HALCON

a product of MVTec

# Surface-Based Matching



**HALCON 21.05** *Progress*

MVTec Software GmbH

*Building Vision for Business*

This technical note describes how and when to use the different surface-based matching approaches provided in HALCON, Version 21.05.0.0.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

Copyright © 2019-2021 by MVTec Software GmbH, München, Germany




Protected by the following patents: US 7,062,093, US 7,239,929, US 7,751,625, US 7,953,290, US 7,953,291, US 8,260,059, US 8,379,014, US 8,830,229. Further patents pending.

Microsoft, Windows, Windows Server 2008/2012/2012 R2/2016, Windows 7/8/8.1/10, Microsoft .NET, Visual C++, and Visual Basic are either trademarks or registered trademarks of Microsoft Corporation.

All other nationally and internationally recognized trademarks and tradenames are hereby recognized.

More information about HALCON can be found at: http://www.halcon.com/

# About This Technical Note

This technical note describes the surface-based matching functionality applicable with HALCON by giving guidance on how to approach your use case and how to apply the according HALCON operators.

Furthermore, common problem sources are discussed and tips and tricks are provided.

# Contents


**1 Overview of Surface-Based Matching** **7**
1.1 Introduction . . . 7
1.2 Surface-Based Matching Methods . . . 7

**2 Workflow of Surface-Based Matching** **11**
2.1 Creating the Surface Model . . . 11
2.2 Finding the Surface Model . . . 11
2.2.1 Approximate Matching . . . 13
2.2.2 Sparse Pose Refinement . . . 13
2.2.3 Dense Pose Refinement . . . 13
2.3 Visualizing the Result . . . 14

**3 Data Requirements** **15**
3.1 XYZ-Mappings . . . 15
3.2 Surface Normal Vectors . . . 15

**4 Troubleshooting** **17**
4.1 Normals . . . 17
4.2 Shape of the Model . . . 17
4.3 Duplicate Points . . . 18
4.4 Checklist . . . 19
4.4.1 Points to Consider for Both Surface-Based Matching and Edge-Supported Surface-Based Matching . . . 20
4.4.2 Points to Consider for Edge-Supported Surface-Based Matching Only . . . 21

**5 Tips & Tricks for Improved Speed, Robustness and Accuracy** **22**
5.1 Speed . . . 22
5.1.1 Model . . . 22
5.1.2 Scene . . . 22
5.2 Robustness . . . 23
5.3 Accuracy . . . 25

**6 Background Removal** **26**
6.1 Remove Background Plane . . . 26
6.1.1 Orthogonal Camera Setup . . . 26
6.1.2 Tilted Camera Setup . . . 26
6.2 Remove Background of Arbitrary Shape . . . 27
6.3 Remove Bin . . . 28
6.3.1 Remove Bin Based on a Meshed CAD Model . . . 28
6.3.2 Remove Bin Based on Thresholding . . . 28

# Chapter 1

# Overview of Surface-Based Matching

## 1.1 Introduction

Surface-based matching is a method used to localize objects in 3D space. In order to determine the position of the wanted object, you need

- a **3D surface model**, which determines the shape and features of the object to be localized and
- a **3D scene**, which corresponds to a 3D point cloud representing the 3D space the model is searched for in.



Figure 1.1: The surface model of an object (left), a 3D point cloud (center), where the surface model should be located in, and the final matching result (right), where the found instances of the model are marked green.

Along with the model and the scene, a number of parameters is regarded in order to set the proceedings of the matching algorithm. This way, the matching process can be adapted to the features and particularities of the object and the scene.

As a result you receive the pose(s) of your object, i.e., its position(s) and orientation(s) in the provided scene (see the chapter reference "Poses").

To learn how surface-based matching differs from further 3D matching methods in HALCON see the chapter reference "3D Matching".

## 1.2 Surface-Based Matching Methods

Surface-based matching can be used to cover a variety of different use cases. These different use cases depend on the object to be found, as for certain objects edge information should be considered. We distinguish the cases needing:

- **only 3D surface information**,

- **3D surface** and **3D edge information**,
- **3D surface, 3D edge**, and **2D edge information**, or
- **3D surface** and **2D edge information**.

An overview of the different operation modes of surface-based matching, some use cases and considered data are given in table 1.1. In general, you distinguish the 3D scene and the 3D model to be found in it.

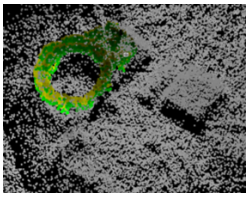
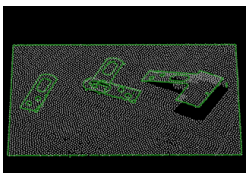
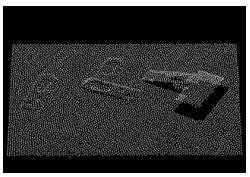
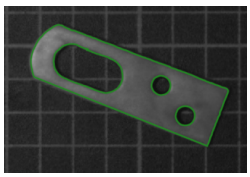
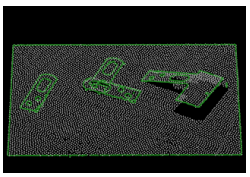
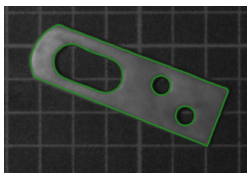
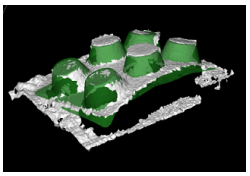
| | Data | Example | Use Cases |
|---|---|---|---|
| **(Standard) Surface-Based Matching** | Surface |  | Large and round shapes, extended in all three dimensions |
| **Edge-Supported Surface-Based Matching** | Surface + 3D edge |  | Shapes with reduced 3D information, i.e., flat objects, boxes, objects that contain 3D information especially at edges and speed is in focus, or the 2D edges do not have an additional value. |
| | Surface + 2D edge |   | Shapes with reduced 3D information, i.e., flat objects with 3D information especially at edges. However, the 3D data are noisy close to edges. |
| | Surface + 3D edge + 2D edge |   | Shapes with reduced 3D information, i.e., flat objects, boxes, or objects that contain 3D information especially at edges and accuracy is in focus. |
| **Deformable Surface-Based Matching** | Surface |  | Objects that can occur in deformed shapes. |

Table 1.1: Summary of the surface-based matching operation modes and their corresponding use cases.

The standard surface-based matching using only surface information should be employed in case you need to locate objects with large and round shapes extended in all three dimensions, i.e., the 3D points representing an instance of the object in the scene do not all lie in one plane. Some exemplary objects can be found in figure 1.2.

However, the standard surface-based matching using surface information only should not be applied on objects with reduced 3D information, such as flat objects. A box viewed orthogonally from above appears as a flat object in the scene. Therefore, if the surface-based matching is applied with only surface information, the box will be found in the background with high probability (as long as the background is flat). This is due to the fact that surface information contains only the 3D point coordinates and their sampling distances as well as their normal directions as features. Therefore, the score of the box found in the background will have a similar value to the score of boxes found correctly (see figure 1.3). For such cases, the edge-supported surface-based matching should be applied (see figure 1.4).

Edge-supported surface-based matching is best applied on objects like shown in figure 1.4. The following sets of information can be used for an edge-supported surface-based matching:

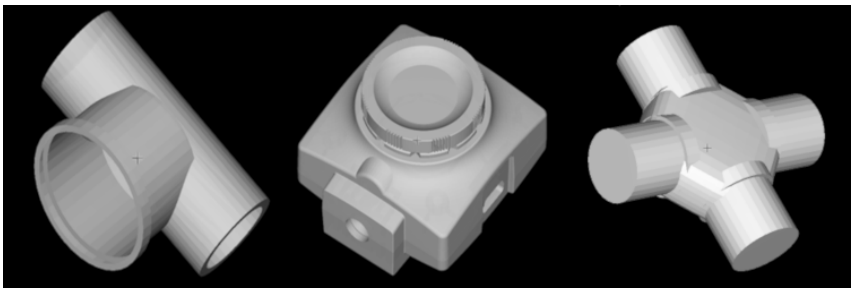


Figure 1.2: Objects suited for **standard surface-based matching**.

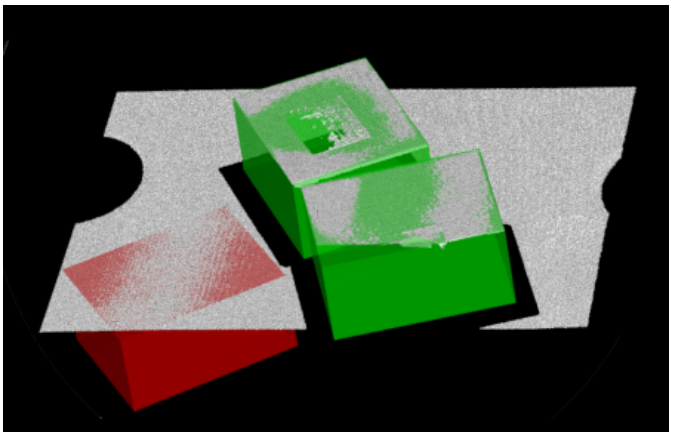


Figure 1.3: Without edge support, for objects with reduced 3D information, like boxes, candidates are falsely located in the background (red) instead of the actual instances (green).

- **Surface + 3D edge information**
  In case speed is crucial, surface and 3D edge information should be used. This set of information can also be chosen in case 2D edge information has no positive but negative impact. This occurs for instance, when additional 2D edges from, e.g., a bar code are present on the object (see figure 1.5).
- **Surface + 2D edge information**
  If point cloud information is too noisy in order to compute adequate 3D edges or in case the point cloud is not ordered, i.e., no XYZ-mappings are available (see section 3.1 on page 15), surface information and 2D edges should be used.
- **Surface + 3D edge + 2D edge information**
  When robustness and accuracy are crucial, both 3D and 2D edge information should be used as long as adequate edges can be extracted.

While standard-surface-based matching and edge-supported surface-based matching share the same set of operators, there is a specific set of operators for deformable surface-based matching solely. In the following chapters, this technical note focuses on surface-based matching with non-deformable models. For further information regarding deformable surface-based matching specifically refer to the respective operator references, `find_deformable_surface_model` and `create_deformable_surface_model` in particular, or the HDevelop example program `find_deformable_surface_model.hdev`.

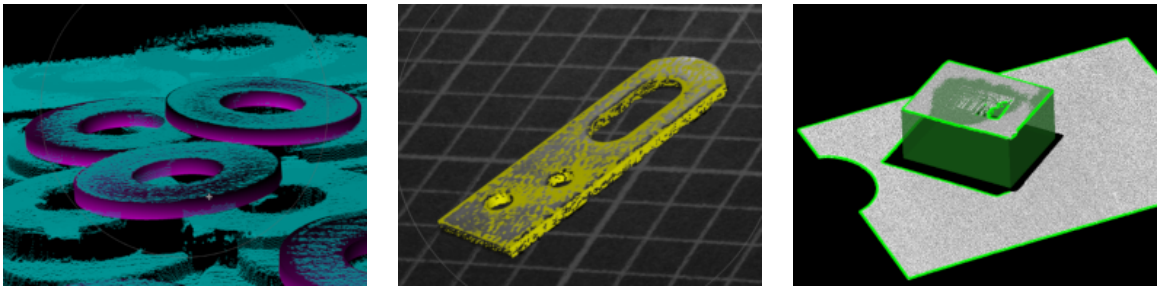


Figure 1.4: **Edge-supported surface-based matching** should be used in case objects with reduced 3D information need to be located, i.e., flat objects, almost symmetric objects, and objects that have larger shapes extended only in two dimensions.

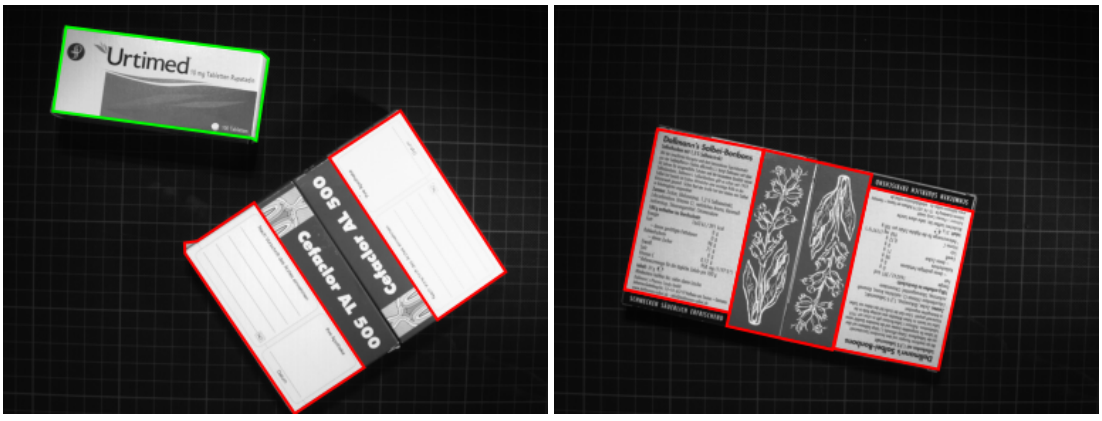



Figure 1.5: When using 2D edges, labels, data codes etc. can have a negative impact on the extraction of the outer box edges.

# Chapter 2

# Workflow of Surface-Based Matching



To give guidance on the general proceedings the workflow of surface-based matching is described step by step. Note that there are also a number of HDevelop example programs that can be consulted, showing a variety of surface-based matching approaches.

## 2.1 Creating the Surface Model

In order to find your object within a 3D scene you need a 3D model of the object, so its spatial features can be matched. There are various ways to get a 3D object model, it might, e.g., be loaded from a CAD file, or extracted manually from a 3D point cloud. Nevertheless, in order to apply it for surface-based matching you need to create a surface model for it, using `create_surface_model`. If you already have a surface model you can just read the model using `read_surface_model`.

Note, that besides the 3D point data the 3D object model needs to contain further information, such as point normals, a triangular or polygon mesh, or 2D-mappings (see the operator reference of `create_surface_model` for details).

For the creation of the surface model, the 3D object model is sampled according to the sampling distance determined in the parameter `RelSamplingDistance`. For the refinement of the pose the model is sampled once more, controlled by the generic parameter *'pose_ref_rel_sampling_distance'* (see figure 2.1). Note that it is usually not necessary to change these two parameters, as the default values work for a wide variety of models.

During the surface model creation, you can also train or compute further features of the model, depending on your matching approach. You can, e.g., train the model for 3D edge-support or invert the model normals using generic parameters. For details refer to the operator reference of `create_surface_model`.

## 2.2 Finding the Surface Model

Once the surface model of the object to be matched has been created with `create_surface_model`, you can use `find_surface_model` to look for the object in your scene. The scene is usually acquired by a 3D sensor, whereby certain requirements are put on the data (see chapter 3 on page 15). A preprocessing step, used to remove the background (see chapter 6 on page 26) can enhance the performance significantly. Note that the workflows of the standard surface-based matching and the edge-supported surface-based matching are the same.

Generally the surface-based matching is performed in three main steps, which are carried out by `find_surface_model`:

1. **Approximate matching:**
   The approximate poses of the instances of the surface model in the scene are searched. Point clouds are sampled and a subset of key points is defined. For each key point, the optimum pose of the surface model is computed under the assumption that the key point lies on the surface of the object. From all key points, the poses with the best scores are then selected and used as approximate poses.

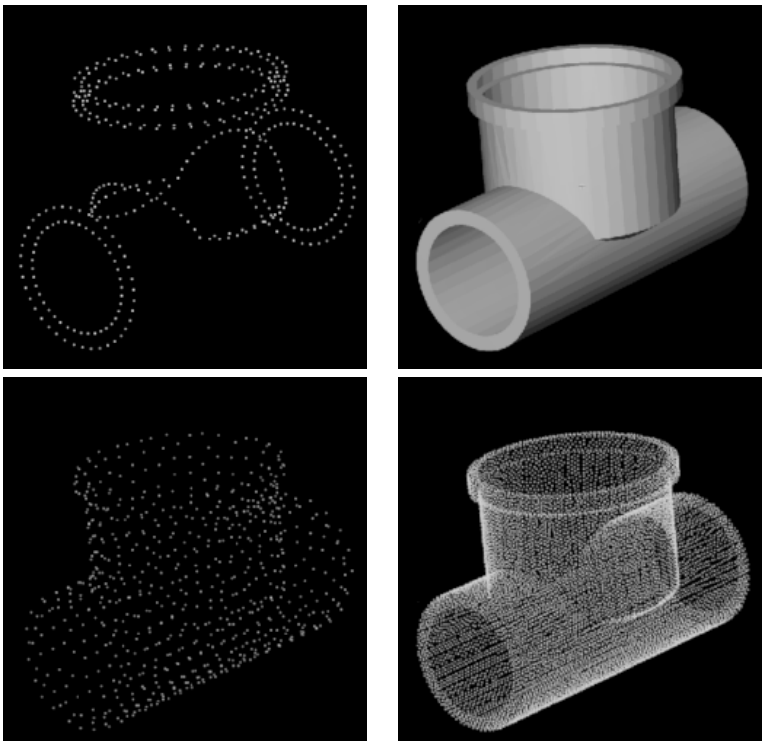
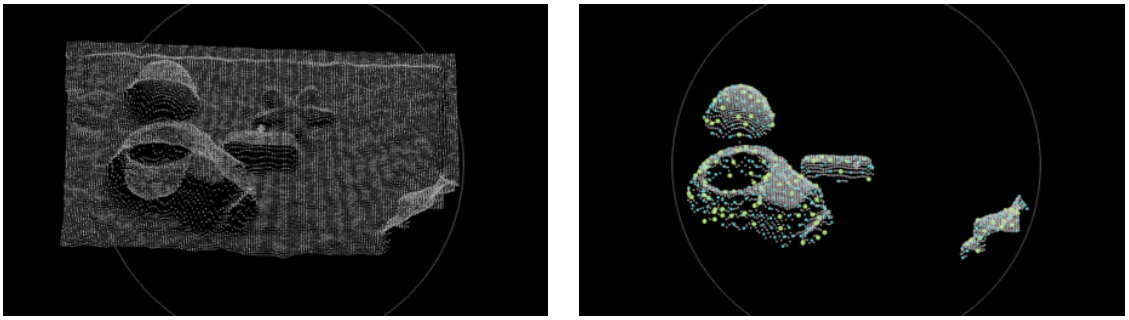


Figure 2.1: The 3D object model contains a number of points (top left) and a polygon mesh (top right). For the surface model the 3D model is sampled for the following steps, once for the approximate matching (bottom left) and once for the two pose refinement steps (bottom right).

2. **Sparse pose refinement:**
   In this step, the approximate poses found in the previous step are refined by minimizing the distances between the sampled scene points and the planes of the model points. This increases the accuracy of the poses and the significance of the score value.

3. **Dense pose refinement:**
   The poses found in the previous steps are accurately refined. This step works similar to the sparse pose refinement and minimizes the distances between all scene points and the planes of the closest model points.

Those three steps are explained in detail in the following subsections.



a) Scene point cloud

b) Reduced scene (gray), sampled points (cyan) and key points (yellow)

Figure 2.2: The scene, as it is acquired (a) and the point sets that are used for the different steps of the matching process (b). Note that in the second image the background has been filtered out, using the XYZ-mappings.

### 2.2.1 Approximate Matching

First, points are sampled uniformly from the scene passed with the parameter `Object3D`. The sampling distance is controlled with the parameter `RelSamplingDistance`.

Then, a set of key points (see figure 2.2) is selected from the sampled scene points. The number of selected key points is controlled with the parameter `KeyPointFraction`.

For each selected key point, the optimum pose of the surface model is computed under the assumption that the key point lies on the surface of the object. This is done by pairing the key point with all other sampled scene points (see figure 2.2) and finding the point pairs on the surface model that have a similar distance and relative orientation. The similarity is defined by the parameters *'feat_step_size_rel'* and *'feat_angle_resolution'* in `create_surface_model`. The pose for which the largest number of points from the sampled scene lie on the object is considered to be the best pose for this key point. The number of sampled scene points on the object is considered to be the score of the pose.

If the model was trained for edge-supported surface-based matching, edges are extracted from the 3D scene, similar to the operator `edges_object_model_3d`, and sampled. In addition to the sampled 3D surface, the reference points are then paired with all sampled edge points and similar point-edge-combinations are identified on the surface model.

From all key points the poses with the best scores are then selected and used as approximate poses.

If the pose refinement is disabled, the score described above is returned for each pose in `Score`. The value of the score depends on the amount of surface area of the instance that is visible in the scene and on the sampling rate of the scene. Only poses whose score exceeds `MinScore` are returned. To determine a good threshold for `MinScore`, it is recommended to test the matching on several scenes. The maximum number of returned poses is set with the generic parameter *'num_matches'*.

Note that the resulting poses from this step are only approximate. The error in the pose is proportional to the sampling rates of the surface model given in `create_surface_model`, and is typically less than 5% of the object's diameter.

### 2.2.2 Sparse Pose Refinement

The sparse pose refinement uses the sampled scene points from the approximate matching. The pose is optimized such that the distances from the sampled scene points to the plane of the closest model point are minimal. The plane of each model point is defined as the plane perpendicular to its normal.

Additionally, if the model was trained for edge-supported surface-based matching and it was not disabled using the parameter *'use_3d_edges'*, the pose is also optimized such that the sampled edge points in the scene align with the edges of the surface model.

The sparse pose refinement is enabled by default. It can be disabled by setting the generic parameter *'sparse_pose_refinement'* to *'false'*. For large scenes with much clutter, i.e., scene parts that do not belong to the object of interest, it is faster to disable the sparse pose refinement.

The score of each pose is recomputed after the sparse pose refinement.

### 2.2.3 Dense Pose Refinement

This step works similar to the sparse pose refinement and minimizes the distances between the scene points and the planes of the closest model points. The difference is that

- a second sampled model of the scene is used (hereby, the number of points depends on the parameter *'pose_ref_sub_sampling'*), and
- if the model was created for edge-supported surface-based matching and it was not disabled using the parameter *'use_3d_edges'* (see above), all extracted scene edge points are used for the refinement, instead of only the sampled edge points.

Taking all points from the scene increases the accuracy of the refinement but is slower than refining on the subsampled scene points. The dense pose refinement is enabled by default, but can be disabled with the generic parameter *'dense_pose_refinement'*.

After the dense pose refinement, the score of each match is recomputed. The threshold for considering a point to be 'on' the object is set with the generic parameter value *'pose_ref_scoring_dist_rel'* or *'pose_ref_scoring_dist_abs'*. When using the edge-supported matching, the parameters *'pose_ref_scoring_dist_edges_rel'* or *'pose_ref_scoring_dist_edges_abs'* control the corresponding thresholds for edges.

The final accuracy of the refined pose depends on several factors. The internal refinement algorithm has an accuracy of up to 1e-7 times the size (diameter) of the model. This maximal accuracy is only achieved for best possible conditions. Factors for the final accuracy are, e.g., the shape of the model, the number of scene points, the noise of the scene points, the visible part of the object instance, and the position of the object.

The number of matches to be returned can be controlled by the generic parameter *num_matches* but also depends on whether the matches exceed the minimum score `MinScore`.

## 2.3 Visualizing the Result

If your matching approach was successful you can check the result by visualizing it with the procedure `visualize_object_model_3d` (see figure 2.3). Therefore, display your model in the scene after translating and rotating it according to the result pose(s) that are returned by `find_surface_model` or `find_surface_model_image` in the parameter `Pose`. If used for the matching, you can also display the 3D edges in this visualization.

```
rigid_trans_object_model_3d (ObjectModel3D, Pose, ObjectModel3DRigidTrans)
visualize_object_model_3d (WindowHandle, [Scene, ObjectModel3DRigidTrans], [], [], [], \
                           [], [], [], [], VisPose)
```

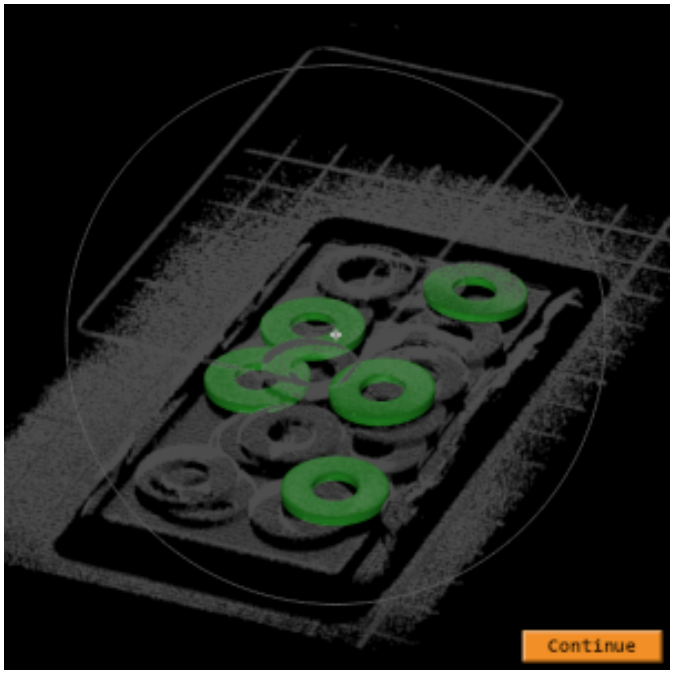



Figure 2.3: Use the HDevelop procedure `visualize_object_model_3d` to inspect your results (green).

# Chapter 3

# Data Requirements

In order to obtain fast, accurate and robust matching results, the required data should be available and conform to the underlying conventions. These data include *XYZ-mappings* (section 3.1) and the *surface normal vectors* (section 3.2). They are explained in the following sections.

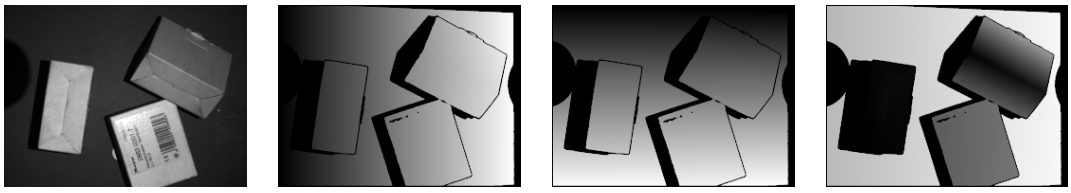


## 3.1 XYZ-Mappings

XYZ-mappings are XYZ-images that map 3D coordinates into 2D coordinates, i.e., the X-coordinates are encoded as gray-values in the X-image, the Y-coordinates are encoded as gray-values in the Y-image and the Z-coordinates are encoded as gray-values in the Z-image (see example in figure 3.1). As the order of the 3D coordinates might be important for further processing steps, it is coherently determined by the 2D image coordinates.

When handling 3D object models, for instance when a model is prepared for surface-based matching, using 2D-mappings instead of the point cloud data can increase speed and robustness distinctly.

In the context of surface-based matching, mappings are especially used in order to compute normal vectors and 3D edges. If the model is trained for edge-supported surface-based matching, mappings are explicitly required for the scene. The mappings can be created using the operator `object_model_3d_to_xyz`.



a) Scene image b) X-image c) Y-image d) Z-image

Figure 3.1: Image of a scene and the corresponding XYZ-images.

## 3.2 Surface Normal Vectors

A point normal is a normalized vector attached to the point coordinate. It defines the surface tangent on the point. In HALCON point normals can be computed with:

- `surface_normals_object_model_3d` and `smooth_object_model_3d`
  Note that `smooth_object_model_3d` redistributes the points to describe a smoother surface manifold.
- `sample_object_model_3d`
  The operator `sample_object_model_3d` with the mode *'fast_compute_normals'* re-computes normal vectors if a mesh or mappings are available; therefore, any available normal vectors are ignored in those cases.

For robust and successful matching results, it is important that the normals of the model and the normals of the scene both point approximately in the same direction. Furthermore, for edge-supported surface-based matching model and scene normals need to point inwards.

Note that `find_surface_model` and `find_surface_model_image` ignore any polygonal or triangular mesh data, points and normals are used instead. If the scene does not have normals, they will be computed and aligned based on the parameter *'scene_normal_computation'* of `find_surface_model` and `find_surface_model_image`. Note that, if the model was trained for edge-supported surface-based matching, the normals are computed from the mappings of the scene, regardless of whether there are normals provided already or not (from HALCON 20.11 onwards). For the matching, it is not recommended to use scenes that do not contain any normals or XYZ-mappings. Check the normals, among various further features, of your scene by using the procedure `debug_find_surface_model`.

# Chapter 4

# Troubleshooting

If matching results are not as expected, potential sources of problems can be identified by visualizing and debugging with the procedure `debug_find_surface_model`. For the standard surface-based matching, major problems are

- that the model normals and surface normals do not both point either inwards or outwards from the model surface,
- scaling issues, or
- that the shape of the object does not have enough 3D information.

For the edge-supported surface-based matching, in addition to the aforementioned issues, users may encounter problems related to

- the presence of duplicate points.

In the following, these problem sources are discussed. At the end, a tabulated summary is provided for problem-solving.



## 4.1 Normals

The direction and orientation (inwards or outwards) of the normals of the model and scene must be consistent for the matching process: For both standard and edge-supported surface-based matching, model normals and scene normals should point approximately in the same direction. This can be checked visually using the procedure `debug_find_surface_model` (see figure 4.1).

For the standard surface-based matching, the normals' orientation can be adjusted by setting the parameter *'model_invert_normals'* to be *'true'* when creating it with `create_surface_model`.

For edge-supported surface-based matching the normals need to point inwards. This should be the case for the scene normals anyway, as they are calculated from the 2D-mappings.

## 4.2 Shape of the Model

Only for shapes containing enough 3D information, i.e., shapes consisting of tilted or curved surfaces facing many directions, standard surface-based matching returns robust results. Some models have parts that could very well fit in several locations of the scene, for instance, planar objects present in a planar scene (e.g., a box viewed from above). In those cases, it is difficult to fix all dimensions of the pose, since the matching score hardly varies under largely different translation or rotation values. However, this ill-posed behavior can be avoided by using edge-supported surface-based matching (see figure 4.2).

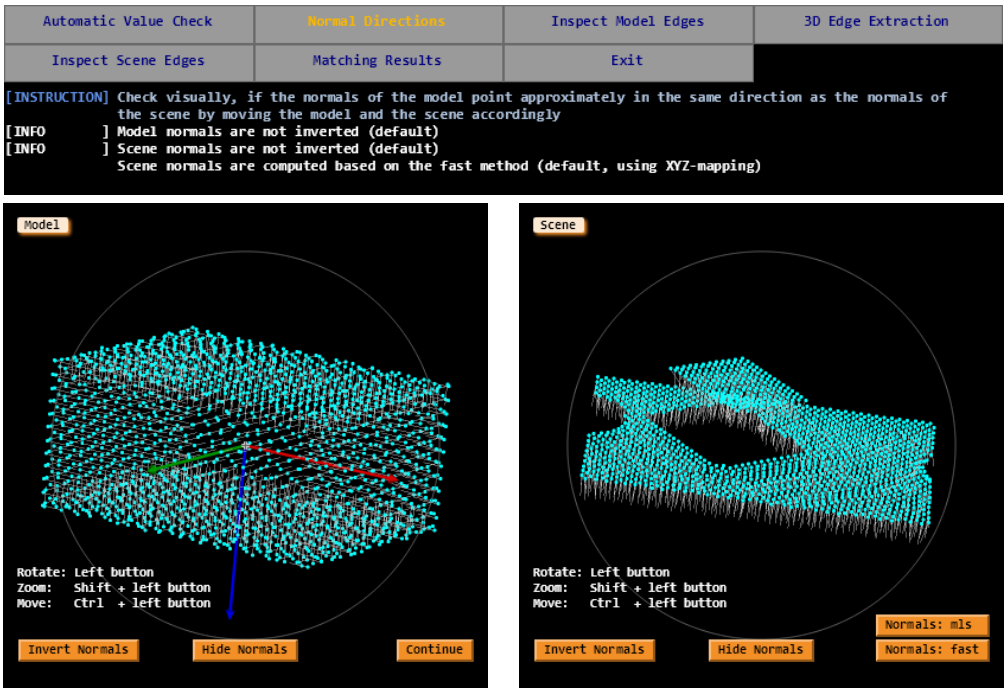
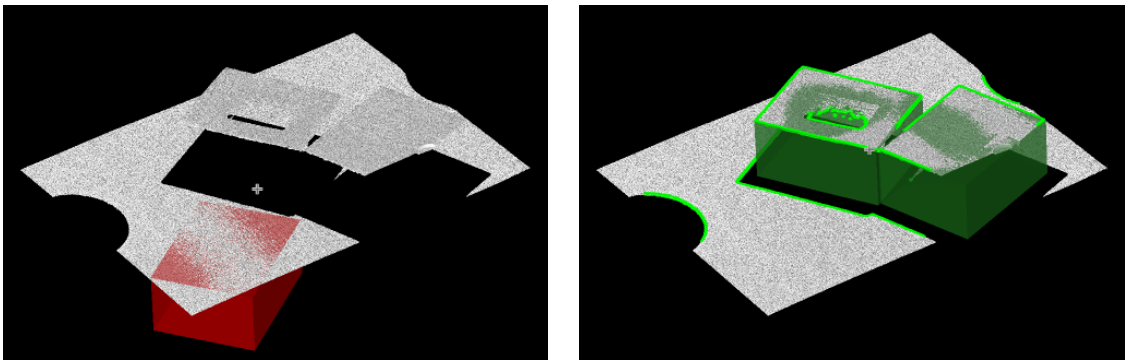



Figure 4.1: The procedure `debug_find_surface_model` for checking surface-based matching, given a scene and a model to be found. In this example, normal consistency is checked.



a) Matching a planar object without edge-support. b) Successful matching with edge-support.

Figure 4.2: If the model shape does not differ significantly from the spatial features of the scene, edge-supported surface-based matching can be used.

## 4.3 Duplicate Points

The presence of duplicate points, i.e., points with identical coordinates, often depends on the sensor's handling of data during the 3D acquisition process. In some cases, parts of the sensor do not register any information, e.g., caused by shading or the reflective features of a surface, which leads to invalid data points. Some 3D sensors remove invalid points, some code them as `NaN`, and others code them with value `0`. If those values are assigned with `0`, all the invalid data points will be reconstructed as the $(0, 0, 0)$ coordinate point. In order to check if duplicate points are available, you can use the 'Automatic Value Check' of the procedure `debug_find_surface_model` or the gray-value histogram of the Z-mapping (see figure 4.3). If duplicate points are present, the matching process will slow down and the 3D-edge extraction will fail.

Duplicate points can be removed by an appropriate threshold on the Z-mapping or by applying the operator `select_points_object_model_3d`. In the example of figure 4.3 the following two approaches will solve the problem.

**Mapping based method:**

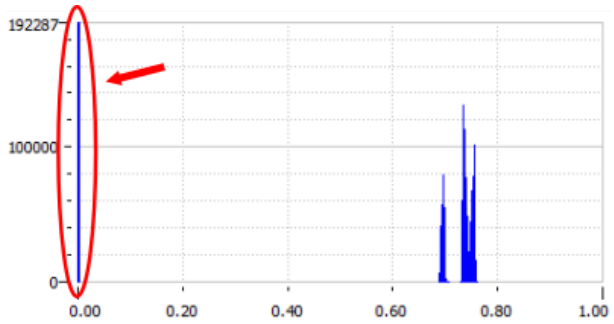



Figure 4.3: Check for duplicate points with the gray-value histogram of the Z-mapping represented by the peek at value 0.

```
object_model_3d_to_xyz (XMapping, YMapping, ZMapping, Scene, 'from_xyz_map', [], [])
threshold (ZMapping, Region, 0.000001, 1)
reduce_object_model_3d_by_view (Region, Scene, CamParam, Pose, SceneReduced)
```

**3D-based approach:**

```
select_points_object_model_3d (Scene, 'num_neighbors_fast 0.000001', 0, 1, SceneReduced)
```

## 4.4 Checklist

The following lists give some hints on how to encounter common issues that can occur when implementing and running your application.

### 4.4.1 Points to Consider for Both Surface-Based Matching and Edge-Supported Surface-Based Matching

| Surface-Based Matching and Edge-Supported Surface-Based Matching | |
|---|---|
| **Normal directions** | see section 4.1 on page 17 |
| **Shape of the model** | see section 4.2 on page 17 |
| **Duplicate points** | see section 4.3 on page 18 |
| **Scaling** | Use `read_object_model_3d` with correct scaling. |
| **Sampling of the model** | Check the sampling of the model with the following code:<br>`get_surface_model_param (SurfaceModelID, 'sampled_model', SampledModel)`<br>`visualize_object_model_3d(WindowHandle, SampledModel, [], [], [], [], [], [], [], PoseOut)`<br>Make sure the essential parts of the object are still represented in the surface model. |
| **Sampling of the scene** | Check the sampling of the scene with the following code:<br>`get_surface_matching_result (SurfaceMatchingResultID, 'sampled_scene', 0, SampledScene)`<br>`visualize_object_model_3d(WindowHandle, SampledScene, [], [], [], [], [], [], [], PoseOut)`<br>The object should be covered by at least 150 scene points. In case that fine details of the model should be considered, the sampling distance needs to be lowered. When long and thin objects need to be matched, the sampling distance must be adapted such that enough information is available in the shortest object extension (see figure 5.1). |
| **Score** | If only low scores are returned check the following potential causes:<br>• If the model contains structures on the inside that cannot be seen, those invisible structures should be removed.<br>• If the model normals do not point approximately in the same direction as the scene normals, they should be adapted (see section 4.1 on page 17).<br>• The model needs to fit the objects in the scene.<br>• The model is sampled a second time for the refinement. This sampling can be controlled by the parameter value *'pose_ref_rel_sampling_distance'*. In order to obtain more accurate results, this parameter can be set to a lower value. Its default value is `0.01`.<br>• If only few of the object instance is visible for the camera, use the view-based score, which considers only that part of an object, that can actually be seen from the camera viewpoint. |

### 4.4.2 Points to Consider for Edge-Supported Surface-Based Matching Only

| Edge-Supported Surface-Based Matching (in addition to the points above) | |
|---|---|
| **Model normals** | If model normals do not point inwards, invert the model normals by setting the parameter *'model_invert_normals'* of `create_surface_model` to *'true'*. As the scene normals are calculated from the 2D-mappings (from HALCON 20.11 onwards), the scene normals should point inwards anyway. Else, set the parameter *'invert_scene_normals'* of `find_surface_model` to *'true'* (from HALCON 19.11 onwards) or reconsider the consistency of the used 2D-mapping. |
| **Score** | If only low scores are returned check the following potential causes:<br>• Check if 3D edges are computed in a suitable way. 3D edges can be accessed by `get_surface_matching_result` with the parameter *'sampled_3d_edges'*. If 3D edges can not be computed adequately due to the underlying noise, they can also be determined with `edges_object_model_3d` and filtered by a suitable method. Then, the 3D edges can be set as a generic parameter in `find_surface_model`, i.e., with *'3d_edges'* as generic parameter and the filtered edges as its value.<br>• Retrieve the results of the matching using `get_surface_matching_result` and check the values for *'score_surface'*, *'score_3d_edges'* and *'score_2d_edges'*.<br>  • If the *'score_3d_edges'* is very low, increase the value of the parameter *'pose_ref_scoring_dist_rel'* / *'pose_ref_scoring_dist_abs'* when calling `find_surface_model`. This can be due to the fact that 3D edges of the objects in the scene are slightly deformed or the point cloud is slightly noisy close to 3D edges.<br>  • If the *'score_2d_edges'* is very low, check if the set camera parameters and camera pose are correct. In order to do so, project the model into the 2D intensity image. In case the objects in the scene are slightly deformed, increase the value of the parameter *'max_deformation'* of `find_surface_model_image`. |

# Chapter 5

# Tips & Tricks for Improved Speed, Robustness and Accuracy

Generally, it is highly recommended to start with the default parameters. In case the matching results are not as expected, the most important parameters to modify for improved speed, robustness, and accuracy are explained below.

## 5.1 Speed

In order to speed up the matching process, there are two major approaches:

Use a straightforward parameter set, e.g., choose the fast option when determining the normal computation of the model, or reduce the number of points by an appropriate pre-processing step on the point cloud. Features regarding 3D model and scene, which can affect the speed are listed in the following sections.

### 5.1.1 Model

**Complexity of the model:** The more surfaces the model contains, the slower the training and matching process. This is especially the case if *'train_3d_edges'* is enabled. In such cases, the model should be simplified as much as possible, for instance, with `simplify_object_model_3d`.

**Inner structure of the Model:** A slowdown can be due to the fact that the model contains non-visible surfaces inside. Such triangles need to be removed.

### 5.1.2 Scene

**Sampling distance:** The larger the sampling distance, the smaller the amount of data points and thus the faster the object localization process. However, the sampling distance should be set such that at least 150 points are distributed on the object. Moreover, in case smaller or thinner shapes are to be considered, the sampling distance should be set such that enough points are distributed over the smaller or thinner shape as well. For instance, if a long and thin object should be localized, the sampling distance should be set such that it is at least one fourth of the smaller extension of the object.

**Shape of the model:** In case of flat objects, the optimization process will take longer. In such cases, it is recommended to segment the scene and to apply the surface matching in a loop of the segmented components.

**Disable 3D edge alignment:** When applying surface-based matching with surface information, 3D and 2D edge information, 3D edge alignment can be disabled, which can speed up the matching, especially if handling noisy point clouds. This can be done by setting *'use_3d_edges'* to be *'false'*.

**Use XYZ-mappings:** XYZ-mapping-based processing is much faster than point cloud processing due to the already available information structure or order. Note that you can also enhance the preprocessing of your 3D data, by applying it to the XYZ-images.

**Duplicate points:** Duplicate points lead to slower processing. The presence of duplicate points can be checked and removed as described in section 4.3 on page 18.

**Number of points:** If there are a lot of scene points that lie on the object instance and/or the point cloud is very dense, it is recommended to increase the value set for the generic parameter *'pose_ref_sub_sampling'* (its default value is *2*, i.e., every second scene point is used in the dense pose refinement).

**Remove outliers:** Use `select_points_object_model_3d` with the parameter *'num_neighbors_fast'* to remove outliers.

**Remove noise:** Noise can be removed with one of the following approaches:

- `median_image` on the Z-mapping. Note that if smoothing is applied on the Z-image , artifacts appear in the XY-plane at Z=0. Therefore, a threshold should be carried out afterwards with a value that excludes all Z=0 values.
- Apply a standard blob analysis on the Z-mapping. Remove smaller blobs from the image domain and reduce the scene to the new domain by applying `reduce_object_model_3d_by_view` or `xyz_to_object_model_3d`.

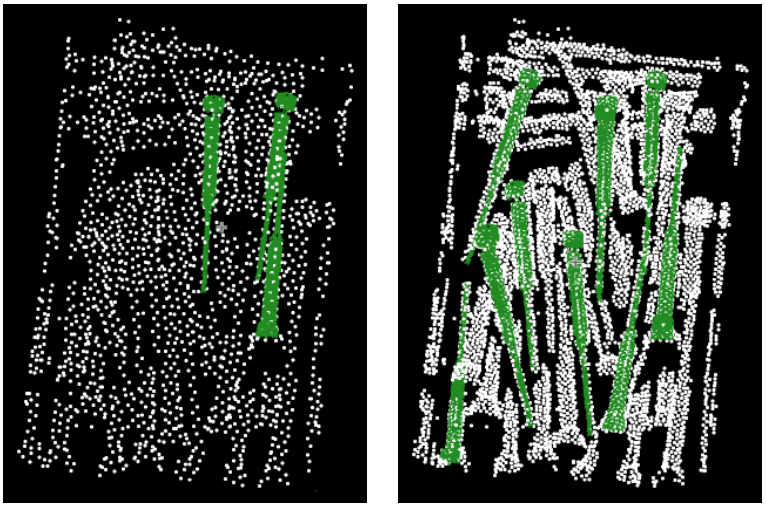
**Remove background:** Generally, the fewer the points to be processed, the faster the matching. In most cases, an adequate preprocessing can speed up the application significantly. Such a preprocessing can be applied on the XYZ-mappings or on the point cloud. For further information see chapter 6 on page 26.

## 5.2 Robustness

Higher robustness can especially be achieved by improving the significance of the computed scores. There are three major parameters that can be used in order to improve the matching concerning its robustness.

**Key point fraction:** Increasing the number of key points increases the number of approximate matching poses, and therefore, more poses are available to be analyzed in the following two refinement steps. Note that this increases the processing time as well.

**Sampling distance:** If the model contains a finer structure or a high density of geometrical information, reducing the sampling distance leads to more robust results. Note that the distance should be defined such that the characteristic features of the model are displayed fine enough (see figure 5.1 for an example case with a long and thin object).



a) *RelSamplingDistance*= 0.03 b) *RelSamplingDistance*= 0.015

Figure 5.1: With the determined distance between the sampling points of a scene, the object characteristics still need to be represented.

**Point thresholding:** In case 3D edges with nonlinear trajectories are available, i.e., edges of a box are rather round, matching results and scores can be improved by increasing the values of *'pose_ref_dist_threshold_edges_rel'* or *'pose_ref_dist_threshold_edges_abs'*.

Robustness and scores can further be affected by the parameters listed below.

**Duplicate points:** Duplicate points can also lead to less robust results. Therefore, it is highly recommended to remove them before any further processing (see section 4.3 on page 18).

**Normal directions:** If model normals and scene normals do not point approximately in the same direction, matching results are not robust or matching will not work at all (see section 4.1 on page 17).

**Shape:** Shapes with reduced 3D information lead to lower robustness. Therefore, in such cases it is highly recommended to preprocess the data in a suitable way and use edge-supported surface-based matching.

**View-based score:** The view-based score only considers those model points, that are potentially visible from the determined viewpoint. Therefore the robustness of the matching results increases as this score is more expressive.

**Maximum overlap:** Due to efficiency reasons, the generic parameters *'max_overlap_dist_rel'*/ *'max_overlap_dist_abs'* are based on the distance of the centers of the axis oriented bounding boxes of neighboring objects in the scene. Therefore, in case of long and thin objects, the generic parameters should be set to a much lower value (at about the smallest extension of the object, see figure 5.2). However, this also leads to overlapping matching results, which need to be filtered out in a post-processing step.

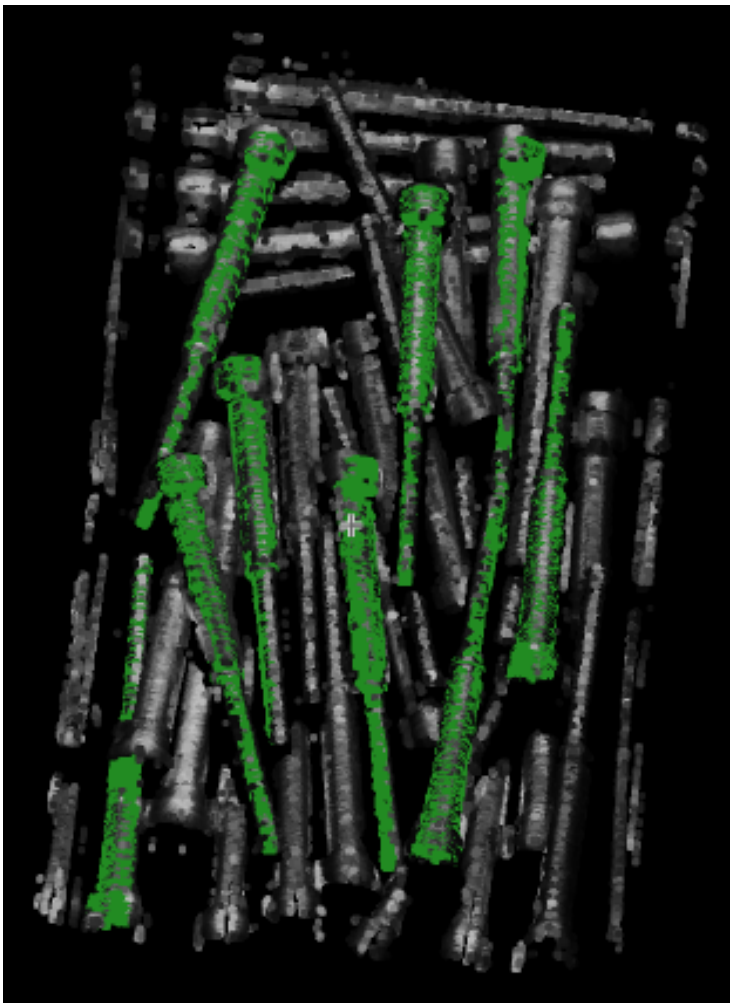


Figure 5.2: The object model has a length of 45 mm and a maximal diameter of 4.5 mm. Therefore the *'max_overlap_dist_abs'* is set to 4.5 mm and the sampling distance is set to 0.6 mm.

**Self-similarities:** The pose of objects that are almost symmetric are sometimes hard to distinguish for `find_surface_model`. Enable the generic parameter *'train_self_similar_poses'* when creating the surface model, in order to avoid this.

## 5.3 Accuracy

The internal refinement algorithm has a maximum achievable accuracy.

This accuracy limitation is due to numerical constraints linked to the internal processing of data and variables. The maximum error is at around 1e-3 (or 0.1%), but can be practiced at around 1e-7 times the size (diameter) of the model, or 0.00001%. Note that this is the maximum accuracy that can be achieved if all the conditions are perfect.

The model size affects the (absolute) accuracy because coordinates are stored as float values. This is one reason why the accuracy is relative to the model size. For example, if the scale of the model and the scenes is changed (e.g., by multiplying all model and scene coordinates with the factor 1.000.000, the absolute accuracy would also change by the factor 1.000.000). In contrast, the accuracy in relation to the model diameter would remain the same.

The following features can affect the accuracy:

**Shape of the model:** Some models have shapes which allow certain degrees of freedom for the pose. For example, planar objects can translate and rotate on a scene plane without changing the score too much. Note that sometimes this can be avoided by using edge-supported surface-based matching. This applies to all self-similar or symmetric shapes: Cylinders, spheres, planar objects etc.

**Visible part of the object instance:** Even if the shape of the object is suitable, for example, in the case of a box, it can happen that only a small part of the object is visible, for instance, a small part of one side of the box. In this case, the translation and rotation inside that one box side cannot be properly determined. Ideally, several orthogonal sides of the object instance should be visible, so that the translation (and rotation) are fixed in all directions.

**Number of scene points that lie on the model:** If only few scene points represent an instance of the object, no high accuracy can be achieved.

**Point noise from the sensor:** More accurate input points will also lead to a more accurate pose. Note for example that time-of-flight sensors can cause noise deviations up to 1 cm, whereas sheet-of-light setups might lead to 0.1 mm noise; so, coarsely, there are 2 orders of magnitude of difference.

**Object position in scene coordinates:** The further away an instance is from the origin of the scene, the smaller the possible accuracy is. For example, if an object is 1.000 times its diameter away from the origin of the scene, the accuracy drops by a factor of up to 1.000. This is because the coordinates are stored as float values, and they will lose some of their significant bits. It is always advisable to keep the scene origin at about 10-100 times the diameter of the model away from its instances at most.

**Refinement sampling distance:** The sampling distance of the model in the refinement step also affects the final accuracy. It can be set using the generic parameter *'pose_ref_rel_sampling_distance'* of the operator `create_surface_model`. With a smaller value, the accuracy can improve, especially if the model contains parts with high curvature or high frequency. It is possible to inspect the impact on the model used in the refinement step using the operator `get_surface_model_param` with the parameter *'sampled_pose_refinement'*. The sampled point cloud should represent the object, including its fine structures. Parts that are not represented by this point cloud are not used during the refinement.

# Chapter 6

# Background Removal

As already mentioned several times, removing the background from your scene before the matching can be useful in many cases. There are various different methods available in order to remove background or to reduce the scene to the objects to be located. In the following, different approaches are described that show how to remove the background plane or bins from the 3D scene.

You can also step through the HDevelop examples

- `remove_bin_for_3d_object_localization.hdev` and
- `remove_background_for_3d_object_localization.hdev`,

which demonstrate the workflows explained in the following sections.

## 6.1 Remove Background Plane

### 6.1.1 Orthogonal Camera Setup

In case height is not changing and the 3D sensor or acquisition setup is mostly orthogonal to the background plane, the background plane can be removed by applying a simple threshold on the Z-mapping (if available) or on the 3D point cloud.

A threshold on the Z-mapping can be applied using the following operators:

```
threshold (ImageZ, Regions, MinThres, MaxThres)
reduce_object_model_3d_by_view (Regions, Scene, [], [], SceneReduced)
```

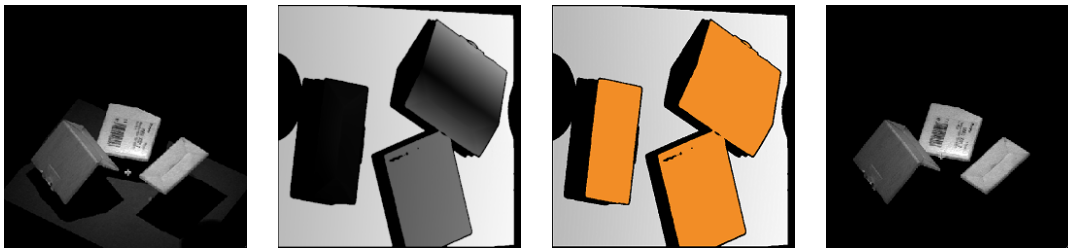
To remove the background plane without using a Z-mapping the following operator calls can be executed:

```
select_points_object_model_3d (Scene, 'point_coord_z', MinThres, MaxThres, ObjectModel3DReduced)
```

### 6.1.2 Tilted Camera Setup

In case the 3D sensor is tilted with respect to the background plane, in a first step, it can be approximated by a plane estimation. In a second step the point cloud will be leveled such that the above-mentioned method can be used in order to remove the background plane.

1. Acquire a reference scene. Then, define a ROI on gray values that describe the background plane. The approximate background plane can be estimated by the following lines of codes:



a) Scene b) Z-image c) Applied threshold d) Reduced scene

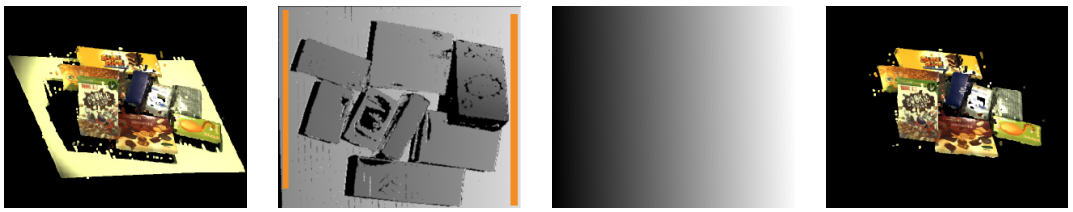
Figure 6.1: A scene is reduced to the contained objects by applying a threshold on its Z-image.

```
fit_surface_first_order (ROI, ImageZ, 'regression', 5, 2, Alpha, Beta, Gamma)
get_image_size (Z, Width, Height)
gen_image_surface_first_order (BackgroundPlane, 'real', Alpha, Beta, Gamma, \
                               Width/2, Height/2, Width, Height)
```

2. Remove the background plane from the online acquired scene by subtracting the estimated plane. This way the scene is leveled and hence a hardcoded threshold can be applied:

```
sub_image (ImageZ, BackgroundPlane, ImageSub, 1, 0)
threshold (ImageSub, Regions, MinThres, MaxThres)
reduce_object_model_3d_by_view (Regions, Scene, [], [], SceneReduced)
```

The HDevelop example `remove_background_for_object_location.hdev` demonstrates this workflow for a given application.



a) Scene b) Z-image c) Estimated plane d) Reduced scene

Figure 6.2: By estimating the pose of the background plane in 3D space, it can be subtracted out.

## 6.2 Remove Background of Arbitrary Shape

In some cases, the background may contain more than just a plane. If it remains steady for all scenes, you can acquire a reference scene (without the wanted objects in it) in order to subtract it from your actual scenes. Follow these steps to remove a background of arbitrary shape:

1. Get the Z-image of the reference scene and your actual scene and subtract them.

```
* Offline step, performed once:
object_model_3d_to_xyz (XRef, YRef, ZRef, ReferenceScene, 'from_xyz_map', [], [])
* Online steps, performed for every scene:
object_model_3d_to_xyz (X, Y, Z, Scene, 'from_xyz_map', [], [])
sub_image (ZRef, Z, ImageSub, 1, 0)
```

2. Apply a threshold and reduce noise in order to receive a region of interest. Make sure your threshold is lower than the height of the wanted objects.

```
threshold (ImageSub, Region, 0.5, 1e+10)
connection (Region, ConnectedRegions)
* Remove noise based on shape features, e.g. area:
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 2000, 1e+10)
union1 (SelectedRegions, RegionUnion)
```

3. Reduce your scene by view, using your region of interest.

```
reduce_object_model_3d_by_view (RegionUnion, Scene, [], [], SceneReduced)
```

## 6.3 Remove Bin

The most common approach regarding the removal of the bin in the scene is locating the bin based on a meshed CAD model or isolating the bin from the background as above-mentioned and then to apply a threshold in order to remove the bin.

### 6.3.1 Remove Bin Based on a Meshed CAD Model

The bin removal based on a meshed CAD model can be realized based on the following steps:

1. Create a surface model with the meshed CAD model that models the bin (see figure 6.3 b)).

```
create_surface_model (Model, 0.03, 'model_invert_normals', 'true', SurfaceModelID)
```

2. Localize the bin.

```
find_surface_model (SurfaceModelID, Scene, 0.05, 0.2, 0, 'true',[], [], Pose, \
                    Score, SurfaceMatchingResultID)
```

3. Move the meshed CAD model into the scene.

```
rigid_trans_object_model_3d (Model, Pose, ObjectModel3DRigidTrans)
```

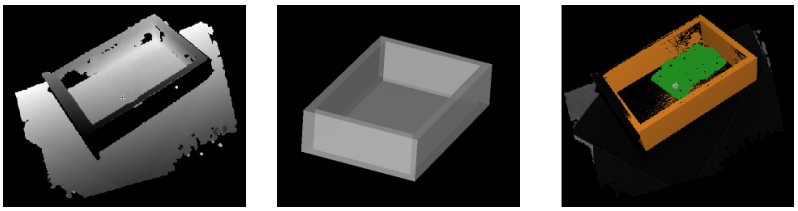
4. Reduce the scene to the bin's interior. Therefore, subtract the Z-mappings of the scene and the found CAD model, remove the background by applying a threshold and reduce the scene by view, considering the resulting region.

```
object_model_3d_to_xyz (XModel, YModel, ZModel, ObjectModel3DRigidTrans, \
                        'cartesian_faces', [CameraParam],[0,0,0,0,0,0,0])
object_model_3d_to_xyz (XScene, YScene, ZScene, Scene, 'from_xyz_map', [],[])
sub_image (ZModel, ZScene, ImageSub, 1, 0)
threshold (ImageSub, Regions, 0.001,1)
reduce_object_model_3d_by_view (Regions, Scene, [], [], SceneReduced)
```

### 6.3.2 Remove Bin Based on Thresholding

In case that no meshed CAD model of the bin is available, the following approach can be used in order to reduce the scene to the objects inside the bin:

1. Remove the background plane e.g. with a method described in section 6.1 on page 26.



a) Scene b) CAD model c) Located bin

Figure 6.3: The bin can be found in the scene and removed from it using a CAD model of the bin.

2. Estimate the extents of the bin. Now that the scene is reduced to the bin containing the objects to be localized, the bin is estimated approximately with respect to the scene coordinate system (see figure 6.4 b) ).

```
smallest_bounding_box_object_model_3d (Scene, 'oriented', BoundingBoxPose, \
                                       Length1, Length2, Length3)
gen_box_object_model_3d (BoundingBoxPose, Length1, Length2, Length3, BoundingBox)
```

3. Transform the bin into the center of the estimated smallest bounding box.

```
pose_invert (BoundingBoxPose, PoseInverted)
rigid_trans_object_model_3d (Scene, PoseInverted, SceneRigidTrans)
```

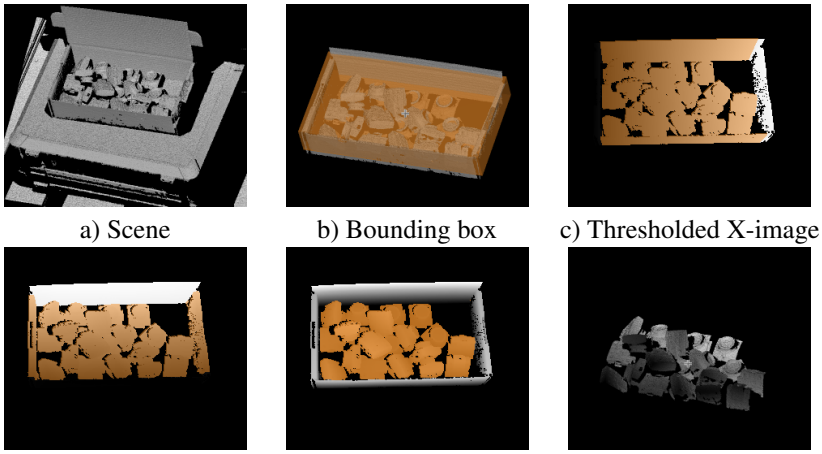
4. Apply a 2D threshold on the X and Y mappings (see figure 6.4 c) and d) ).

```
object_model_3d_to_xyz (X, Y, Z, SceneRigidTrans, 'from_xyz_map', [], [])
threshold (X, RegionX, -Length1 / 2 * 0.85, Length1 / 2 * 0.85)
threshold (Y, RegionY, -Length2 / 2 * 0.85, Length2 / 2 * 0.85)
```

   Alternatively, you can apply a threshold on X- and Y-coordinates using `select_points_object_model_3d`. In this case the next step can be skipped.

5. Compute the region of interest by intersecting the X- and Y-mappings and the domain of the scene without background (see the results in figure 6.4 e) and f) ).

```
intersection (RegionX, RegionY, RegionIntersection)
intersection (RegionIntersection, SceneDomain, ROI)
reduce_object_model_3d_by_view (ROI, Scene, 'xyz_mapping', [], SceneReduced)
```



a) Scene b) Bounding box c) Thresholded X-image

d) Thresholded Y-image e) Intersected regions f) Reduced scene

Figure 6.4: After removing the background plane the domain is reduced based on the 2D-mappings of the scene.