optonic
vision · automation

Handbook

mikado

Control

Version 3.4

A product of Optonic
www.mikado-robotics.de

Optonic GmbH | Zollhallenstr. 11 | 79106 Freiburg | Deutschland/Germany
Tel.: +49 761-79095-0 | E-Mail: info@optonic.com
© Copyright Optonic GmbH, 2015-2025

CONTENTS

1 Introduction **1**
- 1.1 Quick start . . . 1
- 1.2 The different tools . . . 1

2 Use **3**
- 2.1 Navigation . . . 3
- 2.2 Home page . . . 4
- 2.3 Operator page . . . 5
- 2.4 Setup page . . . 5
- 2.5 Status page . . . 5
  - 2.5.1 Control buttons . . . 5
  - 2.5.2 Program schedule . . . 6
- 2.6 Status bar . . . 6
  - 2.6.1 System status . . . 7
- 2.7 Password protection . . . 8
- 2.8 Program editor . . . 8
  - 2.8.1 Mikado Programs . . . 8
  - 2.8.2 Graphical user interface . . . 9
  - 2.8.3 Toolbar . . . 10
  - 2.8.4 Create program . . . 11
  - 2.8.5 Programm-Validator . . . 11
- 2.9 Variable editor . . . 12
  - 2.9.1 Variablentabelle . . . 12
  - 2.9.2 Editing variables . . . 13
  - 2.9.3 Adding and filtering variables . . . 16
  - 2.9.4 Import variables from file . . . 17
  - 2.9.5 Visualization of the robot pose . . . 17
- 2.10 Product management . . . 20
  - 2.10.1 Overview . . . 20
  - 2.10.2 Name . . . 21
  - 2.10.3 Active product . . . 21
  - 2.10.4 Object search . . . 21
  - 2.10.5 Gripping poses . . . 24
  - 2.10.6 Collision Check for Grasp-Poses . . . 27
- 2.11 Greifpose . . . 30
  - 2.11.1 Overview . . . 30
  - 2.11.2 Gripping process . . . 30
  - 2.11.3 Gripping motion search . . . 32
- 2.12 Scene management . . . 42
  - 2.12.1 Overview . . . 42
  - 2.12.2 Obstacle . . . 43
  - 2.12.3 Box . . . 43
  - 2.12.4 Region of Interest (ROI) . . . 45
- 2.13 Object recognition tool . . . 46

2.13.1 Prerequisites . . . 46
2.13.2 First steps . . . 47
2.13.3 Camera selection . . . 47
2.13.4 Save last recording . . . 47
2.13.5 Visualization of the scene . . . 48
2.13.6 Halcon-specific object recognition parameters . . . 49
2.14 Handle inspection . . . 51
2.14.1 Overview . . . 51
2.14.2 Inspection of the gripping movement search . . . 51
2.15 Best handles . . . 54
2.15.1 Prioritization . . . 54
2.15.2 Best handles display . . . 66
2.16 Profiler . . . 67
2.17 Logs . . . 67
2.17.1 Overview . . . 67
2.18 Camera settings . . . 68
2.18.1 Live image 2D . . . 68
2.18.2 Camera tab . . . 69
2.18.3 Hand-eye calibration . . . 70
2.19 Robot control . . . 74
2.19.1 Direct movements . . . 74
2.19.2 Commissioning mode . . . 75
2.19.3 Gripper . . . 75
2.19.4 Move . . . 76
2.19.5 Robot parameters . . . 76
2.20 Robot tracking . . . 76
2.21 Overview of robot inputs and outputs . . . 77
2.21.1 Adding/removing I/Os . . . 78
2.21.2 Setting outputs and labeling the I/Os . . . 78
2.22 System-Tools . . . 79
2.22.1 System . . . 79
2.22.2 Mikado . . . 80
2.22.3 Developer tools . . . 80
2.22.4 Profiling . . . 80

3 Tasks **83**
3.1 Procedure . . . 83
3.1.1 Switch to sequence . . . 83
3.1.2 If-Group . . . 83
3.1.3 Exit program . . . 85
3.1.4 Error . . . 85
3.1.5 Wait . . . 85
3.1.6 Waiting until standstill . . . 85
3.1.7 Interrupt program . . . 86
3.2 Numbers . . . 86
3.2.1 Increase number . . . 86
3.2.2 Reduce number . . . 86
3.2.3 Arithmetic operation . . . 87
3.2.4 Logical operation . . . 87
3.2.5 Comparison operation . . . 89
3.3 Fundamentals . . . 90
3.3.1 Copy . . . 90
3.3.2 Swap values . . . 91
3.3.3 Reset variable . . . 91
3.3.4 Reset all variables . . . 91
3.3.5 Set initial value . . . 91
3.3.6 Negate Boolean variable . . . 92
3.3.7 Copy entry from list . . . 92

3.3.8 Copy next entry from list . . . 92
3.3.9 Add to list . . . 93
3.3.10 Set list element . . . 93
3.3.11 Delete element from list . . . 93
3.3.12 Liste leeren . . . 94
3.3.13 Liste leather? . . . 94
3.3.14 Count list elements . . . 94
3.4 Robot control . . . 95
3.4.1 Move robot to . . . 95
3.4.2 Move robot Cartesian to . . . 95
3.4.3 Move Robot relatively . . . 96
3.4.4 Move Robot Cartesian relatively . . . 97
3.4.5 Move Tcp to . . . 98
3.4.6 Move Tcp Cartesian to . . . 98
3.4.7 Move Tcp relatively . . . 99
3.4.8 Move Tcp Cartesian relatively . . . 100
3.4.9 Move part to . . . 101
3.4.10 Move part Cartesian to . . . 102
3.4.11 Move Piece relatively . . . 103
3.4.12 Move Piece Cartesian relatively . . . 104
3.4.13 Read robot position . . . 104
3.4.14 Read part position . . . 105
3.4.15 Read TCP position . . . 105
3.4.16 Is robot position accessible with part? . . . 106
3.4.17 Is robot position reachable? . . . 106
3.4.18 Open/close hand . . . 106
3.4.19 Setting the gripper status . . . 107
3.4.20 Set active gripper . . . 109
3.4.21 Execute robot program . . . 109
3.4.22 Set robot output signal . . . 110
3.4.23 Read robot output signal . . . 111
3.4.24 Read robot input signal . . . 111
3.4.25 Wait for robot input signal . . . 111
3.4.26 Wait for robot output signal . . . 112
3.4.27 Wait for fieldbus input signal . . . 112
3.4.28 Set Fieldbus Output Signal Async . . . 113
3.5 Image processing . . . 114
3.5.1 Set active camera . . . 114
3.5.2 Load camera configuration . . . 114
3.5.3 Kalibrierung laden camera . . . 114
3.5.4 Record point cloud . . . 115
3.5.5 Record point cloud asynchronously . . . 115
3.5.6 Capture for Series . . . 116
3.5.7 Readjusting the camera pose . . . 116
3.5.8 Determine the number of points . . . 117
3.5.9 Save image . . . 118
3.5.10 Set active ROI . . . 119
3.5.11 Determine pattern pose . . . 119
3.6 Calibration . . . 120
3.6.1 Start calibration . . . 120
3.6.2 Recording for calibration . . . 120
3.6.3 Finalize calibration . . . 120
3.6.4 Validate hand-eye calibration . . . 121
3.7 Scene object . . . 124
3.7.1 Set active box . . . 124
3.7.2 Search scene object . . . 125
3.7.3 Read scene object position . . . 126
3.7.4 Set scene object position . . . 126

3.7.5 Reset scene object position . . . 127
3.8 Object handling . . . 127
3.8.1 Set active product . . . 127
3.8.2 Start handle search . . . 127
3.8.3 Cancel handle search . . . 128
3.8.4 Find object . . . 128
3.8.5 Get Picks . . . 129
3.8.6 Pick object . . . 131
3.8.7 Pick-Search-Statistics . . . 133
3.8.8 Discard part . . . 135
3.8.9 Clear Recognized Objects and Pointcloud . . . 135
3.8.10 Add static object . . . 135
3.8.11 Remove static object . . . 136
3.8.12 Attach static object . . . 136
3.8.13 Read out product dimensions . . . 137
3.8.14 Adjust Object Pose . . . 137
3.9 Transformations . . . 138
3.9.1 Split pose . . . 138
3.9.2 Create pose . . . 139
3.9.3 Rotate Pose Global . . . 139
3.9.4 Rotating pose Local . . . 141
3.9.5 Move Pose Global . . . 143
3.9.6 Move Pose Local . . . 145
3.9.7 Pose transformieren . . . 146
3.9.8 Inverse Transformation . . . 147
3.10 Extensions . . . 147
3.10.1 JavaScript . . . 147
3.10.2 Execute Halcon Script . . . 147

4 Tutorials **149**
4.1 An initial process in ten minutes . . . 149
4.1.1 Set up recording pose . . . 149
4.1.2 Set up scene . . . 149
4.1.3 Set up product . . . 149
4.1.4 Save product . . . 150
4.1.5 Create minimum program . . . 150
4.1.6 Teach in poses and adjust variables . . . 150
4.1.7 Let it run and vary a little . . . 150
4.2 A new product up to object recognition . . . 152
4.2.1 Set up new product / product list . . . 152
4.2.2 Optimize object recognition . . . 154
4.3 Meshlab . . . 156
4.3.1 Basics: Loading and saving STL . . . 156
4.3.2 Center product coordinate system . . . 157
4.3.3 Scale STL . . . 159
4.3.4 STL Reduce number of triangles . . . 161
4.3.5 Remove STL surfaces . . . 163
4.3.6 STL Automatically remove internal surfaces . . . 168
4.4 Integrate Halcon Script: Data matrix recognition . . . 173
4.4.1 Preparation . . . 173
4.4.2 Program schedule . . . 174
4.4.3 Create Hdev script for the recognition of data matrices / QR codes . . . 175
4.4.4 Execute and test program . . . 177
4.5 Example: User-defined log file from Mikado . . . 180
4.5.1 Suitable Mikado program . . . 180
4.5.2 Variables of the Mikado program . . . 181
4.5.3 Script details . . . 181
4.6 Example: Sort Mikado PoseList . . . 182

5 Simulation **183**
5.1 Roboter-Simulation . . . 183
5.2 Recorded image data (Filecam) . . . 183
5.3 Simulation based on mesh files . . . 184
5.3.1 Simulator - Configuration . . . 184
5.3.2 Simulator - Automatic mode . . . 185
5.3.3 Simulator - Adjusting the camera settings with scene from Simulator . . . 186

6 Robot models **193**
6.1 Set up ABB Controller . . . 193
6.1.1 System requirements . . . 193
6.1.2 Network configuration . . . 194
6.1.3 Installing the programs . . . 194
6.1.4 Setting up the programs . . . 196
6.1.5 Updating the robot programs . . . 198
6.1.6 Robot Control . . . 198
6.1.7 Configuring gripper and IO with Robot Web Services . . . 199
6.1.8 Giving up control over the robot temporarily . . . 200
6.1.9 Troubleshooting . . . 202
6.2 Set up Doosan Controller . . . 203
6.2.1 Network setup . . . 203
6.2.2 General notes . . . 203
6.2.3 Automatic mode change . . . 203
6.2.4 Query the name of the current tool . . . 204
6.2.5 Query the name of the current tcp . . . 204
6.2.6 Sources . . . 204
6.3 Set up Fanuc Controller . . . 204
6.3.1 Prerequisites . . . 204
6.3.2 Network configuration . . . 205
6.3.3 Installing the Mikado interface . . . 206
6.3.4 Parameterizing the Mikado interface . . . 206
6.3.5 Start system . . . 207
6.3.6 Optional: Gripper control via I/Os . . . 207
6.3.7 Optional: Adjust robot speed in Mikado . . . 207
6.3.8 Optional: Fine-tune movement parameters . . . 207
6.3.9 Optional: Execute robot program . . . 208
6.3.10 Optional: Give up robot control . . . 208
6.3.11 Troubleshooting . . . 209
6.3.12 Testing the connection . . . 209
6.4 Set up KUKA Controller . . . 210
6.4.1 General . . . 210
6.4.2 System requirements . . . 210
6.4.3 Network configuration . . . 211
6.4.4 KSS: Prepare KRC . . . 212
6.4.5 iiQKA.OS2: Prepare KRC . . . 214
6.4.6 Optional: Execute native robot program . . . 220
6.4.7 Transfer robot control . . . 221
6.4.8 Speed and fine-tuning . . . 221
6.4.9 Troubleshooting . . . 222
6.5 Set up Mitsubishi Controller . . . 222
6.5.1 Network configuration . . . 223
6.5.2 Configure connection . . . 229
6.5.3 Configure programs . . . 231
6.5.4 Troubleshooting: Verbindung testen . . . 235
6.5.5 Optional: Manual control . . . 237
6.5.6 Optional: Adjust base coordinate system . . . 237
6.6 Using Mitsubishi robots . . . 239
6.6.1 Parameters and controls . . . 239

6.6.2 Turning on the robot . . . 240
6.6.3 Other notes . . . 240
6.7 Set up Nachi Controller . . . 240
6.7.1 Network configuration . . . 240
6.7.2 Preparation . . . 240
6.7.3 Copying the programs/user tasks . . . 243
6.7.4 Compiling the programs/user tasks . . . 243
6.7.5 Setting the priority of the user tasks . . . 243
6.7.6 Starting the robot program . . . 244
6.8 Set up Stäubli Controller . . . 244
6.8.1 Network configuration . . . 244
6.8.2 Installation . . . 244
6.8.3 Set up server sockets . . . 245
6.8.4 Start . . . 245
6.8.5 Set up grippers and IOs . . . 245
6.9 Set up Techman Controller . . . 245
6.9.1 Network configuration . . . 245
6.9.2 Establish Modbus connection . . . 246
6.9.3 Create project . . . 246
6.9.4 Test project . . . 247
6.9.5 Set project speed and activate automatic mode . . . 248
6.9.6 Troubleshooting . . . 249
6.9.7 Writing Techman variables and executing programs . . . 249
6.10 Set up Universal Robots Controller . . . 249
6.10.1 System requirements . . . 249
6.10.2 Network configuration . . . 249
6.10.3 Setting up the communication program . . . 250
6.10.4 Set up autostart . . . 252
6.10.5 Joint angle limits . . . 253
6.10.6 Speed and acceleration . . . 253
6.10.7 Extracting and using robot calibration data . . . 253
6.10.8 Optional: Execute native robot program . . . 254
6.11 Using UR robots . . . 256
6.11.1 Turning on the robot . . . 256
6.11.2 Using URCap grippers . . . 257
6.11.3 Parameters and controls . . . 259
6.11.4 Other notes . . . 259
6.11.5 Troubleshooting Universal Robot . . . 259
6.12 Setting up the Yaskawa Motoman Controller . . . 262
6.12.1 Preparation . . . 262
6.12.2 Network configuration . . . 263
6.12.3 Setting up the robot controller . . . 264
6.12.4 Weiteres & Troubleshooting . . . 267
6.12.5 Enter acceleration limits in joint_limits.yaml . . . 268
6.12.6 Error codes / Alarms . . . 268
6.12.7 Giving up control over the robot temporarily . . . 270
6.12.8 Running other INFORM jobs on the robot . . . 270
6.12.9 Call INIT_ROS and RET to the caller . . . 271
6.12.10 Setting of the required outputs without INIT_ROS . . . 271
6.12.11 INIT_ROS as default job, other jobs might get run using CALL JOB . . . 271
6.12.12 Reading and writing 16-bit registers . . . 272

7 Installation **273**
7.1 Overview . . . 273
7.1.1 Hardware overview . . . 273
7.1.2 Software overview . . . 274
7.1.3 Integration procedure (step-by-step instructions) . . . 275
7.2 Configuration Backup . . . 277

7.3 Create a new cell . . . 277
7.3.1 Create new Mikado Control cell . . . 277
7.4 Customize an existing cell . . . 278
7.4.1 Speed settings . . . 278
7.4.2 Configuration files . . . 281
7.4.3 Launchfiles . . . 288
7.4.4 Cell geometry . . . 289
7.5 Directory structure . . . 299
7.6 Mikado Select/Control assembly guideline . . . 300
7.6.1 Mikado Controller Hardware . . . 300
7.6.2 Connecting the Ensenso cameras . . . 300
7.6.3 Fastening the calibration plate . . . 301
7.7 Gripper integration . . . 302
7.7.1 Gripper types . . . 302
7.7.2 Integrate gripper . . . 303
7.7.3 Mikado configuration . . . 307
7.7.4 Gripper connection variants . . . 307
7.7.5 Calibrate TCP . . . 307
7.8 Set up hardware . . . 309
7.8.1 Overview of connections/cabling . . . 309
7.8.2 Display connection, operation without connected display . . . 311
7.8.3 Network adapter . . . 311
7.8.4 Connection check . . . 313
7.8.5 Set up Ensenso cameras . . . 314
7.8.6 Setting up the robot controller . . . 318
7.9 Fieldbus interface . . . 318
7.9.1 Brief description of the interface . . . 318
7.9.2 General communication process . . . 318
7.9.3 Communication status . . . 320
7.9.4 Synchronizing PLC variables and Mikado variables . . . 321
7.9.5 Rotation conventions . . . 322
7.9.6 Error codes . . . 323
7.9.7 Commands . . . 325
7.9.8 Example sequence for select cells . . . 328
7.9.9 Updates of the fieldbus interface . . . 329
7.9.10 Standard Fieldbus Interface . . . 330
7.9.11 Specific configuration for Profinet . . . 332
7.9.12 Spezifische Konfiguration für EtherCAT . . . 332
7.9.13 Spezifische Konfiguration für EtherNet/IP . . . 333
7.9.14 Specific configuration for Beckhoff ADS interface . . . 333
7.10 About the operating system . . . 335

8 Extending Mikado Control **337**
8.1 JavaScript Task . . . 337
8.1.1 General information . . . 337
8.1.2 `task` . . . 338
8.1.3 `dataStore` . . . 338
8.1.4 `picksearch` . . . 338
8.1.5 `kinematics` . . . 344
8.1.6 `product` . . . 346
8.1.7 `robot` . . . 348
8.1.8 `scene` . . . 351
8.1.9 `file` . . . 352
8.1.10 `program` . . . 353
8.1.11 `lab` . . . 353
8.1.12 `pointCloud` . . . 365
8.2 Halcon Script Task . . . 367
8.2.1 mikado_get_2d_image . . . 367

8.2.2 mikado_publish_2d_image . . . 368
8.2.3 mikado_get_point_cloud . . . 368
8.2.4 mikado_write_variable . . . 368
8.2.5 mikado_read_variable . . . 369
8.2.6 ensenso_set_get_json . . . 370
8.3 User-defined object recognition with Halcon . . . 372
8.3.1 Hdev script . . . 372
8.3.2 Customizing the configuration file . . . 373
8.4 Own pages with QML . . . 374
8.4.1 Konfiguration . . . 374
8.4.2 Introduction to QML . . . 374
8.4.3 Further information . . . 375
8.5 External control with ROS/Python . . . 376
8.5.1 Example program . . . 376
8.5.2 Mikado ROS Services . . . 376
8.5.3 Mikado ROS Actions . . . 383
8.5.4 Systemstatus . . . 385
8.5.5 Create custom ROS Service . . . 385
8.5.6 Launchfile . . . 387

9 Updates **389**
9.1 Mikado . . . 389
9.1.1 Version 3.4.0 . . . 389
9.1.2 Version 3.3.0 . . . 391
9.1.3 Version 3.2.0 . . . 399
9.1.4 Version 3.1.2 . . . 401
9.1.5 Version 3.0.5 . . . 405
9.1.6 Version 3.0.4 . . . 405
9.1.7 Version 3.0.3 . . . 406
9.1.8 Version 3.0.2 . . . 406
9.1.9 Version 3.0.1 . . . 406
9.1.10 Version 3.0.0 . . . 406
9.1.11 Version 2.1.0 . . . 411
9.2 Scene objects . . . 412
9.2.1 Version 5 . . . 412
9.2.2 Version 4 . . . 414
9.2.3 Version 3 . . . 414
9.2.4 Version 2 . . . 414

10 History of changes **415**

11 Copyright **417**
11.1 GNU LESSER GENERAL PUBLIC LICENSE . . . 417
11.2 BSD 3-Clause License . . . 419

CHAPTER

ONE

INTRODUCTION

Mikado Control is a graphical software for bin picking. It bundles solutions for all sub-problems from image acquisition and object recognition to path planning, robot control and communication with external controllers. Despite these sometimes complex tasks, Mikado Control provides you with intuitive and easily configurable access to your bin picking application. The use as a pure sensor system for position determination, e.g. for depalletizing, is also a field of application for Mikado and is marketed as Mikado Select.

This documentation will help you to operate the Mikado Controller and understand how it works. The chapters *Use*, *Tasks* and *Simulation* are important for using an existing system, as are the practical exercises in the chapter *Tutorials*.

The chapter *Installation* guides you through the setup of a new cell. The section *Integration procedure (step-by-step instructions)* provides a step-by-step overview of the setup steps. The installation of the hardware components is described in the section *Mikado Select/Control assembly guideline*.

The setup of the supported robot models is described manufacturer-specifically in the *Robot models* section. The section *Extending Mikado Control* shows the available interfaces and extension options of Mikado.

1.1 Quick start

The quickest way to get into the Mikado interface is to use one of the supplied cells with simulated robot movement and saved or simulated camera recordings. To do this, start your controller and make sure that the license dongles are plugged in. There will already be a shortcut on the desktop to open one of the ready-made cells in the Mikado program, e.g. `motek_demo_2017`, and a shortcut for the corresponding camera simulation next to it. First start Mikado and then the simulator. You will now see a fully set up cell with scene, product and program in which you can practice your first steps.

1.2 The different tools

The Mikado controller is supplied with various software tools. This is the core Mikado program, which is linked once on the desktop for each cell definition on the controller. These links are created automatically for each newly created cell. If a controller goes into continuous operation, Mikado can be executed directly with a selected cell definition via Autostart.

In addition to the Mikado link, a simulator link is also created for each cell definition, for more details see chapter *Simulation*.

To create a new cell, use the Mikado Assistant. You will find this in the taskbar or in the start menu after entering `Assistant`. The exact steps are documented in the chapter *Create a new cell*. You can also use the Assistant to make in-depth adjustments to existing cells. The various options can be found in the chapter *Customize an existing cell*.

You can run the NxView program supplied for configuring the Ensenso camera from the taskbar or the Start menu. If the network configuration is correct, all accessible Ensenso cameras are listed here. Please note that only one application has access to one camera at a time, i.e. either Mikado or NxView. The operating instructions for the camera can be found in the Ensenso Manual.

CHAPTER

TWO

USE

This chapter explains how to use Mikado Control and how to create your own workflows. In addition, all tools of the Mikado interface are explained.

2.1 Navigation

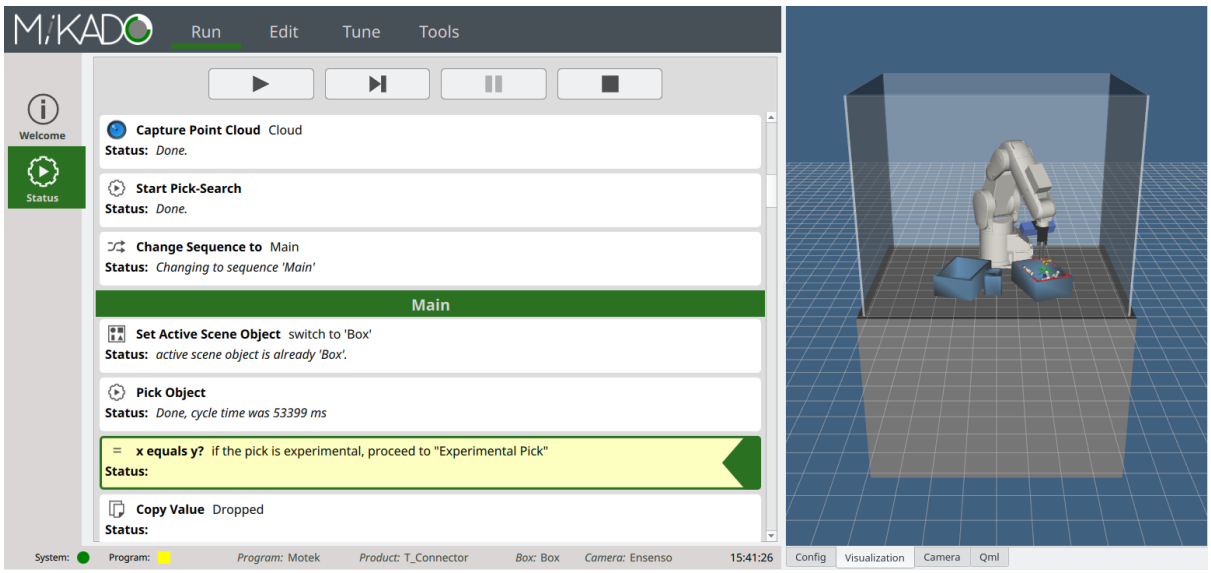


Fig. 1: Mikado interface with status page

The user interface is divided into five areas:

- **Above:** ***Main navigation bar***
  Displays all main categories, which in turn contain the individual navigation pages. Click on one of the categories and the category pages will be expanded. Now select the desired page to switch to it. The following main categories are available:
  - *Run*: Contains pages that are relevant during operation (status monitoring and individual operator pages).
  - *Edit*: Allows you to set up processes and manage products and scenes.
  - *Tune*: Used to improve object recognition, handle definitions or performance.
  - *Tools*: Other tools such as logs, robot control, camera tools, etc.

  If your system requires password protection, a lock symbol will appear on the right-hand side of the main navigation bar, which you can use to log in and out, see chapter *Password protection*.

Hint: You can use the function keys F1 - F12 to switch directly between the pages (F1: Status, F2: Program editor, ...)

- **Links:** ***Navigation pages***
  Displays all navigation pages of the current main category. Here you can quickly switch to another page within the category.
- **Center:** ***Main window***
  Contents of the currently selected page are displayed in this main area of Mikado Control.
- **Right:** ***Visualization***
  Display of robot and cell. In this view, content is also displayed depending on the currently selected page (e.g. product, handles, point cloud, findings, etc.). The 3D view within the visualization can be navigated with the mouse:
  - Left mouse button: Rotate
  - Right mouse button: Zoom
  - Middle mouse button: Move

  While the corresponding mouse button is pressed, the view can be changed by moving the mouse. While the mouse button is held down, a yellow transparent disc can be seen in the visualization scene, which represents the current zoom or rotation center point.

  Various tabs can be selected below the visualization. In the 'Config' tab, the visualization can be adjusted as required. For example, point clouds can be enlarged or colored here. Buttons for saving and resetting the current view can also be found at the bottom of the 'Config' tab. When saving, only the current view is saved (e.g. status view of the scene or product view, depending on which Mikado page you are currently on). The saved view is loaded when you start Mikado or by clicking the 'Reset' button.

  The 'Camera' tab shows the last image taken by the camera(s) (details on the camera settings under *Camera settings*).
- **Below:** ***Status bar***
  Information on the current status of the program sequence (see *Status bar*).

2.2 Home page

On the welcome page of Mikado Control you can load your products and program sequences via quick access. All you need to do is click on the corresponding list entry.

New products or programs that you create in Mikado Control are automatically added to the lists the next time you start the Mikado software. All you have to do is save the product/program in the standard directory for products/programs.

Note: The default directory is preselected when a product/program is saved. Further information on the directory structure can be found in the section *Directory structure*.

2.3 Operator page

The ‘Operator’ page is used to implement individual or project-specific requirements for the Mikado| interface. Control elements such as input fields and buttons that are necessary or helpful for an individual application are displayed on this page. If no user page is configured, it is not displayed in the navigation. Otherwise, it can be found below the welcome page in the ‘Run’ menu.

2.4 Setup page

The ‘Setup’ page can be customized. The documentation for the *Operator page* applies accordingly to the setup page.

If password protection is activated, this page must be unlocked before it can be accessed. This means, for example, that setting options for application-specific parameters can be offered to a restricted group of users.

2.5 Status page

The status page is the central view for an operator. The program sequence can be controlled here and the current status of the executed task is displayed. (Screenshot of the Mikado interface with status page, see above).

2.5.1 Control buttons

The following buttons can be used to control the program sequence:

- ***Start***
  Start program execution. If the program was paused, execution is continued at the paused point
- ***Execute next task***
  The next step is executed, after which the program sequence pauses. This allows you to execute the sequence step by step, e.g. to test a new program.
- ***Pause***
  The program sequence is stopped, but the program status is retained and can be continued.
- ***Stop and reset***
  Stops the program sequence and resets the program status. All variables are reset to the initial state and the program cursor is set to the first task.



Fig. 2: Control buttons (from left: Start, Next task, Pause, Stop)

Note: Buttons are deactivated to prevent incorrect entries. For example, the pause button cannot be pressed if the program sequence has not yet been started. If a button is deactivated, the symbol is displayed in light gray.

2.5.2 Program schedule

The central part of the status overview can be seen below the control buttons. All sequences and their tasks of the current Mikado program are displayed here. Sequences are shown as green boxes with sequence names and group the tasks (white boxes) underneath them.

You will find any user comments assigned next to the task type. If a task has already been executed, you will see the result of the last execution of this task in the 'Status' field. If the task is currently being executed, the displayed box turns light green and the 'Status' field shows the execution status.

If the execution of the program has been paused, the following task, which will be executed next when the program is resumed, is marked with the program cursor and colored yellow. The program cursor is displayed as a green frame.

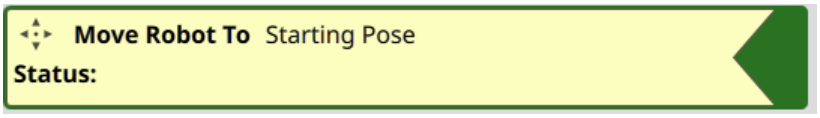


Fig. 3: Program cursor

In the event of an error, the program sequence is aborted and the task that reported the error is colored red or orange. In this case, an error message is displayed in the 'Status' field. Some tasks support the option to retry the program. In this case, the task is not colored red but orange. Examples of this are the move tasks in '*Robot control*' or *Record point cloud*. To obtain further information on an error, it is advisable to take a look at the *Logs*.

Task visualization

The colored representations of a task on the status page have the following meanings:

- *white*

  Inactive. Task has not yet been executed or has been executed successfully.

- *green*

  In execution.

- *yellow*

  This task is executed next when the process is continued.

- *orange*

  An error has occurred during the execution of the task, but it is possible to try again. To do this, continue the process.

- *rot*

  An error has occurred. The process must be stopped and restarted to continue.

2.6 Status bar

The status bar is an integral part of the Mikado interface and is displayed at the bottom of every view.

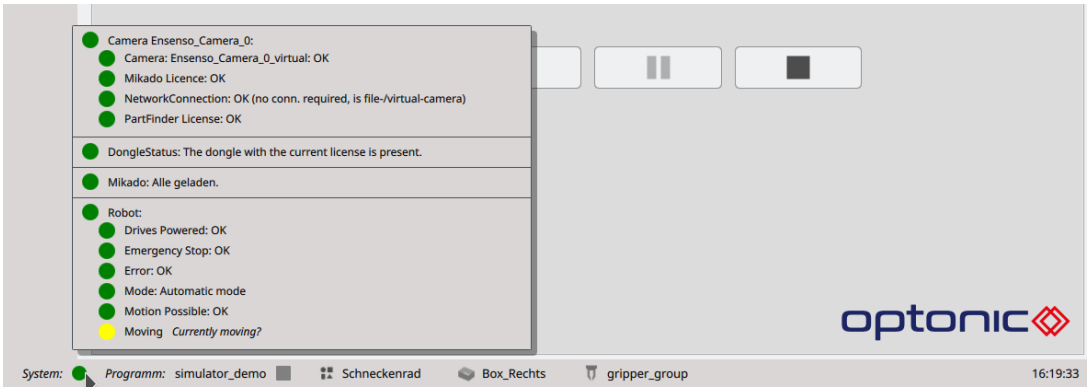


Fig. 4: Status bar and system status

The following displays can be seen in the status bar (from the left):

- **System status**
  See *System status*.
- **Program**
  Loaded Mikado program. This can be modified and managed using *Program editor*.
- **Product**
  Active product. The active product can be changed and set via the *Product management* (see *Active product*)
- **Box**
  Active box. The active crate can change during program execution (see *Set active box*). You can see the currently active crate in this area of the status bar.
- **Camera**
  Active camera. The active camera can change during program execution (see *Set active camera*). You can see the currently active camera in this area of the status bar.
- **Time**
  The current time.

2.6.1 System status

The system status serves as an indicator of whether all the hardware components of your system are working correctly. If all configured components such as **robot**, **camera systems**, **license dongles** or **fieldbus systems** are working correctly, this status indicator turns **green** and Mikado can be considered **ready**. However, if Mikado Control detects an impermissible status in the system, the status indicator will turn **red** and your program sequence may not function correctly.

If you would like to see a more detailed breakdown of the status of the individual components, move your mouse pointer over the status indicator. An overview of the individual components now appears, each with a status indicator and comment.

2.7 Password protection

For cells with activated password protection, all Gui elements that would allow changes to the cell modeling or flow control are only released after the correct password has been entered.

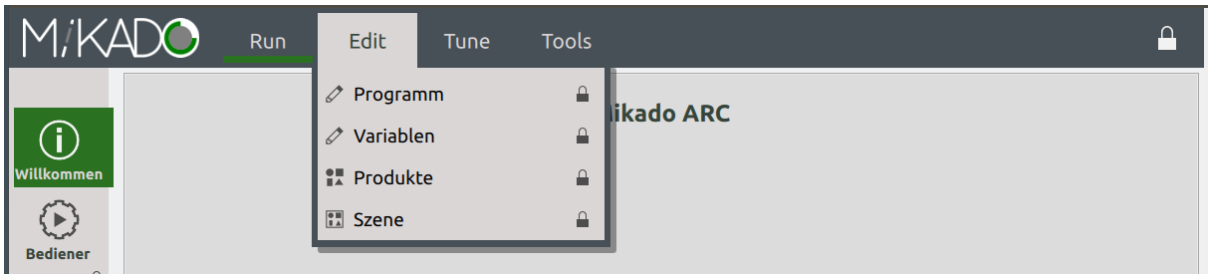


Fig. 5: Screenshot menu with activated password protection

Unlocked areas in the menu are indicated by small lock symbols. To unlock, click on the large lock symbol in the menu bar at the top right and enter the password in the following dialog.

Activating access control and setting the password is described in *access_control.yaml*.

2.8 Program editor

The program editor provides the necessary tools to create a Mikado program sequence. It can be found via the main menu under 'Edit' → 'Program'.

2.8.1 Mikado Programs

A Mikado program consists of sequences, tasks and variables. Together, these form an executable program that performs tasks such as reaching into a box. Products and scene objects that can be used in the program are managed independently of the program.

Sequences

A Mikado program sequence takes place in different sequences. The program starts with the first sequence and repeats this again and again. A sequence change or transition takes place when a task triggers this. An example of this is the task '*Switch to sequence*', which stops the execution of the active sequence and switches to another sequence. Another example of a sequence change are tasks that jump to another sequence under certain conditions.

Sequences are created and edited using the program editor. They must be clearly named.

Tasks

Tasks are the basic building blocks of a Mikado program and are always part of a sequence. When the execution of a sequence starts, it begins with the first task. The tasks are then processed one after the other, with each task fulfilling a task. Tasks are parameterized using variables. Variables can also be changed by a task.

A task can also have sequence changes. These are listed below a hyphen. If a corresponding condition occurs, the task can trigger a sequence change.

Tasks are added to sequences and edited using the program editor. Both variables and sequence changes are configured via the interface.

All available tasks are listed and explained in the *Tasks* chapter.

Hint: Both variables and sequence changes are not mandatory for every task.

Variables

Variables represent the dynamic part of the Mikado program. They are read in or modified by tasks. If a program is restarted, all variable values are reset to the defined initial value, but many variable values are modified by the tasks during execution. Further information on variables can be found in the *Variable editor* section.

2.8.2 Graphical user interface

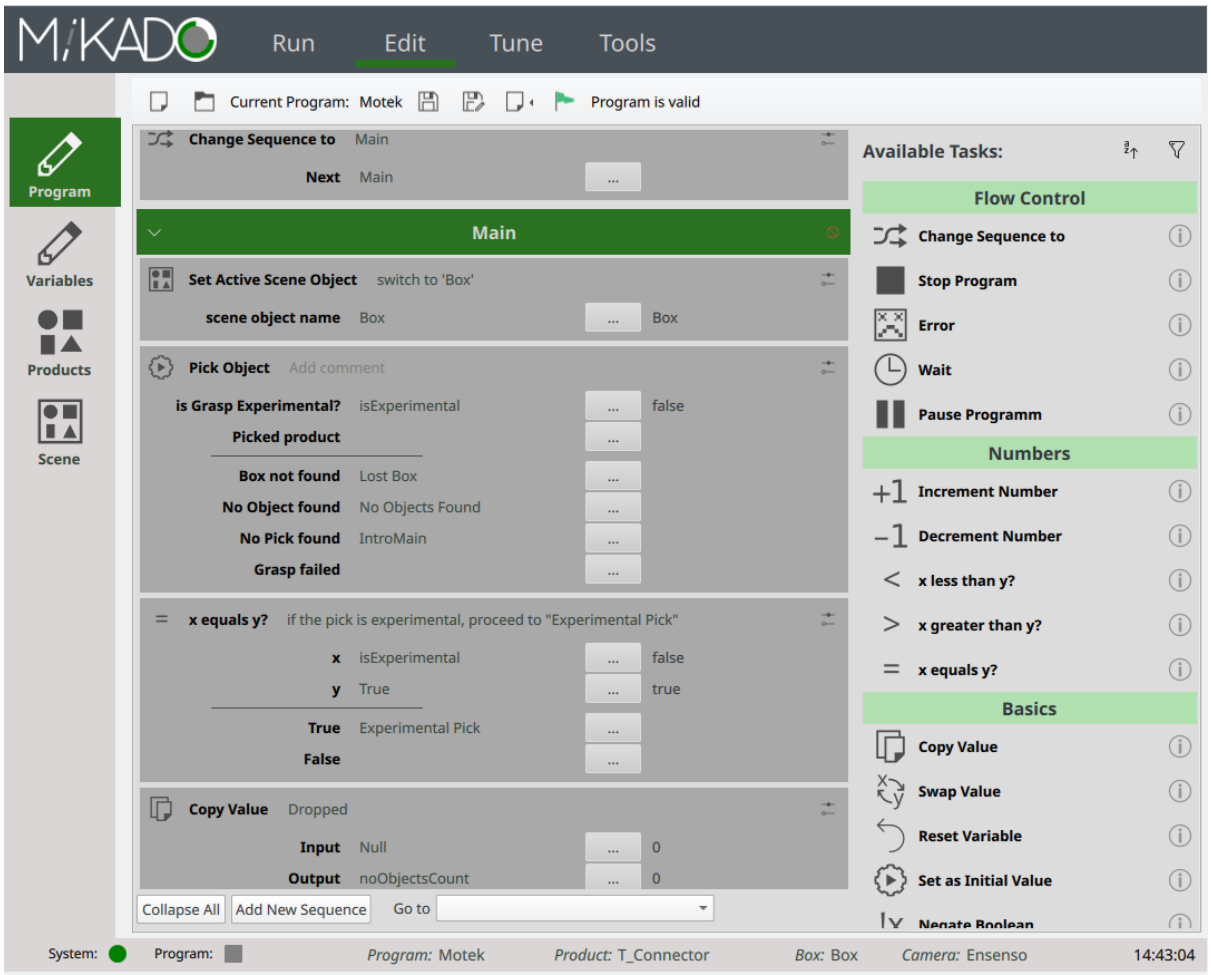


Fig. 6: Program editor

The program editor interface consists of the following areas:

- **Toolbar** (top bar)

  See *Toolbar*

- **Editor** (left area)

  Interface to compile the Mikado program

- **Task library** (right bar)

List with all available task types. The tasks are listed by category. Click on the info icon to display a brief description. To add a task to the current Mikado program, drag and drop the desired task to the desired position.

  - Alphabetical sorting: If this button is activated, the categories are hidden and all tasks are listed in alphabetical order.
  - Filter: If this button is activated, an input field is displayed which can be used to filter the available tasks.

- **Sequence bar** (bottom bar)

  The sequence bar is located below the editor:

  - *Collapse all / Expand all*

    Folds all sequences in/out. If all sequences are collapsed, you will see a clear arrangement of the sequences. In this view, you can also move the sequences using 'Drag & Drop' with the help of the now visible handles.

  - *Add sequence*

    Create new sequence.

  - *Jump to*

    Select a sequence here to jump to it directly in the editor. This allows you to navigate more quickly in extensive programs.

2.8.3 Toolbar



Fig. 7: Toolbar of the program editor

The toolbar is located in the upper area with the following elements (from left to right):

- **New program**

  Discards the currently loaded program and creates a new one.

Warning: Unsaved changes will be lost.

- **Open program**

  Opens a dialog with which an existing Mikado program can be loaded. The variables of the selected program are also loaded.

Warning: Unsaved changes will be lost.

- **Current program**

  The program name is displayed in this field.

- **Save program**

  Saves all changes to sequences, tasks and variables.

- **Save program as …**

  Saves all changes to sequences, tasks and variables in a new file.

- **Reset all variables to their initial value**

  Click on this button to reset the values of all variables to their initial value. This does not stop the current Mikado program.

- **Check the program for errors**

  See *Programm-Validator*.

2.8.4 Create program

To create a new sequence, click on the 'Add sequence' button in the lower sequence bar. A new sequence is appended to the current program. Clicking on the title of the sequence (by default "New sequence") allows you to change the sequence name. The buttons in the sequence heading can also be used to collapse or delete the sequences.

Sequences can be put together using 'drag & drop'. Select a task from the task list on the right and, holding down the mouse button, drag it to the position where you want to insert it. An instance of the selected task appears at the desired location. You can now optionally add a comment to the task. Depending on the task, you can or must assign variables and sequences for sequence changes to the task. If a mandatory variable is missing, this is indicated by a note. Further information can also be found in the task description.

A line is displayed in the task for each variable/sequence change. Here you can see the name of the variable or sequence change, followed by the current selection. The button for each entry opens a selection dialog with which you can select an existing variable/existing sequence change. In this dialog, there is also a text field that can be used to create a new variable/sequence. Finally, the current value of the selected variable is displayed next to the button.

Tasks have a meta button next to these elements at the top right, which can be used to display an extended view. In this view, tasks can be moved, duplicated and deleted. Moving or duplicating a task is also done by 'drag & drop'.

Hint: It's quicker this way:

- Click on the variable name: Direct entry of the variable name. If this does not exist, a new variable is created.
- Click on the variable value: Opens a quick dialog in which the variable properties can be edited.

Hint: Don't regret anything: If you delete a task or a sequence, the action will be executed with a delay of a few seconds. This gives you one last opportunity to undo the action if it was not intended!

2.8.5 Programm-Validator

The current Mikado program is automatically checked for some errors using the program validator. The result is indicated by a green or red flag in the toolbar. In the event of an error, the number of misconfigured tasks is also displayed. On the one hand, sequence changes and variables are checked for existence and on the other hand, it is checked whether the types of the task variables are correct. Some tasks have special conditions, e.g. only a string can be appended to a StringList, but not a pose. You will find more detailed information on troubleshooting either in a summary dialog when you click on the flag or in red in the preview area of the individual tasks.

This validation also takes place automatically before each start of a Mikado program and displays a message if the validation fails. If necessary, the program can then be executed despite the error until the first erroneous task is reached.

2.9 Variable editor

An overview of all variables can be found under the ‘Edit’ menu item on the ‘Variables’ page. For each variable, it is explained how its values can be edited and displayed.

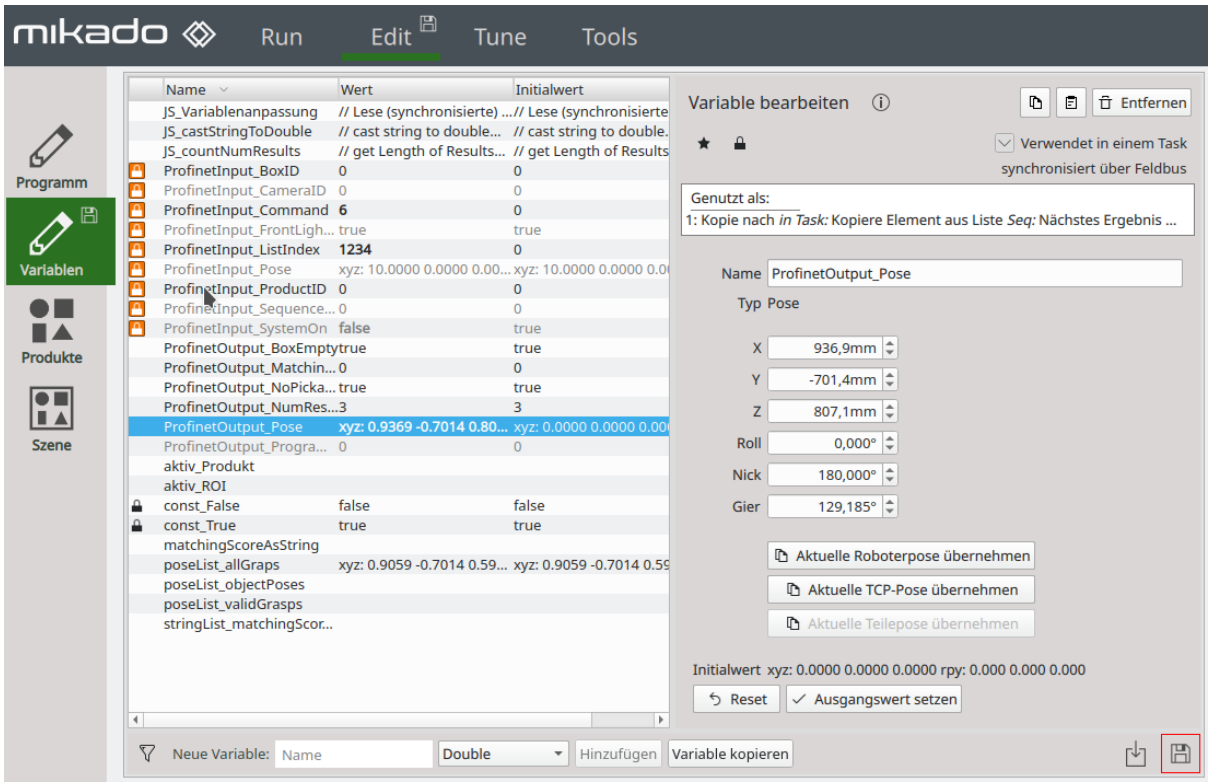


Fig. 8: Variable editor

2.9.1 Variablentabelle

The variables are listed in tabular form on the left-hand side of the page. Further columns contain the current value and the initial value (see below for a more detailed explanation). The list is sorted alphabetically, with variables starting with capital letters first and then variables starting with lower case letters.

If variable values differ from their initial values, the variable value is displayed in bold font.

2.9.2 Editing variables

Click on an entry in the variable table to display a view for editing the variable to the right of the table. This view contains the following elements:

Property indicator

Variables can have additional properties, which can be (de)activated by clicking on the buttons shown below.

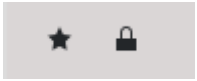


Fig. 9: Buttons for (de)activating properties

Favorite variables

Favorites are always displayed at the top of the variable table. In programs with many variables, this feature can help you find important variables more quickly.

Read-only variables

Read-only variables have the following properties:

(1) The value cannot be changed by the Mikado program. If a task attempts to change the value of a write-protected variable, a program error is thrown. This also means that write-protected variables cannot be reset.

The origin of the write-protected status is subdivided. If the status is user-defined, symbolized by , the following property becomes active:

(2) Value and initial value are synchronized. Users can set the value on the variables page by temporarily removing the write protection.

Or it is an input variable of the active interface. Input variables are automatically write-protected and are symbolized by . External variables have the following property:

(3) They cannot be modified by the user. This status can only be achieved at runtime if an interface is active. This status is not saved.

Appearance

Clicking on the button to the left of the occurrence counter opens a list with a detailed description of the occurrence. Click on an entry to jump to the corresponding point in the program sequence.

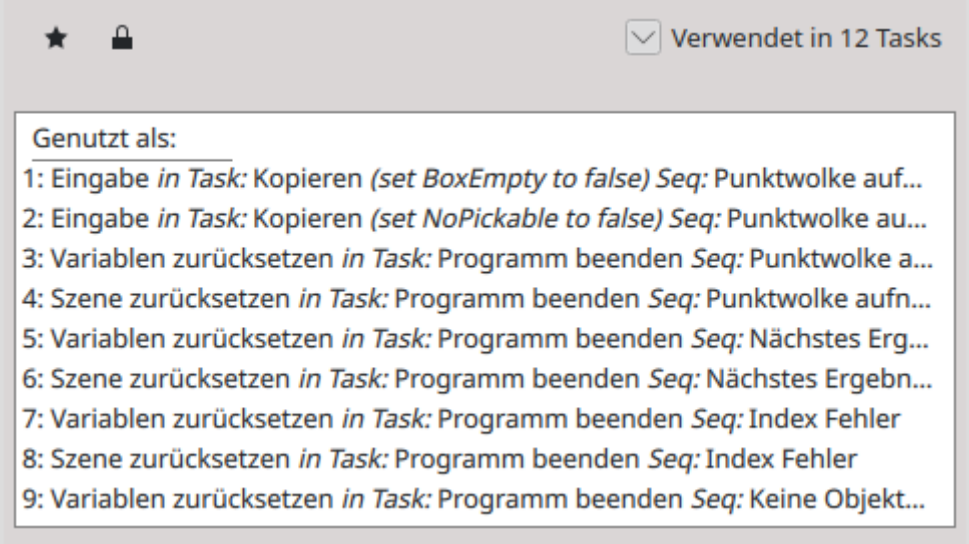


Fig. 10: Example of variable occurrence

Name

Variable name

Typ

An overview of the types with a brief explanation of each can be found in the following table.

Variablentyp	Description	Example values
Integer	Integer	`1`
Double	Decimal number	`1.5`
String	Text	Box
StringList	List of text elements	Box1 Box2
JointPosition	Robot position represented by the individual joint angle values in degrees	`90 0 90 0 90 45`
Pose	Pose consisting of position (x, y, z) in m and orientation (roll, pitch, yaw) in radians e.g. for grab pose, product pose etc. *Note*: A pose is a form of representation of the homogeneous transformation matrix. Both representations can be transformed into each other.	`xyz: 0.1 0 0.5 rpy: 0 0 3.14`
PoseList	List of pose elements	`xyz: 0.1 0 0.5 rpy: 0 0 3.14` `xyz: 0.2 0 0.6 rpy: 3.14 0 3.14`
Bool	Boolean value (`true` or `false`)	`true`

JScript	A text-like data type that contains JavaScript code	`var a = dataStore.data("a")`
		`a = a + 1`
		`dataStore.setData("a", a)`

Value

The current value of the variable. The value of a variable can change at any time as the program runs.

If it is a StringList, the text field is multi-line and list entries are separated by a line break.

Initial value

The value that is assigned to a variable when the Mikado program is started or reset. This value is saved when a program is saved and is generally not changed during a program run.

Possible actions

- ***Reset***
  Resets the current value of the variable to the initial value.
- ***Remove***
  Deletes the variable and its content.
- ***Confirm***
  Confirm input. Only visible for list input fields, as simple input fields are confirmed by pressing the 'Enter' key (or by switching to another field).
- ***Adopt current robot pose***
  Sets the value of the variable to the pose that the robot is currently assuming (only visible for the 'Pose' variable type).
- ***Take over current part pose***
  Sets the value of the variable to the pose that a gripped part is currently in (only visible for the 'Pose' variable type). The prerequisite for this function is that the robot has successfully gripped a part and is currently holding it in the gripper.
- ***Adopt current joint values***
  Sets the value of the variable to the joint angle that the robot is currently at (only visible for the 'Joint-Position' variable type).

Hint: Always confirm an entry with the Enter key or switch to the next input field, only then will the input values be accepted.

2.9.3 Adding and filtering variables

Below the variable table are input fields that can be used both to add new variables and to filter existing variables.

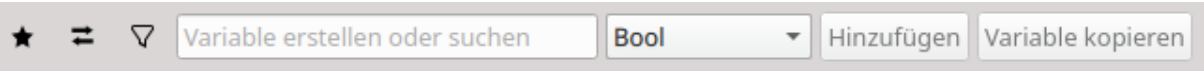


Fig. 11: Adding and filtering variables

The buttons on the left can be used to activate or deactivate certain filters for the existing variables.

- ★ Show favorites only
- ⇄ Show fieldbus variables only
- ▽ Only show variables of the currently selected type. This refers to the variable type selected further to the right in the selection.

The text field in the middle is used to enter a name for a new variable and also acts as a name filter for existing variables. The selection next to it is used to choose the type of a new variable or the associated filter.

To add a new variable, first enter a variable name that has not yet been used in the text field. Then select the desired variable type. Click on "Add" to create the new variable. It will appear in the table and can then be edited as usual.

The "Copy variable" button can also be used to copy the currently selected variable.

2.9.4 Import variables from file

The ‘Import from file’ button can be found at the bottom of the page. You can use this function to add a complete variable table from another Mikado program to the existing variable list. This does not change the main program. The following cases can occur:

- **New variable name:**
  Variable is added
- **Variable name already exists, variable type matches:**
  The current value is replaced by the value from the file to be read, initial value remains unchanged
- **Variable name already exists, variable type does not match:**
  Variable remains unchanged, the variable from the file is not imported

2.9.5 Visualization of the robot pose

As soon as you have created a variable that represents a robot pose (variable type ‘Pose’ or ‘JointPosition’), you can display the value of this variable in the visualization as a preview. To do this, simply click on the variable in the table and the entry will be highlighted in color.

Visualization of a joint position

Variables of type ‘JointPosition’ are represented in the visualization by a transparent robot to distinguish it from the actual robot pose.

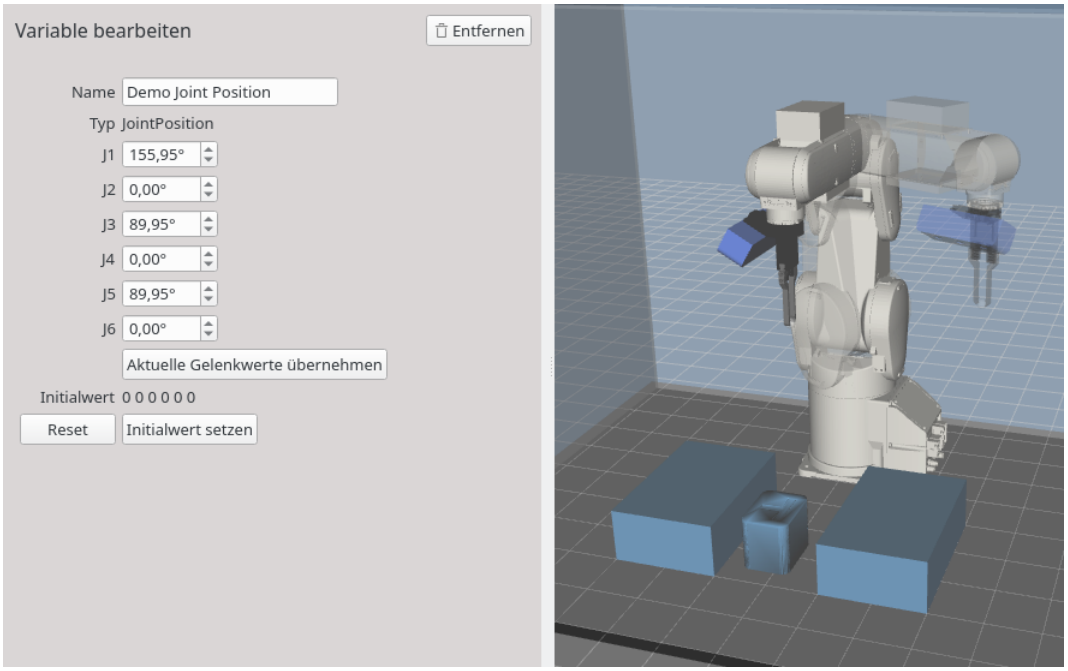


Fig. 12: Visualization of a joint position. On the left is the actual current robot position, on the right is the transparent model as a preview of the selected joint position.

Visualization of a pose

Variables of type ‘Pose’ can be visualized in different modes:

1. ***Wrist***
   If the variable is currently assigned a valid value that the robot wrist could also theoretically assume (without taking collisions into account), another (virtual) robot is drawn in the visualization of the scene. This robot model is displayed slightly transparently. It shows the pose/position that the robot would assume when moving off.

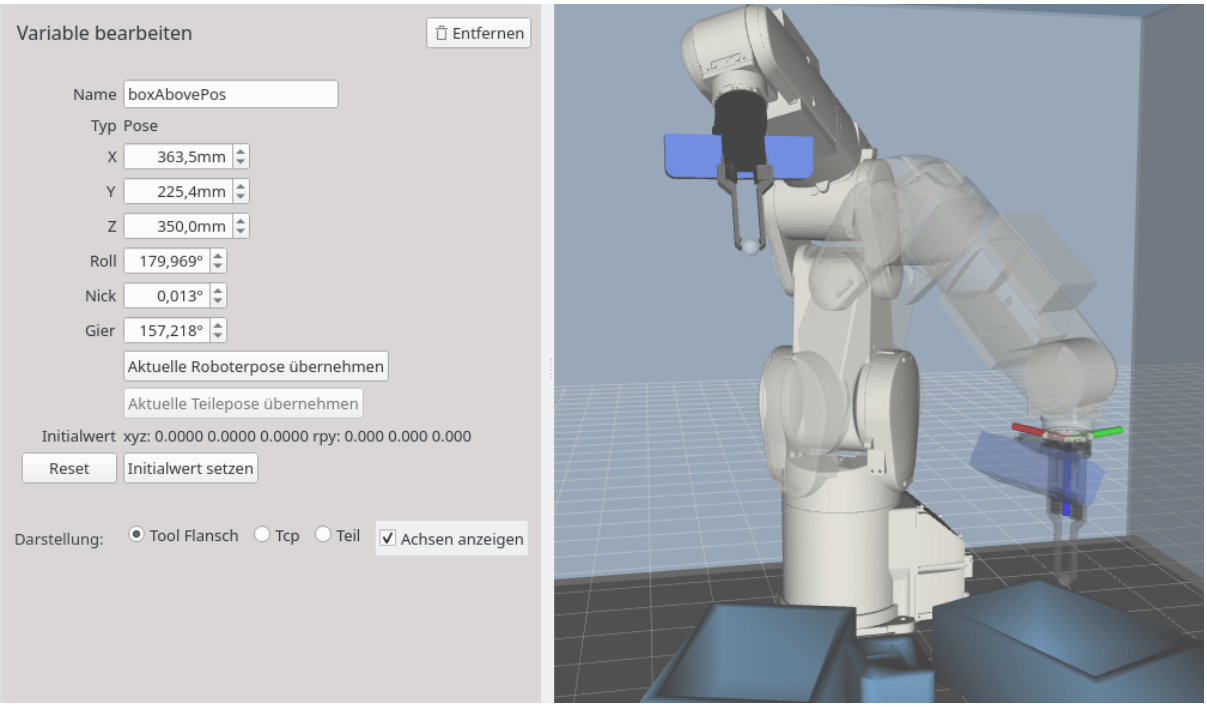


Fig. 13: Visualization of a robot pose in ‘wrist’ mode

2. ***TCP***
   If the variable is currently assigned a valid value that the end effector (TCP) could also theoretically assume (without taking collisions into account), another (virtual) robot is drawn in the visualization of the scene. This robot model is displayed slightly transparently. It shows the pose/position that the robot would assume when moving off.

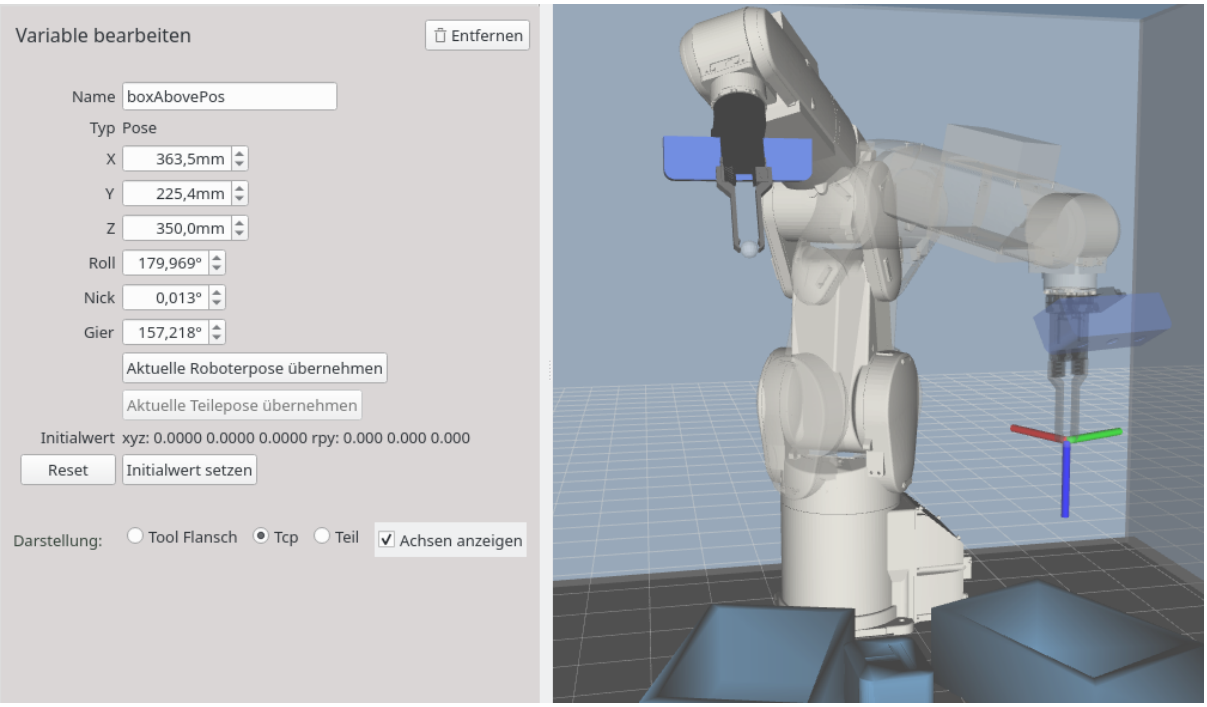


Fig. 14: Visualization of a robot pose in 'TCP' mode

3. ***Part***

   If the variable is assigned a valid value that the currently gripped part could also theoretically assume (without taking collisions into account), another (virtual) robot is drawn in the visualization of the scene. This robot model is displayed slightly transparently. It shows the pose/position that the robot would assume when approaching. The virtual robot is only displayed if the robot has successfully gripped a part and is currently holding it in the gripper. If this is not the case, the currently active product is displayed at the pose(s) without a virtual robot.

4. ***Show axes***

   Activate this setting to additionally display the pose as a simple, three-color axis system. This option is active by default, but can be hidden using the checkbox.

   Note that axes are automatically displayed if you select a PoseList without using the *Part* visualization mode.

If an error occurs in a selected visualization (e.g. robot cannot reach pose due to geometry), this is displayed in red at the bottom next to the visualization modes.

Visualization of collisions

If you select a joint position or pose from the variable list, a preview is displayed in the visualization (see *Visualization of the robot pose*). In order to be able to check whether the robot or gripper would collide with other objects in the environment when setting up the position or pose, potential collisions are displayed directly in the visualization window. Colliding parts of the robot are colored red.

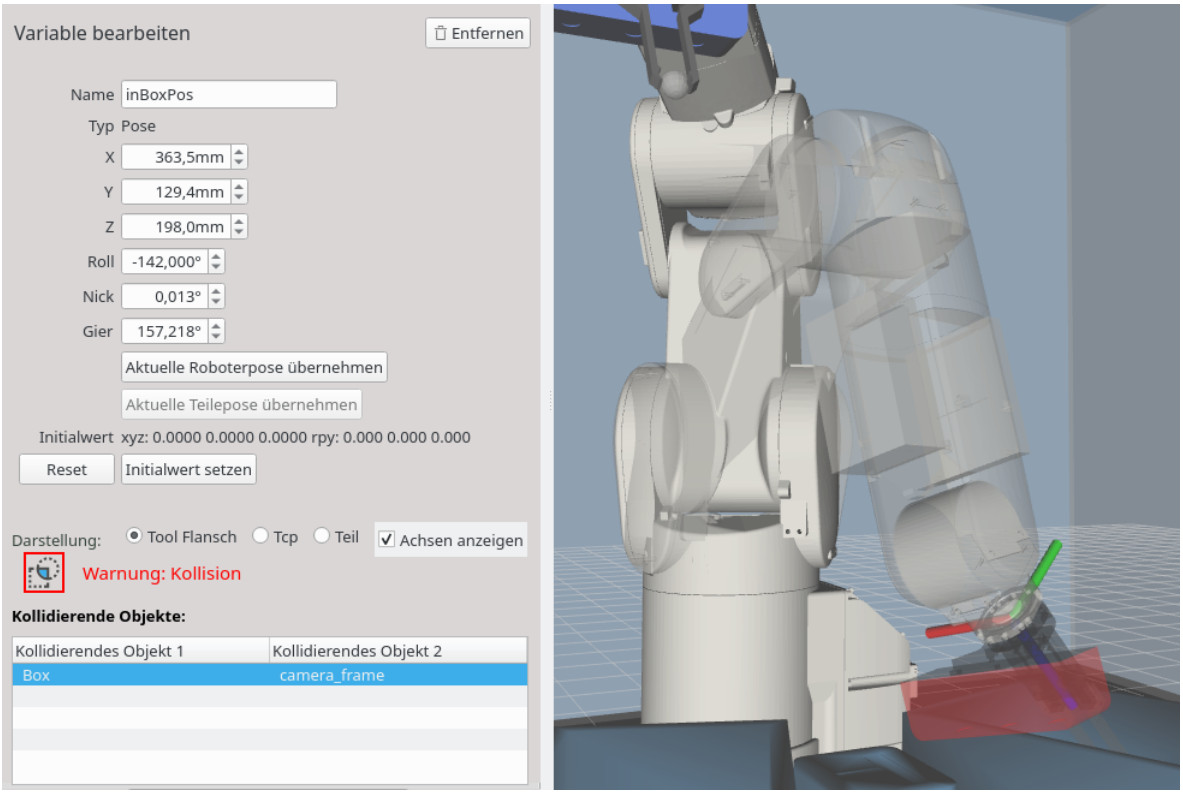


Fig. 15: Visualization with collisions on the gripper

A message with a corresponding symbol also appears, indicating that the selected pose cannot be reached in the currently selected mode due to collisions. The individual collisions are displayed in a list below. Please note that this is only an indication. In some cases, the pose can still be used for a functioning and collision-free sequence. For example, collisions can occur with scene objects that change their position in the sequence. Another example is poses that are transformed again within a program sequence.

2.10 Product management

2.10.1 Overview

A product describes a physical object that is to be found and possibly gripped using 3D recognition. It consists of a name and various parameters for the object search and the gripping process. A distinction must be made between the (abstract/descriptive) “product” on the one hand and the concrete “object” on the other: The same product can be found multiple times during an object search. Each object is an instance of the product (with a specific position in space).

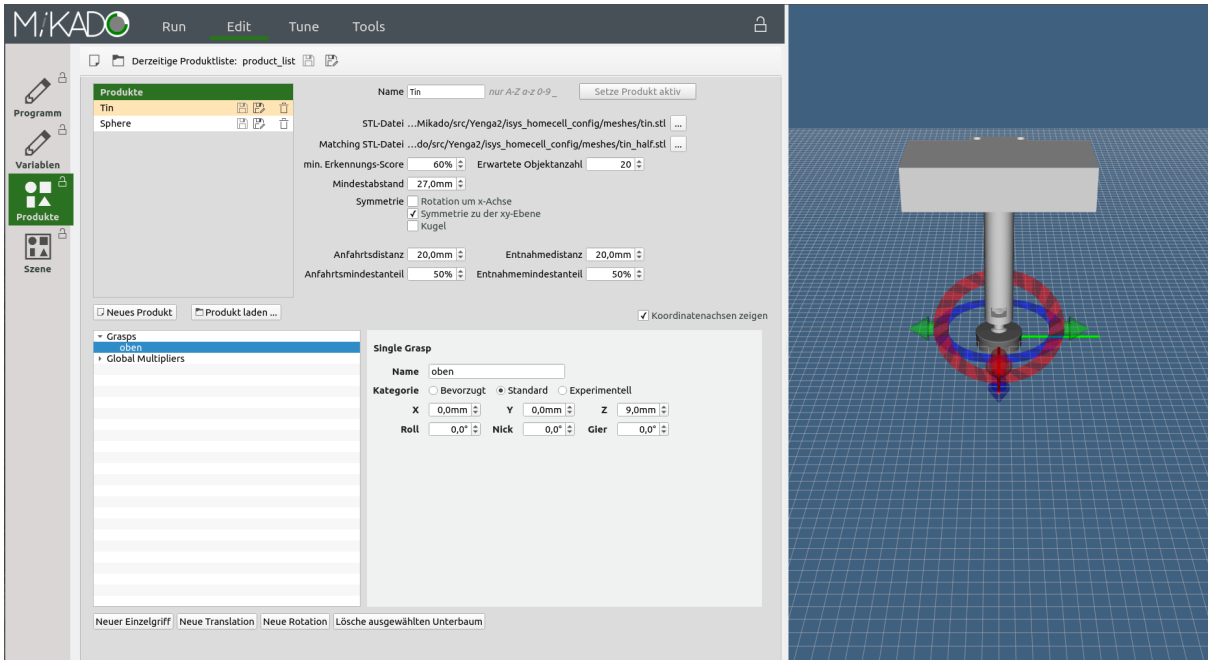


Fig. 16: Screenshot of product management with single handle

2.10.2 Name

The product name identifies the part to be searched for and must fulfill the following criteria:

- Must be unique, i.e. two products must not have the same name
- Starts with a letter
- Consists of alphanumeric characters (a-z, A-Z, 0-9) or underscores ('_')

2.10.3 Active product

During program execution, exactly one product is always defined as the 'Active product' (provided at least one product exists in product management). If product management only knows one product, this is automatically the active product. If a product-related action is executed, the active product is used for this action.

Example: If you use the task *Start handle search*, possible handles for the active product are searched for. However, if you define the variable 'Product' in the program editor, the system searches for the product you have defined, regardless of the active product.

The active product can either be set in the product administration using the 'Use as active product' button, or as part of the Mikado program using the *Set active product* task.

2.10.4 Object search

Each product contains the following parameters for the object search:

STL file / STL file (recognition)”

The mesh file (CAD data) of the object to be used for the display and object search is specified here. If nothing is specified under “STL file (recognition)”, the file specified under “STL file” is used for recognition. Otherwise, the file specified under “STL file (recognition)” is used for recognition and the file specified under “STL file” is only used for display. This means that a simplified mesh can be used for recognition, while the full mesh is shown in the display.

The following requirements and instructions must be observed when creating/using an STL file:

- Use millimeters as the unit of measurement for the STL
- Use meshes with as few triangles (faces) as possible - as few as possible, but enough to ensure that all important structures of the product are still well represented. The more triangles are used, the longer the computing time for collision checks, for example.
- Select a sensible origin of the product coordinate system, for example at the center of gravity of the product.
- If the product has symmetries, position the origin of the product coordinate system on the symmetry axis or plane.
- If possible, remove all triangles that are in the product or are never visible.

Search algorithm

The search algorithm used for object recognition is defined here. Mikado Control offers various search algorithms to choose from:

- Ensenso Partfinder (recommended), a search algorithm developed by Optonic, see also Partfinder documentation.
- Halcon Surface Based, a search algorithm developed by MVTec. An additional license is required for this.

Min. detection score

The object search determines a score for each potential object in the scene. This score is the measure of the correspondence between the potential object findings and the model. The minimum value that a potential object must exceed in order to be considered a recognized object is specified here.

Expected number of objects

To speed up the object search, you can enter an expected number of objects here. In addition, the probability of finding an object increases if this value is set higher. If you want to limit the maximum number of objects to be recognized, you can set an upper limit here. If you have activated the reuse of previous matches, this restriction only applies to new finds (more on the topic of *reuse of previous matches* in section *Object recognition tool*).

Minimum distance

The minimum distance between two objects found is specified here. This has the following effects when searching for objects: If there are several potential objects close to each other, the one with the best score is selected and all other objects that are within the minimum distance are discarded.

Symmetries

Products might have rotational symmetries, i.e. when rotating the product around some axis its shape or appearance does not change. Mikado supports continuous rotational symmetry around the main axis X, Y, and Z as well as discrete rotational symmetry with a 180° rotation around one of these axes. Configuring symmetries is not strictly necessary but might speed up computations and helps avoiding unnecessary rotational movements.

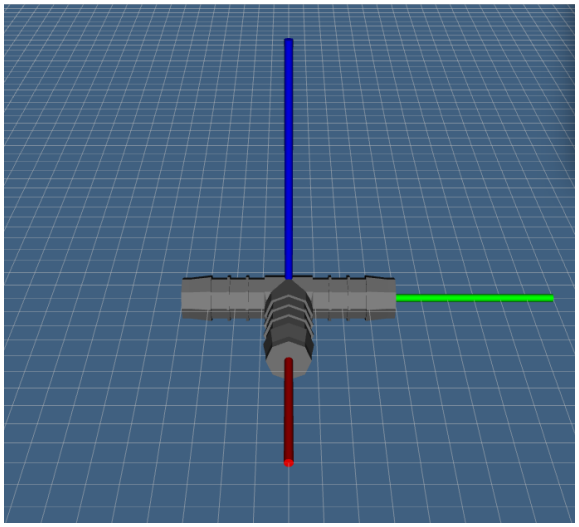


Fig. 17: 180° rotational symmetry around the X-axis (red)

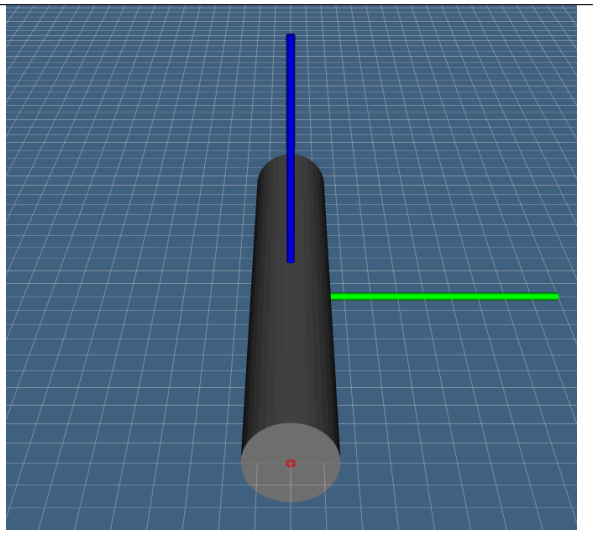


Fig. 18: Rotational symmetry around the X-axis (red) and 180° rotational symmetry around Y (green) and Z (blue)

Hint: The prerequisite for this is that the origin and orientation of the product coordinate system have been configured appropriately in the CAD data.

When using the object recognition with PartFinder the viewpoint setting fromProductSymmetry can be used to automatically reduce the number of rendered perspectives.

Now the found object poses will be aligned to the world coordinate system, irresponsibly of the recognition algorithm used. For example objects with a continuous rotational symmetry around X will be rotated such that their Z-axis points up as much as possible. For discrete 180° rotational symmetry around X, poses will be rotated such that Z points in positive direction of the world coordinate system as well. This applies similarly for symmetries around the Y-axis. For symmetries around the Z-axis, poses are aligned such that object x-axis points in positive world x-direction.

Hint: This allows greatly reducing the number of grasp poses as we no longer need to set-up grasps from the downward facing side. However it is still recommended to configure some slightly rotated grasps even if the product has continuous rotational symmetry as collisions might obstruct a grasps directly from above.

Additionally symmetries are considered for movements of the robot in *Move part to*, *Move part Cartesian to* as well as the reachability query *Is robot position accessible with part?*. For example a cylindrical piece with a 180° rotational symmetry around the Z-axis will not be rotated needlessly but rather the solution closest to the starting position is used.

Withdrawal distance

The distance between the gripping and picking pose is defined here. These poses are defined in the *Gripping process* section.

Minimum withdrawal percentage

Here, the minimum removal distance is defined as a percentage of the removal distance.

When checking the removal, the distance from the gripping pose to the removal pose is checked for collisions. It is possible to set that only a certain percentage of the distance must be covered, rather than the entire distance. These poses are defined in the *Gripping process* section.

Example: If a minimum extraction percentage of 70 % is set and a collision would occur after 80 %, the actual extraction pose is set to the pose after 79 % and free planning (towards the final pose) is carried out from this pose. This makes it possible to define the removal distance for the ideal case and, on the other hand, to define a minimum distance that must be travelled cartesian.

Travel distance

The distance between the pick and approach pose is defined here. These poses are defined in the *Gripping process* section.

Minimum travel time

The minimum distance for gripping is defined here as a percentage of the approach distance. The description in *Minimum withdrawal percentage* also applies analogously to this parameter.

2.10.5 Gripping poses

Gripping poses" are the poses in the product coordinate system to which the gripper may apply in order to grip the product. The gripping poses are defined via gripping multipliers, which can be used to multiply groups of gripping poses with each other. They are described in more detail in the following section.

External and internal grasps

Depending on the gripper, different types of grasps can be used:

- ***External grasps***
  The robot opens the gripper before it performs the gripping movement. At the gripping point, it closes the gripper to grip the product from the outside.
- ***Internal grasp***
  The robot closes the gripper before it performs the gripping movement. At the gripping point, it opens the gripper to grip the product from the inside (e.g. a hole or a recess in the product).

Which grasp types you can actually use depends on the gripper used. Create new grasps in the corresponding category to define the desired grasp type.

Gripping multipliers

Grab multipliers contain various combinations of the following properties:

- Name
- Category (Preferred, Standard, Experimental, <inherited>)
- Position (x, y, z)
- Orientation (Roll, Pitch, Yaw)
- Multiplier (natural number)
- symmetrical?

The name of a gripping multiplier must be unique for a specific product.

The category can be used to set which grips should be checked first (in the order "Preferred" > "Standard" > "Experimental"). If no category is set for a grip multiplier, it "inherits" the category of the grip multiplier above it. For rotation and translation multipliers, the category only affects "real" rotations and translations, i.e. not 1 (rotation) or 0 (translation) (see below).

If no handle of the categories "Preferred" or "Standard" is found, but only a handle of the category "Experimental", the corresponding output variable in the *Pick object* task is set to true. This means that further processing can be made dependent on whether the handle was experimental or not. The picked objects can then, for example, be realigned or dropped into the box to be picked again.

The "Multiplier" property specifies the number of gripping poses within a gripping multiplier. A single grip, for example, always has a multiplier of 1, whereas rotation and translation multipliers always have a multiplier of at least 2.

The "symmetrical" property only exists for rotation multipliers (see below).

The individual multipliers available are explained in more detail below.

Single handle

A single grip always has a multiplier of 1. The position and orientation can be set for the single grip - i.e. a gripping pose is defined. As the multipliers always include the identity transformation, single grips must also be used for pure movement or rotation. Further rotations or translations can then be multiplied onto these.

Translation multiplier

A translation multiplier always has a multiplier of at least 2 and a position (translation) that acts as a generator of the various translations. The resulting translations are {0, T, 2*T, … , (m-1)*T}, where m is the multiplier value and T is the translation generator.

Rotation multiplier

A rotation multiplier always has a multiplier of at least 2 and an orientation (rotation) that acts as a generator of the various rotations. It also has the Boolean property "symmetrical". If this is not set, the calculated rotations are {1, R, 2*R, … , (m-1)*R}, where m is the multiplier value and R is the rotation generator. If "symmetrical" is set, the resulting rotations are {-(m-1)*R, … -2*R, -R, 0, R, 2*R, … , (m-1)*R}.

Multiplication

If you have two grab multipliers and multiply them, all poses of one multiplier are multiplied by the poses of the second multiplier.

Example: Gripping multiplier M1: Translation multiplier (m=2, x=50mm, y= 0mm, z=0mm) Gripping multiplier M2: Translation multiplier (m=3, x= 0mm, y=50mm, z=0mm)

M1*M2 contains the following poses (x, y, z; roll, pitch, yaw):

```
(  0,   0, 0;  0, 0, 0)
(  0,  50, 0;  0, 0, 0)
(  0, 100, 0;  0, 0, 0)
( 50,   0, 0;  0, 0, 0)
( 50,  50, 0;  0, 0, 0)
( 50, 100, 0;  0, 0, 0)
```

In general, the number of poses in the product of two grasp multipliers is equal to the product of the number of poses of the individual grasp multipliers. (#(M1*M2) = #(M1) * #(M2))

Multiplier tree

It is possible to define several groups of gripping poses on a product, whereby each group in turn consists of several multipliers. This is possible in the multiplier tree: By adding a new gripping multiplier at the top level, a new group of gripping poses is defined. By adding a gripping multiplier A to an existing multiplier B, the multiplier is multiplied by B. New groups can be added at any level.

Example tree:

```
- A
  - R
- B
  - T1
    - C
  - T2
    - D
```

Assuming A, B, C are single handles, R is a rotation with multiplier 5 (non-symmetrical) and T1, T2 are translations with multiplier 3, the following handles result:

```
- A
- A * R
- A * R^2
- A * R^3
- A * R^4
- B * C
- B * T1 * C
- B * T1^2 * C
- B * D
- B * T2 * D
- B * T2^2 * D
```

Global multipliers

Multipliers can be defined here, which are attached to each leaf of the gripping multiplier tree.

Example:

```
- Greifpunkte:
  - A
    - A1
    - A2
  - B
    - B1
- Globale Multiplikatoren
  - M1
    - M11
  - M2
```

This results in the following tree:

```
- A
  - A1
    - M1
      - M11
    - M2
  - A2
    - M1
      - M11
    - M2
- B
  - B1
    - M1
      - M11
    - M2
```

2.10.6 Collision Check for Grasp-Poses

If the Grasp-Multipliers lead to grasps that collide with the product, the Grasp-Tree displays a red bar as well as the number of colliding grasps. The colliding grasps are assigned to those Grasp-Multipliers that cause them. If such a Grasp-Multiplier is expanded, the assignment becomes more detailed. The following two screenshots show an example of this at different expansion depths.

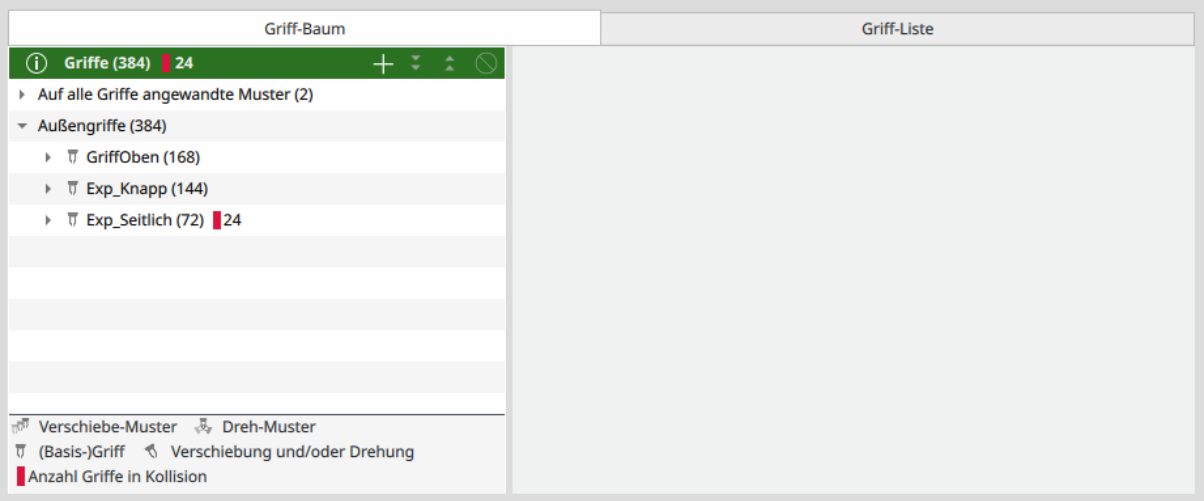


Fig. 19: Screenshot Grasp-Tree with Number of colliding Grasps (Expansion-Depth: 1)

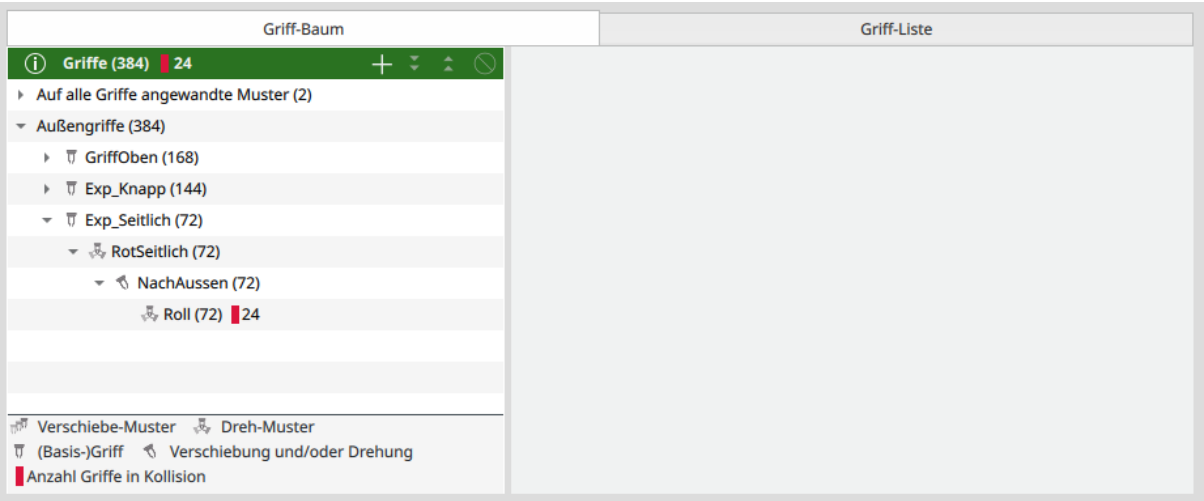


Fig. 20: Screenshot Grasp-Tree with Number of colliding Grasps (Expansion-Depth: 4)

Next to the Grasp-Tree is the tab Grasp-List. This holds all generated Grasps. Here too the collisions are displayed in the header via a red bar and the number of colliding grasps.

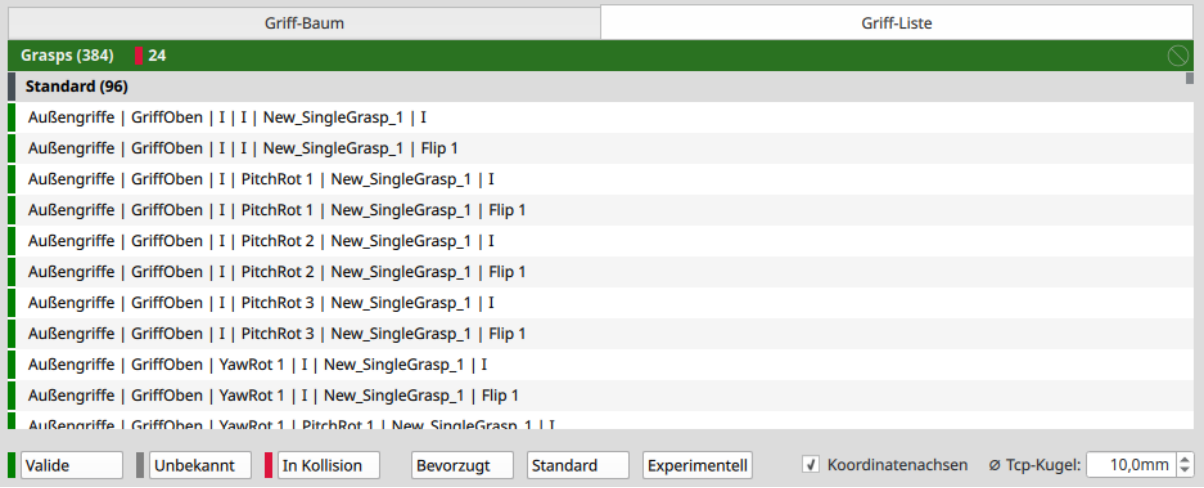


Fig. 21: Screenshot Grasp-List (unfiltered)

Below the list there are buttons for filtering the grasps (similar to the *Handle inspection*). These can be used, for example, to display only the colliding Grasps. By deselecting "Valid" collision free grasps will be filtered out.

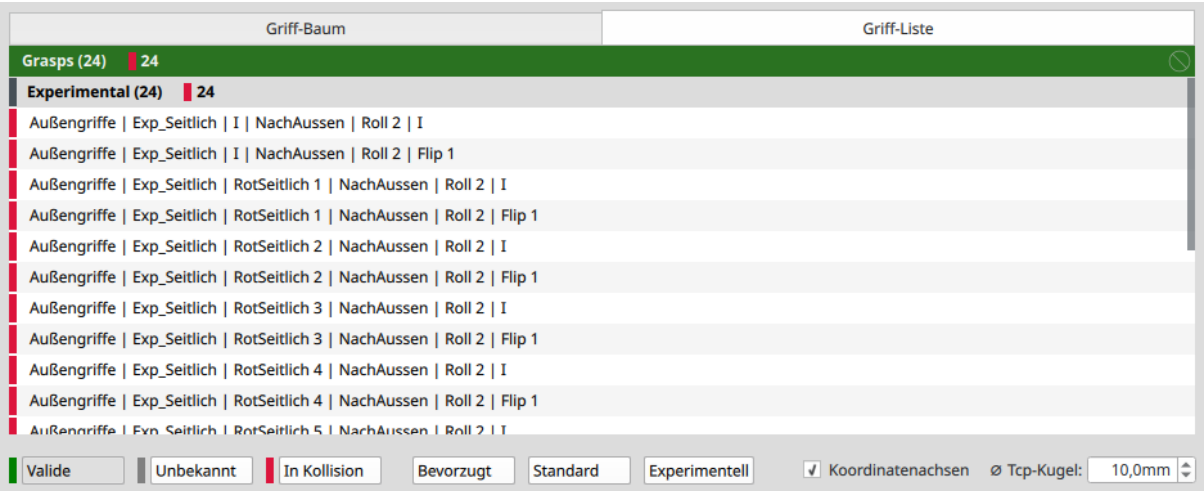


Fig. 22: Screenshot Grasp-List (only colliding)

When a grasp from the Grasp-List is selected, it is displayed in the visualization. For a colliding grasp, the collision-points between gripper and product are marked via small red spheres.

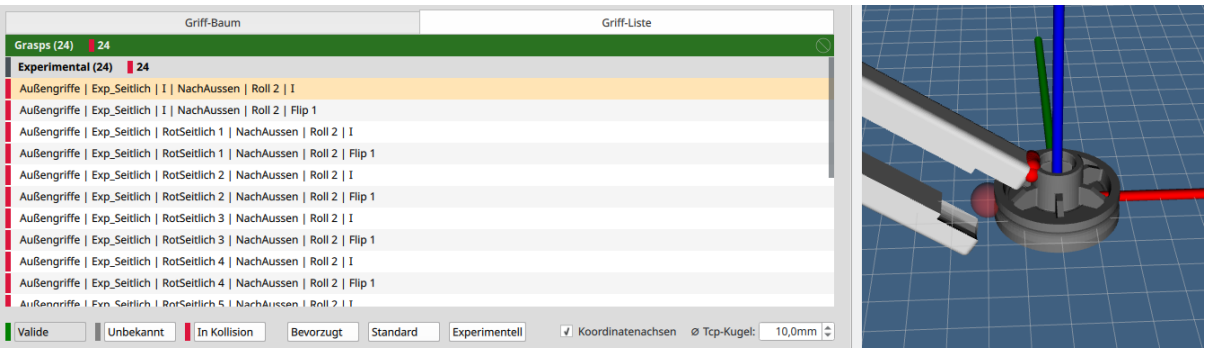


Fig. 23: Screenshot Grasp-List and Visualization of collision points

Setting up handles for interchangeable grippers

This section is only relevant for cells with multiple grippers.

If a multi-gripper is configured for a cell, the grasp setting looks like the following screenshot.

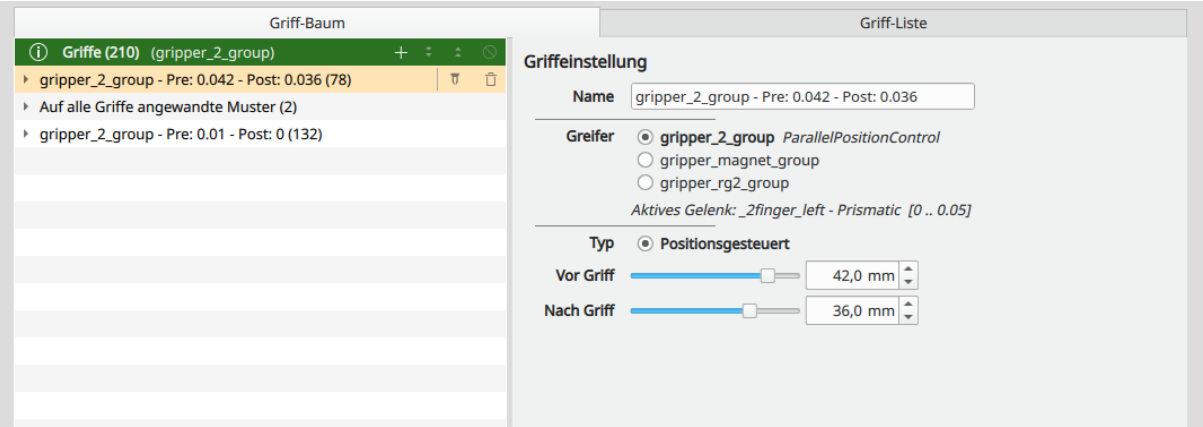


Fig. 24: Setting up grasps for multiple grippers

The associated gripper group can be set for each grasp setting. All grasps corresponding to this grasp setting are assigned to the selected gripper group.

Several grasp settings can be defined for one product - and therefore also grasps associated with different grippers. The pick search only ever searches for grasps that are assigned to the active gripper.

2.11 Greifpose

2.11.1 Overview

The handle inspection tool is used to analyze the search for potential gripping movements. If N objects are found in a region and M pick points (see *Gripping poses*) are defined for the corresponding *product*, there are basically NxM potential pick movements – one for each combination of object and pick point. Each of these movements is first checked for potential collisions. As soon as the rest of the program requires a gripping movement (e.g. when a "gripping object" task is reached), the "best" movement is selected from all the valid movements found up to that point. The "quality" of a movement depends on several criteria (see below), including the position-height of the associated object and the length of the gripping movement.

2.11.2 Gripping process

The gripping movement is basically divided into the following stages:

Start up
Starting pose → Approach pose

Grasping
Approach pose → Grab pose (linear travel in Cartesian space)

Remove
Gripping pose → Removal pose (linear movement in Cartesian space)

Back up
Removal pose → End pose

Gripping and removal are movements along a straight line in 3D space, whereas the robot can basically move freely when approaching and returning.

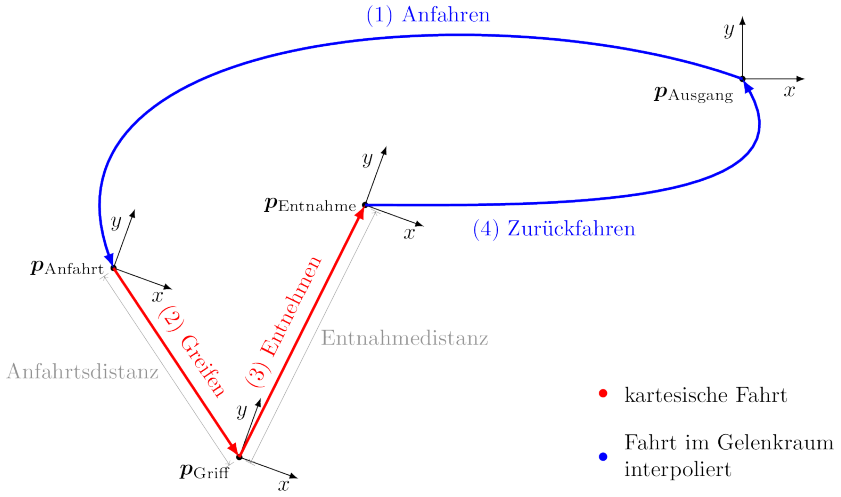


Fig. 25: Schematic representation of the gripping process

Initial and final pose

The start and end poses are given by the transition pose of the surrounding object (box) belonging to the search. The start and end poses are therefore identical.

Approach pose

To calculate the approach pose, a certain distance is calculated "backwards" from the gripper pose. "Backwards" means away from the product, i.e. in the z-direction in the coordinate system of the gripper pose (z-direction of the "gripper_tcp" Urdf object). The distance between the approach and gripper pose is set via parameters, see section *Search-Strategies*.

Greifpose

If N product objects were found and M *gripping points* are defined for the corresponding product, the gripping pose in the world coordinate system results from the pose of the nth product object and from the mth product-specific gripping point (in the product coordinate system). The product-specific *grip points* must have been created manually beforehand.

Removal pose

Different strategies are tried out in a fixed sequence to calculate the extraction pose, see section *Search-Strategies*. If extraction in one of the directions is not possible, the next direction is checked. The gripping process for a specific gripping pose is only considered to have failed if none of the variants leads to a feasible gripping movement. The distance between the removal and gripping pose is set via parameters, see section *Search-Strategies*.

Note: In general, path planning prevents any collisions between the robot, cell, scene objects, product objects and 3D point cloud. However, during picking (picking pose → picking pose), movements are also permitted that involve collisions between the objects to be picked (for more information, see section *Gripping motion search*). As these objects are not created as fixed scene objects, it is assumed that they are movable and insensitive to touch. If a completely collision-free movement is found, this is always preferred over a collision-prone one. This removal strategy increases the probability of a successful grasp.

2.11.3 Gripping motion search

The overall trajectory consists of four partial trajectories. Gripping and removal are movements along a straight line in 3D space, whereas the robot can basically move freely when approaching and returning.

The search for a valid gripping movement is carried out in the following steps, analogous to the above classification:

(a) Collision check of the gripping pose (without attached object)

(b) Gripping: linear (straight) movement in Cartesian space (not in joint space)

(c) Approach

(d) Collision check of the gripping pose with attached object

(e) Withdrawal

(f) Return journey

In (a) and (d), only one pose is checked at a time; in the other four steps, the sub-trajectories mentioned above are created (provided a valid plan can be found in each case). There are different search approaches for the individual sub-trajectories. If the search does not find a valid trajectory for a step, the gripping movement is discarded. The various search strategies for the partial trajectories are listed below:

(a) Collision check of the gripping pose (without attached object)

(b) Gripping: linear (straight) movement in Cartesian space (not in joint space)

(c) Start up

  (i) by interpolation in the joint space

  (ii) per free path planning

(d) Collision check of the gripping pose with attached object

(e) Removal (linear (straight) travel in Cartesian space (not in joint space))

  (i) collision-free

    - in grab direction (*ExtractInGraspDirection*)
    - top (*ExtractUpwards*)
    - in the direction of the final pose (*ExtractTowardsFinalPose*)
    - towards the center of the crate at a 60° angle to the (x,y) plane (*ExtractTowardsCenter<Angle[°]>*)

  (ii) *Object/octomap collisions* are permitted

    - in grab direction (*ExtractInGraspDirection_AllowOOCollisions*)
    - nach oben (*ExtractUpwards_AllowOOCollisions*)
    - in the direction of the final pose (*ExtractTowardsFinalPose_AllowOOCollisions*)
    - in the direction of the center of the crate at a 60° angle to the (x,y) plane (*ExtractTowardsCenter<Angle[°]>_AllowOOCollisions*)

  (iii) *Overlap collisions* are allowed

    - in grasp direction (*ExtractInGraspDirection_AttObjObjColl*)
    - upwards (*ExtractUpwards_AttObjObjColl*)
    - in the direction of the final pose (*ExtractTowardsFinalPose_AttObjObjColl*)
    - in the direction of the center of the crate at a 60° angle to the (x,y) plane (*ExtractTowardsCenter<Angle[°]>_AttObjObjColl*)

  The order of the extract-functions can be set via the parameter *extract_fn_order* (see section *Search-Strategies*). Only collision-free functions are to be listed. All others will be generated automatically.

(f) Back up

(i) by interpolation in the joint space (*RetractInterpolating*)

(ii) via free path planning (*RetractFreely*)

Note: The extraction trajectories *ExtractTowardsCenter10* and *ExtractTowardsCenter40* are deactivated by default. They can be activated on the parameter-page (*Search-Strategies*) by setting the following parameters:

```
ExtractTowardsCenter10: true
ExtractTowardsCenter40: true
```

Table 1: **Overview of the different removal strategies**

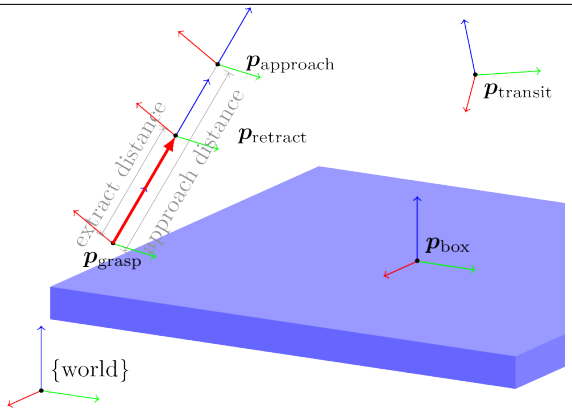


Fig. 26: *ExtractInGraspDirection*

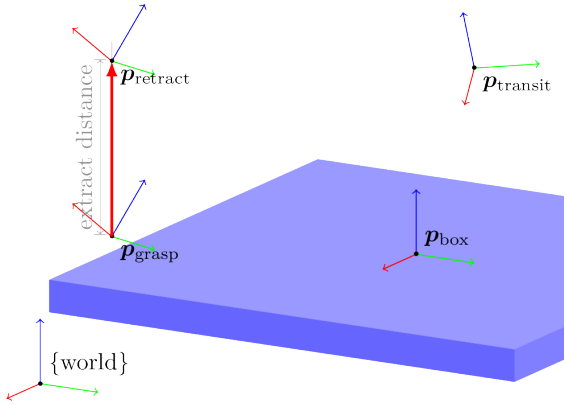


Fig. 27: *ExtractUpwards*

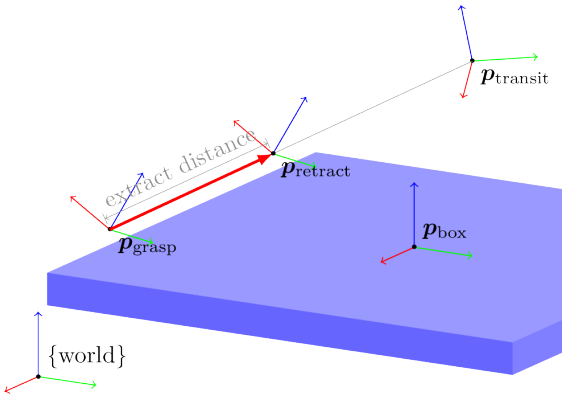


Fig. 28: *ExtractTowardsFinalPose*

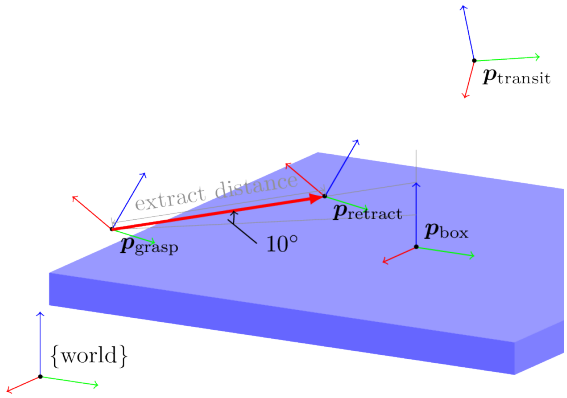


Fig. 29: *ExtractTowardsCenter10*

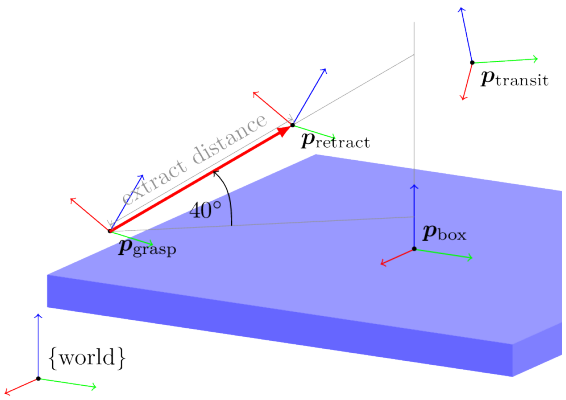


Fig. 30: *ExtractTowardsCenter40*

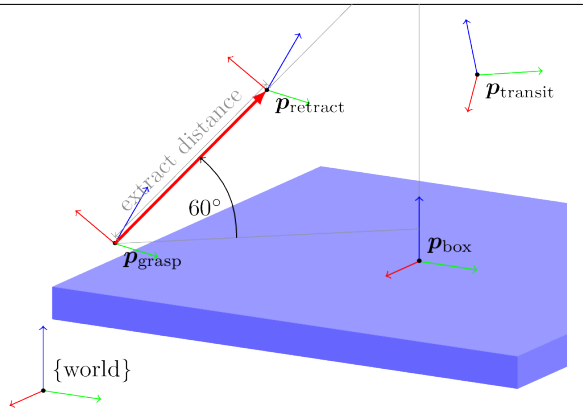


Fig. 31: *ExtractTowardsCenter60*

Object/octomap collisions

These are collisions of the object to be gripped with either

(a) other product objects (but not with scene objects), or with

(b) der Octomap.

This type of collision can be allowed via the parameter *allow_OO_collisions_during_extraction*, see section *Search-Strategies*.

Overlap collisions

These are collisions of the object to be gripped at the gripping pose. Due to inaccuracies in the position detection of the objects, it can happen that two objects slightly overlap ("collide") in the scene. If this cannot really be the case, but is merely an artifact of object recognition, these collisions can be ignored. If overlap collisions are permitted during removal, the collisions that already occur at the gripping point are ignored along the removal trajectory until they no longer occur. If they occur again afterwards, they are no longer ignored. The collision depth up to which overlap collisions are permitted can be set in the *pick_search.yaml* via the parameters:

- Set *max_allowed_collision_depth* generally or via
- Set *specific_max_allowed_collision_depths* product-specific.

If two objects overlap to such an extent that the collision depth is greater than the specified threshold, these collisions are not ignored, even if they already occur at the gripping point.

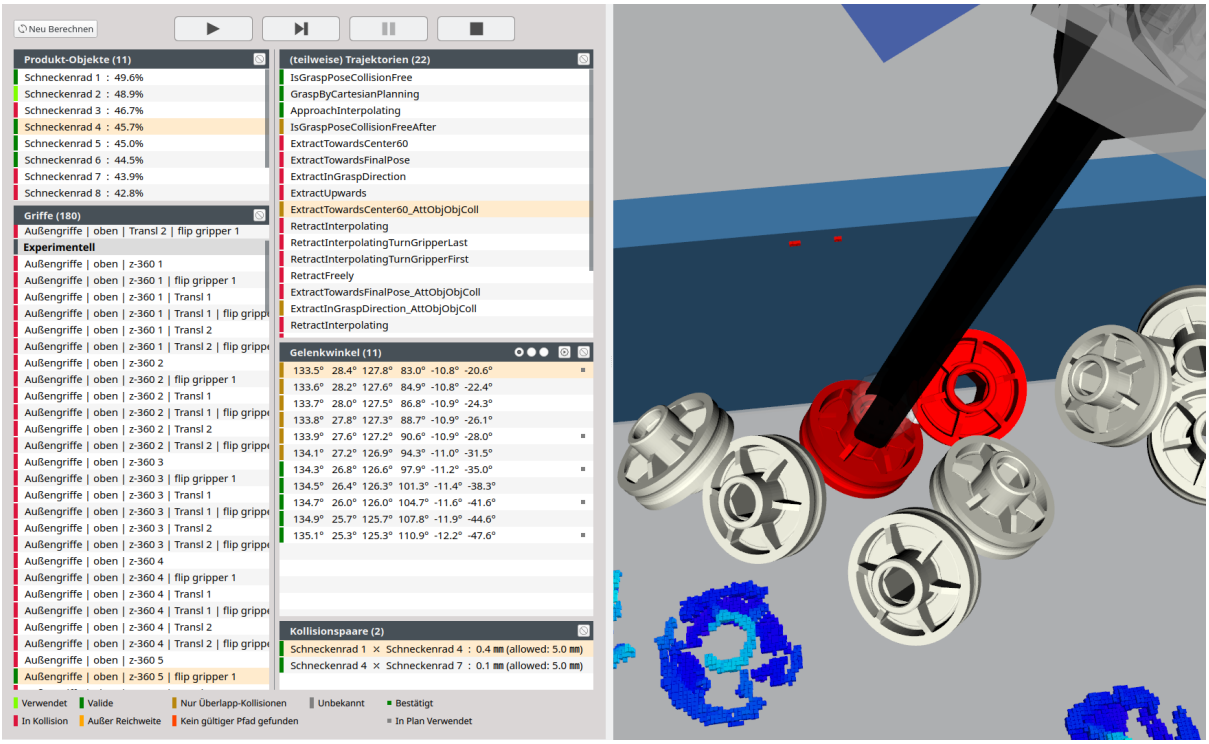


Fig. 32: Screenshot example of overlap collisions

Interpolation Parameters

In order to check trajectories in detail, further trajectory-points are repeatedly added between two trajectory-points, if they are too far apart. The thresholds determining whether two points are deemed "close enough", can be set via the following parameters (the values shown are the default-values):

```
interpolation_max_translation_diff: 0.001 // 1mm
interpolation_max_rotation_diff: 0.01745329251 // 1°
cartesian_max_translation_diff: 0.01 // 10mm
cartesian_max_rotation_diff: 0.08726646259 // 5°
```

The *cartesian_max_…_diff*-parameters are used to determine the step-size of cartesian (partial) trajectories. This includes only those points that will be sent to the robot. The *interpolation_max_…_diff*-parameters are used for all interpolation-trajectories, as well as for any additional interpolation steps between cartesian points of the cartesian trajectories.

Smaller values, especially for the *interpolation_max_…_diff*-parameters, will lead to more trajectory-points. This in turn will lead to larger computation times.

Exclude certain search functions

It is possible to switch off individual extract or retract functions. The parameter *search_fn_blacklist* in the *pick_search.yaml* is used for this purpose. It expects a list of function names that should not be used. If one of the collision-free extraction functions is listed (e.g. *ExtractInGraspDirection*), all associated extraction functions are also automatically removed (*ExtractInGraspDirection_AllowOOCollisions*, *ExtractInGraspDirection_AttObjObjColl*).

Search-Strategies

Gripper

- ***Filter Max. Angle of Wrist (relative to a vertical line)***
  Filter out grasps whose wrist-angle is above the max. angle the wrist is allowed to have (relative to a vertical line)
- ***Max. Angle of Wrist (relative to a vertical line)***
  max. angle the wrist is allowed to have (relative to a vertical line)
- ***Filter Max. Angle of Tcp (relative to a vertical line)***
  Filter out grasps whose tcp-angle is above the max. angle the tcp is allowed to have (relative to a vertical line)
- ***Max. Angle of Tcp (relative to a vertical line)***
  max. angle the tcp is allowed to have (relative to a vertical line)

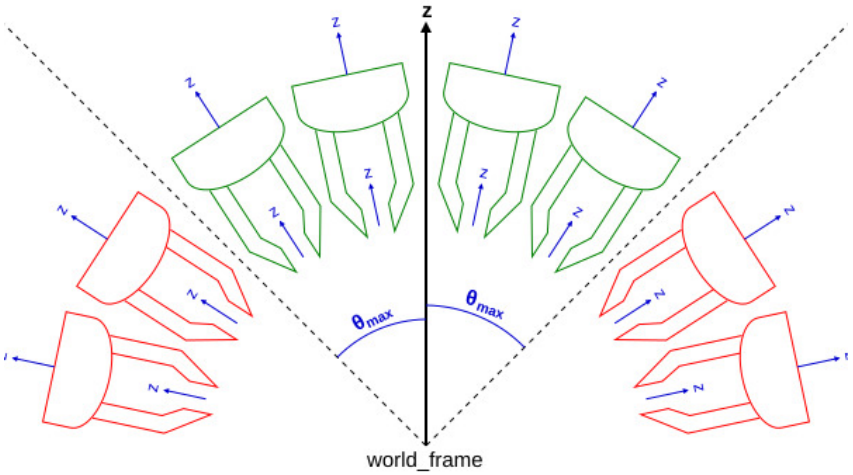


Fig. 33: Max. Angle of Wrist/Tcp (relative to a vertical line)

Grasping

- ***Grasping distance***
  (maximum) distance from the grasp-pose for which the grasping-motion is straight in Cartesian space
- ***Minimum grasping percentage***
  minimum fraction of the grasping-distance that must be reached (collision-free)

Interpolation Functions

There are three kinds of Interpolation Functions:

- ***without Intermediate Points***
  The Standard Interpolation, where the entire trajectory from start- to approach-pose (or, during retract, from extract- to end-pose) is interpolated in joint-space.
- ***Cartesian Intermediate Points***
  In the cartesian interpolation intermediate points are added between start- & approach-pose (retract- & goal-pose) in equal distance on a straight line between the two endpoints. The subparameter *Number of Points* is used to set the number of intermediate points. The intermediate points only define a position, but not the orientation of the Tcp. The subparameter *Rotation Type* allows for three settings, determining how the orientation of the Tcp shall change during the robot motion:

  a) *At Transit-Pose*: The orientation stays the same during position-changes and only changes at the transit-pose (start-pose for approach / end-pose for retract)

  b) *During Motion*: The orientation is interpolated during the robot-motion (along with the position).

  c) *In Box*: The orientation stays the same during position-changes and only changes at the approach-/extract-pose.

- ***Split Joints***

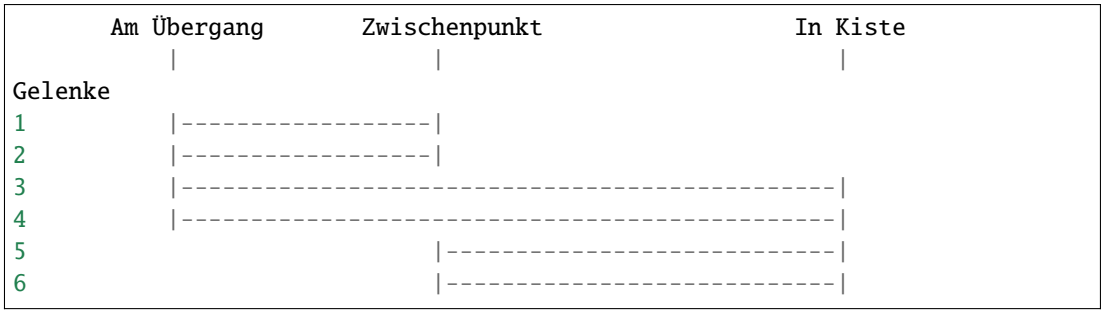
  This type of interpolation allows for interpolating some joints before others by splitting the interpolation into two segments. Each joint may be interpolated in the first segment, the second segment or over both segments. If all joints are interpolated over both segments, that's just ordinary interpolation.

  If one or more joints are interpolated in both segments, a percentage is needed to determine, at what point relative to the overall interpolation the other joints are to start/finish interpolation (see diagram below). We thus require two lists of joint-indices (one for each segment) and in some cases a percentage (when a joint occurs in both lists). Note: If a joint-index is not contained in either list, it will be included in both.

  For example

```
Gelenke an Übergang: [1, 2, 3]
Gelenke in Kiste: [3, 5, 6]
Prozentsatz: 40%
```

  will result in the following interpolation:



  Note that joint 4 is interpolated over both segments just like joint 3, as it is in neither list and therefore added to both. Note also, that the intermediate joint-position consists of the following positions:

  (a) the In-Box-positions of joints 1 and 2,

  (b) the At-Transit-positions of joints 5 and 6, and

  (c) the intermediate positions of joints 3 and 4, where each intermediate position is a 40%-interpolation between the At-Transit- and the In-Box-positions of those joints.

  A more realistic example is

```
Name: TurnGripperAtTransit
Gelenke an Übergang: [6]
Gelenke in Kiste: [1, 2, 3, 4, 5]
```

  This will turn the last joint (the gripper-joint) at the transit-pose and use the remaining joints to drive to/from the grasp-pose. This can be useful, when wanting to avoid turning the gripper inside the box. Another option with similar effect is to move joint 5 to the Joints At Transit too.

Approach

- ***Approach Interpolation-Function Order***
  Order in which the listed interpolations-functions are tested. Functions that aren't listed, are not tested. For a description of the interpolation-functions see *Interpolation Functions*.
- ***Disallow Free Planner for Approach***
  Prevents free planer from being used. In this case only interpolation-functions are tested.

Withdrawal

- ***Withdrawal distance***
  (maximum) distance from the grasp-pose for which the extract-motion ist straight in Cartesian space
- ***Minimum withdrawal percentage***
  minimum fraction of the extract-distance that must be reached (collision-free)
- ***Allow Collisions with other Objects & Octomap***
  allowed collisions between attached object and other objects as well as the octomap during Extract
- ***Allow Max-Depth Collisions***
  during extract, allow collisions at grasp-point up to a given collision-depth
- ***Max. Allowed Collision-Depth***
  max. collision-depth that is allowed during extract
- ***Specific Max. Allowed Collision-Depth***
  allows setting specific max-allowed-collision-depth for a pair of objects. Possible objects are all products, boxes and obstacles.
- ***Extract Function Order***
  Set the order, in which various extract-functions are tried for each pick
- ***Extract Num. XY-Directions***
  For the extract-functions "At Angle" and "Sideways" N extract-function are created, each with one of N directions equally distributed in a circle, where the 0°-direction points towards the center of the box. This parameter sets the number of such generated functions (N).
- ***Sort Extracts By***
  If this parameter is set, rather than testing the corresponding retract-function after each successful extract-function, first all extract-functions are tested and then sorted by this criterion. Then the retract-functions are tested until a valid one is found.

Return journey

- ***Disable Retract***
  stop planning after extract, making the retract-pose the final-pose
- ***Retract Interpolation-Function Order***
  Order in which the listed interpolations-functions are tested. Functions that aren't listed, are not tested. For a description of the interpolation-functions see *Interpolation Functions*.
- ***Disallow Free Planner for Retract***
  Prevents free planer from being used. In this case only interpolation-functions are tested.

Position Equality

- ***Joint-Space-Interpolation Max. Translation Diff.***
  max. translation-difference between two points to not require further intermediate point during joint-space interpolation. Determines how detailed a trajectory is sampled for checking collisions.
- ***Joint-Space-Interpolation Max. Rotation Diff.***
  max. rotation-difference between two points to not require further intermediate point during joint-space interpolation. Determines how detailed a trajectory is sampled for checking collisions.
- ***Cartesian Interpolation Max. Translation Diff.***
  max. translation-difference between two points to not require further intermediate point during cartesian interpolation. Determines how detailed a cartesian trajectory is created and sent to the robot.
- ***Cartesian Interpolation Max. Rotation Diff.***
  max. rotation-difference between two points to not require further intermediate point during cartesian interpolation. Determines how detailed a cartesian trajectory is created and sent to the robot.

The parameters for the cartesian interpolation may affect the robot speed during grasping / extraction. Smaller thresholds lead to more points, which may lead to slower robot motion.

The parameters of the joint-interpolation may affect the computation time for the pick-search. The value should be adapted to the size of the product, the box, and the robot. For thin tin plates for example a value of thickness/3 may be suitable, while for large parts one collision-check per cm may be enough.

Contacts

- ***Number of Contacts during Collision-Check***
  Maximum total number of contacts to be computed during a single collision-check (sum over all collision-pairs). Should be large (~100k) if collisions with octomap are likely, otherwise can be lower (~100). Larger values can lead to a slower pick-search.
- ***Max. Contacts per Collision-Pair***
  Maximum total number of contacts per collision-pair to be computed during a single collision-check. Should be large (~10k) if collisions with octomap are likely, otherwise can be lower (~30). Larger values can lead to a slower pick-search.

Search-Strategies

Pick Search

- ***Search-Strategies***
  Search-Strategy-Parameter-Set for each Grasp-Category
- ***Pick-Search Timeout***
  Timeout before pick-search is stopped
- ***Successful Picks per Object Before Deferral***
  When this number of successful picks has been reached for an object, picks of other objects are prioritized for the remaining pick-search.

- ***Max Successful Picks In Total***
  The pick-search is stopped after this number of successful picks (overall).

Conditions

- ***Requirements for Accepting Pick***
  Requirements a pick must meet to be accepted, see *Conditions*.

Extra-Time

- ***Extra-Time***
  Extra amount of time the pick-search is running, while the extra-time criteria have not been fulfilled yet by the best pick
- ***Requirements for extra search-time***
  Requirements a pick must meet to be used immediately (not just after the extra time), see *Conditions*.

Ordered Sorting Criteria:

For a description of each criterion see *Best handles*.

- ***Ordered Sorting Criteria***
  (order of) criteria to be used for sorting the successful picks
- ***Bin-Sizes***
  Bin-sizes to use for the binning-criteria
- ***Weights***
  Weights to use for *weighted sum of bins* (one of the potential order-criteria). Only the relative size of the weights matters, not the size of the numbers. The weights are only relevant if *weighted sum of bins* is in the list of order-criteria.

Skip Object Search

- ***Allow Skipping New Object-Search***
  allow skipping the next object-search, if the following requirements are met
- ***Min. Valid Picks***
  min. number of valid picks (overall) to allow skipping the next object-search
- ***Min. Valid Picks Per Object***
  min. number of valid picks (per object) to allow skipping the next object-search

- ***Min. Valid Objects***
  min. number of valid objects to allow skipping the next object-search

Trajectory-Distance Threshold

The trajectory-distance threshold is determined using the trajectory distances of the previous successful grasps. The mean value and the standard deviation are calculated from these. The threshold is (mean value + *Num. Std. Dev standard deviation). Furthermore, there is the parameter *Fixed Threshold*, which can be used to set a fixed threshold value.

- ***Threshold Default***
  Default-value for the trajectory-distance-threshold (used for binary binning in pick-selection-order)
- ***Num. Std. Dev***
  Number of standard deviations the trajectory-distance-threshold must lie above the mean
- ***Fixed Threshold***
  fixed value for the trajectory-distance-threshold (makes the two parameters above obsolete)

Position Equality

- ***Box Pose Max. Translation Diff.***
  in box-search: If the translation-difference to the previous pose is larger than this threshold, the transit-pose is re-calculated and all grasps are discarded.
- ***Box Pose Max. Rotation Diff.***
  in box-search: If the rotation-difference to the previous pose is larger than this threshold, the transit-pose is re-calculated and all grasps are discarded.
- ***Transit Pose Max. Translation Diff.***
  in box-search: If the translation-difference to the previous pose is larger than this threshold, the transit-pose is re-calculated and all grasps are discarded.
- ***Transit Pose Max. Rotation Diff.***
  in box-search: If the rotation-difference to the previous pose is larger than this threshold, the transit-pose is re-calculated and all grasps are discarded.
- ***Object Pose Max. Translation Diff.***
  in object-search: maximum rotation-difference to the pose of a previously found object to be considered the same object. This value only determines the order of the picks and can be set to relatively large values so that, for example, interlocked objects are not retried too often. To facilitate this, the sort-criterion `number of pick-attempts on object` (see *Available sorting criteria*) can be used both for sorting and for the extra-time.
- ***Object Pose Max. Rotation Diff.***
  in object-search: maximum rotation-difference to the pose of a previously found object to be considered the same object. This value only determines the order of the picks and can be set to relatively large values so that, for example, interlocked objects are not retried too often. To facilitate this, the sort-criterion `number of pick-attempts on object` (see *Available sorting criteria*) can be used both for sorting and for the extra-time.
- ***Pick Max. Translation Diff.***
  in object-search: maximum translation-difference to the pose of a previously found object to be allowed to reuse its picks.

- ***Pick Max. Rotation Diff.***
  in object-search: maximum rotation-difference to the pose of a previously found object to be allowed to reuse its picks. This value influences the picking accuracy and should not be too large.

2.12 Scene management

2.12.1 Overview

Scene objects describe objects and regions that are not included in the static description of the cell. These can be boxes, pure regions (so-called ROIs, "regions of interest") or optional obstacles.

A scene object always has a unique name and a (default) pose. Other properties depend on the type of object. If an object has an "extension" (height, width, length), please note that the x, y and z coordinates of the default pose are the center coordinates of the region/object.

Hint: Before Mikado 3.0, the z-coordinate indicated the height of the bottom of the object. Since Mikado 3.0, the z-coordinate refers to the center of the object.

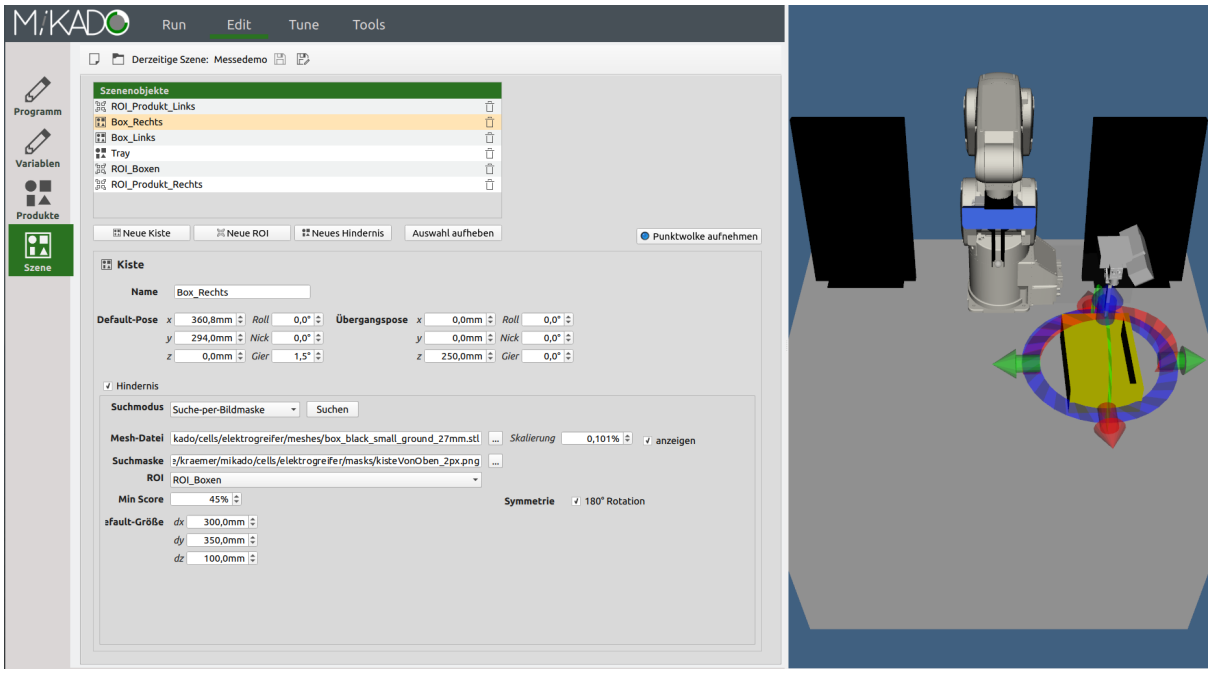


Fig. 34: Screenshot of scene management (example of a recording box)

The default pose of a scene object can either be defined via the input fields or using the interactive markers in the visualization. If you click on the colored arrows, the position can be moved by holding down the mouse button. The colored rings allow you to rotate the rotation around the respective axis of the scene object.

Hint: The colored control elements can only be touched where they are not covered by another (partially transparent) object.

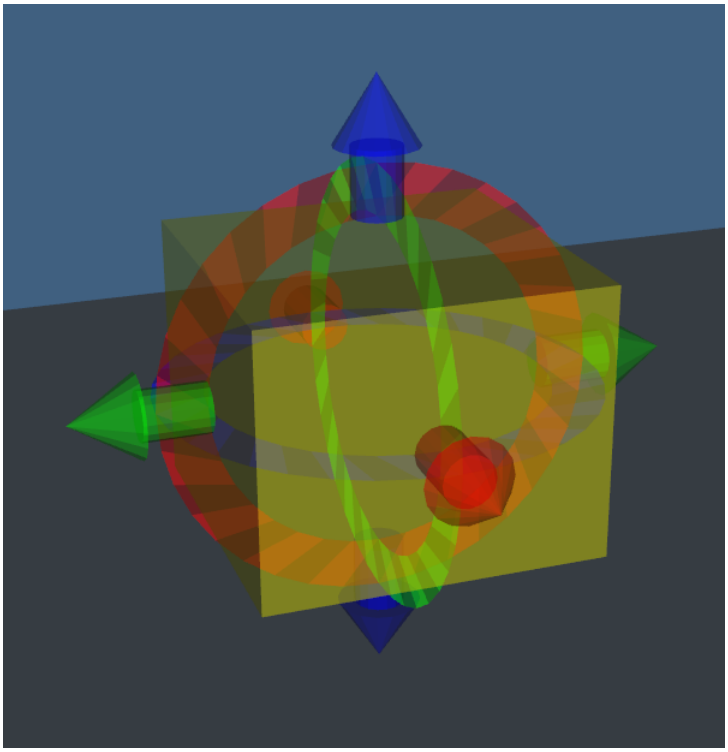


Fig. 35: Interactive marker for scene objects (control elements in red, green and blue)

Hint: The input fields define the rotation values directly in world coordinates. If the interactive markers are used, the rotation values are converted directly into roll, pitch and yaw and several values may change simultaneously. This enables a more intuitive definition of the rotation setting via the interactive markers.

2.12.2 Obstacle

An *obstacle* is a physical object in the scene. It can be viewed as fixed or searched for using 2D search.

2.12.3 Box

Boxes are used in the *pick search* to describe the environment of the parts to be gripped. They contain the so-called **transit pose** which acts as the start and end pose for the grasp planning. The **transit pose** is specified relative to the pose of the box, not relative to the robot.

Boxes with an interfering contour (i.e. with an activated **obstacle**) from which products are to be picked always require a mesh file for collision avoidance.

Hint: If there is no physical box, e.g. because it is being picked from a table, the hook can be removed at *obstacle*.

Box search

For boxes that are not fixed in exactly the same position, there is a layer tracking function using 2D search. The top edge of the box is usually searched for, as this remains recognizable regardless of the box fill level, while the inner walls and base, for example, can be heavily covered by products or flexible packaging materials.

A 2D mask must be specified for the 2D search. This mask can be created in the Mikado application as follows:

1. Create **ROI** This should mainly only contain the upper edge of the box in the Z extension and cover the entire area in which the box can be located in the X and Y directions. The ROI may also be used in the program sequence to search for boxes again.

   When creating the mask, the box should be aligned as orthogonally as possible in the image. Select the appropriate ROI under search parameters.

2. *Visualize* Select the Camera tab and select **projected_heightmap** in the selection menu. Then click on **Search**.

3. **Save** Right-click on the preview image to save either the raw version or a cropped and cropped version.

4. **Post-processing** The box mask should be exactly at right angles to the image. Variable noise and therefore incorrect contours should be reworked and removed or corrected. When searching for crates with variable height or with a high fill level where parts may protrude into the search ROI area, these areas should be masked out as transparent using the alpha channel. They are then not included in the search evaluation. The cropped variant masks out all pixels that do not contain the edge of the box by default.

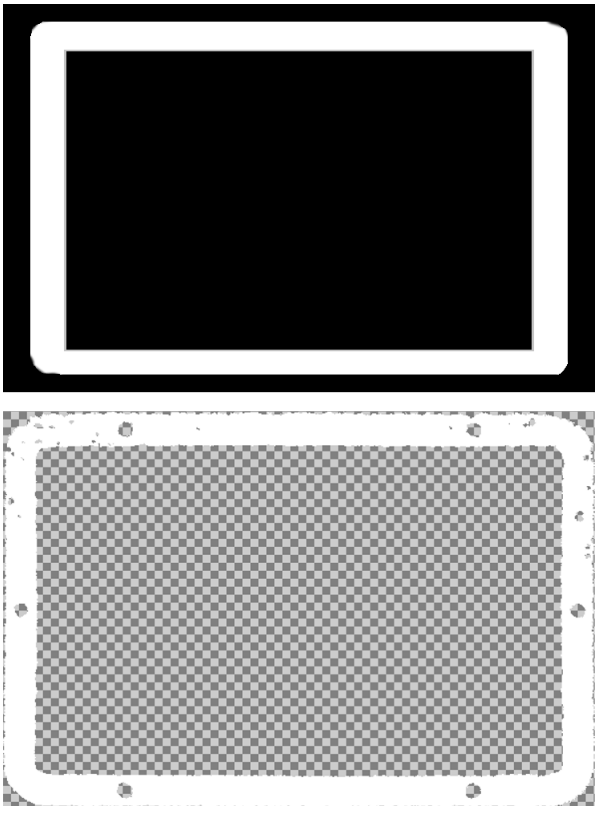


Fig. 36: Two examples of a 2D search mask for a crate. Above: the black background is included in the evaluation. Below: the transparent background is ignored during the search, any height values can be entered at these points.

In the example masks shown here, a crate is searched for using the top edge. In the 2D projection of the height data, the top edge of the crate stands out clearly from the surrounding floor and can therefore be used as a search mask. For the mask search, please note that the center of the mask represents the center of the object.

To include an obstacle in the collision avoidance, it is possible to specify a mesh file that describes the object. The origin of a mesh file must be centrally located in the base of the object.

If no mesh file is specified, the object can be described as a cuboid: **placeholder size**: Length **dx**, Width **dy**, Height **dz**. All obstacles with the search mode Search by image mask are initially considered as placeholder cuboids in the path planning until the first successful search operation in the program sequence. This cube should therefore cover the entire area within which the obstacle could potentially be located.

The parameters of the 2D search:

- **ROI**

  The ROI used to search for the obstacle can be selected in this dropdown. First create an *ROI* so that it is listed here. Make sure that the ROI is chosen generously in x- and y-extension. For good results, your search mask must be able to be mapped completely (i.e. including the border area) in the area of the ROI.

- **Min Score**

  Confidence value for the expected minimum match of search mask and camera image. This is output under **Tools > Logs** or in the console. A high value can lead to failed searches.

- **180° symmetry**

  If the object has rotational symmetry around the z-axis, this option can be activated. This property can be used to define clear transition poses despite symmetry. For example, you can activate this property for rectangular boxes.

2.12.4 Region of Interest (ROI)

An ROI describes a region in space - usually without an associated physical object. It is required, for example, in the object search to restrict the search space within which objects are to be searched for. The use of an ROI serves on the one hand to improve performance and on the other hand to reduce false finds (e.g. by hiding the background). In addition to its default pose, an ROI has size properties: length **dx**, width **dy**, height **dz**. If an ROI is selected, it has a direct effect on the following recordings and object searches.

Table 2: Functionality of a region of interest (ROI)

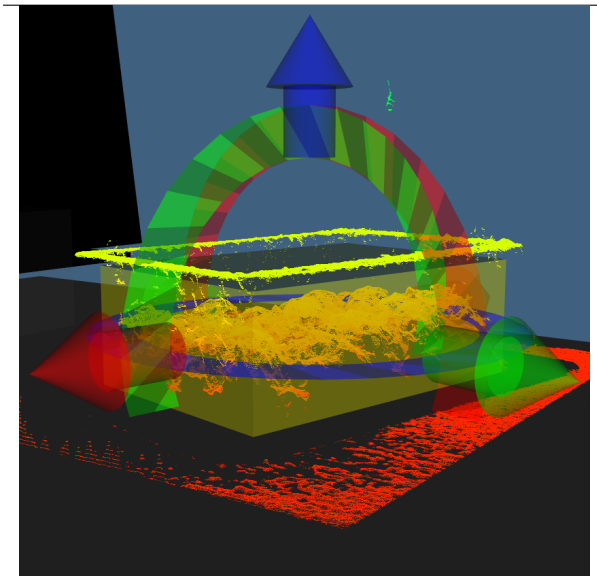


Fig. 37: Point cloud without ROI applied.

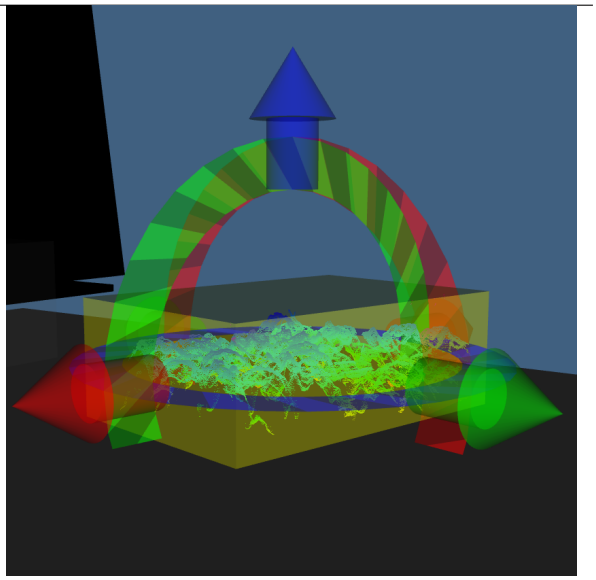


Fig. 38: Point cloud with ROI applied.

Hint: The pose of an ROI can be adjusted at program runtime using the *Set scene object position* task. This is useful, for example, to adapt the ROI to the pose of a tracked box. It can be reset to its original pose using the *Reset scene object position* task.

2.13 Object recognition tool

To optimize the recognition of your products in 3D data, you can change the object search parameters live in the object recognition tool. You can inspect the results of the changes directly. This tool also offers various views that allow you to understand the results and take measures to achieve better results.

You can use the object recognition tool by going to 'Tune' → 'Recognition' in the main menu. You will now see input options and buttons on the left and a visualization of the scene on the right (if not, switch to the 'Visualization' tab at the bottom right).

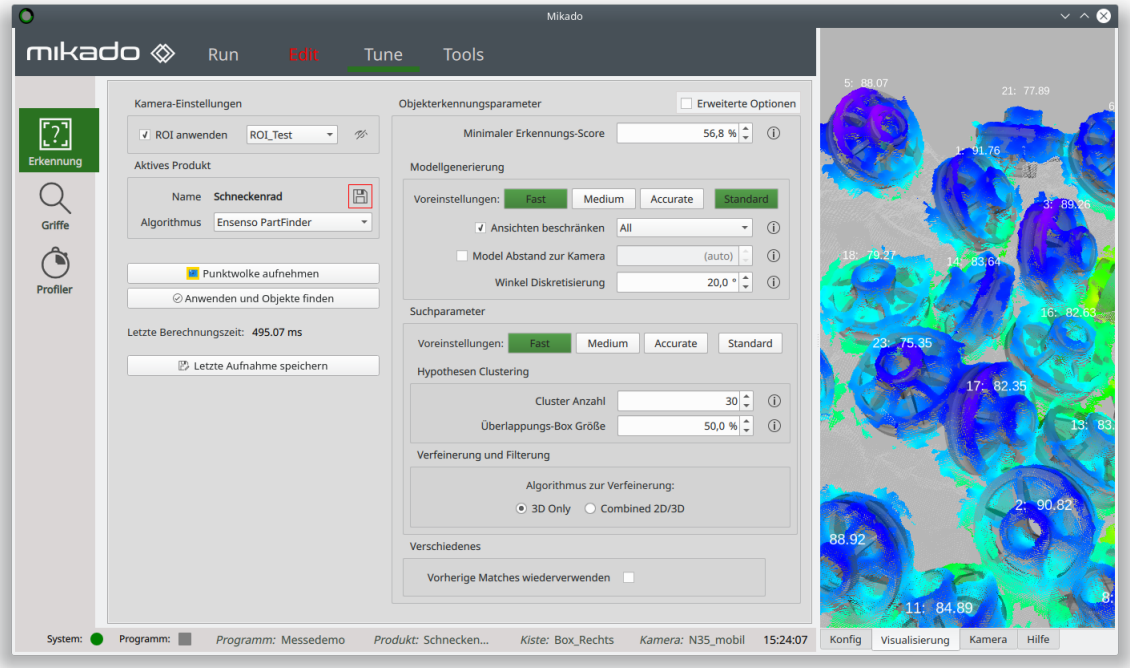


Fig. 39: Screenshot of object recognition tool

Note: The tool cannot be used while a program is running.

2.13.1 Prerequisites

Before using the object detection tool, make sure that the following steps have been carried out:

1. **Configure the camera according to the requirements**
   Before you optimize the parameters for the object search, you should ensure that the 3D data is of sufficiently good quality. Look at the recorded point cloud to first check whether the hardware setup meets the requirements. If the data is very noisy or incomplete, create a new, suitable *Camera configuration*.

2. **Create product**
   To search for an object, it must first be created as a product in the *Product management* (see also tutorial *A new product up to object recognition*). Mandatory product parameters for using the object recognition tool are:

   - *STL file / STL file (recognition)"*
   - *Search algorithm*. We recommend using Ensenso Partfinder.
   - Set the product to active, see *Active product*

3. **Create search area (optional)**
   To avoid false finds, it is advisable to reduce the search area (see *Region of Interest (ROI)*) to the expected area. For example, no components can be located in the crate wall or in the crate floor, which is why the wall and floor should be excluded from the search.

2.13.2 First steps

By clicking on the 'Record point cloud' button, the 3D camera creates a new image and records the point cloud in the *Visualization of the scene* on the right. After a recording has been made, the object search can be started. A click on 'Apply and find objects' triggers the search for the currently active product and displays the results in the visualization. The total calculation time of the last search is also displayed under the buttons (*Last calculation time*).

2.13.3 Camera selection

On the one hand, a camera is selected here (provided that Mikado Control is used with several 3D cameras), and on the other hand, a specific search area can be restricted using a *Region of Interest (ROI)*.

- ***Active camera*:**
  Select the 3D camera from which you would like to obtain data for object recognition. You can of course also use other cameras for the same object search later in the program. This option is only available for setups in which several cameras are defined.
- ***Apply ROI*:**
  If you activate this option, the generated point cloud is already filtered before the object search. Only points that lie within the specified ROI are displayed and used for the search. Try to define ROIs that are as restrictive as possible; this can greatly speed up detection and also ensure more reliable results.

2.13.4 Save last recording

If you would like to save the raw data from the last recording for diagnostic or analysis purposes, you can use the 'Save last recording' button. Select the desired directory in the selection dialog. Several recordings can also be saved in the same directory.

2.13.5 Visualization of the scene

The view settings can be used to change the display of the finds and the 3D data. The display of the point cloud before the object search is always the same: the points are colored according to their height (Z value).

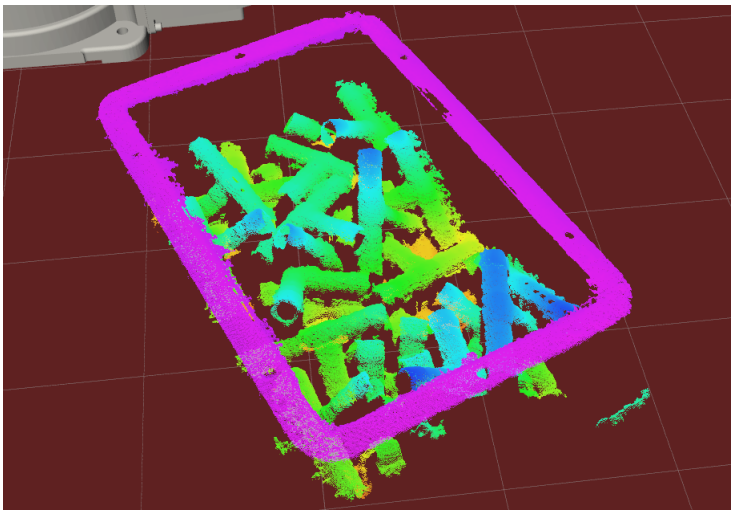


Fig. 40: Screenshot of object recognition tool point cloud after image acquisition

Found objects are drawn in grey in the visualization/point cloud as a CAD model and the corresponding scores are displayed. Points that are close to a find are shown in color (also according to their Z value). All other points are shown in gray.

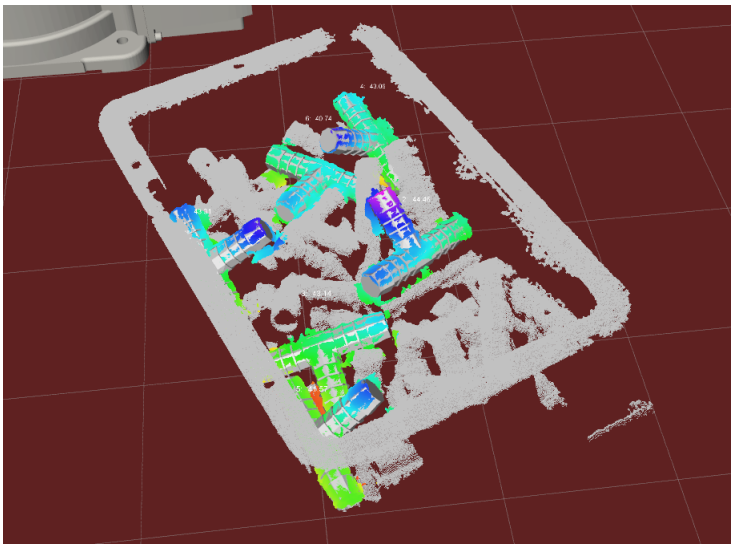


Fig. 41: Default view of the object recognition tool

Note: The points highlighted in color are cut out of the point cloud during normal program execution and are therefore excluded from collision monitoring. The size of the cutout can be set via the parameter *dilation_radius_object_cutting* in *object_recognition.yaml*.

2.13.6 Halcon-specific object recognition parameters

In this area, you can optimize the parameterization of object recognition. The initial values of the parameters correspond to the *configuration* of the project (see *object_recognition.yaml*). Please note that global parameters are specified in *object_recognition.yaml*, i.e. the parameters specified there apply to all products. You can adapt and test product-specific parameters using the object recognition tool.

Hint: All object detection parameters on this page are immediately active, even for the normal program sequence!

- ***Relative sampling distance (scene points)*:**
  Scanning distance of the 3D data relative to the diameter of the surface model (product). [0.0 - 1.0]
- ***Share of decisive points*:**
  Proportion of sampled scene points that are used as key points for approximate matching. [0.0 - 1.0]
- ***Scoring Abstand*:**

  Maximum distance between model point and scene point, so that the point pair contributes positively to the score.

  Typical: local sensor resolution/2.
- ***Use normal information*:**
  Activates/deactivates the use of normal information for approximate matching
- ***Calculation of the normal*:**
  Method for calculating the normals.
  - 'fast': fast procedure, quality of the standards is sufficient
  - 'mls': computationally intensive procedure with very good results
- ***Exact pose refinement*:**
  Activates/deactivates the refinement step of the object search
  - *Sub-sampling rate*: The higher the sub-sampling rate is selected, the fewer scene points are used for exact refinement.
  - *Number of steps*: Maximum number of iterations for exact refinement. The number of steps specified here represents a limit. If a stable result has already been determined earlier, the search is also terminated before this number is reached.
- ***Reuse previous matches*:**
  Activates/deactivates the use of findings from the previous object search.

  Hint: This setting should be deactivated during the optimization of the parameters, as the results of the individual object searches are otherwise dependent on each other. Only after you have found suitable values for all object detection parameters should you activate this function and fine-tune the following parameters.

  - *Sub-Sampling-Rate*: s.o.
  - *Number of steps*: see above.
  - *Threshold value of the relative distance*: Sets the distance threshold relative to the diameter of the model. Only 3D points whose distance to the object instance is less than this threshold are included in the refinement. Points that are further away from the instance are ignored. [0.0 - 1.0]
  - *Use normal information*: Activates/deactivates the use of normal information during matching

View (Halcon)

After an object search has been carried out, additional views can be selected for the Halcon search algorithm in addition to the standard view:

- ***Key points*:**
  Instead of displaying the complete point cloud, only the key points are visualized. In addition, the findings are displayed with the point model of the product - this makes it easier to recognize how a product is scanned for the approximate search. It is recommended that the parameters are selected so that approx. 50-100 key points are located on the object.

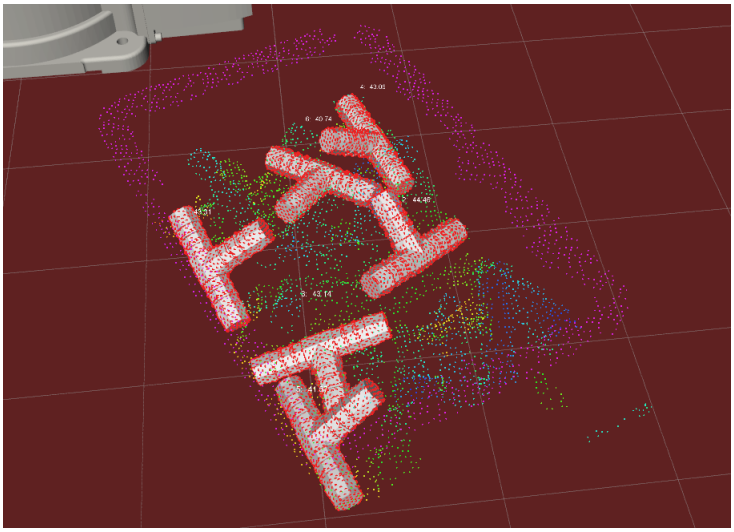


Fig. 42: Key points view of the object recognition tool

- ***Sampled points* (refinement):**
  The display is similar to the view of the key points. However, the point cloud and the product are displayed here in the form used for exact pose refinement.

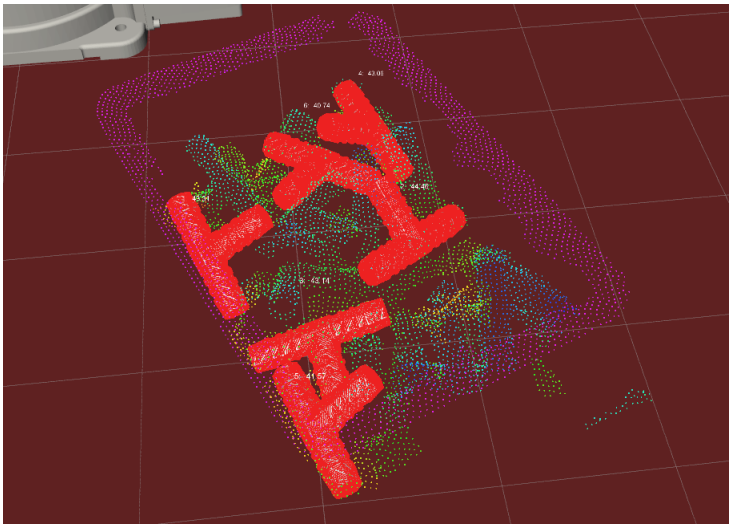


Fig. 43: Refinement view of the object recognition tool

2.14 Handle inspection

2.14.1 Overview

The handle inspection tool is used to analyze the search for potential gripping movements. If N objects are found in a region and M pick points (see *Gripping poses*) are defined for the corresponding *product*, there are basically NxM potential pick movements – one for each combination of object and pick point. Each of these movements is first checked for potential collisions. As soon as the rest of the program requires a gripping movement (e.g. when a "gripping object" task is reached), the "best" movement is selected from all the valid movements found up to that point. The "quality" of a movement depends on several criteria (see below), including the position-height of the associated object and the length of the gripping movement.

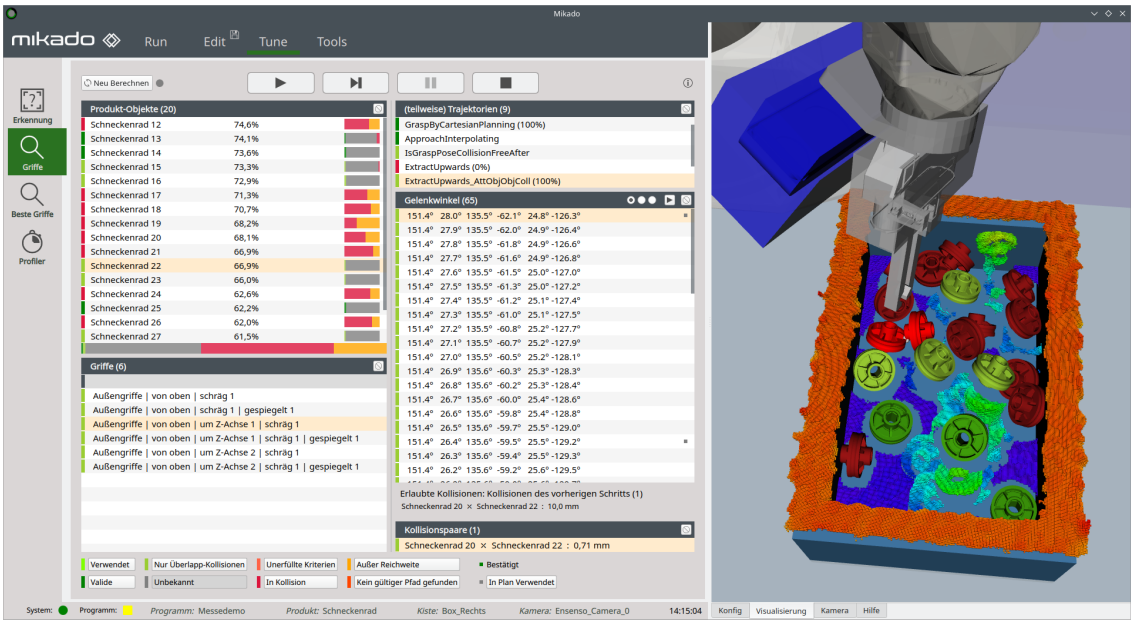


Fig. 44: Screenshot handle inspection

Prioritization of the handles found

see *Prioritization*

2.14.2 Inspection of the gripping movement search

The number of robot poses (states) to be inspected results from the product of the number of objects found (N) with the number of gripping positions per object (M) multiplied by the number of intermediate robot poses to be inspected per gripping movement. The latter varies with the gripping movement: For longer movements, more intermediate poses need to be checked than for shorter movements.

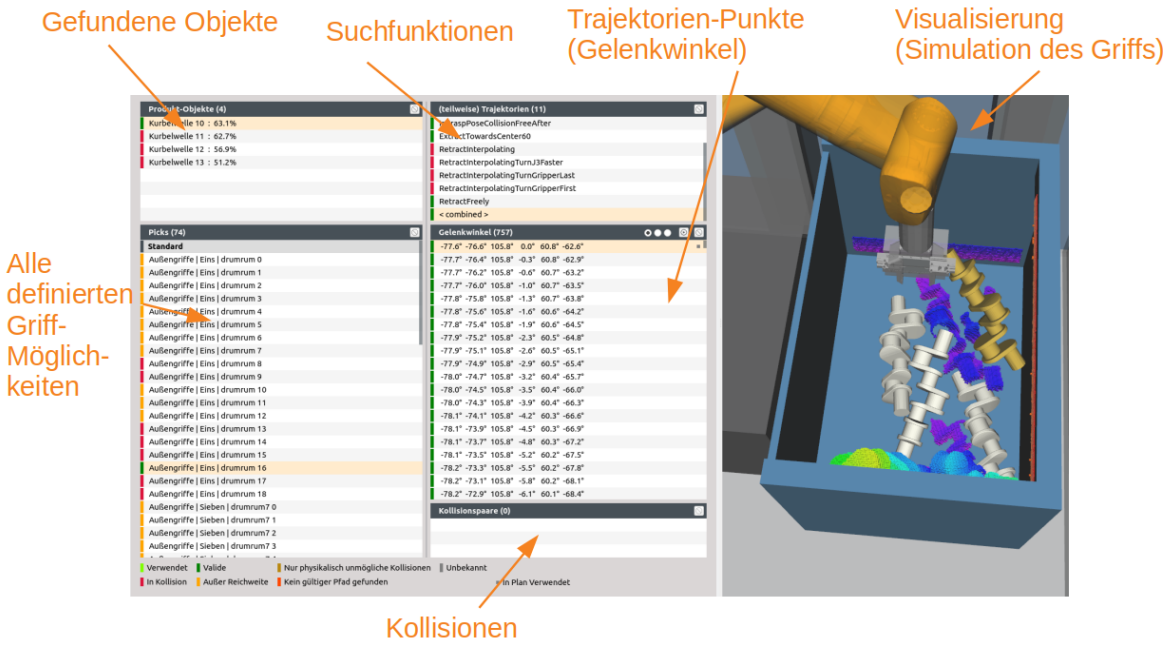


Fig. 45: Screenshot of handle inspection with description

As this quickly becomes confusing, a step-by-step selection is made in the inspection menu:

First, the object for which the potential gripping movement is to be displayed must be selected. The currently selected object is colored yellow in the visualization on the right - all other objects are displayed in blue.

If an object is selected, the corresponding handles are displayed in the next list (center left). The handle for which the potential gripping movement is to be displayed is selected here.

Once a handle – and therefore a potential gripping movement – has been selected, the individual search functions are displayed in the next list (top right).

If a search function is selected, the individual steps of the associated movement are displayed in the list below. The three radio buttons to the right of the list can be used to select what exactly is displayed in the list: The joint state of the robot, the pose of the end effector or the pose of the wrist. The movement can be played back in its entirety using the “Play” button at the top right of the list. If the selected step contains one or more collisions, these are displayed in the list at the bottom right.

If a collision is selected, the colliding parts are colored red in the display on the right (exception: the point cloud). Furthermore, the corner points of the collision area are marked with small white spheres.

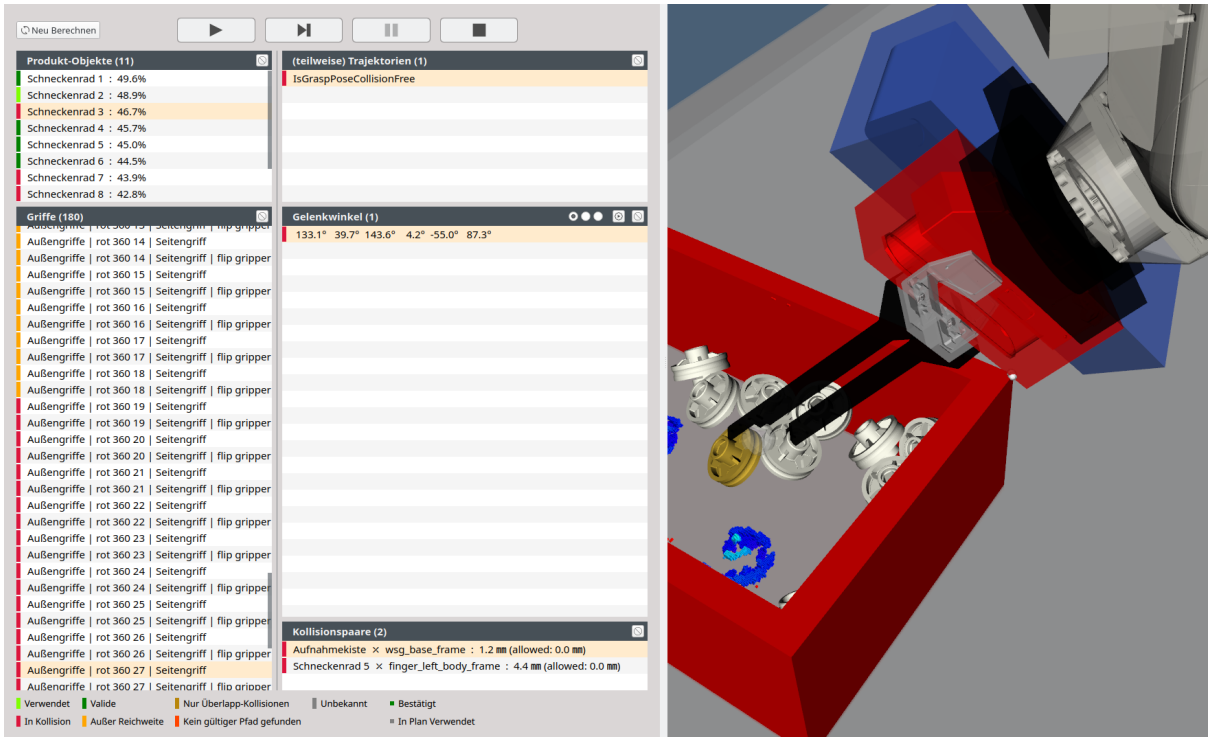


Fig. 46: Screenshot of collision display in the handle inspection

The special case in which the gripping position is outside the reach of the robot is characterized by the fact that the list of movements is empty. In this case, the robot is displayed in its basic state (all axis angles at 0°) in the visualization on the right-hand side. In addition, a gripper is shown in red at the unreachable gripping position in the visualization.

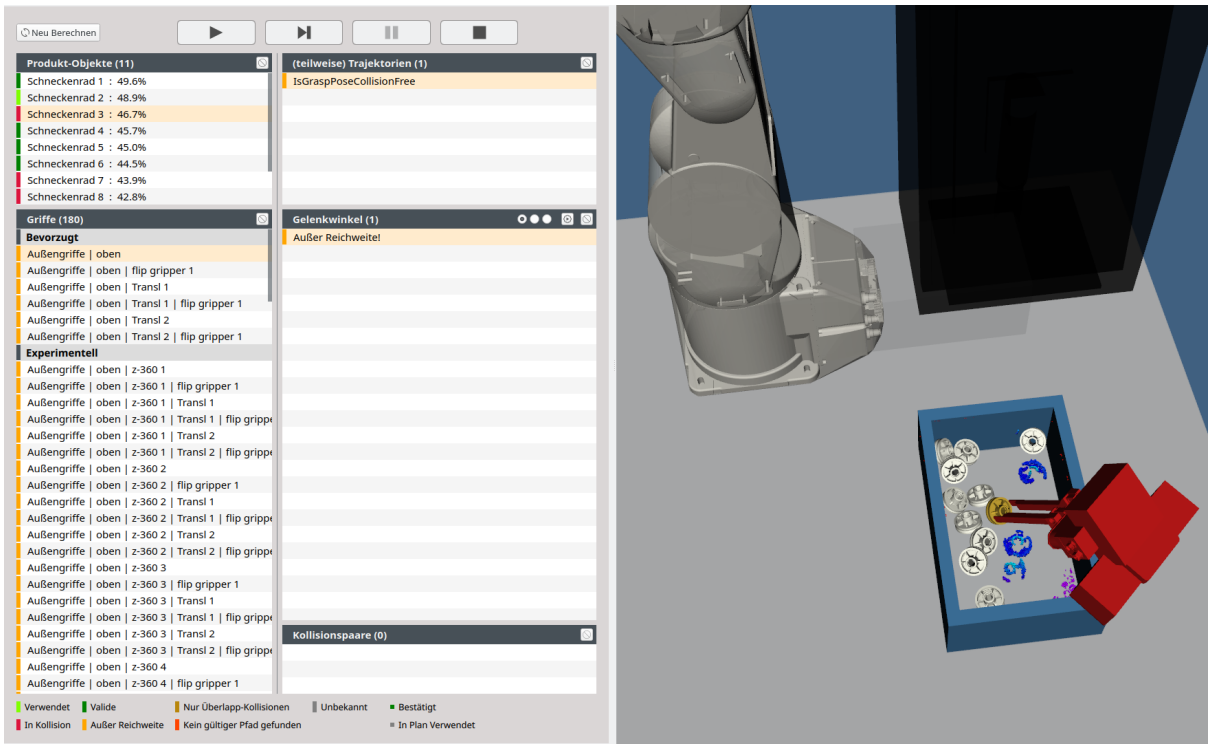


Fig. 47: Screenshot of unreachable handles in the handle inspection

Search status

A status is displayed in color for each object, each grab point, each search step, each search function and each movement step. The colors have the following meaning (see also the legend in the screenshots):

- light green: *used* (the object that was grabbed or the pick that was actually used)
- dark green: *valid*
- dark-gold: *valid, but with overlap collision(s)*
- dark red: *invalid due to collision(s)*
- matt-orange: *invalid, because not reachable*
- bright-orange: (only free path planning) *invalid, as no valid path found*
- gray: *not yet calculated*

In the list of trajectory points, there are also colored markings for some points on the right: * gray: *is sent to the robot controller* (In a simple interpolation run, for example, many intermediate points are checked, but only the start & end point are sent to the robot controller, as each robot controller takes over the interpolation independently). * Green: *has been validated* (If a previously found object is found again during an object search, the handles that have already been checked are checked again. This is particularly relevant if a handle was found using free path planning. In this case, free path planning is not carried out again during validation, but the previously found trajectory points are only checked again. Points marked in green indicate that the points have been validated).

In the case of movement steps, there are only the *valid* and *invalid* states. These indicate whether there is a collision for this movement step. In the other cases, there is also the *undefined* state.

For a search function, *undefined* indicates that the function has not yet been used (calculated). This occurs, for example, if a function already returns *valid* and therefore the other functions are no longer needed to determine whether there is a collision-free path.

For a search level, *invalid* indicates that all functions have been checked and have returned *invalid*. Conversely, *valid* indicates that at least one function has returned *valid*. *Undetermined* indicates that no *valid* function has yet been found, but that there are other functions in the *undetermined* state.

For a search movement, *valid* indicates that all associated search stages are *valid*. Conversely, *invalid* indicates that at least one search stage is in the *invalid* state. *Undetermined* indicates that no search stage is yet in the *invalid* state, but not all stages have been checked to completion.

For an object, *invalid* indicates that all associated potential grasping movements are *invalid*. Conversely, *valid* indicates that at least one *valid* associated gripping movement has been found. *indeterminate* indicates that no *valid* gripping movement has been found yet, but that there are other gripping movements in the *indeterminate* state.

For the grasping movement search as a whole, *invalid* indicates that all potential grasping movements are *invalid* for all objects found. Conversely, *valid* indicates that there is at least one *valid* grasping movement for at least one object.

2.15 Best handles

2.15.1 Prioritization

Prioritization takes place via the parameter *Ordered Sorting Criteria*. This parameter is a list of predefined strings, where each string specifies a sorting criterion. The list determines the criteria to be used for sorting and the order in which the criteria are prioritized.

Binning is used to compare all handles with regard to the various criteria. Binning involves grouping floating point values. This involves dividing the range of values from the respective best value into equally sized ranges (so-called bins). The bin value 0 is always the best bin (the first bin from the best value) and increasing bin values represent worse values. The size of these bins can be set using the parameter *Bin-Sizes*.

Available sorting criteria

The following predefined string values are available:

Table 3: Boolean criteria:

Bin name	Meaning of the bin value
`pick not tried before`	0: Handle not yet tried to grip 1: Handle already tried to grip
`is below trajectory-distance-threshold`	0: is below trajectory distance threshold 1: is over trajectory distance threshold value
`has no real collisions`	0: has no real collisions 1: has (permitted) real collisions
`has no fake collisions`	0: has no pseudo-collisions (see *Pseudo-collisions*) 1: has (permitted) pseudo-collisions

The Boolean criteria can be transferred to the bin values: 0 stands for "criterion fulfilled", 1 stands for "criterion not fulfilled".

Table 4: upwards limited criteria:

`grasp-category`	0: Preferred (B) 1: Standard (S) 2: Experimentell (E)

Categories can also be transferred to bin values. The best category (preferred) receives the value 0, the second best the value 1, etc.

Table 5: Binning criteria:

`number of pick-attempts on object (increasing)`	0: object not yet attempted to be gripped 1..: number of pick-attempts on object
`Score`	0: Bin of the highest matching score 1..: Bins for lower matching scores Default-Bin-Size: 0.1 (10%)

continues on next page

Table 5 – continued from previous page

`Wrist-Angle to Vertical`	0: Bin of lowest angle between wrist-pose and a vertical line (see *Angle to Vertical*) 1..: Bins for higher angles Default-Bin-Size: 0.087266 (5°)
`Tcp-Angle to Vertical`	0: Bin of lowest angle between tcp-pose and a vertical line (see *Angle to Vertical*) 1..: Bins for higher angles Default-Bin-Size: 0.087266 (5°)
`Travel distance`	0: Bin of the smallest travel distance (see *Trajectory-Distance Threshold*) 1..: Bins for larger travel distances Default-Bin-Size: 1.0471975512 (60° = 6 joints * 10°)
`total collision-depth`	0: bin for smallest total collision-depth (see *Collision-Depth*) 1…: bins for larger total collision-depths Default-Bin-Size: 0.001 (1mm)
`average collision-depth`	0: bin for smallest avg. collision-depth (see *Collision-Depth*) 1…: bins for larger avg. collision-depths Default-Bin-Size: 0.0001 (0.1mm)
`maximal collision-depth`	0: bin for smallest max. collision-depth (see *Collision-Depth*) 1…: bins for larger max. collision-depths Default-Bin-Size: 0.0002 (0.2mm)
`object-x (decreasing)`	0: Bin of the highest x-value of all object-poses (in world-coordinates) 1..: Bins for lower x-values Default-Bin-Size: 0.05 (5cm)
`object-y (decreasing)`	0: Bin of the highest y-value of all object-poses (in world-coordinates) 1..: Bins for lower y-values Default-Bin-Size: 0.05 (5cm)
`object-z (decreasing)`	0: Bin of the highest z-value of all object-poses (in world-coordinates) 1..: Bins for lower z-values Default-Bin-Size: 0.05 (5cm)

continues on next page

Table 5 – continued from previous page

`object-x (increasing)`	0: Bin of the lowest x-value of all object-poses (in world-coordinates) 1..: Bins for higher x-values Default-Bin-Size: 0.05 (5cm)
`object-y (increasing)`	0: Bin of the lowest y-value of all object-poses (in world-coordinates) 1..: Bins for higher y-values Default-Bin-Size: 0.05 (5cm)
`object-z (increasing)`	0: Bin of the lowest z-value of all object-poses (in world-coordinates) 1..: Bins for higher z-values Default-Bin-Size: 0.05 (5cm)
`abs(object-x) (increasing)`	0: Bin of the lowest absolute x-value of all object-poses (in world-coordinates) 1..: Bins for higher absolute x-values Default-Bin-Size: 0.05 (5cm)
`abs(object-y) (increasing)`	0: Bin of the lowest absolute y-value of all object-poses (in world-coordinates) 1..: Bins for higher absolute y-values Default-Bin-Size: 0.05 (5cm)
`abs(object-x) (decreasing)`	0: Bin of the highest absolute x-value of all object-poses (in world-coordinates) 1..: Bins for lower absolute x-values Default-Bin-Size: 0.05 (5cm)
`abs(object-y) (decreasing)`	0: Bin of the highest absolute y-value of all object-poses (in world-coordinates) 1..: Bins for lower absolute y-values Default-Bin-Size: 0.05 (5cm)
`Tcp-x (decreasing)`	0: Bin of the highest x-value of all Tcp-poses (in world-coordinates) 1..: Bins for lower x-values Default-Bin-Size: 0.05 (5cm)

continues on next page

Table 5 – continued from previous page

`Tcp-y (decreasing)`	0: Bin of the highest y-value of all Tcp-poses (in world-coordinates) 1..: Bins for lower y-values Default-Bin-Size: 0.05 (5cm)
`Tcp-z (decreasing)`	0: Bin of the highest z-value of all Tcp-poses (in world-coordinates) 1..: Bins for lower z-values Default-Bin-Size: 0.05 (5cm)
`Tcp-x (increasing)`	0: Bin of the lowest x-value of all Tcp-poses (in world-coordinates) 1..: Bins for higher x-values Default-Bin-Size: 0.05 (5cm)
`Tcp-y (increasing)`	0: Bin of the lowest y-value of all Tcp-poses (in world-coordinates) 1..: Bins for higher y-values Default-Bin-Size: 0.05 (5cm)
`Tcp-z (increasing)`	0: Bin of the lowest z-value of all Tcp-poses (in world-coordinates) 1..: Bins for higher z-values Default-Bin-Size: 0.05 (5cm)
`abs(Tcp-x) (increasing)`	0: Bin of the lowest absolute x-value of all Tcp-poses (in world-coordinates) 1..: Bins for higher absolute x-values Default-Bin-Size: 0.05 (5cm)
`abs(Tcp-y) (increasing)`	0: Bin of the lowest absolute y-value of all Tcp-poses (in world-coordinates) 1..: Bins for higher absolute y-values Default-Bin-Size: 0.05 (5cm)
`abs(Tcp-x) (decreasing)`	0: Bin of the highest absolute x-value of all Tcp-poses (in world-coordinates) 1..: Bins for lower absolute x-values Default-Bin-Size: 0.05 (5cm)

continues on next page

Table 5 – continued from previous page

`abs(Tcp-y) (decreasing)`	0: Bin of the highest absolute y-value of all Tcp-poses (in world-coordinates) 1..: Bins for lower absolute y-values Default-Bin-Size: 0.05 (5cm)
`wrist-x (decreasing)`	0: Bin of the highest x-value of all wrist-poses (in world-coordinates) 1..: Bins for lower x-values Default-Bin-Size: 0.05 (5cm)
`wrist-y (decreasing)`	0: Bin of the highest y-value of all wrist-poses (in world-coordinates) 1..: Bins for lower y-values Default-Bin-Size: 0.05 (5cm)
`wrist-z (decreasing)`	0: Bin of the highest z-value of all wrist-poses (in world-coordinates) 1..: Bins for lower z-values Default-Bin-Size: 0.05 (5cm)
`wrist-x (increasing)`	0: Bin of the lowest x-value of all wrist-poses (in world-coordinates) 1..: Bins for higher x-values Default-Bin-Size: 0.05 (5cm)
`wrist-y (increasing)`	0: Bin of the lowest y-value of all wrist-poses (in world-coordinates) 1..: Bins for higher y-values Default-Bin-Size: 0.05 (5cm)
`wrist-z (increasing)`	0: Bin of the lowest z-value of all wrist-poses (in world-coordinates) 1..: Bins for higher z-values Default-Bin-Size: 0.05 (5cm)
`abs(wrist-x) (increasing)`	0: Bin of the lowest absolute x-value of all wrist-poses (in world-coordinates) 1..: Bins for higher absolute x-values Default-Bin-Size: 0.05 (5cm)

continues on next page

Table 5 – continued from previous page

`abs(wrist-y) (increasing)`	0: Bin of the lowest absolute y-value of all wrist-poses (in world-coordinates) 1..: Bins for higher absolute y-values Default-Bin-Size: 0.05 (5cm)
`abs(wrist-x) (decreasing)`	0: Bin of the highest absolute x-value of all wrist-poses (in world-coordinates) 1..: Bins for lower absolute x-values Default-Bin-Size: 0.05 (5cm)
`abs(wrist-y) (decreasing)`	0: Bin of the highest absolute y-value of all wrist-poses (in world-coordinates) 1..: Bins for lower absolute y-values Default-Bin-Size: 0.05 (5cm)
`object-x rel. to box (decreasing)`	0: Bin of the highest x-value of all object-poses (in box-coordinates) 1..: Bins for lower x-values Default-Bin-Size: 0.05 (5cm)
`object-y rel. to box (decreasing)`	0: Bin of the highest y-value of all object-poses (in box-coordinates) 1..: Bins for lower y-values Default-Bin-Size: 0.05 (5cm)
`object-z rel. to box (decreasing)`	0: Bin of the highest z-value of all object-poses (in box-coordinates) 1..: Bins for lower z-values Default-Bin-Size: 0.05 (5cm)
`object-x rel. to box (increasing)`	0: Bin of the lowest x-value of all object-poses (in box-coordinates) 1..: Bins for higher x-values Default-Bin-Size: 0.05 (5cm)
`object-y rel. to box (increasing)`	0: Bin of the lowest y-value of all object-poses (in box-coordinates) 1..: Bins for higher y-values Default-Bin-Size: 0.05 (5cm)

continues on next page

Table 5 – continued from previous page

`object-z rel. to box (increasing)`	0: Bin of the lowest z-value of all object-poses (in box-coordinates) 1..: Bins for higher z-values Default-Bin-Size: 0.05 (5cm)
`abs(object-x) rel. to box (increasing)`	0: Bin of the lowest absolute x-value of all object-poses (in box-coordinates) 1..: Bins for higher absolute x-values Default-Bin-Size: 0.05 (5cm)
`abs(object-y) rel. to box (increasing)`	0: Bin of the lowest absolute y-value of all object-poses (in box-coordinates) 1..: Bins for higher absolute y-values Default-Bin-Size: 0.05 (5cm)
`abs(object-x) rel. to box (decreasing)`	0: Bin of the highest absolute x-value of all object-poses (in box-coordinates) 1..: Bins for lower absolute x-values Default-Bin-Size: 0.05 (5cm)
`abs(object-y) rel. to box (decreasing)`	0: Bin of the highest absolute y-value of all object-poses (in box-coordinates) 1..: Bins for lower absolute y-values Default-Bin-Size: 0.05 (5cm)
`Tcp-x rel. to box (decreasing)`	0: Bin of the highest x-value of all Tcp-poses (in box-coordinates) 1..: Bins for lower x-values Default-Bin-Size: 0.05 (5cm)
`Tcp-y rel. to box (decreasing)`	0: Bin of the highest y-value of all Tcp-poses (in box-coordinates) 1..: Bins for lower y-values Default-Bin-Size: 0.05 (5cm)
`Tcp-z rel. to box (decreasing)`	0: Bin of the highest z-value of all Tcp-poses (in box-coordinates) 1..: Bins for lower z-values Default-Bin-Size: 0.05 (5cm)
`Tcp-x rel. to box (increasing)`	0: Bin of the lowest x-value of all Tcp-poses (in box-coordinates) 1..: Bins for higher x-values Default-Bin-Size: 0.05 (5cm)

continues on next page

Table 5 – continued from previous page

`Tcp-y rel. to box (increasing)`	0: Bin of the lowest y-value of all Tcp-poses (in box-coordinates) 1..: Bins for higher y-values Default-Bin-Size: 0.05 (5cm)
`Tcp-z rel. to box (increasing)`	0: Bin of the lowest z-value of all Tcp-poses (in box-coordinates) 1..: Bins for higher z-values Default-Bin-Size: 0.05 (5cm)
`abs(Tcp-x) rel. to box (increasing)`	0: Bin of the lowest absolute x-value of all Tcp-poses (in box-coordinates) 1..: Bins for higher absolute x-values Default-Bin-Size: 0.05 (5cm)
`abs(Tcp-y) rel. to box (increasing)`	0: Bin of the lowest absolute y-value of all Tcp-poses (in box-coordinates) 1..: Bins for higher absolute y-values Default-Bin-Size: 0.05 (5cm)
`abs(Tcp-x) rel. to box (decreasing)`	0: Bin of the highest absolute x-value of all Tcp-poses (in box-coordinates) 1..: Bins for lower absolute x-values Default-Bin-Size: 0.05 (5cm)
`abs(Tcp-y) rel. to box (decreasing)`	0: Bin of the highest absolute y-value of all Tcp-poses (in box-coordinates) 1..: Bins for lower absolute y-values Default-Bin-Size: 0.05 (5cm)
`wrist-x rel. to box (decreasing)`	0: Bin of the highest x-value of all wrist-poses (in box-coordinates) 1..: Bins for lower x-values Default-Bin-Size: 0.05 (5cm)
`wrist-y rel. to box (decreasing)`	0: Bin of the highest y-value of all wrist-poses (in box-coordinates) 1..: Bins for lower y-values Default-Bin-Size: 0.05 (5cm)
`wrist-z rel. to box (decreasing)`	0: Bin of the highest z-value of all wrist-poses (in box-coordinates) 1..: Bins for lower z-values Default-Bin-Size: 0.05 (5cm)

continues on next page

Table 5 – continued from previous page

`wrist-x rel. to box (increasing)`	0: Bin of the lowest x-value of all wrist-poses (in box-coordinates) 1..: Bins for higher x-values Default-Bin-Size: 0.05 (5cm)
`wrist-y rel. to box (increasing)`	0: Bin of the lowest y-value of all wrist-poses (in box-coordinates) 1..: Bins for higher y-values Default-Bin-Size: 0.05 (5cm)
`wrist-z rel. to box (increasing)`	0: Bin of the lowest z-value of all wrist-poses (in box-coordinates) 1..: Bins for higher z-values Default-Bin-Size: 0.05 (5cm)
`abs(wrist-x) rel. to box (increasing)`	0: Bin of the lowest absolute x-value of all wrist-poses (in box-coordinates) 1..: Bins for higher absolute x-values Default-Bin-Size: 0.05 (5cm)
`abs(wrist-y) rel. to box (increasing)`	0: Bin of the lowest absolute y-value of all wrist-poses (in box-coordinates) 1..: Bins for higher absolute y-values Default-Bin-Size: 0.05 (5cm)
`abs(wrist-x) rel. to box (decreasing)`	0: Bin of the highest absolute x-value of all wrist-poses (in box-coordinates) 1..: Bins for lower absolute x-values Default-Bin-Size: 0.05 (5cm)
`abs(wrist-y) rel. to box (decreasing)`	0: Bin of the highest absolute y-value of all wrist-poses (in box-coordinates) 1..: Bins for lower absolute y-values Default-Bin-Size: 0.05 (5cm)

Table 6: User-defined binning:

`user-defined bin`	lower bin is better

The function `picksearch.setUserBins(bins: [int])` can be used in a JavaScript-Task to set a bin-value for every found object. The number of given `bins` has to exactly match the number of found objects. When `user-defined bin` is included in the `Ordered Sorting Criteria`, the bins set via the above function will be used during pick-sorting.

Table 7: weighted binning:

`weighted sum of bins`	lower weighted sum is better

If one criterion is not to be compared strictly after the next, but several criteria are to be combined, the `weighted_binning` parameter can be used. For this case, there is also the separate parameter `weights`, which allows setting the weights of the individual criteria. All criteria that are not listed in this parameter implicitly have a weight of 0. To calculate the weighted bin, the bins are calculated for all criteria whose weight is not 0, then multiplied by the respective weight and finally added together.

If two picks have the same (binning) value in all criteria, they are sorted according to the travel distance (without binning).

Table 8: Debug criteria (0 can be used as a condition):

`always 1`	All handles have a bin value of 1
`has real collisions`	Opposite of `has no real collisions`
`has (permitted) pseudo-collisions`	Opposite of `has no pseudo-collisions`

Angle to Vertical

The Angle to Vertical (tcp or wrist) can be used to restrict how inclined the gripper may be when grasping.

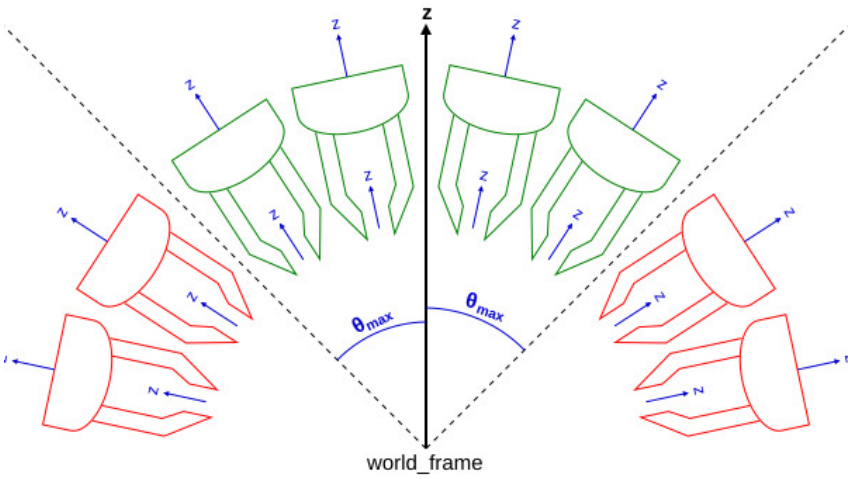


Fig. 48: Maximum Angle to Vertical

Conditions

The bin values can be used to implement conditions. For example, it may be that you only want to grab parts from the top layer. This means that you want the part for `z (decreasing)` (or `z rel. to box (decreasing)`) to have the bin value 0. This can be implemented using the parameter `Requirements for Pick`, by adding an entry for `z rel. to box (decreasing)` and setting the maximum value to `1 x bin-size` (for this criterion). For boolean criteria there is no value to be set, they must simply be fulfilled.

If instead you want the part to be in one of the top two positions, you would select a maximum value of `2 x bin-size`. In general for decreasing (increasing) criteria one sets the maximum (minimum) value a part or pick may have.

Handles that do not meet the requirements are discarded. In the pick inspection, they are given the color for "Unfulfilled criteria".

In addition to these "hard" constraints, "soft" constraints can also be set, which cause the pick search to run a little longer if no pick has yet been found that meets all the conditions. The parameter for this is `Requirements for Extra-Time`, whereby the additional time granted can be set via `Extra-Time`. The extra time starts from the first successful pick found (if it doesn't already meet the criteria).

If the parameter is not set, the following additional time conditions are used by default:

Object not yet attempted to pick?	
is below travel distance threshold	
Grasp-Category	1 (Preferred or Standard)

This means that as long as all valid grasps found are experimental, are above the travel threshold or belong to a part that has already been attempted to be grasped, the search for better picks continues – until the set additional time runs out.

Collision-Depth

For each collision, the collision-depth is the depth of how deep two colliding objects are overlapping. The *total collision-depth* is the sum of all collision-depths of all collisions of all trajectory-steps of the *Extract*-stage (see *Gripping process*) (the only stage that allows collisions). The *average collision-depth* is the *total collision-depth* divided by the number of trajectory-steps in the *Extract*-stage. The *max collision-depth* is the largest collision-depth of a single collision of all trajectory-steps.

Note that different partial trajectories of the *Extract*-stage can be of different length. This can lead to one partial trajectories having a larger *total collision-depth* than another, but at the same time a smaller *average collision-depth*.

Note further that the computation of all collision-depths significantly increases the computation time of the pick-search. Therefore it is recommended that these selection-criteria are only used when necessary. The computation-time is also influenced by the number of trajectory-points, see also *Interpolation Parameters*.

Pseudo-collisions

Pseudo-collisions are collisions of parts with each other before they are gripped. This can happen when parts are directly adjacent to each other. In this case, the (small) inaccuracy of the object recognition can lead to two parts being recognized as being inserted into each other. However, as this is only an artifact of object recognition and the parts cannot actually be inserted into each other, such collisions are classified as “pseudo-collisions”.

2.15.2 Best handles display

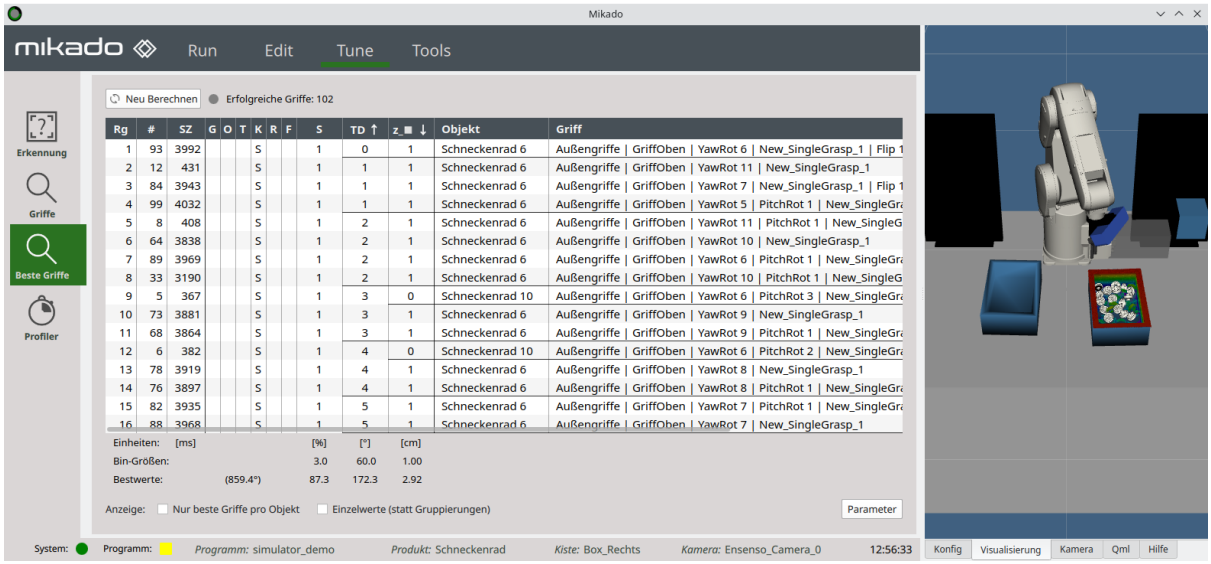


Fig. 49: Screenshot Best-Griffe

On the “Best handles” page, the valid handles found so far are listed in sorted order. The corresponding table consists of three column areas:

- The first three columns show the rank (Rg) within the sorting, the order in which the handles were found (#) and the search time (SZ) in milliseconds (from the start of the handle search).
- The last two columns indicate the names of the object that would be gripped with the handle and the handle itself.
- In between, the values of the respective handle are displayed in the respective search criteria. The order of the columns depends on the order of the aforementioned `pick_selection_order` parameter. Please note:
  - For Boolean criteria, an “X” is displayed if a criterion is not fulfilled. Otherwise the field is empty.
  - For unlimited criteria, the bin values are displayed. If the “Single values” checkbox (below the table) is ticked, the actual values are displayed instead.
  - The headings are abbreviated to save space. If you move the mouse over a heading, the whole name is displayed.
  - To make it easier to recognize from which criterion two consecutive handles differ, a dividing line is drawn between the handles from the first different grouping.
  - Double-click on a line to jump to the corresponding handle in the handle inspection.

Below the table, the units, the bin sizes and the respective best values are displayed for each binning column. Two display options can be set below this:

- By ticking “Only best handles per object”, the list is filtered so that only the best handle is displayed for each object.
- If you tick “Individual values (instead of groupings)”, the actual values are displayed instead of the bins.

At the bottom right, click on the parameter button to display the sorting parameters for the current search run. These can be copied to `pick_search.yaml` and changed there.

At the top left, as on the handle inspection page, there is a button to repeat the search. There is also a status display, which is green when the handle search is running and gray otherwise. The number of successful grips is displayed next to it.

2.16 Profiler

The runtimes of the individual tasks can be displayed on the Profiler page. The times are particularly interesting for tasks that execute robot movements or otherwise take up a lot of time and help you to optimize your process.

The interface shows you the complete program sequence similar to the *Status page*. The sequences and tasks are listed and the current status of a task is displayed. The last runtime of each individual task is displayed in milliseconds. Please note the following points:

- If the task has not yet been executed, the last runtime is 0 ms. For very simple tasks (e.g. copying a variable), the runtime is very short, so that 0 ms is also displayed.
- It may not be possible to display the runtime of the object and handle search itself, as this usually takes place parallel to the program. The duration of an object search can be determined using the *Object recognition tool*.
- The runtimes of the object search and handle search can vary greatly. The duration of the object search varies depending on the environment, ROI and parameterization. The handle search depends on the number of objects found and can also take longer if the objects are positioned unfavorably.

2.17 Logs

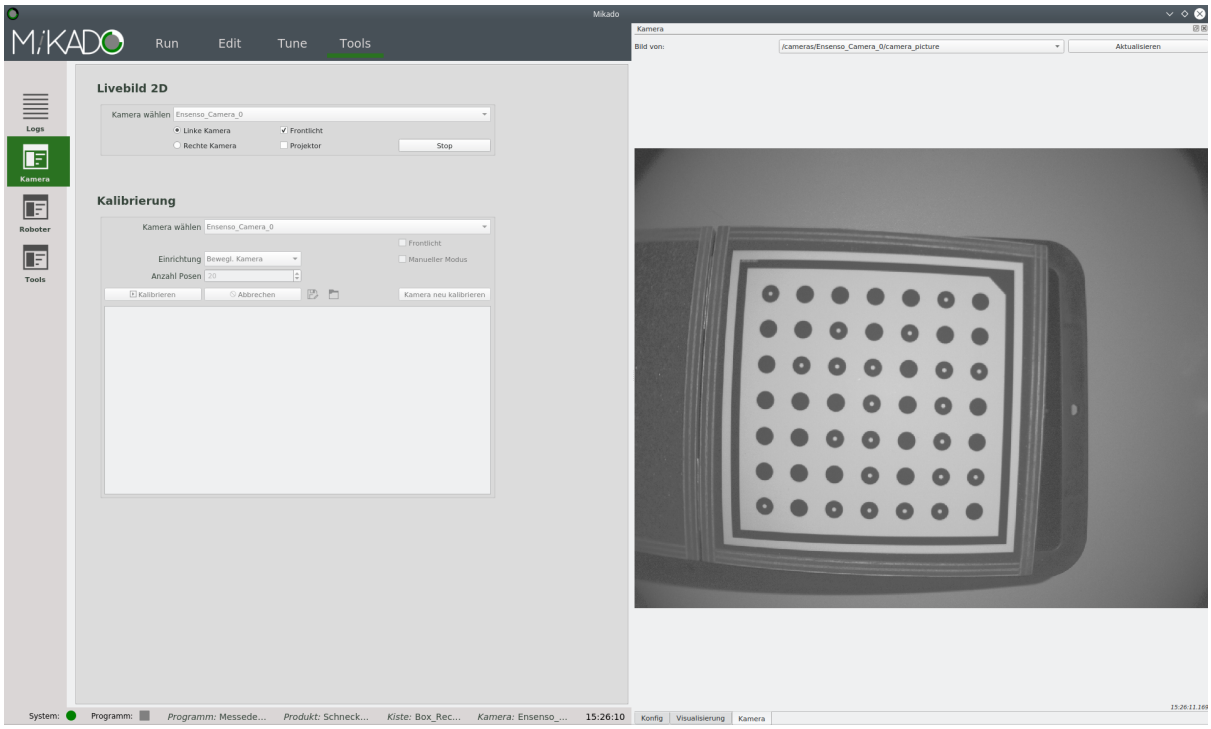
2.17.1 Overview

The last 10,000 log messages are displayed on the log page. It is also possible to filter these. The filter options are

- Select the log level of the messages to be displayed:
  - Debug
  - Warning
  - Info
  - Error
  - Fatal mistake
- Select the time period for which the messages to be displayed have been logged
- Text or pattern matching on messages
- Select the ROS nodes from which the messages originated

2.18 Camera settings

The camera settings page offers controls for a live view, direct view of the camera data and the option to calibrate the camera. The view can be found in the menu under ‘Tools’ → ‘Camera’.



Note: The tool provided here is only intended as a supplement to camera-specific tools. To use an Ensenso camera, for example, you must first configure the camera using the Ensenso software (NxView) (see *Camera configuration*). This will ensure that correct 3D data is captured using your 3D sensor. You can then use Mikado Control to check the images or to switch between different configurations.

2.18.1 Live image 2D

At the top of the interface are the controls for directly controlling the 2D image recordings. You can view the image using the *camera tab*.

- ***Select camera*:**
  If you have several cameras connected to the system, you can select the camera you want to use here
- ***Left camera / Right camera*:**
  With this selection, you can specify whether you want to display the live image from the left or right 2D camera of the stereo camera pair

- ***Front light*:**
  Activate this option if you want to actively illuminate the scene with the front light for every shot. If your model does not have a front light, this parameter is hidden and grayed out for file cameras.
- ***Projector*:**
  Activate the projector if you want to project the pattern onto the scene during a recording. This pattern supports the capture of 3D data. The selection option is grayed out for file cameras.
- ***Start or Stop*:**
  This button is used to start or stop live recording. Calibration cannot be performed while recording is started.

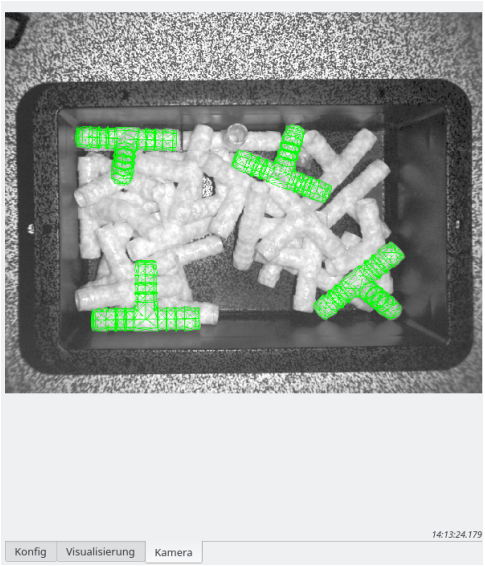
Hint: Changes to the camera parameters only affect the view. Starting the program while the 2D live image is still active will notify you that the live image is being switched off. The last active front light/projector settings are also reset.

2.18.2 Camera tab

You can view the image of a camera in the Camera tab. It is located on the right-hand side of the interface. If live recording is activated (see *Live image 2D*), you will see the live stream of the camera here.

- ***Image by*:**

  You can select the image channel in this selection field. If you have several cameras connected, you can change the camera here.
- ***Update*:**
  Click on this button to search for new picture channels. As a rule, however, picture channels should be fully available from the start of the program.



- A timestamp can be seen at the bottom right of the ‘Camera’ tab. Here you can see, for example, whether the live stream is currently activated.
- If an object search has been carried out (even while a program is running), the underlying camera image is displayed on the image channel of the corresponding camera with the findings marked. The findings are shown in green as a grid.
- You can switch to the live image at any time; the tabs for ‘Visualization’ and ‘Camera’ in the bottom right-hand side of the interface can be operated independently of the menu items.

- If the image channel is changed, the currently displayed image is discarded and only when new image data arrives is it displayed

2.18.3 Hand-eye calibration

This section describes how to perform a hand-eye calibration. The TCP calibration procedure is described in section *Calibrate TCP*. Mikado Control supports different calibration variants for hand-eye calibration (HAK), which are selected depending on the installation and cell configuration. Before a calibration, it must be ensured that the calibration pattern is in a suitable position and that the camera (or cameras) are looking as centrally as possible at the pattern. The following table provides an overview of the different variants.

Table 9: **Overview of the different HAK variants**

Moving camera	Automatic implementation for mobile camera
Fixed camera	Automatic implementation for fixed camera
Transfer	For multi-camera setup: Quick calibration of the secondary camera(s)
Set origin	For select cells or simulated robots: Set origin to pattern pose
Roboter	Outputs the transformation chain of the robot

Tip: Use the *Camera tab* and possibly the *Robot control* to align the camera.

Note: The position of the camera defined in *Cell geometry* has no influence on the hand-eye calibration.

User interface

The individual elements of the interface are explained below:

- ***Select camera*:**
  If you have several cameras connected to the system, you can select the camera you want to use here
- ***Setup*:**
  Select the setup method that corresponds to your setup here. If you change the setup type, additional / different buttons and input fields are displayed according to your selection. A detailed description of the individual types can be found under *Setup types*.
- ***Calibrate*:**
  This button starts the calibration

  Hint: During calibration, you can see the individual images in the *Camera tab* in the image channel of the camera to be calibrated. The position of the calibration pattern is also shown if it is successfully detected.

- ***Cancel*:**
  Cancel current calibration
- ***Camera recalibrieren*:**
  Performs a dynamic stereo calibration. This improves the recognition of the calibration pattern and acquisition of the 3D data. A calibration pattern must be placed as centrally as possible in the camera's field of view in order to perform the calibration.

  Recalibration is particularly useful after the camera has been finally installed/screwed onto the system during commissioning.

- ***Output field*:**
  The text field is used to output status messages during calibration. Here you can also see the progress of the calibration procedure, for example.

Setup types

Depending on the application and vision system, you must select the correct calibration procedure. All types are listed below and the settings that can be made are explained in each case.

Moving camera

In hand-eye calibration with a mobile camera, the transformation between the camera coordinate system and the coordinate system of the robot flange is determined. The position of the robot flange can be converted into world coordinates using the measured joint angles of the robot. This means that the camera image can also be converted into world coordinates with the aid of hand-eye calibration (HAK).

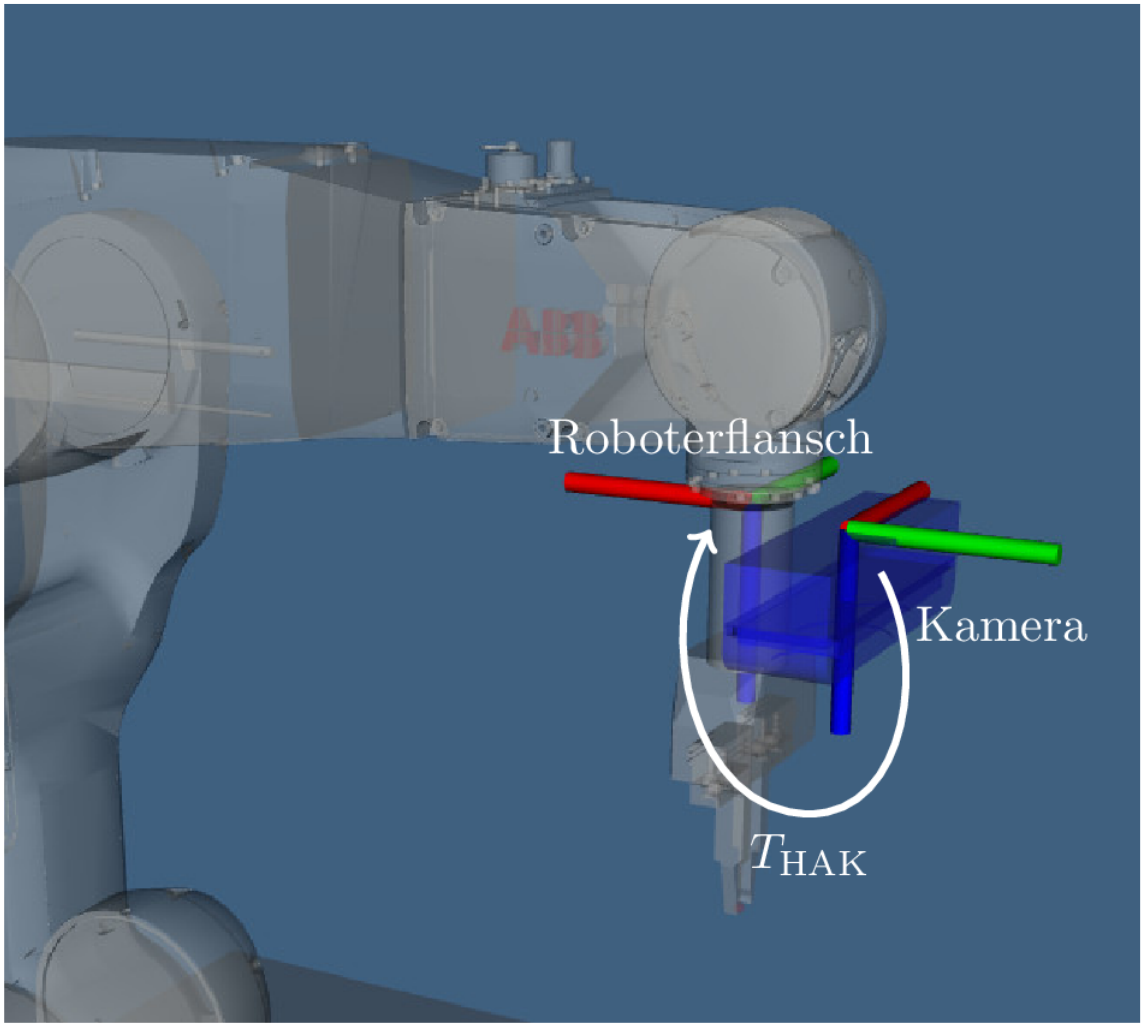


Fig. 50: Transformation determined by the hand-eye calibration for a moving camera.

The camera is located on the robot arm and is therefore movable. The calibration pattern is placed in the center of the work area. During calibration, various robot poses are approached randomly. If the calibration pattern is visible at this point, the image is included in the calibration. The process is repeated until the required number of poses is reached. Once the calibration data has been successfully calculated, it is saved in the camera.

Further input fields:

- **Manual mode**: No automatic calibration, further details in section *Manual mode*.
- **Number of poses**: Number of valid poses that are to be approached for calibration. Poses for which the calibration pattern cannot be recognized are not counted.

Hint: If you switch to the 'Visualization' tab while calibrating a moving camera, you will see the scene and the robot. The poses that the robot has moved to for the recordings are also shown here as individual axis representations.

Fixed camera

In hand-eye calibration with a fixed camera, the transformation between the camera coordinate system and the world coordinate system is determined. This transformation can be used to convert the camera images into world coordinates.

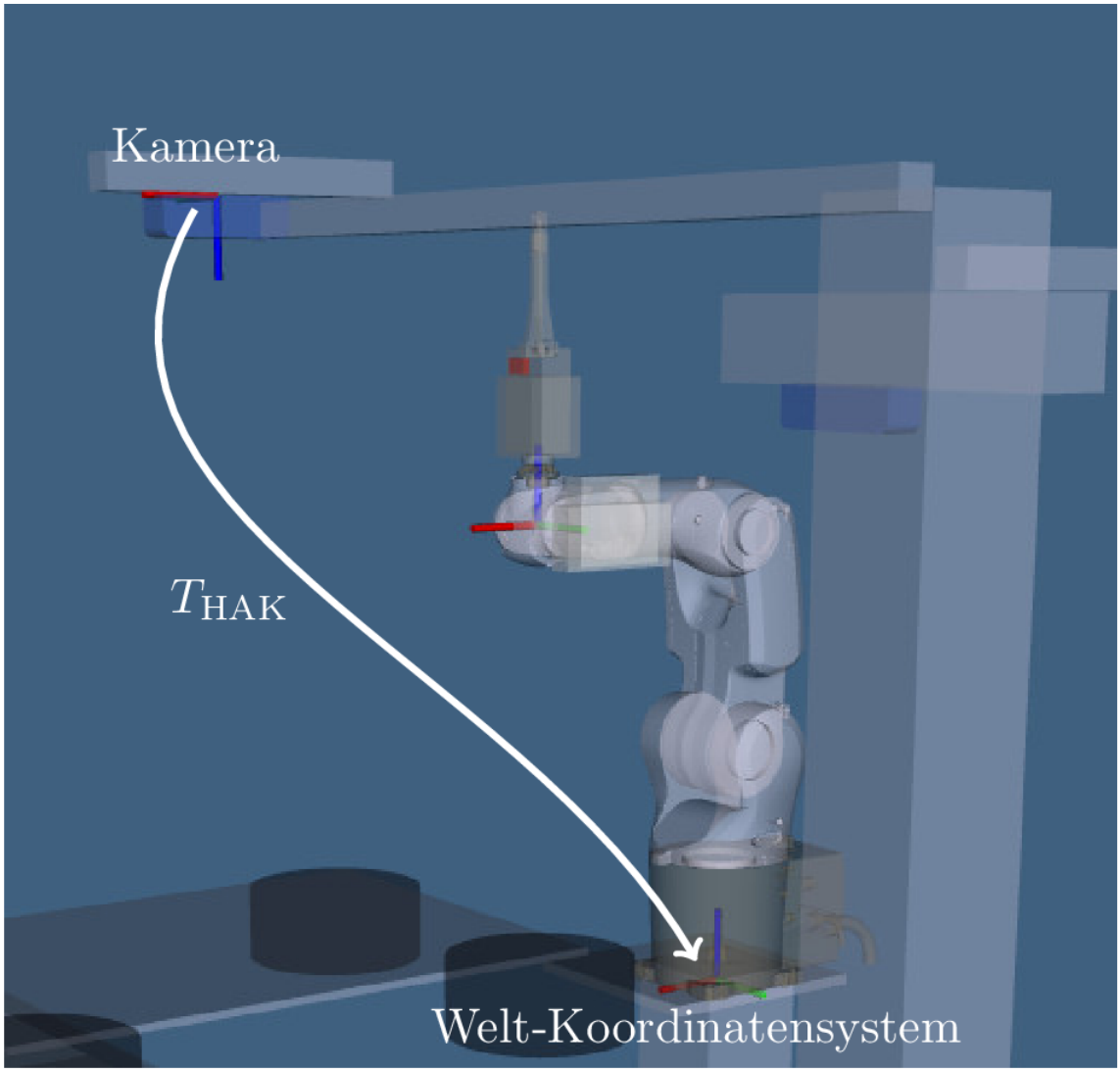


Fig. 51: Transformation determined by hand-eye calibration with a fixed camera.

The calibration procedure is the same as in the 'Moving camera' setup. The difference is that the camera is mounted

statically at a fixed location. The calibration pattern must therefore be attached to the robot arm. The various poses are now approached by the robot and move the calibration pattern in the camera's field of view. Instructions for attaching the calibration pattern can be found under *Fastening the calibration plate*.

Further input fields:

- **Manual mode**: No automatic calibration, further details in section *Manual mode*.
- **Number of poses**: Number of valid poses that are to be approached for calibration. Poses for which the calibration pattern cannot be recognized are not counted.

Transfer

The *Transfer* mode enables the transfer of calibration data from one camera to a second camera that is permanently installed. The setup is only used in systems with multiple cameras. In the *Reference camera* field, you can select the camera to be used as the source. Both a moving and a fixed camera can be selected as the reference camera.

To transfer the calibration, both cameras must view a common area. Position the calibration pattern in this area and start the calibration.

Further input fields:

- **Reference camera**: Select an already calibrated camera (moving or fixed) to be used as the source for the calibration.

Set origin

You can also set the origin directly using the calibration pattern. The center point on the surface of the calibration plate is used as the origin by default in this mode. By additionally specifying a *Pattern Pose*, you can transform the origin relative to the calibration pattern and thus set an offset.

Further input fields:

- **Pattern Pose**: Set a relative offset to the calibration pattern here. This allows you to place the origin at a position that cannot be physically reached with the calibration pattern.

Note: In the Ensenso calibration sample, the Z-axis is perpendicular to the top of the sample side of the calibration plate, the X-axis points parallel to the edges in the direction of the 'Ensenso' label, the Y-axis describes the edges at right angles to it.

Roboter

Outputs the transformation chain of the robot. This transformation chain is automatically used during hand-eye calibration to check the calibration of the robot joints. If a significant improvement in the quality of the hand-eye calibration can be achieved by recalibrating the robot joints, Mikado issues a corresponding message.

Furthermore, this transformation chain can be used in external tools to calibrate the zero positions of the robot joints.

Manual mode

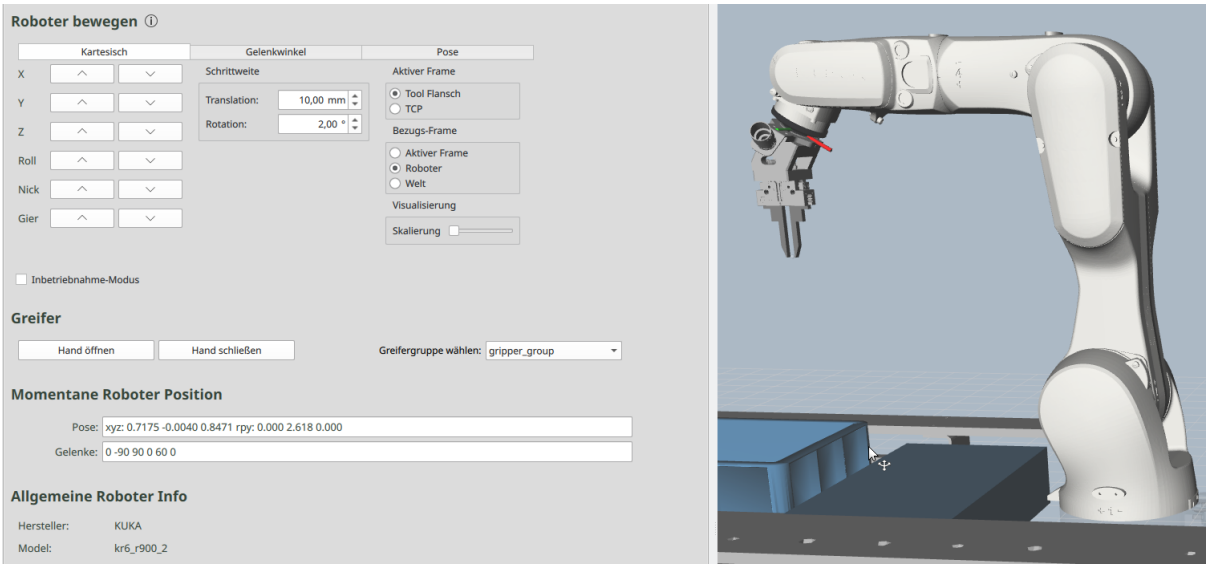
In manual mode, you can control the individual calibration steps yourself. This option is available for the 'Moving camera' and 'Fixed camera' setup types. Manual mode allows you, for example, to manually move the robot to a suitable position for the individual recordings. If you activate this mode, you will see four new buttons:

- **Start**: The calibration process is initialized. The current robot pose determines the starting position of the calibration.
- **Recording**: This triggers a camera recording. If the calibration pattern cannot be found, a message is displayed in the output field.
- **New pose**: The robot is moved to a random pose. If you have not yet performed a pre-calibration, the calibration pattern may not be fully visible at this pose. Alternatively, you can also move the robot manually directly via the robot controller in order to move to the next desired recording position.
- **Pre-calibration**: If there are enough successful recordings/measurements, pre-calibration can be started. After this, only random poses that retain the calibration pattern in the field of view are approached.

2.19 Robot control

The 'Robot' page provides an interface and operating elements for your connected robot. The robot can be controlled and repositioned directly, and you can also determine the current position. Users with advanced knowledge can also send robot-specific commands, depending on the robot model. The robot is controlled directly using the various buttons. Each movement is collision-checked with all objects found by the object recognition or modeled in the scene.

There are three sections, **Direct Movements** , **Grab** and **Move**.



2.19.1 Direct movements

- **Active link**:

  Defines the reference point around which rotation is to take place for the movements to be executed rotationally. If, for example, **Tcp** is selected, rotation is performed around the coordinate origin of this link.

- **Coordinate system**:

  The axes of this coordinate system are used to execute the movements. If 'Active link' is selected, the coordinate origin of the active link is used. If 'Robot' is selected, the base link is used.

As an aid, you can imagine that the coordinate axes of the selected coordinate system are moved to the coordinate origin of the active link and then used as the basis for the movements.

Warning: The 'robot' coordinate system does not necessarily correspond to the world coordinate system!

- **X / Y / Z**:

  Clicking on an arrow increases/decreases the value on the respective axis by the specified increment (default value: 10 mm). The movements are executed in the 'Active link' or 'Robot' coordinate system.

- **Roll / Pitch / Yaw**:

  Clicking on an arrow increases/decreases the value on the respective axis by the specified increment (default value: 2 degrees). The movements are carried out in the 'Active link' or 'Robot' coordinate system.

- **J1 - J6**:

  Click on + / - to increase/decrease the value of the respective joint. This moves the robot arm in joint angles.

- **Crotch width**:

  The increment is specified here for Cartesian movements in the X/Y/Z direction. For rotational and joint movements, the increment is also specified here in degrees.

- **Show**:

  This allows you to display the coordinate systems of 'Active link' and 'Coordinate system' (see above).

- **Scaling coordinate system**:

  This allows you to scale the displayed coordinate systems.

Warning:

- Make sure that there are no objects in the cell or near the robot that are not modeled in the scene while you are controlling the robot.
- You can also hold down a button, but you should be aware that the robot may execute the last command when you release it and move a little further. If you want a higher positioning accuracy, it is best to use the original manual robot control (teach panel).

2.19.2 Commissioning mode

This setting is only available, if the *developer mode* was activated. With active commissioning mode the *system status* will not treat the robot status *manual mode / teaching mode* as an error. Thus the Mikado system status can become ready, although this normally requires the robot to be in automatic mode. After changing this setting it is applied on the next robot status change.

This setting can be permanently activated in the Mikado Assistant.

2.19.3 Gripper

- **Open hand**:

  Opens the robot's gripper

- **Close hand**:

  Closes the robot's gripper

- **Select gripper group**:

  Allows the selection of the active gripper, if multiple grippers have been set up in the Assistant.

2.19.4 Move

- A pose or joint angle position can be selected using the combo box button. The poses/joint angle positions of the currently loaded Mikado program are available for selection.
- Activate the ‘Move’ option to get a pose preview and activate the mode.
- Select which robot part the robot should use to move to the specified pose. However, this has no effect on joint angle positions.
- Click on *Move* to move the robot to the displayed target position.

You will find the following output elements under the buttons for direct control:

- **Pose**:

  Position of the flange/wrist of the robot as a pose in the text display of Mikado Control.

- **Joints**:

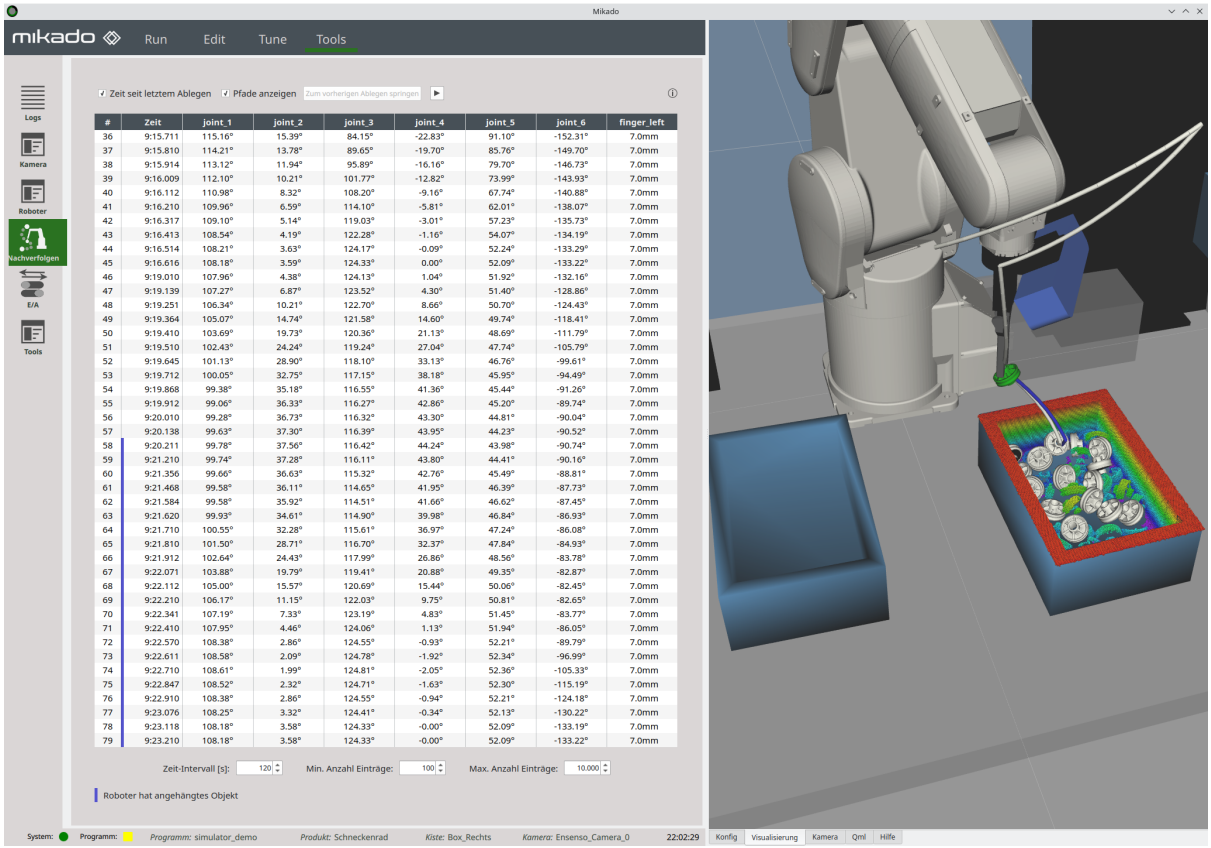
  Position values of the robot as joint angle in degrees.

2.19.5 Robot parameters

If the robot has model-specific parameters or functionalities, these will also be shown on this page. A description of the model-dependent control elements can be found in the *Robot models* chapter.

2.20 Robot tracking

The ‘Tracking’ page allows you to track the robot’s movements. The robot states of the last N seconds are saved temporarily for this purpose, whereby you can set N. The robot states are displayed in a list and can be selected.



The screenshot above shows the tracking page. The meaning of the various page elements is explained below.

- **Time since last filing:** Allows you to change the display of the time column. If the check mark is set, the times are shown relative to the last previous filing (or to the original state if no filing has taken place). Otherwise, the time is simply shown as a time.
- **Display paths:** If the check mark is set, the robot's movement steps contained in the list are combined into a path and visualized. Partial paths are colored blue if the robot has an attached object. If the check mark is not set, the paths are hidden.
- **Jump to previous filing:** If a line is selected in the list, you can jump to the previous filing (as seen from this line). If "Time since last filing" is set, you jump to the line that has the time value 0 (or to the beginning of the list if the last previous filing has already been removed from the list).
- **Replay button:** Replays the movement of the robot in the visualization, with the same time intervals as they occur between the individual steps. The only exception is that the maximum time interval for playback is limited to 1s so that the user does not have to wait too long if the robot has not moved for a while.
- **Time interval [s]:** The time interval specifies how far back the tracking should go. For example, at 120 seconds, the robot states of the last two minutes are temporarily stored and displayed. Exceptions are cases where there were fewer than "Min. number of states" or more than "Max. number of states" during this period (see the following two points).
- **Min. number of states:** This parameter specifies the minimum number of the last states to be temporarily stored and displayed. If there were fewer (different) states in the specified time interval, older states are also displayed.
- **Max. Number of states:** This parameter specifies the maximum number of the last states to be temporarily stored and displayed. If there were more (different) states in the specified time interval, the entire time interval is not displayed.
- **Left bar:** If a state in the list on the left has a blue bar, this means that this state has an attached object. This makes it easier to find phases with and without an attached object.
- **Visualization:** If a state is selected in the list, it is displayed in the visualization without transparency. At the same time, the current robot state (the last entry in the list) is displayed semi-transparently. If no state is selected in the list, the last (current) state is always displayed without transparency.

2.21 Overview of robot inputs and outputs

The page offers functions for displaying robot inputs and outputs. In addition, robot outputs can be set and the page can be configured as required. To use the functions described below, there must be a connection to the robot. The page is intended to support troubleshooting in the overall process. The view can be found in the menu under 'Tools' → 'I/O'.

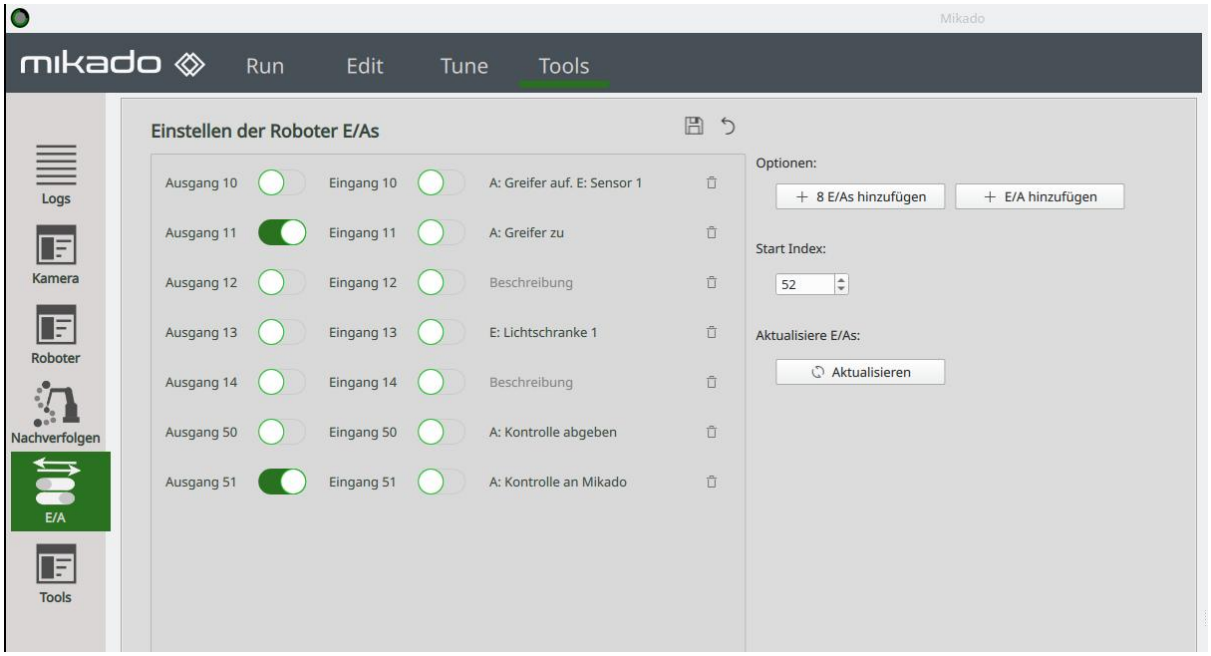


Fig. 52: Screenshot of setting the robot I/Os

2.21.1 Adding/removing I/Os

On the right-hand side of the screenshot above, you can see various elements for adding and removing inputs and outputs.

- ***Add 8 I/Os*:**
  Adds 8 inputs and outputs starting with the start index shown in the spin box below. If the spin box is set to 10, inputs and outputs 10 - 17 are added.
- ***Add E/As*:**
  Adds an input and output with the start index that can be seen in the spin box below.
- ***Update I/Os*:**
  Reads all inputs and outputs in the list and displays their result. If the circle with the white base and the green border points to the right, the output is *true*; if it points to the left, it is *false*.

2.21.2 Setting outputs and labeling the I/Os

The list can be edited and saved to provide the best possible overview of the inputs and outputs that are important for the application.

- ***Set outputs*:**
  By clicking on the circle with the white base and the green border to the right of the output, it can be set to *false* or *true*.
- ***Labeling*:**
  In order not to lose track of the inputs and outputs, they can be labeled. If the same number is assigned to an input and output, it can be helpful to make a note of this in the labeling.
- ***Save the list*:**
  In order to have the list and label permanently available even after restarting Mikado, it can be saved using the diskette symbol above the list. Only one list can be saved per Mikado cell.

- ***Reset the list*:**
  To reset a change to the last saved status, press the arrow next to the diskette symbol.

If there is no connection to the robot or it is faulty, no input or output can be set or read via Mikado. In this case, the circle is outlined in red and contains a red cross, as shown in the following image.

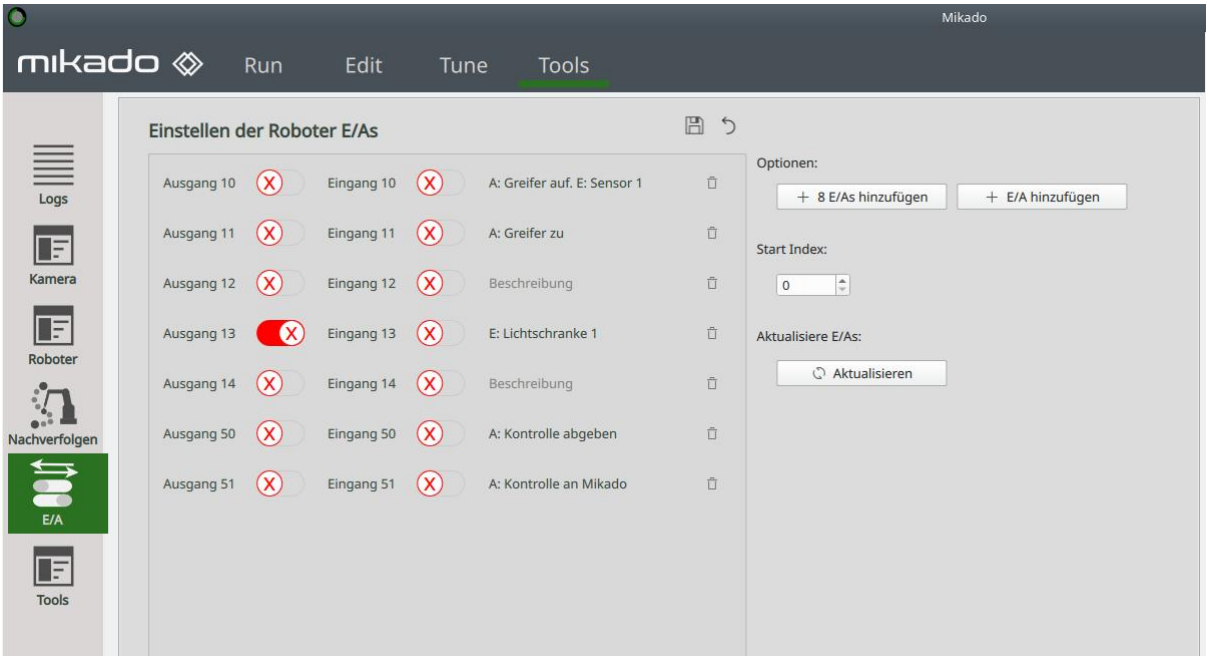


Fig. 53: Screenshot Error when reading the inputs and outputs

2.22 System-Tools

The System tools page is located under the menu item 'Tools' → 'Tools'. The following elements can be found here:

2.22.1 System

Elements for general interaction with Mikado:

- **Exit Mikado:** Exits the Mikado application after displaying a dialog for confirmation
- **Shut down controller:** Opens the dialog of the operating system to shut down or restart the controller. Without any further user interaction, the Mikado controller is automatically shut down after a waiting time of 30 seconds.
- **Full screen mode / Exit full screen mode:** Switch the Mikado interface between full screen mode and window mode.

Note: The elements '*End Mikado*' and '*Full screen mode*' are locked when password protection is switched on and are only enabled once the correct password has been entered.

2.22.2 Mikado

Information about the Mikado software version is visible in this section. This includes, for example, the Mikado edition used, the installed software version and the directory of the currently open Mikado cell.

2.22.3 Developer tools

The Mikado interface can be set to developer mode using the “Activate developer mode” checkbox. This mode is intended for internal use and is used, for example, for testing the software.

2.22.4 Profiling

With the help of so-called profiling, certain activities of the Mikado system can be recorded and subsequently analyzed. The activities of the system are time-stamped and saved in a proprietary binary format in a file of the type *.nxlog*. These files can then be viewed via the external application NxProfiler, which visualizes the previously recorded activities of the system on a timeline. Profiling can be used, for example, to identify time-consuming function calls in order to optimize the cycle time of an application.

The Mikado system is divided into three different sections whose activities are recorded separately. These are:

- **Mikado:** Contains the activities of the Mikado main program, in particular e.g. the Mikado program sequence, handle search, path planning of the robot etc.
- **Ensenso:** Interface between the Mikado application and the Ensenso camera with camera-specific functionalities (e.g. calculating the octomap, cutting out objects from the point cloud, applying ROIs, forwarding function calls to the NxLib)
- **NxLib:** Activities of the Ensenso NxLib API (program library on which the Ensenso interface is based). Contains deeper information about the interaction with the camera, for example processing functions (stereo matching), PartFinder search algorithm and network communication between controller and camera.



Fig. 54: Screenshot Tools Profiling

The type and number of recorded activities are influenced by the selected log level. This can be set individually for each of the three sections mentioned using the drop-down list. The following levels can be selected in ascending order:

- **Info:** Basic information
- **Debug:** More information about partial steps of commands
- **Trace:** Complete logging of all calls

Hint: As the log level increases, all information from the lower levels continues to be recorded, but additional information may be added. Log level *Debug* therefore also contains all information from level *Info*, for example.

The following interaction options are available for each section (Mikado / Ensenso / NxLib):

- **Log level menu:** Select the desired log level (see above). The menu is deactivated if the profiler is already active.

- **Start:** Starts profiling for this section with the set log level. If the profiler is active, a timer is automatically started which writes the collected data to the designated file every 15 seconds.
- **Stop:** Stops profiling for the section and saves the collected but unsaved data in the corresponding file.
- **Open log in NxProfiler:** Opens the saved file for the corresponding section in the external application *NxProfiler*.

Warning: When the Mikado application is restarted, the recorded log files of a Mikado cell from the previous session are emptied and overwritten. The files therefore only contain information about the currently active or last started session of this cell. However, different cells contain separate files that are not overwritten by other cells.

CHAPTER

THREE

TASKS

This chapter describes all available *tasks* that can be used in a Mikado program sequence. Each of the following tasks is first explained by a description, followed by up to two tables.

- The **Variables** table lists all variables that a task requires or offers. The variable type and, if applicable, a short description complete the description in the two additional columns.
- The second table **Sequence change** contains all sequence transitions that can be triggered by a task. The first column contains the specified variable name as it is displayed in the *Program editor*. The second column shows the condition that must be fulfilled when the task is executed for the sequence change to take place. A more detailed explanation of the condition can be found in the third column.

Further information on creating a program sequence can be found under *Program editor*.

3.1 Procedure

3.1.1 Switch to sequence

Switches to another sequence.

This task can be used to ensure that a specific sequence is executed during program execution. If, for example, a sequence is to be executed a maximum of once, this task can be placed at the end of this sequence in order to then switch to the next sequence to be executed.

Sequence change

Name	Condition	Remark
Next	–	–

3.1.2 If-Group

Evaluates conditions in a specific order one after the other and executes the corresponding subsequence when a condition is met.

Each If-group consists of at least one `if` element. Optionally, any number of `else if` elements and a single `else` element at the end can be added.

The individual conditions of the If-group are evaluated in sequence from top to bottom. If one of the conditions is true, the corresponding subsequence is executed and everything following it in the if group is ignored. If none of the given conditions are true and the if group reaches the `else` element during evaluation, this subsequence is executed. This means that `else` is a fallback option that is always executed if none of the previous conditions are true.



Fig. 1: Screenshot of an If group in the program editor

Variables

Name	Type	Description
lhs	Bool, Integer, Double, String, Pose, JointPosition, Product, Box, ROI, Obstruction, PickSearch-ParamSet	Left-hand side
Operation	Parameter	Selection of the comparison operation. The available operations depend on the variable type. For all variables: • == • != Only for integer and double variables: • < • > • <= • >= Only for bool variables: • `AND` • `OR` • `NAND` • `NOR`
rhs	Bool, Integer, Double, String, Pose, JointPosition, Product, Box, ROI, Obstruction, PickSearch-ParamSet	Right-hand side

3.1.3 Exit program

Ends the currently running program. The running sequence is aborted and program execution stops. By default, all variables are reset to their initial value and the scene is reset. Program execution starts again from the first sequence when the program is restarted.

Variablen

Name	Typ	Description
Reset variables	Bool	Specifies whether all Mikado variables should be reset to their initial value (optional, default value: 'true')
Reset scene	Bool	Specifies whether the scene with its collision objects should be reset to the initial state (optional, default value: 'true')

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.1.4 Error

Exit the program with an error.

Behaves in the same way as the *Exit program* task, but this task signals an error status. If, for example, external systems access Mikado Control, it may be important to know whether a program has been successfully completed or stopped due to an error.

3.1.5 Wait

Waits for a certain time before continuing with the program sequence.

The time to wait is specified in milliseconds and must be stored in a variable.

Variablen

Name	Typ	Description
Time span [ms]	Integer	Time span in milliseconds that is to be maintained

3.1.6 Waiting until standstill

This function is used to achieve a pre-run stop.

This task blocks the running program until all movements triggered by previous movement tasks (e.g. *Move robot to*) have been completed.

Mikado combines some partial trajectories into one continuous movement. In the event of interruptions (e.g. emergency stop, error), unwanted shortcuts can occur when moving from several points. This can be prevented by this task, as the robot must stop at this task and subsequent movements are not combined with previous ones.

3.1.7 Interrupt program

The program is stopped at the location of this task. Execution can then be continued using the ‘Start’ or ‘Execute next task’ buttons in the user interface (see *Control buttons*).

When using a fieldbus interface, the `RunningBit` (see *Mikado status*) remains `true` in this paused program status. The Mikado program is therefore still being executed from the communication partner’s point of view, as it is only paused but not finished in this state.

Variablen

Name	Typ	Description
Reason	String	The content of the variable is displayed on the status page as text in the task status.

Tip: Use this task for error analysis, for example, to interrupt the process in certain situations.

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.2 Numbers

3.2.1 Increase number

Increase a number by one. The result is assigned to the variable ‘Number’.

Variablen

Name	Typ	Description
Number	Integer, Double	This variable is increased by one.

3.2.2 Reduce number

Decrease a number by one. The result is assigned to the variable ‘Number’.

Variablen

Name	Typ	Description
Number	Integer, Double	This variable is reduced by one.

3.2.3 Arithmetic operation

Perform a mathematical operation with two input variables. The basic arithmetic operations of addition, subtraction, multiplication and division are available.

Variables

Name	Type	Description
Input 1	Integer, Double	–
Input 2	Integer, Double	–
Operation	Parameter	Selection of the mathematical operation: • + Addition • - Subtraction • × Multiplication • / Division
Result	Integer, Double	–

Hint: The required type of the result variable depends on the types of the two input variables. If both input values are of type `Integer`, the result must also be of type `Integer`. In this case, integer operations are used.

In all other cases in which one or both input values are of type `double`, the result must be of type `double`. Otherwise, the conversion of a floating point number into an integer could lead to an unintentional loss of information. Mixed forms are also permitted for the input variables, i.e. an `integer` variable and a `double` variable can be calculated together.

Warning: When dividing with two integers, integer division is used. Only the quotient without any fractional part is retained and any remainder is discarded.

Example: 7 / 4 = 1, the remainder 3 is discarded.

3.2.4 Logical operation

Perform a logical operation with two input variables. These operations can be used to logically combine two `Bool` variables. The operations AND, OR, XOR, NAND, NOR and XNOR are available. The behaviour of these operations can be seen in the truth tables shown below.

Variables

Name	Type	Description
Input 1	Bool	–
Input 2	Bool	–
Operation	Parameter	Selection of the logical operation: • `AND` • `OR` • `XOR` • `NAND` • `NOR` • `XNOR`
Result	Bool	–

Truth tables

The following tables list all possible combinations of two inputs and the corresponding result values for each of the available Boolean operations.

AND

The result is True only if both inputs are True (logical ‘and’ operation).

Input 1	Input 2	Result (AND)
False	False	False
False	True	False
True	False	False
True	True	True

OR

The result is True if at least one input is True (logical ‘or’ operation).

Input 1	Input 2	Result (OR)
False	False	False
False	True	True
True	False	True
True	True	True

XOR

The result is True if exactly one input is True, but not both (exclusive ‘or’ operation).

This operation is also known as logical inequality, because the result is only true if the two inputs are different (i.e. one is true and the other false, but not both true or both false at the same time).

Input 1	Input 2	Result (XOR)
False	False	False
False	True	True
True	False	True
True	True	False

NAND

The result is False only if both inputs are True; otherwise, it’s True.

The NAND operation is the logical complement of the AND operation, i.e. it produces exactly the opposite result of the logical AND operation.

Input 1	Input 2	Result (NAND)
False	False	True
False	True	True
True	False	True
True	True	False

NOR

The result is True only if both inputs are False.

The NOR operation is the logical complement of the OR operation, i.e. it produces exactly the opposite result of the logical OR operation.

Input 1	Input 2	Result (NOR)
False	False	True
False	True	False
True	False	False
True	True	False

XNOR

The result is True if both inputs are the same (both True or both False).

This operation is therefore also known as logical equality and is the logical complement to the XOR operation.

Input 1	Input 2	Result (XNOR)
False	False	True
False	True	False
True	False	False
True	True	True

3.2.5 Comparison operation

This task allows a jump to another sequence when a certain condition is met.

The two input variables and the operation parameter are evaluated. If the result of this comparison is 'true', the task jumps to the sequence of the sequence change `True`, if specified. If the result of the comparison is "false", the task jumps to the sequence of the sequence change 'False', if specified.

If one of the conditions occurs and the corresponding optional sequence change is not set, the program execution continues with the next task of the active sequence. This means that there is no jump to another sequence.

Hint: The variable types of the two input variables `x` and `y` must match.

Variables

Name	Type	Description
x	Bool, Integer, Double, String, Pose, JointPosition, Product, Box, ROI, Obstruction, PickSearchParamSet	–
Operation	Parameter	Selection of the comparison operation. The available operations depend on the variable type. For all variables: • == • != Only for integer and double variables: • < • > • <= • >= Only for bool variables: • AND • OR • NAND • NOR
y	Bool, Integer, Double, String, Pose, JointPosition, Product, Box, ROI, Obstruction, PickSearchParamSet	–

Sequence change

Name	Condition	Remark
True	The expression is true	Sequence change is optional
False	The expression is false	Sequence change is optional

3.3 Fundamentals

3.3.1 Copy

Copies the value of the input variable to the output variable.

Make sure that the type of the input variable matches the type of the output variable.

Hint: If you want to convert types, you can use the *JavaScript* task.

Variablen

Name	Typ	Description
Input	alle	Variable with the value to be copied, remains unchanged
Issue	alle	Variable whose value is overwritten with the value of the input variable

3.3.2 Swap values

Swaps the values of two variables. The variable types must match.

Variablen

Name	Typ	Description
Value 1	alle	Gets value 2
Value 2	alle	Receives value 1

3.3.3 Reset variable

Resets the variable to its initial value. This means that changes to the value of the variable are discarded by the program sequence and the variable contains the defined initial value.

Variablen

Name	Typ	Description
Variable	alle	

3.3.4 Reset all variables

Resets all variables to their initial value. This means that changes to the value of the variables are discarded by the program sequence and the variables are given the defined initial value.

3.3.5 Set initial value

Sets the current value of a variable as the initial value. The program is marked as “changed” and must be saved if the change is to be permanent.

Variablen

Name	Typ	Description
Variable	alle	Variable whose initial value is set

3.3.6 Negate Boolean variable

Negates a boolean variable.

This means that if the variable is true, it is changed to false by this task. If the variable is false before this task is executed, it is set to true.

Variablen

Name	Typ	Description
Variable	Boolean	

3.3.7 Copy entry from list

Copy a specific entry from a list into a variable.

Hint: If you only want to run through the individual elements of the list one after the other, the *Copy next entry from list* is more suitable.

Variablen

Name	Typ	Description
List	PoseList, StringList	Input list
Index	Integer	Index of the entry from the list that is to be read out. For example, for index = 0, the first entry in the list is copied.
Copy to	Pose, String	Output variable: Copy list entry to this variable

Hint: The variable types of the input list and the output variable must match (e.g. PoseList and Pose).

Sequence change

Name	Condition	Remark
Index too large	Index > list size or index < 0	If the index is not valid, the entry cannot be read. It is therefore possible to continue with another sequence in this case.

3.3.8 Copy next entry from list

Copy the next entry from a list.

An internal counter is used for this, which counts up each time the task is executed. When the end of the list is reached, the counter starts again at the first element. You therefore do not need a variable for the list index here as with the *Copy entry from list* task.

Note: If this task is used in several places, they count up independently of each other.

Example application: Alternate several shooting poses when taking pictures

Variablen

Name	Typ	Description
List	PoseList, StringList	Input list
Copy to	Pose, String	Output variable: Copy list entry to this variable

3.3.9 Add to list

Add an element (at the end) to the list.

Variablen

Name	Typ	Description
Element to be added	Pose, String	This element is added to the end of the list
List	PoseList, StringList	List to which the element is to be added

3.3.10 Set list element

Set an element in the given list to a specific value. The old value at this point is overwritten.

Variablen

Name	Typ	Description
Value	Pose, String	Value that is to overwrite the value of an element in the list
Index	Integer	Position in the list at which the element is to be overwritten
List	PoseList, StringList	List to be modified

3.3.11 Delete element from list

Removes an element from the given list at the given position

Hint: Useful for working through a series of target poses, for example.

Variablen

Name	Typ	Description
List	List	list to be edited
Index	Integer	Position of the element to be deleted

Sequence change

Name	Condition	Remark
Invalid index	Index<0 or index >= number of items	0-based indexing of the list elements.

3.3.12 Liste leeren

Empty the given list.

All entries in the list are removed. A new list can then be created using the *Add to list* task, for example.

Variablen

Name	Typ	Description
List	PoseList, StringList	List to be emptied

3.3.13 Liste leather?

Checks whether the given list is empty.

Variablen

Name	Typ	Description
List	PoseList, StringList	List to be checked

Sequence change

Name	Condition	Remark
List is empty	List contains no elements	–
List not empty	List contains at least one element	–

3.3.14 Count list elements

Get the number of elements in a list (StringList or PoseList)

Variables

Name	Type	Description
List	PoseList, StringList	Input: List to get the length of
Number of elements	Integer	Output: Number of elements in the list

3.4 Robot control

3.4.1 Move robot to

Move the robot using free movement planning so that the tool flange (tool 0 or wrist) is positioned at the target pose.

The direct movement to the target position is always checked first. If this is not possible, alternative paths are searched for and used if successful. As the free path search can be computationally intensive, positions that are easy to reach should be selected wherever possible. When moving around a known obstacle, you should therefore consider moving to intermediate positions.

Hint: Robot poses can be defined as a pose or joint position (directly the individual joint angles). Robot poses can also be taught in / taught out. How the robot is controlled using the Mikado interface is described in section '*Robot control*', the teaching of the position into a variable is explained in section '*Editing variables*'.

Variablen

Name	Typ	Description
Goal	Pose, JointPosition	Target position that the robot should move to. The position defines the pose of the tool flange (tool 0 or wrist) of the robot.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if the path planning could not find a collision-free path to the target position (e.g. due to obstacles or an impossible position). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

During a new attempt, the free path planning calculates a new movement that positions the robot flange collision-free at the target pose. However, the movement now starts from the current position at which it stopped in the previous attempt.

3.4.2 Move robot Cartesian to

Drive the robot in a straight line to the target position.

For more information on robot movements, see '*Move robot to*'.

Variablen

Name	Typ	Description
Goal	Pose, JointPosition	Target position that the robot should move to. The position defines the pose of the tool flange (tool 0 or wrist) of the robot.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if a Cartesian movement to the target position is not possible without collision (e.g. due to obstacles). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

The robot moves to the target position again in a straight line during a new attempt. However, it starts from the current position at which it stopped in the previous attempt.

3.4.3 Move Robot relatively

Freely move the robot such that the tool flange (tool 0 or wrist) is moved by the given deltas or transform.

The task is similar to '*Move robot to*', except the goal-pose is not given explicitly but rather a transform is given, which is applied to the current pose in order to compute the goal-pose.

Variables

Name	Type	Description
Delta x	double	intended x-Translation
Delta y	double	intended y-Translation
Delta z	double	intended z-Translation
Delta Roll	double	intended Roll-Rotation
Delta Pitch	double	intended Pitch-Rotation
Delta Yaw	double	intended Yaw-Rotation
Transform	Pose	intended transform of the current pose
in local coordinates?	bool	yes: Transform in local coordinates no: transform in world-coordinates

If a Transform is given, all Delta-Variables must be empty, and vice versa.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if the path planning could not find a collision-free path to the target position (e.g. due to obstacles or an impossible position). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

During a new attempt, the free path planning calculates a new movement that positions the robot flange collision-free at the target pose. However, the movement now starts from the current position at which it stopped in the previous attempt.

3.4.4 Move Robot Cartesian relatively

Drive the robot in a straight line relative to its current position.

For more information on robot movements, see '*Move robot to*'.

Variables

Name	Type	Description
Delta x	double	intended x-Translation
Delta y	double	intended y-Translation
Delta z	double	intended z-Translation
Delta Roll	double	intended Roll-Rotation
Delta Pitch	double	intended Pitch-Rotation
Delta Yaw	double	intended Yaw-Rotation
Transform	Pose	intended transform of the current pose
in local coordinates?	bool	yes: Transform in local coordinates no: transform in world-coordinates

If a Transform is given, all Delta-Variables must be empty, and vice versa.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if a Cartesian movement to the target position is not possible without collision (e.g. due to obstacles). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

The robot moves to the target position again in a straight line during a new attempt. However, it starts from the current position at which it stopped in the previous attempt.

3.4.5 Move Tcp to

Move the robot using free motion planning so that the tool center point (TCP or end effector) or the specified robot link is positioned at the target pose.

For more information on robot movements, see '*Move robot to*'.

Variablen

Name	Typ	Description
Goal	Pose, JointPosition	Target position that the robot should move to. The position defines the pose of the robot's tool center point (TCP or end effector).
Link	String	The link *gripper_tcp* is used as TCP by default. An alternative TCP for the movement can be specified here.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if the path planning could not find a collision-free path to the target position (e.g. due to obstacles or an impossible position). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

When a new attempt is made, the free path planning calculates a new movement that positions the tool center point collision-free at the target pose. However, the movement now starts from the current position at which it stopped in the previous attempt.

3.4.6 Move Tcp Cartesian to

Move the robot in a straight line so that the tool center point (TCP) or the specified robot link is positioned at the target pose.

Optionally, a tolerance in meters can be specified. This means that no error is triggered in two special cases:

1. Planning recognizes a collision with a component of the cell, an object or the point cloud:

   The system now checks whether the TCP can be moved within the specified tolerance. If this is the case, only the possible partial movement is executed.

2. The robot movement is interrupted directly on the robot by an input or similar:

   If the TCP comes to a stop within the tolerance to the destination, the journey is considered successful. Otherwise, the "Collision" sequence change is triggered.

One use case would be, for example, an ini with which the deflection of a spring-loaded suction cup is detected on contact with a surface. The robot program supplied by us must now be modified so that it aborts the current movement on an edge at this input. Typically, an interrupt can be defined that reacts to the signal and initiates the braking movement. We will be happy to assist you with the corresponding adaptation.

For more information on robot movements, see '*Move robot to*'.

Variablen

Name	Typ	Description
Goal	Pose, JointPosition	Target position that the robot should move to. The position defines the pose of the robot's tool center point (TCP or end effector).
Toleranz	Double	Permitted distance from the end position to the target position at which planning or travel is considered successful.
Link	String	The link *gripper_tcp* is used as TCP by default. An alternative TCP for the movement can be specified here.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if the path planning could not find a collision-free path to the target position (e.g. due to obstacles or an impossible position). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.
	Journey was interrupted	If the TCP is still outside the tolerance, the system switches to this sequence when the movement is interrupted externally.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

The movement starts from the current position at which you stopped in the previous attempt.

3.4.7 Move Tcp relatively

Move the robot using free motion planning so that the tool center point (TCP or end effector) or the specified robot link is moved by the given deltas or transform.

For more information on robot movements, see '*Move robot to*'.

Variables

Name	Type	Description
Delta x	double	intended x-Translation
Delta y	double	intended y-Translation
Delta z	double	intended z-Translation
Delta Roll	double	intended Roll-Rotation
Delta Pitch	double	intended Pitch-Rotation
Delta Yaw	double	intended Yaw-Rotation
Transform	Pose	intended transform of the current pose
Link	String	The link *gripper_tcp* is used as TCP by default. An alternative TCP for the movement can be specified here.
in local coordinates?	bool	yes: Transform in local coordinates no: transform in world-coordinates

If a Transform is given, all Delta-Variables must be empty, and vice versa.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if the path planning could not find a collision-free path to the target position (e.g. due to obstacles or an impossible position). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

When a new attempt is made, the free path planning calculates a new movement that positions the tool center point collision-free at the target pose. However, the movement now starts from the current position at which it stopped in the previous attempt.

3.4.8 Move Tcp Cartesian relatively

Move the Tcp or the given link in a straight line relative to its current position.

Optionally, a tolerance in meters can be specified. This means that no error is triggered in two special cases:

1. Planning recognizes a collision with a component of the cell, an object or the point cloud:

   The system now checks whether the TCP can be moved within the specified tolerance. If this is the case, only the possible partial movement is executed.

2. The robot movement is interrupted directly on the robot by an input or similar:

   If the TCP comes to a stop within the tolerance to the destination, the journey is considered successful. Otherwise, the "Collision" sequence change is triggered.

   One use case would be, for example, an ini with which the deflection of a spring-loaded suction cup is detected on contact with a surface. The robot program supplied by us must now be modified so that it aborts the current movement on an edge at this input. Typically, an interrupt can be defined that reacts to the signal and initiates the braking movement. We will be happy to assist you with the corresponding adaptation.

For more information on robot movements, see '*Move robot to*'.

Variables

Name	Type	Description
Delta x	double	intended x-Translation
Delta y	double	intended y-Translation
Delta z	double	intended z-Translation
Delta Roll	double	intended Roll-Rotation
Delta Pitch	double	intended Pitch-Rotation
Delta Yaw	double	intended Yaw-Rotation
Transform	Pose	intended transform of the current pose
in local coordinates?	bool	yes: Transform in local coordinates no: transform in world-coordinates
Tolerance	Double	Permitted distance from the end position to the target position at which planning or travel is considered successful.
Link	String	The link *gripper_tcp* is used as TCP by default. An alternative TCP for the movement can be specified here.

If a Transform is given, all Delta-Variables must be empty, and vice versa.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if the path planning could not find a collision-free path to the target position (e.g. due to obstacles or an impossible position). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.
	Journey was interrupted	If the TCP is still outside the tolerance, the system switches to this sequence when the movement is interrupted externally.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

The movement starts from the current position at which you stopped in the previous attempt.

3.4.9 Move part to

Move the robot using free movement planning so that the attached / gripped part is positioned at the target pose.

For more information on robot movements, see '*Move robot to*'.

If the product of the currently attached object contains a symmetry (see *Symmetries*), poses that have been rotated according to those symmetries will be considered as well. As an example consider a cylindrical object with a continuous rotational symmetry around X and a discrete 180° rotational symmetry around Z. Here the task will consider poses with a rotation around the longitudinal axis as well as their 180° rotated variants. The motion will end at the collision-free solution that is closest in joint-space.

Variablen

Name	Typ	Description
Goal	Pose, JointPosition	Target position that the robot should move to. The position defines the pose of the currently attached / gripped part.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if the path planning could not find a collision-free path to the target position (e.g. due to obstacles or an impossible position). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

The free path planning calculates a new movement for a new attempt, which positions the part collision-free at the target pose. However, the movement now starts from the current position at which it stopped in the previous attempt.

3.4.10 Move part Cartesian to

Move the attached / gripped part from the current pose to the target position in a straight movement.

For more information on robot movements, see '*Move robot to*'.

If the product of the currently attached object contains a symmetry (see *Symmetries*), poses that have been rotated according to those symmetries will be considered for the final position as well. As an example consider a cylindrical object with a continuous rotational symmetry around X and a discrete 180° rotational symmetry around Z. Here the task will consider poses with a rotation around the longitudinal axis as well as their 180° rotated variants. The motion will end at the collision-free solution that is closest in joint-space.

Variablen

Name	Typ	Description
Goal	Pose, JointPosition	Target position that the robot should move to. The position defines the pose of the currently attached / gripped part.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if a Cartesian movement to the target position is not possible without collision (e.g. due to obstacles). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

The robot moves the part to the target position again in a straight line during a new attempt. However, it starts from the current position at which it stopped in the previous attempt.

3.4.11 Move Piece relatively

Move the robot using free movement planning so that the attached / gripped part is moved by the given deltas or transform.

The task is similar to '*Move part to*', except the goal-pose is not given explicitly but rather a transform is given, which is applied to the current pose in order to compute the goal-pose.

Variables

Name	Type	Description
Delta x	double	intended x-Translation
Delta y	double	intended y-Translation
Delta z	double	intended z-Translation
Delta Roll	double	intended Roll-Rotation
Delta Pitch	double	intended Pitch-Rotation
Delta Yaw	double	intended Yaw-Rotation
Transform	Pose	intended transform of the current pose
in local coordinates?	bool	yes: Transform in local coordinates no: transform in world-coordinates

If a Transform is given, all Delta-Variables must be empty, and vice versa.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if the path planning could not find a collision-free path to the target position (e.g. due to obstacles or an impossible position). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

When a new attempt is made, the free path planning calculates a new movement that positions the piece collision-free at the target pose. However, the movement now starts from the current position at which it stopped in the previous attempt.

3.4.12 Move Piece Cartesian relatively

Move the robot using so that the attached / gripped part is moved by the given deltas or transform in a straight line.

For more information on robot movements, see '*Move robot to*'.

If the product of the currently attached object contains a symmetry (see *Symmetries*), poses that have been rotated according to those symmetries will be considered for the final position as well. As an example consider a cylindrical object with a continuous rotational symmetry around X and a discrete 180° rotational symmetry around Z. Here the task will consider poses with a rotation around the longitudinal axis as well as their 180° rotated variants. The motion will end at the collision-free solution that is closest in joint-space.

Variables

Name	Type	Description
Delta x	double	intended x-Translation
Delta y	double	intended y-Translation
Delta z	double	intended z-Translation
Delta Roll	double	intended Roll-Rotation
Delta Pitch	double	intended Pitch-Rotation
Delta Yaw	double	intended Yaw-Rotation
Transform	Pose	intended transform of the current pose
in local coordinates?	bool	yes: Transform in local coordinates no: transform in world-coordinates

If a Transform is given, all Delta-Variables must be empty, and vice versa.

Sequence change

Name	Condition	Remark
Collision	Collision-free path could not be found	If a sequence is specified here, the program switches to this sequence if a Cartesian movement to the target position is not possible without collision (e.g. due to obstacles). If this field remains empty, the program execution is stopped in this case without the robot having moved to the target position.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

The robot moves to the target position again in a straight line during a new attempt. However, it starts from the current position at which it stopped in the previous attempt.

3.4.13 Read robot position

Reads out the current pose of the robot (wrist) and saves it in a variable.

Variablen

Name	Typ	Description
Roboterposition	Pose	Output variable: Current position of the wrist

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.4.14 Read part position

Reads out the current pose of an attached part and saves it in a variable. If no part is attached, the program jumps to error handling.

Variablen

Name	Typ	Description
Part position	Pose	Output variable: Current position of the piece

Sequence change

Name	Condition	Remark
No part grabbed	There is currently no part attached	If no part is appended, its position cannot be read. Therefore, error handling must take place in this case or the program is terminated with an error

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.4.15 Read TCP position

Reads out the current pose of the TCP (tool-center-point) and saves it in a variable.

Variables

Name	Type	Description
TCP Position	Pose	Output variable: current TCP position

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.4.16 Is robot position accessible with part?

Check whether a given partial pose can be reached by the robot and is collision-free.

If the product of the currently attached object contains a symmetry (see *Symmetries*), poses that have been rotated according to those symmetries will be considered as well. As an example consider a cylindrical object with a continuous rotational symmetry around X and a discrete 180° rotational symmetry around Z. Here the task will consider poses with a rotation around the longitudinal axis as well as their 180° rotated variants.

Variablen

Name	Typ	Description
Goal	Pose	Target pose of the part to be tested

Sequence change

Name	Condition	Remark
Not reachable	Target pose not reachable	–

3.4.17 Is robot position reachable?

Check whether the given robot pose can be reached from the specified robot link (e.g. **gripper_tcp** for the TCP) and is collision-free.

Variablen

Name	Typ	Description
Goal	Pose	Target pose whose achievability is to be tested
Link	String	Link of the robot that is to reach the destination

Sequence change

Name	Condition	Remark
Not reachable	Target pose not reachable	–

3.4.18 Open/close hand

Opens or closes the gripper. If the input variable is set to ‘true’, the gripper is opened; if it is set to ‘false’, it is closed.

If the gripper is already in the desired state, this task does not cause any change.

Variablen

Name	Typ	Description
Open hand	Bool	'true': Gripper should be opened (or remain open) 'false': Gripper should be closed (or remain closed)

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.4.19 Setting the gripper status

Sets the gripper to an extended configured state (also called **additional gripper state**).

This state change can be used to set and wait for IO signals, for example. The use of extended gripper states makes it possible to model complex grippers that can assume other states in addition to open/closed.

Variablen

Name	Typ	Description
Condition	String	Name of the new gripper state, must be defined in *gripper.yaml* within *additional_states*. Example: 'Standing' and 'Folded' for a gripper with additional swivel function.

Note: The `joint_values` of a gripper with `additional_state` must currently be published by a project-specific script with a `joint_state_publisher`. Contact Mikado support for details.

Gripping behavior

If a gripper has configured, extended states, there is a separate grip category for each state. In the following example, the gripper has a pneumatic swivel joint to fold the gripper fingers. This joint has two states: "Standing" and "Folded". Grips can be defined separately for each of these states.

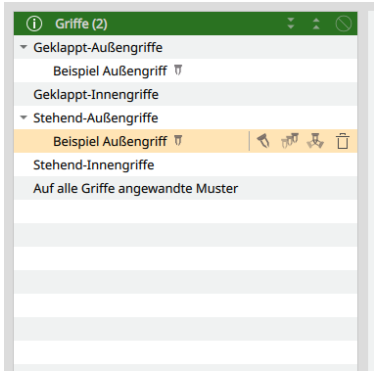


Fig. 2: Extended states in the handle editor

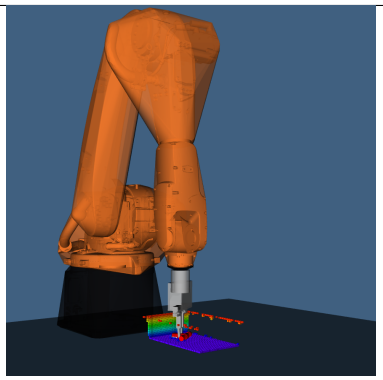


Fig. 3: External handle in "upright" position

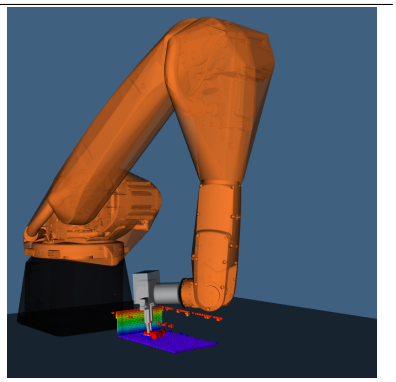


Fig. 4: External handle in "Folded" state

The best handle is always selected (see *Best handles*). However, the grip search only implicitly considers the extended states in its prioritization criteria. In the example above, the best grip (of the two possible) is the "standing" grip, as the wrist is higher and the trajectory distance is much shorter.

Rg	#	SZ	G	O	T	K	R	F	S	z ↓	HG-z ↓	TD ↑	Objekt	Griff
1	1	38				S	X		79.0	3.93	24.65	383.7	Produkt_1 3	Stehend-Außengriffe
2	2	163			X	S	X		79.0	3.93	15.85	680.1	Produkt_1 3	Geklappt-Außengriffe

Fig. 5: Prioritization of the two states

If the best grip belongs to a state other than the current state, the *Pick object* task internally calls the *Setting the gripper status* task. This means that the gripper state is automatically changed in this case (or the signal is sent).

Summary:

1. Handle search delivers “best handle”
2. Does the best grip belong to a state other than the current state?
3. If no -> continue to 5.
4. Set the gripper status according to the “best grip”
5. Grab part

Detailed description

There are a total of 4 possible actions that can be performed when setting an extended status:

- `set_outputs_true`: sets the IO ports corresponding to the given signal numbers to `true`
- `set_outputs_false`: sets the IO ports corresponding to the given signal numbers to `false`
- `wait_for_inputs_true`: blocked until `true` is present at the corresponding IO ports
- `wait_for_inputs_false`: blocked until `false` is present at the corresponding IO ports

In addition, changed joint positions of the gripper can be defined for extended states using the following two parameters:

- `joint_names`: List of additional joints within the gripper
- `joint_values`: Joint values [rad] that the additional joints assume in this state

Example of a corresponding section for an extended *gripper.yaml* configuration:

```yaml
# gripper.yaml

additional_states:
  Stehend:
    set_outputs_true:       []
    set_outputs_false:      [2]  # Roboter IO zum Setzen von Gelenk stehend
    wait_for_inputs_true:   [23] # Stehend bestätigt
    wait_for_inputs_false: []
    joint_names:            ['Kippgelenk_Joint']
    joint_values:           [0.0]  # Wert in [rad] von Kippgelenk_Joint im Zustand
↪"Stehend"
  Geklappt:
    set_outputs_true:       [2]  # Roboter IO zum Setzen von Gelenk geklappt
    set_outputs_false:      []
    wait_for_inputs_true:   [22]  # Klappen bestätigt
    wait_for_inputs_false: []
    joint_names:            ['Kippgelenk_Joint']
    joint_values:           [-1.570796]  # Wert in [rad] von Kippgelenk_Joint im␣
↪Zustand "Geklappt"
```

If defined in `gripper.yaml`, these `additional_states` are automatically added to newly created products. An example of a handle definition with 'upright' and 'folded' states:

```
# <name>.product
[...]

grasps:
  - name: Außengriff - stehend
    type: GraspSetting
    graspType: SimpleExternal
    additionalStates:
      - Stehend   # muss übereinstimmen mit Namen aus gripper.yaml
    endEffector: gripper_group
    subitems:
        []
  - name: Außengriff - geklappt
    type: GraspSetting
    graspType: SimpleExternal
    additionalStates:
      - Geklappt  # muss übereinstimmen mit Namen aus gripper.yaml
    endEffector: gripper_group
    subitems:
        []
```

Note: For existing products, the additional gripper state for a specific handle category must be entered manually in the <productName>.product file.

3.4.20 Set active gripper

Changes the active gripper to one of the configured grippers. The change can only be effected if no product is attached to the gripper.

To set up handles for interchangeable grippers, see *Setting up handles for interchangeable grippers*.

Variablen

Name	Typ	Description
Gripper	String	Name of the configured gripper

3.4.21 Execute robot program

Execute a native robot program.

Native robot programs are stored and/or programmed directly on a robot controller and define specific sequences in the native programming language. This allows additional capabilities of the robot system to be used, for example. To create such a program, please consult the documentation for your robot.

Prerequisites for correctly creating and preparing robot-specific programs are explained in the corresponding sections in the chapter *Robot models*.

Add your own robot program for:

- *Universal Robot*
- *Kuka*

- Please contact us if your robot manufacturer is not yet listed

Variablen

Name	Typ	Description
Program name	String	File name or program name of the native robot program on the controller
Integer Argument Name	String	Variable name of an integer variable on the controller
Integer argument	Integer	Value of the above integer variable
Double Argument Name	String	Variable name of a floating point variable on the controller
Double Argument	Double	Value of the floating point variable above
String Argument Name	String	Variable name of a text variable on the controller
String Argument	String	Value of the above text variable
Pose Argument Name	String	Variable name of a Pose variable on the controller
Pose Argument	Pose	Value of the above pose variable
Return value	Double	The robot program can modify the return value and thus provide feedback to Mikado Control. The return value is stored in the variable specified here after the task has been executed.

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.4.22 Set robot output signal

Sets the value of a robot (or PLC) output.

The output bits of a robot can be used for other system components such as LEDs or doors etc.

Warning: Robot output signals can be modified both by robot programs and via the Mikado sequence controller. Caution is required if both components write to the same output.

Variablen

Name	Typ	Description
Signal number	Integer	The output signals of a robot controller are clearly numbered.
Value	Bool	The desired value of the output signal (‘true’: 1 or ‘high’, ‘false’: 0 or ‘low’)
Pulse duration [ms]	Integer	This variable can be set if required. The output can be set for a period of time and then assumes the previous value.

3.4.23 Read robot output signal

Reads the value of a robot (or PLC) output.

The current status of an output signal can thus be read and verified.

Hint: Robot output signals can be modified both by robot programs and via the Mikado sequence controller.

Variablen

Name	Typ	Description
Signal number	Integer	The output signals of a robot controller are clearly numbered.
Issue	Bool	The read-in value of the input signal is written to this variable (‘true’: 1 or ‘high’, ‘false’: 0 or ‘low’)

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.4.24 Read robot input signal

Reads the value of a robot (or PLC) input

The input bits of a robot can be connected to other system components such as push buttons, light barriers or encoder sensors, etc.

Variablen

Name	Typ	Description
Signal number	Integer	The output signals of a robot controller are clearly numbered.
Issue	Bool	The read-in value of the input signal is written to this variable (‘true’: 1 or ‘high’, ‘false’: 0 or ‘low’)

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.4.25 Wait for robot input signal

Continuously read robot input signal and wait until it matches the given value.

Variablen

Name	Typ	Description
Signal number	Integer	Signal number of the robot controller
Value	Integer, Bool	The desired value that the input signal should assume.
Timeout [ms]	Integer	Maximum waiting time in milliseconds. If the signal does not assume the desired value within this time, the sequence change 'Timeout reached' will take place.

Sequence change

Name	Condition	Remark
Timeout reached	The signal has not reached the desired value within the timeout period	

3.4.26 Wait for robot output signal

Continuously read robot output signal and wait until it matches the given value.

Variablen

Name	Typ	Description
Signal number	Integer	Signal number of the robot controller
Value	Integer, Bool	The desired value that the output signal should assume.
Timeout [ms]	Integer	Maximum waiting time in milliseconds. If the signal does not assume the desired value within this time, the sequence change 'Timeout reached' will take place.

Sequence change

Name	Condition	Remark
Timeout reached	The signal has not reached the desired value within the timeout period	

3.4.27 Wait for fieldbus input signal

Read a fieldbus input variable continuously and wait until it matches the expected value. Fieldbus input variables are Mikado variables that are synchronized via fieldbus (see *Synchronizing PLC variables and Mikado variables*). These variables can be identified by the  symbol when synchronization is active.

Variables

Name	Type	Description
Fieldbus Variable	Integer, Bool	Mikado variable synchronized as fieldbus input (Profinet, EtherNet/IP, etc.)
Expected Value	Integer, Bool	The desired value that the fieldbus variable should assume
Timeout [ms]	Integer	Maximum waiting time in milliseconds (optional). If the fieldbus variable does not assume the expected value within this time, the optional sequence change 'Timeout reached' is executed or the task goes into the error state.

Hint: Fieldbus variable and expected value must be of the same variable type.

Warning: It is strongly recommended to specify a timeout value. Otherwise, there is a risk that the task will block further execution of the program indefinitely.

Sequence change

Name	Condition	Remark
Timeout reached	The fieldbus variable has not assumed the desired value within the timeout period	Optional

3.4.28 Set Fieldbus Output Signal Async

This task sets a fieldbus output variable to a specified target value as soon as the robot leaves a certain region (ROI). Fieldbus output variables are Mikado variables that are synchronized via fieldbus (see *Synchronizing PLC variables and Mikado variables*). These variables can be identified by the ▫ symbol when synchronization is active.

The task triggers a pre-run stop to prevent incorrect use and then starts monitoring the specified region. This means that it should be called when the robot is already within the ROI, but before the movement to leave the ROI is started.

If the robot is already outside the specified ROI when this task is called, the fieldbus output is immediately set to the target value.

Variables

Name	Type	Description
ROI	ROI	The fieldbus output signal is set when this region is left
Fieldbus variable	Bool, Integer, Double, Pose	Mikado variable synchronized as fieldbus output (Profinet, EtherNet/IP, etc.)
Target value	Bool, Integer, Double, Pose	The value that the fieldbus variable should take on

Hint: Fieldbus variables and target value must be of the same type.

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.5 Image processing

3.5.1 Set active camera

Change the camera used for recordings and the like.

Tasks that perform tasks with the help of a camera use the camera that is active at the time the task is executed. If your setup contains several cameras, you can use this to switch between your cameras. If only one camera is part of your setup, this is automatically the active camera.

Example: For a picking application, you have two recording areas, each with one camera. As soon as you have emptied one area, you switch to the other recording area. To do this, you can define a switch sequence in your Mikado program that sets the active camera, among other things.

Variablen

Name	Typ	Description
Camera	Camera, Integer	Name or ID of the camera that is now to be marked as the active camera. Note: Camera names of the available cameras can be seen at the bottom of the *Status bar*.

3.5.2 Load camera configuration

Load camera configuration file and use it to replace the active configuration. The new configuration is applied to the currently active camera.

The specified file name is interpreted relative to the **config** directory in the project folder.

Hint: The configuration file can be created using the *Ensenso NxView application*.

Variablen

Name	Typ	Description
File name	String	File name of the configuration file for your camera. The file must be located in the 'config' subdirectory of the project directory. *Example: '123456.ensparam'*

3.5.3 Kalibrierung laden camera

Load a *hand-eye calibration* for the current camera, replacing the active one.

The specified path is evaluated relative to the **config** directory of the project.

The *hand-eye calibration* can be performed either in the camera menu or with the *Start calibration* task.

Variablen

Name	Typ	Description
File name	String	Name of the file with the HAK for the camera. The file must be located in the 'config' subdirectory of the project directory. *Example: 'last_calibration_Ensenso_Camera_0.json'*

3.5.4 Record point cloud

Records a point cloud with the currently active camera. The recording can then be used to search for objects.

This task only triggers the camera to take the image(s). The point cloud is not yet visible in the visualization. As soon as an object search (or handle search) has been carried out, the point cloud and objects found are inserted into the visualization.

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.5.5 Record point cloud asynchronously

Records a point cloud with the currently active camera. Fixed cameras can therefore be triggered in the background.

The camera only triggers the recording(s) with this task. In contrast to the *Record point cloud* task, the recording is executed in parallel to the sequence. This means, for example, that robot movements can take place during the recording. If the camera data is required in a subsequent task, the system waits for this task to be completed.

The point cloud is not yet visible in the visualization. As soon as an object search (or handle search) has been carried out, the point cloud and objects found are inserted into the visualization.

Hint: It is not permitted to record point clouds in the process while a recording has not yet been completed. Such a situation leads to a program error. Therefore, when using asynchronous recordings, ensure that the subsequent recording is started at the earliest after the previous one has been completed.

Hint: If a sequence is stopped during an asynchronous recording, the recording will still be completed. Do not resume the sequence until this recording has been completed at the earliest.

Variablen

Name	Typ	Description
Delay	Integer	Waiting time in [ms] before the recording is triggered. Application example: Before the recording is triggered, the robot should be moved out of the field of view of a fixed camera.

3.5.6 Capture for Series

Allows capturing and combining multiple point clouds. Each time this tasks is run, the existing point cloud is extended by a new one. The series needs to be finished by a final *Record point cloud* that creates the combined point cloud.

Use this task to combine several recordings (e.g. from different recording perspectives). Except for the last capture, every partial capture must be made with this *Capture for Series* task. As soon as you use the *Record point cloud* (or *Record point cloud asynchronously*) task, this capture is merged with all previous multiple captures and is available for the next step.

Hint: The pose estimation algorithm PartFinder does not support this feature. Only the last point cloud will be used, independently of the previous series.

Hint: Multiple images combine all the resulting point clouds, thereby greatly increasing the number of points. In this case, overlapping areas of the images in particular have twice the number of points. Please note that although this can improve an object search, it can also delay it considerably.

3.5.7 Readjusting the camera pose

Searches for the reference objects and calculates the camera offset.

At least 3 reference objects are required, 4 or more objects is recommended.

The orientation of a found reference object is not included in the calculation. Only the object position is used for the calculation. This depends on the product coordinate system. For this reason, spheres are very suitable as reference objects (with the origin of the product coordinate system at the center of the sphere).

Hint: If there are symmetries in the reference object, the ambiguity must not affect the product position. Normally, it should be sufficient to use the product center point as the origin of the product coordinate system.

Variablen

Name	Typ	Description
Reference object name	String	Product name of the reference object to be searched for (must exist as a product)
Reference poses	PoseList	List of positions at which the reference objects are expected. These can be determined and taught in using the *Find object* task, for example.
ROI names	StringList	Names of the ROI areas that are to be used for the individual reference objects. The order of the entries must match the reference poses.
Max Std. deviation in the relative position [mm]	Double	Maximum permitted standard deviation in the relative position in millimetres. If this limit is exceeded, the sequence change 'Std. dev. > Maximum' (see below) takes place.
Measured poses	PoseList	The poses of the reference objects determined by object recognition
Offset	Pose	The result is stored in this variable. The offset represents the deviation calculated between the measured reference objects and the given reference poses (no orientation).

Sequence change

Name	Condition	Remark
Reference object not found	One of the reference objects could not be found within the specified ROI	
Std. dev > Maximum	Standard deviation > Max Std. dev. in the relative position [mm]	Occurs if, for example, the camera has moved a lot. If no sequence is specified here, the execution of the tasks continues normally even if the maximum is exceeded.

3.5.8 Determine the number of points

Determine the number of points from the last recording.

Hint: To count points in a specific region, first use the '*Set active ROI*' task. The '*Logical operation*' task can be used for evaluation, for example.

This task can be used, for example, to check whether unexpected objects or obstacles have appeared in a certain area. The Z value of the highest and lowest point found is output as an aid for the check. This can be useful, for example, when putting a part back into the box so that the part is not dropped from an unchecked height. Or to check whether an object/obstacle is located in a certain area.

Variablen

Name	Typ	Description
Minimum number	Integer	Input variable: Minimum number of points per shift to be considered the highest Z value. (Default: 1, minimum: 1)
Quantity	Integer, Double	Output variable: Number of 3D points in the active ROI of the active camera is written to this variable
Highest Z	Double	Output variable: Specifies the value of the point with the highest Z value. A region of interest (ROI) is taken into account.
Lowest Z	Double	Output variable: Specifies the value of the point with the smallest Z value. A region of interest (ROI) is taken into account.

Detailed description

Reflections can cause the highest point in the point cloud to deviate significantly from the actual highest point. Isolated scraps of point clouds located in the search area above the products can lead to unexpected results. If problems occur with a noisy point cloud, the calculation of the highest point can be changed by setting the input variable *Minimum number*. If the value of this input variable is greater than 1, a histogram of all Z values is calculated. The search area is divided into 0.5 mm thick layers. Each point within the search area is assigned to a layer. The return value for the highest Z value is the upper edge of the first layer from the top that counts at least the *minimum number* of points. The lowest Z value is determined in the same way. The following figure outlines the principle of calculating the highest point.

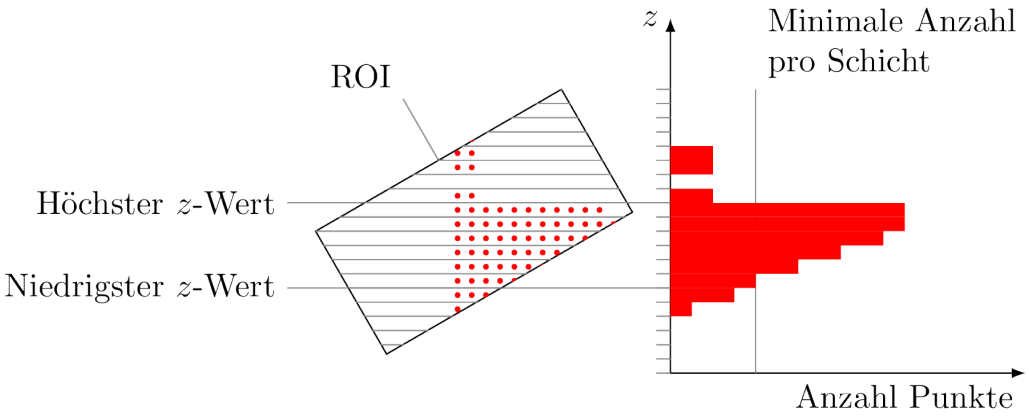


Fig. 6: Principle sketch of the task *Determine number of points*

Warning: Due to the calculation method when the input variable *Minimum number* is set, the return value for the highest or lowest point can have an inaccuracy of up to 0.5 mm.

Note: If no ROI is active, *Minimum number* has no effect.

3.5.9 Save image

Saves the last raw image data set that can be used as an Ensenso file camera. The procedure for creating a new file camera is described in the Ensenso chapter under *File camera*. Typical use cases are situations where no handle or object can be found. This scenario can then be viewed and improved later with the help of a file camera.

Variables

Name	Type	Description
Path	String, Product, Box	Directory path in which the images are to be stored (e.g. `data/filecam1`). Relative to `~/mikado/filecams`.
Prefix	String, Product, Box	Prefix of the file names for the images and configuration files (e.g. `capture_before_pick`)

Hint: The task requires a lot of time, as the data is written to the hard disk synchronously. Therefore, it should only be used sparingly, e.g. to save the last point cloud after errors.

Hint: A product or box can also be specified directly to automatically organize images by type in a subfolder with the name of the product or box.

Hint: Only paths inside *~/mikado/filecams* are permitted. To prevent a disk overload, this directory is automatically restricted to 30GB, if the `mikado-log-purger` package is installed. When new recordings go beyond this threshold, the oldest recordings in this directory will be deleted to make space for new ones.

Warning: From EnsensoSDK Version 4.x onwards, the `Prefix` argument no longer has any effect. It is ignored and has no influence on the file names.

3.5.10 Set active ROI

Set the region of interest (ROI) of the current camera. The ROI is defined via a scene object of type 'ROI', which can be set up on the *'Scene' page*.

Variables

Name	Type	Description
ROI	ROI, Integer	Name of the ROI object or ID of the ROI object that is to be used immediately. The ROI objects are managed in the *Scene management* area.

3.5.11 Determine pattern pose

Captures an image and searches for a calibration pattern in it. Attention: Switch off the projector first.

The result pose refers to the center of the top of the plate. The Z-axis is perpendicular to the surface.

Hint: This task can be used, for example, to check the position of a part on the robot arm (e.g. gripper tip).

- Read out the pattern pose using the camera
- Move the robot TCP to a position relative to the pattern pose
- Measure the distance and compare it with the target pose used

Variablen

Name	Typ	Description
Pattern pose	Pose	Output variable: Pose of the calibration pattern.

Sequence change

Name	Condition	Remark
Pattern not found	Pattern could not be found in the camera image	If no pattern was found in the camera recording, the pose cannot be read out. Therefore, error handling must take place in this case or the program is terminated with an error

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.6 Calibration

3.6.1 Start calibration

This task starts the *hand-eye calibration*. As with *manual mode*, the robot is not automatically moved to a new pose after each recording.

The movement to new poses can be defined by the Mikado program, e.g. by using a list of suitable poses that cover the workspace and many possible orientations of the calibration pattern to the camera.

The *Recording for calibration* task can now be used to record images of the calibration pattern. The task *Finalize calibration* can be used to complete the process as soon as enough suitable recordings are available.

If a calibration has already been started and not yet completed, the previous calibration is canceled.

Hint: The success of hand-eye calibration is highly dependent on the variety of poses used. The better the field of vision is covered and the different orientations of the pattern are captured, the more accurate the result.

3.6.2 Recording for calibration

This task adds a new image to the current calibration process (*Hand-eye calibration*). The calibration was usually started beforehand by the *Start calibration* task.

The images taken with this task should record the calibration pattern in as many different poses as possible in order to increase the accuracy of the calibration.

Error cases

- The task fails if there is no active calibration process.
- The task fails and can be triggered again if a calibration process has been started but the recording does not contain a valid calibration pattern and no sequence has been defined for the “Not visible” result.

Sequence change

Name	Condition	Remark
Not visible	The calibration pattern is not visible or is covered too much	The program should switch to a different pose and execute the task again

3.6.3 Finalize calibration

This task ends the *hand-eye calibration*. The calibration was usually started beforehand by the *Start calibration* task.

Before this task is executed, the *Recording for calibration* task should be used to record images of the calibration pattern in various poses.

The task fails if there is no active calibration process or no valid recordings of the calibration pattern have been recorded.

If the calibration is completed successfully, the result is saved in the file **config/last_calibration_Ensenso_Camera_0.json** and the evaluation of the calibration accuracy is saved in the file **config/last_hand_eye_calibration_errors.json** in the cell folder.

Hint: The success of hand-eye calibration is highly dependent on the variety of poses used. The better the field of vision is covered and the different orientations of the pattern are captured, the more accurate the result.

3.6.4 Validate hand-eye calibration

This task checks the quality of the hand-eye calibration of the active camera. The following steps are carried out for this purpose:

- Move the robot to the specified validation pose.
- Wait until the robot has reached the pose.
- Record point cloud.
- Search for validation object in point cloud.
- Read the pose of the validation object from the URDF.
- Determine the deviation between the model pose and the pose recognized in the point cloud. The distance error e_{Trans} and the angle error e_{Rot} are determined.
- Compare the errors determined with the accuracy of the last hand-eye calibration performed.

Hint: The task can also be used to check the TCP calibration by specifying a part of the gripper or the entire gripper group as the validation object.

Warning: The success of the validation is heavily dependent on the quality of the object recognition of the validation object used or the corresponding product!

Variablen

Name	Typ	Description
Validation object	String	Name of the validation object. A product with this name must be set up (products are defined in the *Product management*) **and** a URDF object with this name must exist (URDF objects are defined in the *Cell geometry*). For a fixed camera setup, link 4 of the robot arm is clearly visible on most robots and is therefore suitable as a validation object.
Validation pose	Pose	Robot flange pose adopted for validation
Permitted deviation	Double	Scaling f of the limit value for successful validation. (Default is 1 for 100% of the mean error of the last HAK)

Sequence change

Name	Condition	Remark
Validation failed	Unexpected error occurred	E.g. no URDF object / product set up, product not found in point cloud, …
Limit value exceeded	$e_{\mathrm{Trans}} > f \cdot \bar{e}_{\mathrm{Trans}}$ oder $e_{\mathrm{Rot}} > f \cdot \bar{e}_{\mathrm{Rot}}$	Determined distance and/or angle error is greater than during the last HAK. A detailed description of the error calculation can be found here *Calculation of the deviation*

Detailed description

Collision monitoring and grip planning in Mikado are always carried out in global coordinates, i.e. the point cloud and the robot pose are transformed into the global coordinate system. To be able to do this, Mikado requires very precise information about the robot's joint position on the one hand and very precise information about the camera's mounting position on the other. The process of hand-eye calibration determines the transformation between the camera and the robot base in the case of a fixed camera. In the case of a mobile camera, the transformation between the robot flange and the camera is determined by the HAK. If the robot pose visible in the point cloud differs from the visualized robot pose, there can be two reasons for this:

- the robot calibration is incorrect: the joint angles received from the robot controller do not correspond to the actual joint angles or
- the mounting position of the camera determined by the HAK does not correspond to the actual mounting position.

The following figure provides an overview of the various transformation chains. The transformations marked in blue are determined by the hand-eye calibration (*Hand-eye calibration*), in each case for the two cases fixed or mobile camera. The transformations marked in black are determined by the joint angles obtained from the robot controller. The orange marked transformation from the robot flange to the gripper (*Calibrate TCP*) is determined by the *Cell geometry* and can only be modified in the Assistant.

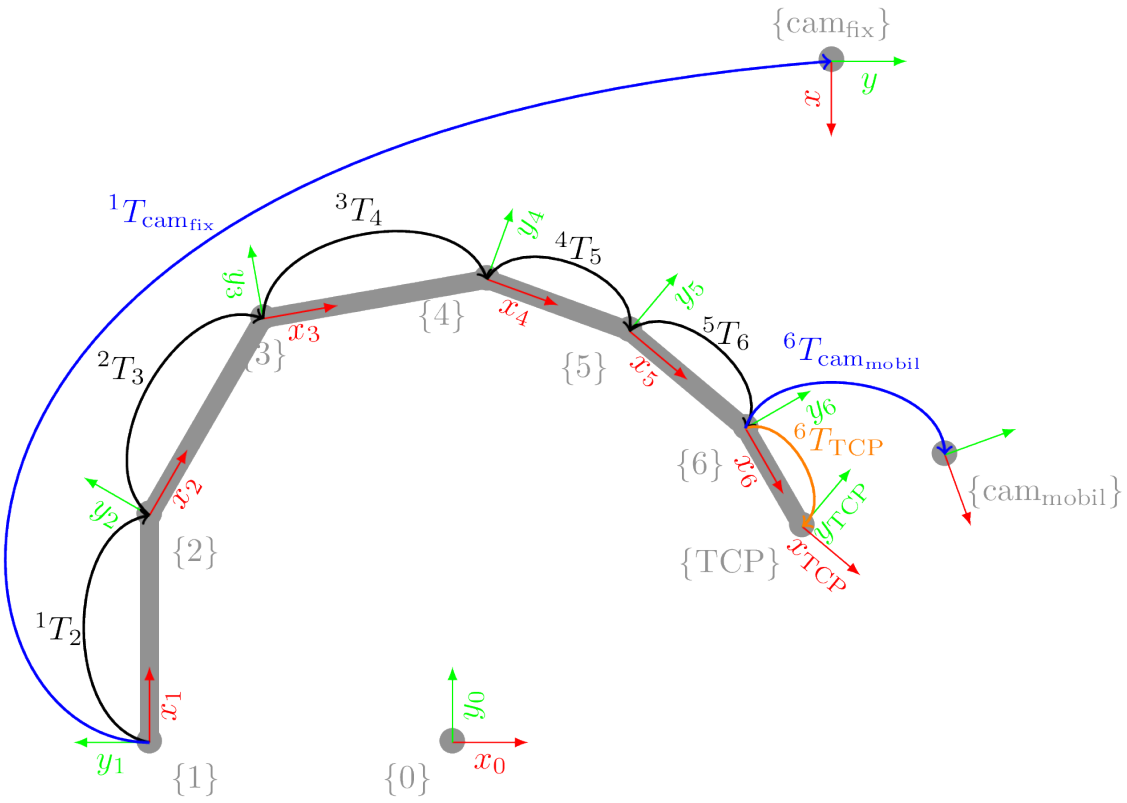


Fig. 7: Overview of the relevant transformations

To check the quality of the hand-eye calibration, the point cloud is compared with the Mikado-internal model of the cell. To do this, a validation object, which is also an integral part of the cell, is searched for in the point cloud and checked to see whether the recognized pose matches the model pose. The following figure shows a successful validation of the hand-eye calibration with *link_4* as the validation object. The robot model (shown here in blue with half opacity) matches the point cloud (shown by the colored dots). To quantify the match, the detection algorithm specified in the validation product is used to determine the exact pose of *link_4* in the point cloud (the detected product is highlighted in white in the figure). This pose is then compared with the model pose of *link_4*.

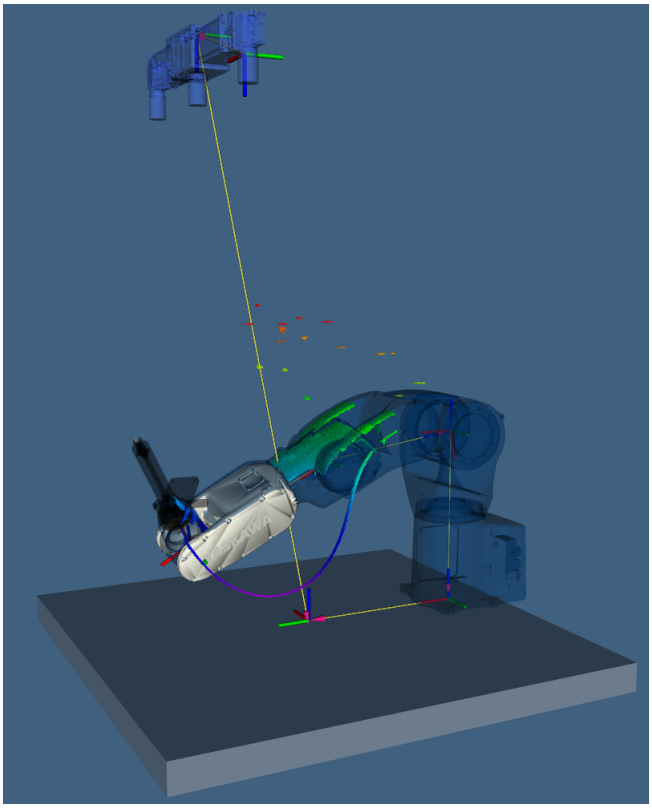
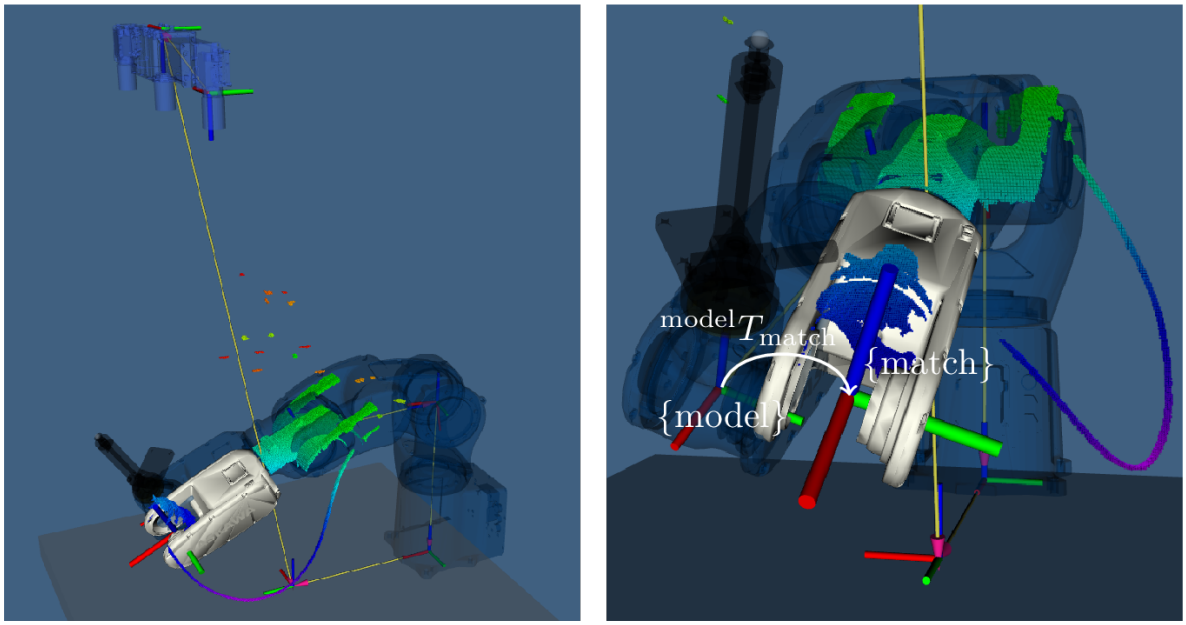


Fig. 8: Successful validation of the hand-eye calibration

Calculation of the deviation

If the hand-eye calibration has become invalid due to a change in the mounting position of the camera (caused by a collision, for example), this can be quickly recognized with the help of this task. The following illustration shows a failed validation. The model pose (again shown in blue with half opacity) differs greatly from the point cloud. The deviation between the model pose and the recognized pose of the validation object is correspondingly large.



(a) Keine Übereinstimmung von Punktwolke und Modell

(b) Detailansicht

Fig. 9: Failed validation of the hand-eye calibration

Here ${}^{\text{model}}T_{\text{match}}$ denotes the transformation between the model pose and the pose recognized in the point cloud, which is called T in the following for the sake of simplicity. The distance error e_{Trans} now results from the Euclidean norm of the translation part t of the transformation T:

$$e_{\text{Trans}} = |t|_2 \ ,$$

where t results from the transformation matrix:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \ .$$

Each 3-dimensional rotation can be represented by an axis of rotation and an angle of rotation θ around this axis. This property is used to determine the angular error and the rotation component R is first converted into this axis-angle representation. The angular error is defined as the magnitude of the rotation angle:

$$e_{\text{Rot}} = |\theta| \ .$$

It should be noted that the angular error only provides information about the rotation around the axis of rotation, but no information about the orientation of this axis.

3.7 Scene object

3.7.1 Set active box

If more than one box exists, this can be used to set the active one.

The *Scene management* section explains how scene objects such as boxes or containers can be created and managed.

If there is only one scene object of the type *box*, this is automatically the active box.

In this case, it is not necessary to integrate this task into your program sequence.

The active box is used for the pick search, for example (see *Start handle search*), as long as no other box is explicitly selected in the respective task.

Variables

Name	Type	Description
Box	Box, Integer	Name of the box or ID of the box that should now be active

3.7.2 Search scene object

Searches for a scene object using the method configured for the object on the scene setup page.

Please note that an image is first captured using the camera for a search and the 2D image search is then triggered. For more information on this search procedure and how it can be set up, see Section *Scene management* / *Obstacle*.

Hint: The current position of a scene object can be read out using the *Read scene object position* task.

Variables

Name	**Type**	**Description**
Object	Box, Integer	The name of the scene object or the ID of the scene object to be searched for (optional)
Min Score	Double	Minimum score for object recognition (optional)

Sequence change

Name	**Condition**	**Remark**
Not found	The object could not be found reliably enough in the current image	Common cause if an object is not found: Object is not completely in the camera's field of view at the time of recording. If no sequence is defined here, the program sequence continues normally with the next task.

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.7.3 Read scene object position

Write position/pose of a scene object in variable.

If the 2D search of the scene object is activated, two scenarios are created for this task:

1. The scene object has not yet been localized (see Task *Search scene object*):
   In this case, the task returns the default pose of the object as the position.
2. The scene object has already been localized (see *Search scene object*, *Find object*):
   The current position (and therefore the result of the last search for the object) is written to the output variable.

Variables

Name	Type	Description
Object	Box, ROI, Obstacle, Integer	The name of the scene object or the ID of the scene object (optional)
Position	Pose	Output variable: The position or pose of the object is written to this variable by the task.

3.7.4 Set scene object position

Set position/pose of a scene object.

The position of a scene object can be replaced by any pose. For example, obstacles can be moved to new positions in the sequence or poses of ROIs can be adjusted.

Hint: This task can be used, for example, to adjust the position of an ROI to a moving crate. The crate is localized with *Search scene object* and the current pose of the crate is read out using the *Read scene object position* task. The read pose is then used as input for this task to set the ROI position.

Variables

Name	Type	Description
Object	Crate, ROI, Obstruction, Integer	The name or ID of a scene object (optional)
New position	Pose	The position or pose to be used for the scene object.

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.7.5 Reset scene object position

Reset the position/pose of a scene object to the initial position.

This task is usually used together with the *Set scene object position* task to reset an object previously modified in the program to its original position.

Variables

Name	Type	Description
Object	ROI, Box, Obstacle, Integer	Optional: The name or ID of the scene object that is to be reset. If this entry is left blank, the active box is used.

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.8 Object handling

3.8.1 Set active product

If several products exist, this task can be used to switch back and forth between them.

Some tasks that require a product use the active product by default unless an explicit product is specified for the task (e.g. *Start handle search*).

Variables

Name	Type	Description
Product	Product, Integer	The product to be actively set. Product names are defined in the *Product management*.

3.8.2 Start handle search

Start the *Gripping motion search*. This is started in the background and runs until all possible handles have been checked or the *Pick object* task is executed. The search can be canceled with the task *Cancel handle search*.

Before executing this task, it is necessary to provide a point cloud using the *Record point cloud* task. The handle search runs as a parallel process in the background, so the running Mikado program is not interrupted. If *Pick object* is called before all handles have been tested, the task behaves as follows, depending on the state of the search:

- If a possible handle has already been found, the search is aborted and the possible handle is executed
- If no possible handle has yet been found, the task waits until a possible handle has been found or all possibilities have been calculated.

The handle search takes place in the following steps:

1. *Box search*, if the box used is configured for a search and the box search variable is not set to 'false' → Box is drawn/positioned in visualization
2. Object search (see *Object recognition tool*) → Found objects and remaining point cloud are displayed in the visualization

3. Handle search → All *Gripping poses* of all objects found are tested, planned and checked for possible collisions. Results can be viewed in the *Handle inspection*.

Hint: Experimental handles are only checked and calculated once all 'normal' handles have been tested without success.

Input-Variables

Name	Typ	Description
Product	Product | Integer	Name or ID of the product to be searched for. If the field is left empty, the active product is used. Product names are defined in the *Product management*.
Box	Box | Integer	Name of the box in which to search. If the field is left empty, the active box is used. Box names are defined in the *Scene management*.
Gripper	Gripper | StringList | Integer | IntegerList	One or more names or IDs for the grippers for which picks are to be found. The gripper of the currently best pick can be read out via the *Pick-Search-Statistics*. If the field is left empty, the last started pick-search is used.
Box search	Bool	Specifies whether the *Box search* should be carried out before the handle/object search in the current recording (optional, default value: 'true').
Transition pose	Pose | JointPosition	Specifies which transition pose is to be used for the handle search (optional, default value: 'Transition pose of active crate').
Pick-Search Parameters	Pick-Search-Parameters | Integer	Name or ID of the Pick-Search Parameters, which are to be used. If the field is left empty, uses the parameters defined in the used product.

3.8.3 Cancel handle search

Cancel the current handle search, e.g. to change the active product or the crate used.

A handle search is started via the task *Start handle search* and runs in the background parallel to the execution of the Mikado program.

Application example: A sensor signal (e.g. read in with *Read robot input signal*) should cause the program to search for another product. You can now cancel the handle search, change the product and immediately start a handle search with the other product.

3.8.4 Find object

Returns a list of all poses of the products found, their recognition scores and all possible handles. The handles are not sorted by quality.

In contrast to the *Start handle search* task, the search is not carried out in the background, but is started with the task. The program is only continued once all results have been calculated.

Application example: If only object positions are to be measured and forwarded to external systems via an interface.

Variables

Name	Type	Description
Product	Product | Integer	Name or ID of the product to be searched for in the last recording. If the field is left empty, the active product is used. Product names are defined in the *Product management*.
Consider in Collision-world	Bool	default: true. If set to false, the found objects will not be added to the collision-world.
Pick-Search Parameters	Pick-Search-ParamSet	Contains the maximum translation- and rotation-difference, which is used to determine whether an object is found at its previous pose.
Object Poses	PoseList	Output variable: List of poses of the objects found. The poses refer to the origin of the product coordinate system.
Object Scores	StringList	Output variable: List of scores of the objects found.
Potential Grasps	PoseList	Output variable: List of all grip poses for all finds, without collision check. The poses refer to 'tool0', i.e. the robot's wrist during the grip.

Sequence change

Name	Condition	Remark
No object found	No object found	

3.8.5 Get Picks

Returns a list of all poses of the products found, their recognition scores and one valid pick-pose per object. The picks are sorted by the sorting criteria, see *Best handles*.

The search has to be started using *Start handle search* before using this task. The program will be continued when for each object either a pick is found or all picks have failed.

Input-Variables

Name	Type	Description
Product	Product | Integer	Name or ID of the product searched for. If the field is left empty, the active product is used. Product names are defined in the *Product management*.
Box	Box | Integer	Name or ID of the box in which the product was searched. If the field is left empty, the active box is used. Box names are defined in the *Scene management*.
Gripper	Gripper | StringList | Integer | IntegerList	One or more names or IDs for the grippers to determine the pick-search, for which information is to be read out. If the field is left empty, the last started pick-search is used.

Output-Variables

Name	Type	Description
Object Poses	PoseList	List of poses of the objects found. The poses refer to the origin of the product coordinate system.
Object Scores	DoubleList	List of scores of the objects found.
Valid Grasps	PoseList	List of valid (collision-checked) handle poses. Only the best grip is returned for each object, whereby the list is sorted according to the criteria described in *Best handles*. The poses refer to 'tool0', i.e. the robot's wrist when grasping.
Valid Grasps Names	StringList	List of names of the valid (collision-checked) grasps. See also: Output variable "Valid Grasps"
Num. Objects Recognized	Integer	Number of Objects found
Num. Objects Pickable	Integer	Num. of Objects for which a valid Pick has been found
Num. Objects Pickable (Preferred)	Integer	Num. of Objects for which a valid Pick of category "Preferred" has been found
Num. Objects Pickable (Standard)	Integer	Num. of Objects for which a valid Pick of category "Standard" has been found
Num. Objects Pickable (Experimental)	Integer	Num. of Objects for which a valid Pick of category "Experimental" has been found
Object Name	String	Name of the Object of the best Pick
Object Pose	Pose	Pose of the Object of the best Pick
Collision Type	Integer	Collision-Type of the best Pick (allowed collisions: 2, no collisions: 0)
Gripper Name	Gripper | String	Gripper of the best Pick (useful for searches with multiple grippers)
Gripper Id	Integer	ID of the Gripper of the best Pick (useful for searches with multiple grippers)
Pick Category	String	Category of the best Pick ("Preferred" / "Standard" / "Experimental")
Pick Subcategory	Integer	Subcategory of the best Pick (0-10)
Grasp-Name	String	Grasp-Name of the best Pick
Grasp-Offset	Pose	Offset of the Grasp of best Pick relative to its Object
Grasp-Tcp-Pose	Pose	Tcp-Pose of the grasp-point of the best Pick
Grasp Tool0-Pose	Pose	Tool0-Pose of the grasp-point of the best Pick
Start Tcp-Pose	Pose	Tcp-Pose of the start-point of the best Pick
Start Tool0-Pose	Pose	Tool0-Pose of the start-point of the best Pick
Approach Tcp-Pose	Pose	Tcp-Pose of the approach-point of the best Pick
Approach Tool0-Pose	Pose	Tool0-Pose of the approach-point of the best Pick
Retract Tcp-Pose	Pose	Tcp-Pose of the retract-point of the best Pick
Retract Tool0-Pose	Pose	Tool0-Pose of the retract-point of the best Pick
Final Tcp-Pose	Pose	Tcp-Pose of the end-point of the best Pick
Final Tool0-Pose	Pose	Tool0-Pose of the end-point of the best Pick
Total Collision-Depth	Double	Total collision-depth of all steps of the best Pick
Maximum Collision-Depth	Double	Maximum collision-depth of all steps of the best Pick
Average Collision-Depth	Double	Average collision-depth of all steps of the best Pick

Sequence change

Name	Condition	Remark
No box found	No box found	
No object found	No object found	
No pick found	No valid pick found	No pickable objects available

3.8.6 Pick object

Pick an object from the box. The results of *Gripping motion search* are awaited, and then the best pick found is executed.

If necessary, this task blocks until a valid pick is found or a timeout occurs during the pick search (more on the timeout parameter in section *pick_search.yaml*).

Further information on picking parts can be found in the task *Start handle search*.

Hint: For a higher throughput speed, the point cloud should be captured as early as possible and the pick search should be started with the task *Start handle search*. Ideally, the *Gripping motion search* has already found a valid pick when the "*Pick object*" task is reached. This is the only way the robot can execute a pick directly without delay. Otherwise, it must wait for the calculation.

Hint: To store the part again, you must use the task *Discard part*!

Move to transit pose

There are two different variants for the robot's behaviour when moving to the transit pose, between which you can choose using an optional input variable. The robot can either always move to the transit pose when this task is executed – regardless of whether a valid pick is available – or only if a valid pick is actually available.

Which of the two variants is more suitable depends heavily on the specific application. If the '*Pick object*' task has to wait for the pick search, it is usually faster if the robot always moves to the transit pose. Once a valid pick is found, the robot is already in the transit pose directly above the box and can immediately execute the pick.

In certain situations, however, it may be useful for the robot to wait at its previous position for a valid pick and only then move to the transit pose. This is the case, for example, when several boxes are to be checked for picks one after the other. In this case, moving directly to the transit pose would cause the robot to move to each box, even if ultimately there is no pick available. If, on the other hand, the robot waits at its previous position, it can move directly from there to a box with a valid pick, thus saving a few unnecessary movements.

Input-Variables

Name	Typ	Description
Product	Product | Integer	Name or ID of the product searched for. If the field is left empty, the active product is used. Product names are defined in the *Product management*.
Box	Box | Integer	Name or ID of the box in which the product was searched. If the field is left empty, the active box is used. Box names are defined in the *Scene management*.
Move to transit only if pick	Bool	Behaviour regarding the movement to the transit pose, see *Move to transit pose* (optional, default: 'false'). If 'false', the robot always moves to the transit pose. If 'true', only in the case of a valid pick.

Output-Variables

Name	Typ	Description
is the grasp experimental?	Bool	is set to 'true' if the executed pick is an experimental pick
Num. Objects Recognized	Integer	Number of Objects found
Num. Objects Pickable	Integer	Num. of Objects for which a valid Pick has been found
Num. Objects Pickable (Preferred)	Integer	Num. of Objects for which a valid Pick of category "Preferred" has been found
Num. Objects Pickable (Standard)	Integer	Num. of Objects for which a valid Pick of category "Standard" has been found
Num. Objects Pickable (Experimental)	Integer	Num. of Objects for which a valid Pick of category "Experimental" has been found
Object Name	String	Name of the Object of the executed Pick
Object Pose	Pose	Pose of the object of the executed Pick
Collision Type	Integer	Collision-Type of the executed Pick (allowed collisions: 2, no collisions: 0)
Gripper Name	Gripper | String	Gripper of the executed Pick (useful for searches with multiple grippers)
Gripper Id	Integer	ID of the Gripper of the executed Pick (useful for searches with multiple grippers)
Pick Category	String	Category of the executed Pick ("Preferred" / "Standard" / "Experimental")
Pick Subcategory	Integer	Subcategory of the executed Pick (0-10)
Grasp-Name	String	Name of the Grasp of the executed Pick
Grasp-Offset	Pose	Offset of the Grasp of the executed Pick relative to its Object
Grasp-Tcp-Pose	Pose	Tcp-Pose of the grasp-point of the executed Pick
Grasp Tool0-Pose	Pose	Tool0-Pose of the grasp-point of the executed Pick
Start Tcp-Pose	Pose	Tcp-Pose of the start-point of the executed Pick
Start Tool0-Pose	Pose	Tool0-Pose of the start-point of the executed Pick
Approach Tcp-Pose	Pose	Tcp-Pose of the approach-point of the executed Pick
Approach Tool0-Pose	Pose	Tool0-Pose of the approach-point of the executed Pick
Retract Tcp-Pose	Pose	Tcp-Pose of the retract-point of the executed Pick
Retract Tool0-Pose	Pose	Tool0-Pose of the retract-point of the executed Pick
Final Tcp-Pose	Pose	Tcp-Pose of the end-point of the executed Pick
Final Tool0-Pose	Pose	Tool0-Pose of the end-point of the executed Pick
Total Collision-Depth	Double	Total collision-depth of all steps of the executed Pick
Maximum Collision-Depth	Double	Maximum collision-depth of all steps of the executed Pick
Average Collision-Depth	Double	Average collision-depth of all steps of the executed Pick

Sequence change

Name	Condition	Remark
Box not found	Box not found	If the crate used is not fixed, but is searched for in the point cloud (see *Scene management* / *Obstacle*), it is possible that the crate was not found reliably enough.
No object found	No object found	If no reliable detection was determined in the point cloud.
No handle found	No handle found	If no handle is found that can be carried out without collision.
Handle failed	Handle failed	If the handle could not be executed successfully. Example: Handle sensor reports an error or a movement could not be executed successfully.

Behavior when trying again

This task supports the option of retrying execution in the event of an error.

If the handle was not executed successfully (e.g. timeout during movement at reduced speed or similar), the task is executed again as follows.

- If the error occurs before a part has been gripped:

  Another attempt is made to perform the grip.

- If the error occurs during or after a part has been gripped:
  - The gripper is opened and falls back into the crate.
  - A movement back to the transition pose is planned and executed.
  - If the movement to the transition pose was executed successfully, the Mikado program sequence is continued with the sequence change 'grip failed'.

3.8.7 Pick-Search-Statistics

Retrieve information about the given pick-search and its best pick so far.

The task does not stop the search and does not fix the pick-choice, so better picks may still appear later.

This task is useful for pick-searches with multiple grippers. The task can be used e.g. to read out the best picks' gripper, which can then be used to change the current gripper if necessary. The *Pick object* only considers picks belonging to the then current gripper.

Input-Variables

Name	Type	Description
Product	Product | Integer	Name or ID of the product searched for. If the field is left empty, the active product is used. Product names are defined in the *Product management*.
Box	Box | Integer	Name or ID of the box in which the product was searched. If the field is left empty, the active box is used. Box names are defined in the *Scene management*.
possible End-Effector(s)	Gripper | StringList | Integer | IntegerList	One or more names or IDs for the grippers to determine the pick-search, for which information is to be read out. If the field is left empty, the last started pick-search is used.

Output-Variables

Name	Type	Description
Num. Objects Recognized	Integer	Number of Objects found
Num. Objects Pickable	Integer	Num. of Objects for which a valid Pick has been found
Num. Objects Pickable (Preferred)	Integer	Num. of Objects for which a valid Pick of category “Preferred” has been found
Num. Objects Pickable (Standard)	Integer	Num. of Objects for which a valid Pick of category “Standard” has been found
Num. Objects Pickable (Experimental)	Integer	Num. of Objects for which a valid Pick of category “Experimental” has been found
Object Name	String	Name of the Object of the best Pick found so far
Object Pose	Pose	Pose of the object of the best pick found so far
Collision Type	Integer	Collision-Type of the best Pick found so far (allowed collisions: 2, no collisions: 0)
Gripper Name	Gripper | String	Gripper of the best Pick found so far (useful for searches with multiple grippers)
Gripper Id	Integer	ID of the Gripper of the best Pick found so far (useful for searches with multiple grippers)
Pick Category	String	Category of the best Pick found so far (“Preferred” / “Standard” / “Experimental”)
Pick Subcategory	Integer	Subcategory of the best Pick found so far (0-10)
Grasp-Name	String	Grasp-Name of the best Pick found so far
Grasp-Offset	Pose	Offset of the Grasp of best Pick found so far relative to its Object
Grasp-Tcp-Pose	Pose	Tcp-Pose of the grasp-point of the best Pick found so far
Grasp Tool0-Pose	Pose	Tool0-Pose of the grasp-point of the best Pick found so far
Start Tcp-Pose	Pose	Tcp-Pose of the start-point of the best Pick found so far
Start Tool0-Pose	Pose	Tool0-Pose of the start-point of the best Pick found so far
Approach Tcp-Pose	Pose	Tcp-Pose of the approach-point of the best Pick found so far
Approach Tool0-Pose	Pose	Tool0-Pose of the approach-point of the best Pick found so far
Retract Tcp-Pose	Pose	Tcp-Pose of the retract-point of the best Pick found so far
Retract Tool0-Pose	Pose	Tool0-Pose of the retract-point of the best Pick found so far
Final Tcp-Pose	Pose	Tcp-Pose of the end-point of the best Pick found so far
Final Tool0-Pose	Pose	Tool0-Pose of the end-point of the best Pick found so far
Total Collision-Depth	Double	Total collision-depth of all steps of the best pick found so far
Maximum Collision-Depth	Double	Maximum collision-depth of all steps of the best pick found so far
Average Collision-Depth	Double	Average collision-depth of all steps of the best pick found so far

Sequence change

Name	Condition	Remark
No box found	No box found	
No object found	No object found	
No pick found	No valid pick found	No pickable objects available

3.8.8 Discard part

Opens the gripper and removes the attached object from the collision world.

Hint: If you have picked a part with the *Pick object* task, it must also be dropped with the 'Drop part' task. If the *Open/close hand* task is used to release a picked part, the gripper will open, but the part will still be present in the collision world and will still move with the robot.

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.8.9 Clear Recognized Objects and Pointcloud

Remove all found objects and the point cloud (octomap) from the collision world.

Example application: If crates with different products are used in the same position (e.g. by manually swapping crates), you must delete the current finds when changing crates. Otherwise, finds are used in Mikado for collision planning that no longer exist in this case.

Warning: This task cancels all ongoing pick searches. This means that before anything else can be grasped, a new search must be started with the task '*Start handle search*'. Otherwise, executing the '*Pick object*' task will result in an error.

3.8.10 Add static object

Inserts an instance of a product at any position in the world. This is bypassed during path planning, for example, and is used for visualization. The added object can then be attached to the TCP (*Attach static object*) or removed again (*Remove static object*).

Example: An object has been placed on a clipboard in a known pose. As soon as a part is deposited (*Discard part*), it is no longer taken into account by Mikado in the path planning. This task can be used to avoid collisions between the robot (and any attached parts) and the static object during subsequent movements.

Hint: The object ID must be unique and must not yet exist, i.e. the object ID must not match the name of a scene object (crate, obstacle) or a find ("product name NUMBER") in order to avoid unwanted behavior.

Variablen

Name	Typ	Description
Product	Product, Integer	Name of the product that serves as a template
Object ID	String	Identifier of the new object (for *Remove static object*)
Object pose	Pose	Position of the created object

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.8.11 Remove static object

Removes an object previously *added by a task* from the scene. It is then no longer taken into account as a collision object.

Hint: A static object that was attached to the TCP with the *Attach static object* task cannot be removed with this task. It can only be deposited with the *Discard part* task and removed from the scene at the same time.

Variablen

Name	Typ	Description
Object ID	String	Identifier of the object to be removed (from *Add static object*)

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.8.12 Attach static object

Attaches a previously added static object (*Add static object*) to the TCP. This is taken into account during path planning and in the visualization, for example, and is moved together with the robot.

The object can be dropped again with the *Discard part* task. It is then automatically removed from the scene and is no longer available as a scene object.

Example: An object was placed on a clipboard in a known pose and added as a static object to avoid collisions. The object is blindly grabbed from the clipboard and attached to the TCP by this task. In this way, collisions between the object and the cell can be avoided during subsequent movements, and movements can be carried out in part coordinates (*Move part to*).

Hint: To attach an object to the gripper, another part must not already be gripped. The object ID must exist. If no static object with the corresponding object ID exists, the task fails.

Variablen

Name	Typ	Description
Object ID	String	Identifier of the object (from *Add static object*)

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.8.13 Read out product dimensions

Reads the dimensions of the axis-aligned bounding box (AABB) of the active product.

Hint: The dimensions of the AABB can also be accessed via the JavaScript API, see *getDimensions*.

Variablen

Name	Typ	Description
X	Double	Output variable (optional): Dimension in X-direction of the product coordinate system [m]
Y	Double	Output variable (optional): Dimension in Y-direction of the product coordinate system [m]
Z	Double	Output variable (optional): Dimension in Z-direction of the product coordinate system [m]

Detailed description

Calculates the smallest possible enclosing box for the active product and returns its dimensions. The orientation of the cuboid is aligned with the product coordinate system (Axis-aligned bounding box AABB). In the following sketch, the AABB is indicated by a yellow transparent cuboid.

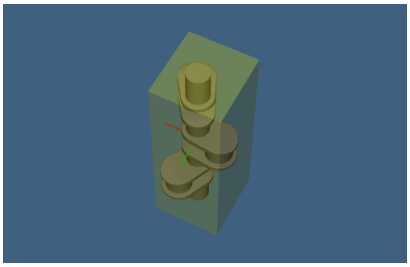


Fig. 10: Principle sketch of the task *Read out product dimensions*

3.8.14 Adjust Object Pose

Adjusts the pose of the attached (grabbed) part. This is taken into account in the collision world and used for path planning during movements using *move piece (Cartesian) to* (*Move part to*).

Example: A part is removed from the box using a magnetic gripper and must then be placed in a turning station. When the part is pulled out, it shifts slightly so that it can no longer be fed correctly to the turning station. Now the part can be moved under a camera, the correct pose can be determined using *Find object* (*Find object*) and readjusted using *Adjust object pose* (*Adjust Object Pose*). The part can then be fed with the correct position and orientation.

Hint: The input variables relating to translation and rotation tolerance can be used to execute a sequence change if the pose of the part found differs too much from the expected pose.

Variables

Name	Type	Description
Goal Pose	Pose	Pose to which the attached part is moved to
Tolerance translation	Double	Permissible translational difference [m] between the pose of the attached part and the goal pose
Tolerance rotation	Double	Permissible rotational difference [°] between the pose of the attached part and the goal pose

Sequence change

Name	Condition	Comment
Tolerance exceeded	Deviation between part pose and target pose > Tolerance	If the deviation of the translation or rotation of the pose of the attached part and the goal pose is greater than the permissible tolerance, you can continue with a different sequence

Hint: This task triggers a pre-run stop. This means that all robot movements accumulated from previous tasks are executed and must be completed before this task is executed.

3.9 Transformations

3.9.1 Split pose

This task is used to split a `Pose` into its individual position and orientation components. The results are the xyz distances and Euler angles, i.e. roll, pitch and yaw.

Variablen

Name	Typ	Description
Pose	Pose	–
x	Double	Distance in meters (opt)
y	Double	Distance in meters (opt)
z	Double	Distance in meters (opt)
Roll	Double	Angle in degrees (opt)
Pitch	Double	Angle in degrees (opt)
Yaw	Double	Angle in degrees (opt)

Hint: Individual output values can be omitted if they are not required.

3.9.2 Create pose

This task can be used to create a `Pose` from a combination of different input values. The input values can be an input pose and individual position and orientation components (x, y, z, roll, pitch, yaw). All inputs are optional.

If an input pose is used, it forms the basis for the new pose. All additional individual components set for this pose are replaced by the given value. Components that are not set remain unchanged from the input pose.

If no input pose is used, the new pose is based on a zero pose. This means that all optional position and orientation components that are not set are given the value `0 m` or `0°`.

Variablen

Name	Typ	Description
Input pose	Pose	optional
x	Double	Distance in meters (opt)
y	Double	Distance in meters (opt)
z	Double	Distance in meters (opt)
Roll	Double	Angle in degrees (opt)
Pitch	Double	Angle in degrees (opt)
Yaw	Double	Angle in degrees (opt)
Output pose	Pose	–

Hint: The direction of rotation is determined by the right-thumb rule: If an imaginary hand grips around an axis of the coordinate system so that the thumb points away from the origin, then the remaining fingers towards the tip specify the positive direction of rotation.

Hint: Rotation by several differential angles is carried out according to the RPY convention and thus successively for the roll, pitch and yaw angles.

3.9.3 Rotate Pose Global

This task is used to rotate poses in Euler angles, i.e. roll, pitch and yaw. The difference angles refer to the global coordinate system of the robot.

Variablen

Name	Typ	Description
Pose	Pose	Input Pose
Roll	Double	Angle in degrees
Pitch	Double	Angle in degrees
Yaw	Double	Angle in degrees
Result	Pose	Result Pose

Hint: The direction of rotation is determined by the right-thumb rule: If an imaginary hand grips around an axis of the coordinate system so that the thumb points away from the origin, then the remaining fingers towards the tip specify the positive direction of rotation.

Hint: Rotation by several differential angles is carried out according to the RPY convention and thus successively for the roll, pitch and yaw angles.

Example

A rotation of the yaw angle by 90° is shown in an example. Changes in the yaw angle correspond to rotations around the z-axis (blue). The image shows both the axes of the global coordinate system, which lies at the robot origin. As well as those of the local coordinate system of the pose to be changed, which is located here in the robot flange. If you now look at the difference after the task has been executed, you will see a change in the orientation of the pose coordinate system and robot flange. The pose was rotated (locally), so the position of the pose has not changed. The yaw rotation referred to the blue, local z-axis.

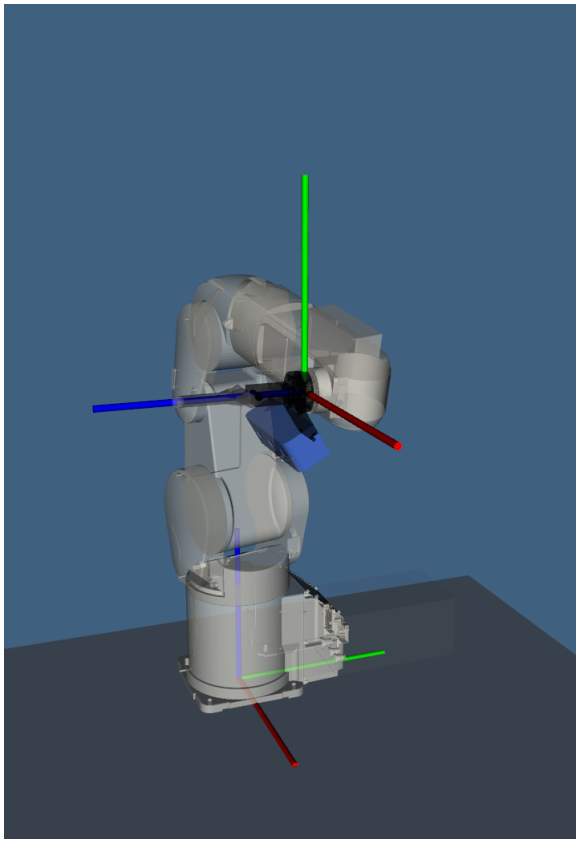


Fig. 11: Screenshot: initial Pose

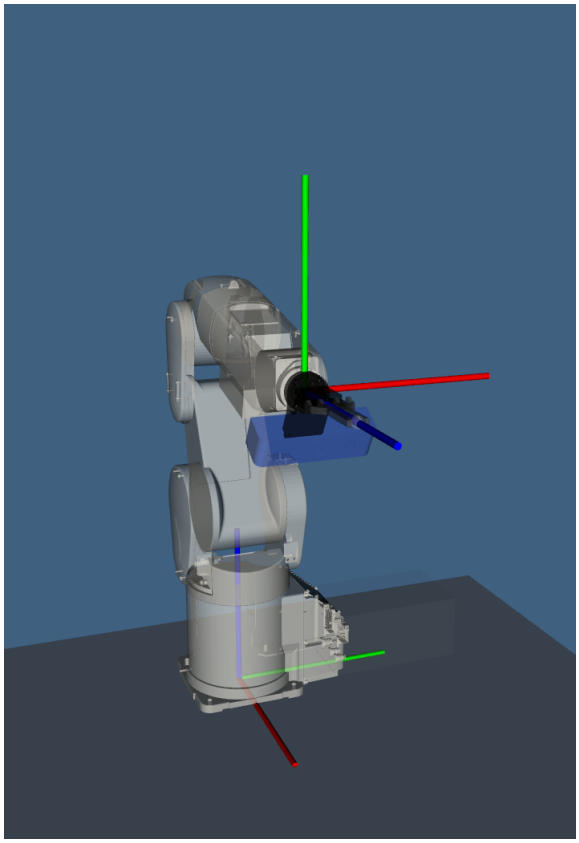


Fig. 12: Screenshot: Pose after global rotation

3.9.4 Rotating pose Local

This task is used to rotate poses in Euler angles, i.e. roll, pitch and yaw. The difference angles refer to the local coordinate system of the robot flange.

Variablen

Name	Typ	Description
Pose	Pose	Input Pose
Roll	Double	Angle in degrees
Pitch	Double	Angle in degrees
Yaw	Double	Angle in degrees
Result	Pose	Result Pose

Hint: The direction of rotation is determined by the right-thumb rule: If an imaginary hand grips around an axis of the coordinate system so that the thumb points away from the origin, then the remaining fingers towards the tip specify the positive direction of rotation.

Hint: Rotation by several differential angles is carried out according to the RPY convention and thus successively for the roll, pitch and yaw angles.

Example

A rotation of the yaw angle by 90° is shown in an example. Changes in the yaw angle correspond to rotations around the z-axis (blue). The image shows both the axes of the global coordinate system, which lies at the robot origin. As well as those of the local coordinate system of the pose to be changed, which is located here in the robot flange. If you now look at the difference after the task has been executed, you will see a change in the orientation of the robot flange and pose coordinate system. The pose was rotated locally, so the position has not changed. The yaw rotation referred to the local z-axis (blue), which has also not changed as a result.

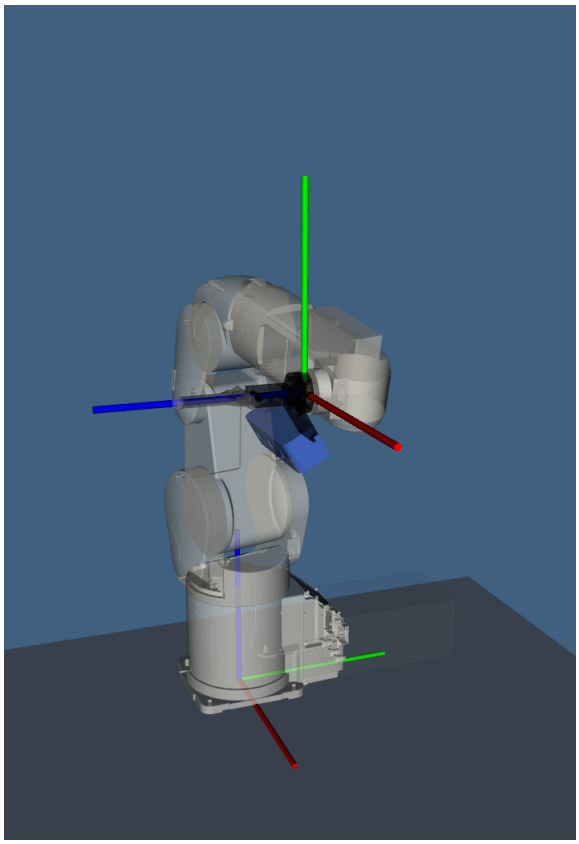


Fig. 13: Screenshot: initial Pose

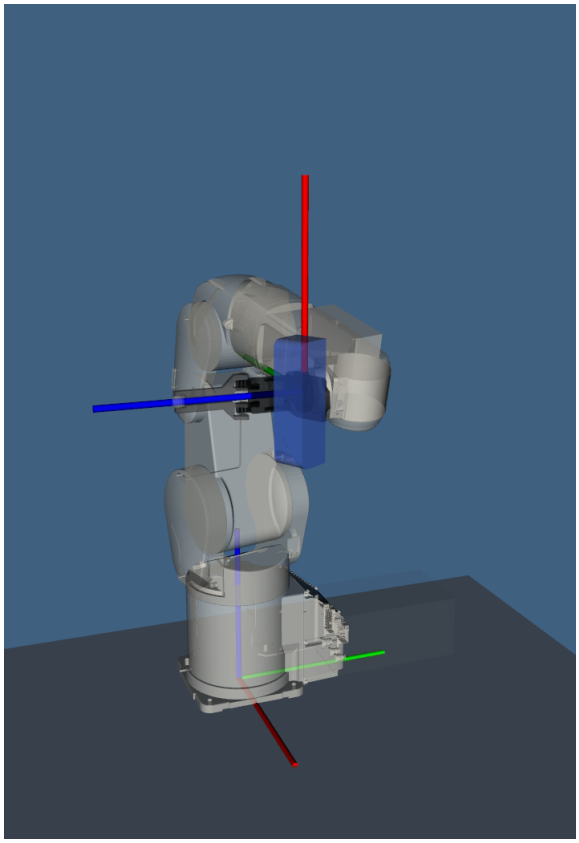


Fig. 14: Screenshot: Pose after local rotation around z-axis (yaw)

3.9.5 Move Pose Global

This task is used to move poses, i.e. to apply a translation to them. The pose is moved by difference distances as input values of this task in the global coordinate system of the input pose.

Variablen

Name	Typ	Description
Pose	Pose	Input Pose
x	Double	Distance in meters
y	Double	Distance in meters
z	Double	Distance in meters
Result	Pose	Result Pose

Example

In one example, a translation in the Z direction is performed globally by 0.1m. The image shows both the axes of the global coordinate system, which is located in the robot origin. As well as those of the local coordinate system of the pose to be changed, which in this case is in the robot flange. If you now look at the difference after the task has been executed, you will see a change in the position of the pose coordinate system and robot flange. The pose has been moved globally in the direction of the blue z-axis (in the robot origin).

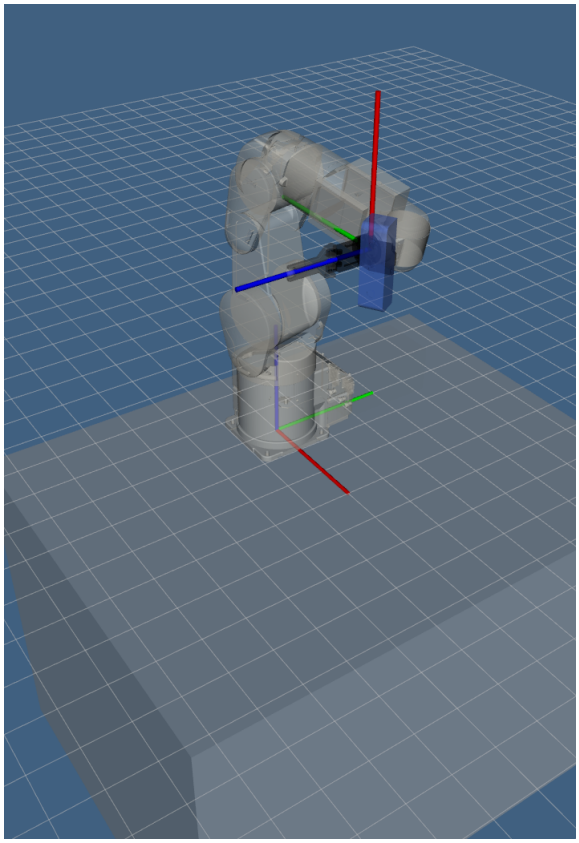


Fig. 15: Screenshot: initial Pose

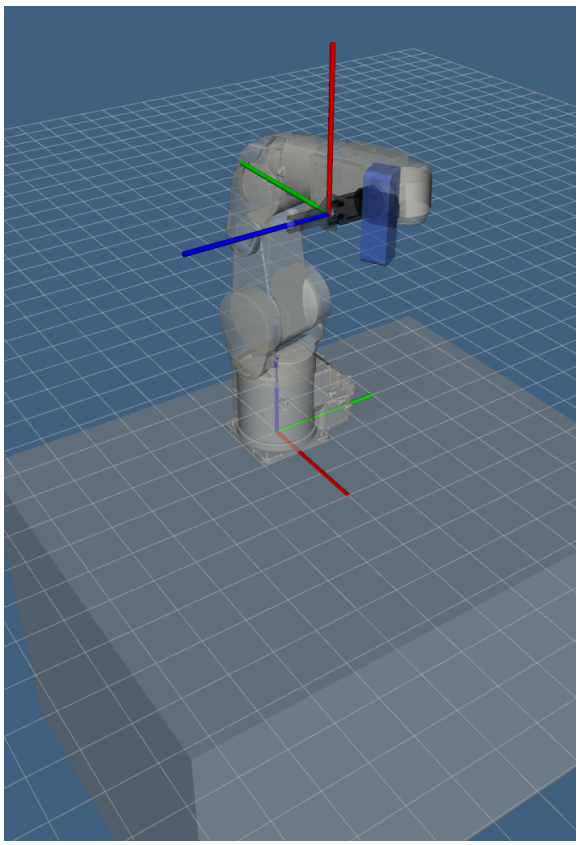


Fig. 16: Screenshot: Pose after global shift

3.9.6 Move Pose Local

This task is used to move poses, i.e. to apply a translation to them. The pose is moved by difference distances as input values of this task in the local coordinate system of the input pose.

Variablen

Name	Typ	Description
Pose	Pose	Input Pose
x	Double	Distance in meters
y	Double	Distance in meters
z	Double	Distance in meters
Result	Pose	Result Pose

Example

In one example, a translation in the Z direction is performed locally by 0.1m. The image shows both the axes of the global coordinate system, which is located in the robot origin. As well as those of the local coordinate system of the pose to be changed, which in this case is located in the robot flange. If you now look at the difference after the task has been executed, you will see a change in the position of the pose coordinate system and robot flange. The pose has been moved locally in the direction of the blue z-axis (in the robot flange).

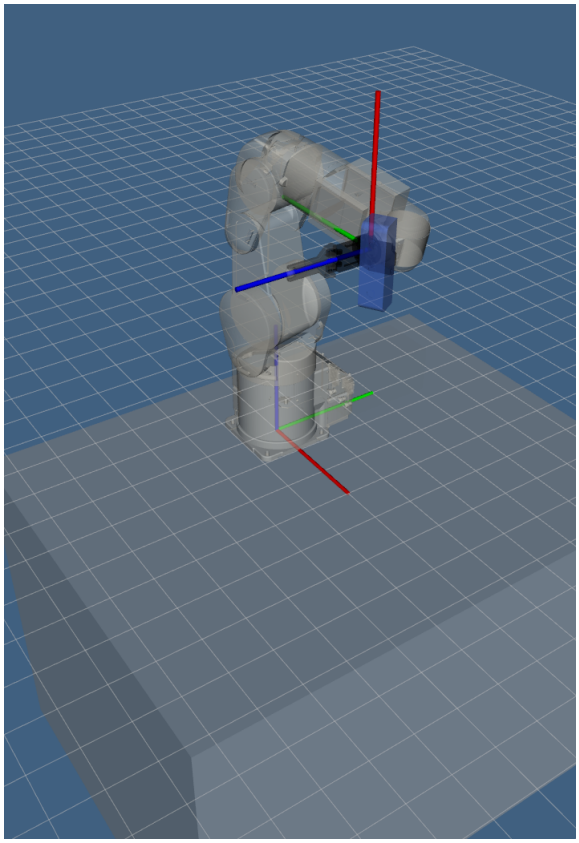


Fig. 17: Screenshot: initial Pose

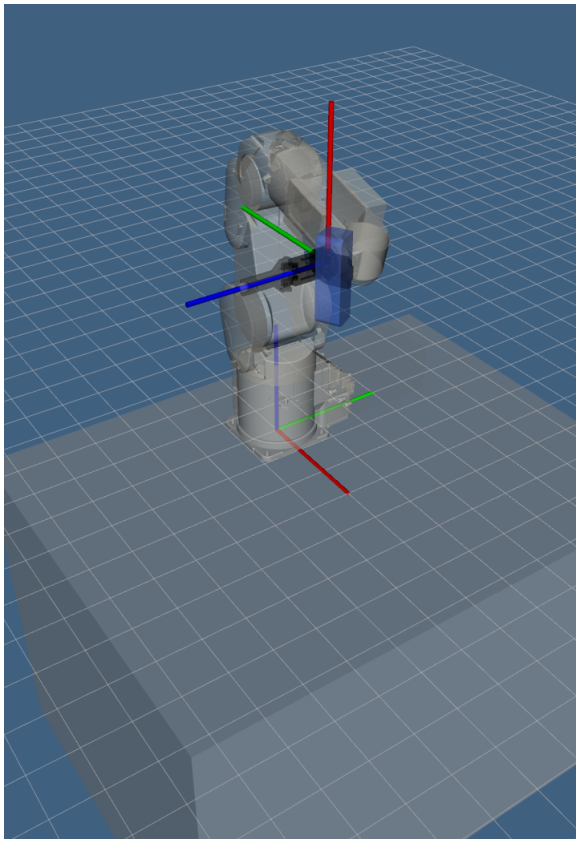


Fig. 18: Screenshot: Pose after local displacement

3.9.7 Pose transformieren

Multiplies two poses/transformations, for example to apply an offset to a recording pose.

Hint: The shift (and rotation) of the transformation pose takes place relative to the input pose. By swapping the input values, a global transformation can be carried out instead of a local one: to achieve a shift in the Z direction of the world coordinate system, for example, use the Z offset pose as the input pose and the pose to be shifted as the transformation.

Variablen

Name	Typ	Description
Pose	Pose	Input pose
Transformation	Pose	Input transformation as a pose
Result	Pose	Result of the transformation as a pose (= pose * transformation)

3.9.8 Inverse Transformation

Calculate the inverse transformation of a given transformation/pose.

Variablen

Name	Typ	Description
Transformation	Pose	Input transformation / pose
Inverses	Pose	Output variable: Inverse transformation of the input transformation

3.10 Extensions

3.10.1 JavaScript

Execute user-defined JavaScript code to implement additional actions that are not available as a task. These can be mathematical calculations, complex logic or interfaces to other systems, for example. All variables can be accessed. Sequence changes can also be carried out.

Further information on this topic in section *Extending Mikado Control* / *JavaScript Task*.

Variablen

Name	Typ	Description
Code	JScript	The code snippet to be executed as source code

3.10.2 Execute Halcon Script

Execute a Halcon script using the HDevEngine.

If application-specific image processing steps are to be implemented using the Halcon library, native Halcon scripts can be integrated into the Mikado process. To do this, the Halcon program must be stored as an .hdev file in the 'scripts' folder of the project directory.

You have access to the last camera recordings and all Mikado variables.

Further information on this topic in section *Extending Mikado Control* / *Halcon Script Task*.

Variablen

Name	Typ	Description
Path to the .hdev	String	Name or path of the .hdev file (absolute path or relative path to the project directory)

CHAPTER

FOUR

TUTORIALS

This chapter contains instructions and examples to illustrate the use of Mikado.

4.1 An initial process in ten minutes

The following steps allow you to quickly create a minimal Mikado working environment. This is based on a set-up cell, calibrated and configured camera.

4.1.1 Set up recording pose

- In the main menu, click on Tools and then Change robot
- Move the robot to a suitable recording pose by jogging

4.1.2 Set up scene

- Create crate 1, with parameterization: fixed crate
- Load 3D model
- Record point cloud
- Bring the box into overlap with the point cloud
- Create ROI "Search area" for object search and move it so that the bottom of the box is cut out, e.g. to z=7 mm
- Place crate 2 as above. Move to a suitable storage position

4.1.3 Set up product

Now a product is set up, i.e. the model is loaded, a simple grasp is defined, configured and recognized. Meshes may need to be preprocessed, which is *described separately*. Setting up a product and recognizing it in the point cloud is explained in more detail *in another tutorial* :

- On the Edit -> Product page, a new product can be created and the name customized.
- The most important parameters are then adjusted:

Name	Value
Mesh	Path to the mesh file
min. Recognition score	Start value low, 20%, then tune
Travel distance	cartesian approach to bypass object contour
Removal distance	~Partial height, Cartesian "pull out", collisions permitted
Symmetry	select according to the object
Minimum distance	~ half partial diameter, how close can two objects be?

- A suitable outside/inside handle must be defined, for example by selecting the corresponding standard handle and adjusting it using the graphic marker
- the standard inner/outer handle that is not required can be removed
- Save product list
- Save product

4.1.4 Save product

- Switch to the object recognition page
- Check the “Roi” box
- Record point cloud + apply and find objects
- Adjust minimum score of the product

Hint: In the event of detection problems, please check whether a point cloud appears after clicking on Record point cloud and whether the new product is selected as the active product (lower status bar). A standard tuning procedure is to start from a low detection score and iteratively increase it until no more false detections are shown. This should be tried out for various scenarios.

4.1.5 Create minimum program

A new “Sequence” sequence is created, into which the following tasks are gradually dragged:

- Set Active ROI, parameterization: create new string variable, fill with “search range” and set initial value
- Move robot (new pose variable “Recording pose”, accept current robot position, set initial value)
- Record point cloud
- Start handle search
- Grab part
- Move robot (new pose variable “drop pose”, do not teach in yet)
- Discard part

4.1.6 Teach in poses and adjust variables

- Switch to the Edit tab and submenu item Variables
- Drop pose: “Accept current robot pose”, manually adjust values, set initial value

4.1.7 Let it run and vary a little

- For example, error handling should be carried out by jumping to corresponding further sequences in the Gripping Task part. In the minimum case, these then contain an error task that gives the operator a suitable instruction or, in advanced cases, vary the sequence (different pick-up pose or crate)
- More handles can be added. In the handle inspector, the user can see which handles do not work and why (step through the program and stop after “Start handle search”)
- For example, an experimental handle can be inserted. For handling in the program, a bool variable must be inserted in the “Part glreifen” task. The program then checks whether this has been set to true and possibly jumps to a sequence with a different drop variant

- A variation that optimizes the cycle time is the following: After the grip, jump to a new “grip loop” sequence, record the point cloud here, start the grip search, then move to the drop pose, drop the part and grip the part. Then jump to “Grip loop”. The new grip is calculated in the background.

4.2 A new product up to object recognition

In this Mikado tutorial, you will learn how to set up a new product using the Mikado interface. All you need is a suitable CAD model in STL file format. You can find out how to optimally prepare your CAD model in a *separate tutorial*.

Note: This tutorial refers to the search algorithm Ensenso Partfinder.

4.2.1 Set up new product / product list

- Switch to the "Edit" → "Products" page

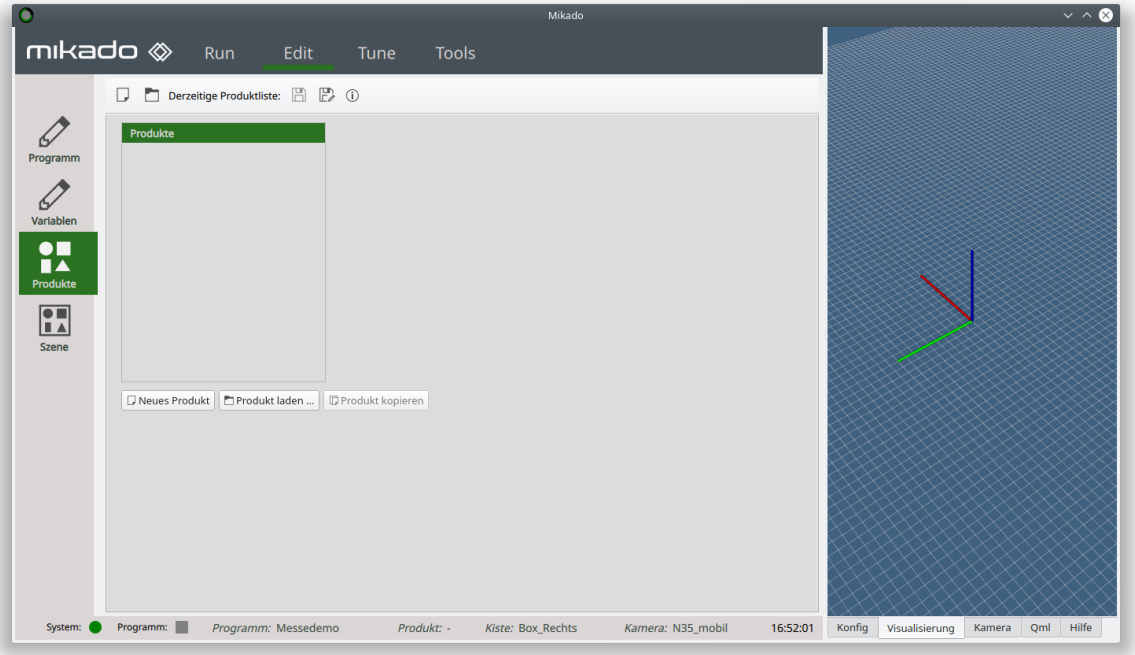


Fig. 1: Product page: Empty view if no products have been created yet

- The "New product" button adds a new product with the name "Product1" to the product list

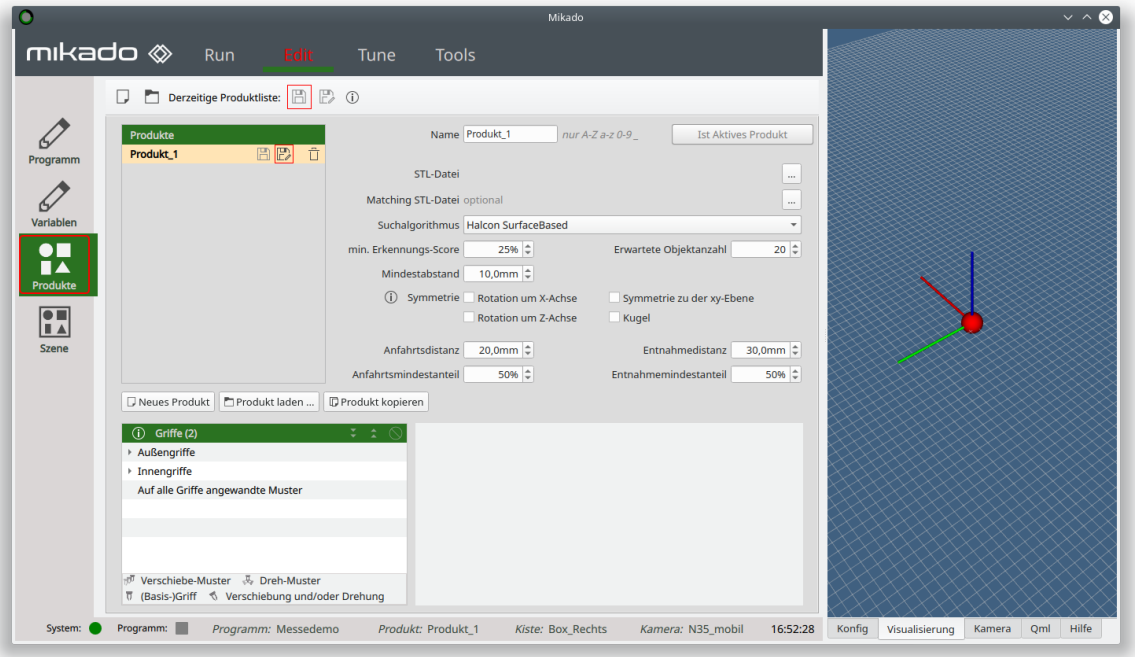


Fig. 2: Product page: View when a new product has been created

- The product name is entered in the “Name” field
- The CAD model in STL format can be specified under “STL file”. To do this, press the corresponding selection button labeled “…”. The loaded product model is now displayed in the visualization.

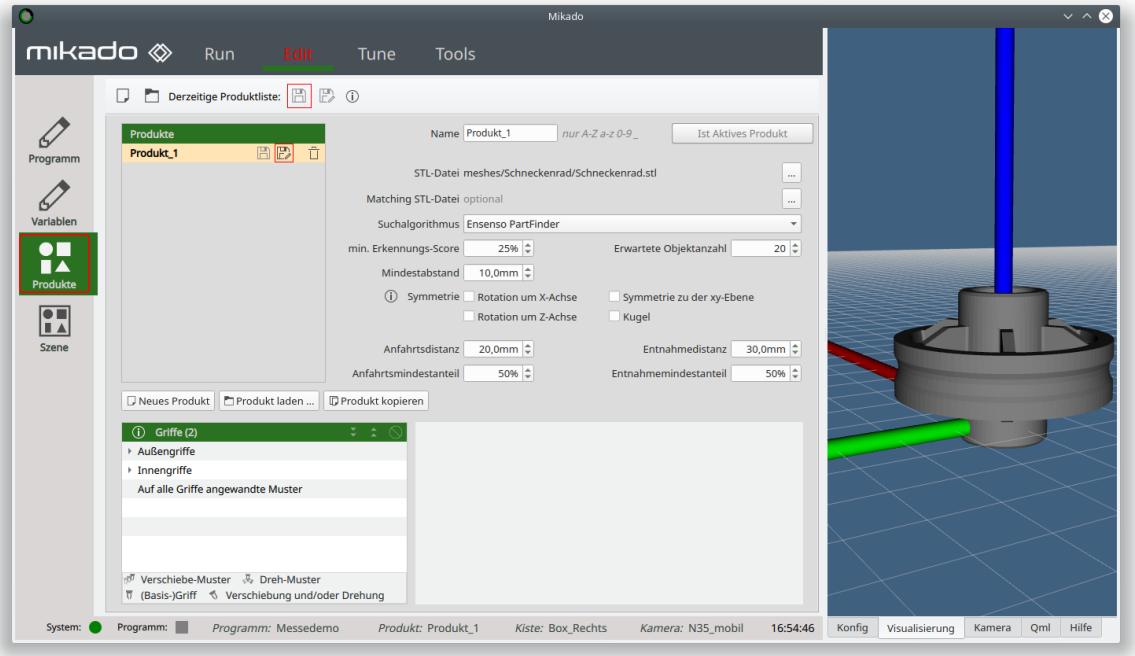


Fig. 3: Product page: View when a CAD model has been selected

- Adjust the parameters that are relevant for object detection:
  - Min. detection score: Initially select a very low value (approx. 10-25%). A suitable value will be

determined in a later step using the object detection tool.

  - Minimum distance: Robustness and performance improve with a suitable minimum distance. Set the largest possible value at which adjacent parts can still be detected.

- Details on all product parameters are described in the *Product management* chapter.
- Save
  - First save the product (floppy disk symbol in the product list) and select a suitable file name
  - Now you can also save the product list (upper diskette symbol directly below the main navigation bar)

4.2.2 Optimize object recognition

- Set active product
  - Switch to the "Edit" → "Products" page
  - Select the desired product from the product list
  - Click on the "Use as active product" button
  - In the *Status bar* you can check whether the correct product is now active
  - Switch to the "Tune" → "Detection" page
- Mikado offers buttons for selecting presets for the object recognition parameters. A distinction is made here between 'Model generation' and 'Search parameters', whereby the default settings 'Fast', 'Medium' and 'Accurate' are available. We recommend setting both parameter sets to 'Fast' to start with.
- Restrict perspectives. During model generation, the product is viewed from different perspectives. For each perspective, the product is converted into a corresponding point cloud, which is compared with the actual point cloud in the search process. If a product has rotational symmetries or is only visible from a certain direction, it makes sense to restrict the views accordingly. This both reduces the computing time and avoids false matches.
- Apply ROI → Select the desired ROI so that background data is pre-filtered.

Hint: An ROI is not absolutely necessary, but can have a strong influence on search duration and reliability. If an ROI has not yet been set up: Further information in chapter *Region of Interest (ROI)*.

- "Record point cloud" → "Apply and find objects"
- All detections above the set minimum detection score are visualized. The detection score is always higher than a detected object instance.

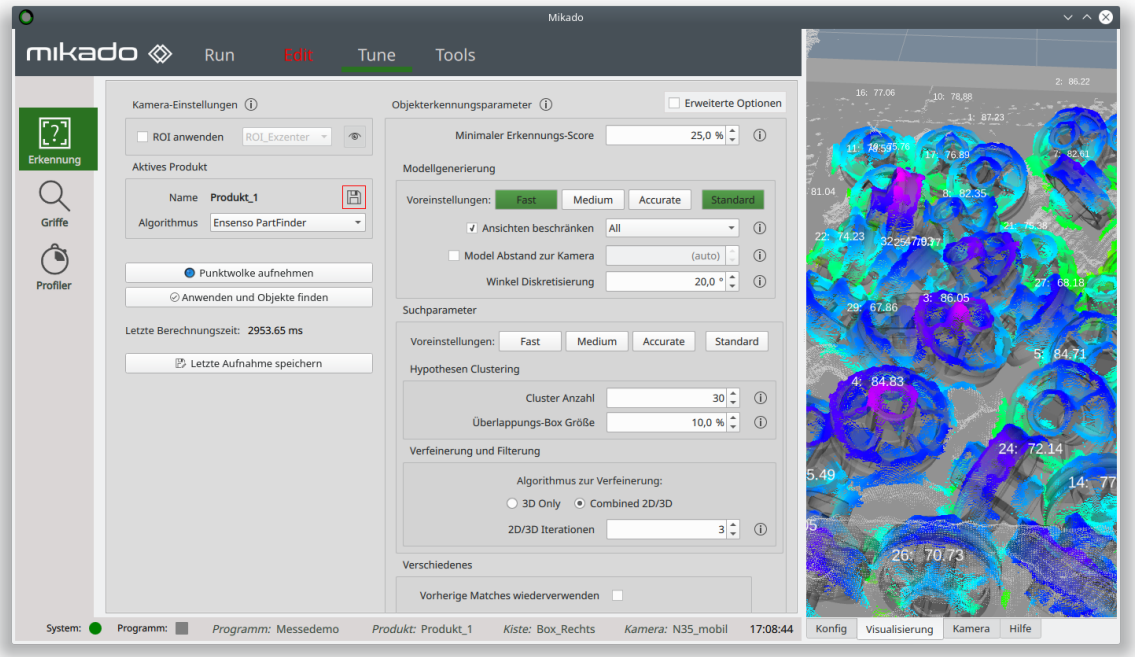


Fig. 4: Object recognition: Found parts are drawn in gray/transparent

- You can now evaluate which scores can be achieved and at which values inaccuracies or false positives occur. In our example with the “worm gears”, the scores for parts without concealment are over 75%. This is therefore a suitable minimum score here.
- If situations arise in your application where no part is detected, the score can also be set lower in this example. However, it is important to ensure that the value is not set too low.
- The parameters can also be adjusted based on the default settings. If the quality of the findings is too inaccurate with ‘Fast’, first select ‘Medium’ and then ‘Accurate’.
- You can also adjust the parameters individually. Behind each parameter in the Mikado interface is an info button that links to the corresponding documentation. At the top right is the ‘Advanced options’ button, which activates additional parameters.
- Now carry out further tests to test your new minimum score and parameters. Use different fill levels and different arrangements of the parts to test the robustness of the settings.
- Increase the minimum score as soon as you receive a false find. The new minimum score should be higher than the score of a false find.

4.3 Meshlab

MeshLab (http://www.meshlab.net/) is a free tool that can be installed and used on a Mikado controller. MeshLab can be used to view and edit mesh-based CAD files such as the STL file format. The following tutorials explain CAD editing options that enable optimum results in object recognition.

Of course, you can also use a tool of your choice to implement the modifications to your CAD model.

4.3.1 Basics: Loading and saving STL

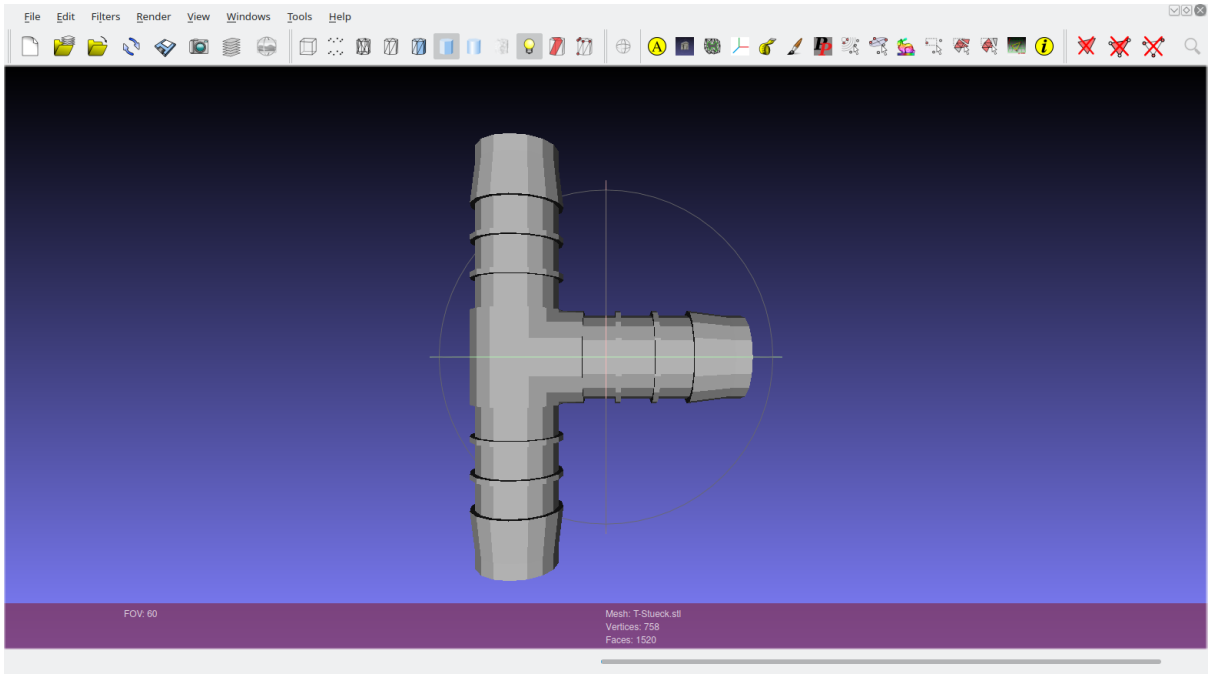


Fig. 5: MeshLab GUI

- Open MeshLab and load the desired product
  - Main menu → File → Import Mesh…
  - Select the desired file (e.g. in the mikado/cells/<cell name>/meshes/<file name>.stl directory)
  - Confirm “Post-Open Processing” message with OK
- Export STL file:
  - File → Export Mesh As…
  - Enter new file name
  - Select STL as format → Save

Hint: Save modified CAD files under a new name so that you can restore old versions at any time.

4.3.2 Center product coordinate system

- Display product coordinate system
  - Render → Show axis

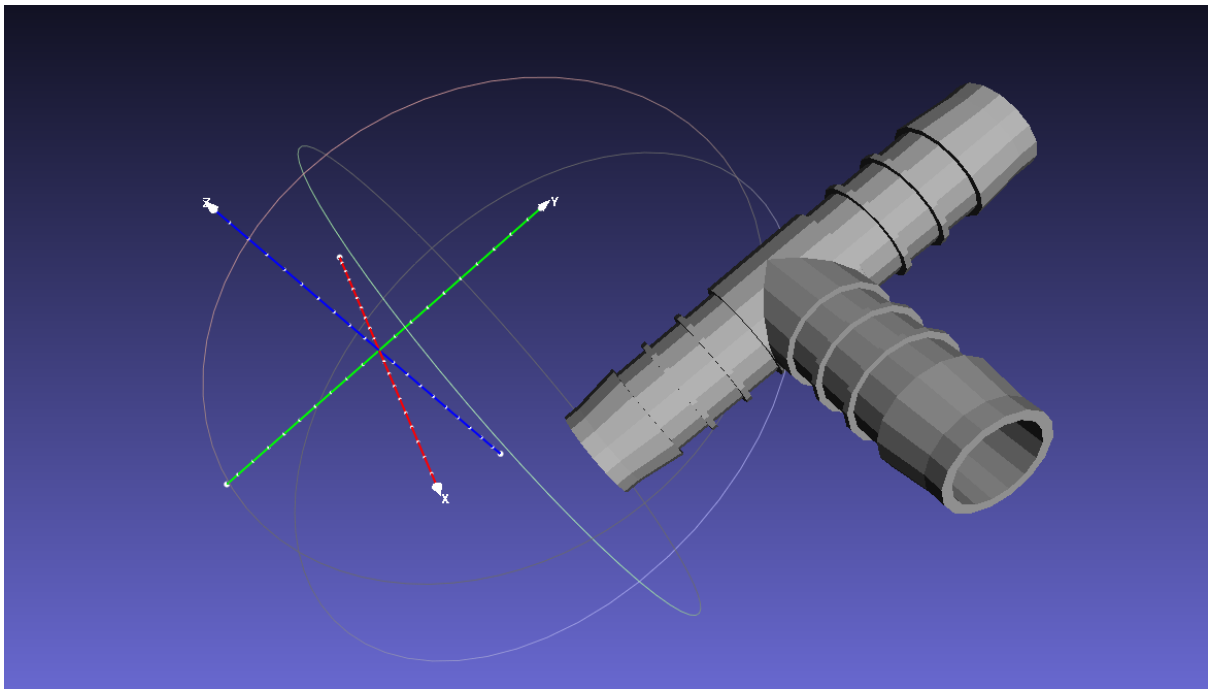


Fig. 6: MeshLab: Display coordinate system

- Center coordinate system
  - Filters → Normals, Curvatures and Orientation → Transform: Move, Translate, Center
  - “translate center of bbox to the origin”
  - Apply

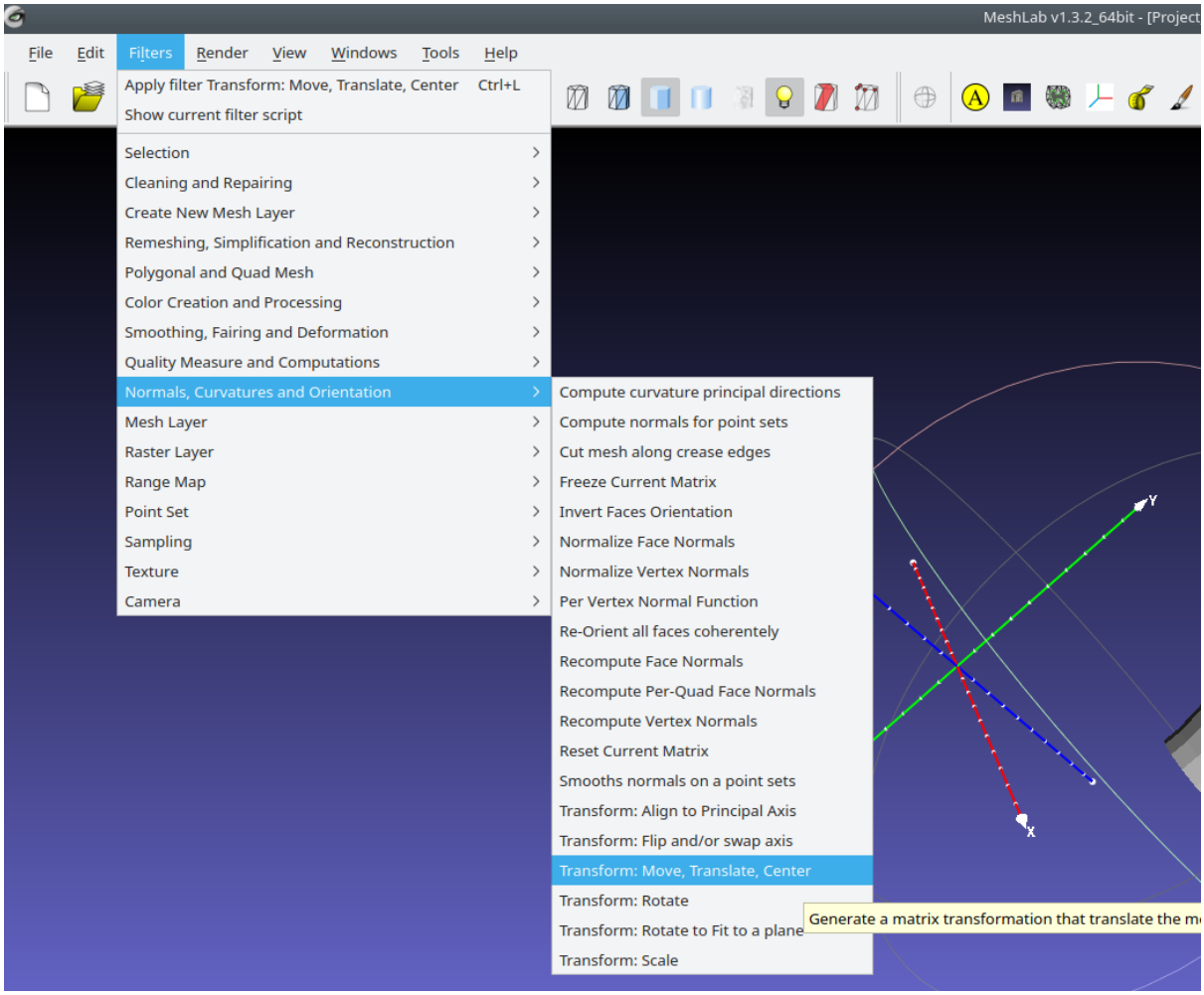


Fig. 7: MeshLab: Transform coordinate system

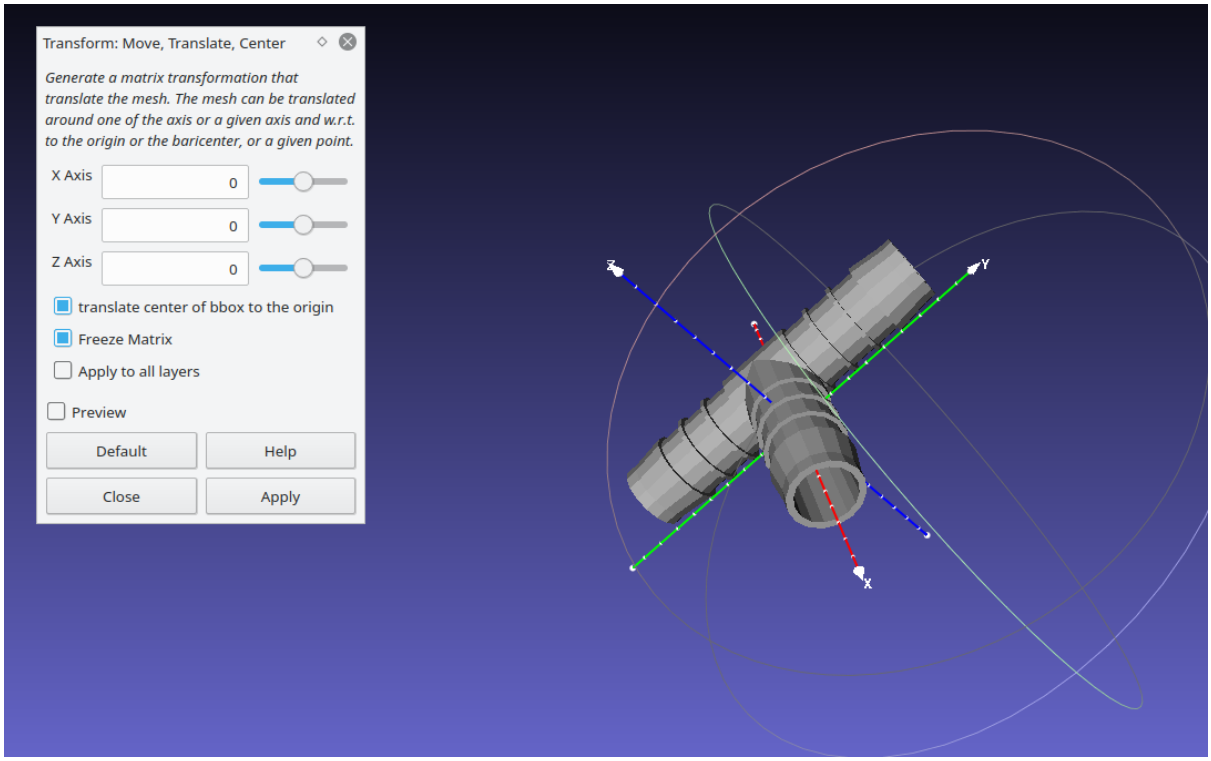


Fig. 8: MeshLab: Coordinate system centered

Hint:

- The coordinate system can also be moved and rotated manually using the "Transform" filter. If the STL is in a unit other than millimetres, this can also be converted here using the "Transform: Scale" filter (see *Scale STL*).
- A preview is possible for most filters by activating the "Preview" option.

4.3.3 Scale STL

If you want to change the size of your CAD or change the unit, you can scale the STL using Meshlab.

- Load CAD (see *Basics: Loading and saving STL*)
- Scaling CAD
  - Filters → Normals, Curvatures and Orientation → Transform: Scale

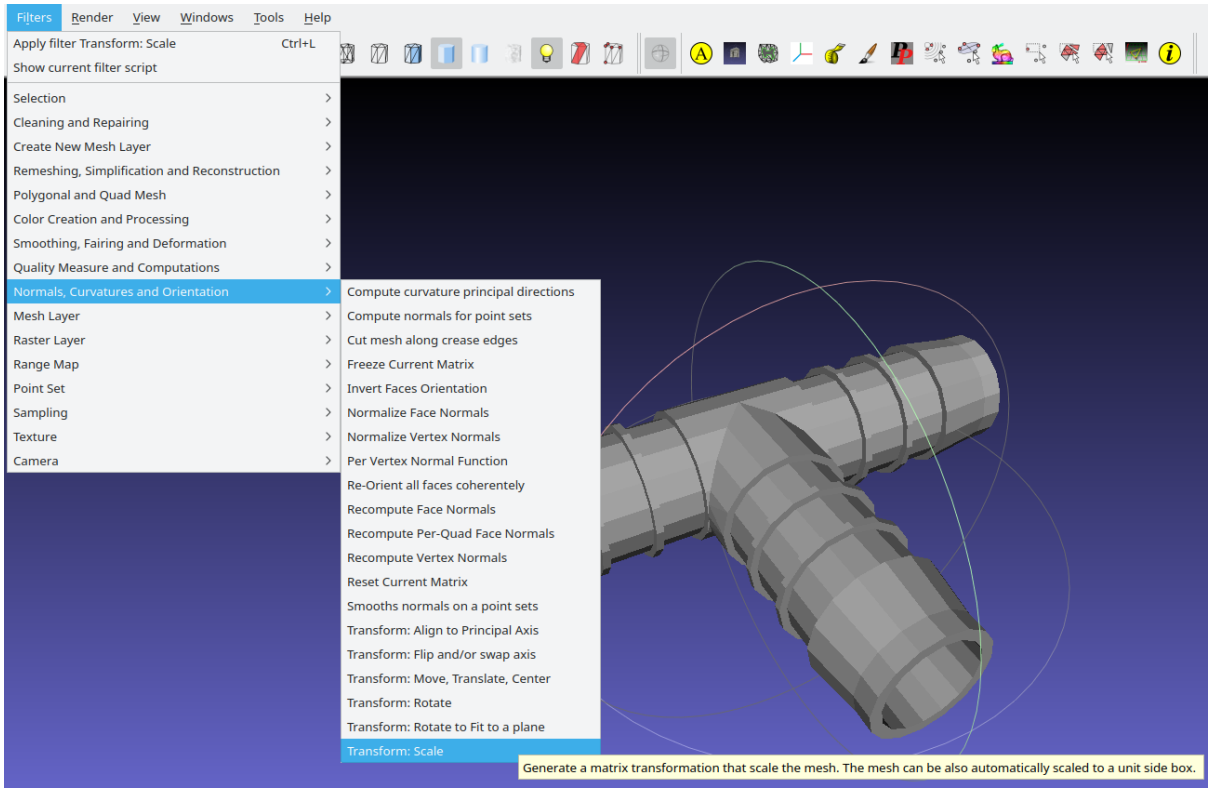


Fig. 9: MeshLab: Filter for scaling

- Activate “Uniform scaling”
- Activate “Preview” so that changes are displayed immediately
- Adjust the factor for “X axis” (1.0 → 100%, 0.01 → 1%). For example, if you want to convert a CAD that was defined in meters to millimeters, select the value 1000.0
- Apply

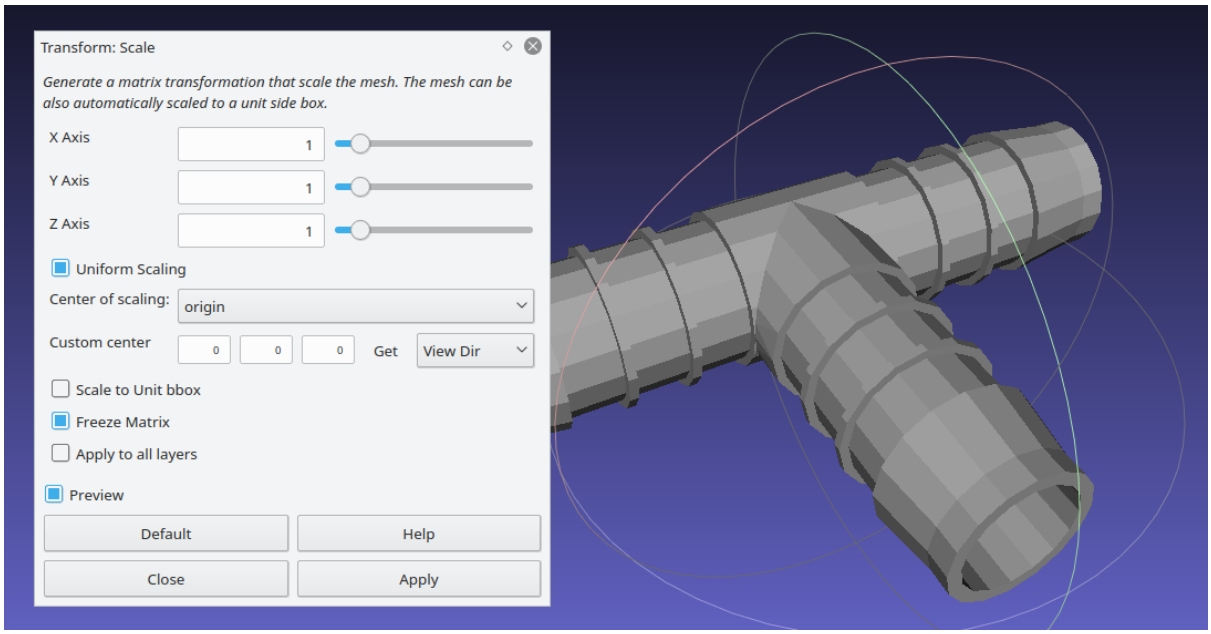


Fig. 10: MeshLab: Filter for scaling - Settings

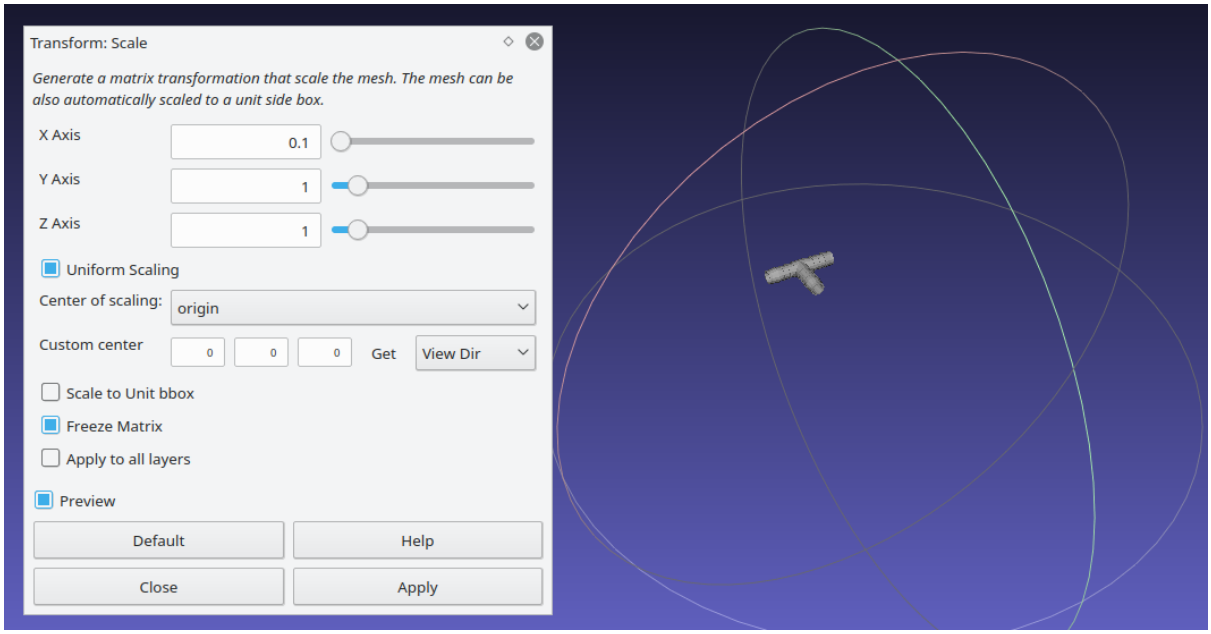


Fig. 11: MeshLab: Scaled STL

- Export STL (see *Basics: Loading and saving STL*)

4.3.4 STL Reduce number of triangles

The number of triangles of a product influences the loading time of a product and in particular the collision checks during path planning. It is recommended to reduce the number of triangles until important product features and structures are still recognizable.

- The current number of triangles is shown in the status display at the bottom
- Hauptmenü → Filters → Remeshing, Simplification and Reconstruction → Quadric Edge Collapse Decimation

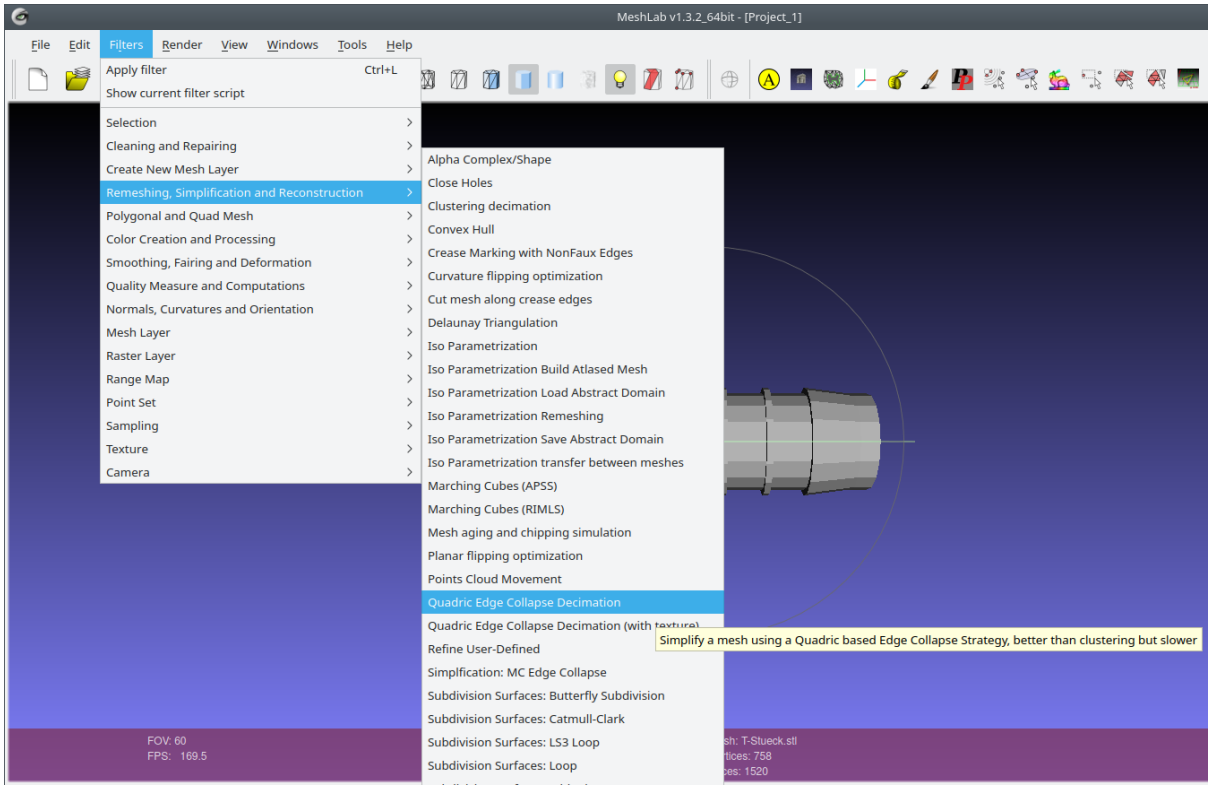


Fig. 12: MeshLab: Decimation filter

- Specify the desired number of triangles → Apply
  - Before applying a different parameterization, first reload the model ("Reload" button: )

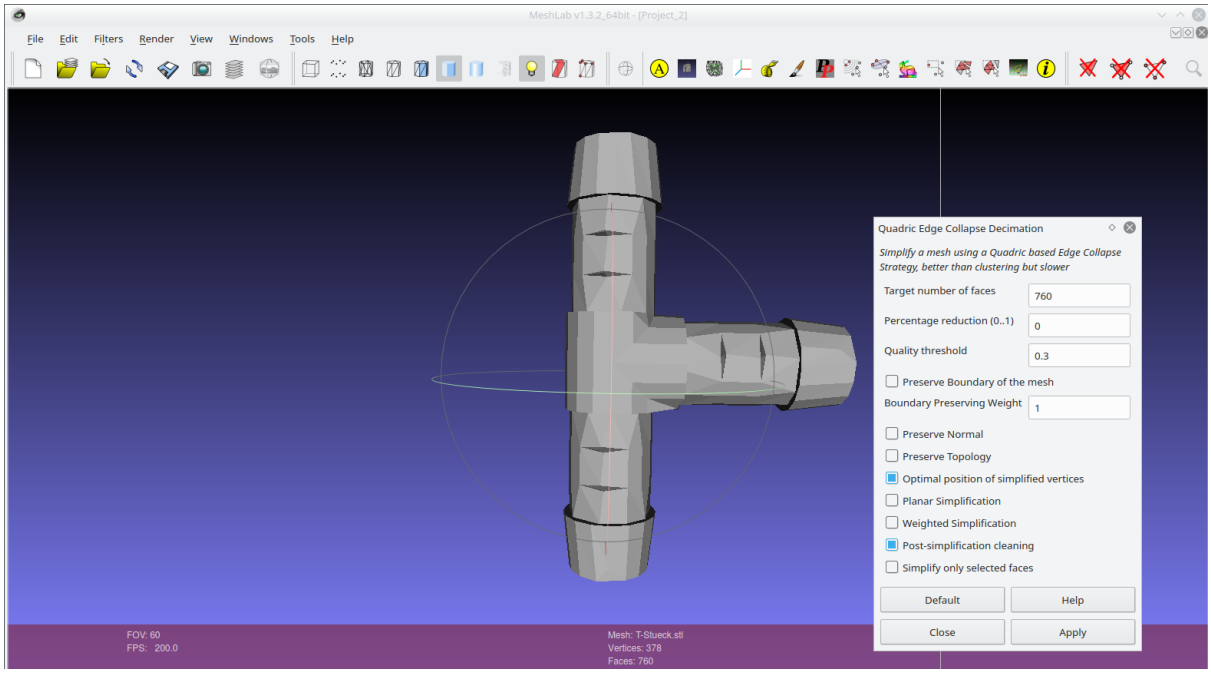


Fig. 13: MeshLab: Decimated model of the T-piece

4.3.5 Remove STL surfaces

Many CAD design data contain some surfaces and structures that are never visible from a camera view. These include internal structures or, depending on the application, the underside of a product. This property can also be used to reduce the CAD for symmetrical parts. A sphere, for example, is always visible from a single camera as a hemisphere at most. In this case, the lower half of a sphere can be removed for an object search. This increases the maximum score that can be achieved. This leads to more stable and faster object position detection.

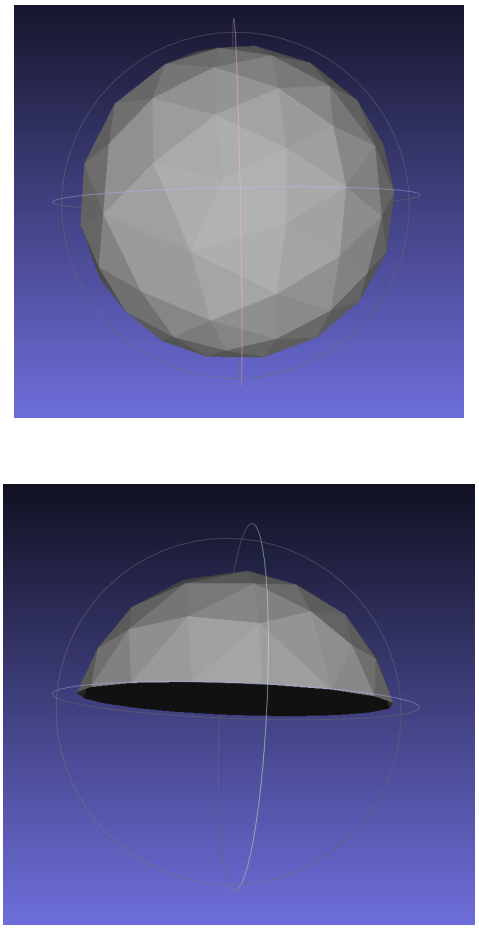


Fig. 14: Example: For object matching of a sphere, a hemisphere leads to higher scores. However, the collision model must still represent the complete product.

Hint: The *Product management* chapter describes how different CADs can be set up for object searches and collision checks.

The following steps describe how the number of triangles can be decimated.

1. **Load product** (see *Basics: Loading and saving STL*)
2. **Select areas** (marked areas are colored red)

    a) Option 1: **Rectangle selection**
    Select faces in a rectangular region (“Select Faces in a rectangular region”)

    - Select selection symbol
    - Select the area of the CAD with the mouse button held down. All triangles that project into the area are added to the selection.

Hint: Notes on rectangle selection:

- If you hold down the “Ctrl” key, the selected area is added to the existing selection
- If you hold down the “Shift” key, the selected area is removed from the existing selection
- The “ESC” button can be used to switch between selection and camera mode
- To exit the selection mode, click on the selected symbol again

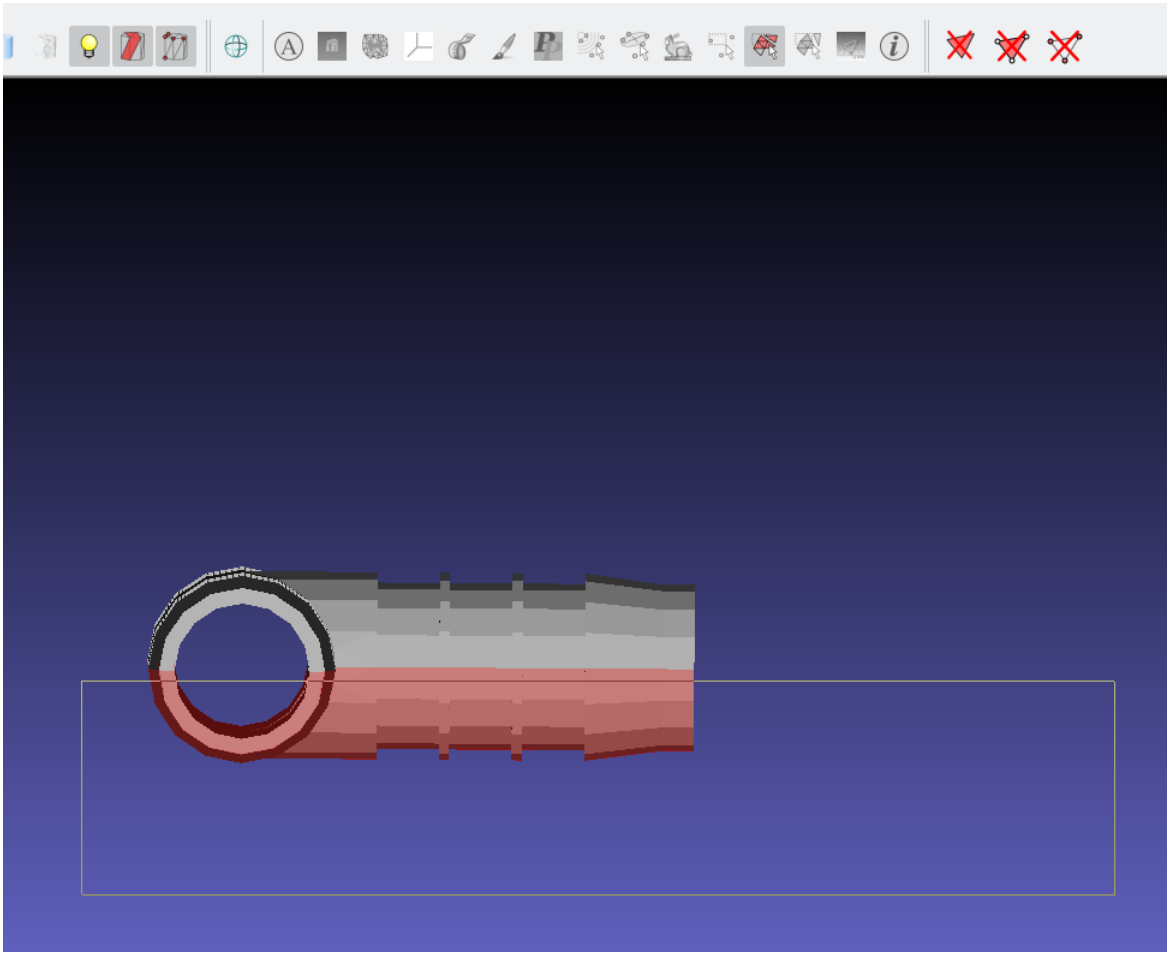


Fig. 15: MeshLab: Rectangle selection

b) Option 2: **Brush tool**
Select areas with the brush tool (“Z-Painting”)

- Click on the brush symbol
- Select marking brush (red brush)
- “Brush” areas with the mouse

Hint: Notes on the brush tool:

- Left mouse button adds areas to the selection
- Right mouse button removes the selection areas
- Brush type and size can be set using the brush options
- The “ESC” button can be used to switch between brush and camera mode

- To exit the selection mode, click on the selected symbol again

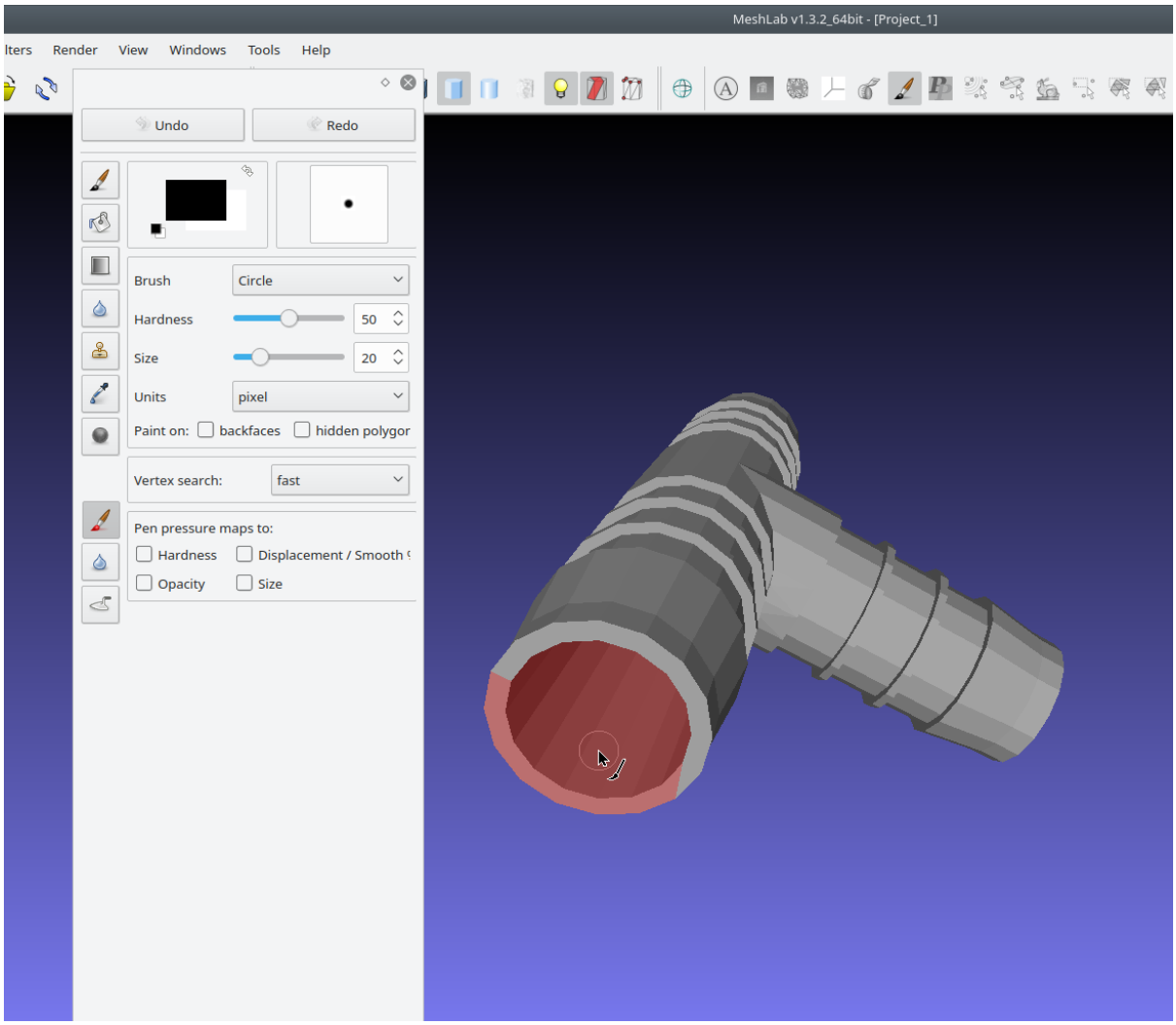


Fig. 16: MeshLab: Brush selection

3. **Remove areas**

   a) **Method 1:**
   Remove structures selected on the product (Remove selected areas)

   - Selektierte Dreiecke entfernen (“Delete the current set of selected faces and all the vertices surrounded by that faces.”)

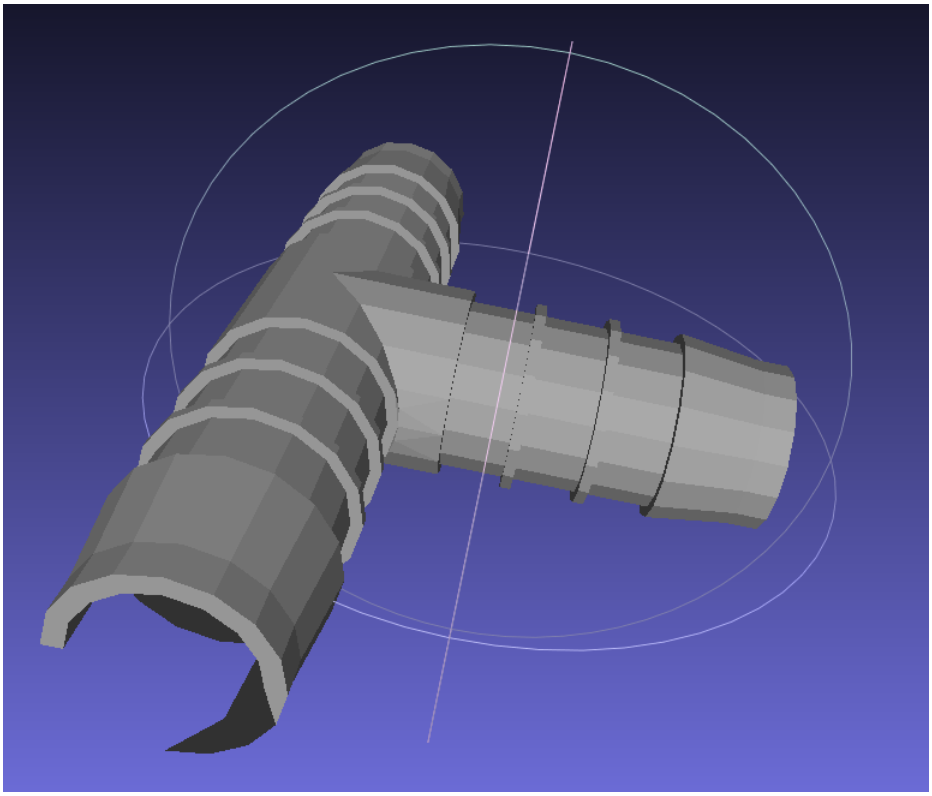


Fig. 17: MeshLab: Selected triangles removed

b) **Method 2:**
Select what is visible to the camera on the product (remove all areas that are not part of the selection)

- This method is used to select what is visible to the camera on the product (remove all areas that do not belong to the selection)
- Invert selection
  - Filters → Selection → Invert Selection
  - Apply

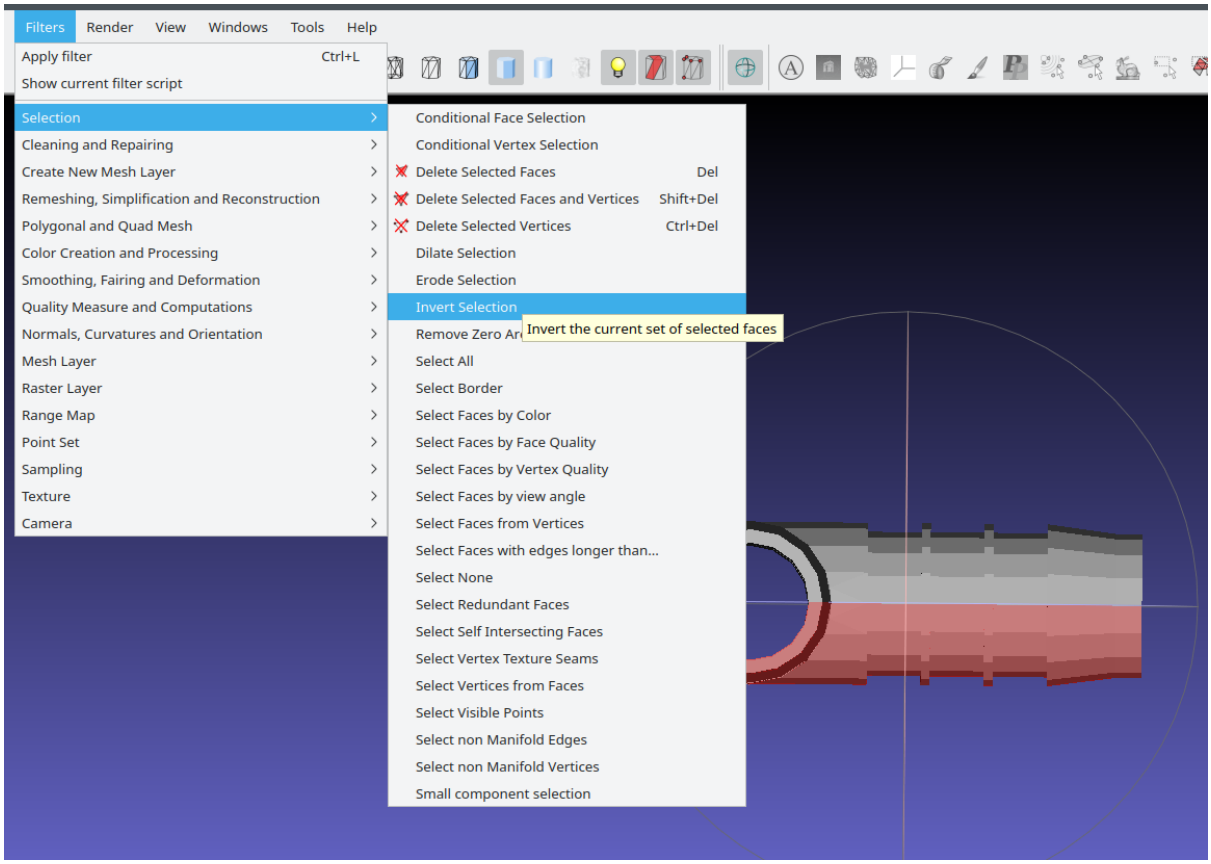


Fig. 18: MeshLab: Invert selection

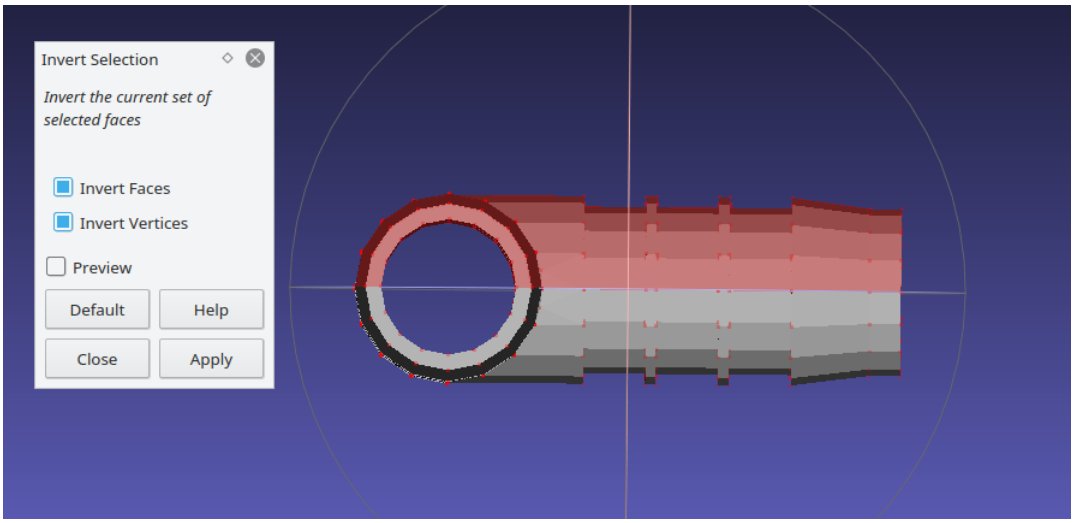


Fig. 19: MeshLab: Selection inverted

- Selektierte Dreiecke entfernen (“Delete the current set of selected faces and all the vertices surrounded by that faces.”)

4. **STL exportieren** (s. *Basics: Loading and saving STL*)

Hint:

- View the selection from all sides before removing it. Elongated or misshapen triangles often lead to unwanted selections.

- Also save intermediate versions.
- Under certain circumstances, it can be helpful to change the Field Of View (FOV). To do this, hold down the "Shift" key while scrolling with the mouse wheel.

4.3.6 STL Automatically remove internal surfaces

Section *Remove STL surfaces* describes how to select and remove surfaces / triangles of a product using manual tools. If you want to remove the internal and therefore invisible triangles of your product, you can also have MeshLab do this automatically. MeshLab offers algorithms that detect hidden triangles.

The following instructions explain the individual steps using the example product 'locking bolt'. The manufacturer's CAD data also includes an internal bolt, which is not visible from the outside. In addition, deep holes are also rarely part of the point cloud, which is why triangles can also be removed here. These surfaces are removed in the following steps using Meshlab.

1. **Load product** (see *Basics: Loading and saving STL*)

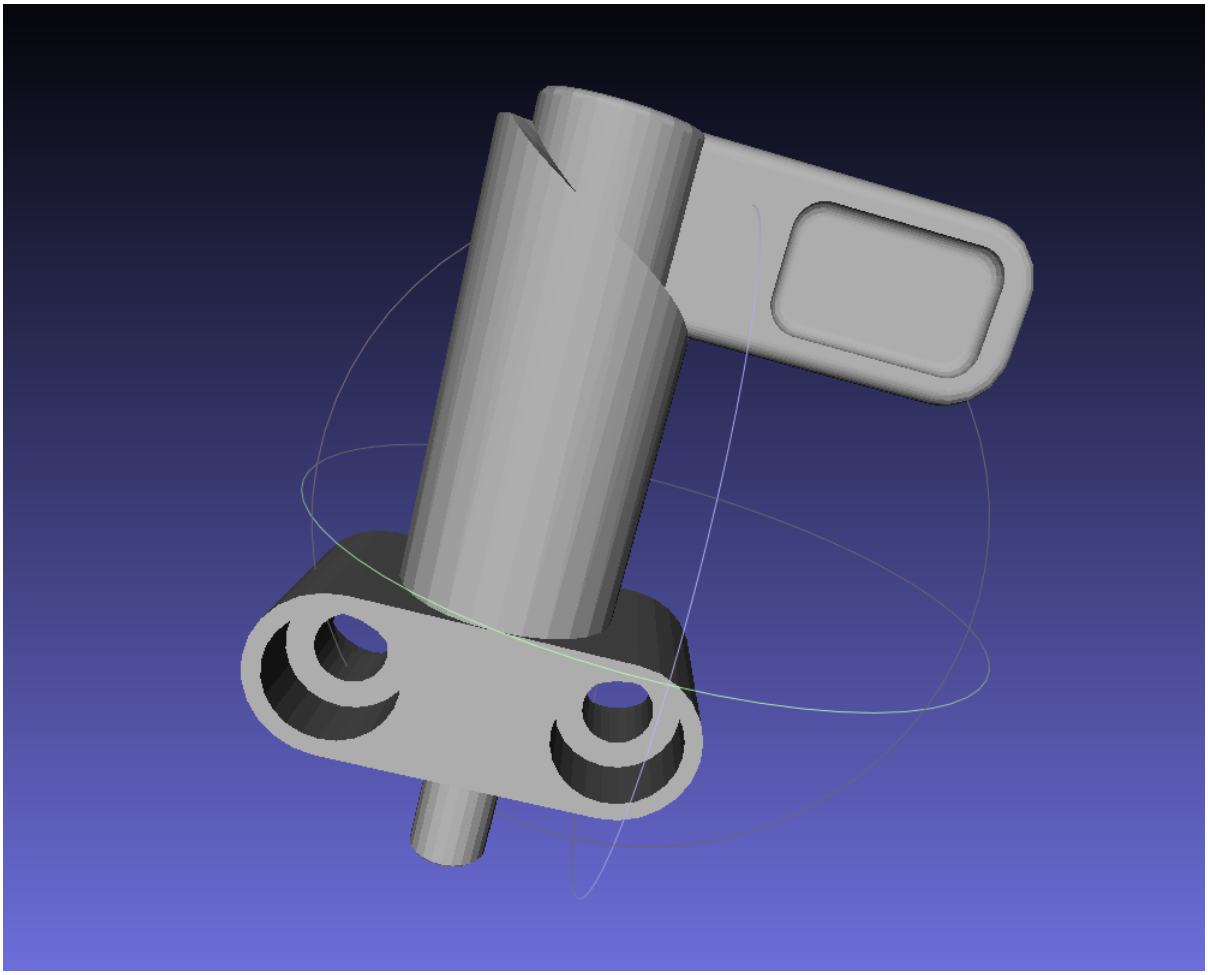


Fig. 20: MeshLab: An example product 'Locking bar CAD'

2. **Calculate ambient occlusion**

   In this step, a kind of "visibility rating" is calculated for each triangle. In the next step, this value serves as the basis for deciding whether the triangle can be removed or not.

   a. Open the dialog for the Ambient Occlusion Filter under

Filters → Color Creation and Processing → Ambient Occlusion Per Vertex

b. Activate ‘Use GPU acceleration’, then confirm with ‘Apply’ The calculation is started and a visibility rating is assigned to each vertex as a quality feature. The areas are now displayed brightly if visibility is good and darkly if visibility is low.

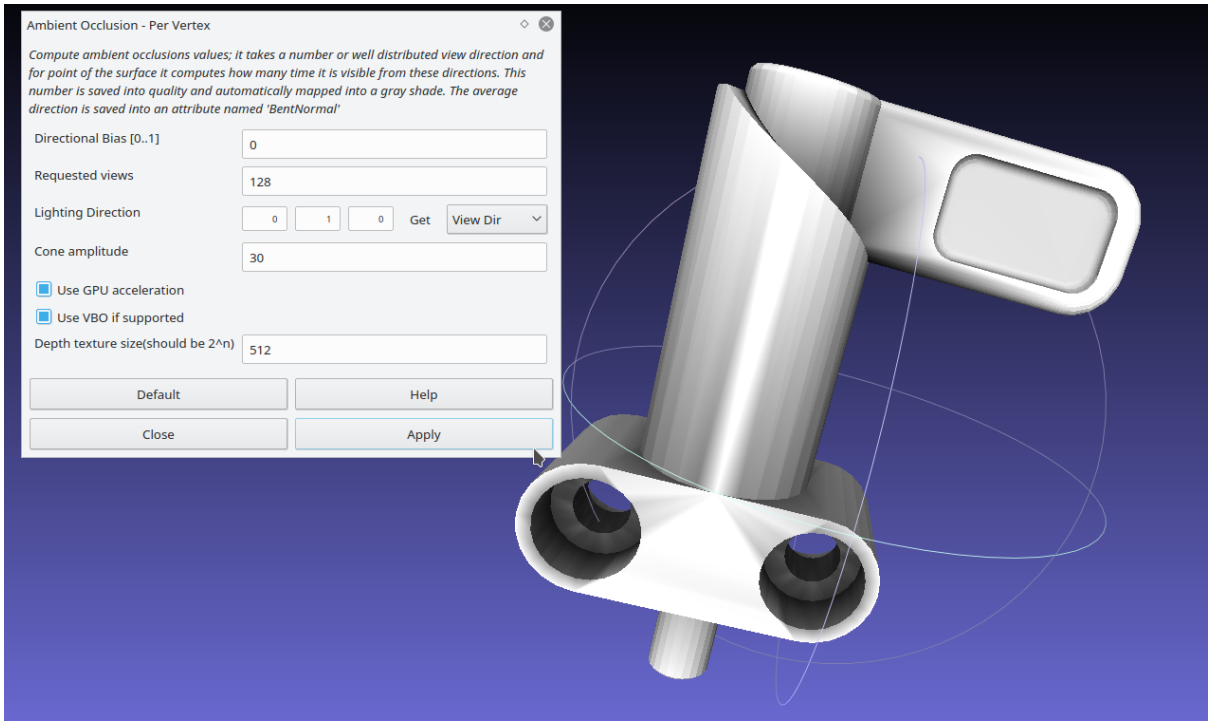


Fig. 21: MeshLab: “Visibility assessment” using the ambient occlusion filter

3. **Select areas based on visibility**

The results of the previous step can now be used as the basis for a selection/selection of triangles.

a. Filters → Selection → Select Faces by Vertex Quality

b. Set the ‘Min Quality’ parameter to the lowest value.

c. Activate “Preview”

d. Adjust ‘Max Quality’ until only the desired areas are selected (colored red)

e. ‘Apply’

Hint:

- It may not be possible to clearly separate partially covered areas from the areas to be removed. In this case, you can also use manual tools to subsequently adjust the selection (see *Remove STL surfaces*).
- Under ‘Render’ → ‘Render Mode’, you can switch between different types of visualization of the CAD model. The ‘Wireframe’ display, for example, also allows internal surfaces to be visualized as a triangular mesh.

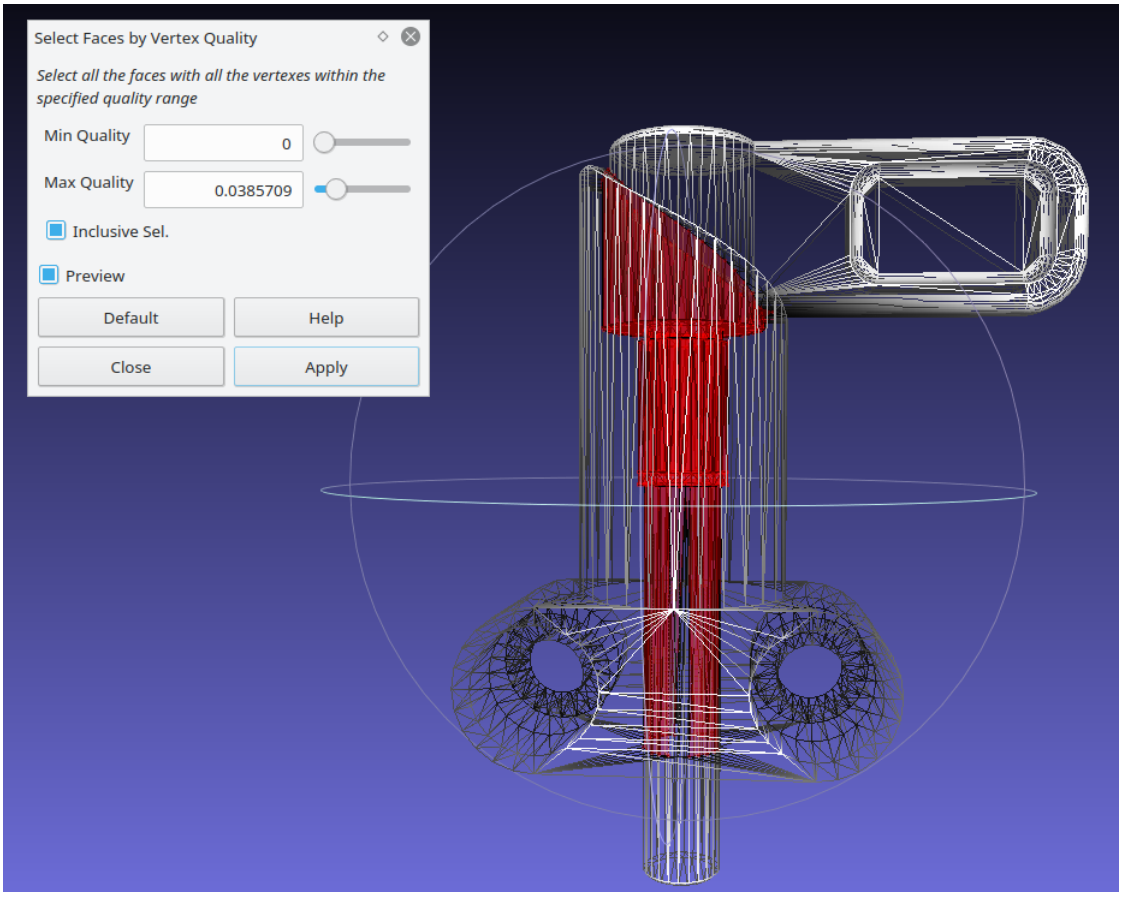


Fig. 22: MeshLab: Selection based on the visibility of an area with a low threshold value

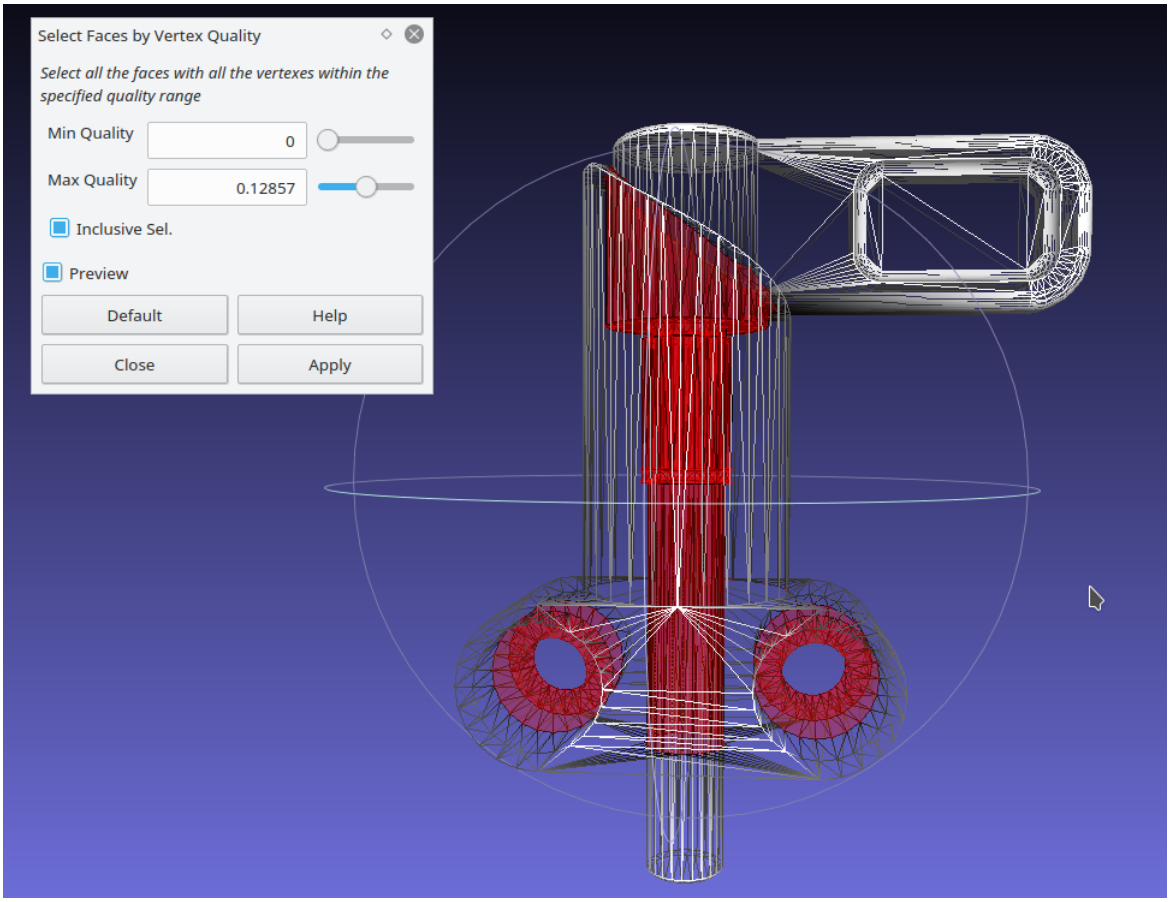


Fig. 23: MeshLab: Selection based on the visibility of a surface with a higher threshold value: further inner surfaces are selected. Selected surfaces are now also added in the lower area, as the structures are located in a depression.

4. **Remove areas**

   Selektierte Dreiecke entfernen (“Delete the current set of selected faces and all the vertices surrounded by that faces.”)

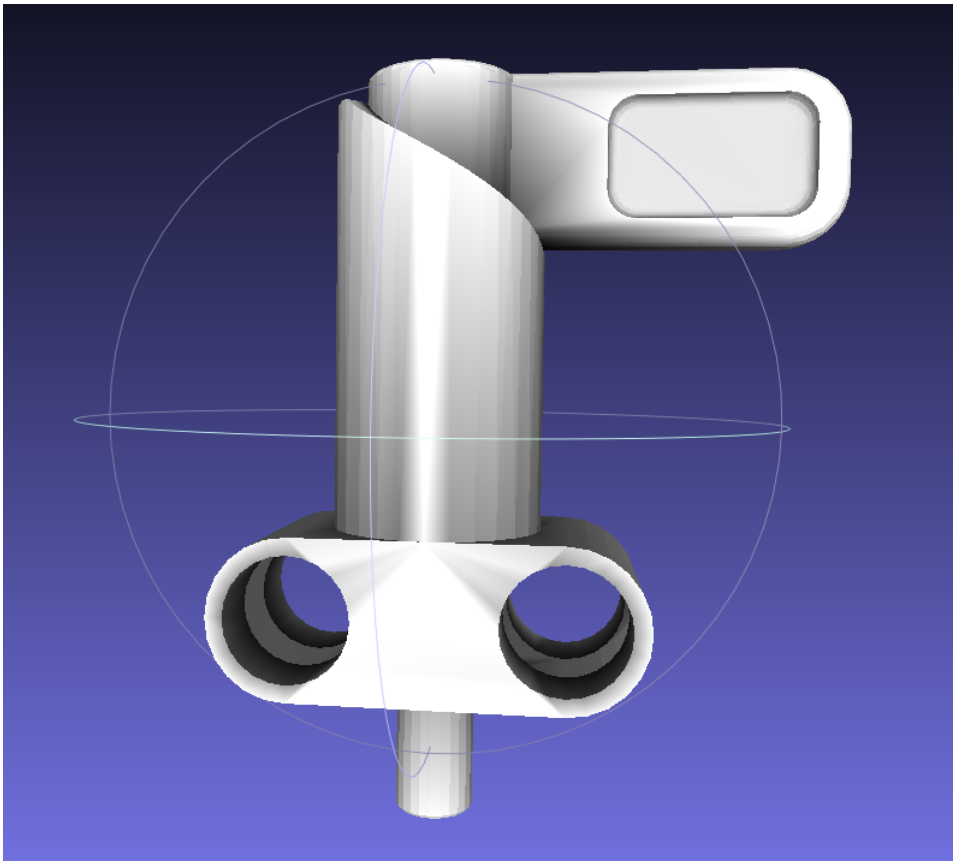


Fig. 24: MeshLab: The edited STL now has hardly any internal structures and can now be reworked manually or saved. As a high threshold value was selected in this example, the rings inside the lower holes have also been removed.

5. **STL exportieren** (s. *Basics: Loading and saving STL*)

4.4 Integrate Halcon Script: Data matrix recognition

The task *Execute Halcon Script* allows you to integrate your own Halcon scripts (Hdev scripts) into a Mikado| program sequence. The following tutorial uses an example to explain the steps required to implement your own recognition of a data matrix or a QR code using the Halcon library. The image data is read from one of the two mono cameras from the stereo image of the Ensenso camera.

In a practical application, such a function is helpful, for example, to trigger a product change via the barcode. The following barcodes were used:



Fig. 25: Datamatrix code for the article '12345678'



Fig. 26: Datamatrix code for the article '98765432'



Fig. 27: QR code Mikado (content is a link to the Mikado website)

4.4.1 Preparation

As a product change is to take place using the barcode, you first need a suitable product list. In this example, the products start with the prefix `Product_` followed by the article number, which is read in by the barcode. How to set up products is explained in the *Product management* chapter.

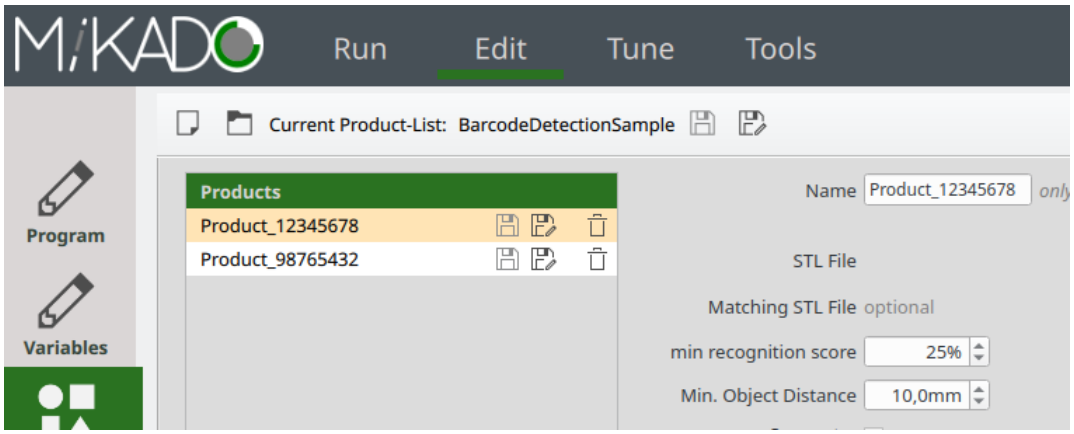


Fig. 28: Sample product list

You will also need a camera configuration for your Ensenso camera that is suitable for capturing a mono image (in particular, you should deactivate the projector and adjust the exposure time for good contrast). Finally, save the configuration file in the 'config' directory of your robot cell, e.g. under 'barcodeDetection.json'.

Hint: Explanations of the directories can be found under *Directory structure*.

4.4.2 Program schedule

In the first step, the program sequence is created as shown using the *program editor*.

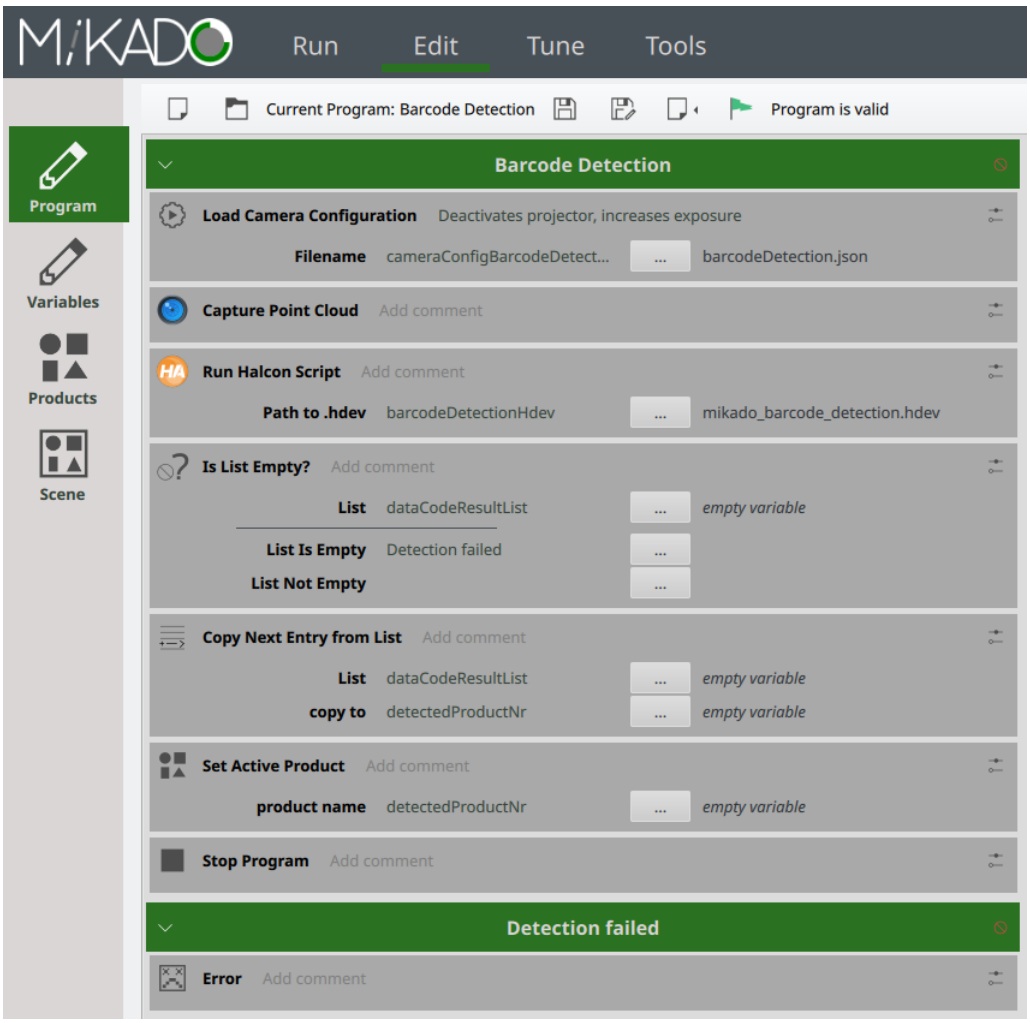


Fig. 29: Program editor: Program sequence for the integration of an Hdev script

- *Startsequenz: 'Barcode Detection'*
  Is executed once at program start and stops at the end of the sequence.
  - *Load camera configuration*
    Changes the configuration for the Ensenso camera. The configuration `barcodeDetection.json` is optimized for 2D image acquisition. In this case, the projector is deactivated and the exposure time is adjusted for good contrasts.
  - *Record point cloud*
    Triggers the camera for a recording. In the following steps, no 3D point cloud is evaluated, but this task also triggers the recordings of the mono cameras.

- *Execute Halcon Script*
  The actual barcode recognition takes place in this step. The HDev script `mikado_barcode_detection.hdev` is executed and can access image data from the Ensenso. The script itself is created in the next step.
- *Liste leather?*
  The HDev script for barcode recognition will access the Mikado variables and write recognized barcodes as StringList to the variable `dataCodeResultList`. If the list remains empty, no barcode has been recognized. In this case, the process jumps to the second sequence called `Detection failed` and ends here in the error state. If the list contains at least one entry, there is no sequence change and the process continues with the next task.
- *Copy next entry from list*
  Copies the first entry of the `dataCodeResultList` list into the `detectedProductNr` variable. This means that only the first recognized barcode is evaluated.
- *Set active product*
  Executes the product change. Note: If the product does not exist, an error is output at this point and the program execution is aborted.
- *Exit program*
  Stop program.

- *Sequenz: 'Detection failed"*
  Is executed if the barcode recognition does not find any barcodes.
  - *Error*
    Stops the program in the error state.

4.4.3 Create Hdev script for the recognition of data matrices / QR codes

The next step is to create the Halcon script, which is used to recognize the various barcodes.

Hint: All Mikado-specific halcon operators are explained in chapter *Halcon Script Task*.

To do this, open HDevelop and create a new HDev script. Save the file under the name `mikado_barcode_detection.hdev` in the `scripts` folder of the robot cell. At the beginning of the program, the rectified mono image of the Ensenso's right camera is read in and stored in the variable `Image2D` (Z. 4).

```
1 * Read right image from last capture of Ensenso camera and
2 * detect data matrix codes / QR codes in image.
3
4 mikado_get_2d_image(Image2D, 'Ensenso_Camera_0', 'right', 'rectified')
```

Then use the operator `find_data_code_2d` to search the image for data matrices (lines 5 - 7). The standard parameterization serves its purpose here, but can of course also be optimized through training. Information on this topic can be found in the Halcon documentation.

As the products in the example product list begin with the prefix `Product_`, the decoded article numbers are extended accordingly (lines 11 - 13)

```
6  create_data_code_2d_model('Data Matrix ECC 200', [], [], DataCodeHandle)
7  find_data_code_2d(Image2D, dataMatrixXLDs, DataCodeHandle, [], [], resultHandles,␣
   ↪decodedDataStrings)
8  clear_data_code_2d_model(DataCodeHandle)
9
10 * Prepend 'Product_' to article number (as this is our Mikado product name)
```

(continues on next page)

(continued from previous page)

```
for Index := 0 to |decodedDataStrings|-1 by 1
    decodedDataStrings[Index] := 'Product_' + decodedDataStrings[Index]
endfor
```

Since the script should also recognize and decode QR codes, we start the search again with the corresponding parameterization (lines 15 - 17).

```
create_data_code_2d_model('QR Code', [], [], DataCodeHandle)
find_data_code_2d(Image2D, qrXLDs, DataCodeHandle, [], [], resultHandles,␣
↪decodedQrStrings)
clear_data_code_2d_model(DataCodeHandle)
```

After the results are assembled (line 19), the decoded information is passed to Mikado as a list of strings (line 20). When creating the Mikado program flow in the step above, the variable 'dataCodeResultList' was created for this purpose.

```
decodedStrings := [decodedDataStrings, decodedQrStrings]
mikado_write_variable('dataCodeResultList', 'StringList', decodedStrings, [])
```

As the last part of the script, you can pass an image to Mikado that is to display results or analysis information, for example. In this case, you use a series of operators to draw the recognized outlines of the barcodes in colour in the image (from line 24). Data matrices are framed in green, QR codes in orange.

```
* Publish an image with results
get_image_size(Image2D, Width, Height)
open_window(0, 0, Width, Height, 0, 'buffer', '', WindowHandle)
disp_image(Image2D, WindowHandle)
set_line_width(WindowHandle, 5)
set_color(WindowHandle, 'green')
disp_xld(dataMatrixXLDs, WindowHandle)
set_color(WindowHandle, 'orange')
disp_xld(qrXLDs, WindowHandle)
dump_window_image(OutputImage, WindowHandle)
mikado_publish_2d_image(OutputImage)
```

The following screenshot shows the entire script as a screenshot from the HDevelop environment.

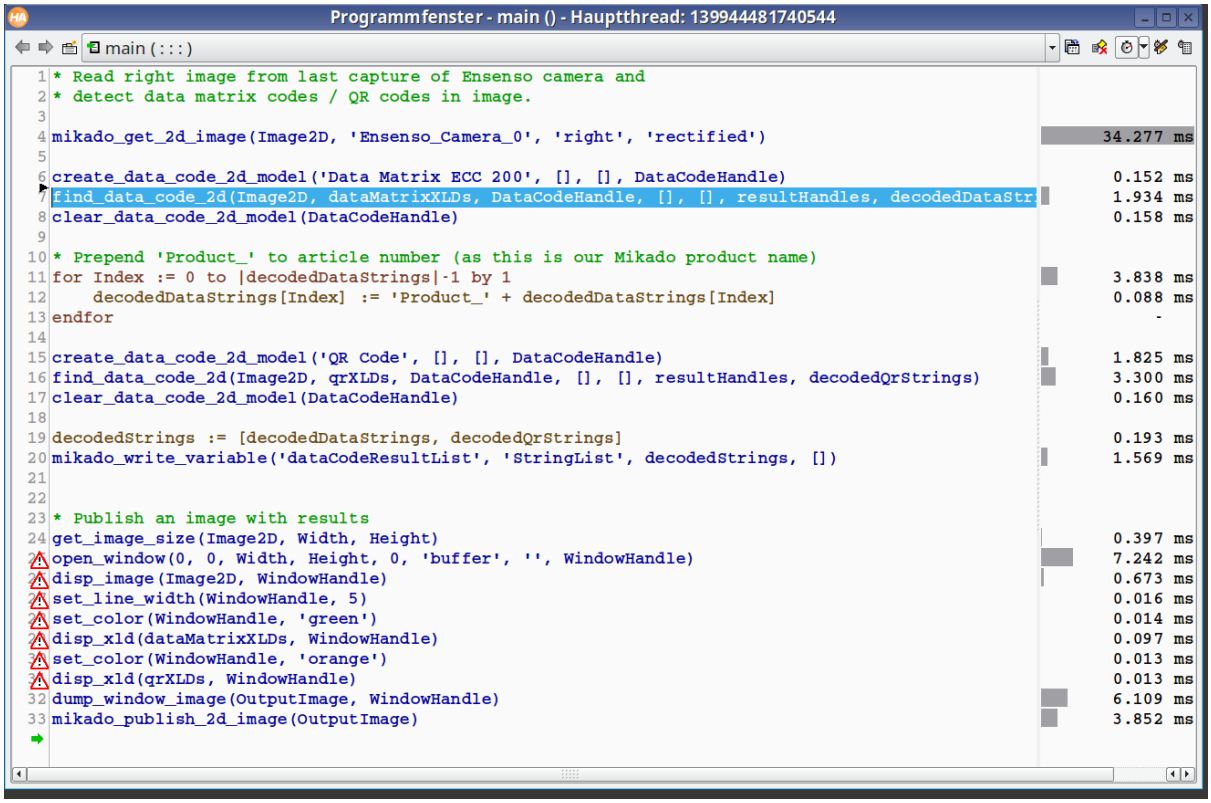


Fig. 30: Hdev-Script ‘mikado_barcode_detection.hdev’

Hint: Some Halcon operators at the end of the script are marked with a red warning symbol. The warning relates to the execution of the script within the HDevelop environment. However, as the script is later executed in a separate Hdev engine, these warnings can be ignored.

Tip: If you run HDevelop directly on the Mikado controller, you can also test the script directly in HDevelop. You also have the full support of the development environment for the Mikado operators. Simply start Mikado in parallel to HDevelop. Both the Mikado program flow and the individual Halcon operators can now be executed step by step. This helps you to develop and test the HDev script.

4.4.4 Execute and test program

The result can be seen in the following screenshots. If the process is executed while the camera is pointed at one of the two data matrices, the active product is successfully changed to the coded article number.

If you would like to display the corresponding image with the results drawn in, switch to the ‘Camera’ tab at the bottom right before starting the program. There, select the entry ‘/mikado_hdevengine_node/halcon_picture’ in the combo box at the top under ‘Show Image:’. Now, after executing your Hdev script, the captured image is displayed and decoded barcodes are framed in color.

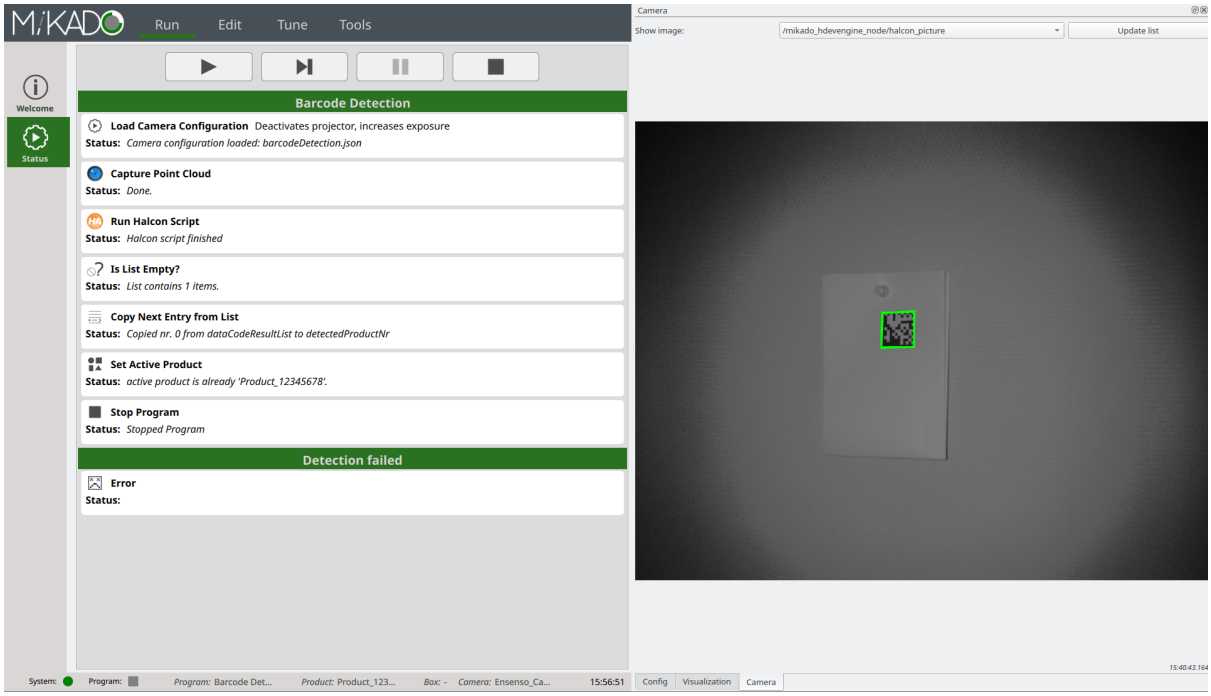


Fig. 31: Status view when using the data matrix of the article '12345678'

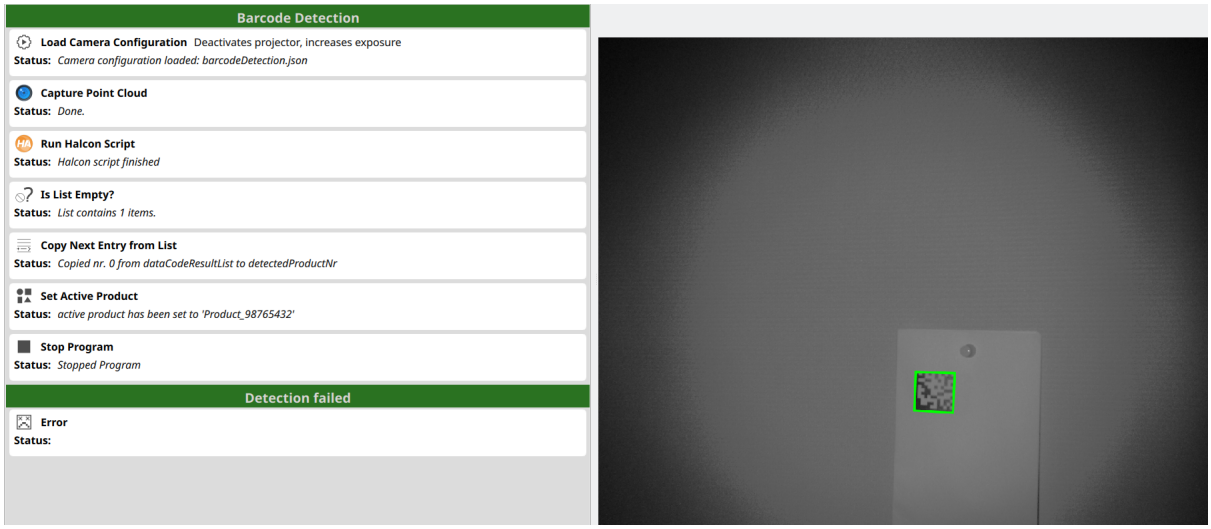


Fig. 32: Status view when using the data matrix of the article '98765432'

If the camera is pointed at the QR code, the decoded information does not contain an article number, but a URL to the Mikado website. As there is obviously no product with this name, the program sequence ends in an error as shown in the screenshot.

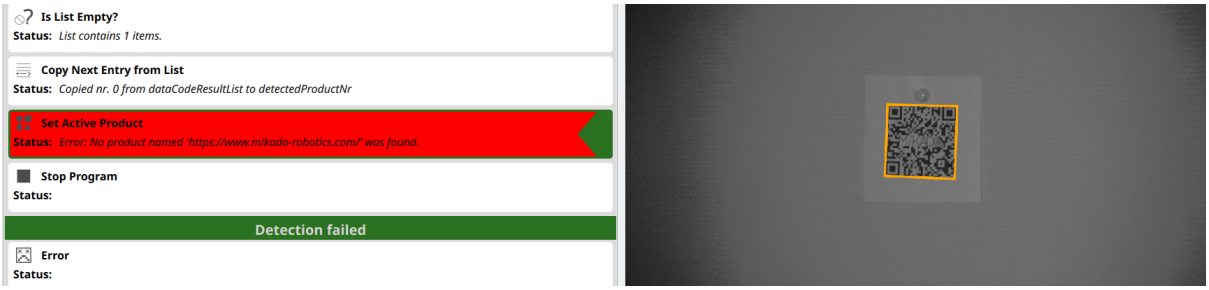


Fig. 33: Status view when using the Mikado QR code

4.5 Example: User-defined log file from Mikado

This tutorial shows how log files can be written in .csv format from the Mikado program flow. Use cases can be tests for statistical evaluation. The *JavaScript* task is used, whose interface is also described in the extension chapter *JavaScript Task*.

The entire script first logs a header line and then data lines. This is logged once in each program run and appended to a log file. The log file can then be imported using conventional table editors, for example. The individual script modules are explained below.

```
//Variablen aus Mikado laden
var testlauf = dataStore.getJsObject("nameTestlauf")
var ausgabeDateiFunde = dataStore.getJsObject("ausgabeOrdner")
var durchlaufN = dataStore.getJsObject("durchlaufN")
var fundeListe = dataStore.getJsObject("objektListe")

// Konstanten, .csv Spalten Seperator, Zielpfad
var sep = ";"
ausgabeDateiFunde = ausgabeDateiFunde + testlauf + ".csv"

//Logge in Mikado Konsole
console.log("JS Auswertung: Durchlauf " + durchlaufN + " schreibe Datei nach: " +␣
↪ausgabeDateiFunde)

//Überschrift, wird nur im ersten Durchlauf geschrieben
var ausgabeString = durchlaufN == 0 ? "Zeit;Durchlauf;Anzahl Funde" + "\n" : ""

//Zeitstempel über Mikadoschnittstelle "lab" bekommen
var timeString = lab.getCurrentDateTime()

//Auswertung
ausgabeString = ausgabeString + timeString + sep + durchlaufN + sep + fundeListe.
↪length + "\n"

//Ausgabe in Datei
var append = true //überschreibt Datei, falls false
lab.writeStringToFile(ausgabeDateiFunde, ausgabeString, append)

console.log("DEBUG JsScript Logging: \n" + ausgabeString)
```

4.5.1 Suitable Mikado program

The Mikado program contains a simple task sequence within a sequence:

- *Find objects*: (requires a fully configured product), for testing purposes *fundPosen* can also be a static pose list initialized with dummy values, in which case this task is omitted
- *Javascript*: Loads and executes the discussed script
- *Increase number*: Increase the variable *throughputN*
- *number greater than*: Comparison of the *throughputN* variable with the *maxN* variable, jump to the beginning if this has not yet been reached, i.e. in the *false* case
- *Exit program* Task

4.5.2 Variables of the Mikado program

Several variables must be created in the Mikado program on which the script is based:

- *nameTestrun* (String): Identifier of this run, also used as the name of the target file
- *outputFolder* (String): Absolute path to the output folder, ends with "/", e.g. "/home/mikado/"
- *runN* (integer): Counter of the logging iteration
- *maxN* (Integer): Maximum counter after which the logging iteration should be completed
- *fundPosen* (PoseList): List containing poses of found objects

4.5.3 Script details

The script is now explained step by step. Divided into initial actions, operation and final actions.

```
1  //Variablen aus Mikado laden
2  var testlauf = dataStore.getJsObject("nameTestlauf")
3  var ausgabeDateiFunde = dataStore.getJsObject("ausgabeOrdner")
4  var durchlaufN = dataStore.getJsObject("durchlaufN")
5  var fundeListe = dataStore.getJsObject("fundPosen")
6
7  // Konstanten, .csv Spalten Seperator, Zielpfad
8  var sep = ";"
9  ausgabeDateiFunde = ausgabeDateiFunde + testlauf + ".csv"
```

In this initial block, the contents of the micadovariables are read first. The output file path is also compiled from this.

```
12 //Logge in Mikado Konsole
13 console.log("JS Auswertung: Durchlauf " + durchlaufN + " schreibe Datei nach: " +␣
   ↪ausgabeDateiFunde)
14
15 //Überschrift, wird nur im ersten Durchlauf geschrieben
16 var ausgabeString = durchlaufN == 0 ? "Zeit;Durchlauf;Anzahl Funde" + "\n" : ""
17
18 //Zeitstempel über Mikadoschnittstelle "lab" bekommen
19 var timeString = lab.getCurrentDateTime()
20
21 //Auswertung
22 ausgabeString = ausgabeString + timeString + sep + durchlaufN + sep + fundeListe.
   ↪length + "\n"
```

The "calculation operations" are carried out in this block. These include logging in the Mikado logs in order to obtain feedback on the script execution. Then the string of the heading is assembled, using a tertiary operator via *runN* but only in the initial run. The current log line is filled under "Evaluation", the values are separated by separators. The final line break using "n" will write further content to the following line in the next run.

```
25 //Ausgabe in Datei
26 var append = true //überschreibt Datei, falls false
27 lab.writeStringToFile(ausgabeDateiFunde, ausgabeString, append)
```

Finally, the file is written to the file system. In an extended version, the *append* variable could also be set to *false* the first time if an initial overwriting of the log file is desired.

4.6 Example: Sort Mikado PoseList

This example shows how a Mikado variable of type *PoseList* can be sorted according to certain criteria. One possible use case is, for example, to sort the result of the *Find object* task in Mikado according to the z-coordinate in order to give priority to the parts at the top. In the example shown here, a list of poses is therefore sorted in descending order according to the z-coordinate. However, by modifying the example, any *PoseList* variables can also be sorted in a different way, e.g. ascending instead of descending or according to the x or y coordinate as well as the coordinate angles.

Attention: In the example shown, the original unsorted Mikado variable "ExamplePoseList" is overwritten with the sorted result. If the original variable is to be retained instead, a change is also necessary.

```
1  // Zu sortierende PoseList-Variable aus Mikado laden
2  var MikadoVariableToBeSorted = 'ExamplePoseList'
3  var PoseList = dataStore.getJsObject(MikadoVariableToBeSorted)
4
5  // Länge überprüfen, Listen mit weniger als 2 Einträgen können nicht sortiert werden
6  if (PoseList.length <= 1) {
7      task.status = "PoseList mit 0 oder 1 Einträgen kann nicht sortiert werden!"
8  }
9  // Sortieren und Ergebnis zurück in Mikado-Variable schreiben
10 else {
11     PoseList.sort((PoseA, PoseB) => PoseB.z - PoseA.z);
12     task.status = "PoseList \"" + MikadoVariableToBeSorted + "\" absteigend nach der␣
   ↪z-Koordinate sortiert"
13     dataStore.setJsObject(MikadoVariableToBeSorted, PoseList)
14 }
```

The JavaScript method Array *sort()* with a user-defined comparison function is used to sort the pose list. The behavior of the sorting can be changed accordingly by modifying the comparison function in line 11. The individual components of the poses can be accessed using the keywords `x`, `y`, `z`, `roll`, `pitch` and `yaw`. The following are examples of some alternative sorting options:

- Sort by z-coordinate ascending

```
PoseList.sort((PoseA,PoseB) => PoseA.z - PoseB.z);
```

- Sort by x-coordinate descending

```
PoseList.sort((PoseA,PoseB) => PoseB.x - PoseA.x);
```

- Sort by yaw angle descending

```
PoseList.sort((PoseA,PoseB) => PoseB.yaw - PoseA.yaw);
```

CHAPTER

FIVE

SIMULATION

Almost all sub-components of Mikado can be set to simulation mode either individually or all at the same time. For example, it is possible to use recorded images (Filecam) to optimize a process with a simulated robot in the office. The integrated physics simulation enables containers to be filled with parts to be handled on the basis of the mesh models. This means that there is no need to obtain camera images beforehand. This is ideal, for example, for evaluating different gripper variants without having the grippers manufactured in advance.

Hint: Only very limited conclusions can be drawn from the results of the simulation about the cycle times that can be achieved in reality. The required acceleration values and reaction times of the robot controller are generally not accessible and also depend on the specific configuration. If you have a license (or test version) of the simulation software of your robot manufacturer, you can of course also use Mikado with it, please contact us for details.

5.1 Roboter-Simulation

In the configuration file 'config/manipulator.yaml', the variable 'simulated_robot: true' can be used to switch to simulation mode. Mikado will now no longer make contact with the real robot, but will follow the trajectories offline.

5.2 Recorded image data (Filecam)

You can save image data manually in the application Mikado, automatically from a program sequence, or also in NxView in order to load it again later as Filecam. To view such image data, start NxView, "Create…" → "Filecam". Select the folder with the saved images and make a note of the name of the camera, this is automatically "File[serial number]". Open your cell in the Mikado Assistant and enter the new name under "Parameters" → "Cameras". Alternatively, you can also enter the line "camera_serial_no: File[serial number]" in the "config/camera_0.yaml" file. The Filecam is now used instead of the real camera. The images are rotated with each shot. If the images were saved directly by Mikado, the point cloud will also be displayed at the correct position in the room. If the Filecam was saved by NxView, this only works with calibrated stationary cameras.

Hint: Preferably use the "Save last image" button under "Tune" → "Detection" or the *"Save image" task* in the program sequence. This is the only way to save the current position of cameras mounted on the robot. If you nevertheless create filecams in NxView, please ensure that you do not save them as a ZIP file.

5.3 Simulation based on mesh files

This chapter describes the add-on program *Mikado Simulator*, which you can use to provide synthetic camera images as input to Mikado. For this purpose, a box is filled with mesh files of the product in a physics simulation and then images are captured with a so-called "virtual camera". The stereo matching and object recognition function in the same way as real image data.

Hint: In general, the simulator must be started after Mikado. You will find a separate link for this on the desktop or in the cell directory.

5.3.1 Simulator - Configuration

This page can be accessed via the main menu 'Setup', submenu 'General'.

Configure the simulator and add and manipulate individual objects.

General setup

This item allows you to restore a previous simulator state. The following settings are loaded:

- Crate objects with drop zone
- Product objects with individual product properties

The following parameters can be set permanently in the respective cell folder in launch/simulator.launch:

- Camera serial number and features
- Fixed or movable camera
- Default product mesh file
- Automatic mode: Default parameter
- Automatic mode: Default for "Start immediately after simulator start"

Camera Setup

This sub-item shows the currently set camera serial number and whether it is fixed or movable. The value for the noise level can be selected from 0 to a maximum of 30 and is applied after confirmation. This setting can no longer be adjusted during automatic mode.

Box setup

So far it is possible to work with exactly one box.

This sub-item allows you to adjust the parameters of the active crate. On the one hand, this means the pose consisting of xyz coordinates and rpy rotation angles. Secondly, the dimension and height of the drop zone can be defined for each crate. Its orientation is linked to the orientation of the crate.

Product setup

This sub-item allows you to select the product that is to be simulated and dropped over the box. To define the product, a mesh file must be specified and whether it has 180° rotational symmetry around the x-axis. The latter also allows detections rotated by 180° to be classified as correct in the evaluation. The 'Reflectance' property can be used as a material property to simulate how strongly the object should reflect the camera's projector light. To add parts, a corresponding number must be selected. With 'Add', the parts are then placed randomly in the drop zone in terms of both position and rotation. A uniform distribution is used as the probability distribution. By default, 20 parts are simulated at once. If more are added, the simulation is carried out in several tranches.

Hint: The physics simulation takes place over a set period of time. The 'simulate_total_duration' parameter in the cell definition under launch/simulator.launch can be adjusted for high or very low launch heights.

Hint: The number of simulated parts should be selected so that initially no or only a few collisions occur during random positioning in the drop zone.

Hint: Convex parts can sometimes lead to collisions between objects and the crate in the physics simulation.

Objects

This status display shows all objects currently in the system. Recognized objects have a bold name, successfully picked objects have a crossed-out name. Clicking on an individual object highlights it in color in the visualization and shows its details, such as position, orientation, symmetry and reflective properties. The pose of the object can be adjusted here using the input fields and applied by clicking on 'Apply'. Such changes to the pose are not checked with the physics simulation. When manually adjusting an object position, make sure that there is no overlap between objects, containers or scene objects in the created scenario.

5.3.2 Simulator - Automatic mode

You can access this page via the main menu 'Run', submenu 'Automatic'.

The simulator can be used as an endurance tester in automatic mode. Several simulated fillings of the crate are processed by Mikado and statistics are generated.

Procedure

Several test runs, so-called *fills*, are carried out. In each run, a box is filled with the specified number of parts and Mikado is reset and restarted. The following parameters must be set for this:

- ***Case fillings*:**
  The number of runs for which the box is refilled.
- ***Objects per filling*:**
  The number of objects that are dropped into the crate per filling.
- ***Allowed angle error*:**
  Each object detection by Mikado is only accepted if the absolute angle error is below this limit. Objects that are not accepted cannot be grasped and are considered a false grasp if Mikado attempts to grasp them.
- ***Permitted distance error*:**
  Like *Permitted angle error*, only this parameter describes the maximum permitted absolute error distance by which the object detection may be off.

- ***Permitted misuse in sequence*:**
  How many objects may not be successfully grabbed in succession until this run is aborted and recorded as failed.
- ***Allowed remaining quantity of objects*:**
  If Mikado stops with an error, it checks whether the maximum permitted remaining quantity of parts in the crate has not been reached. If so, the run is ended and recorded as a success.
- ***Pause on Mikado error*:**
  If active, the simulator interrupts continuous operation if an error state is reached in Mikado before the desired degree of emptying is reached. Continuous operation can be resumed by clicking on the 'Resume' icon. This option allows problematic situations in Mikado to be analyzed in more detail. The object positions during the interruption can also be saved so that they can be reproduced later.

Evaluation

The simulator displays both the statistics for the current filling to be emptied and for the entirety of the current test. The overall statistics always refer to all completed runs.

Individual values are described in more detail below:

- ***Degree of emptying*:**
  At the end (successful/abort on error) of each run, the proportion of removed parts to the total number of parts per fill is saved. The *degree of emptying* corresponds to the average of all runs.
- ***Average time per handle*:**
  Average time of the ratio of the total time of a run to the picks made by the robot during the run. This value is independent of the success of the individual picks.
- ***Average time per withdrawal*:**
  Like … *per grip*, but only successful grips apply here. This value therefore corresponds to a combination of gripping/recognition reliability and average cycle time.

5.3.3 Simulator - Adjusting the camera settings with scene from Simulator

If there is no robot and no camera yet and you want to create a simulation, it is often necessary to adjust the camera settings. The following section shows the steps for creating the new camera settings.

Procedure

1. cell creation: A simulated cell is used as an example, in which a box is filled with crankshafts.

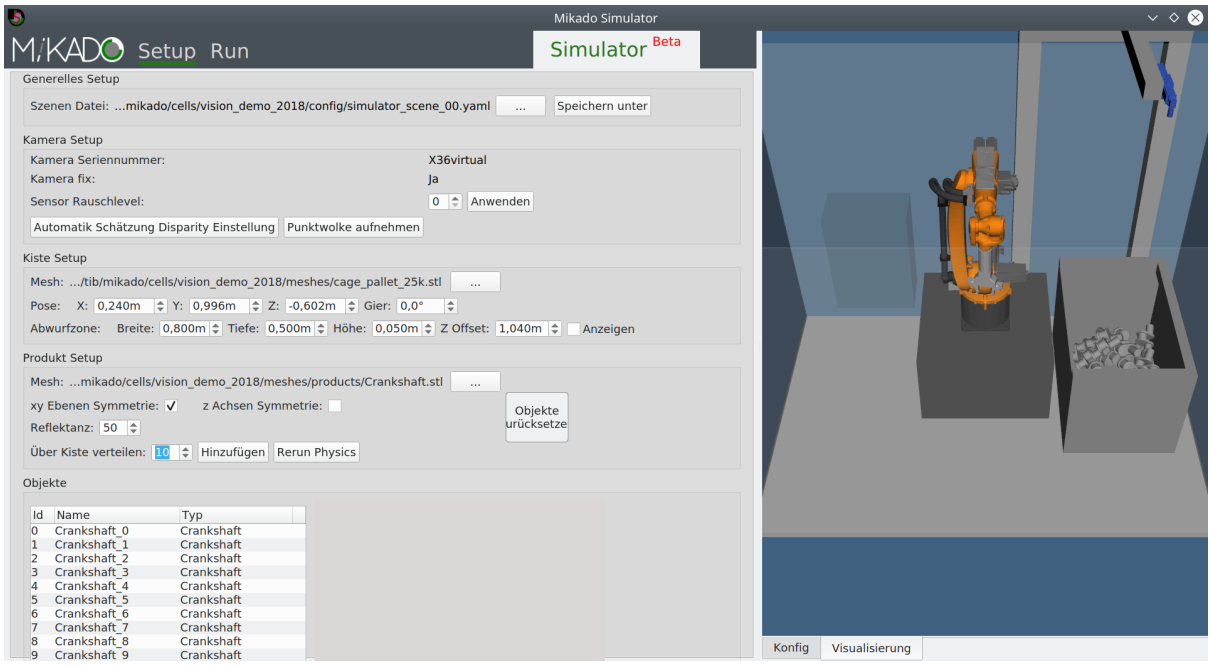


Fig. 1: Screenshot from the Mikado Simulator

Open NxTree: The scene from the simulator can be opened in NxTree. To do this, Mikado and the simulator must remain open. After starting NxTree, the following window opens. The selection should be made as in the following window.

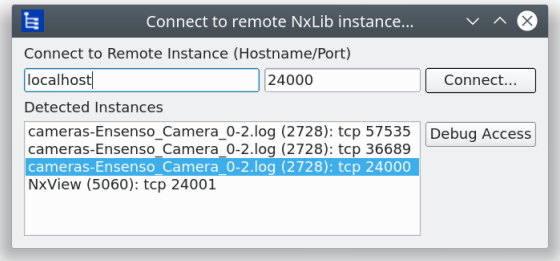


Fig. 2: NxTree startup window

3. open NxView: The camera to be set must then be opened in NxView.

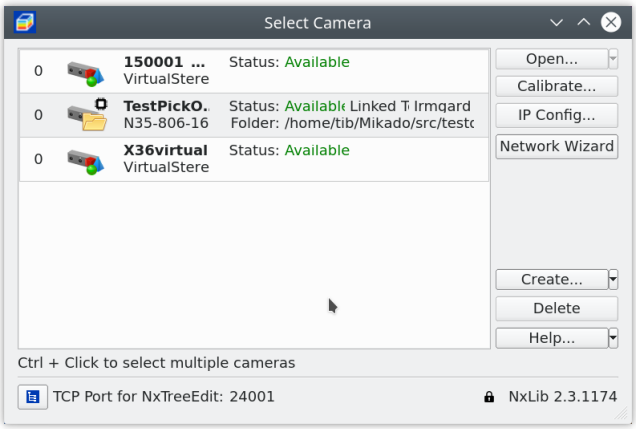


Fig. 3: Selection of the camera to be set in NxView

4. open NxTree in NxView: An exemplary scene is displayed in NxView. To transfer the simulator scene, NxTree must be started in NxView. This step can be seen in the following illustration.

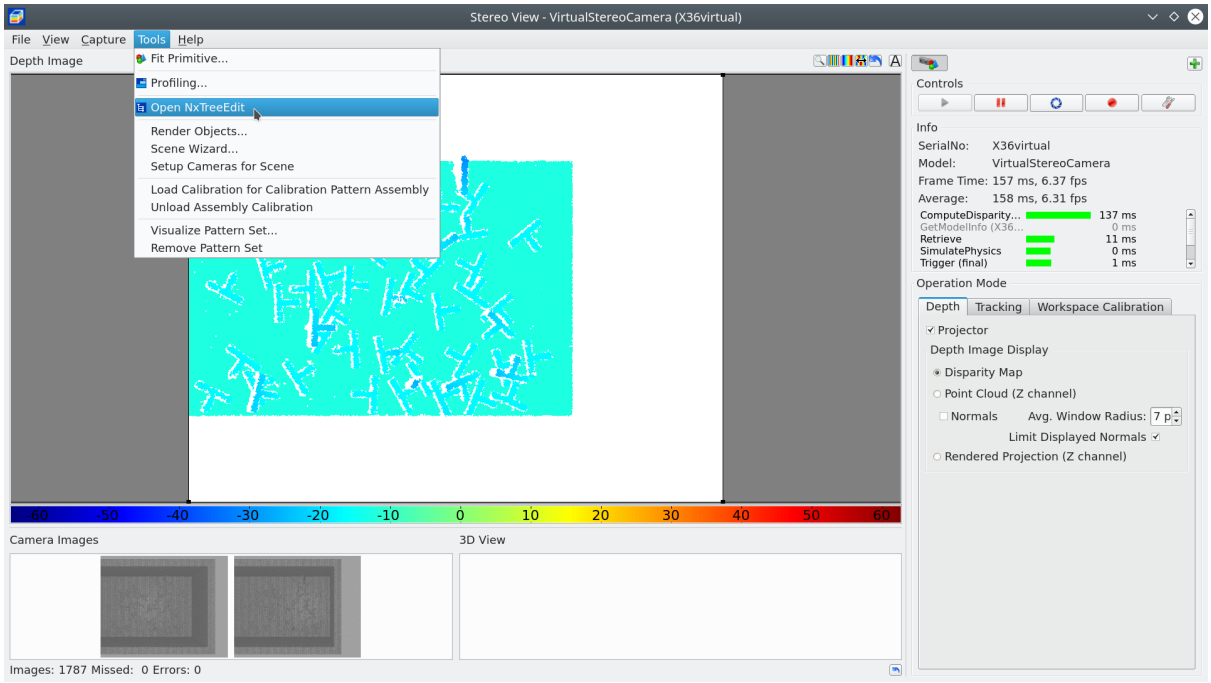
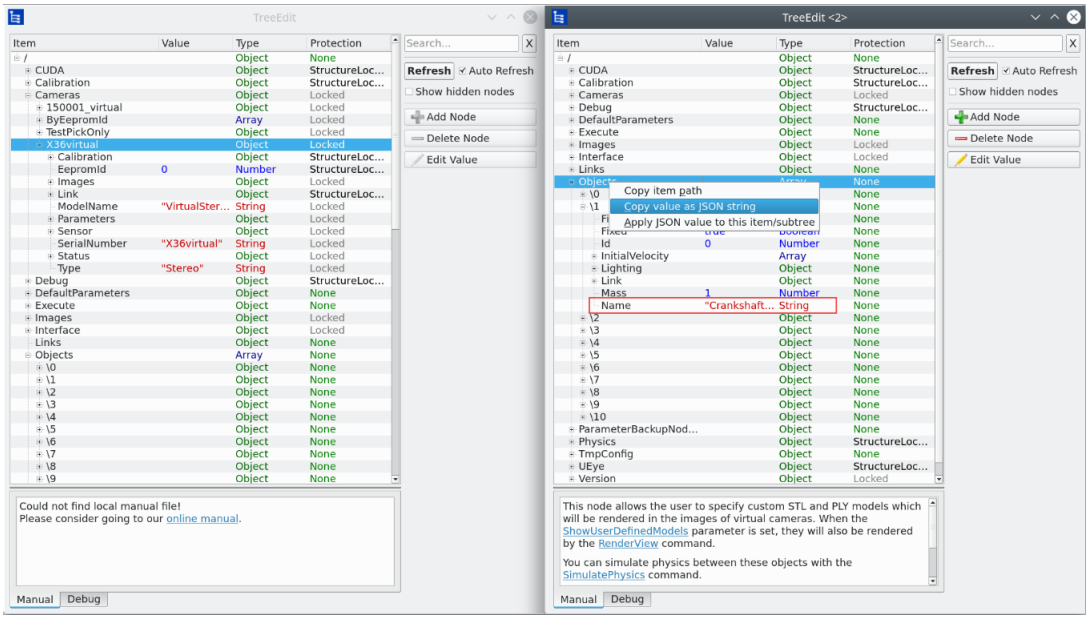
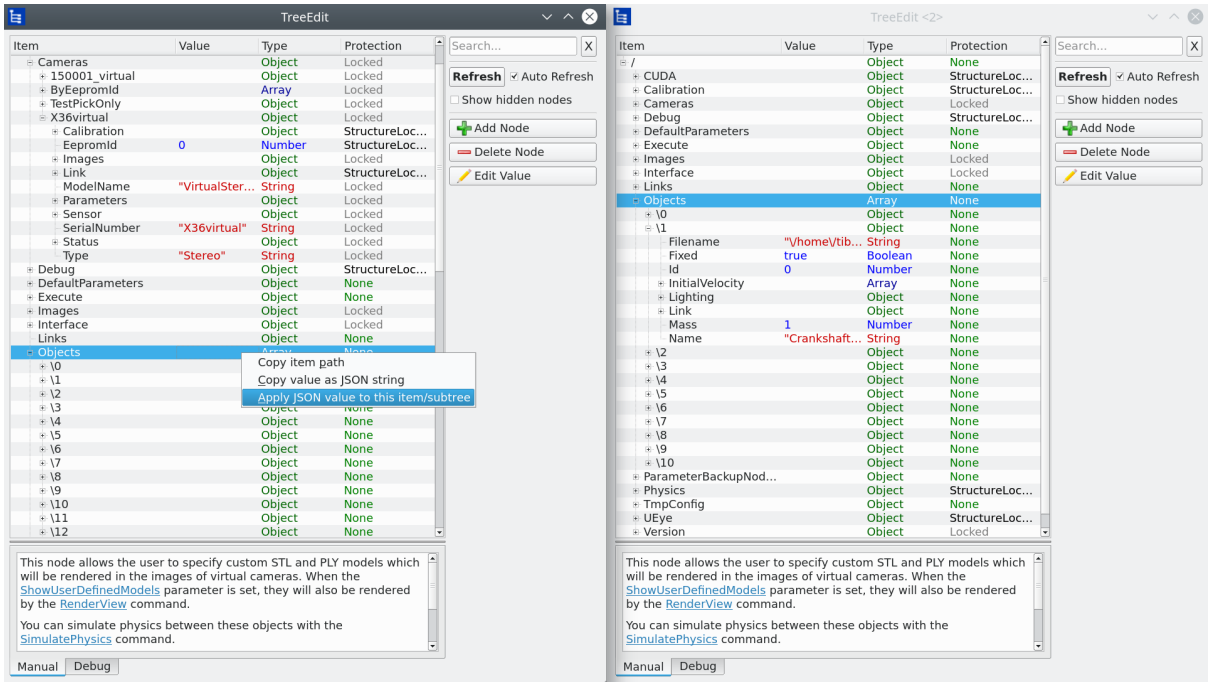


Fig. 4: Screenshot NxView

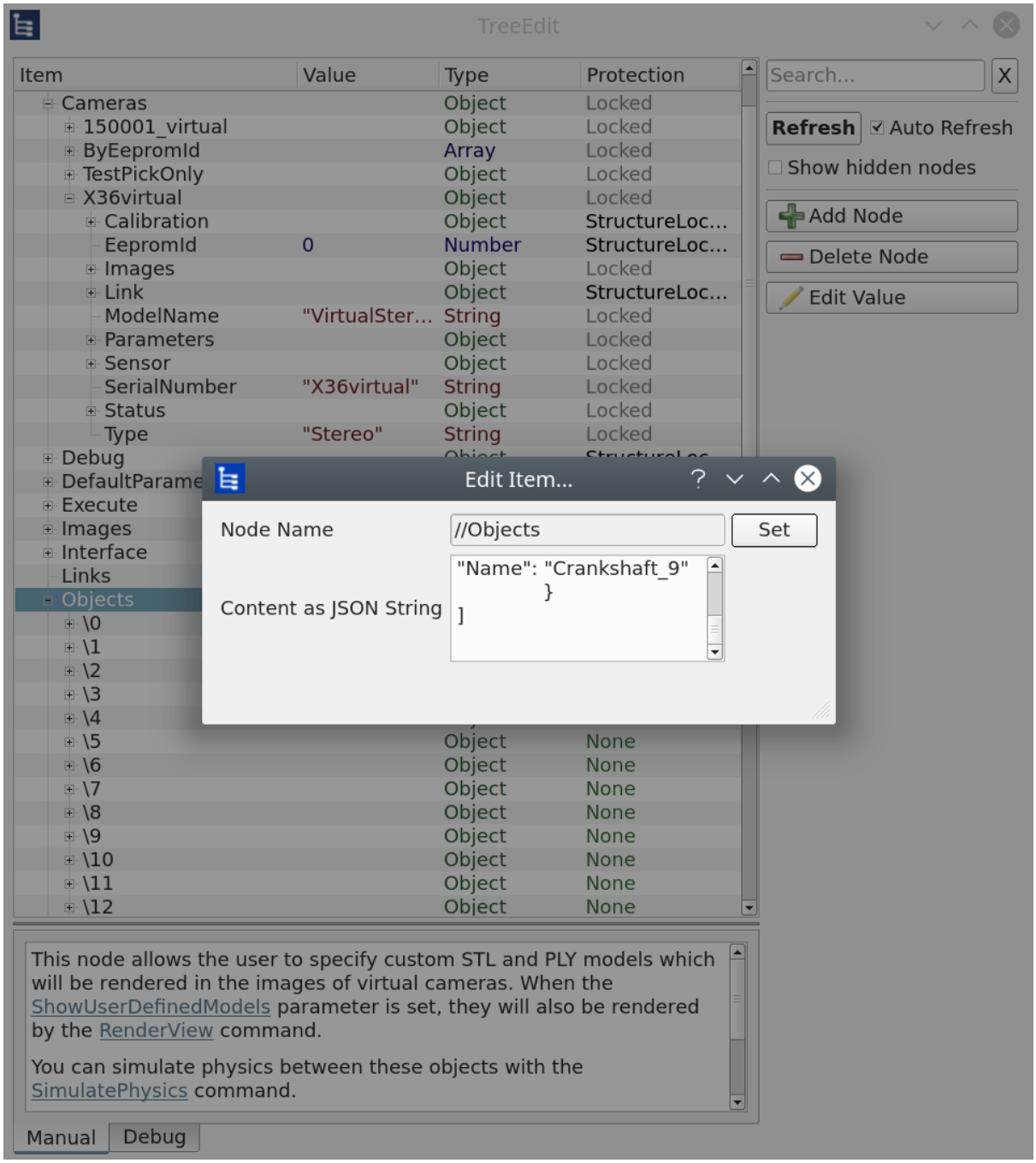
Copying the elements: The following illustrations show NxTree of the scene in NxView (left) and the scene in the simulator (right). If you lose track of which window belongs to which scene, you can check the objects, for example. In the simulator scene (right), crankshafts have been thrown into the crate. These can be found in the right-hand window under Objects (see red rectangle). This window therefore belongs to the simulator. To transfer these objects, they must be copied and pasted as shown in the following illustrations. The command **Copy value as JSON string** appears after **Objects** has been clicked with the right mouse button.



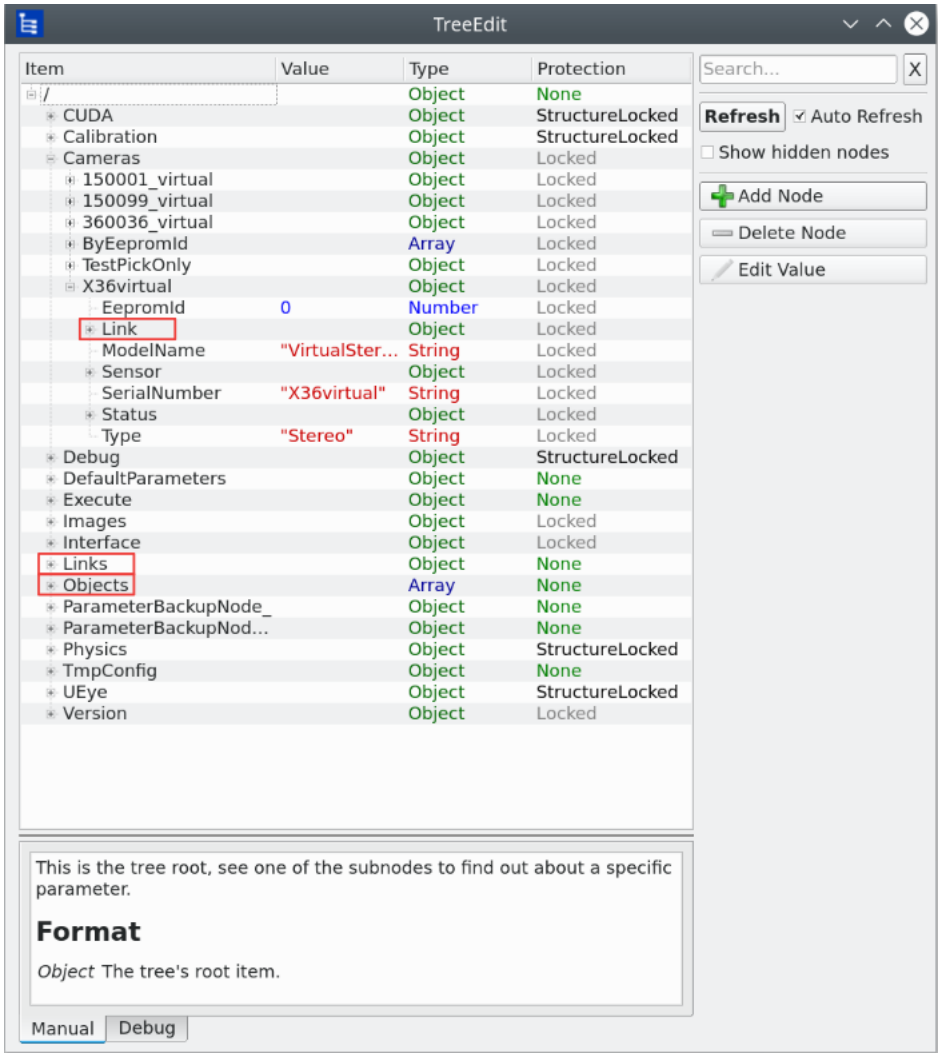
The objects must then be inserted into the scene associated with NxView (left-hand window). To do this, right-click on **Objects** and select **Apply JSON value to this item/subtree**.



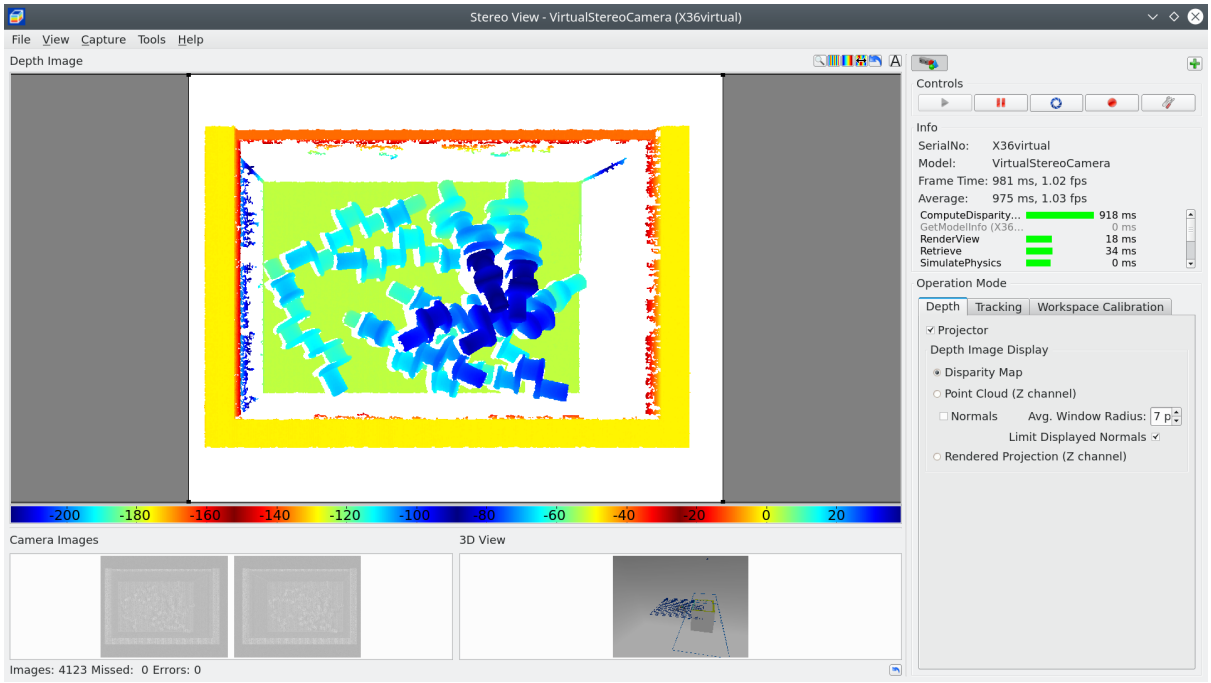
In the window that opens, the **Content as JSON String** field must be filled. To do this, click on the field with the right mouse button and then on **Insert**. You will then see the content in the field.



This procedure for **Objects** must be carried out analogously for **Links** and the **Link** of the corresponding camera (see red rectangles).



6. adjusting the camera settings: When closing NxTree, the scene from the simulator is now visible. The camera parameters can now be adjusted so that the crankshafts are recognized as well as possible.


Stereo View - VirtualStereoCamera (X36virtual)
File View Capture Tools Help
Depth Image
Controls
Info
SerialNo: X36virtual
Model: VirtualStereoCamera
Frame Time: 981 ms, 1.02 fps
Average: 975 ms, 1.03 fps
ComputeDisparity... 918 ms
GetModelInfo (X36... 0 ms
RenderView 18 ms
Retrieve 34 ms
SimulatePhysics 0 ms
Operation Mode
Depth Tracking Workspace Calibration
Projector
Depth Image Display
Disparity Map
Point Cloud (Z channel)
Normals
Avg. Window Radius: 7 p
Limit Displayed Normals
Rendered Projection (Z channel)
-200 -180 -160 -140 -120 -100 -80 -60 -40 -20 0 20
Camera Images
3D View
Images: 4123 Missed: 0 Errors: 0

CHAPTER

SIX

ROBOT MODELS

Mikado allows the use of robots from many different manufacturers. This section lists how to configure the robot controllers of the individual manufacturers so that they can communicate with Mikado without any problems.

Robots from different manufacturers differ in their features and additional functions. The respective sections explain the options offered by the respective robot models and manufacturers in Mikado.

Parameters or control elements that are model-dependent and are displayed via the interface are integrated on the *Robot control* page.

6.1 Set up ABB Controller

- *System requirements*
- *Network configuration*
- *Installing the programs*
- *Setting up the programs*
- *Updating the robot programs*
- *Robot Control*
- *Configuring gripper and IO with Robot Web Services*
- *Giving up control over the robot temporarily*
- *Troubleshooting*

6.1.1 System requirements

The robot-side part of the ABB connection was written in RAPID and uses both parallel tasks and sockets. The following options are required:

- 623-1: Multitasking
- 616-1: PC Interface

At least **RobotWare** OS version 5.13 is required.

RobotStudio is required for this setup. It is available in a free Basic version and is used in the following instructions.

Robot Web Services (RWS) is required for the direct control of grippers and IOs.

RWS 1.0 is included in RobotWare OS 6.x.

RWS 2.0 is installed in RobotWare OS 7.0 and above. Mikado systems older than 3.3 might require an update to support IO via RWS 2.0.

Alternatively, grippers and IOs can also be controlled via Profinet or other fieldbusses.

6.1.2 Network configuration

To enable communication between the mikado controller and the robot controller, a network connection is required. To achieve this, connect the ABB Controller and the Mikado Controller using a network cable (ethernet).

Hint: On the ABB controller, it is strongly recommended to use the WAN port. If your controller has multiple WAN ports, use WAN1.

A network connection between the two systems is required for communication between the robot controller and Mikado controller. To do this, first connect an Ethernet port of the ABB controller to an Ethernet port of the Mikado controller. A fixed IP address and a corresponding subnet mask must then be set for both the Mikado Controller and the robot controller. The network configuration of the Mikado Controller is explained in more detail in section *Network adapter*. On the ABB controller, you can configure the IP address of the public network by going to `Settings` → `Network` → `Public Network` (default is 192.168.3.2). For more details on how to configure the network on the robot please check out the manufacturer's user manual.

Hint: Make sure that the IP addresses of the Mikado Controller and ABB Controller are in the same address space and that the subnet masks match.

Then enter the IP address of the robot controller in the **Mikado Assistant** under `Robot` → `Connection` in the corresponding field and save the settings.

To check the network connection see *Connection issues*.

6.1.3 Installing the programs

In order for the ABB robot to be controlled by Mikado, the corresponding programs must be copied to the robot controller and set up there. You can find the required RAPID programs at

`/opt/isys-vision/mikado/robot_programs/abb/rapid`

or using the corresponding button in the Mikado Assistant:

`Robot` → `Set up ABB Controller` → `Open folder`

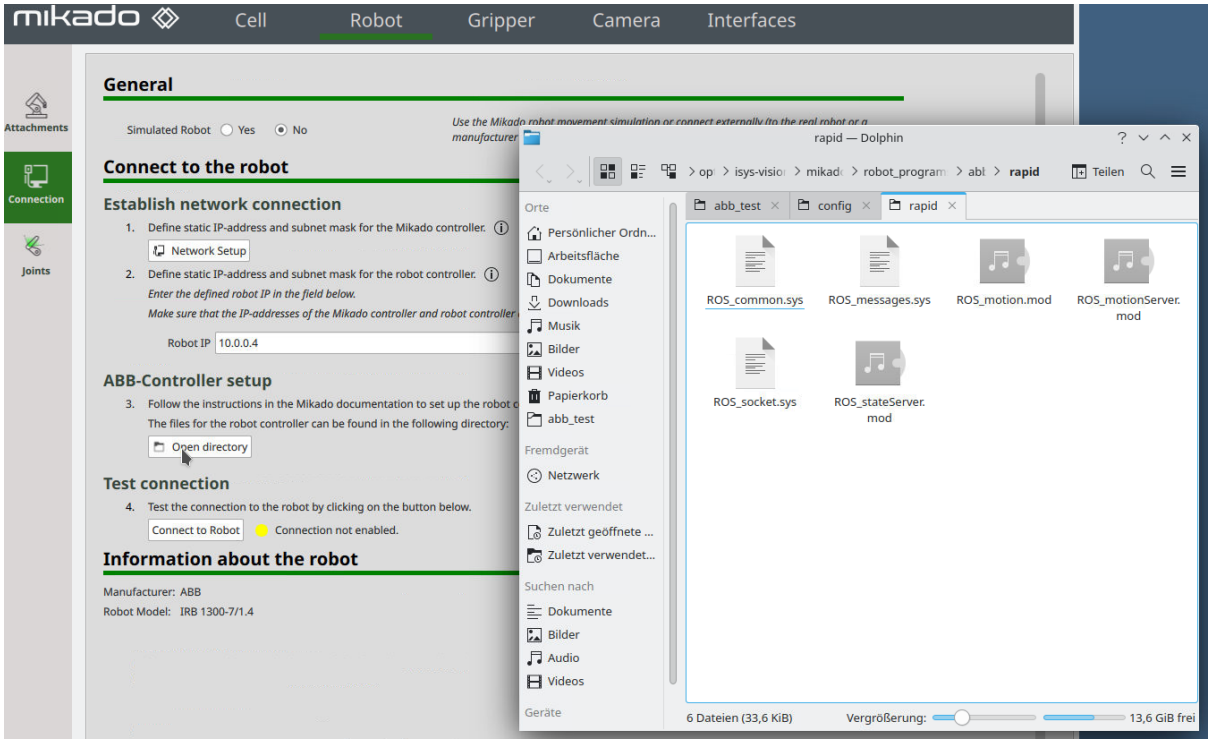


Fig. 1: Mikado Rapid files on the Mikado controller

Copy the files to a USB drive, plug it into the controller or flex pendant and copy the files onto the robot controller, into a folder named `ROS`, e.g. under `HOME`.

Then copy all files from the usb drive there.

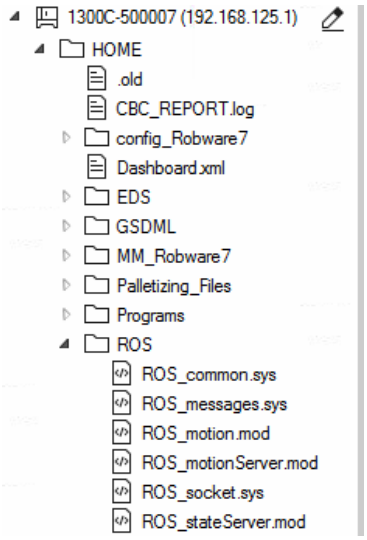


Fig. 2: Mikado Rapid files in RobotStudio

The following section describes the configuration of these programs on the robot controller in more detail.

6.1.4 Setting up the programs

The driver consists of three programs running in parallel:

- ROS_stateServer.mod Transmits the current joint position and the state of the robot
- ROS_motionServer.mod Receives trajectories from Mikado
- ROS_motion.mod Executes the received trajectories

Separate tasks must now be created for these three programs. To do this, go to `Controller` → `Configuration` → `Controller` → `Task`. You can now add the required tasks by right-clicking and selecting `New Task`:

Name	Type	TrustLevel	Motion Task
ROS_StateServer	Semistatic	NoSafety	No
ROS_MotionServer	Semistatic	SysStop	No
T_ROB1	Normal		Yes

If the task `T_ROB1` exists on your controller and already contains motion-control modules, rename the `main()` function of `ROS_motion` to `ROS_main()` and set the entry point to `ROS_main()`.

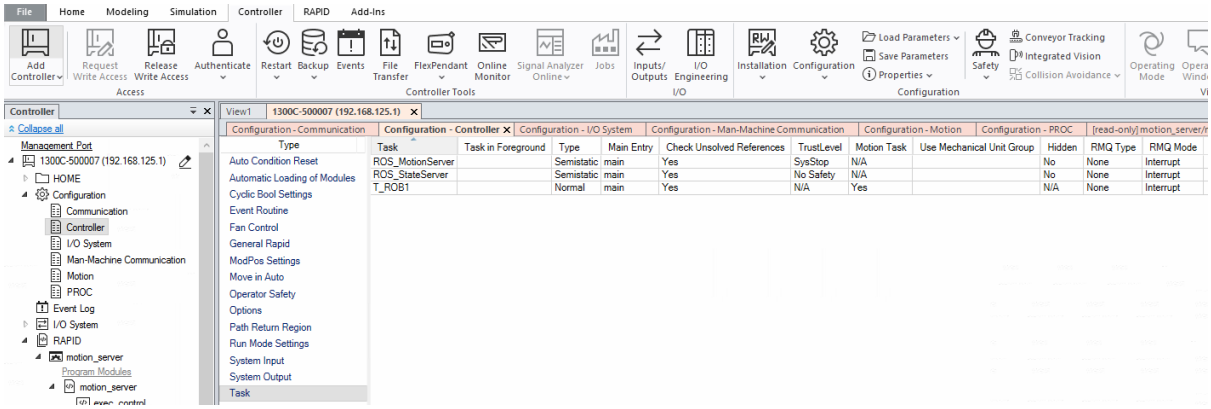


Fig. 3: Tasks to add in Robot Studio

If you plan to apply changes to `ROS_StateServer` or `ROS_MotionServer``during the setup process, it is recommended to set their task types to ``Normal` first. When going into production mode, the task types of both servers need to be changed to `Semistatic`.

Now you need to create some signals, go to `I/O System` → `Signal` and add the following signals by right-clicking and selecting `New Signal`:

Name	Type
signalExecutionError	Digital Output
signalMotionPossible	Digital Output
signalMotorOn	Digital Output
signalRobotActive	Digital Output
signalRobotEStop	Digital Output
signalRobotNotMoving	Digital Output
signalRosMotionTaskExecuting	Digital Output

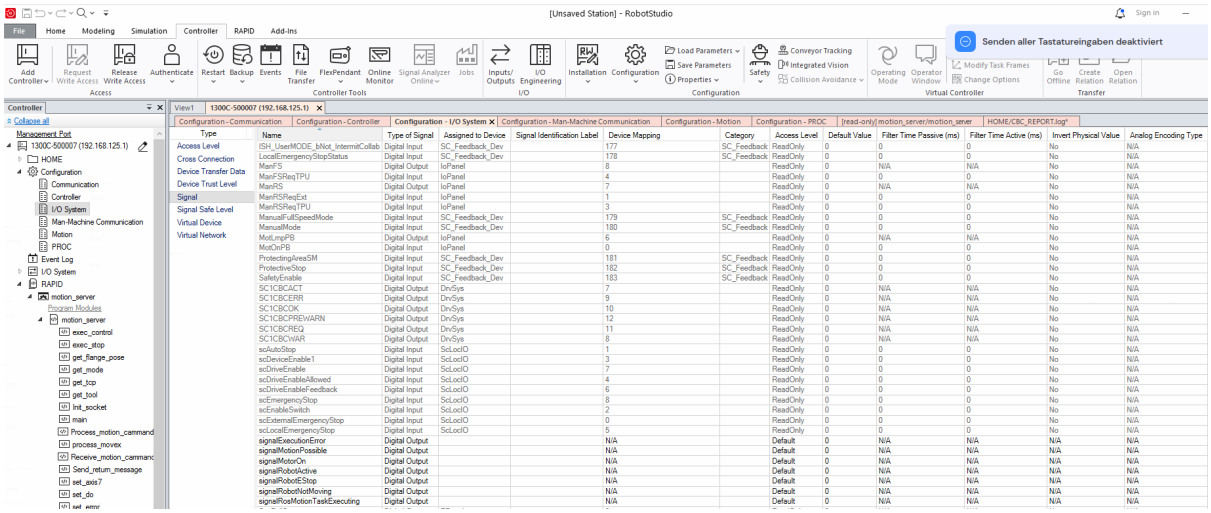


Fig. 4: Signals to add in RobotStudio

These must now be connected to the system outputs. To do this, go to `Controller` → `System Output` and right-click and add `New System Output`:

Signal name	Status	Arg 1	Arg 2
signalExecutionError	Execution Error	N/A	T_ROB1
signalMotionPossible	Runchain OK	N/A	N/A
signalMotorOn	Motors On State	N/A	N/A
signalRobotActive	Mechanical Unit Active	ROB_1	N/A
signalRobotEStop	Emergency Stop	N/A	N/A
signalRobotNotMoving	Mechanical Unit Not Moving	ROB_1	N/A
signalRosMotionTaskExecuting	Task Executing	N/A	T_ROB1

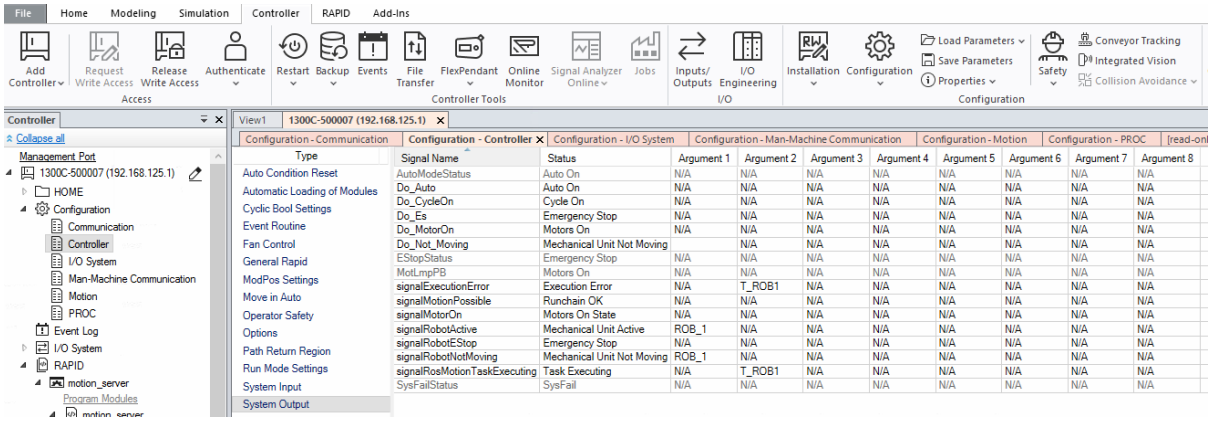


Fig. 5: System output signals in RobotStudio

Now auto-start must be activated for these programs, go to `Controller` → `Automatic Loading of Modules` and add `New Automatic Loading of Modules` by right-clicking:

Warning: To ensure the functionality of the driver, it is essential to make sure that the `ROS_common` file is installed as *shared* and is not loaded separately into the tasks. None of the tasks may contain an instance of `ROS_common`, as otherwise the data from this instance will be accessed instead of the shared data.

File	Task	Installed	All Tasks	Hidden	Shared
HOME:/ROS/ROS_common.sys	•	NO	NO	NO	YES
HOME:/ROS/ROS_messages.sys	ROS_MotionServer	NO	NO	NO	NO
HOME:/ROS/ROS_motionServer.mod	ROS_MotionServer	NO	NO	NO	NO
HOME:/ROS/ROS_socket.sys	ROS_MotionServer	NO	NO	NO	NO
HOME:/ROS/ROS_messages.sys	ROS_StateServer	NO	NO	NO	NO
HOME:/ROS/ROS_socket.sys	ROS_StateServer	NO	NO	NO	NO
HOME:/ROS/ROS_stateServer.mod	ROS_StateServer	NO	NO	NO	NO
HOME:/ROS/ROS_motion.mod	T_ROB1	NO	NO	NO	NO

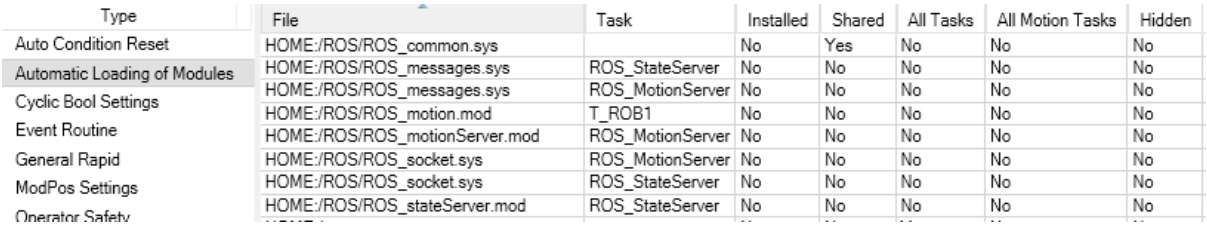


Fig. 6: Automatic loading of modules in RobotStudio

After restarting the system (ideally P-Start), the robot can now be controlled by Mikado.

6.1.5 Updating the robot programs

If you want to replace the robot programs with programs from a different version, upload the files of the new version to the robot as described in section *Installing the programs* and perform a P-start.

`ABB` → `Restart` → `Advanced` → `P-Start` → `OK`

6.1.6 Robot Control

To control robots with Mikado, use the Flex Pendant:

- Auto Mode
- Motors On
- Reset program pointer
- Start

The robot status should look as follows after successful setup:

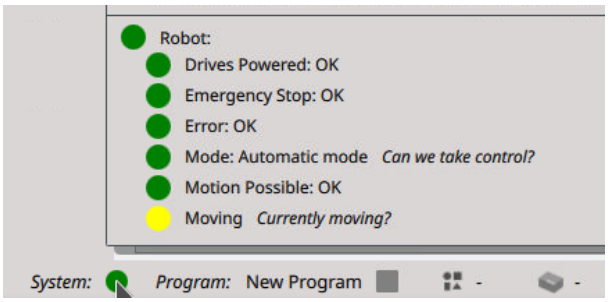


Fig. 7: Robot Status after succesfull setup

If Mikado shows the status `No Motion Possible`: is another machine running RobotStudio and currently in control of the robot? That machine needs to release control in order for Mikado to be able to gain control over the robot.

On status `Timeout` in Mikado:

- Can the robot be pinged?
  - `ping robot-ip` in Konsole
- Are the three tasks ROS_StateServer, ROS_MotionServer and T_ROB1 running on the robot?
  - If necessary activate motors, reset program pointer and press Start on the Flex Pendant

6.1.7 Configuring gripper and IO with Robot Web Services

ABB's Robot Web Service (RWS) can be used to control simple grippers and IOs in general.

When using Mikado Assistant version 3.3.1 or newer, the RWS version can be selected on the robot connection setup page.

If necessary, modify the following data in the *config/manipulator.yaml* file of your cell:

```
rws_username: Default User   #default
rws_password: robotics       #default
rws_digital_in_path: MIK_DI
rws_digital_out_path: MIK_DO
rws_version: 2
```

We recommend to update RWS user names and passwords on the ABB controller. If user name or password have been changed, the entries *rws_username* and *rws_password* have to be updated accordingly.

Mikado uses internal numbers for the individual inputs and outputs, these are appended to the prefix *rws_digital_in_path*, so if input *23* is read in Mikado, *MIK_DI23* is read on the ABB IRC in this example. Same for the outputs: a Mikado write to output 12 is mapped to ABB output *MIK_DO12*.

If RWS is used, the access level has to be changed in RobotStudio, as the default level only allows reading access.

By mapping the IOs we can unlock access to individual signals:

- Configuration > I/O System > Signal
- Right Click > New Signal ...
- Create new IO with a chosen name. Important: use the scheme *prefix* + *number*, e.g. *MIK_DO1*
- Type of Signal: Digital Output
- Assigned to Device: leave empty
- Access Level: All (allows remote access)

Now a cross connection can be created:

- Configuration > I/O System > Cross Connection
- Right Click > New cross connection ...
- Name of choice
- Resultant: name of the actual IO, e.g. *ABB_0_DO1* or a fieldbus IO

The IO signal numbers for gripper operation can now be set in Mikado Assistant.

6.1.8 Giving up control over the robot temporarily

It is possible to give up control of the robot from Mikado in order to perform other tasks/movements directly on the robot and then hand robot control back to Mikado.

This requires adjustments in both the Mikado program and the robot program `ROS_motion`. A flag is also required for the transfer, which can be read and written by both Mikado and the robot. In this example, the output *MIK_DO3* is used for this purpose.

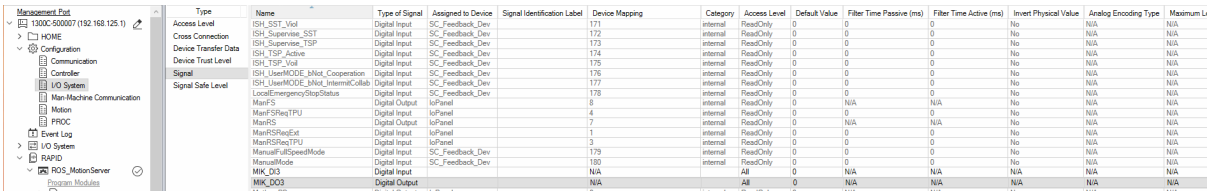


Fig. 8: View of the digital output *MIK_DO3* used in the example in RobotStudio

You must now make the following changes to your Mikado program:

- Insert a *Set Robot Output Signal* task at the point in the Mikado program where you want to transfer control. Give the task the output number and a variable with the value *true* as arguments.
- Create a new sequence called *Wait for robot control*.
- Underneath the *Set robot output signal* task, insert a sequence change that switches to the sequence you just created.
- The sequence *Waiting for robot control* should now read the robot output repeatedly in a loop. To do this, insert a *Read robot output* task, read the same output as above, and write the result to a variable.
- Insert an *IF* statement that checks the result variable. If it is *false*, this means that the robot has returned control to Mikado. In this case, you can continue with the actual sequence after updating the robot position (as shown in the screenshot using the *Read robot position* task). If the output is still *true*, we wait a while and then query again.

Logic to be executed by Mikado while the robot is in control can be inserted after setting the output and before switching to the wait sequence.

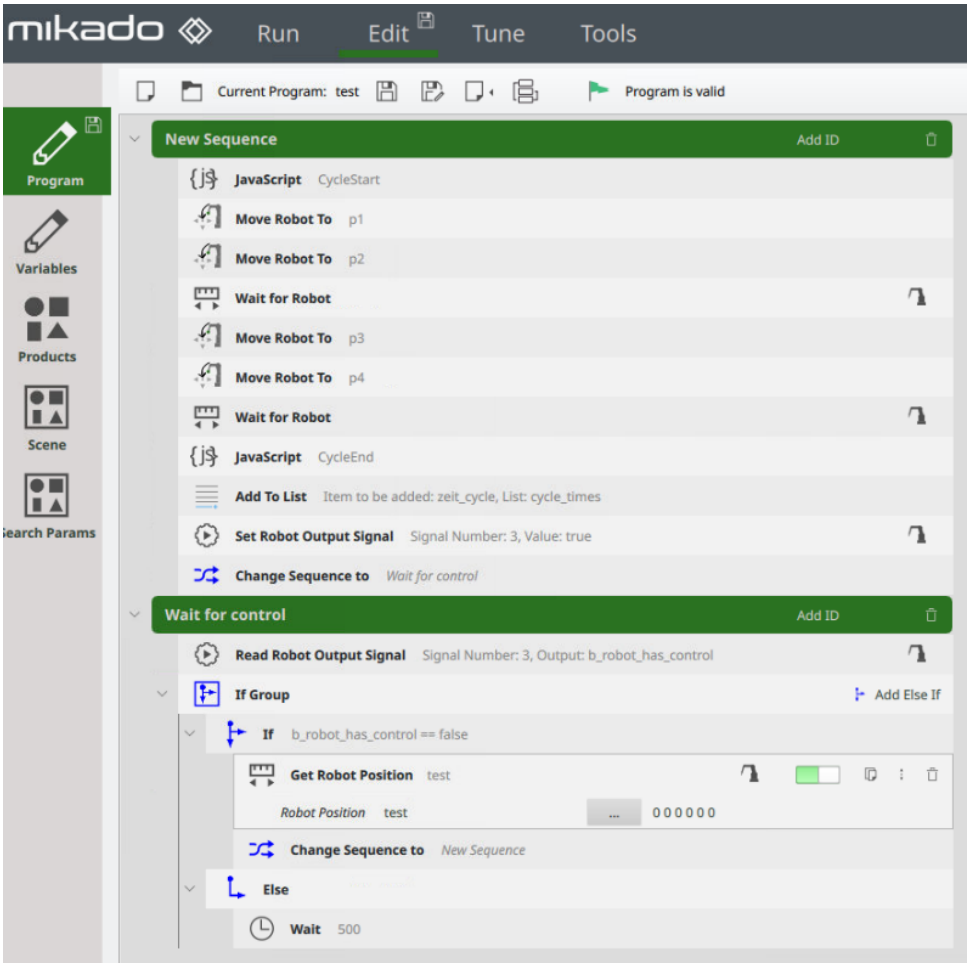


Fig. 9: Example of a Mikado program with control transfer to the ABB robot

Additionally, the `ROS_motion` robot program must be adapted. To do this, insert the output query in the *WHILE* loop of the *main* function:

```
[...]

IF(ROS_new_trajectory) THEN
  init_trajectory;
ENDIF

! check if robot should take control
! we are setting a digital output in mikado and then wait until it is reset
! in the meantime the robot can do whatever it wants
IF MIK_DO3 = 1 THEN
  TPWrite "Robot taking control";
  ! insert your commands here
  !MoveJ p9, v1000, z50, tool0;
  WaitRob \InPos;
  InvertDO MIK_DO3;
  TPWrite "Mikado taking control";
ENDIF

IF (trajectory_size > 0) THEN
[...]
```

Once the robot has executed its commands, the output must be reset to *false* so that Mikado can regain control. In

the example this is done by the command `InvertDO MIK_DO3`.

6.1.9 Troubleshooting

Increasing robot speed

To adjust the robots movement speed, open the program `ROS_motion` and adjust the value of the variable `move_speed`. Proceed with caution, especially if you have also made adjustments to the variable `stop_mode`.

Debug messages on the teach pendant

To facilitate the set up process, debug messages are enabled on the teach pendant by default (update frequency, number of trajectory points transmitted, etc.). If you want to disable them, you can do so by changing the persistent variable `isDebug` to false inside of `ROS_motionServer`. Messages that are not written to the teach pendant as messages using *TPWrite*, but rather to the event log using *ErrWrite* \W, are not affected by this.

RWS Debug Messages

TODO Um mehr Details zu RWS IO-Vorgängen zu erhalten kann das Logging-Level des RWS Adapters erhöht werden.

Add the following line in *launch/rosconsole.config*:

```
log4j.logger.ros.abb_rws_mikado_adapter=DEBUG
```

Connection issues

In case of connection problems, we recommend to first check whether the robot controller can be reached from the Mikado controller. To do this, open a console on the Mikado controller (CTRL + ALT + T) and enter the following command in the console (replace the IP with the IP of the robot controller):

```
ping 192.168.1.42
```

The ping can be canceled by pressing *Ctrl* + *C*. A positive response would look like this:

```
ping 192.168.1.42
PING 192.168.1.42 (192.168.1.42) 56(84) bytes of data.
64 bytes from 192.168.1.42: icmp_seq=1 ttl=64 time=1.18 ms
64 bytes from 192.168.1.42: icmp_seq=2 ttl=64 time=0.448 ms
^C
--- 192.168.1.42 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.448/0.816/1.185/0.369 ms
```

If the ping is successful, the robot controller can be reached. If that is the case, check whether both server programs (state and motion server) are running on the robot and where the program pointers are located. If both programs are waiting for a client connection, try restarting Mikado.

6.2 Set up Doosan Controller

Nothing needs to be installed or copied to the Doosan robot controller.

6.2.1 Network setup

A network connection between the two systems is required for communication between the robot controller and Mikado Controller. To do this, first connect an Ethernet port of the robot controller to an Ethernet port of the Mikado Controller.

- Under `Settings` → `Network` → `Controller` select e.g. `192.168.127.100` as the IP address of the Doosan controller and `255.255.255.0` as the subnet mask
- After setting up the IP address, the Doosan robot must be restarted.
- The network card of the Mikado controller connected to the Doosan controller must be assigned an IP in the same subnet, e.g. `192.168.127.200`. The network configuration of the Mikado controller is explained in more detail in the *Network adapter* section.
- The IP addresses of the Mikado controller and the robot controller must not be the same.
- Check the connection to the robot on the Mikado controller in the console: `ping <ROBOT_IP>`, e.g. `ping 192.168.127.100`

Caution: * The IP address of the robot controller must not be `10.0.0.x` in the network, as it is used internally. * Do not use Doosan controller `LAN socket 4` as it is used internally by the robot.

6.2.2 General notes

- Before the robot can be moved with Mikado, the servos should be energized. You will hear a clicking sound when this happens.
- It is recommended to switch off the robot's energy-saving mode: * Completely deactivate the energy-saving mode under `Settings` → `Idle Servo Off` * The default timeout for the energy-saving mode is 5 minutes. After this timeout, the robot switches to the Safety Off state.

6.2.3 Automatic mode change

The robot first starts in manual mode and then only moves slowly. The maximum speed can be increased by switching to automatic mode.

```
# auf Automatikmodus gehen um das Geschwindigkeitslimit zu erhöhen
# zwischen /dsr01 und /system steht das Robotermodell (kleingeschrieben)
# für das Modell M 1013 ergibt sich also:
rosservice call /dsr01a0509/system/set_robot_mode "robot_mode: 1"

# robot_mode: 1 = Automatikmodus
# robot_mode: 0 = manueller Modus
```

The tool weight can only be set in manual mode.

6.2.4 Query the name of the current tool

```
# zwischen /dsr01 und /system steht das Robotermodell (kleingeschrieben)
# für das Modell M 1013 ergibt sich also:
rosservice call /dsr01m1013/tool/get_current_tool "{}"
```

6.2.5 Query the name of the current tcp

```
# zwischen /dsr01 und /system steht das Robotermodell (kleingeschrieben)
# für das Modell M 1013 ergibt sich also:
rosservice call /dsr01m1013/tcp/get_current_tcp "{}"
```

6.2.6 Sources

- Doosan ROS Manual
- Network settings and control transfer
- Network Settings
- Network Connections
- Energy Saving

6.3 Set up Fanuc Controller

This chapter guides you through the installation and configuration of a Fanuc controller for use with a Mikado system.

6.3.1 Prerequisites

- Robotercontroller:

  Fanuc controllers for robot control such as the R-30iB with software version v7 or higher are supported. Contact us if you would like to use different hardware or software.

- Reset:

  To avoid interactions with other programs or configurations, you should reset the controller to the factory settings before connecting to Mikado if it is no longer in the delivery state. A backup of the controller configuration is also recommended (see your controller manual).

- Teach panel:

  A teach panel (e.g. the *iPendant*) is required for the one-off setup and for manual operation.

- Install the following options on the Fanuc controller:
  - *R648* - User Socket Messaging
  - *R632* or *R651* - Karel: Fanuc robots that were ordered and delivered in Europe (delivered with FRL PARAMS (*R651*)) have the option *R632* installed by default. Ordering it seperately is therefore not necessary.

  The installed options can be viewed on the teach panel under *Menu* → *NEXT* → *STATUS* → *Version ID*, there on *CONFIG* (or F3).

- Setting `$Karel_ENB`:
  - *Menu* → *System* → *Variables*: Check that `$Karel_ENB = 1` is set

- Registers, variables, server tags: The following slots must be kept free on the controller:
  - Server Tags: 3 - 5
  - Digital Outputs: 145 - 147
  - Integer Registers: 71-76
  - Position registers: 61 - 90

  If the position registers are not available, they can be reconfigured. After successfully installing the robot files on the controller, select the Karel program `ROS_TRAJ`. Then press the *DATA* button and select the Karel variables. If the correct program has been selected, you should now see the data structure `cfg_` along with other variables. Select it with *ENTER* and adjust the following two variables as needed:
  - *pt_buff_size* (default: 30): Number of position registers used by the driver. It is not recommended to change this value unless absolutely necessary.
  - *first_pr* (default: 61): The first position register used. Make sure that at least *pt_buff_size* subsequent position registers are available.
- Kollisionsgrenzen / Dual Check Safety (DCS):

  If you have defined collision limits in the Fanuc controller or use Dual Check Safety, please note that these limits are not taken into account in the collision and path planning of Mikado. Therefore, always select a configuration in which the collision limits of the Fanuc controller are less restrictive than those in the Mikado configuration.
- USB stick: It is best to use a USB stick as the installation medium. Alternatively, you can also transfer files via the network (FTP) (see Fanuc manual).

6.3.2 Network configuration

First, an Ethernet connection must be established. To do this, connect the Fanuc robot controller from its Ethernet port 1 to the corresponding Mikado controller Ethernet port. A new connection should now be displayed in the Mikado controller in the menu bar at the bottom right. In the case of a direct connection, this must now be set to a fixed IP and subnet mask. How a Mikado controller receives a fixed IP is explained in section *Network adapter*.

The IP address configuration of the robot controller now also has to be checked and updated.

- *Menu* → *Setup* → *Host Comm*
- *TCP/IP* → *F3 Detail*
- *Port#1 IP Addr*: if necessary, update the IP address to match the Mikado controllers robot network connection. The robot controller and the Mikado controller cannot use the same IP address.

Hint: Make sure that the IP addresses of the Mikado Controller and Fanuc Controller are in the same address space and that the subnet masks match.

Then enter the IP address of the robot controller in the **Mikado Assistant** under *Robot* → *Connection* in the corresponding field and save the settings.

You can find a connection check description in *Testing the connection*.

6.3.3 Installing the Mikado interface

Version compatibility has been tested with version v7.70/v8.30/v9.30, the installation file may also be compatible with higher versions.

- First copy all the required installation files to the root directory of a USB stick (FAT32 formatted). You will find the files on the Mikado controller in the directory `/opt/isys-vision/mikado/robot_programs/fanuc/fanuc_install_packages/fanuc_mikado_driver_VX.XX/` or using the corresponding button in the Mikado Assistant ("Parameters" -> "Robot" -> "Open directory")

Hint: It is important that the files are copied to the root directory of the USB drive, as the installation script will not be able to find the files otherwise.

- Insert the USB stick into the teach panel of the robot and navigate to the content
  - *Menu* → *File* → *File* → *All*
  - *F5 UTIL* → *Set Device* → *UT1:* (USB Port of the Teach Pendant)
- Select the installation file `install.cm` and execute the installation by pressing the *Enter* key
- To update an existing driver installation, you first have to remove the previous installation:
  - *FNCTN* → *Abort ALL (2x)*
  - First run `uninstall.cm`, then `install.cm`
- After successful execution of the installation script, restart the robot controller

6.3.4 Parameterizing the Mikado interface

- Flags and registers:

  As mentioned in section *Prerequisites*, some DOs and registers are used for communication. Please check that these are not in use and comment on the DO and register descriptions if necessary.
- (Optional): Enable HTTP access and KCL. This enables the automatic start of the Fanuc programs with Mikado.
  - Prerequisite: HTTP access is enabled in the settings
  - Use the Teach-Pendant to navigate to *Menu* → *SETUP* → *Host Comm* → *HTTP*
  - Select the line in which Resource: `KCL:\*` is entered
  - Activate KCL by executing *Unprotect*
  - (if possible) Set up the following access data in the corresponding fields: User: `ROS`, Password: `ROS`

Hint: Section *Optional: Execute robot program* explains what you also need to configure so that native robot programs can be integrated into your Mikado program

---

6.3.5 Start system

The 'MIKADO' program starts all the necessary subroutines and thus establishes the connection to Mikado.

Hint: If you want to stop the running Fanuc program, execute *ABORT ALL* (to be found under *Functions*).

6.3.6 Optional: Gripper control via I/Os

If you want to control your gripper via inputs and outputs, you can implement this as standard via digital inputs and outputs (DI and DO). The configuration options for this are listed in the *gripper.yaml* section.

However, if you want to use the robot I/Os (RI and RO), you must set up a mapping to DIs and DOs as follows:

- Create new background logic program (BG Logic)
- Example pseudocode:

```
RO[1] = DO[101]
RO[2] = DO[102]
DI[101] = RI[1]
```

- In the Mikado configuration, the gripper in this example is controlled via the output signals 101 and 102. Signal input 101 is read from robot input 1.

6.3.7 Optional: Adjust robot speed in Mikado

The execution speed of robot movements can be adjusted directly from Mikado. However, a fixed speed of 70% is used by default. For more information about speed and other movement parameters, see the *Optional: Fine-tune movement parameters* section.

To adjust the speed directly from Mikado, entries must be added to the `manipulator.yaml` file. This file is located in the cell's config folder.

Add the following lines at the end of the file:

```
connection_info_is_traj_velocity: true
# velocity override specifies the target velocity of each motion between 1.0 and 100.0
↪%
velocity_override: 60.0
```

The velocity override accepts values between 1.0 and 100.0. When set, all trajectory points are executed at the specified target speed. The speed can also be adjusted during Mikado execution using the JavaScript function `lab.setRobotVelocityOverride(double velocityOverride)`. However, the speed setting always applies to an entire trajectory.

6.3.8 Optional: Fine-tune movement parameters

The robot's movements can be fine-tuned using three parameters to adjust speed, motion smoothness, and path accuracy. Suitable values for these parameters depend heavily on the specific application and the trajectories used. They can therefore be adjusted manually as needed.

Warning: Users who are unfamiliar with programming Fanuc robots are strongly advised against modifying these parameters. Exercise extreme caution when tuning movement parameters to prevent damage or injury. After changing the parameters, always start the motion sequence at a reduced speed and increase it gradually.

The FANUC motion instructions can be found in the TP program ROS_MOVESM. It contains two different types of movements: a movement with a blending value enabled and a fine movement (for the final trajectory point):

```
24:  J PR[R[71:curr_pr]] R[77:velocity]% CNT50 ACC60    ;
25:  JMP LBL[44] ;
26:  LBL[55:last_pt] ;
27:  J PR[R[71:curr_pr]] R[77:velocity]% FINE ACC60    ;
```

1. **Velocity:** The target speed of the movement is specified immediately after the target position register and is expressed as a percentage. It is read from a register (register 77 by default) and is set to 70% unless configured otherwise. To adjust the speed, either replace the indirect register assignment with a direct assignment (e.g. 60%) or change the speed from the Mikado software, as described in the *Optional: Adjust robot speed in Mikado* section. Depending on the trajectory characteristics and blending values, the target speed is often not reached during short movements.
2. **Blending Value (CNT Value):** The blending value determines how precisely each point along the trajectory must be reached. The higher the CNT value, the more the robot is allowed to deviate from trajectory points in order to maintain a higher speed. This allows for faster and smoother movements at the expense of accuracy and path fidelity. The higher this value, the further the robot may deviate from the collision-checked path, which can lead to collisions. Therefore, increase the CNT value only in small increments and test how this affects the execution of the trajectories. Changes to this parameter are made at your own risk.
3. **Acceleration:** Acceleration is also specified as a percentage, but uses the ACC prefix. The higher the acceleration value, the faster the robot attempts to reach the specified speed. For dense trajectories with closely spaced points, a high acceleration value can result in jerky movements.

6.3.9 Optional: Execute robot program

If you want to execute your own robot program within your workflow, this should be implemented using a control handover, as described under *Optional: Give up robot control*. You can then insert your program, for example, within the MIKADO robot program above the last JMP command, so that it is executed and Mikado subsequently regains control.

If it is necessary to distinguish between several programs that need to be called, you can achieve this by setting robot outputs and querying them in the robot program.

6.3.10 Optional: Give up robot control

In some workflows, it may be necessary for Mikado to relinquish control of the robot so that other robot programs can be executed. To do this, the Mikado motion program ROS_MOVESM must be exited, which can be triggered by setting output DO[147].

After exiting ROS_MOVESM, program execution continues in the robot program MIKADO: It jumps to a dedicated section of the program where you can simply insert your own program calls (see LBL[2] in the MIKADO program).

If you want to exit the MIKADO program completely, remove the JMP command from the last line of the program. However, if you do this, you will need to call the MIKADO program again later before Mikado can control the robot again.

To trigger the relinquishing of control, add a *Set robot output signal* task at the end of a Mikado sequence and select signal number 147. When the MIKADO robot program becomes active again, it switches DO[147] off. You can wait for this change in the Mikado program sequence using the task *Wait for robot output signal* (available since Mikado 3.4.0) before continuing with motion tasks. Since the robot may have moved in the meantime, it is important to insert a *Read robot position* task into the program sequence, as shown in the following screenshot.

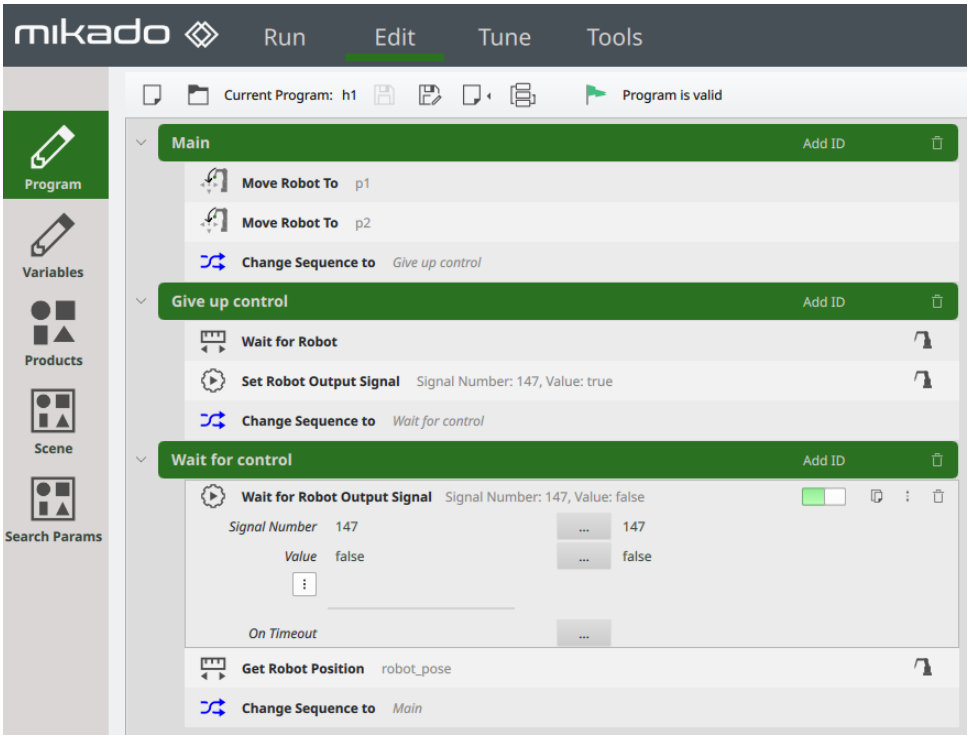


Fig. 10: Example of a Mikado program giving up control to the Fanuc robot

6.3.11 Troubleshooting

Debug messages on the teach pendant

For commissioning purposes, debug messages for the teach pendant are enabled by default on the robot and are displayed on the User screen. If you want to disable them, you can do so setting Karel variables. Select the Karel program `ROS_TRAJ`. Then press the *DATA* button, select the Karel variables, and press the *ENTER* key to open the configuration structure `cfg_`. Set the variable `is_debug` to *FALSE*. Proceed in the same way with the `is_debug` variable in the Karel file `ROS_STATE`.

Robot moves slowly or jerkily

The robot's motion behavior can be adjusted using various movement parameters. These are described in the *Optional: Fine-tune movement parameters* section.

6.3.12 Testing the connection

Troubleshooting if there is no connection to the robot after configuration and restart. In the *Status bar* area, you can see whether the connection to the robot has been successfully established.

- Check: Robot controller and Mikado controller have the same subnet mask and different fixed IPs in the same address space
- Check: The status LED of the network port does not indicate an error
- Check: Can the robot be reached via *ping* under the entered IP

The necessary tools to execute the ping command from the Mikado controller:

A console must be started on the Mikado controller. The ping command to be entered is then, for example:

```
ping 192.168.1.42
```

Pinging can be canceled by pressing *Ctrl* + *C*. A positive response would look like this:

```
ping 192.168.1.42
PING 192.168.1.42 (192.168.1.42) 56(84) bytes of data.
64 bytes from 192.168.1.42: icmp_seq=1 ttl=64 time=1.18 ms
64 bytes from 192.168.1.42: icmp_seq=2 ttl=64 time=0.448 ms
^C
--- 192.168.1.42 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.448/0.816/1.185/0.369 ms
```

6.4 Set up KUKA Controller

The following describes how to prepare the robot for using Mikado, in particular the use of the various KRL scripts. For steps that differ when using iiQKA.OS2, a separate paragraph can be found in the following instructions. If there is no separate listing, the steps are identical for both OS versions.

6.4.1 General

Kuka System Software (KSS)

Two interpreters run on the robot: The submit interpreter (`C:\KRC\ROBOTER\KRC\R1\System\mikado_update.sub`) and the robot interpreter (`C:\KRC\ROBOTER\KRC\R1\Program\mikado.src`). The submit interpreter is usually started automatically and then runs in the background, the robot program must be selected. These both communicate with the Mikado controller via EthernetKRL.

iiQKA.OS2

Two interpreters run on the robot: The submit interpreter (`C:\KRC\ROBOTER\KRC\R1\TP\Mikado\Service.sub` and `C:\KRC\ROBOTER\KRC\R1\TP\Mikado\Status.sub`) and the robot interpreter (`C:\KRC\ROBOTER\KRC\R1\Program\mikado.src`). The submit interpreter is usually started automatically and then runs in the background, the robot program must be selected. These both communicate with the Mikado controller via EthernetKRL.

6.4.2 System requirements

Kuka System Software (KSS)

Mikado communicates with the KUKA controller via EthernetKRL and uses splines for movement commands. Therefore, at least KSS 8.3 must be installed as well as the additional package EthernetKRL 2.2 or higher. Since two submit programs are running for performance reasons, the MultiSubmit extension must be installed for KSS 8.3 and 8.4; from KSS 8.5 onwards this is standard.

iiQKA.OS2

- **iiQKA.OS2** Version 9.3+
- **Ethernet KRL** Version 6.2.1+
- the Windows applications **iiQWorks.Cockpit**, **iiQWorks.App Builder**, and **iiQWorks.Sim** (the Basic version is sufficient)

6.4.3 Network configuration

A network connection must be established for communication between the Mikado controller and the robot controller. By default, the **KLI** interface (**KUKA Line Interface**) of the robot controller with the following connections is intended for this purpose:

- KR C4: `X66`
- KR C5: `XF5`

> **Warning:** Without additional option packages, these network connections must be used. Other RJ45 connectors on the robot controller do not work at all for communication with Mikado (e.g. KSI) or must be activated using additional option packages. Alternative connection options available with option packages are explained in more detail below.

Define a fixed IP address and an associated subnet mask for both the Mikado controller and the robot controller. The network configuration of the Mikado controller is explained in more detail in the *Network adapter* section.

Information on the network configuration of the KUKA controller can be found in the manufacturer's user manual.

The IP address of the robot controller must then be stored in the Mikado cell. To do this, navigate to `Parameters` → `Robot` in the Mikado Assistant and enter the IP address of the robot controller in the corresponding field.

Hint: Make sure that the IP addresses of the Mikado Controller and KUKA Controller are in the same address space and that the subnet masks match. If, as recommended, a separate network connection is used for the camera, this must be in a different address space in order to avoid ambiguous packet transmission.

The IP address of the KLI interface of the robot controller must **not** be within the following ranges reserved by KUKA:

- 192.168.0.0 - 192.168.0.255
- 172.16.0.0 - 172.16.255.255
- 172.17.0.0 - 172.17.255.255

Alternative connection options:

If the above-mentioned default network connections for the KLI interface are otherwise occupied (e.g. by the robot's Profinet communication), the following alternatives can be considered:

- An external ethernet switch can be used on `XF5` (KR C5) or `X66` (KR C4) to increase the number of available connectors for the KLI interface.
- KR C5 controllers have an internal network switch for the KLI interface, whose connections can be activated using option packages.
  - **KUKA.EtherNetSwitch Dual-Port**: connector `XF6` can also be used in addition to `XF5`
  - **KUKA.EtherNetSwitch Quad-Port**: connectors `XF3`, `XF4` and `XF6` can also be used in addition to `XF5`

- Depending on the configuration, KR C4 controllers may also have a switch for the KLI interface. If present, these are the connections `X67.1`, `X67.2`, and `X67.3`. However, depending on the configuration of the controller, these connectors can also be used for other optional interfaces and are not always clearly assigned to the KLI interface.

  To find out whether these connections can be used for KLI, please consult the manual for your specific configuration.

- The **KUKA.OptionalNetworkInterface** option can be used to activate the **KONI** interface (`XF7`) on KR C5 controllers. This can also be used for communication via Ethernet KRL. In contrast to the *KUKA.EtherNetSwitch* options for the KLI interface, KONI offers an additional, completely independent network interface that can be configured independently (own IP address, subnet, etc.). This makes it possible, for example, to use different network interfaces for Ethernet KRL communication and Profinet communication of the robot. This can prevent communication problems when Profinet communication traffic is very high, see *Troubleshooting*.

  KONI configuration: Ports 54600, 54601, and 54602 must **not** be listed in the list of ports under `Network Configuration` → `KONI` → `Advanced` → `NAT`. These ports are required for communication with Mikado and must therefore not be used for any other purpose.

- Depending on the configuration, KRC4 controllers may also have a **KONI** interface for configuring a standalone network interface that is separate from the KLI interface. If available, these are the `X64.1`, `X64.2`, and `X64.3` connections.

  To find out whether KONI is available, please consult the manual for your specific configuration variant.

6.4.4 KSS: Prepare KRC

To be able to control the KUKA robot with Mikado Control, the corresponding robot programs must be copied to the robot controller, configured and started. The steps required for this are explained in more detail below.

If your robot uses iiQKA.OS2, different steps are required for the installation of the software, which are described in section *iiQKA.OS2: Prepare KRC*.

KSS: Copying the files

First, the required robot programs must be copied to the robot controller. These programs can be found on the Mikado controller under:

`/opt/isys-vision/mikado/robot_programs/kuka/`

or using the button in the Mikado Assistant (`Parameters` → `Robot` → `Open directory`). There are three subfolders within the directory, each of which corresponds to a folder on the robot controller:

- `EthernetKRL/*` → `C:\KRC\ROBOTER\Config\User\Common\EthernetKRL\*`
- `Program/*` → `C:\KRC\ROBOTER\KRC\R1\Program\*` or `KRC:\R1\Program\*`
- `System/*` → `C:\KRC\ROBOTER\KRC\R1\System\*` or `KRC:\R1\System\*`

You will find the network settings in the EthernetKRL folder. Since a KUKA controller can have several network connections, the IP of the connection used for communication with Mikado must be stored in the configuration files (the IP address of the robot controller, not that of the Mikado controller!) To do this, you must enter the IP address of the KUKA controller in line 12 of the files `MikadoXmlService.xml`, `MikadoXmlStatus.xml` and `MikadoXmlTraj.xml`. This can be done quite easily with the text editor Kate, for example, after you have copied the files from the above-mentioned directory to a USB stick (the folder /opt and all files contained therein are write-protected, which is why editing is only possible after copying). Somewhat less conveniently, the files can also be edited directly on the KUKA controller. After making changes to the EthernetKRL XML files, a restart of the KUKA controller is necessary.

The system folder contains programs that are executed by the submit interpreter (see below).

The Program folder contains programs that are executed by the robot interpreter. In particular, this is the "mikado.src" program, which in turn calls other .src files.

To copy files from the USB stick with the SmartPAD to the KUKA controller: `Display` → `Window` → `Navigator`. Files can be selected in the Navigator and then copied via `Edit` → `Copy` and `Edit` → `Paste`. A USB mouse on the SmartPAD simplifies operation. With sufficient user rights, the HMI interface can also be minimized and Windows Explorer can be used.

Hint: For security reasons, USB sticks may not be visible in some user groups.

Switch to at least the `Expert` user group if the USB stick is not shown as a storage device on the SmartPAD. If it is still not shown, use a different USB port (e.g., directly on the robot controller instead of on the SmartPad) or a different USB stick.

KSS: Submit Interpreter [S]

Switch to T1 mode and then to the Expert user level.

First, deselect all previously installed Mikado submits (`mikado_update`, `mik_service_sps`, `mik_status_sps`). Then, in the submit interpreter, select the file `mikado_update.sub` for slot EX1.

Go to `S` → `Stop` and `Deselect`, then `Display/Assign` and select the new submit `mikado_update.sub`. The robot should now be displayed in the correct position in Mikado Control. This submit is sending updates to Mikado if the file `mikado.src` is not being executed in the robot interpreter or if the robot is in operation mode `T1` or `T2`. While the robot is in automatic mode and `mikado.src` is running, the robot interpreter takes over communication with Mikado for performance reasons.

To ensure that the module is also automatically selected and started when the robot controller is restarted, it must be selected in the cold start configuration. To do this, switch from the `Current display/assignment` tab to `Cold start configuration` and select the module for slot EX1 there too.

KSS: Robot Interpreter [R]

Now select the main program `mikado.src` for the robot interpreter and start it. To do this, navigate to the "Program" folder on the SmartPAD (see above), select the program and press the `Select` button at the bottom of the screen. The program is now preselected for execution and must then be started as follows, depending on the operating mode:

- Manual modes **T1 / T2**: Press and hold the enabling button on the back of the SmartPAD. Then press and hold the `Start` button. (It may be necessary to press the `Start` button several times until the program has started successfully). Both buttons must be held down permanently in manual modes so that the robot program is executed and Mikado can perform movements. This operating mode is therefore suitable, for example, for setup (performing calibration, teaching poses, etc.), as the robot can be stopped immediately by releasing one of the two buttons.
- Automatic modes **Auto / External**: Press the `Start` button (several times if necessary). It is not necessary to hold the enabling switch on the back of the SmartPAD in automatic modes. The program is started and then continues to run automatically without additional user interaction (i.e. unlike manual modes, no button needs to be held down permanently).

If the robot program has started successfully, the symbol for the robot interpreter `R` is displayed in green at the top of the SmartPAD screen. If the symbol has a different color (e.g. orange or red), the robot program has not yet started successfully.

If you now make a small movement from axis 6 in Mikado on the robot side, for example, this should be executed by KUKA.

6.4.5 iiQKA.OS2: Prepare KRC

To be able to control the KUKA robot with Mikado Control, the corresponding robot programs must be copied to the robot controller, configured and started. The steps required for this are explained in more detail below.

If your robot uses KSS, different steps are required for the installation of the software, which are described in section *KSS: Prepare KRC*.

iiQKA.OS2: Installing Ethernet KRL

1. If not already done, it is necessary to install the Ethernet KRL option package. To do this, open **iiQWorks.Cockpit** and install Ethernet KRL as shown in the screenshot:

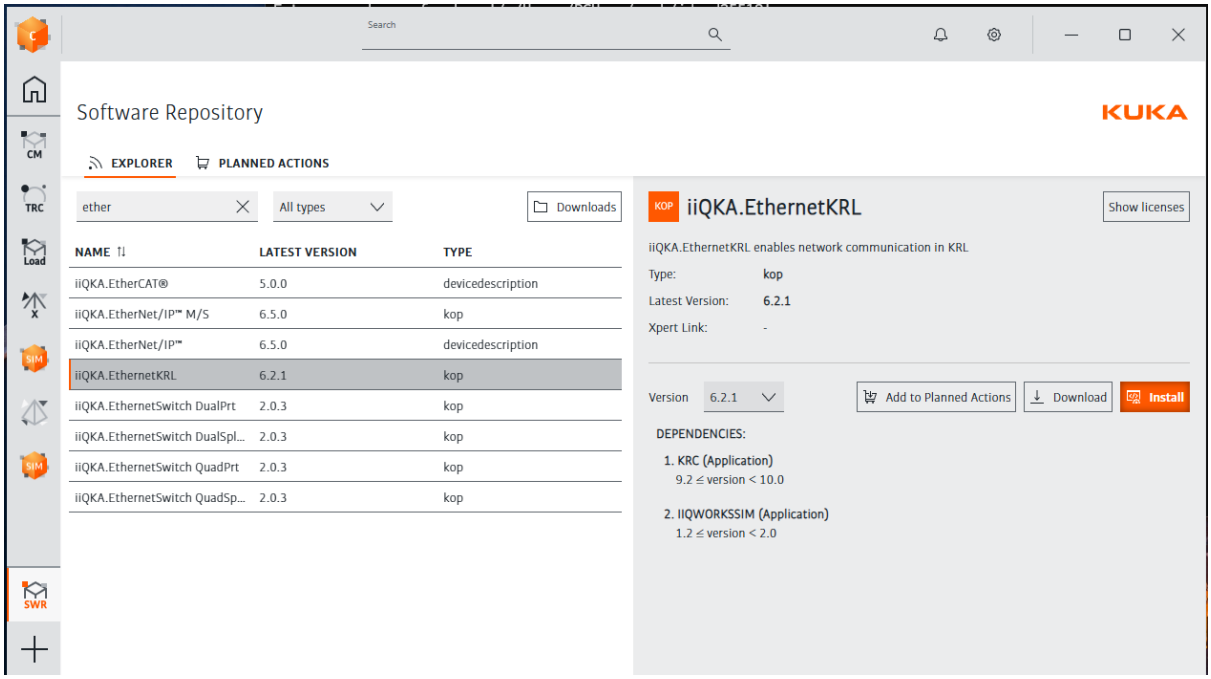


Fig. 11: Installation of the Ethernet KRL option package in **iiQWorks.Cockpit**

2. Afterwards, the option package still needs to be installed on the corresponding robot. To do this, open the corresponding instance with iiQWorks.Sim as shown in the screenshot:

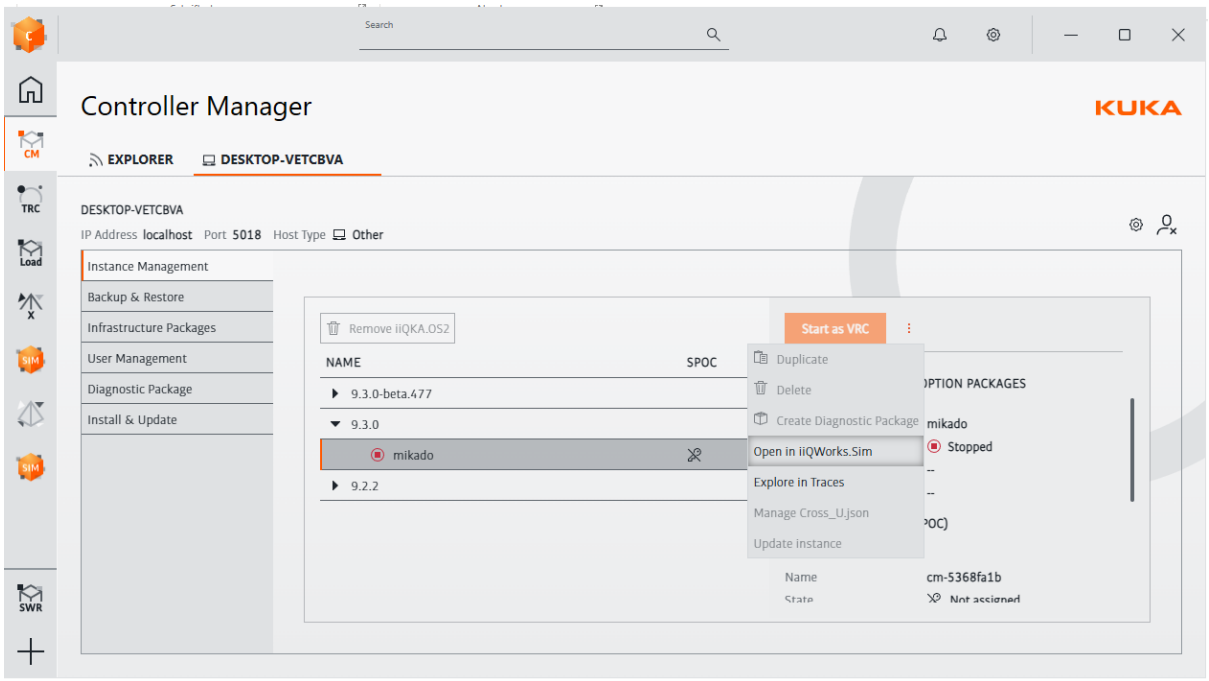


Fig. 12: Opening the instance in **iiQWorks.Sim**

3. Select the robot in the Device Tree and add the Ethernet KRL option package in the properties window on the right-hand side. Refer to the markings in the screenshot for this:

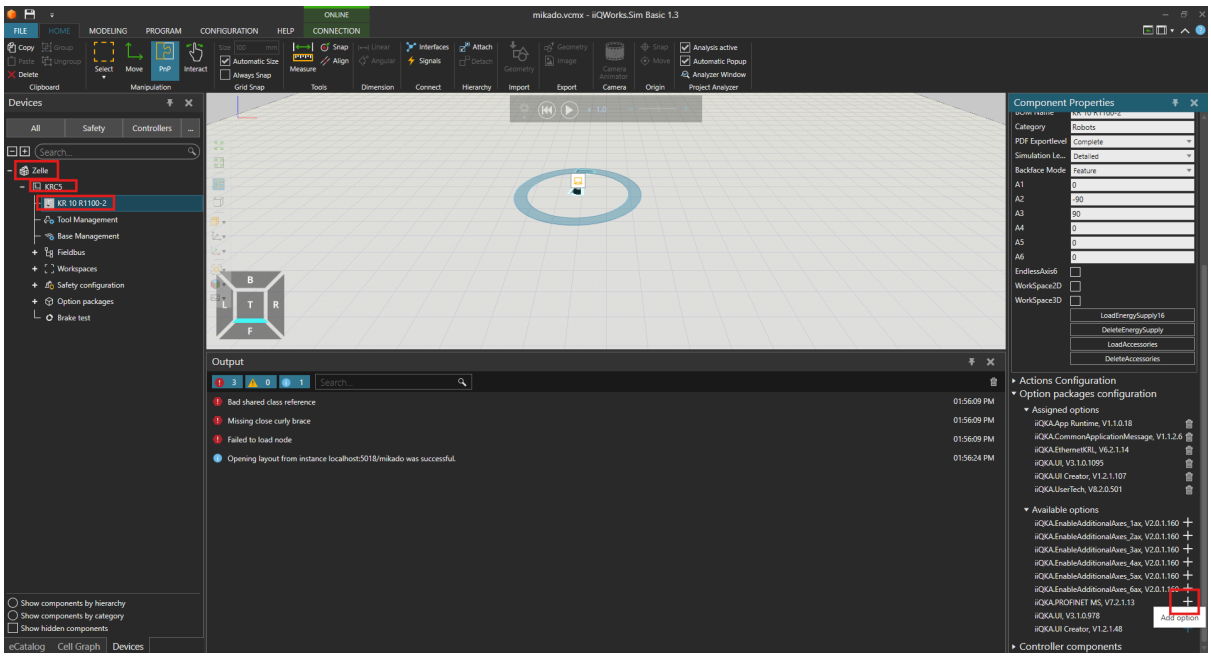


Fig. 13: Adding the option package in **iiQWorks.Sim**

4. The configuration changes must now be applied to the robot. To do this, switch to the `CONNECTION` tab as shown in the following screenshot and deploy the changes to the controller

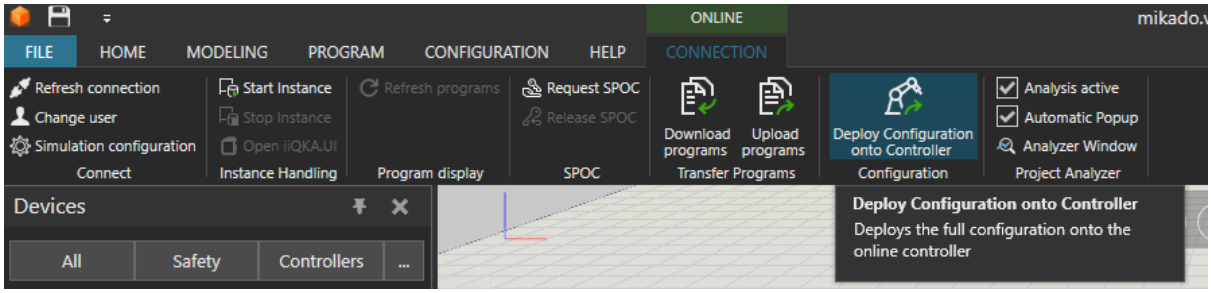


Fig. 14: Deploy the configuration changes to the controller

iiQKA.OS2: Installing Mikado

The **iiQWorks.App Builder** is required to install the Mikado package.

1. On the Mikado controller, open the folder `/opt/isys-vision/mikado/robot_programs/kuka_iiqka.os2/`. The easiest way to do this is to open the Mikado Assistant, navigate to the robot connection page, and open the directory using the corresponding button (`Robot` → `Connection` → `KUKA-Controller setup` → `Open directory`).
2. Copy the contents of the folder `/opt/isys-vision/mikado/robot_programs/kuka_iiqka.os2/` to a USB flash drive.
3. Copy the folder from the USB drive to a Windows computer that is connected to the robot.
4. Open the **iiQWorks App Builder**
5. Select the `Open project folder` option as shown in the screenshot and open the folder you just copied.

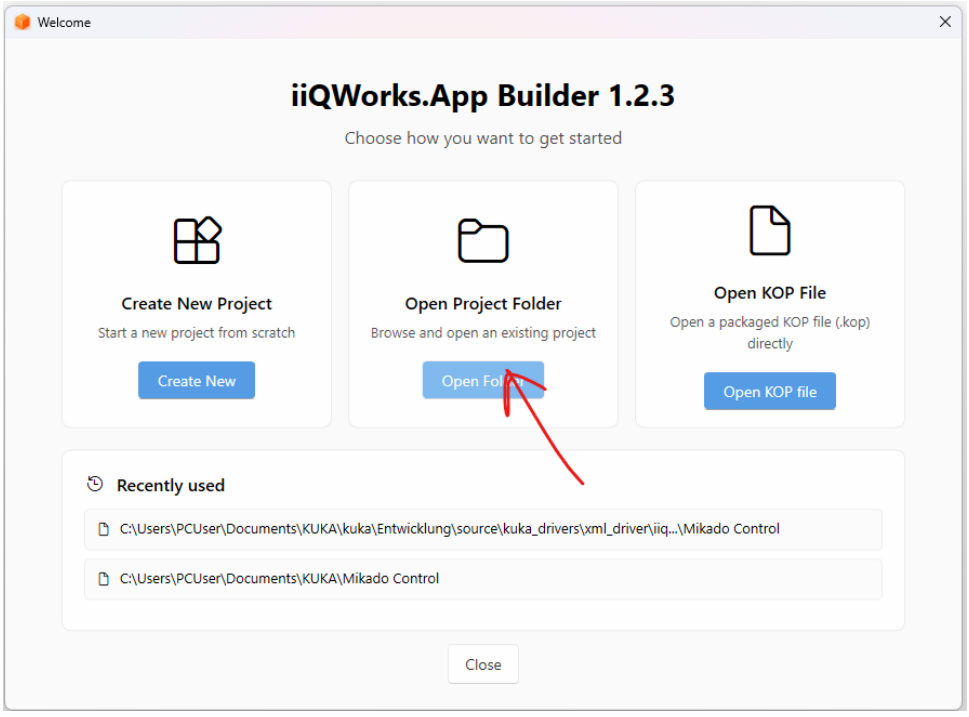


Fig. 15: Open the folder copied in the previous step in the App Builder

6. Select the controller of your robot via `Extras` → `Select KRC/VRC for deployment`. Be sure to select the correct instance as well!

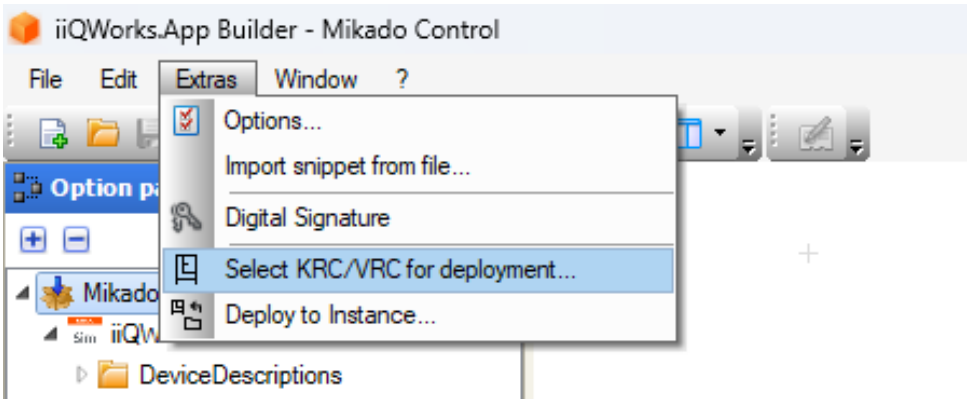


Fig. 16: Select your robot controller for deployment

7. Deselect all programs and submits on the robot.
8. Deploy the files to the robot: `Extras` → `Deploy to Instance`.
9. Request the SPOC (“Request SPOC”) and confirm the request on the teach panel.
10. Confirm the request on the teach panel.
11. Click `Deploy` to transfer the files.

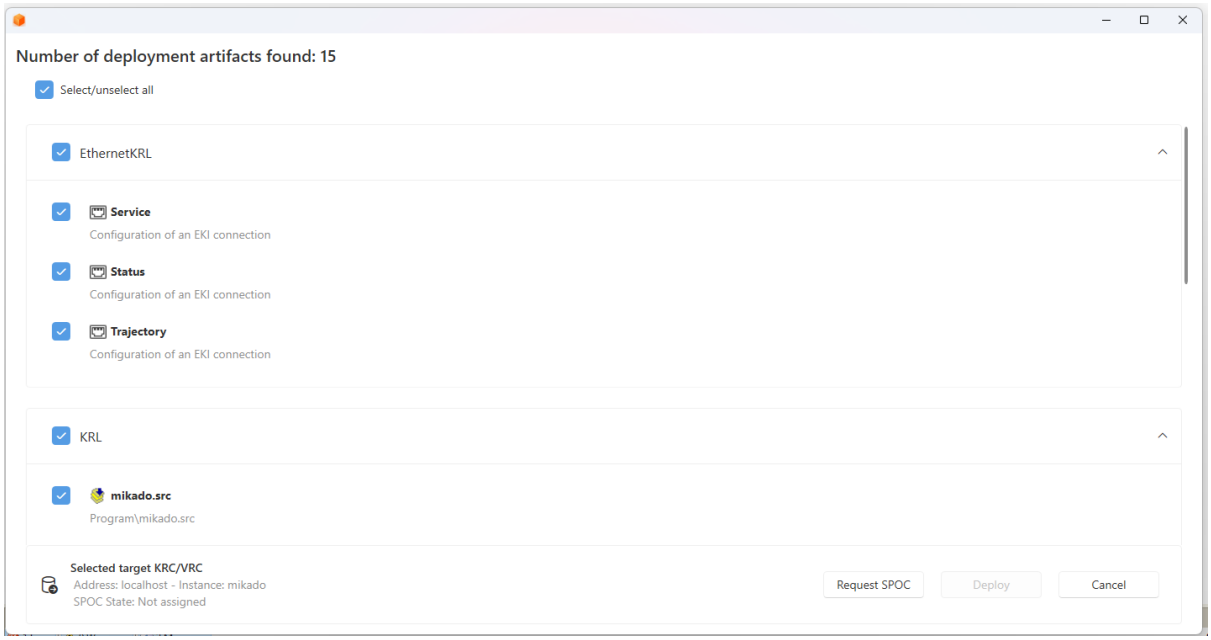


Fig. 17: Click `Deploy` to transfer the files.

iiQKA.OS2: Submit Interpreter [S]

In the Submit Interpreter, select the file `Service.sub` for slot EX1 and the file `Status.sub` for slot EX2. Proceed as follows:

1. Switch to T1 mode.
2. Switch to the Expert user level.
3. Deselect all submits via `S` → `Deselect all`.

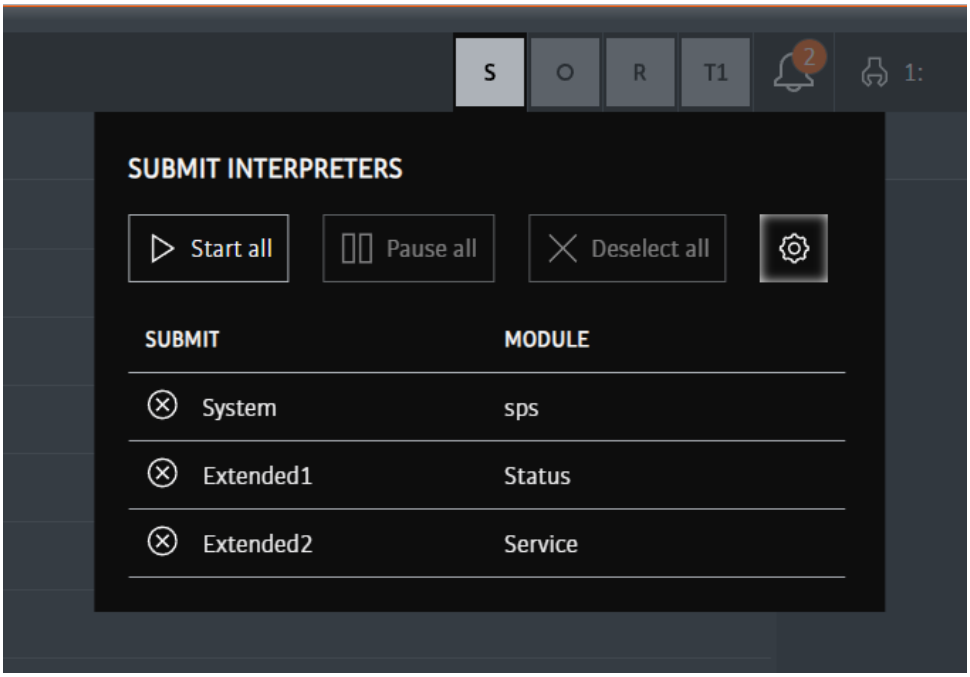


Fig. 18: Any currently running submits must be deselected.

4. Open the Submit assignment menu via the gear icon.

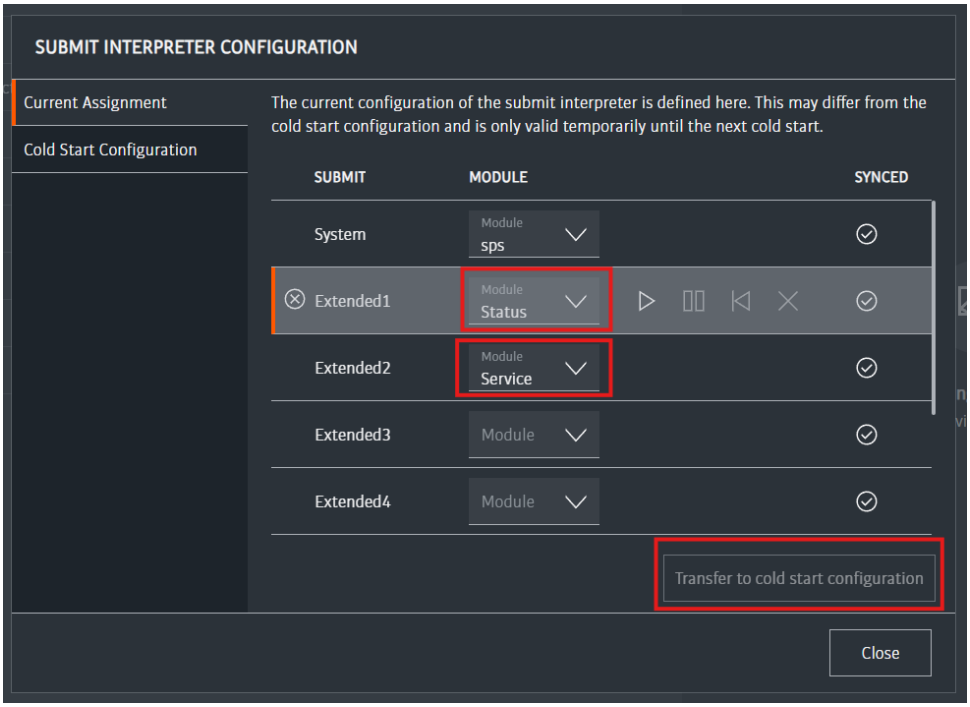


Fig. 19: Open the assignment menu and configure slots EX1 and EX2.

5. In the Submit Interpreter, select the file `Service.sub` for slot EX1.
6. In the Submit Interpreter, select the file `Status.sub` for slot EX2.
7. Press the button `Transfer to cold start configuration` so that the submits are also loaded correctly after a cold start.
8. Close the assignment menu and press `Start all` to start the execution of the submits.

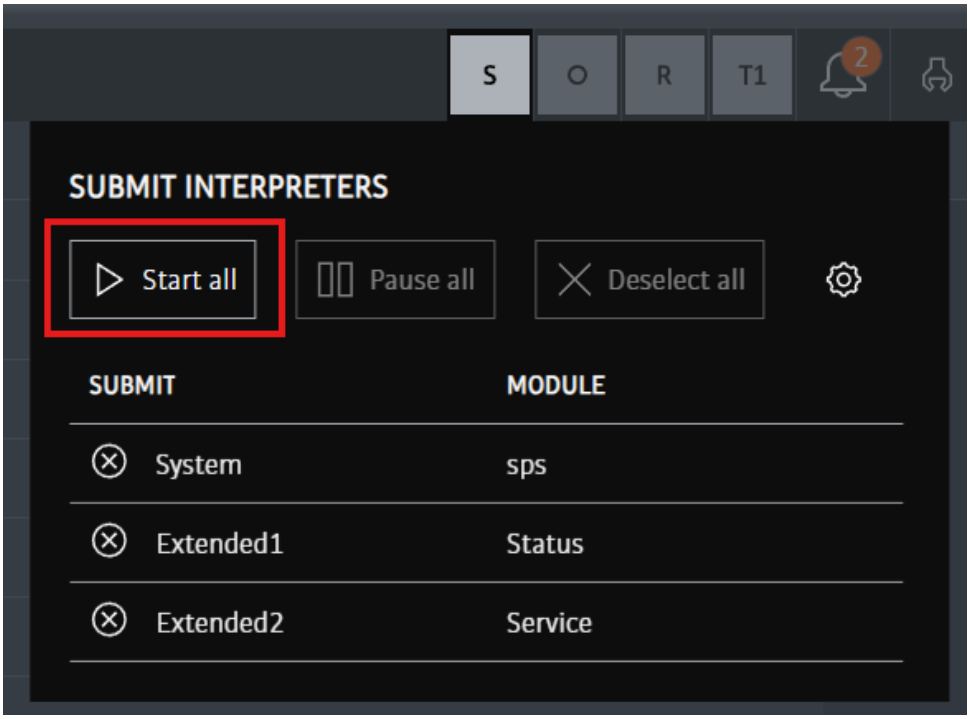


Fig. 20: Start the submits by clicking `Start all`.

iiQKA.OS2: Robot Interpreter [R]

The `mikado.src` program running in the robot interpreter is responsible for receiving and executing trajectories. To start it, please proceed as follows:

1. Select the `mikado` program from the “Program” folder.

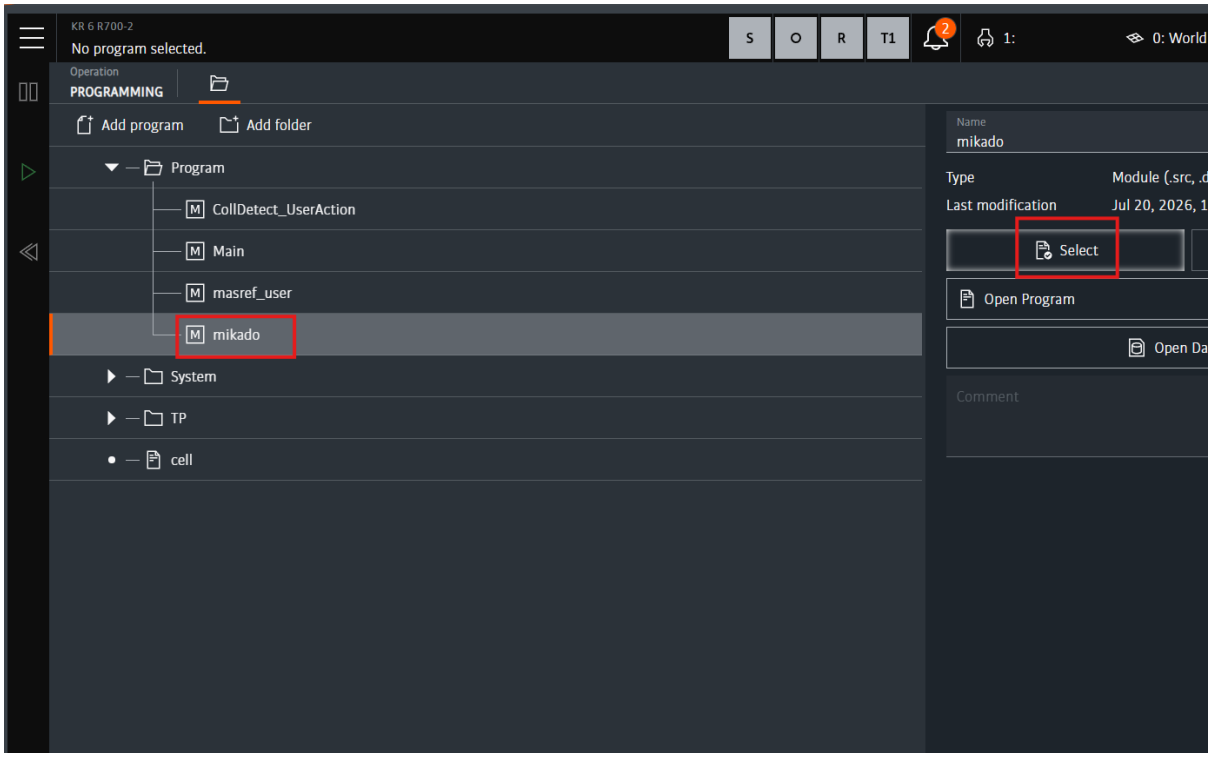


Fig. 21: Select the `mikado` program.

2. To make the robot ready for operation, hold down the dead-man switch (if you are in T1 mode), or switch on the drives as shown in the screenshot (in automatic mode).

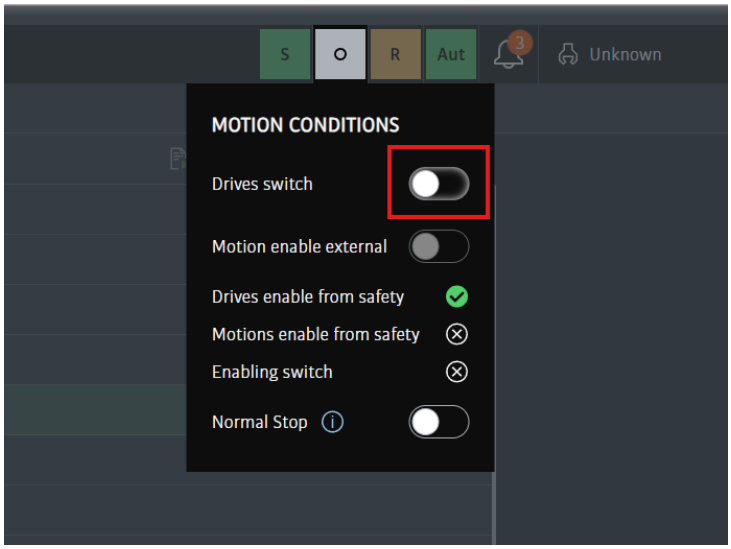


Fig. 22: In automatic mode, you must activate the drives using the switch shown here.

3. Now hold down the green "Play" button on the panel until the message "BCO reached" appears. Then press the "Play" button again.
4. In automatic mode, the loop of `mikado.src` should now be executed. In T1 mode, you must continuously hold down the dead-man switch and the "Play" button.

If the robot program has started successfully, the symbol for the robot interpreter `R` is displayed in green at the top of the panel. If the symbol has a different color (e.g. orange or red), the robot program has not yet started successfully.



Fig. 23: Status bar with the `mikado` program running in automatic mode.

If the symbol is a different color (e.g. orange or red), the robot program has not been started successfully yet.

If you now make, for example, a small movement of axis 6 on the robot page in Mikado, it should be executed by the KUKA robot.

6.4.6 Optional: Execute native robot program

Native programs can be executed on the controller using the "*Execute robot program*" task.

If a new program named "XYZ" is to be added, the program should be saved in the file `KRC/R1/Program/prog_<NAME>.src` with a function name of the same name (`prog_<NAME>`). Afterwards, the program must be added to the file `C:/KRC/ROBOTER/KRC/R1/Program/run_program.src` in the `run_program_impl` section. There, copy and adapt either Variant A for functions with arguments or Variant B for procedures without arguments from the comment block.

Hint: By default, the template results in a failure being reported back. Change the line `result.success = false` to `true` to report a successful execution to Mikado. With a semicolon `;` you can also comment out the error

message `result.errorMsg`, or you can delete the line completely.

Hint: If you execute a movement in the native program, the program must be ended with a pre-run stop. This can be triggered in KUKA by a `wait for true`. This is the only way Mikado can plan subsequent movements with the correct starting position.

6.4.7 Transfer robot control

In some applications, it may be desirable for robot movements to be controlled partly by Mikado and partly by a robot program. This means that Mikado only executes part of the robot's entire movement sequence and then transfers control so that other sequences can be executed on the robot, for example, to move to positions taught on the robot.

The robot must be in a standstill during each control transfer!

Transfer control to mikado

In order for Mikado to execute movement commands, the function `mikado.src/mikado()` must be executed in the KRC robot interpreter. This function starts an endless loop on the robot controller, which waits for new movement commands from Mikado and executes them.

Mikado controls the robot while this function is being executed.

Transfer control to the robot

To transfer control from Mikado to the robot controller, the following function can be called at the desired point in the Mikado sequence using a *JavaScript* task.

```
robot.triggerRobotService("/give_up_robot_control")
```

This call interrupts the above-mentioned infinite loop within `mikado.src/mikado()` and exits the function. The robot can then continue as desired.

6.4.8 Speed and fine-tuning

The speed at which movements are executed can be set via a number of parameters. In addition, the speed can be set on the KUKA controller/teach pendant, which is then multiplied by the Mikado values.

In general, the following parameters can be set in `config/manipulator.yaml` for all trips:

```
velocity_scaling: 0.5
max_velocity_scaling: 0.7
acceleration_scaling: 0.5
max_acceleration_scaling: 0.7

#rescaling in kuka driver
min_velocity_clipping: 0.1
max_velocity_clipping: 100.0
velocity_scaling_factor: 2.5
```

The parameter `velocity_scaling` scales the maximum speed specified in the URDF. If this is adjustable in Mikado, `max_velocity_scaling` sets the upper limit for the user. The value is between 0 and 1, which corresponds to the full speed or 100% `acceleration_scaling` sets the corresponding acceleration factor.

For the Cartesian (linear) part of a handle, slower movements can be defined separately in the individual `.product` files in the `products` folder. These also act as factors and therefore work in combination with velocity_scaling. `ApproachVelocityScaling: 0.5` specifies the velocity factor before gripping and `RetractVelocityScaling: 0.3` after gripping when lifting. To prevent Mikado from overwriting the changes, it is best to close it before adjusting the values in these `.product` files.

Based on the maximum speed in the URDF or `joints_limits.yaml`, the above factors and the possible accelerations, Mikado now calculates how long the individual sections of the journey will take. From this duration, a percentage of the maximum speed is recalculated for the KUKA. Unfortunately, this calculation generally leads to lower speeds than expected, so the result can be increased with `velocity_scaling_factor`, `max_velocity_clipping` sets an upper limit, `min_velocity_clipping` a lower one.

6.4.9 Troubleshooting

Timeouts and connection losses during high network usage

During high load on the network interface used by Mikado caused by additional communication, timeouts and connection losses caused by missing reply messages from the robot can occur.

As a real-time protocol, Profinet communication generally has a higher priority on the robot controller than the Ethernet KRL interface used for communication with Mikado. In most applications, this is not a problem.

However, if both Profinet and Mikado (Ethernet KRL) communicate via the KLI interface, a very high Profinet communication volume can ultimately result in the Ethernet KRL interface no longer being able to send or receive messages as expected due to its lower priority.

If this happens, it in turn leads to timeouts and connection interruptions on the Mikado side. These usually manifest themselves in the form of error messages of the following type:

```
[kuka-driver][communication][...][read] reading timed out after 300 ms!
[kuka-driver][keep_alive][...] reading timed out!
```

This problem seems to occur particularly with very short cycle times (transmission cycle) for the Profinet protocol.

Example: A Profinet transmission cycle of 2 ms with a KR C5 controller can place such a heavy load on the robot's network interface that the described problem occurs.

There are two possible solutions to avoid this situation:

- Use the KONI interface for Ethernet KRL communication to completely separate Profinet and Ethernet KRL from each other.
- Reduce Profinet communication as much as possible to relieve the load on the robot's network interface (e.g. by slowing down the Profinet cycle time).

6.5 Set up Mitsubishi Controller

Tested with CR75x-D controller, software version S7 17-02-28.

6.5.1 Network configuration

First, an Ethernet connection must be established. To do this, connect the Mitsubishi robot controller from its Ethernet port to a Mikado controller Ethernet port. A new connection should now be displayed in the menu bar at the bottom right of the Mikado controller. In the case of a direct connection, this must now be set to a fixed IP and subnet mask. How a Mikado controller receives a fixed IP is explained in section *Network adapter*.

The IP configuration must now also be adjusted on the robot controller side. To do this, open the *Parameters* sub-item of the main menu on the teach panel.

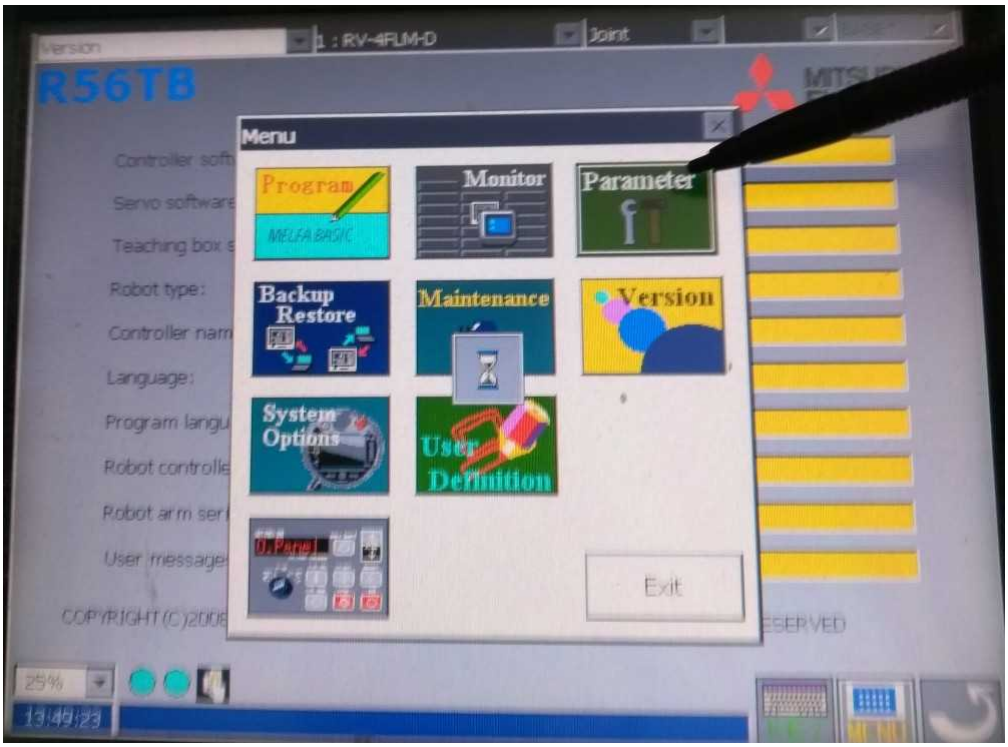


Fig. 24: Image Mitsubishi parameters Open page

Select the Ethernet menu here.

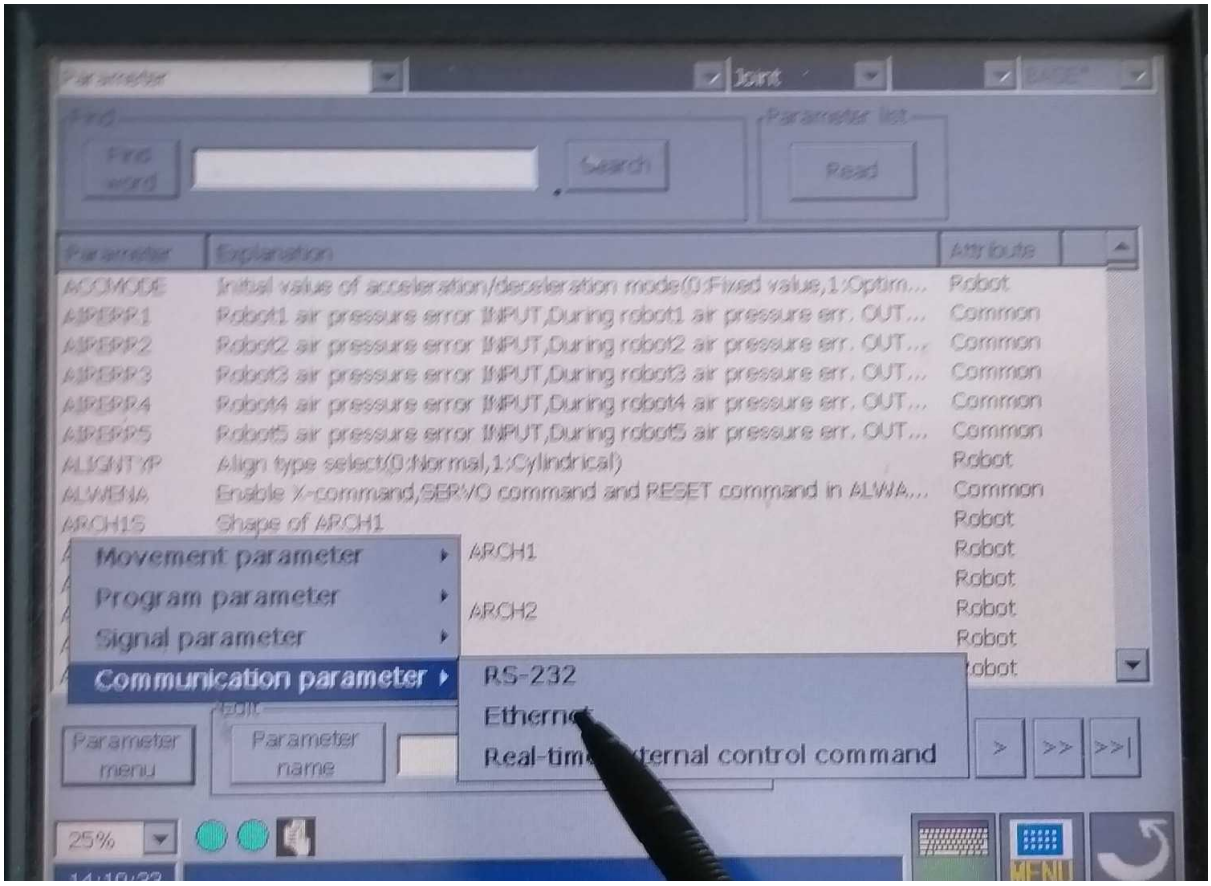


Fig. 25: Bild Mitsubishi TP Ethernet

In the view that now appears, you can set the IP address and the subnet mask.

Hint: An IP address in the same address space as the Mikado controller must be selected for the robot controller and the subnet masks must be identical.

In addition, the ports must be set as described below so that a data connection can be established as expected.

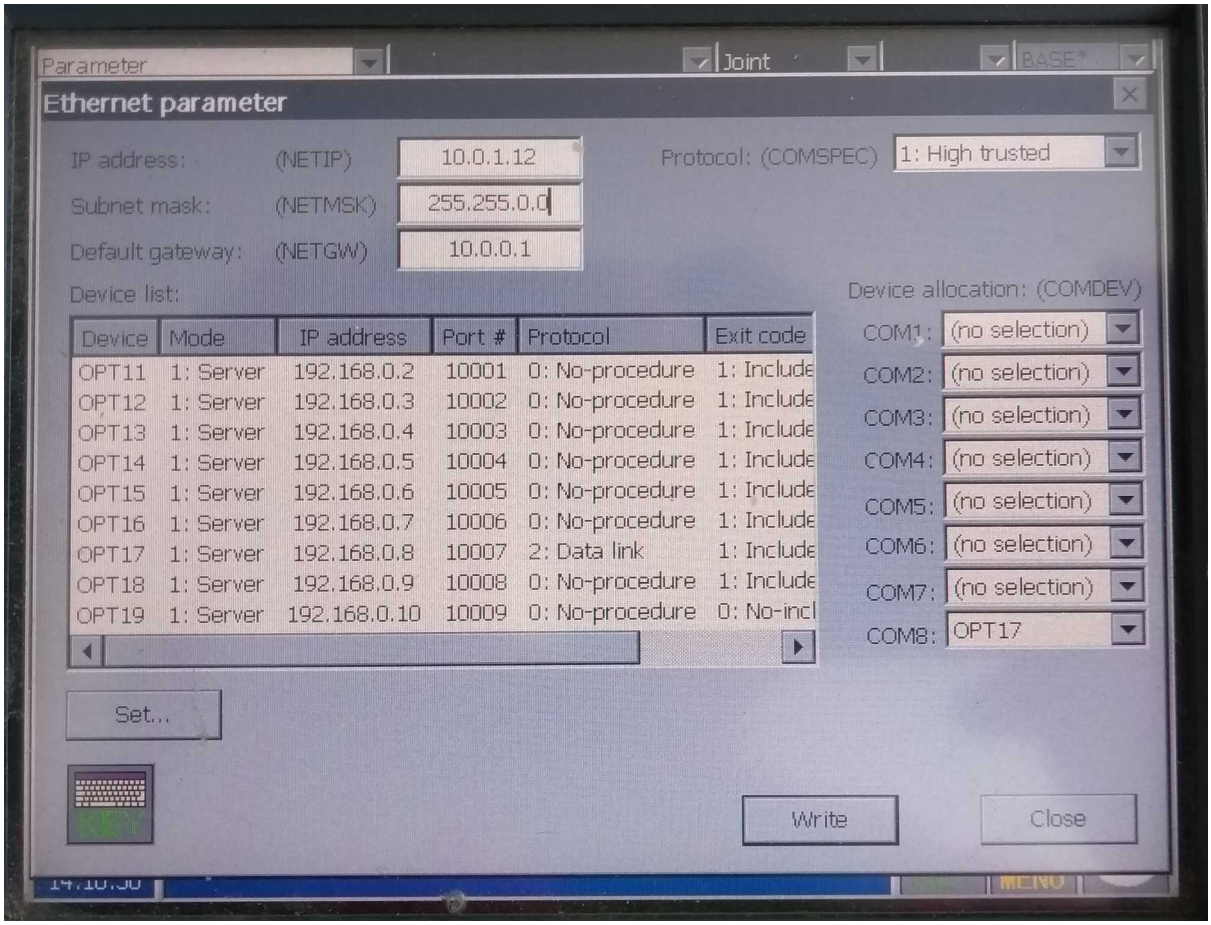


Fig. 26: Image Mitsubishi example IP configuration

Select the OPT17 device or port 10007 and press the **Set…** button. A selection menu appears, for which you accept the following values:

Name	**Value**
Mode	Server
Port #	10007
Protocol	Data link
Packet type	CR
Allocation	COM8

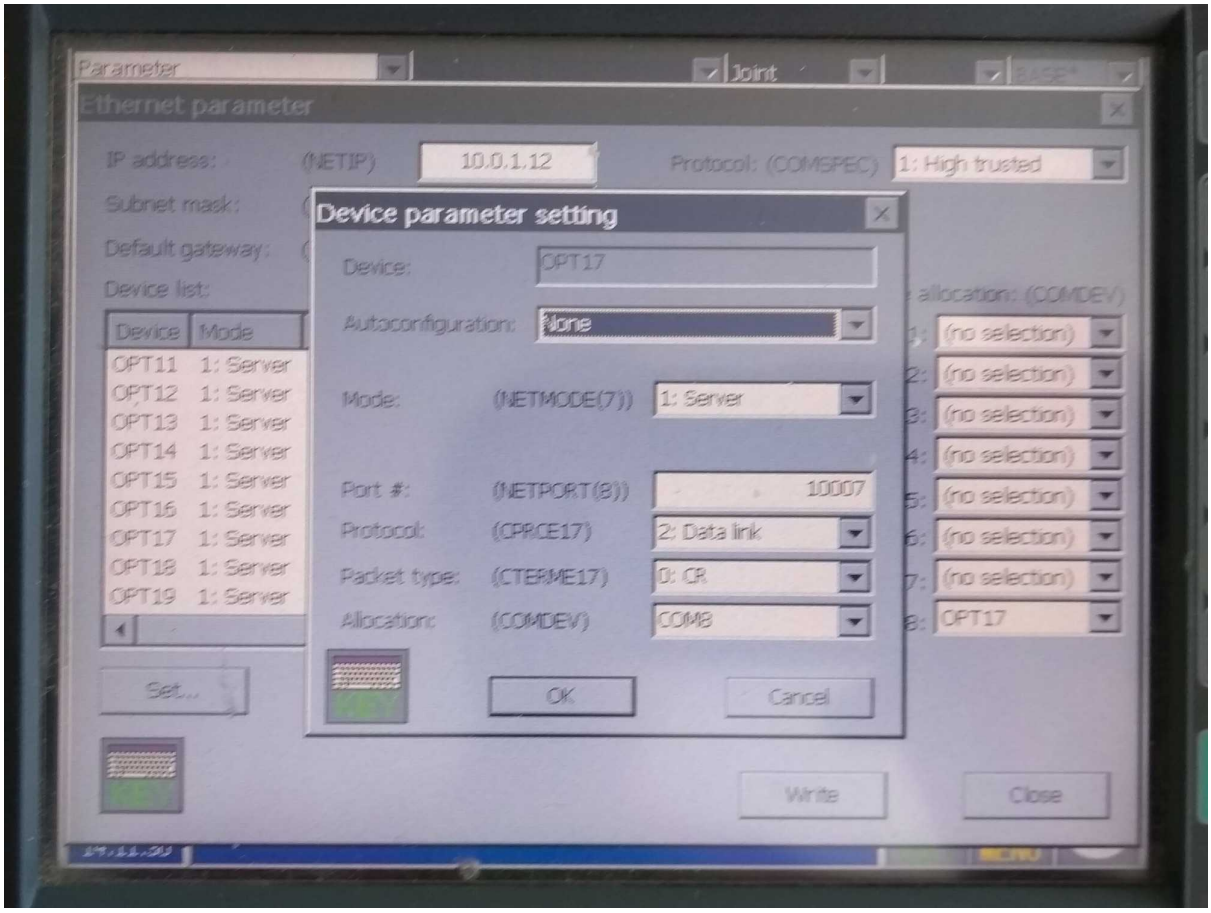


Fig. 27: Image Mitsubishi example port configuration

No role assignment is necessary for all other ports, or existing ones should be switched off by setting the *Protocol* parameter to the value *No procedure*:

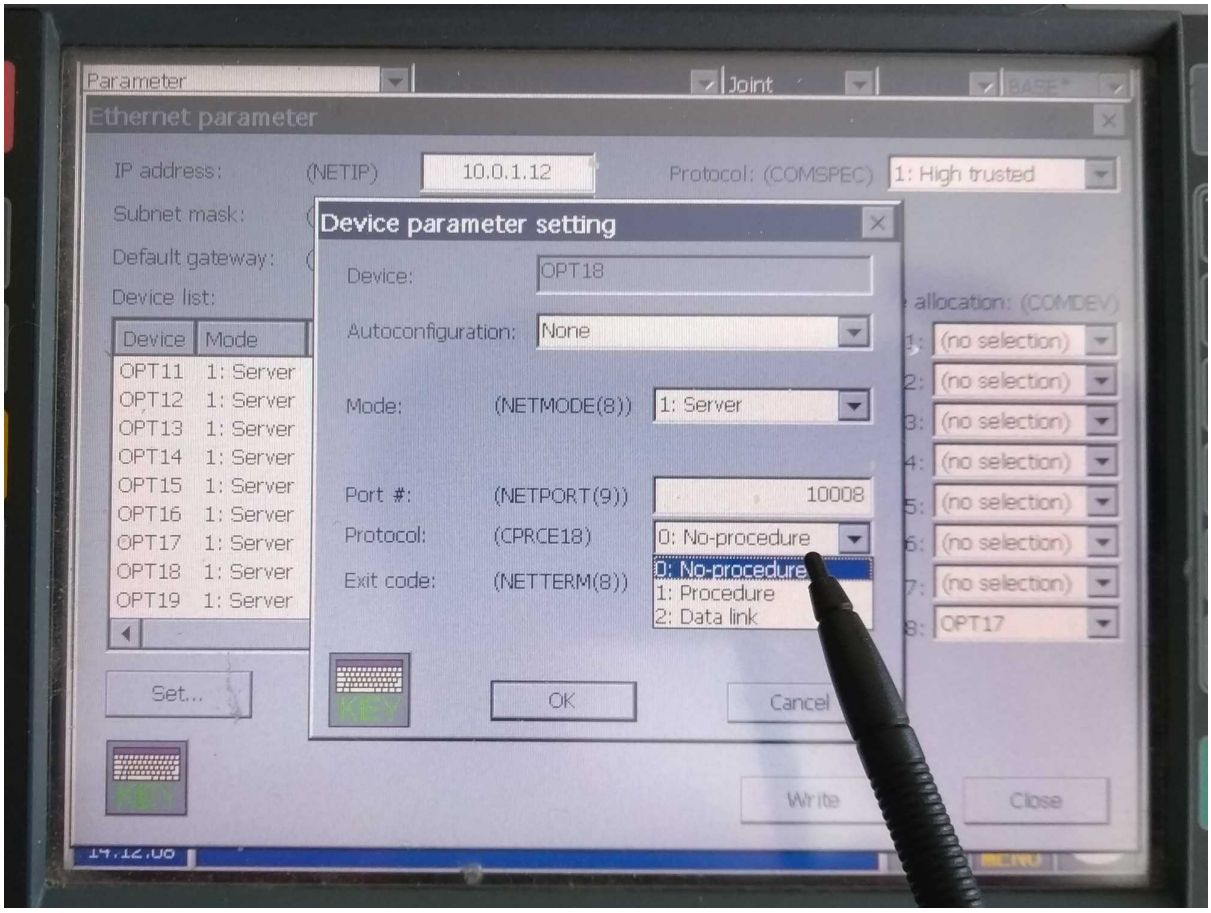


Fig. 28: Image Mitsubishi example Set port neutral

The changes must now be overwritten by the teach panel on the robot controller.

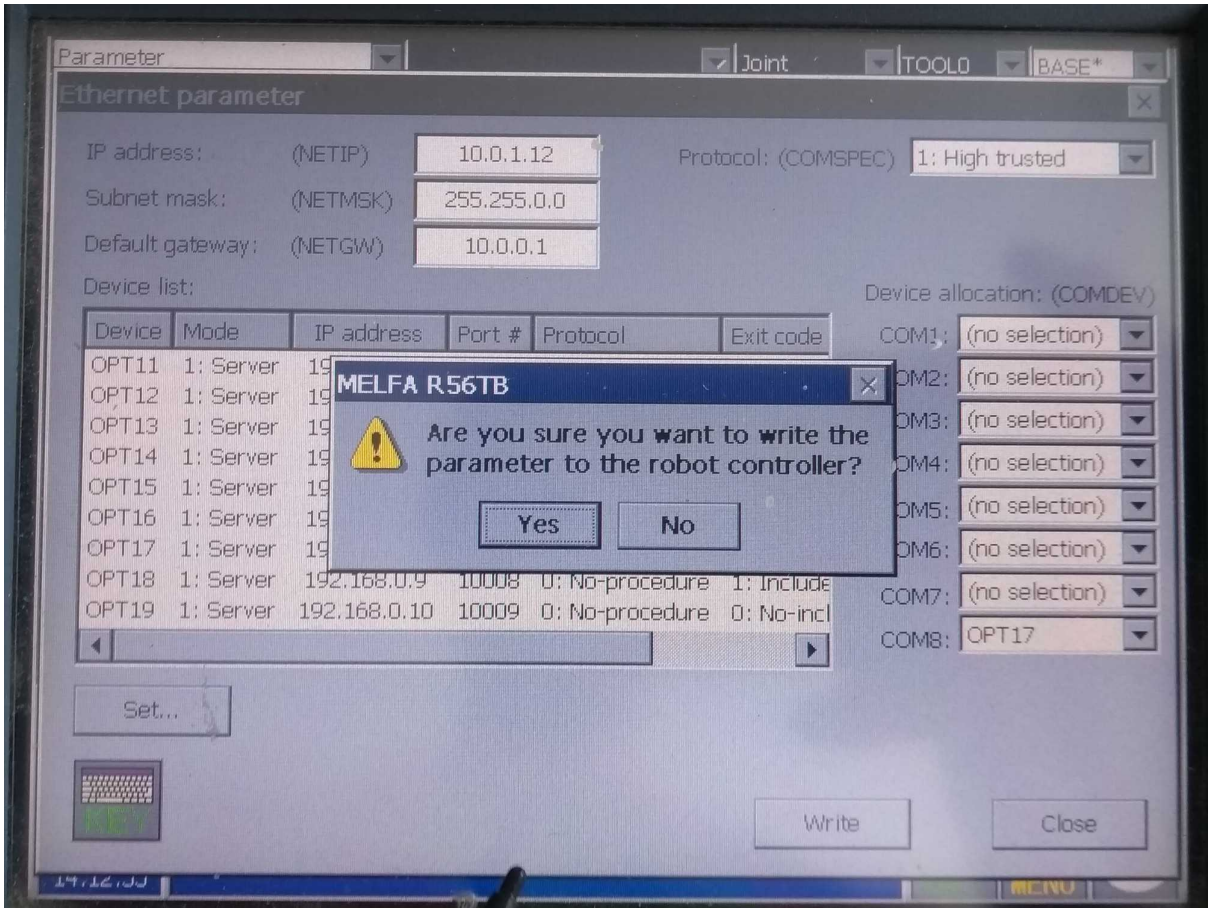


Fig. 29: Image Mitsubishi Parameter Write

The controller must then be restarted.

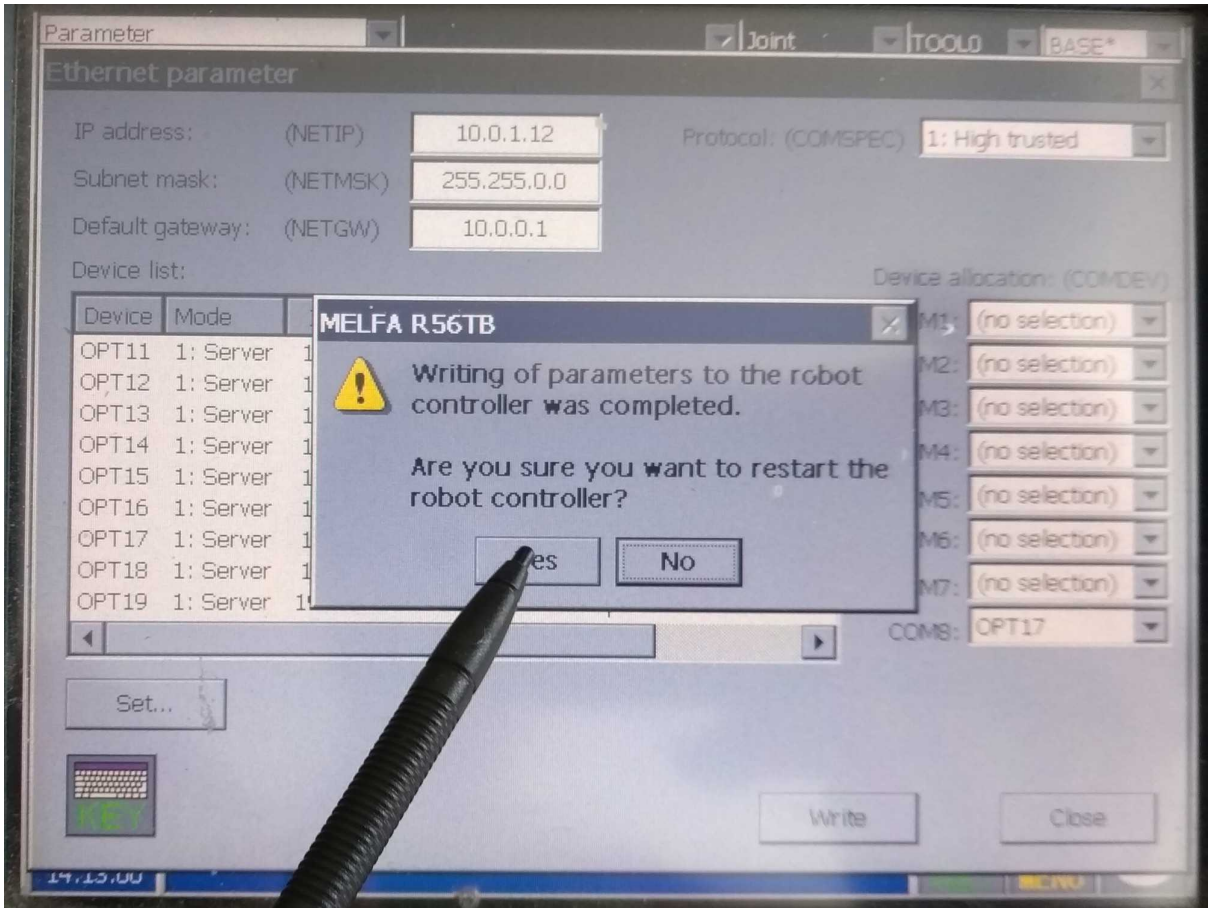


Fig. 30: Image Mitsubishi Controller restart

Now enter the selected IP address of the robot controller in the Mikado Assistant under “Parameters” -> “Robot” in the corresponding field and save the settings.

6.5.2 Configure connection

The Mikado controller expects explicit line termination to be switched on. You can check or set this via the parameter menu. Enter the name *NETTERM* here and select *Read*.

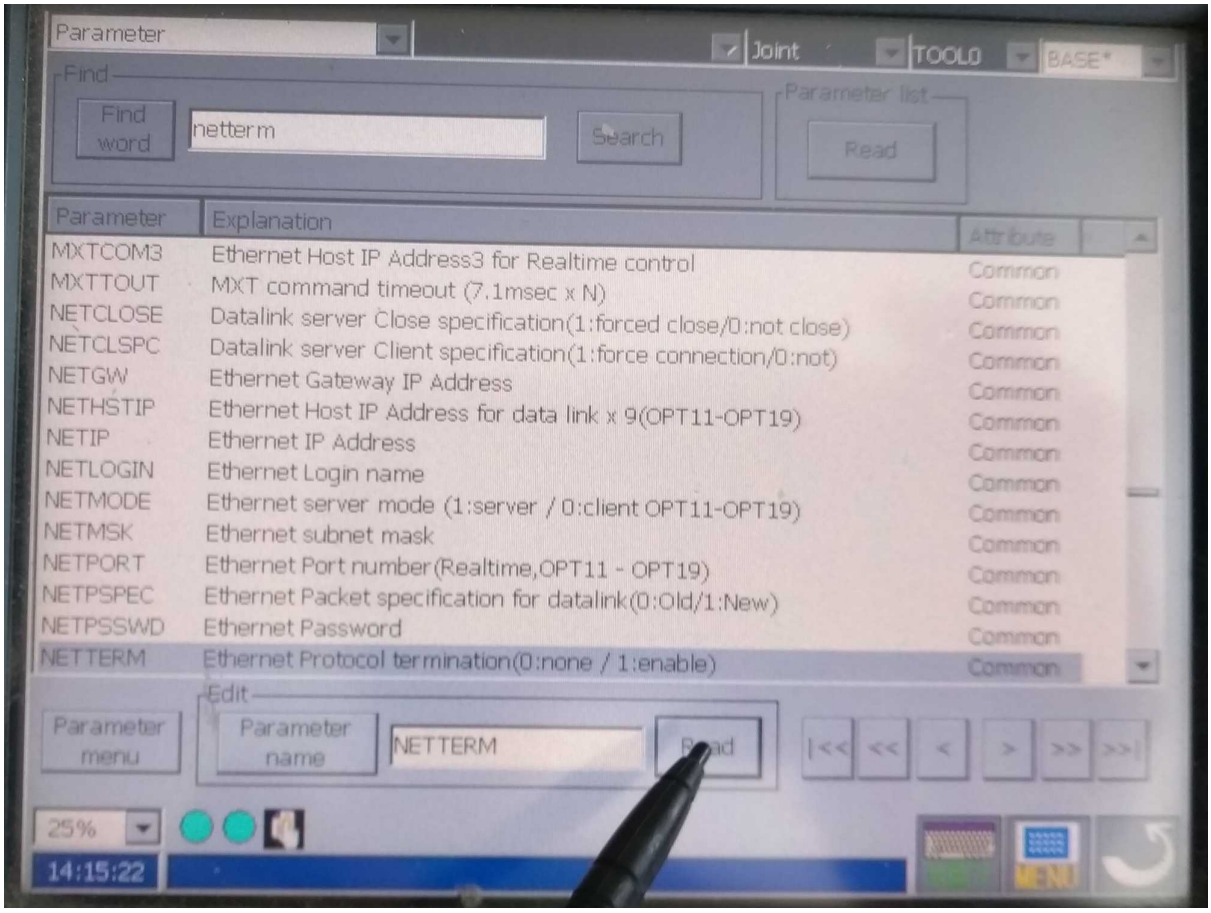


Fig. 31: Image Find Mitsubishi Netterm parameters

You will then see the current setting in the *Parameter edit* dialog. Bits 1-8 should be set to 1. **Otherwise, the values must be manually set to 1 and written to the controller.**

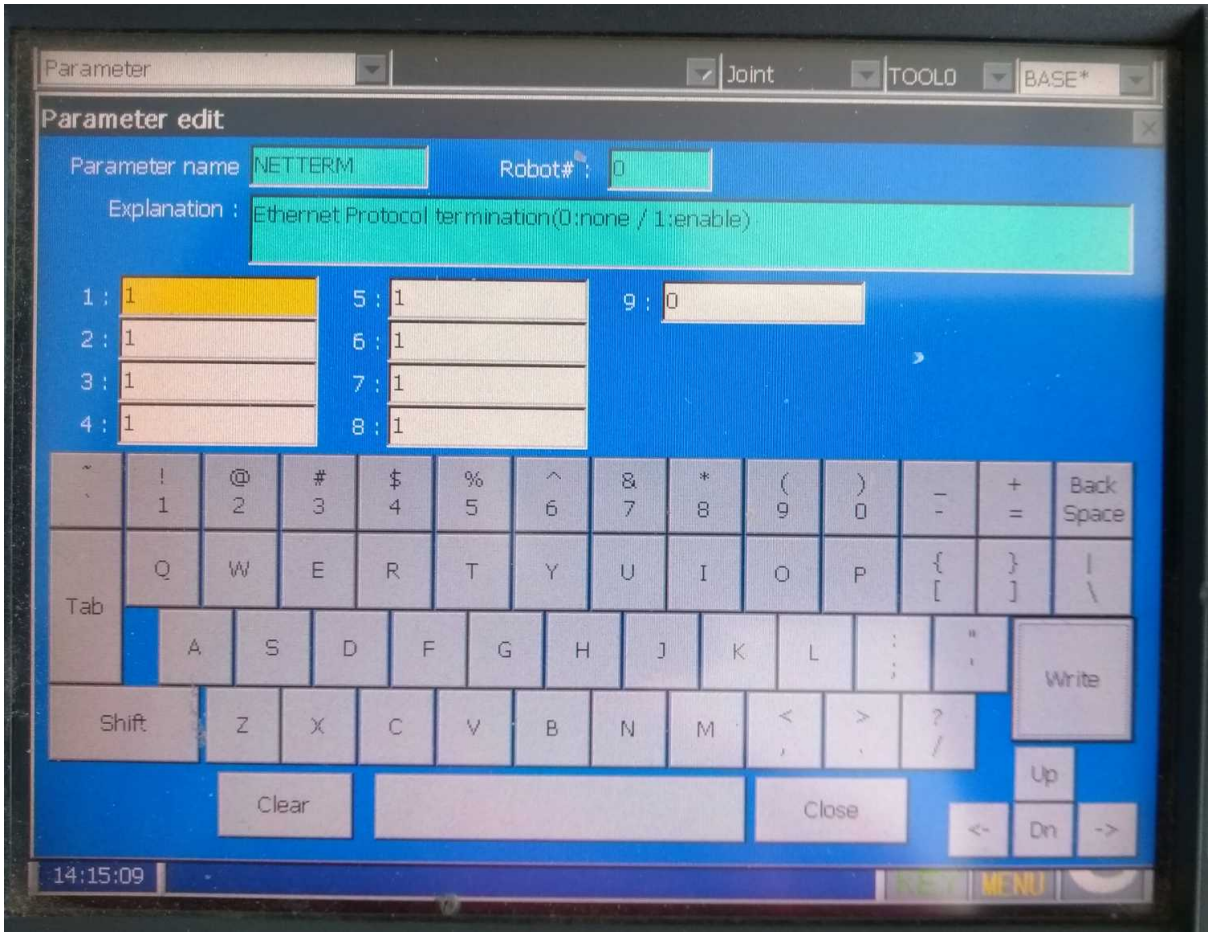


Fig. 32: Bild Mitsubishi Netterm Parameter editieren

6.5.3 Configure programs

During runtime, the Mikado controller runs a program on the robot controller that takes over control. This program must not be blocked by other programs running from the start.

Open the *Slot table parameter* dialog from the parameter menu.

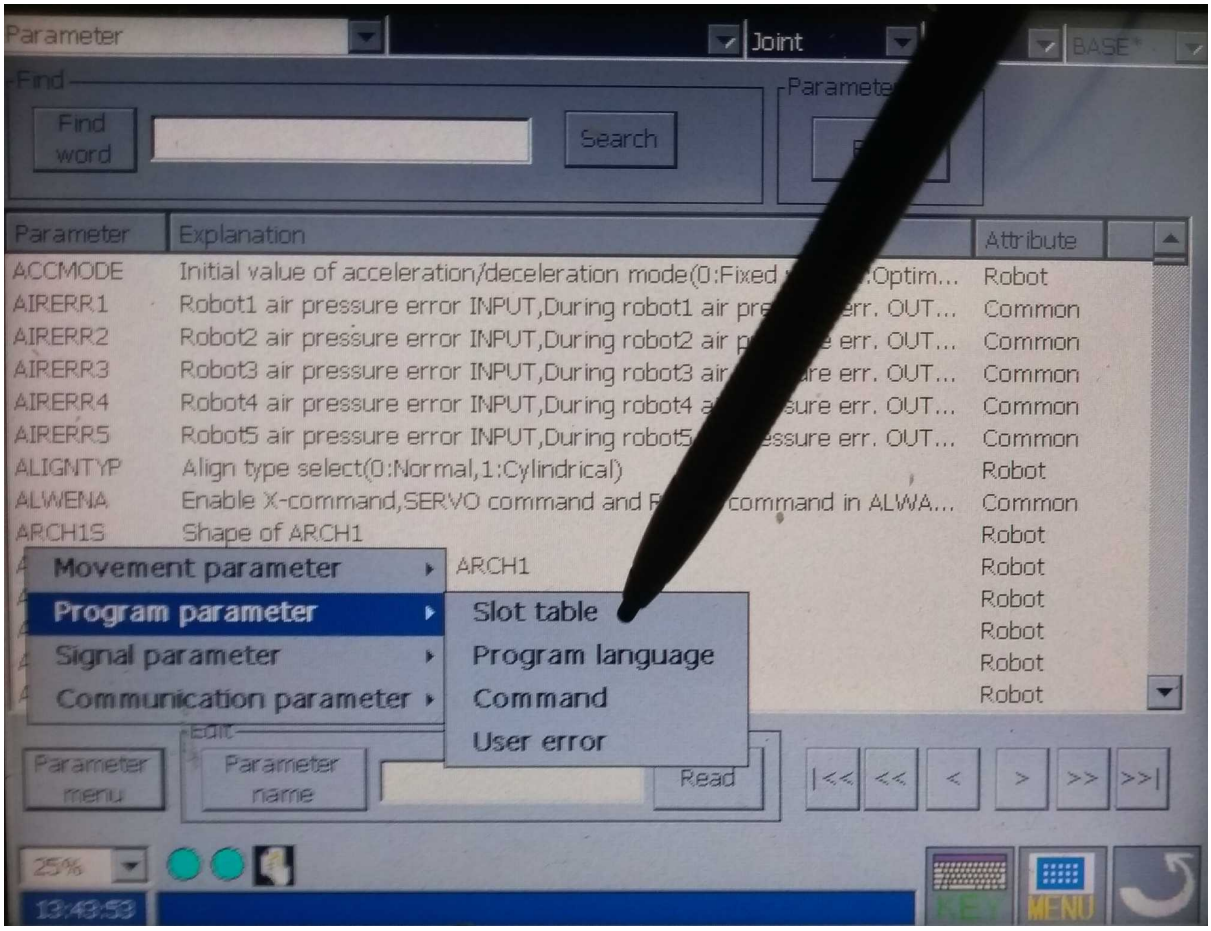


Fig. 33: Image Open Mitsubishi Program Slot dialog

If necessary, remove programs from the automatic start and restart the controller.

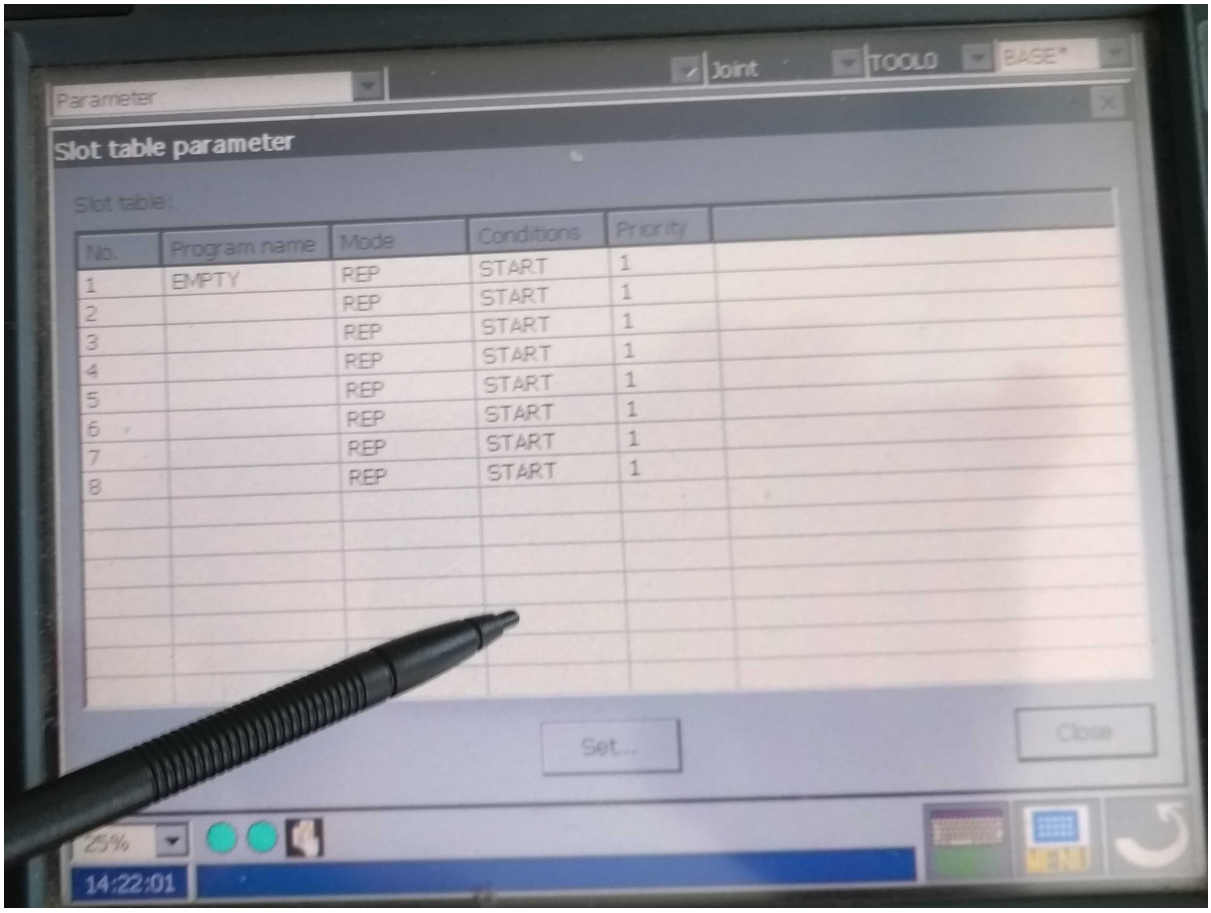


Fig. 34: Bild Mitsubishi Controller Slot table parameter

The Mikado controller requires sufficient free memory space on the robot controller. To do this, select *Program list* from the selection menu.

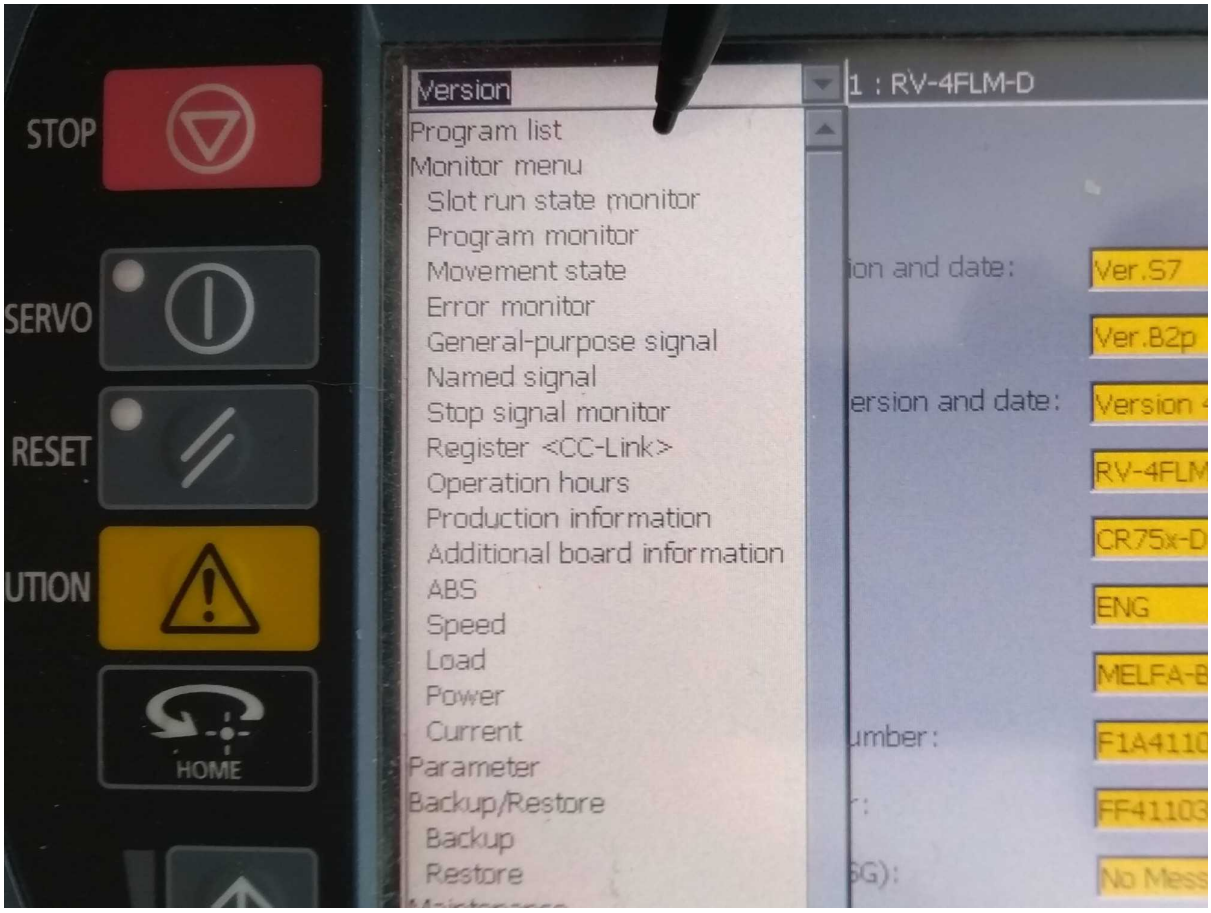


Fig. 35: Image Select Mitsubishi Controller Program List

A program overview opens showing the currently available programs and the free memory space.

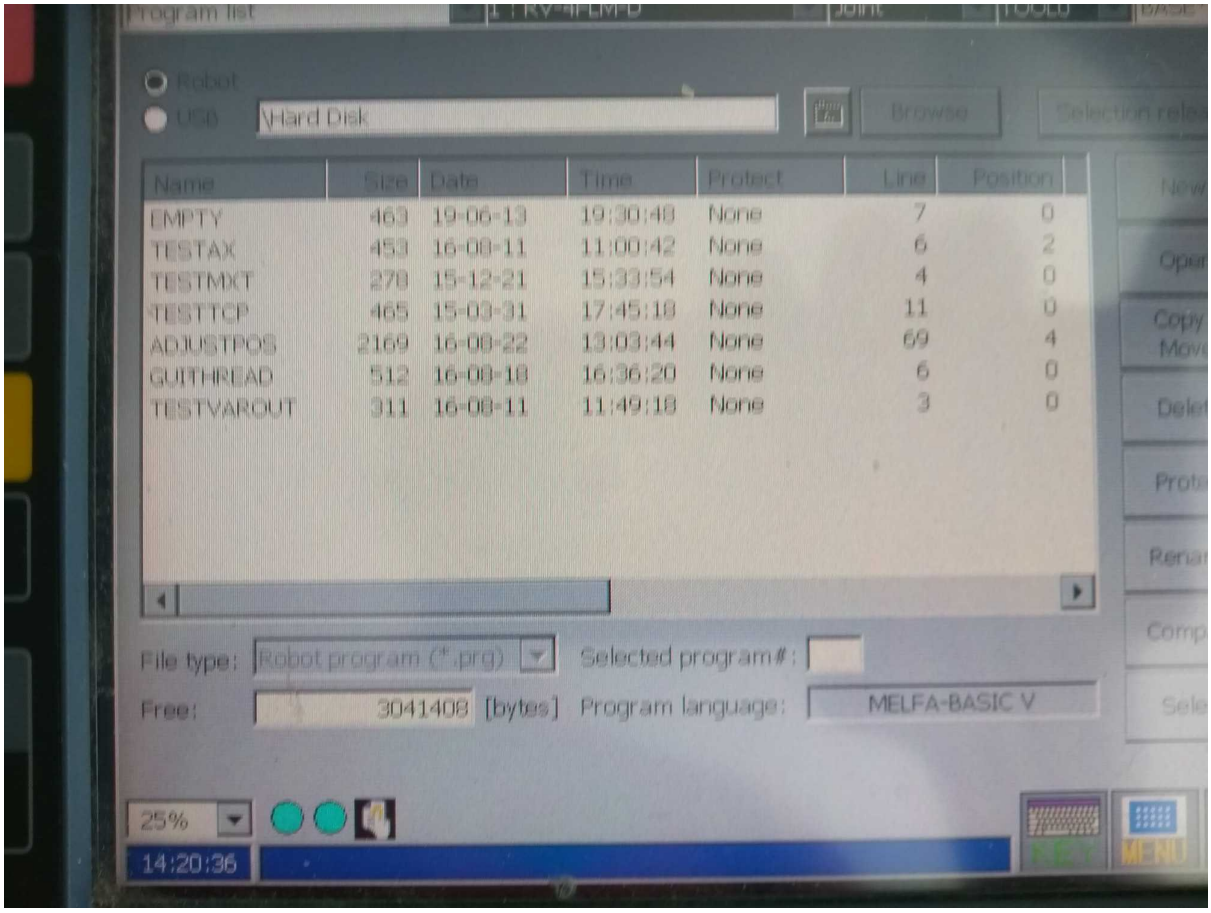


Fig. 36: Image Mitsubishi Controller Memory

6.5.4 Troubleshooting: Verbindung testen

Troubleshooting if there is no connection to the robot after configuration and restart. In the *Status bar* area, you can see whether the connection to the robot has been successfully established.

- Check: Robot controller and Mikado controller have the same subnet mask and different fixed IPs in the same address space
- Check: The status LED of the network port does not indicate an error
- Check: Can the robot be reached via *ping* under the entered IP
- Check: Is the robot capable of communicating via R3 protocol
- Check: Is a carriage return sent with R3 communication

The necessary tools for executing the Ping command and sending R3 commands from the Mikado controller:

A console must be started on the Mikado controller. The ping command to be entered is then, for example:

```
ping 10.0.0.4
```

Pinging can be canceled by pressing *Ctrl* + *C*. A positive response would look like this:

```
ping 10.0.0.4
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=1.18 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.448 ms
^C
```

(continues on next page)

(continued from previous page)

```
--- 10.0.0.4 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.448/0.816/1.185/0.369 ms
```

To send R3 commands, open a *telnet* connection. A connection test runs via port 10002 and should return *QOk* as the response:

```
telnet [Roboter IP] 10002
```

Commands can then be sent directly to the robot via port 10003:

```
telnet [Roboter IP] 10003
```

For example, the speed factor could be read as:

```
1;1;OVRD
```

For example, the robot hand / gripper could be controlled with the following commands:

```
1;1;CNTLON
1;1;HNDON1
```

The controller's responses must always be completed with a carriage return, i.e. the cursor must move to the beginning of the new line.

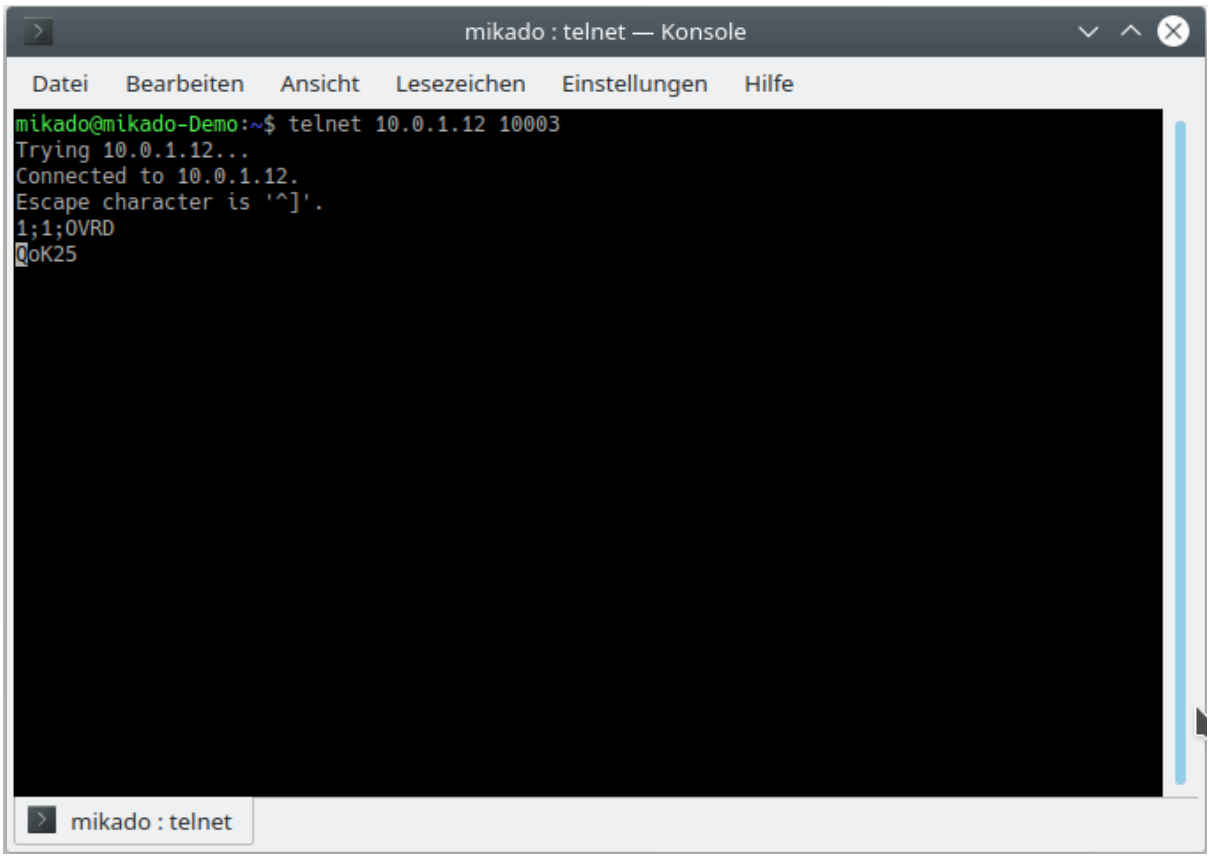


Fig. 37: Image Mitsubishi Telnet OVRD Read parameters

6.5.5 Optional: Manual control

If the gripper is to be controlled via robot I/O, this must be adapted or tested. The hand I/Os can be used for this purpose. If you have connected your gripper to hand I/O **Port 1**, for example, enter the corresponding I/O numbers in the gripper configuration (*gripper.yaml*). At Mitsubishi, the manual I/O numbers are in the range 900-907, both for inputs and outputs. Alternatively, you can of course also use the general I/Os of the Mitsubishi controller (these usually start at index 0, see documentation of the robot controller used).

Extract from the example configuration for a gripper on 'Hand Port 1':

```
open_hand_signal_number: 900
close_hand_signal_number: 901
is_closed_signal_number: 900
is_open_signal_number: 901
```

You can now check on the teach pendant whether the gripper actions are triggered. To do this, open the manual menu (separate button at the top right of the teach panel) and press the Open and Close buttons, i.e. **+** and **-**. Make sure that the function is not inverted, i.e. that Open actually triggers an opening and not a closing.

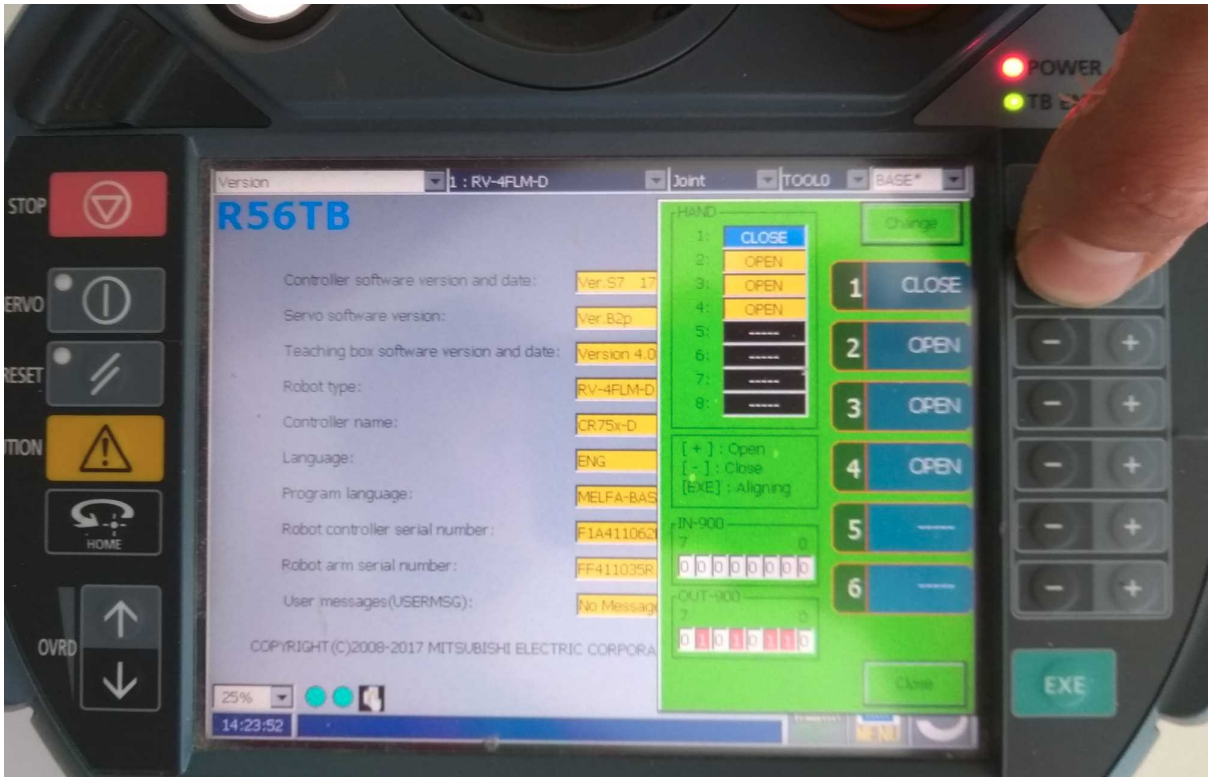


Fig. 38: Check picture Mitsubishi hand

6.5.6 Optional: Adjust base coordinate system

With *Create a new cell*, Mikado offers the option of setting the cell coordinate system independently of that of the robot. This is useful for overhead assembly, for example, in order to obtain an intuitive world coordinate system. In order to display poses in the same coordinate system on the teach panel, the origin of the robot must also be adjusted accordingly. This is not absolutely necessary, but facilitates the integration and teaching of poses. With the Mitsubishi robot controller, adjust the robot coordinate system as follows. Select the item *Movement Parameter* and then the sub-item *Tool*.

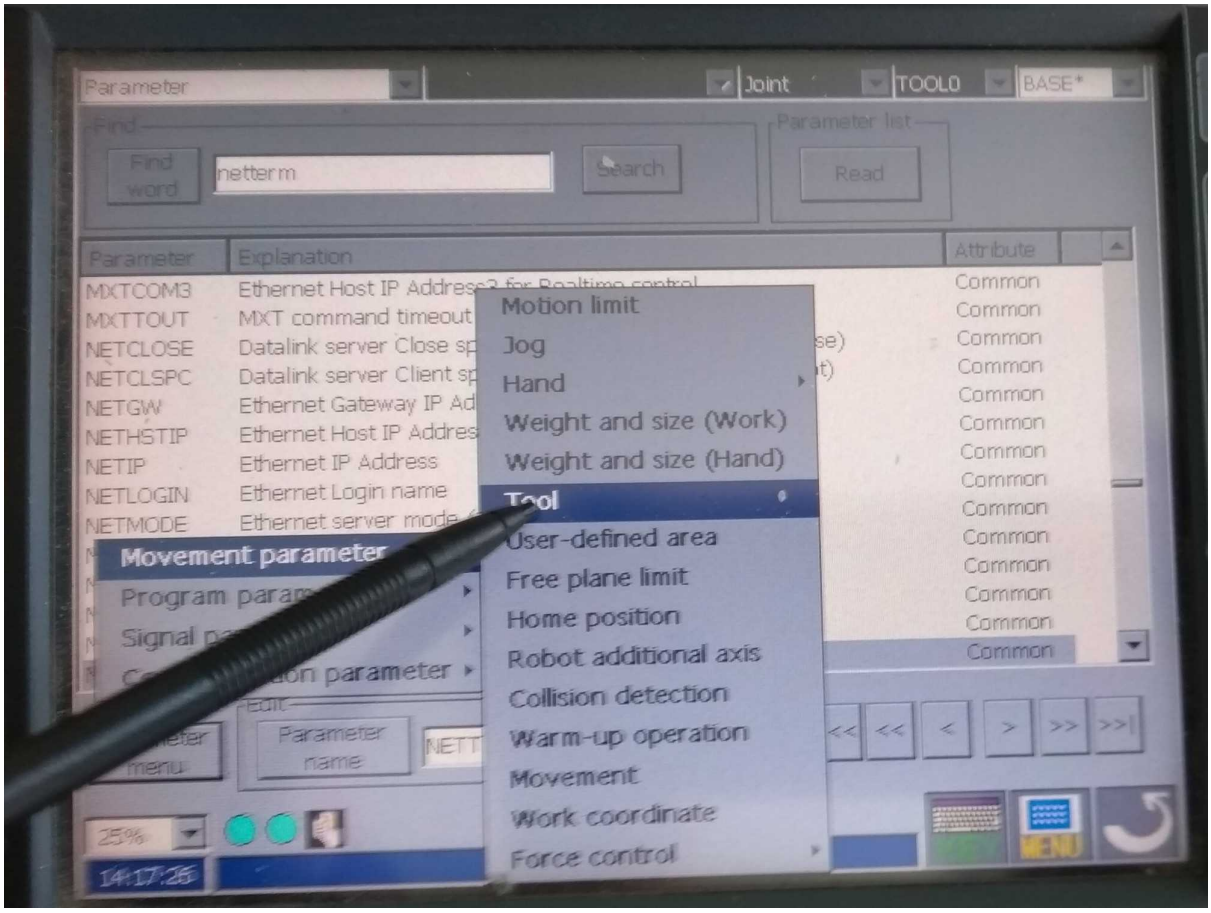


Fig. 39: Image Mitsubishi menu coordinate system

In the following dialog, you can then change the origin of the robot accordingly. In the example shown, the base is rotated by 90° around the z-axis.

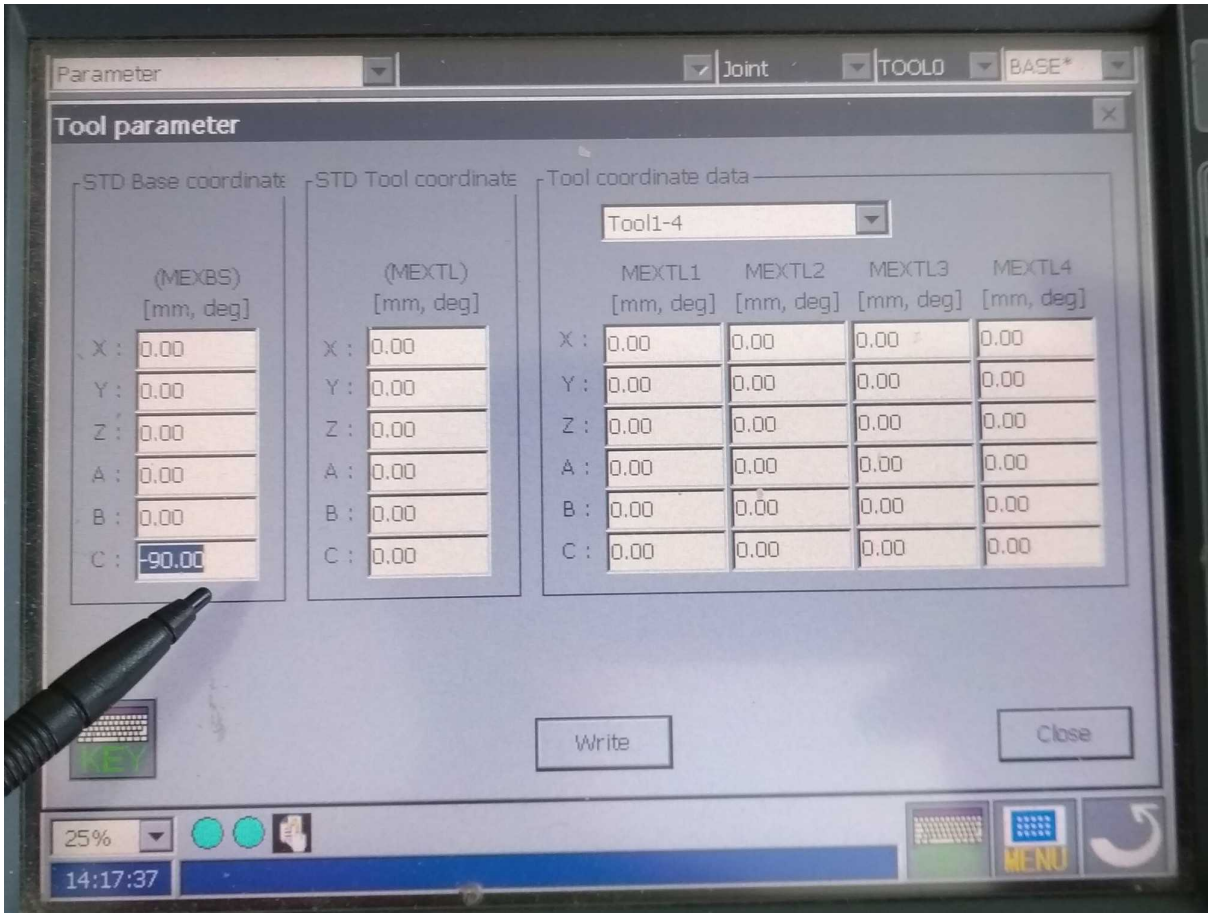


Fig. 40: Image Mitsubishi Dialog Adjust coordinate system

The changes are applied using the *Write* command. The correctness can be easily checked on the Mikado robot page, which always displays the currently read robot pose. It should now match the one displayed in the teach panel.

6.6 Using Mitsubishi robots

6.6.1 Parameters and controls

- **Speed factor**:

  Set speed for robot movements. Clicking on *Apply* sets the set value on the robot controller. Please note that maximum values are stored in the *System configuration*, so that you may not be able to exceed a limit value.

- **Reclaim robot control**:

  Robot control can be withdrawn from Mikado Control under certain circumstances (e.g. if you switch to manual mode on the robot). Press this button to transfer control back to Mikado Control.

- **Send to robot**:

  Mitsubshi-specific R3 commands can be entered in the text field next to this button and sent to the robot. The response message is then displayed in raw form.

  You can use this control element to send any commands to the robot directly via the user interface. For example, you can control the speed of the robot or address specific robot IOs. Further information on R3 and the individual commands can be found in your robot documentation.

6.6.2 Turning on the robot

- Mikado and robot can be started at the same time.
- Teach Pendant must be deactivated and the controller must operate in “Automatic” mode

6.6.3 Other notes

- Teaching position:
  - Switch to manual mode.
  - Activate teach pendant and switch on servos.
  - Move to position in jog mode.
  - Click on “Accept current robot pose” in Mikado.
- Values of the current robot pose:
  - The robot pose of the controller may differ from the pose displayed in the Mikado interface. However, this is only relevant if you want to use native robot programs in combination with Mikado.
  - Make sure that the base coordinate system is defined correctly (Mitsubishi Teach Pendant: Parameters → Movement Parameters → Tool). Set the transformation from world coordinates to the robot coordinate system here.
  - x/y/z coordinates may have a slight offset due to factory calibration.

6.7 Set up Nachi Controller

6.7.1 Network configuration

A network connection between the two systems is required for communication between the robot controller and Mikado Controller. To do this, first connect an Ethernet port of the Nachi Controller to an Ethernet port of the Mikado Controller. A fixed IP address and a corresponding subnet mask must then be set for both the Mikado Controller and the robot controller. The network configuration of the Mikado Controller is explained in more detail in section *Network adapter*. We are working on a detailed description of the network configuration of the Nachi controller. For the time being, information on the network configuration of the controller can be found in the manufacturer’s user manual.

Hint: Make sure that the IP addresses of the Mikado Controller and Nachi Controller are in the same address space and that the subnet masks match.

Then enter the IP address of the robot controller in the Mikado Assistant under “Parameters” -> “Robot” in the corresponding field and save the settings.

6.7.2 Preparation

This description was created for a Nachi Controller Ver. `4 63 05-30-2019`. The instructions may be incomplete for other versions. If you have any problems, please contact Mikado Support.

The preparation for transferring the programs and user tasks is described so that it can be implemented with the help of the Teach Panel. First of all, it is recommended to press the `R` key and then enter `314` in the field that opens. The controller then asks for the password for the `Specialist` authorization, which is required for parts of the Nachi setup. Enter `12345` as the password and then press the `Enter` button. On the teach panel, you can go back one step in the menu by pressing the `R` button. Most of the buttons on the teach panel are assigned twice for reasons of space. If `Enable` and another button are pressed at the same time, the green function on the button is

executed. If, for example, the `0` button is pressed together with `Shift`, the result is a + (plus sign) and without a `0` (scalar).



Fig. 41: Photo of a Nachi Teach Panel in which the double assignment of the buttons can be seen

6.7.3 Copying the programs/user tasks

In order for the Nachi robot to be controlled by Mikado, the corresponding programs must be copied to the robot controller and compiled there. You will find the required programs on the Mikado controller under

`/opt/isys-vision/mikado/robot_programs/nachi/`

or directly via the corresponding button in the Mikado Assistant ("Parameters" -> "Robot" -> "Open directory"). Compiling the programs only works if the name of the robot is part of the program name. The example shown here is a `MZ07L-01`. If you have a different robot, please change the name of the robot programs. Now copy the programs from the above-mentioned directory to a USB stick for transfer to the Nachi. There are 2 USB interfaces on the Nachi where data can be transferred. One on the teach panel and one on the controller itself. Experience has shown that the port on the teach panel is unreliable, which is why we recommend transferring data to the controller.

To copy the files, please click on the Teach Panel:

`Service Utilities` → `7 File Manager` → `1 File Copy` → select your USB stick under `Device (src)` and click on the right arrow on the Teach Panel → select the programs on the right side by clicking on `Enter` → `Execute`.

6.7.4 Compiling the programs/user tasks

The driver consists of 2 robot programs that are called one after the other and two user tasks that run in parallel:

- `MZ07L-01-A.008` → Robot program for resetting the variables
- `MZ07L-01-A.012` → Robot program for controlling the robot
- `USERTASK-A.000` → Usertask to receive the data from Mikado
- `USERTASK-A.001` → User task for sending the data to Mikado
- `VAR.INC` → List of required variables

The list with additional variables only needs to be copied, but not compiled. To compile the other files, please proceed as follows:

`Service Utilities` → `15 ASCII File Edit` → `Select/mark file` and open with `Execute` → `Complete` → Confirm the question whether the file should be compiled. If the query does not appear after confirming, you must check whether the name of the file is correct (is the name of the robot correct, is the extension correct, ...).

6.7.5 Setting the priority of the user tasks

The user tasks run in parallel to each other. By changing the priority, the receiving of data can be given a higher priority than the sending of data. As trajectories in Mikado often contain over 100 joint poses, the receiving of data must be assigned a higher priority. This works as follows:

`Service Utilities` → `4 Monitor 2` → `37 User Task` → `1 User Task Monitor`.

The priority of the user tasks must be changed here. To do this, set the priority of `Prog 0` to `6` and that of `Prog 1` to `4`. To do this, the number in the `Priority` field must be marked and can now be changed by clicking the button with the sheet and pen (next to the `FN` button) on the Teach Panel. After changing the priorities, exit the editing mode by clicking the button with the pen and paper again.

If you want to stop a user task, you must press the `Enable` button and `Off` (located on the digit `2`), as the `Off` function is highlighted in green. After starting the controller, user tasks are started automatically.

6.7.6 Starting the robot program

After switching the lever on the teach panel and controller to `Playback` mode, the program can be started. The correct program can be selected by clicking the program button on the teach panel. This can also be used to reset the program after a stop due to an emergency stop, for example. The robot can be switched on by clicking `Enable` and `Motor ON`. Once the correct program has been selected and the robot is energized, the program can be started by simultaneously pressing `Enable` + `Shift` + `GO`. The Nachi can now be controlled by Mikado.

The gripper (or suction cup) is controlled via outputs, which you set in the Mikado cell directory in the file `config/gripper.yaml`:

```
open_hand_signal_number: 23
close_hand_signal_number: 24
```

6.8 Set up Stäubli Controller

The driver can be used with a Stäubli 6-axis robot and the CS8 or CS9 controller. At least VAL 3 version s7.7.2 is required.

6.8.1 Network configuration

A network connection between the two systems is required for communication between the robot controller and Mikado controller. To do this, first connect an Ethernet port of the Stäubli controller to an Ethernet port of the Mikado controller. A fixed IP address and a corresponding subnet mask must then be set for both the Mikado Controller and the robot controller. The network configuration of the Mikado Controller is explained in more detail in section *Network adapter*. We are working on a detailed description of the network configuration of the Stäubli controller. For the time being, information on the network configuration of the controller can be found in the manufacturer's user manual.

Hint: Make sure that the IP addresses of Mikado Controller and Stäubli Controller are in the same address space and that the subnet masks match.

Then enter the IP address of the robot controller in the Mikado Assistant under "Parameters" -> "Robot" in the corresponding field and save the settings.

6.8.2 Installation

In order for the Stäubli robot to be controlled by Mikado, two folders with VAL3 programs must be copied to the Stäubli controller, you will find these programs on the Mikado controller under

`/opt/isys-vision/mikado/robot_programs/staubli`"

or using the corresponding button in the Mikado Assistant ("Parameters" -> "Robot" -> "Open directory"). Either a USB stick (< 2GB for CS8), an FTP connection or the transfer manager in Stäubli Robotics Studio can be used for copying.

6.8.3 Set up server sockets

Two sockets must now be set up, the procedure differs slightly between CS8 and CS9:

For *CS8*, from the main menu, select Control panel –> I/O –> Socket –> TCP Servers

For *CS9*, select from Home IO –> Socket –> TCP Servers –> "+"

Name	Port	Timeout	End of string	Nagle
Feedback	11002	-1	13	Off
Motion	11000	-1	13	Off

6.8.4 Start

The driver can now be started; the procedure here also differs slightly between CS8 and CS9:

In CS8, press the *Start/Run* button, select *ros_server* and confirm with F8 (Ok).

For CS9, press VAL# –> Memory –> Select *ros_server* –> Play

6.8.5 Set up grippers and IOs

Grippers and IOs can be written to Stäubli via Modbus/TCP or via a fieldbus connection (e.g. ProfiNet), please consult the Stäubli documentation for setup on the Stäubli side and contact us for details for setup on the Mikado side.

6.9 Set up Techman Controller

The Techman Controller can be set up in just a few steps. During operation, there are a few details to consider for native programs, for example. The following configuration was tested with controller software version 1.72, 1.76 und 2.14.

6.9.1 Network configuration

First, an Ethernet connection must be established between the robot controller and the Mikado controller. This must be done on port 1* on the Techman Controller, as it only works with the Listen Node. A fixed IP address and a corresponding subnet mask must then be set for both the Mikado Controller and the robot controller. The network configuration of the Mikado Controller is explained in more detail in section *Network adapter*. For the Techman Controller, you can adjust the TCP/IP settings under the menu item *System*, then *Network*. Further information on the network configuration of the controller can be found in the manufacturer's user manual.

Hint: Make sure that the IP addresses of the Mikado Controller and Techman Controller are in the same address space and that the subnet masks match.

Hint: Port 1 corresponds to *Local Area Connection 9* on the network side. The robot controller should be accessible from the Mikado controller via *ping* if communication is successful.

6.9.2 Establish Modbus connection

The current Modbus status can be called up and set under the menu item *Settings*, then *Connection*. The IP address of the Mikado controller must be permitted on the Techman controller and the Modbus status must be activated. The following steps are necessary for this:

- The Modbus status must first be deactivated.
- Then add the IP address of the Mikado controller with activated *writer permission* under *IP filter*.
- And finally reactivate the Modbus status. The *Status* should be followed by *Enable*.

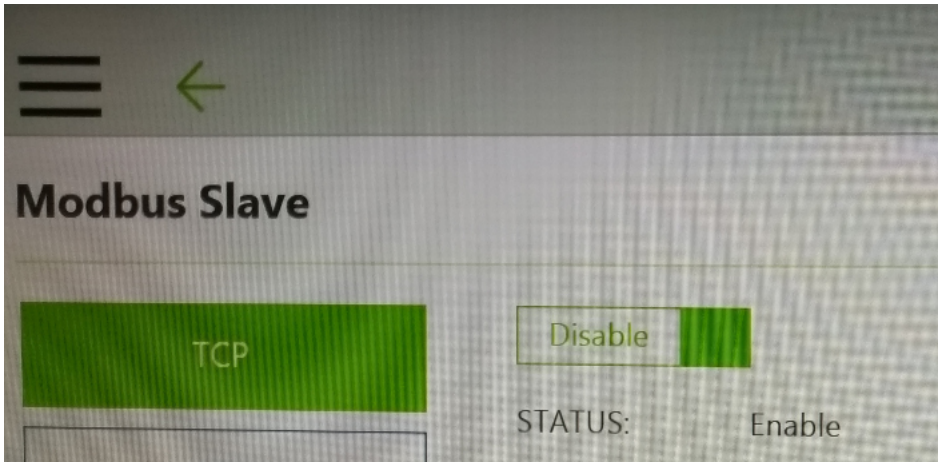


Fig. 42: Figure: Modbus Enable setting

Hint: The console can be opened in the top right-hand corner. If you deactivate and reactivate the Modbus server, its current IP is also written there. After changing the TCP/IP connection settings, the Modbus server must also be restarted.

6.9.3 Create project

Mikado requires a project flow with Listen Node in order to be able to communicate at runtime. This must be active so that Mikado can execute robot movements. However, custom programs can also be executed (see below). A minimal flow would consist of *Start*, *Listen* and *Stop* nodes. A somewhat more complex sequence is shown below.

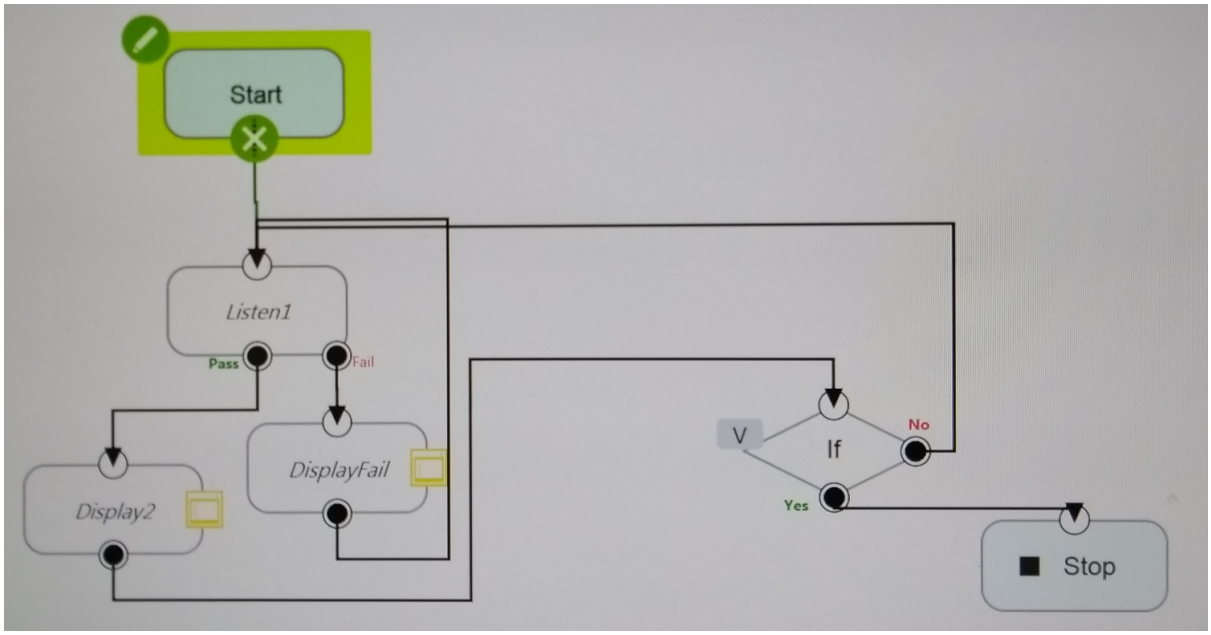


Fig. 43: Listen Node with debug display and possibility to return

The process allows debug messages to be shown to the user via display nodes. More important, however, is the process via an *if* node. This tests the state of the *var_stop* variable and decides where the program continues. The variable can be written from the Mikado program, see *Execute robot program* and thus the Techman flow can be controlled.

6.9.4 Test project

Via *Diagnosis*, then *Stepping* the program/flow can be started as a test and run through step by step. The TCP/IP connection with address should also be displayed in the console as soon as the Listen Node is reached. From this point on, the connection to Mikado should also work. I.e. if you now start a Mikado cell or if it is already running, the status light for the robot should turn green.

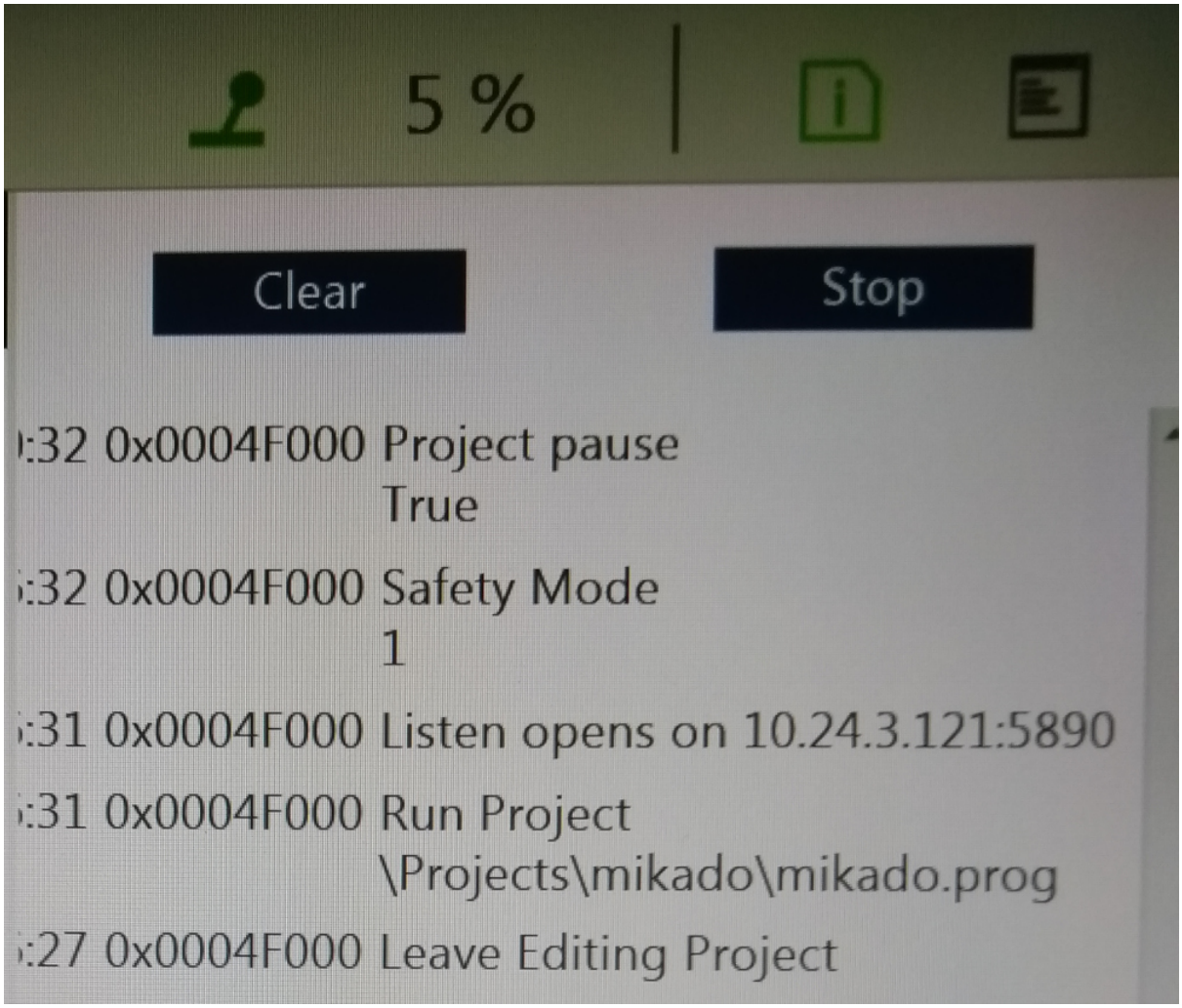


Fig. 44: Listen Node in the console

6.9.5 Set project speed and activate automatic mode

All executable projects can be viewed in the “Run Setting”. The **current** display in the file means that this project is currently being executed, the **tested** display means that this project may be executed in automatic mode.

If the project is to be executed in automatic mode, it must pass the test run. The test run is carried out in manual mode. The following steps are necessary for this:

- Go to the page **project** and open the chosen project file
- On the open project page, press the Play button on the teach panel.
- Set the speed to the desired value with **+/-** on the teach panel.
- Once you have set your preferred speed, press and hold the M/A mode switch button (do not stop the project).
- When the two display positions on the teach panel start to flash, press the **+/-** buttons in this order: **+ - + + -**.
- Now the project can be stoppen
- After unlocking, the robot switches to auto mode to complete the test run. The robot saves the speed at this point as the project speed.
- Go to the *Run Settings* page to check that the project speed has been updated
- Switch to automatic mode using **M/A + - + + -**

Hint: Remember that the project speed will be reset to 10% if you carry out the test run again in manual mode. You must perform these steps again to set the project speed.

6.9.6 Troubleshooting

As the robot program required for communication with Mikado contains a continuous loop, the test run will never end. In order to obtain the **tested** status nevertheless and thus be able to start the program in automatic mode, the new program can initially consist of the blocks

Start -> Wait -> Stop

exist. The following steps are then carried out:

- The test run is started (press the Play button on the teach panel when the project page is open).
- While the wait block is active, set the project speed and confirm with **M/A + - + + -**.
- After the time has expired, the project is **tested**.

As the **tested** status is not lost by changes, the wait block can now be replaced by a list block (or by several blocks as in the example illustration *Listen Node with debug display and possibility to return*).

6.9.7 Writing Techman variables and executing programs

To write variables and execute programs, the *Execute robot program* task is used in the Mikado program. Due to Techman's sequential flow programming, native robot programs must be executed slightly differently than with other robots. Mikado writes the variables and then leaves the Listen Node, giving up control. With the help of the written variables, the Techman program/flow can then execute further movements in a subflow, for example. Finally, you have to go back to a Listen Node on the Techman side.

6.10 Set up Universal Robots Controller

This section describes the preparations and settings required to control a UR robot with Mikado.

6.10.1 System requirements

The Mikado connection for Universal Robots is compatible with UR robots of the CB3-series from V3.7 and the e-series from V5.1.

6.10.2 Network configuration

A network connection between the two systems is required for communication between the robot controller and Mikado Controller. To do this, first connect an Ethernet port of the Universal Robots Controller to an Ethernet port of the Mikado Controller. A fixed IP address and a corresponding subnet mask must then be set for both the Mikado Controller and the robot controller. The network configuration of the Mikado Controller is explained in more detail in section *Network adapter*.

The IP address of the UR controller can be accessed via the *Settings* menu (list icon at the top right) of the Polyscope interface. An IP address can be assigned there under the heading *System* → *Network* → *Static IP address*.

Make sure that the IP addresses of the Mikado Controller and Universal Robots Controller are in the same address space and that the subnet masks match. The IP addresses of the devices must not be identical.

Warning: In the *Installation* area, the *Fieldbus* → *Ethernet/IP* interface must not be activated as it interferes with the network connection to the Mikado controller.

Then enter the IP address of the robot controller in the Mikado Assistant under "Robot" -> "Connection" in the corresponding field and save the settings.

6.10.3 Setting up the communication program

In order for the robot to be controlled by Mikado, a communication program must be copied to the Universal Robots Controller and set up there. To do this, carry out the following steps:

- Copy the URCap "ExternalControl" program to a USB stick. You can find the program via the button in the Mikado Assistant ("Parameters" -> "Robot" -> "Open directory") or directly in the following directory: `/opt/isys-vision/mikado/share/ur_robot_driver/resources/externalcontrol-X.X.X.urcap`
- Install URCap on the UR controller:
  - CB3: *Setup Robot* → *URCaps* → + button → select externalcontrol-X.X.X.urcap → Restart
  - e-series: Top right on the menu icon → *Settings* → *System* → *URCaps* → + button → select externalcontrol-X.X.X.X.urcap → Restart
- Configure URCap
  - *Program Robot* → *Installation* → *External Control*
  - Enter the IP of the Mikado controller
- Create new UR program
  - Node "*External Control*" (listed under *Structure*) insert
  - Save program (e.g. under the program name *mikado*)
- PolyScope 5.x: Activate *Remote Control Mode*
  - Top right on the menu icon → *Settings* → *System* → *Remote Control* → *Enable*

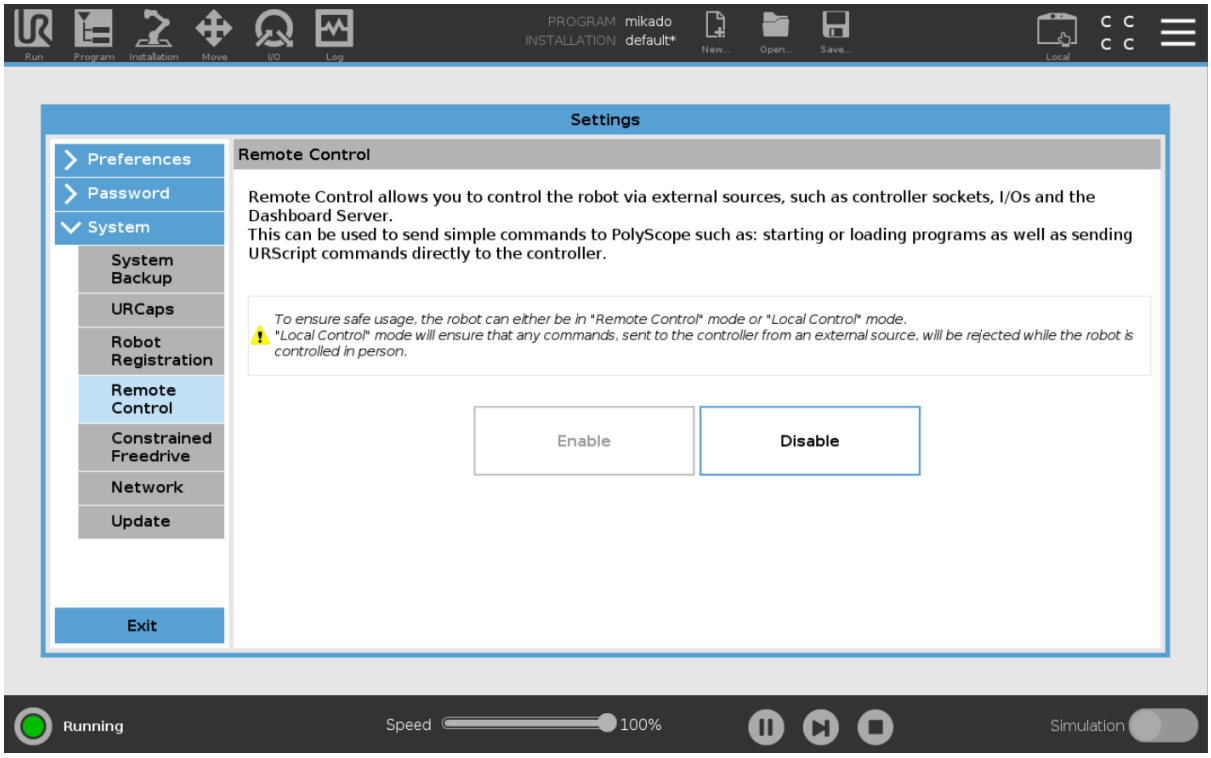


Fig. 45: Settings for remote control mode

– *Exit* → Switch mode from *Local* to *Remote Control* at the top right

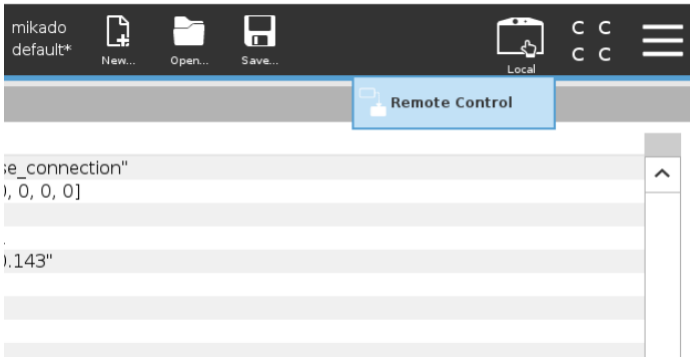


Fig. 46: Switch between *Local* and *Remote Control* mode

- **PolyScope 5.10 and newer**: activate services
  - In newer PolyScope versions on factory settings, multiple services that are necessary for Mikado to connect to the UR robot are deactivated
  - To reactivate these, so the robot state can be visualized in mikado:
    1. Hamburger menü (upper right) → *Settings*
    2. *Security* → *Services*
    3. Unlock menu with admin password. (default: “easybot”, this might need to be changed beforehand under *Password* → *Admin*)
    4. Activate **Dashboard Server**, **Primary Client Interface**, **Secondary Client Interface** and **Real-Time Data Exchange (RTDE)**
    5. Lock menu and *Exit*

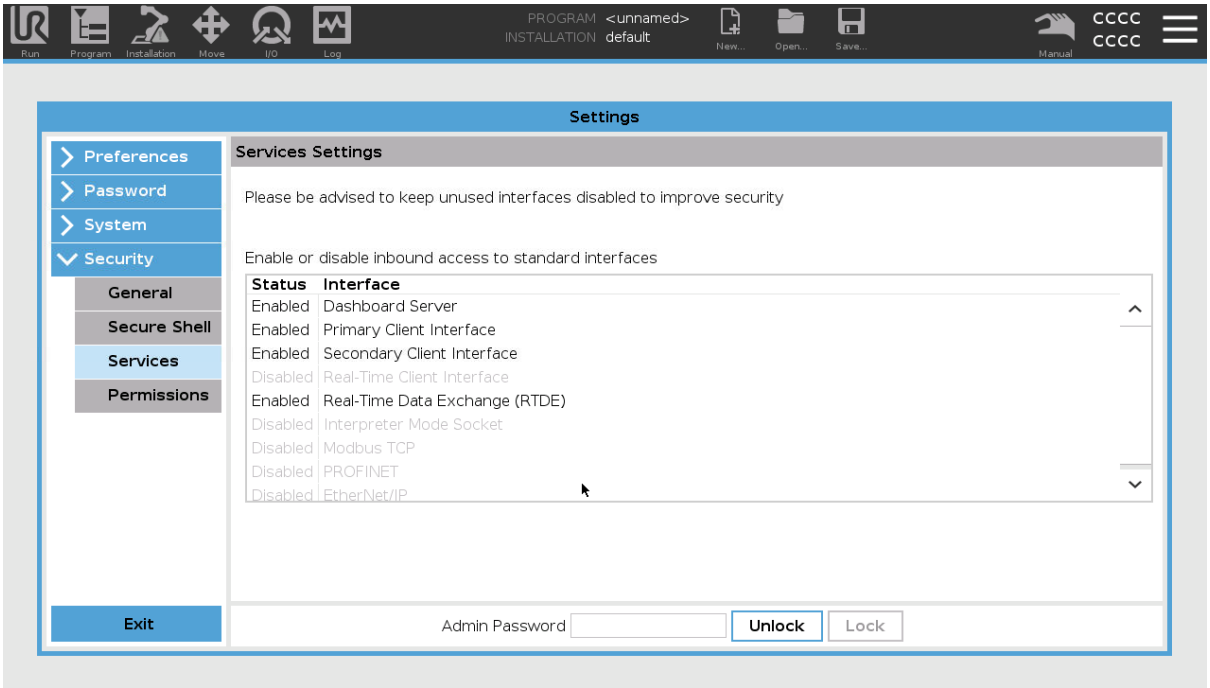


Fig. 47: PolyScope 5.23 Services

6.10.4 Set up autostart

Autostart can be activated so that Mikado can control and move the robot directly when the application is started. In this case, Mikado triggers the UR so that the communication program is also started when the robot axes are enabled.

Set up and configure Autostart:

- On the UR Controller under *Installation* → *Startup* → Check *Load default program* → Select communication program (e.g. *mikado.urp*)
- Save installation
- Open the autostart program in local mode. This program will be loaded automatically on future restarts.
- Activate *Remote Control Mode*
- If necessary, adjust the following parameters in the corresponding cell in the `launch/manipulator.launch.xml` file:

```
<param name="enable_autostart" value="true"/>
<param name="autostart_ur_program_name" value="mikado.urp"/>
```

  - *enable_autostart*: Enable/disable the autostart function. Mikado waits at program start until a robot status could be read and only then triggers the communication program. *(default value: 10s)*
  - *autostart_ur_program_name*: Program name of the autostart program (usually name of the communication program including .urp extension) *(default value: mikado.urp)*
- Now Mikado will automatically start the autostart program on Mikado startup.

Hint: UR offers the option of energizing and enabling the robot directly after the controller start via digital inputs. Use this function (can be found under *Installation* → *Startup*) to boot the system into an operational state.

6.10.5 Joint angle limits

The Mikado path planning supports joint angle ranges of up to 360°. As a UR has spinning joints, the value range in the robot controller (Polyscope) must be reduced. You can define limits for the *Joints* in the robot safety settings. Use the same value ranges here as you have set in the configuration (*joint_limits.yaml*) of Mikado. This prevents you from accidentally teaching robot poses that are outside the limits.

Please note: Give the joint limits in the controller a little more leeway (+/- 4°), as a safety stop can be triggered earlier and the Freedrive counteracts before the limits are reached. Example: Configuration Mikado: from -180° to 180° → Configuration Polyscope: from -184° to 184°

Hint: If you want to go beyond the joint angle value range for certain applications, you can also move the value range. You just need to make sure that the joint angle limits on the controller and the Mikado configuration match.

It is also possible to use a native robot program to enable rotations beyond the limits. For example, you can use the task "*Execute robot program*" for certain movements in your application and then return control to Mikado.

6.10.6 Speed and acceleration

Make sure that you always set a sufficiently low speed/acceleration in Mikado so that the robot cannot exceed the maximum speed/acceleration set in the UR controller. If the UR controller has to limit the speed, this can lead to the error message "Goal distance too high" and the robot may stop before it reaches the planned goal. In this case, you must either increase the limits in the UR controller or reduce the values in Mikado on the *Robot control* page.

6.10.7 Extracting and using robot calibration data

All UR robots are calibrated at the factory before delivery to compensate for inaccuracies in the robot kinematics. This factory calibration can be extracted and used by Mikado to calculate the inverse kinematics. In this way, inaccuracies are also compensated for when the robot is controlled by Mikado.

The use of the robot calibration data for Mikado is not mandatory, but is strongly recommended, as the position of the end effector may otherwise deviate by several centimetres.

The following steps are necessary for this:

1. Start the UR robot and ensure that there is a network connection to the Mikado controller (see *Network configuration*)
2. Open the console window on the Mikado controller and execute the following command. Here, <ROBOTER IP> and <ZELLNAME> must be replaced by the IP address of the robot and the name of the Mikado cell to be used. This command is used to read out the calibration of the robot and save it in the specified file:

   ```
   roslaunch ur_calibration calibration_correction.launch robot_ip:=<ROBOTER IP>␣
   ↪target_filename:="${HOME}/mikado/cells/<ZELLNAME>/config/ur_calib_data.yaml"
   ```

   The name of the target file can be freely selected if required. However, this must be taken into account accordingly in the following steps.
3. The calibration data is now available as a file and needs to be linked to the robot cell. To do this, open your cell in the Mikado-Assistant and switch to the *Cell → Cell Model* page. Select *Show Urdf* and then the `world.xacro` file. Now replace the path to the calibration file (*kinematics_parameters_file*) as shown in the following example. <CELLNAME> must also be replaced here with the name of your Mikado cell:

   ```
   <xacro:ur10e_robot kinematics_parameters_file="$(find ur_description)/config/
   ↪ur10e/default_kinematics.yaml" prefix="${arm_prefix}"/>
   ```

   is replaced by:

```
<xacro:ur10e_robot kinematics_parameters_file="$(find <ZELLNAME>)/config/ur_
↪calib_data.yaml" prefix="${arm_prefix}"/>
```

4. Furthermore, open the file `launch/manipulator.launch.xml` in the cell directory and replace the line:

```
<arg name="kinematics_config" default="$(find ur_description)/config/ur10e/
↪default_kinematics.yaml" ... />
```

   with:

```
<arg name="kinematics_config" default="$(arg project_dir)/config/ur_calib_data.
↪yaml" ... />
```

5. Now check whether the changes were successful by starting Mikado. If the calibration data is correctly integrated, the following message will appear in the logs when you start Mikado: `Calibration checked successfully.`

   If an error occurs, the message `The calibration parameters of the connected robot do not match those in the specified kinematics configuration file [...]` appears.

Hint: The hand-eye-calibration should be repeated after the robot calibration data has been applied. The calibration data allows to locate the camera position more precisely.

Warning: Take care to always update both files that mention the new calibration data:

- `launch/manipulator.launch.xml`
- `urdf/world.xacro`

6.10.8 Optional: Execute native robot program

Native programs can be executed on the controller using the "*Execute robot program*" task.

The following steps are necessary for this:

- Create a new UR program using the teach pendant
- If you want to transfer variables to the program, create the corresponding installation variables:
  - These can be found under *Program Robot* → *Installation* → *General* → *Variables*
  - Do not forget to save the installation configuration with the installation variables as well

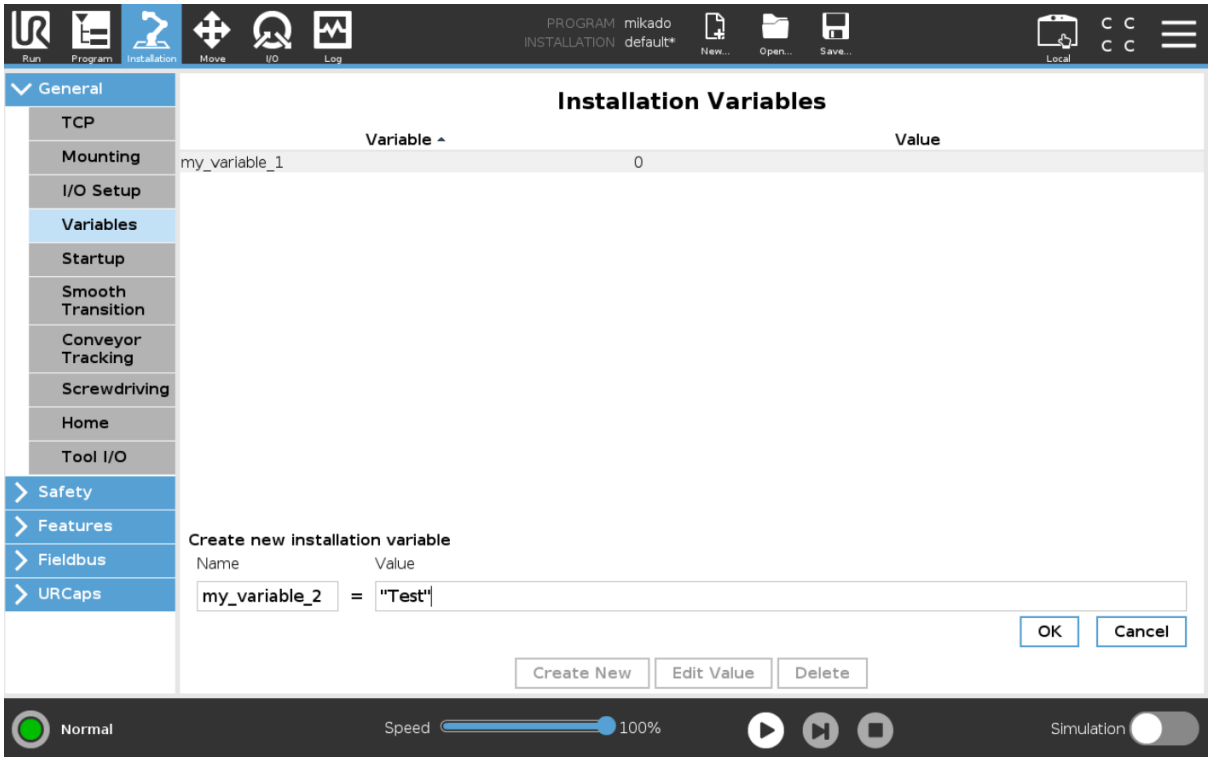


Fig. 48: UR Installationsvariablen

- Insert the UR command **Stop** at the end of your UR program. At this point, the UR program stops and returns control to Mikado.

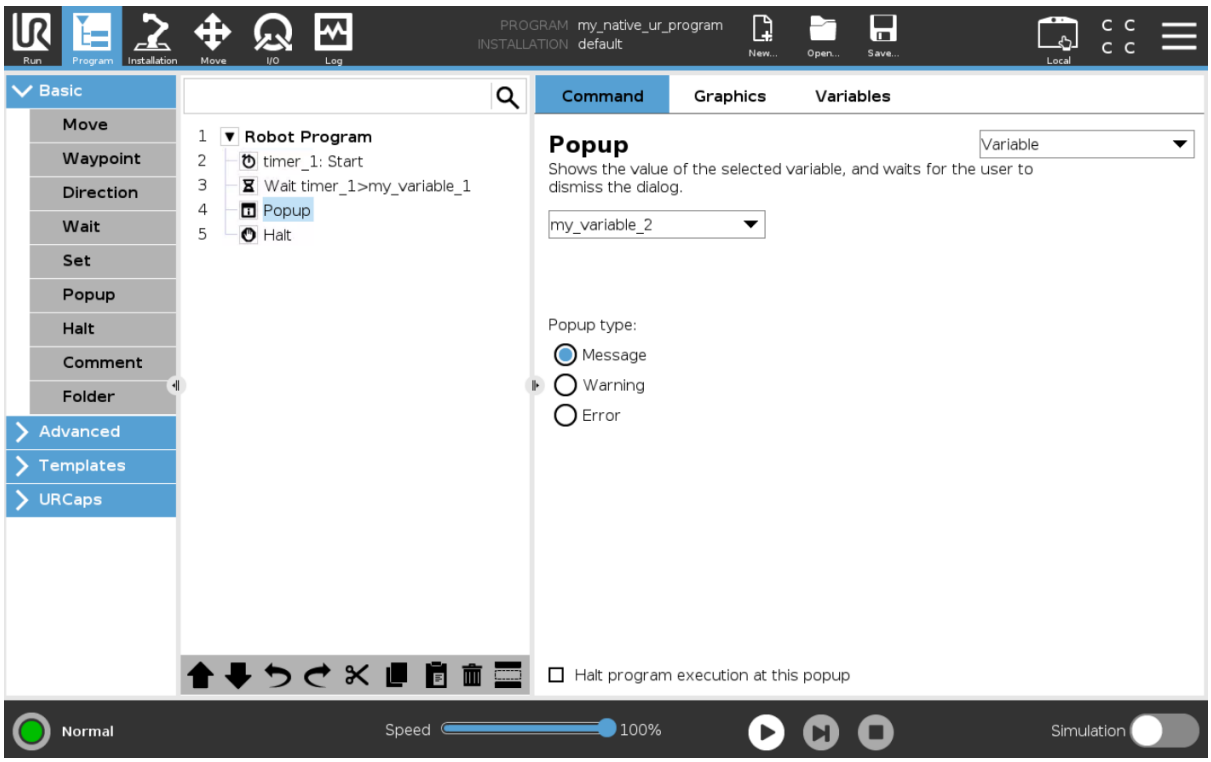


Fig. 49: Example of a native UR program

- Once you have completed and tested your UR program, load and start the control program again

- Now you can use the task "*Execute robot program*" in Mikado to integrate your native UR program into the process

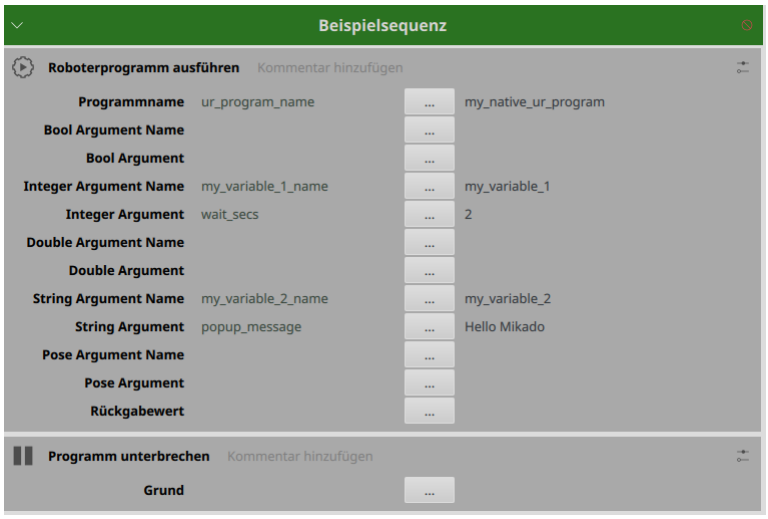


Fig. 50: Example Mikado program with task "*Execute robot program*"

Hint: In the currently available Mikado version, you do not have the option of returning a return value to Mikado. By default, you will receive the value *0* as the return value of the task.

6.11 Using UR robots

6.11.1 Turning on the robot

- Start Mikado and robot controller
- Energize and unlock the robot
- Make sure that the *Remote Control* mode is selected on the teach pendant **at the start time of Mikado**

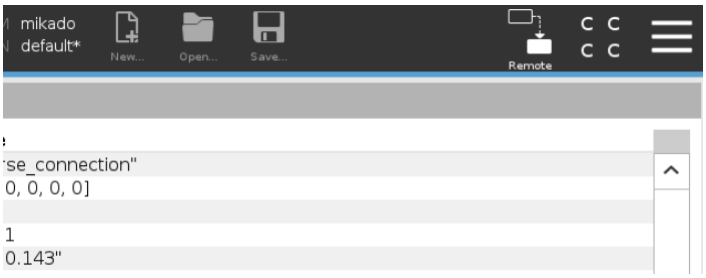


Fig. 51: The *Remote Control Mode* can be selected at the top right of the UR Teach Pendant

- Load and start the *Mikado* program on the UR Controller
  - A connection to Mikado is established. If this is not successful, an error message appears. In the event of an error, check the network connection and start the Mikado interface.
  - The execution of movements can now be stopped or paused on the teach pendant. However, this can lead to timeouts occurring in Mikado and thus tasks being aborted due to the timeout.

Hint:

- Note that the Mikado communication program is terminated as soon as you move the robot manually. Restart the program as soon as you want to return control to Mikado.
- You can optionally use an autostart for the UR program (see *Set up autostart*).

6.11.2 Using URCap grippers

To use additional URCaps, e.g. for gripper control:

- the **ExternalControl** URCap has to be executed permanently, so Mikado can control the robot
- to additionally e.g. control a gripper using URCaps, the program can reacto to Events or run an additional Thread to perform additional actions depending on robot outputs.

Using Events the program could for example look like this:

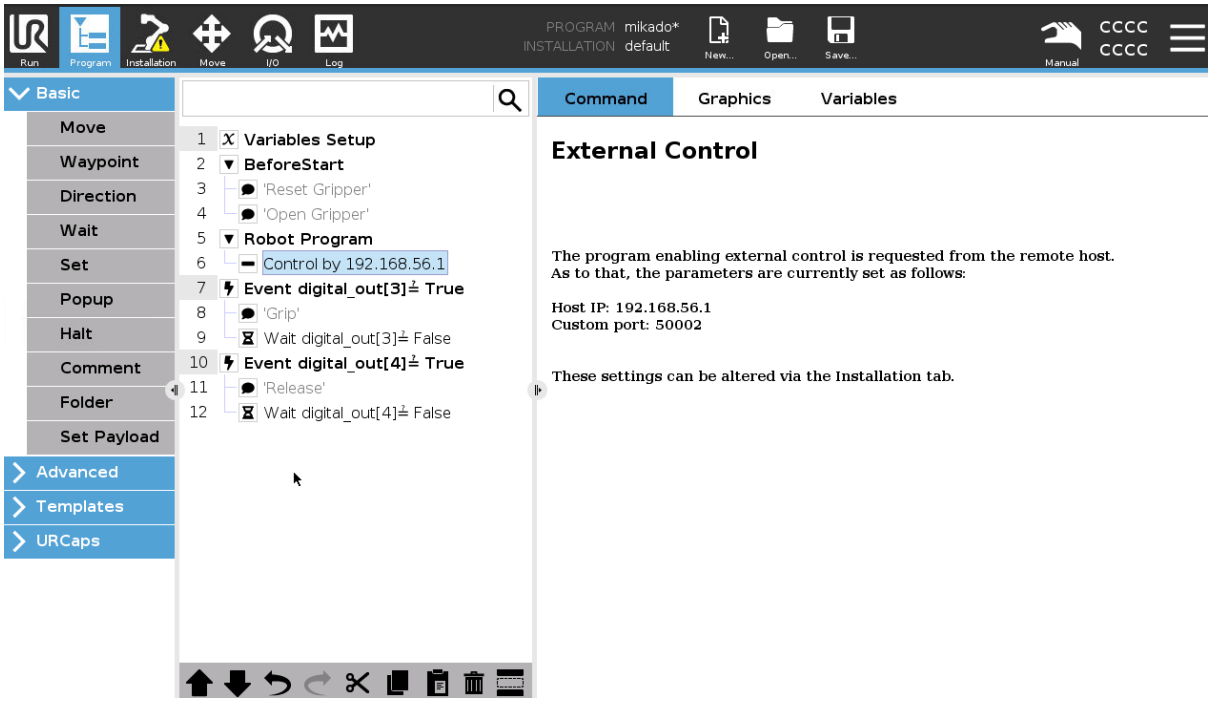


Fig. 52: URCap Gripper Events for one gripper

It is possible to control multiple grippers, when the currently used gripper is indicated by a digital output. In the following example digital output 5 and 6 are used to activate the respective gripper.

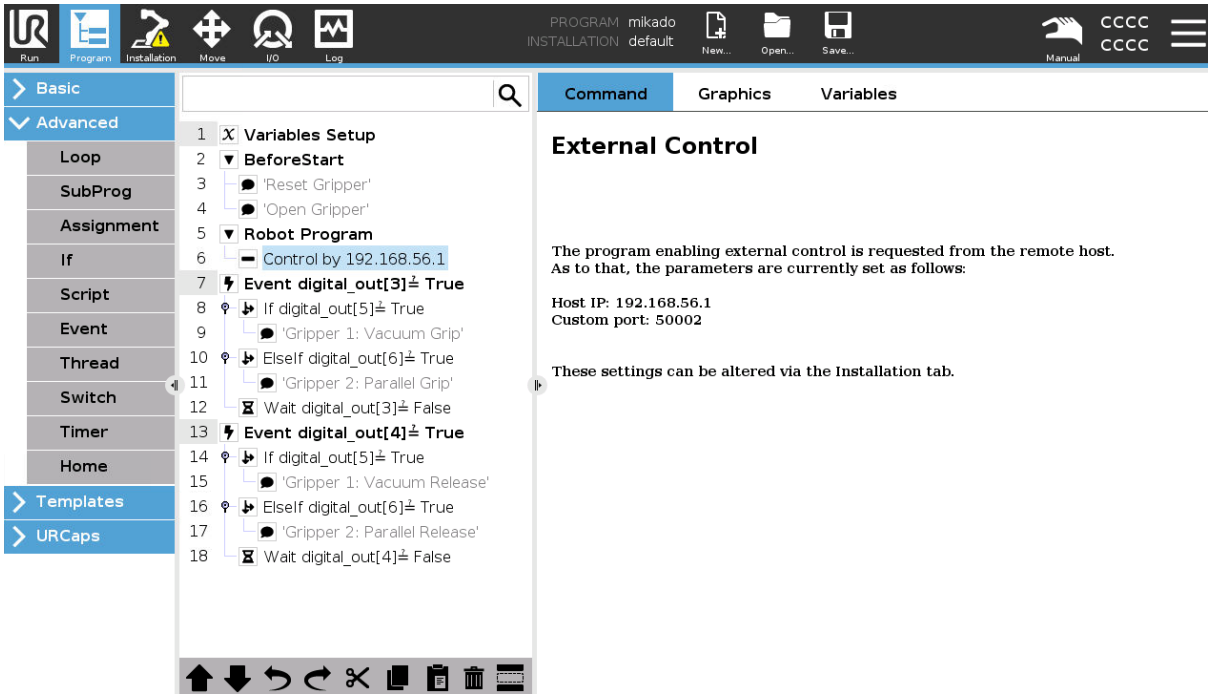


Fig. 53: URCap Gripper Events for multiple grippers

For just two grippers this can be simplified, so e.g. just the two states of one digital output can be used to choose between the two grippers.

When using one Thread instead of Events, the program could look like this:

```
Hauptprogramm
  ExternalControl

Thread:
  if Output[0] == True:
    schließe Greifer
    Wait until Output[0] == False

  if Output[1] == True:
    öffne Greifer
    Wait until Output[1] == False

  Script: sync()
```

6.11.3 Parameters and controls

- **Speed - Acceleration**:

  Two input fields allow you to set the speed and acceleration factor for robot movements. Please note that maximum values are stored in the *System configuration* so that you may not be able to exceed a limit value.

6.11.4 Other notes

- Teaching position:
  - You can teach positions using either buttons on the teach pendant (polyscope) of the robot or the Freedrive mode.
  - Click on “Accept current robot pose” in Mikado.
  - Note that you must restart the UR communication program after manual robot movements so that Mikado can send commands to the robot again.

6.11.5 Troubleshooting Universal Robot

- The robot cannot be moved via Mikado
  - Check whether there is a connection to the robot
  - Are the axles energized?
  - Is the communication program running on the Teach Pendant?
  - Is the UR in remote control mode?
  - Has the communication program entered the address of the Mikado controller as the IP?
- The UR communication program cannot be started and displays the error message``The connection to the remote PC could not be established``
  - Has the communication program entered the address of the Mikado controller as the IP?
  - Are the IP addresses of the robot and Mikado controller correctly configured and in the same subnet?
- The Mikado system status for the robot is red
  - Check whether the axes are energized and the communication program has started successfully
- A native program cannot be started.
  - Check the program name, variable names and the existence and data type of the installation variables.
  - Important: The system connects to the UR Controller during the start-up process of the Mikado interface. At this point, the *Remote Control Mode* must be selected on the teach pendant.
  - If a robot movement is to be executed in the native program, the first waypoint must match the current robot position. Otherwise you will receive an error.
- Autostart of the Mikado communication program does not work
  - The system connects to the UR Controller during the start-up process of the Mikado interface. At this point, the autostart program needs to be the current program and the *Remote Control Mode* must be selected on the teach pendant
- No connection to the robot possible and the Mikado console reports: `Variable 'tool_digital_output_mask' is currently controlled by another RTDE client.`
  - Make sure that no other UR program besides the URCap “ExternalControl” uses RTDE (Real-time Data Exchange).
  - When the resource conflict is caused by a required URCap (e.g. for a gripper), the Mikado UR driver has to be modified, to give up control over the resources used by multiple URcaps:

* the file rtde_input_recipe.txt has to be made writeable and the modified:

```
roscd ur_robot_driver/resources
sudo chmod 777 ./rtde_input_recipe.txt
kate ./rtde_input_recipe.txt
```

* in the file rtde_input_recipe.txt the following lines have to be removed, as they are used by another URCap:

```
tool_digital_output_mask
tool_digital_output
```

* See also https://github.com/UniversalRobots/Universal_Robots_ROS_Driver/issues/239

- No connection to the robot possible and the Mikado console reports: `Variable 'speed_slider_mask' is currently controlled by another RTDE client.`
  - In the *Installation* area, the *Fieldbus* → *Ethernet/IP* interface must not be activated as it interferes with the network connection to the Mikado controller.
    * See also https://github.com/UniversalRobots/Universal_Robots_ROS_Driver/issues/204
- A previously working OnRobot URCap stops working, e.g. with a message "Waiting for valid … timed out. Program halted"
  - Navigate to *Installation* → *URCaps* → *OnRobot* to change the offsets of the used RTDE registers to remove the resource conflict of the URCaps.
- After **updating** the `mikado-robots-ur` package, the Mikado cell no longer starts and displays an error message in the console, usually `resource not found: ur_e_description`. This situation occurs when the Mikado cell was originally created with an older version of the UR driver and is now no longer compatible with the newly installed driver version. The manual adjustments listed below are necessary to make the Mikado cell functional with the current driver.

  Please note: The following changes are all shown using an `ur10e` as an example. If a different robot is used in the cell, the examples must be adapted to the model actually used.

  - **urdf/world.xacro**

    The following lines:

```
<xacro:include filename="$(find ur_e_description)/urdf/ur10e.urdf.xacro"/>
<xacro:ur10e_robot joint_limited="true" kinematics_file="${xacro.load_yaml('
↪$(find ur_e_description)/config/ur10e_default.yaml')}" prefix="${arm_
↪prefix}"/>
```

    must be replaced by these lines:

```
<xacro:include filename="$(find ur_description)/urdf/inc/ur10e_macro.xacro"/>
<xacro:ur10e_robot kinematics_parameters_file="$(find ur_description)/config/
↪ur10e/default_kinematics.yaml" prefix="${arm_prefix}"/>
```

  - **launch/trajectory_execution.launch.xml**

    The following line:

```
<rosparam unless="$(arg simulated_robot)" file="$(find ur10_e_moveit_config)/
↪config/controllers.yaml"/>
```

    must be replaced by this line:

```
<rosparam unless="$(arg simulated_robot)" file="$(find ur10e_moveit_config)/
↪config/ros_controllers.yaml"/>
```

– **launch/manipulator.launch.xml**

The following section:

```
<group unless="$(arg sim)">
  <arg name="kinematics_config" default="$(find ur_e_description)/config/
↪ur10e_default.yaml" doc="Kinematics config file used for calibration␣
↪correction. This will be used to verify the robot's calibration is␣
↪matching the robot_description."/>

  <arg name="max_joint_difference" default="0.0001" doc="Minimal accuracy of␣
↪each joint of robot"/>
  <arg name="controller_config_file" default="$(find ur_robot_driver)/config/
↪ur10e_controllers.yaml" doc="Config file used for defining the ROS-Control␣
↪controllers."/>
  <arg name="use_tool_communication" default="false" doc="On e-Series robots␣
↪tool communication can be enabled with this argument"/>
  <arg name="headless_mode" default="false" doc="Automatically send URScript␣
↪to robot to execute. On e-Series this does require the robot to be in
↪'remote-control' mode. With this, the URCap is not needed on the robot."/>
  <arg name="limited" default="true" doc="Use the description in limited␣
↪mode (Every axis rotates from -PI to PI)"/>
  <include file="$(find ur_robot_driver)/launch/ur_common.launch">
      <arg name="use_tool_communication" value="$(arg use_tool_communication)
↪"/>
      <arg name="controller_config_file" value="$(arg controller_config_file)
↪"/>
      <arg name="kinematics_config" value="$(arg kinematics_config)"/>
      <arg name="robot_ip" value="$(arg robot_ip)"/>
      <arg name="limited" value="$(arg limited)"/>
      <arg name="headless_mode" value="$(arg headless_mode)"/>
      <arg name="max_joint_difference" value="$(arg max_joint_difference)" />
  </include>
</group>
```

must be replaced by this section:

```
<group unless="$(arg sim)">
  <arg name="kinematics_config" default="$(find ur_description)/config/ur10e/
↪default_kinematics.yaml" doc="Kinematics config file used for calibration␣
↪correction. This will be used to verify the robot's calibration is␣
↪matching the robot_description."/>
  <arg name="max_joint_difference" default="0.001" doc="Maximum allowed␣
↪difference between target and actual joint positions for low bandwidth␣
↪trajectory follower"/>
  <arg name="controller_config_file" default="$(find ur_robot_driver)/config/
↪ur10e_controllers.yaml" doc="Config file used for defining the ROS-Control␣
↪controllers."/>
  <arg name="use_tool_communication" default="false" doc="On e-Series robots␣
↪tool communication can be enabled with this argument"/>
  <arg name="headless_mode" default="false" doc="Automatically send URScript␣
↪to robot to execute. On e-Series this does require the robot to be in
↪'remote-control' mode. With this, the URCap is not needed on the robot."/>
  <arg name="ur_hardware_interface_node_required" default="false" doc="Shut␣
↪down ros environment if ur_hardware_interface-node dies."/>
  <include file="$(find ur_robot_driver)/launch/ur_common.launch">
      <arg name="use_tool_communication" value="$(arg use_tool_communication)
↪"/>
```

(continues on next page)

(continued from previous page)

```
      <arg name="controller_config_file" value="$(arg controller_config_file)
↪"/>
      <arg name="kinematics_config" value="$(arg kinematics_config)"/>
      <arg name="robot_ip" value="$(arg robot_ip)"/>
      <arg name="headless_mode" value="$(arg headless_mode)"/>
      <arg name="max_joint_difference" value="$(arg max_joint_difference)" />
      <arg name="ur_hardware_interface_node_required" value="$(arg ur_
↪hardware_interface_node_required)"/>
  </include>
</group>
```

– **launch/planning_context.launch.xml**

The following line:

```
<rosparam command="load" file="$(find ur10_e_moveit_config)/config/
↪kinematics.yaml"/>
```

must be replaced by this line:

```
<rosparam command="load" file="$(find ur10e_moveit_config)/config/kinematics.
↪yaml"/>
```

6.12 Setting up the Yaskawa Motoman Controller

- *Setting up the Yaskawa Motoman Controller*
- *Network configuration*
- *Copying the files*
- *Import MotoROS*
- *Giving up control over the robot temporarily*
- *Call INIT_ROS and RET to the caller*
- *Setting of the required outputs without INIT_ROS*
- *INIT_ROS as default job, other jobs might get run using CALL JOB*

6.12.1 Preparation

This description was created for the Yaskawa Motoman controller models DX200, FS100, YRC1000 and YRC1000micro (also YRC1000u). If you have any problems, please contact Mikado Support. The setup procedure differs depending on whether a **Standard Pendant** (small screen at the top, keypad at the bottom) or a **Smart Pendant** (large touchscreen, buttons on the right in the frame) is used.

Note on the Smart Pendant: Since Smart Pendant software version 2.1, the Smart Pendant has a `Classic Interface` interface which emulates the Standard Pendant. This means that instructions for the Standard Pendant can also be applied to the Smart Pendant. The `Classic Interface` can be found under `Menu` → `System Settings` → `Classic Interface` → `Start Classic Interface`.

Software Pendant: All instructions for the **Standard Pendant** can alternatively be implemented with the Windows software **Software Pendant**.

Software requirements

The minimum requirements for the controller software, which can be viewed under `System Info` → `Version`:

Controller	Version
YRC1000	YAS2.80
YRC1000micro	YBS2.45
DX200	DN2.21
FS100	FS3.30

A software version that is too low can, among other things, trigger alarm `1020`; see *Error codes / Alarms*.

6.12.2 Network configuration

A network connection between the two systems is required for communication between the robot controller and Mikado controller. To do this, first connect an Ethernet port of the Yaskawa controller to an Ethernet port of the Mikado controller (**YRC1000**: LAN2 oder LAN3). A fixed IP address and a corresponding subnet mask must then be set for both the Mikado Controller and the robot controller. The network configuration of the Mikado Controller is explained in more detail in section *Network adapter*. Network settings for the Yaskawa controller can be made as follows:

For this step, the controller must be started in `Maintenance Mode`.

Standard Pendant: To start the controller in `Maintenance Mode`, the `Main Menu` button must be held down continuously while the controller is freshly switched on and booting up. The button can be released when the logo appears on the pendant screen. Alternatively, a restart can be triggered directly into maintenance mode via `System Info` → `CPU Reset`.

Smart Pendant: From version 2.1, the Smart Pendant can emulate the interface of the Standard Pendant via `Menu` → `System Settings` → `Classic Interface`. By selecting `Maintenance Mode` instead of `Connect` via `Special Mode` → `Maintenance Mode`, this emulated interface can be started in **Maintenance Mode**.

Im Maintenance Mode:

Standard counterpart: Set security level to `Management` (or `Advanced`) `System` → `Security`, default password: 16 times the digit `9`), then the IP address of the controller can be set under `System` → `Setup` → `Option Function` → `Lan Interface Setting`. Any changes must then be confirmed by pressing `Enter` to apply them.

Smart Pendant: Under `Menu` → `System Settings` → `Controller` → `IP Address` the current IP address of the Motoman controller can be read out and changed if necessary.

Hint: Make sure that the IP addresses of the Mikado Controller and Yaskawa Controller are in the same address space and that the subnet masks match.

To enable Mikado to establish a connection, the IP address of the Yaskawa Motoman controller must be known. To do this, enter the IP address of the robot controller in the Mikado Assistant during the cell creation process or in the Robot section of an existing cell (`Robot` → `Connection`).

6.12.3 Setting up the robot controller

Overview

The controller side of the Yaskawa Motoman ROS driver is called **MotoROS** and is a MotoPlus application. It uses a socket interface and several parallel applications. The standard setup for Mikado uses only a 6-joint manipulator without additional axes. The MotoROS application acts on the Motoman controller as a server for clients on the Mikado PC.

Copying the files

The files required for the setup are located in the folder: `/opt/isys-vision/mikado/robot_programs/motoman/motoman_files`.

The directory can also be opened via the Mikado Assistant using the corresponding button (`Robot` → `Connection` → `Open directory`).

- The `MotoRos` subfolder contains the MotoROS program files. As the programs are controller-specific, they are located in further subfolders corresponding to the various robot controller models. For the YRC1000 controller, for example, the required files are located in `MotoRos/YRC1000/...`. From the available files, you need to select a single suitable MotoROS program.
  - The file name consists of the controller type, the version number and the update rate at which the robot position is sent to Mikado, e.g. `MotoRosYRC1_19122_200hz.out`
  - MotoROS files with a name that ends on `_noStartJob` are configured for applications with control transfer, see *Giving up control over the robot temporarily*.
- The `Inform` subfolder contains the job file `INIT_ROS.JBI` and the configuration file `IONAME.DAT`. In this case too, the correct files must be selected from several available options in the relevant subfolders. For standard single-arm setups, the files from the subfolder `Inform/DX200, FS100, YRC1000/single_arm/` should be used. The other subfolders and files are intended for special setups, e.g. those with additional external axes.

The selected files must be copied to a USB storage device and then inserted into the Pendant or the robot controller for file transfer. All files must be stored in the root directory of the storage device, or in a subdirectory with a simple name.

The next section explains how to copy these files to the robot controller.

Import MotoROS

FS100, DX200, YRC:

For this step, the controller must be started in `Maintenance Mode`.

Standard Pendant: To start the controller in `Maintenance Mode`, the `Main Menu` button must be held down continuously while the controller is freshly switched on and booting up. The button can be released when the logo appears on the pendant screen.

Smart Pendant: From version 2.1, the Smart Pendant can emulate the interface of the Standard Pendant via `Menu` → `System Settings` → `Classic Interface`. By selecting `Maintenance Mode` instead of `Connect` via `Special Mode` → `Maintenance Mode`, this emulated interface can be started in **Maintenance Mode**.

Im Maintenance Mode:

1. Increase the security level to `Management` (or `Advanced`): `System` → `Security`, default password: 16 times the digit `9`
2. go to `System` → `Setup`, select `Option Function`
   - Set `MotoPlus Func` to `USED`

- If the message `Initialise related files? SRAMDRV.DAT` appears, you can confirm it by selecting `YES`

- Set `MOTOMAN DRIVER` to `USED`

3. select the USB memory under `Motoplus APL` → `Device`. Note: The port `USB1:Controller` is located on the controller, `USB:Pendant` or `USB:PHG` are located on the back of the Teach Pendant
4. if the file is not in the root directory of the USB memory device, navigate to the corresponding folder using `Motoplus APL` → `Folder`
5. under `Motoplus APL` → `Load (User App)` select the file `MotoRosXXXXX.out` with `Select` and confirm with `Enter`
6. under `Motoplus APL` → `File List` verify that MotoROS has been installed and that no other MotoPlus applications have been loaded onto the controller

Restart in Online Mode:

1. Set key switch to `TEACH` mode
2. Exit Maintenance Mode
    1. **Standard Pendant**: Restart controller to start again without maintenance mode
        - `System` → `CPU Reset`, `Online`, select `Reset`, or power off and on
    2. **Classic Interface**: Exit maintenance mode via the power button on the top right to exit the Classic Interface
3. **Only for DX200 and YRC**:
    - Within 30 seconds of the reboot, the alarm `8001 [10] Speed FB enabled, reboot now` should appear. After restarting again, no more alarms should appear. If further alarms appear, the MotoROS Troubleshooting page can help.
4. Check MotoROS status:
    1. go to `MotoPlus APL` → `MotoPlus MONITOR` → `Application Run Flow`
        - ensure, that exactly one version of MotoROS is installed and active
    2. go to `MotoPlus APL` → `MotoPlus MONITOR` → `Application Property List`
        - ensure there is exactly one entry for MotoROS, with two sub-entries:

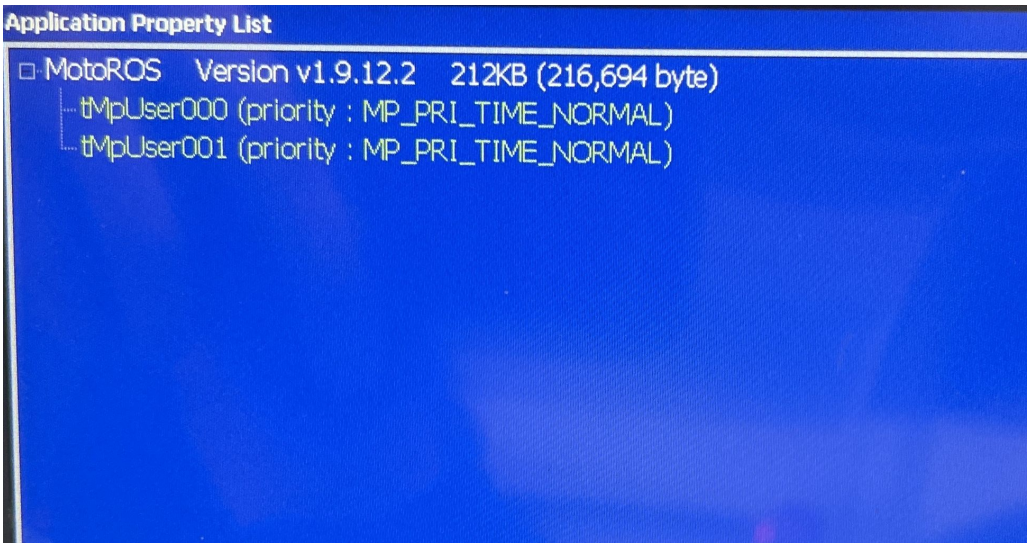


Fig. 54: Application Property List with MotoROS version number

3. If there are conflicts with other installed MotoPlus applications (e.g. because of thread priorities), they should be visible here.

Set up INFORM code

The MotoROS application uses the `INIT_ROS` `INFORM` job. The file `INIT_ROS.JBI` mentioned in section *Copying the files* must now be copied to the controller together with the `IONAME.DAT` file. The `IONAME.DAT` file contains the names of MotoROS output signals.

For the next steps, you should insert a USB memory with the two files into the teach pendant and start the controller normally (not in `Mainenance Mode`). Both files should be in the same folder.

Standard Pendant or **Software Pendant** or **Classic Interface**:

1. Increase the security level to `Management` (or `Advanced`) `System Info` → `Security`, default password: 16 times the digit `9`
2. select the USB memory under `Ext Memory` → `Device`
3. navigate to the correct folder in `Ext Memory` → `Folder`
4. under `Ext Memory` → `Load` → `Job` select and load the `INIT_ROS.JBI` job
5. verify under `Job` → `Select Job` that `INIT_ROS` has been installed correctly

Smart Pendant:

1. zu `Menu` → `Utility` → `File Transfer` navigieren
2. Select `To Controller`
3. if the file is not in the root directory of the USB memory device, navigate to the corresponding folder using `Change Folder`
4. In the job list, check the box in front of the `INIT_ROS.JBI` job and select `Copy Files to Controller`
5. Answer the file transfer dialog with `Yes`

Controller configuration

Now you can verify whether the robot controller can receive commands in **Remote** mode:

Standard Pendant or **Classic Interface**:

1. select `Teach` mode on the pendant using the key switch
2. Increase the security level to `Management` (or `Advanced`) `System Info` → `Security`, default password: 16 times the digit `9`
3. select `In/Out` → `Pseudo Input Sig` on the counterpart
4. Activate the pseudo input `CMD REMOTE SEL`. The signal number is `#87015` (**YRC**) or `#82015` (**FS100**, **DX200**)
    1. move the cursor to the circle next to the appropriate input
    2. Press `Interlock + Select` together to activate the input
5. If GPIO via Yaskawa IO-Board is used: ensure, that Pseudo Input Sig `#87013` (**YRC**) or `#82013` (**FS100**, **DX200**) is active
6. Select `Setup` → `Operate Cond`. Move the cursor to the `Cycle Switch in xxx Mode` entries. Set all these entries to `Cycle`.
7. select `Remote` mode on the pendant using the key switch

Smart Pendant:

1. start the **Classic Interface** via `Menu` → `System Settings` → `Classic Interface` and select `Connect`

2. select `Teach` mode on the pendant using the key switch
3. Select `Setup` → `Operate Cond`. Move the cursor to the `Cycle Switch in xxx Mode` entries. Set all these entries to `Cycle`.
4. select the power button at the top right and return to the Smart Pendant interface with `Disconnect`
5. select `Remote` mode on the pendant using the key switch

`CMD REMOTE SEL` is always active with the Smart Pendant and does not need to be explicitly activated.

Connection test with Mikado

The robot controller should be fully powered up before Mikado is started. After starting Mikado, the robot should audibly activate the servomotors and display a green status in the bottom left of the *status display* if `Remote` mode has been selected on the counterpart via the key switch.

If Mikado cannot establish a connection after completing the configuration, the robot controller must be restarted.

Hint: Mikado can move the robot only while it is in the **remote** mode. Mikado will always show a red `Motion Possible` status while the robot is in **teach** or **auto** mode.

6.12.4 Weiteres & Troubleshooting

Error code description: http://wiki.ros.org/motoman_driver/Troubleshooting

A Mikado-independent guide to setting up MotoROS on the controller: ROS Wiki Tutorial

Yaskawa Motoman Dokumentation: https://www.motoman.com/en-us/service-training/product-documentation

Yaskawa Motoman download area for the software counterpart: https://software.yaskawa.eu.com/programs

Yaskawa Motoman Smart Pendant updates to install version 2.1 or newer: https://yaskawapartners.com/download/smart-pendant-software/

Servomotor

In the standard configuration of the controller, the servomotors are switched to a power-saving mode after a certain period of inactivity. This is ended automatically as soon as the robot receives a new movement task.

I/O address ranges

The Mikado application only uses `General Input` and `General Output` addresses for I/O access.

The first valid Mikado I/O address for Motoman robots has index 1, but the Motoman Teach counterparts use the 1-based notation `Out#001` and `In#001` for general purpose I/O signals. This notation uses continuous indices, not to be confused with the global Motoman addresses of the format `#87015`.

Example: The Mikado output with index 1 is mapped to the Motoman address `#10010` which is also referred to as `Out#001` on the counterpart.

Yaskawa Motoman addresses are written in decimal notation, but the last digit describes an 8-bit byte, which is why only the digits 0-7 are valid here. The address range 10010 - 15127 therefore contains, for example, 512 bytes and 4096 valid bit addresses.

Mikado I/O addresses are continuously mapped to valid Motoman addresses, therefore Mikado input 9 corresponds to Motoman addresses `In#009` and `` `#00020 ``.

Table 5: **Address ranges of various controllers**

Controller	General Input	General Output	Pseudo Input
FS100	10 - 1287	10010 - 11287	82010 - 82207
DX200	10 - 5127	10010 - 15127	82010 - 82207
YRC1000(u)	10 - 5127	10010 - 15127	82010 - 87207

MotoROS uses some fixed outputs, these are named in the `IONAME.DAT`:

```
OUT#889 ROS_READY
OUT#890 ROS_DONE
OUT#891 INIT_DONE
OUT#892 CONNECTION_SRV
OUT#893 MOTION_SERVER
OUT#894 STATE_SERVER
OUT#895 IO_SERVER
OUT#896 FAILURE
```

6.12.5 Enter acceleration limits in joint_limits.yaml

The acceleration limits of the Motoman manipulators are application-dependent and not freely accessible to the public, which is why Mikado initially uses an acceleration limit of 1 rad/s^2. Especially for small and fast manipulators, it is recommended to explicitly specify higher acceleration limits for the joints in `joint_limits.yaml` by setting `has_acceleration_limits` to `true` and `max_acceleration` to the desired limit in rad/s^2.

```
joint_1_s:
  min_position: -2.967
  max_position: 2.967
  has_acceleration_limits: true
  max_acceleration: 8.11
```

Caution: Acceleration limits that are set too high can lead to problems such as incorrectly detected collisions and movement aborts as well as alarm messages on the robot controller.

In the `joint_limits.yaml` file, the robot's range of motion can also be restricted by reducing the `min` and `max` position of individual joints in order to prevent undesirable robot configurations when moving to Cartesian poses.

6.12.6 Error codes / Alarms

After an alarm has occurred, details of the fault can be displayed on the control panel. There you will also find an error subcode, information on the affected joint and possible causes of the error. Information on previous alarms can be found in the standard counterpart under `System Info` → `Alarm History`. There you can switch between the display of main alarms and subordinate alarms.

If a subordinate alarm has occurred (**error code 4000 or greater**), this must be acknowledged by the user on the teach pendant before Mikado can move the robot again.

Hint: After a main alarm (**error code 0000-3999**) has occurred, the Yaskawa controller must be restarted (CPU reset or main switch Off&On).

- Alarms `1020`, `1050`, and `4207`: This error indicates that the MotoROS application cannot be loaded. The cause may be a corrupted file, a faulty USB drive, or an incompatible Yaskawa controller software version; see *Software requirements*.
- Error `4315: Collision detected` and error `1303: Arithmetic Error` indicate unexpected collisions or unexpectedly high resistances. If these messages occur without the robot touching its surroundings,

the collision detection threshold values (Smart Pendant: `Robot` → `Collision detection` → `Mode 8` (default)) should be checked and adjusted if necessary. By adapting the tool properties mass and center of gravity to reality, the reliability of collision detection can be increased and the threshold values can be lowered.

- `4414: Excessive Segment`: If the *acceleration limits* in the file `joint_limits.yaml` are set too high, the error `4414: Excessive Segment` may also occur in addition to these two errors. The error subcode indicates the affected robot joint with an acceleration limit that is set too high.

If the start position sent to the robot deviates from the current position of the robot, various error messages may occur:

- MotoROS Error `3011: Trajectory start position doesn't match current robot position`.
  - If this error occurs without any apparent external influence (e.g., due to minor positional changes at the end of a rapid movement), it is possible to allow the robot driver to perform new attempts if necessary, i.e., to attempt again to start the movement. When the robot arm is still "settling" into position, this allows repeat attempts, so the movement can be successfully executed after the expected start position has been reached.

    To do this, you can configure a parameter in the file `launch/manipulator.launch.xml`:

    * The value for `retry_traj_on_invalid_start_pos` must be set to a positive number, e.g., `"5"`:

```
<include file="$(find motoman_driver)/launch/robot_interface_streaming_
↪yrc1000.launch">
  <arg name="robot_ip" value="$(arg robot_ip)" />
  <arg name="retry_traj_on_invalid_start_pos" value="5" />
</include>
```

    * In older cells, created with versions of the `mikado-robots-motoman` package prior to *3.2.0*, the line `<arg name="retry_traj_on_invalid_start_pos" value="0" />` does not yet exist.

      It must be inserted immediately after the line `<include...$(motoman_driver)/launch/robot_interface_streaming...`.

> **Warning:** The value should not be set higher than 10, as otherwise unexpected deviations in the robot's position might be ignored, and a movement command might be sent to the robot due to this setting, although it is in an unexpected starting position.

- For MotoROS error `5012`, the log contains the description `Incremental move rejected on MotoROS side (5012)`. In a connected Telnet console, the messages `mpExRcsIncrementMove returned: -1` and `ERROR: Controller is not ready (code: 5012). Can't process ROS_MSG_JOINT_TRAJ_PT_FULL` visible.
- These errors can occur in particular if the workpiece held by the gripper is a double fit with another part of the robot cell: The double fit can lead to a passive movement of the robot arm, as well as to an opposite movement as soon as the gripper opens again. Since the gripper opening is often directly followed by a robot movement command, there may be a deviation from the last received robot position and the error patterns mentioned above.
- For detailed error analysis it is possible to establish a telnet connection to the robot from a Konsole window on the Mikado controller.

  After Mikado restart a message `Starting new connection to` should be visible. If the telnet connection attempt fails or no such messages are visible, the network communication is probably faulty.

  If `ping <ROBOT_IP>` works, but `telnet <ROBOT_IP>` or the robot connection of Mikado fails (no message `Starting new connection` in the telnet output), then one possible cause is that multiple network interfaces of the Mikado controller are configured to be in the same sub-net / IP range.

```
$ telnet 192.168.255.1

Trying 192.168.255.1...
Connected to 192.168.255.1.
Escape character is '^]'.

VxWorks login: MOTOMANrobot
Password:

# nach Mikado Neustart: wenn solche Meldungen sichtbar sind,
# funktioniert zumindest die Netzwerkkommunikation:

Starting new connection to the ... Server
```

The Mikado robot status `Motion Possible` should turn green, as soon as the telnet connection prints `In StartTrajMode`. With this message MotoROS indicates it is ready to move the robot.

Warning: Avoid pressing the key combination `Ctrl+C` while the telnet session is in the foreground, as this command is sent to the robot and causes issues there. The connection can be terminated using `Ctrl+D`.

6.12.7 Giving up control over the robot temporarily

After installation of the MotoROS program and the `INIT_ROS` job on the robot, `INIT_ROS` will always run automatically.

Usually Mikado then takes over control of the robot motion.

The task *Execute robot program* can be used to give up and take back motion control of the robot.

- When providing the program name `"robot_disable"`, control of the robot is temporarily given up. This guarantees that Mikado will not interfere when external PLCs or other robot jobs move the robot.
- Running the task with program name `"robot_enable"` reverts this and will regain motion control for Mikado. If this task fails, the robot status signal of Mikado will show this as an error.

To prohibit Mikado from taking control of the robot after startup, the value **control_on_startup** should be set to **false** in `config/manipulator.yaml`.

If the output `ROS_READY` is not set when `robot_enable` is executed, MotoROS will report `Error 5009 "ROS not ready"`.

The same error is generated, when `ROS_READY` is not set during Mikado launch and the file `config/manipulator.yaml` has **control_on_startup** set to `true` or if it is missing.

6.12.8 Running other INFORM jobs on the robot

There are different options for the combination of the MotoPlus Application MotoROS and *INFORM*:

1. MotoROS is installed in a version with `_noStartJob` Suffix. This MotoROS will never force the execution of an *INFORM* Job.
    a. An INIT_ROS Job with a terminating `RET` return statement can be used. To transfer control to Mikado, invoke the job using `CALL JOB:INIT_ROS`. For details, see *Call INIT_ROS and RET to the caller*.
    b. No INIT_ROS job is used. The necessary Outputs `ROS_READY` and `ROS_DONE` are set in a different INFORM Job, see *Setting of the required outputs without INIT_ROS*.

2. MotoROS is installed in a version without `_noStartJob` suffix. MotoROS then automatically starts the *INFORM* Job `INIT_ROS` and takes over robot control, independent of which *INFORM* job was running before that moment. The execution of other *INFORM* Jobs can be done from INIT_ROS using `CALL JOB`, see *INIT_ROS as default job, other jobs might get run using CALL JOB*.

6.12.9 Call INIT_ROS and RET to the caller

Run `CALL JOB:INIT_ROS` from another robot program to allow Mikado to move the robot.

Finally, control is given up again and `RET` causes the calling robot program to continue execution. Such a `INIT_ROS.JBI` example job file can be found in `Inform/DX200, FS100, YRC1000/single_arm_noStartJob_ret/`.

`ROS_DONE` is set as soon as `robot_disable` has been executed. For details see *Giving up control over the robot temporarily*.

6.12.10 Setting of the required outputs without INIT_ROS

Here no `INIT_ROS` job is used. The required outputs `ROS_READY` and `ROS_DONE` are set in a different INFORM job.

It's necessary for the *INFORM* job to execute a WAIT statement while Mikado controls the robot, as this condition is checked by MotoROS.

`ROS_DONE` is set as soon as `robot_disable` has been executed. For details see *Giving up control over the robot temporarily*.

```
' reset MotoROS signals
DOUT OT#(890) OFF
DOUT OT#(889) OFF
' wait for MotoROS to notice
TIMER T=0.05
' set ROS_READY
' allows control by MotoROS
DOUT OT#(889) ON
WAIT OT#(890)=ON
' reset ROS_DONE
DOUT OT#(890) OFF
' Mikado gave up robot control
```

6.12.11 INIT_ROS as default job, other jobs might get run using CALL JOB

It's possible to interrupt the execution of Mikado INFORM jobs on the robot controller temporarily in order to run other jobs.

For this, the alternative job file `INIT_ROS.JBI` from the folder `Inform/DX200, FS100, YRC1000/single_arm_call_job_main` can be used. More information on the installation of job files can be found under *Copying the files*.

This example `INIT_ROS.JBI` job checks two signals on start-up and, depending on the result, either hands control of the robot over to Mikado or calls the `MAIN` job instead:

```
IF ( IN#(1871)=ON AND IN#(1872)=OFF ) THEN
 CALL JOB:MAIN
ENDIF
```

Switching to another job only works when Mikado is not already in control of the robot or has released it, as described under *Giving up control over the robot temporarily*.

6.12.12 Reading and writing 16-bit registers

The task *Execute robot program* provides use of various special functions of Yaksawa robots in Mikado:

- With the program name `"read_mreg"`, a 16-bit register value can be read from the robot controller
  - The index of the register can be provided as the integer argument
  - The value read from the register will be in the output parameter **return value**
- With the program name `"read_mreg"`, a 16-bit register value can be read from the robot controller
  - The index of the register can be provided as the integer argument
  - The value to be written is passed as the double argument

CHAPTER

SEVEN

INSTALLATION

This chapter explains how to set up your robot cell in Mikado Control and obtain an executable system. You will also find information on how the properties of the system can later be adapted to your needs.

7.1 Overview

This overview lists the various components that make up the Mikado environment from a hardware and software perspective. The links and interfaces between the components are also shown schematically with the help of a diagram.

7.1.1 Hardware overview

The hardware setup of a typical Mikado system consists of the following hardware components. In order for all systems to communicate successfully with the Mikado controller, they must be set up accordingly on the controller. The *Assistant* and the *Configuration files* are available for setting up the individual components.

- Mikado-Controller
  - Central component: Takes over robot, gripper and camera control
  - Object recognition, path planning, inverse kinematics, sequence control, fieldbus connection
- Roboter
  - TCP/IP connection to Mikado controller
  - Depending on the manufacturer, the corresponding communication module is loaded with Cell
  - Robot controller must be configured depending on the manufacturer (e.g. as described in section *Set up Mitsubishi Controller*)
- Gripper
  - CAD model of the mold (for collision checks)
  - Parameterization of the opening, IO control or separate driver
  - Either connected via robot IOs, TCP/IP or external IOs (or PLC)
- Camera
  - CAD model (for collision checks)
  - Hand-eye calibration via Mikado surface
  - Selection: Real, file camera or simulated
  - TCP/IP connection to Mikado controller
- PLC / control system
  - Interface to PLC or external systems

- Fieldbus connection to Mikado controller, e.g. via Profinet (may require optional interface module in the Mikado controller)
- Control from outside (start sequences)
- Data exchange / results

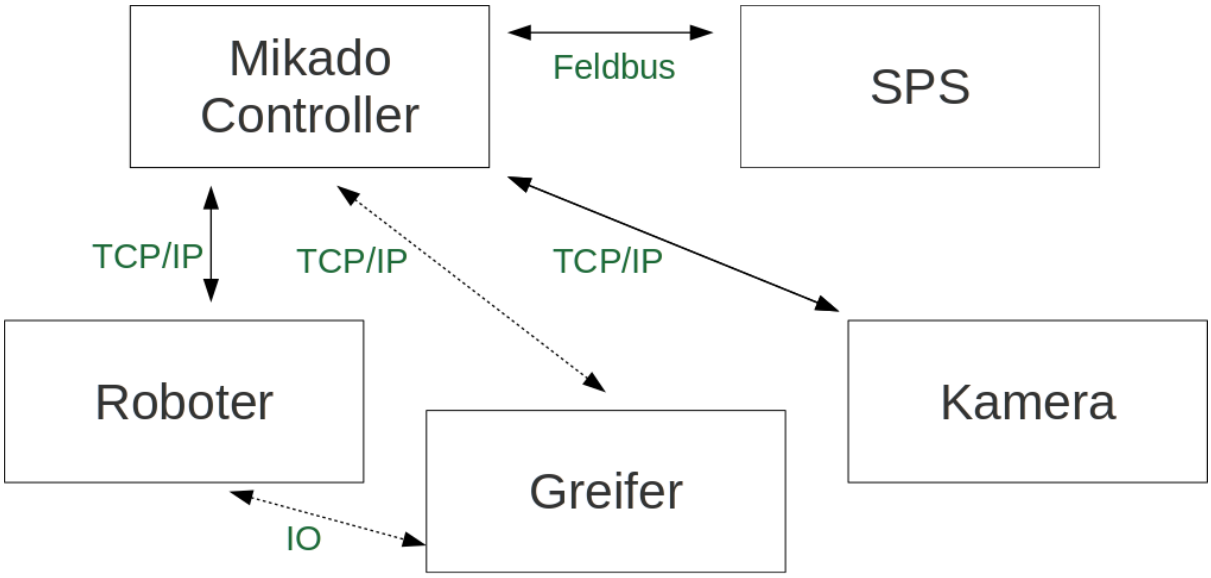


Fig. 1: Mikado Control: Overview of hardware components with interfaces

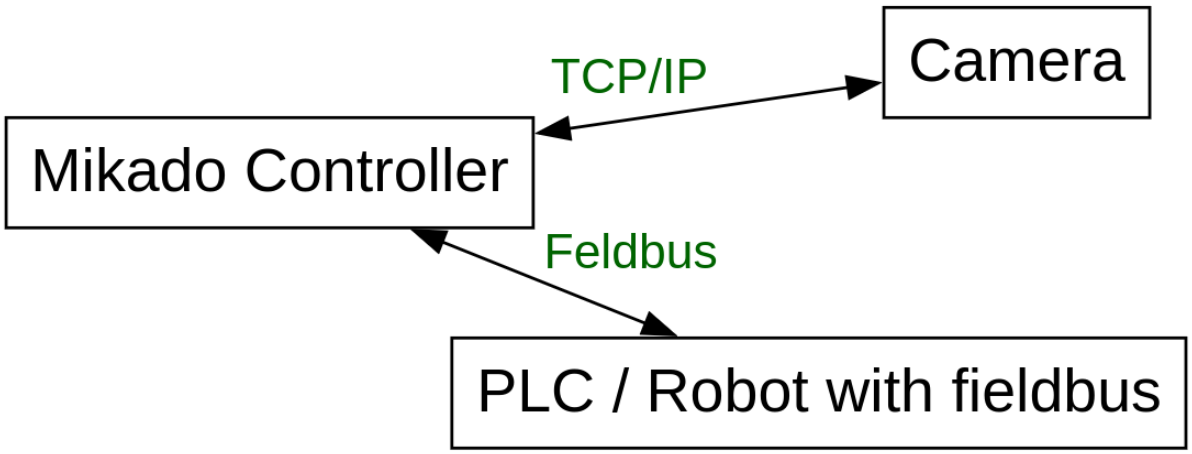


Fig. 2: Mikado Select: Overview of hardware components with interfaces

7.1.2 Software overview

The Mikado system can be divided into five different software modules. These components also work together, which is shown in the following graphic.

- Mikado (*Use*)
  - User and configuration interface
- Mikado Simulator (*Simulation*)
  - Replace real camera with file camera / virtual camera
  - Filling crates with physics simulation

  - Test gripper concept
  - Simulates stereo images of filled boxes
- Mikado Assistant (*Create a new cell*)
  - Create robot cell
  - Modeling grippers
- Scripts and own extensions (*Extending Mikado Control*)
  - Own Halcon scripts
  - Adapter for fieldbus connection
  - Other options such as JavaScript and Python
- Ensenso NxView (Ensenso Manual)
  - Configure and test 3D camera
  - Create file cameras
  - Export configuration for Mikado

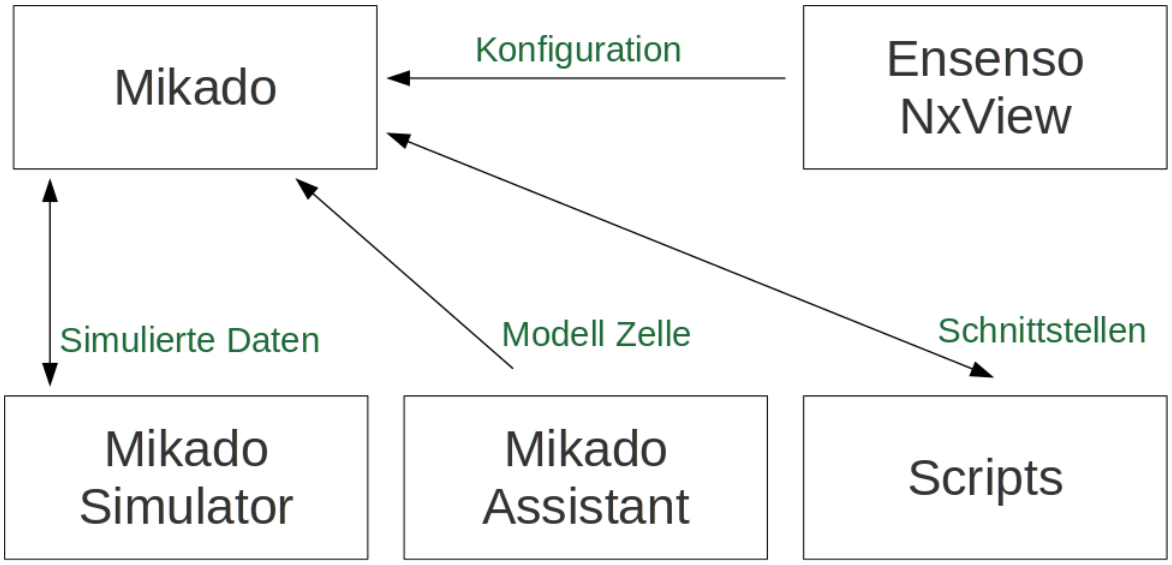


Fig. 3: Overview of software components with interfaces

7.1.3 Integration procedure (step-by-step instructions)

A compact step-by-step guide that can be used as a guide for new integrations is listed below. The 'Tools' column contains notes and links to the sections of the manual that are most important for this step.

Step	Designation	Tools
1	Set up network adapter	*Network manager*
2	Create and customize cell	*Mikado Assistant cell creation* and *URDF editing help*
3	Integrate gripper	*Mikado Assistant cell creation* and *CAD gripper*
4	Integrate robot and camera	*robot controller*
5	Create camera configuration(s)	NxView Manual
6	Hand-eye calibration	*Mikado calibration* and calibration plate
7	Determine the TCP offset of the gripper	*Calibrate TCP* and calibration plate

continues on next page

Table 1 – continued from previous page

Step	Designation	Tools
8	Define scene, product, grasps and program sequence	*Mikado GUI*
9	Optimization and testing	*Mikado GUI* and optionally *Mikado Simulator*
10	Optional: Set up interface	*Extending Mikado Control*

7.2 Configuration Backup

A backup at file system level includes the folder */home/mikado/mikado/cells* and contains all system-specific settings.

Warning: The password used for access control (*Password protection* bzw. *access_control.yaml*) is included in this backup and can be read in plain text.

7.3 Create a new cell

The *Mikado Assistant* is available to create a new cell. To start it, open the program menu (K at the bottom left) or press the Super/Windows key, type *Assist...* and click on the *Mikado Assistant* starter found. The Assistant then allows you to open existing cells as well as create new cells. The created cells then contain:

- Ready-linked interfaces for a selected robot, gripper and camera type
- Model for collision testing
- Default settings for Mikado parameters

7.3.1 Create new Mikado Control cell

After starting the Assistant, click on *New Control Cell*. In the following selection menu, the cell can be roughly pre-modeled in just a few steps and the important settings for the robot and gripper type can be made. Once the model has been created, it can then be adapted precisely to customer requirements using the editor in the Assistant. When selecting parameters, please note that the cell name, robot and gripper type cannot be changed afterwards.

Hint: The poses are always calculated in the Mikado program in the world coordinate system. The world coordinate system is set to that of the robot by Mikado standard. This has the advantage that the same coordinates can be used in both the Teach Panel and the Mikado program. If you want an independent coordinate system, e.g. for overhead robot assembly, you must switch to *Advanced mode* (in the visualization tab *Configuration*, *Advanced mode*). The setting option *World identical to robot coordinate system* then appears, which can be set to false.

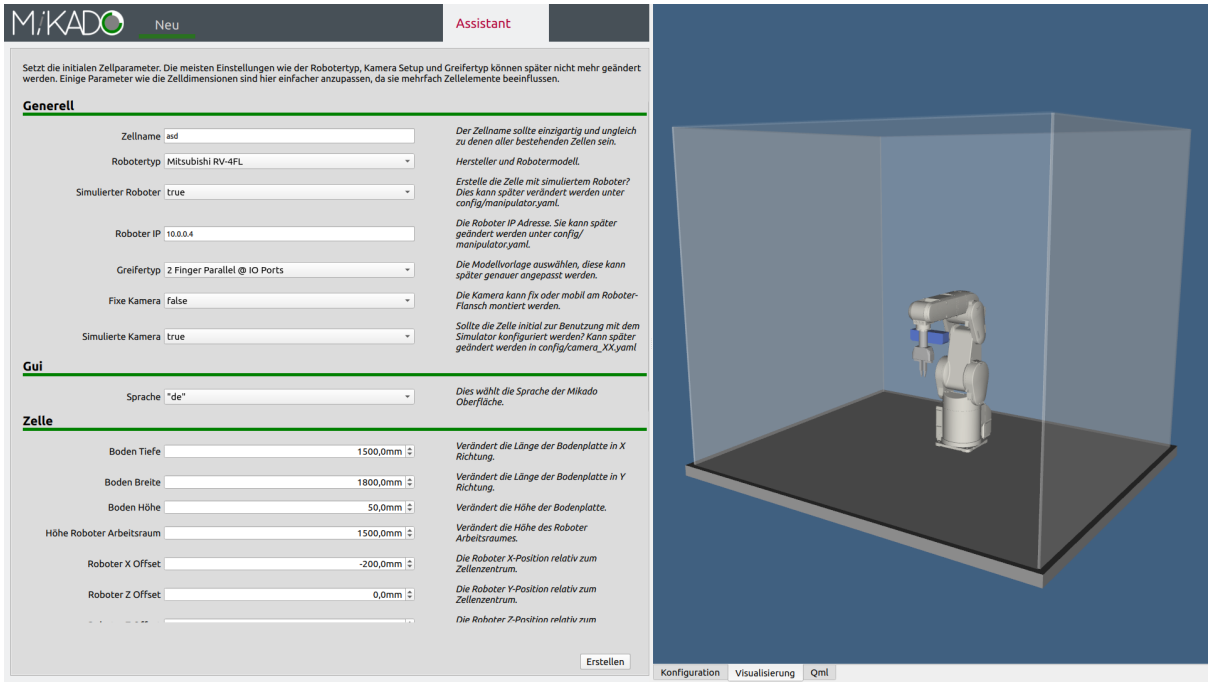


Fig. 4: Screenshot Assistant overview cell creation

7.4 Customize an existing cell

An existing cell has a large number of global parameters that make it configurable.

7.4.1 Speed settings

For each joint of the robot, it must be defined how far it may be moved, how fast it may be moved and how much it may be accelerated. Four limit values are defined for each joint:

- max_position: maximum position of the joint in rad
- min_position: minimum position of the joint in rad
- max_velocity: maximum travel speed of the joint in rad/s
- max_acceleration: maximum acceleration of the joint in rad/s^2

The meaning of the last two values is shown in the figure below.

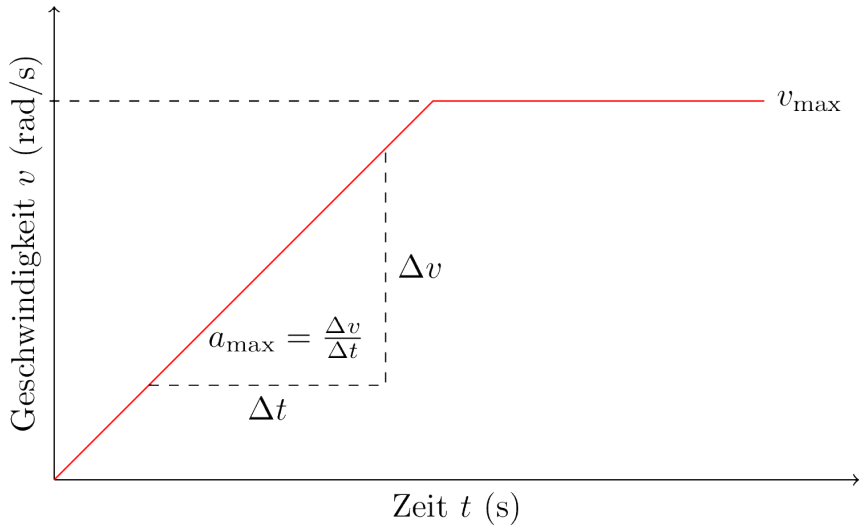


Fig. 5: Meaning of maximum speed and acceleration

The travel speed of the robot when controlled by Mikado depends on various factors.

Absolute values

First of all, each robot joint has physical limits that must not be exceeded. The position and speed limits are usually specified by the manufacturer. These manufacturer specifications are stored in the Urdf of the robot used, which can be found in the installation directory:

```
/opt/isys-vision/mikado/share/<robot_model>_support/urdf/<robot_model>_macro.xacro
```

The respective limit values for position and speed are defined in the joints (see *Joints*) of each robot link in the following form:

```
<limit lower="${radians(-120)}" upper="${radians(120)}" effort="0" velocity="$
↪{radians(623)}" />
```

It is noticeable that there is no value for the gradient of the acceleration ramp. This is due to the fact that the maximum permissible acceleration is heavily dependent on the weight of the gripper used and the gripped part. Therefore, there are no universally valid manufacturer specifications for this. However, Mikado requires this value to calculate the trajectory points, which is why a maximum acceleration of 1 rad/s^2 is assumed as standard. This value is relatively low and must be increased incrementally until the desired cycle time is reached or the robot throws an error.

Mikado provides the configuration file *joint_limits.yaml* to make it easy to set the four limit values.

The absolute values for max_position, min_position, max_velocity and max_acceleration can be set here.

Note: The values defined in `joint_limits.yaml` for min_position, max_position and max_velocity overwrite the corresponding values in the URDF, regardless of whether they are larger or smaller.

Relative values

Typically, the speed of a robot is set on the teach panel using a scaling factor. This scaling factor also appears in Mikado and can be set in the configuration file *manipulator.yaml*:

- `velocity_scaling` is multiplied by the applicable, absolute max_velocity.
- Acceleration_scaling is multiplied by the applicable absolute max_acceleration.

These two scalings are permanently defined in `manipulator.yaml`, but can also be changed temporarily at Mikado runtime. For some robots (e.g. Mitsubishi) this is possible under Tools/Robots. A temporary change is always possible by means of a JavaScript task using *setRobotVelocityScaling* or *setRobotAccelerationScaling*.

Different behavior with different robots

If a robot controller receives a sequence of trajectory points (e.g. a path planned by Mikado), this is processed piece by piece. However, some robot controllers (e.g. Fanuc) have a limited planning horizon. Accordingly, only a limited number of subsequent trajectory points are considered when moving to the next point. The safety-relevant assumption is made that the last point considered corresponds to the end point of the path. Typically, the robot must come to a standstill at the end point of the path. For paths with closely spaced trajectory points, this means that the maximum speed is never increased.

Mikado's free path planner generates a large number of trajectory points to ensure that no collisions occur. In combination with a controller that could also stop at every point approached, this leads to a very slow ride. Due to this cause, nothing can be changed here via the scaling factors. However, the use of the free path planner can be avoided by adding suitable intermediate poses. The direct path (in the joint space) between the individual intermediate poses must be collision-free. Otherwise, the path planner is used again. The following illustration shows an example of an intermediate pose inserted between the start and end pose.

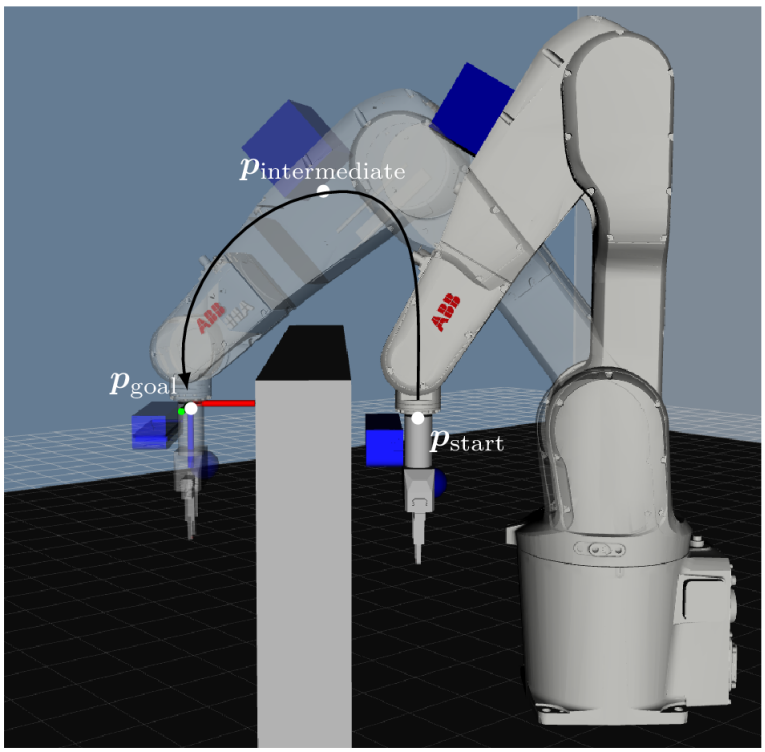


Fig. 6: Intermediate pose inserted as an example to avoid the use of the free path planner

Note: If the robot moves very slowly during freely planned movements, you can avoid activating the free path

planner by adding suitable intermediate poses.

The following links refer to robot-specific speed tuning:

Table 2: **Links to robot-specific speed tuning**

Kuka Roboter:	*Speed and fine-tuning*
Universal Robots:	*Speed and acceleration*

7.4.2 Configuration files

The "config/" folder contains a number of files for setting interface properties, as well as for parameterizing the object search and path planning. Of particular interest are *access_control.yaml*, *mikado.yaml* and *status.yaml*.

access_control.yaml

- *enabled*: (true/false) specifies whether simple access control should be enabled. A password must then be entered for all pages of the interface on which changes can be made. The effect is also described in chapter *Password protection*.
- *password*: (string) Defines the password. This access control should only be understood as protection against accidental access and not as a security mechanism against deliberate manipulation. The password is stored in plain text in the configuration file *access_control.yaml* and can be easily read out. Therefore, no password from another system should be reused.

In Mikado Assistant these settings can be found in section Cell > Application > Access Control.

calibration.yaml

Some parameters for automatic calibration or for generating new recording poses can be set in this file:

- *targetLink*: Name of the link that is used to generate the robot poses. (Optional, default: wrist of the robot)
- *max_translation_before_precalib*: Maximum translation of the generated poses to the initial pose before a precalibration was performed [in meters].
- *max_translation_after_precalib*: Maximum translation of the generated poses to the initial pose after a precalibration has been performed [in meters].
- *rotation_before_precalib*: Rotation of generated poses to the initial pose before a precalibration was performed [in radians].
- *rotation_after_precalib*: Rotation of generated poses to the initial pose after a precalibration has been performed [in radians].
- *check_pose_visibility*: (true/false) This can be used to explicitly deactivate the visibility test after precalibration (optional, default: true)
- *max_failed_moves*: A randomly generated pose can occasionally not be reached by the robot (e.g. in case of collision or no inverse kinematics found). This limit value defines the maximum number of attempts at which the calibration process is aborted.
- *max_failed_visible*: After a successful pre-calibration, each generated pose is checked to see whether the calibration pattern will be in the field of view. This limit value defines the maximum number of attempts the calibration may start per pose. If no pose is found after this, the calibration is aborted.

Hint: The values after pre-calibration should allow more leeway to achieve a better final calibration result.

camera_X.yaml

A parameter file is created for each camera. This contains:

- *camera_serial*: The camera serial number used to establish communication.
- *fixed_camera*: Determines whether the camera is treated as fixed or movable on the robot arm
- *camera_config*: Path to the configuration file loaded at startup (see *Camera configuration*), for example `cellname:config/150001.ensparam`. File cameras do not use this.
- *mono_cameras* (optional): If you want to use an additional monocular camera, it can be integrated here (e.g. for use with *Halcon Scripts*)
  - *serial_no*: Serial number of the monocular camera (e.g. `4101230815`)
  - *config*: Path to the configuration file for this monocular camera loaded at startup, e.g. `cellname:/config/4101230815.ensparam`

Warning: File cameras in a *.zip are not supported and must be unpacked into a folder.

gripper.yaml

In the configuration file *gripper.yaml*, the gripper type is specified at the top with the parameter *type*. There are the following categories:

- *type*:
  - *None:* An immobile "gripper" with no other special features
  - *Magnetic:* a magnetic gripper
  - *Vacuum:* a vacuum gripper, here the vacuum is already switched during the approach movement to avoid displacement by the vacuum cup (especially with bellows suction cups).
  - *ParallelTwoStates:* a simple pneumatic gripper with *open* and *closed* states. Can be used for both internal and external grippers. Of course, it also works with simple electric grippers.
  - *ParallelPositionControl:* an electric gripper that can be controlled to any position.
  - *ParallelForceControl:* an electric gripper with force measurement and force control.

The remaining parameters are listed in the following table. Some parameters are optional depending on the gripper type and may therefore not be part of your configuration file. The *Robot type* column indicates for each parameter whether it exists for a specific robot type. The *Gripper type* column indicates whether a parameter exists for a specific gripper type. Here *IO* denotes grippers controlled by IOs, *Finger* denotes grippers with at least two fingers and *WSG* denotes grippers of the WSG series from Weiss.

Table 3: **Overview of the gripper parameters**

		Robot type	Gripper type
Outputs			
open_hand_signal_number	Signal number (IO number) with which the gripper can be switched to the open state	alle	IO
open_low_active	If the open state of the gripper is low-active, set this parameter to *true*	Mitsubishi, Techman	IO
close_hand_signal_number	Signal number (IO number) with which the gripper can be switched to the closed state	alle	IO
close_low_active	If the closed state of the gripper is low-active, set this parameter to *true*	Mitsubishi, Techman	IO

continues on next page

Table 3 – continued from previous page

Inputs			
is_open_signal_number	Signal number (IO number) at which the open status of the gripper is signaled	Mitsubishi, Techman, UR	IO
is_open_low_active	If the sensor used for the open state is low-active, set this parameter to *true*	Mitsubishi, Techman, UR	IO
is_closed_signal_number	Signal number (IO number) at which the closed state of the gripper is signaled	Mitsubishi, Techman, UR	IO
is_closed_low_active	If the sensor used for the closed state is low-active, set this parameter to *true*	Mitsubishi, Techman, UR	IO
Finger configuration			
joint_finger_name	Name of the joint of the leading gripper finger (e.g. 'finger_left'). Can be set to '' if gripper has no movable joints.	alle	Fingers
joint_value_open	[1] Value of the joint in meters when the gripper is closed (e.g. 0.0). This value is used to control electric grippers with linear axes, for example.	alle	Fingers
joint_value_closed	Page 283, 1 Value of the joint in meters when the gripper is open (e.g. 0.07).	alle	Fingers
Schedule			
wait_time_before_pick	Waiting time in seconds at the transition pose of a gripping process if the gripper state for the grip needs to be changed	alle	alle
wait_time_pick	Waiting time in seconds directly after the gripper has gripped a part	alle	alle
finger_state_publish_rate	Update rate of the finger (or gripper) joint in Hz	all except UR, Techman	alle
WSG			
speed	Speed of the gripper axis in m/s	alle	WSG
acceleration	Acceleration of the gripper axis in m/s^2	alle	WSG
force	Gripping force of the gripper axis in Newtons	alle	WSG

joint_limits.yaml

This file defines the limits for the individual joints. The following parameters can be specified:

- *min_position* minimum Position
- *max_position* maximum position (both in radians, i.e. π (pi) or 3.141 corresponds to 180°)
- *has_velocity_limits* (true/false) speed limits are available
- *max_velocity* maximum speed (wheel/second)
- *has_acceleration_limits* (true/false) acceleration limits are available
- *max_acceleration* maximum acceleration (wheel/second^2)

Unfortunately, the following restrictions apply:

- we currently use a control system for some robot types that is based purely on trajectories (i.e. contains no speed or time components). The maximum speed and acceleration then results from the settings of the robot controller (or its performance).
- At present, planners also do not take long acceleration ramps into account, for example. They optimize purely for the shortest route.

[1] The values for *joint_value_closed* and *joint_value_open* must also be entered in the *URDF* of the gripper as joint limits ('lower' and 'upper')! Use the Mikado Assistant to adapt the URDF of the gripper. Details on the URDF parameterization of joints can be found in section *Joints*.

The values specified here overwrite the theoretical limits of the hardware as specified in the URDF/XACRO in the robot package. We also recommend setting joint limits on the robot controller and setting slightly tighter limits in *joint_limits.yaml*.

manipulator.yaml

- *simulated_robot:* true/false Use simulated robot instead of connecting to the real robot
- *robot_ip:* e.g. "192.168.2.1" Enter the set IP of the robot here
- *manufacturer:* e.g. "Universal Robots" Manufacturer of the robot, activates any special functions in Mikado Control.
- *model:* e.g. "UR5" so far only for documentation purposes
- *velocity_scaling:* (0.01-1.0) Scaling factor to the maximum speed defined in the URDF. Can be temporarily adjusted in the interface under Tools/Robots.
- *max_velocity_scaling:* (0.01-1.0) Upper limit for the temporary value in the surface
- *Acceleration_scaling:* (0.01-1.0) Scaling factor to the maximum acceleration defined in the URDF. Can also be temporarily adjusted in the interface.
- *max_acceleration_scaling:* (0.01-1.0) Upper limit for the temporary value in the surface

mikado.yaml

This file defines some properties of the interface:

- *separate_windows*: false Visualization and user interface in separate windows, distributed across multiple monitors
- *program_file_name*: "project:programs/Motek.yaml" a program sequence that is to be loaded directly (optional)
- *product_list_file_name*: "project:products/T-Stueck_productList.yaml" a product list that is loaded directly (optional)
- *scene_objects_file_name*: "project:scenes/EnvObjects.yaml" a file with the scene objects (optional)
- *debug_qml*: true/false In Qml debug mode, the program interface and in particular the system-specific pages can be reloaded without restarting the program. A new "Qml" tab appears below the visualization for this purpose.

The prefix *project:* means that the information is relative to the project directory, i.e. the directory of the robot cell.

object_recognition.yaml

This file contains parameters for object recognition that apply to all products. Some parameters cannot be influenced via the Mikado interface, as these must already be set at system startup. The following parameters must be set:

- ***voxel_size***
  Voxel size. A 3D point is considered a "voxel" in collision planning, i.e. a small cube. The side length of such a cube can be defined here. [mm] *Default: 2.0*

  Note: The voxel size (octomap) displayed in the view can be changed in *sensor_manager.launch.xml*.

- ***dilation_radius***
  Dilation radius object search. If an object is found, all 3D points around this object are cut out for the

object search. The maximum distance within which the points are removed can be set here. [Number of voxels] *Default: 0*

- ***dilation_radius_object_cutting***
  Dilation radius when cutting out matches. If an object is found, all points around this object are cut out for collision planning. The maximum distance within which the points are removed can be set here. Should be larger than the dilation radius of the object search. [Number of voxels] *Default: 2*

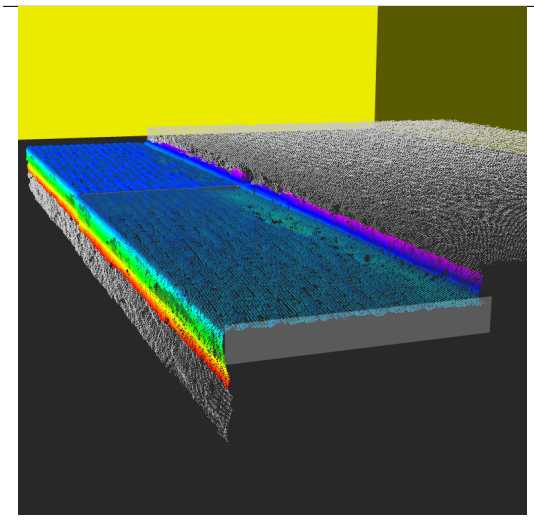


Fig. 7: *dilation_radius_object_cutting: 4*. For a voxel size of 3 mm, this results in a cutout of 4*3 mm = 12 mm.

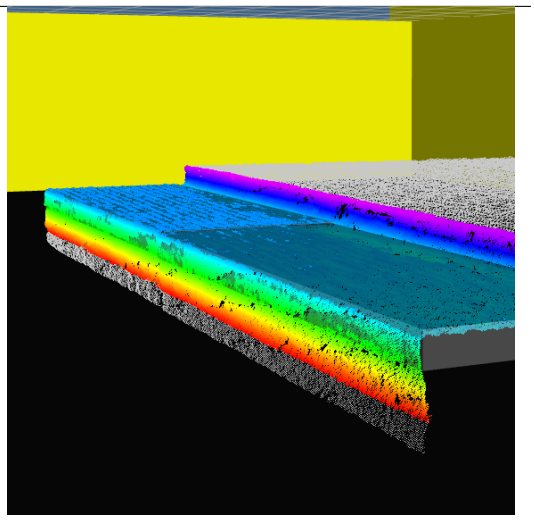


Fig. 8: *dilation_radius_object_cutting: 8*. For a voxel size of 3 mm, this results in a cutout of 8*3 mm = 24 mm.

- ***cuda_enabled***
  Use CUDA. If the Mikado controller is equipped with an NVIDIA graphics card, stereo matching can be accelerated using CUDA. [bool] *Default: true*
- ***useFloat16***
  Accelerates the SCM and SGM calculations (Matching Method/ComputeDisparityMap) if the GPU supports half arithmetic. [bool] *Default: true*
- ***halcon_available***
  Halcon available. If no Halcon license is available, all Halcon-dependent features can be deactivated here. [bool] *Default: false*

Halcon specific parameters:

- ***relative_sampling_distance***
  Relative sampling distance (products). Sampling distance of the surface model (product) relative to its diameter. [0.0 - 1.0] *Default 0.0375*
- ***pose_ref_rel_sampling_distance***
  Relative sampling distance for refinement (products). Sampling distance of the surface model (product) relative to its diameter for the refinement step of the object search. Should be selected larger than the parameter *Relative sampling distance (products)* [0.0 - 1.0] *Default 0.015*

ompl_planning.yaml

Parameters for the available path planning algorithms can be adjusted here. Mikado Control currently uses "RRT-Connect" by default and this cannot be switched.

pick_search.yaml

Some of the following terms are explained in more detail in section *Handle inspection*.

- ***allow_OO_collisions_during_extraction:***
  Activate/deactivate object/octomap collisions
- ***allow_max_depth_collisions:***
  Grasped object may collide with other objects during the removal (extract) movement
- ***max_allowed_collision_depth:***
  Maximum "depth" of a permitted collision
- ***specific_max_allowed_collision_depths:***
  Maximum "depth" of a permitted collision for two specific products/objects
- ***pick_selection_order:***
  Prioritization of the handles can be influenced here by changing the order of the list elements
  - *score_bin_size:* Size of the grouping intervals by score
  - *z_bin_size:* Size of the grouping intervals by z-value in *world* coordinates
- ***search_fn_blacklist:***
  Switch off individual extract or retract functions
- ***allow_skipping_new_object_search:***
  Skip 3D matching for new objects if there are enough possible handles available by reusing previous finds. The following criteria must be met for the object search not to be triggered:
  - *min_valid_picks:* Minimum number of valid picks in total
  - *min_valid_picks_per_object:* Number of handles that must be found on an object for the object to be classified as "grabbable"
  - *min_valid_objects:* Minimum number of "tangible" objects
- ***object_recognition_max_translation_diff:***
  Reuse of finds: Maximum distance of the previous find for it to be classified as the same find
- ***object_recognition_max_rotation_diff:***
  Reuse of finds: Maximum rotation to the previous find so that it is classified as the same find
- ***interpolation_max_translation_diff:***
  Maximum increment of two robot states in interpolated movements (distance)
- ***interpolation_max_rotation_diff:***
  Maximum increment of two robot states in interpolated movements (rotation)
- ***pick_search_timeout_in_seconds:***
  If a task is executed in the Mikado process that has to wait for the result of the handle search (e.g. the task '*Pick object*'), the maximum calculation time defined here applies. If this time is exceeded without a valid handle being found, the process terminates with an error.

sensors_ensenso.yaml

This is where you configure how the point cloud recorded by the camera is to be taken into account for path planning. In the default setting we use, only the last image is taken into account and known geometries such as the robot arm ("selfie") are *not* filtered out. The products found, however, are filtered so that this mode is well suited for use in a crate, for example. If *occupancy_map_monitor/PointCloudOctomapUpdater* is used for *sensor_plugin* instead, each point cloud is interpreted as an "update" of the previous scene. In this way, a collision geometry of the cell is created over several images with a moving camera. (This cannot yet be saved and is therefore only mentioned here as a nice gimmick).

status.yaml

This file defines which components (in addition to some fixed components for cameras and the Mikado Control license) make up the system status. Typically, there will be an entry for the robot, one or more for the cameras and possibly another for fieldbus components such as ProfiNet. Here is an example for a robot and a ProfiNet component:

```
status:
  - name: Robot
    type: RobotStatus
    topic: robot_status
  - name: Profinet
    type: RosDiagnosticStatus
    topic: profinet/status
    enabled_if_rosparam: /profinet
```

The system can easily be supplemented with your own components, e.g. for specific hardware or software. All you have to do is regularly (< 10s) publish a message of the type *RosDiagnosticStatus* on a topic of your choice (google ros publisher or ask us for a sample program).

view_settings.yaml

A complete standard configuration of the views of each 3D visualization in Mikado. For example, the position and orientation of the scene camera are saved here. Information on using the save function for the view settings can be found in the *Navigation* chapter.

world.srdf

This file contains some additional information on the structure of the cell that cannot be specified in the *Cell geometry*. Among other things, all exceptions to collision monitoring are listed here. By default, Mikado allows collisions for the following categories:

- all objects attached to *tool0* may collide with each other,
- all objects attached to *workcell_base* may collide with each other,
- all neighboring links of the robot may collide in pairs and
- the group of all objects attached to *tool0* may collide with Link_5 and Link_6 of the robot.

This speeds up the system, e.g. with stationary objects, but may also be necessary if, for example, the collision geometries of two neighboring sections of the robot constantly or temporarily overlap minimally. Further exceptions can be permitted via the "Add new exception" field. To delete a user-defined exception, select the corresponding entry in the list and press the "Delete" button. Only user-defined exceptions can be deleted.

Note: If a new object is attached directly to a robot link, Mikado does not generate an automatic exception to the collision check, as none of the above categories cover this case. If the new object and parent (robot) link overlap,

the robot can no longer be moved by Mikado because a collision has been detected (parent object and the new object). In this case, the exception must be created manually.

Warning: The collision check between the parts listed here is completely deactivated. Therefore, only deactivate the collision check for parts that cannot collide in reality. Careless deactivation can result in serious material damage. Therefore, make sure that deactivated collisions cannot occur, for example via maximum axis angles (see *URDF*)!

Here are two examples:

```
<disable_collisions link1="base_link" link2="link_1" reason="Adjacent" />
<disable_collisions link1="base_link" link2="pane_back" reason="Never" />
```

The collision between "base_link" (the base of the robot) and "link_1" (the first segment of the robot) is deactivated, as they touch slightly and are only connected by a vertical axis of rotation. They can therefore never be brought into an actual collision regardless of robot movements. The collision between "base_link" and "pane_back" (rear wall of the cell) is deactivated as both are fixed.

The entries for the *gripper_group* and the *disable_collisions* table are also interesting. The *gripper_group* determines which parts of the gripper are shown in the gripper point editor. Here is an example from the template:

```
<group name="gripper_group">
    <link name="gripper_frame" />
    <link name="camera_frame" />
    <link name="gripper_cables_frame" />
    <link name="gripper_tcp" />
</group>
```

7.4.3 Launchfiles

The individual software components of a Mikado system are started via so-called launch files. These are XML files with a relatively easy-to-read structure. If you have used the Mikado Assistant, no changes to the launch files should be necessary. An exception is, for example, changing the robot type at a later date. Each cell has a file *launch/main.launch* which describes the actual launch process of the system. Optional arguments can be passed to this file, e.g. "sim:=true" to start Mikado Control in simulation mode. *main.launch* is typically called via the bash script "zellname.bash", which in turn is started via "zellname.desktop".

The following description of the possible entries in a launch file is deliberately brief; for further information, please refer to the ROS wiki (http://wiki.ros.org/roslaunch/XML).

- <arg> a parameter *name* within the launch file that can be specified when the launch file is called; if it is not specified, *default* is used. Is then used later via "$(arg name)" either in case distinctions (if/unless) or as a value.
- <group> can be used to group <node>s and parameters, either to make a complete block optional via if/unless or to move the nodes and parameters into a namespace.
- <include> loads another launch file; its parameters can also be set via <arg>
- <node> starts a program under the unique name *name*, it originates from the package *pkg* and has the file name *type*. If the program terminates as planned or due to a crash, it can be restarted using *respawn="true "*. However, if *required="true "* is specified, the entire launch file will be terminated in this case.
- <param> sets a single parameter on the central parameter tree.
- <rosparam> is usually used to load an entire configuration file (.yaml) onto the parameter tree. You can also use *ns* to write to a "subfolder" of the tree.

sensor_manager.launch.xml

The resolution of the octomap displayed can be set here, which can be particularly relevant for systems without a high-performance graphics card. The octomap resolution displayed can be adjusted in the following line:

```
<param type="double" name="octomap_resolution" value="0.001"/>
```

The following screenshots show the difference between a 1 mm resolution and a 4 mm resolution.

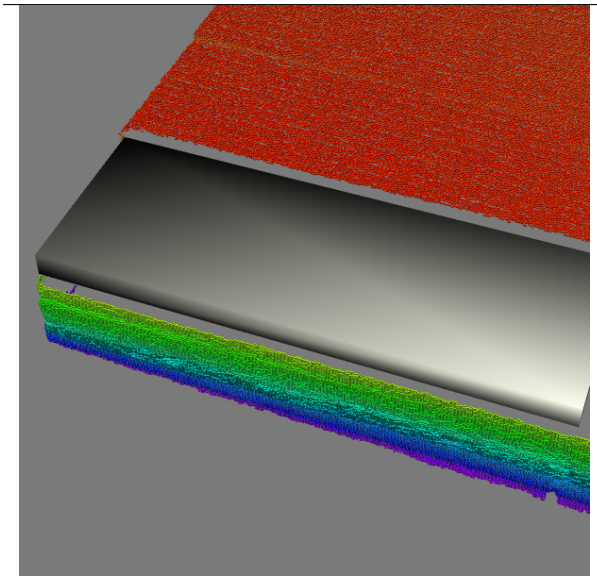


Fig. 9: *octomap_resolution = 0.001*

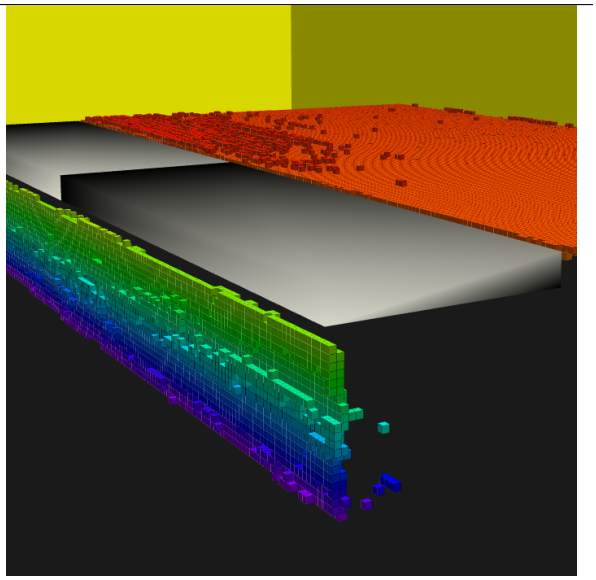


Fig. 10: *octomap_resolution = 0.004*

Note: The resolution of the octomap has nothing in common with the voxel size defined in *object_recognition.yaml*.

7.4.4 Cell geometry

The program-independent geometry (and kinematics) of the robot cell is described via a file in the Unified Robot Description Format (*URDF*). This format is an XML dialect and can be edited with any text editor.

Mikado organizes the URDF description of the robot cell in a total of five different files: *Global*, *Cell*, *Gripper*, *Robot* and *Camera*, whereby changes are only possible in the first three files mentioned. The modifiable URDF files are stored in the *urdf* subdirectory of the project directory. This subdivision helps to keep the individual modules of a robot cell clear and easy to replace.

The figure below shows the hierarchical structure of the file structure. The *Global* file integrates the other four files. The *J* (for joint/joint) indicates the file in which the geometric connection between the corresponding modules is defined.

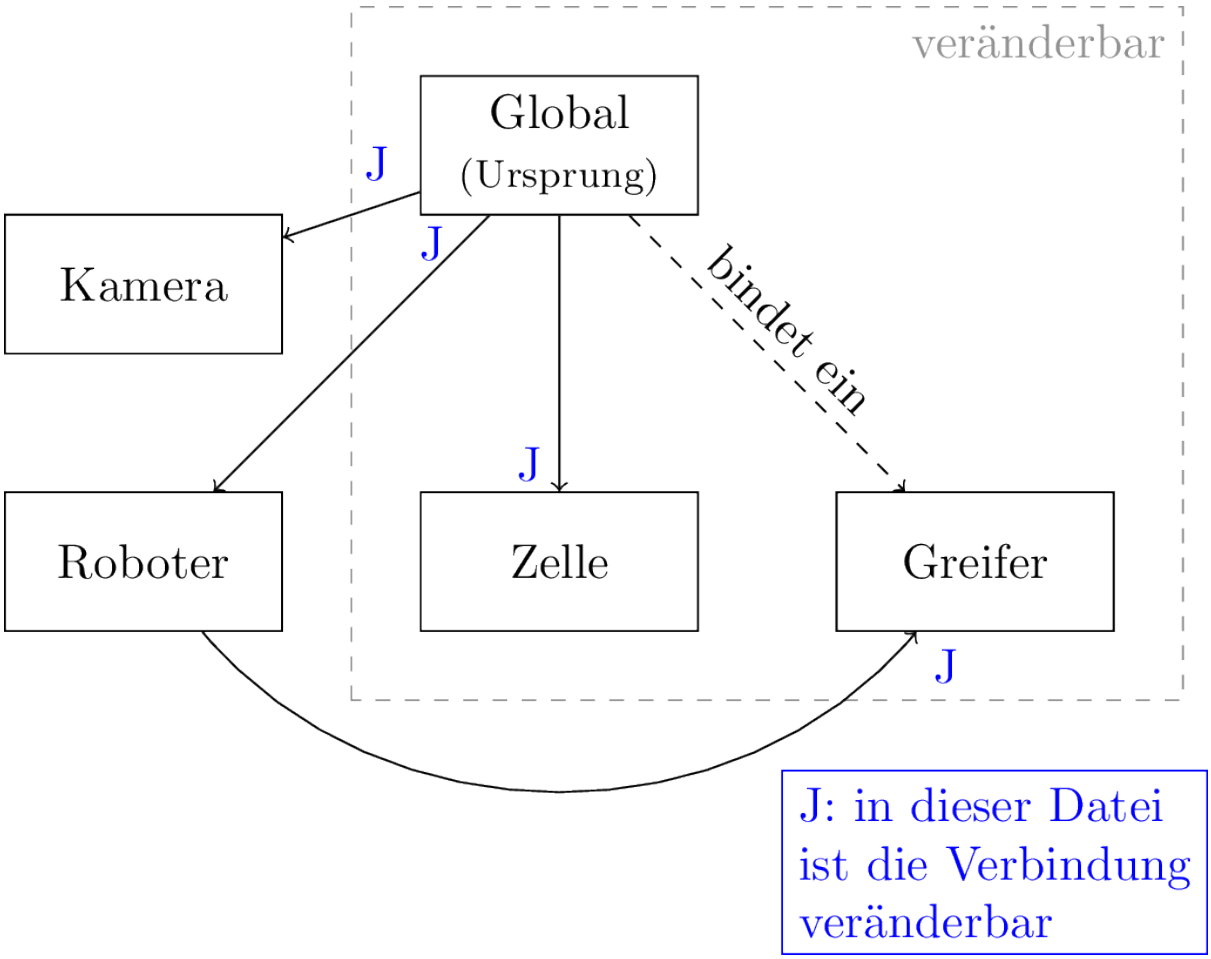


Fig. 11: Organization of the URDF files

The following table provides a brief overview of the intended content of the individual files:

Table 4: **Overview of the individual URDF files**

Global	The global area connects the robot, the cell and the gripper. It can be used to add further components to the robot.
Cell	All objects that relate to the static work cell are defined in the Work cell section. Only add stationary objects here that are not connected to the robot.
Gripper	All objects that are connected to the gripper are defined in the Gripper section. Only add objects here that are mounted directly or indirectly on the flange of the robot. The tool center point (gripper_tcp), which is decisive for the grip definition, is also defined here.
Roboter	The geometry of the robot is defined here. The content of this file is defined by the manufacturer and cannot be changed.
Camera	The geometry of the camera is defined here. The content of this file is defined by Optonic and cannot be changed.

With the Assistant, Mikado offers various tools to modify the cell geometry in the most user-friendly way possible. In the following a short introduction to *URDF* is given and the possibilities to structure it with *XACRO* are shown. Furthermore, the *URDF Insert Help* and the *URDF editing help* are introduced.

URDF

The Unified Robot Description Format (URDF) description of a robot cell begins with an enclosing <robot> tag. This has an attribute *name*, which must match the tag in the associated *world.srdf*. This is followed by a series of <link> and <joint> tags. A <link> represents a node, an object or less abstractly e.g. the table/base plate of the cell. <link>s are connected to each other via <joint>s. These can be either fixed or movable and form a tree structure. We always start this with the root object <link name=”world_frame”/>. This must always exist and must not be renamed! The figure below shows an example cell described with URDF. The coordinate systems of the individual links are represented by the three-colored coordinate crosses with a corresponding white label. The joints represented by yellow arrows show how the individual links are connected to each other.

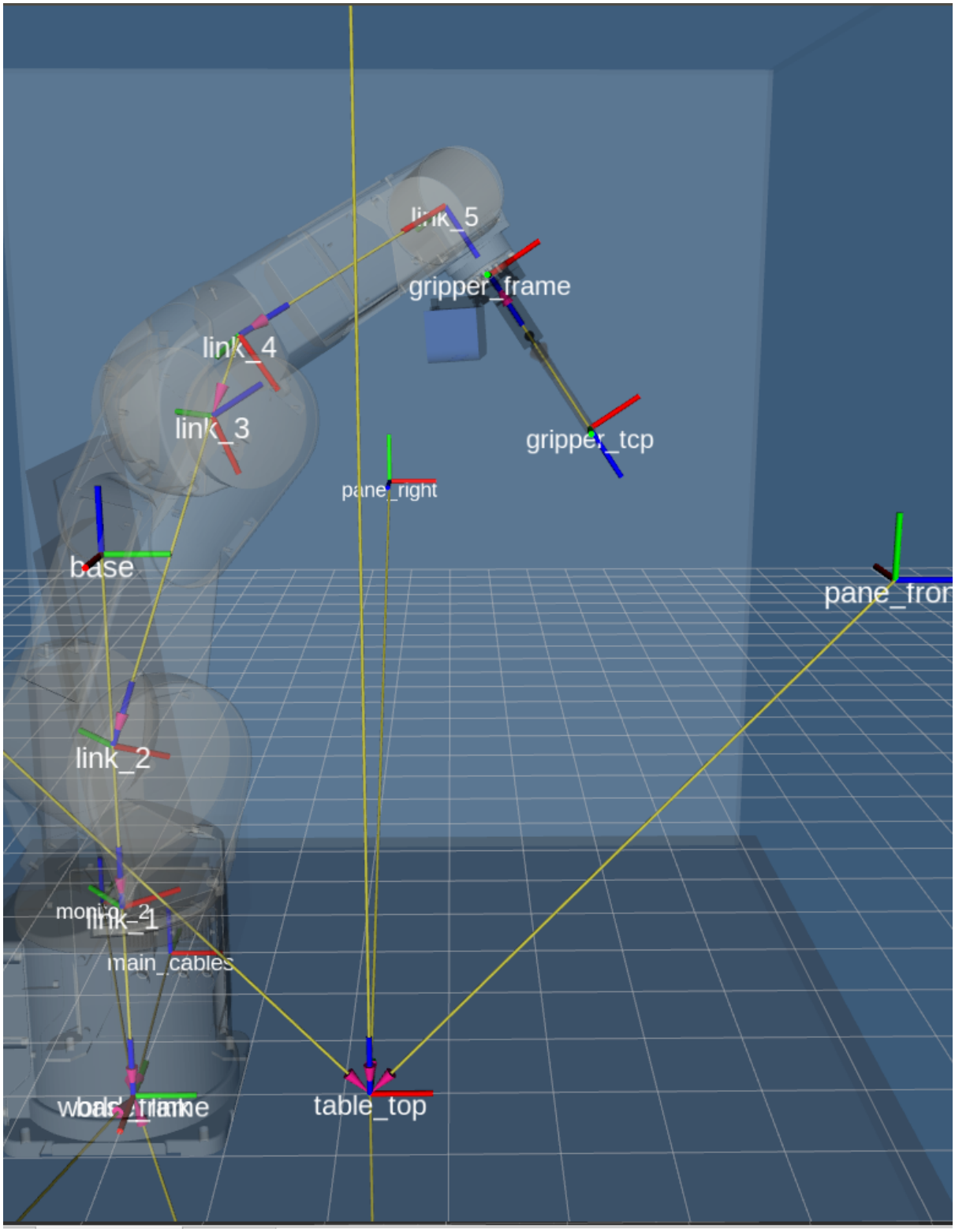


Fig. 12: Visualization of a robot cell described with URDF

Links

A small example explains the structure of a link:

```
<link name="mein_link">
   <visual>
     <origin xyz="0 0 0" rpy="0 0 0" />
     <geometry>
       <mesh filename="package://mein_projekt/meshes/mein_link.stl" />
     </geometry>
     <material name="Cyan">
       <color rgba="0 1.0 1.0 1.0"/>
     </material>
   </visual>

   <collision>
     <origin xyz="0 0 0" rpy="0 0 0"/>
     <geometry>
       <cylinder radius="1" length="0.5"/>
     </geometry>
   </collision>
</link>
```

The “my_link” node consists of a CAD/mesh file that describes the visual appearance and has been colored in cyan/turquoise (the name is freely selectable, the RGBA values are decisive). Mesh files must be available either as STL (simple triangles, no color information) or as Collada (.dae, textures etc) and in meters. The visual geometry can be moved via <origin xyz=”0 0 0” rpy=”0 0 0” />, but it is clearer to move the CAD file appropriately (e.g. in Meshlab, SolidWorks, Blender, . . .). The simplest possible description should be used for the collision geometry, but it must always be the same size or larger than the actual physical geometry. A cylinder with a radius of one meter and a height of half a meter was simply used here. The origin is in the center of the cylinder. If the origin of a URDF primitive is shifted, it should be noted that this shift always takes place in the original coordinate system of the primitive. The following image illustrates the shifting of the origin of a URDF primitive.

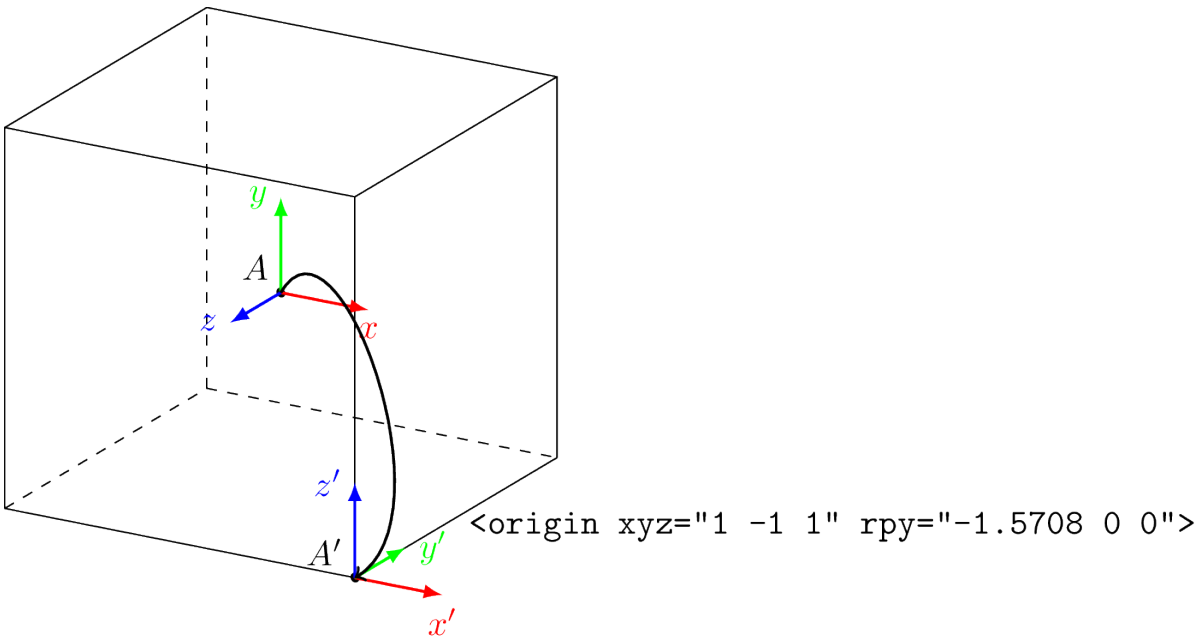


Fig. 13: Moving the origin of a URDF primitive

The primitives <box>, <cylinder>, <sphere> and CAD models <mesh> can be used for both visual and collision geometry:

```
<box size="0.1 0.2 0.3" />
<cylinder radius="0.1" size="0.2" />
<sphere radius="0.3" />
<mesh filename="package://mein_projekt/meshes/beispiel.dae" scale="0.001" />
```

It is quite possible to use multiple geometric objects in a link, but this should be avoided as it is insufficiently tested and therefore not officially supported. This applies to both collision geometries and visual geometries.

Joints

The individual links are connected to each other via joints. In the simplest case, this is a rigid connection:

```
<joint name="world_to_table" type="fixed">
  <parent link="world_frame"/>
  <child link="table_base"/>
  <origin xyz="0.3 0 -0.5" rpy="0 0 0"/>
</joint>
```

This defines that a link with the name “table_base” is moved by 30cm in the x-direction and by -50cm in the z-direction. It should be noted that the shift specified in “<origin>” node is always interpreted in the coordinate system of the parent object. The following image illustrates the displacement in a joint.

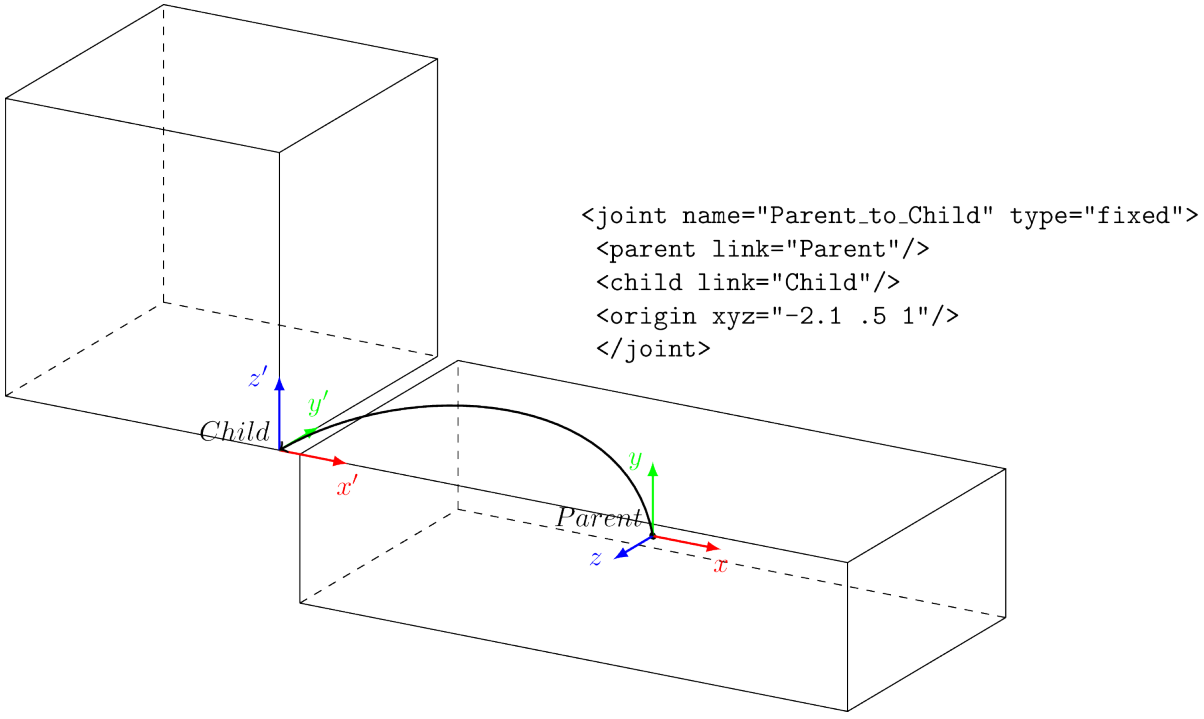


Fig. 14: The importance of displacement within a joint

Industrial robots typically have 6 axes of rotation, such an axis would then be described by:

```
<joint name="joint_3" type="revolute">
   <origin xyz="0 0 0.310" rpy="0 0 0"/>
   <parent link="link_2"/>
   <child link="link_3"/>
   <axis xyz="0 1 0"/>
   <limit lower="${-16*deg_to_rad}" upper="${164*deg_to_rad}" velocity="${250*deg_to_
```

(continues on next page)

(continued from previous page)

```
↪rad}" effort="0" />
 </joint>
```

The type “revolute” indicates that this is a rotation joint. Its origin has been shifted by 31cm from link2 and it is rotated around the y-axis. As the robots themselves are typically modeled by Optonic, the last entry <limit> is of particular interest here. *lower* specifies the minimum value that the joint can assume, *upper* the maximum. These values must be specified in radians/arc measure. To improve comprehensibility, XACRO has been used here to convert from degrees. The limits do not have to be symmetrical and often are not.

Warning: The limits should also always be set in the controller as Mikado Control is not safety-relevant. It is important to set the limits in the controller approx. 1° larger than in Mikado Control (3° for UR). If Mikado Control attempts to move to a position that is not permitted in the controller, this usually results in very unpleasant emergency braking that has to be acknowledged manually (for safety reasons). If the limits are set identically, this can still occur due to rounding errors.

velocity specifies the maximum achievable speed of the joint in radians per second. This should also be set slightly lower than the maximum speed configured in the controller. In principle, *effort* specifies the maximum force of the motor, but is not used in Mikado Control. The maximum acceleration can be configured via the file *joint_limits.yaml*. To avoid confusion, the same values should be specified there.

Linear joints are also used in grippers in particular, here is an example:

```
<joint name="finger_left" type="prismatic">
    <parent link="gripper_frame"/>
    <child link="finger_left_frame"/>
    <origin xyz="0.0035 0 0.2045" rpy="${m_pi} 0 0"/>
    <axis xyz="1 0 0"/>
    <limit lower="0" upper="0.006" effort="${max_effort}" velocity="0.01" />
</joint>
```

Here, “prismatic” indicates a linear displacement, in this case along the x-axis. The limits now indicate meters instead of radians. Especially with a parallel gripper, you want to move both fingers together. There are so-called “mimic” joints (pantomime joints) for this purpose. The right finger could then be coupled to the left finger as follows:

```
<joint name="finger_right" type="prismatic">
    <parent link="gripper_frame"/>
    <child link="$finger_right_frame"/>
    <origin xyz="-0.0035 0 0.2045" rpy="${m_pi} 0 ${m_pi}"/>
    <axis xyz="1 0 0"/>
    <limit lower="0" upper="0.006" effort="${max_effort}" velocity="0.01" />
    <mimic joint="finger_left"/>
</joint>
```

The joint is therefore identical to the left finger except for an additional rotation of 180° around the z-axis. The <mimic> tag causes the position of the joint to be linked to that of “finger_left”. Optionally, a *multiplier* and an *offset* could also be applied to the value.

The official documentation for URDF can be found at http://wiki.ros.org/urdf/XML/model. However, not all tags and joint types mentioned there are officially supported by Mikado Control.

XACRO

If you open the urdf folder in Dolphin (file manager), you will notice that there are only files with the extension *.xacro*. XACRO stands for **X**ml M**acro** and is a tool for creating flexible and compact XML that is modular and does not constantly repeat itself. If you write the complete cell description including robots in a URDF file, it becomes quite large and it is cumbersome to change the size of the cell, for example. We therefore use a macro language that extends URDF with the option of defining variables, calculating values (e.g. degrees in radians) and integrating additional files. A pure URDF file is then automatically generated from this XACRO file in the launch file. To create a URDF file manually, use the command:

```
xacro --inorder datei.xacro > datei.urdf
```

This can be particularly helpful for testing a change. Now to some of the functions of XACRO:

Calculations

Variables and the basic arithmetic operations can be used:

```
<xacro:property name="diameter" value="4.3" />
<sphere radius="${0.5 * diameter}" />
```

Here, a sphere with a diameter of 4.3m is created by halving the diameter and using it as the radius. Some constants like *pi* are already defined and even arbitrary Python code can be used. Another example (but there is no circle tag in Mikado):

```
<xacro:property name="R" value="2" />
<xacro:property name="alpha" value="${30/180*pi}" />
<circle circumference="${2 * pi * R}" pos="${sin(alpha)} ${cos(alpha)}" />
```

It always makes sense to define a variable *deg_to_rad* and then use the much more intuitive degree values:

```
<xacro:property name="deg_to_rad" value="${1.0/180*pi}" />
```

URDF Insert Help

To facilitate the insertion of Urdf primitives, the Mikado Assistant offers insertion help. This can be displayed for the three Urdf pages *Cell*, *Gripper* and *Global* using the “Insert help” checkbox at the bottom left of the window. The following screenshot shows a completed Urdf input help:

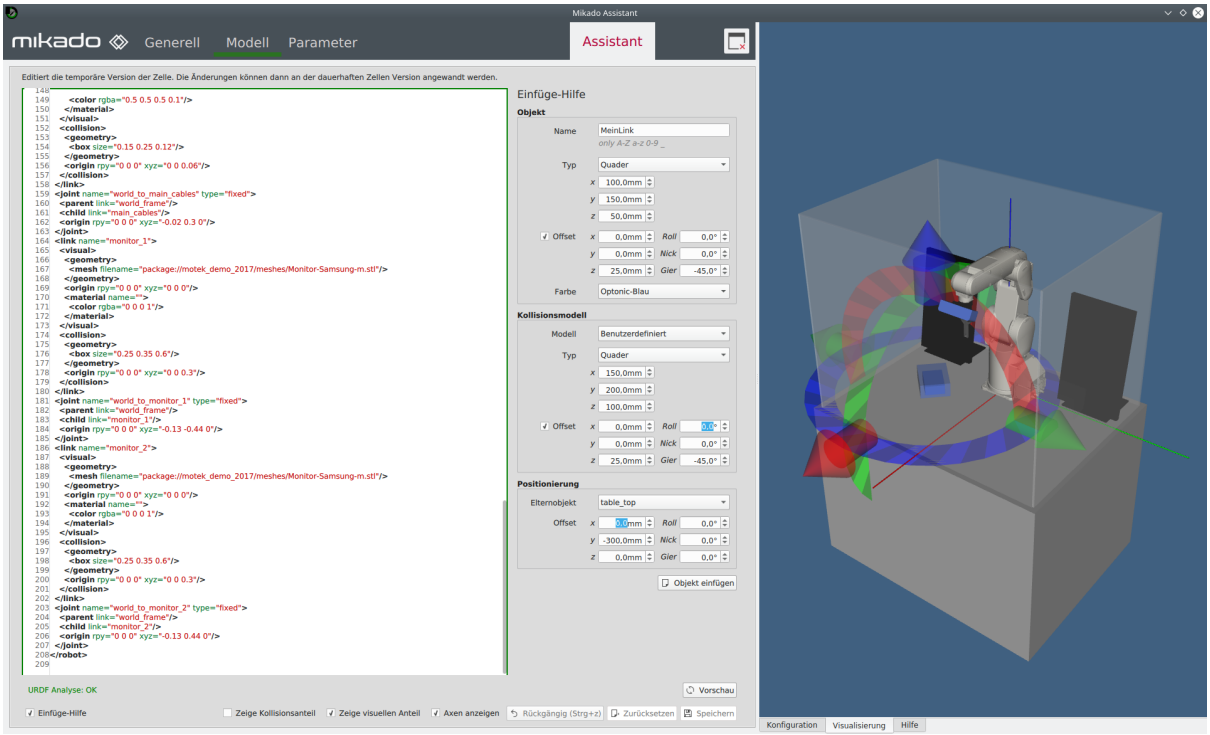


Fig. 15: Screenshot Assistant URDF Insert Help

In the example shown, a cuboid with the dimensions 100x150x50 mm is to be inserted at the bottom of the cell. To do this, the z-value of the origin is set to 25 mm using the “Offset” button in the “Object” box. As the cuboid in this example is positioned at an angle in the cell, the “Yaw” input field is set to -45°.

A cuboid should also be used as the collision model, but with slightly larger dimensions. This can be done in the “Collision model” box in the same way as the “Object” settings.

In the “Positioning” box, you can set which parent object the new object should be attached to. All objects currently contained in the cell are offered for selection via the drop-down menu. In this example, “base” is selected as the new cuboid is to stand on the floor. The “Offset” buttons can be used to set the relative displacement and rotation of the origins of the parent object and the new object.

The visualization on the right-hand side shows a preview of the new object. If the user presses the ‘Insert object’ button, the following code block is generated for our example and inserted into the Urdf editor on the left-hand side:

```
<link name='MeinLink'>
  <visual>
    <geometry>
      <box size='0.1 0.15 0.05'/>
    </geometry>
    <material name=''>
      <color rgba='0.255 0.412 0.882 1'/>
    </material>
    <origin xyz='0 0 0.025' rpy='0 0 -0.785398'/>
  </visual>
  <collision>
    <geometry>
      <box size='0.15 0.2 0.1'/>
    </geometry>
    <origin xyz='0 0 0.025' rpy='0 0 -0.785398'/>
  </collision>
</link>
```

(continues on next page)

(continued from previous page)

```
<joint name='table_top_to_MeinLink' type='fixed'>
  <parent link='table_top' />
  <child link='MeinLink' />
  <origin xyz='0 -0.3 0' rpy='0 0 0'/>
</joint>
```

URDF editing help

The URDF editing help enables an exclusively graphically based modification of the cell geometry. The left-hand side of the screenshot below shows the tree structure of the URDF objects. A URDF object can be selected here. The selected object can then be modified using the mask on the right-hand side. Similar to *URDF Insert Help*, the visual and collision model as well as the positioning of the object can be set using the corresponding buttons. An object can also be renamed or its color changed.

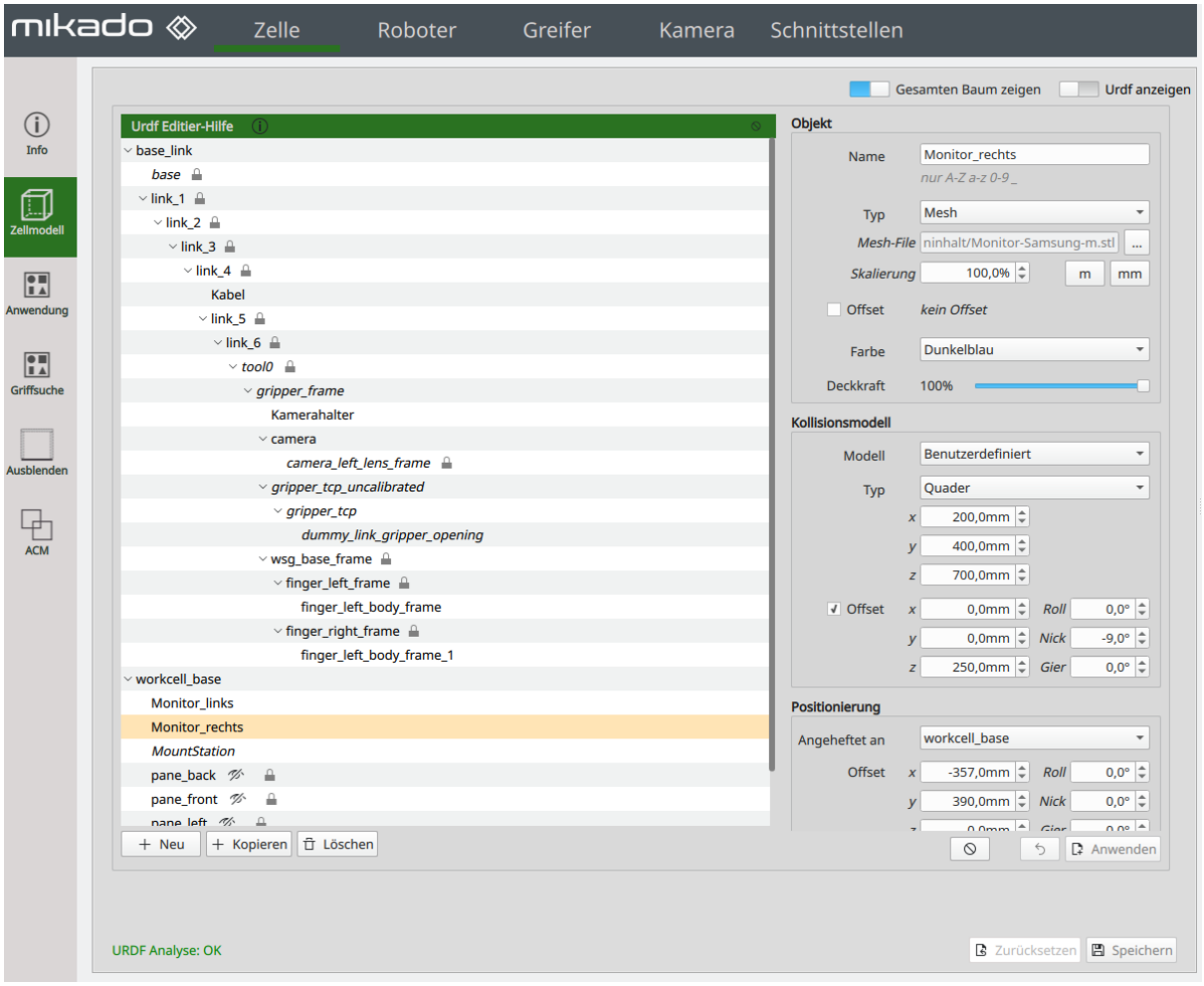


Fig. 16: Screenshot Assistant editing help

Changes to a URDF object must be confirmed by pressing the “Apply” button. If the button is not pressed, the changes to the current input mask will be lost when you select another object or leave the page. To save the changes permanently, press the “Save” button at the bottom right.

In the example shown, the different objects have different font positions and symbols. The following table provides a brief overview of their meaning.

Table 5: **Editing help legend**

Upright font	The object has a collision geometry.
slanted font	The object is virtual. It has no collision geometry. Please note that it can still have a visual geometry.
	The object cannot be edited. There may be various reasons for this: - Das Objekt wird aus einem anderem Ros-Paket eingebunden. - Die Editier-Funktion ist durch das Setzen des Attributs <`link name="example" editable="false"`> manuell gesperrt. - Das Objekt wird durch ein xacro-Makro erzeugt.
	The object is hidden in the simplified view. The configuration of the simplified view can be found in the menu under *Hide*.

Mikado Key-Links

The following Urdf objects are required by Mikado:

- `gripper_tcp` Gripping point of the gripper. Is used for the grip definition and the gripping process. A gripping pose is always approached in the positive z-direction of the coordinate system of this object.
- `tool0` Robot flange. Is set in the respective robot package. This object cannot be changed.
- `camera_left_lens_frame` Position of the left mono camera. Only used by the Mikado simulator.

7.5 Directory structure

In general, a Mikado system consists of a general system-unspecific part that is already pre-installed (as a Debian package) and the specific description of your cell. Several cells can be set up side by side on one PC, but only one application should be started at a time. Additional cells, even with other robot types, for example, can be used to simulate a planned new system.

All existing cells can be found in the `mikado/cells` folder in the `/home/mikado/mikado/cells` folder as subfolders. A cell with the name `example cell` has the following structure:

- `config/` contains configuration files for the interface as well as for object search and path planning parameters
- `launch/` contains information on which software components the system is made up of and how these are to be started
- `masks/` contains masks (images), e.g. for searching for scene objects via their contour
- `meshes/` contains 3D model data of the products, crates, obstacles etc. used.
- `products/` contains the product descriptions and product lists
- `programs/` contains the actual program sequence as it was created in the program editor
- `scenes/` contains descriptions of the moving objects in the cell, i.e. boxes, obstacles and ROIs
- `urdf/` contains the description of the unchangeable components of the cell, i.e. robot, gripper, safety fence, ...
- `package.xml` defines the name of the cell, for the sake of simplicity it should be the same as the folder name

This structure should be adhered to in order to guarantee Mikado's ability to run.

7.6 Mikado Select/Control assembly guideline

7.6.1 Mikado Controller Hardware

The Mikado Control controller is available in a compact and a larger, more powerful 19-inch version. Both controllers should be installed in a protected environment, e.g. in a control cabinet.

19-inch variant

The powerful version of the Mikado controller is supplied as a 19-inch rack with 4 U (approx. 18 cm high). The power switch and two USB ports are located on the front behind a lockable flap. All other connections are located on the rear. Power is also supplied via a 110-240 V power cable. Mouse and keyboard can be connected via USB. There are 3x DisplayPort, HDMI and DVI connections for monitors. Cameras and robots can be connected via four Gigabit network ports on a plug-in card. A further onboard Gigabit port is available for remote access or integration into the overall system. The Hilscher cifX card can also be used here for communication with the PLC etc.

Compact version

The dimensions of the controller are approx. 250 x 160 x 160 mm, but there should still be approx. 5 cm space to the front for the cables. Mounting is possible using the supplied plates with a hole pattern of 270 x 135 mm. Power is supplied via a 110-240 V power cable, keyboard, mouse and screens (DVI/HDMI) can be connected via the usual connections. Two Gigabit Ethernet ports are available for the connection to the camera and robot. The plug-in card is a Hilscher cifX card that can be used for ProfiNet integration, for example. Alternatively, a 4-port network card can also be installed.

7.6.2 Connecting the Ensenso cameras

Only a brief summary can be given here, further details can be found in the Ensenso Manual.

N-Series

The compact cameras in the N series can be connected via PoE (Power over Ethernet) as well as via a separate power supply (24 V). In both cases, data is transmitted via Gigabit Ethernet. This requires a sufficiently shielded cable, at least Cat 5e. Unfortunately, many robots are only equipped with a 100 MBit cable without sufficient shielding. With transmission times in the range of seconds, a minimum cycle time can no longer be achieved.

The mounting should be done with three M4 screws in a triangular arrangement to avoid distortion of the camera body. The drilling pattern and the dimensions of the camera can be found here in the Ensenso manual.

X-Series

The X series is always powered via an external 24 V power supply unit. Power supply via PoE is not possible here. As the projector consumes a relatively high amount of power during recording, a cable with a larger cable diameter should be used for any extensions. The mono cameras are connected to the projector and supplied with 12 V from it.

There are two options for data transmission with the X series:

- The two cameras can be connected to the projector's integrated switch. The switch in turn is connected directly to the Mikado controller.
- Alternatively, only one of the two mono cameras can be connected to the switch. The other camera and the switch are then connected directly to the Mikado controller.

The second variant doubles the data transfer rate and the recording time can be reduced accordingly. The projector is mounted with four M4 screws, the drilling pattern and dimensions can be found here in the Ensenso manual.

7.6.3 Fastening the calibration plate

For hand-eye calibration (HAK), an Ensenso calibration pattern must be attached to the flange or gripper in the case of a stationary camera. With a moving camera, however, this can simply be placed in the cell.

Stationary camera

In principle, the position and axis angle of the robot during calibration should be as close as possible to a typical gripping position. As can be seen in the figure below, axis 6, i.e. also the flange, should be aligned in the negative Z direction and the calibration plate (pattern) in the direction of the camera. Furthermore, the mounting of the calibration plate should be sufficiently stable and not move, but the exact position is irrelevant. The boom for the camera must be long enough so that the camera has a good view of the entire pattern or at least the majority of it. Last but not least, an uncomplicated conversion is helpful so that recalibration can be carried out quickly and easily at any time.

The simplest design is an elongated plate with drill holes for the robot flange and a glued or clamped calibration plate next to the flange. The disadvantage here is the need to dismantle the gripper. The pattern must not be drilled into, as there is a risk that it will warp. The pattern is a measuring instrument and should also be protected from scratches after use.

If the gripper has a stable form fit, a part with a glued pattern can also be gripped. Attaching a section of top-hat rail to the side of the gripper and gluing a corresponding counterpart to the pattern has also proven successful.

Hint: Provide a backstop for the ceramic patches in case the gripper is opened unintentionally.



Fig. 17: Example of attaching a calibration plate to the robot flange for a stationary camera

Moving camera

With a mobile camera attached to the robot flange or gripper, the pattern can be placed anywhere in the workspace. It is best to place it in the center of the subsequent pick-up area, e.g. on or in the parts box. The pattern does not need to be fixed in place, it simply needs to rest in a non-slip position during the robot movements during the calibration process.

7.7 Gripper integration

This chapter on gripper integration shows the options offered by Mikado Control for integrating your gripper. Supported gripper types and how you can integrate and configure the CAD model of the gripper are explained.

7.7.1 Gripper types

Mikado Control supports several gripper types such as parallel, vacuum, magnetic or electric grippers. A list and description of the gripper categories can be found in section *gripper.yaml*.

7.7.2 Integrate gripper

As soon as you have attached a gripper to the robot, you can add it to the robot model. You will need a suitable CAD model of the gripper for this.

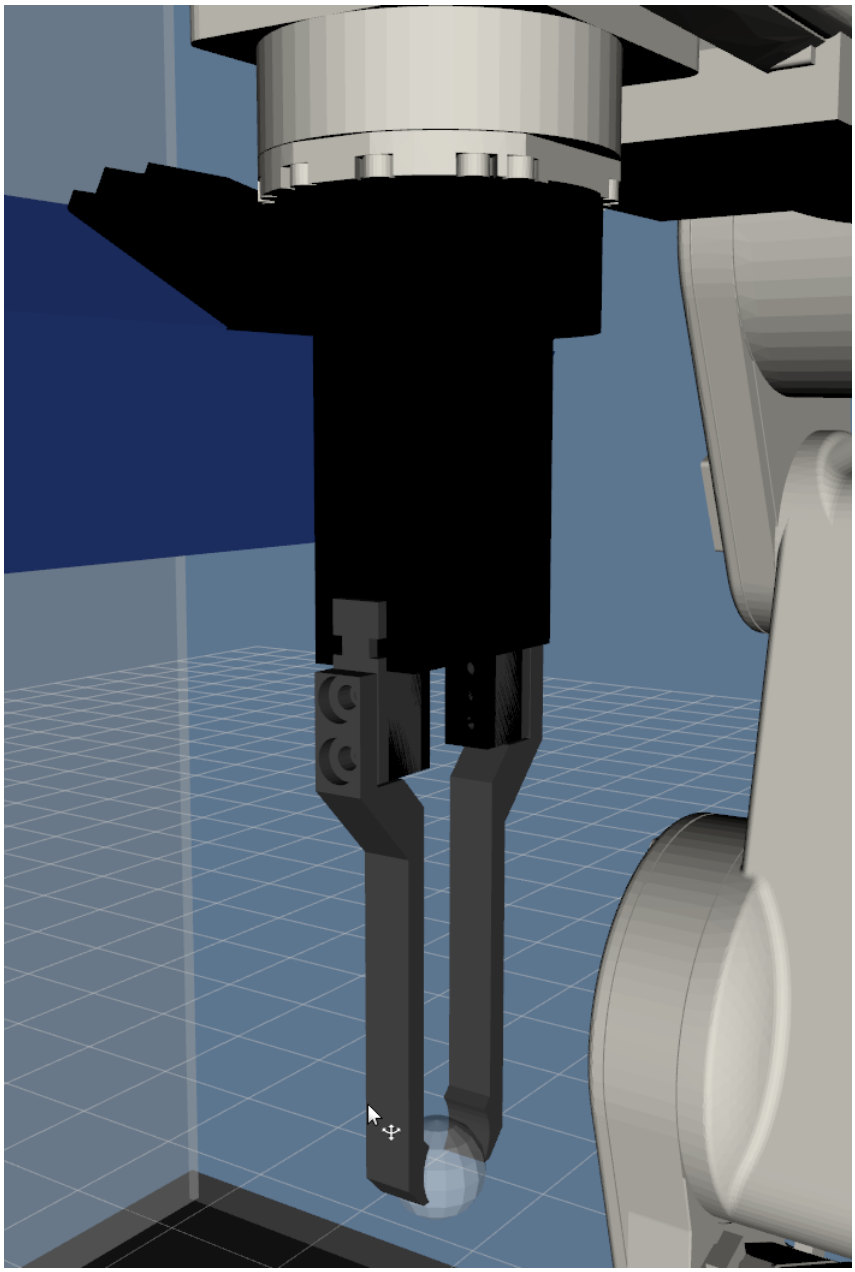


Fig. 18: Integrated gripper in the Mikado visualization (example)

Modify gripper

If you only want to make minor changes to the gripper model of an already integrated system, you can simply modify the integrated STL file with a CAD tool of your choice. The *Mikado Assistant* gives you access to the URDF file that defines your gripper. There you can see which STL files are used and adapt the models.

Hint: Modify existing CAD models instead of creating completely new ones, so you no longer have to worry about scaling or positioning the gripper.

Prepare CAD

A model of the gripper is required so that Mikado Control can carry out gripper and path planning. You therefore need a CAD model of the gripper in STL file format. Make sure that you reduce the number of triangles to the necessary minimum. An extremely high number of triangles can greatly delay the collision checks and thus lead to long execution times. Reduce the model so that all structures lie within the model but are still precise enough for the gripping processes. It usually makes sense to approximate the base body more closely than the gripper jaws, for example.

If you are using a gripper type with moving elements (e.g. a parallel gripper), you will need the individual parts as STL files. A parallel gripper therefore consists of three STL files: Base unit, left gripper finger (or jaw) and right gripper finger. If the gripper has no moving elements (vacuum or magnetic gripper), one STL file containing the complete gripper is sufficient.

Now save the files in the *meshes/* directory of your cell project directory.

Hint: If the gripper fingers are symmetrical, you can use the same model for both gripper fingers. They then only differ in their positioning on the robot.

The example image of the Mikado visualization above shows a parallel gripper consisting of the following components:

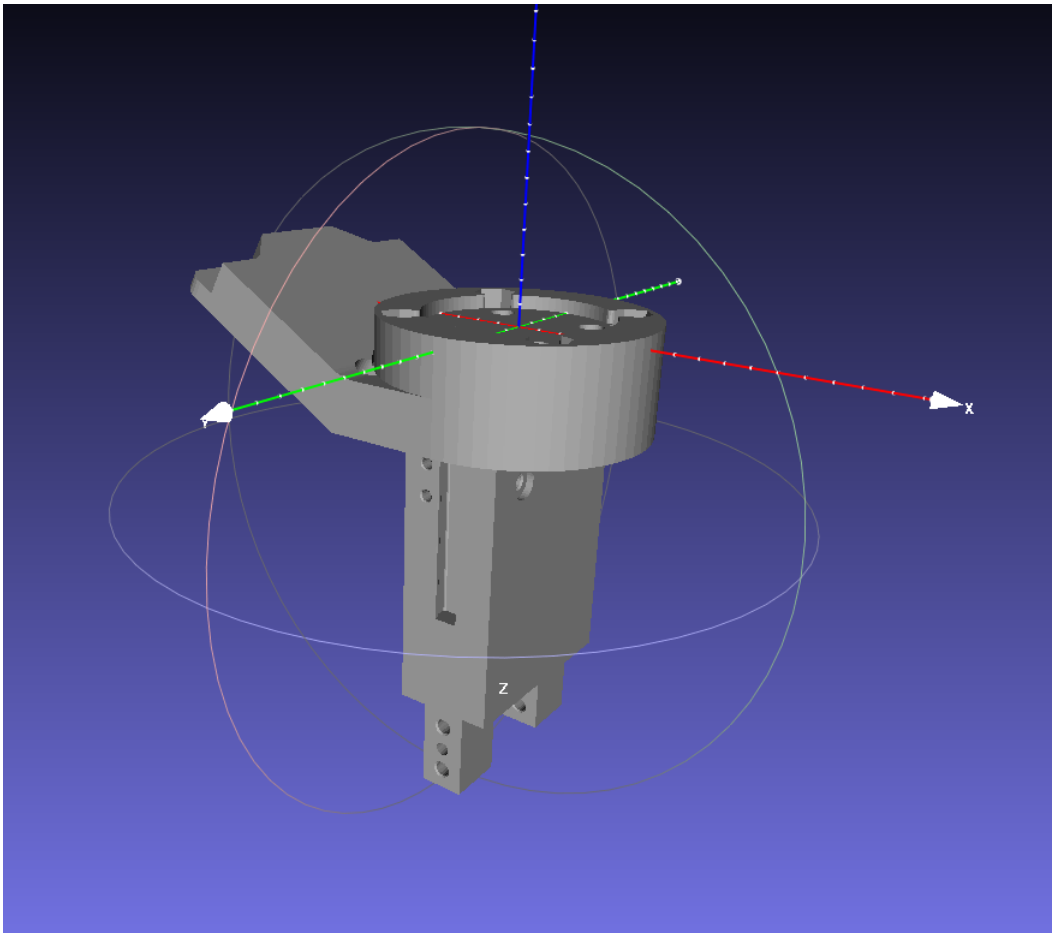


Fig. 19: Basic geometry of the example gripper (shown in black above)

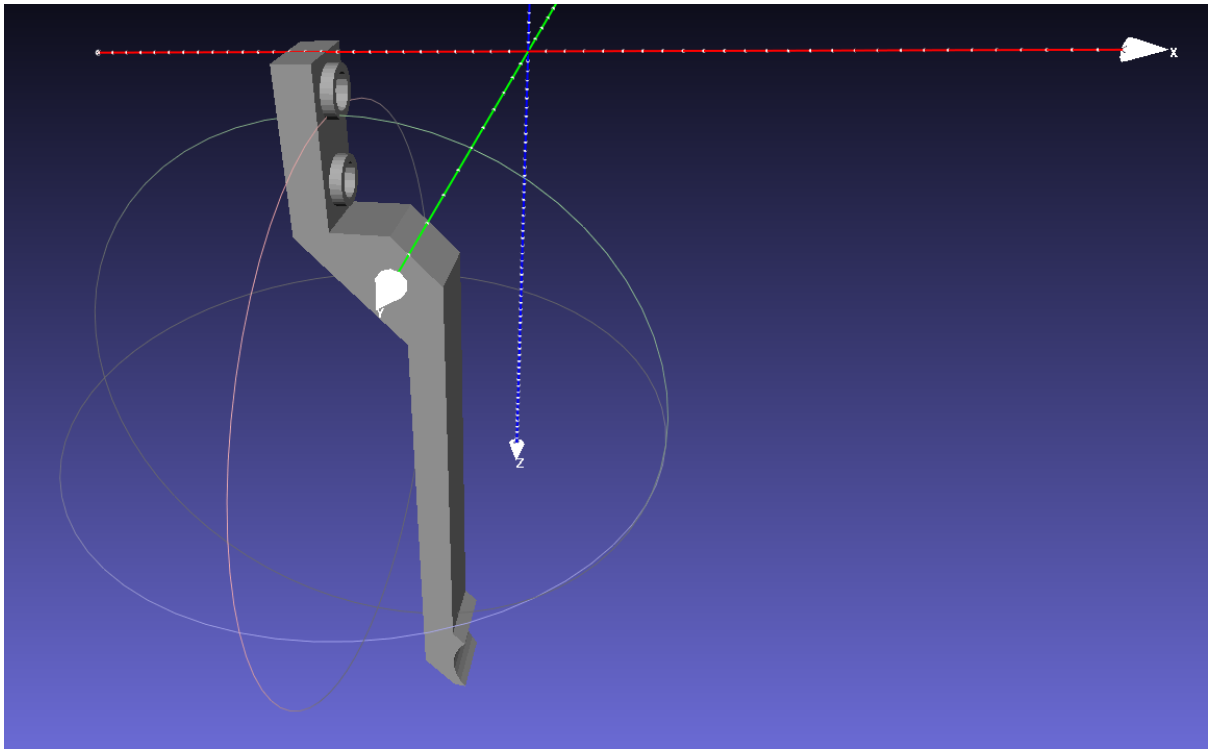


Fig. 20: Finger geometry of the example gripper (shown in gray above): Coordinate origin is on the z-axis of the robot tool flange. This facilitates positioning.

Integrate CAD

Once all the required CAD files are available, you can integrate them into your cell using the Mikado Assistant.

As soon as you have created your cell via the Mikado Assistant, a gripper template is already included. To use your own gripper model, you can now adjust the file paths that are defined in the *URDF file* of the gripper model:

```
<mesh filename="package://mein_projekt/meshes/greifer_dateiname.stl" />
```

Hint: In addition to STL files, you can also integrate gripper models in DAE format.

Greifermodell positionieren

You will find that the gripper is often incorrectly scaled or not positioned in the desired location.

- **Adjust scaling**

  Set the 'scale' property if the STL file does not use the meter unit. In this example, the unit mm was used in the STL, hence the factor 0.001:

  ```
  <mesh filename="package://mein_projekt/meshes/greifer_in_mm.stl" scale="0.001" />
  ```

- **Adjust gripper position**

  Just as with cell geometry, individual bodies/parts are connected and positioned using joints. The position of the gripper and the gripper parts attached to it can therefore be changed in the same way. Further details are described in detail in section *URDF*.

  Do not forget to also adjust the joint `<joint name="gripper_to_tcp" type="fixed">`. This joint defines the position of the gripping point on the tool and is visualized in Mikado as a transparent sphere.

7.7.3 Mikado configuration

To ensure that your gripper is controlled correctly, define the properties of a gripper in the configuration file, which is explained in the *gripper.yaml* section.

7.7.4 Gripper connection variants

Most of the gripper variants can be integrated via robot IOs or a fieldbus. Which IO assignment you use for the open and close actions is also entered in the Mikado configuration (see *gripper.yaml*).

If you are using a gripper that is controlled via TCP/IP, you must assign an IP to the gripper via the manufacturer's software that is located in the Mikado controller network. Now enter the IP in the *main.launch* file in the *launch/* directory. Within the file you will find the following line in which you must enter the new IP address.

```
<include file="...">
    <arg name="ip" value="10.0.1.13" />
```

7.7.5 Calibrate TCP

During TCP calibration, the transformation of the gripper assembly to the robot flange is determined.

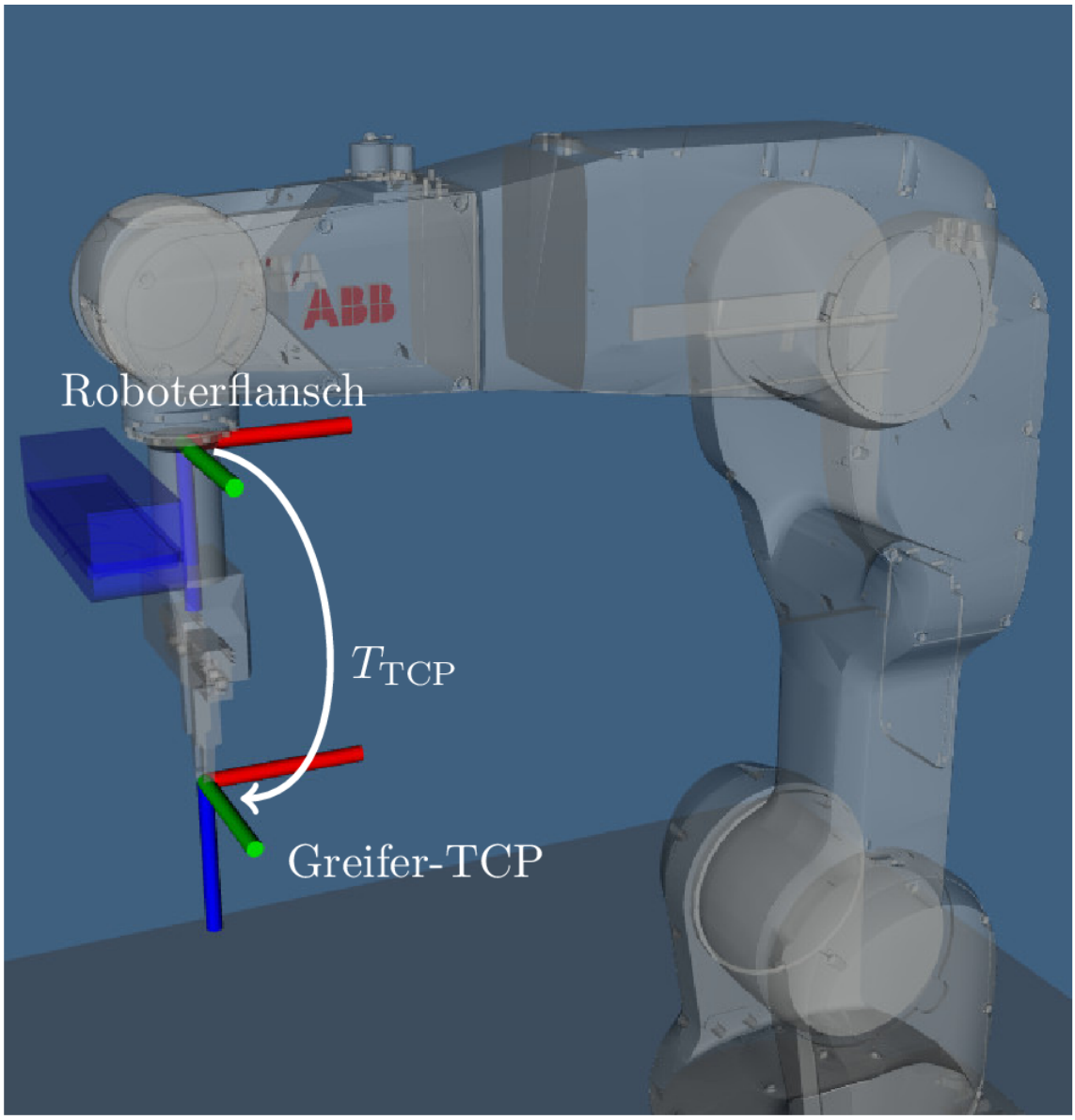


Fig. 21: Transformation determined by the TCP calibration.

A check of the TCP offsets after a gripper has been set up or reassembled is strongly recommended. Any deviation of the assembly from the values entered in the gripper position must be recognized when gripping or approaching parts.

A simple Mikado sequence helps you to visually check whether the gripper position entered is correct.

- Start the Mikado Control interface after you have set up your gripper
- Place the calibration plate in front of the robot in the camera's field of view in the recording pose
- Download the following Mikado program from *programs/examples/tcp_calibration_example.program*, or create it manually
  - Move robot to a suitable recording pose
  - Determine pattern pose
  - Reverse pattern pose (roll 180°)
  - Move pattern pose in z-direction (e.g. by +10 mm)

  - Move TCP of the robot to the determined pose
- The robot should now be positioned with the TCP of the gripper centrally above the middle point of the calibration plate and at the specified distance from the surface of the plate (e.g. 10 mm)
- Measure the deviations from the expected central position and apply the deviations to the offset of *gripper_tcp* in the Mikado Assistant gripper editor
- The offset from *gripper_tcp_uncalibrated* to *gripper_tcp* indicates the offset between the TCP position expected in CAD and the actual measured TCP position
- After saving the changes in the Assistant, the Mikado GUI must be restarted to read in the new values
- Repeat the process to ensure that the new offset values have led to an improvement in positioning accuracy

7.8 Set up hardware

This section contains instructions for connecting specific hardware to Mikado Control.

7.8.1 Overview of connections/cabling

The Mikado controller must be connected to the camera, but there are different things to consider here depending on the camera model. The compact N-series cameras can alternatively be connected via PoE (Power over Ethernet) or via a separate power supply (24V). Data is transmitted via Gigabit Ethernet. This requires a sufficiently shielded cable, at least Cat 5e.

The wiring of a cell containing an X36 camera is shown below. In particular, the minimum voltage of > 20 V at a current of 5 A at the camera must be observed. When calculating the voltage drop, the cable length in particular must be taken into account. There is a return path, as there is a positive and negative pole. The voltage drop also depends on the current, the cross-section of the cable and the specific electrical resistance of copper. The network cable or the fiber optic cable from the Mikado controller to the camera or from the Mikado controller to the robot are not included in the scope of delivery, as the length depends on the system design.

$$\Delta V = l_{Kabel} * I * \rho_{Kupfer} * \frac{1}{A_{Kabel}}$$

$$\Delta V = (l_{Hin} + l_{Rück}) * 5A * \frac{0{,}017\Omega * mm^2}{m} * \frac{1}{0{,}5mm^2}$$

$$\Delta V = (10m + 10m) * 5A * \frac{0{,}017\Omega * mm^2}{m} * \frac{1}{0{,}5mm^2}$$

$$\Delta V = 3{,}4\ V$$

Fig. 22: Example for calculating the voltage drop as a function of the cross-section and cable length

As cables with a cross-section > 0.5 mm^2 can only be accommodated in an M8 connector with great effort, it is advisable to choose an M12 cable with an M12 to M8 adapter connected to the end.

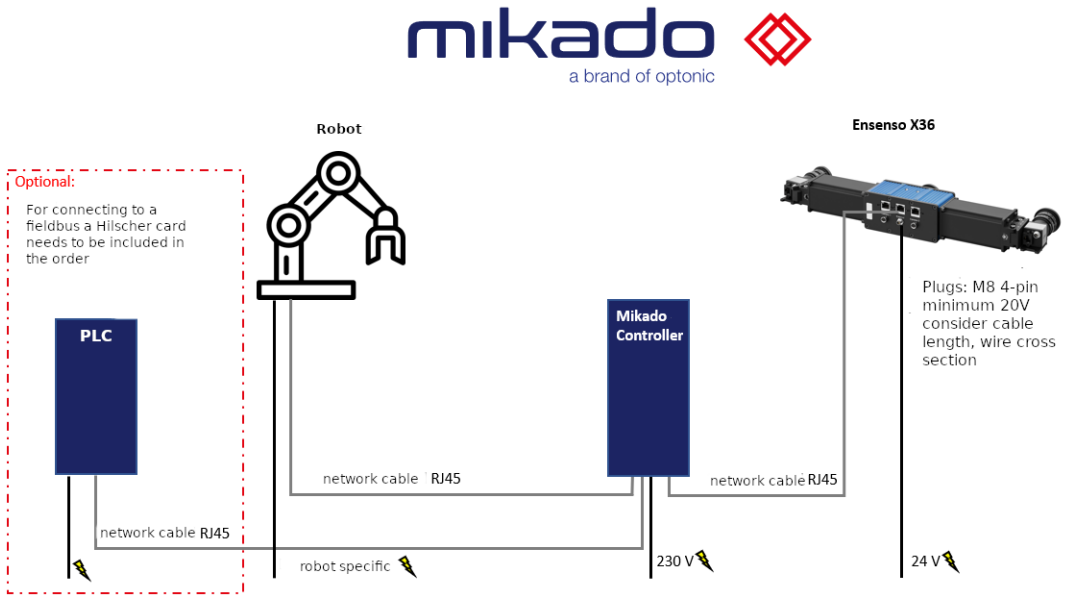


Fig. 23: Connection overview when using an Ensenso X36

While a network switch is integrated in an Ensenso X36, the XQ model has an external switch that is supplied wired directly to the XQ camera. This does not require an M8 plug, but three cables. There are connectors on the switch into which the cables and their ferrules are plugged. The images are transmitted from the switch to the Mikado controller via a fiber optic cable to ensure high transmission speeds. The 10 GBit network card required for this and the adapter for the Mikado controller are included in the scope of delivery.

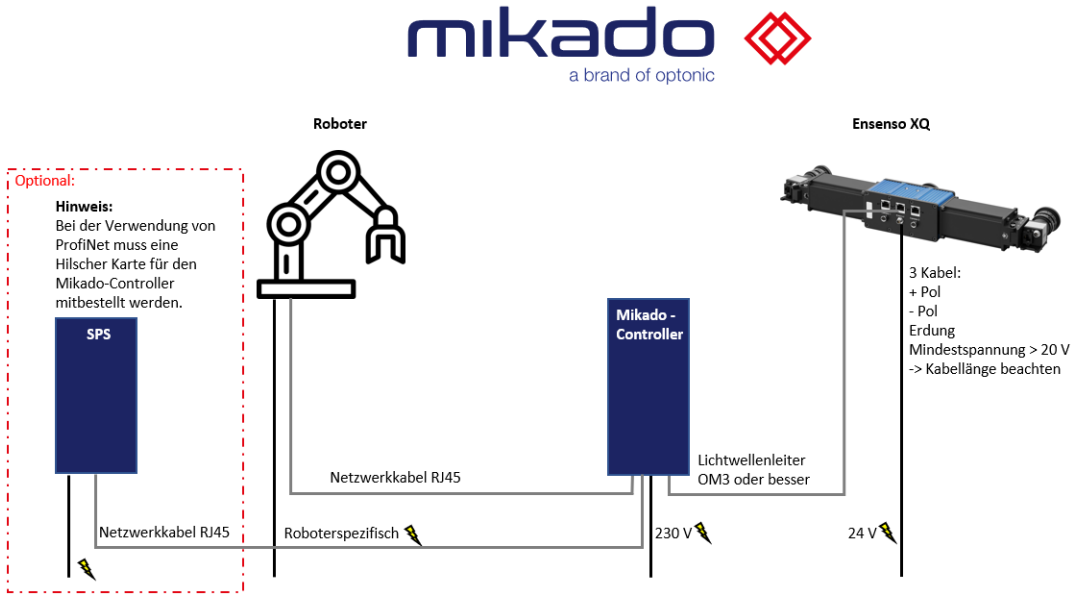


Fig. 24: Connection overview when using an Ensenso XQ (photo shows an X36, not an XQ)

If you want Mikado to communicate with your PLC via a fieldbus such as ProfiNet, please contact us when ordering so that a corresponding interface card can be included in the planning. If you want to communicate via OPC UA, for example, no additional hardware is usually required.

7.8.2 Display connection, operation without connected display

To operate, the Mikado Controller requires either a connected monitor, an HDMI or DisplayPort dongle, or a configuration for a virtual monitor.

A dummy connector and the virtual monitor configuration also have a preferred maximum screen resolution.

To set up the virtual monitor, run the script `sudo install-headless-display` in the Konsole.

The script sets up a virtual HDMI monitor with a resolution of 1920x1080 pixels. This resolution is also used when establishing a remote maintenance connection to the Mikado controller.

Once the script has been enabled, the computer will immediately check whether a real display (or a dongle) is connected every time it restarts. If so, it will use that display instead of the virtual monitor.

If no monitor or dongle is connected, the virtual HDMI monitor is used.

A **significant side effect** of the virtual monitor is that automatic monitor detection is disabled when a real monitor is connected. For example, if a DisplayPort monitor is connected, it will not be recognized until the next controller restart. If an HDMI monitor is connected before the restart, the display will only use the fixed resolution of the virtual monitor (default: 1920×1080), and the connected monitor may scale or fit the image accordingly.

To disable the automatic setup of the virtual monitor at system startup again, run the following command in the console: `sudo install-headless-display --disable`.

7.8.3 Network adapter

The Mikado controller must be connected to components such as the Ensenso camera and robots via a TCP/IP interface. How to set this up correctly is explained below.

Fixed IP

In this configuration, each component is connected to a dedicated network port on the Mikado controller. A fixed IP is then assigned to the controller and the components. On the Mikadocontroller side, this works by first switching to the network settings, for example by clicking on the network icon at the bottom right of the taskbar and then clicking on the settings icon in the dialog:

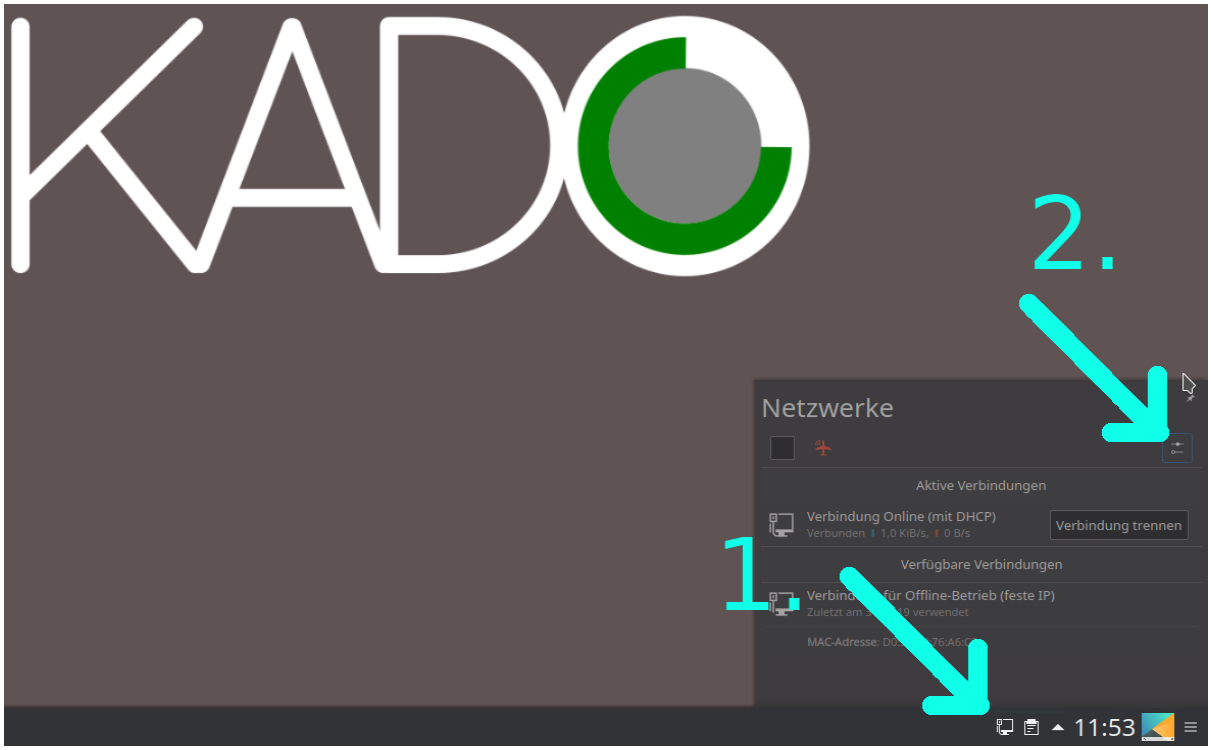


Fig. 25: Open screenshot of network settings

To set a fixed IP, follow the five steps:

1. Selecting the appropriate adapter
2. Selection of the IP v4 tab
3. Selection method: Manual
4. Create a new entry in the list using + *Add* and enter an IP and subnet mask.
5. *Apply* and *OK*

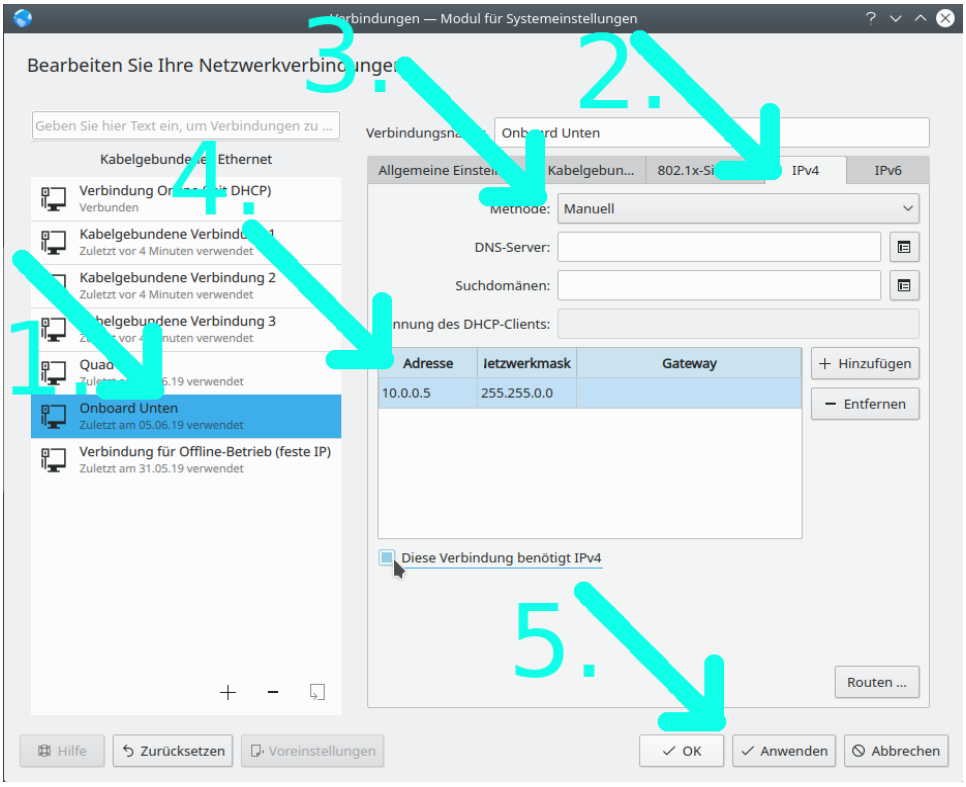


Fig. 26: Screenshot Configure fixed IP network settings

Warning: The different network adapters of the Mikado controller should not be in the same address space, as this makes packet transmission ambiguous. For example, configure one network port to 192.168.5.1 and another to 192.168.2.1.

7.8.4 Connection check

In case of connection problems, we recommend to first check whether the network device can be reached from the Mikado controller. To do this, open a console on the Mikado controller (CTRL + ALT + T) and enter the following command in the console (replace the IP with the IP of the connected network device):

```
ping 192.168.1.42
```

The ping can be canceled by pressing *Ctrl* + *C*. A positive response would look like this:

```
ping 192.168.1.42

PING 192.168.1.42 (192.168.1.42) 56(84) bytes of data.
64 bytes from 192.168.1.42: icmp_seq=1 ttl=64 time=1.18 ms
64 bytes from 192.168.1.42: icmp_seq=2 ttl=64 time=0.448 ms
^C
--- 192.168.1.42 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.448/0.816/1.185/0.369 ms
```

If the ping is successful within a few milliseconds, the connected network device is reachable under the given IP address.

If the ping results in error messages, there are multiple possible causes:

- The network device with that IP address is not ready to communicate.
- There is no network connection between the devices. Maybe a network connector is not fully seated within its corresponding RJ-45 socket.
- The IP addresses of the Mikado controller and the connected network devices are not within the same subnet address space.

7.8.5 Set up Ensenso cameras

Setting up a camera is basically divided into three steps.

1. Setting up the network interface and thus communication with the camera
2. Optimization of the camera configuration for the scenario, in particular exposure time, FlexView and parameters for stereo matching
3. Hand-eye calibration for the micado cell

Setting up the camera network adapter

The first step is to establish a connection to the camera. To do this, it must be assigned an IP in the subnet space of the Mikado network adapter. Once the network adapter has been set to the desired IP, this can be done using the Network Wizard in NxView.

If no cameras are displayed in NxView, please check the following points:

- LEDs on the Ensenso camera flash to indicate that the camera is energized
- Ensenso N-Series: Start `sudo idscameramanager` via the console. This is a stand-alone tool for configuring the camera driver. If the camera is still not found after a few seconds, the "ETH-ConfigurationMode" must be activated under "Daemon Control" for the network adapter used. The camera found in this way can be assigned a fixed IP in the same address space using "Manual ETH configuration".
- Use the latest version of the Ensenso Manual to look up further information on the correct configuration and connections of your system.

If several cameras are used in your application, you must connect both devices to separate network adapters on your Mikado controller. Both adapters must be in their own subnet.

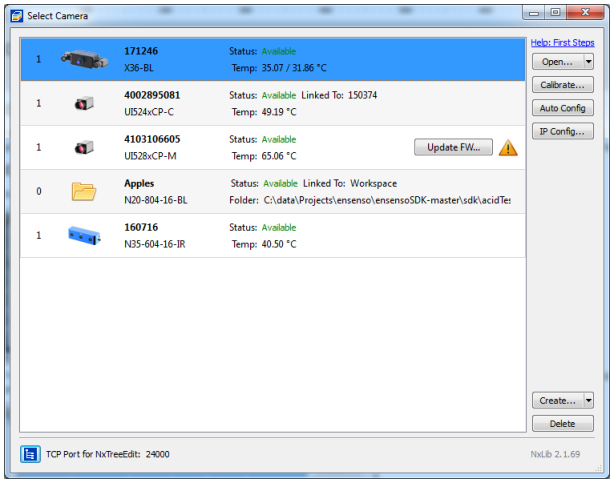
Hint: For example, if the fixed IP of a network adapter is 192.168.2.42 (with subnet mask 255.255.255.0), the same address space only means a permitted difference in the last block of numbers. A suitable IP for the camera would then be 192.168.2.3, for example.

Warning: No two network adapters should be in the same address space, as this makes packet transmission ambiguous. For example, configure one port to 192.168.5.1 and another to 192.168.2.1.

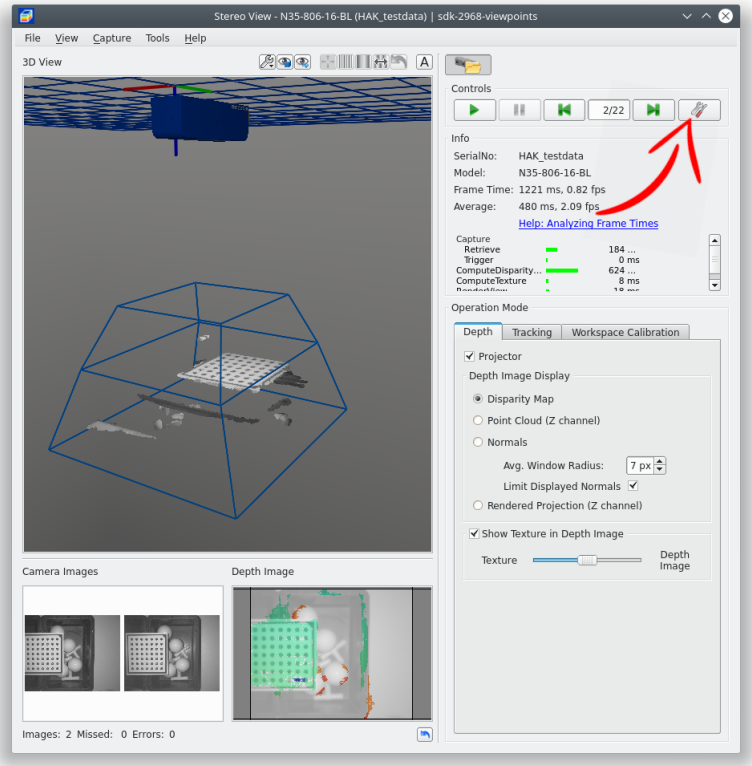
Camera configuration

The camera configuration refers to the internal parameter set of the 3D sensor used. The NxView software is used to configure the camera. The following steps are required to create a camera configuration:

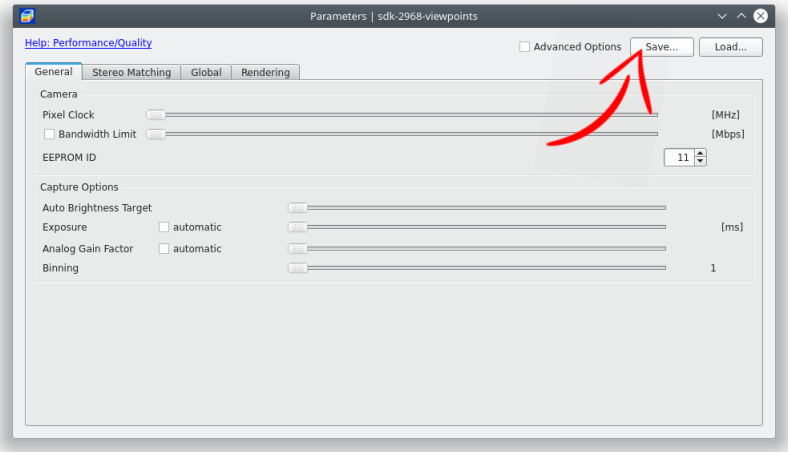
1. Open NxView and select the desired camera.



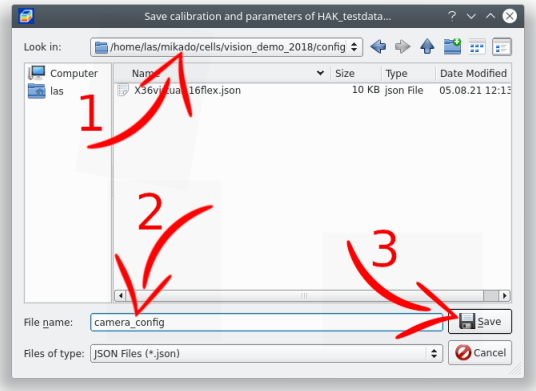
2. As soon as the camera is open, click on the parameter icon. A new window opens.



3. All camera parameters can be set in the parameter window. As soon as you have found a parameter set that produces the required data quality, save the parameter set. To do this, click on the Save button. A suitable name can be the serial number of the camera, e.g. `229020.ensparam`.



4. A dialog box opens in which you select the file path of your Mikado cell (1), choose a descriptive name for the camera configuration file (2) and finally click on Save (3).



Now open the cell in the Mikado Assistant and select the newly created configuration in the menu item ‘Parameters → Cameras → Config file’.

Further detailed information can be found in the NxView Manual.

Note: With an Ensenso X series, an intrinsic calibration may still be necessary.

Hand-eye calibration

Starting the hand-eye calibration from the user's perspective is described in more detail in the chapter *Hand-eye calibration*. For this to work, the movement parameters must be defined during calibration. These are listed in chapter *calibration.yaml*.

File camera

File cameras offer good optimization options for newly created cells as well as error reporting options. It is possible to collect the necessary raw data both from Mikado and using NxView. It is especially recommended for setups with a moving camera (cameras on the robot arm) to collect the data from Mikado, either with the corresponding task in the program flow (see task '*Save image*') or on the *recognition page* using the Save last shot button. Mikado also saves a link file for each recording, which contains the current robot position for moving cameras. Uncalibrated file cameras display the point cloud in the visualization as if the camera were positioned in the coordinate origin looking upwards. Nevertheless, object recognition can also be carried out with an uncalibrated camera without any problems, for example as part of a preliminary inspection.

The following raw data is written for each recording:

Pictures	*.png	Number of FlexView x (left, right)
Kameraparameter	*_calib.json	1x
Roboterposition	Links*.txt	1x

Creating a file camera from recorded data can be done via NxView. In the interface, select 'Create' → 'File Camera' and the path of the folder containing the raw data. You can then adjust the serial number of the camera, which is set to 'File[serial number]' by default.

Now the serial number of the camera must be adjusted in Mikado, see chapter *camera_X.yaml*. In this configuration file, also pay attention to the parameter `fixed_camera`, as this also indicates whether the camera was movable or fixed for file cameras. Mikado then uses the recordings from the file camera in the order in which they were recorded. Once the last shot has been taken, the file camera starts again from the beginning. The camera parameters that are stored with the file camera are used, *not those loaded when Mikado is started or in the program sequence*.

Warning: File cameras that are in a *.zip can be created in NxView, but are not supported by Mikado. These must be unzipped and the file camera created with the unzipped files.

Multi-Camera Setup

If you would like to integrate an additional camera into your application, the following steps must be carried out for installation:

1. Connect the camera to the controller and set up communication as described in *Setting up the camera network adapter*.
2. Create a second camera configuration and also save it in the `config` directory.
3. Copy the file `camera_0.yaml` in the `config` directory, rename the copy (e.g. `camera_1.yaml`, `camera_2.yaml`, etc.) and open it. Now adjust the entries there (see section *camera_X.yaml*).
4. Open the file `main.launch` in the `launch` directory and insert the following section under the line `<!-- Cameras -->`. Make sure that you give the camera a new unique name and enter the correct configuration file.

```
<node name="Ensenso_Camera_1" pkg="ensenso" type="ensenso_node" output="screen"␣
↪ns="cameras">
  <rosparam command="load" file="$(arg project_dir)/config/camera_1.yaml" />
</node>
```

5. Start Mikado and check in *System status* whether the camera could be opened correctly.

The camera is now set up. You can now continue with the hand-eye calibration (see *Hand-eye calibration*). The Mikado interface will now display a selection dialog for the camera to be used at the appropriate points. The active camera can be adjusted in your Mikado process via the task '*Set active camera*'. The currently active camera can be seen in the *system bar* at the bottom.

7.8.6 Setting up the robot controller

The setup of the robot controller is described in chapter *Robot models*.

7.9 Fieldbus interface

Mikado can be connected to other devices via fieldbus, e.g. to a PLC.

7.9.1 Brief description of the interface

The core functions of Mikado can be called up via the fieldbus interface. A distinction is made between general functions and functions for controlling a loaded Mikado program.

General functions

- Carry out the hand-eye calibration.

Functions for controlling a Mikado program

- Start and stop the Mikado program or an individual sequence within the Mikado program.
- The synchronization of Mikado variables with the PLC.

7.9.2 General communication process

Communication is realized by means of commands (see *Commands*) that are sent from the PLC to Mikado. The status of the communication is described using status bits (see *Communication status*). The following process description applies to all commands. The condition for communication is `OutReadyBit = 1`, which depends on the Mikado **system status**.

Table 6: General communication process

Time	SPS	Mikado
$< t_1$	Wait for `OutReadyBit = 1` Wait for `OutBusyBit = 0` Set parameters	Waiting on command
t_1	Set `InCmd` Wait for `OutBusyBit = 1`	

t_2		`OutErrorCode = 0`
		`OutBusyBit = 1` (fieldbus adapter processes command)
		Only when starting the program or a sequence:
		`OutRunningBit = 1`

t_3	Reset `InCmd` to `0` Wait for `OutBusyBit = 0`	
t_4		Write `OutErrorCode` `OutBusyBit = 0` (fieldbus adapter has processed command)
t_5	Reading `OutErrorCode` ***Nur when starting the program or a sequence:*** Wait for `OutRunningBit = 0`	
t_6	***Nur when starting the program or a sequence:*** Reading `OutErrorCode`	***Only when the program or a sequence was started before:*** `OutRunningBit = 0` (MIK program ends)
from here	read variables if necessary	

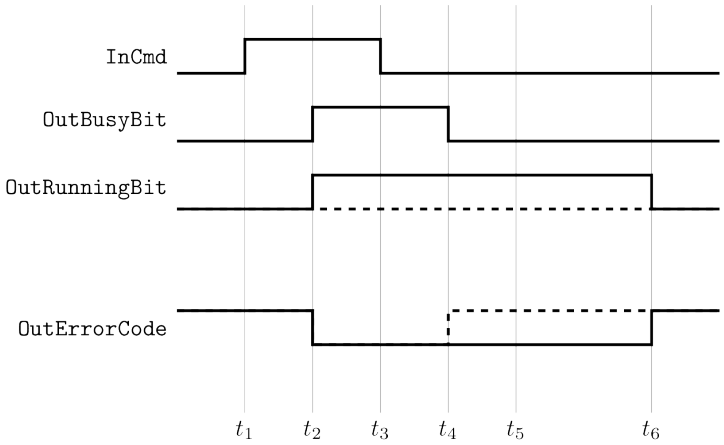


Fig. 27: General signal trace, alternative cases dashed

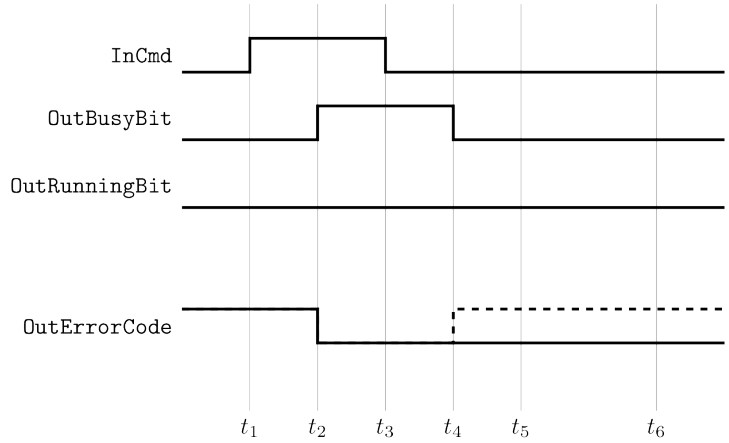


Fig. 28: Signal trace for commands like StartCalib, CaptureCalib, FinishCalib

- If a command is set (`InCmd` $\neq$ `0`), Mikado responds with `OutBusyBit = 1` and first resets the `OutErrorCode`. The PLC responds to `OutBusyBit = 1` with `InCmd = 0` and then waits for `OutBusyBit = 0`.
- The PLC evaluates the `OutErrorCode`. If this has the value 0, the command has been executed successfully.
- Only commands `RunMikadoProg` and `StartSequence` cause `OutRunningBit = 1`. This signals that Mikado is processing the current program. As soon as `OutRunningBit = 0`, the `OutErrorCode` and the corresponding output variables can be read out.

7.9.3 Communication status

LifeBit

A LifeBit is used to check the status of communication between Mikado and the PLC. The PLC reads the value of `OutLifeBit` and writes it unchanged to `InLifeBit`. The fieldbus adapter reads the value of `InLifeBit` and writes the negated value to `OutLifeBit`. If the value of `OutLifeBit` does not change regularly, communication is considered interrupted and a corresponding error is displayed in the Mikado status.

Mikado status

The status of Mikado is transmitted to the PLC using three different bits. The following table summarizes the meaning of the individual bits.

Table 7: Meaning of the individual status bits

Name	Explanation	Value
`OutBusyBit`	Status of the fieldbus adapter.	True = Command is processed. False = Adapter is ready to accept new command
`OutReadyBit`	System status of Mikado.	True = Mikado ready for operation False = Mikado not ready
`OutRunningBit`	Program status of Mikado	True = Mikado program is running (corresponds to green or yellow in Mikado) False = program is not running (corresponds in Mikado: orange, red or grey)

Note: New commands are only accepted if `OutBusyBit` = False

7.9.4 Synchronizing PLC variables and Mikado variables

PLC variables can be mapped to Mikado variables and vice versa. The mapping between PLC variables and Mikado variables can be adapted and extended in the Mikado Assistant under “Interfaces”. Corresponding setup pages are available for the various supported communication protocols (ADS, EtherCAT, EtherNet/IP, Profinet). The standard fieldbus configuration of newly created cells contains all the necessary entries for executing the commands.

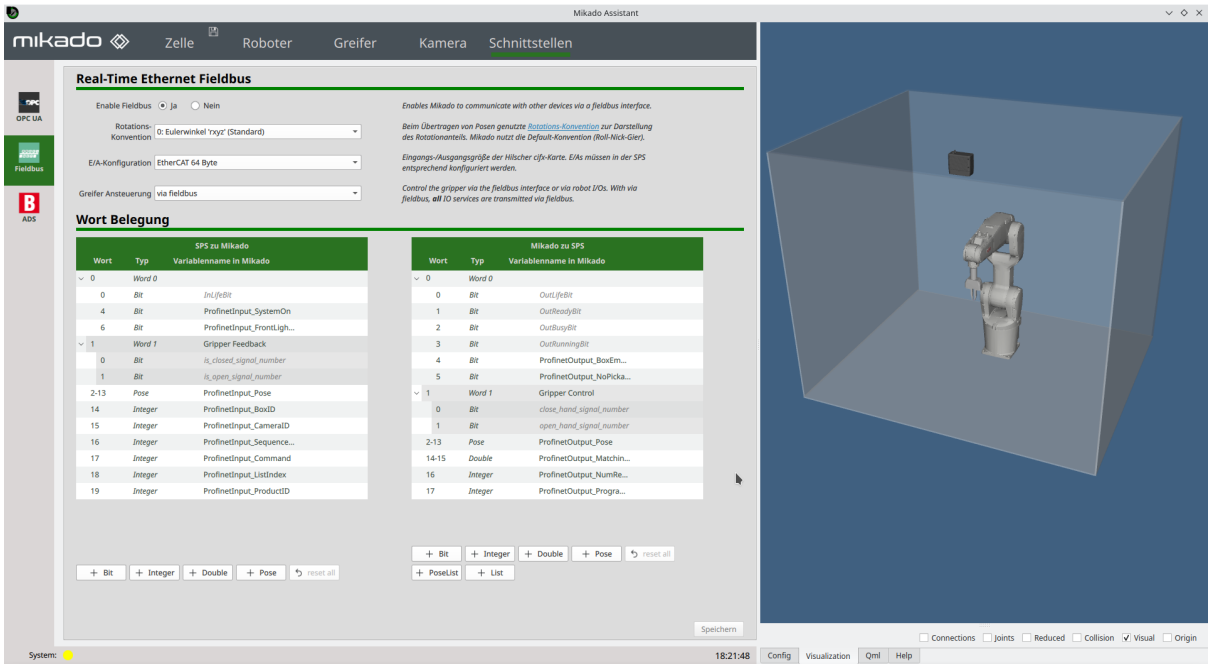


Fig. 29: Assistant for adapting the word assignment and synchronized Mikado variables via fieldbus

The fieldbus configuration is saved in the configuration file `config/profinet.yaml` or `config/ads.yaml`. The file is structured as follows:

```
inputs:  # list of synchronized variables PLC -> Mikado
    - name: <VariableID>  # internal unique ID
      type: <VariableType>  # type of variable, e.g. Bool, Double, Pose, ...
      # Mikado address:
      mikadoVarName: <VariableNameMikadoSide> # variable name in Mikado as stated on␣
↪Edit>Variables page
      # Fieldbus address:
      word: <AddressProfinetSide>  # only applicable for Profinet / EtherCAT /␣
↪EtherNet/IP variables
      bit: <AddressProfinetSide>   # only applicable for Profinet / EtherCAT /␣
↪EtherNet/IP variables of type 'Bool'
      listSize: <size of list>     # only applicable for Profinet / EtherCAT /␣
↪EtherNet/IP variables of type 'PoseList or StringList'
      remoteName: <VariableNameAdsSide>  # only applicable for ads variables
    - ...
outputs:  # list of synchronized variables Mikado -> PLC
    - ...
```

All names of synchronized variables mentioned in this documentation refer to `<VariableID>`. If, for example, the variable `InSequenceID` is mentioned, it is assumed that the configuration file used contains an entry `name: InSequenceID`.

7.9.5 Rotation conventions

The rotation convention defines how the orientation of poses is transferred to the interface or read from the interface. The orientation is always specified in the form of Euler angles, which can be interpreted in different ways depending on the convention (see below).

The rotation convention is specified as a combination of four letters: **r** (**r**elative, intrinsic, rotated coordinate system) or **s** (**s**tatic ,extrinsic, fixed coordinate system) followed by the order of the three rotation axes **x/y/z**. This theoretically results in 24 different combinations in which the orientation of poses can be represented in Euler angles.

The following table lists the rotation conventions for some robot manufacturers. The convention must be set to match the robot used in the Mikado Assistant under the "Interfaces" tab so that the alignment of poses can be interpreted correctly.

Mikado uses the convention *rxyz* (roll-nick-yaw / RPY) as standard. If this convention is selected, the Mikado poses are passed on to the interface unchanged. If a different setting is selected, the rotation part of a pose is converted by the interface into the corresponding format.

The rotation convention *uvw*, which is used natively by Universal Robots, has a special position. Instead of converting this to Euler angles on the robot side, it can be used directly by selecting the corresponding convention. The parameters **u**, **v** and **w** represent a vector around which the pose is rotated. The type of rotation description corresponds to the *Axis-Angle Representation*.

Table 8: Supported conventions

ID	Convention	Robot manufacturer	R1	R2	R3
0	*rxyz*	(Standard) Stäubli	*X* (Roll)	*Y'* (Nick/Pitch)	*Z"* (Gier/Yaw)
1	*sxyz*	Fanuc Yaskawa Motoman Universal Robots Fruitcore Mitsubishi Techman	*X* (A)	*Y* (B)	*Z* (C)
2	*rzyx*	Kuka ABB Nachi	*Z* (A)	*Y'* (B)	*X"* (C)
3	*rzyz*	Doosan Kawasaki Omron	*Z*	*Y'*	*Z"*
4	*uvw*	Universal Robots	*U*	*V*	*W*

7.9.6 Error codes

The output word `OutErrorCode` encodes error messages. The following table summarises all possible error codes and explains their meaning and recommended corrective actions.

Table 9: Possible error messages

Code	Error name	Description	Recommended action
0	`NoError`	Initial state. No error present	-
>0	Error in the Mikado program	Project-specific error in the Mikado program	For details, see the Mikado program or project-specific documentation
-1	`ServiceCallFailed`	Command failed due to an internal error.	Contact Mikado Support
-4	`NoReplyValue`	No return value received (internal error)	Contact Mikado Support
-5	`TooManyReplyValues`	Number of return values higher than expected (internal error)	Contact Mikado Support
-6	`FatalErrorMikadoProgram`	The Mikado program has a fatal error. The program cannot continue from this point.	Reset the Mikado program and restart the program / sequence
-8	`UnknownCommand`	Unknown command received	Check `InCmd`. For valid commands, see *Commands*.
-9	`LifebitClientMissing`	Lifebit of the client was not recognized.	Check the fieldbus connection between Mikado and the communication partner and the configuration of the 'LifeBit'
-11	`TriggeringFailed`	Failed to call a Mikado sequence. This can occur, for example, if Mikado is not ready for operation or if a program is already running.	Ensure that Mikado is ready for operation (`ReadyBit = true`) and that no program is already running (`RunningBit = false`)
-12	`ReadingVariableFailed`	Internal error when reading data from the fieldbus interface	Contact Mikado Support
-13	`WritingVariableFailed`	Internal error when reading data from the fieldbus interface	Contact Mikado Support

Table 10: Possible error messages, continued

Code	Error name	Description	Recommended action
-15	`CancelCalibrationFailed`	Aborting an already started calibration failed	Ensure that calibration has been started previously (*StartCalib*) and that no program is running (`RunningBit = false`)
-16	`CaptureForCalibrationFailed`	Command *CaptureCalib* failed. Capture for ongoing calibration unsuccessful	Ensure that calibration has been started previously (*StartCalib*) and that no program is running (`RunningBit = false`)
-17	`FinishCalibrationFailed`	Command *FinishCalib* failed. Calculating and saving calibration was not successful	Ensure that calibration has been started previously (*StartCalib*) and that no program is running (`RunningBit = false`)
-18	`StartCalibrationFailed`	Command *StartCalib* failed. Calibration could not be started	Ensure that no calibration is already in progress and no program is running (`RunningBit = false`)
-19	`SequenceDoesNotExist`	Command *StartSequence* failed. Mikado sequence with the specified ID does not exist	Check `SequenceID` and call command *StartSequence* with a valid ID
-20	`MissingConfigurationItem`	Invalid fieldbus configuration. A required entry is missing.	Contact Mikado Support
-60	`RetryErrorMikadoProgram`	The Mikado program has an error, but the execution can be retried. This can occur if the connection to the robot or camera is briefly interrupted while Mikado is attempting to execute an action.	Execute the command *RunMikadoProg*. In many cases, the cause of the error will have been automatically corrected by the time the retry is attempted, and the process can be continued successfully. If not, another error will be triggered.
-61	`RetryActionErrorMikadoProgram`	Robot movement failed. A movement command from Mikado could not be successfully executed by the robot. This can occur if the connection to the robot is lost or if the robot is stopped unexpectedly, for example due to an error or a safety stop (emergency stop).	If applicable, first rectify the cause of the unexpected stop and then execute the command *RunMikadoProg*. The program will continue. If the movement still cannot be executed successfully, an error will be triggered again.
-100	`UnexpectedError`	Unexpected error occurred	Contact Mikado Support

7.9.7 Commands

All commands have at least `OutErrorCode` as a return value. Some commands have additional input and return values. The additional input and return values are listed in the detailed description of the commands, if they exist.

Note: The input values must be present on the fieldbus **before** the command is set so that they can be taken into account. The return values are present on the fieldbus as soon as `OutBusyBit == 0` and the configuration file contains the corresponding entries (see *Synchronizing PLC variables and Mikado variables*).

The following table lists all available commands:

Table 11: Available commands

InCmd	Command name	Description
0	`Idle`	Waiting
Program specific commands		
1	*RunMikadoProg*	Start/continue Mikado program from the current task
2	*StartSequence*	Start a specific sequence of the Mikado program. SequenceID must be set beforehand
3	*PauseMikadoProg*	Stop process after execution of the current task
4	*StopMikadoProg*	Stop the Mikado program
5	*ResetMikadoProg*	Stop Mikado program and reset all variables and states in the Mikado
General commands		
8	*StartCalib*	Start hand-eye calibration
9	*CaptureCalib*	Triggers image acquisition and measurement of the calibration pattern
10	*FinishCalib*	Calculate and save hand-eye calibration
11	*readPoseAndPublishTool0*	Sets the pose of the robot flange in Mikado
14	*CancelCalib*	Cancel hand-eye calibration

Program specific commands

Program-specific commands refer to the current Mikado program and can only be executed if a program is loaded. These commands have no additional return values. The desired results depend on the implementation in the Mikado program and are communicated individually via the synchronized variables (see *Synchronizing PLC variables and Mikado variables*).

Conversely, the behaviour of the Mikado program can be influenced via the fieldbus by using synchronized variables as inputs for tasks in the Mikado program. It should be noted that a synchronized variable used as task input in the Mikado program sequence is read out at the time of task execution. Therefore, synchronized variables on the PLC side should only be changed if no Mikado program is running.

Note: To avoid errors in the program sequence, synchronized variables on the PLC side should only be changed if no Mikado program is running (`OutRunningBit = 0`).

RunMikadoProg

This command is used to start or continue the current Mikado program. After successful execution of the command, the `RunningBit = True` changes and signals the running Mikado program. As soon as this is reset by Mikado, the synchronized variables can be evaluated.

StartSequence

This command executes the Mikado sequence with the ID stored in the word `SequenceID`. From program version 3, the ID of a sequence can be set in the text field to the right of the sequence name. In earlier program versions, the sequence name must be the same as the ID sent, for example the integer "42".

After successful execution of the command, the `RunningBit = 1` changes and signals the running Mikado program. As soon as this is reset by Mikado, the synchronized variables can be evaluated.

Table 12: Additional input and return values

Input value		Return value
`InSequenceID`	ID of the sequence to be started	*none*

PauseMikadoProg

This command pauses the execution of the current program.

StopMikadoProg

This command stops the execution of the current program.

ResetMikadoProg

This command stops the execution of the current program and resets the status of the program and all variables.

General Commands

General commands can be executed independently of the Mikado program. The return values are at the address defined in the configuration file as soon as `OutBusyBit == 0`.

StartCalib

This command starts the hand-eye calibration.

Table 13: Additional input and return values

Input value		Return value
`InCameraID`	ID of the camera to be used	*none*
`InFrontLightOn`	Switch on the light during calibration?	

CaptureCalib

This command is used to collect poses for hand-eye calibration. The calibration process must have been started beforehand using the `StartCalib` command. Between 10 and 20 poses should be collected. This command must therefore be executed frequently.

Table 14: Additional input and return values

Input value		Return value
`InPose`	Current pose of tool0	*none*

Example procedure:

- Robot strikes a pose
- PLC writes the current flange pose (tool0) to `InPose`
- The command `CaptureCalib` is triggered
- Mikado reads the current robot pose from `InPose` and takes a picture

FinishCalib

This command is used to calculate the hand-eye calibration. This is the last step in the hand-eye calibration process. This takes place after a sufficient number of collected pattern poses with the help of `CaptureCalib`.

Table 15: Additional input and return values

Input value	Return value	
none	`OutCalibMaxDist`	Maximum pose error distance
	`OutCalibMeanDist`	Mean pose error distance
	`OutCalibMeanX`	Mean pose error x
	`OutCalibMeanY`	Mean pose error y
	`OutCalibMeanZ`	Mean pose error z
	`OutCalibMeanAngle`	Mean pose error angle

CancelCalib

This command cancels the active hand-eye calibration.

readPoseAndPublishTool0

Note: This command is generally only used for select cells.

This command reads in a pose transmitted by the PLC and sets it in Mikado as the current pose of the robot flange (tool0). The position of the robot during image acquisition must be known for the Mikado program sequence, whereby Mikado Select itself has no information about the robot status. It is not necessary to use this command for hand-eye calibration, as the `CaptureCalib` command already performs the same functions in the background.

Table 16: Additional input and return values

Input value		Return value
`InPose`	Current pose of tool0	*none*

7.9.8 Example sequence for select cells

The example program `select_findObjects.program` is used below to show how a Mikado program can be controlled via the fieldbus. The sample program can be found in the program library from Mikado version 2.1.

Prerequisites:

- A product list must be loaded that contains a product with the name `P_0`.
- A scene must be loaded that contains an ROI with the name `ROI_0`.

Table 17: Example procedure for a select cell

#	SPS	Mikado
Only for mobile camera: Set robot pose (for static camera continue directly with #3)		
1	Set `InPose` to the current pose of the robot flange (tool0) Set `InCmd = 11`	
2		Adopts the values in `InPose` as the current robot pose
Find objects		
3	Set `InSequenceID = 1` Set `InBoxID= 0` Set `InProductID = 0` Set `InCmd = 2` (`RunSequence`)	
4		Start sequence with ID = 1 (*Find objects*) Set `OutNumResults` End of the sequence: `OutRunningBit = 0`
Communicate results		
5	Lese `OutNumResults` Set `InListIndex = 0` Set `InSequenceID = 2` Set `InCmd = 2` (`RunSequence`)	
6		Start sequence with ID = 2 (*get Result*) Set `OutPose` w.r.t. `InListIndex` End of the sequence: `OutRunningBit = 0`
7	Lese `OutPose` Set `InListIndex = 1` Set `InSequenceID = 2` Set `InCmd = 2` (`RunSequence`)	
8		*repeat step 6*
9	*repeat step 5 with* `InListIndex++`	
10	Lese `OutPose` Set `InListIndex = OutNumResults` Set `InSequenceID = 2` Set `InCmd = 2` (`RunSequence`)	
11		*repeat step 6*
12	Lese `OutPose`	

7.9.9 Updates of the fieldbus interface

Version 3.1.0

(mikado-profinet v3.1.0, mikado_io_beckhoff v3.1.0)

Note: If PLC-side value changes are made at program runtime (`RunningBit = 1`), unexpected behavior may occur with this version, as they are immediately visible to the program.

The new fieldbus adapter version brings the significant innovation that synchronized variables are constantly updated. In previous versions, variables had to be updated with the `syncAll` command. This change has the following consequences:

- The user-defined live property of variables no longer applies because all variables are now "live".
- The `syncAll` command is superfluous and is therefore no longer listed in this documentation. The command can still be sent. The `OutErrorCode` returned in this case is `NoError`.
- If PLC-side value changes are made at program runtime, unexpected behavior may occur.

The following table shows an example of how PLC-side changes can lead to unexpected behavior at an inopportune time. Assume a program is loaded in Mikado that calls the task `Find Sceneobject`, among others. The synchronized variable `InBoxID` is used as task input.

Table 18: Example of PLC-side changes at an unfavorable time

Time	SPS	Mikado	Comment
t_1	Set `InBoxID = 42` Set `InCmd = 1` (start program)		PLC sets the variable `InBoxID = 42` and sends start command

t_2		`OutBusyBit = 1`
		`OutRunningBit = 1`
		`OutBusyBit = 0`
			Adapter executes command
			Mikado starts program
			Command execution completed

t_3	`InBoxID = 0` (**false**) `InCmd = 0`		**This is an unfavorable time.** Variables used in the program sequence should not be changed at runtime.
t_4		Program calls `FindSceneObject (InBoxID)`	Mikado has now arrived at the task call. The current `InBoxID = 0` is used.
t_5		`OutRunningBit = 0`	End of program reached
t_6	`InBoxID = 0` (**correct**)		**The right time to reset variables.**

Version 3.1.2

Since *mikado-profinet* version 3.1.2 the package also supports the protocols EtherCAT and EtherNet/IP in addition to Profinet.

The protocol can be configured in Mikado Assistant on the fieldbus interface page.

7.9.10 Standard Fieldbus Interface

The different fieldbus interfaces have many commonalities. Following you will find the possible data types and the *Standard Word Mapping*.

This interface is fully supported for the protocols Profinet, EtherCAT and EtherNet/IP. Other protocols deviate from the standard as described in the respective section, like e.g Beckhoff ADS which maps variables to names instead of memory addresses.

An overview of the fieldbus interface documentation can be found *here*.

Byte Order

The byte order corresponds to the Siemens big-endian convention. When interpreting the words, a byte swap must therefore take place depending on the configuration of the master system.

Fieldbus Data Types

In principle, data is transmitted word by word, i.e. in 2-byte packets. As mentioned above, the big-endian byte order must be observed when interpreting the data. The following data types are possible:

- The word is broken down into 16 individual bits, whereby each bit is interpreted as a bool.
- The word is interpreted as an integer.
- `Double`: Two words (32 bits) are interpreted together as a floating point number.
- `Pose`: 12 consecutive words which are interpreted as six double values (X, Y, Z, R1, R2, R3).
- `PoseList` (Mikado to PLC only): `12 x <list length>` consecutive words per Pose, which are interpreted as a list of Poses. A list of length 5 will e.g. require 60 consecutive words.
- `StringList` (Mikado to PLC only): `2 x <list length>` consecutive words, which are interpreted as a list of double values. Strings are automatically converted to Double. A String that is not convertible to Double can not be transmitted using this interface.

Standard Word Assignment

Most of the assigned fieldbus words are directly mapped to Mikado variables.

Excluded from this are the words marked in grey in the Figure *Overview of the standard mapping of input and output byte arrays*, whose values are set independently of the Mikado program by the fieldbus adapter.

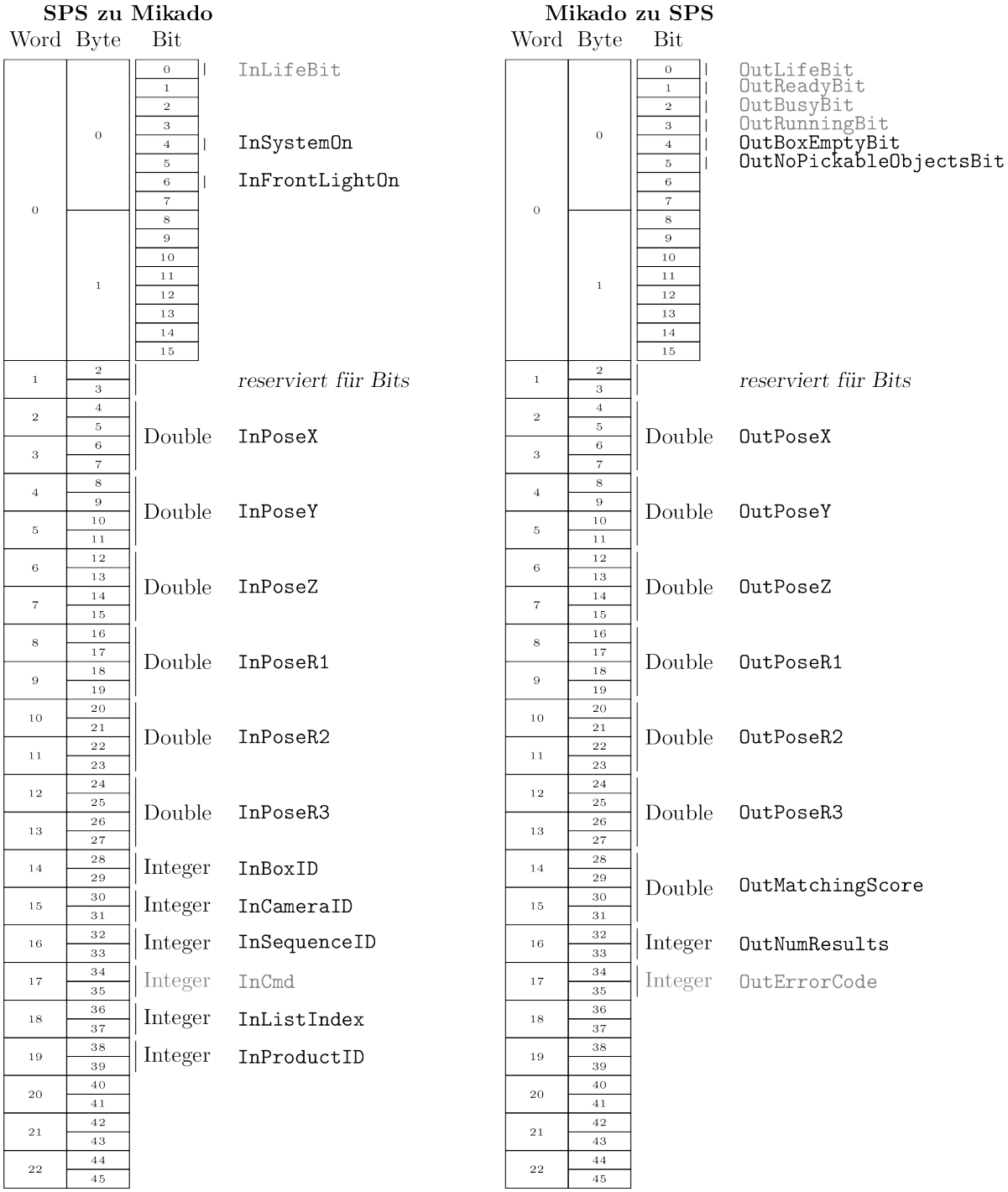


Fig. 30: Overview of the standard mapping of input and output byte arrays

7.9.11 Specific configuration for Profinet

This section describes the steps required to set up the hardware for communication via Profinet.

An overview of the fieldbus interface documentation can be found *here*.

GSDML

The interface description for ProfiNet connection is defined via the GSDML file. After installing the Debian package `mikado-profinet`, this file can be found under the following path:

`/opt/cifx/gsdml/GSDML-V2.35-HILSCHER-CIFX RE PNS-20190108.xml`

The following Profinet IO device of the GSDML must be integrated into the master:

`CIFX RE/PNS V3.5 - V3.5.x`

The Mikado controller provides the following input/output layout depending on the chosen configuration. Inputs and outputs are defined from the perspective of the Profinet master:

- 64 byte input and output
- 128 byte input and output
- 256 byte input and output
- 512 byte input and output

These Inputs and Outputs are written and read according to the word mapping, see *Standard Word Assignment*.

Standard Fieldbus Interface

Byte order, available data types and the standard word mapping follow the *Standard Fieldbus Interface*.

7.9.12 Spezifische Konfiguration für EtherCAT

This section describes the steps required to set up the hardware for communication via EtherCAT.

An overview of the fieldbus interface documentation can be found *here*.

Interface description

The interface description for a EtherCAT connection is defined via XML file. After installing the Debian package `mikado-profinet`, this file can be found under the following path:

`/opt/cifx/ethercat/Hilscher CIFX RE ECS V4.6.X.{BYTE_SIZE}.xml`

The following EtherCAT device is to be added on the master: `Hilscher CIFX RE ECS`

Depending on the selected configuration, the Mikado controller provides the following input/output layouts. Inputs and outputs are defined from the perspective of the EtherCAT master:

- 64 byte input and output
- 128 byte input and output
- 256 byte input and output

These inputs and outputs are written or read according to the configured word assignment, see *Standard Word Assignment*.

Standard Fieldbus Interface

Byte order, available data types and the standard word mapping follow the *Standard Fieldbus Interface*.

7.9.13 Spezifische Konfiguration für EtherNet/IP

Dieser Abschnitt beschreibt die notwendigen Schritte zur Einrichtung der Hardware für die Kommunikation via EtherNet/IP.

An overview of the fieldbus interface documentation can be found *here*.

Interface description

The interface description for a EtherNet/IP connection is defined in an EDS file. After installing the Debian package `mikado-profinet`, this file can be found under the following path:

`/opt/cifx/ethernet_ip/HILSCHER CIFX-RE EIS V1.1.EDS`

The following EtherNet/IP device is to be added on the master:

`Hilscher CIFX 50-RE EIS`

The Mikado controller provides the following input/output layout depending on the chosen configuration. Inputs and outputs are defined from the perspective of the EtherNet/IP master:

- 64 byte input and output
- 128 byte input and output
- 256 byte input and output

These Inputs and Outputs are written and read according to the word mapping, see *Standard Word Assignment*.

> **Warning:** The Mikado EtherNet/IP interface is configured by default to assign an IP address via `DHCP`. This setting cannot currently be changed by the user. If you require a static IP address for the interface for your application, please contact Mikado Support.

Standard Fieldbus Interface

Byte order, available data types and the standard word mapping follow the *Standard Fieldbus Interface*.

7.9.14 Specific configuration for Beckhoff ADS interface

This section describes the steps required to set up communication via Beckhoff ADS.

An overview of the fieldbus interface documentation can be found *here*.

Data types

Data type Mikado	Data type PLC	Remark
`Bool`	`BOOL`	
`Integer`	`INT`	
`Double`	`REAL`	
`Pose`	`ARRAY [1..6] OF REAL`	The six values represent X, Y, Z, R1, R2, R3 of the Mikado pose
`PoseList`	`ARRAY [1..6*n] OF REAL`	The length of the array must be a multiple of 6. Each 6 neighboring values represent a pose.
`String`	`STRING`	
`StringList`	`ARRAY [1..n] OF REAL`	**Mikado to PLC only**. The length of the array is not subject to any restrictions. The values within the Mikado-StringList are converted to doubles.

Note: If a PoseList or StringList in Mikado is longer than the available space on the PLC side, the Mikado-side list is truncated to the available length.

Configuration

In addition to the fieldbus adapter parameters, the following ADS-specific parameters can be set in the configuration file `ads.yaml`:

```
# for publishing joint state:
publish_robot_state: false # if true, the robot joint state is published every cycle
current_joint_position_array_name: MAIN.OutCurrentJointPositions # Name of main robot
↪'s joint positions on PLC
main_robot_prefix: "" # prefix as stated in the xacro-macro of main robot (works at␣
↪least for staeubli)

# publishing additional robots (optional):
additional_robot1_prefix: "example_prefix" # prefix as stated in xacro-macro of␣
↪additional robot
additional_robot1_variable: MAIN.OutJointPosOfRobotWithExamplePrefix # Name of␣
↪additional robot's joint positions on PLC

additional_robot2_prefix: "example_prefix2"
additional_robot2_variable: MAIN.OutJointPosOfRobotWithExamplePrefix2
# .. and so on

# network settings:
local_net_id: ""
remote_ip: ""
remote_net_id: ""
remote_port: 851

# IOs
input_array_name: GVL.inputBytes  # IO-Name on PLC
output_array_name: GVL.outputBytes # IO-Name on PLC
```

(continues on next page)

(continued from previous page)

```
# robot control:
enable_robot_control: false # if true, the robot is controlled via ADS-interface
publish_robot_status: false # if true, the robot status is published every cycle
trajectory_array_name: MAIN.InTrajectory
in_motion_name: MAIN.OutInMotion
writing_trajectory_allowed_name: MAIN.OutWritingTrajectoryAllowed
```

7.10 About the operating system

Mikado Select/Control comes pre-installed on a Mikado controller. A customised Kubuntu 22.04 is used as the operating system. Automatic login is enabled for the user *mikado*. The default password is *mikado* as well. This password is required for installing updates or for customization of the system at a deeper level.

Warning: We recommend replacing the initial password during start-up. Please take into consideration any corporate guidelines that might be applicable. The password can be changed via System Settings > Users > Change Password or via the command *passwd* in any terminal.

CHAPTER

EIGHT

EXTENDING MIKADO CONTROL

8.1 JavaScript Task

The *JavaScript Task* allows you to execute your own JavaScript/ECMAScript code as a task. In addition to all the functionalities of JavaScript, this task allows direct access to some Mikado variables via predefined modules. The following table provides an overview of the interfaces provided by the modules.

Table 1: **Overview of the interfaces in the JavaScript task**

task	Allows access to the visualization of the task and the setting of follow-up sequences.
dataStore	Allows read and write access to all Mikado variables.
picksearch	Allows access to the results of the handle search.
kinematics	Module for kinematics, collision checking and reachability considerations.
product	Allows access to product-specific properties.
robot	Access to the status of the robot
scene	Allows access to scene objects such as crates, ROIs and obstacles
file	Allows access to files
program	Allows access to Mikado programs
lab	Allows access to Mikado-specific, rarely used but still useful functions.
pointCloud	Allows access to attributes of the point cloud

Ready-to-use templates for specific use cases are listed here:

- *Example: User-defined log file from Mikado*.
- *Example: Sort Mikado PoseList*.

8.1.1 General information

You can print text to the log console within a JavaScript task using:

```
console.log("My log message")
```

Further links to JavaScript in general:

http://doc.qt.io/qt-5/qtqml-javascript-functionlist.html

http://doc.qt.io/qt-5/qjsengine.html#Extension-enum

http://www.ecma-international.org/ecma-262/7.0/index.html

8.1.2 task

The task module can be accessed via the variable `task`. The status text can be edited via `task.status`. If `task.setNextSequence` is set, the sequence with this name is executed next; if the string is empty, the current sequence is continued:

```
task.status = "Running"
task.setNextSequence("Run to goal")
```

In case of problems the Mikado program flow can be stopped permanently by calling `task.setError("description")` or temporarily by calling `task.setErrorMightRetry("Beschreibung")`.

8.1.3 dataStore

Variables can be accessed using the `dataStore` object, using `dataStore.data(name)` or `dataStore.setData(name, value)` respectively. See the following sample code:

```
var a = dataStore.data("a")
a = a + 1
var success = dataStore.setData("a", a)
```

The function `dataStore.setData(name, value)` returns `true` if the value was written successfully. In the event of an error, the return value is `false` and details of the corresponding error can be found in the log (e.g. in the event of a formatting error).

However, for some variable types (Pose, JointPosition and PoseList), the `dataStore.data(name)` function returns internal data types. In this case, the `dataStore.getJsObject(name)` function can be used instead. This returns a JavaScript-typical array or object. It can then be edited directly and, if necessary, written back with `dataStore.setJsObject(name, value)`:

```
var b = dataStore.getJsObject('einePose')
if (b.x < 0)
   b.x = 1.2
dataStore.setJsObject('einePose', b)

var c = dataStore.getJsObject('eineJointPosition')
console.log("J1 ist bei " + c[0])
```

The `dataStore.addVariable(name, type)` function can also be used to create new variables. `dataStore.setVariableInitialValue(name)` causes the current value of the variable to be adopted as the initial value.

8.1.4 picksearch

The `picksearch` module provides access to the results of the pick- and object-search. The following functions are available:

- *getPickedObjectPose*
- *getBestPick*
- *getFoundObjectPoses*
- *setUserBins*
- *getSucceededPicks*
- *getLastUsedGrasp*
- *rerunPickSearch*

- *awaitBestPickPerObject*

getPickedObjectPose

Read the pose of the currently picked object in the world coordinate system. A *Pick object* task must have successfully picked an object beforehand. If no object was picked or the object has already been deposited, the function returns `null`.

Use: `Pose = picksearch.getPickedObjectPose()`

```
var pickedObjectPose = picksearch.getPickedObjectPose()
if (pickedObjectPose != null) {
 dataStore.setJsObject("pickedObjectPose", pickedObjectPose)
}
```

getBestPick

Read information about the best known pick at the time of the call. A *Start handle search* task must have been executed beforehand.

Optionally accepts the name of a product (`productName`), the name of a box (`boxName`) and the name of one or more end-effectors (`endEffectorName(s)`) as arguments to select a specific pick-search. If the function is called without product- and box-name, the last search started is used. If no end-effector is given, the last started pick-search corresponding to the chosen search is used.

Returns a JavaScript object with the following properties:

- All information that is also contained in a `Grasp`, see e.g. *getLastUsedGrasp*
- `objectName` (`String`): Name of the grasped object
- `objectPose` (`Pose`): Pose of the grasped object in the world coordinate system
- `graspTcpPose` (`Pose`): TCP pose of the grasp in the world coordinate system
- `graspTool0Pose` (`Pose`): Tool0 pose of the grasp in the world coordinate system
- `startTcpPose` (`Pose`): Start pose of the grasp (see *Gripping process*)
- `approachTcpPose` (`Pose`): Approach pose of the grasp (see *Gripping process*)
- `retractTcpPose` (`Pose`): Rectract pose of the grasp (see *Gripping process*)
- `finalTcpPose` (`Pose`): Final pose of the grasp (see *Gripping process*)
- `distanceJointSpace` (`Double`): Trajectory distance in joint space

Usage:

- `pick = picksearch.getBestPick(Integer timeout)`

```
task.status = ""

// eine Sekunde warten falls noch kein Griff gefunden wurde
var bestPickObj = picksearch.getBestPick(1000)

// Informationen über den besten Griff ausgeben
if (bestPickObj != null) {
 task.status = "Object: "+bestPickObj["objectName"]+"\n"
 + "Best pick: "+bestPickObj["graspName"]+"\n"
 + "Category: "+bestPickObj["category"]+"\n"
 + "Distance: "+bestPickObj["distanceJointSpace"]
```

(continues on next page)

(continued from previous page)

```
 dataStore.setJsObject("bestObjectPose", bestPickObj["objectPose"])
 dataStore.setJsObject("bestPickTcp", bestPickObj["graspTcpPose"])
 dataStore.setJsObject("bestPickTool0", bestPickObj["graspTool0Pose"])
 dataStore.setJsObject("bestPickStartTcp", bestPickObj["startTcpPose"])
 dataStore.setJsObject("bestPickApproachTcp", bestPickObj["approachTcpPose"])
 dataStore.setJsObject("bestPickRetractTcp", bestPickObj["retractTcpPose"])
 dataStore.setJsObject("bestPickFinalTcp", bestPickObj["finalTcpPose"])
} else {
 task.status = "Kein Griff gefunden"
}
```

- `pick = picksearch.getBestPick(Integer timeout, String productName, String boxName)`

```
var bestPickObj = picksearch.getBestPick(1000, "product", "box")
```

- `pick = picksearch.getBestPick(Integer timeout, String productName, String boxName, String endEffectorName)`

```
var bestPickObj = picksearch.getBestPick(1000, "product", "box", "gripper_group")
```

- `pick = picksearch.getBestPick(Integer timeout, String productName, String boxName, StringList endEffectorNames)`

```
var bestPickObj = picksearch.getBestPick(1000, "product", "box", ["gripper_group1
↪", "gripper_group2"])
```

getFoundObjectPoses

Returns a list of the positions of all objects found.

Optionally accepts the name of a product (`productName`) and the name of a box (`boxName`) as arguments to select a specific pick-search. If the function is called without arguments, the last pick-search started is used.

Use without arguments: `PoseList = picksearch.getFoundObjectPoses()`

```
// eine Liste aller gefundenen Objekte auslesen und in eine Mikado
// Variable speichern
var foundPoses = picksearch.getFoundObjectPoses()
dataStore.setJsObject("foundPoses", foundPoses)
```

Use with arguments: `PoseList = picksearch.getFoundObjectPoses(String productName, String boxName)`

```
// eine Liste aller gefundenen Objekte einer bestimmten Griffsuche auslesen und in␣
↪eine Mikado
// Variable speichern
var foundPoses = picksearch.getFoundObjectPoses("productName", "boxName")
dataStore.setJsObject("foundPoses", foundPoses)
```

setUserBins

This function can be used to set a bin-value for each object, which can be used with the `user_binning` sorting criteria (see *Prioritization*). The number of entries in the list (`bins`) must correspond exactly to the number of objects found.

Optionally accepts the name of a product (`productName`), the name of a box (`boxName`) and the name of one or more end-effectors (`endEffectorName(s)`) as arguments to select a specific pick-search. If the function is called without product- and box-name, the last search started is used. If no end-effector is given, the last started pick-search corresponding to the chosen search is used.

Use without arguments: `picksearch.setUserBins(IntegerList bins)`

```
// User-Bins setzen ([0, 1, 2, ...]). Die Anzahl der Objekte wird über die Objekt-
↪Liste ausgelesen.
var numObjects = picksearch.getFoundObjectPoses().length
var bins = []
for (var i = 0; i < numObjects; ++i) bins.push(i)
picksearch.setUserBins(bins)
```

Usage with Arguments:

- `picksearch.setUserBins(IntegerList bins, String productName, String boxName)`

  ```
  // User-Bins einer bestimmten Griffsuche setzen ([0, 1, 2, ...]). Die Anzahl der␣
  ↪Objekte wird über die Objekt-Liste ausgelesen.
  var numObjects = picksearch.getFoundObjectPoses("productName", "boxName").length
  var bins = []
  for (var i = 0; i < numObjects; ++i) bins.push(i)
  picksearch.setUserBins(bins, "productName", "boxName")
  ```

- `picksearch.setUserBins(IntegerList bins, String productName, String boxName, String endEffectorName)`

  ```
  ...
  picksearch.setUserBins(bins, "productName", "boxName", "gripper_group")
  ```

- `picksearch.setUserBins(IntegerList bins, String productName, String boxName, StringList endEffectorNames)`

  ```
  ...
  picksearch.setUserBins(bins, "productName", "boxName", ["gripper_group1",
  ↪"gripper_group2"])
  ```

getSucceededPicks

Returns a list of all previously successful picks. The sorting follows the normal PickSearch criteria.

Optionally accepts the name of a product (`productName`), the name of a box (`boxName`) as arguments to select a specific search. Furthermore, optionally accepts the name of one or more end-effectors (`endEffectorName(s)`) as arguments to select a specific pick-search. If the function is called without product- and box-name, the last search started is used. If no end-effector is given, the last started pick-search corresponding to the chosen search is used.

Use without arguments: `picks = picksearch.getSucceededPicks()`

```
// Liste aller erfolgreichen Griffe auf der Konsole ausgeben
var picks = picksearch.getSucceededPicks()

for (var i = 0; i < picks.length; ++i) {
```

(continues on next page)

(continued from previous page)

```
 console.info( "Object: "+picks[i].objectName+"\n"
+ "grasp: "+picks[i].graspName+"\n"
+ "Category: "+picks[i].category+"\n"
+ "Distance: "+picks[i].distanceJointSpace);
}
```

Usage with Arguments:

- `picks = picksearch.getSucceededPicks(String productName, String boxName)`

```
// Liste aller erfolgreichen Griffe auf der Konsole ausgeben
var picks = picksearch.getSucceededPicks("productName", "boxName")

for (var i = 0; i < picks.length; ++i) {
 console.info( "Object: "+picks[i].objectName+"\n"
+ "grasp: "+picks[i].graspName+"\n"
+ "Category: "+picks[i].category+"\n"
+ "Distance: "+picks[i].distanceJointSpace);
}
```

- `picks = picksearch.getSucceededPicks(String productName, String boxName, String endEffectorName)`

```
var picks = picksearch.getSucceededPicks("productName", "boxName", "gripper_group
↪")
...
```

- `picks = picksearch.getSucceededPicks(String productName, String boxName)`

```
var picks = picksearch.getSucceededPicks("productName", "boxName", ["gripper_
↪group1", "gripper_group2"])
...
```

getLastUsedGrasp

Read information about the current (last used) grasp. An object must have been picked beforehand with the *Pick object* task .

Returns a JavaScript object with the following properties:

- `graspName` (`String`): Name of the grasp
- `category` (`String`): Grasp category (“Preferred” / “Standard” / “Experimental”)
- `objectToTcpTransform` (`Pose`): TCP pose of the grasp in the local coordinate system of the object
- `objectToTool0Transform` (`Pose`): Tool0 pose of the grasp in the local coordinate system of the object
- `graspOffset` (`Pose`): Transformation between grasped object and gripper tcp
- `isInwards` (`Bool`): Grasp is inwards (true) oder outwards (false)
- `gripper` (`String`): Name of the gripper used for this grasp
- `tcpName` (`String`): Name of the tcp used for this grasp

Use: `grasp = picksearch.getLastUsedGrasp()`

```
// Informationen über den aktuellen Griff ausgegeben
var grasp = picksearch.getLastUsedGrasp()
```

(continues on next page)

(continued from previous page)

```
task.status = "Verwendeter Griff: "+grasp["graspName"]+"\n"
+ "Kategorie: "+grasp["category"]+"\n"
```

rerunPickSearch

Rerun a previously started pick-search without repeating the object search. This function is mainly used in applications with several grippers (or TCPs) to calculate picks for different grippers one after the other without having to repeat the object search each time.

Optionally accepts the name of a product (`productName`), the name of a box (`boxName`) as arguments to select a specific search. Furthermore, optionally accepts the name of one or more end-effectors (`endEffectorName(s)`) as arguments to select a specific pick-search. If the function is called without product- and box-name, the last search started is used. If no end-effector is given, the last started pick-search corresponding to the chosen search is used.

Usage without arguments:

- `rerunPickSearch()`

```
var success = picksearch.rerunPickSearch()
```

- `rerunPickSearch(String searchParams)`

```
var success = picksearch.rerunPickSearch("Suchparameter mit pick_search.yaml")
```

Usage with Arguments:

- `rerunPickSearch(String productName, String boxName)`

```
picksearch.rerunPickSearch("productName", "boxName")
```

- `rerunPickSearch(String productName, String boxName, String endEffectorName)`

```
picksearch.rerunPickSearch("productName", "boxName", "gripper_group")
```

- `rerunPickSearch(String productName, String boxName, String endEffectorName, String searchParams)`

```
picksearch.rerunPickSearch("productName", "boxName", "gripper_group",
↪"Suchparameter mit pick_search.yaml")
```

- `rerunPickSearch(String productName, String boxName, StringList endEffectorNames)`

```
picksearch.rerunPickSearch("productName", "boxName", ["gripper_group1", "gripper_
↪group2"])
```

- `rerunPickSearch(String productName, String boxName, String endEffectorName)`

```
picksearch.rerunPickSearch("productName", "boxName", ["gripper_group1", "gripper_
↪group2"], "Suchparameter mit pick_search.yaml")
```

awaitBestPickPerObject

Wait until the pick-search is finished and return a list of the best valid pick per object. The returned list contains JavaScript objects with the properties of a `Pick`, see *getBestPick*.

Optionally accepts the name of a product (`productName`) and the name of a box (`boxName`) as arguments to select a specific pick-search. If the function is called without arguments, the last pick-search started is used.

Use without arguments: `GraspList = awaitBestPickPerObject()`

```
var bestPickPerObjectList = picksearch.awaitBestPickPerObject()
```

Usage with Arguments:

- `GraspList = awaitBestPickPerObject(String productName, String boxName)`

  ```
  var bestPickPerObjectList = picksearch.awaitBestPickPerObject("productName",
  ↪"boxName")
  ```

- `GraspList = awaitBestPickPerObject(String productName, String boxName, String endEffectorName)`

  ```
  var bestPickPerObjectList = picksearch.awaitBestPickPerObject("productName",
  ↪"boxName", "gripper_group")
  ```

- `GraspList = awaitBestPickPerObject(String productName, String boxName, StringList endEffectorNames)`

  ```
  var bestPickPerObjectList = picksearch.awaitBestPickPerObject("productName",
  ↪"boxName", ["gripper_group1", "gripper_group1"])
  ```

8.1.5 kinematics

The `kinematics` module allows access to functions for calculating the forward and backward kinematics. It also provides functions for checking the achievability of a target pose for a given start pose. The following functions are available:

- *forwardKinematics*
- *inverseKinematics*
- *isMovePossible*
- *isMovePossibleCartesian*

forwardKinematics

Calculates the pose of a target link for given joint angles. The pose of "tool0" is calculated by default.

Returns an object with the following keys:

- `success` (`Bool`): Forward kinematics could be solved?
- `result` (`Pose`): Pose of the target link. Is only set if successfully executed.

Usage: `Var = forwardKinematics(JointPose, String = "tool0")`

```
// Berechne TCP-Pose zu gegebenen Gelenkwinkeln
var jointPose = dataStore.getJsObject('EineJointPose')
var map = kinematics.forwardKinematics(jointPose, "gripper_tcp")
```

(continues on next page)

(continued from previous page)

```
if (map.success) {
    dataStore.setJsObject('Pose_TCP', map.result)
}
```

inverseKinematics

Calculates the robot's joint angles so that a target link assumes the given pose. By default, "tool0" is used as the target link. Optionally, an initial value can be set for the joint angle optimization.

Returns an object with the following keys:

- `success` (`Bool`): Have errors occurred? The following return values are only set if the execution was successful.
- `ikCollisionFree` (`Bool`): Collision-free solution found?
- `ikSolutionFound` (`Bool`): Solution found?
- `result` (`JointPose`): Gelenkwinkel des Roboters.

Usage: `Var = inverseKinematics(Pose, JointPose = [0,0,0,0,0,0], String = "tool0")`

```
// Berechne die Gelenkwinkel zu gegebener TCP-Pose
var tcpPose = dataStore.getJsObject('Pose_TCP')
var map = kinematics.inverseKinematics(tcpPose, [0,0,0,0,0,0] ,"gripper_tcp")
dataStore.setJsObject('JPose_Griff', map.result)

if (!map.ikSolutionFound) {
    task.setNextSequence("Griff ist nicht erreichbar")
} elseif (!map.ikCollisionFree) {
    task.setNextSequence("Griff ist kollisionsbehaftet")
}
```

isMovePossible

Checks whether a collision-free path can be found from the start pose to the target pose. By default, the current robot pose is used as the start pose. Both the start and target pose must be passed as JointPose. If necessary, use *inverseKinematics* to convert a pose into a JointPose.

Returns an object with the following keys:

- `success` (`Bool`): Have errors occurred? The following return values are only set if the execution was successful.
- `isPossible` (`Bool`): Collision-free path found?
- `reason` (`String`): Information on why the destination may not be reachable.

Verwendung: `Var = isMovePossible(JointPose goal, JointPose start = Var())`

```
var goal = dataStore.getJsObject('EineJointPose')
var start = dataStore.getJsObject('AndereJointPose')
var map = kinematics.isMovePossible(goal, start)

console.log("move possible: " + map.isPossible)
console.log("reason: " + map.reason)
```

isMovePossibleCartesian

Checks whether a Cartesian, collision-free path can be found from the start pose to the target pose. A Cartesian path is the direct line between the start and target pose in Cartesian space. The target pose must be passed as a pose.

The start pose must be transferred as a JointPose. By default, the current robot pose is used as the start pose. If necessary, use *inverseKinematics* to convert a pose into a JointPose.

Optionally, a target link can be transferred, which is to be moved cartesian to a given pose. By default, "tool0" is used as the target link.

Returns an object with the following keys:

- `success` (`Bool`): Have errors occurred? The following return values are only set if the execution was successful.
- `isPossible` (`Bool`): Collision-free path found?
- `goalJointPosition` (`JointPosition`): last point of trajectory (goal) but only given if isPossible = true (type: JointPosition)
- `reason` (`String`): Information on why the destination may not be reachable.

Verwendung: `Var = isMovePossibleCartesian(Pose goal, JointPose start = Var(), String targetLink = "tool0")`

```
// Überprüfen ob sich der Greifer 10cm senkrecht nach unten fahren lässt
var start_pose = dataStore.getJsObject('Pose_TCP')
var start_pose_IK = kinematics.inverseKinematics(start_pose, [0,0,0,0,0,0], "gripper_
↪tcp") // convert to JointPose
dataStore.setJsObject('start_pose_IK', start_pose_IK.result)

var goal_pose = dataStore.getJsObject('Pose_TCP')
goal_pose.z = goal_pose.z - 0.1
var map = kinematics.isMovePossibleCartesian(goal_pose, start_pose_IK.result,
↪"gripper_tcp")
dataStore.setJsObject('goal_pose', map.goalJointPosition)

console.log("move possible: " + map.isPossible)
console.log("reason: " + map.reason)
```

8.1.6 product

The `product` module provides access to product-specific properties. The following functions are available:

- *activeProduct*
- *getUsedEndEffectorNames*
- *getGraspOffset*
- *getDimensions*
- *getAllProductNames*

activeProduct

Reads or sets the current product. The current product can also be set using the *Set active product* task.

Usage: `String = activeProduct` or `activeProduct = String`

```
// Setzen des aktiven Produkts
product.activeProduct = "P_1"

// Lesen des aktiven Produkts
task.status = "active product is " + product.activeProduct
```

getUsedEndEffectorNames

For applications with several grippers, this function can be used to query a list of all grippers that are configured for the active product. A gripper is configured for a product on the product page.

Note: This function is not relevant for applications with only one gripper.

Usage: `StringList = getUsedEndEffectorNames()`

```
// gibt Liste an angelegten Greifern von aktivem Produkt zurück
var greiferListe = product.getUsedEndEffectorNames()
```

getGraspOffset

Reading the grip pose (or the position of the part in the gripper) of a specific grip for the active product.

Use: `Pose = getGraspOffset(String)`

```
// Lese die Pose eines bestimmten Griffs
var graspPos = product.getGraspOffset("Außengriffe | Basisgriff")
task.status = "x: " + graspPos.x + " y: " + graspPos.y  + " z: " + graspPos.z
    + " r: " + graspPos.roll  + " p: " + graspPos.pitch  + " y: " + graspPos.yaw

// Überschreibe die Pose des angehängten Teils
lab.setCurrentGraspOffset(graspPos)
```

In the example code above, the pose of the “basic handle” is read. The handle definition matching the example code is shown in the screenshot below. The handle pose is then used to overwrite the pose of the currently attached part. This can be useful, for example, if the part is centered in the gripper due to positive locking. See also the example for *setCurrentGraspOffset*.

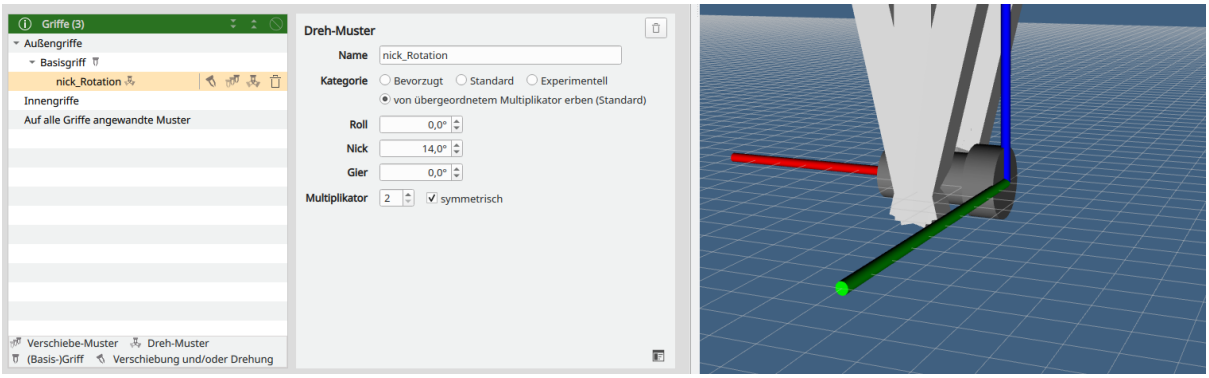


Fig. 1: Example entries in the handle editor

getDimensions

Calculates the smallest possible enclosing cuboid for the specified product and returns its dimensions. The orientation is aligned with the product coordinate system (Axis-align Bounding Box AABB). If no product is specified, the dimensions of the active product are calculated.

Returns an object with the following keys:

- `x` (`Double`): Dimension in X-direction of the product coordinate system [m]
- `y` (`Double`): Dimension in Y-direction of the product coordinate system [m]
- `z` (`Double`): Dimension in Z-direction of the product coordinate system [m]

Use: `Var = getDimensions(String productName = "")`

```
// Auslesen der aktuellen Produktabmessungen
var dims = product.getDimensions()
task.status = "active product has following dimensions:\nx: " + dims.x + "\ny: " +␣
↪dims.y + "\nz: " + dims.z

// Auslesen von spezifischen Produktabmessungen
var dims = product.getDimensions("P_1")
task.status = task.status + "\nSpecific product has following dimensions:\nx: " +␣
↪dims.x + "\ny: " +  dims.y + "\nz: "  + dims.z
```

Hint: The product dimensions can also be accessed using the *Read out product dimensions* task.

---

getAllProductNames

Read out list with all product names of the active product list.

Use: `productList = product.getAllProductNames()`

```
var productList = product.getAllProductNames()

// Beispiel: Überprüfe, ob ein bestimmtes Produkt existiert
if(productList.includes("Produkt_1")){
    task.status = "Produkt_1 ist vorhanden"
} else {
    task.status = "Produkt_1 ist nicht vorhanden"
}
```

8.1.7 robot

The following functions are available:

- *getLinkTransform*
- *readInput*
- *readOutput*
- *writeOutput*
- *getCurrentTipLink*
- *getCurrentEndEffectorGroupName*

- *isObjectAttached*
- *runRobotProgram*

getLinkTransform

Returns the absolute transformation of a link relative to the world frame based on the target position after all previous tasks have been executed. To read out the actual position without preprocessing, *getTF* can be used.

Use: `pose = robot.getLinkTransform(LinkName)`:

```
var pose = robot.getLinkTransform("gripper_tcp")

if (pose.z < 0.1) {
    console.log("Greifer ist nahe am Boden")
    task.setNextSequence("Abstand gewinnen")
}
```

readInput

Retrieves the value of a robot input or field-bus bit depending on configuration.

Use: `value = robot.readInput(Number)`

```
var input = robot.readInput(23)
```

readOutput

Retrieves the value of a robot output or field-bus bit depending on configuration.

Use: `value = robot.readOutput(Number)`

```
var output = robot.readOutput(23)
```

writeOutput

Sets the value of a robot output or field-bus bit depending on configuration.

Use: `value = robot.writeOutput(Number, Value)`:

```
var output = robot.writeOutput(23, true)
```

getCurrentTipLink

Read the name of the currently active TCP. Particularly useful for robot cells with interchangeable grippers, where there are several grippers and therefore several TCPs, but only one of which is active at any one time.

Use: `tipLinkName = robot.getCurrentTipLink()`:

```
var tipLinkName = robot.getCurrentTipLink()
task.status = "Aktiver TCP: " + tipLinkName
```

getCurrentEndEffectorGroupName

Read the name of the currently active gripper. Particularly useful for robot cells with interchangeable grippers, where there are several grippers, but only one of them is active at any one time.

Use: `endEffectorName = robot.getCurrentEndEffectorGroupName()`:

```
var endEffectorName = robot.getCurrentEndEffectorGroupName()
task.status = "Aktiver Greifer: " + endEffectorName
```

isObjectAttached

Reads out whether an object is currently attached to the gripper in Mikado, i.e. an object has been gripped. Returns `true` if an object is attached, otherwise `false`.

Use: `objectAttached = robot.isObjectAttached()`:

```
var objectAttached = robot.isObjectAttached()

if(objectAttached) {
    task.status = "Objekt am Greifer angehängt"
} else {
    task.status = "Kein Objekt am Greifer angehängt"
}
```

runRobotProgram

Allows running a native robot program and get back its result. In contrast to *Execute robot program* task, this function can pass any number of arguments of each variable type. Also Mikado does not enforce that no motions are commanded in parallel. If needed this should be enforced using *Waiting until standstill*. Poses are transmitted in the coordinate system fixed to the robots base.

Use: `result = robot.runRobotProgram(name, arguments)`:

```
args = [{
    name: "test123",
    value: false,
    type: "bool"
},
{
    name: "testabc",
    value: true,
    type: "bool"
},
{
    name: "testInt",
    value: 23,
    type: "integer"
},
{
    name: "testDouble",
    value: 3.14159,
    type: "double"
},
{
    name: "testabc",
    value: {x: 1, y: 0.23, z: 2, roll: 3, pitch:4, yaw: 6},
```

(continues on next page)

(continued from previous page)

```
    type: "pose"
},
{
    name: "testabcd",
    value: dataStore.getJsObject("upperRight"),
    type: "pose"
}
]
var result = robot.runRobotProgram("test", args)
task.status = result
```

8.1.8 scene

The following functions are available:

- *getSceneObjectDimensions*
- *setSceneObjectDimensions*
- *getSceneObjectPose*
- *setSceneObjectPose*
- *roiContainsPoint*

getSceneObjectDimensions

Returns the size of an ROI or the size of the placeholder (default size) for an obstacle or a box. The size of a mesh/CAD is not taken into account.

Use: `size = scene.getSceneObjectDimensions(Name)`

```
var size = scene.getSceneObjectDimensions("ROI1")
console.log("ROI1 ist "+size.x+" * "+size.y+" * "+size.z+" [m] groß")
```

setSceneObjectDimensions

Changes back the size of an ROI or the size of the placeholder (default size) for an obstacle or a box. The size/scaling of a mesh/CAD is not changed.

Verwendung: `scene.setSceneObjectDimensions(Name, valueX, valueY, valueZ)`:

```
var size = scene.getSceneObjectDimensions("ROI1")
scene.setSceneObjectDimensions("ROI1", size.x, size.y, 0.23)
```

getSceneObjectPose

Returns the pose of a scene object (in meters and radians).

Use: `Pose = scene.getSceneObjectPose(Name)`:

```
var pose = scene.getSceneObjectPose("ROI1")
console.log("ROI1 ist bei x: "+pose.x+" y: "+pose.y+" z: "+pose.z+" roll: "+pose.roll+
↪" pitch: "+pose.pitch+" yaw: "+pose.yaw)
```

setSceneObjectPose

Sets the position of a scene object.

Usage: `scene.setSceneObjectPose(Name, Pose)`:

```
var pose = scene.getSceneObjectPose("ROI1")
console.log("ROI1 ist bei x: "+pose.x+" y: "+pose.y+" z: "+pose.z+" roll: "+pose.roll+
↪" pitch: "+pose.pitch+" yaw: "+pose.yaw)
pose.z += 0.1
scene.setSceneObjectPose("ROI1", pose)
```

roiContainsPoint

Check whether a point in three-dimensional space lies within a specific ROI. `name` is the name of the ROI to be checked. The coordinates `x`, `y` and `z` must be passed in meters as a unit. Returns `true` if the point is within the ROI, otherwise `false`.

Use: `scene.setSceneObjectDimensions(name, x, y, z)`:

```
var pose = dataStore.getJsObject('einePose')
var result = scene.roiContainsPoint("ROI", pose.x, pose.y, pose.z)
if(result){
    task.status = "Punkt liegt innerhalb der ROI"
} else {
    task.status = "Punkt liegt außerhalb der ROI"
}
```

8.1.9 `file`

The `file` module provides access to text files. The following operators are available:

- *readCsvAsDict*

readCsvAsDict

This function reads in a text file in CSV format. It returns a JavaScript dictonary (keys/values). The first column is used as key, the remaining columns are returned as list.

The very first line is ignored as it typically contains a table header comment.

Relative file names are interpreted with respect to the cell directory.

Usage: `dict = file.readCsvAsDict(filename, separator)`

```
//path aus der Variablen „output_folder“ laden
var path = dataStore.data("ausgabe_ordner")
var filename = path + "test.csv"
var dict = file.readCsvAsDict(filename, ";");
console.log("JS Dictionary Ausgabe: " + JSON.stringify(dict));
task.status = "Wert für Produkt 1: "+dict["Produkt1"][0]
```

In this example the file `test.csv` is read from a configurable folder (Mikado Variable `ausgabe_ordner`). The first parameter (column) after the name is printed to the Task status.

8.1.10 program

The JavaScript `program` module provides access to Mikado programs. The following functions are available:

- *save*

save

Save the current Mikado program. This function can be used, for example, to automatically save changes to initial values of variables during program execution so that they are retained even after the application is restarted. An error is triggered if saving fails.

Use: `program.save()`

8.1.11 lab

The JavaScript Lab provides access to rarely used but nevertheless useful functions. The following functions are available:

- *invertPose*
- *getPositionAverage*
- *getPoseDistanceTrans*
- *getPoseDistanceRot*
- *getPoseDifference*
- *getPoseRotAxis*
- *writeStringToFile*
- *getCurrentDateTime*
- *getPoseAsJson*
- *getTF*
- *sendRobotCommand*
- *getRobotVelocityScaling*
- *setRobotVelocityScaling*
- *getRobotAccelerationScaling*
- *setRobotAccelerationScaling*
- *getCurrentGraspName*

- *getCurrentGraspOffset*
- *setCurrentGraspOffset*
- *getCurrentGraspPieceToToolTransform*
- *mult*
- *clearAllRecognizedObjects*
- *updateOctomap*
- *setSceneObjectDimension*
- *callExtensionService*
- *setGetJsonFromCamera*

invertPose

Inverts a Pose.

Use: `Pose = invertPose(Pose)`

```
var pose = dataStore.getJsObject('einePose')
var pose_inv = lab.invertPose(pose)
```

getPositionAverage

Calculates the mean value of the x-y-z coordinates of all poses in a PoseList.

Use: `Pose = getPositionAverage(PoseList)`

```
var poselist = dataStore.getJsObject('einePoseList')
var averagePosition = lab.getPositionAverage(poselist)
console.log(
    "Mittlere Position ist: "
    + averagePosition.x + " "
    + averagePosition.y + " "
    + averagePosition.z)
```

getPoseDistanceTrans

Calculates the Euclidean distance between the positions of two poses.

Use: `Double = getPoseDistanceTrans(Pose, Pose)`

```
var pose1 = dataStore.getJsObject('einePose')
var pose2 = dataStore.getJsObject('anderePose')
var Abstand = lab.getPoseDistanceTrans(pose1, pose2)
```

getPoseDistanceRot

Calculates the relative rotation between the orientations of two poses.

Use: `Double = getPoseDistanceRed(Pose, Pose)`

```
var pose1 = dataStore.getJsObject('einePose')
var pose2 = dataStore.getJsObject('anderePose')
var Verdrehung = lab.getPoseDistanceRot(pose1, pose2)
```

getPoseDifference

Calculates the difference between two poses according to the following scheme, where R_i describes the rotational component and t_i the translational component of the pose i:

$$T = \begin{bmatrix} R_1 R_2^{-1} & t_1 - t_2 \\ 0 & 1 \end{bmatrix} .$$

Use: `Pose = getPoseDifference(Pose, Pose)`

```
var pose1 = dataStore.getJsObject('einePose')
var pose2 = dataStore.getJsObject('anderePose')
var poseDiff = lab.getPoseDifference(pose1, pose2)
dataStore.setJsObject("poseDiff", poseDiff)
```

getPoseRotAxis

Calculates the angular deviation between the z-axis of a pose with the rotation component R and the z-axis of the global coordinate system:

$$\alpha = \cos^{-1}(Rz \cdot z) .$$

By default, the angular deviation from the z-axis is calculated. By adding a second parameter, the operator can also be used to calculate the deviation from the x or y axis.

Use: `Double = getPoseRotAxis(Pose, String = "Z")`

```
var pose1 = dataStore.getJsObject('einePose')
var zRotOffset = lab.getPoseRotAxis(pose1)
var xRotOffset = lab.getPoseRotAxis(pose1, "X")

// überprüfen ob Teil aufrecht gegriffen werden kann
var Teilpose = dataStore.getJsObject("Teilpose")
var rot_z = 180/Math.PI*lab.getPoseRotAxis(Teilpose, "Z")
var TeilAufrecht = (rot_z < 10) ? true : false // Abweichung kleiner als 10 deg?
dataStore.setData("TeilAufrecht", TeilAufrecht)
task.status = "TeilAufrecht: " + TeilAufrecht
```

writeStringToFile

Writes a string to a file.

Verwendung: `Bool = writeStringToFile(String outputFilePath, String fileContent, Bool appendToExisting = false)`

```
// save data
var append = false // überschreibt Datei, falls false
var path = dataStore.data("Ausgabe_Ordner")
var content = "Test Output"
var filename = path + "test.csv"

lab.writeStringToFile(filename, content, append)
console.log("JS Auswertung: AUSGABE nach: "  + filename)
```

getCurrentDateTime

Returns the current time.

Use: `String = getCurrentDateTime()`

```
var dateTime = lab.getCurrentDateTime() // dateTime = <YYYY>-<MM>-<DD>T<hh>:<mm>:<ss>
```

getPoseAsJson

Converts the given pose into Json format so that it can be used in NxTreeEdit, for example.

Use: `String = getPoseAsJson(Pose)`

```
var pose1 = dataStore.getJsObject('einePose')
var poseAsJson = lab.getPoseAsJson(pose1)
console.log("Json Darstellung:\n" + poseAsJson)
```

getTF

Calculates the transformation between two Urdf objects. Urdf objects are defined in *Cell geometry*.

Usage: `Pose = getTF(String originFrameName, String goalFrameName)`

```
var tool0_zu_gripperTcp = lab.getTF('tool0', 'gripper_tcp')
dataStore.setJsObject("tool0_zu_gripperTcp", tool0_zu_gripperTcp)
```

sendRobotCommand

Sends an R3 command to the connected robot.

> **Warning:** Only works with Mitsubishi robots.

Use: `String = sendRobotCommand(String)`

```
var response = lab.sendRobotCommand('1;1;OVRD=25') // setze Geschwindigkeit auf 25%
```

getRobotVelocityScaling

Reads out the currently used speed scaling for the robot's path planning.

Use: `Double = getRobotVelocityScaling()`

```
var velocityScaling = lab.getRobotVelocityScaling()
```

setRobotVelocityScaling

Sets the velocity scaling for the robot's path planning, see *Speed settings* for details.

Use: `setRobotVelocityScaling(Double)`

```
lab.setRobotVelocityScaling(0.4) // setze Geschwindigkeit-Skalierung auf 40%
```

getRobotAccelerationScaling

Reads out the currently used acceleration scaling for the robot's path planning.

Use: `Double = getRobotAccelerationScaling()`

```
var accScaling = lab.getRobotAccelerationScaling()
```

setRobotAccelerationScaling

Sets the acceleration scaling for the robot's path planning, see *Speed settings* for details.

Usage: `setRobotAccelerationScaling(Double)`

```
lab.setRobotAccelerationScaling(0.4) // setze Beschleunigungs-Skalierung auf 40%
```

getCurrentGraspName

Reads the name of the last handle used. The names of the handles are displayed in the *Handle inspection* and are derived from the *Gripping poses* of a product. If the pick poses are named in the same way, the handle name can be used to draw conclusions about the position of the part in the picker.

Use: `String = getCurrentGraspName()`

```
var name = lab.getCurrentGraspName();
var drehung = 0.0
if (name.includes("180 gedreht")) {
    drehung = 180.
}
dataStore.setData("Var_Drehwinkel", drehung);
dataStore.setData("Var_GraspName", name);
task.status = "CurrentGraspName = " + name + " Var_Drehwinkel = " + dataStore.data(
↪"Var_Drehwinkel")
```

getCurrentGraspOffset

Reads the pose representation of the transformation matrix from attached part to handle point (see *Details* for description of the position of the handle point) of the last used handle.

See also *getCurrentGraspPieceToToolTransform* for the transformation from the attached part to Tool0.

In the following example, the position of a part in the gripper is analyzed. The situation shown in the figure below is considered. The coordinate system (red, green, blue corresponds to x, y, z) of the gripper TCP and the part is shown. For the depositing process of the part, we are interested in the orientation of the part in the gripper. In the case shown, the z-axis of the part can point either in the positive or negative y-direction of the gripper TCP. This results from the definition of the 'gripper_tcp' (see *Mikado Key-Links*) and the grips created for the product (part). In our example, both axes point in opposite directions. The following sample code provides the desired result.

Use: `Pose = getCurrentGraspOffset()`

```
// Check ob die z-Achse des momentan gegriffenen Teils in positive y-Richtung des TCP␣
↪zeigt

// Transformation von Teil zu Griffpunkt
var part_T_grasp = lab.getCurrentGraspOffset()

// Translation null setzen -> "verschiebt" Teil in Griffpunkt
part_T_grasp.x = 0
part_T_grasp.y = 0
part_T_grasp.z = 0

// mit Einheitsvektor Z multiplizieren
var zDir = lab.mult(part_T_grasp, [0,0,1]) // (part)e_z = (part)T_(grasp) * (grasp)e_z

var aligned = (zDir[1] > 0) // Vergleich mit der Y-Achse des Griffpunkts

task.status = "z-Achse Teil und y-Achse TCP zeigen in die gleiche Richtung?: " +␣
↪aligned
```

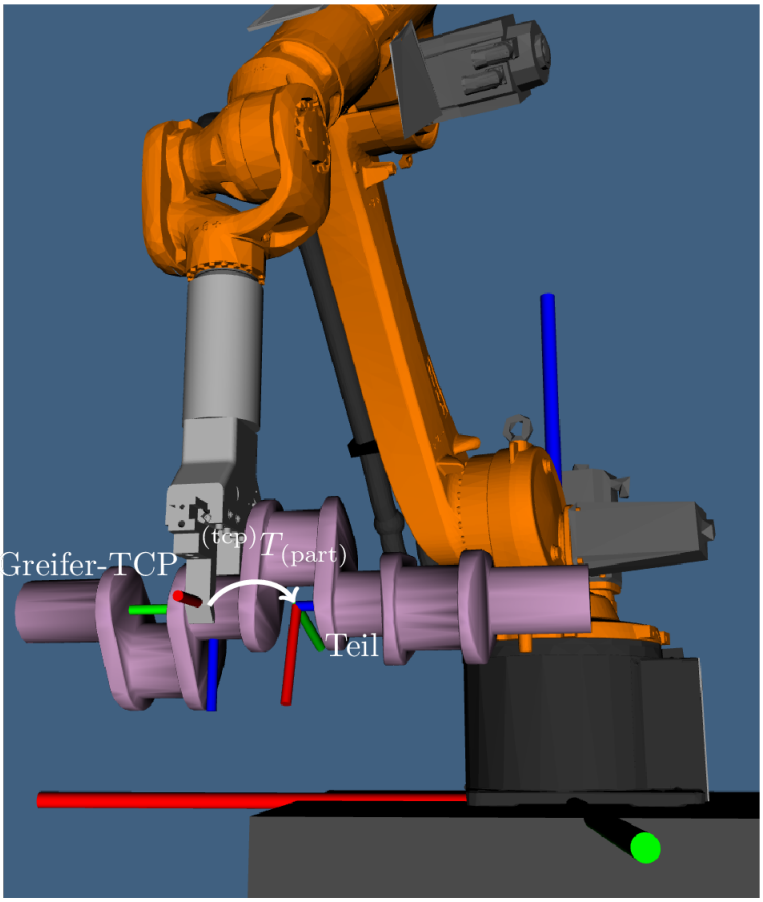


Fig. 2: Illustration of the example for the analysis of the currently accessed part

Details

The following sketch illustrates the position of the grip point:

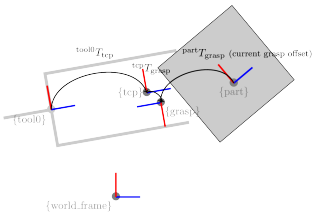


Fig. 3: Sketch of the position of the grip point

The grasp point transformation ${}^{\text{part}}T_{\text{grasp}}$ describes the relative position of the attached part to the grasp point. It should be noted here that there is a y flip ${}^{\text{tcp}}T_{\text{grasp}}$ between the reference system TCP and the reference system grasp:

$$T = \begin{bmatrix} R(180°, y) & 0 \\ 0 & 1 \end{bmatrix}.$$

setCurrentGraspOffset

Overwrites the transformation matrix from attached part to handle point (see *Details*).

This can be useful if the physical position of the part in the gripper deviates from the modeled position. This deviation can be caused, for example, by an integrated centering device or form fit of the gripper used. Or also due to inaccuracies in the object recognition and consequently in the gripping process. The actual position of the part in the gripper can, for example, be determined by an additional camera image ("grip check").

This operator can be used to update the model representation and subsequent tasks such as *Move part to* change their behavior accordingly.

See also *getCurrentGraspOffset* for the corresponding read function.

Note: This operator only has an effect if a part is currently in the gripper.

Note: The collision check and path planning only uses the new position of the part in the gripper from this JavaScript task onwards. If there are still jobs in the robot action queue at the time this task is executed, these were planned with the old part position. The corrected part position is not taken into account retroactively for trips that have already been planned.

Use: `bool = setCurrentGraspOffset(Pose)`

```
// Transformation von Teil zu Grasp
var part_T_grasp = lab.getCurrentGraspOffset()

// Nick null setzen -> Teil-Koordinatensystem wird an TCP-Koordinaten System␣
↪ausgerichtet
part_T_grasp.pitch = 0

// aktualisiere Pose des gegriffenen Teils
lab.setCurrentGraspOffset(part_T_grasp)
```

In the figure below, the gripper is centered by positive locking, i.e. it can be assumed that each gripped part is aligned with the gripper fingers (left image). However, since the part was approached slightly tilted with the gripped part, this tilting is also taken into account in the model representation (middle image). This leads to a significant deviation between the model and the actual scene. To correct this deviation, the model representation can be updated by applying the example code above (see right image).

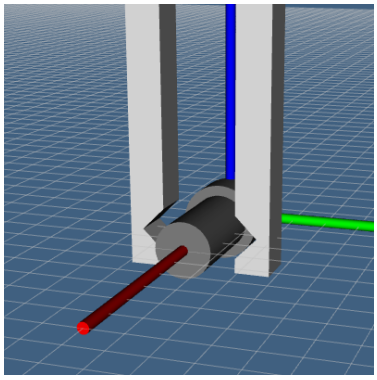


Fig. 4: Positive fit results from gripper and part geometry

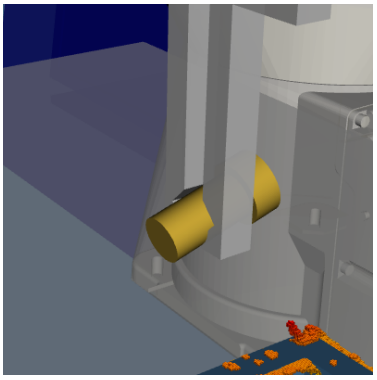


Fig. 5: Initial situation

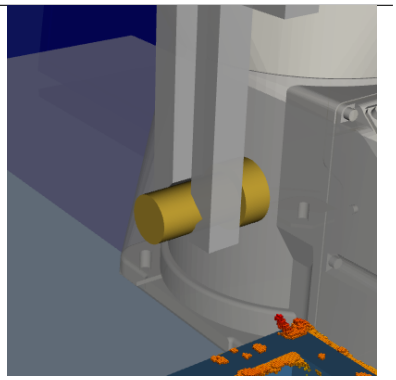


Fig. 6: Situation after using the `setCurrentGraspOffset` operator

If the position of the gripped part in the world coordinate system is known, for example by using the *Find object*

task, the position of the attached part in the gripper can be readjusted using the following transformation:

$$ {}^{\text{match}}T_{\text{grasp}} = \left(\underbrace{{}^{\text{tcp}}T_{\text{world}}}_{\text{urdf}} \quad \underbrace{{}^{\text{world}}T_{\text{match}}}_{\text{findObject}} \right)^{(-1)} \quad \underbrace{{}^{\text{tcp}}T_{\text{grasp}}}_{180^\circ \text{ flip}} . $$

The following JavaScript example demonstrates the readjustment of the gripped part using the return of the *Find object* task:

```
// create the y-flip transformation
var yFlip = lab.getTF("world_frame", "world_frame") // get a random pose object
yFlip.x = 0
yFlip.y = 0
yFlip.z = 0
yFlip.roll = 0
yFlip.pitch = 3.14159  // 180 deg
yFlip.yaw = 0

// get match position
var matches = dataStore.getJsObject("Pose_im_Greifer") // PoseList as returned by␣
↪FindObject Task

if (matches.length != 1) {
    task.status = "Expected exactly 1 match, but got " + matches.length
}
var world_T_match = matches[0]

// transform to tcp coordinate frame
var tcp_T_world = lab.getTF("gripper_tcp", "world_frame")
var tcp_T_match = lab.mult(tcp_T_world, world_T_match)

// compare to model
var part_T_grasp = lab.getCurrentGraspOffset()
var grasp_T_part = lab.invertPose(part_T_grasp)
var tcp_T_part = lab.mult(yFlip, grasp_T_part) // respect the transformation between␣
↪grasp and tcp
var dist_trans = lab.getPoseDistanceTrans(tcp_T_part, tcp_T_match)
var dist_rot = lab.getPoseDistanceRot(tcp_T_part, tcp_T_match)
task.status = "Distance model -> match is: trans: " + Math.round(dist_trans *1000 *␣
↪100) / 100 + "mm, rot: " + Math.round(dist_rot* 180/3.14159 * 100) / 100 + "deg"
+ "\ntcp_T_match: " + "\nx: " + tcp_T_match.x*1000 + "\ny: " + tcp_T_match.y*1000  +
↪"\nz: " + tcp_T_match.z*1000
    + "\nr: " + tcp_T_match.roll  + "\np: " + tcp_T_match.pitch  + "\ny: " + tcp_T_
↪match.yaw


// set current Grasp Offset
var match_T_tcp = lab.invertPose(tcp_T_match)
var match_T_grasp = lab.mult(match_T_tcp, yFlip)

lab.setCurrentGraspOffset(match_T_grasp)
```

getCurrentGraspPieceToToolTransform

Calculates the transformation between the currently gripped part and tool0.

See also *getCurrentGraspOffset* for the transformation from the appended part to the TCP.

Use: `Pose = getCurrentGraspPieceToToolTransform()`

```
var pieceToToolTF = lab.getCurrentGraspPieceToToolTransform()
// Pose könnte per Feldbus-Schnittstelle an SPS übertragen werden
```

mult

Multiplies the transformation representation of a pose by the transformation representation of a pose or a 3D vector.

Use: `[Pose, Vector3] = mult(Pose, [Pose, Vector3])`

```
// Pose mit Vector multiplizieren
var pose = dataStore.getJsObject('einePose')
var vec = lab.mult(pose, [1,0,0])
console.log(vec)

// Pose mit Pose multiplizieren
var pose2 = dataStore.getJsObject('anderePose')
var pose_res = lab.mult(pose, pose2)
console.log(pose_res.x, pose_res.y, pose_res.z, pose_res.roll, pose_res.pitch, pose_
↪res.yaw)
```

The following illustration shows an example of the analysis of a found object. The coordinate system (red, green, blue corresponds to x, y, z) of the world and the part is shown. The following example code indicates whether the z-axis of the part and the x-axis of the world point in the same direction (in the example shown, both axes point in the same direction).

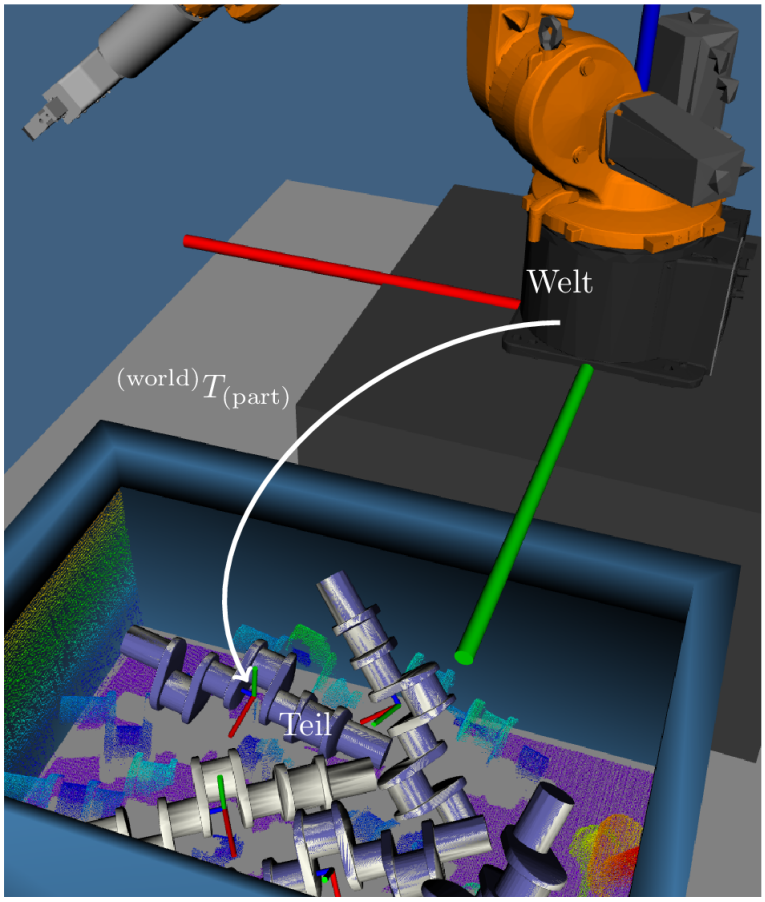


Fig. 7: Illustration of the example for analyzing a found part

Example: Examine parts pose

```
// Check ob die z-Achse des gefundenen Teils in positive x-Richtung der Welt zeigt

// Transformation von TCP zum Teil
// Annahme Teilepose 'currentMatch' wurde vorher mit dem Task "Finde Objekt" ermittelt
var pose_worldToPart = dataStore.getJsObject("currentMatch");
// Translation null setzen -> "verschiebt" Teil in Ursprung von Welt
pose_worldToPart.x = 0
pose_worldToPart.y = 0
pose_worldToPart.z = 0

// mit Einheitsvektor Z multiplizieren
var zDir = lab.mult(pose_tcpToPart, [0,0,1]) // (world)e_z = (world)T_(part) *␣
↪(part)e_z

// Ergebnis ist Z-Richtung des Teils beschrieben im Welt-KOS
var aligned = (zDir[0] > 0) // Vergleich mit der X-Achse der Welt

task.status = "z-Achse Teil und x-Achse Welt zeigen in die gleiche Richtung?: " +␣
↪aligned
```

clearAllRecognizedObjects

Deletes all objects found from the collision world. After calling this operator, a completely new search is started despite the "Reuse previous matches" flag being activated (see also *Halcon-specific object recognition parameters*).

Verwendung: `Bool = clearAllRecognizedObjects()`

```
var success = lab.clearAllRecognizedObjects()
```

updateOctomap

Calculates and publishes the octomap from the current point cloud. It can therefore be used for collision checking during placing, for example. The optional parameter `waitUntilUpdateReceived` specifies whether the call should block.

Use: `Bool = updateOctomap(Bool waitUntilUpdateReceived=True)`

```
var success = lab.updateOctomap()
```

setSceneObjectDimension

Sets the default size of a scene object. Scene objects can be *Obstacle*, *Region of Interest (ROI)* or *Box*.

Verwendung: `Bool = setSceneObjectDimension(String sceneObjectName, Double x, Double y, Double z)`

```
// Anpassen der Größe einer ROI
var pose = dataStore.getJsObject('ROI_Dimensions')
var success = lab.setSceneObjectDimension('ROI_Bildaufnahme', pose.x, pose.y, pose.z)
```

callExtensionService

Can be used to call a ROS Service Call of type `mikado_extension_msgs::RunExtensionBasic` and wait for the results. The service could be implemented in Python and can be used to implement project specific functionality.

Use: `dict = lab.callExtensionService(String name)`

```
result = lab.callExtensionService("/screw_check/check")

if (result.success)
    task.status = "Check erfolgreich"
else
    task.status = "Check gescheitert. Fehler: "+result.msg
```

You can find a sample template for implementing a project-specific ROS service in the chapter *Create custom ROS Service*.

setGetJsonFromCamera

This function can be used to write and/or read directly into the variable tree of the nxLib (nxTreeEdit) of an Ensenso camera. It can be used, for example, to read or change parameters or to execute actions of the nxLib directly via the `/command` node.

Equivalent to the procedure in the Halcon task *ensenso_set_get_json*.

Use: `String = setGetJsonFromCamera(String camera, String setJson, String getPath)`

```
// Setze die Eigenschaft "Test/abc" auf den Wert "123"
task.setGetJsonFromCamera("Ensenso_Camera_0", "{"Test":{"abc": "123"}}", "")

// Lese die Eigenschaft "Version" des NxTree
string version = task.setGetJsonFromCamera("Ensenso_Camera_0", "", "/Version")
```

8.1.12 pointCloud

The following functions are available:

- *getXyzMinMax*

getXyzMinMax

This function allows access to the properties of the point cloud. It provides its minimum and maximum X, Y and Z values as well as the number of points. If an ROI is active, the output refers to the selected ROI.

The function can be used, among other things, to check whether a crate has been completely emptied. Another use case would be to place parts back into the crate depending on the filling level by calculating a discharge height depending on the maximum height determined.

Returns an object with the following keys:

- `x_min` (`Double`): Minimum in X-direction of the product coordinate system [m]
- `x_max` (`Double`): Maximum in X-direction of the product coordinate system [m]
- `y_min` (`Double`): Minimum in Y-direction of the product coordinate system [m]
- `y_max` (`Double`): Maximum in Y-direction of the product coordinate system [m]
- `z_min` (`Double`): Minimum in Z-direction of the product coordinate system [m]
- `z_max` (`Double`): Maximum in Z-direction of the product coordinate system [m]
- `number_points` (`Integer`): Number of points of the point cloud (in the active ROI)

Use: `pointCloud = getXyzMinMax()`

```
// Berechne Eigenschaften der aktuellen (ROI-)Punktwolke
var pointCloudInfo = pointCloud.getXyzMinMax()

// Informationen über Punktwolke ausgeben
console.log(JSON.stringify(pointCloudInfo))

// Zähle Punkte in ROI zur Überprüfung ob sich Bauteil darin befindet
var punkteInROI = pointCloudInfo.number_points

if (punkteInROI < 100) {         // Durch Rauschen kann es vorkommen, dass sich␣
↪vereinzelt Punkte in der ROI befinden.
```

(continues on next page)

(continued from previous page)

```
    var ablageFrei = true         // Die Überprüfung "punkteInROI == 0" wäre somit␣
↪fehleranfällig.
} else {
    var ablageFrei = false
}

// Schreibe Wert in Mikado Variable "ablage_frei"
dataStore.setData("ablage_frei", ablageFrei)
console.log(task.status = "Ablage ist frei: " + ablageFrei)
```

The example above checks whether there are points in the active ROI. If yes, the position is considered occupied, if not, the position is free.

8.2 Halcon Script Task

The Halcon image processing library from MVTec Software GmbH makes it possible to develop your own image processing algorithms in a Halcon-specific script language. If user-defined Halcon scripts are to be added to an application, simple integration can be carried out using the *Run Halcon Script* task. To do this, you develop your own algorithms in the Halcon development environment, HDevelop. An extension provides the same access to the Mikado system as later in the task. To develop, you need a separate developer license for HDevelop, currently in version 13. If the program developed in this way works satisfactorily, it can now be integrated into a Mikado process via the task *Run Halcon Script*.

If you would like to implement your own algorithms for object recognition using Halcon, see section *User-defined object recognition with Halcon*.

The following HDevelop operators are provided by the Mikado extension for HDevelop:

8.2.1 mikado_get_2d_image

Read out the last capture of an Ensenso camera integrated in Mikado. Use tasks such as '*Load camera configuration*' and '*Record point cloud*' to execute the image capture and then use this Halcon operator to access the capture.

If a monocular camera is connected to Mikado, this command can be used to capture an image. Please note that the function call only triggers a new image at the same time for monocular cameras. Use the 'CameraTarget' parameter to transmit the serial number of the camera and thus request the raw or rectified image.

The following table describes the individual parameters of the operator.

Parameter	Parameter name	Description	Example
Issue	Image2D	The image is stored in this variable after the operator is called	OutputImage
Input	CameraName	Name of the camera	'Ensenso_Camera_0'
	CameraTarget	Right or left camera (Ensenso). Serial number for mono cameras.	'left', 'right', '4101230815'
	ImageType	Image type	'raw', 'rectified', 'withOverlay'

Hint: Examples for using the HDevelop operator:

- `mikado_get_2d_image(Image2D, 'Ensenso_Camera_0', 'left', 'raw')`
- `mikado_get_2d_image(Image2D, 'Ensenso_Fix', 'right', 'raw')`
- `mikado_get_2d_image(Image2D, 'Ensenso_Robot', 'left', 'withOverlay')`
- `mikado_get_2d_image(Image2D, 'Ensenso_Camera_0', '4101230815', 'raw')`

Warning: This operator does not trigger a capture. You must trigger the capture beforehand, e.g. with the task *Record point cloud*.

8.2.2 mikado_publish_2d_image

Publishes a 2D image for display in Mikado. The image, whether grayscale or RGB(A), can then be viewed in the Mikado interface in the right half of the live camera view by selecting it from the list.

The following table describes the individual parameters of the operator.

Parameter	Parameter name	Description	Example
Input	Image2D	Image to be displayed	Image2D

Hint: Examples for using the HDevelop operator:

- `mikado_publish_2d_image (Image2D)`

8.2.3 mikado_get_point_cloud

Read out the last point cloud of an Ensenso camera integrated in Mikado. Use tasks like '*Load camera configuration*' and '*Record point cloud*' to execute the image capture and then use this Halcon operator to access the capture. As Halcon does not deal well with NaN values, these are set to zero. These can be filtered with the additional output region.

The following table describes the individual parameters of the operator.

Parameter	Parameter name	Description	Example
Issue	ImageX	The X-component of the point cloud	OutputX
	ImageY	The Y-component of the point cloud	OutputY
	ImageZ	The Z-component of the point cloud	OutputZ
	Region	Valid points of the point cloud	ValidRegion
Input	CameraName	Name of the camera	'Ensenso_Camera_0'

Hint: Examples for using the HDevelop operator:

- `mikado_get_point_cloud (ImageX, ImageY, ImageZ, ValidRegion, 'Ensenso_Camera_0')`

Warning: This operator does not trigger a capture. You must trigger the capture beforehand, e.g. using the *Record point cloud* task.

8.2.4 mikado_write_variable

Set the value of a Mikado variable.

The following table describes the individual parameters of the operator.

Parameter	Parameter name	Description
Input	VariableName	Name of the variable to be changed
	VariableValueType	Type of variable
	VariableStringValue	Value of the variable if the variable type is String or StringList
	VariableNumericValue	Value of the variable if the variable type is an integer, bool or double

All Mikado variable types are listed in the following table. The assignment of which value is used for which variable type is listed in this table together with examples.

Variablentyp	Writing possible	Input via VariableStringValue / VariableNumericValue	Examples
Integer	✓	Numeric	1 42
Double	✓	Numeric	1.1 3.14
String	✓	String	'myValue'
StringList	✓	String	['StringListWert1', 'StringListWert2']
JointPosition	–		
Pose	✓	Numeric	Halcon pose, is converted internally into radians and meters
PoseList	✓	Numeric	Tuple from Halcon Posen
Bool	✓	Numeric	0 1
JScript	–		

Hint: Examples for using the HDevelop operator:

- `mikado_write_variable('MeineMikadoIntegerVariable', 'Integer', [], 42)`
- `mikado_write_variable('MeineMikadoStringVariable', 'String', 'test', [])`
- `mikado_write_variable('MeineMikadoStringListVariable', 'StringList', ['test1', 'test2', 'test3'], [])`

8.2.5 mikado_read_variable

Read out the value of a Mikado variable.

The following table describes the individual parameters of the operator.

Parameter	Parameter name	Description	Examples
Input	VariableName	Name of the Mikado variable to be read	'VarName'
Issue	VariableValueType	Type of variable read	String', 'Integer', ...
	VariableStringValue	Value of the variable as a string	'myValue', '42', ['StringListWert1', 'StringListWert2']

continues on next page

Table 7 – continued from previous page

Parameter	Parameter name	Description	Examples
	VariableNumericValue	Value of the variable if the Mikado variable had a numeric type	42, 3.14, 0

All Mikado variable types are listed in the following table. The assignment of which value is described for which variable type is listed in this table together with examples.

Variablentyp	Output via VariableStringValue / VariableNumericValue	Examples
Integer	Numeric	1 42
Double	Numeric	1.1 3.14
String	String	'myValue'
StringList	String	['StringListValue_1', 'StringListValue2']
JointPosition	String	'90.0 0.0 0.0 10.1 90.0 0.0'
Pose	String	'xyz: 0.00 0.10 0.21 rpy: 3.14 0.00 1.65'
PoseList	String	'xyz: 0.00 0.07 0.00 rpy: 0.00 0.00 1.54nxyz: 0.10 0.20 0.30 rpy: 3.14 0.00 0.00'
Bool	Numeric	0 1

Hint: Examples for using the HDevelop operator:

- `mikado_read_variable ('MeineMikadoVariable', VariableValueType, VariableStringValue, VariableNumericValue)`
- `mikado_read_variable ('ProduktName', VariableValueType, VariableStringValue, VariableNumericValue)`

8.2.6 ensenso_set_get_json

This procedure can be used to write and/or read directly into the variable tree of the nxLib of an Ensenso camera. It can be used, for example, to read or change parameters or to execute actions of the NxLib directly via the `/command` node. This procedure is equivalent to the function call in the JavaScript task *setGetJsonFromCamera*.

The following table describes the individual parameters of the operator.

Parameter	Parameter name	Description	Examples
Input	CameraName	Name of the camera	'Ensenso_Camera_0'
	SetJson	Json subtree to be written. Empty if necessary.	'{}', '{“Execute”:{“Aufnahme”: {“Command”: “Capture”}}}'

continues on next page

Table 9 – continued from previous page

Parameter	Parameter name	Description	Examples
	GetJsonPath	Path whose json is to be read	'/Version', '', '/Execute/Result'
Issue	SetSuccess	Was the writing/setting successful?	true', 'false'
	SetError	Error message when setting	'', 'CouldNotInterpretJsonText', ...
	GetError	Error message when reading	"", "ItemInexistent".

Hint: Examples for using the HDevelop operator:

- `ensenso_set_get_json('Ensenso_Camera_0', '', '/Version', SetSuccess, GetValue, SetError, GetError)`
- `ensenso_set_get_json('Ensenso_Camera_0', '{"Test":{"abc": "123"}}', '', SetSuccess, GetValue, SetError, GetError)`
- `ensenso_set_get_json('Ensenso_Camera_0', '{"Execute":{"Aufnahme": {"Command": "Capture"}}}', '/Execute/Aufnahme/Result/', SetSuccess, GetValue, SetError, GetError)`

Listing 1: Example: Reading out the camera parameters for N30, N35, N40, N45 and S10

```
* valid for N30, N35, N40, N45 and S10

* get size of sensor pixels in microns.
ensenso_set_get_json ('Ensenso_Camera_0', '', \
                      '/Cameras/205044/Sensor/PixelSize',\
                      _, PixelSize, SetError, GetError)
S := PixelSize // same for x and y


* get width / height of sensor
ensenso_set_get_json ('Ensenso_Camera_0', '', \
                      '/Cameras/205044/Sensor/Size',\
                      _, SensorSize, SetError, GetError)
sensorWidth := SensorSize[0]
sensorHeight := SensorSize[1]

* get rectified camera coordinate system
ensenso_set_get_json ('Ensenso_Camera_0', '', \
                      '/Cameras/205044/Calibration/Dynamic/Stereo/Left/Camera',\
                      _, CameraCOS, SetError, GetError)
Fx := CameraCOS[0][0]
Fy := CameraCOS[1][1]
F := Fx*S

gen_cam_par_area_scan_polynomial (F, 0, 0, 0, 0, 0, S, S,\
                                  sensorWidth/2, sensorHeight/2, sensorWidth,␣
↪sensorHeight,\
                                  CamParams)
```

8.3 User-defined object recognition with Halcon

Mikado offers an interface to Halcon (for information on Halcon, see *Halcon Script Task*) to integrate your own algorithms for object recognition. To integrate your scripts you must

- Create an .hdev script that contains the corresponding procedures
- Extend the object_recognition.yaml (information about configuration files in section *Configuration files*) to tell Mikado for which products which script should be called

8.3.1 Hdev script

The first prerequisite for your own object recognition is a suitable Hdev script that can be integrated into Mikado. We recommend installing HDevelop directly on the Mikado controller so that you can test directly on the controller and use the Mikado extension.

Define a new program with the following procedures using the development environment for Halcon:

- `load_model`
  - Input control parameters (product properties)
  - *ModelName*: Product name of the product to be loaded.
  - *MeshFilename*: File path of the CAD file of the product.
  - *RecognitionMeshFilename* (optional): File path of the CAD file of the product to be used for object recognition. Value is empty if not specified.
  - *ParamNames* (optional): List of parameter names that are specified in object_recognition.yaml for this product.
  - *ParamValues* (optional): List of parameter values that are specified in object_recognition.yaml for this product.
  - Minimal example:

```
global tuple SurfaceModelID

if (RecognitionMeshFilename == '')
    RecognitionMeshFilename := MeshFilename
endif

read_object_model_3d(RecognitionMeshFilename, 1.0, ParamNames, ParamValues,␣
↪ObjModel3D, Status)

create_surface_model(ObjModel3D, 0.03, [], [], SurfaceModelID)

return ()
```

Hint: Using `global tuple`, tuples can be declared that are also available in other procedures such as the `find_objects` procedure.

- `find_objects`
  - Input control parameters (product properties)
  - *ModelNames*: Product name of the product to be recognized.
  - *MinimumScores*: Minimum score of the product.
  - *MaxTotalMatches*: Expected number of finds (sum of all product finds).

- Output control parameters (return values to Mikado)
- *RecognizedModels*: Product names of the finds. If this list remains empty, the first product is used by default for all find poses.
- *Poses*: Recognized object poses
- *Scores*: Score values of the individual objects
- Minimal example:

  In the following example, the function *mikado_get_point_cloud* is used to retrieve the point cloud of the last recording and use it for the object search.

```
* Simple Example: assumes that only one model is searched; ignore ModelNames␣
↪and just use first MinScore

mikado_get_point_cloud(ImageX, ImageY, ImageZ, Region, 'Ensenso_Camera_0')

reduce_domain (ImageX, Region, ImageX)
reduce_domain (ImageY, Region, ImageY)
reduce_domain (ImageZ, Region, ImageZ)

xyz_to_object_model_3d(ImageX, ImageY, ImageZ, ObjectModel3D)

global tuple SurfaceModelID

find_surface_model (SurfaceModelID, ObjectModel3D, 0.05, 0.2,␣
↪MinimalScores[0], 'false', ['num_matches'], [MaxTotalMatches], Poses,␣
↪Scores, SurfaceMatchingResultID)

return ()
```

8.3.2 Customizing the configuration file

The object_recognition.yaml configuration file contains the parameterization for object recognition. You can also integrate your user-defined Hdev script at this point.

1. Exit the Mikado interface.
2. Open the 'object_recognition.yaml' file in the 'config' directory
3. Search for the section `models:` and create entries for the products with their own object recognition in the following form:

```
[...]

models:
    MyProductName1:
        use_halcon_script: true
        script_filename: "/home/mikado/cells/my_mikado_cell/scripts/
↪MyHalconObjectRecognition1.hdev"

        # Params are passed through to halcon function
        params:
            my_param_1: false
            my_param_2: 'text_parameter'
    MyProductName2:
        use_halcon_script: true
        script_filename: "/home/mikado/cells/my_mikado_cell/scripts/
↪MyHalconObjectRecognition2.hdev"
```

4. Start Mikado.

Hint: You can check whether your configuration is correct by loading your product in the Mikado interface and then looking at the logs. There you will find an info entry: "[HDevEngine node] Request received to load model for object recognition via halcon script: [..]".

8.4 Own pages with QML

The user interface of Mikado is programmed in QML and can easily be expanded to include two customer-specific pages. For example, a system-specific status is displayed to the operator on the first page. On another page, more detailed information can be added or parameters can be set. If activated, this second page is password-protected.

8.4.1 Konfiguration

In the folder `qml` you will find the file `pages.yaml` in which you can configure which two files are to be displayed as pages. After restarting Mikado, these will now appear between *Welcome* and *Status* in *Run* mode. You can edit these files with any text editor; *kate* is pre-installed on the controller. If you set the parameter `debug_qml: true` in the `config/mikado.yaml` file, another page appears on the right-hand side of the screen where you can reload your code live. This allows you to make quick and easy adjustments to the pages, ideally on a second monitor.

8.4.2 Introduction to QML

QML is a declarative language very similar to HTML, and JavaScript can also be used for dynamic aspects. Let's start with a simple example:

```
import QtQuick 2.12

Item {

    // Example: Header of user page
    Rectangle {
        id: header
        y: 20
        width: 700
        height: 50
        color: "blue"
        radius: 20
        anchors.horizontalCenter: parent.horizontalCenter

        Text {
            anchors.centerIn: parent
            text: "MIKADO - User Page"
            verticalAlignment: Text.AlignVCenter
            horizontalAlignment: Text.AlignHCenter
            font.bold: true
            font.pixelSize: 32
            color: "white"
        }
    }
}
```

The basic package for QtQuick is loaded here first. It provides some basics such as rectangles (`Rectangle`), text displays (`Text`) and images (`Image`). The object hierarchy then begins with a minimum element (`Item`), which is

automatically adapted to the window in the Mikado interface and therefore does not require a size specification. This contains a blue rectangle with rounded corners (`radius: 20`). Interesting here is the relative positioning in the middle of the page via an `anchor`. This rectangle again contains a white text.

It gets a little more exciting when we add a button and trigger a sequence when clicked:

```
import QtQuick 2.12
import QtQuick.Controls 2.12

//...

Button {
   text: "Starte Test-Sequenz"
   onClicked: startSequence("Test")
}

function startSequence(name) {
   if (core.selectSequence(name)){
     core.run();
     return true
   } else {
     print("Sequenz existiert nicht")
     return false
   }
}
```

Here, the JavaScript function `startSequence` is executed after a click (`onClicked`). This accesses Mikado via the `core` object and attempts to switch the program to the sequence named `Test`. If this does not exist, a message is output to the console (and the logs). Otherwise the Mikado program sequence is activated.

Some other predefined elements can be found in the following excerpts.

Display Mikado variable:

```
VariableLabel {
  variable: "TeilZaehler"
}
```

Customize Mikado variable in text format:

```
VariableTextField {
  id: varTextField
  variable: "Zeit"
  setAsInitial: true
}
```

Otherwise, all other blocks from e.g. Qt Quick Controls 2 can also be used.

8.4.3 Further information

The QML book 'Qt5 Cadaques' at https://qmlbook.github.io and the official QML documentation at https://doc.qt.io/qt-5/qmlreference.html provide a very comprehensive and comprehensible introduction to QML. We use the Qt Quick Controls 2 whose documentation can be found at https://doc.qt.io/qt-5/qtquickcontrols-index.html.

8.5 External control with ROS/Python

Both for external control by a PLC or a control system and for the integration of special hardware components such as individual grippers, swivel units, ... Mikado offers an ROS interface. This API can be accessed with different languages, in the following Python is used as it generally results in compact and understandable source code. If you prefer to work with C, C++, C# or node.js, please ask us directly for instructions and examples.

8.5.1 Example program

Let's start with a simple example program:

```
1  from mikado_msgs.srv import *
2  import rospy
3
4  # Variable aus Mikado lesen
5  get_var = rospy.ServiceProxy('read_variable', ReadVariable)
6  max_ansicht = get_var('MaxAnsicht')
7  print "MaxAnsicht = " + max_ansicht.value_ints[0]
8
9  # Mikado Sequenz starten
10 run_sequence = rospy.ServiceProxy('run_sequence', RunSequence)
11 run_sequence(name='Init', check_existence_only=False,
12                               stop_after_first=False, blocking=False)
13 # oder
14 run_sequence('Init', False, False, False)
```

In this example, the API for communication with Mikado Control is loaded first (1). An example then explains how a variable is read from Mikado via a *ServiceCall*, i.e. a function call. First a function object is defined (5) and then its function is called (6). The return value is stored in `max_view`. As variables in Mikado can have different types, this is in the format of a collection object. The `value_ints` property can now be used to access an array of all integer return values, for example.

In the second section, we start a Mikado sequence called `Init` (a name assigned in the Mikado program). Again, a function object is created (10) and then called. Either the detailed variant (11) with named arguments or the compact variant (14) can be used.

8.5.2 Mikado ROS Services

With a running Mikado, the ROS service interface can be conveniently inspected and tested on the command line. `rosservice list` displays the available services, `rosservice info /run` for example displays the information for the function to start the Mikado process and with `rosservice call /run ...` it can be called directly. Press the Tab key twice to complete the list of arguments. The most important commands are listed below:

Mikado

`mikado_msgs.`**`pause`**`()`

Pauses the currently running Mikado program after the currently running task has ended. The function is asynchronous, i.e. the robot can still move after the function has ended, for example. To ensure that Mikado has been paused, please use the `/engine_state` topic.

`mikado_msgs.`**`read_variable`**`()`

Reads a Mikado variable and returns its value. Call in code: rospy.ServiceProxy('read_variable', ReadVariable)

Parameters

name (*string*) – Name of the variable

Returns

Wert der Variable success (bool): Variable existiert value_type (string): Typ der Variable (`Integer`, `Double`, `String`, `StringList`, `Pose`, `PoseList`, `JointPosition`, `Bool`, `JScript`) value_string (string): Textdarstellung der Variable value_strings (string[]): value_ints (int[]): value_floats (double[]): value_poses (geometry_msgs::Pose[]):

`mikado_msgs.`**`run`**`()`

Aufruf in Code: rospy.ServiceProxy('run', Run)

Parameters

`step` (*`bool`*) – Pause after a step (single step) Starts execution at the beginning of the Mikado program or at the current position of the program pointer. The `step` argument specifies whether only the next task is to be executed or whether automatic mode is to be started.

`mikado_msgs.`**`run_sequence`**`()`

Aufruf in Code: rospy.ServiceProxy('run_sequence', RunSequence)

Parameters

- **`name`** (*`string`*) – Name of the sequence to be executed
- **`check_existence_only`** (*`bool`*) – the function ends with `success = true` if the sequence exists
- **`stop_after_first`** (*`bool`*) – Executes only the first task in the desired sequence, can then be executed in individual steps with `run(step=true)`

Starts the program execution with a desired sequence `name`.

`mikado_msgs.`**`stop`**`()`

Aufruf in Code: rospy.ServiceProxy('stop', Stop)

Terminates the program execution, resets all variables to their initial value and removes dynamic data. This includes point clouds, object finds, handles, positions of scene objects … Call in code: rospy.ServiceProxy('stop', Stop)

`mikado_msgs.`**`write_variable`**`()`

Aufruf in Code: rospy.ServiceProxy('write_variable', WriteVariable)

Parameters

- **`name`** (*`string`*) –
- **`value_type`** (*`string`*) – `Integer`, `Double`, `String`, `StringList`, `Pose`, `PoseList`, `JointPosition`, `Bool`, `JScript`
- **`value_strings`** (*`string[]`*) –
- **`value_ints`** (*`int[]`*) –
- **`value_floats`** (*`float[]`*) –
- **`value_poses`** (*`Pose[]`*) – Sets the value of a Mikado variable that must have the specified `value_type`. The value must be in the corresponding fields. `JointPosition` is read from `value_floats`, `JScript` (JavaScript) from `value_strings`.

Ensenso

isys_object_recognition_msgs.**calibrate**()

/cameras/Ensenso_Camera_0/calibrate

Parameters

- **store** (*bool*) – Saving the calibration in the camera
- **calibration_setup** (*string*) – Moving or Fixed, is the camera moving (attached to the robot) or in a fixed position in the cell

Performs the camera calibration with the images captured with capture_for_calibration.

isys_object_recognition_msgs.**calibrate_workspace**()

/cameras/Ensenso_Camera_0/calibrate_workspace

Parameters
defined_pose_transform (*Pose*) –

Sets the coordinate system of the camera based on a calibration plate so that the pattern is located at the defined_pose_transform pose. (Only useful for Mikado Select, replaces an existing hand-eye calibration)

isys_object_recognition_msgs.**capture_for_calibration**()

/cameras/Ensenso_Camera_0/capture_for_calibration

Parameters

- **reset** (*bool*) – Discard previous recordings and start from the beginning
- **tool0_pose** (*Pose*) – Position of the robot flange (tool0)

Makes a recording and saves the positions of the pattern and robot for subsequent calibration with calibrate.

isys_object_recognition_msgs.**estimate_pattern_pose**()

/cameras/Ensense_Camera_0/estimate_pattern_pose

Returns
success (bool): a pattern was visible and its position could be determined pattern_pose (geometry_msgs.Pose):

Captures a camera image, searches it for an Ensenso calibration pattern and returns its position as pattern_pose.

isys_object_recognition_msgs.**get_captured_images**()

/cameras/Ensenso_Camera_0/get_captured_images

Parameters

- **left_image_type** (*int*) –
- **right_image_type** (*int*) –

Returns
success (bool) left_image, right_image (sensor_msgs.Image)

Returns an image from the last capture, type can be 0 (NO_IMAGE), 1 (RAW_IMAGE), 2 (RECTIFIED_IMAGE) or 3 (OVERLAY_IMAGE) and refers to the analog stages of Ensenso image processing.

isys_object_recognition_msgs.**get_point_cloud**()

/cameras/Ensenso_Camera_0/get_point_cloud

Returns
point_clouds (sensor_msgs.PointCloud2) List of point clouds

Returns the last recorded point cloud. In the case of multiple recordings, these are returned as individual point clouds.

isys_object_recognition_msgs.**is_calibration_pattern_visible**()

/cameras/Ensenso_Camera_0/is_calibration_pattern_visible

Parameters

- **calibration_setup** (*string*) – Moving or Fixed, is the camera moving (attached to the robot) or in a fixed position in the cell
- **tool0_pose** (*Pose*) – Pose of the robot flange

Checks whether the pattern is in the camera's field of view for the given pose. This requires at least a pre-calibration using run_precalibration or a complete calibration.

isys_object_recognition_msgs.**load_camera_config**()

/cameras/Ensenso_Camera_0/load_camera_config

Parameters

- **filename** (*string*) –
- **load_capture_parameters** (*bool*) –
- **load_link** (*bool*) –

Loads a camera configuration saved in NxView in JSON format into the current camera. Only the capture parameters (load_capture_parameters), only the transformation/link (load_link) or both can be loaded.

isys_object_recognition_msgs.**recalibrate**()

/cameras/Ensenso_Camera_0/recalibrate

Performs an estimation and recalibration of the vergence angle. A calibration pattern must be visible for this.

isys_object_recognition_msgs.**run_precalibration**()

/cameras/Ensenso_Camera_0/run_precalibration

Parameters

calibration_setup (*string*) – Moving or Fixed, is the camera moving (attached to the robot) or in a fixed position in the cell

Performs an initial rough hand-eye calibration. After this, is_calibration_pattern_visible can be used to check the visibility of random calibration positions, for example.

isys_object_recognition_msgs.**save_camera_config**()

/cameras/Ensenso_Camera_0/save_camera_config

Parameters

filename (*string*) –

Writes the current camera configuration to a JSON file.

isys_object_recognition_msgs.**save_raw_data**()

/cameras/Ensenso_Camera_0/save_raw_data

Parameters

- **path** (*string*) – Target folder, relative to the Home directory (personal folder in Dolphin)
- **prefix** (*string*) – The file name is preceded by a prefix
- **rectified** (*bool*) – If the image is to be saved in a rectified version, set this parameter to true

Saves the current recording, the resulting set of images can be used for a Filecam camera, for example.

isys_object_recognition_msgs.**set_capture_mode**()

/cameras/Ensenso_Camera_0/set_capture_mode

Parameters

- **front_light** (*bool*) –
- **projector** (*bool*) –

For 2D recordings with the Ensenso camera, the projector can be switched off and the frontlight switched on if necessary. Must be reset for 3D recordings.

isys_object_recognition_msgs.**set_get_json**()

/cameras/Ensenso_Camera_0/set_get_json

Parameters

- **set_json** (*string*) –
- **wait_for_null_json** (*string*) –
- **get_json** (*string*) –

Returns

bool set_success: Write successful string get_result: string set_error: string get_error:

This function makes it easy to interact with the NxLib. Parameters can be written with `set_json`, but commands can also be started. To do this, you write an `execute` node and specify the path to its `command` node in `wait_for_null_json`. When the command has been executed, this node is then set to `null` by the NxLib. Using `get_json`, either a parameter can be read or the `Result` node of a command can be queried.

isys_object_recognition_msgs.**set_roi_rotated_box**()

/cameras/Ensenso_Camera_0/set_roi_rotated_box

Parameters

- **box_pose** (*geometry_msgs.Pose*) –
- **box_size** (*geometry_msgs.Vector3*) –

Sets a region of interest (ROI) to which the point cloud is clipped.

isys_object_recognition_msgs.**start_capture**()

/cameras/Ensenso_Camera_0/start_capture

Parameters

left_camera (*bool*) – Should the left or right camera be used

Returns

success: true

Starts a continuous recording (2D), the left or right 2D raw image is published to the topic `/cameras/Ensenso_Camera_0/camera_image` and displayed in Mikado on the camera page.

isys_object_recognition_msgs.**stop_capture**()

/cameras/Ensenso_Camera_0/stop_capture

Returns

success: true

Ends the continuous image recording run

Robots/Hardware

extra_industrial_msgs.**control_hand**(*bool open*)

Aufruf in Code: rospy.ServiceProxy('control_hand', ControlHand)

Parameters
open (*bool*) –

Switches the gripper.

extra_industrial_msgs.**read_input_signal**(*int signal_number*)

Parameters
signal_number (*int*) –

Returns
returnValue: bool success: bool

Code example
read_input_signal = rospy.ServiceProxy('read_input_signal', ReadSignal)

res = read_input_signal(signal_number=23)

if res.success:

print res.returnValue

Reads the value of a digital input.

extra_industrial_msgs.**read_output_signal**(*int signal_number*)

Aufruf in Code: rospy.ServiceProxy('run_output_signal', ReadSignal)

Parameters
signal_number (*int*) –

Returns
returnValue (bool): success (bool):

Code example
read_output_signal = rospy.ServiceProxy('read_output_signal', ReadSignal)

res = read_output_signal(signal_number=23)

if res.success:

print res.returnValue

Reads the value of a digital output.

extra_industrial_msgs.**set_output_signal**(*int signal_number*, *bool value*)

Aufruf in Code: rospy.ServiceProxy('set_output_signal', SetSignal)

Parameters

- **signal_number** (*int*) –
- **value** (*bool*) –

Returns
success (bool):

Sets a digital output.

extra_industrial_msgs.**run_program**()

Parameters

- **program_name** (*string*) –
- **int_args** (*named_int[]*) – Ein Array von Kombinationen aus [string variable_name, int int_value]

- **float_args** (*named_float[]*) – Ein Array von Kombinationen aus [string variable_name, float float_value]
- **string_args** (*named_string[]*) – Ein Array von Kombinationen aus [string variable_name, string string_value]
- **pose_args** (*named_pose[]*) – Ein Array von Kombinationen aus [string variable_name, pose pose_value]

Returns
returnValue (int): A freely assignable return value success (bool): If the execution was successful

Executes a classic robot program that is stored on the robot controller.

industrial_msgs.**industrial_io_read**()

Aufruf in Code: rospy.ServiceProxy('industrial_io_read', IORead)

Parameters
items (*industrial_msgs.IOReadRequestItem[]*) – Ein Array von [int type, int index]

Returns
items (industrial_msgs.IOReadReplyItem[]): Ein Array von [int type, int index, int result, int value]

Allows several inputs or outputs to be read at the same time, this is particularly interesting for recording entire data blocks on a fieldbus. `type` specifies the type of field, possible values are

DigitalIn = 1 (read one input bit)

DigitalOut = 2 (read one output bit)

AnalogueIn = 3

AnalogueOut = 4

GroupedIn = 5 (One input word, 16 bit, read)

GroupedOut = 6 (One output word, 16 bit, read)

Not every end device supports all types; for fieldbuses (Hilscher), for example, only the digital inputs/outputs and word-wise read/write are available.

industrial_msgs.**industrial_io_write**()

Parameters
items (*industrial_msgs.IOWriteRequestItem[]*) – Ein Array von [int type, int index, int value]

Returns
items (industrial_msgs.IOWriteReplyItem[]): Ein Array von [int type, int index, int result]

Writes several outputs or fields in one pass, particularly interesting for fieldbus interfaces. Sets one or more fields with the type `type`, the address `index` to the value `value`. The return value contains the information for each field as to whether it could be written (result = 1) or not (result = 0). Which `type` belongs to input or output can be found in the list under industrial_io_read.

8.5.3 Mikado ROS Actions

In addition to the ROS services, ROS actions also exist as an external interface. These can be used via action clients and allow asynchronous commands and feedback. For detailed information on the use of ROS actions, please refer to the official ROS documentation.

The ROS actions offered by Mikado are listed below.

`mikado_msgs.calibration.`**`cancel`**`()`

Cancel hand-eye calibration

Returns
error_code

EC_SUCCESS	0
EC_ERROR	1
EC_ACTION_RUNNING	2
EC_DISABLED	3

`mikado_msgs.calibration.`**`capture`**`()`

Trigger recording for hand-eye calibration

Parameters
`current_pose` (*`geometry_msgs.Pose`*) – Current robot pose

Returns
error_code

EC_SUCCESS	0
EC_ERROR	1
EC_ACTION_RUNNING	2
EC_DISABLED	3
EC_NO_CALIBRATION_STARTED	4

`mikado_msgs.calibration.`**`finish`**`()`

Calculate hand-eye calibration and save it on the camera if successful.

In addition to the error number, the return values contain values for the calibration result (average deviations).

Returns
error_code

EC_SUCCESS	0
EC_ERROR	1
EC_ACTION_RUNNING	2
EC_DISABLED	3
EC_NO_CALIBRATION_STARTED	4

Returns
meanDist [mm]

Returns
meanX [mm]

Returns
meanY [mm]

Returns
meanZ [mm]

Returns
meanAngle [radian]

Returns
maxDist [mm]

mikado_msgs.calibration.**set_origin**()

Setting the origin using a calibration plate

Parameters

- **pose_of_measured_object** (*geometry_msgs.Pose*) – Pose of the calibration plate at the time of recording relative to the desired origin
- **camera_name** (*string*) – Camera name (e.g. "Ensenso_Camera_0")
- **frontlight** (*bool*) – Activate the front light of the camera for recording (if available)

Returns
error_code

EC_SUCCESS	0
EC_ERROR	1
EC_ACTION_RUNNING	2
EC_DISABLED	3
EC_CALIBRATION_IN_PROGRESS	4
EC_UNKNOWN_CAMERA_NAME	5

mikado_msgs.calibration.**start**()

Initialize calibration process (manual mode) or start automatic (automatic mode).

Parameters
calibration_mode –

CALIB_MODE_MANUAL	0
CALIB_MODE_AUTO	1

Parameters

- **pose_list_name** (*string*) – (optional) Variable name of the PoseList to be used as the list of calibration poses
- **camera_name** (*string*) – Camera name (e.g. "Ensenso_Camera_0")
- **frontlight** (*bool*) – Activate the front light of the camera for recording (if available)

Returns
error_code

EC_SUCCESS	0
EC_ERROR	1
EC_ACTION_RUNNING	2
EC_DISABLED	3
EC_CALIBRATION_IN_PROGRESS	4
EC_UNKNOWN_CALIB_MODE	5
EC_UNKNOWN_CAMERA_NAME	6

8.5.4 Systemstatus

A script can contribute its own information to the system status in Mikado, e.g. an error status leads to a query before the program is executed and is also displayed in the list on the left-hand side of the status bar. To do this, an entry must be added to `config/status.yaml`:

```
status:
    - name: Robot
      type: RobotStatus
      topic: robot_status
    - name: Profinet
      type: RosDiagnosticStatus
      topic: profinet/status
      enabled_if_rosparam: /profinet
    - name: Beispiel Status
      type: RosDiagnosticStatus
      topic: sps_status
```

A message must now be published on the newly added topic at least every 10s. Here is a minimal example

```
#!/usr/bin/env python
import rospy
from diagnostic_msgs.msg import DiagnosticArray, DiagnosticStatus

pub = rospy.Publisher('sps_status', DiagnosticArray, queue_size=10)
rospy.init_node('sps_script', anonymous=True)
rate = rospy.Rate(10) # 10hz
while not rospy.is_shutdown():
    diag_array = DiagnosticArray()
    status = DiagnosticStatus()
    status.name = "SPS"
    status.level = DiagnosticStatus.OK
    status.message = "Es laueft bestens"
    diag_array.status = [status]
    pub.publish(diag_array)
    rate.sleep()
```

In return, the script can also read a summary of the current system status on the `/system_status` topic.

8.5.5 Create custom ROS Service

The following code example can be used as a basis for creating your own ROS node with any project-specific functionality. The node provides a ROS service of type `mikado_extension_msgs::RunExtensionBasic`, which in turn can be called within the Mikado program using, for example, the `lab.callExtensionService()` function (see chapter *callExtensionService*)

```
#!/usr/bin/env python3
import rospy
from mikado_extension_msgs.srv import RunExtensionBasic, RunExtensionBasicResponse
from mikado_python_interface.mikado_com import read_from_mikado_throwing, write_to_
↪mikado_throwing
from mikado_python_interface.constants import NoError

NODE_NAME = "custom_node"
```

(continues on next page)

(continued from previous page)

```
SERVICE_NAME = "berechne_produkt_id_plus_100"
MIKADO_BAUTEIL_ID_VARIABLE = "ProfinetInput_ProductID"
MIKADO_ERGEBNIS_VARIABLE = "i_ergebnisBerechnung"

def custom_calculation():

    success = True
    # Lese Mikadovariable
    bauteil_id = read_from_mikado_throwing((MIKADO_BAUTEIL_ID_VARIABLE, "Integer"))

    # Führe Berechnung durch
    ergebis = int(bauteil_id) + 100

    # Schreibe Ergebnis
    write_to_mikado_throwing((MIKADO_ERGEBNIS_VARIABLE, "Integer"), ergebis)


def no_crash_wrapper_custom_calculation(_) -> RunExtensionBasicResponse:

    success = True
    msg = "Custom Message: OK"
    try:
        rospy.loginfo("Recieved service request")
        custom_calculation()
    except Exception as e:
        msg = f"Service Call failed - {e}."
        rospy.logerr(msg)
        success = False
    finally:
        retval = RunExtensionBasicResponse()
        retval.msg = msg
        retval.success = success
        retval.can_retry = False
        return retval

def main():
    try:
        rospy.init_node(NODE_NAME)
        rospy.wait_for_service('read_variable')
        rospy.Service(f"/{NODE_NAME}/{SERVICE_NAME}", RunExtensionBasic, no_crash_
↪wrapper_custom_calculation)
        rospy.spin()
    except rospy.ROSInterruptException as e:
        print(e)


if __name__ == "__main__":
    main()
```

Explanation: The node in the example is named “custom_node” and provides a ROS service `/custom_node/calculate_product_id_plus_100`. When this service is called, the logic of the function `custom_calculation()` is invoked. In the example shown, this function reads the Mikado variable “ProfinetInput_ProductID”, adds 100, and writes the result to the Mikado variable “i_ergebnisBerechnung”.

Function call (example)

```
lab.callExtensionService("/custom_node/berechne_produkt_id_plus_100")
```

To implement your own logic, you can add your own code to the `custom_calculation(...)` function. To start the node automatically with the cell, follow the instructions in the chapter *Launchfile*.

8.5.6 Launchfile

To start and end a script automatically with Mikado, save it in the `Scripts` folder of the cell (create the folder if necessary). Now open the file `launch/main.launch` and add the following text block:

```
<node name="custom_script" pkg="meine_zelle" type="Custom.py" output="screen" />
```

Here, our script with the file name `ProfinetMikadoAdapter.py` is started from the cell `my_cell` under the freely selectable name[1] `adapter_script`.

Hint: For the node to start, the file must be executable. To make a file executable, open the console in the `Scripts` subfolder (right-click -> Open Terminal) and enter the following command:

```
chmod +x ./Custom.py
```

---

[1] lower case letters, numbers and underscores are permitted. However, the name is only used to identify the program if necessary.

CHAPTER

NINE

UPDATES

This chapter describes changes in Mikado versions in detail. A complete and always up to date of all components of the Mikado system can be found on our website at https://optonic.com/support.

9.1 Mikado

Below you can find all changes for all Mikado versions.

9.1.1 Version 3.4.0

Mikado version 3.4.0 extends the pick search, adds new programming options, and improves usability when starting, backing up, and setting up programs.

Extended Pick Search

The *Get Picks*, *Pick object*, and *Pick-Search-Statistics* tasks provide additional output variables. This allows programs to directly access information such as the number of recognized objects, the number of pickable objects, object pose, pick, gripper, and collision type.

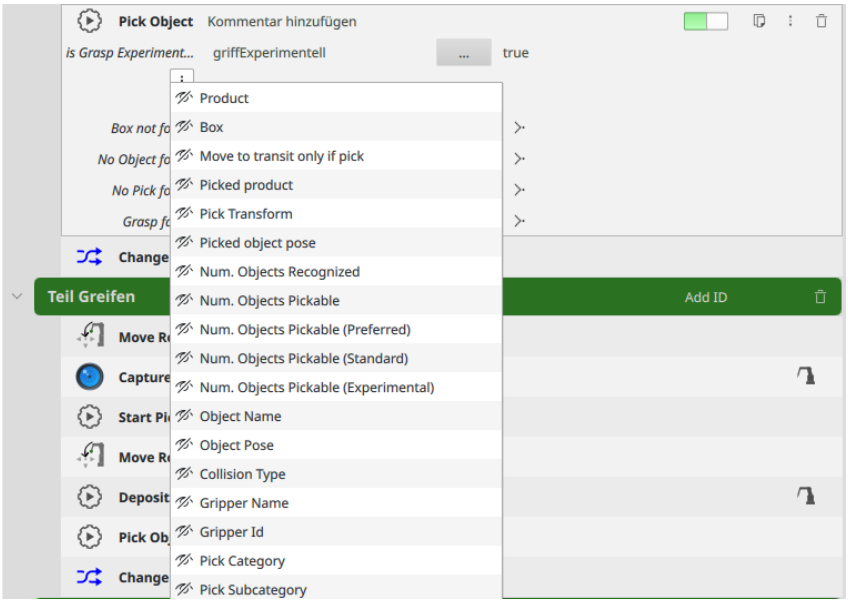


Fig. 1: Additional output variables in the `Pick Object` task.

A fixed Z height relative to the box can now be specified for the pick search. The Cartesian movements for picking and extraction are planned up to this height, making path planning easier to control for rods, tight boxes, or higher edges.

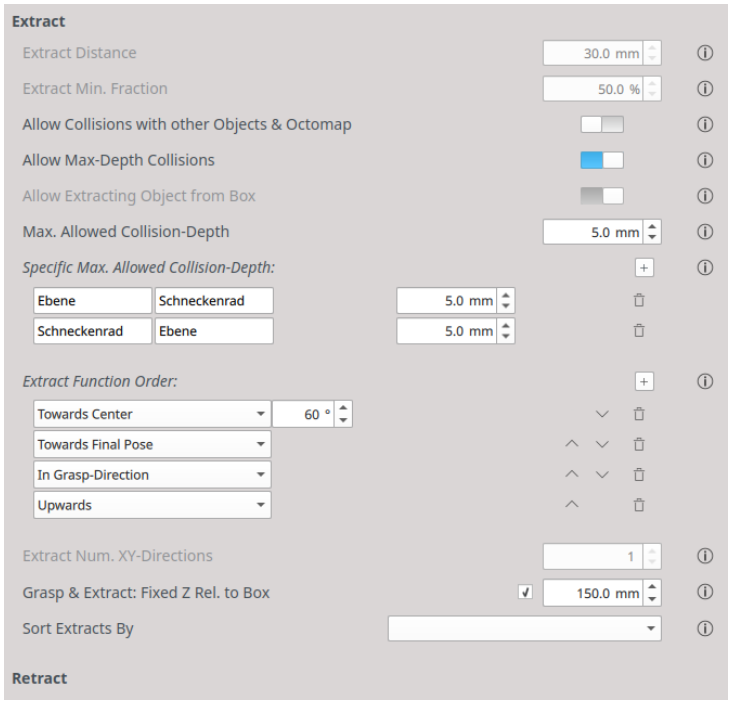


Fig. 2: Setting a fixed Z height relative to the box for picking and extraction paths.

Pick trajectories can also be smoothed. This creates smoother transitions between approach, pick, extraction, and retract.

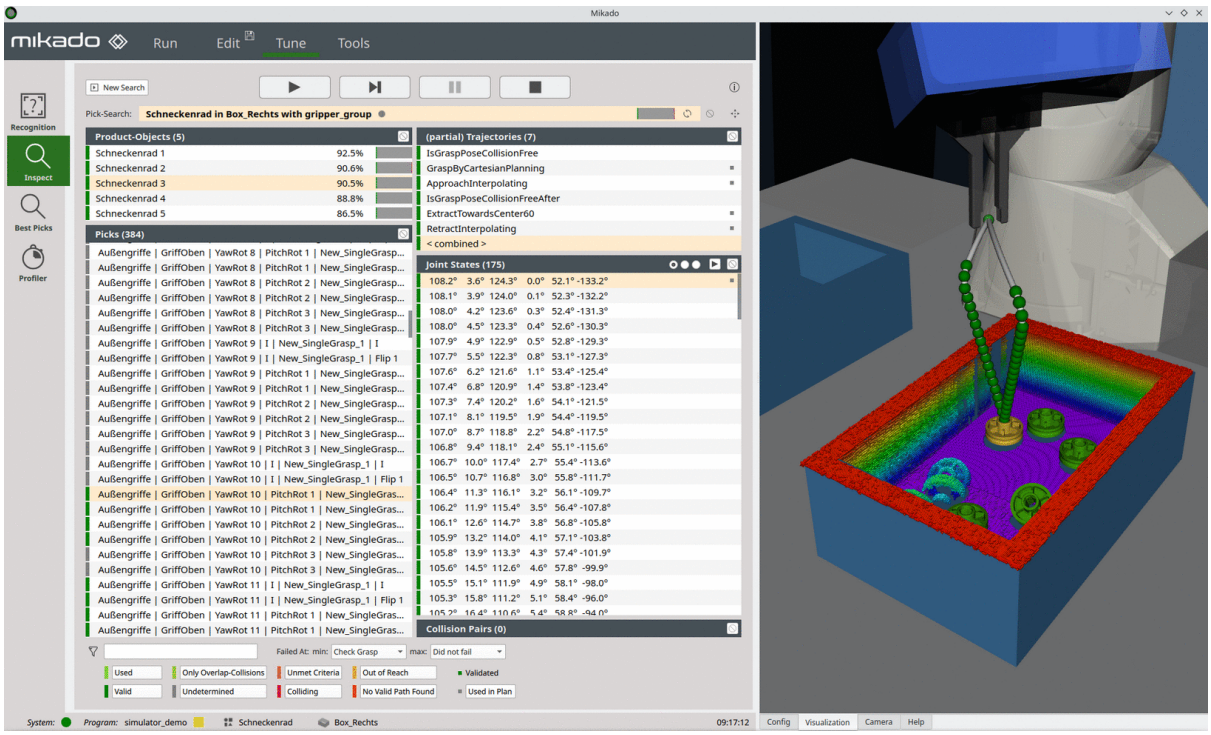


Fig. 3: Smoothed pick trajectory in the pick inspection.

New Tasks and Interfaces

- **Robot movements:** New tasks enable relative movements of the robot, TCP, or picked part.
- **Robot signals:** The new *Wait for robot output signal* task waits for robot output signals.
- **JavaScript:** The pose of the picked object can be queried via *picksearch.getPickedObjectPose()*.
- **FANUC:** The FANUC-specific page can show the robot web interface and open it in the browser.

Improved Usability

- **Welcome page:** Startup files for programs, products, and scene objects can be selected directly.
- **System page:** The current cell can be backed up as a ZIP archive.
- **Program settings:** Language selection, window size, window state, and full-screen mode are saved on the respective controller after changes in Mikado. This makes it easy to use your own language during offline commissioning or troubleshooting, for example, without having to change the customer's cell.
- **Task Editor:** Optional variables without a value are initially hidden and can be added through a menu when needed.

9.1.2 Version 3.3.0

Mikado version 3.3.0 introduces various new tasks, a new fieldbus page, new error reporting functionalities, and includes numerous usability and stability improvements.

Greater flexibility for vacuum grippers

Starting with Mikado 3.3.0, an area can be defined for vacuum grippers that is not considered a collision object during grasp planning. As shown in the example below, a plane is defined relative to the TCP (with a definable Z offset). Among other things, this functionality allows flexible suction cups of vacuum grippers to be excluded from collision checks during grasp planning. The new function can be set on the gripper page of the Mikado Assistant.

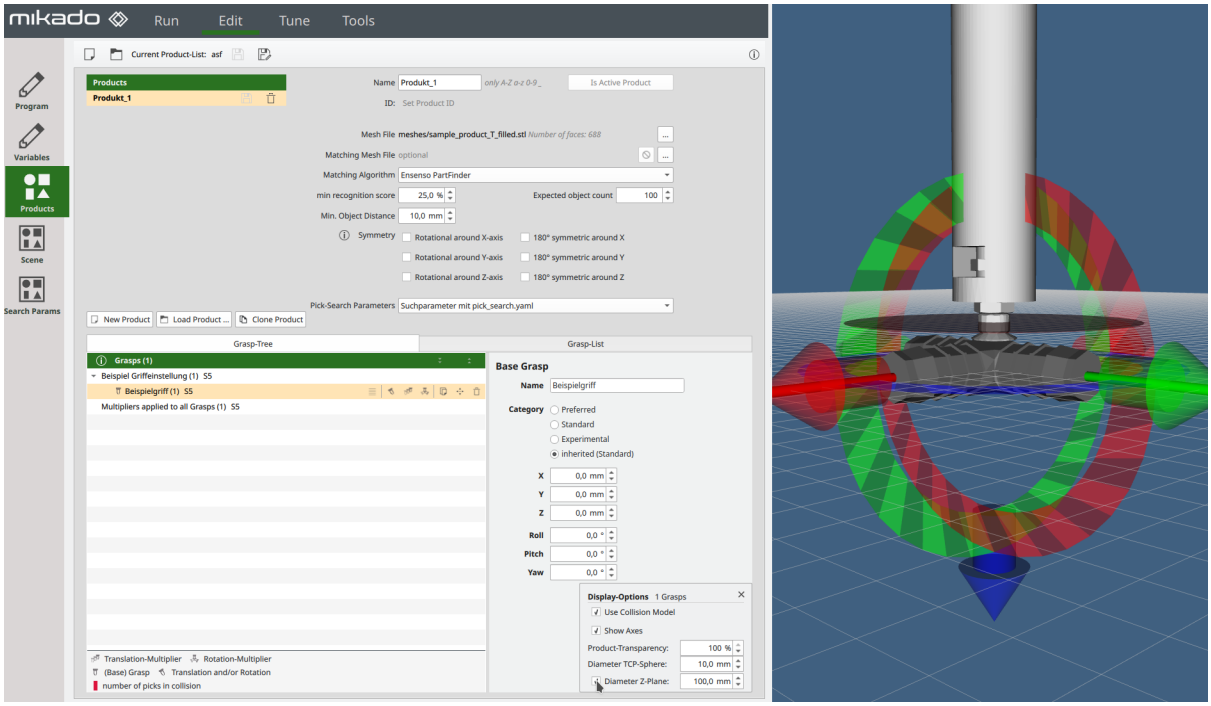


Fig. 4: Visualization of the area on the gripper that is not checked for collisions during the pick search. Everything below the dark gray plane is not treated as a collision object during the pick search.

In addition, for different robot manufacturers there now is the option of not activating the vacuum gripper at the transit position, but rather when a specified distance is reached during movement. This minimizes unwanted suction or displacement of parts before the actual gripping position is reached.

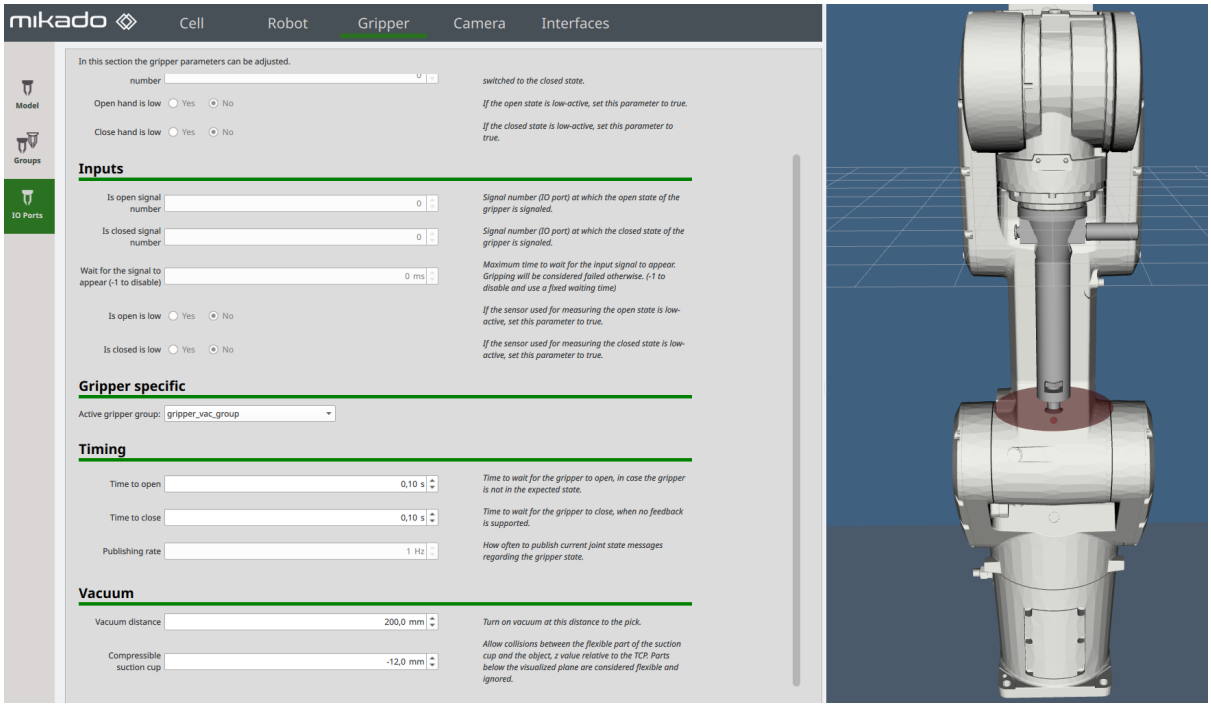


Fig. 5: Setting options for distance-based activation of a vacuum gripper and compression of the suction cup in the Mikado Assistant.

Additional filter options

To make complex programs easier to use, filter options have been added in various places. Variables and sequences can now be found more easily by entering their names in the corresponding text fields.

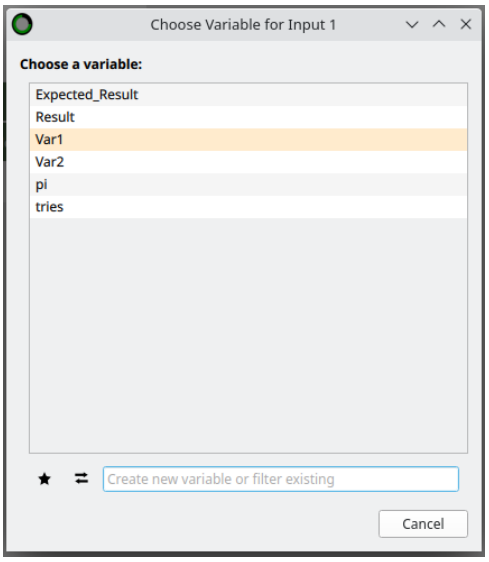


Fig. 6: New filter in the variable selection dialog

New filter options are available in the following locations:

- **Variable selection:** Filtering of boxes, ROIs, products, and other variables
- **Variable Editor:** Filter by Type
- **Sequence selection:** Filter by name
- **Program Editor:** Filtering when jumping to sequences
- *Best picks:** New filter to display only picks that were available at the time of pick selection
- **Pick inspector:** Filter by pick name

New and revised tasks

Three new calculation tasks are available in version 3.3.0: *Arithmetic operation*, *Logical operation* and *Comparison operation*. The *Arithmetic operation* task allows simple arithmetic operations to be performed with two variables (integer or double), the *Logical operation* task can be used to implement boolean operations, and *Comparison operation* allows two variables to be compared.

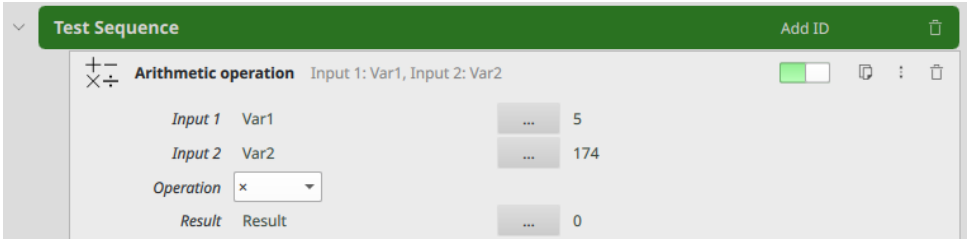


Fig. 7: New *Arithmetic operation* task

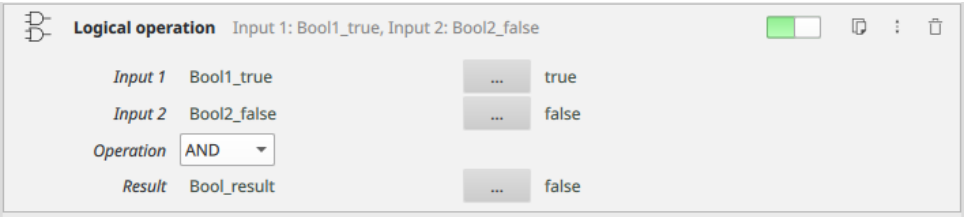


Fig. 8: New *Logical operation* task

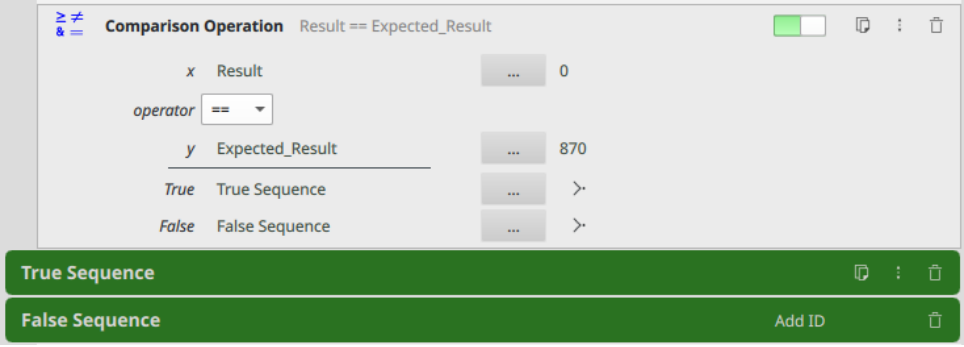


Fig. 9: New *Comparison operation* task

Another new feature is the *Set Fieldbus Output Signal Async* task, which sets a fieldbus output as soon as the robot has left a specified ROI. This enables convenient and fast feedback to the PLC when the robot has left a specific area.

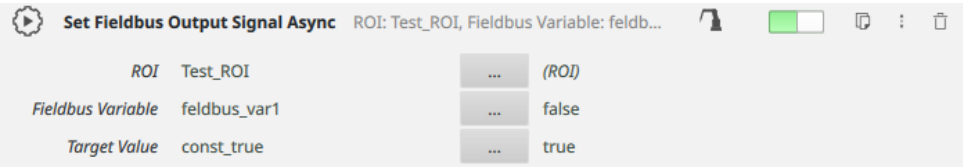


Fig. 10: The new *Set Fieldbus Output Signal Async* task

Further changes to tasks:

- *Create pose* can now also be used to overwrite a single component (e.g., the Z value)
- *Find object*, *Start handle search* and *Get Picks* no longer make changes to the active product
- In the *Pick object* task, an optional parameter can be set to ensure that the move to the transit pose is only executed if a pick was found

Extended functionality for JavaScript

In order to implement complex functionalities even better, the JavaScript task has been expanded and the following improvements have been added:

- When renaming a Mikado variable, it is also renamed in *JavaScript* tasks
- XMLHttpRequests can be allowed optionally
- For picks from the picks search, both the TCP poses and the Tool0 poses are now accessible
- `lab.callService` allows the use of any ROS services
- Native robot programs can be called from JavaScript

Improvements for the pick search

The pick search has been expanded to include various new parameters and sorting criteria. In addition, feedback to the user has been improved:

- *Start handle search* now accepts a parameter for the (future) gripper
- **Position equality**: Separate parameters for tolerance in recognizing unsuccessfully grasped and re-recognized objects: `Part pose max. translation difference`, `Part pose max. rotation difference`, `Grasp max. translation difference`, and `Grasp max. rotation difference`
- **Sorting criteria:** New sorting criteria `Object angle to vertical` and `Number of pick attempts`
- **Pick inspector:** New button to center visualization to the middle of the box
- **Pick inspector:** Notify if transit pose is unreachable or other problems occured
- **Pick search:** Improved consideration of user binning

Improved traceability for fieldbus commands

To make it easier to track commands received via a fieldbus, these are logged on the new `Fieldbus` page. It can be found in Mikado under the `Run` menu item and provides information about the type of command received, start time, duration, and its status.

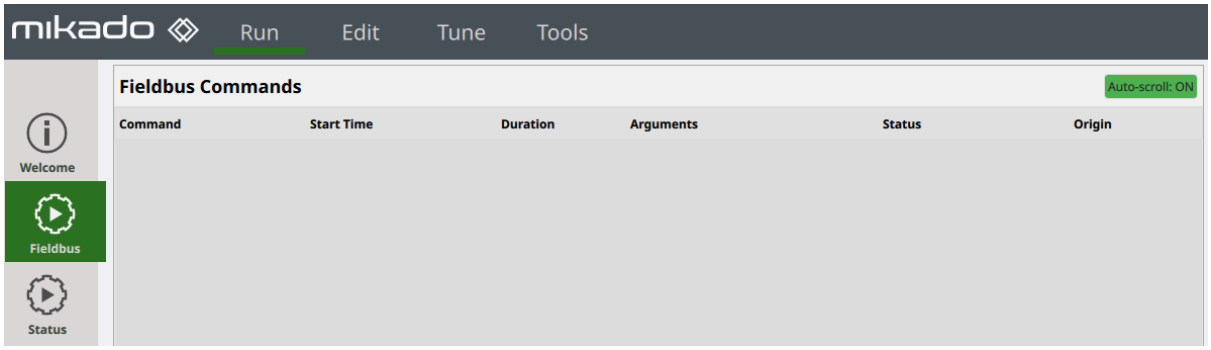


Fig. 11: All incoming commands are recorded and logged on the new fieldbus page.

Further fieldbus changes:

- **Fieldbus commands:** New commands for starting and evaluating a pick search directly via the fieldbus
- **Mikado Status:** Improved status messages regarding the loading of programs, products, and scenes

Object recognition

In version 3.3.0, the naming of symmetries has been revised and additional symmetries have been added. All symmetries are taken into account for object recognition and the `Move part to` task. On the recognition page, the new visualization of the views taken into account by PartFinder in the model generation tab, allow to better understand the effects of the symmetry selection.

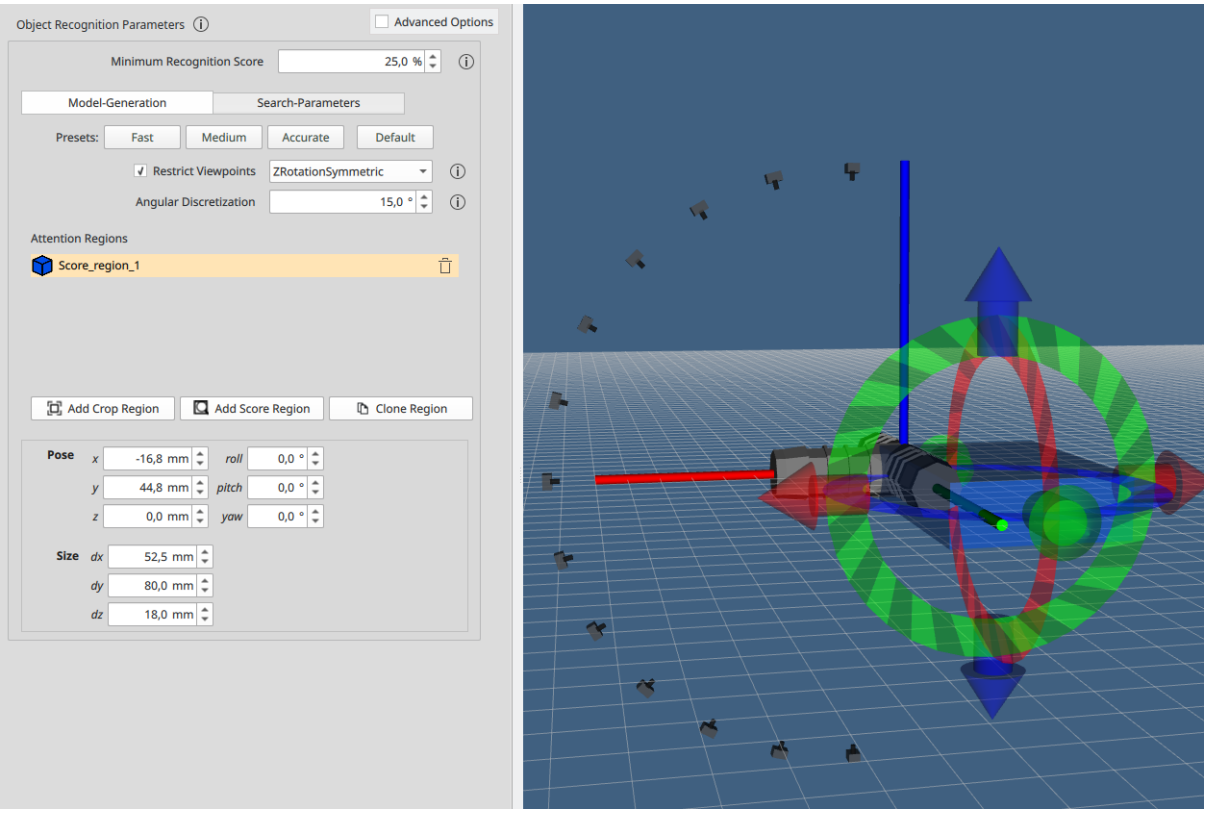


Fig. 12: New visualization of the views considered for object recognition and example of a `score region`

In addition, adjustments were made to Mikado to support Ensenso SDK version 4.2, resulting in a significant improvement in object recognition.

The following points have also been improved:

- **Save image:** The current product name can be used as a prefix when saving Filecams
- **Model generation:** Areas of the model can now be defined (`Score Region`) to be prioritized in the calculation of the recognition score
- **Product Mesh:** OBJ meshs can now be used directly as product meshes
- **Hypothesis clustering:** up to 4096 clusters allowed (as in Partfinder)

Debugging and Error Reporting

To simplify analysis in the event of an error, a button has been added to the Tools->Tools page to create an error report. After clicking on the button, you will be prompted to describe the error you have observed. After clicking on "OK," an archive file with relevant debug files is automatically created, which makes it much easier to investigate the error.

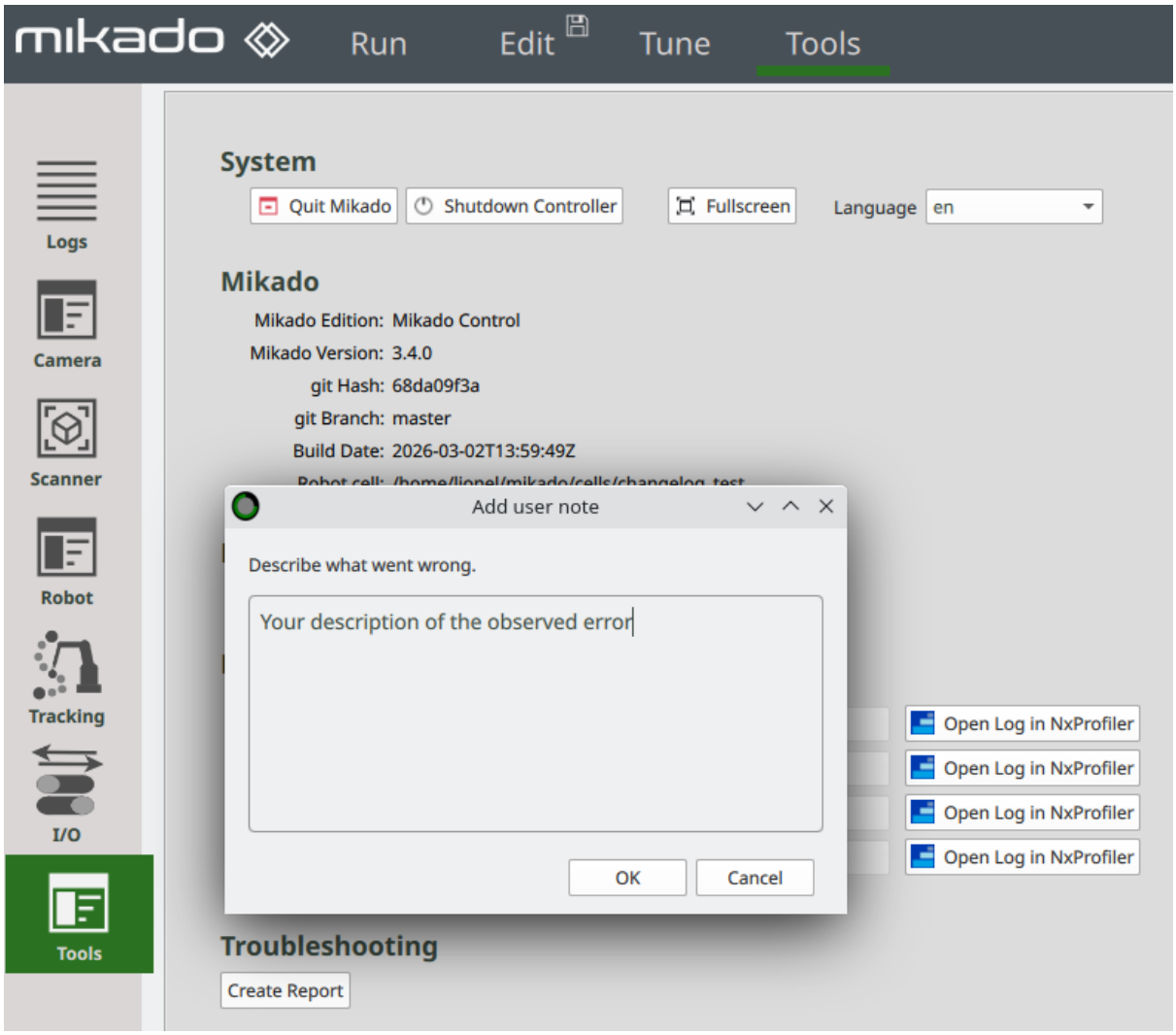


Fig. 13: Creating an error report on the Tools->Tools page

The log viewer under Tools->Logs has been modernized. The standalone log GUI can now also load saved logs for analysis.

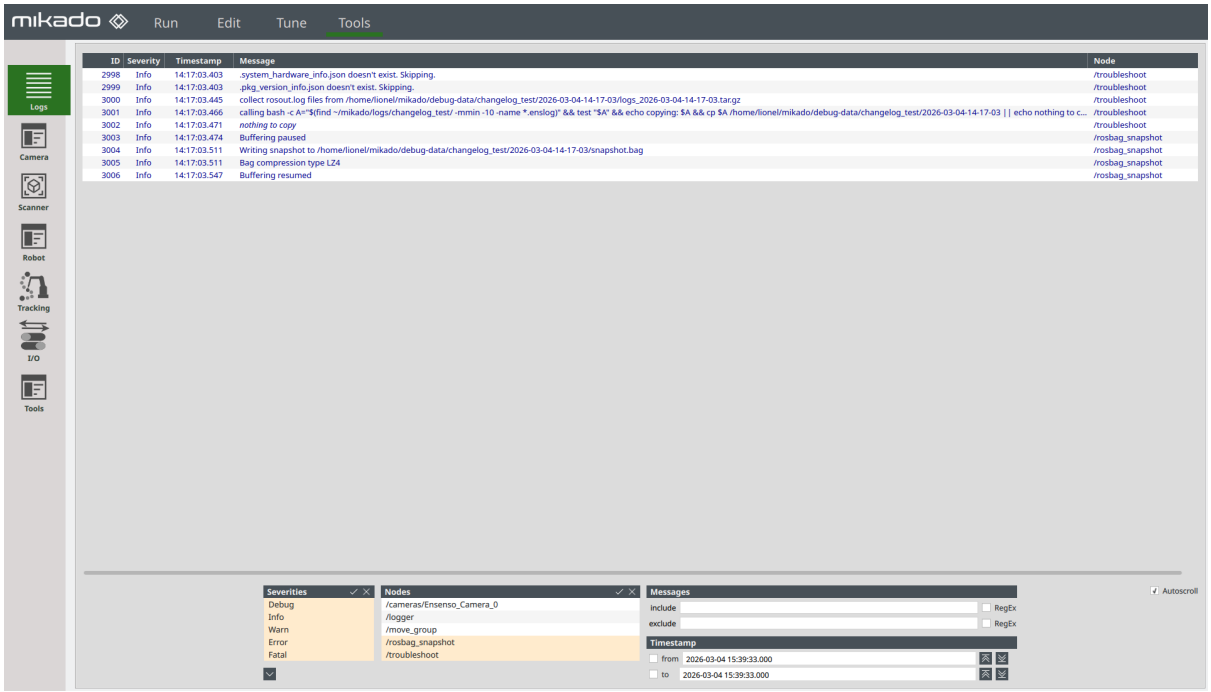


Fig. 14: View of the modernized Tools->Logs page

Better usability

- **Variables:** Cameras and grippers now have their own variable types
- **Grasp Editor:** Simplified setup with optional transparent products and markers adjusted to product size
- **Kiosk mode:** Exiting kiosk mode with Alt+Tab etc. is suppressed. The Windows/Meta key can also be suppressed using a KWin window rule. The status and logins in kiosk mode are logged.
- **Robot page:** The active gripper is now*displayed and can be changed
- **Setup mode:** New setup mode in which manual robot operation is not communicated as an error in the robot status
- **Scene objects:** Camera selection next to the capture button

Miscellaneous

- **Logging:** In addition to software versions, hardware and IP addresses are now also logged in the folder of a Mikado cell
- **Robot page:** Display of information about the robot model and manufacturer
- **Status bar:** Techman error codes are displayed in hexadecimal, as specified in the manual
- **Search strategies:** The step size of cartesian movements is adjustable
- **Kuka:** For Kuka robots, parameters for influencing speed have been added to the robot page
- **MeshScanner:** Do not always save meshes in OBJ format
- **Mesh names:** More special characters allowed in mesh file names
- **Program editor:** Warning if program cannot be written due to full hard disk
- **Scene objects:** Improved feedback when searching for box fails
- **Optimizations:** Optimizations in loading and validating programs as well as in calculating picks

- **Bug fixes:** Various bug fixes and stability improvements

9.1.3 Version 3.2.0

Graphical PickSearch tuning

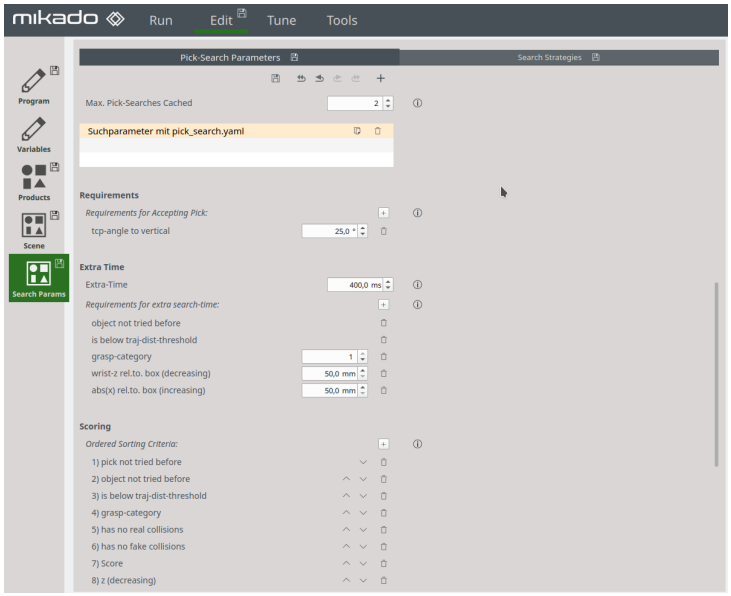


Fig. 15: Tuning the parameters of a pick-search

It is now possible to configure the parameters of a pick-search directly in Mikado. Depending on specific needs parameter sets can be set per product or switched on demand. Use cases may include enforcing a strict order of depalletizing or prioritizing certain extraction strategies during a pick. The rudimentary configuration options that previously existed in Mikado Assistant have been removed.

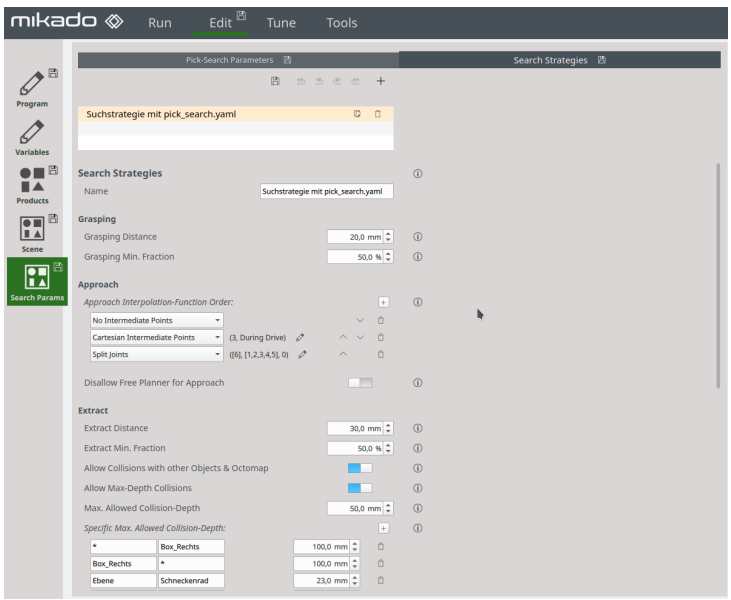


Fig. 16: Configuring the extraction strategies

Some new extraction strategies were added as well. Especially sliding out objects in their xy-planes has proven valuable for randomly stacked cut-metal parts.

Switching between several pick-searches

Multiple pick-searches can now be switched between to allow efficient alternating picks from multiple boxes. All pick-searches can be inspected and optimized using the pick inspector or best picks tool. Multiple grippers can be tested without requiring a new object search. This happens either automatically when rerunning Start Pick-Search or by using JavaScript (see *rerunPickSearch*).

Numeric IDs for grippers, cameras, products and scene objects

To simplify external control of the process flow using a PLC it is now possible to assign numeric IDs in addition to the existing names. This was added for grippers, cameras, products and scene objects.

Improved communication of joint limits

Issues due to violated joint limits are now more readily visible. Error messages were added at program start, when jogging or in the editor for joint position variables.

New PartFinder Features

Some new parameters of the PartFinder object recognition algorithm added with Ensenso SDK 4.1 were made accessible. This includes TexturePenalty or ComplementViewpoints. Additionally it is now possible to restrict the object recognition to certain regions of the model without artistically editing the CAD mesh model. These so called attention regions can be configured intuitively now.

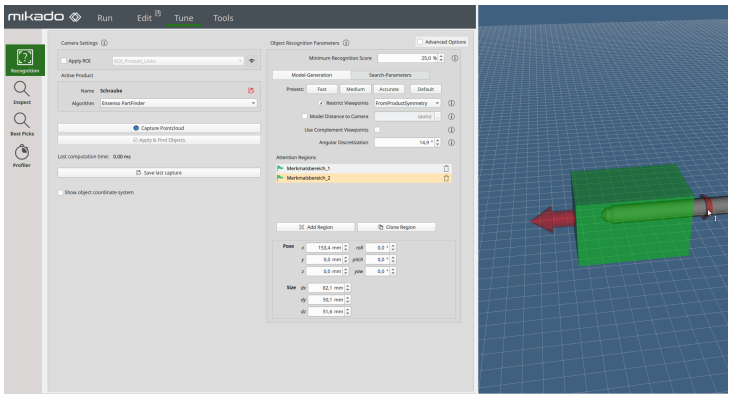


Fig. 17: Example attention region: only the tip of a screw is of interest

For rapid interpretation of object recognition results it is now possible to display their local coordinate axes.

Accessing Picks in Mikado Select

The collision-checked pick search in Mikado Select is now performed asynchronously as in Mikado Control via Start Pick-search and the new Get Picks task. The previous Find Objects task is automatically replaced by the two tasks during the upgrade if the pick-search was previously used.

Further innovations

- Program: Tasks can now be disabled or commented out and are skipped during execution
- Jogging: Filter variable names
- Filter variables in variable picker dialogue
- JS: reading and writing IOs (see *robot*)
- JS: set the state of a task to Error or ErrorMightRetry (see *task*)
- JS: lab.callExtensionService for arbitrary extension scripts (see *callExtensionService*)
- WaitForFieldbusInputSignalTask: task to wait for value changes in a (fieldbus) variable (see *Wait for fieldbus input signal*)
- SetInitialValueTask: optionally save the program (see *Set initial value*)
- Display variables from “addons” as well
- Status: jump to currently active task
- Yaskawa specific functions and error codes
- FindObjectTask no longer needlessly cancels pick-searches running in parallel
- Number of contact points in collision checking can be configured
- Default number of contact points during collision checking was increased to get more realistic overlap results
- PickInspector: more details why picks where filtered
- PickInspector: communicate clearly when grasps collide with the object to be grasped
- Variable page now also displays when variables are used in JavaScript tasks
- Show the number of triangles in a scene objects mesh
- Immediately cancel asynchronous capturing if the robot fails to move
- Use new profiler file format
- separate package mikado-robots-common for helper programs
- Usability improvements in the grasp editor
- Filecams are now stored in mikado/filecams and are limited in size
- New task: Reset scene object position

9.1.4 Version 3.1.2

References to scene objects and products

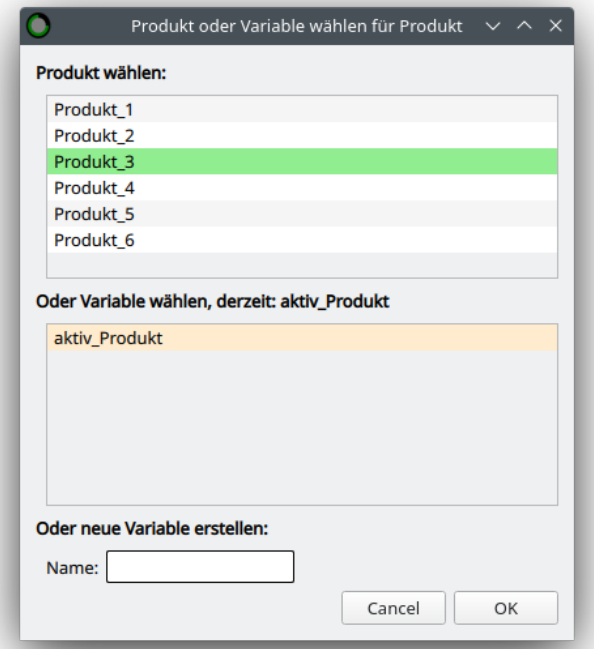


Fig. 18: Selection of a program variable or a product reference

Products and scene objects can be referenced directly in tasks. This feature is particularly useful if the same ROI is always to be used within a task or the same product is to be searched for. It is still possible to set the value via a program variable.

Revision Robot tracking

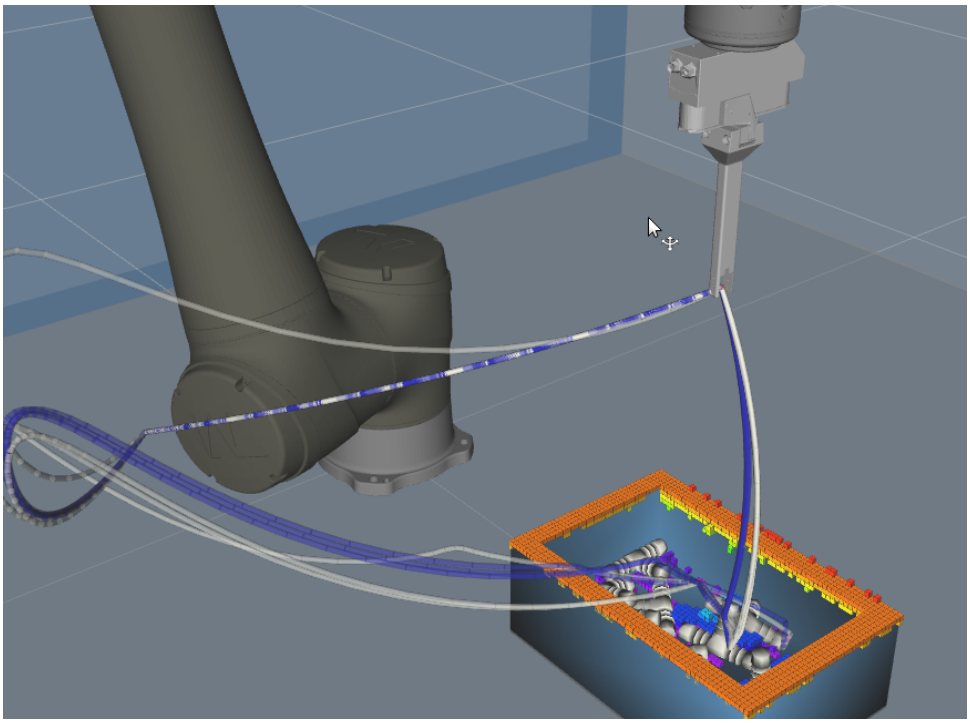


Fig. 19: Visualized path of the active TCP

On the *Robot tracking* page, the path of the active ToolCenterPoint (TCP) is visualized as a line. The page also supports the use of multiple grippers.

Position controlled grippers

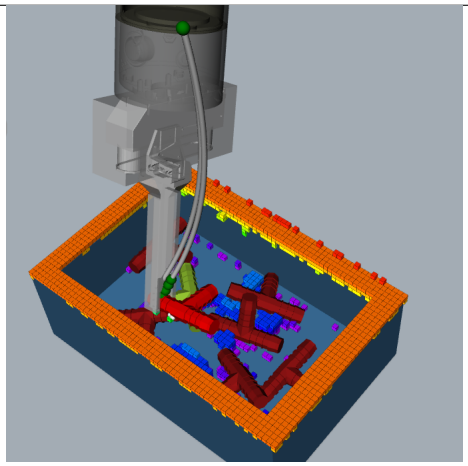


Fig. 20: Grasp with small opening width

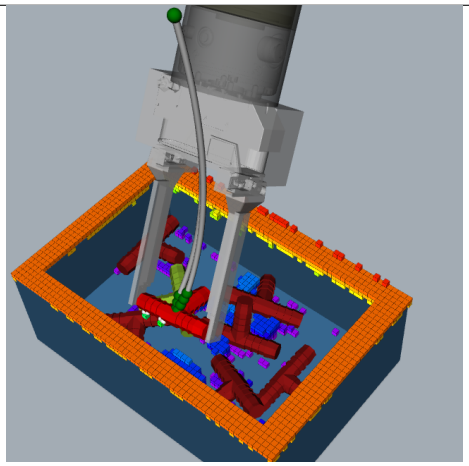


Fig. 21: Grasp with large opening width

For grippers with position control, the opening width of the gripper fingers can be set for each Grasp-Setting. To do this, the gripper type must be set to `type: ParallelPositionControl` in the corresponding `gripper.yaml`.

Mesh-Scanner

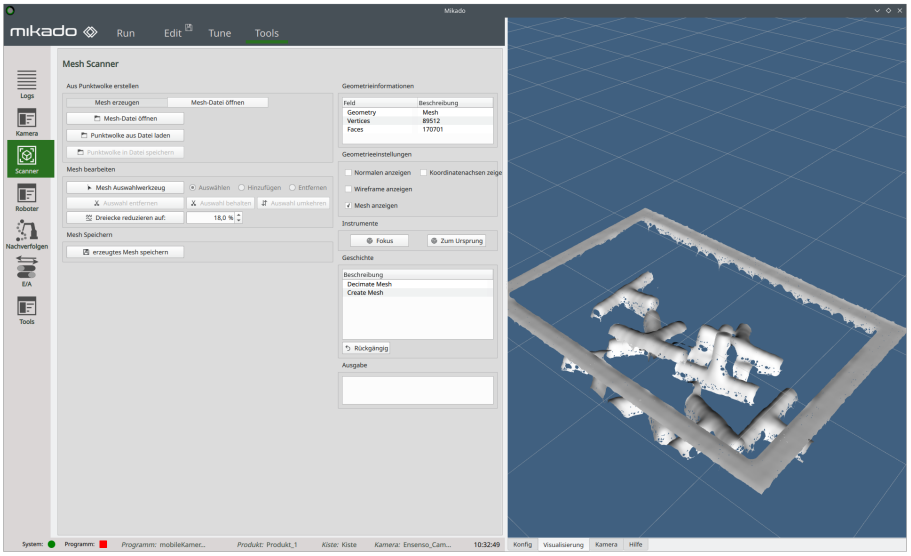


Fig. 22: Screenshot Scanner Page

The scanner page offers the option of creating a mesh from a point cloud and post-processing the created mesh. The mesh can then be used in the product search. The scanner is able to:

- Capture point cloud or load point cloud/mesh
- Create mesh from point clouds
- Simplify mesh
- Cut out parts of the mesh
- Save mesh

Collision check of grasps in the grasp editor

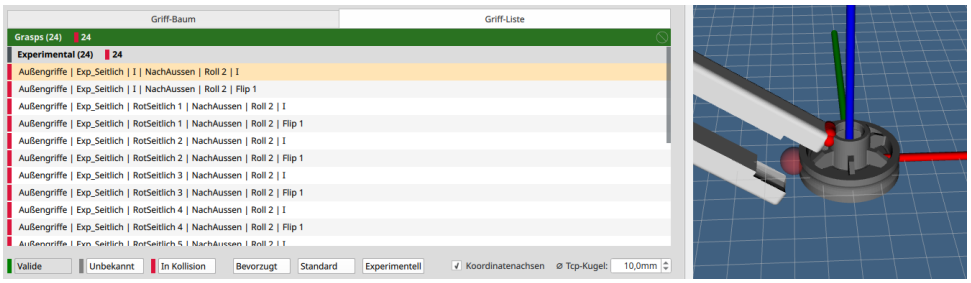


Fig. 23: Grasp list and visualization of collision points

Newly created grasps are checked for collisions with the part when they are created (see *Collision Check for Grasp-Poses*). Next to the grasp tree there is the tab grasp list. This list contains all grasps generated by the grasp tree. If you select a grasp in the grasp list, it is displayed in the visualization. If a grasp collides, the detected collision points between the gripper and the product are marked with small red spheres.

Minor optimizations

- Significantly faster Profinet interface thanks to event-based backend. For Details refer to: *Version 3.1.0*
- GUI language can be changed without restarting Mikado
- Robot status display tidied up, more error details
- Status and program editor: more informative placeholder texts
- Update Octomap now also possible from JS, without pick search / FindObjects (see *updateOctomap*)
- Pick inspector: pick search can be started manually
- Picks can be sorted by *Collision-Depth*
- JavaScript: Read CSV file into dictionary, e.g. for product parameters (refer to *file*)
- Warning if a product uses very large meshes
- Erroneous variables are displayed on the variable page, e.g. writing to read-only variables or variables that have an incorrect type when used
- JavaScript: Tcp coordinates for grasps available, i.e. Start, Approach, Retract and Final (siehe *picksearch*)
- QML: it is now possible to start any process, e.g. Okular for a project specific manual
- Interface to implement object recognition with a project-specific algorithm
- PickSearch/JS: Additional freely usable evaluation criterion for objects (user binning)
- Adjusting the position of a gripped part
- Move robot to: possible collisions directly in the status of the task
- Grasp editor: Copying and moving grasps
- X > Y? and similar tasks with blue symbols for logic/jumps
- Task to read out the length of a list
- FindObjects: optional sequence change if box was searched for and not found
- JS: More information on last used grasp
- Picks can be sorted by collision depth

9.1.5 Version 3.0.5

- Error corrections in the handling of newly created variables

9.1.6 Version 3.0.4

- Cuda can be activated for the PartFinder product search
- Very short Cartesian moves are no longer extended to 10 points

9.1.7 Version 3.0.3

- For developers, “Find Objects” is also visible in Mikado Control
- Find Objects now also has an optional “Box” parameter
- PartFinder: maximum number of clusters increased to 200
- new Task: Move Tool Center Point (TCP) Cartesian, optionally with tolerance to the target.
- Display information about the currently used FileCam on the object detection page
- Grasp translations can also be set to symmetric now

9.1.8 Version 3.0.2

- Picks: entire trajectory can be visualized in preview
- read-only variables
- Variable page: favorite variables

9.1.9 Version 3.0.1

- Detailed display of object search / pick search pipeline
- ADS interface support
- NxLogs GUI for profiling

9.1.10 Version 3.0.0

Collision check of the gripper for Mikado Select

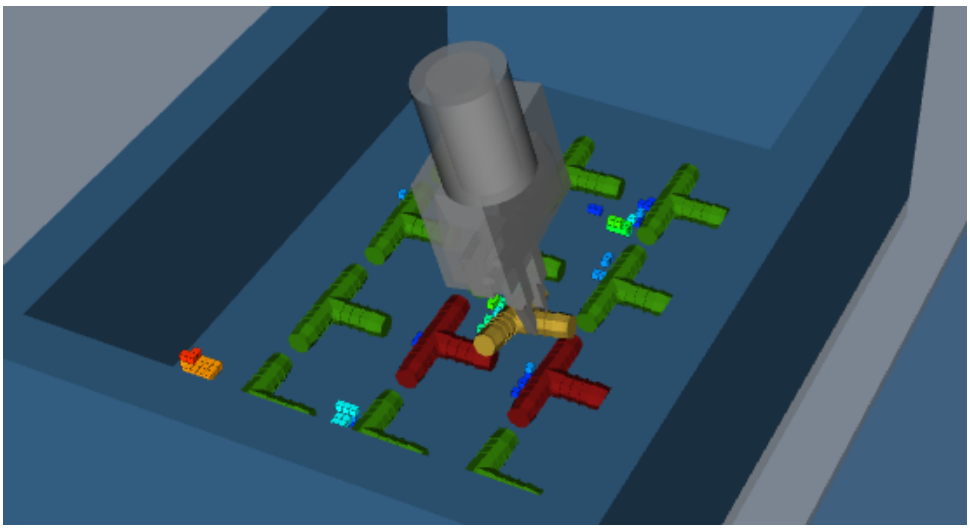


Fig. 24: Collision check of the gripper for Mikado Select

Mikado Select supports the finding of collision-checked handles. The task *Find object* has a new return value `Valide Griffe` for this purpose. This pose list contains the best grasp per found and graspable object. The list is sorted according to user-specific criteria (see *Best handles*).

Visualization of the handle prioritization

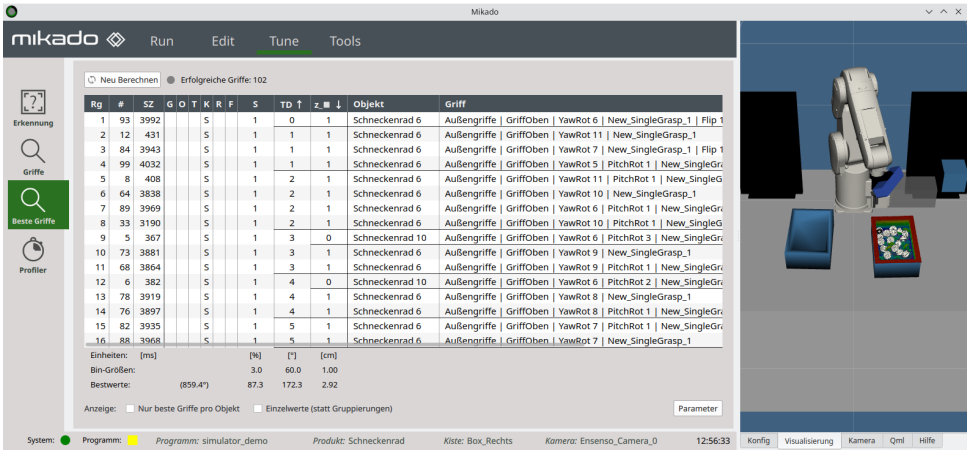


Fig. 25: Screenshot Best-Griffe

Sorting and exclusion criteria for handles can be defined globally (for all products) in `pick_search.yaml`. Binning is used to compare all handles according to the various criteria. The new *Best handles* page provides an analysis tool for the sorting criteria.

Support for tool changers

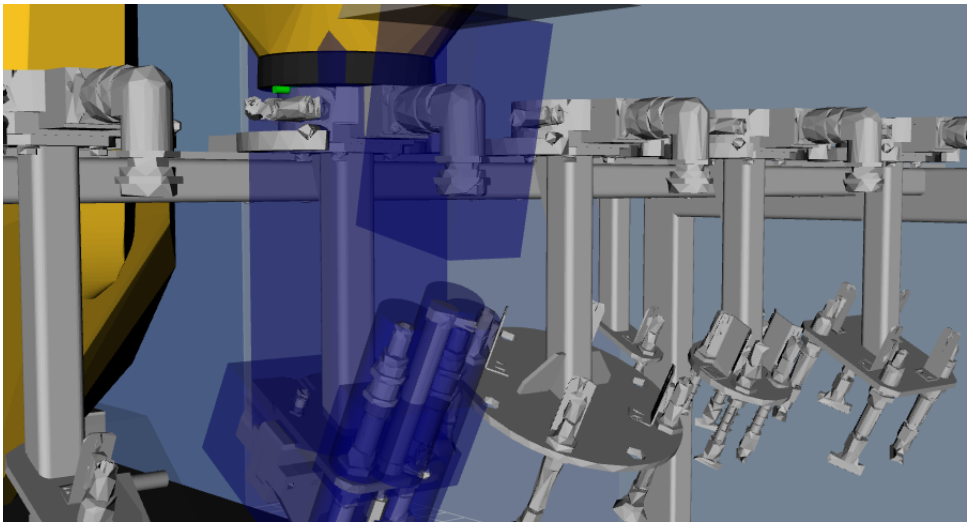


Fig. 26: Support for tool changers

Mikado Control supports changing grippers during the runtime of a robot cell. The new task *Set active gripper* can be used to switch between the defined gripper geometries within the robot program. Products now have gripper-dependent grip definitions, see *Setting up handles for interchangeable grippers*. The grip search only checks grips that have been defined for the actively set gripper.

Setting up a cell with multiple grippers will be handled by Mikado support for the time being. The addition of new grippers in Mikado Assistant is in progress and is planned for the next release. Editing cells already set up in Mikado Assistant is fully supported.

Additional joints also for parallel grippers

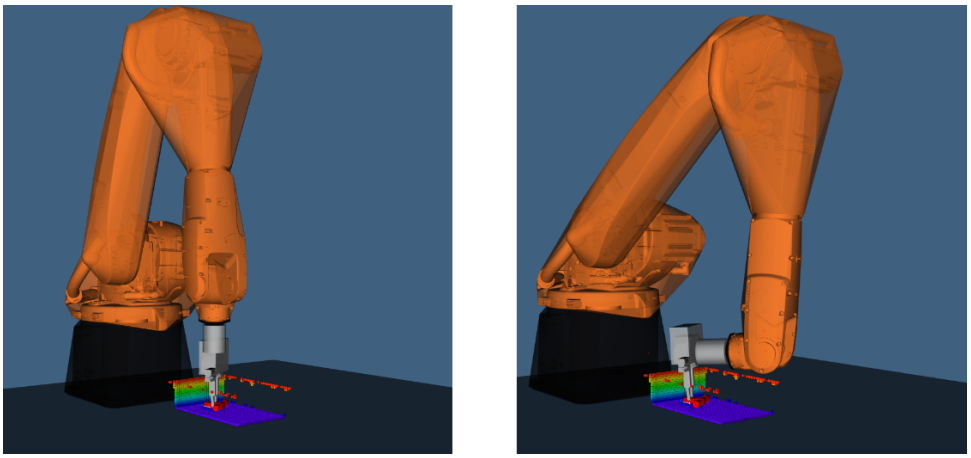


Fig. 27: Tilting joint for parallel gripper

Additional joints on the gripper are also supported for parallel grippers. Previously, only magnetic and vacuum grippers were possible (geometry does not change when opening/closing). Further details can be found in the documentation of the *Setting the gripper status* task.

Library with sample programs

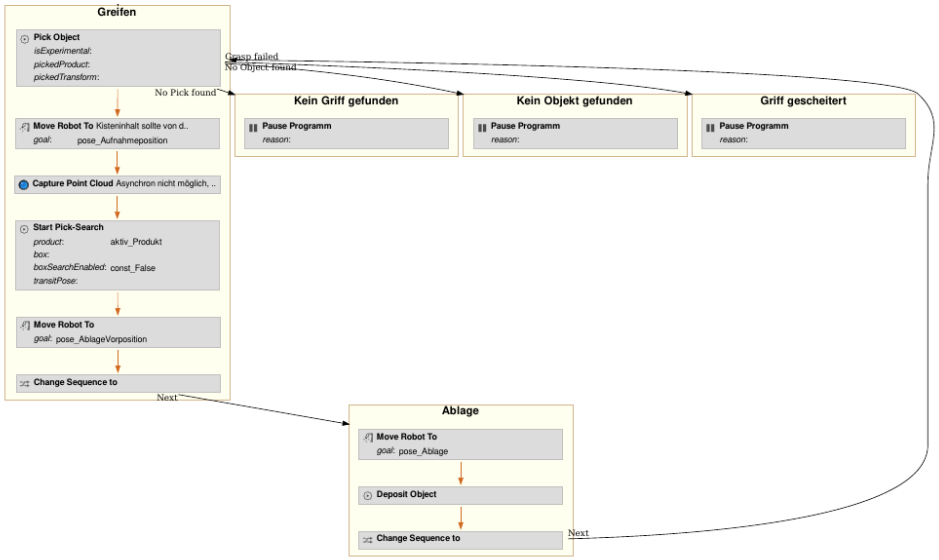


Fig. 28: Sample program for mobile camera

The program library is located on the Welcome page and can be opened by clicking on the folder icon to the left of the ‘Programs’ heading. It offers sample programs that are suitable as a starting point for project-specific programs. These include:

- **Select**
  - TCP calibration program
  - Finding objects and transferring the results via Profinet interface
- **Control**

- TCP calibration program
- Asynchronous image recording with stationary camera
- Synchronous image recording with mobile camera

If-Else tasks for conditional execution of partial sequences

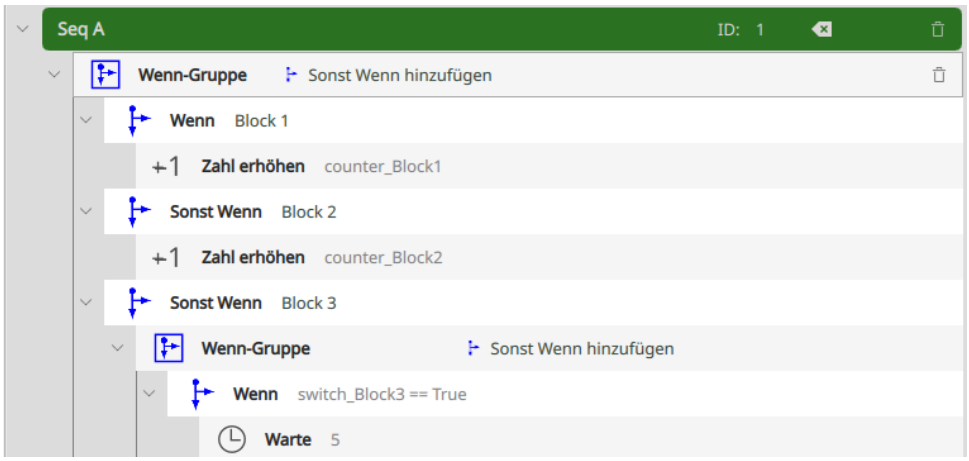


Fig. 29: Nested if-groups in the program editor

The new *If-Group* task enables the conditional execution of partial sequences. This feature also enables, among other things:

- Copying sequences
- Subdividing sequences into partial sequences
- Parallel deletion of multiple tasks
- Redesign of the program editor

Rotation on jogging side

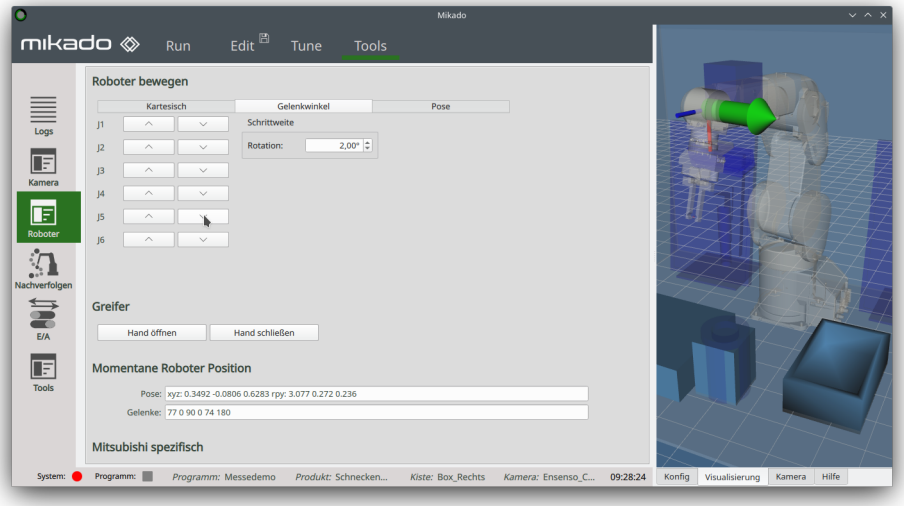


Fig. 30: Jogging page: Preview of the rotation axis with green arrow

In addition to Cartesian movements in the x, y and z directions, the robot can be rotated piece by piece on the robot page (see *Robot control*). The coordinate system in which the robot is rotated can be selected manually. The following coordinate systems are available:

- World - Coordinate system
- Robot - Coordinate system
- Tool0 (or robot wrist) - Coordinate system

Further innovations:

- The movement is displayed as a preview on the right-hand side
- Redesign of the robot page

Depalletization rules

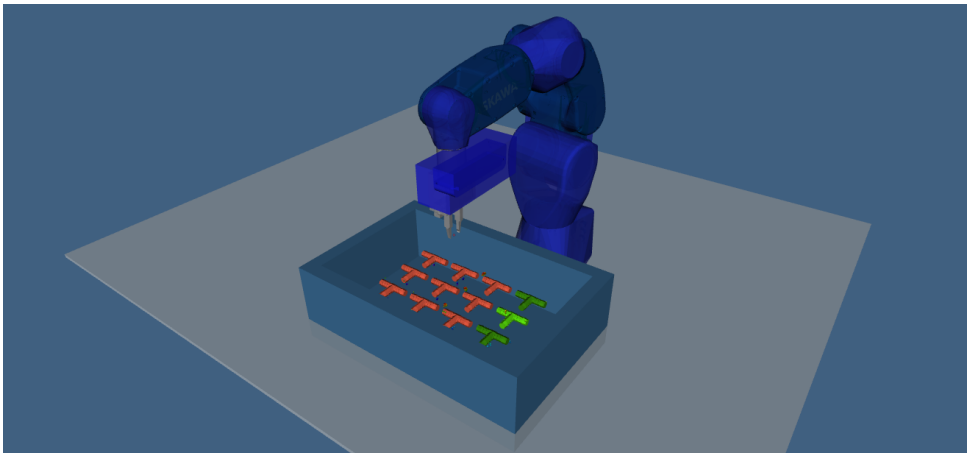


Fig. 31: Only picks with the smallest x-value are permitted

The new parameter `max_bins_required` enables the exclusion of certain findings (see *Conditions*). For example, products with a high x-value relative to other products can be excluded. This enables depalletizing from left to right.

More information on failed picks

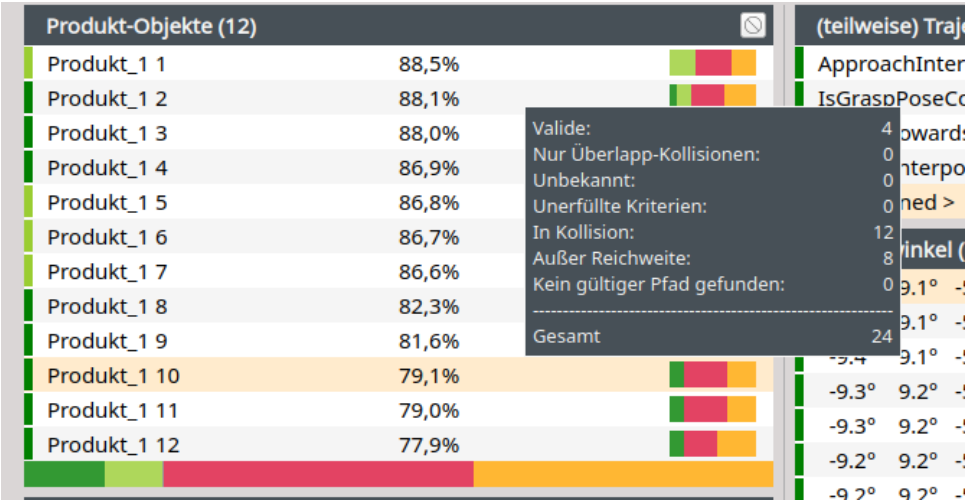


Fig. 32: Re-design of the Pick Inspector page

The pick inspection page has been revised and now offers the following additional features:

- Some collision pairs have additional explanatory text
- Bar charts show the current status of the handle search as a whole and the product-specific status

JavaScript: API for scene objects

```
var size = scene.getSceneObjectDimensions("ROI1")
scene.setSceneObjectDimensions("ROI1", size.x, size.y, 0.23)
```

Fig. 33: Setting the z-value of an ROI to a fixed value

The size and position of scene objects can now be read and written via the JavaScript API (see *scene*). The following new functions are available:

- `getSceneObjectDimensions`
- `setSceneObjectDimensions`
- `getSceneObjectPose`
- `setSceneObjectPose`

9.1.11 Version 2.1.0

- Integration of Ensenso PartFinder object recognition
- Object detection parameters are saved and set on a product-specific basis
- Total number of handles is displayed with the product
- Both individual grip patterns and all resulting grips can be displayed

Pick inspection:

- Objects are colored according to their tangibility and can be clicked on directly
- Picks can be filtered by success

Extensions:

- The results of the handle search can now be called up via JavaScript
- JavaScript kinematics API to check the accessibility of multiple poses or paths, for example
- The dimensions of the point cloud within an ROI can be queried via JavaScript
- Calibrations via a ROS interface (and thus e.g. via OpcUA or ProfiNet)

9.2 Scene objects

9.2.1 Version 5

Warning: If your program uses a *Read scene object position* or *Set scene object position*, the behavior of the program may change. Please check your program accordingly.

The meaning of the `default pose` of a scene object has been changed. In version smaller than 5, the default pose describes the center of the floor of a scene object. The `default size` parameter was used to determine the floor. This meant that the actual pose of the scene object depended on the default pose and the default size. The dependency on the default size has been removed. As of version 5, the default pose for scene objects with cuboid geometry (such as ROIs) describes the position of the center of gravity of the volume instead of the center of the floor as before. For scene objects with mesh geometry (such as boxes), the default pose now directly describes the position of the origin of a mesh in the world coordinate system. In addition, the new parameter `Mesh-Offset` has been introduced to be able to move the origin of a mesh from Mikado.

With the version update, the various pose parameters for scene objects are automatically changed to the new meaning. The following operation is performed:

1. Shift the `default pose` by half the z-component of the default size d_z in the positive z-direction of the local coordinate system,
2. Shifting the `transition pose` by the same amount and in the opposite direction and
3. Set the z-component of the `mesh offset` to $-\frac{1}{2}d_z$ (same amount as for 1 and 2).

The new meaning of the default pose is shown in the following illustration.

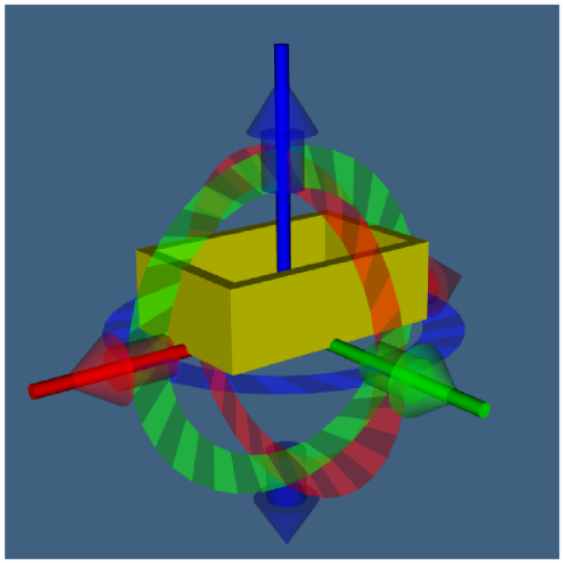


Fig. 34: Old meaning of the `default pose`

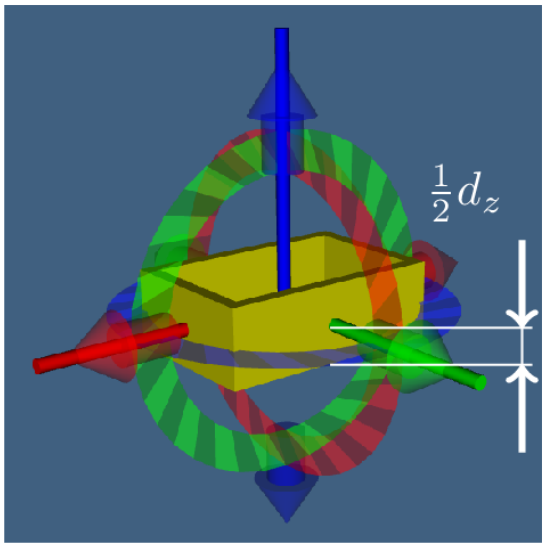


Fig. 35: New meaning of the `default pose`

The default size is set in Mikado as shown in the following screenshots.

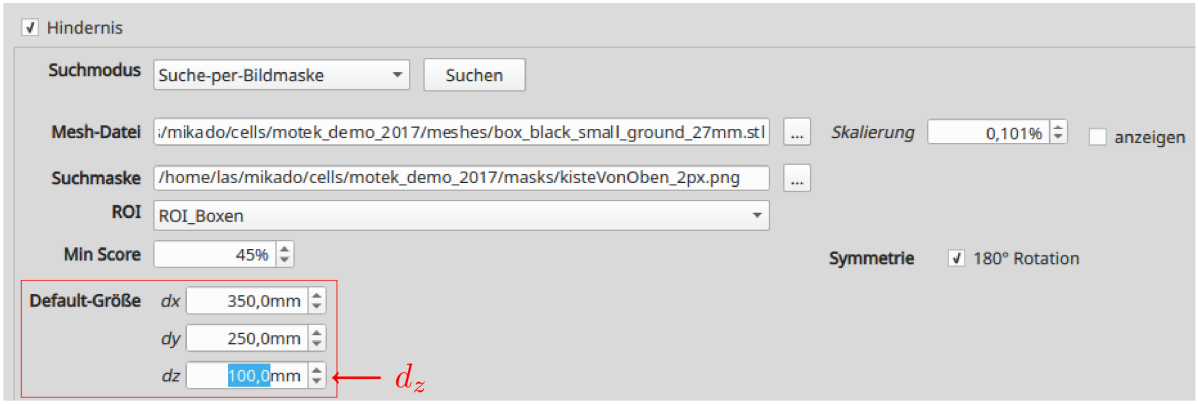


Fig. 36: Setting the default size of a crate

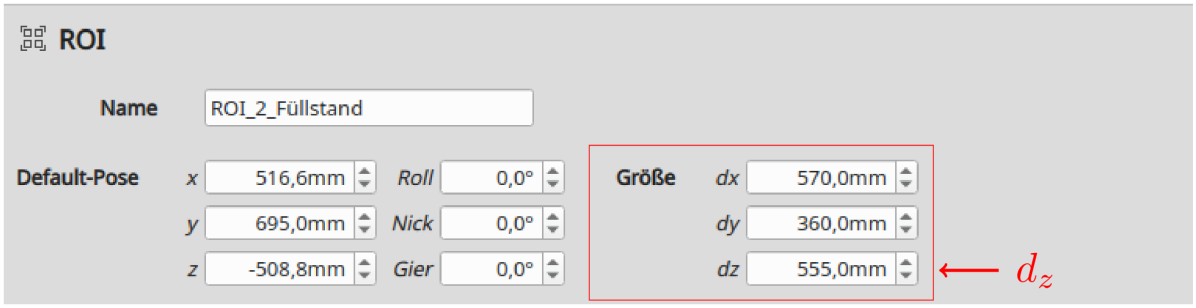


Fig. 37: Setting the default size of an ROI

9.2.2 Version 4

New function `ROI` added, which is required when searching for scene objects. Please select an ROI object for each scene object that uses `Search-By-Image`.

9.2.3 Version 3

New function `Search symmetries` added, which is used when searching for scene objects. Disabled by default.

9.2.4 Version 2

The standard file extension for scene objects has changed. Please save your Mikado scene object files with the extension `*.scene` from now on. The paths for the start program in config/mikado.yaml must be edited manually.

CHAPTER

TEN

HISTORY OF CHANGES

The version numbers listed here follow the releases of the Mikado manual in the package **mikado-docs-user-<language>**.

Table 1: List of all changes

Date	Version	Details
18.12.2017	1.0.1	Initial Version
18.01.2018	1.2.1	Mikado Version 1.2
23.02.2018	1.2.2	Mikado Version 1.2 Revision
09.04.2018	1.3.1	Mikado Version 1.3
10.09.2018	1.4.1	Mikado Version 1.4
18.10.2018	1.4.2	Mikado Version 1.4 Revision
18.02.2019	1.5.1	Mikado Version 1.5
05.03.2019	1.5.2	Mikado Version 1.5 Revision
27.06.2019	1.6.1	Mikado Version 1.6
12.07.2019	1.6.2	Mikado Version 1.6 Revision
20.07.2019	1.6.3	Integration expanded
07.10.2019	1.6.4	Mikado Version 1.6 Revision and assembly instructions
09.12.2019	1.7.1	Mikado Version 1.7
23.06.2020	1.7.2	Mikado Version 1.7 Revision
28.01.2021	2.0.1	Mikado Version 2.0
13.12.2021	2.1.1	Mikado Version 2.1
07.03.2022	2.1.2	Detail corrections for Assistant
24.05.2022	2.1.3	JavaScript API documented, notes on robot speed
04.08.2022	2.1.4	Yaskawa Motoman supplements. Other robot models explained in more detail
05.10.2022	2.1.5	More details on Pick search
26.10.2022	2.1.6	Small corrections
16.03.2023	2.1.7	PartFinder, assembly instructions and TCP calibration
12.04.2023	2.1-8	Documentation of all Tasks
01.07.2023	3.0-0	Mikado Version 3.0
06.02.2024	3.0.1	Robot setup is described directly under *Robot models*. Documentation for Doosan Setup, reading CSV with JavaScript, task "Move TCP Cartesian", "Robot tracking". Fieldbus rotation conventions. ADS data types and read-only variables.
12.03.2024	3.0.2	Manual translated to English, Description of getCurrentGraspOffset and link from Mikado GUI to “Wait for Robot Task” fixed.
24.05.2024	3.1.0	Pick prioritization according to *Collision-Depth*, *Collision Check for Grasp-Poses*, new features in Mikado *Version 3.1.2*

CHAPTER

ELEVEN

COPYRIGHT

© Copyright Optonic GmbH, 2015-2025 - All rights reserved. - No part of this publication may be reproduced in any form or by any means, mechanical or electronic, without the express written permission of Optonic GmbH. Optonic, is a trademark or registered trademark of Optonic GmbH. Other brand names mentioned herein are the property of their respective owners.

Mikado Control uses the following free libraries under the terms of LGPL V. 3.0:

- Qt: http://www.qt.io
- breeze icon theme: https://github.com/KDE/breeze-icons

Optonic will provide you with the source code of these components in the form used upon request within a period of three years after receipt of this software. The exact wording of the LGPL V.3 can be found at the end of this overview.

Mikado Control uses the following free libraries under the terms of the BSD:

- MoveIt: http://moveit.ros.org
- rviz: https://github.com/ros-visualization/rviz
- weitere ros Pakete: cv_bridge, diagnostic_msgs, eigen_conversions, geometric_shapes, moveit_visual_tools, octomap_msgs, octomap_rviz_plugins, tf_conversions

The list of ROS packages listed is not exhaustive. Detailed information can be found in the installed packages and will be added here.

Below is the exact wording of the LPGL V.3 and the BSD license:

11.1 GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

- a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or
- b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

- a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

- a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.
- b) Accompany the Combined Work with a copy of the GNU GPL and this license document.
- c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.
- d) Do one of the following:
  - 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.
  - 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user’s computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

- e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.
- b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

11.2 BSD 3-Clause License

Copyright (c) 2008-2013, Willow Garage, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.