optonic
vision · automation

Handbuch

mikado

Control

Version 3.4

Ein Produkt von 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

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Schnellstart . . . 1
- 1.2 Die verschiedenen Werkzeuge . . . 2

2 Verwendung **3**
- 2.1 Navigation . . . 3
- 2.2 Startseite . . . 4
- 2.3 Bediener-Seite . . . 5
- 2.4 Einrichtungs-Seite . . . 5
- 2.5 Statusseite . . . 5
  - 2.5.1 Steuerbuttons . . . 5
  - 2.5.2 Programmablauf . . . 6
- 2.6 Statusleiste . . . 6
  - 2.6.1 Systemstatus . . . 7
- 2.7 Passwortschutz . . . 8
- 2.8 Programm-Editor . . . 8
  - 2.8.1 Mikado-Programme . . . 8
  - 2.8.2 Grafische Oberfläche . . . 9
  - 2.8.3 Toolbar . . . 10
  - 2.8.4 Programm erstellen . . . 11
  - 2.8.5 Programm-Validator . . . 12
- 2.9 Variablen-Editor . . . 12
  - 2.9.1 Variablentabelle . . . 13
  - 2.9.2 Bearbeitung von Variablen . . . 13
  - 2.9.3 Variablen Hinzufügen und Filtern . . . 16
  - 2.9.4 Variablen aus Datei importieren . . . 17
  - 2.9.5 Visualisierung der Roboter-Pose . . . 17
- 2.10 Produktverwaltung . . . 20
  - 2.10.1 Überblick . . . 20
  - 2.10.2 Name . . . 21
  - 2.10.3 Aktives Produkt . . . 21
  - 2.10.4 Objektsuche . . . 22
  - 2.10.5 Greifposen . . . 24
  - 2.10.6 Kollisionsprüfung der Greifposen . . . 28
- 2.11 Griff-Suche . . . 30
  - 2.11.1 Überblick . . . 30
  - 2.11.2 Greifprozess . . . 30
  - 2.11.3 Greifbewegungssuche . . . 32
- 2.12 Szenenverwaltung . . . 43
  - 2.12.1 Überblick . . . 43
  - 2.12.2 Hindernis . . . 45

2.12.3 Kiste . . . 45
2.12.4 Region of Interest (ROI) . . . 47
2.13 Objekterkennungs-Tool . . . 48
2.13.1 Voraussetzungen . . . 49
2.13.2 Erste Schritte . . . 50
2.13.3 Kamera-Auswahl . . . 50
2.13.4 Letzte Aufnahme speichern . . . 50
2.13.5 Visualisierung der Szene . . . 50
2.13.6 Halcon-spezifische Objekterkennungsparameter . . . 51
2.14 Griff-Inspektion . . . 53
2.14.1 Überblick . . . 53
2.14.2 Inspektion der Greifbewegungssuche . . . 54
2.15 Beste Griffe . . . 57
2.15.1 Priorisierung . . . 57
2.15.2 Beste-Griffe-Anzeige . . . 69
2.16 Profiler . . . 70
2.17 Logs . . . 71
2.17.1 Überblick . . . 71
2.18 Kamera-Einstellungen . . . 71
2.18.1 Livebild 2D . . . 72
2.18.2 Kamera-Tab . . . 72
2.18.3 Hand-Auge-Kalibrierung . . . 73
2.19 Robotersteuerung . . . 78
2.19.1 Direkte Bewegungen . . . 78
2.19.2 Inbetriebnahme-Modus . . . 79
2.19.3 Greifer . . . 79
2.19.4 Bewegen . . . 80
2.19.5 Roboterparameter . . . 80
2.20 Roboter Nachverfolgen . . . 80
2.21 Übersicht Roboterein- und ausgänge . . . 81
2.21.1 Hinzufügen/Entfernen von E/As . . . 82
2.21.2 Setzen von Ausgängen und Beschriftung der E/As . . . 82
2.22 System-Tools . . . 83
2.22.1 System . . . 83
2.22.2 Mikado . . . 84
2.22.3 Entwickler-Tools . . . 84
2.22.4 Profiling . . . 84

3 Tasks **87**
3.1 Ablauf . . . 87
3.1.1 Zu Sequenz wechseln . . . 87
3.1.2 Wenn-Gruppe . . . 88
3.1.3 Programm beenden . . . 89
3.1.4 Fehler . . . 89
3.1.5 Warten . . . 89
3.1.6 Warten bis zum Stillstand . . . 89
3.1.7 Programm unterbrechen . . . 90
3.2 Zahlen . . . 90
3.2.1 Zahl erhöhen . . . 90
3.2.2 Zahl verringern . . . 90
3.2.3 Arithmetische Operation . . . 91
3.2.4 Logische Operation . . . 91
3.2.5 Vergleichsoperation . . . 94
3.3 Grundlegendes . . . 95
3.3.1 Kopieren . . . 95
3.3.2 Werte tauschen . . . 95
3.3.3 Variable zurücksetzen . . . 95
3.3.4 Alle Variablen zurücksetzen . . . 96

3.3.5 Ausgangswert setzen . . . 96
3.3.6 Negiere Boolsche Variable . . . 96
3.3.7 Kopiere Eintrag aus Liste . . . 96
3.3.8 Kopiere nächsten Eintrag aus Liste . . . 97
3.3.9 Zur Liste hinzufügen . . . 97
3.3.10 Setze Listenelement . . . 97
3.3.11 Lösche Element aus Liste . . . 98
3.3.12 Liste leeren . . . 98
3.3.13 Liste leer? . . . 98
3.3.14 Zähle Listenelemente . . . 99
3.4 Roboterkontrolle . . . 99
3.4.1 Bewege Roboter nach . . . 99
3.4.2 Bewege Roboter kartesisch nach . . . 100
3.4.3 Bewege Roboter relativ . . . 101
3.4.4 Bewege Roboter kartesisch relativ . . . 101
3.4.5 Bewege Tcp nach . . . 102
3.4.6 Bewege Tcp kartesisch nach . . . 103
3.4.7 Bewege Tcp relativ . . . 104
3.4.8 Bewege Tcp kartesisch relativ . . . 105
3.4.9 Bewege Teil nach . . . 106
3.4.10 Bewege Teil kartesisch nach . . . 107
3.4.11 Bewege Teil relativ . . . 107
3.4.12 Bewege Teil kartesisch relativ . . . 108
3.4.13 Roboterposition auslesen . . . 109
3.4.14 Teileposition auslesen . . . 110
3.4.15 TCP Position auslesen . . . 110
3.4.16 Ist Roboter Position mit Teil erreichbar? . . . 110
3.4.17 Ist Roboter Position erreichbar? . . . 111
3.4.18 Hand öffnen/schließen . . . 111
3.4.19 Greiferzustand einstellen . . . 112
3.4.20 Setze aktiven Greifer . . . 114
3.4.21 Roboterprogramm ausführen . . . 114
3.4.22 Roboterausgangssignal setzen . . . 115
3.4.23 Roboterausgangssignal lesen . . . 116
3.4.24 Robotereingangssignal lesen . . . 116
3.4.25 Auf Robotereingangssignal warten . . . 116
3.4.26 Auf Roboterausgangssignal warten . . . 117
3.4.27 Auf Feldbuseingangssignal warten . . . 117
3.4.28 Feldbusausgangssignal asynchron setzen . . . 118
3.5 Bildverarbeitung . . . 119
3.5.1 Setze aktive Kamera . . . 119
3.5.2 Kamera Konfiguration laden . . . 119
3.5.3 Kamera Kalibrierung laden . . . 120
3.5.4 Punktwolke aufnehmen . . . 120
3.5.5 Punktwolke asynchron aufnehmen . . . 120
3.5.6 Mehrfachaufnahme . . . 121
3.5.7 Kamerapose nachjustieren . . . 121
3.5.8 Anzahl der Punkte ermitteln . . . 122
3.5.9 Bild speichern . . . 124
3.5.10 Setze aktive ROI . . . 124
3.5.11 Patternpose bestimmen . . . 125
3.6 Kalibrierung . . . 125
3.6.1 Starte Kalibrierung . . . 125
3.6.2 Aufnahme für Kalibrierung . . . 126
3.6.3 Kalibrierung abschließen . . . 126
3.6.4 Hand-Auge-Kalibrierung validieren . . . 127
3.7 Szenenobjekt . . . 130
3.7.1 Setze aktive Kiste . . . 130

3.7.2 Suche Szenenobjekt . . . 131
3.7.3 Szenenobjekt-Position auslesen . . . 132
3.7.4 Szenenobjekt-Position setzen . . . 132
3.7.5 Szenenobjekt-Position zurücksetzen . . . 133
3.8 Objekt-Handhabung . . . 133
3.8.1 Setze aktives Produkt . . . 133
3.8.2 Griffsuche starten . . . 133
3.8.3 Griffsuche abbrechen . . . 134
3.8.4 Finde Objekt . . . 135
3.8.5 Griffe auslesen . . . 135
3.8.6 Teil greifen . . . 137
3.8.7 Griffsuchen-Statistik . . . 140
3.8.8 Teil ablegen . . . 142
3.8.9 Lösche Funde und Punktwolke . . . 142
3.8.10 Statisches Objekt hinzufügen . . . 142
3.8.11 Statisches Objekt entfernen . . . 143
3.8.12 Statisches Objekt anhängen . . . 143
3.8.13 Produktabmessungen auslesen . . . 144
3.8.14 Passe Objektpose an . . . 145
3.9 Transformationen . . . 145
3.9.1 Pose auftrennen . . . 145
3.9.2 Pose erstellen . . . 146
3.9.3 Rotiere Pose Global . . . 147
3.9.4 Rotiere Pose Lokal . . . 149
3.9.5 Verschiebe Pose Global . . . 151
3.9.6 Verschiebe Pose Lokal . . . 152
3.9.7 Pose transformieren . . . 154
3.9.8 Inverse Transformation . . . 154
3.10 Erweiterungen . . . 154
3.10.1 JavaScript . . . 154
3.10.2 Halcon Script ausführen . . . 155

4 Tutorials **157**
4.1 Ein erster Ablauf in zehn Minuten . . . 157
4.1.1 Aufnahmepose einrichten . . . 157
4.1.2 Szene einrichten . . . 157
4.1.3 Produkt einrichten . . . 158
4.1.4 Produkt speichern . . . 158
4.1.5 Minimalprogramm erstellen . . . 158
4.1.6 Posen einlernen und Variablen anpassen . . . 159
4.1.7 Laufen lassen und etwas variieren . . . 159
4.2 Ein neues Produkt bis zur Objekterkennung . . . 160
4.2.1 Neues Produkt / Produktliste einrichten . . . 160
4.2.2 Objekterkennung optimieren . . . 162
4.3 Meshlab . . . 164
4.3.1 Grundlagen: STL laden und speichern . . . 164
4.3.2 Produktkoordinatensystem zentrieren . . . 165
4.3.3 STL skalieren . . . 167
4.3.4 STL Anzahl der Dreiecke reduzieren . . . 169
4.3.5 STL Flächen entfernen . . . 171
4.3.6 STL Innenliegende Flächen automatisch entfernen . . . 176
4.4 Halcon Script einbinden: Datamatrix Erkennung . . . 181
4.4.1 Vorbereitung . . . 181
4.4.2 Programmablauf . . . 182
4.4.3 Hdev-Script für eine Erkennung von Datamatrizen / QR-Codes erstellen . . . 183
4.4.4 Programm ausführen und testen . . . 185
4.5 Beispiel: Benutzerdefinierte Logdatei aus Mikado . . . 188
4.5.1 Passendes Mikado Programm . . . 188

4.5.2 Variablen des Mikado Programms . . . 189
4.5.3 Skript Details . . . 189
4.6 Beispiel: Mikado PoseList sortieren . . . 190

5 Simulation **191**
5.1 Roboter-Simulation . . . 191
5.2 Aufgezeichnete Bilddaten (Filecam) . . . 191
5.3 Simulation basierend auf Mesh-Dateien . . . 192
5.3.1 Simulator - Konfiguration . . . 192
5.3.2 Simulator - Automatischer Modus . . . 193
5.3.3 Simulator - Anpassung der Kameraeinstellungen mit Szene aus Simulator . . . 195

6 Roboter-Modelle **201**
6.1 ABB Controller einrichten . . . 201
6.1.1 Systemanforderungen . . . 202
6.1.2 Netzwerkkonfiguration . . . 202
6.1.3 Installation der Programme . . . 202
6.1.4 Einrichtung der Programme . . . 204
6.1.5 Updaten der Roboterprogramme . . . 206
6.1.6 Roboter steuern . . . 206
6.1.7 Greifer und IOs einrichten mit Robot Web Services . . . 207
6.1.8 Kontrollabgabe . . . 208
6.1.9 Troubleshooting . . . 210
6.2 Doosan Controller einrichten . . . 211
6.2.1 Netzwerksetup . . . 211
6.2.2 Allgemeine Hinweise . . . 211
6.2.3 Moduswechsel Automatik . . . 211
6.2.4 Name des aktuellen Tools abfragen . . . 212
6.2.5 Name des aktuellen TCP abfragen . . . 212
6.2.6 Quellen . . . 212
6.3 Fanuc Controller einrichten . . . 212
6.3.1 Voraussetzungen . . . 212
6.3.2 Netzwerkkonfiguration . . . 213
6.3.3 Mikado-Schnittstelle installieren . . . 213
6.3.4 Mikado-Schnittstelle parametrieren . . . 214
6.3.5 System starten . . . 214
6.3.6 Optional: Greifer-Ansteuerung über I/Os . . . 214
6.3.7 Optional: Natives Roboterprogramm ausführen . . . 215
6.3.8 Troubleshooting: Verbindung testen . . . 216
6.4 KUKA Controller einrichten . . . 216
6.4.1 Systemanforderungen . . . 216
6.4.2 Netzwerkkonfiguration . . . 216
6.4.3 KRC vorbereiten . . . 218
6.4.4 Optional: Natives Roboterprogramm ausführen . . . 219
6.4.5 Roboterkontrolle übergeben . . . 220
6.4.6 Geschwindigkeit und Fine-Tuning . . . 220
6.4.7 Troubleshooting . . . 221
6.5 Mitsubishi Controller einrichten . . . 221
6.5.1 Netzwerkkonfiguration . . . 221
6.5.2 Verbindung konfigurieren . . . 227
6.5.3 Programme konfigurieren . . . 229
6.5.4 Troubleshooting: Verbindung testen . . . 233
6.5.5 Optional: Handansteuerung . . . 235
6.5.6 Optional: Basiskoordinatensystem anpassen . . . 236
6.6 Mitsubishi Roboter verwenden . . . 237
6.6.1 Parameter und Steuerelemente . . . 237
6.6.2 Roboter einschalten . . . 238
6.6.3 Sonstige Hinweise . . . 238

6.7 Nachi Controller einrichten . . . 238
6.7.1 Netzwerkkonfiguration . . . 238
6.7.2 Vorbereitung . . . 239
6.7.3 Kopieren der Programme/Usertasks . . . 241
6.7.4 Kompilieren der Programme/Usertasks . . . 241
6.7.5 Einstellen der Priorität der Usertasks . . . 241
6.7.6 Starten des Roboterprogramms . . . 242
6.8 Stäubli Controller einrichten . . . 242
6.8.1 Netzwerkkonfiguration . . . 242
6.8.2 Installation . . . 242
6.8.3 Serversockets einrichten . . . 243
6.8.4 Starten . . . 243
6.8.5 Greifer und IOs einrichten . . . 243
6.9 Techman Controller einrichten . . . 243
6.9.1 Netzwerkkonfiguration . . . 243
6.9.2 Modbus Verbindung herstellen . . . 244
6.9.3 Projekt erstellen . . . 244
6.9.4 Projekt testen . . . 245
6.9.5 Projektgeschwindigkeit einstellen und Automatik-Modus aktivieren . . . 246
6.9.6 Troubleshooting . . . 247
6.9.7 Techman Variablen schreiben und Programme ausführen . . . 247
6.10 Universal Robots Controller einrichten . . . 247
6.10.1 Systemanforderungen . . . 247
6.10.2 Netzwerkkonfiguration . . . 248
6.10.3 Kommunikationsprogramm einrichten . . . 248
6.10.4 Autostart einrichten . . . 250
6.10.5 Gelenkwinkelgrenzen . . . 251
6.10.6 Geschwindigkeit und Beschleunigung . . . 251
6.10.7 Roboter-Kalibrierdaten auslesen und verwenden . . . 251
6.10.8 Optional: Natives Roboterprogramm ausführen . . . 252
6.11 UR Roboter verwenden . . . 254
6.11.1 Roboter einschalten . . . 254
6.11.2 URCap Gripper verwenden . . . 255
6.11.3 Parameter und Steuerelemente . . . 257
6.11.4 Sonstige Hinweise . . . 257
6.11.5 Troubleshooting Universal Robot . . . 257
6.12 Yaskawa Motoman Controller einrichten . . . 260
6.12.1 Vorbereitung . . . 260
6.12.2 Netzwerkkonfiguration . . . 261
6.12.3 Einrichtung des Roboter Controllers . . . 261
6.12.4 Weiteres & Troubleshooting . . . 264
6.12.5 Beschleunigungslimits in joint_limits.yaml eintragen . . . 265
6.12.6 Fehlercodes / Alarme . . . 265
6.12.7 Kontrolle über den Roboter vorübergehend abgeben . . . 266
6.12.8 Andere Jobs auf dem Roboter ausführen . . . 267
6.12.9 Yaskawa 16-Bit Register lesen und schreiben . . . 267

7 Installation **269**
7.1 Übersicht . . . 269
7.1.1 Hardware-Übersicht . . . 269
7.1.2 Software-Übersicht . . . 270
7.1.3 Integrationsablauf (Schritt-für-Schritt Anleitung) . . . 271
7.2 Konfigurations-Backup . . . 272
7.3 Erstellen einer neuen Zelle . . . 272
7.3.1 Neue Mikado Control Zelle Erstellen . . . 272
7.4 Anpassen einer vorhandenen Zelle . . . 273
7.4.1 Geschwindigkeitseinstellungen . . . 273
7.4.2 Konfigurationsdateien . . . 276

7.4.3 Launchfiles . . . 284
7.4.4 Zellgeometrie . . . 285
7.5 Verzeichnisstruktur . . . 295
7.6 Mikado Select / Control Montageanleitung . . . 296
7.6.1 Mikado Controller Hardware . . . 296
7.6.2 Anschluss der Ensenso Kameras . . . 296
7.6.3 Befestigung der Kalibrierplatte . . . 297
7.7 Greifer-Integration . . . 298
7.7.1 Greifertypen . . . 299
7.7.2 Greifer einbinden . . . 299
7.7.3 Mikado-Konfiguration . . . 303
7.7.4 Greifer Anschlussvarianten . . . 303
7.7.5 TCP kalibrieren . . . 303
7.8 Hardware einrichten . . . 305
7.8.1 Übersicht Anschlüsse/Verkabelung . . . 305
7.8.2 Netzwerkadapter . . . 307
7.8.3 Verbindungstest . . . 308
7.8.4 Ensenso-Kameras einrichten . . . 309
7.8.5 Roboter-Controller einrichten . . . 313
7.9 Feldbus-Schnittstelle . . . 313
7.9.1 Kurzbeschreibung der Schnittstelle . . . 313
7.9.2 Genereller Ablauf der Kommunikation . . . 313
7.9.3 Status der Kommunikation . . . 315
7.9.4 Synchronisieren von SPS-Variablen und Mikado-Variablen . . . 316
7.9.5 Rotations-Konvention . . . 317
7.9.6 Fehler-Codes . . . 318
7.9.7 Kommandos . . . 320
7.9.8 Beispielablauf für Select-Zellen . . . 323
7.9.9 Aktualisierungen der Feldbus-Schnittstelle . . . 324
7.9.10 Standard-Feldbus-Schnittstelle . . . 325
7.9.11 Spezifische Konfiguration für Profinet . . . 327
7.9.12 Spezifische Konfiguration für EtherCAT . . . 327
7.9.13 Spezifische Konfiguration für EtherNet/IP . . . 328
7.9.14 Spezifische Konfiguration für Beckhoff ADS-Schnittstelle . . . 328
7.10 Informationen zum Betriebssystem . . . 330

8 Mikado Control erweitern **331**
8.1 JavaScript Task . . . 331
8.1.1 Allgemeines . . . 332
8.1.2 `task` . . . 332
8.1.3 `dataStore` . . . 332
8.1.4 `picksearch` . . . 333
8.1.5 `kinematics` . . . 338
8.1.6 `product` . . . 341
8.1.7 `robot` . . . 343
8.1.8 `scene` . . . 346
8.1.9 `file` . . . 347
8.1.10 `program` . . . 347
8.1.11 `lab` . . . 348
8.1.12 `pointCloud` . . . 359
8.2 Halcon Script Task . . . 361
8.2.1 mikado_get_2d_image . . . 361
8.2.2 mikado_publish_2d_image . . . 362
8.2.3 mikado_get_point_cloud . . . 362
8.2.4 mikado_write_variable . . . 362
8.2.5 mikado_read_variable . . . 363
8.2.6 ensenso_set_get_json . . . 364
8.3 Benutzerdefinierte Objekterkennung mit Halcon . . . 366

8.3.1 Hdev-Skript . . . 366
8.3.2 Anpassen der Konfigurationsdatei . . . 367
8.4 Eigene Seiten mit QML . . . 368
8.4.1 Konfiguration . . . 368
8.4.2 Einführung in QML . . . 368
8.4.3 Weiterführende Informationen . . . 370
8.5 Externe Ansteuerung mit ROS/Python . . . 370
8.5.1 Beispielprogramm . . . 370
8.5.2 Mikado ROS Services . . . 371
8.5.3 Mikado ROS Actions . . . 377
8.5.4 Systemstatus . . . 379
8.5.5 Eigenen ROS Service erstellen . . . 380
8.5.6 Launchfile . . . 381

9 Aktualisierungen **383**
9.1 Mikado . . . 383
9.1.1 Version 3.4.0 . . . 383
9.1.2 Version 3.3.0 . . . 386
9.1.3 Version 3.2.0 . . . 394
9.1.4 Version 3.1.2 . . . 396
9.1.5 Version 3.0.5 . . . 400
9.1.6 Version 3.0.4 . . . 400
9.1.7 Version 3.0.3 . . . 401
9.1.8 Version 3.0.2 . . . 401
9.1.9 Version 3.0.1 . . . 401
9.1.10 Version 3.0.0 . . . 401
9.1.11 Version 2.1.0 . . . 406
9.2 Szenen-Objekte . . . 407
9.2.1 Version 5 . . . 407
9.2.2 Version 4 . . . 408
9.2.3 Version 3 . . . 408
9.2.4 Version 2 . . . 408

10 Änderungshistorie **409**

11 Copyright **411**
11.1 GNU LESSER GENERAL PUBLIC LICENSE . . . 411
11.2 BSD 3-Clause License . . . 413

KAPITEL 1

Einleitung

Mikado Control ist eine grafische Software für den Griff in die Kiste (Bin Picking). Sie bündelt Lösungen für alle Teilprobleme von Bildaufnahme und Objekterkennung über Pfadplanung und Robotersteuerung bis hin zur Kommunikation mit externen Steuerungen. Trotz dieser teils komplexen Aufgabengebiete ermöglicht Ihnen Mikado Control einen intuitiven und bequem konfigurierbaren Zugang zu Ihrer Bin-Picking-Applikation. Die Verwendung als reines Sensorsystem zur Lagebestimmung z.B. zur Depalettierung ist ebenfalls ein Anwendungsgebiet von Mikado und wird als Mikado Select vertrieben.

Diese Dokumentation hilft Ihnen den Mikado Controller zu bedienen und die Funktionsweise zu verstehen. Für die Verwendung eines bestehenden Systems sind die Kapitel *Verwendung*, *Tasks* und *Simulation* wichtig, sowie die praktischen Übungen aus dem Kapitel *Tutorials*.

Durch die Einrichtung einer neuen Zelle führt Sie das Kapitel *Installation*. Der Abschnitt *Integrationsablauf (Schritt-für-Schritt Anleitung)* gibt eine Schritt-für-Schritt Übersicht über die Einrichtungsschritte. Die Montage der Hardwarekomponenten wird im Abschnitt *Mikado Select / Control Montageanleitung* beschrieben.

Die Einrichtung der unterstützen Robotermodelle wird im Bereich *Roboter-Modelle* herstellerspezifisch beschrieben. Die Sektion *Mikado Control erweitern* zeigt die verfügbaren Schnittstellen und Erweiterungsmöglichkeiten von Mikado auf.

1.1 Schnellstart

Der schnellste Weg in die Mikadooberfläche ist eine der mitgelieferten Zellen mit simulierter Roboterbewegung und gespeicherten oder simulierten Kameraaufnahmen zu benutzen. Starten Sie hierzu Ihren Controller und stellen Sie sicher, dass die Lizenzdongles eingesteckt sind. Auf dem Desktop liegen dann bereits eine Verknüpfung um eine der vorgefertigten Zellen im Mikado Programm zu öffnen, z.B. `motek_demo_2017`, und daneben eine Verknüpfung für die zugehörige Kamerasimulation. Starten Sie zuerst Mikado und dann den Simulator. Sie sehen nun eine fertig eingerichtete Zelle mit Szene, Produkt und Programm, in der Sie erste Schritte üben können.

1.2 Die verschiedenen Werkzeuge

Der Mikado Controller wird mit verschiedenen Software-Werkzeugen ausgeliefert. Dies ist das Kern Mikado Programm, das für jede Zelldefinition auf dem Controller einmal auf dem Desktop verknüpft ist. Für jede neu erstellte Zelle werden diese Verknüpfungen automatisch angelegt. Geht ein Controller in den Dauerbetrieb kann Mikado mit einer ausgewählten Zelldefinition per Autostart direkt ausgeführt werden.

Neben der Mikado Verknüpfung wird ebenso für jede Zelldefinition eine Simulator Verknüpfung angelegt, Näheres hierzu im Kapitel *Simulation*.

Zum Anlegen einer neuen Zelle benutzen Sie den Mikado Assistant. Diesen finden Sie in der Taskleiste oder im Startmenü nach Eingabe von `Assistant`. Die genauen Schritte sind im Kapitel *Erstellen einer neuen Zelle* dokumentiert. Mit dem Assistant können Sie ebenfalls bestehende Zellen tiefgreifend anpassen. Die verschiedenen Möglichkeiten finden Sie im Kapitel *Anpassen einer vorhandenen Zelle*.

Das zum Konfigurieren der Ensenso Kamera mitgelieferte Program NxView können Sie über die Taskleiste oder das Startmenü ausführen. Bei korrekter Netzwerkkonfiguration werden Ihnen hier alle erreichbaren Ensensokameras aufgeführt. Bitte beachten Sie, dass immer nur eine Anwendung Zugriff auf eine Kamera hat, also entweder Mikado oder NxView. Die Bedienungsanleitung für die Kamera finden Sie im Ensenso Manual.

KAPITEL 2

Verwendung

In diesem Kapitel wird erläutert wie Mikado Control verwendet wird und eigene Abläufe erstellt werden. Zudem werden alle Werkzeuge der Mikado-Oberfläche erläutert.

2.1 Navigation

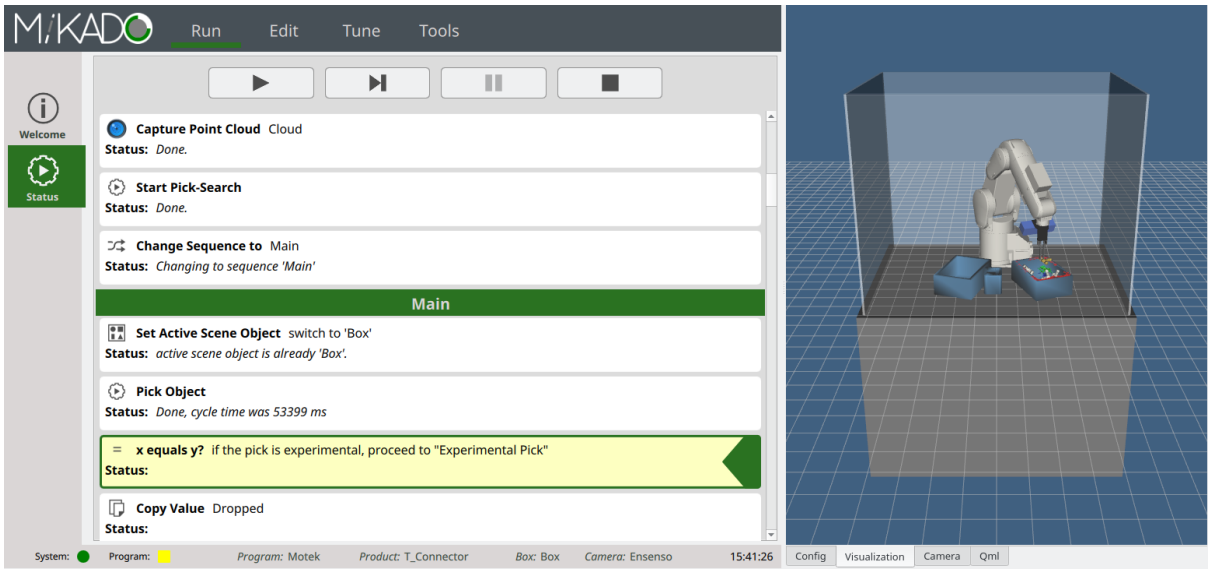


Abb. 1: Mikado-Oberfläche mit Statusseite

Die Benutzeroberfläche ist in fünf Bereiche eingeteilt:

- **Oben:** ***Haupt-Navigationsleiste***
  Zeigt alle Hauptkategorien an, die wiederum die einzelnen Navigationsseiten beinhalten. Klicken Sie auf eine der Kategorien und die Seiten der Kategorie werden ausgeklappt. Wählen Sie nun die gewünschte Seite an, um auf diese zu wechseln. Folgende Hauptkategorien stehen Ihnen zur Verfügung:
  - *Run*: Beinhaltet Seiten, die während des Betriebs relevant sind (Statusüberwachung und individuelle Bediener-Seiten).

- *Edit*: Ermöglicht die Einrichtung der Abläufe und Verwaltung von Produkten und Szene.
- *Tune*: Dient der Verbesserung der Objekterkennung, der Griffdefinitionen oder der Performance.
- *Tools*: Sonstige Werkzeuge wie Logs, Robotersteuerung, Kamera Tools, etc.

Falls Ihr System einen Passwortschutz erfordert, erscheint am rechten Rand der Haupt-Navigationsleiste ein Schloss-Symbol, mit dem Sie sich ein- und ausloggen können, siehe Kapitel *Passwortschutz*.

Hinweis: Mit den Funktionstasten F1 - F12 können Sie zwischen den Seiten direkt wechseln (F1: Status, F2: Programmeditor, ...)

- **Links:** ***Navigationsseiten***
  Zeigt alle Navigationsseiten der aktuellen Haupt-Kategorie an. Hier können Sie schnell innerhalb der Kategorie zu einer anderen Seite wechseln.
- **Mitte:** ***Hauptfenster***
  Inhalte der aktuell ausgewählten Seite werden in diesem Hauptbereich von Mikado Control angezeigt.
- **Rechts:** ***Visualisierung***
  Darstellung von Roboter und Zelle. In dieser Ansicht weren ebenso Inhalte in Abhängigkeit von der aktuell gewählten Seite angezeigt (z.B. Produkt, Griffe, Punktwolke, Funde, etc). In der 3D-Ansicht innerhalb der Visualisierung kann mit der Maus navigiert werden:
  - Linke Maustaste: Rotieren
  - Rechte Maustaste: Zoom
  - Mittlere Maustaste: Bewegen

  Während die entsprechende Maustaste gedrückt wird, kann die Ansicht mittels Mausbewegungen verändert werden. Während die Maustaste gedrückt gehalten wird, ist in der Visualisierungs-Szene eine gelb-transparente Scheibe zu sehen, die den aktuellen Zoom- bzw. Rotations-Mittelpunkt darstellt.

  Unterhalb der Visualisierung sind verschiedene Reiter auswählbar. Im ‚Config' Tab kann die Visualisierung bei Bedarf angepasst werden. Beispielsweise lassen sich hier Punktwolken vergrößern oder einfärben. Außerdem sind im unteren Bereich des ‚Config'-Tabs Buttons zum Speichern und Zurücksetzen der aktuellen Ansicht zu finden. Beim Speichern wird lediglich die aktuelle Ansicht gesichert (z.B. Statusansicht der Szene oder Produktansicht, je nachdem, auf welcher Mikado-Seite Sie sich momentan befinden). Die gespeicherte Ansicht wird beim Start von Mikado oder durch den Button ‚Zurücksetzen' geladen.

  Der ‚Kamera' Tab zeigt die letzte Bildaufnahme der Kamera(s) (Details zu den Kamera-Einstellungen unter *Kamera-Einstellungen*).
- **Unten:** ***Statusleiste***
  Informationen über den aktuellen Zustand des Programmablaufs (s. *Statusleiste*).

2.2 Startseite

Auf der Willkommensseite von Mikado Control können Sie Ihre Produkte und Programmabläufe über einen Schnellzugriff laden. Hierzu genügt ein Klick auf den entsprechenden Listeneintrag.

Neue Produkte oder Programme, die Sie in Mikado Control anlegen, werden beim nächsten Start der Mikado-Software automatisch in die Listen aufgenommen. Hierzu müssen Sie das Produkt/Programm lediglich in das Standardverzeichnis für Produkte/Programme abspeichern.

Bemerkung: Das Standardverzeichnis wird beim Speichern eines Produkts/Programms vorausgewählt. Nähere Informationen zu der Verzeichnisstruktur ist im Abschnitt *Verzeichnisstruktur* zu finden.

2.3 Bediener-Seite

Die Seite ‚Bediener' dient dazu, individuelle bzw. projektspezifische Anforderungen an die Mikado-Oberfläche umzusetzen. Steuerelemente wie z.B. Eingabefelder und Buttons, die für eine einzelne Anwendung notwendig oder hilfreich sind, werden auf dieser Seite angezeigt. Wird keine Bediener-Seite konfiguriert, so wird diese auch nicht in der Navigation angezeigt. Ansonsten ist sie unterhalb der Willkommensseite im Menü ‚Run' zu finden.

2.4 Einrichtungs-Seite

Die Seite ‚Einrichtung' kann individuell gestaltet werden. Die Dokumentation zur *Bediener-Seite* gilt entsprechend für die Einrichtungs-Seite.

Ist der Passwortschutz aktiviert, muss diese Seite vor dem Zugriff freigeschaltet werden. Hier können also beispielsweise Einstellmöglichkeiten für applikationsspezifische Parameter einem eingeschränkten Benutzerkreis angeboten werden.

2.5 Statusseite

Die Statusseite stellt die zentrale Ansicht für einen Bediener dar. Der Programmablauf kann hier gesteuert werden und es wird der aktuelle Zustand des ausgeführten Tasks dargestellt. (Screenshot der Mikado-Oberfläche mit Statusseite s.o.).

2.5.1 Steuerbuttons

Für die Programmablaufs-Steuerung können folgende Buttons verwendet werden:

- ***Start***
  Programmablauf starten. Falls das Programm pausiert war, wird die Ausführung an der pausierten Stelle fortgesetzt
- ***Nächsten Task ausführen***
  Der nächste Schritt wird ausgeführt, danach pausiert der Programmablauf. So können Sie den Ablauf Schritt für Schritt ausführen, um z.B. ein neues Programm zu testen.
- ***Pause***
  Der Programmablauf wird angehalten, der Programmzustand bleibt jedoch erhalten und kann fortgesetzt werden.
- ***Stopp und Reset***
  Stoppt den Programmablauf und setzt den Programmzustand zurück. Alle Variablen werden auf den Initialzustand zurückgesetzt und der Programmcursor wird auf den ersten Task gesetzt.



Abb. 2: Steuerbuttons (von links: Start, Nächster Task, Pause, Stopp)

Bemerkung: Buttons sind deaktiviert, um Fehleingaben zu vermeiden. Der Pause-Button kann zum Beispiel nicht betätigt werden, wenn der Programmablauf noch nicht gestartet wurde. Ist ein Button deaktiviert, wird das Symbol hellgrau dargestellt.

2.5.2 Programmablauf

Unter den Steuerbuttons ist der zentrale Teil der Statusübersicht zu sehen. Hier werden alle Sequenzen und deren Tasks des aktuellen Mikado-Programms angezeigt. Sequenzen sind als grüne Kästen mit Sequenznamen eingezeichnet und gruppieren unter sich die Tasks (weiße Kästen).

Einen ggf. vergebenen Benutzerkommentar finden Sie neben dem Tasks-Typ. Wenn ein Task bereits ausgeführt wurde, sehen Sie im Feld ‚Status' das Ergebnis der letzten Ausführung dieses Tasks. Wird der Task aktuell ausgeführt, färbt sich der dargestellte Kasten hellgrün und das Feld ‚Status' zeigt den Status der Ausführung an.

Wurde die Ausführung des Programms pausiert, so wird der folgende Task, der beim Fortsetzen des Programms als nächstes ausgeführt wird, mit dem Programmcursor markiert und gelb eingefärbt. Der Programmcursor wird als grüner Rahmen dargestellt.

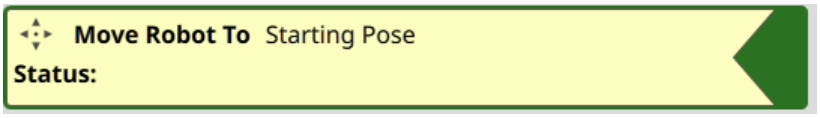


Abb. 3: Programmcursor

Im Fehlerfall bricht der Programmablauf ab und der Task, der den Fehler gemeldet hat wird rot oder orange eingefärbt. In diesem Fall ist im Feld ‚Status' eine Fehlernachricht zu sehen. Einige Tasks unterstützen die Möglichkeit einen erneuten Versuch auszuführen. In diesem Fall wird der Task nicht rot sondern orange eingefärbt. Beispiele hierfür sind die Bewege-Tasks in *Roboterkontrolle* oder *Punktwolke aufnehmen*. Um weitere Informationen zu einem Fehler zu erhalten, empfiehlt es sich einen Blick in die *Logs* zu werfen.

Task Visualisierung

Die farbigen Darstellungen eines Tasks in der Statusseite haben folgende Bedeutungen:

- *weiß*

  Inaktiv. Task wurde noch nicht oder erfolgreich ausgeführt.

- *grün*

  In Ausführung.

- *gelb*

  Dieser Task wird bei Fortsetzen des Ablaufs als nächstes ausgeführt.

- *orange*

  Bei der Ausführung des Tasks ist ein Fehler aufgetreten, ein erneuter Versuch ist aber möglich. Hierzu den Ablauf fortsetzen.

- *rot*

  Ein Fehler ist aufgetreten. Der Ablauf muss gestoppt und neu gestartet werden um fortzusetzen.

2.6 Statusleiste

Die Statusleiste ist ein fester Bestandteil der Mikado-Oberfläche und auf jeder Ansicht im unteren Bereich eingeblendet.

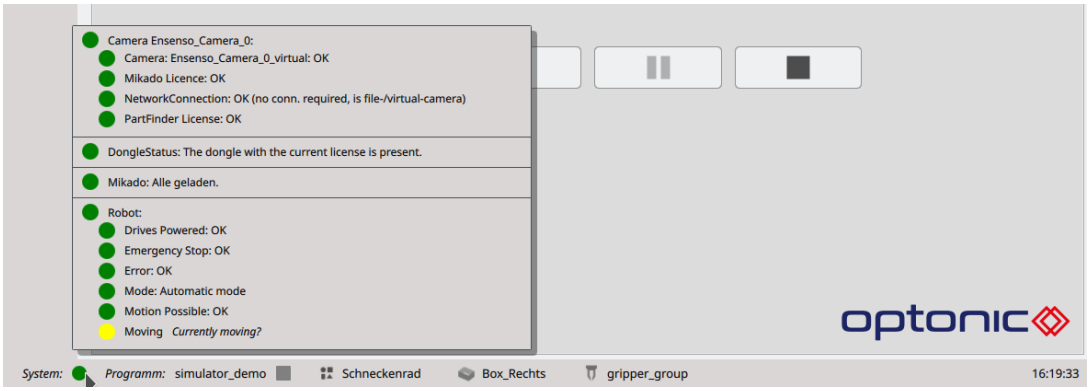


Abb. 4: Statusleiste und Systemstatus

Folgende Anzeigen sind in der Statusleiste zu sehen (von links):

- **Systemstatus**
  Siehe *Systemstatus*.
- **Programm**
  Geladenes Mikado-Programm. Dieses kann *Programm-Editor* modifiziert und verwaltet werden.
- **Produkt**
  Aktives Produkt. Das aktive Produkt kann über die *Produktverwaltung* verändert und gesetzt werden (siehe *Aktives Produkt*).
- **Kiste**
  Aktive Kiste. Die aktive Kiste kann sich während der Programmausführung verändern (siehe *Setze aktive Kiste*). In diesem Bereich der Statusleiste sehen Sie die aktuell aktive Kiste.
- **Kamera**
  Aktive Kamera. Die aktive Kamera kann sich während der Programmausführung verändern (siehe *Setze aktive Kamera*). In diesem Bereich der Statusleiste sehen Sie die aktuell aktive Kamera.
- **Uhrzeit**
  Die aktuelle Uhrzeit.

2.6.1 Systemstatus

Der Systemstatus dient als Indikator der allgemeinen Betriebsbereitschaft von Mikado. Wenn alle konfigurierten Komponenten des Systems wie z.B. **Roboter**, **Kamerasysteme**, **Lizenzdongles** oder **Feldbussysteme** ordnungsgemäß funktionieren, so ist der Statusindikator **grün** gefärbt und Mikado gilt als betriebsbereit. Erkennt Mikado Control jedoch einen unzulässigen Zustand im System, so wird der Statusindikator **rot** und Ihr Programmablauf wird unter Umständen nicht korrekt funktionieren.

Möchten Sie eine detailliertere Aufschlüsselung der Zustände der einzelnen Komponenten sehen, bewegen Sie Ihren Mauszeiger über den Statusindikator. Nun erscheint eine Übersicht der einzelnen Komponenten mit jeweils eigenem Statusindikator und ggf. Anmerkungen.

2.7 Passwortschutz

Für Zellen mit aktiviertem Passwortschutz werden alle Gui-Elemente, die Änderungen an der Zellmodellierung oder Ablaufsteuerung erlauben würden, nur nach Eingabe des richtigen Passworts freigegeben.

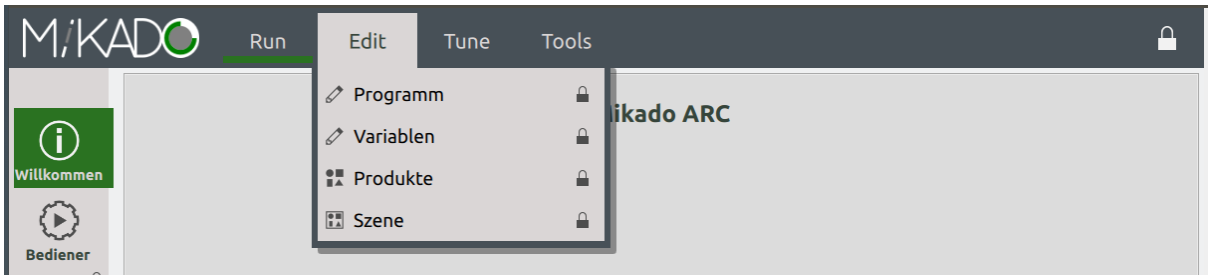


Abb. 5: Screenshot Menü mit aktiviertem Passwortschutz

Im Menü werden nicht freigegebene Bereiche durch kleine Schlosssymbole gekennzeichnet. Zum Freischalten klickt man auf das große Schlosssymbol in der Menüleiste oben rechts und gibt im folgenden Dialog das Passwort ein.

Das Aktivieren und Einstellen des Passworts wird unter *access_control.yaml* beschrieben.

2.8 Programm-Editor

Der Programm-Editor bietet die nötigen Werkzeuge, um einen Mikado-Programmablauf zusammenzustellen. Er ist über das Hauptmenü unter ‚Edit' → ‚Programm' zu finden.

2.8.1 Mikado-Programme

Ein Mikado-Programm besteht aus Sequenzen, Tasks und Variablen. Diese bilden gemeinsam ein ausführbares Programm, das Aufgaben wie z.B. den Griff in eine Kiste realisiert. Unabhängig vom Programm werden Produkte und Szenenobjekte verwaltet, die im Programm verwendet werden können.

Sequenzen

Ein Mikado-Programmablauf findet in verschiedenen Sequenzen statt. Das Programm startet mit der ersten Sequenz und wiederholt diese immer wieder. Ein Sequenzwechsel oder -übergang findet dann statt, wenn ein Task dies veranlasst. Ein Beispiel hierfür ist der Task ‚*Zu Sequenz wechseln*', der die Ausführung der aktiven Sequenz stoppt und zu einer anderen Sequenz wechselt. Ein weiteres Beispiel für einen Sequenzwechsel sind Tasks, die bei bestimmten Bedingungen zu einer anderen Sequenz springen.

Sequenzen werden mithilfe des Programm-Editors erstellt und editiert. Sie müssen eindeutig benannt werden.

Tasks

Tasks sind die Grundbausteine eines Mikado-Programms und immer Teil einer Sequenz. Startet die Ausführung einer Sequenz, so wird mit dem ersten Task begonnen. Danach werden die Tasks der Reihe nach abgearbeitet, wobei jeder Task eine Aufgabe erfüllt. Tasks werden mittels Variablen parametriert. Ebenso können Variablen von einem Task verändert werden.

Weiterhin kann ein Task Sequenzwechsel besitzen. Diese werden unterhalb eines Trennstrichs aufgelistet. Tritt eine entsprechende Bedingung ein, so kann der Task somit einen Sequenzwechsel auslösen.

Tasks werden mithilfe des Programm-Editors zu Sequenzen hinzugefügt und editiert. Sowohl Variablen wie auch Sequenzwechsel werden über die Oberfläche konfiguriert.

Alle verfügbaren Tasks sind in Kapitel *Tasks* aufgelistet und erläutert.

Hinweis: Sowohl Variablen als auch Sequenzwechsel sind nicht bei jedem Task zwingend erforderlich.

Variablen

Variablen stellen den dynamischen Teil des Mikado-Programms dar. Sie werden von Tasks eingelesen oder auch verändert. Wird ein Programm neu gestartet, so werden alle Variablenwerte auf den definierten Initialwert zurückgesetzt, während der Ausführung werden viele Variablenwerte jedoch von den Tasks modifiziert. Weitere Informationen zu Variablen sind im Abschnitt *Variablen-Editor* zu finden.

2.8.2 Grafische Oberfläche

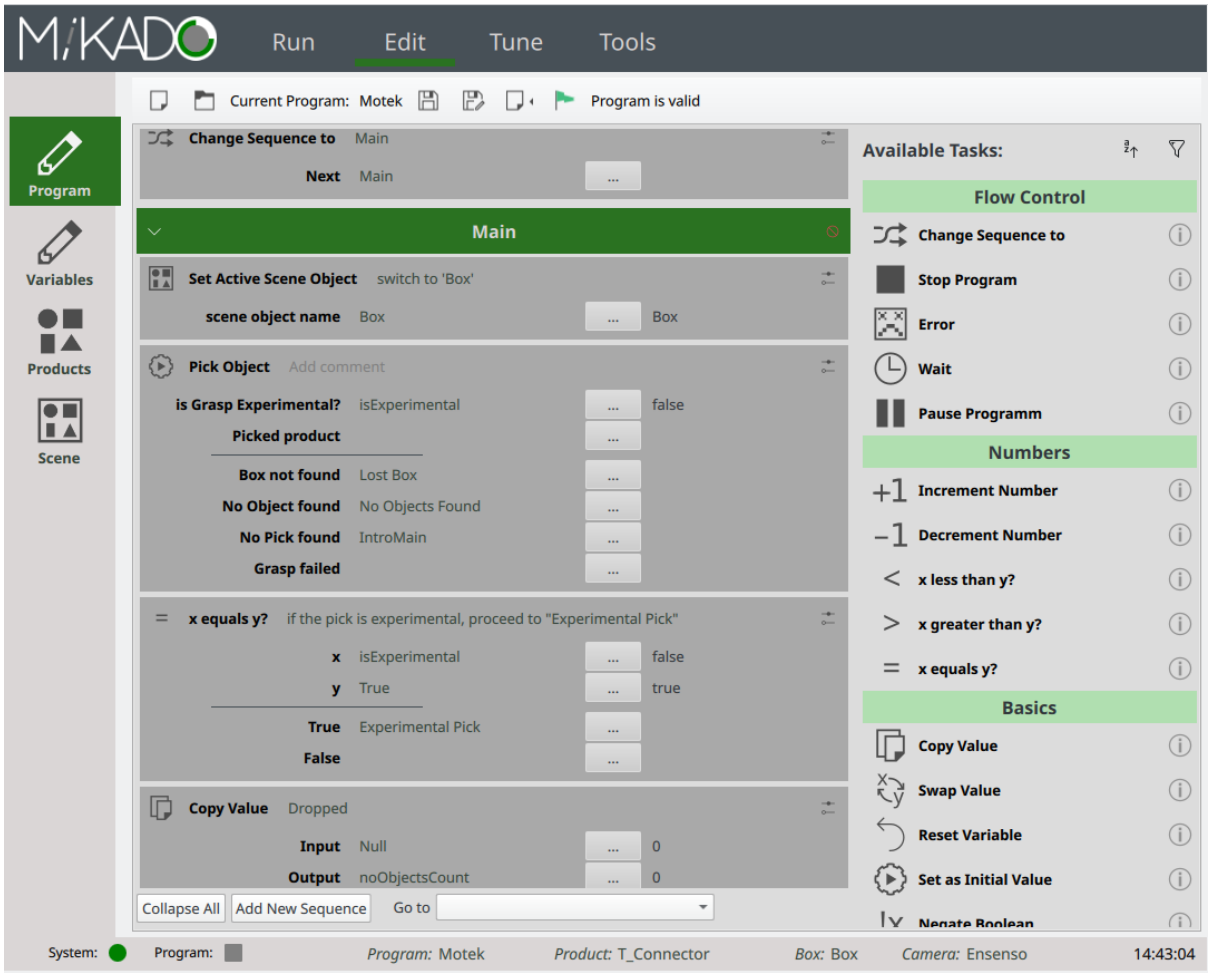


Abb. 6: Programm-Editor

Die Oberfläche des Programm-Editors besteht aus folgenden Bereichen:

- **Toolbar** (obere Leiste)

  Siehe *Toolbar*

- **Editor** (linker Bereich)

  Oberfläche um das Mikado-Programm zusammenzustellen

- **Task-Bibliothek** (rechte Leiste)

  Liste mit allen verfügbaren Task-Typen. Die Tasks sind nach Kategorien gelistet. Ein Klick auf das Info-Symbol blendet eine Kurzbeschreibung ein. Um einen Task dem aktuellen Mikado-Programm hinzuzufügen, den gewünschten Task per ‚Drag & Drop' an die gewünschte Position ziehen.

  – Alphabetische Sortierung: Ist dieser Button aktiviert, so werden die Kategorien ausgeblendet und alle Tasks werden in alphabetischer Reihenfolge aufgelistet.

  – Filter: Ist dieser Button aktiviert, so wird ein Eingabefeld angezeigt, welches zum Filtern der verfügbaren Tasks verwendet werden kann.

- **Sequenzleiste** (untere Leiste)

  Die Sequenzleiste befindet sich unter dem Editor:

  – *Alles einklappen / Alles ausklappen*

    Klappt alle Sequenzen ein/aus. Sind alle Sequenzen eingeklappt, erhalten Sie eine übersichtliche Anordnung der Sequenzen. In dieser Ansicht können Sie die Sequenzen zudem per ‚Drag & Drop' mithilfe der nun sichtbaren Anfasser verschieben.

  – *Füge Sequenz hinzu*

    Neue Sequenz erstellen.

  – *Springe zu*

    Wählen Sie hier eine Sequenz aus, um sie im Editor direkt anzuspringen. Somit können Sie in umfangreichen Programmen schneller navigieren.

2.8.3 Toolbar



Abb. 7: Toolbar des Programm-Editors

Die Toolbar befindet sich im oberen Bereich mit folgenden Elementen (von links nach rechts):

- **Neues Programm**

  Verwirft das aktuell geladene Programm und erstellt ein neues.

Warnung: Nicht gespeicherte Änderungen gehen hierbei verloren.

- **Programm öffnen**

  Öffnet einen Dialog, mit dem ein vorhandenes Mikado-Programm geladen werden kann. Es werden ebenfalls die Variablen des ausgewählten Programms geladen.

Warnung: Nicht gespeicherte Änderungen gehen hierbei verloren.

- **Aktuelles Programm**

  Der Programmname wird in diesem Feld angezeigt.

- **Programm speichern**

  Speichert alle Änderungen der Sequenzen, Tasks und Variablen.

- **Programm speichern unter ...**

  Speichert alle Änderungen der Sequenzen, Tasks und Variablen in einer neuen Datei.

- **Alle Variablen auf ihren Ausgangswert zurücksetzen**

  Mit einem Klick auf diesen Button setzen Sie die Werte aller Variablen auf ihren Initialwert zurück. Das aktuelle Mikado-Programm wird hierbei nicht gestoppt.

- **Das Programm auf Fehler überprüfen**

  Siehe *Programm-Validator*.

2.8.4 Programm erstellen

Um eine neue Sequenz zu erstellen, klicken Sie in der unteren Sequenzleiste auf den Button ‚Füge Sequenz hinzu'. Eine neue Sequenz wird an das aktuelle Programm angehängt. Ein Klick auf den Titel der Sequenz (standardmäßig „Neue Sequenz") erlaubt, den Sequenznamen zu ändern. Mit den Buttons in der Sequenzüberschrift lassen sich die Sequenzen außerdem einklappen oder löschen.

Abläufe einer Sequenz lassen sich mittels ‚Drag & Drop' zusammenstellen. Wählen Sie einen Task aus der rechten Task-Liste aus und ziehen Sie diesen mit gedrückter Maustaste an die Stelle, an der Sie ihn einfügen möchten. Eine Instanz des gewählten Tasks erscheint an der gewünschten Stelle. Sie können den Task nun optional mit einen Kommentar versehen. Je nach Task können oder müssen Sie dem Task Variablen und Sequenzen für Sequenzwechsel zuweisen. Fehlt eine zwingend notwendige Variable wird dies durch einen Hinweistext angezeigt. Weitere Informationen finden Sie auch in der Taskbeschreibung.

Für jede Variable/jeden Sequenzwechsel wird eine Zeile im Task angezeigt. Hier sehen Sie den Namen der Variable oder des Sequenzwechsels, gefolgt von der aktuellen Auswahl. Der Button jedes Eintrags öffnet ein Auswahldialog, mit dem Sie eine existierende Variable/existierenden Sequenzwechsel auswählen können. In diesem Dialog gibt es zudem ein Textfeld, mit dem eine neue Variable/eine neue Sequenz erstellt werden kann. Zuletzt wird der aktuelle Wert der ausgewählten Variable neben dem Button dargestellt.

Tasks haben neben diesen Elementen oben rechts einen Meta-Button, mit dem eine erweiterte Ansicht eingeblendet werden kann. In dieser Ansicht lassen sich Tasks verschieben, duplizieren und löschen. Einen Task zu verschieben oder zu duplizieren erfolgt ebenfalls durch ‚Drag & Drop'.

Hinweis: So geht's schneller:

- Klick auf den Variablennamen: Direkteingabe des Variablennamens. Existiert dieser nicht, so wird eine neue Variable angelegt.
- Klick auf den Variablenwert: Öffnet einen Schnell-Dialog in dem die Variableneigenschaften editiert werden können.

Hinweis: Bereuen Sie nichts: Löschen Sie einen Task oder eine Sequenz, so wird die Aktion um einige Sekunden verzögert ausgeführt. So haben Sie eine letzte Möglichkeit, die Aktion zu widerrufen falls diese nicht beabsichtigt war!

2.8.5 Programm-Validator

Das aktuelle Mikado-Programm wird automatisch unter Verwendung des Programm-Validators auf einige Fehler überprüft. Das Ergebnis wird anhand einer grünen oder roten Flagge in der Toolbar angezeigt. Im Fehlerfall wird auch die Anzahl der fehlkonfigurierten Tasks angezeigt. Es werden zum einen Sequenzwechsel und Variablen auf Existenz geprüft und zum anderen wird kontrolliert, ob die Typen der Task-Variablen korrekt sind. Einige Tasks haben noch gesonderte Bedingungen, z.B. kann an eine StringList nur ein String angehängt werden, nicht jedoch eine Pose. Genauere Hinweise zur Fehlerbehebung finden Sie entweder in einem zusammenfassenden Dialog, wenn Sie auf die Flagge klicken, oder bei den einzelnen Tasks in rot im Vorschaubereich.

Diese Validierung findet außerdem automatisch vor jedem Start eines Mikado-Programms statt und zeigt einen Hinweis an, falls die Validierung fehlschlägt. Das Programm kann dann bei Bedarf trotz Fehler bis zum Erreichen des ersten fehlerhaften Tasks ausgeführt werden.

2.9 Variablen-Editor

Eine Übersicht aller Variablen ist unter dem Menüpunkt ‚Edit' auf der Seite ‚Variablen' zu finden. Zu jeder Variable wird erläutert, wie deren Werte bearbeitet und dargestellt werden können.

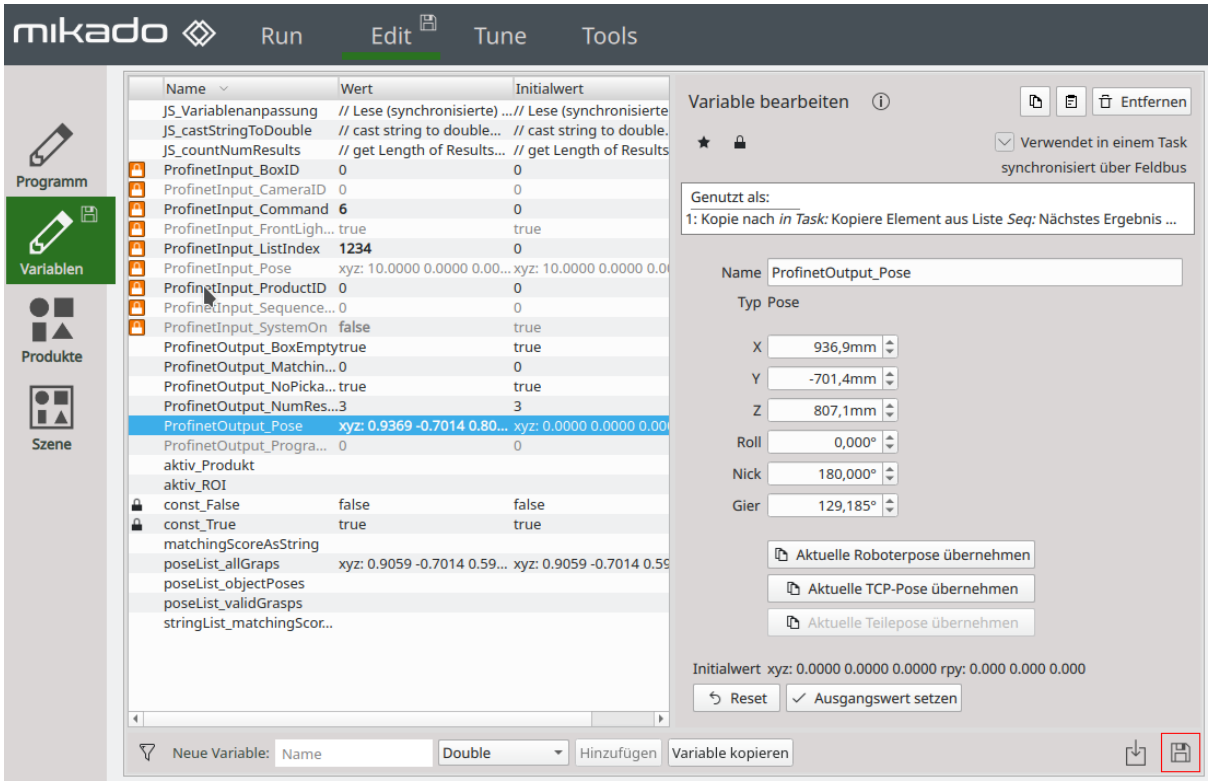


Abb. 8: Variablen-Editor

2.9.1 Variablentabelle

Im linken Teil der Seite werden die Variablen in tabellarischer Form aufgelistet. Weitere Spalten beinhalten den aktuellen Wert sowie den Initialwert (genauere Erläuterung s.u.). Die Liste ist alphabetisch sortiert, wobei zuerst Variablen mit Großbuchstaben beginnend und danach Variablen mit Kleinbuchstaben beginnend aufgelistet sind.

Weichen Variablenwerte von ihren Initialwerten ab, so wird der Variablenwert in fetter Schriftart dargestellt.

2.9.2 Bearbeitung von Variablen

Mit einem Klick auf einen Eintrag der Variablentabelle wird rechts neben der Tabelle eine Ansicht zur Bearbeitung der Variable sichtbar. Diese Ansicht enthält folgende Elemente:

Eigenschaftskennzeichen

Variablen können zusätzliche Eigenschaften haben, welche durch Klicken auf die unten abgebildeten Buttons (de-)aktiviert werden können.

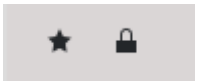


Abb. 9: Buttons zum (De-)Aktivieren von Eigenschaften

Favorisierte Variablen

Favorisierte werden in der Variablentabelle immer zuoberst angezeigt. Bei Programmen mit vielen Variablen kann dieser Eigenschaft helfen wichtige Variablen schneller wiederzufinden.

Schreibgeschützte Variablen

Schreibgeschützte Variablen haben folgende Eigenschaften:

(1) Der Wert kann vom Mikado-Programm nicht geändert werden. Versucht ein Task den Wert einer schreibgeschützten Variable zu ändern, wird ein Programm-Fehler geworfen. Dies beinhaltet auch, dass Schreibgeschützte Variablen nicht zurückgesetzt werden können.

Es wird unterteilt, woher der Schreibgeschützte Status kommt. Ist der Status Benutzer-definiert, symbolisiert durch , wird folgende Eigenschaft aktiv:

(2) Wert und Initialwert werden synchronisiert. Benutzer können den Wert auf der Variablen-Seite einstellen indem der Schreibschutz vorübergehend aufgehoben wird.

Oder es handelt sich um eine Eingangs-Variable der aktiven Schnittstelle. Eingangs-Variablen sind automatisch schreibgeschützt und werden durch  symbolisiert. Externe Variablen besitzen folgende Eigenschaft:

(3) Sie können vom Nutzer nicht modifiziert werden. Dieser Status kann nur zur Laufzeit erreicht werden, wenn eine Schnittstelle aktiv ist. Dieser Zustand wird nicht mitgespeichert.

Auftreten

Durch Klicken auf den Button links neben dem Auftreten Zähler, öffnet sich eine Liste mit der detaillierten Auftreten-Beschreibung. Mit einem Klick auf einen Eintrag wird an die entsprechende Stelle im Programmablauf gesprungen.

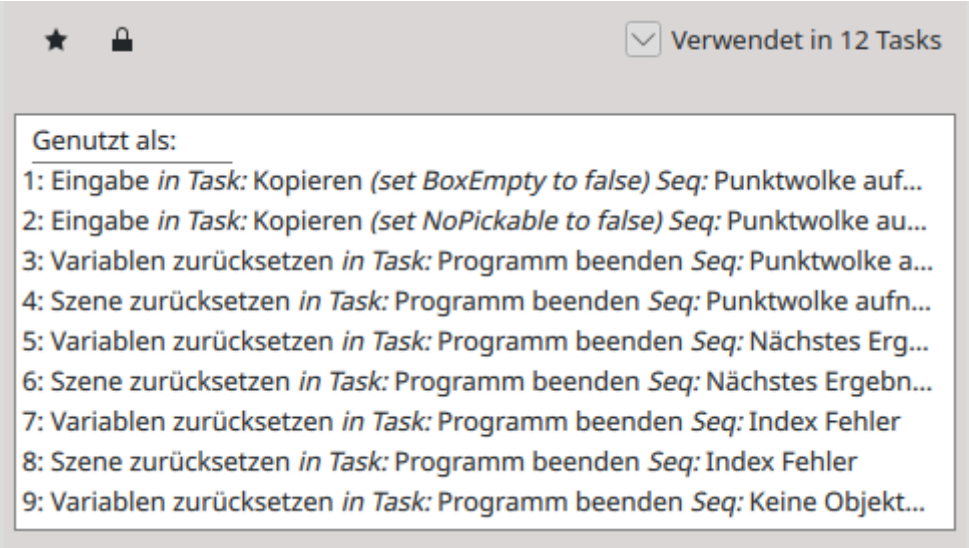


Abb. 10: Beispiel für Variablen-Auftreten

Name

Variablenname

Typ

Eine Übersicht über die Typen mit je einer kurzen Erläuterung finden Sie in der folgenden Tabelle.

Variablentyp	Beschreibung	Beispielwerte
Integer	Ganzzahl	`1`
Double	Dezimalzahl	`1.5`
String	Text	`Kiste`
StringList	Liste an Textelementen	`Kiste1` `Kiste2`
JointPosition	Roboterposition repräsentiert durch die einzelnen Gelenkwinkelwerte in Grad	`90 0 90 0 90 45`
Pose	Pose bestehend aus Position (x, y, z) in m und Orientierung (Roll, Nick, Gier) im Bogenmaß z.B. für Greifpose, Produktpose etc. *Hinweis*: Eine Pose ist eine Darstellungsform der homogenen Transformationsmatrix. Beide Repräsentationen lassen sich ineinander umwandeln.	`xyz: 0.1 0 0.5 rpy: 0 0 3.14`
PoseList	Liste an Pose-Elementen	`xyz: 0.1 0 0.5 rpy: 0 0 3.14` `xyz: 0.2 0 0.6 rpy: 3.14 0 3.14`
Bool	Boolscher Wert (`true` oder `false`)	`true`

JScript	Ein textähnlicher Datentyp, der JavaScript Code beinhaltet	`var a = dataStore.data("a")`
		`a = a + 1`
		`dataStore.setData("a", a)`

Wert

Der aktuelle Wert der Variable. Der Wert einer Variable kann sich durch den Ablauf des Programms jederzeit ändern.

Handelt es sich um eine StringList, ist des Textfeld mehrzeilig und Listeneinträge werden durch einen Zeilenumbruch getrennt.

Initialwert

Der Wert, der beim Start des Mikado-Programms oder beim Reset einer Variable zugewiesen wird. Dieser Wert wird beim Speichern eines Programms mitgespeichert und wird i.d.R. während eines Programmablaufs nicht verändert.

Mögliche Aktionen

- ***Reset***
  Setzt den aktuellen Wert der Variable auf den Initialwert zurück.
- ***Entfernen***
  Löscht die Variable und deren Inhalt.
- ***Bestätigen***
  Eingabe bestätigen. Nur für Listen-Eingabefelder sichtbar, da bei einfachen Eingabefeldern ein Bestätigen über die Eingabetaste ‚Enter' ausgeführt wird (oder durch das Wechseln in ein anderes Feld).
- ***Aktuelle Roboterpose übernehmen***
  Setzt den Wert der Variable auf die Pose, die der Roboter momentan einnimmt (nur für den Variablentyp ‚Pose' sichtbar).
- ***Aktuelle Teilepose übernehmen***
  Setzt den Wert der Variable auf die Pose, die ein gegriffenes Teil momentan einnimmt (nur für den Variablentyp ‚Pose' sichtbar). Voraussetzung für diese Funktion ist, dass der Roboter ein Teil erfolgreich gegriffen hat und momentan im Greifer festhält.
- ***Aktuelle Gelenkwerte übernehmen***
  Setzt den Wert der Variable auf die Gelenkwinkel, die der Roboter momentan einnimmt (nur für den Variablentyp ‚JointPosition' sichtbar).

Hinweis: Bestätigen Sie eine Eingabe immer mit der Eingabetaste oder wechseln Sie in das nächste Eingabefeld, erst danach werden Eingabewerte übernommen.

2.9.3 Variablen Hinzufügen und Filtern

Unterhalb der Variablentabelle befinden sich Eingabefelder, die sowohl zum Hinzufügen neuer Variablen als auch zum Filtern vorhandener Variablen verwendet werden können.

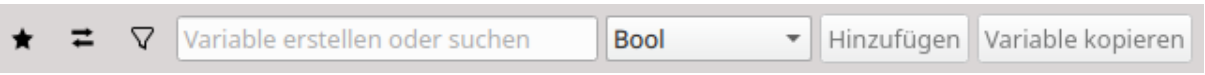


Abb. 11: Variablen Hinzufügen und Filtern

Über die Buttons links können bestimmte Filter für die vorhandenen Variablen aktiviert bzw. deaktiviert werden.

- ★ Nur Favoriten anzeigen
- ⇄ Nur Feldbus-Variablen anzeigen
- ▽ Nur Variablen des aktuell ausgewählten Typs anzeigen. Dies bezieht sich auf den weiter rechts in der Auswahl gewählten Variablen-Typ.

Das Textfeld in der Mitte dient zur Eingabe eines Namens für eine neue Variable und fungiert gleichzeitig als Namensfilter für vorhandene Variablen. Mit der Auswahl daneben wird der Typ einer neuen Variablen oder des zugehörigen Filters gewählt.

Um eine neue Variable hinzuzufügen, muss zunächst ein noch nicht verwendeter Variablenname in das Textfeld eingegeben werden. Anschließend muss der gewünschte Variablentyp ausgewählt werden. Durch Klicken auf „Hinzufügen" wird die neue Variable erzeugt. Sie erscheint oben in der Tabelle und kann anschließend wie gewohnt bearbeitet werden.

Mit dem Button „Variable kopieren" kann außerdem die aktuell ausgewählte Variable kopiert werden.

2.9.4 Variablen aus Datei importieren

Im unteren Bereich der Seite ist ein Button ‚Aus Datei importieren' zu finden. Mithilfe dieser Funktion können Sie eine komplette Variablentabelle aus einem anderen Mikado-Programm der existierenden Variablenliste hinzufügen. Dabei verändern Sie das Hauptprogramm nicht. Folgende Fälle können dabei auftreten:

- **Neuer Variablenname:**
  Variable wird hinzugefügt
- **Variablenname existiert bereits, Variablentyp stimmt überein:**
  Der aktuelle Wert wird durch den Wert aus der einzulesenden Datei ersetzt, Initialwert bleibt unverändert
- **Variablenname existiert bereits, Variablentyp stimmt nicht überein:**
  Variable bleibt unverändert, die Variable aus der Datei wird nicht importiert

2.9.5 Visualisierung der Roboter-Pose

Sobald Sie eine Variable angelegt haben, die eine Roboterpose darstellt (Variablentyp ‚Pose' oder ‚JointPosition'), können Sie den Wert dieser Variable in der Visualisierung als Vorschau darstellen. Hierzu genügt ein Klick auf die Variable in der Tabelle und der Eintrag wird farbig markiert.

Visualisierung einer JointPosition

Variablen vom Typ ‚JointPosition' werden in der Visualisierung durch einen transparenten Roboter dargestellt um ihn von der tatsächlichen Roboterpose unterscheiden zu können.

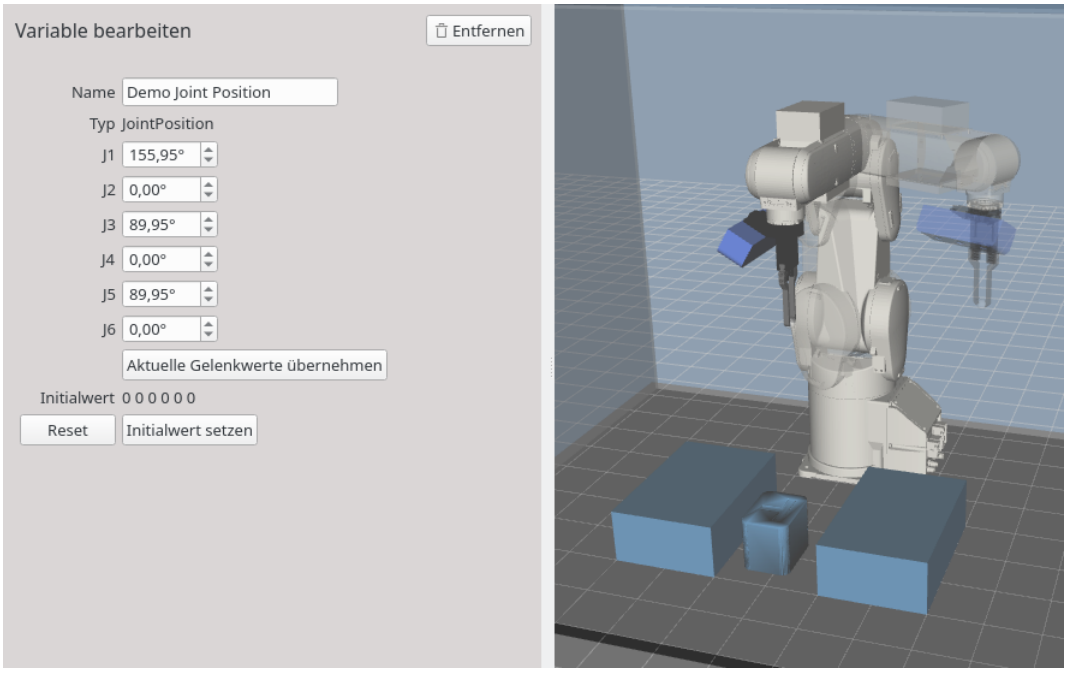


Abb. 12: Visualisierung einer JointPosition. Links die tatsächliche momentane Roboterposition, rechts das transparente Modell als Vorschau der ausgewählten JointPosition.

Visualisierung einer Pose

Variable vom Typ ‚Pose' können in verschiedenen Modi visualisiert werden:

1. ***Handgelenk***
   Falls der Variablen momentan ein valider Wert zugewiesen ist, den das Roboter-Handgelenk auch theoretisch einnehmen könnte (ohne Berücksichtigung von Kollisionen), wird in der Visualisierung der Szene ein weiterer (virtueller) Roboter eingezeichnet. Dieses Robotermodell wird leicht transparent dargestellt. Er zeigt die Pose/Position, die der Roboter beim Anfahren einnehmen würde.

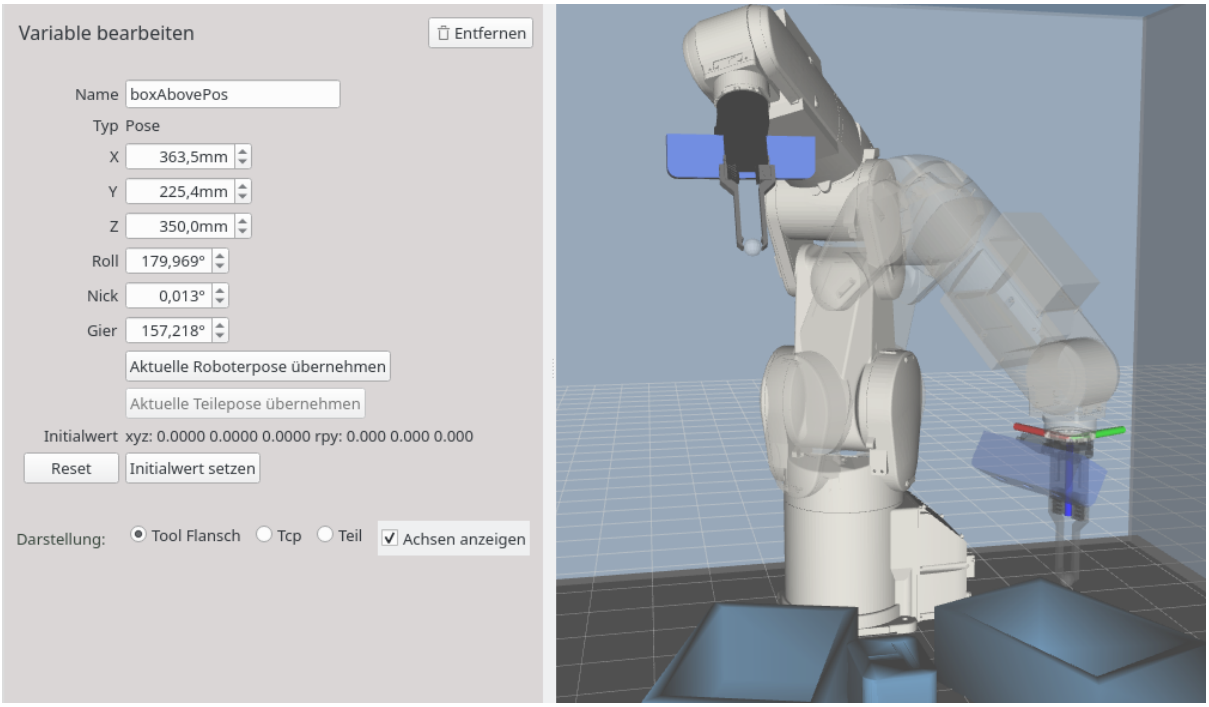


Abb. 13: Visualisierung einer Roboterpose im ‚Handgelenk' Modus

2. ***TCP***
   Falls der Variable momentan ein valider Wert zugewiesen ist, den der Endeffektor (TCP) auch theoretisch einnehmen könnte (ohne Berücksichtigung von Kollisionen), wird in der Visualisierung der Szene ein weiterer (virtueller) Roboter eingezeichnet. Dieses Robotermodell wird leicht transparent dargestellt. Er zeigt die Pose/Position, die der Roboter beim Anfahren einnehmen würde.

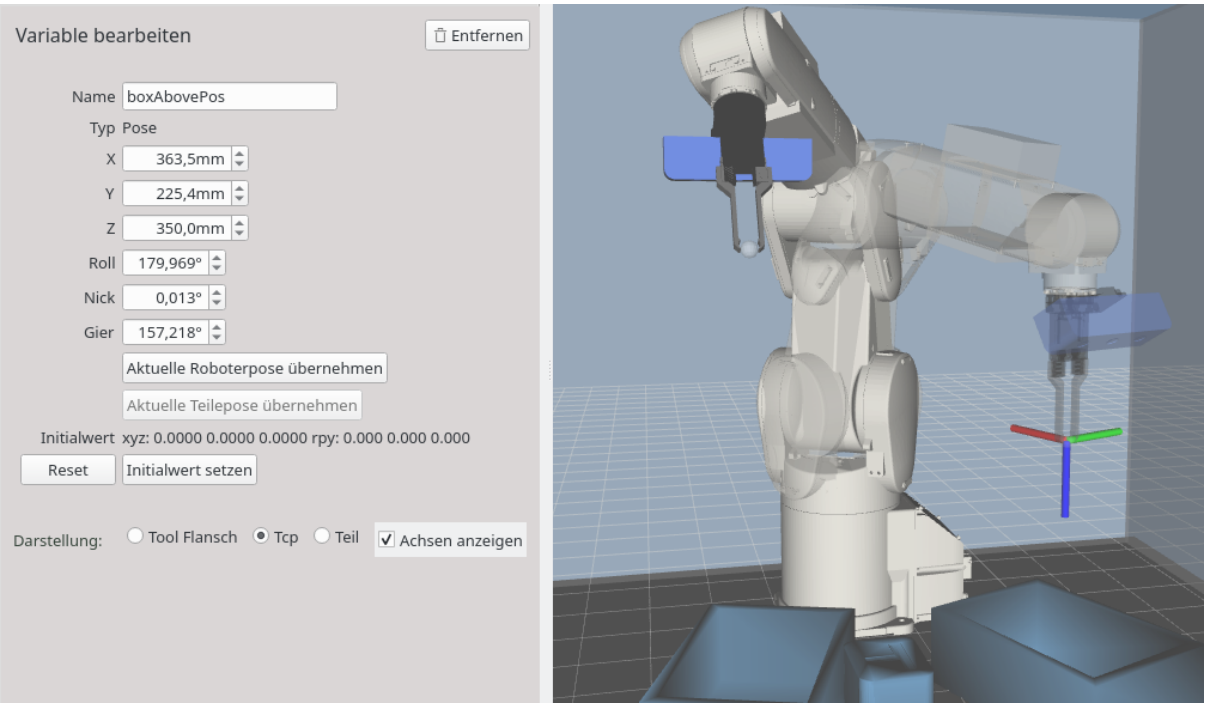


Abb. 14: Visualisierung einer Roboterpose im ‚TCP' Modus

3. ***Teil***

   Falls der Variablen ein valider Wert zugewiesen ist, den das aktuell gegriffene Teil auch theoretisch einnehmen könnte (ohne Berücksichtigung von Kollisionen), wird in der Visualisierung der Szene ein weiterer (virtueller) Roboter eingezeichnet. Dieses Robotermodell wird leicht transparent dargestellt. Er zeigt die Pose/Position, die der Roboter beim Anfahren einnehmen würde. Der virtuelle Roboter wird lediglich dann angezeigt, wenn der Roboter ein Teil erfolgreich gegriffen hat und momentan im Greifer festhält. Ist dies nicht der Fall, so wird das momentan aktive Produkt an der/den Pose/n ohne virtueller Roboter eingeblendet.

4. ***Achsen anzeigen***

   Aktivieren Sie diese Einstellung, um die Pose zusätzlich als ein einfaches, dreifarbiges Achssystem darzustellen. Standardmäßig ist diese Option aktiv, kann allerdings durch die Checkbox ausgeblendet werden.

   Beachten Sie, dass Achsen automatisch angezeigt werden, wenn Sie eine PoseList selektieren ohne den Visualisierungs-Modus *Teil* zu verwenden.

Falls bei einer gewählten Visualisierung ein Fehler auftritt (z.B. Roboter kann Pose aufgrund der Geometrie nicht erreichen), so wird dieser unten, neben den Visualisierungsmodi, rot angezeigt.

Visualisierung von Kollisionen

Wählen Sie eine JointPosition oder Pose aus der Variablenliste, so wird eine Vorschau in der Visualisierung angezeigt (s. *Visualisierung der Roboter-Pose*). Um bereits beim Einrichten der Position oder Pose prüfen zu können, ob der Roboter oder Greifer mit anderen Objekten der Umgebung kollidieren würde, werden potentielle Kollisionen direkt im Visualisierungsfenster angezeigt. Kollidierende Teile des Roboters werden rot eingefärbt.

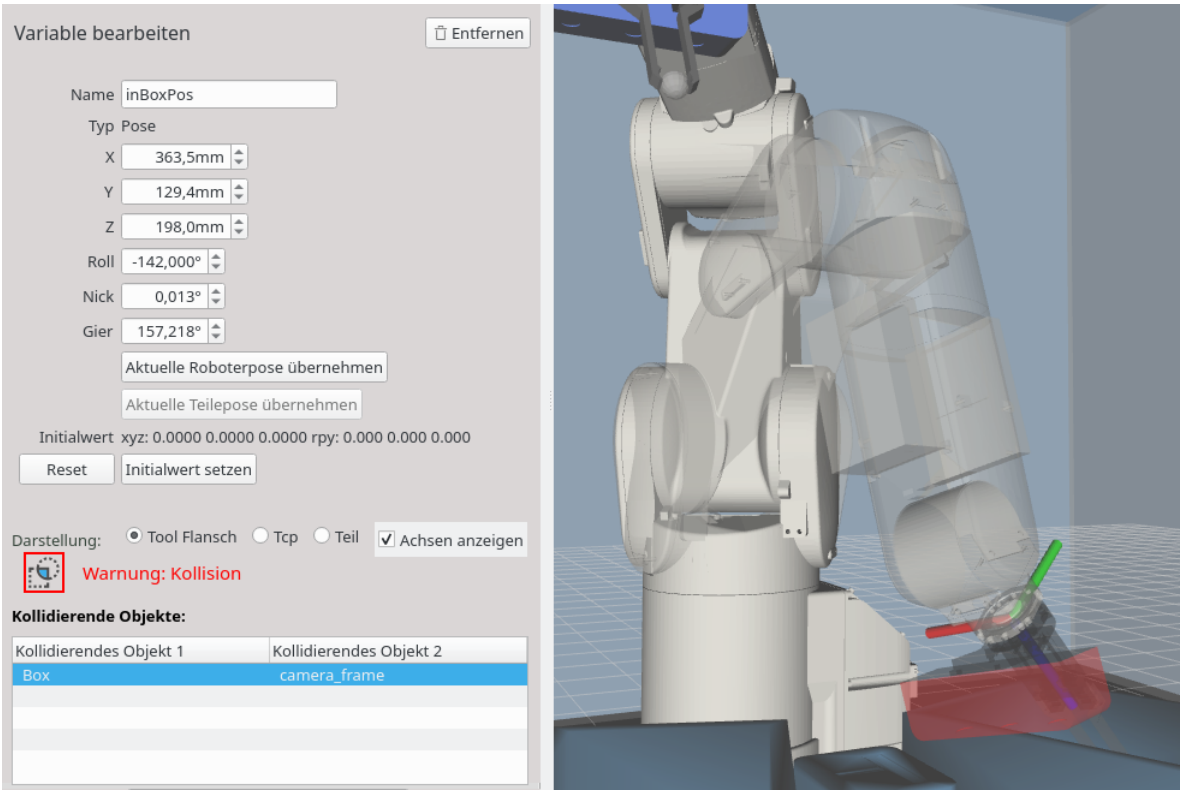


Abb. 15: Visualisierung mit Kollisionen am Greifer

Weiterhin erscheint eine Hinweismeldung mit entsprechendem Symbol, das darauf hinweist, dass die gewählte Pose im aktuell ausgewählten Modus aufgrund von Kollisionen nicht erreichbar ist. Die einzelnen Kollisionen werden in einer Liste darunter angezeigt. Beachten Sie, dass dies lediglich ein Hinweis ist. In manchen Fällen kann die Pose trotzdem für einen funktionierenden und kollisionsfreien Ablauf verwendet werden. Beispielsweise können Kollisionen mit Szenenobjekten auftreten, die im Ablauf aber ihre Position verändern. Ein weiteres Beispiel sind Posen die innerhalb eines Programmablaufs nochmal transformiert werden.

2.10 Produktverwaltung

2.10.1 Überblick

Ein Produkt beschreibt ein physikalisches Objekt, das über die 3D-Erkennung gefunden und ggf. gegriffen werden soll. Es besteht aus einem Namen sowie diversen Parametern für die Objektsuche sowie den Greifprozess. Es ist zu unterscheiden zwischen dem (abstrakten/beschreibenden) „Produkt“ auf der einen und dem konkreten „Objekt“ auf der anderen Seite: Dasselbe Produkt kann bei einer Objektsuche mehrfach gefunden werden. Jedes Objekt ist dabei eine Instanz des Produktes (mit einer konkreten Lage im Raum).

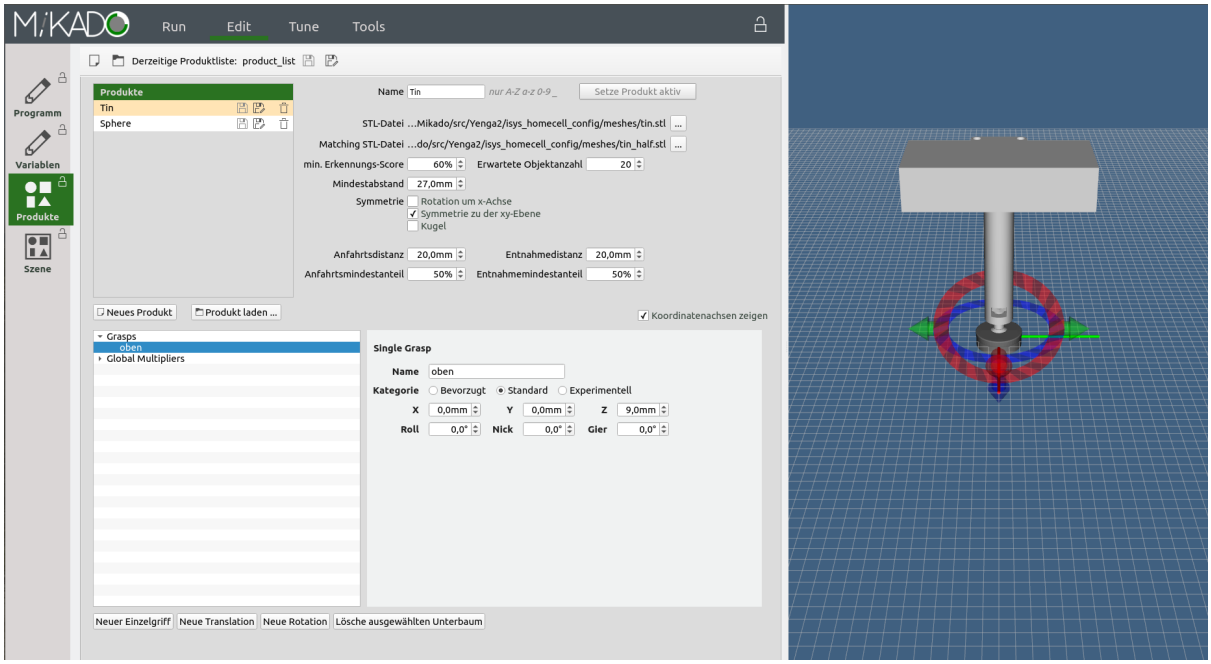


Abb. 16: Screenshot Produktverwaltung mit Einzelgriff

2.10.2 Name

Der Produktname identifiziert das zu suchende Teil und muss folgende Kriterien erfüllen:

- Muss eindeutig sein, d.h. zwei Produkte dürfen nicht den gleichen Namen haben
- Beginnt mit einem Buchstaben
- Besteht aus alphanumerischen Zeichen (a-z, A-Z, 0-9) oder Unterstrichen (‚_‘)

2.10.3 Aktives Produkt

Während der Programmausführung ist immer genau ein Produkt als ‚Aktives Produkt‘ definiert (sofern in der Produktverwaltung mindestens ein Produkt existiert). Kennt die Produktverwaltung lediglich ein Produkt, so ist dieses automatisch das aktive Produkt. Wird eine produktbezogene Aktion ausgeführt, so wird das aktive Produkt für diese Aktion verwendet.

Beispiel: Verwenden Sie den Task *Griffsuche starten*, so werden mögliche Griffe für das aktive Produkt gesucht. Wenn Sie jedoch im Programmeditor die Variable ‚Produkt‘ definieren, so wird unabhängig vom aktiven Produkt nach dem von Ihnen definierten Produkt gesucht.

Das aktive Produkt lässt sich entweder in der Produktverwaltung über den Button ‚Als aktives Produkt verwenden‘ setzen, oder als Teil des Mikado-Programms mithilfe des Tasks *Setze aktives Produkt*.

2.10.4 Objektsuche

Jedes Produkt enthält die folgenden Parameter für die Objektsuche:

STL-Datei / STL-Datei (Erkennung)

Hier wird die Mesh-Datei (CAD-Daten) des Objektes angegeben, die für die Darstellung sowie die Objektsuche verwendet werden soll. Wird unter „STL-Datei (Erkennung)“ nichts angegeben, wird die unter „STL-Datei“ angegebene Datei für die Erkennung verwendet. Andernfalls wird die unter „STL-Datei (Erkennung)“ angegebene Datei für die Erkennung und die unter „STL-Datei“ angegebene Datei nur für die Darstellung verwendet. Somit lässt sich ein vereinfachtes Mesh für die Erkennung verwenden, während in der Anzeige das volle Mesh dargestellt wird.

Folgende Voraussetzungen und Hinweise sind bei der Erstellung/Verwendung einer STL-Datei zu beachten:

- Verwenden Sie als Maßeinheit für das STL Millimeter
- Verwenden Sie Meshes mit möglichst wenigen Dreiecken (Faces) - so wenige wie möglich, aber ausreichend viele um alle wichtigen Strukturen des Produkts weiterhin gut abzubilden. Je mehr Dreiecke verwendet werden, desto höher die Rechenzeit für z.B. Kollisionsprüfungen.
- Wählen Sie einen sinnvollen Ursprung des Produktkoordinatensystems, beispielsweise im Schwerpunkt des Produkts.
- Weist das Produkt Symmetrien auf, so positionieren Sie den Ursprung des Produktkoordinatensystems auf der Symmetrieachse oder -ebene.
- Entfernen Sie möglichst alle Dreiecke, die im Produkt liegen oder nie sichtbar sind.

Suchalgorithmus

Hier wird der für die Objekterkennung verwendete Suchalgorithmus festgelegt. Mikado Control bietet verschiedene Suchalgorithmen zur Auswahl:

- Ensenso Partfinder (empfohlen), ein von Optonic entwickelter Suchalgorithmus, siehe auch Partfinder Dokumentation.
- Halcon Surface Based, ein von MVTec entwickelter Suchalgorithmus. Hierfür ist eine zusätzliche Lizenz notwendig.

Min. Erkennungs-Score

Die Objektsuche ermittelt für jedes potenzielle Objekt in der Szene einen Score. Dieser Score ist das Maß der Übereinstimmung der potenziellen Objektfunde mit dem Modell. Hier wird der Minimalwert angegeben, den ein potenzialles Objekt überschreiten muss, um als erkanntes Objekt zu gelten.

Erwartete Objektanzahl

Um die Objektsuche zu beschleunigen, können Sie hier eine zu erwartende Objektanzahl angeben. Außerdem steigt die Wahrscheinlichkeit ein Objekt zu finden, wenn dieser Wert höher gewählt wird. Falls Sie die Anzahl an maximal zu erkennende Objekten einschränken möchten, können Sie hier eine Obergrenze festlegen. Wenn Sie die Wiederverwendung vorheriger Matches aktiviert haben, gilt diese Beschränkung nur für neue Funde (mehr zum Thema *Wiederverwendung vorheriger Matches* in Abschnitt *Objekterkennungs-Tool*).

Mindestabstand

Hier wird der Mindestabstand zwischen zwei gefundenen Objekten angegeben. Dies hat folgendene Auswirkungen bei der Objektsuche: Gibt es mehrere nah beieinanderliegende potenzielle Objekte, wird das mit dem besten Score ausgewählt und alle weiteren Objekte, die innerhalb des Mindestabstands liegen, werden verworfen.

Symmetrien

Produkte verfügen teilweise über Rotationssymmetrien. Das bedeutet vereinfacht gesagt: Wenn man das Produkt um eine Achse dreht, verändert sich sein Aussehen nicht. Mikado unterstützt dabei kontinuierliche Rotationssymmetrien um die Hauptachsen X, Y und Z sowie diskrete Rotationssymmetrien mit 180°-Rotation um eine der Hauptachsen. Das Einrichten der Symmetrien ist nicht zwingend notwendig, kann aber Rechenzeit sparen und unnötige Drehbewegungen vermeiden.

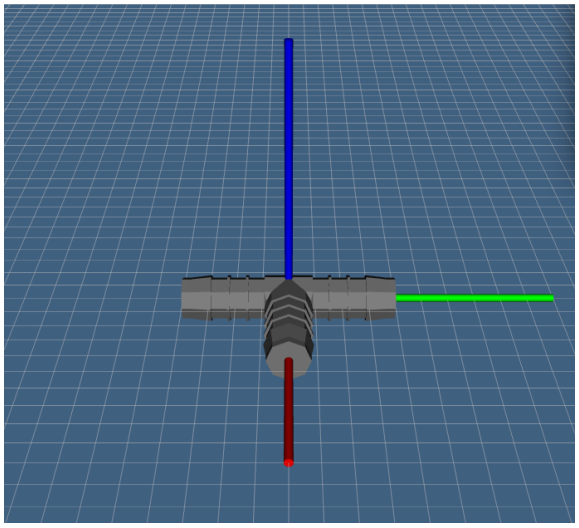


Abb. 17: 180°-Rotationssymmetrie um die X-Achse (rot)

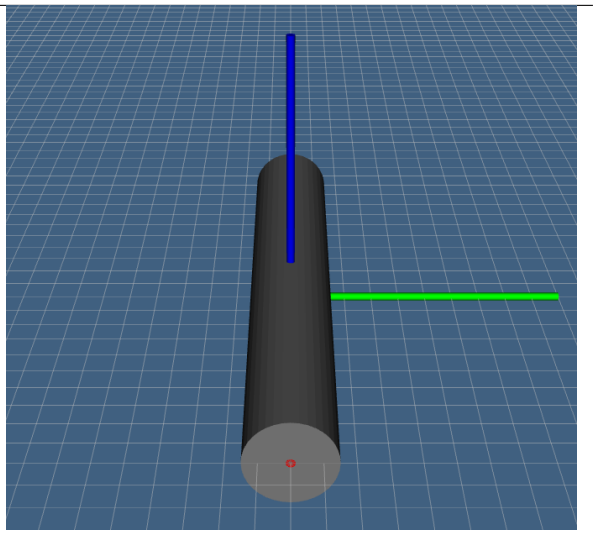


Abb. 18: Rotationssymmetrie um die X-Achse (rot) und 180°-Rotationssymmetrie um Y (grün) und Z (blau)

Hinweis: Voraussetzung hierfür ist, dass der Ursprung und die Orientierung des Produktkoordinatensystems in den CAD-Daten hierzu passend konfiguriert wurden.

Bei der Objekterkennung mit PartFinder kann über die Einstellung fromProductSymmetry erreicht werden, dass die Anzahl der gerenderten Perspektiven passend reduziert wird.

Anschließend werden die gefundenen Objekte, unabhängig vom Objekterkennungsalgorithmus, entsprechend der Symmetrie ausgerichtet. Beispielsweise werden Funde für ein Produkt mit kontinuierlicher Rotationssymmetrie um die X-Achse so gedreht, dass die Z-Achse möglichst senkrecht nach oben (Welt-Z) zeigt. Bei diskreter 180°-Symmetrie um X wird ebenfalls so gedreht, dass Z in die positive Raumrichtung zeigt. Dies gilt ebenfalls für die Symmetrien der Y-Achse. Bei Symmetrien zur Z-Achse werden die Teile so ausgerichtet, dass die X-Achse in Richtung der Welt X-Achse zeigt.

Hinweis: Hierdurch kann die zu definierende Anzahl der Greifposen reduziert werden (bei symmetrischen Objekten müssen somit die unten liegenden Greifpunkte nicht zusätzlich erstellt werden). Dies ist aber z.B. kein kompletter Ersatz für eine Rotations-Multiplikation eines definierten Griffes.

Zusätzlich wird die Symmetrie berücksichtigt bei Bewegungen des Roboters über *Bewege Teil nach*, *Bewege Teil kartesisch nach* sowie bei der Erreichbarkeitsabfrage *Ist Roboter Position mit Teil erreichbar?*. Hier wird beispiels-

weise ein zylindrisches Teil mit 180°-Rotationssymmetrie um die Z-Achse nicht unnötig gedreht, sondern die der Ausgangsposition näherliegende Lösung verwendet.

Entnahmedistanz

Hier wird die Distanz zwischen Greif- und Entnahmepose festgelegt. Diese Posen sind im Abschnitt *Greifprozess* definiert.

Entnahmemindestanteil

Hier wird die Mindestdistanz beim Entnehmen als Prozentsatz der Entnahmedistanz festgelegt.

Beim Überprüfen des Entnehmens wird jeweils von der Greifpose aus die Strecke hin zur Entnahmepose auf Kollisionen überprüft. Dabei kann eingestellt werden, dass nicht die gesamte Strecke zurückgelegt werden muss, sondern nur ein bestimmter Prozentsatz. Diese Posen sind im Abschnitt *Greifprozess* definiert.

Beispiel: Ist ein Entnahmemindestanteil von 70 % eingestellt und nach 80 % würde eine Kollision auftreten, wird die tatsächliche Entnahmepose auf die Pose nach 79 % gesetzt und von dieser aus wird frei (zur Endpose hin) geplant. Dies ermöglicht es, die Entnahmedistanz für den Idealfall zu definieren und andererseits jeweils eine Mindestdistanz zu definieren, die kartesisch abgefahren werden muss.

Anfahrtsdistanz

Hier wird die Distanz zwischen Greif- und Anfahrtspose festgelegt. Diese Posen sind im Abschnitt *Greifprozess* definiert.

Anfahrtsmindestanteil

Hier wird die Mindestdistanz beim Greifen als Prozentsatz der Anfahrtsdistanz festgelegt. Die Beschreibung in *Entnahmemindestanteil* gilt analog auch für diesen Parameter.

2.10.5 Greifposen

Als „Greifposen“ werden die Posen im Produkt-Koordinatensystem bezeichnet, an denen der Greifer ansetzen darf, um das Produkt zu greifen. Die Greifposen werden über Greifmultiplikatoren festgelegt, mittels derer man Gruppen von Greifposen miteinander multiplizieren kann. Sie werden im folgenden Abschnitt näher beschrieben.

Außen- und Innengriffe

Je nach Greifer können verschiedene Arten von Griffe verwendet werden:

- ***Außengriffe***
  Der Roboter öffnet den Greifer bevor er die Griffbewegung ausführt. Am Griffpunkt schließt er den Greifer um das Produkt von außen zu greifen.
- ***Innengriffe***
  Der Roboter schließt den Greifer bevor er die Griffbewegung ausführt. Am Griffpunkt öffnet er den Greifer um das Produkt von innen zu greifen (z.B. ein Loch oder eine Aussparung im Produkt).

Welchen Grifftyp Sie tatsächlich verwenden können, hängt vom verwendeten Greifer ab. Legen Sie neue Griffe in der entsprechenden Kategorie an, um den gewünschten Grifftyp zu definieren.

Greifmultiplikatoren

Greifmultiplikatoren enthalten verschiedene Kombinationen der folgenden Eigenschaften:

- Name
- Kategorie (Bevorzugt, Standard, Experimentell, <geerbt>)
- Position (x, y, z)
- Orientierung (Roll, Nick, Gier)
- Multiplikator (natürliche Zahl)
- symmetrisch?

Der Name eines Greifmultiplikators muss für ein bestimmtes Produkt eindeutig sein.

Anhand der Kategorie kann eingestellt werden, welche Griffe zuerst überprüft werden sollen (in der Reihenfolge „Bevorzugt“ > „Standard“ > „Experimentell“). Wird für einen Greifmultiplikator keine Kategorie eingestellt, so „erbt“ er die Kategorie des darüberliegenden Greifmultiplikators. Bei Rotations- und Translationsmultiplikatoren wirkt sich die Kategorie nur auf „echte“ Rotationen und Translationen aus, also nicht auf 1 (Rotation) bzw 0 (Translation) (siehe unten).

Wird kein Griff der Kategorien „Bevorzugt“ oder „Standard“ gefunden, sondern nur ein Griff der Kategorie „Experimentell“, so wird im *Teil greifen*-Task die entsprechende Ausgabevariable auf true gesetzt. Dadurch kann die Weiterverarbeitung davon abhängig gemacht werden, ob der Griff experimentell war oder nicht. Die gegriffenen Objekte können dann beispielsweise neu ausgerichtet oder in die Kiste fallen gelassen werden, um sie erneut zu greifen.

Die Eigenschaft „Multiplikator“ gibt die Anzahl der Greifposen innerhalb eines Greifmultiplikators an. Ein Einzelgriff bspw. hat immer einen Multiplikator von 1. Rotations- und Translationsmultiplikatoren hingegen haben immer einen Multiplikator von mindestens 2.

Die Eigenschaft „symmetrisch“ gibt es nur für Rotationsmultiplikatoren (s.u.).

Im Folgenden werden die einzelnen zur Verfügung stehenden Multiplikatoren näher erläutert.

Einzelgriff

Ein Einzelgriff hat immer einen Multiplikator von 1. Es können für den Einzelgriff die Position und Orientierung eingestellt werden – es wird also eine Greifpose definiert. Da die Multiplikatoren immer die Identitätstransformation beinhalten, müssen zum reinen Verschieben oder Rotieren auch Einzelgriffe verwendet werden. Auf diese können dann weitere Rotationen oder Translationen draufmultipliziert werden.

Translationsmultiplikator

Ein Translationsmultiplikator hat immer einen Multiplikator von mindestens 2 sowie eine Position (Translation), die als Generator der verschiedenen Translationen fungiert. Die daraus resultierenden Translationen sind {0, T, 2*T, … , (m-1)*T}, wobei m der Multiplikator-Wert und T der Translationsgenerator ist.

Rotationsmultiplikator

Ein Rotationsmultiplikator hat immer einen Multiplikator von mindestens 2 sowie eine Orientierung (Rotation), die als Generator der verschiedenen Rotationen fungiert. Zusätzlich hat er noch die bool'sche Eigenschaft „symmetrisch". Ist diese nicht gesetzt, so sind die berechneten Rotationen {1, R, 2*R, … , (m-1)*R}, wobei m der Multiplikator-Wert und R der Rotationsgenerator ist. Ist „symmetrisch" gesetzt, so sind die resultierenden Rotationen {-(m-1)*R, … -2*R, -R, 0, R, 2*R, … , (m-1)*R}.

Multiplikation

Hat man zwei Greifmultiplikatoren und multipliziert diese, werden alle Posen des einen mit den Posen des zweiten Multiplikators multipliziert.

Beispiel: Greifmultiplikator M1: Translationsmultiplikator (m=2, x=50mm, y= 0mm, z=0mm) Greifmultiplikator M2: Translationsmultiplikator (m=3, x= 0mm, y=50mm, z=0mm)

M1*M2 enthält folgende Posen (x, y, z; Roll, Nick, Gier):

```
(  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)
```

Generell ist die Anzahl der Posen im Produkt zweier Greifmultiplikatoren gleich dem Produkt der Anzahl der Posen der einzelnen Greifmultiplikatoren. (#(M1*M2) = #(M1) * #(M2))

Multiplikator-Baum

Es besteht die Möglichkeit, dass an einem Produkt mehrere Gruppen von Greifposen definiert werden sollen, wobei jede Gruppe wiederum aus mehreren Multiplikatoren besteht. Dies ist im Multiplikator-Baum möglich: Durch Hinzufügen eines neuen Greifmultiplikators auf der obersten Ebene, wird eine neue Gruppen von Greifposen definiert. Durch Hinzufügen eines Greifmultiplikators A zu einem bereits bestehenden Multiplikator B wird dieser mit B multipliziert. Das Hinzufügen neuer Gruppen ist auf jeder Ebene möglich.

Beispielbaum:

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

Angenommen A, B, C sind Einzelgriffe, R ist eine Rotation mit Multiplikator 5 (nicht-symmetrisch) und T1, T2 sind Translationen mit Multiplikator 3, so ergeben sich folgende Griffe:

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

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
- B * D
- B * T2 * D
- B * T2^2 * D
```

Globale Multiplikatoren

Hier können Multiplikatoren definiert werden, die an jedes Blatt des Greifmultiplikator-Baums angehängt werden.

Beispiel:

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

Dies ergibt folgenden Baum:

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

2.10.6 Kollisionsprüfung der Greifposen

Sollte ein Griff-Multiplikator zu Griffen führen die mit dem Produkt kollidieren, so wird über dem Griff-Baum die Anzahl der kollidierenden Griffe neben einem roten Balken angezeigt. Innerhalb des Baums werden die kollidierenden Griffe dem Griff-Multiplikator zugeordnet, welcher sie verursacht. Durch ausklappen der Multiplikatoren wird die Zuordnung der erkannten Kollisionen detaillierter. Als Beispiel dienen die folgenden beiden Screenshots mit unterschiedlichen Ausklapptiefen.

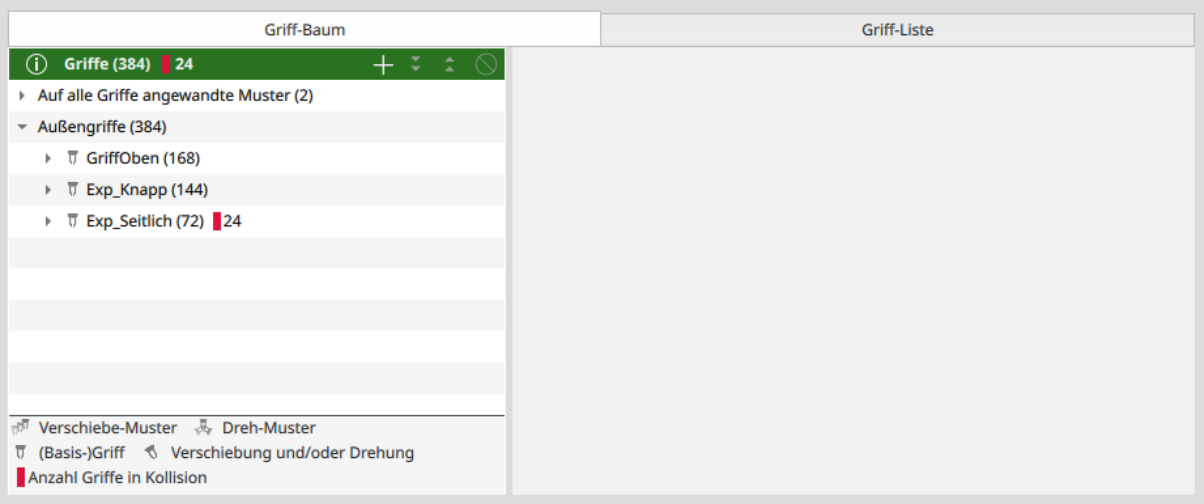


Abb. 19: Screenshot Griff-Baum mit Anzahl der kollidierenden Griffe (Ausklapptiefe: 1)

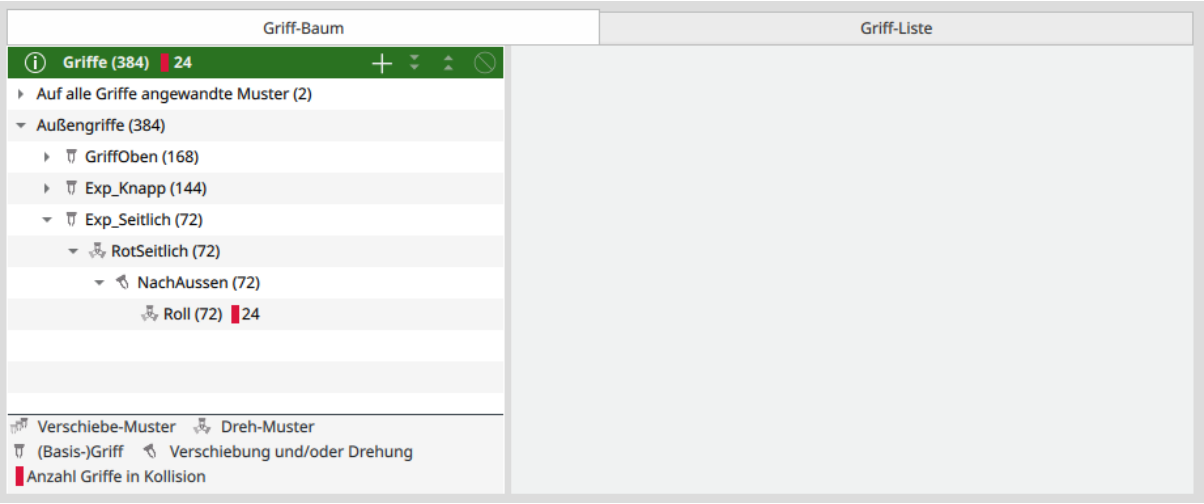


Abb. 20: Screenshot Griff-Baum mit Anzahl der kollidierenden Griffe (Ausklapptiefe: 4)

Neben dem Griff-Baum gibt es auch den Reiter Griff-Liste. Diese Liste enthält alle durch den Griff-Baum generierten Griffe. Auch hier werden Kollisionen in der Überschrift mit einem roten Balken und der Anzahl der kollidierenden Griffe angezeigt.

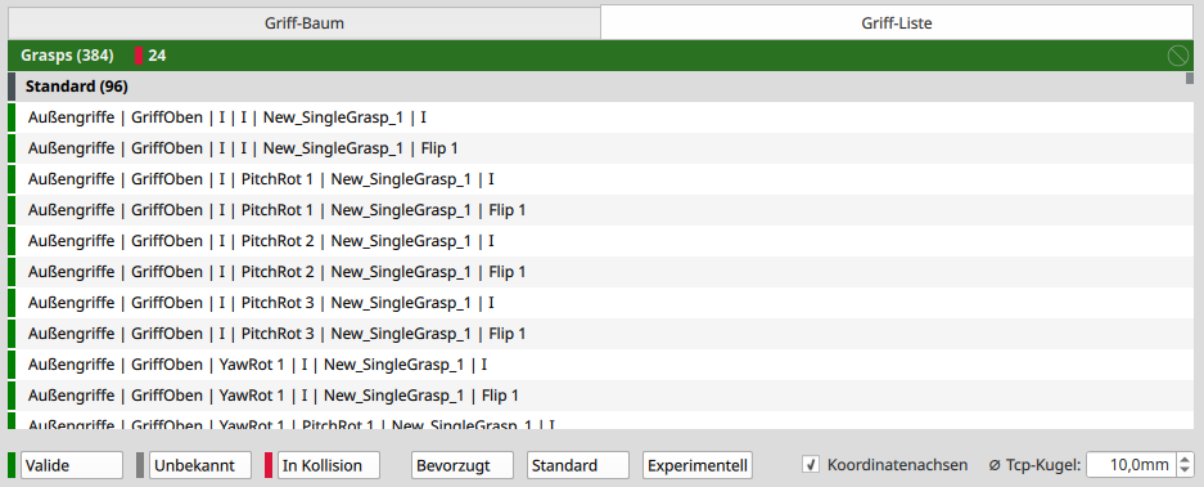


Abb. 21: Screenshot Griff-Liste (ungefiltert)

Unter der Liste befinden sich (wie auch in der *Griff-Inspektion*) Knöpfe zum Filtern der Griffe. Dies kann bspw. verwendet werden, um sich nur die kollidierenden Griffe anzeigen zu lassen. So werden durch Abwählen von „Valide“ die kollisionsfreien Griffe ausgeblendet.

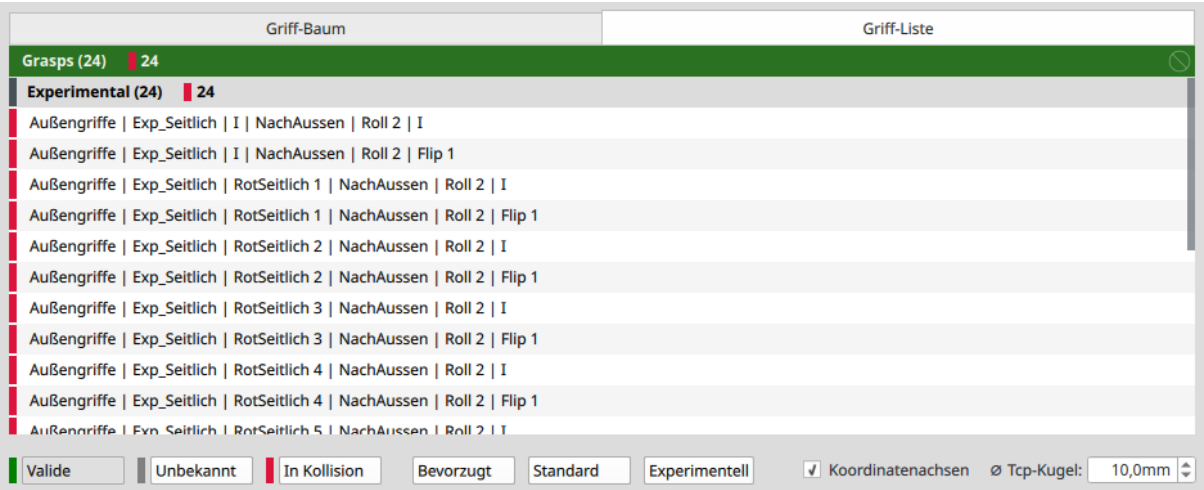


Abb. 22: Screenshot Griff-Liste (nur kollidierende)

Wählt man in der Griff-Liste einen Griff aus, wird dieser in der Visualisierung angezeigt. Bei einem kollidierenden Griff werden die erkannten Kollisionspunkte zwischen Greifer und Produkt durch kleine rote Kugeln markiert.

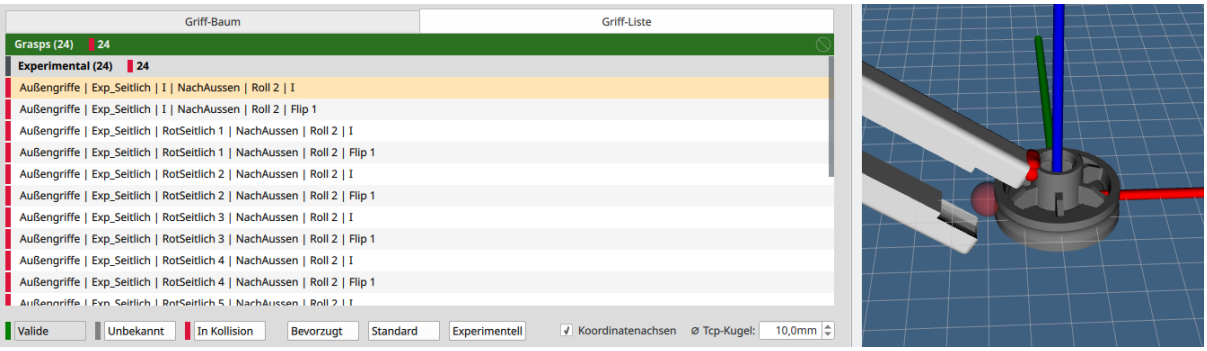


Abb. 23: Screenshot Griff-Liste und Visualisierung der Kollisionspunkte

Einrichten von Griffen bei Wechselgreifern

Dieser Abschnitt ist nur relevant für Zellen mit Wechselgreifer.

Ist für eine Zelle ein Wechselgreifer eingerichtet, sieht die Griffeinstellung wie im folgenden Screenshot aus.

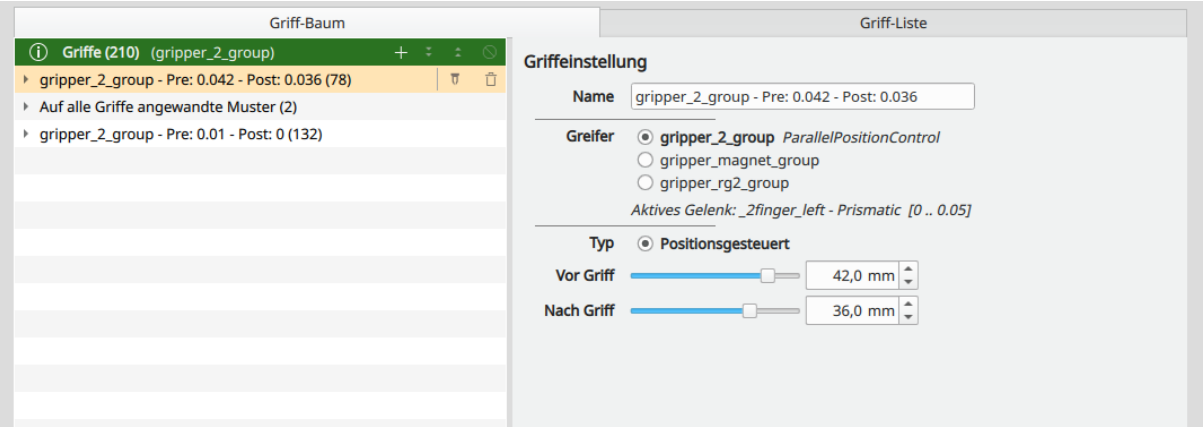


Abb. 24: Einrichten von Griffen bei mehreren Greifern

Für jede Griffeinstellung lässt sich die zugehörige Greifergruppe einstellen. Alle dieser Griffeinstellung zugehörigen Griffe werden der angewählten Greifergruppe zugeordnet.

Es lassen sich für ein Produkt mehrere Griffeinstellung definieren - und somit auch Griffe mit unterschiedlichen Greifern. Bei der Griffsuche wird stets nur nach Griffen gesucht, die dem aktiven Greifer zugeordnet sind.

2.11 Griff-Suche

2.11.1 Überblick

Die Griff-Suche sucht nach kollisionsfreien Griffen inklusive der gesamten Greifbewegung. Werden in einer Region N Objekte gefunden und sind für das entsprechende *Produkt* M Greifpunkte (siehe *Greifposen*) definiert, so gibt es grundsätzlich NxM potenzielle Griffe bzw Greifbewegungen – eine für jede Kombination von Objekt und Greifpunkt. Jede dieser Bewegungen wird zunächst auf potenzielle Kollisionen untersucht. Sobald der weitere Programmablauf einen Griff benötigt (bspw. bei Erreichen eines „Greife-Objekt“-Tasks), wird von allen bis dahin gefundenen validen Bewegungen die „beste“ ausgewählt. Die „Güte“ einer Bewegung richtet sich dabei nach mehreren Kriterien (siehe unten), u.a. nach der Lage-Höhe des zugehörigen Objekts und der Länge der Greifbewegung.

2.11.2 Greifprozess

Die Greifbewegung ist grundsätzlich in folgende Stufen unterteilt:

Anfahren
: Ausgangspose → Anfahrtspose

Greifen
: Anfahrtspose → Greifpose (lineare Fahrt im kartesischen Raum)

Entnehmen
: Greifpose → Entnahmepose (lineare Fahrt im kartesischen Raum)

Zurückfahren
: Entnahmepose → Endpose

Greifen und Entnehmen sind dabei Bewegungen entlang einer Geraden im 3D-Raum, wohingegen sich der Roboter beim An- und Zurückfahren grundsätzlich frei bewegen kann.

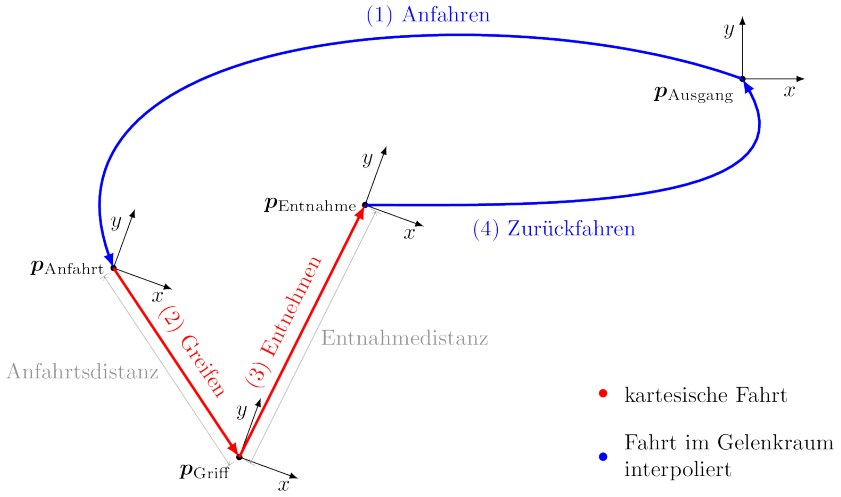


Abb. 25: Schematische Darstellung des Greifprozesses

Ausgangs- und Endpose

Die Ausgangs- und Endpose sind durch die Übergangspose der zur Suche gehörenden Kiste gegeben. Ausgangs- und Endpose sind somit identisch.

Anfahrtspose

Zur Berechnung der Anfahrtspose wird eine bestimmte Distanz von der Greifpose aus „rückwärts“ gerechnet. „Rückwärts“ meint dabei vom Produkt weg, also in z-Richtung im Koordinatensystem der Greifpose (z-Richtung des Urdf-Objekts „gripper_tcp“). Die Distanz zwischen Anfahrts- und Greifpose wird über Parameter eingestellt, siehe Abschnitt *Suchstrategien*.

Greifpose

Wurden N Produkt-Objekte gefunden und es sind für das entsprechende Produkt M *Greifpunkte* definiert, so ergibt sich die Greifpose im Welt-Koordinatensystem aus der Pose des n-ten Produkt-Objekts und aus dem m-ten produktspezifischen Greifpunkt (im Produkt-Koordinatensystem). Die produktspezifischen *Greifpunkte* müssen zuvor manuell angelegt worden sein.

Entnahmepose

Für die Berechnung der Entnahmepose werden unterschiedliches Strategien in einer festen Reihenfolge ausprobiert, siehe Abschnitt *Suchstrategien*. Sollte das Herausziehen in eine der Richtungen nicht möglich sein, wird die nächste Richtung überprüft. Der Greifprozess für eine bestimmte Greifpose gilt erst als gescheitert, wenn keine der Varianten zu einer realisierbaren Greifbewegung führt. Die Distanz zwischen Entnahme- und Greifpose wird über Parameter eingestellt, siehe Abschnitt *Suchstrategien*.

Bemerkung: Im Allgemeinen verhindert die Bahnplanung jegliche Kollisionen zwischen Roboter, Zelle, Szenenobjekten, Produktobjekten und 3D-Punktwolke. Während des Entnehmens (Greifpose → Entnahmepose) werden allerdings auch Bewegungen zugelassen, die Kollisionen zwischen den zu greifenden Objekten beinhalten (nähere Informationen in Abschnitt *Greifbewegungssuche*). Da diese Objekte nicht als feste Szenenobjekte angelegt sind, gilt die Annahme, dass diese beweglich und berührungsunempfindlich sind. Wird eine komplett kollisionsfreie

Bewegung gefunden, wird diese gegenüber einer kollisionsbehafteten stets bevorzugt. Diese Entnahme-Strategie erhöht die Wahrscheinlichkeit eines erfolgreichen Griffs.

2.11.3 Greifbewegungssuche

Die berechnete Gesamttrajektorie besteht aus vier Teiltrajektorien. Greifen und Entnehmen sind dabei Bewegungen entlang einer Geraden im 3D-Raum, wohingegen sich der Roboter beim An- und Zurückfahren grundsätzlich frei bewegen kann.

Die Suche nach einer validen Greifbewegung erfolgt analog zur obigen Einteilung in folgenden Schritten:

(a) Kollisionsprüfung der Greifpose (ohne angehängtem Objekt)

(b) Greifen: lineare (gerade) Fahrt im kartesischen Raum (nicht im Gelenkraum)

(c) Anfahrt

(d) Kollisionsprüfung der Greifpose mit angehängtem Objekt

(e) Entnahme

(f) Rückfahrt

Bei (a) und (d) wird jeweils nur eine Pose geprüft, bei den anderen vier Schritten werden die oben erwähnten Teil-Trajektorien erstellt (sofern jeweils ein valider Plan gefunden werden kann). Für die einzelnen Teiltrajektorien gibt es wiederum verschiedene Suchansätze. Findet die Suche für eine Stufe keine valide Trajektorie, so wird die Greifbewegung verworfen. Die verschiedenen Suchstrategien für die Teiltrajetorien sind im Folgenden aufgeführt:

(a) Kollisionsprüfung der Greifpose (ohne angehängtem Objekt)

(b) Greifen: lineare (gerade) Fahrt im kartesischen Raum (nicht im Gelenkraum)

(c) Anfahren

  (i) per Interpolation im Gelenkraum

  (ii) per Freier Bahnplanung

(d) Kollisionsprüfung der Greifpose mit angehängtem Objekt

(e) Entnahme (lineare (gerade) Fahrt im kartesischen Raum (nicht im Gelenkraum))

  (i) kollisionsfrei

    - in Greif-Richtung (*ExtractInGraspDirection*)
    - nach oben (*ExtractUpwards*)
    - in Richtung der Endpose (*ExtractTowardsFinalPose*)
    - in Richtung Kistenmitte im nutzerdefinierten Winkel zur (x,y)-Ebene (*ExtractTowardsCenter<Winkel[°]>*)

  (ii) *Objekt/Octomap-Kollisionen* sind erlaubt

    - in Greif-Richtung (*ExtractInGraspDirection_AllowOOCollisions*)
    - nach oben (*ExtractUpwards_AllowOOCollisions*)
    - in Richtung der Endpose (*ExtractTowardsFinalPose_AllowOOCollisions*)
    - in Richtung Kistenmitte im nutzerdefinierten Winkel zur (x,y)-Ebene (*ExtractTowardsCenter<Winkel[°]>_AllowOOCollisions*)

  (iii) *Überlappkollisionen* sind erlaubt

    - in Greif-Richtung (*ExtractInGraspDirection_AttObjObjColl*)
    - nach oben (*ExtractUpwards_AttObjObjColl*)
    - in Richtung der Endpose (*ExtractTowardsFinalPose_AttObjObjColl*)

- in Richtung Kistenmitte im nutzerdefinierten Winkel zur (x,y)-Ebene (*ExtractTowardsCenter<Winkel[°]>_AttObjObjColl*)

Die Reihenfolge der Entnahme-Funktionen kann über die *Entnahmereihenfolge* (siehe Abschnitt *Suchstrategien*) festgelegt werden. Dabei werden nur die kollisionsfreien Funktionen eingetragen. Die anderen werden automatisch generiert.

(f) Zurückfahren

(i) per Interpolation im Gelenkraum (*RetractInterpolating*)

(ii) per Freier Bahnplanung (*RetractFreely*)

Bemerkung: Die Entnahme-Trajektorien *ExtractTowardsCenter10* und *ExtractTowardsCenter40* sind standardmäßig deaktiviert. Sie können auf der Parameter-Seite (*Suchstrategien*) eingeschaltet werden über das Setzen der Parameter:

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

Tab. 1: **Übersicht der verschiedenen Entnahme-Strategien**

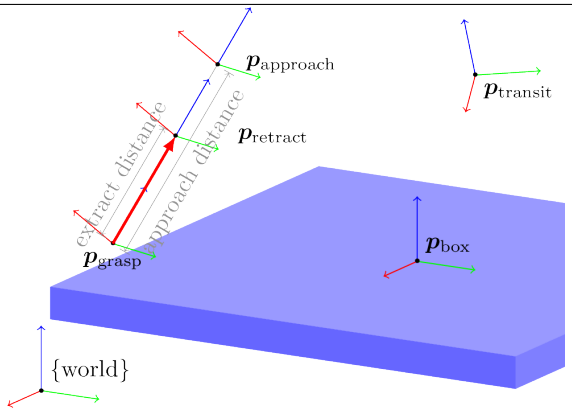


Abb. 26: *ExtractInGraspDirection*

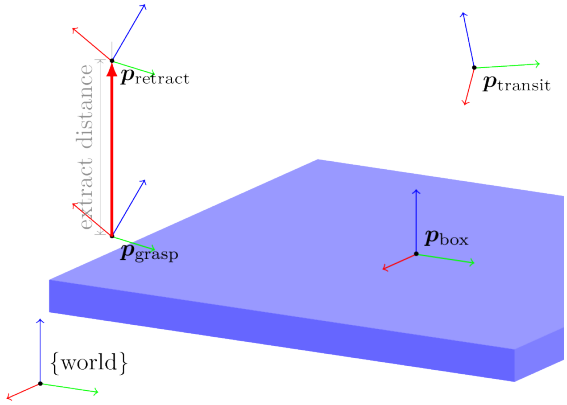


Abb. 27: *ExtractUpwards*

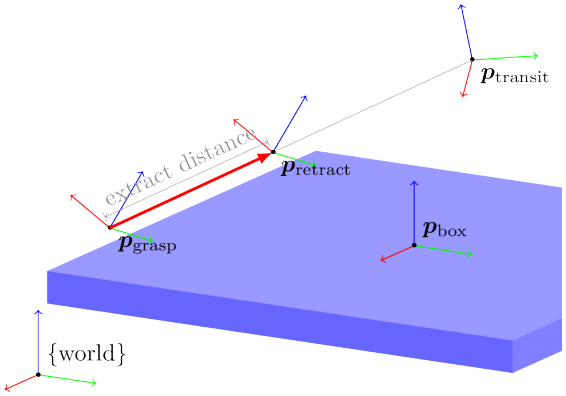


Abb. 28: *ExtractTowardsFinalPose*

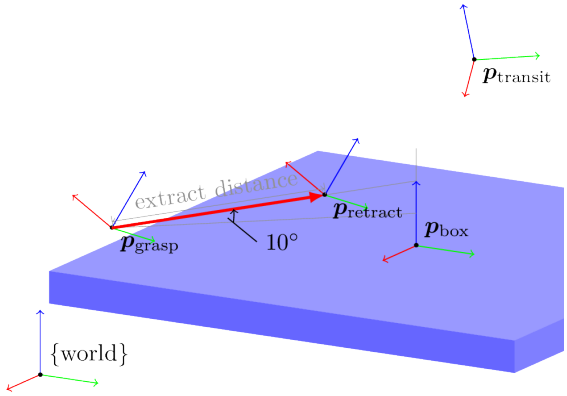


Abb. 29: *ExtractTowardsCenter10*

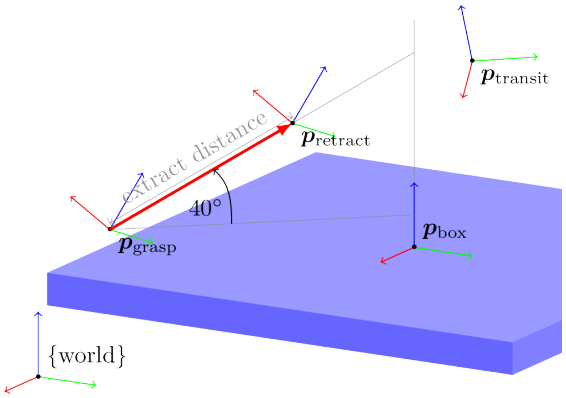


Abb. 30: *ExtractTowardsCenter40*

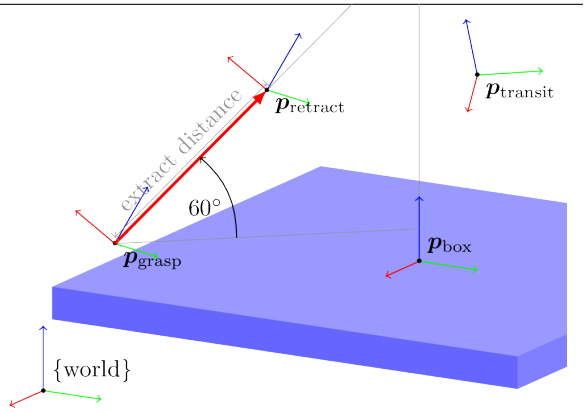


Abb. 31: *ExtractTowardsCenter60*

Objekt/Octomap-Kollisionen

Dies sind Kollisionen des zu greifenden Objekts mit entweder

(a) anderen Produkt-Objekten (aber nicht mit Szenenobjekten), oder mit

(b) der Octomap.

Diese Art von Kollisionen kann über den Parameter *allow_OO_collisions_during_extraction* erlaubt werden, siehe Abschnitt *Suchstrategien*.

Überlappkollisionen

Dies sind Kollisionen des zu greifenden Objekts bereits an der Greifpose. Aufgrund von Ungenauigkeiten in der Lageerkennung der Objekte kann es vorkommen, dass sich zwei Objekte in der Szene leicht überlappen („kollidieren“). Wenn dies reell nicht der Fall sein kann, sondern nur ein Artefakt der Objekterkennung ist, können diese Kollisionen ignoriert werden. Sind Überlappkollisionen bei der Entnahme erlaubt, so werden die Kollisionen, die bereits am Greifpunkt auftreten, entlang der Entnahmetrajektorie solange ignoriert, bis sie nicht mehr auftreten. Treten sie danach erneut auf, werden sie nicht mehr ignoriert. Die Kollisionstiefe, bis zu der Überlappkollisionen erlaubt sind, lässt sich in der *pick_search.yaml* über die Parameter:

- *max_allowed_collision_depth* allgemein einstellen bzw. über
- *specific_max_allowed_collision_depths* Produkt-spezifisch einstellen.

Überlappen sich zwei Objekte so stark, dass die Kollisionstiefe größer als der angegebene Threshold ist, so werden diese Kollisionen nicht ignoriert, auch wenn sie bereits am Greifpunkt auftreten.

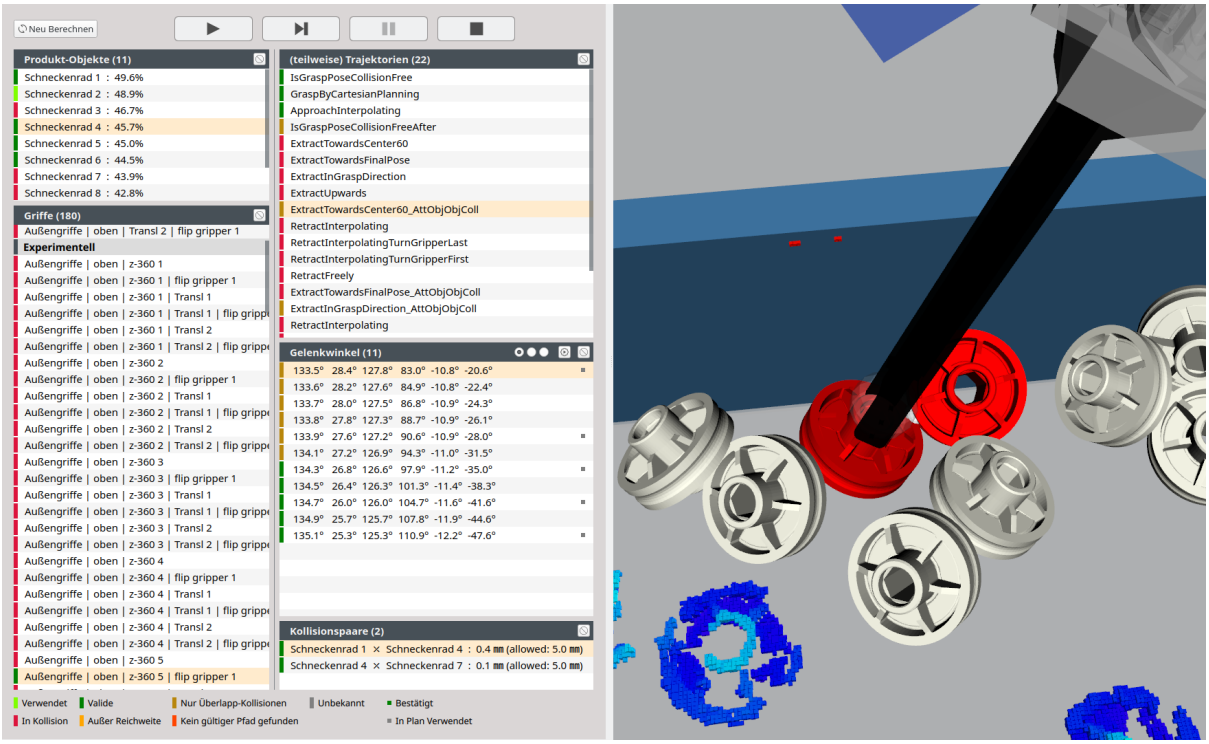


Abb. 32: Screenshot Beispiel für Überlappkollisionen

Interpolations-Parameter

Um Fahrten kleinschrittig zu prüfen, wird zwischen zwei Trajektorienpunkten ein weiterer Punkt eingefügt. Dies wird solange wiederholt, bis benachbarte Punkte nah genug beisammen liegen. Die Grenzwerte, ab wann zwei Punkte als „nah genug“ gelten, lassen sich über folgende Parameter einstellen (gezeigt werden die Default-Werte):

```
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°
```

Die *catesian_max_…_diff*-Parameter werden verwendet um die Schrittgröße der kartesischen Fahrten zu berechnen. Dies beinhaltet alle Schritte, die auch an den Roboter gesendet werden. Die *interpolation_max_…_diff*-Parameter werden für alle Interpolationsfahrten verwendet, sowie für zusätzliche Interpolationsschritte zwischen den kartesischen Punkten einer kartesischen Fahrt.

Kleinere Werte, insbesondere für die *interpolation_max_…_diff*-Parameter führen folglich zu mehr Trajektorienpunkten, was wiederum zu einer höheren Berechnungszeit führt.

Ausschließen bestimmter Suchfunktionen

Es ist möglich, einzelne Entnahme- (Extract-) oder Rückfahr- (Retract-)Funktionen abzuschalten. Hierzu dient der Parameter *search_fn_blacklist* in der *pick_search.yaml*. Er erwartet eine Liste mit Funktionsnamen, die nicht verwendet werden sollen. Wird eine der kollisionsfreien Entnahmefunktionen aufgeführt (bspw. *ExtractInGraspDirection*), werden automatisch auch alle zugehörigen Entnahmefunkionen entfernt (*ExtractInGraspDirection_AllowOOCollisions*, *ExtractInGraspDirection_AttObjObjColl*).

Suchstrategien

Griff

- ***Griffe mit zu hohem Handgelenks-Winkel verwerfen***
  Griffe herausfiltern, deren Handgelenks-Winkel höher ist als der maximale Winkel, den das Handgelenk zu Vertikalen haben darf
- ***Maximaler Winkel des Handgelenks (relativ zur Vertikalen)***
  maximaler Winkel, den das Handgelenk zu Vertikalen haben darf
- ***Griffe mit zu hohem Tcp-Winkel verwerfen***
  Griffe herausfiltern, deren Tcp-Winkel höher ist als der maximale Winkel, den der Tcp zu Vertikalen haben darf
- ***Maximaler Winkel des Tcp (relativ zur Vertikalen)***
  maximaler Winkel, den der Tcp zu Vertikalen haben darf

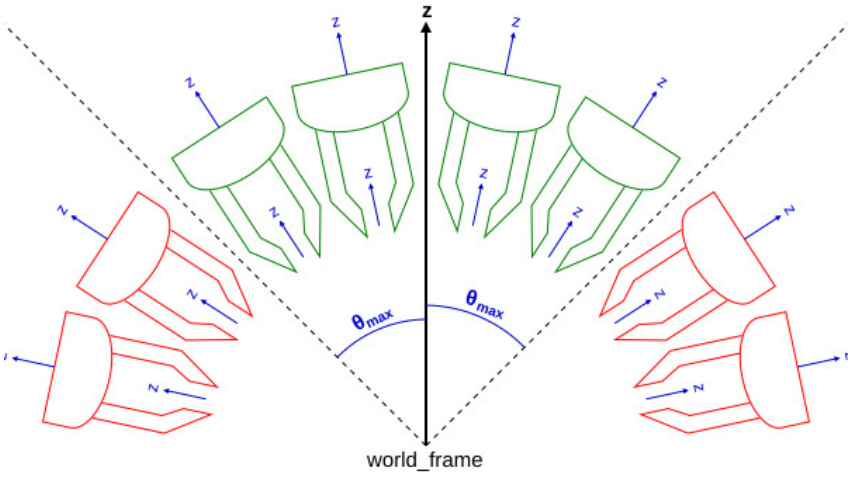


Abb. 33: Maximaler Winkel des Handgelenks/Tcp (relativ zur Vertikalen)

Greifen

- ***Greifdistanz***
  (maximale) Strecke zur Greifpose, die beim Greifen kartesisch gefahren wird
- ***Greifmindestanteil***
  minimaler Anteil der Greifdistanz, der erreicht werden muss (kollisionsfrei)

Interpolations-Funktionen

Es gibt drei Arten von Interpolationsfunktionen:

- ***ohne Zwischenpunkte***
  Die Standard-Interpolation, bei der die gesamte Fahrt von Start- zu Anfartspose (bzw bei der Rückfahrt Entnahme- zu Zielpose) im Gelenkraum interpoliert wird.
- ***Kartesische Zwischenpunkte***
  Bei der kartesischen Interpolation werden zwischen Start- & Anfahrtspose (bzw. zwischen Entnahme- und Zielpose) jeweils Zwischenpunkte hinzugefügt, die in gleichem Abstand auf gerader Linie zwischen den jeweiligen beiden Endpunkten liegen. Über den Unterparameter *Anzahl Zwischenpunkte* kann man die Anzahl dieser Zwischenpunkte festlegen. Die Zwischenpunkte definieren jeweils nur eine Position, aber keine Orientierung des Tcp. Unter dem Punkt *Rotation* gibt es drei Einstell-Möglichkeiten, wie sich die Orientierung des Tcp während der Fahrt verhalten soll:

  a) *Am Übergang*: Die Orientierung bleibt während der Fahrt gleich und dreht sich nur an der Übergangspose (Startpose bei der Anfahrt / Zielpose bei der Rückfahrt)

  b) *Während Fahrt*: Die Orientierung wird während der Fahrt interpoliert (wie die Position auch).

  c) *In der Kiste*: Die Orientierung bleibt während der Fahrt gleich und dreht sich nur an der Anfahrts- bzw Entnahme-Pose.
- ***Gelenke aufteilen***
  Mit dieser Interpolationsart können einzelne Gelenke vor anderen interpoliert werden, indem die Interpolation in zwei Segmente unterteilt wird. Jedes Gelenk kann im ersten, im zweiten oder über beide Segmente interpoliert werden. Wenn alle Gelenke über beide Segmente interpoliert werden, erhält man die übliche Interpolation (keine Unterteilung).

  Wenn mindestens ein Gelenk über beide Segmente interpoliert wird, wird ein Prozentsatz benötigt, der angibt, an welchem Punkt relativ zur Gesamt-Interpolation die anderen Gelenkbewegungen anfangen/aufhören sollen (siehe Diagram weiter unten). Es bedarf daher zweier Listen von Gelenk-Indizes (eine für jedes Segment) sowie in manchen Fällen einen Prozentsatz (wenn ein Gelenk in beiden Listen vorkommt). Dabei ist zu beachten: Wenn ein Gelenk-Index in keiner der beiden Listen vorkommt, wird es automatisch in beide hinzugefügt.

  Zum Beispiel führt

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

  zu folgender Interpolation:

```
       Am Übergang       Zwischenpunkt                  In Kiste
           |                  |                            |
Gelenke
1          |------------------|
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
2           |------------------|
3           |--------------------------------------------|
4           |--------------------------------------------|
5                              |---------------------------|
6                              |---------------------------|
```

Wie zu sehen ist, wird Gelenk 4 genauso wie Gelenk 3 über beide Segmente interpoliert, da es in keiner der Listen steht und daher beiden hinzugefügt wird. Desweiteren ist zu sehen, dass die Zwischen-Gelenkposition aus folgenden Einzel-Positionen besteht:

(a) den Gelenkpositionen in der Kiste für Gelenke 1 und 2,

(b) den Gelenkpositionen am Übergang für Gelenke 5 und 6, und

(c) den Zwischenpositionen für Gelenke 3 und 4, wobei jede Zwischenposition eine 40%-Interpolation zwischen Am-Übergang-Position und In-Kiste-Position dieser Gelenke ist.

Ein realistischeres Beispiel ist

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

Hier wird das letzte Gelenk (das Greifer-Gelenk) an der Übergangspose gedreht und mit den restlichen Gelenken zur/von der Griffpose gefahren. Dies kann nützlich sein, wenn der Greifer nicht innerhalb der Kiste gedreht werden soll. Eine weitere Option mit ähnlichem Effekt ist, das Gelenk 5 ebenfalls in die Gelenke an Übergang zu schieben.

Anfahrt

- ***Reihenfolge der Anfahrts-Interpolations-Funktionen***
  Reihenfolge, in der die aufgeführten Interpolations-Funktionen getestet werden. Nicht enthaltene Interpolationsfunktionen werden nicht getestet. Für eine Beschreibung der Interpolations-Funktionen siehe *Interpolations-Funktionen*.
- ***Kein Freier Planer für Anfahrt***
  Verhindert, dass der freie Planer verwendet wird. In diesem Fall werden nur Interpolationsfunktionen getestet.

Entnahme

- ***Entnahmedistanz***
  (maximale) Strecke zur Greifpose, die bei der Entnahme kartesisch gefahren wird
- ***Entnahmemindestanteil***
  minimaler Anteil der Entnahmedistanz, der erreicht werden muss (kollisionsfrei)
- ***Kollisionen mit anderen Objekten & Octomap erlauben***
  erlaube Kollisionen zwischen angehängtem Objekt und anderen Objekten sowie der Octomap während der Entnahme
- ***Max.-Tiefen-Kollisionen erlauben***
  erlaube Kollisionen am Greifpunkt bis zur eingestellten Kollisionstiefe während der Entnahme
- ***Maximale erlaubte Kollisionstiefe***
  maximale Kollisionstiefe, die während der Entnahme erlaubt ist
- ***Spezifische maximale erlaubte Kollisionstiefe***
  erlaubt das Setzen spezifischer maximaler Kollisionstiefen für Paare von Objekten. Mögliche Objekte sind alle Produkte, Kisten und Hindernisse.
- ***Reihenfolge der Entnahme-Funktionen***
  Reihenfolge, in der verschiedene Entnahme-Funktionen geprüft werden sollen
- ***Entnahme Anzahl XY-Richtungen***
  Bei den Entnahme-Funktionen „Im Winkel“ und „Seitlich“ werden N Richtungen im Kreis gleichverteilt, wobei 0° in Richtung Kistenmitte zeigt, und für jede Richtung wird eine Entnahmefunktion generiert. Dieser Parameter legt die Anzahl N fest.
- ***Entnahme-Funktionen sortieren nach***
  Ist dieser Parameter gesetzt, wird nicht nach jeder erfolgreichen Entnahme-Funktion zunächst die Rückfahrt getestet, sondern zunächst werden alle Entnahme-Funktionen geprüft und dann nach diesem Kriterium sortiert. Dann werden jeweils die Rückfahrten getestet, bis eine erfolgreiche gefunden wurde.

Rückfahrt

- ***Rückfahrt deaktivieren***
  nur bis zur Rückfahrtspose planen und dort anhalten
- ***Reihenfolge der Rückfahrts-Interpolations-Funktionen***
  Reihenfolge, in der die aufgeführten Interpolations-Funktionen getestet werden. Nicht enthaltene Interpolationsfunktionen werden nicht getestet. Für eine Beschreibung der Interpolations-Funktionen siehe *Interpolations-Funktionen*.
- ***Kein Freier Planer für Rückfahrt***
  Verhindert, dass der freie Planer verwendet wird. In diesem Fall werden nur Interpolationsfunktionen getestet.

Positionsgleichheit

- ***Gelenkraum-Interpolation Max. Translationsunterschied***
  maximaler Translationsunterschied zwischen zwei Punkten um keinen weiteren Zwischenpunkt zu brauchen (in der Gelenkraum-Interpolation). Wirkt sich darauf aus, wie kleinschrittig eine Fahrt auf Kollisionen geprüft wird.
- ***Gelenkraum-Interpolation Max. Rotationsunterschied***
  maximaler Rotationsunterschied zwischen zwei Punkten um keinen weiteren Zwischenpunkt zu brauchen (in der Gelenkraum-Interpolation) Wirkt sich darauf aus, wie kleinschrittig eine Fahrt auf Kollisionen geprüft wird.
- ***Kartesische Interpolation Max. Translationsunterschied***
  maximaler Translationsunterschied zwischen zwei Punkten um keinen weiteren Zwischenpunkt zu brauchen (in der kartesischen Interpolation). Wirkt sich darauf aus, wie kleinschrittig die kartesische Fahrt erzeugt und an den Roboter gesendet wird.
- ***Kartesische Interpolation Max. Rotationsunterschied***
  maximaler Rotationsunterschied zwischen zwei Punkten um keinen weiteren Zwischenpunkt zu brauchen (in der kartesischen Interpolation) Wirkt sich darauf aus, wie kleinschrittig die kartesische Fahrt erzeugt und an den Roboter gesendet wird.

Die Parameter zur kartesischen Interpolation wirken sich ggf auf die Fahrgeschwindigkeit des Roboters aus. Bei kleineren Schwellwerten braucht es mehr Punkte, was ggf zu einer langsameren Fahrt führen kann.

Die Parameter zur Gelenkraum-Interpolation wirken sich auf die Berechnungszeit der Griffsuche aus. Der Wert sollte an die Größe des Produkts, die Kiste und des Roboters angepasst werden. Bei dünnen Blechen eigenet sich ein Wert von Blechdicke/3, bei großen Teilen reicht möglicherweise ein Kollisionscheck pro cm.

Kontakte

- ***Anzahl Kontakte während Kollisionsprüfung***
  Maximale Gesamtzahl der Kontakte, die während einer einzelnen Kollisionsprüfung berechnet werden. Sollte groß (~100k) sein, wenn Kollisionen mit der Octomap wahrscheinlich sind, andernfalls kann der Parameter kleiner sein (~100). Größere Werte können zu einer langsameren Griffsuche führen.
- ***Max. Anzahl Kontakte pro Kollisionspaar***
  Maximale Anzahl der Kontakte pro Kollisionspaar, die während einer einzelnen Kollisionsprüfung berechnet werden. Sollte groß (~10k) sein, wenn Kollisionen mit der Octomap wahrscheinlich sind, andernfalls kann der Parameter kleiner sein (~30). Größere Werte können zu einer langsameren Griffsuche führen.

Suchstrategien

Griffsuche

- ***Suchstrategien***
  Suchstrategien-Parametersätze für jede Greifkategorie
- ***Griffsuchen-Timeout***
  Timeout, nach dem die Suche abgebrochen wird
- ***Erfolgreiche Griffe pro Objekt vor Rückstellung***
  Wenn diese Anzahl an erfolgreichen Griffen für ein Objekt erreicht wurde, werden Griff anderer Objekte bei der restlichen Suche bevorzugt gesucht.
- ***Max Erfolgreiche Griffe Gesamt***
  Nach dieser Anzahl von erfoglreichen Griffen (insgesamt) wird die Griffsuche angehalten.

Bedingungen

- ***Bedingungen für Griff***
  Bedingungen, die ein Griff erfüllen muss, um angenommen zu werden, siehe *Bedingungen*.

Zusatzzeit

- ***Zusatzzeit***
  Zusatzzeit, die die Suche nach dem ersten gefundenen Griff läuft, solange die Zusatzzeit-Bedingungen vom besten Griff noch nicht erfüllt worden sind
- ***Bedingungen für zusätzliche Suchzeit***
  Bedingungen, die ein Griff erfüllen muss, um direkt (nicht erst nach der Zusatzzeit) verwendet zu werden, siehe *Bedingungen*.

Sortierung:

Für eine Beschreibung der einzelnen Sortierkriterien siehe *Beste Griffe*.

- ***Sortier-Kriterien***
  (Reihenfolge der) Sortierkriterien der erfolgreichen Griffe
- ***Bin-Größen***
  Bin-Größen für die Binning-Kriterien
- ***Gewichte***
  Gewichte für die *gewichtete Summe der Bins* (eines der potenziellen Sortierkriterien) Wichtig ist das Verhältnis der einzelnen Gewichte, nicht die Höhe der Zahlen. Die Gewichte sind nur relevant, wenn die *gewichtete Summe der Bins* in der Liste der Sortierkriterien auftaucht.

Objektsuche Überspringen

- ***Überspringen der nächsten Objektsuche erlauben***
  das Überspringen der nächsten Objektsuche erlauben, sofern die folgenden Bedingungen erfüllt sind
- ***Mindestzahl gefundener Griffe***
  minimale Anzahl valider Griffe (insgesamt) um das Überspringen der nächsten Objektsuche zu erlauben
- ***Mindestzahl gefundener Griffe pro Objekt***
  minimale Anzahl valider Griffe (pro Objekt) um das Überspringen der nächsten Objektsuche zu erlauben
- ***Mindestzahl gefundener Objekte***
  minimale Anzahl valider Objekte um das Überspringen der nächsten Objektsuche zu erlauben

Fahrdistanz Schwellwert

Der Fahrdistanz-Schwellwert wird anhand der Fahrdistanzen der vorherigen erfolgreichen Griffe ermittelt. Aus diesen wird der Mittelwert sowie die Standardabweichung berechnet. Der Schwellwert liegt bei (Mittelwert + *Anzahl Std.-Dev.* * Standardabweichung) Desweiteren gibt es den Parameter *Fester Schwellwert*, mit dem ein fester Schwellwert gesetzt werden kann.

- ***Schwellwert-Default***
  Default-Wert für den Fahrdistanz-Schwellwert (wird für binäres Binning in der Griffsortierung verwendet)
- ***Anzahl Std.-Dev.***
  Anzahl der Standardabweichungen, die der Schwellwert über dem Durchschnitt liegen muss
- ***Fester Schwellwert***
  fester Wert für den Schwellwert (macht die beiden Parameter über diesem obsolet)

Positionsgleichheit

- ***Kistenpose Max. Translationsunterschied***
  bei Kistensuche: Liegt der Translationsunterschied zur vorherigen Pose höher als dieser Schwellwert, wird die Übergangspose neu berechnet und alle bisherigen Griffe werden verworfen.
- ***Kistenpose Max. Rotationsunterschied***
  bei Kistensuche: Liegt der Rotationsunterschied zur vorherigen Pose höher als dieser Schwellwert, wird die Übergangspose neu berechnet und alle bisherigen Griffe werden verworfen.
- ***Übergangspose Max. Translationsunterschied***
  bei Kistensuche: Liegt der Translationsunterschied zur vorherigen Pose höher als dieser Schwellwert, wird die Übergangspose neu berechnet und alle bisherigen Griffe werden verworfen.
- ***Übergangspose Max. Rotationsunterschied***
  bei Kistensuche: Liegt der Rotationsunterschied zur vorherigen Pose höher als dieser Schwellwert, wird die Übergangspose neu berechnet und alle bisherigen Griffe werden verworfen.

- ***Teilpose Max. Translationsunterschied***
  bei Teilsuche: Maximaler Translationsunterschied zur Pose eines vorher gefundenen Teils, um als dasselbe Teil eingestuft zu werden. Dieser Wert entscheidet nur über die Reihenfolge der Griffe und kann relativ groß eingestellt werden, um z.B. verkeilte Teile nicht zu häufig erneut zu probieren. Dafür kann das Sortier-Kriterium `Anzahl Greifversuche an Objekt` (siehe *Verfügbare Sortier-Kriterien*) sowohl bei der Sortierung als auch bei der Extrazeit verwendet werden.
- ***Teilpose Max. Rotationsunterschied***
  bei Teilsuche: Maximaler Rotationsunterschied zur Pose eines vorher gefundenen Teils um als dasselbe Teil eingestuft zu werden. Dieser Wert entscheidet nur über die Reihenfolge der Griffe und kann relativ groß eingestellt werden, um z.B. verkeilte Teile nicht zu häufig erneut zu probieren. Dafür kann das Sortier-Kriterium `Anzahl Greifversuche an Objekt` (siehe *Verfügbare Sortier-Kriterien*) sowohl bei der Sortierung als auch bei der Extrazeit verwendet werden.
- ***Griff Max. Translationsunterschied***
  bei Teilsuche: Maximaler Translationsunterschied zur Pose eines vorher gefundenen Teils, um die Griffe jenes Teils wiederverwenden zu können.
- ***Griff Max. Rotationsunterschied***
  bei Teilsuche: Maximaler Rotationsunterschied zur Pose eines vorher gefundenen Teils, um die Griffe jenes Teils wiederverwenden zu können. Dieser Wert fließt in die Griffgenauigkeit ein und sollte nicht zu hoch gewählt werden.

2.12 Szenenverwaltung

2.12.1 Überblick

Szenen-Objekte beschreiben Objekte und Regionen, die nicht in der statischen Beschreibung der Zelle enthalten sind. Dies können Kisten sein, reine Regionen (sogenannte ROIs, Regions of Interest) oder Hindernisse.

Ein Szenen-Objekt hat immer einen eindeutigen Namen sowie eine (Default-)Pose. Weitere Eigenschaften sind abhängig von der Art des Objektes. Hat ein Objekt eine „Ausdehnung“ (Höhe, Breite, Länge), so ist zu beachten, dass die x-, y- und z-Koordinaten der Default-Pose die Zentrums-Koordinaten der Region / des Objektes sind.

Hinweis: Vor Mikado 3.0 gab die z-Koordinate die Höhe des Bodens des Objektes an. Seit Mikado 3.0 bezieht sich die z-Koordinate auf die Objektmitte.

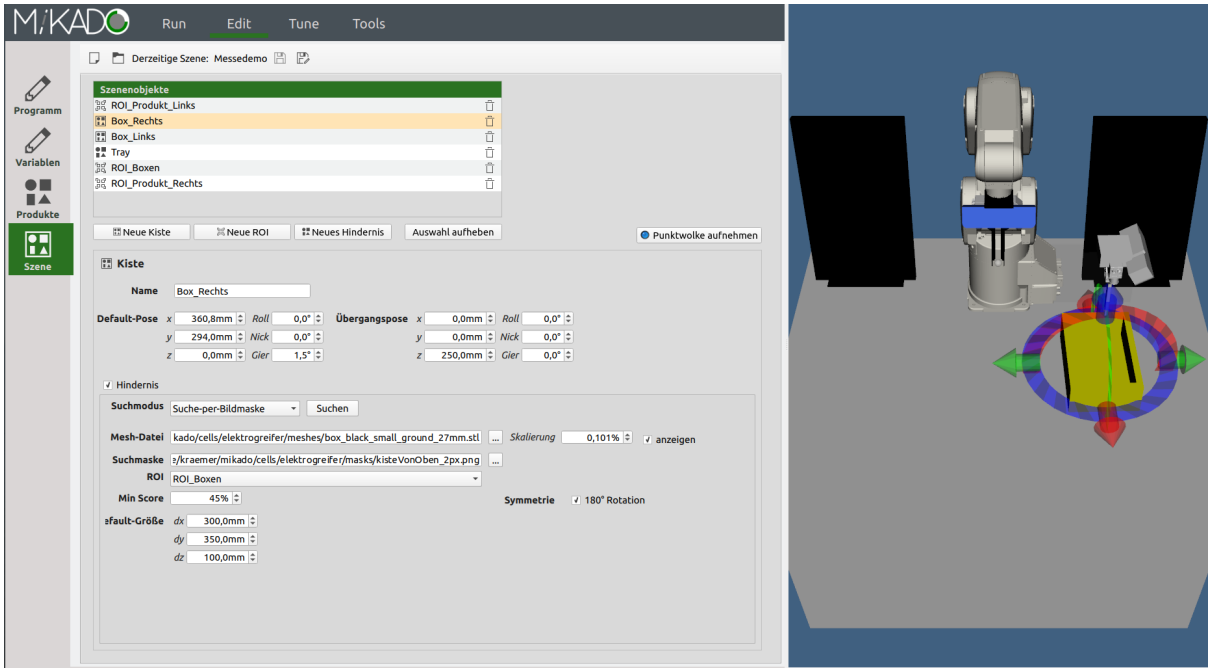


Abb. 34: Screenshot Szenenverwaltung (Beispiel einer Aufnahmekiste)

Die Default-Pose eines Szenenobjekts kann entweder über die Eingabefelder definiert werden oder mithilfe der interaktiven Marker in der Visualisierung. Klicken Sie die farbigen Pfeile an, so kann die Position mit gedrückter Maustaste verschoben werden. Die farbigen Ringe ermöglichen es, die Rotation um die jeweilige Achse des Szenenobjekts zu rotieren.

Hinweis: Die farbigen Steuerelemente können nur dort angefasst werden, wo sie nicht von einem anderen (evtl. teiltransparenten) Objekt verdeckt werden.

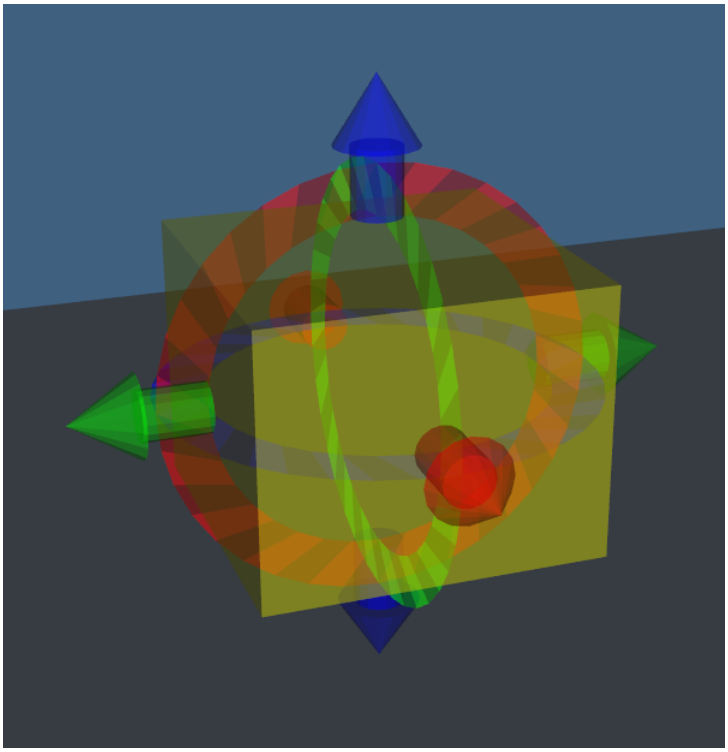


Abb. 35: Interaktiver Marker für Szenenobjekte (Steuerelemente in rot, grün und blau)

Hinweis: Die Eingabefelder definieren die Rotationswerte direkt in Weltkoordinaten. Werden die interaktiven Marker verwendet, werden die Rotationswerte direkt in Roll, Nick, Gier umgerechnet und gegebenenfalls ändern sich dadurch mehrere Werte gleichzeitig. Dies ermöglicht eine intuitivere Definition der Rotationseinstellung über die interaktiven Marker.

2.12.2 Hindernis

Ein **Hindernis** ist ein Kollisions-Objekt in der Szene das bei der Bahnplanung berücksichtigt wird. Es kann als fix betrachtet oder per 2D-Suche gesucht werden.

2.12.3 Kiste

Kisten werden bei der *Griffsuche* verwendet um die Umgebung der zu greifenden Teile zu beschreiben. Sie enthalten die sogenannte **Übergangspose** als Start- und Zielpose für die Greifplanung fungiert. Die **Übergangspose** wird relativ zur Pose der Kiste angegeben, nicht relativ zum Roboter.

Kisten mit Störkontur (d.h. mit aktiviertem **Hindernis**) aus denen Produkte gegriffen werden sollen benötigen immer eine Mesh-Datei für die Kollisionsvermeidung.

Hinweis: Wenn keine physische Kiste vorhanden ist, weil z.B. von einem Tisch gegriffen wird, dann kann der Haken bei **Hindernis** entfernt werden.

Kistensuche

Für Kisten, die nicht immer an exakt die gleiche Stelle gestellt oder fixiert werden gibt es eine Lagenachführung per 2D Suche. Dabei wird in der Regel der obere Kistenrand gesucht, da dieser unabhängig vom Kistenfüllstand erkennbar bleibt, während z.B. die Innenwände und der Boden stark von Produkten oder flexiblen Packstoffen verdeckt werden können.

Für die 2D-Suche muss eine 2D-Maske angegeben werden. Diese Maske kann wie folgt in der Mikado Anwendung erstellt werden:

1. **ROI anlegen** Diese sollte in Z-Ausdehnung hauptsächlich nur den oberen Kistenrand enthalten, und in X- und Y-Richtung den gesamten Bereich abdecken in dem sich die Kiste befinden kann. Die ROI wird ggf. auch im Programmablauf zur erneuten Kistensuche verwendet.

   Bei der Masken-Erstellungsaufnahme sollte die Kiste möglichst orthogonal ausgerichtet im Bild stehen. Die passende ROI unter Suchparameter auswählen.

2. **Visualisieren** Rechts unten den Reiter Kamera auswählen und im Auswahlmenü den Eintrag **projected_heightmap** auswählen. Dann auf **Suchen** klicken.

3. **Speichern** Per Rechtsklick auf das Vorschaubild kann entweder die Rohversion gespeichert werden oder eine automatisch zugeschnittene und freigestellte Variante.

4. **Nachbearbeiten** Die Kistenmaske sollte exakt rechtwinklig im Bild stehen. Variables Rauschen und dadurch fehlerhafte Konturen sollten nachbearbeitet und entfernt bzw. nachgebessert werden. Bei der Suche von Kisten mit variabler Höhe oder mit hohem Füllstand wo Teile in den Bereich der Such-ROI ragen können, sollten diese ggf. Bereiche per alpha-Channel als transparent ausmaskiert werden. Sie gehen dann nicht in die Suchbewertung mit ein. Die zugeschnittene Variante maskiert standardmäßig alle Bildpunkte die nicht den Kistenrand enthalten aus.

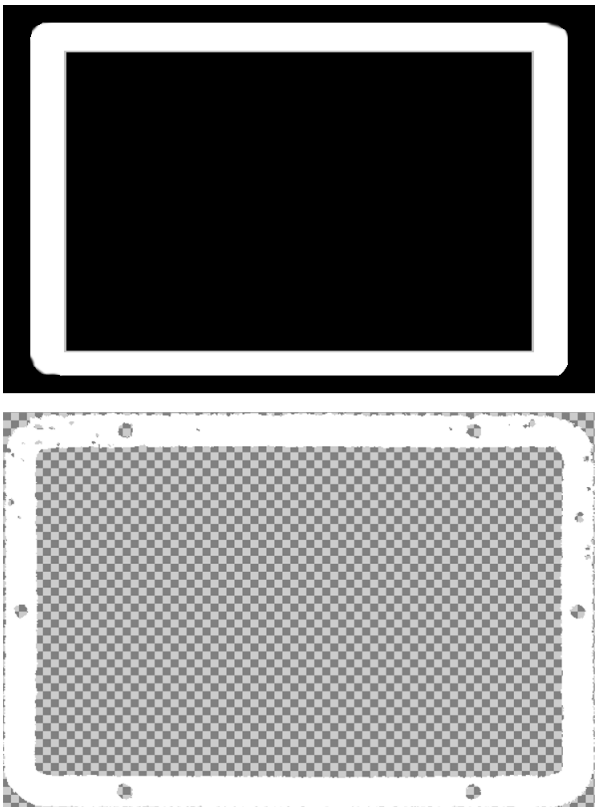


Abb. 36: Zwei Beispiele einer 2D Suchmaske für eine Kiste. Oben: der schwarze Hintergrund geht in die Bewertung mit ein. Unten: der transprarente Hintergrund wird während der Suche ignoriert, an diesen Stellen können beliebige Höhenwerte stehen.

In den hier abgebildeten Beispielmasken wird eine Kiste anhand des oberen Rands gesucht. Der nach oben ragende

Kistenrand hebt sich in der 2D-Projektion der Höhendaten deutlich vom umgebenden Boden ab und kann somit als Suchmaske verwendet werden. Für die Maskensuche ist zu beachten, dass die Mitte der Maske den Mittelpunkt des Objekts darstellt.

Um ein Hindernis als Störkontur in der Kollisionsvermeidung einzuplanen kann eine entsprechende Mesh-Datei angegeben werden. Der Ursprung der Mesh-Datei sollte zentral im Boden des Objekts liegen.

Falls keine Mesh-Datei angegeben wird, kann das Objekt als Quader beschrieben werden: **Platzhalter-Größe**: Länge **dx**, Breite **dy**, Höhe **dz**. Alle Hindernisse mit dem Suchmodus Suche-per-Bildmaske werden bis zum ersten erfolgreichen Suchvorgang im Programmablauf zunächst als Platzhalter-Quader in der Bahnplanung berücksichtigt. Dieser Quader sollte daher den gesamten Bereich abdecken, innerhalb dessen sich das Hindernis potentiell befinden könnte.

Die Parameter der 2D-Suche:

- **ROI**

  Die ROI, die für die Suche des Hindernisses verwendet wird kann in diesem Dropdown ausgewählt werden. Legen Sie zuerst eine *ROI* an, damit diese hier gelistet wird. Achten Sie darauf, dass die ROI in x- und y-Ausdehnung großzügig gewählt ist. Ihre Suchmaske muss für gute Ergebnisse vollständig (sprich inklusive Randbereich) im Bereich der ROI abgebildet werden können.

- **Min Score**

  Konfidenzwert für die erwartete Mindest-Übereinstimmung von Suchmaske und Kamerabild. Die Ausgabe erfolgt unter **Tools > Logs** oder in der Konsole. Ein hoher Wert kann zu fehlgeschlagenen Suchvorgängen führen.

- **180° Symmetrie**

  Weist das Objekt eine 180°-Rotationssymmetrie um die z-Achse auf, so kann diese Option aktiviert werden. Mithilfe dieser Eigenschaft können trotz der Symmetrie eindeutige Übergangsposen definiert werden. Diese Einstelltung empfiehlt sich beispielsweise für symmetrische rechteckige Kisten.

2.12.4 Region of Interest (ROI)

Eine ROI beschreibt eine Region im Raum – in der Regel ohne ein zugehöriges physikalisches Objekt. Sie wird bspw. in der Objektsuche benötigt um den Suchraum einzuschränken, innerhalb dessen nach Objekten gesucht werden soll. Die Verwendung einer ROI dient zum einen der Verbesserungen der Performance und zum anderen der Reduzierung von Fehlfunden (z.B. durch Ausblendung des Untergrunds). Zusätzlich zu ihrer Default-Pose hat eine ROI Größeneigenschaften: Länge **dx**, Breite **dy**, Höhe **dz**. Wird eine ROI angewählt, so hat sie direkt Auswirkung auf die folgenden Aufnahmen und Objektsuchen.

Tab. 2: Funktionsweise einer Region of Interest (ROI)

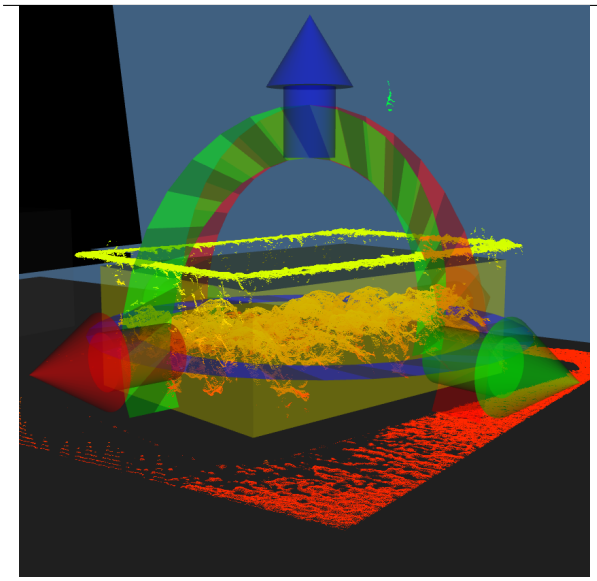


Abb. 37: Punktwolke ohne angewandte ROI.

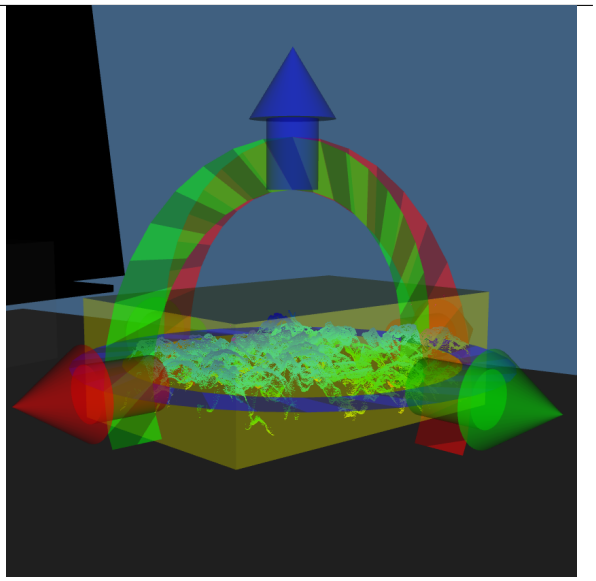


Abb. 38: Punktwolke mit angewandter ROI.

Hinweis: Die Pose einer ROI kann mittels des Tasks *Szenenobjekt-Position setzen* zur Programmlaufzeit angepasst werden. Dies ist beispielsweise nützlich, um die ROI an die Pose einer nachgeführten Kiste anzupassen. Mittels des Tasks *Szenenobjekt-Position zurücksetzen* kann sie auf ihre ursprüngliche Position zurückgesetzt werden.

2.13 Objekterkennungs-Tool

Um die Erkennung Ihrer Produkte in 3D-Daten zu optimieren, können Sie die Parameter der Objektsuche live im Objekterkennungs-Tool verändern. Die Ergebnisse der Änderungen können Sie direkt inspizieren. Zudem bietet dieses Werkzeug verschiedene Ansichten, die es ermöglichen die Ergebnisse zu verstehen und Maßnahmen zur Erzielung besserer Resultate einzuleiten.

Das Objekterkennungs-Tool können Sie verwenden, indem Sie im Hauptmenü auf ‚Tune' → ‚Erkennung' wechseln. Nun sehen Sie links Eingabemöglichkeiten und Buttons, rechts eine Visualisierung der Szene (Falls nicht, wechseln Sie rechts unten auf den Reiter ‚Visualisierung').

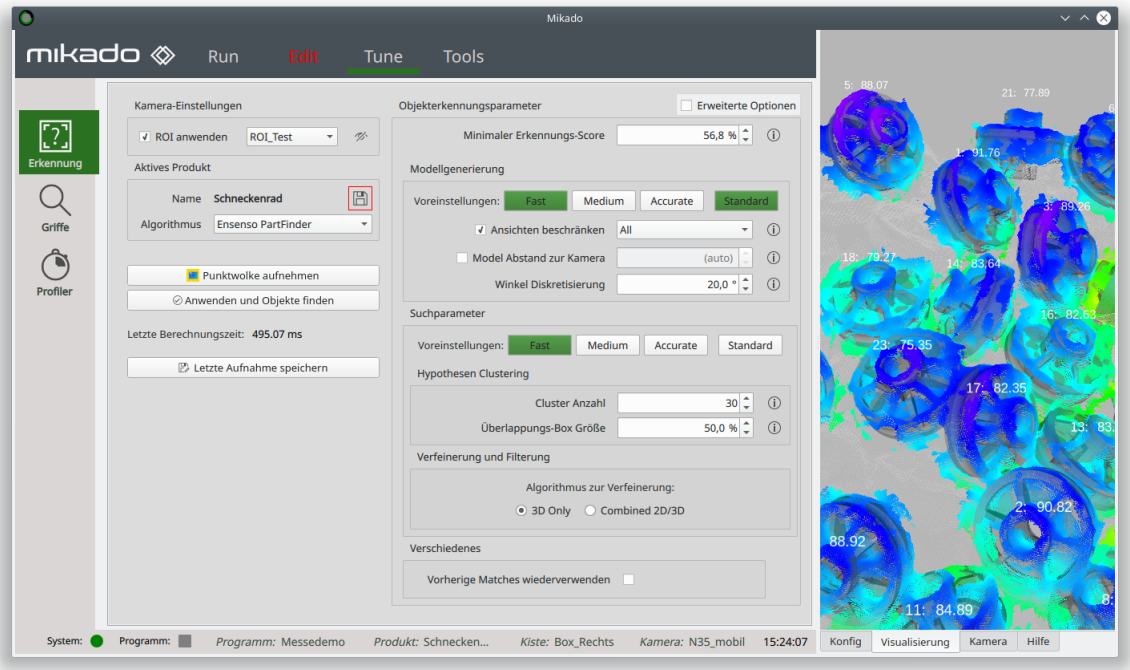


Abb. 39: Screenshot Objekterkennungs-Tool

Bemerkung: Das Tool lässt sich nicht während der Ausführung eines Programms verwenden.

2.13.1 Voraussetzungen

Bevor Sie das Objekterkennungs-Tool verwenden, stellen Sie sicher, dass folgende Schritte durchgeführt sind:

1. **Kamera entsprechend den Anforderungen konfigurieren**
   Bevor Sie die Parameter für die Objektsuche optimieren, sollten Sie sicherstellen, dass die 3D-Daten eine ausreichend gute Qualität aufweisen. Betrachten Sie die aufgenommene Punktwolke, um zunächst zu kontrollieren, ob das Hardware-Setup den Anforderungen entsprechend eingerichtet ist. Sollten die Daten stark verrauscht oder lückenhaft sein, erstellen Sie eine neue, geeignete *Kamerakonfiguration*.

2. **Produkt anlegen**
   Um nach einem Objekt zu suchen, muss dies erst in der *Produktverwaltung* als Produkt angelegt worden sein (siehe auch Tutorial *Ein neues Produkt bis zur Objekterkennung*). Zwingend notwendige Produkt-Parameter für die Benutzung des Objekterkennung-Tools sind:

   - *STL-Datei / STL-Datei (Erkennung)*
   - *Suchalgorithmus*. Wir empfehlen Ensenso Partfinder zu verwenden.
   - Das Produkt aktiv setzen, siehe *Aktives Produkt*

3. **Suchbereich anlegen (optional)**
   Um Fehlfunde zu vermeiden, empfiehlt es sich den Suchbereich (siehe *Region of Interest (ROI)*) auf den erwarteten Bereich zu reduzieren. Beispielsweise können sich keine Bauteile in der Kistenwand oder im Kistenboden befinden, weshalb Wand und Boden von der Suche ausgeschlossen werden sollten.

2.13.2 Erste Schritte

Mit einem Klick auf den Button ‚Punktwolke aufnehmen' erstellt die 3D-Kamera eine neue Aufnahme und zeichnet die Punktwolke in die *Visualisierung der Szene* rechts ein. Nachdem eine Aufnahme gemacht wurde, kann die Objektsuche gestartet werden. Ein Klick auf ‚Anwenden und Objekte finden' löst die Suche nach dem momentan aktiven Produkt aus und stellt die Ergebnisse in der Visualisierung dar. Außerdem wird die Gesamtberechnungszeit der letzten Suche unter den Buttons angezeigt (*Letzte Berechnungszeit*).

2.13.3 Kamera-Auswahl

Hier wird zum einen eine Kamera ausgewählt (vorausgesetzt Mikado Control wird mit mehreren 3D-Kameras eingesetzt), zum anderen kann ein bestimmter Suchbereich mittels einer *Region of Interest (ROI)* eingeschränkt werden.

- ***Aktive Kamera*:**
  Wählen Sie die 3D-Kamera, von der Sie Daten für die Objekterkennung beziehen möchten. Sie können später im Programmablauf natürlich auch andere Kameras für dieselbe Objektsuche verwenden. Diese Option ist nur für Setups verfügbar, in denen mehrere Kameras definiert sind.
- ***ROI anwenden*:**
  Aktivieren Sie diese Option, so wird die generierte Punktwolke bereits vor der Objektsuche gefiltert. Nur Punkte, die innerhalb der angegebenen ROI liegen, werden dargestellt und für die Suche verwendet. Versuchen Sie möglichst restriktive ROIs zu definieren, dies kann eine Erkennung stark beschleunigen und auch für zuverlässigere Ergebnisse sorgen.

2.13.4 Letzte Aufnahme speichern

Möchten Sie aus Diagnose- oder Analysezwecken die Rohdaten der letzten Aufnahme sichern, können Sie den Button ‚Letzte Aufnahme speichern' verwenden. Wählen Sie im Auswahldialog das gewünschte Verzeichnis aus. Es können im selben Verzeichnis auch mehrere Aufnahmen abgelegt werden.

2.13.5 Visualisierung der Szene

Mit den Ansichtseinstellungen lässt sich die Darstellung der Funde und der 3D-Daten verändern. Die Darstellung der Punktwolke vor der Objektsuche ist stets gleich: Die Punkte sind nach ihrer Höhe (Z-Wert) eingefärbt.

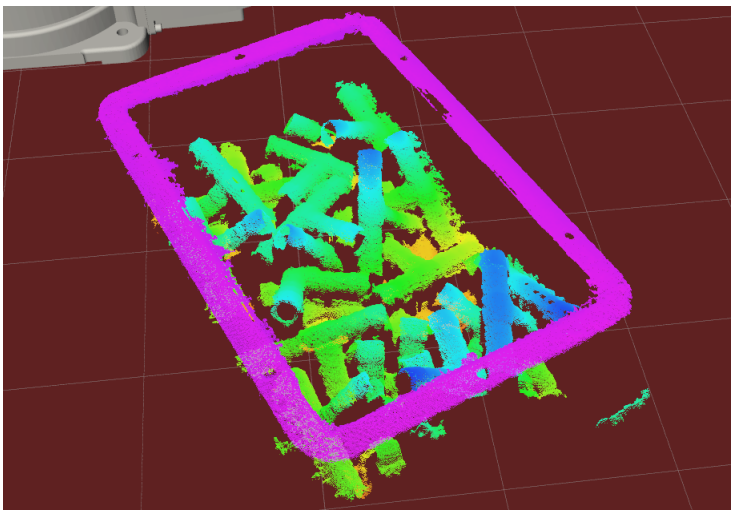


Abb. 40: Screenshot Objekterkennungs-Tool Punktwolke nach Bildaufnahme

Gefundene Objekte werden in der Visualisierung/Punktwolke als CAD-Modell grau eingezeichnet und die dazugehörigen Scores eingeblendet. Punkte, die nahe an einem Fund liegen, sind dabei farblich dargestellt (ebenfalls nach ihrem Z-Wert). Alle weiteren Punkte werden grau dargestellt.

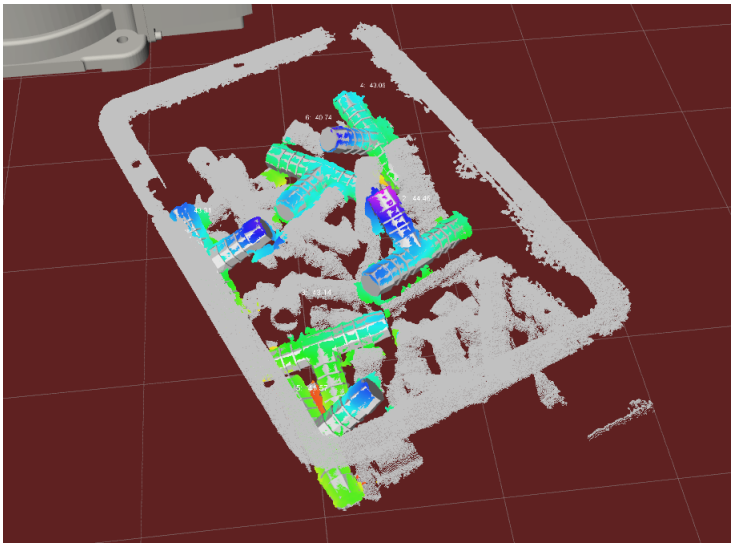


Abb. 41: Default-Ansicht des Objekterkennungs-Tools

Bemerkung: Die farblich hervorgehobenen Punkte werden im normalen Programmablauf aus der Punktwolke ausgeschnitten und werden somit von der Kollisionsüberwachung ausgeschlossen. Die Größe des Ausschnitts lässt sich über den Parameter *dilation_radius_object_cutting* in *object_recognition.yaml* einstellen.

2.13.6 Halcon-spezifische Objekterkennungsparameter

In diesem Bereich können Sie die Parametrierung der Objekterkennung optimieren. Die Initialwerte der Parameter entsprechen der *Konfiguration* des Projekts (siehe *object_recognition.yaml*). Beachten Sie, dass in *object_recognition.yaml* globale Parameter spezifiziert werden, d.h. die dort spezifizierten Parameter gelten für alle Produkte. Produktspezifische Parameter können Sie mithilfe des Objekterkennungs-Tool anpassen und testen.

Hinweis: Alle Objekterkennungsparameter dieser Seite sind sofort aktiv, auch für den normalen Programmablauf!

- ***Relative Sampling-Distanz (Szenenpunkte)*:**
  Abtastdistanz der 3D-Daten relativ zum Durchmesser des Oberflächenmodells (Produkt). [0.0 - 1.0]
- ***Anteil entscheidender Punkte*:**
  Anteil der abgetasteten Szenepunkte, die als Schlüsselpunkte für das näherungsweise Matching verwendet werden. [0.0 - 1.0]
- ***Scoring Abstand*:**

  Maximalabstand zwischen Modellpunkt und Szenenpunkt, sodass das Punktpaar positiv zum Score beiträgt.
  Typisch: örtliche Sensorauflösung/2.
- ***Verwende Normaleninformation*:**
  Aktiviert/Deaktiviert beim näherungsweisen Matching die Verwendung von Normaleninformation
- ***Berechnung der Normalen*:**
  Verfahren zur Berechnung der Normalen.

  - ‚fast‘: schnelles Verfahren, Qualität der Normalen ist ausreichend
  - ‚mls‘: rechenintensives Verfahren mit sehr guten Ergebnissen

- ***Exakte Posenverfeinerung*:**
  Aktiviert/Deaktiviert den Verfeinerungsschritt der Objektsuche
  - *Sub-Sampling-Rate*: Je höher die Sub-Sampling Rate gewählt ist, desto weniger Szenenpunkte werden für die exakte Verfeinerung verwendet.
  - *Anzahl der Schritte*: Maximale Anzahl der Iterationen für die exakte Verfeinerung. Die Anzahl der hier festgelegten Schritte stellt ein Limit dar. Falls ein stabiles Ergebnis bereits früher ermittelt wurde, wird die Suche auch vor Erreichen dieser Anzahl beendet.
- ***Vorherige Matches wiederverwenden*:**
  Aktiviert/Deaktiviert die Verwendung von Funden der vorhergehenden Objektsuche.

  Hinweis: Diese Einstellung sollte während der Optimierung der Parameter deaktiviert werden, da die Ergebnisse der einzelnen Objektsuchen sonst voneinander abhängig sind. Erst nachdem Sie geeignete Werte für alle Objekterkennungsparameter gefunden haben, sollten Sie diese Funktion aktivieren und die Feinabstimmung der folgenden Parameter vornehmen.

  - *Sub-Sampling-Rate*: s.o.
  - *Anzahl der Schritte*: s.o.
  - *Schwellwert der relativen Distanz*: Setzt den Abstandsschwellwert relativ zum Durchmesser des Modells. Nur 3D-Punkte deren Abstand zur Objektinstanz diesen Schwellwert unterschreiten gehen in die Verfeinerung ein. Punkte, die weiter von der Instanz entfernt sind, werden ignoriert. [0.0 - 1.0]
  - *Verwende Normaleninformation*: Aktiviert/Deaktiviert beim Matching die Verwendung von Normaleninformation

Ansicht (Halcon)

Nachdem eine Objektsuche durchgeführt wurde, lassen sich für den Suchalgorithmus Halcon zusätzlich zur Standard-Ansicht noch weitere Ansichten auswählen:

- ***Schlüsselpunkte*:**
  Anstatt die komplette Punktwolke darzustellen, werden nur die Schlüsselpunkte visualisiert. Außerdem werden die Funde mit dem Punktmodell des Produkts dargestellt - so lässt sich besser erkennen, wie ein Produkt für die näherungsweise Suche abgetastet wird. Empfohlen wird, dass die Parameter so gewählt werden, dass ca. 50-100 Schlüsselpunkte auf dem Objekt liegen.

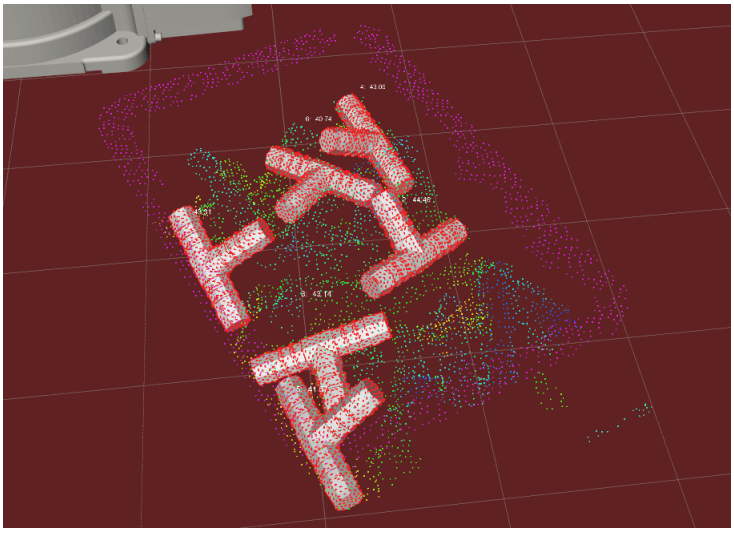


Abb. 42: Schlüsselpunkte-Ansicht des Objekterkennungs-Tools

- ***Abgetastete Punkte* (Verfeinerung):**
  Die Darstellung ist ähnlich zur Ansicht der Schlüsselpunkte. Allerdings werden hier die Punktwolke und das Produkt in der Form dargestellt, wie sie für die exakte Posenverfeinerung verwendet wird.

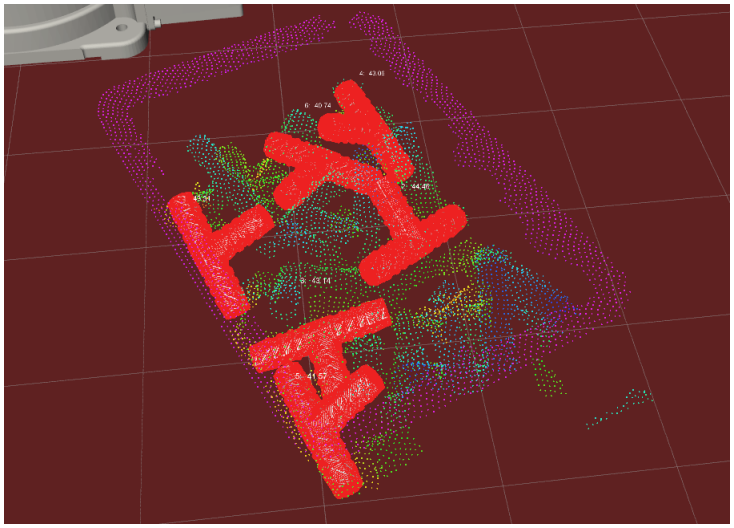


Abb. 43: Verfeinerung-Ansicht des Objekterkennungs-Tools

2.14 Griff-Inspektion

2.14.1 Überblick

Das Griff-Inspektionstool dient zur Analyse der Suche nach potenziellen Greifbewegungen. Werden in einer Region N Objekte gefunden und sind für das entsprechende *Produkt* M Greifpunkte (siehe *Greifposen*) definiert, so gibt es grundsätzlich NxM potenzielle Greifbewegungen – eine für jede Kombination von Objekt und Greifpunkt. Jede dieser Bewegungen wird zunächst auf potenzielle Kollisionen untersucht. Sobald der weitere Programmablauf eine Greifbewegung benötigt (bspw. bei Erreichen eines „Greife-Objekt“-Tasks), wird von allen bis dahin gefundenen validen Bewegungen die „beste“ ausgewählt. Die „Güte“ einer Bewegung richtet sich dabei nach mehreren Kriterien (siehe unten), u.a. nach der Lage-Höhe des zugehörigen Objekts und der Länge der Greifbewegung.

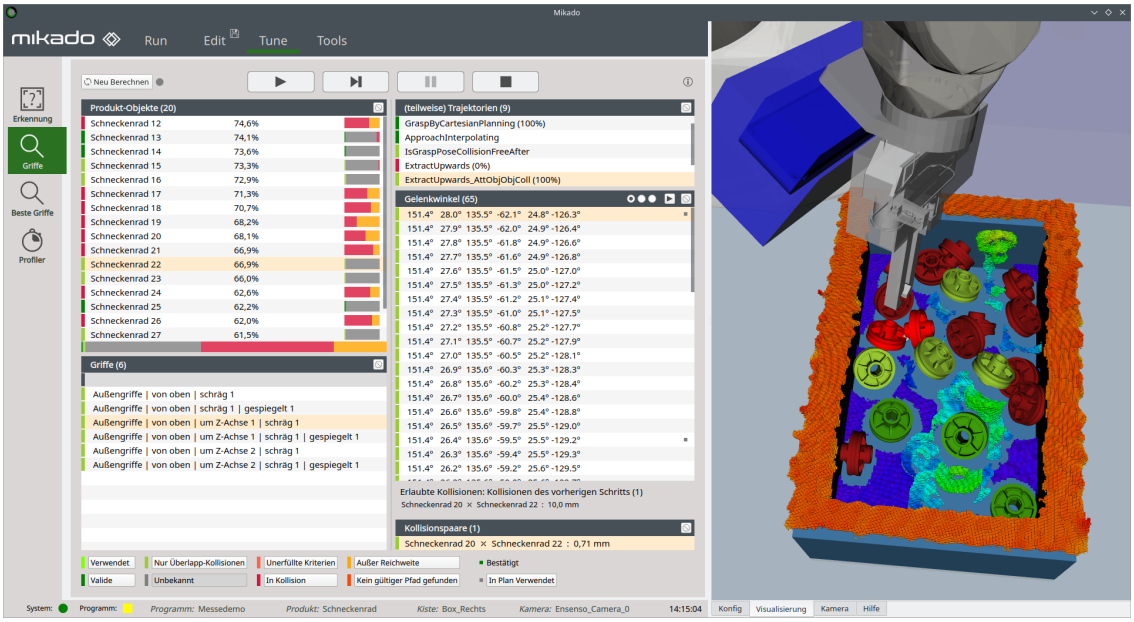


Abb. 44: Screenshot Griff-Inspektion

Priorisierung der gefundenen Griffe

siehe *Priorisierung*

2.14.2 Inspektion der Greifbewegungssuche

Die Anzahl der zu inspizierenden Roboterposen (-zustände) ergibt sich aus dem Produkt der Anzahl der gefundenen Objekte (N) mit der Anzahl der Greifpositionen pro Objekt (M) multipliziert mit der Anzahl der zu prüfenden Roboter-Zwischenposen pro Greifbewegung. Letztere variiert mit der Greifbewegung: Bei längeren Fahrten müssen mehr Zwischenposen geprüft werden als bei kürzeren Fahrten.

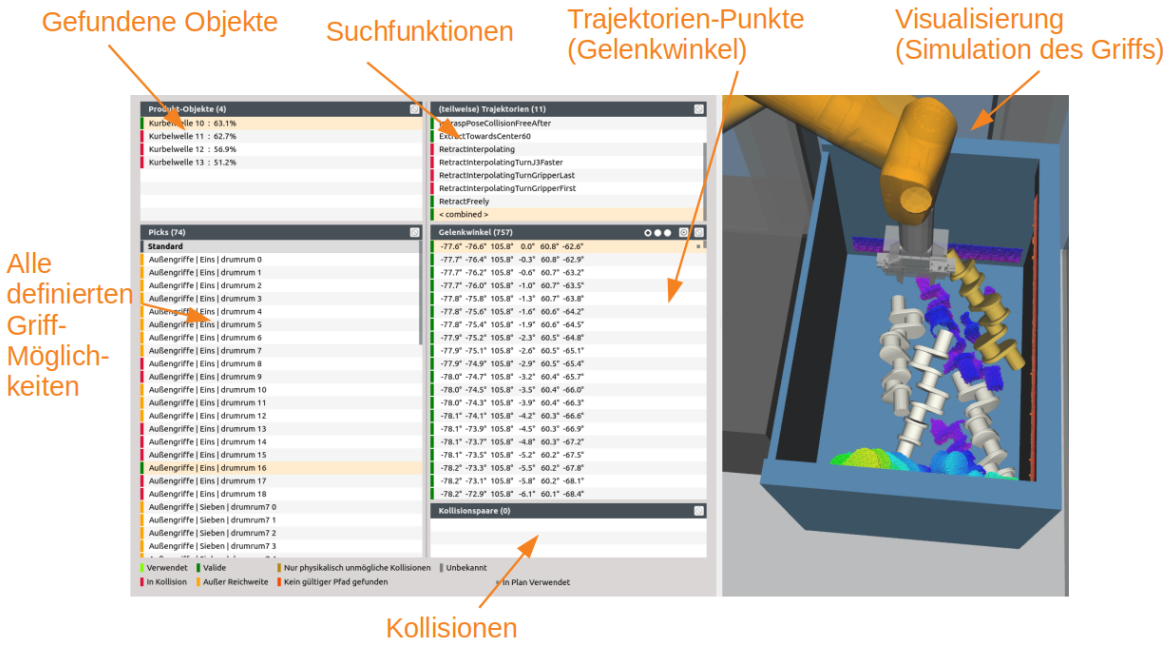


Abb. 45: Screenshot Griff-Inspektion mit Beschreibung

Da dies schnell unübersichtlich wird, erfolgt im Inspektionsmenü eine schrittweise Auswahl:

Zunächst muss das Objekt ausgewählt werden, für welches die potenzielle Greifbewegung angezeigt werden soll. Das aktuell ausgewählte Objekt wird in der Visualisierung rechts gelb eingefärbt – alle anderen Objekte werden blau dargestellt.

Ist ein Objekt ausgewählt, werden in der nächsten Liste (links mittig) die zugehörigen Griffe angezeigt. Hier wird der Griff ausgewählt, für den die potenzielle Greifbewegung angezeigt werden soll.

Ist ein Griff – und somit eine potenzielle Greifbewegung – ausgewählt, werden in der nächsten Liste (oben rechts) die einzelnen Suchfunktionen angezeigt.

Ist eine Suchfunktion ausgewählt, so werden in der Liste darunter die einzelnen Schritte der zugehörigen Bewegung angezeigt. Dabei kann über die drei Optionsfelder rechts neben der Liste gewählt werden, was genau in der Liste angezeigt wird: Der Gelenkzustand des Roboters, die Pose des Endeffektors oder die Pose des Handgelenks. Über den „Abspielen“-Knopf oben rechts neben der Liste kann die Bewegung als Ganzes abgespielt werden. Enthält der selektierte Schritt eine oder mehrere Kollisionen, werden diese in der Liste unten rechts angezeigt.

Ist eine Kollision ausgewählt, so werden die kollidierenden Teile in der Darstellung rechts rot eingefärbt (Ausnahme: die Punktwolke). Des Weiteren werden die Eckpunkte der Kollisionsfläche mit kleinen weißen Kugeln gekennzeichnet.

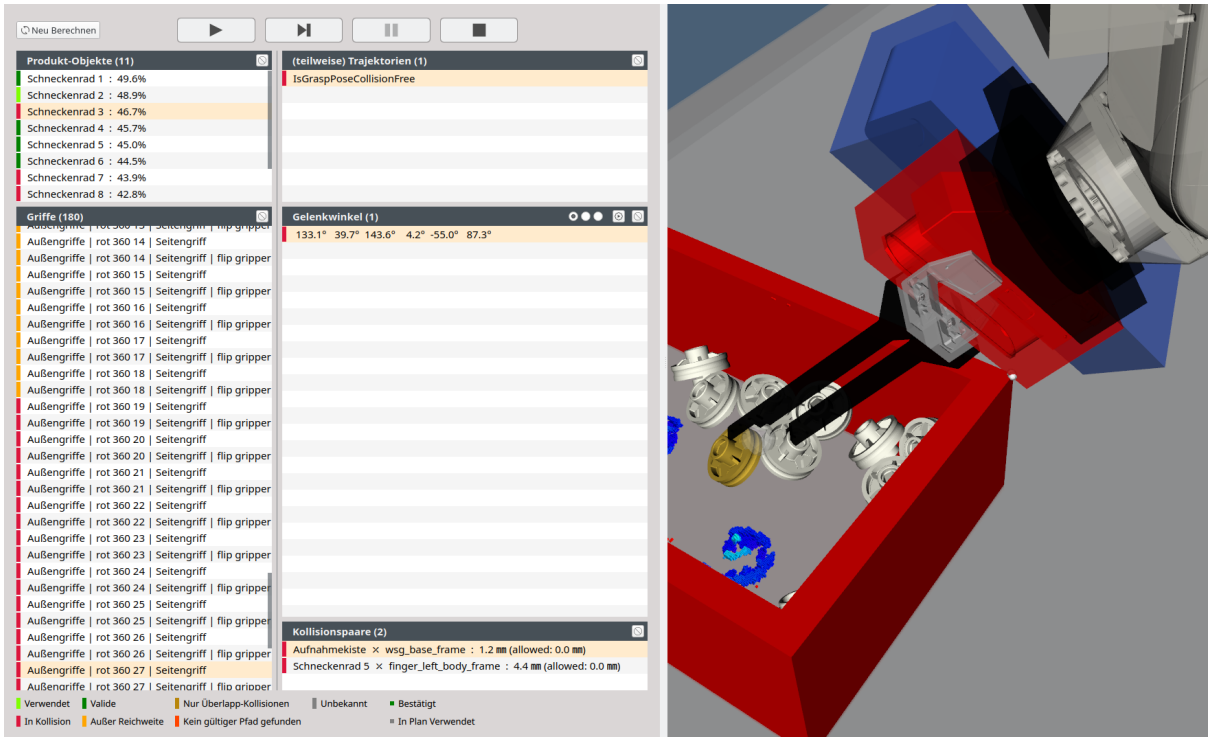


Abb. 46: Screenshot Kollisionsanzeige in der Griff-Inspektion

Der Sonderfall, dass die Greifposition außerhalb der Reichweite des Roboters liegt, wird dadurch gekennzeichnet, dass die Liste der Bewegungen leer ist. In der Visualisierung auf der rechten Seite wird der Robotor in diesem Fall in seinem Grundzustand (alle Achswinkel auf 0°) angezeigt. Zusätzlich wird in der Visualisierung ein Greifer in rot an der nicht erreichbaren Greifposition dargestellt.

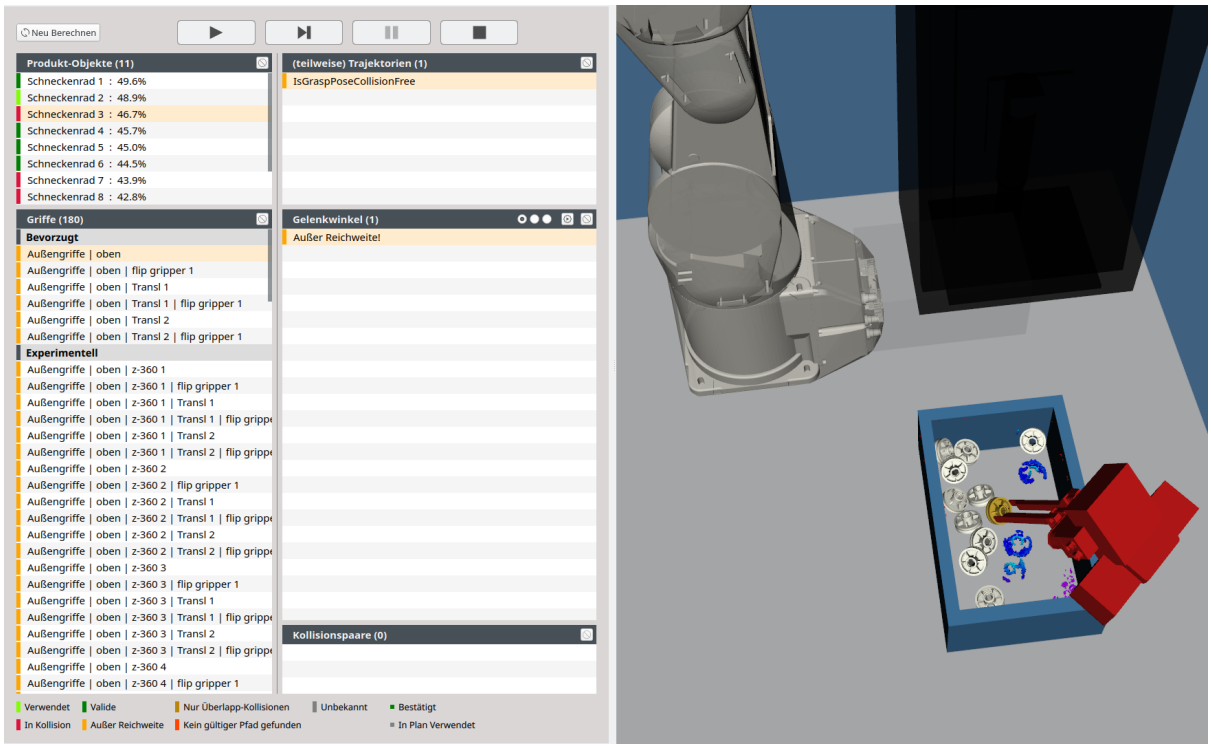


Abb. 47: Screenshot Darstellung unerreichbarer Griffe in der Griff-Inspektion

Suchzustand

Für jedes Objekt, jeden Greifpunkt, jede Such-Stufe, jede Such-Funktion und jeden Bewegungsschritt wird ein Zustand farbig angezeigt. Die Farben haben folgende Bedeutung (siehe auch Legende in den Screenshots):

- hell-grün: *verwendet* (das Objekt, das gegriffen wurde, bzw. der Griff, der tatsächlich verwendet wurde)
- dunkel-grün: *valide*
- dunkel-gold: *valide, aber mit Überlappkollision(en)*
- dunkel-rot: *invalide aufgrund von Kollision(en)*
- matt-orange: *invalide, da nicht erreichbar*
- grell-orange: (nur freie Bahnplanung) *invalide, da kein valider Pfad gefunden*
- grau: *wurde noch nicht berechnet*

In der Liste der Trajektorienpunkte gibt es außerdem bei manchen Punkten rechts farbige Markierungen: * grau: *wird an die Roboter-Steuerung geschickt* (Bei einer einfachen Interpolationsfahrt bspw. werden zwar viele Zwischenpunkte geprüft, aber nur Start- & Endpunkt an die Roboter-Steuerung geschickt, da jeder Roboter-Steuerung die Interpolation selbstständig übernimmt.) * grün: *wurde validiert* (Wird ein zuvor bereits gefundenes Objekt bei einer Objektsuche wiedergefunden, werden die bereits geprüften Griffe nochmal geprüft. Dies ist insbesondere dann relevant, wenn ein Griff per freier Bahnplanung gefunden wurde. Dann wird bei der Validierung nicht abermals eine freie Bahnplanung durchgeführt, sondern die zuvor gefundenen Trajektorienpunkte werden nur nochmal geprüft. Grün markerite Punkte zeigen an, dass die Punkte validiert wurden.)

Im Falle der Bewegungsschritte gibt es nur die Zustände *valide* und *invalide*. Die geben an, ob für diesen Bewegungsschritt eine Kollision vorliegt. In den anderen Fällen gibt es zusätzlich noch den Zustand *unbestimmt*.

Für eine Such-Funktionen gibt *unbestimmt* an, dass die Funktion noch nicht verwendet (berechnet) wurde. Dies tritt bspw. auf, wenn eine Funktion bereits *valide* zurückgibt und dementsprechend die weiteren Funktionen nicht mehr gebraucht werden,um festzustellen, ob es einen kollisionsfreien Pfad gibt.

Für eine Such-Stufe gibt *invalide* an, dass alle Funktionen geprüft wurden und *invalide* zurückgegeben haben. Umgekehrt gibt *valide* an, dass mindestens eine Funktion *valide* zurückgegeben hat. *Unbestimmt* gibt an, dass noch keine *valide* Funktion gefunden wurde, es aber noch weitere Funktionen im Zustand *unbestimmt* gibt.

Für eine Suchbewegung gibt *valide* an, dass alle zugehörigen Such-Stufen *valide* sind. Umgekehrt gibt *invalide* an, dass mindestens eine Such-Stufe im Zustand *invalide* ist. *Unbestimmt* gibt an, dass noch keine Such-Stufe im Zustand *invalide* ist, aber noch nicht alle Stufen zu Ende geprüft wurden.

Für ein Objekt gibt *invalide* an, dass alle zugehörigen potenziellen Greifbewegungen *invalide* sind. Umgekehrt gibt *valide* an, dass mindestens eine *valide* zugehörige Greifbewegung gefunden wurde. *Unbestimmt* gibt an, dass noch keine *valide* Greifbewegung gefunden wurde, es aber noch weitere Greifbewegungen im Zustand *unbestimmt* gibt.

Für die Greifbewegungssuche als Ganzes gibt *invalide* an, dass für alle gefundenen Objekte, alle potenziellen Greifbewegungen *invalide* sind. Umgekehrt gibt *valide* an, dass es mindestens eine *valide* Greifbewegung für mindestens ein Objekt gibt.

2.15 Beste Griffe

2.15.1 Priorisierung

Die Priorisierung erfolgt über den Parameter *Sortier-Kriterien*. Dieser Parameter ist eine Liste von vordefinierten Strings, wobei jeder String ein Sortier-Kriterium angibt. Die Liste legt fest, nach welchen Kriterien sortiert wird und in welcher Reihenfolge die Kriterien priorisiert werden.

Um alle Griffe bezüglich der verschiedenen Kriterien vergleichen zu können, wird Binning verwendet. Beim Binning werden Gleitkommazahl-Werte gruppiert. Dabei wird der Werte-Bereich vom jeweiligen Bestwert aus in gleichgroße Bereiche (sogenannte Bins) eingeteilt. Der Bin-Wert 0 ist immer die beste Bin (die erste Bin ab dem Bestwert) und steigende Bin-Werte stehen für schlechtere Werte. Die Größe dieser Bins lässt sich über den Parameter *Bin-Größen* setzen.

Verfügbare Sortier-Kriterien

Folgende vordefinierten String-Werte stehen zur Verfügung:

Tab. 3: Bool'sche Kriterien:

Bin-Name	Bedeutung des Bin-Wertes
`Griff noch nicht versucht zu greifen?`	0: Griff noch nicht versucht zu greifen 1: Griff bereits versucht zu greifen
`ist unter Fahrdistanz-Schwellwert`	0: ist unter Fahrdistanz-Schwellwert 1: ist über Fahrdistanz-Schwellwert
`hat keine echten Kollisionen`	0: hat keine echten Kollisionen 1: hat (erlaubte) echte Kollisionen
`hat keine Überlapp-(Pseudo-)Kollisionen`	0: hat keine Pseudo-Kollisionen (siehe *Pseudo-Kollisionen*) 1: hat (erlaubte) Pseudo-Kollisionen

Die Bool'sche Kriterien lassen sich auf die Bin-Werte übertragen: 0 steht für „Kriterium erfüllt", 1 steht für „Kriterium nicht erfüllt".

Tab. 4: nach oben limitierte Kritieren:

`Griff-Kategorie`	0: Bevorzugt (B) 1: Standard (S) 2: Experimentell (E)

Kategorien lassen sich ebenfalls auf Bin-Werte übertragen. Die beste Kategorie (Bevorzugt) erhält den Wert 0, die zweitbeste den Wert 1, usw.

Tab. 5: Binning-Kriterien:

`Anzahl Greifversuche an Objekt (aufsteigend)`	0: Objekt noch nicht versucht zu greifen 1..: Anzahl Greifversuche dieses Objekts
`Match-Score`	0: Bin des höchsten Matching-Scores 1..: Bins für niedrigere Matching-Scores Default-Bingröße: 0.1 (10%)

Fortsetzung auf der nächsten Seite

Tab. 5 – Fortsetzung der vorherigen Seite

`Handgelenks-Winkel zur Vertikalen`	0: Bin des niedrigsten Winkels zwischen der Handgelenkspose und einer vertikalen Geraden (siehe *Winkel zur Vertikalen*) 1..: Bins für höhere Winkel Default-Bingröße: 0.087266 (5°)
`Tcp-Winkel zur Vertikalen`	0: Bin des niedrigsten Winkels zwischen der Tcp-Pose und einer vertikalen Geraden (siehe *Winkel zur Vertikalen*) 1..: Bins für höhere Winkel Default-Bingröße: 0.087266 (5°)
`Fahrdistanz`	0: Bin der kleinsten Fahrdistanz (siehe *Fahrdistanz Schwellwert*) 1..: Bins für größere Fahrdistanzen Default-Bingröße: 1.0471975512 (60° = 6 joints * 10°)
`Gesamt-Kollisionstiefe`	0: Bin der niedrigsten Gesamt-Kollisionstiefe (siehe *Kollisionstiefe*) 1..: Bins für höhere Gesamt-Kollisionstiefen Default-Bingröße: 0.001 (1mm)
`durchschnittliche Kollisionstiefe`	0: Bin der niedrigsten Durchschnitts-Kollisionstiefe (siehe *Kollisionstiefe*) 1..: Bins für höhere Durchschnitts-Kollisionstiefen Default-Bingröße: 0.0001 (0.1mm)
`maximale Kollisionstiefe`	0: Bin der niedrigsten Maximal-Kollisionstiefe (siehe *Kollisionstiefe*) 1..: Bins für höhere Maximal-Kollisionstiefen Default-Bingröße: 0.0002 (0.2mm)
`Fund-x (absteigend)`	0: Bin des höchsten x-Wertes aller Teile-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere x-Werte Default-Bingröße: 0.05 (5cm)
`Fund-y (absteigend)`	0: Bin des höchsten y-Wertes aller Teile-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere y-Werte Default-Bingröße: 0.05 (5cm)
`Fund-z (absteigend)`	0: Bin des höchsten z-Wertes aller Teile-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere z-Werte Default-Bingröße: 0.05 (5cm)

Fortsetzung auf der nächsten Seite

Tab. 5 – Fortsetzung der vorherigen Seite

`Fund-x (aufsteigend)`	0: Bin des niedrigsten x-Wertes aller Teile-Posen im Welt-Koordinatensystem 1..: Bins für höhere x-Werte Default-Bingröße: 0.05 (5cm)
`Fund-y (aufsteigend)`	0: Bin des niedrigsten y-Wertes aller Teile-Posen im Welt-Koordinatensystem 1..: Bins für höhere y-Werte Default-Bingröße: 0.05 (5cm)
`Fund-z (aufsteigend)`	0: Bin des niedrigsten z-Wertes aller Teile-Posen im Welt-Koordinatensystem 1..: Bins für höhere z-Werte Default-Bingröße: 0.05 (5cm)
`abs(Fund-x) (aufsteigend)`	0: Bin des niedrigsten absoluten x-Wertes aller Teile-Posen im Welt-Koordinatensystem 1..: Bins für höhere absolute x-Werte Default-Bingröße: 0.05 (5cm)
`abs(Fund-y) (aufsteigend)`	0: Bin des niedrigsten absoluten y-Wertes aller Teile-Posen im Welt-Koordinatensystem 1..: Bins für höhere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`abs(Fund-x) (absteigend)`	0: Bin des höchsten absoluten x-Wertes aller Teile-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere absolute x-Werte Default-Bingröße: 0.05 (5cm)
`abs(Fund-y) (absteigend)`	0: Bin des höchsten absoluten y-Wertes aller Teile-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`Tcp-x (absteigend)`	0: Bin des höchsten x-Wertes aller Tcp-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere x-Werte Default-Bingröße: 0.05 (5cm)

Fortsetzung auf der nächsten Seite

Tab. 5 – Fortsetzung der vorherigen Seite

`Tcp-y (absteigend)`	0: Bin des höchsten y-Wertes aller Tcp-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere y-Werte Default-Bingröße: 0.05 (5cm)
`Tcp-z (absteigend)`	0: Bin des höchsten z-Wertes aller Tcp-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere z-Werte Default-Bingröße: 0.05 (5cm)
`Tcp-x (aufsteigend)`	0: Bin des niedrigsten x-Wertes aller Tcp-Posen im Welt-Koordinatensystem 1..: Bins für höhere x-Werte Default-Bingröße: 0.05 (5cm)
`Tcp-y (aufsteigend)`	0: Bin des niedrigsten y-Wertes aller Tcp-Posen im Welt-Koordinatensystem 1..: Bins für höhere y-Werte Default-Bingröße: 0.05 (5cm)
`Tcp-z (aufsteigend)`	0: Bin des niedrigsten z-Wertes aller Tcp-Posen im Welt-Koordinatensystem 1..: Bins für höhere z-Werte Default-Bingröße: 0.05 (5cm)
`abs(Tcp-x) (aufsteigend)`	0: Bin des niedrigsten absoluten x-Wertes aller Tcp-Posen im Welt-Koordinatensystem 1..: Bins für höhere absolute x-Werte Default-Bingröße: 0.05 (5cm)
`abs(Tcp-y) (aufsteigend)`	0: Bin des niedrigsten absoluten y-Wertes aller Tcp-Posen im Welt-Koordinatensystem 1..: Bins für höhere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`abs(Tcp-x) (absteigend)`	0: Bin des höchsten absoluten x-Wertes aller Tcp-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere absolute x-Werte Default-Bingröße: 0.05 (5cm)

Fortsetzung auf der nächsten Seite

Tab. 5 – Fortsetzung der vorherigen Seite

`abs(Tcp-y) (absteigend)`	0: Bin des höchsten absoluten y-Wertes aller Tcp-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-x (absteigend)`	0: Bin des höchsten x-Wertes aller Flansch-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere x-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-y (absteigend)`	0: Bin des höchsten y-Wertes aller Flansch-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere y-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-z (absteigend)`	0: Bin des höchsten z-Wertes aller Flansch-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere z-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-x (aufsteigend)`	0: Bin des niedrigsten x-Wertes aller Flansch-Posen im Welt-Koordinatensystem 1..: Bins für höhere x-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-y (aufsteigend)`	0: Bin des niedrigsten y-Wertes aller Flansch-Posen im Welt-Koordinatensystem 1..: Bins für höhere y-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-z (aufsteigend)`	0: Bin des niedrigsten z-Wertes aller Flansch-Posen im Welt-Koordinatensystem 1..: Bins für höhere z-Werte Default-Bingröße: 0.05 (5cm)
`abs(Flansch-x) (aufsteigend)`	0: Bin des niedrigsten absoluten x-Wertes aller Flansch-Posen im Welt-Koordinatensystem 1..: Bins für höhere absolute x-Werte Default-Bingröße: 0.05 (5cm)

Fortsetzung auf der nächsten Seite

Tab. 5 – Fortsetzung der vorherigen Seite

`abs(Flansch-y) (aufsteigend)`	0: Bin des niedrigsten absoluten y-Wertes aller Flansch-Posen im Welt-Koordinatensystem 1..: Bins für höhere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`abs(Flansch-x) (absteigend)`	0: Bin des höchsten absoluten x-Wertes aller Flansch-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere absolute x-Werte Default-Bingröße: 0.05 (5cm)
`abs(Flansch-y) (absteigend)`	0: Bin des höchsten absoluten y-Wertes aller Flansch-Posen im Welt-Koordinatensystem 1..: Bins für niedrigere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`Fund-x rel. zur Kiste (absteigend)`	0: Bin des höchsten x-Wertes aller Teile-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere x-Werte Default-Bingröße: 0.05 (5cm)
`Fund-y rel. zur Kiste (absteigend)`	0: Bin des höchsten y-Wertes aller Teile-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere y-Werte Default-Bingröße: 0.05 (5cm)
`Fund-z rel. zur Kiste (absteigend)`	0: Bin des höchsten z-Wertes aller Teile-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere z-Werte Default-Bingröße: 0.05 (5cm)
`Fund-x rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten x-Wertes aller Teile-Posen im Kisten-Koordinatensystem 1..: Bins für höhere x-Werte Default-Bingröße: 0.05 (5cm)
`Fund-y rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten y-Wertes aller Teile-Posen im Kisten-Koordinatensystem 1..: Bins für höhere y-Werte Default-Bingröße: 0.05 (5cm)

Fortsetzung auf der nächsten Seite

Tab. 5 – Fortsetzung der vorherigen Seite

`Fund-z rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten z-Wertes aller Teile-Posen im Kisten-Koordinatensystem 1..: Bins für höhere z-Werte Default-Bingröße: 0.05 (5cm)
`abs(Fund-x) rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten absoluten x-Wertes aller Teile-Posen im Kisten-Koordinatensystem 1..: Bins für höhere absolute x-Werte Default-Bingröße: 0.05 (5cm)
`abs(Fund-y) rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten absoluten y-Wertes aller Teile-Posen im Kisten-Koordinatensystem 1..: Bins für höhere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`abs(Fund-x) rel. zur Kiste (absteigend)`	0: Bin des höchsten absoluten x-Wertes aller Teile-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere absolute x-Werte Default-Bingröße: 0.05 (5cm)
`abs(Fund-y) rel. zur Kiste (absteigend)`	0: Bin des höchsten absoluten y-Wertes aller Teile-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`Tcp-x rel. zur Kiste (absteigend)`	0: Bin des höchsten x-Wertes aller Tcp-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere x-Werte Default-Bingröße: 0.05 (5cm)
`Tcp-y rel. zur Kiste (absteigend)`	0: Bin des höchsten y-Wertes aller Tcp-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere y-Werte Default-Bingröße: 0.05 (5cm)
`Tcp-z rel. zur Kiste (absteigend)`	0: Bin des höchsten z-Wertes aller Tcp-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere z-Werte Default-Bingröße: 0.05 (5cm)

Fortsetzung auf der nächsten Seite

Tab. 5 – Fortsetzung der vorherigen Seite

`Tcp-x rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten x-Wertes aller Tcp-Posen im Kisten-Koordinatensystem 1..: Bins für höhere x-Werte Default-Bingröße: 0.05 (5cm)
`Tcp-y rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten y-Wertes aller Tcp-Posen im Kisten-Koordinatensystem 1..: Bins für höhere y-Werte Default-Bingröße: 0.05 (5cm)
`Tcp-z rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten z-Wertes aller Tcp-Posen im Kisten-Koordinatensystem 1..: Bins für höhere z-Werte Default-Bingröße: 0.05 (5cm)
`abs(Tcp-x) rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten absoluten x-Wertes aller Tcp-Posen im Kisten-Koordinatensystem 1..: Bins für höhere absolute x-Werte Default-Bingröße: 0.05 (5cm)
`abs(Tcp-y) rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten absoluten y-Wertes aller Tcp-Posen im Kisten-Koordinatensystem 1..: Bins für höhere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`abs(Tcp-x) rel. zur Kiste (absteigend)`	0: Bin des höchsten absoluten x-Wertes aller Tcp-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere absolute x-Werte Default-Bingröße: 0.05 (5cm)
`abs(Tcp-y) rel. zur Kiste (absteigend)`	0: Bin des höchsten absoluten y-Wertes aller Tcp-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-x rel. zur Kiste (absteigend)`	0: Bin des höchsten x-Wertes aller Flansch-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere x-Werte Default-Bingröße: 0.05 (5cm)

Fortsetzung auf der nächsten Seite

Tab. 5 – Fortsetzung der vorherigen Seite

`Flansch-y rel. zur Kiste (absteigend)`	0: Bin des höchsten y-Wertes aller Flansch-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere y-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-z rel. zur Kiste (absteigend)`	0: Bin des höchsten z-Wertes aller Flansch-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere z-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-x rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten x-Wertes aller Flansch-Posen im Kisten-Koordinatensystem 1..: Bins für höhere x-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-y rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten y-Wertes aller Flansch-Posen im Kisten-Koordinatensystem 1..: Bins für höhere y-Werte Default-Bingröße: 0.05 (5cm)
`Flansch-z rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten z-Wertes aller Flansch-Posen im Kisten-Koordinatensystem 1..: Bins für höhere z-Werte Default-Bingröße: 0.05 (5cm)
`abs(Flansch-x) rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten absoluten x-Wertes aller Flansch-Posen im Kisten-Koordinatensystem 1..: Bins für höhere absolute x-Werte Default-Bingröße: 0.05 (5cm)
`abs(Flansch-y) rel. zur Kiste (aufsteigend)`	0: Bin des niedrigsten absoluten y-Wertes aller Flansch-Posen im Kisten-Koordinatensystem 1..: Bins für höhere absolute y-Werte Default-Bingröße: 0.05 (5cm)
`abs(Flansch-x) rel. zur Kiste (absteigend)`	0: Bin des höchsten absoluten x-Wertes aller Flansch-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere absolute x-Werte Default-Bingröße: 0.05 (5cm)

Fortsetzung auf der nächsten Seite

Tab. 5 – Fortsetzung der vorherigen Seite

`abs(Flansch-y) rel. zur Kiste (absteigend)`	0: Bin des höchsten absoluten y-Wertes aller Flansch-Posen im Kisten-Koordinatensystem 1..: Bins für niedrigere absolute y-Werte Default-Bingröße: 0.05 (5cm)

Tab. 6: Nutzer-definiertes Binning:

`vom Nutzer gesetzte Bin`	niedrigere Bin ist besser

In einem JavaScript-Task kann über die Funktion `picksearch.setUserBins(bins: [int])` für jedes gefundene Objekt ein Bin-Wert gesetzt werden. Dabei muss die Anzahl der `bins` exakt mit der Anzahl der gefundenen Objekte übereinstimmen. Ist zusätzlich in der Liste der `Sortier-Kriterien` der Eintrag `vom Nutzer gesetzte Bin` enthalten, so werden die gesetzten Bins für die Sortierung mitverwendet.

Tab. 7: gewichtetes Binning:

`gewichtete Summe der Bins`	niedrigere gewichtete Summe ist besser

Wenn nicht strikt ein Kriterium nach dem nächsten verglichen werden soll, sondern mehrere Kriterien kombiniert werden sollen, kann der Eintrag `gewichtete Bin` verwendet werden. Für diesen Fall gibt es zusätzlich den separaten Parameter `Gewichte`, über den man die Gewichte der einzelnen Kriterien einstellen kann. Alle Kriterien, die nicht in diesem Parameter aufgeführt werden, haben implizit das Gewicht 0. Zur Berechnung der gewichteten Bin werden die Bins für alle Kriterien berechnet, deren Gewicht nicht 0 ist, dann mit dem jeweiligen Gewicht multipliziert und schließlich zusammenaddiert.

Haben zwei Griffe in allen Kriterien denselben (Binning-)Wert, wird nach der Fahrdistanz sortiert (ohne Binning).

Tab. 8: Debug-Kriterien (0 kann als Bedingung verwendet werden):

`immer 1`	Alle Griffe haben den Bin-Wert 1
`hat echte Kollisionen`	Gegenteil von `hat keine echten Kollisionen`
`hat Überlapp-(Pseudo-)Kollisionen`	Gegenteil von `hat keine Überlapp-(Pseudo-)Kollisionen`

Winkel zur Vertikalen

Mit dem Winkel zur Vertikalen (Tcp oder Handgelenk) lässt sich beschränken, wie schräg gegriffen werden darf.

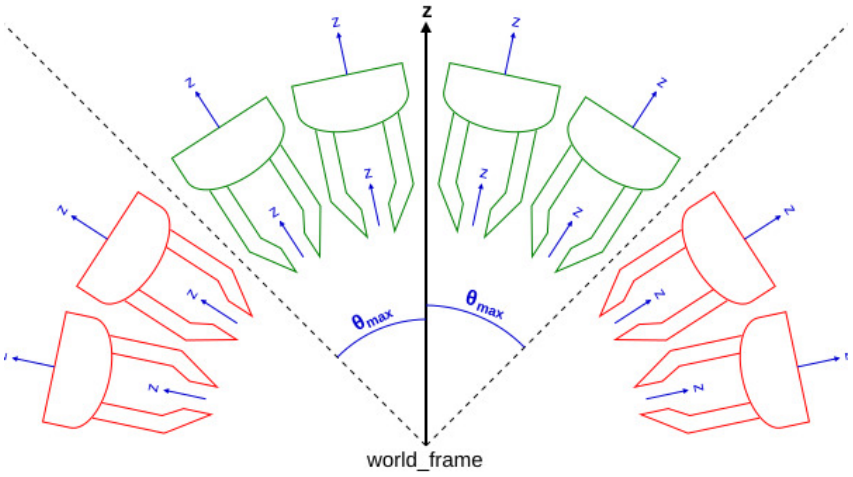


Abb. 48: Maximaler Winkel zur Vertikalen

Bedingungen

Die Bin-Werte können verwendet werden, um Bedingungen abzubilden. Zum Beispiel kann es sein, dass man nur Teile aus der obersten Lage greifen möchte. Das heißt, man möchte, dass das Teil für `z (absteigend)` (oder `z rel. zur Kiste (absteigend)`) den Bin-Wert 0 hat. Dies lässt sich über den Parameter `Bedingungen für Griff` abbilden, indem der Eintrag `z rel. zur Kiste (absteigend)` hinzugefügt wird und den Maximal-Wert `1 x Bin-Größe` (für dieses Kriterium) erhält. Bei bool'schen Kriterien gibt es keinen Wert einzustellen, diese müssen einfach erfüllt sein.

Möchte man stattdessen, dass das Teil in einer der beiden oberen Lagen liegt, würde man eine Maximal-Wert von `2 x Bin-Größe` wählen. Grundsätzlich stellt man bei absteigenden (bzw aufsteigenden) Kriterien den Wert ein, den ein Griff oder Teil für maximal haben darf (bzw mindestens haben muss).

Griffe, die die Bedingungen nicht erfüllen, werden verworfen. In der Griff-Inspektion bekommen sie die Farbe für „Unerfüllte Kriterien".

Neben diesen „harten" Bedingungen sind auch „weiche" Bedingungen einstellbar, die dazu führen, dass die Griffsuche noch etwas weiter läuft, wenn bisher noch kein Griff gefunden wurde, der die Bedingungen alle erfüllt. Der Parameter hierfür heißt `Bedingungen für Zusatzzeit`, wobei die gewährte `Zusatzzeit` ein eigener Parameter ist. Die Zusatzzeit beginnt ab dem ersten gefundenen erfolgreichen Griff (sofern dieser die Bedingungen nicht erfüllt).

Ist der Parameter nicht gesetzt, werden default-mäßig folgende Zusatzzeit-Bedingungen verwendet:

Objekt noch nicht versucht zu greifen?	
ist unter Fahrdistanz-Grenzwert	
Griff-Kategorie	1 (Bevorzugt oder Standard)

Das heißt, solange alle gefundenen validen Griffe experimentell sind, über dem Fahrdistanzschwellwert sind oder zu einem Teil gehören, das bereits versucht wurde zu greifen, wird weiter nach besseren Griffen gesucht – bis die eingestellte Zusatzzeit ausläuft.

Kollisionstiefe

Für jede Kollision gibt die Kollisionstiefe an, wie tief zwei kollidierende Objekte einander überlappen. Die *Gesamt-Kollisionstiefe* ist die Summe der Kollisionstiefen aller Kollisionen aller Fahrt-Schritte der *Entnehmen*-Stufe (siehe *Greifprozess*) (der einzigen Stufe, die Kollisionen erlaubt). Die *Durchschnitts-Kollisionstiefe* ist die *Gesamt-Kollisionstiefe* geteilt durch die Anzahl der Fahrt-Schritte der *Entnehmen*-Stufe. Die *Maximal-Kollisionstiefe* ist die größte Einzel-Kollisionstiefe einer Kollision aller Fahrt-Schritte.

Zu beachten ist hierbei, dass verschiedene *Entnehmen*-Teilfahrten unterschiedlich lang sein können. Daher kann es vorkommen, dass eine *Entnehmen*-Teilfahrt eine größere *Gesamt-Kollisionstiefe* hat als eine andere, aber gleichzeitig eine kleinere *Durchschnitts-Kollisionstiefe*.

Ebenfalls zu beachten ist, dass das Berechnen aller Kollisionstiefen die Berechnungszeit der Griff-Suche signifikant erhöht. Es empfiehlt sich daher, diese Sortier-Kriterien nur zu verwenden, wenn sie unbedingt benötigt werden. Die Berechnungszeit wird dabei auch von der Anzahl der Fahrtpunkte beeinflusst, siehe auch *Interpolations-Parameter*.

Pseudo-Kollisionen

Pseudo-Kollisionen sind Kollisionen von Teilen miteinander, bevor sie gegriffen werden. Dies kann passieren, wenn Teile direkt aneinander liegen; dann kann die (kleine) Ungenauigkeit der Objekterkennung dazu führen, dass zwei Teile als ineinander steckend erkannt werden. Da dies aber nur ein Artefakt der Objekterkennung ist und die Teile in Wirklichkeit nicht ineinanderstecken können, werden solche Kollisionen als „Pseudo-Kollisionen“ klassifiziert.

2.15.2 Beste-Griffe-Anzeige

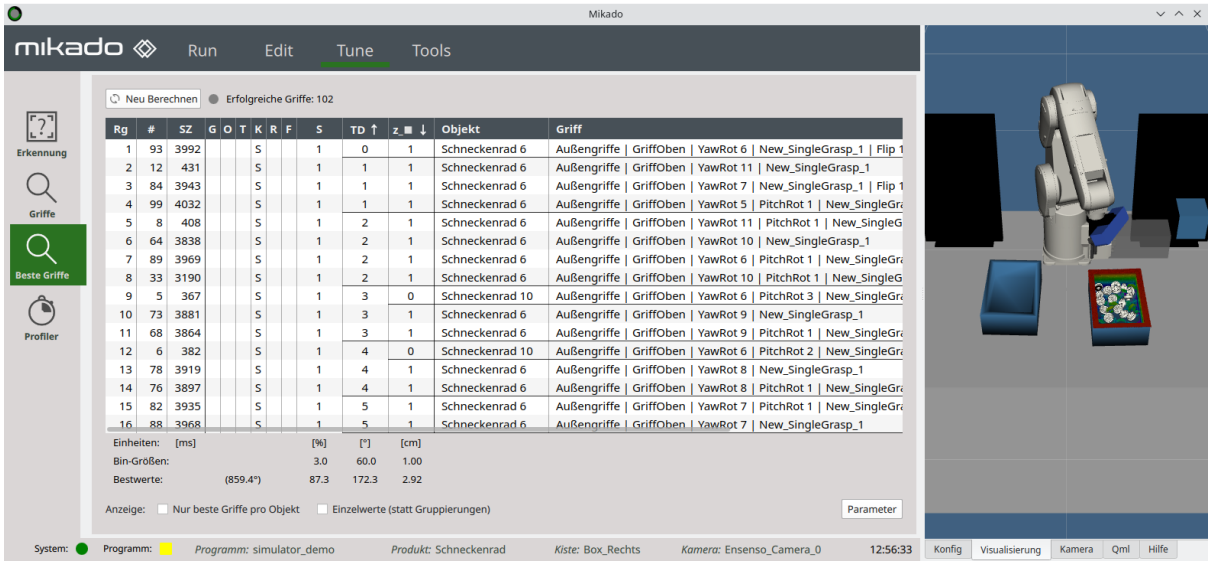


Abb. 49: Screenshot Beste-Griffe

Auf der „Beste Griffe“-Seite werden die bisher gefundenen validen Griffe in sortierter Reihenfolge aufgelistet. Die entsprechende Tabelle besteht aus drei Spalten-Bereichen:

- Die ersten drei Spalten geben den Rang (Rg) innerhalb der Sortierung, die Reihenfolge, in der die Griffe gefunden wurden (#), sowie die Suchzeit (SZ) in Millisekunden an (ab Start der Griffsuche).
- Die letzten zwei Spalten geben die Namen des Objekts, das mit dem Griff gegriffen werden würde, und des Griffs selbst an.
- Dazwischen werden die Werte des jeweiligen Griffs in den jeweiligen Suchkritieren angezeigt. Die Reihenfolge der Spalten richtet sich dabei nach der Reihenfolge des zuvor erwähnten Parameters `pick_selection_order`. Dabei ist zu beachten:
  - Bei bool’schen Kriterien wird ein „X“ angezeigt, wenn ein Kriterium nicht erfüllt ist. Andernfalls ist das Feld leer.
  - Bei unlimitierten Kriterien werden die Bin-Werte angezeigt. Wird das Häkchen bei „Einzelwerte“ (unter der Tabelle) gesetzt, werden stattdessen die tatsächlichen Werte angezeigt.
  - Die Überschriften sind abgekürzt um Platz zu sparen. Wenn man mit der Maus über eine Überschrift fährt, wird der ganze Name angezeigt.
  - Um leichter erkennen zu können, ab welchem Kriterium sich zwei aufeinanderfolgende Griffe unterscheiden, ist eine Trennlinie zwischen den Griffen ab der ersten unterschiedlichen Gruppierung eingezeichnet.
  - Bei Doppelklick auf eine Zeile wird zum entsprechenden Griff in der Griff-Inspektion gesprungen.

Unter der Tabelle werden für jede Binning-Spalte die Einheiten, die Bin-Größen, sowie die jeweiligen Bestwerte angezeigt. Darunter wiederum lassen sich zwei Anzeige-Optionen einstellen:

- Mit einem Häkchen bei „Nur beste Griffe pro Objekt“ wird die Liste gefiltert, sodass für jedes Objekt nur noch der beste Griff angezeigt wird.
- Mit einem Häkchen bei „Einzelwerte (statt Gruppierungen)“ werden nicht die Bins, sondern die tatsächlichen Werte angezeigt.

Unten rechts lassen sich per Klick auf den Parameter-Button die Sortier-Parameter für den aktuellen Suchdurchlauf anzeigen. Diese können in die `pick_search.yaml` kopiert und dort verändert werden.

Oben links gibt es, wie auf der Griff-Inspektion-Seite, einen Button um den Suchdurchlauf zu wiederholen. Daneben gibt es eine Status-Anzeige, die grün ist, wenn die Griffsuche läuft, und andernfalls grau ist. Daneben wird die Anzahl der bisher erfolgreichen Griffe angezeigt.

2.16 Profiler

Auf der Profiler-Seite können die Laufzeiten der einzelnen Tasks angezeigt werden. Die Zeiten sind insbesondere für Tasks interessant, die die Roboterbewegungen ausführen oder anderweitig viel Zeit beanspruchen und helfen Ihnen bei der Optimierung Ihres Ablaufs.

Die Oberfläche zeigt Ihnen den kompletten Programmablauf ähnlich zur *Statusseite*. Die Sequenzen und Tasks werden aufgeführt und der aktuelle Status eines Tasks wird angezeigt. Die letzte Laufzeit jedes einzelnen Tasks wird in Millisekunden angezeigt. Beachten Sie dabei folgende Punkte:

- Wurde der Task noch nicht ausgeführt, beträgt die letzte Laufzeit 0 ms. Bei sehr einfachen Tasks (z.B. Kopieren einer Variable) ist die Laufzeit sehr gering, so dass ebenfalls 0 ms angezeigt wird.
- Die Laufzeit der Objekt- und Griffsuche selbst kann unter Umständen nicht angezeigt werden, da diese meist parallel zum Programm stattfindet. Die Dauer einer Objektsuche kann mit dem *Objekterkennungs-Tool* ermittelt werden.
- Die Laufzeiten von Objektsuche und Griffsuche können stark variieren. Je nach Umgebung, ROI und Parametrierung variiert die Dauer der Objektsuche. Die Griffsuche ist abhängig von der Anzahl an Funden und kann zudem länger dauern, wenn die Objekte ungünstig positioniert sind.

2.17 Logs

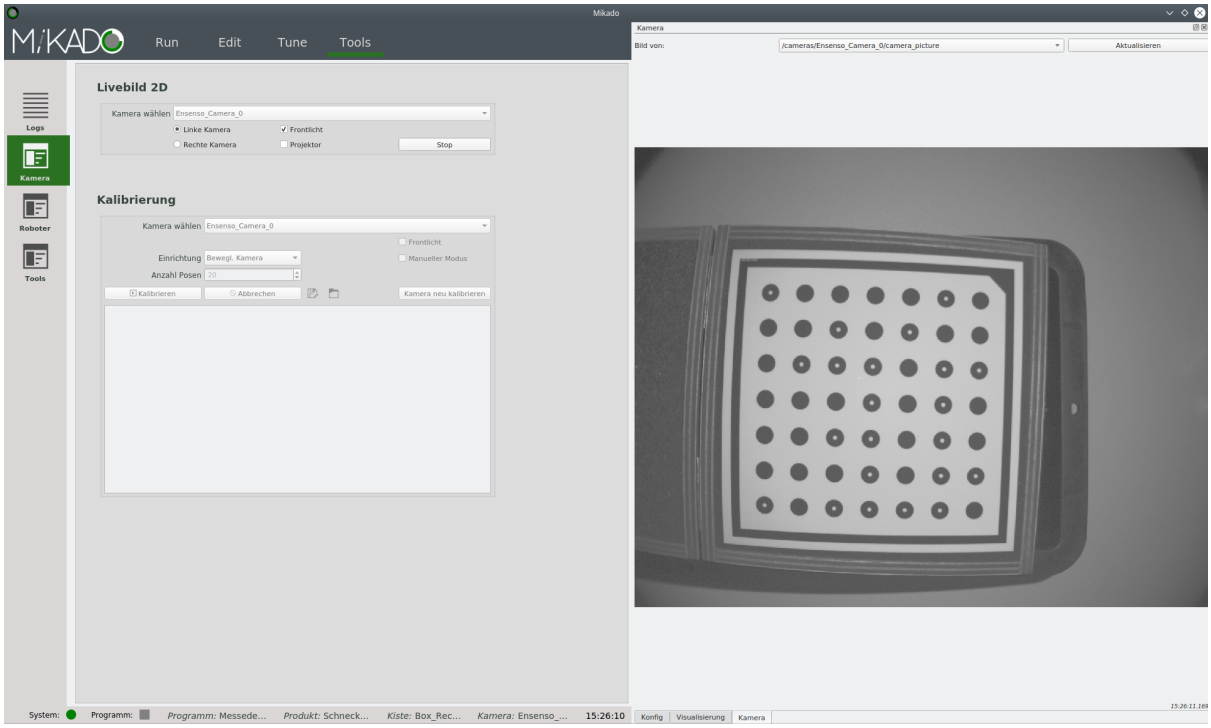
2.17.1 Überblick

Auf der Log-Seite werden die letzten 10.000 Log-Nachrichten angezeigt. Zudem gibt es die Möglichkeite, diese zu filtern. Als Filter-Option gibt es

- Auswählen der Log-Level der anzuzeigenden Nachrichten:
  - Debug
  - Warnung
  - Info
  - Fehler
  - Fataler Fehler
- Auswählen des Zeitraums, für den die anzuzeigenden Nachrichten geloggt wurden
- Text- oder Pattern-Matching auf Nachrichten
- Auswählen der ROS-Nodes, aus denen die Nachrichten kamen

2.18 Kamera-Einstellungen

Die Kameraeinstellungsseite bietet Steuerelemente für eine Live-Ansicht, Direktansicht der Kameradaten sowie die Möglichkeit, die Kamera zu kalibrieren. Die Ansicht ist im Menü unter ‚Tools' → ‚Kamera' zu finden.



Bemerkung: Das hier zur Verfügung gestellte Werkzeug ist lediglich als Ergänzung zu kameraspezifischen Tools gedacht. Um z.B. eine Ensenso Kamera zu verwenden, müssen Sie die Kamerakonfiguration zuerst mithilfe der Ensenso Software (NxView) vornehmen (siehe *Kamerakonfiguration*). So stellen Sie sicher, dass korrekte 3D-Daten mithilfe Ihres 3D-Sensors erfasst werden. Mikado Control kann dann verwendet werden, um die Aufnahmen zu überprüfen oder um zwischen verschiedenen Konfigurationen zu wechseln.

2.18.1 Livebild 2D

Im oberen Bereich der Oberfläche sind die Steuerelemente, um die 2D Bildaufnahmen direkt zu kontrollieren. Das Bild können Sie mithilfe des *Kamera Tabs* betrachten.

- ***Kamera auswählen*:**
  Falls Sie mehrere Kameras am System angeschlossen haben, können Sie hier die Kamera wählen, die Sie verwenden möchten
- ***Linke Kamera / Rechte Kamera*:**
  Mit dieser Auswahl können Sie festlegen, ob Sie das Livebild der linken oder der rechten 2D-Kamera des Stereo-Kamera-Paars anzeigen wollen
- ***Frontlicht*:**
  Aktivieren Sie diese Option, wenn Sie die Szene mit dem Frontlicht bei jeder Aufnahme aktiv beleuchten möchten. Falls Ihr Modell kein Frontlicht besitzt, wird dieser Parameter ausgeblendet und für Filekameras ausgegraut.
- ***Projektor*:**
  Aktivieren Sie den Projektor, wenn Sie bei einer Aufnahme das Pattern auf die Szene projizieren möchten. Dieses Pattern unterstützt die Erfassung von 3D-Daten. Die Auswahlmöglichkeit wird für Filekameras ausgegraut.
- ***Start bzw. Stop*:**
  Mit diesem Button wird die Liveaufnahme gestartet respektive wieder beendet. Eine Kalibrierung kann während einer gestarten Aufnahme nicht durchgeführt werden.

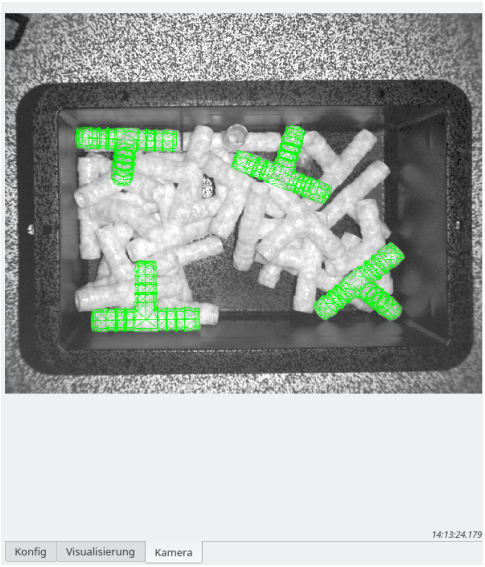
Hinweis: Veränderungen an den Kameraparametern wirken sich nur auf die Ansicht aus. Ein Programmstart, während das 2D Livebild noch aktiv ist wird darüber benachrichtigen, dass das Livebild abgeschaltet wird. Dabei wird auch auf die zuletzt aktiven Frontlicht/Projektor Einstellungen zurückgesetzt.

2.18.2 Kamera-Tab

Im Kamera-Tab können Sie das Bild einer Kamera betrachten. Er befindet sich auf der rechten Seite der Oberfläche. Ist die Liveaufnahme aktiviert (s. *Livebild 2D*), so sehen Sie hier den Livestream der Kamera.

- ***Bild von*:**

  In diesem Auswahlfeld können Sie den Bildkanal wählen. Haben Sie mehrere Kameras angeschlossen, so können Sie die Kamera hier wechseln.
- ***Aktualisieren*:**
  Ein Klick auf diesen Button sucht nach neuen Bildkanälen. In der Regel sollten jedoch Bildkanäle ab Programmstart vollständig verfügbar sein.



- Unten rechts im ‚Kamera' Tab ist ein Zeitstempel zu sehen. Hier erkennen Sie z.B., ob der Livestream momentan aktiviert ist.
- Wurde eine Objektsuche (auch während der Ausführung eines Programms) durchgeführt, so wird auf dem Bildkanal der entsprechenden Kamera das zugrunde liegende Kamerabild mit eingezeichneten Funden dargestellt. Dabei sind die Funde als Gitternetz grün abgebildet.
- Sie können jederzeit auf das Livebild wechseln, die Tabs für ‚Visualisierung' und ‚Kamera' in der rechten Hälfte der Oberfläche unten lassen sich unabhängig von den Menüpunkten bedienen.
- Wird der Bildkanal gewechselt, so wird das momentan dargestellte Bild verworfen und erst wenn neue Bilddaten eintreffen, werden diese dargestellt

2.18.3 Hand-Auge-Kalibrierung

Dieser Abschnitt beschreibt die Durchführung einer Hand-Auge-Kalibrierung. Die Durchführung der TCP-Kalibrierung ist in Abschnitt *TCP kalibrieren* beschrieben. Mikado Control unterstützt verschiedene Kalibrierungsvarianten für die Hand-Auge-Kalibrierung (HAK), die je nach Installation und Zellenkonfiguration gewählt werden. Vor einer Kalibrierung muss sichergestellt werden, dass sich das Kalibriermuster an einer geeigneten Position befindet und die Kamera (oder die Kameras) möglichst zentral auf das Muster blicken. Folgende Tabelle gibt einen Überblick zu den verschiedenen Varianten.

Tab. 9: **Übersicht der verschiedenen HAK-Varianten**

Bewegliche Kamera	Automatische Durchführung für mobile Kamera
Fixe Kamera	Automatische Durchführung für fixe Kamera
Transfer	Für Multi-Kamera-Setup: Schnelle Kalibrierung der Neben-Kamera(s)
Ursprung setzen	Für Select-Zellen oder simulierte Roboter: Setze Ursprung an Patternpose
Roboter	Gibt die Transformationskette des Roboters aus

Tipp: Nutzen Sie den *Kamera-Tab* und eventuell die *Robotersteuerung* für die Ausrichtung der Kamera.

Bemerkung: Die in *Zellgeometrie* definierte Position der Kamera hat keinen Einfluss auf die Hand-Auge-Kalibrierung.

Bedienoberfläche

Im Folgenden sind die einzelnen Elemente der Oberfläche erläutert:

- ***Kamera auswählen*:**
  Falls Sie mehrere Kameras am System angeschlossen haben, können Sie hier die Kamera wählen, die Sie verwenden möchten
- ***Setup*:**
  Wählen Sie hier die Setup-Methode aus, die Ihrem Aufbau entspricht. Wechseln Sie den Setup-Typ, so werden entsprechend Ihrer Auswahl weitere / andere Schaltflächen und Eingabefelder angezeigt. Eine detaillierte Beschreibung der einzelnen Typen finden Sie unter *Setup-Typen*.
- ***Kalibrieren*:**
  Dieser Button startet die Kalibrierung

  Hinweis: Während der Kalibrierung sehen sie im *Kamera-Tab* im Bildkanal der zu kalibrierenden Kamera die einzelnen Aufnahmen. Die Lage des Kalibriermusters wird bei erfolgreicher Erkennung ebenfalls eingezeichnet.

- ***Abbrechen*:**
  Aktuelle Kalibrierung abbrechen
- ***Kamera rekalibrieren*:**
  Führt eine dynamische Stereokalibrierung durch. Diese verbessert die Erkennung des Kalibriermusters und Erfassung der 3D-Daten. Für die Durchführung muss ein Kalibriermuster möglichst mittig im Sichtfeld der Kamera platziert sein.

  Eine Rekalibrierung ist vor allem sinnvoll, nachdem die Kamera bei einer Inbetriebnahme final am System installiert/angeschraubt wurde.
- ***Ausgabefeld*:**
  Das Textfeld dient der Ausgabe von Statusmeldungen während der Kalibrierung. Hier sehen Sie z.B. auch den Fortschritt der Kalibrierungsprozedur.

Setup-Typen

Je nach Anwendung und Vision-System müssen Sie das korrekte Kalibrierverfahren wählen. Im Folgenden sind alle Typen aufgelistet und jeweils erläutert, welche Einstellungen vorgenommen werden können.

Bewegliche Kamera

Bei der Hand-Auge-Kalibrierung mit mobiler Kamera wird die Transformation zwischen dem Kamera-Koordinatensystem und dem Koordinatensystem des Roboterflansch bestimmt. Die Position des Roboterflansch lässt sich mit Hilfe der gemessenen Gelenkwinkel des Roboters in Weltkoordinaten überführen. Somit kann mit Hilfe der Hand-Auge-Kalibrierung (HAK) das Kamerabild ebenfalls in Weltkoordinaten überführt werden.

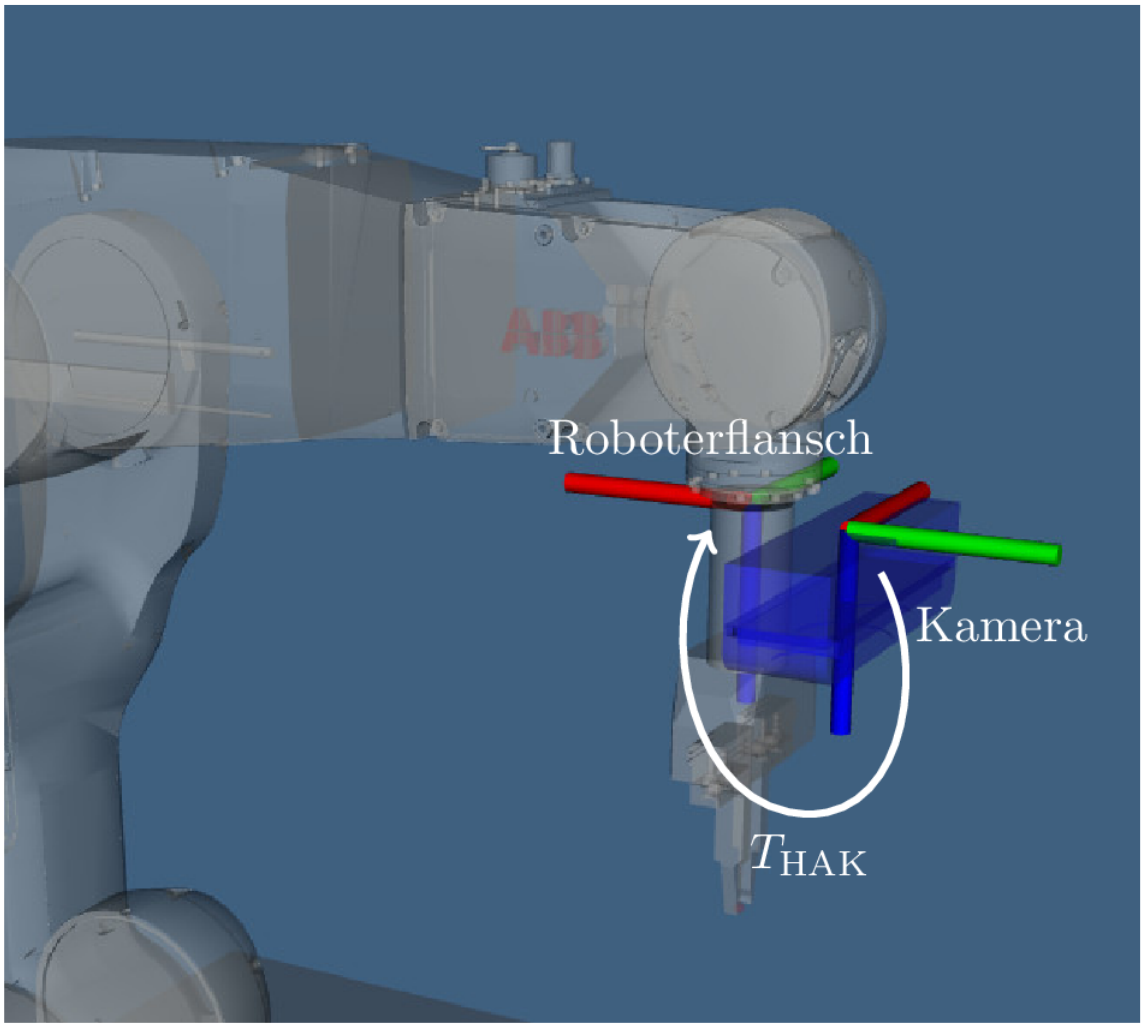


Abb. 50: Durch die Hand-Auge-Kalibrierung bestimmte Transformation bei einer beweglichen Kamera.

Die Kamera befindet sich am Roboterarm und ist daher beweglich. Das Kalibriermuster wird mittig im Arbeitsbereich platziert. Während der Kalibrierung werden verschiedene Roboter-Posen zufallsbasiert angefahren. Ist das Kalibriermuster an dieser Stelle sichtbar, fließt die Aufnahme in die Kalibrierung ein. Der Vorgang wird so lange wiederholt, bis die Anzahl an geforderten Posen erreicht ist. Nach erfolgreicher Berechnung der Kalibrierdaten, werden diese in der Kamera gespeichert.

Weitere Eingabefelder:

- **Manueller Modus**: Keine Automatik-Kalibrierung, weitere Details in Sektion *Manueller Modus*.
- **Anzahl Posen**: Anzahl der validen Posen, die für die Kalibrierung angefahren werden sollen. Posen, bei denen das Kalibriermuster nicht erkannt werden kann, werden dabei nicht gezählt.

Hinweis: Wechseln Sie während der Kalibrierung einer beweglichen Kamera in den ‚Visualisierung'-Tab, sehen Sie die Szene und den Roboter. Zusätzlich werden hier die Posen als einzelne Achs-Darstellungen eingezeichnet, die der Roboter für die Aufnahmen angefahren hat.

Fixe Kamera

Bei der Hand-Auge-Kalibrierung mit fixer Kamera wird die Transformation zwischen dem Kamera-Koordinatensystem und dem Welt-Koordinatensystem bestimmt. Mit Hilfe dieser Transformation lassen sich die Kamerabilder in Weltkoordinaten überführen.

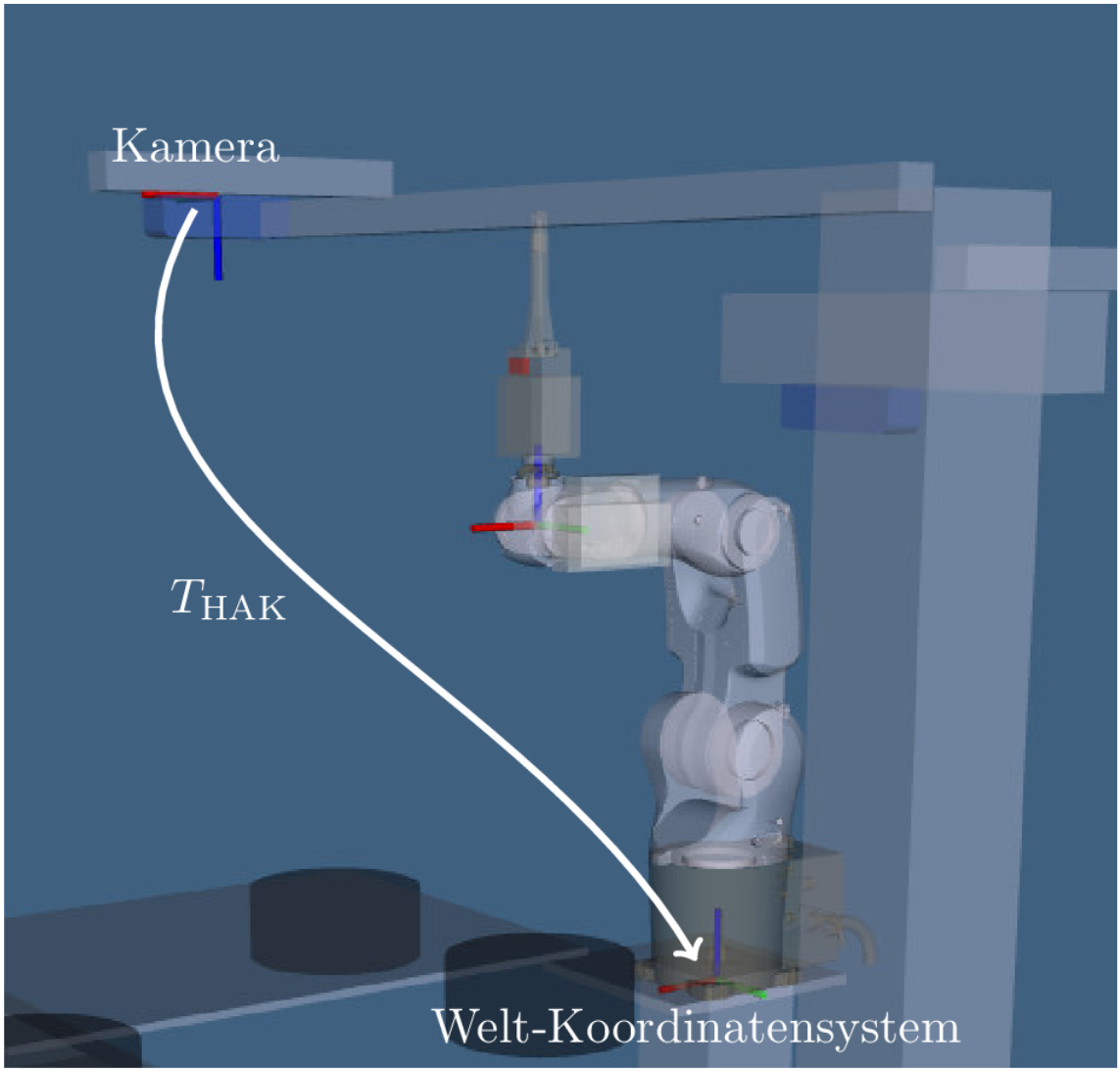


Abb. 51: Durch die Hand-Auge-Kalibrierung bestimmte Transformation bei einer fixen Kamera.

Der Ablauf der Kalibrierung ist derselbe wie im Setup ‚Bewegliche Kamera'. Der Unterschied ist, dass die Kamera statisch an einem festen Ort angebracht wurde. Daher muss das Kalibriermuster am Roboterarm angebracht werden. Die verschiedenen Posen werden nun vom Roboter angefahren und bewegen das Kalibriermuster im Sichtfeld der Kamera. Eine Anleitung zum Anbringen des Kalibriermusters ist unter *Befestigung der Kalibrierplatte* zu finden.

Weitere Eingabefelder:

- **Manueller Modus**: Keine Automatik-Kalibrierung, weitere Details in Sektion *Manueller Modus*.
- **Anzahl Posen**: Anzahl der validen Posen, die für die Kalibrierung angefahren werden sollen. Posen, bei denen das Kalibriermuster nicht erkannt werden kann, werden dabei nicht gezählt.

Transfer

Der *Transfer*-Modus ermöglicht die Übertragung der Kalibrierdaten einer Kamera in eine zweite Kamera, die fix installiert ist. Das Setup wird nur in Systemen mit mehreren Kameras verwandt. Im Feld *Referenzkamera* können Sie die Kamera wählen, die als Quelle verwendet werden soll. Als Referenzkamera kann sowohl eine bewegliche als auch eine fixe Kamera gewählt werden.

Um die Kalibrierung zu übertragen, müssen beide Kameras einen gemeinsamen Bereich einsehen. Positionieren Sie in diesem Bereich das Kalibriermuster und starten die Kalibrierung.

Weitere Eingabefelder:

- **Referenzkamera**: Wählen Sie hier eine bereits kalibrierte Kamera (beweglich oder fix), die als Quelle für die Kalibrierung verwendet werden soll.

Ursprung setzen

Den Ursprung können Sie mithilfe des Kalibriermusters auch direkt setzen. Der Mittelpunkt auf der Oberfläche der Kalibrierplatte wird in diesem Modus standardmäßig als Ursprung verwendet. Indem Sie zusätzlich eine *Pattern Pose* angeben, können Sie den Ursprung relativ zum Kalibriermuster transformieren und somit einen Offset einstellen.

Weitere Eingabefelder:

- **Pattern Pose**: Stellen Sie hier einen relativen Offset zum Kalibriermuster ein. So können Sie den Ursprung auch an eine Position legen, der physikalisch nicht mit dem Kalibriermuster erreichbar ist.

Bemerkung: Beim Ensenso-Kalibriermuster steht die Z-Achse senkrecht nach oben auf der Musterseite der Kalibrierplatte, die X-Achse zeigt parallel zu den Kanten in Richtung ‚Ensenso'-Aufschrift, die Y-Achse beschreibt die dazu rechtwinklig stehenden Kanten.

Roboter

Gibt die Transformationskette des Roboters aus. Diese Transformationskette wird bei der Hand-Auge-Kalibrierung automatisch verwendet, um die Kalibrierung der Robotergelenke zu überprüfen. Kann durch eine Rekalibrierung der Robotergelenke eine signifikante Verbesserung der Qualität der Hand-Auge-Kalibrierung erreicht werden, gibt Mikado eine entsprechende Meldung aus.

Weiterhin kann diese Transformationskette in externen Tools dazu verwendet werden die Nullstellungen der Robotergelenke zu kalibrieren.

Manueller Modus

Im Manuellen Modus können Sie die einzelnen Kalibrierungsschritte selbst steuern. Diese Option steht Ihnen bei den Setup-Typen ‚Bewegliche Kamera' und ‚Fixe Kamera' zur Verfügung. Der Manuelle Modus erlaubt Ihnen zum Beispiel, den Roboter für die einzelnen Aufnahmen manuell in eine geeignete Position zu fahren. Aktivieren Sie diesen Modus, so sehen Sie vier neue Buttons:

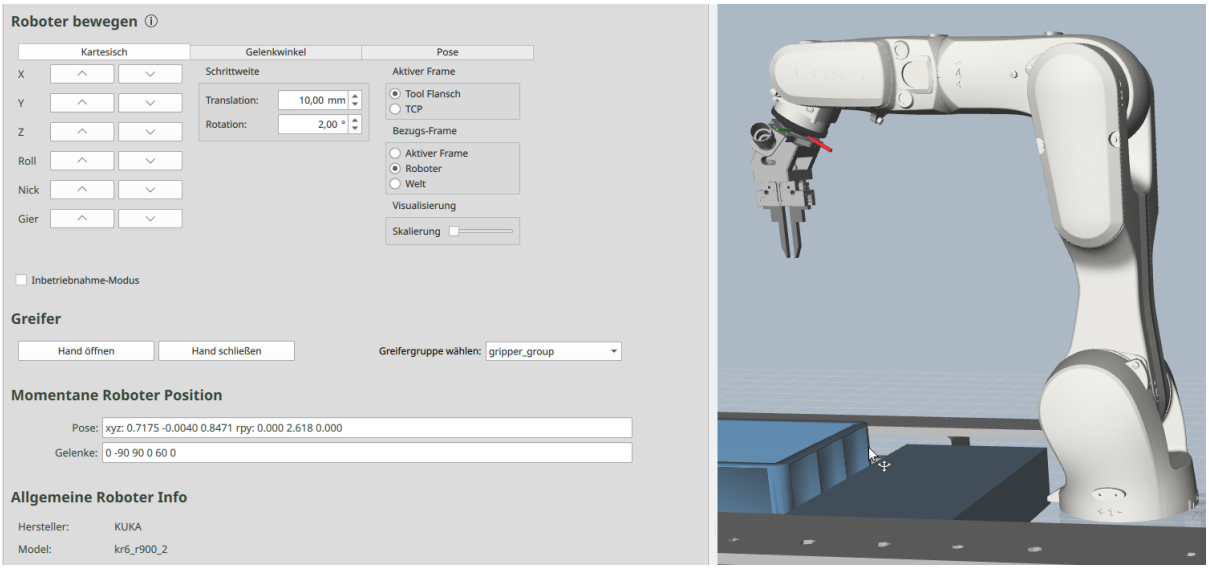
- **Start**: Der Vorgang der Kalibrierung wird initialisiert. Die aktuelle Roboterpose bestimmt die Ausgangsposition der Kalibrierung.
- **Aufnahme**: Hiermit wird eine Kamera-Aufnahme ausgelöst. Falls das Kalibriermuster nicht gefunden werden kann, wird eine Meldung im Ausgabefeld angezeigt.
- **Neue Pose**: Der Roboter wird zu einer zufälligen Pose bewegt. Falls Sie noch keine Vorkalibrierung durchgeführt haben, könnte es sein, dass das Kalibriermuster an dieser Pose nicht vollständig sichtbar ist. Alternativ können Sie den Roboter auch direkt über die Robotersteuerung manuell bewegen, um die nächste gewünschte Aufnahmeposition anzufahren.

- **Vorkalibrieren**: Sind ausreichend erfolgreiche Aufnahmen/Messungen vorhanden, kann die Vorkalibrierung gestartet werden. Nach dieser werden nur noch zufällige Posen angefahren, die das Kalibriermuster im Sichtfeld behalten.

2.19 Robotersteuerung

Die Seite ‚Roboter' bietet eine Schnittstelle und Bedienelemente zu Ihrem angeschlossenen Roboter. Der Roboter kann direkt gesteuert und umpositioniert werden, wobei Sie ebenfalls die aktuelle Position ermitteln können. Nutzer mit erweiterten Kenntnissen können, je nach Robotermodell, auch roboterspezifische Kommandos absetzen. Die direkte Ansteuerung des Roboters erfolgt über die verschiedenen Schaltflächen. Jede Fahrt erfolgt kollisionsgeprüft mit allen Objekten, die von der Objekterkennung gefunden oder in der Szene modelliert wurden.

Es gibt drei Sektionen, **Direkte Bewegungen** , **Greifer** und **Bewegen**.



2.19.1 Direkte Bewegungen

- **Aktiver Link**:

  Definiert für die rotatorisch auszuführenden Bewegungen den Bezugspunkt um den gedreht werden soll. Wenn z.B. **Tcp** ausgewählt wird, wird um den Koordinatenursprung dieses Links gedreht.

- **Koordinatensystem**:

  Die Achsen dieses Koordinatensystems werden für die Ausführung der Bewegungen verwendet. Falls ‚Aktiver Link' ausgewählt wird, so wird der Koordinatenursprung des aktiven Links verwendet. Falls ‚Roboter' ausgewählt wird, wird der Basislink verwendet.

  Als Hilfestellung kann man sich vorstellen, dass die Koordinatenachsen des ausgewählten Koordinatensystems in den Koordinatenursprung des aktiven Links geschoben werden und anschließend als Basis für die Bewegungen verwendet wird.

Warnung: Das Koordinatensystem ‚Roboter' entspricht nicht zwangsweise dem Weltkoordinatensystem!

- **X / Y / Z**:

  Ein Klick auf einen Pfeil erhöht/verringert den Wert auf der jeweiligen Achse um die angegebene Schrittweite (Standardwert: 10 mm). Die Bewegungen werden im Koordinatensystem ‚Aktiver Link' oder ‚Roboter' ausgeführt.

- **Roll / Nick / Gier**:

  Ein Klick auf einen Pfeil erhöht/verringert den Wert auf der jeweiligen Achse um die angegebene Schrittweite (Standardwert: 2 Grad). Die Bewegungen werden im Koordinatensystem ‚Aktiver Link' oder ‚Roboter' ausgeführt.

- **J1 - J6**:

  Ein Klick auf + / - erhöht/verringert den Wert des jeweiligen Gelenks. Damit wird der Roboterarm in Gelenkwinkeln bewegt.

- **Schrittweite**:

  Für kartesische Bewegungen in X/Y/Z-Richtung wird hier die Schrittweite festgelegt. Für rotatorische und Gelenkbewegungen wird hier ebenso die Schrittweite in Grad angegeben.

- **Zeigen**:

  Hiermit können Sie die Koordinatensysteme von ‚Aktiver Link' und ‚Koordinatensystem' (s.o.) anzeigen lassen.

- **Skalierung Koordinatensystem**:

  Hiermit können Sie die angezeigten Koordinatensysteme skalieren.

Warnung:

- Achten Sie darauf, dass sich keine Objekte in der Zelle oder Nähe des Roboters befinden, die nicht in der Szene modelliert sind, während Sie den Roboter steuern.
- Sie können einen Button auch gedrückt halten, allerdings sollten Sie darauf achten, dass der Roboter beim Loslassen eventuell den letzten Befehl ausführt und noch ein Stück weiterfährt. Falls Sie eine höhere Positioniergenauigkeit wünschen, nutzen Sie am besten die originale manuelle Robotersteuerung (Teach Panel).

2.19.2 Inbetriebnahme-Modus

Diese Einstellung ist nur verfügbar, wenn zuvor der *Entwicklermodus* aktiviert wurde. Mit aktiviertem Inbetriebnahme-Modus wird im *System-Status* der Roboterstatus *Handbetrieb / Manueller Modus* nicht als Fehler gewertet. Mikado kann somit trotz dieser Modi in Betriebsbereitschaft versetzt werden, während im regulären Betrieb sonst ein Automatikmodus des Roboters vorausgesetzt wird. Nach dem Ändern der Einstellung ist eine Statusänderung des Roboters notwendig, damit die Einstellung wirksam wird.

Im Mikado Assistant kann diese Einstellung dauerhaft aktiviert werden.

2.19.3 Greifer

- **Öffne Hand**:

  Öffnet den Greifer des Roboters

- **Schließe Hand**:

  Schließt den Greifer des Roboters

- **Greifergruppe wählen**:

  Erlaubt die Auswahl des aktiven Greifers, falls im Assistenten mehrere Greifer angelegt wurden.

2.19.4 Bewegen

- Mithilfe der Kombobox-Schaltfläche kann eine Pose oder Gelenkwinkelposition ausgewählt werden. Zur Auswahl stehen die Posen/Gelenkwinkelpositionen des aktuell geladenen Mikado-Programms.
- Aktivieren Sie die Option ‚Bewegen' um eine Posenvorschau zu erhalten und den Modus zu aktivieren.
- Wählen Sie, mit welchem Roboterteil sich der Roboter zur angegebenen Pose bewegen soll. Dies hat jedoch keine Auswirkung auf Gelenkwinkelpositionen.
- Mit einem Klick auf *Bewegen* führt der Roboter die Bewegung zur angezeigten Zielposition aus.

Unter den Schaltflächen zur Direktsteuerung finden Sie folgende Ausgabeelemente:

- **Pose**:

  Position des Flansches/Handgelenks des Roboters als Pose in der Textdarstellung von Mikado Control.

- **Gelenke**:

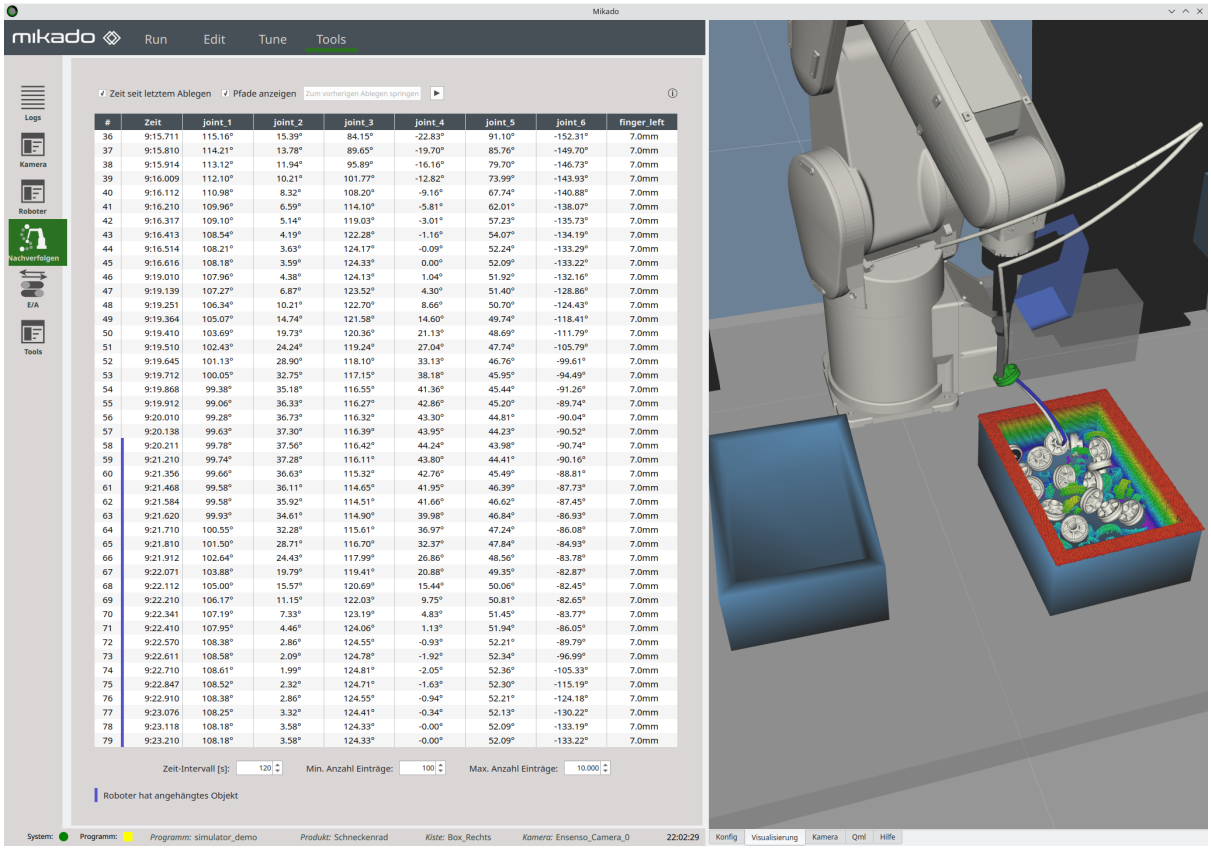
  Positionswerte des Roboters als Gelenkwinkel in Grad.

2.19.5 Roboterparameter

Falls der Roboter modellspezifische Parameter oder Funktionalitäten hat, werden diese ebenfalls auf dieser Seite zu sehen sein. Eine Beschreibung der modellabhängigen Steuerelemente finden Sie im Kapitel *Roboter-Modelle*.

2.20 Roboter Nachverfolgen

Die Seite ‚Nachverfolgen' bietet die Möglichkeit, die ausgeführten Bewegungen des Roboters nachzuverfolgen. Dafür werden die Roboterzustände der letzten N Sekunden zwischengespeichert, wobei N in der Oberfläche einstellbar ist. Die Roboterzustände werden in einer Liste angezeigt und sind auswählbar.



Der obenstehende Screenshot zeigt die Nachverfolgen-Seite. Im Folgenden wird die Bedeutung der verschiedenen Seiten-Elemente erklärt.

- **Zeit seit dem letzten Ablegen:** Ermöglicht, die Darstellung der Zeit-Spalte zu wechseln. Ist das Häkchen gesetzt, werden die Zeiten relativ zum letzten vorherigen Ablegen angegeben (bzw zum Ursprungszustand, wenn noch kein Ablegen stattgefunden hat). Andernfalls wird die Zeit schlicht als Uhrzeit angegeben.
- **Pfade anzeigen:** Ist das Häkchen gesetzt, werden die in der Liste enthaltenen Bewegungsschritte des Roboters zu einem Pfad zusammengefügt und visualisiert. Dabei sind Teilpfade blau eingefärbt, wenn der Roboter ein angehängtes Objekt hat. Ist das Häkchen nicht gesetzt, werden die Pfade ausgeblendet.
- **Zum vorherigen Ablegen springen:** Ist eine Zeile in der Liste ausgewählt, kann damit zum (von dieser Zeile ausgesehen) vorherigen Ablegen gesprungen werden. Ist „Zeit seit dem letzten Ablegen“ gesetzt, springt man somit zu der Zeile, die den Zeit-Wert 0 hat (oder an den Listenanfang, wenn das letzte vorherige Ablegen bereits aus der Liste gefallen ist).
- **Abspiel-Button:** Gibt in der Visualisierung die Bewegung des Roboters wieder, mit den gleichen Zeitabständen wie sie zwischen den Einzelschritten auftreten. Die einzige Ausnahme ist, dass beim Abspielen der maximale Zeitabstand auf 1s beschränkt ist, damit der Nutzer nicht zu lange warten muss, wenn der Roboter sich eine Zeit lang nicht bewegt hat.
- **Zeit-Intervall [s]:** Der Zeit-Intervall gibt an, wie weit das Nachverfolgen zurückreichen soll. Zum Beispiel werden bei 120 Sekunden die Roboterzustände der letzten zwei Minuten zwischengespeichert und angezeigt. Ausnahmen bilden die Fälle, in denen in diesem Zeitraum weniger als „Min. Anzahl Zustände“ oder mehr als „Max. Anzahl Zustände“ vorhanden waren (siehe die folgenden zwei Punkte).
- **Min. Anzahl Zustände:** Dieser Parameter gibt an, wieviele der letzen Zustände mindestens zwischengespeichert und angezeigt werden sollen. Wenn im angegebenen Zeit-Intervall weniger (verschiedene) Zustände vorhanden waren, werden auch ältere Zustände angezeigt.
- **Max. Anzahl Zustände:** Dieser Parameter gibt an, wieviele der letzen Zustände höchstens zwischengespeichert und angezeigt werden sollen. Wenn im angegebenen Zeit-Intervall mehr (verschiedene) Zustände vorhanden waren, wird somit nicht das ganze Zeit-Intervall angezeigt.
- **Balken links:** Wenn ein Zustand in der Liste links einen blauen Balken hat, bedeutet dies, dass dieser Zustand ein angehängtes Objekt hat. So lassen sich Phasen mit und ohne angehängtem Objekt leichter finden.
- **Visualisierung:** Wird ein Zustand in der Liste ausgewählt, so wird dieser in der Visualisierung ohne Transparenz angezeigt. Gleichzeitig wird der aktuelle Roboter-Zustand (der letzte Eintrag in der Liste) halbtransparent angezeigt. Ist kein Zustand in der Liste ausgewählt, wird stets der letzte (der aktuelle) Zustand ohne Transparenz angezeigt.

2.21 Übersicht Roboterein- und ausgänge

Die Seite bietet Funktionen zum Anzeigen von Roboterein- und ausgängen. Zusätzlich dazu können Roboterausgänge gesetzt und die Seite nach eigenen Wünschen konfiguriert werden. Zur Nutzung der im Folgenden beschriebenen Funktionen muss eine Verbindung zum Roboter bestehen. Die Seite soll im Hinblick auf die Fehlersuche im Gesamtablauf unterstützen. Die Ansicht ist im Menü unter ‚Tools‘ → ‚E/A‘ zu finden.

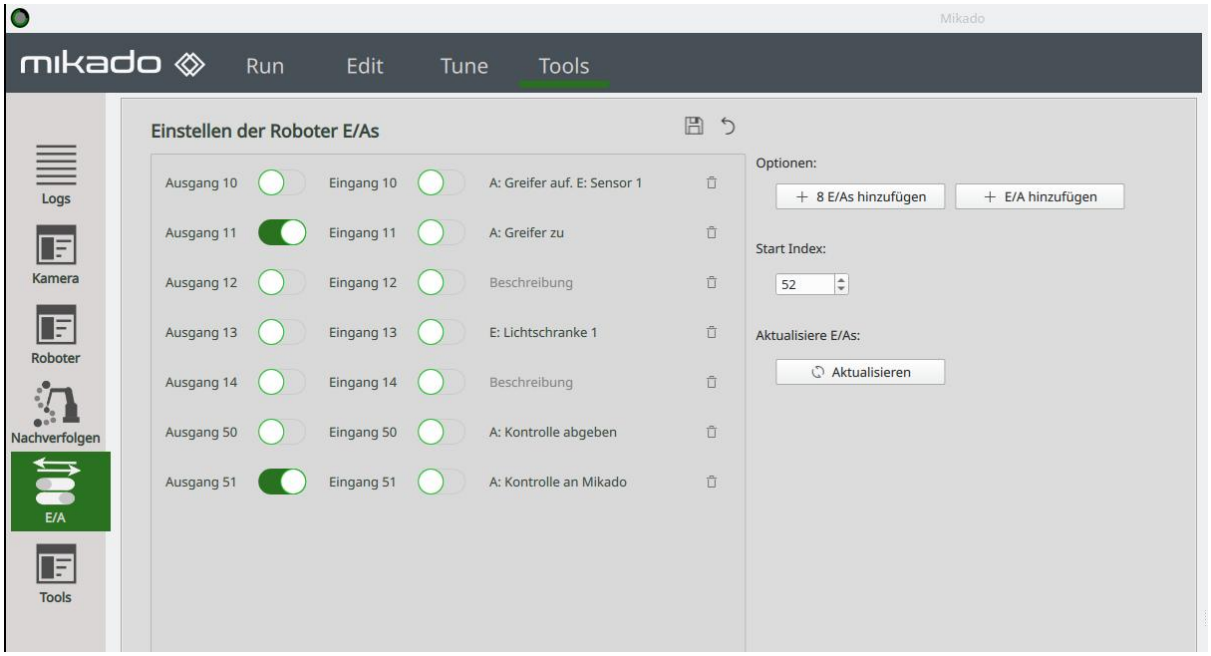


Abb. 52: Screenshot Einstellen der Roboter E/As

2.21.1 Hinzufügen/Entfernen von E/As

Auf der rechten Seite des oben dargestellten Screenshots sind verschiedene Elemente zum Hinzufügen und Entfernen von Ein- und Ausgängen zu sehen.

- ***8 E/As hinzufügen*:**
  Fügt 8 Ein- und Ausgänge hinzu beginnend mit dem Start Index, der in der Spinbox darunter zu sehen ist. Steht die Spinbox auf 10, werden die Ein- und Ausgänge 10 - 17 hinzugefügt.
- ***E/As hinzufügen*:**
  Fügt einen Ein- und Ausgang mit dem Start Index hinzu, der in der darunter stehenden Spinbox zu sehen ist.
- ***Aktualisiere E/As*:**
  Liest alle in der Liste vorhandenen Ein- und Ausgänge und zeigt deren Ergebnis an. Zeigt der Kreis mit der weißen Grundfläche und der grünen Umrandung nach rechts ist der Ausgang *true*, zeigt er nach links ist er *false*.

2.21.2 Setzen von Ausgängen und Beschriftung der E/As

Um einen möglichst guten Überblick über die für die Anwendung wichtigen Ein- und Ausgänge zu haben kann die Liste bearbeitet und gespeichert werden.

- ***Ausgänge setzen*:**
  Durch das Anklicken der Kreises mit der weißen Grundfläche und der grünen Umrandung rechts neben dem Ausgang kann dieser auf *false* oder *true* gesetzt werden.
- ***Beschriftung*:**
  Um den Überblick über die Ein- und Ausgänge nicht zu verlieren können diese beschriftet werden. Wird ein Ein- und Ausgang der gleichen Zahl belegt kann es hilfreich sein dies in der Beschriftung zu vermerken.

- ***Speichern der Liste*:**
  Um die Liste und Beschriftung dauerhaft auch nach dem Neustart von Mikado zur Verfügung zu haben kann diese mit dem Disketten Symbol über der Liste gespeichert werden. Es kann pro Mikado Zelle nur eine Liste abgespeichert werden.
- ***Zurücksetzen der Liste*:**
  Um eine Änderung auf den letzten gespeicherten Stand zurückzusetzen kann der Pfeil neben dem Disketten Symbol betätigt werden.

Besteht keine Verbindung zum Roboter oder diese ist fehlerhaft, kann über Mikado kein Ein- oder Ausgang gesetzt oder gelesen werden. In diesem Fall ist der Kreis rot umrandet und beinhaltet ein rotes Kreuz, wie im folgenden Bild zu sehen ist.

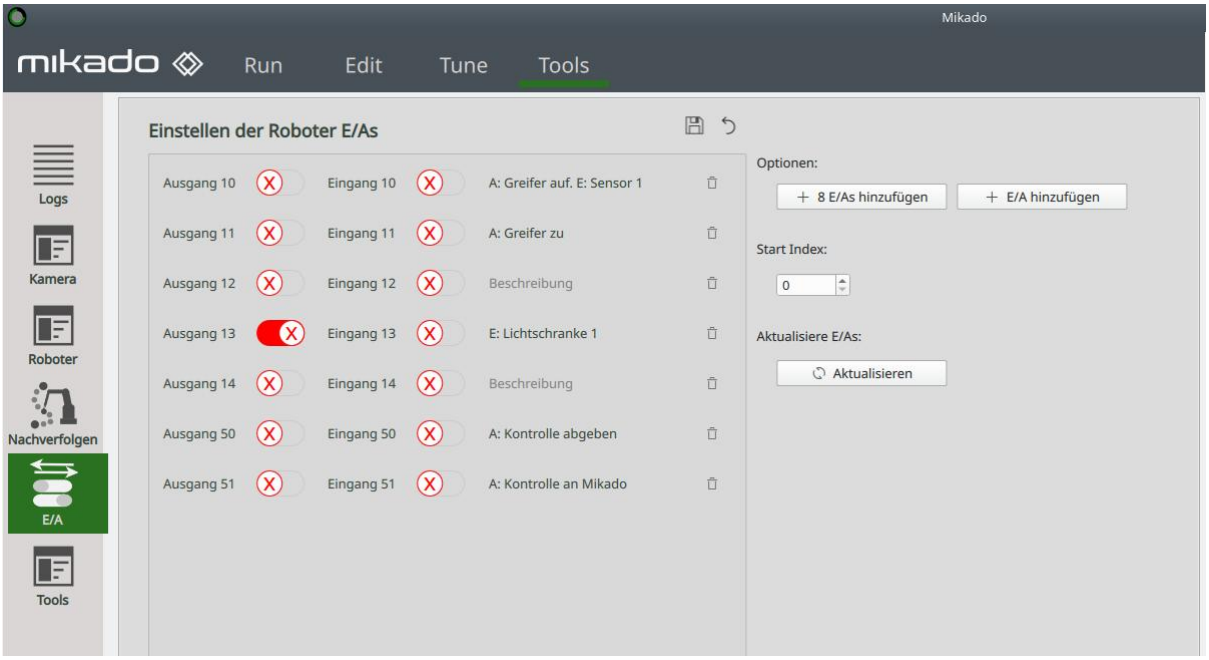


Abb. 53: Screenshot Fehler beim Lesen der Ein- und Ausgänge

2.22 System-Tools

Unter dem Menüpunkt ‚Tools' → ‚Tools' befindet sich die Seite System-Tools. Hier finden sich folgende Elemente:

2.22.1 System

Elemente zur allgemeinen Interaktion mit Mikado:

- **Mikado beenden:** Beendet die Mikado Anwendung nach Anzeige eines Dialogs zur Bestätigung
- **Controller herunterfahren:** Öffnet den Dialog des Betriebssystems zum Herunterfahren bzw. Neustarten des Controllers. Ohne eine weitere Nutzerinteraktion wird der Mikado Controller nach einer Wartezeit von 30 Sekunden automatisch heruntergefahren.
- **Vollbildmodus / Vollbildmodus verlassen:** Umschalten der Mikado-Oberfläche zwischen Vollbildmodus und Fenstermodus.

Bemerkung: Die Elemente ‚*Mikado beenden*' und ‚*Vollbildmodus*' sind mit eingeschaltetem Passwortschutz gesperrt und werden erst nach Eingabe des korrekten Passwortes freigegeben.

2.22.2 Mikado

In diesem Abschnitt sind Informationen über den Softwarestand von Mikado sichtbar. Diese beinhalten beispielsweise die verwendete Mikado-Edition, den installierten Softwarestand und das Verzeichnis der aktuell geöffneten Mikado-Zelle.

2.22.3 Entwickler-Tools

Über die Checkbox „Entwicklermodus aktivieren“ lässt sich die Mikado-Oberfläche in den Entwicklermodus versetzen. Dieser Modus ist für den internen Gebrauch bestimmt und wird z.B. für Test-Zwecke der Software verwendet.

2.22.4 Profiling

Mit Hilfe des sogenannten Profiling können bestimmte Aktivitäten des Mikado Systems aufgezeichnet und anschließend analysiert werden. Hierbei werden die Aktivitäten des Systems mit Zeitstempeln versehen und in einem proprietären Binärformat in einer Datei vom Typ *.nxlog* abgespeichert. Diese Dateien lassen sich dann über die externe Anwendung NxProfiler einsehen, welches die zuvor aufgezeichneten Aktivitäten des Systems auf einer Zeitachse visualisiert. Durch das Profiling lassen sich somit beispielsweise zeitintensive Funktionsaufrufe erkennen, um die Taktzeit einer Anwendung zu optimieren.

Das Mikado System ist dabei in drei verschiedene Abschnitte unterteilt, deren Aktivitäten getrennt voneinander aufgezeichnet werden. Diese sind:

- **Mikado:** Beinhaltet die Aktivitäten des Mikado-Hauptprogramms, insbesondere z.B. den Mikado-Programmablauf, Griffsuche, Bahnplanung des Roboters etc.
- **Ensenso:** Schnittstelle zwischen der Mikado-Anwendung und der Ensenso Kamera mit Kamera-spezifischen Funktionalitäten (z.B. Berechnen der Octomap, Ausschneiden von Objekten aus der Punktwolke, Anwenden von ROIs, Weiterleiten von Funktionsaufrufen an die NxLib)
- **NxLib:** Aktivitäten der Ensenso NxLib API (Programmbibliothek, die der Ensenso-Schnittstelle zugrunde liegt). Beinhaltet tieferliegende Informationen über die Interaktion mit der Kamera, beispielsweise Verarbeitungsfunktionen (Stereo-Matching), PartFinder Suchalgorithmus und Netzwerkkommunikation zwischen Controller und Kamera.



Abb. 54: Screenshot Tools Profiling

Die Art und Anzahl der aufgezeichneten Aktivitäten werden von der jeweils ausgewählten Log-Stufe beeinflusst. Diese lässt sich für jeden der drei genannten Abschnitte individuell per Dropdown-Liste einstellen. Es sind hierbei aufsteigend folgende Stufen auswählbar:

- **Info:** Grundlegende Informationen
- **Debug:** Mehr Informationen über Teilschritte von Befehlen
- **Trace:** Vollständige Protokollierung aller Aufrufe

Hinweis: Mit steigender Log-Stufe werden jeweils auch alle Informationen der darunterliegenden Stufen weiterhin aufgezeichnet, es kommen jedoch gegebenenfalls zusätzliche Informationen hinzu. Log-Stufe *Debug* beinhaltet somit beispielsweise auch alle Informationen aus Stufe *Info*.

Für jeden Abschnitt (Mikado / Ensenso / NxLib) bieten sich somit folgende Interaktionsmöglichkeiten:

- **Log-Stufe Menü:** Auswahl der gewünschten Log-Stufe (siehe oben). Das Menü ist deaktiviert bei bereits aktivem Profiler.
- **Start:** Startet das Profiling für diesen Abschnitt mit der eingestellten Log-Stufe. Bei aktivem Profiler wird automatisch ein Timer gestartet, der die angesammelten Daten alle 15 Sekunden in die vorgesehene Datei schreibt.
- **Stop:** Beendet das Profiling für den Abschnitt und speichert die gesammelten aber noch ungespeicherten Daten in die entsprechende Datei.
- **Log in NxProfiler öffnen:** Öffnet die gespeicherte Datei für den entsprechenden Abschnitt in der externen Anwendung *NxProfiler*.

Warnung: Beim Neustart der Mikado-Anwendung werden die aufgezeichneten Log-Dateien einer Mikado-Zelle aus der vorhergehenden Sitzung geleert und überschrieben. Die Dateien beinhalten somit jeweils nur Informationen über die momentan aktive bzw. zuletzt gestartete Sitzung dieser Zelle. Unterschiedliche Zellen beinhalten jedoch getrennte Dateien, die jeweils nicht von anderen Zellen überschrieben werden.

KAPITEL 3

Tasks

Dieses Kapitel beschreibt alle verfügbaren *Tasks*, die in einem Mikado-Programmablauf verwendet werden können. Jeder der folgenden Tasks wird zuerst durch eine Beschreibung erläutert, gefolgt von bis zu zwei Tabellen.

- Die Tabelle **Variablen** listet alle Variablen auf, die ein Task benötigt oder anbietet. Der Variablentyp und ggf. eine Kurzbeschreibung ergänzen die Beschreibung in den beiden zusätzlichen Spalten.
- Die zweite Tabelle **Sequenzwechsel** beinhaltet alle Sequenzübergänge, die von einem Task ausgelöst werden können. Die erste Spalte enthält den vorgegebenen Variablennamen, wie er im *Programm-Editor* angezeigt wird. Die zweite Spalte zeigt die Bedingung an, die bei Ausführung des Tasks erfüllt sein muss, damit der Sequenzwechsel stattfindet. Eine nähere Erläuterung der Bedingung ist ggf. in der dritten Spalte zu finden.

Weitere Informationen zum Erstellen eines Programmablaufs finden Sie unter *Programm-Editor*.

3.1 Ablauf

3.1.1 Zu Sequenz wechseln

Wechselt zu einer anderen Sequenz.

Um sicherzustellen, dass während der Programmausführung eine bestimmte Sequenz ausgeführt wird, kann dieser Task verwendet werden. Soll z.B. eine Sequenz maximal ein mal ausgeführt werden, kann dieser Task am Ende dieser Sequenz platziert werden, um danach in die nächste auszuführende Sequenz zu wechseln.

Sequenzwechsel

Name	Bedingung	Anmerkung
Nächste	–	–

3.1.2 Wenn-Gruppe

Wertet Bedingungen in einer bestimmten Reihenfolge nacheinander aus und führt bei Eintreten einer Bedingung die entsprechende Subsequenz aus.

Jede Wenn-Gruppe besteht mindestens aus einem `Wenn` Element. Optional können zusätzlich beliebig viele `Sonst Wenn` Elemente sowie ein einziges `Sonst` Element am Ende hinzugefügt werden.

Die einzelnen Bedingungen der Wenn-Gruppe werden der Reihe nach von oben nach unten evaluiert. Wenn eine der Bedingungen wahr ist, wird die entsprechende Subsequenz ausgeführt und alles Folgende in der Wenn-Gruppe wird ignoriert. Wenn keine der gegebenen Bedingungen wahr ist und die Wenn-Gruppe bei der Auswertung das `Sonst` Element erreicht, wird diese Subsequenz ausgeführt. Das heißt `Sonst` ist eine Rückfalloption, die immer dann ausgeführt wird, wenn keine der vorherigen Bedingungen wahr ist.



Abb. 1: Screenshot einer Wenn-Gruppe im Programm-Editor

Variablen

Name	Typ	Beschreibung
lhs	Bool, Integer, Double, String, Pose, Gelenkwinkel, Produkt, Kiste, ROI, Hindernis, PickSearchParamSet	Linke Seite
Operation	Parameter	Auswahl der Vergleichsoperation. Die verfügbaren Operationen sind abhängig vom Variablentyp. Für alle Variablen: • == • != Nur für Integer und Double Variablen: • < • >>br>• <= • >= Nur für Bool Variablen: • `AND` • `OR` • `NAND` • `NOR`
rhs	Bool, Integer, Double, String, Pose, Gelenkwinkel, Produkt, Kiste, ROI, Hindernis, PickSearchParamSet	Rechte Seite

3.1.3 Programm beenden

Beendet das momentan laufende Programm. Die laufende Sequenz wird abgebrochen und die Programmausführung stoppt. Dabei werden standardmäßig alle Variablen auf ihren Initialwert zurückgesetzt und die Szene zurückgesetzt. Die Programmausführung beginnt bei erneutem Start des Programms wieder bei der ersten Sequenz.

Variablen

Name	Typ	Beschreibung
Variablen zurücksetzen	Bool	Gibt an, ob alle Mikado-Variablen auf ihren Initialwert zurückgesetzt werden sollen (optional, Defaultwert: ‚true')
Szene zurücksetzen	Bool	Gibt an, ob die Szene mit ihren Kollisionsobjekten auf den Ausgangszustand zurückgesetzt werden soll (optional, Defaultwert: ‚true')

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.1.4 Fehler

Beende das Programm mit einem Fehler.

Verhält sich analog zum Task *Programm beenden*, jedoch signalisiert dieser Task einen Fehlerzustand. Greifen z.B. externe Systeme auf Mikado Control zu, so ist es unter Umständen wichtig zu wissen, ob ein Programm erfolgreich beendet wurde oder aufgrund eines Fehlers gestoppt wurde.

3.1.5 Warten

Wartet für eine bestimmte Zeit, bevor mit dem Programmablauf fortgefahren wird.

Die zu wartende Zeit wird in Millisekunden angegeben und muss in einer Variable hinterlegt werden.

Variablen

Name	Typ	Beschreibung
Zeitspanne [ms]	Integer	Zeitspanne in Millisekunden, die gewartet werden soll

3.1.6 Warten bis zum Stillstand

Diese Funktion dient zum Erwirken eines Vorlaufstopps.

Dieser Task blockiert das laufende Programm so lange, bis alle durch vorherige Bewegungstasks (z.B. *Bewege Roboter nach*) angestoßenen Bewegungen beendet sind.

Mikado kombiniert manche Teiltrajektorien zu einer durchgehenden Bewegung. Im Falle von Unterbrechungen (z.B. Notaus, Fehler) können beim Abfahren von mehreren Punkten unerwünschte Abkürzungen entstehen. Dies kann durch diesen Task unterbunden werden, da der Roboter zwingend bei diesem Task stoppt und anschließende Bewegungen nicht mit vorherigen kombiniert werden.

3.1.7 Programm unterbrechen

Das Programm wird an der Stelle dieses Tasks angehalten. Die Ausführung kann anschließend durch die Schaltflächen „Start" oder „Nächster Task ausführen" in der Benutzeroberfläche fortgesetzt werden (siehe *Steuerbuttons*).

Bei Verwendung einer Feldbus-Schnittstelle bleibt das `RunningBit` (siehe *Mikado-Status*) im pausierten Programmzustand auf `true`. Das Mikado-Programm befindet sich also aus Sicht des Kommunikationspartners weiterhin in der Ausführung, da es in diesem Zustand nur pausiert aber nicht beendet ist.

Variablen

Name	Typ	Beschreibung
Grund	String	Inhalt der Variable wird auf der Status-Seite als Text im Task-Status angezeigt.

Tipp: Verwenden Sie diesen Task zum Beispiel zur Fehleranalyse, um bei bestimmten Situationen den Ablauf zu unterbrechen.

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.2 Zahlen

3.2.1 Zahl erhöhen

Erhöhe eine Zahl um eins. Das Ergebnis wird der Variable ‚Zahl' zugewiesen.

Variablen

Name	Typ	Beschreibung
Zahl	Integer, Double	Diese Variable wird um eins erhöht.

3.2.2 Zahl verringern

Verringere eine Zahl um eins. Das Ergebnis wird der Variable ‚Zahl' zugewiesen.

Variablen

Name	Typ	Beschreibung
Zahl	Integer, Double	Diese Variable wird um eins verringert.

3.2.3 Arithmetische Operation

Eine mathematische Operation mit zwei Eingabevariablen durchführen. Es stehen die Grundrechenarten Addition, Subtraktion, Multiplikation und Division zur Verfügung.

Variablen

Name	Typ	Beschreibung
Eingabe 1	Integer, Double	–
Eingabe 2	Integer, Double	–
Operation	Parameter	Auswahl der mathematischen Operation: • + Addition • - Subtraktion • × Multiplikation • / Division
Ergebnis	Integer, Double	–

Hinweis: Der vorgeschriebene Typ der Ergebnisvariablen hängt von den Typen der beiden Eingabevariablen ab. Wenn beide Eingabewerte vom Typ `Integer` sind, muss auch das Ergebnis vom Typ `Integer` sein. In diesem Fall werden Ganzzahl-Operationen verwendet.

In allen anderen Fällen, in denen einer oder beide Eingabewerte vom Typ `Double` sind, muss das Ergebnis vom Typ `Double` sein. Andernfalls könnte die Umwandlung einer Gleitkommazahl in eine Ganzzahl zu einem unbeabsichtigten Informationsverlust führen. Bei den Eingabevariablen sind auch Mischformen zulässig, d.h. eine `Integer` Variable und eine `Double` Variable können miteinander verrechnet werden.

Warnung: Bei der Division mit zwei Integern wird die Ganzzahldivision verwendet. Hierbei wird nur der ganzzahlige Teil des Quotienten beibehalten und ein eventueller Rest verworfen.

Beispiel: 7 / 4 = 1, der Rest 3 entfällt.

3.2.4 Logische Operation

Eine logische Operation mit zwei Eingabevariablen durchführen. Mit Hilfe dieser Operationen können zwei `Bool` Variablen logisch miteinander kombiniert werden. Es stehen die Operationen AND, OR, XOR, NAND, NOR und XNOR zur Verfügung. Das Verhalten dieser Operationen kann den unten abgebildeten Wahrheitstabellen entnommen werden.

Variablen

Name	Typ	Beschreibung
Eingabe 1	Bool	–
Eingabe 2	Bool	–
Operation	Parameter	Auswahl der logischen Operation: • `AND` • `OR` • `XOR` • `NAND` • `NOR` • `XNOR`
Ergebnis	Bool	–

Wahrheitstabellen

In den folgenden Tabellen sind alle möglichen Kombinationen von zwei Eingaben und die entsprechenden Ergebniswerte für jede der vorhandenen booleschen Operationen aufgeführt.

AND

Das Ergebnis ist nur dann Wahr, wenn beide Eingaben Wahr sind (logische Und-Verknüpfung).

Eingabe 1	Eingabe 2	Ergebnis (AND)
False	False	False
False	True	False
True	False	False
True	True	True

OR

Das Ergebnis ist Wahr, wenn mindestens eine Eingabe Wahr ist (logische Oder-Verknüpfung).

Eingabe 1	Eingabe 2	Ergebnis (OR)
False	False	False
False	True	True
True	False	True
True	True	True

XOR

Das Ergebnis ist Wahr, wenn genau eine Eingabe Wahr ist, aber nicht beide (Exklusive Oder-Verknüpfung).

Diese Operation wird daher auch als logische Kontravalenz bezeichnet, weil das Ergebnis nur Wahr ist, wenn die beiden Eingaben unterschiedlich sind (d.h. eine ist Wahr und die andere Falsch, aber nicht beide zugleich Wahr oder beide zugleich Falsch).

Eingabe 1	Eingabe 2	Ergebnis (XOR)
False	False	False
False	True	True
True	False	True
True	True	False

NAND

Das Ergebnis ist nur dann Falsch, wenn beide Eingaben Wahr sind; andernfalls ist es Wahr.

Die NAND-Verknüpfung ist das logische Komplement der AND-Verknüpfung, d.h. sie erzeugt genau das entgegengesetzte Ergebnis der AND-Verknüpfung.

Eingabe 1	Eingabe 2	Ergebnis (NAND)
False	False	True
False	True	True
True	False	True
True	True	False

NOR

Das Ergebnis ist nur dann Wahr, wenn beide Eingaben Falsch sind.

Die NOR-Verknüpfung ist das logische Komplement der OR-Verknüpfung, d.h. sie erzeugt genau das entgegengesetzte Ergebnis der OR-Verknüpfung.

Eingabe 1	Eingabe 2	Ergebnis (NOR)
False	False	True
False	True	False
True	False	False
True	True	False

XNOR

Das Ergebnis ist Wahr, wenn die beiden Eingaben gleich sind (beide Wahr oder beide Falsch).

Diese Operation wird daher auch als logische Äquivalenz bezeichnet und ist das logische Komplement zur XOR-Verknüpfung.

Eingabe 1	Eingabe 2	Ergebnis (XNOR)
False	False	True
False	True	False
True	False	False
True	True	True

3.2.5 Vergleichsoperation

Dieser Task ermöglicht es, bei Erfüllung einer bestimmten Bedingung zu einer anderen Sequenz zu springen.

Es werden die beiden Eingabevariablen und der Operations-Parameter ausgewertet. Ist das Ergebnis dieses Vergleichs „wahr“, springt der Task zur Sequenz des Sequenzwechsels `Wahr`, sofern diese angegeben ist. Ist das Ergebnis des Vergleichs „falsch“, so springt der Task zur Sequenz des Sequenzwechsels `Falsch`, sofern diese angegeben ist.

Tritt eine der Bedingungen ein und der zugehörige optionale Sequenzwechsel ist nicht gesetzt, fährt die Programmausführung mit dem nächsten Task der aktiven Sequenz fort. Das heißt, es findet dann kein Sprung zu einer anderen Sequenz statt.

Hinweis: Variablentypen der beiden Eingabevariablen `x` und `y` müssen übereinstimmen.

Variablen

Name	Typ	Beschreibung
x	Bool, Integer, Double, String, Pose, Gelenkwinkel, Produkt, Kiste, ROI, Hindernis, PickSearchParamSet	–
Operation	Parameter	Auswahl der Vergleichsoperation. Die verfügbaren Operationen sind abhängig vom Variablentyp. Für alle Variablen: • == • != Nur für Integer und Double Variablen: • < • >>br>• <= • >= Nur für Bool Variablen: • `AND` • `OR` • `NAND` • `NOR`
y	Bool, Integer, Double, String, Pose, Gelenkwinkel, Produkt, Kiste, ROI, Hindernis, PickSearchParamSet	–

Sequenzwechsel

Name	Bedingung	Anmerkung
Wahr	Ausdruck ist wahr	Sequenzwechsel ist optional
Falsch	Ausdruch ist falsch	Sequenzwechsel ist optional

3.3 Grundlegendes

3.3.1 Kopieren

Kopiert den Wert der Eingabe-Variable in die Ausgabe-Variable.

Achten Sie darauf, dass der Typ der Eingabe-Variable mit dem Typ der Ausgabevariable übereinstimmt.

Hinweis: Möchten Sie Typen konvertieren, können Sie den Task *JavaScript* verwenden.

Variablen

Name	Typ	Beschreibung
Eingabe	alle	Variable mit dem zu kopierenden Wert, bleibt unverändert
Ausgabe	alle	Variable, dessen Wert mit dem Wert der Eingabe-Variable überschrieben wird

3.3.2 Werte tauschen

Tauscht die Werte zweier Variablen. Die Variablentypen müssen übereinstimmen.

Variablen

Name	Typ	Beschreibung
Wert 1	alle	Erhält Wert 2
Wert 2	alle	Erhält Wert 1

3.3.3 Variable zurücksetzen

Setzt die Variable auf ihren Initialwert zurück. Hiermit werden Veränderungen am Wert der Variable durch den Programmablauf verworfen und die Variable enthält den fest definierten Initialwert.

Variablen

Name	Typ	Beschreibung
Variable	alle	

3.3.4 Alle Variablen zurücksetzen

Setzt alle Variablen auf ihren Initialwert zurück. Hiermit werden Veränderungen am Wert der Variablen durch den Programmablauf verworfen und die Variablen erhalten den fest definierten Initialwert.

3.3.5 Ausgangswert setzen

Setzt den momentanen Wert einer Variable als Ausgangswert. Das Program wird als „verändert“ markiert und muss noch gespeichert werden falls die Änderung permanent sein soll.

Variablen

Name	Typ	Beschreibung
Variable	alle	Variable, dessen Ausgangswert gesetzt wird

3.3.6 Negiere Boolsche Variable

Negiert eine boolsche Variable.

D.h. ist die Variable wahr (true) wird sie durch diesen Task auf falsch (false) geändert. Ist die Variable vor Ausführung dieses Tasks falsch, wird sie auf true gesetzt.

Variablen

Name	Typ	Beschreibung
Variable	Boolean	

3.3.7 Kopiere Eintrag aus Liste

Kopiere einen bestimmten Eintrag aus einer Liste in eine Variable.

Hinweis: Möchten Sie lediglich die einzelnen Elemente der Liste nacheinander durchlaufen, so ist der *Kopiere nächsten Eintrag aus Liste* besser geeignet.

Variablen

Name	Typ	Beschreibung
Liste	PoseList, StringList	Eingabeliste
Index	Integer	Index des Eintrags aus der Liste, der ausgelesen werden soll. Für z.B. Index = 0 wird der erste Eintrag der Liste kopiert.
Kopie nach	Pose, String	Ausgabevariable: Kopiere Listeneintrag in diese Variable

---

Hinweis: Die Variablentypen der Eingabeliste und der Ausgabevariable müssen übereinstimmen (z.B. PoseList und Pose).

Sequenzwechsel

Name	Bedingung	Anmerkung
Index zu groß	Index > Listengröße oder Index < 0	Falls der Index nicht valide ist, kann der Eintrag nicht gelesen werden. Deshalb besteht die Möglichkeit, in diesem Fall mit einer anderen Sequenz fortzufahren.

3.3.8 Kopiere nächsten Eintrag aus Liste

Kopiere den nächsten Eintrag aus einer Liste.

Hierfür wird ein interner Zähler verwendet, der bei jeder Ausführung des Task hochzählt. Wird das Ende der Liste erreicht, beginnt der Zähler wieder beim ersten Element. Sie benötigen hier also keine Variable für den Listenindex wie beim Task *Kopiere Eintrag aus Liste*.

Achtung: Falls dieser Task an mehreren Stellen verwendet wird, zählen diese unabhängig voneinander hoch.

Beispielanwendung: Mehrere Aufnahme-Posen bei der Bildaufnahme abwechseln

Variablen

Name	Typ	Beschreibung
Liste	PoseList, StringList	Eingabeliste
Kopie nach	Pose, String	Ausgabevariable: Kopiere Listeneintrag in diese Variable

3.3.9 Zur Liste hinzufügen

Füge ein Element (am Ende) der Liste hinzu.

Variablen

Name	Typ	Beschreibung
Hinzuzufügendes Element	Pose, String	Dieses Element wird am Ende der Liste hinzugefügt
Liste	PoseList, StringList	Liste, der das Element hinzugefügt werden soll

3.3.10 Setze Listenelement

Setze ein Element der gegebenen Liste auf einen bestimmten Wert. Der alte Wert an dieser Stelle wird überschrieben.

Variablen

Name	Typ	Beschreibung
Wert	Pose, String	Wert, der den Wert eines Elements in der Liste überschreiben soll
Index	Integer	Position in der Liste, an der das Element überschrieben werden soll
Liste	PoseList, StringList	Zu modifizierende Liste

3.3.11 Lösche Element aus Liste

Entfernt an der gegebenen Stelle ein Element aus der gegebenen Liste

---

Hinweis: Nützlich, um zum Beispiel eine Reihe von Zielposen abzuarbeiten.

Variablen

Name	Typ	Beschreibung
Liste	Liste	zu bearbeitende Liste
Index	Integer	Stelle des zu löschenden Elements

Sequenzwechsel

Name	Bedingung	Anmerkung
Ungültiger Index	Index<0 oder Index >= Anzahl Items	0-basierte Indizierung der Listenelemente beachten.

3.3.12 Liste leeren

Leere die gegebene Liste.

Alle Einträge der Liste werden entfernt. Danach kann z.B. mit dem Task *Zur Liste hinzufügen* eine neue Liste aufgebaut werden.

Variablen

Name	Typ	Beschreibung
Liste	PoseList, StringList	Zu leerende Liste

3.3.13 Liste leer?

Überprüft, ob die gegebene Liste leer ist.

Variablen

Name	Typ	Beschreibung
Liste	PoseList, StringList	Liste, die geprüft werden soll

Sequenzwechsel

Name	Bedingung	Anmerkung
Liste ist leer	Liste enthält keine Elemente	–
Liste nicht leer	Liste enthält mindestens ein Element	–

3.3.14 Zähle Listenelemente

Ermittle die Anzahl der Elemente in einer Liste (StringList oder PoseList).

Variablen

Name	Typ	Beschreibung
Liste	PoseList, StringList	Eingabe: Liste, deren Länge ermittelt wird
Anzahl der Elemente	Integer	Ausgabe: Anzahl der Elemente in der Liste

3.4 Roboterkontrolle

3.4.1 Bewege Roboter nach

Bewege den Roboter mittels freier Bewegungsplanung, sodass der Tool Flansch (Tool 0 oder Handgelenk) an der Zielpose positioniert wird.

Die direkte Bewegung zur Zielposition wird stets zuerst geprüft. Ist diese nicht möglich, so werden alternative Pfade gesucht und bei Erfolg verwendet. Da die freie Pfadsuche rechenintensiv sein kann, sollten möglichst Positionen gewählt werden, die gut erreichbar sind. Bei Bewegungen um ein bekanntes Hindernis sollten Sie aus diesem Grund in Erwägung ziehen Zwischenpositionen anzufahren.

Hinweis: Roboterposen lassen sich als Pose oder JointPosition (direkt die einzelnen Gelenkwinkel) definieren. Weiterhin lassen sich Roboterposen einlernen / teachen. Wie der Roboter mithilfe der Mikado-Oberfläche gesteuert wird, ist in Abschnitt ‚*Robotersteuerung*‘ beschrieben, das Einlernen der Position in eine Variable ist in Sektion ‚*Bearbeitung von Variablen*‘ erläutert.

Variablen

Name	Typ	Beschreibung
Ziel	Pose, JointPosition	Zielposition, die der Roboter anfahren soll. Die Position definiert die Pose des Tool Flansch (Tool 0 oder Handgelenk) des Roboters.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls die Bahnplanung keinen kollisionsfreien Pfad zur Zielposition finden konnte (aufgrund von z.B. Hindernissen oder einer unmöglichen Position). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Die freie Bahnplanung berechnet bei einem erneuten Versuch eine neue Bewegung, die den Roboterflansch kollisionsfrei an der Zielpose positioniert. Die Bewegung startet nun allerdings von der momentanen Position, bei der er im vorigen Versuch stehen geblieben ist.

3.4.2 Bewege Roboter kartesisch nach

Fahre den Roboter auf geradem Wege zur Zielposition.

Weitere Informationen zum Thema Roboterbewegungen s. ‚*Bewege Roboter nach*‘.

Variablen

Name	Typ	Beschreibung
Ziel	Pose, JointPosition	Zielposition, die der Roboter anfahren soll. Die Position definiert die Pose des Tool Flansch (Tool 0 oder Handgelenk) des Roboters.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls eine kartesische Bewegung zur Zielposition nicht kollisionsfrei möglich ist (aufgrund von z.B. Hindernissen). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Der Roboter bewegt sich bei einem erneuten Versuch nochmals zur Zielposition auf geradem Wege. Er startet aber von der momentanen Position, bei der er im vorigen Versuch stehen geblieben ist.

3.4.3 Bewege Roboter relativ

Bewege den Roboter mittels freier Bewegungsplanung, sodass der Tool Flansch (Tool 0 oder Handgelenk) um die Deltas bewegt oder um die Transformation transformiert wird.

Der Task verhält sich wie der ‚*Bewege Roboter nach*', außer dass keine Zielpose, sondern eine Transformation angegeben wird, die auf die aktuelle Pose angewandt wird um die Zielpose zu berechnen.

Variablen

Name	Typ	Beschreibung
Delta x	double	gewünschte x-Translation
Delta y	double	gewünschte y-Translation
Delta z	double	gewünschte z-Translation
Delta Roll	double	gewünschte Roll-Rotation
Delta Nick	double	gewünschte Nick-Rotation
Delta Gier	double	gewünschte Gier-Rotation
Transformation	Pose	gewünschte Transformation der aktuellen Pose
in lokalen Koordinaten?	bool	ja: Transformation in lokalen Koordinaten nein: Transformation in Welt-Koordinaten

Wenn eine Transformation angegeben wird, müssen die Delta-Variablen leer sein, und umgekehrt.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls die Bahnplanung keinen kollisionsfreien Pfad zur Zielposition finden konnte (aufgrund von z.B. Hindernissen oder einer unmöglichen Position). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Die freie Bahnplanung berechnet bei einem erneuten Versuch eine neue Bewegung, die den Roboterflansch kollisionsfrei an der Zielpose positioniert. Die Bewegung startet nun allerdings von der momentanen Position, bei der er im vorigen Versuch stehen geblieben ist.

3.4.4 Bewege Roboter kartesisch relativ

Fahre den Roboter auf geradem Wege relativ zur aktuellen Position.

Weitere Informationen zum Thema Roboterbewegungen s. ‚*Bewege Roboter nach*'.

Variablen

Name	Typ	Beschreibung
Delta x	double	gewünschte x-Translation
Delta y	double	gewünschte y-Translation
Delta z	double	gewünschte z-Translation
Delta Roll	double	gewünschte Roll-Rotation
Delta Nick	double	gewünschte Nick-Rotation
Delta Gier	double	gewünschte Gier-Rotation
Transformation	Pose	gewünschte Transformation der aktuellen Pose
in lokalen Koordinaten?	bool	ja: Transformation in lokalen Koordinaten nein: Transformation in Welt-Koordinaten

Wenn eine Transformation angegeben wird, müssen die Delta-Variablen leer sein, und umgekehrt.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls eine kartesische Bewegung zur Zielposition nicht kollisionsfrei möglich ist (aufgrund von z.B. Hindernissen). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Der Roboter bewegt sich bei einem erneuten Versuch nochmals zur Zielposition auf geradem Wege. Er startet aber von der momentanen Position, bei der er im vorigen Versuch stehen geblieben ist.

3.4.5 Bewege Tcp nach

Bewege den Roboter mittels freier Bewegungsplanung, sodass der Tool Center Point (TCP oder Endeffektor) bzw. der angegebene Roboter-Link an der Zielpose positioniert wird.

Weitere Informationen zum Thema Roboterbewegungen s. ‚*Bewege Roboter nach*‘.

Variablen

Name	Typ	Beschreibung
Ziel	Pose, JointPosition	Zielposition, die der Roboter anfahren soll. Die Position definiert die Pose des Tool Center Points (TCP oder Endeffektor) des Roboters.
Link	String	Standardmäßig wird der Link *gripper_tcp* als TCP verwendet. Hier kann ein alternativer TCP für die Bewegung spezifiziert werden.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls die Bahnplanung keinen kollisionsfreien Pfad zur Zielposition finden konnte (aufgrund von z.B. Hindernissen oder einer unmöglichen Position). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Die freie Bahnplanung berechnet bei einem erneuten Versuch eine neue Bewegung, die den Tool Center Point kollisionsfrei an der Zielpose positioniert. Die Bewegung startet nun allerdings von der momentanen Position, bei der er im vorigen Versuch stehen geblieben ist.

3.4.6 Bewege Tcp kartesisch nach

Bewege den Roboter in gerader Line, sodass der Werkzeugmittelpunkt (englisch: Tool Center Point, kurz TCP) bzw. der angegebene Roboter-Link an der Zielpose positioniert wird.

Optional kann eine Toleranz in Meter angegeben werden. Dadurch wird bei zwei Sonderfällen kein Fehler ausgelöst:

1. Die Planung erkennt eine Kollision mit einem Bestandteil der Zelle, einem Objekt oder der Punktewolke:

   Nun wird überprüft ob der TCP bis innerhalb der angegebenen Toleranz bewegt werden kann. In diesem Fall wird nur die mögliche Teilbewegung ausgeführt.

2. Die Roboterbewegung wird direkt auf dem Roboter durch einen Eingang oder ähnliches abgebrochen:

   Kommt der TCP innerhalb der Toleranz zum Ziel zum Stehen, wird die Fahrt als erfolgreich betrachtet. Ansonsten wird der Sequenzwechsel „Kollision“ ausgelöst.

   Ein Anwendungsfall wäre z.B. ein Ini mit dem das Einfedern eines gefederten Saugers bei Kontakt mit einer Oberfläche erkannt wird. Das von uns gelieferte Roboterprogramm muss nun so modifiziert werden, das es bei einer Flanke auf diesem Eingang die aktuelle Fahrt abbricht. Typischerweise kann ein Interrupt definiert werden, der auf das Signal reagiert und die Bremsbewegung einleitet. Wir unterstützen Sie gerne bei der entsprechenden Anpassung.

Weitere Informationen zum Thema Roboterbewegungen s. ‚*Bewege Roboter nach*‘.

Variablen

Name	Typ	Beschreibung
Ziel	Pose, JointPosition	Zielposition, die der Roboter anfahren soll. Die Position definiert die Pose des Tool Center Points (TCP oder Endeffektor) des Roboters.
Toleranz	Double	Erlaubter Abstand der Endposition zur Zielposition bei der eine Plannung oder Fahrt als erfolgreich gewertet wird.
Link	String	Standardmäßig wird der Link *gripper_tcp* als TCP verwendet. Hier kann ein alternativer TCP für die Bewegung spezifiziert werden.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls die Bahnplanung keinen kollisionsfreien Pfad zur Zielposition finden konnte (aufgrund von z.B. Hindernissen oder einer unmöglichen Position). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.
	Fahrt wurde unterbrochen	Falls sich der TCP noch außerhalb der Toleranz befindet, wird bei externem Bewegungsabbruch zu dieser Sequenz gewechselt.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Die Bewegung startet von der momentanen Position, bei der im vorigen Versuch stehen geblieben wurde.

3.4.7 Bewege Tcp relativ

Bewege den Roboter mittels freier Bewegungsplanung, sodass der Tool Center Point (TCP oder Endeffektor) bzw. der angegebene Roboter-Link um die Deltas bewegt oder um die Transformation transformiert wird.

Weitere Informationen zum Thema Roboterbewegungen s. ‚*Bewege Roboter nach*‘.

Variablen

Name	Typ	Beschreibung
Delta x	double	gewünschte x-Translation
Delta y	double	gewünschte y-Translation
Delta z	double	gewünschte z-Translation
Delta Roll	double	gewünschte Roll-Rotation
Delta Nick	double	gewünschte Nick-Rotation
Delta Gier	double	gewünschte Gier-Rotation
Transformation	Pose	gewünschte Transformation der aktuellen Pose
Link	String	Standardmäßig wird der Link *gripper_tcp* als TCP verwendet. Hier kann ein alternativer TCP für die Bewegung spezifiziert werden.
in lokalen Koordinaten?	bool	ja: Transformation in lokalen Koordinaten nein: Transformation in Welt-Koordinaten

Wenn eine Transformation angegeben wird, müssen die Delta-Variablen leer sein, und umgekehrt.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls die Bahnplanung keinen kollisionsfreien Pfad zur Zielposition finden konnte (aufgrund von z.B. Hindernissen oder einer unmöglichen Position). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Die freie Bahnplanung berechnet bei einem erneuten Versuch eine neue Bewegung, die den Tool Center Point kollisionsfrei an der Zielpose positioniert. Die Bewegung startet nun allerdings von der momentanen Position, bei der er im vorigen Versuch stehen geblieben ist.

3.4.8 Bewege Tcp kartesisch relativ

Bewege den Roboter in gerader Line, sodass der Werkzeugmittelpunkt (englisch: Tool Center Point, kurz TCP) bzw. der angegebene Roboter-Link relativ zur aktuellen Pose bewegt wird.

Optional kann eine Toleranz in Meter angegeben werden. Dadurch wird bei zwei Sonderfällen kein Fehler ausgelöst:

1. Die Planung erkennt eine Kollision mit einem Bestandteil der Zelle, einem Objekt oder der Punktewolke:

   Nun wird überprüft ob der TCP bis innerhalb der angegebenen Toleranz bewegt werden kann. In diesem Fall wird nur die mögliche Teilbewegung ausgeführt.

2. Die Roboterbewegung wird direkt auf dem Roboter durch einen Eingang oder ähnliches abgebrochen:

   Kommt der TCP innerhalb der Toleranz zum Ziel zum Stehen, wird die Fahrt als erfolgreich betrachtet. Ansonsten wird der Sequenzwechsel „Kollision“ ausgelöst.

   Ein Anwendungsfall wäre z.B. ein Ini mit dem das Einfedern eines gefederten Saugers bei Kontakt mit einer Oberfläche erkannt wird. Das von uns gelieferte Roboterprogramm muss nun so modifiziert werden, das es bei einer Flanke auf diesem Eingang die aktuelle Fahrt abbricht. Typischerweise kann ein Interrupt definiert werden, der auf das Signal reagiert und die Bremsbewegung einleitet. Wir unterstützen Sie gerne bei der entsprechenden Anpassung.

Weitere Informationen zum Thema Roboterbewegungen s. ‚*Bewege Roboter nach*‘.

Variablen

Name	Typ	Beschreibung
Delta x	double	gewünschte x-Translation
Delta y	double	gewünschte y-Translation
Delta z	double	gewünschte z-Translation
Delta Roll	double	gewünschte Roll-Rotation
Delta Nick	double	gewünschte Nick-Rotation
Delta Gier	double	gewünschte Gier-Rotation
Transformation	Pose	gewünschte Transformation der aktuellen Pose
in lokalen Koordinaten?	bool	ja: Transformation in lokalen Koordinaten nein: Transformation in Welt-Koordinaten
Toleranz	Double	Erlaubter Abstand der Endposition zur Ziel- position bei der eine Plannung oder Fahrt als erfolgreich gewertet wird.
Link	String	Standardmäßig wird der Link *gripper_tcp* als TCP verwendet. Hier kann ein alternativer TCP für die Bewegung spezifiziert werden.

Wenn eine Transformation angegeben wird, müssen die Delta-Variablen leer sein, und umgekehrt.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls die Bahnplanung keinen kollisionsfreien Pfad zur Zielposition finden konnte (aufgrund von z.B. Hindernissen oder einer unmöglichen Position). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.
	Fahrt wurde unterbrochen	Falls sich der TCP noch außerhalb der Toleranz befindet, wird bei externem Bewegungsabbruch zu dieser Sequenz gewechselt.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Die Bewegung startet von der momentanen Position, bei der im vorigen Versuch stehen geblieben wurde.

3.4.9 Bewege Teil nach

Bewege den Roboter mittels freier Bewegungsplanung, sodass das angehängte / gegriffene Teil an der Zielpose positioniert wird.

Weitere Informationen zum Thema Roboterbewegungen s. ‚*Bewege Roboter nach*‘.

Wenn das Produkt des angehängten Objekts eine Symmetrie besitzt (siehe *Symmetrien*), werden auch entsprechend der Symmetrie gedrehte Posen berücksichtigt. Beispielsweise werden bei einem zylindrischen Objekt mit kontinuierlicher Rotationssymmetrie um X und diskreter 180°-Rotationssymmetrie um Z auch einige Rotationen um die Längsachse des Zylinders sowie die um 180° gedrehten Varianten untersucht. Die Bewegung findet zur kollisionsfreien Lösung mit dem geringsten Abstand zur Vorposition im Gelenkraum statt.

Variablen

Name	Typ	Beschreibung
Ziel	Pose, JointPosition	Zielposition, die der Roboter anfahren soll. Die Position definiert die Pose des momentan angehängten / gegriffenen Teils.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls die Bahnplanung keinen kollisionsfreien Pfad zur Zielposition finden konnte (aufgrund von z.B. Hindernissen oder einer unmöglichen Position). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Die freie Bahnplanung berechnet bei einem erneuten Versuch eine neue Bewegung, die das Teil kollisionsfrei an der Zielpose positioniert. Die Bewegung startet nun allerdings von der momentanen Position, bei der er im vorigen Versuch stehen geblieben ist.

3.4.10 Bewege Teil kartesisch nach

Bewege das angehängte / gegriffene Teil von der aktuelle Pose in einer geraden Bewegung zur Zielposition.

Weitere Informationen zum Thema Roboterbewegungen s. ‚*Bewege Roboter nach*‘.

Wenn das Produkt des angehängten Objekts eine Symmetrie besitzt (siehe *Symmetrien*), werden auch entsprechend der Symmetrie gedrehte Posen für die Zielposition berücksichtigt. Beispielsweise werden bei einem zylindrischen Objekt mit kontinuierlicher Rotationssymmetrie um X und diskreter 180°-Rotationssymmetrie um Z auch einige Rotationen um die Längsachse des Zylinders sowie die um 180° gedrehten Variante untersucht. Die Bewegung findet zur kollisionsfreien Lösung mit dem geringsten Abstand zur Vorposition im Gelenkraum statt.

Variablen

Name	Typ	Beschreibung
Ziel	Pose, JointPosition	Zielposition, die der Roboter anfahren soll. Die Position definiert die Pose des momentan angehängten / gegriffenen Teils.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls eine kartesische Bewegung zur Zielposition nicht kollisionsfrei möglich ist (aufgrund von z.B. Hindernissen). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Der Roboter bewegt das Teil bei einem erneuten Versuch nochmals zur Zielposition auf geradem Wege. Er startet aber von der momentanen Position, bei der er im vorigen Versuch stehen geblieben ist.

3.4.11 Bewege Teil relativ

Bewege den Roboter mittels freier Bewegungsplanung, sodass das angehängte / gegriffene Teil um die Deltas bewegt oder um die Transformation transformiert wird.

Der Task verhält sich wie der ‚*Bewege Teil nach*‘, außer dass keine Zielpose, sondern eine Transformation angegeben wird, die auf die aktuelle Pose angewandt wird um die Zielpose zu berechnen.

Variablen

Name	Typ	Beschreibung
Delta x	double	gewünschte x-Translation
Delta y	double	gewünschte y-Translation
Delta z	double	gewünschte z-Translation
Delta Roll	double	gewünschte Roll-Rotation
Delta Nick	double	gewünschte Nick-Rotation
Delta Gier	double	gewünschte Gier-Rotation
Transformation	Pose	gewünschte Transformation der aktuellen Pose
in lokalen Koordinaten?	bool	ja: Transformation in lokalen Koordinaten nein: Transformation in Welt-Koordinaten

Wenn eine Transformation angegeben wird, müssen die Delta-Variablen leer sein, und umgekehrt.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls die Bahnplanung keinen kollisionsfreien Pfad zur Zielposition finden konnte (aufgrund von z.B. Hindernissen oder einer unmöglichen Position). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Die freie Bahnplanung berechnet bei einem erneuten Versuch eine neue Bewegung, die das Teil kollisionsfrei an der Zielpose positioniert. Die Bewegung startet nun allerdings von der momentanen Position, bei der er im vorigen Versuch stehen geblieben ist.

3.4.12 Bewege Teil kartesisch relativ

Bewege das angehängte / gegriffene Teil in einer geraden Bewegung relativ zur aktuelle Pose.

Weitere Informationen zum Thema Roboterbewegungen s. ‚*Bewege Roboter nach*‘.

Wenn das Produkt des angehängten Objekts eine Symmetrie besitzt (siehe *Symmetrien*), werden auch entsprechend der Symmetrie gedrehte Posen für die Zielposition berücksichtigt. Beispielsweise werden bei einem zylindrischen Objekt mit kontinuierlicher Rotationssymmetrie um X und diskreter 180°-Rotationssymmetrie um Z auch einige Rotationen um die Längsachse des Zylinders sowie die um 180° gedrehten Variante untersucht. Die Bewegung findet zur kollisionsfreien Lösung mit dem geringsten Abstand zur Vorposition im Gelenkraum statt.

Variablen

Name	Typ	Beschreibung
Delta x	double	gewünschte x-Translation
Delta y	double	gewünschte y-Translation
Delta z	double	gewünschte z-Translation
Delta Roll	double	gewünschte Roll-Rotation
Delta Nick	double	gewünschte Nick-Rotation
Delta Gier	double	gewünschte Gier-Rotation
Transformation	Pose	gewünschte Transformation der aktuellen Pose
in lokalen Koordinaten?	bool	ja: Transformation in lokalen Koordinaten nein: Transformation in Welt-Koordinaten

Wenn eine Transformation angegeben wird, müssen die Delta-Variablen leer sein, und umgekehrt.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kollision	Kollisionsfreier Pfad konnte nicht gefunden werden	Wird hier eine Sequenz angegeben, so wird zu dieser Sequenz gewechselt, falls eine kartesische Bewegung zur Zielposition nicht kollisionsfrei möglich ist (aufgrund von z.B. Hindernissen). Bleibt dieses Feld leer, so wird die Programmausführung in diesem Fall angehalten, ohne dass sich der Roboter zur Zielposition bewegt hat.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Der Roboter bewegt das Teil bei einem erneuten Versuch nochmals zur Zielposition auf geradem Wege. Er startet aber von der momentanen Position, bei der er im vorigen Versuch stehen geblieben ist.

3.4.13 Roboterposition auslesen

Liest die momentane Pose des Roboters (Handgelenk) aus und speichert diese in einer Variable.

Variablen

Name	Typ	Beschreibung
Roboterposition	Pose	Ausgabevariable: Momentane Position des Handgelenks

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.4.14 Teileposition auslesen

Liest die momentane Pose eines angehängten Teils aus und speichert diese in einer Variable. Ist kein Teil angehängt wird zu einer Fehlerbehandlung gesprungen.

Variablen

Name	Typ	Beschreibung
Teileposition	Pose	Ausgabevariable: Momentane Position des Teils

Sequenzwechsel

Name	Bedingung	Anmerkung
Kein Teil gegriffen	Es ist momentan kein Teil angehängt	Falls kein Teil angehängt ist, kann dessen Position auch nicht gelesen werden. Deshalb muss in diesem Fall eine Fehlerbehandlung stattfinden oder das Programm wird mit einem Fehler beendet

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.4.15 TCP Position auslesen

Liest die momentane TCP Position des Roboters aus und speichert diese in einer Variable.

Variablen

Name	Typ	Beschreibung
TCP Position	Pose	Ausgabevariable: Momentane Position des TCP

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.4.16 Ist Roboter Position mit Teil erreichbar?

Prüfe, ob gegebene Teil-Pose vom Roboter erreicht werden kann und kollisionsfrei ist.

Wenn das Produkt des angehängten Objekts eine Symmetrie besitzt (siehe *Symmetrien*), werden auch entsprechend der Symmetrie gedrehte Posen berücksichtigt. Beispielsweise werden bei einem zylindrischen Objekt mit kontinuierlicher Rotationssymmetrie um X und diskreter 180°-Rotationssymmetrie um Z auch einige Rotationen um die Längsachse des Zylinders sowie die um 180° gedrehte Variante probiert.

Variablen

Name	Typ	Beschreibung
Ziel	Pose	Zielpose des Teils, die getestet werden soll

Sequenzwechsel

Name	Bedingung	Anmerkung
Nicht Erreichbar	Zielpose nicht erreichbar	–

3.4.17 Ist Roboter Position erreichbar?

Prüfe, ob gegebene Roboterpose vom angegebenen Link des Roboters (z.B. **gripper_tcp** für den TCP) erreicht werden kann und kollisionsfrei ist.

Variablen

Name	Typ	Beschreibung
Ziel	Pose	Zielpose, deren Erreichbarkeit getestet werden soll
Link	String	Link des Roboters, welches das Ziel erreichen soll

Sequenzwechsel

Name	Bedingung	Anmerkung
Nicht Erreichbar	Zielpose nicht erreichbar	–

3.4.18 Hand öffnen/schließen

Öffnet oder schließt den Greifer. Ist die Eingabe-Variable auf ‚true', wird der Greifer geöffnet, auf ‚false' wird er geschlossen.

Hat der Greifer bereits den gewünschten Zustand, bewirkt dieser Task keine Veränderung.

Variablen

Name	Typ	Beschreibung
Hand öffnen	Bool	‚true': Greifer soll geöffnet werden (bzw. bleiben) ‚false': Greifer soll geschlossen werden (bzw. bleiben)

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.4.19 Greiferzustand einstellen

Versetzt den Greifer in einen erweiterten konfigurierten Zustand (auch **additional gripper state** genannt).

Mit diesem Zustandswechsel können beispielsweise IO-Signale gesetzt und abgewartet werden. Die Verwendendung erweiterter Greiferzustände ermöglicht es komplexe Greifer zu modellieren, die neben offen/geschlossen noch weitere noch weitere Zustände annehmen können.

Variablen

Name	Typ	Beschreibung
Zustand	String	Name des neuen Greiferzustands, muss in *gripper.yaml* innerhalb von *additional_states* definiert sein. Beispiel: ‚Stehend' und ‚Geklappt' für einen Greifer mit zusätzlicher Schwenkfunktion.

Bemerkung: Die `joint_values` eines Greifers mit `additional_state` müssen derzeit von einem projektspezifischen Script mit einem `joint_state_publisher` veröffentlicht werden. Kontaktieren Sie den Mikado Support für Details.

Verhalten beim Greifen

Besitzt ein Greifer konfigurierte, erweiterte Zustände, gibt es für jeden Zustand ein eigene Griff-Kategorie. Im folgenden Beispiel besitzt der Greifer ein pneumatisches Schwenkgelenk, um die Greiferfinger zu klappen. Dieses Gelenk hat zwei Zustände: „Stehend" und „Geklappt". Griffe können für jeden dieser Zustände separat definiert werden.

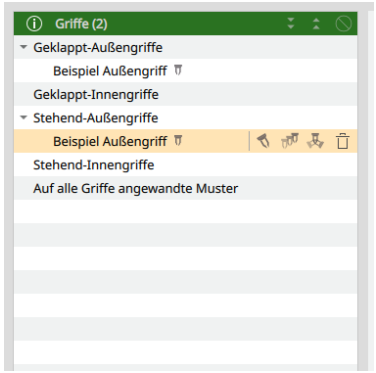


Abb. 2: Erweiterte Zustände im Griffeditor

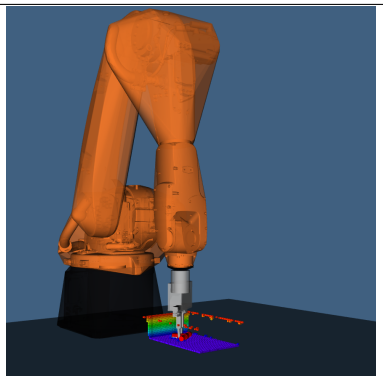


Abb. 3: Außengriff im Zustand „Stehend"

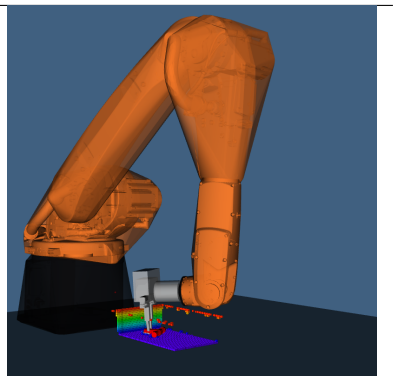


Abb. 4: Außengriff im Zustand „Geklappt"

Gegriffen wird immer mit dem besten Griff (siehe *Beste Griffe*). Die Griffsuche berücksichtigt in ihren Priorisierungskriterien die erweiterten Zustände jedoch nur implizit. Im obigen Beispiel ist (von den zwei möglichen) der „stehende" beste Griff, da das Handgelenk höher gelegen ist und die Trajektoriendistanz wesentlich gering ist.

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

Abb. 5: Priorisierung der beiden Zustände

Sollte der beste Griff einem anderen Zustand angehören, als dem momentanen Zustand, ruft der *Teil greifen*-Task intern den *Greiferzustand einstellen*-Task auf. Dies bedeutet, dass der Greiferzustand in diesem Fall automatisch gewechselt wird (bzw. das Signal dazu gesendet wird).

Zusammenfassung:

1. Griffsuche liefert „besten Griff“
2. Gehört der beste Griff zu einem anderen Zustand als dem momentanen Zustand?
3. Wenn Nein -> weiter zu 5.
4. Greiferzustand entsprechend des „besten Griffs“ einstellen
5. Teil greifen

Detailbeschreibung

Es gibt insgesamt 4 mögliche Aktionen die beim Setzen eines erweiterten Zustands durchgeführt werden können:

- `set_outputs_true`: setzt die der gegebenen Signalnummern entsprechenden IO-Ports auf `true`
- `set_outputs_false`: setzt die der gegebenen Signalnummern entsprechenden IO-Ports auf `false`
- `wait_for_inputs_true`: blockiert bis an den entsprechenden IO-Ports `true` anliegt
- `wait_for_inputs_false`: blockiert bis an den entsprechenden IO-Ports `false` anliegt

Zusätzlich können noch veränderte Gelenkstellung des Greifers für erweiterte Zustände mittels folgenden zwei Parametern definiert werden:

- `joint_names`: Liste von Zusatzgelenken innerhalb des Greifers
- `joint_values`: Gelenkwerte [rad], die die Zusatzgelenke in diesem Zustand annehmen

Beispiel einer entsprechenden Sektion für eine erweiterte *gripper.yaml* Konfiguration:

```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"
```

Diese `additional_states` werden, sofern in der `gripper.yaml` definiert, bei neu angelegten Produkten automatisch hinzugefügt. Ein Beispiel für eine Griffdefinition mit ‚stehendem‘ und ‚geklapptem‘ Zustand:

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

grasps:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
- 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:
      []
```

Bemerkung: Für bereits existierende Produkte muss der zusätzliche Greifer-Zustand für eine bestimmte Griffkategorie manuell in die `<productName>.product`-Datei eingetragen werden.

3.4.20 Setze aktiven Greifer

Wechselt den aktiven Greifer zu einem der konfigurierten Greifer. Der Wechsel kann nur durchgeführt werden, wenn am Greifer kein Teil angehängt ist.

Zum Einrichten von Griffen bei Wechselgreifern siehe *Einrichten von Griffen bei Wechselgreifern*.

Variablen

Name	Typ	Beschreibung
Greifer	String	Name des konfigurierten Greifers

3.4.21 Roboterprogramm ausführen

Führe ein natives Roboterprogramm aus.

Native Roboterprogramme werden direkt auf einem Robotercontroller hinterlegt und/oder programmiert und definieren spezifische Abläufe in der nativen Programmiersprache. Hierdurch können z.b. zusätzliche Fähigkeiten des Robotersystem genutzt werden. Um ein solches Programm zu erstellen, konsultieren Sie bitte die Dokumentation Ihres Roboters.

Voraussetzungen, um robotersepzifische Programme korrekt zu erstellen und vorzubereiten, sind in Kapitel *Roboter-Modelle* in den entsprechenden Sektionen erläutert.

Eigenes Roboterprogramm hinzufügen für:

- *Universal Robot*
- *Kuka*
- *Fanuc*
- Bitte kontaktieren Sie uns, falls Ihr Roboter-Hersteller noch nicht aufgeführt ist

Variablen

Name	Typ	Beschreibung
Programname	String	Dateinamen oder Programmnamen des nativen Roboterprogramm auf dem Controller
Integer Argument Name	String	Variablenname einer Ganzzahl-Variable auf dem Controller
Integer Argument	Integer	Wert der obigen Integer-Variable
Double Argument Name	String	Variablenname einer Gleitkomma-Variable auf dem Controller
Double Argument	Double	Wert der obigen Gleitkomma-Variable
String Argument Name	String	Variablenname einer Text-Variable auf dem Controller
String Argument	String	Wert der obigen Text-Variable
Pose Argument Name	String	Variablenname einer Pose-Variable auf dem Controller
Pose Argument	Pose	Wert der obigen Pose-Variable
Rückgabewert	Double	Das Roboterprogramm kann den Rückgabewert modifizieren und somit Feedback an Mikado Control geben. In der hier angegebene Variable wird der Rückgabewert nach Ausführung des Tasks hinterlegt.

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.4.22 Roboterausgangssignal setzen

Setzt den Wert eines Roboter- (oder SPS-) Ausgangs.

Die Ausgang-Bits eines Roboters können für weitere Systemkomponenten wie z.B. LEDs oder Türen usw. verwendet werden.

Warnung: Roboterausgangssignale können sowohl von Roboterprogrammen, als auch über die Mikado-Ablaufsteuerung modifiziert werden. Vorsicht ist geboten, wenn beide Komponenten auf demselben Ausgang schreiben.

Variablen

Name	Typ	Beschreibung
Signalnummer	Integer	Ausgangssignale eines Roboter-Controllers sind eindeutig durchnummeriert.
Wert	Bool	Der gewünschte Wert des Ausgangssignals (‚true': 1 bzw. ‚high', ‚false': 0 bzw. ‚low')
Puls-Dauer [ms]	Integer	Diese Variable kann bei Bedarf gesetzt werden. Hierbei kann der Output für eine Zeitspanne gesetzt werden und nimmt anschließend den verherigen Wert an.

3.4.23 Roboterausgangssignal lesen

Liest den Wert eines Roboter- (oder SPS-) Ausgangs.

Der momentane Zustand eines Ausgangssignals kann hiermit gelesen und verifiziert werden.

Hinweis: Roboterausgangssignale können sowohl von Roboterprogrammen, als auch über die Mikado-Ablaufsteuerung modifiziert werden.

Variablen

Name	Typ	Beschreibung
Signalnummer	Integer	Ausgangssignale eines Roboter-Controllers sind eindeutig durchnummeriert.
Ausgabe	Bool	Der eingelesene Wert des Eingangssignals wird in diese Variable geschrieben (‚true': 1 bzw. ‚high', ‚false': 0 bzw. ‚low')

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.4.24 Robotereingangssignal lesen

Liest den Wert eines Roboter- (oder SPS-) Eingangs

Die Eingangs-Bits eines Roboters können mit weiteren Systemkomponenten wie z.B. Druckknöpfe, Lichtschranken oder Gebersensorik usw. verbunden werden.

Variablen

Name	Typ	Beschreibung
Signalnummer	Integer	Ausgangssignale eines Roboter-Controllers sind eindeutig durchnummeriert.
Ausgabe	Bool	Der eingelesene Wert des Eingangssignals wird in diese Variable geschrieben (‚true': 1 bzw. ‚high', ‚false': 0 bzw. ‚low')

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.4.25 Auf Robotereingangssignal warten

Fortlaufend Robotereingangssignal lesen und warten, bis es mit dem gegebenen Wert übereinstimmt.

Variablen

Name	Typ	Beschreibung
Signalnummer	Integer	Signalnummer des Roboter-Controllers
Wert	Integer, Bool	Der gewünschte Wert, den das Eingangssignal annehmen soll.
Timeout [ms]	Integer	Maximale Wartezeit in Millisekunden. Nimmt das Signal innerhalb dieser Zeit den gewünschten Wert nicht an, so wird der Sequenzwechsel ‚Timeout erreicht' stattfinden.

Sequenzwechsel

Name	Bedingung	Anmerkung
Timeout erreicht	Innerhalb der Timeout-Zeit hat das Signal nicht den gewünschten Wert angenommen	

3.4.26 Auf Roboterausgangssignal warten

Fortlaufend Roboterausgangssignal lesen und warten, bis es mit dem gegebenen Wert übereinstimmt.

Variablen

Name	Typ	Beschreibung
Signalnummer	Integer	Signalnummer des Roboter-Controllers
Wert	Integer, Bool	Der gewünschte Wert, den das Ausgangssignal annehmen soll.
Timeout [ms]	Integer	Maximale Wartezeit in Millisekunden. Nimmt das Signal innerhalb dieser Zeit den gewünschten Wert nicht an, so wird der Sequenzwechsel ‚Timeout erreicht' stattfinden.

Sequenzwechsel

Name	Bedingung	Anmerkung
Timeout erreicht	Innerhalb der Timeout-Zeit hat das Signal nicht den gewünschten Wert angenommen	

3.4.27 Auf Feldbuseingangssignal warten

Eine Feldbuseingangsvariable fortlaufend lesen und warten, bis diese mit dem erwarteten Wert übereinstimmt. Feldbuseingangsvariablen sind Mikado-Variablen, die per Feldbus synchronisiert werden (siehe *Synchronisieren von SPS-Variablen und Mikado-Variablen*). Diese Variablen sind bei aktiver Synchronisierung anhand des  Symbols erkennbar.

Variablen

Name	Typ	Beschreibung
Feldbus Variable	Integer, Bool	Als Feldbus-Eingang (Profinet, EtherNet/IP, etc.) synchronisierte Mikado-Variable
Erwarteter Wert	Integer, Bool	Der gewünschte Wert, den die Feldbus Variable annehmen soll
Timeout [ms]	Integer	Maximale Wartezeit in Millisekunden (optional). Nimmt die Feldbus Variable innerhalb dieser Zeit nicht den erwarteten Wert an, wird der optionale Sequenzwechsel ‚Timeout erreicht' ausgeführt oder der Task geht in den Fehlerzustand.

Hinweis: Feldbus Variable und erwarteter Wert müssen vom selben Variablentyp sein.

Warnung: Es wird dringend empfohlen, einen Timeout-Wert anzugeben. Andernfalls besteht die Gefahr, dass der Task die weitere Ausführung des Programms auf unbestimmte Zeit blockiert.

Sequenzwechsel

Name	Bedingung	Anmerkung
Timeout erreicht	Innerhalb der Timeout-Zeit hat die Feldbus Variable nicht den gewünschten Wert angenommen	Optional

3.4.28 Feldbusausgangssignal asynchron setzen

Dieser Task setzt eine Feldbusausgangsvariable auf einen vorgegebenen Zielwert, sobald der Roboter eine bestimmte Region (ROI) verlässt. Feldbusausgangsvariablen sind Mikado-Variablen, die per Feldbus synchronisiert werden (siehe *Synchronisieren von SPS-Variablen und Mikado-Variablen*). Diese Variablen sind bei aktiver Synchronisierung anhand des ▫ Symbols erkennbar.

Der Task löst einen Vorlaufstopp aus, um fehlerhafter Benutzung vorzubeugen, und startet dann die Überwachung der angegebenen Region. Das heißt, er sollte aufgerufen werden, wenn sich der Roboter bereits innerhalb der ROI befindet, aber bevor die Fahrt zum Verlassen der ROI gestartet wird.

Befindet sich der Roboter zum Zeitpunkt des Aufrufs dieses Tasks bereits außerhalb der angegebenen ROI, wird der Feldbusausgang sofort auf den Zielwert gesetzt.

Variablen

Name	Typ	Beschreibung
ROI	ROI	Das Feldbusausgangssignal wird bei Verlassen dieser Region gesetzt
Feldbus Variable	Bool, Integer, Double, Pose	Als Feldbus-Ausgang (Profinet, EtherNet/IP, etc.) synchronisierte Mikado-Variable
Zielwert	Bool, Integer, Double, Pose	Der Wert, den die Feldbus Variable annehmen soll

Hinweis: Feldbus Variable und Zielwert müssen vom selben Typ sein.

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.5 Bildverarbeitung

3.5.1 Setze aktive Kamera

Wechsle die Kamera, die für Aufnahmen und Ähnliches verwendet wird.

Tasks, die Aufgaben mit Hilfe einer Kamera erledigen, nutzen die Kamera, die zum Zeitpunkt der Ausführung des Tasks aktiv ist. Falls Ihr Setup mehrere Kameras beinhaltet, können Sie hiermit zwischen Ihren Kameras wechseln. Ist lediglich eine Kamera Teil Ihres Setups, so ist diese automatisch die aktive Kamera.

Beispiel: Sie haben für eine Picking-Anwendung zwei Aufnahmebereiche mit je einer Kamera. Sobald Sie einen Bereich geleert haben, wechseln Sie zum anderen Aufnahmebereich. Um dies zu realisieren, können Sie in Ihrem Mikado-Programm eine Wechsel-Sequenz definieren, die unter anderem die aktive Kamera setzt.

Variablen

Name	Typ	Beschreibung
Kamera	Kamera, Integer	Name oder ID der Kamera, die nun als aktive Kamera gekennzeichnet werden soll. Hinweis: Kameranamen der verfügbaren Kameras sind unten in der *Statusleiste* ersichtlich.

3.5.2 Kamera Konfiguration laden

Lade Kamera-Konfigurationsdatei und ersetze damit die aktive Konfiguration. Die neue Konfiguration wird auf die momentan aktive Kamera angewendet.

Der angegebene Dateiname wird relativ zum Verzeichnis **config** im Projektordner interpretiert.

Hinweis: Die Konfigurationsdatei kann mithilfe der *Ensenso NxView Applikation erstellt werden*.

Variablen

Name	Typ	Beschreibung
Dateiname	String	Dateiname der Konfigurationsdatei für Ihre Kamera. Die Datei muss sich im Unterverzeichnis ‚config' des Projektverzeichnisses befinden. *Beispiel: ‚123456.ensparam'*

3.5.3 Kamera Kalibrierung laden

Läde eine *Hand-Auge-Kalibrierung* für die aktuelle Kamera und ersetzt damit die aktive.

Der angegebene Pfad wird relativ zum Verzeichnis **config** des Projekts ausgewertet.

Die *Hand-Auge-Kalibrierung* kann entweder im Kamera-Menü oder mit dem Task *Starte Kalibrierung* durchgeführt werden.

Variablen

Name	Typ	Beschreibung
Dateiname	String	Name der Datei mit der HAK für die Kamera. Die Datei muss sich im Unterverzeichnis ‚config' des Projektverzeichnisses befinden. *Beispiel: ‚last_calibration_Ensenso_Camera_0.json'*

3.5.4 Punktwolke aufnehmen

Nimmt mit der momentan aktiven Kamera eine Punktwolke auf. Die Aufnahme kann darauf für die Objektsuche verwendet werden.

Durch diesen Tasks löst die Kamera lediglich die Aufnahme(n) aus. Die Punktwolke wird in der Visualisierung noch nicht sichtbar. Sobald eine Objektsuche (oder Griffsuche) durchgeführt wurde, werden Punktwolke und gefundene Objekte in die Visualisierung eingefügt.

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.5.5 Punktwolke asynchron aufnehmen

Nimmt mit der momentan aktiven Kamera eine Punktwolke auf. Fixe Kameras können somit im Hintergrund ausgelöst werden.

Durch diesen Tasks löst die Kamera lediglich die Aufnahme(n) aus. Die Aufnahme wird im Gegensatz zum Task *Punktwolke aufnehmen* parallel zum Ablauf ausgeführt. Somit können z.B. Roboterbewegungen während der Aufnahme stattfinden. Werden die Daten der Kamera in einem nachfolgenden Task benötigt, so wird auf Beendigung dieses Tasks gewartet.

Die Punktwolke wird in der Visualisierung noch nicht sichtbar. Sobald eine Objektsuche (oder Griffsuche) durchgeführt wurde, werden Punktwolke und gefundene Objekte in die Visualisierung eingefügt.

Hinweis: Es ist nicht zulässig, im Ablauf Punktwolken aufzunehmen während eine Aufnahme noch nicht abgeschlossen ist. Eine solche Situation führt zu einem Programmfehler. Achten Sie also bei der Verwendung von asynchronen Aufnahmen darauf, dass die nachfolgende Aufnahme frühestens nach Beendigung der vorhergehenden gestartet wird.

Hinweis: Wird ein Ablauf während einer asynchronen Aufnahme gestoppt, wird die Aufnahme trotzdem abgeschlossen. Setzen Sie den Ablauf frühestens nach Beendigung dieser Aufnahme wieder fort.

Variablen

Name	Typ	Beschreibung
Verzögerung	Integer	Wartezeit in [ms] bevor die Aufnahme auslösen soll. Anwendungsbeispiel: Bevor die Aufnahme auslöst, soll der Roboter aus dem Sichtbereich einer fixen Kamera bewegt werden.

3.5.6 Mehrfachaufnahme

Erlaubt die Aufnahme und Kombination mehrerer Punktwolken. Jeder weitere Aufruf dieses Tasks ergänzt die gesammelte Punktwolke um eine neue Aufnahme. Sobald der Task *Punktwolke aufnehmen* aufgerufen wird, wird die Serie abgeschlossen und die kombinierte Punktwolke erzeugt.

Nutzen Sie diesen Task um mehrere Aufnahmen zu kombinieren (z.B. aus verschiedenen Aufnahmeperspektiven). Bis auf die letzte Aufnahme muss jede Teil-Aufnahme mit diesem Task *Mehrfachaufnahme* erfolgen. Sobald Sie den Task *Punktwolke aufnehmen* (bzw. *Punktwolke asynchron aufnehmen*) verwenden, wird diese Aufnahme mit allen vorherigen Mehrfachaufnahmen zusammengeführt und stehen für den nächsten Schritt zur Verfügung.

Hinweis: Das Objekterkennungs-Verfahren Partfinder berücksichtigt nur die letzte Aufnahme, unabhängig von zuvor ausgeführten Mehrfachaufnahmen.

Hinweis: Mehrfachaufnahmen kombinieren alle entstehenden Punktwolken und erhöhen dadurch die Anzahl der Punkte stark. Insbesondere überlappende Bereiche der Aufnahmen weisen in diesem Fall ein Vielfaches der einfachen Punkteanzahl auf. Beachten Sie, dass dies z.B. die Objektsuche zwar verbessern, aber auch stark verzögern kann.

3.5.7 Kamerapose nachjustieren

Sucht die Referenzobjekte und berechnet den Kamera-Offset.

Es werden mindestens 3 Referenzobjekte benötigt, empfohlen wird eine Anzahl von 4 oder mehr Objekten.

Die Orientierung eines gefundenen Referenzobjekts fließt in die Berechnung nicht ein. Es wird nur die Objektposition für die Berechnung verwendet. Diese ist abhängig vom Produktkoordinatensystem. Aus diesem Grund eignen sich Kugeln sehr gut als Referenzobjekt (mit dem Ursprung des Produktkoordinatensystem im Mittelpunkt der Kugel).

Hinweis: Sind Symmetrien im Referenzobjekt vorhanden, so darf die Mehrdeutigkeit keinen Einfluss auf die Produktposition haben. Im Normalfall sollte es ausreichen, den Produktmittelpunkt als Ursprung des Produktkoordinatensystems zu verwenden.

Variablen

Name	Typ	Beschreibung
Referenzobjekt-Name	String	Produktname des zu suchenden Referenzobjekts (muss als Produkt existieren)
Referenzposen	PoseList	Liste von Positionen an denen die Referenzobjekte erwartet werden. Diese können z.B. mithilfe des Tasks *Finde Objekt* ermittelt und eingelernt werden.
ROI-Namen	StringList	Namen der ROI-Bereiche, die für die einzelnen Referenzobjekte verwendet werden sollen. Die Reihenfolge der Einträge muss zu den Referenzposen passen.
Max Std.-Abw. in der Relativposition [mm]	Double	Maximal erlaubte Standard-Abweichung in der Relativposition in Millimeter. Wird dieses Limit überschritten, so findet der Sequenzwechsel ‚Std.Abw. > Maximum' (s.u.) statt.
Gemessene Posen	PoseList	Die durch die Objekterkennung ermittelten Posen der Referenzobjekte
Offset	Pose	Das Ergebnis wird in dieser Variable abgelegt. Der Offset stellt die Abweichung dar, die zwischen den gemessenen Referenzobjekten und der gegebenen Referenzposen errechnet wurde (keine Orientierung).

Sequenzwechsel

Name	Bedingung	Anmerkung
Referenzobj. nicht gefunden	Einer der Referenzobjekte konnte innerhalb der angegebenen ROI nicht gefunden werden	
Std.Abw. > Maximum	Standardabweichung > Max Std.-Abw. in der Relativposition [mm]	Tritt ein, wenn sich z.B. die Kamera stark bewegt hat. Wird hier keine Sequenz angegeben, wird die Ausführung der Tasks auch bei Überschreitung des Maximums normal fortgesetzt.

3.5.8 Anzahl der Punkte ermitteln

Anzahl der Punkte der letzten Aufnahme ermitteln.

Hinweis: Um Punkte einer bestimmten Region zu zählen, vorher den ‚*Setze aktive ROI*' Task benutzen. Der ‚*Logische Operation*' Task kann z.B. für die Auswertung verwendet werden.

Dieser Task kann z.B. dazu verwendet werden, zu überprüfen ob in einem bestimmten Bereich unerwartet Objekte oder Hindernisse aufgetaucht sind. Als Hilfsmittel für die Überprüfung wird der Z-Wert des am höchsten und niedrigsten gefundenen Punktes ausgegeben. Dies kann z. B beim Zurücklegen eines Teils in die Kiste von Vorteil sein, sodass das Teil nicht aus einer unüberprüften Höhe fallengelassen wird. Oder um zu Überprüfen, ob sich ein Gegenstand/Hindernis in einem bestimmten Bereich befindet.

Variablen

Name	Typ	Beschreibung
Minimale Anzahl	Integer	Eingabevariable: Mindestanzahl an Punkten pro Schicht um als höchster Z-Wert zu gelten. (Default: 1, Minimum: 1)
Anzahl	Integer, Double	Ausgabevariable: Anzahl der 3D-Punkte in der aktiven ROI der aktiven Kamera wird in diese Variable geschrieben
Höchstes Z	Double	Ausgabevariable: Gibt den Wert des Punktes mit dem höchsten Z-Wert an. Eine Region of Interest (ROI) wird dabei berücksichtigt.
Niedrigstes Z	Double	Ausgabevariable: Gibt den Wert des Punktes mit dem kleinsten Z-Wert an. Eine Region of Interest (ROI) wird dabei berücksichtigt.

Detailbeschreibung

Durch Reflexionen kann es passieren, dass der höchste Punkt in der Punktwolke stark vom tatsächlichen höchsten Punkt abweicht. Vereinzelte Punktewolkenfetzen, die sich im Suchbereich oberhalb der Produkte befinden, können zu unerwarteten Ergebnissen führen. Sollten Probleme mit einer verrauschten Punktwolke auftreten, so kann über das Setzen der Eingabevariable *Minimale Anzahl* die Berechnung des höchsten Punktes geändert werden. Ist der Wert dieser Eingabevariable größer als 1, wird ein Histogramm aller Z-Werte berechnet. Der Suchbereich wird dabei in 0.5 mm starke Schichten eingeteilt. Jeder Punkt innerhalb des Suchbereichs wird einer Schicht zugewiesen. Der Rückgabewert für den höchsten Z-Wert ist die obere Kante der ersten Schicht von oben die mindestens die *Minimale Anzahl* an Punkten zählt. Analog dazu wird der niedrigste Z-Wert bestimmt. Folgende Abbildung skizziert das Prinzip der Berechnung des höchsten Punkts.

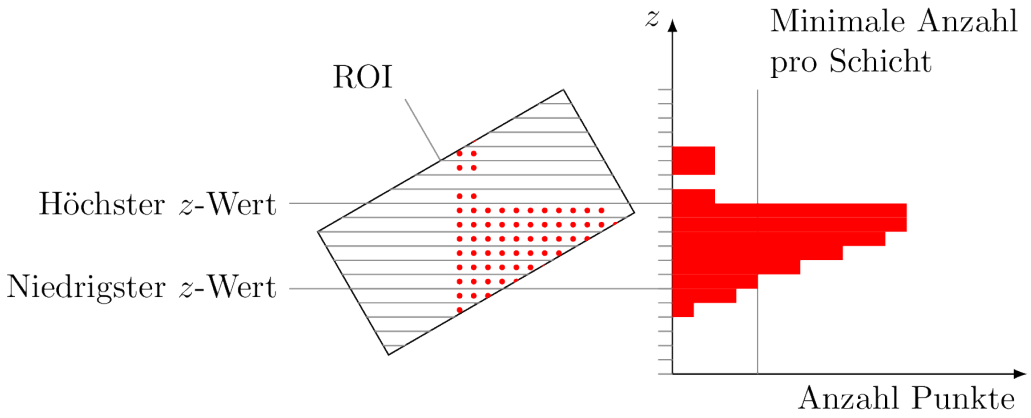


Abb. 6: Prinzipskizze des Tasks *Anzahl der Punkte ermitteln*

Warnung: Bedingt durch die Berechnungsweise bei gesetzter Eingabevariable *Minimale Anzahl* kann der Rückgabewert für den höchsten bzw. niedrigsten Punkt eine Ungenauigkeit von bis zu 0.5 mm besitzen.

Bemerkung: Sollte keine ROI aktiv sein, hat *Minimale Anzahl* keinen Effekt.

3.5.9 Bild speichern

Speichert den letzten Rohbilder-Datensatz, der als Ensenso-Filekamera verwendet werden kann. Der Vorgang zum Erstellen einer neuen Filekamera ist im Ensenso Kapitel beschrieben unter *Filekamera*. Typische Anwendungsfälle sind Situationen, wo kein Griff oder Objekt gefunden werden kann. Mit Hilfe einer Filekamera kann dann später dieses Szenario angeschaut und verbessert werden.

Variablen

Name	Typ	Beschreibung
Pfad	String, Produkt, Kiste	Verzeichnispfad, in dem die Bilder abgelegt werden sollen (z.B. `data/filecam1`). Relativ zu `~/mikado/filecams`.
Präfix	String, Produkt, Kiste	Präfix der Dateinamen für die Bilder und Konfigurationsdateien (z.B. `capture_before_pick`)

Hinweis: Der Task benötigt viel Zeit, da die Daten zeitsynchron auf die Festplatte geschrieben werden. Deshalb sollte er nur sparsam eingesetzt werden, z.B. um nach Fehlerfällen die letzte Punktwolke zu speichern.

Hinweis: Es kann auch direkt ein Produkt oder eine Kiste angegeben werden, um Bilder automatisch sortenrein in einem Unterordner mit dem Namen des Produktes oder der Kiste abzulegen.

Hinweis: Es sind nur Pfade innerhalb von *~/mikado/filecams* erlaubt. Sofern das Paket `mikado-log-purger` installiert ist, wird dieses Verzeichnis automatisch auf eine Größe von 30 GB begrenzt, um ein Volllaufen der Festplatte zu verhindern. Wird diese Grenze überschritten, werden die ältesten Aufnahmen innerhalb des Verzeichnisses zugunsten neuer Aufnahmen gelöscht.

Warnung: Ab EnsensoSDK Version 4.x hat das Argument `Präfix` keine Wirkung mehr. Es wird ignoriert und hat keinen Einfluss auf die Dateinamen.

3.5.10 Setze aktive ROI

Region of Interest (ROI) der aktuellen Kamera setzen. Die ROI wird über ein Szenenobjekt vom Typ ‚ROI' definiert, das auf der *‚Szene'-Seite* eingerichtet werden kann.

Variablen

Name	Typ	Beschreibung
ROI	ROI, Integer	Name des ROI-Objekts oder ID des ROI-Objekts, das ab sofort verwendet werden soll. Die Verwaltung der ROI-Objekte findet im Bereich *Szenenverwaltung* statt.

3.5.11 Patternpose bestimmen

Nimmt ein Bild auf und sucht ein Kalibrierpattern darin. Achtung: Vorher den Projektor abschalten.

Die Ergebnis-Pose bezieht sich auf den Mittelpunkt der Oberseite der Platte. Die Z-Achse steht dabei senkrecht auf der Oberfläche.

Hinweis: Dieser Task kann z.B. verwendet werden, um die Lage eines Teils am Roboterarm (z.B. Greiferspitze) zu prüfen.

- Patternpose mithilfe der Kamera auslesen
- Roboter-TCP an eine Position relativ zur Patternpose bewegen
- Abstand nachmessen und mit der verwendeten Soll-Pose vergleichen

---

Variablen

Name	Typ	Beschreibung
Patternpose	Pose	Ausgabevariable: Pose des Kalibrierpatterns.

Sequenzwechsel

Name	Bedingung	Anmerkung
Pattern nicht gefunden	Pattern konnte im Kamerabild nicht gefunden werden	Falls kein Pattern in der Kameraaufnahme gefunden wurde, so kann die Pose nicht ausgelesen werden. Deshalb muss in diesem Fall eine Fehlerbehandlung stattfinden oder das Programm wird mit einem Fehler beendet

---

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.6 Kalibrierung

3.6.1 Starte Kalibrierung

Dieser Task startet die *Hand-Auge-Kalibrierung*. Analog zum *manuellen Modus* wird der Roboter nach jeder Aufnahme nicht automatisch zu einer neuen Pose bewegt.

Die Bewegung zu neuen Posen kann durch das Mikado Programm definiert werden, z.B. durch Verwendung einer Liste geeigneter Posen welche den Arbeitsraum und viele mögliche Orientierungen des Kalibriermusters zur Kamera abdecken.

Mit Hilfe des Tasks *Aufnahme für Kalibrierung* können nun Aufnahmen des Kalibriermusters aufgezeichnet werden. Mit dem Task *Kalibrierung abschließen* kann der Vorgang abgeschlossen werden sobald genügend geeignete Aufnahmen vorliegen.

Falls bereits eine Kalibrierung gestartet und noch nicht beendet wurde, wird die vorherige Kalibrierung abgebrochen.

Hinweis: Der Erfolg der Hand-Auge-Kalibrierung ist stark abhängig von der Vielfalt der verwendeten Posen. Je besser das Blickfeld abgedeckt und verschiedene Orientierungen des Musters erfasst werden, desto genauer das Ergebnis.

3.6.2 Aufnahme für Kalibrierung

Dieser Task fügt dem aktuellen Kalibriervorgang (*Hand-Auge-Kalibrierung*) eine neue Aufnahme hinzu. Die Kalibrierung wurde in der Regel zuvor durch den Task *Starte Kalibrierung* gestartet.

Die mit diesem Task durchgeführten Aufnahmen sollten das Kalibriermuster in möglichst unterschiedlichen Posen aufzeichnen, um die Genauigkeit der Kalibrierung zu erhöhen.

Fehlerfälle

- Der Task schlägt fehl wenn es keinen aktiven Kalibriervorgang gibt.
- Der Task schlägt fehl und kann erneut angestoßen werden, wenn ein Kalibriervorgang gestartet wurde, die Aufnahme aber kein gültiges Kalibriermuster enthält und keine Sequenz für das Ergebnis „Nicht sichtbar“ definiert wurde.

Sequenzwechsel

Name	Bedingung	Anmerkung
Nicht sichtbar	Das Kalibriermuster ist nicht sichtbar oder zu stark verdeckt	Das Programm sollte eine andere Pose ansteuern und den Task erneut ausführen

3.6.3 Kalibrierung abschließen

Dieser Task beendet die *Hand-Auge-Kalibrierung*. Die Kalibrierung wurde in der Regel zuvor durch den Task *Starte Kalibrierung* gestartet.

Bevor dieser Task ausgeführt wird sollten mit Hilfe des Tasks *Aufnahme für Kalibrierung* Aufnahmen des Kalibriermusters in verschiedenen Posen aufgezeichnet werden.

Der Task schlägt fehl wenn es keinen aktiven Kalibriervorgang gibt oder keine gültigen Aufnahmen des Kalibriermusters aufgezeichnet wurden.

Bei erfolgreichem Abschluss der Kalibrierung wird das Ergebnis z.B. in der Datei **config/last_calibration_Ensenso_Camera_0.json** und die Auswertung der Kalibriergenauigkeit in der Datei **config/last_hand_eye_calibration_errors.json** im Zellordner gespeichert.

Hinweis: Der Erfolg der Hand-Auge-Kalibrierung ist stark abhängig von der Vielfältigkeit der verwendeten Posen. Je besser das Blickfeld abgedeckt und verschiedene Orientierungen des Musters erfasst werden, desto genauer das Ergebnis.

3.6.4 Hand-Auge-Kalibrierung validieren

Dieser Task überprüft die Qualität der Hand-Auge-Kalibrierung der aktiven Kamera. Hierfür werden folgende Schritte durchgeführt:

- Roboter zur vorgegebenen Validierungs-Pose bewegen.
- Warten bis der Roboter die Pose erreicht hat.
- Punktwolke aufnehmen.
- Validierungs-Objekt in Punktwolke suchen.
- Pose des Validierungs-Objekt aus dem URDF auslesen.
- Abweichung von Modell-Pose und der in Punktwolke erkannten Pose ermitteln. Es wird der Distanzfehler e_{Trans} und der Winkelfehler e_{Rot} bestimmt.
- Ermittelte Fehler mit der Genauigkeit der zuletzt durchgeführten Hand-Auge-Kalibrierung vergleichen.

Hinweis: Der Task kann auch verwendet werden um die TCP-Kalibrierung zu überprüfen indem als Validierungs-Objekt ein Teil des Greifers oder die gesamte Greifergruppe angegeben wird.

Warnung: Der Erfolg der Validierung ist stark abhängig von der Qualität der Objekterkennung des verwendeten Validierungs-Objekt bzw. des entsprechenden Produkts!

Variablen

Name	Typ	Beschreibung
Validierungs-Objekt	String	Name des Validierungs-Objekts. Es muss ein Produkt mit diesem Namen eingerichtet sein (Produkte werden in der *Produktverwaltung* definiert) **und** es muss ein URDF-Objekt mit diesem Namen existieren (URDF-Objekte werden in der *Zellgeometrie* definiert). Für ein fixes Kamera-Setup ist bei den meisten Robotern Link 4 des Roboterarms gut sichtbar und somit als Validierungs-Objekt geeignet.
Validierungs-Pose	Pose	Für die Validierung eingenommene Roboter-Flansch Pose
Erlaubte Abweichung	Double	Skalierung f des Grenzwerts für erfolgreiche Validierung. (Default ist 1 für 100% des mittleren Fehlers der letzten HAK)

Sequenzwechsel

Name	Bedingung	Anmerkung
Validierung fehlgeschlagen	Unerwarteter Fehler aufgetreten	Bspw. kein URDF-Objekt / Produkt eingerichtet, Produkt nicht in Punktwolke gefunden, …
Grenzwert überschritten	$e_{\mathrm{Trans}} > f \cdot \bar{e}_{\mathrm{Trans}}$ oder $e_{\mathrm{Rot}} > f \cdot \bar{e}_{\mathrm{Rot}}$	Ermittelter Distanz- und/oder Winkelfehler ist größer als während der letzten HAK. Eine detaillierte Beschreibung zur Berechnung der Fehler ist hier zu finden *Berechnung der Abweichung*

Detailbeschreibung

Die Kollisionsüberwachung sowie die Griffplanung erfolgt in Mikado stets in Weltkoordinaten, d.h. die Punktwolke sowie die Roboterpose werden in das globale Koordinatensystem transformiert. Um dies tun zu können benötigt Mikado zum einen eine sehr genaue Information über die Gelenkstellung des Roboters und zum anderen eine sehr genaue Information über die Montageposition der Kamera. Der Prozess der Hand-Auge-Kalibrierung bestimmt bei einer fixen Kamera die Transformation zwischen Kamera und Roboterfuß. Bei einer mobilen Kamera wird die Transformation zwischen Roboterflansch und Kamera durch die HAK bestimmt. Weicht die in der Punktwolke sichtbare Roboterpose von der visualisierten Roboterpose ab, so kann dies zwei Gründe haben:

- die Roboter-Kalibrierung ist fehlerhaft: die vom Roboter-Controller erhaltenen Gelenkwinkel entsprechen nicht den tatsächlichen Gelenkwinkeln oder
- die durch die HAK ermittelte Montageposition der Kamera entspricht nicht der tatsächlichen Montageposition.

Folgende Abbildung skizziert eine Übersicht zu den verschiedenen Transformationsketten. Die blau gekennzeichneten Transformationen werden durch die Hand-Auge-Kalibrierung (*Hand-Auge-Kalibrierung*) bestimmt, jeweils für die beiden Fälle fixe oder mobile Kamera. Die schwarz gekennzeichneten Transformationen werden durch die vom Roboter-Controller erhaltenen Gelenkwinkel bestimmt. Die orange gekennzeichnete Transformation vom Roboterflansch zum Greifer (*TCP kalibrieren*) wird durch die *Zellgeometrie* bestimmt und kann nur im Assistant modifiziert werden.

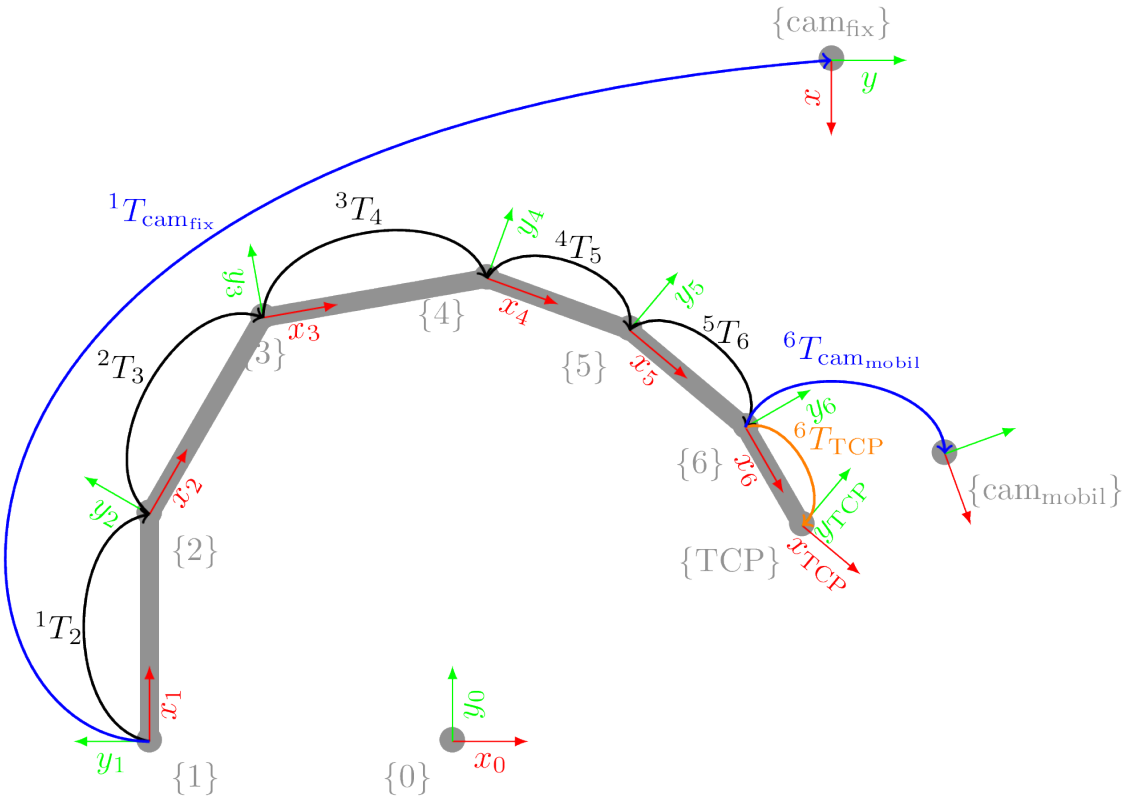


Abb. 7: Übersicht der relevanten Transformationen

Um die Qualität der Hand-Auge-Kalibrierung zu überprüfen wird die Punktwolke mit dem Mikado-internen Modell der Zelle verglichen. Dafür wird ein Validierungs-Objekt, das gleichzeitig integraler Bestandteil der Zelle ist, in der Punktwolke gesucht und überprüft, ob die erkannte Pose mit der Modell-Pose übereinstimmt. Folgende Abbildung zeigt eine erfolgreiche Validierung der Hand-Auge-Kalibrierung mit *link_4* als Validierungs-Objekt. Das Robotermodell (hier in blau mit halber Deckkraft dargestellt) deckt sich mit der Punktwolke (durch die farbigen Punkte dargestellt). Um die Übereinstimmung quantifizieren zu können wird der im Validierungs-Produkt spezifizierte Erkennungsalgorithmus verwendet, um die exakte Pose von *link_4* in der Punktwolke zu bestimmen (das

erkannte Produkt ist in der Abbildung weiß hervorgehoben). Diese Pose wird dann mit der Modell-Pose von *link_4* verglichen.

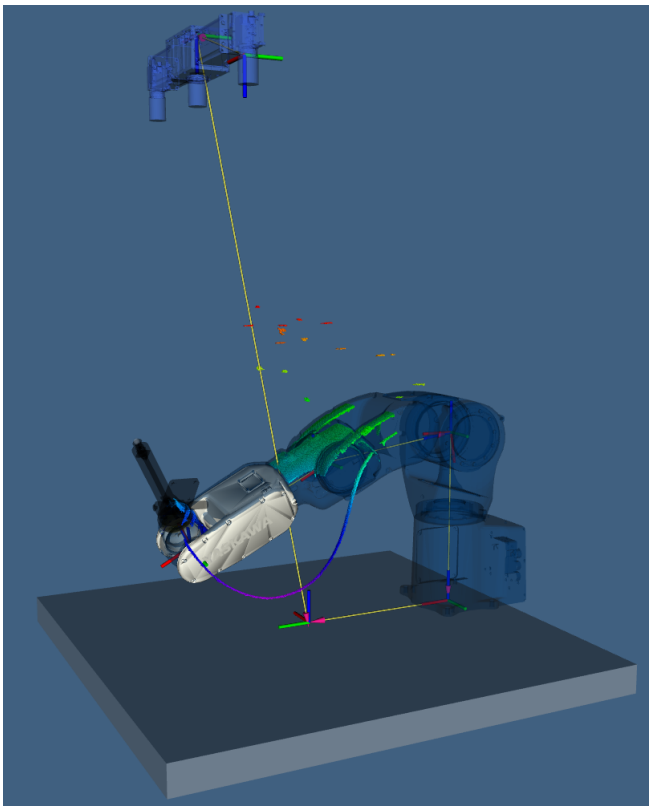
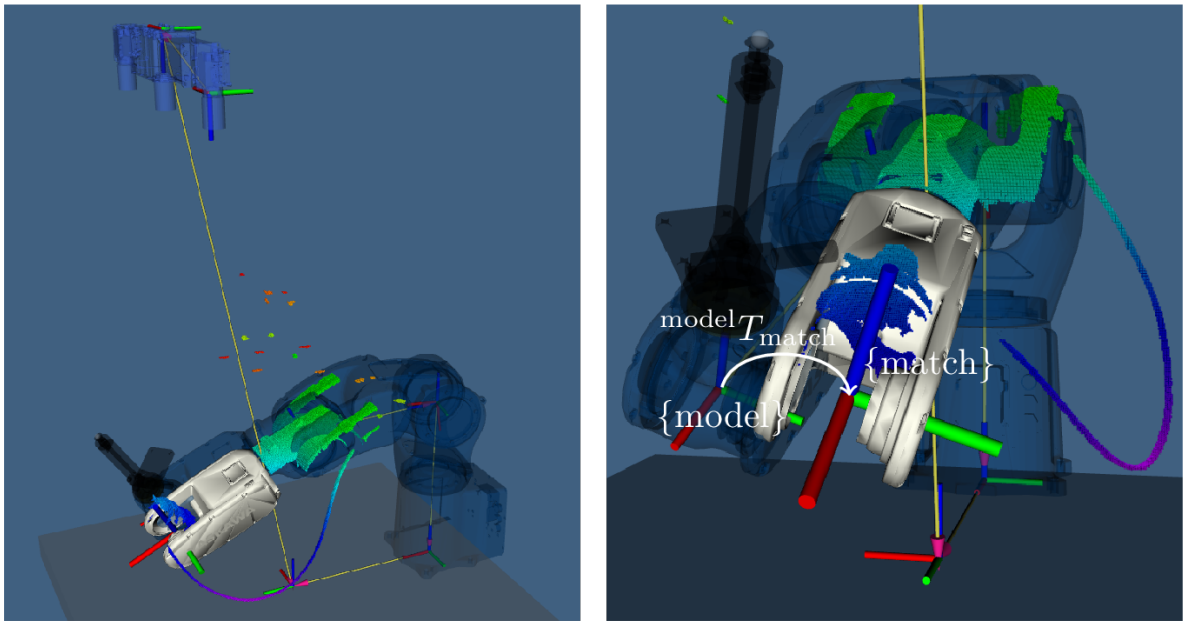


Abb. 8: Erfolgreiche Validierung der Hand-Auge-Kalibrierung

Berechnung der Abweichung

Sollte die Hand-Auge-Kalibrierung durch eine veränderte Montageposition der Kamera (beispielsweise verursacht durch eine Kollision) ungültig geworden sein, lässt sich dies mit Hilfe dieses Task schnell erkennen. Folgende Abbildung zeigt eine fehlgeschlagene Validierung. Die Modell-Pose (wieder in blau mit halber Deckkraft dargestellt) weicht stark von der Punktwolke ab. Entsprechend stark ist die Abweichung zwischen Modell-Pose und erkannter Pose des Validierungs-Objekts.



(a) Keine Übereinstimmung von Punktwolke und Modell

(b) Detailansicht

Abb. 9: Fehlgeschlagene Validierung der Hand-Auge-Kalibrierung

Hierin bezeichnet ${}^{\text{model}}T_{\text{match}}$ die Transformation zwischen Modell-Pose und der in der Punktwolke erkannten Pose, welche im Folgenden der Einfachheit halber T genannt wird. Der Distanzfehler e_{Trans} ergibt sich nun aus der euklidischen Norm des Translation-Anteils t der Transformation T:

$$e_{\text{Trans}} = |t|_2 \ ,$$

wobei sich t aus der Transformationsmatrix ergibt:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} .$$

Jede 3-dimensionale Rotation kann durch eine Drehachse und einen Drehwinkel θ um diese Achse dargestellt werden. Für die Ermittlung des Winkelfehlers wird diese Eigenschaft ausgenutzt und der Rotationsanteil R wird zunächst in eben diese axis-angle Darstellung umgewandelt. Der Winkelfehler ist als Betrag des Drehwinkels definiert:

$$e_{\text{Rot}} = |\theta| \ .$$

Zu beachten ist, dass der Winkelfehler nur Auskunft über die Verdrehung um die Drehachse gibt, jedoch keine Information über die Orientierung dieser Achse.

3.7 Szenenobjekt

3.7.1 Setze aktive Kiste

Falls mehr als eine Kiste existiert, kann hiermit die aktive gesetzt werden.

Im Abschnitt *Szenenverwaltung* wird erläutert, wie Szenenobjekte wie z.B. Kisten oder Behälter angelegt und verwaltet werden können.

Ist nur ein Szenenobjekt des Typs *Kiste* vorhanden, so ist diese automatisch die aktive Kiste.

In diesem Fall ist es nicht nötig, diesen Task in Ihren Programmablauf einzubauen.

Die aktive Kiste wird z.B. bei der Griffsuche verwendet (s. *Griffsuche starten*), solange im jeweiligen Task keine andere Kiste explizit gewählt wird.

Variablen

Name	Typ	Beschreibung
Kiste	Kiste, Integer	Name der Kiste oder ID der Kiste, die nun aktiv sein soll

3.7.2 Suche Szenenobjekt

Sucht ein Szenenobjekt mit der Methode, die für das Objekt auf der Szenen-Einrichtungsseite konfiguriert ist.

Beachten Sie, dass für eine Suche zuerst mithilfe der Kamera eine Bildaufnahme stattfindet und die 2D-Bildsuche ausgelöst wird. Weitere Informationen zu diesem Suchverfahren und wie diese eingerichtet werden kann siehe Abschnitt *Szenenverwaltung* / *Hindernis*.

Hinweis: Die aktuelle Position eines Szenenobjekts kann mithilfe des Tasks *Szenenobjekt-Position auslesen* ausgelesen werden.

Variablen

Name	**Typ**	**Beschreibung**
Objekt	Kiste, Integer	Der Name des Szenenobjekts oder die ID des Szenenobjekts, das gesucht werden soll (optional)
Min Score	Double	Mindestscore für die Objekterkennung (optional)

Sequenzwechsel

Name	**Bedingung**	**Anmerkung**
Nicht gefunden	Das Objekt konnte in der momentane Aufnahme nicht ausreichend zuverlässig gefunden werden	Häufige Ursache, falls ein Objekt nicht gefunden wird: Objekt ist zum Zeitpunkt der Aufnahme nicht komplett im Sichtfeld der Kamera. Ist hier keine Sequenz definiert, wird der Programmablauf normal mit dem nächsten Task fortgesetzt.

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.7.3 Szenenobjekt-Position auslesen

Schreibe Position/Pose eines Szenenobjekts in Variable.

Ist die 2D-Suche des Szenenobjekts aktiviert, so entstehen für diesen Task zwei Szenarien:

1. Das Szenenobjekt wurde noch nicht lokalisiert (s. Task *Suche Szenenobjekt*):
   In diesem Fall liefert der Task die Default-Pose des Objekts als Position.
2. Das Szenenobjekt wurde bereits lokalisiert (s. *Suche Szenenobjekt*, *Finde Objekt*):
   Die aktuelle Position (und somit das Ergebnis der letzten Suche des Objekts) wird in die Ausgabevariable geschrieben.

Variablen

Name	Typ	Beschreibung
Objekt	Box, ROI, Hindernis, Integer	Der Name des Szenenobjekts oder die ID des Szenenobjekts (optional)
Position	Pose	Ausgabevariable: Die Position, bzw. Pose des Objekts wird vom Task in diese Variable geschrieben.

3.7.4 Szenenobjekt-Position setzen

Setze Position/Pose eines Szenenobjekts.

Die Position eines Szenenobjekts kann durch eine beliebige Pose ersetzt werden. So können im Ablauf beispielsweise Hindernisse an neue Positionen bewegt oder Posen von ROIs angepasst werden.

Hinweis: Mithilfe dieses Tasks kann z.B. die Position einer ROI an eine bewegliche Kiste angepasst werden. Die Lokalisierung der Kiste findet mit *Suche Szenenobjekt* statt und mittels des Tasks *Szenenobjekt-Position auslesen* wird die aktuelle Pose der Kiste ausgelesen. Die ausgelesene Pose wird anschließend als Eingabe dieses Tasks zum Setzen der ROI Position verwendet.

Variablen

Name	Typ	Beschreibung
Objekt	Kiste, ROI, Hindernis, Integer	Der Name oder die ID eines Szenenobjekts (optional)
Neue Position	Pose	Die Position, bzw. Pose, die für das Szenenobjekt verwendet werden soll.

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.7.5 Szenenobjekt-Position zurücksetzen

Die Position/Pose eines Szenenobjekts auf die ursprüngliche Position zurücksetzen.

In der Regel wird dieser Task zusammen mit dem *Szenenobjekt-Position setzen* Task verwendet, um ein zuvor im Programmablauf modifiziertes Objekt auf seine Ursprungsposition zurückzusetzen.

Variablen

Name	Typ	Beschreibung
Objekt	ROI, Kiste, Hindernis, Integer	Optional: Der Name oder die ID des Szenenobjekts, das zurückgesetzt werden soll. Wird diese Eingabe leer gelassen, so wird die aktive Kiste verwendet.

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.8 Objekt-Handhabung

3.8.1 Setze aktives Produkt

Wenn mehrere Produkte existieren, kann dieser Task genutzt werden um zwischen ihnen hin und her zu wechseln.

Einige Tasks die ein Produkt benötigen nutzen standardmäßig das aktive Produkt, sofern dem Task kein explizites Produkt mitgegeben wird (z.B. *Griffsuche starten*).

Variablen

Name	Typ	Beschreibung
Produkt	Produkt, Integer	Das aktiv zu setzende Produkt. Produktnamen werden in der *Produktverwaltung* definiert.

3.8.2 Griffsuche starten

Starte die *Greifbewegungssuche*. Diese wird im Hintergrund gestartet und läuft bis alle möglichen Griffe überprüft wurden oder der Task *Teil greifen* ausgeführt wird. Mit dem Task *Griffsuche abbrechen* kann die Suche abgebrochen werden.

Vor Ausführung dieses Tasks ist es notwendig eine Punktwolke durch den Task *Punktwolke aufnehmen* bereitzustellen. Die Griffsuche läuft als paralleler Prozess im Hintergrund, das laufende Mikado-Programm wird also nicht unterbrochen. Wird *Teil greifen* aufgerufen, bevor alle Griffe getestet wurden, verhält sich der Task je nach Zustand der Suche wie folgt:

- Ist bereits ein möglicher Griff gefunden worden, so wird die Suche abgebrochen und der mögliche Griff wird ausgeführt
- Ist noch kein möglicher Griff gefunden worden, so wartet der Task bis ein möglicher Griff gefunden oder alle Möglichkeiten berechnet wurden.

Die Griffsuche findet in folgenden Schritten statt:

1. *Kistensuche*, sofern die verwendete Kiste für eine Suche konfiguriert ist und die Variable Kistensuche nicht auf ‚false' gesetzt ist → Kiste wird in Visualisierung eingezeichnet/positioniert

2. Objektsuche (siehe *Objekterkennungs-Tool*) → Gefundene Objekte und restliche Punktwolke werden in der Visualisierung angezeigt
3. Griffsuche → Alle *Greifposen* aller gefundenen Objekte werden getestet, geplant und auf mögliche Kollisionen geprüft. Ergebnisse sind in der *Griff-Inspektion* einzusehen.

Hinweis: Experimentelle Griffe werden erst dann überprüft und berechnet, sobald alle ‚normalen' Griffe erfolglos getestet wurden.

Eingabe-Variablen

Name	Typ	Beschreibung
Produkt	Produkt | Integer	Name oder ID des Produkts, das gesucht wurde. Wird das Feld leer gelassen, so wird das aktive Produkt verwendet. Produktnamen werden in der *Produktverwaltung* definiert.
Kiste	Kiste | Integer	Name oder ID der Kiste, in der gesucht wurde. Wird das Feld leer gelassen, so wird die aktive Kiste verwendet. Kistennamen werden in der *Szenenverwaltung* definiert.
Greifer	Greifer | StringListe | Integer | IntegerListe	Namen oder IDs der Greifer der Griffsuche, für die Griffe gesucht werden sollen. Statt einer Liste kann auch ein einzelner Name oder eine einzelne ID angegeben werden. Mit dem *Griffsuchen-Statistik* kann der Greifer des bis dahin besten Griffs ausgelesen werden. Wird das Feld leer gelassen, so wird die zuletzt gestartete Griffsuche verwendet.
Kistensuche	Bool	Gibt an, ob die *Kistensuche* vor der Griff-/Objektsuche in der aktuellen Aufnahme durchgeführt werden soll (optional, Defaultwert: ‚true').
Übergangspose	Pose | JointPosition	Gibt an, welche Übergangspose für die Griffsuche verwendet werden soll (optional, Defaultwert: ‚Übergangspose der aktiven Kiste').
Griff-Suchparameter	Griffsuchparameter | Integer	Name oder ID der Griff-Suchparameter, die verwendet werden sollen. Wird das Feld leer gelassen, wird der Parametersatz verwendet, der im verwendeten Produkt hinterlegt ist.

3.8.3 Griffsuche abbrechen

Laufende Griffsuche abbrechen, z.B. um das aktive Produkt oder die verwendete Kiste zu wechseln.

Eine Griffsuche wird über den Task *Griffsuche starten* gestartet und läuft im Hintergrund parallel zur Ausführung des Mikado-Programms.

Anwendungsbeispiel: Ein Sensorsignal (z.B. eingelesen mit *Robotereingangssignal lesen*) soll das Programm dazu veranlassen, nach einem anderen Produkt zu suchen. Sie können nun die Griffsuche abbrechen, das Produkt wechseln und sofort eine Griffsuche mit dem anderen Produkt starten.

3.8.4 Finde Objekt

Gibt eine Liste aller Posen der gefundenen Produkte, ihre Erkennungs-Scores, sowie alle möglichen Griffe zurück. Die Griffe sind nicht nach Güte sortiert.

Im Gegensatz zum Task *Griffsuche starten* wird die Suche nicht im Hintergrund durchgeführt, sondern mit dem Task gestartet. Der Programmablauf wird erst fortgesetzt, sobald alle Ergebnisse berechnet sind.

Anwendungsbeispiel: Wenn lediglich Objektpositionen vermessen und über eine Schnittstelle nach außen zu externen Systemen weitergereicht werden soll.

Variablen

Name	Typ	Beschreibung
Produkt	Produkt | Integer	Name oder ID des Produkts, das in der letzten Aufnahme gesucht werden soll. Wird das Feld leer gelassen, so wird das aktive Produkt verwendet. Produktnamen werden in der *Produktverwaltung* definiert.
Beachte in Kollisionswelt	Bool	default: true. Wird der Wert auf false gesetzt, werden die gefundenen Objekte nicht in der Kollisionswelt berücksichtigt.
Griff-Suchparameter	PickSearchParamSet	Enthält die maximale Translations- und Rotationsdifferenz, die verwendet wird um zu bestimmen, ob ein Objekt an derselben Pose wiedergefunden wurde.
Objektposen	PoseList	Ausgabevariable: Liste an Posen der gefundenen Objekte. Die Posen beziehen sich auf den Ursprung des Produktkoordinatensystems.
Objekt-Scores	StringList	Ausgabevariable: Liste mit den Scores der gefundenen Objekte.
Potentielle Griffe	PoseList	Ausgabevariable: Liste aller Griff-Posen zu allen Funden, ohne Kollisionsprüfung. Die Posen beziehen sich dabei auf ‚tool0', also das Handgelenk des Roboters beim Griff.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kein Objekt gefunden	Kein Objekt gefunden	

3.8.5 Griffe auslesen

Gibt eine Liste aller Posen der gefundenen Produkte, ihre Erkennungs-Scores, sowie der validen Griff-Posen (eine pro Objekt) zurück. Die Griffe sind sortiert nach den Sortier-Kriterien, siehe *Beste Griffe*.

Die Suche muss vorher mit *Griffsuche starten* gestartet werden. Der Programmablauf wird erst fortgesetzt, sobald für jedes Objekt jeweils ein Griff gefunden wurde oder alle Griffe gescheitert sind.

Eingabe-Variablen

Name	Typ	Beschreibung
Produkt	Produkt | Integer	Name oder ID des Produkts, das gesucht wurde. Wird das Feld leer gelassen, so wird das aktive Produkt verwendet. Produktnamen werden in der *Produktverwaltung* definiert.
Kiste	Kiste | Integer	Name oder ID der Kiste, in der gesucht wurde. Wird das Feld leer gelassen, so wird die aktive Kiste verwendet. Kistennamen werden in der *Szenenverwaltung* definiert.
Greifer	Greifer | StringListe | Integer | IntegerListe	Namen oder IDs der Greifer der Griffsuche, für die Informationen ausgelesen werden sollen. Statt einer Liste kann auch ein einzelner Name oder eine einzelne ID angegeben werden. Wird das Feld leer gelassen, so wird die zuletzt gestartete Griffsuche verwendet.

Ausgabe-Variablen

Name	Typ	Beschreibung
Objektposen	PoseList	Liste an Posen der gefundenen Objekte. Die Posen beziehen sich auf den Ursprung des Produktkoordinatensystems.
Objekt-Scores	DoubleList	Liste mit den Scores der gefundenen Objekte.
Valide Griffe	PoseList	Liste der validen (kollisionsgeprüften) Griff-Posen. Pro Objekt wird jeweils nur der beste Griff zurückgegeben, wobei die Liste nach den in *Beste Griffe* beschriebenen Kriterien sortiert ist. Die Posen beziehen sich dabei auf ‚tool0', also das Handgelenk des Roboters beim Griff.
Namen der validen Griffe	StringList	Liste der Namen der validen (kollisionsgeprüften) Griffe. Siehe auch: Ausgabevariable „Valide Griffe"
Anzahl Funde	Integer	Anzahl der gefundenen Objekte
Anzahl greifbarer Funde	Integer	Anzahl der gefundenen Objekte, für die valide Griffe gefunden wurde
Anzahl greifbarer Funde (Bevorzugt)	Integer	Anzahl der gefundenen Objekte, für die valide Griffe der Greif-Kategorie „Bevorzugt" gefunden wurde
Anzahl greifbarer Funde (Standard)	Integer	Anzahl der gefundenen Objekte, für die valide Griffe der Greif-Kategorie „Standard" gefunden wurde
Anzahl greifbarer Funde (Experimentell)	Integer	Anzahl der gefundenen Objekte, für die valide Griffe der Greif-Kategorie „Experimentell" gefunden wurde
Fund-Name	String	Name des Fundes des besten Griffs
Fund-Pose	Pose	Pose des Fundes des besten Griffs
Kollisionsart	Integer	Kollisionsart des besten Griffs (erlaubte Kollisionen: 2, keine Kollisionen: 0)
Greifer-Name	Greifer | String	Greifer des besten Griffs (nützlich bei Suchen mit mehreren Greifern)
Greifer-ID	Integer	ID des Greifers des besten Griffs (nützlich bei Suchen mit mehreren Greifern)
Griff-Kategorie	String	Kategorie des besten Griffs („Bevorzugt" / „Standard" / „Experimentell")
Griff-Unterkategorie	Integer	Unterkategorie des besten Griffs (0-10)
Griff-Name	String	Name des besten Griffs
Griff-Offset	Pose	Offset des besten Griffs relativ zum zugehörigen Fund
Griff-Tcp-Pose	Pose	Tcp-Pose des Greifpunkts des besten Griffs
Griff-Flansch-Pose	Pose	Flansch-Pose des Greifpunkts des besten Griffs
Start-Tcp-Pose	Pose	Tcp-Pose des Startpunktes des besten Griffs

Fortsetzung auf der nächsten Seite

Tab. 1 – Fortsetzung der vorherigen Seite

Name	Typ	Beschreibung
Start-Flansch-Pose	Pose	Flansch-Pose des Startpunktes des besten Griffs
Anfahrt-Tcp-Pose	Pose	Tcp-Pose des Anfahrtspunktes des besten Griffs
Anfahrt-Flansch-Pose	Pose	Flansch-Pose des Anfahrtspunktes des besten Griffs
Rückfahrt-Tcp-Pose	Pose	Tcp-Pose des Rückfahrtspunktes des besten Griffs
Rückfahrt-Flansch-Pose	Pose	Flansch-Pose des Rückfahrtspunktes des besten Griffs
End-Tcp-Pose	Pose	Tcp-Pose des Endpunktes des besten Griffs
End-Flansch-Pose	Pose	Flansch-Pose des Endpunktes des besten Griffs
Gesamtkollisionstiefe	Double	Gesamtkollisionstiefe aller Schritte des besten Griffs
Maximal-Kollisionstiefe	Double	Maximal-Kollisionstiefe aller Schritte des besten Griffs
Durchschnitts-Kollisionstiefe	Double	Durchschnitts-Kollisionstiefe aller Schritte des besten Griffs

Sequenzwechsel

Name	Bedingung	Anmerkung
Keine Kiste gefunden	Keine Kiste gefunden	
Kein Objekt gefunden	Kein Objekt gefunden	
Kein Griff gefunden	Kein valider Griff gefunden	Keine greifbaren Objekte vorhanden

3.8.6 Teil greifen

Greife ein Teil aus der Kiste. Es wird auf die Ergebnisse der *Greifbewegungssuche* gewartet und dann der beste gefundene Griff ausgeführt.

Bei Bedarf blockiert dieser Task, bis ein valider Griff gefunden wurde oder eine Zeitüberschreitung bei der Griffsuche eintritt (weiteres zum Timeout-Parameter in Sektion *pick_search.yaml*).

Weitere Informationen zum Greifen von Teilen sind beim Task *Griffsuche starten* zu finden.

Hinweis: Für eine höhere Durchsatzgeschwindigkeit sollte die Punktwolke so früh wie möglich aufgenommen werden und die Greifplanung mit dem Task *Griffsuche starten* gestartet werden. Idealerweise hat die *Greifbewegungssuche* bereits einen gültigen Griff gefunden, wenn der ‚*Teil greifen*' Task erreicht wird. Nur so kann der Roboter ohne Verzögerung direkt einen Griff ausführen. Andernfalls muss er auf die Berechnung warten.

Hinweis: Um das Teil wieder abzulegen, müssen Sie den Task *Teil ablegen* nutzen!

Fahrt zur Übergangspose

Für das Verhalten des Roboters bezüglich der Fahrt zur Übergangspose gibt es zwei verschiedene Varianten, zwischen denen mittels einer optionalen Eingabevariable gewählt werden kann. Der Roboter kann entweder bei Aufruf dieses Tasks immer zur Übergangspose fahren – unabhängig davon, ob ein gültiger Griff vorhanden ist – oder nur, wenn tatsächlich ein gültiger Griff vorhanden ist.

Welche der beiden Varianten besser geeignet ist, hängt stark vom jeweiligen Anwendungsfall ab. Wenn der ‚*Teil greifen*' Task auf die Griffsuche warten muss, ist es in der Regel schneller, wenn sich der Roboter immer in die Übergangspose bewegt. Wird dann ein gültiger Griff gefunden, befindet sich der Roboter bereits in der Übergangspose direkt über der Kiste und kann den gefundenen Griff sofort ausführen.

In bestimmten Situationen kann es jedoch sinnvoll sein, dass der Roboter an seiner vorherigen Position auf einen gültigen Griff wartet und sich erst dann in die Übergangsposition bewegt. Dies ist beispielsweise der Fall, wenn

mehrere Kisten nacheinander auf Griffe überprüft werden sollen. In diesem Fall würde die direkte Fahrt zur Übergangspose dazu führen, dass der Roboter jede Kiste anfährt, auch wenn letztendlich nicht aus ihr gegriffen werden kann. Wartet der Roboter hingegen an seiner vorherigen Position, kann er von dort aus direkt zu einer Kiste mit einem gültigen Griff fahren und spart somit einige unnötige Fahrten.

Eingabe-Variablen

Name	Typ	Beschreibung
Produkt	Produkt | Integer	Name oder ID des Produkts, das gesucht wurde. Wird das Feld leer gelassen, so wird das aktive Produkt verwendet. Produktnamen werden in der *Produktverwaltung* definiert.
Kiste	Kiste | Integer	Name oder ID der Kiste, in der gesucht wurde. Wird das Feld leer gelassen, so wird die aktive Kiste verwendet. Kistennamen werden in der *Szenenverwaltung* definiert.
Fahrt zum Übergang nur bei Griff	Bool	Verhalten bezüglich der Fahrt zur Übergangspose, siehe *Fahrt zur Übergangspose* (optional, default: ‚false'). Wenn ‚false', fährt der Roboter immer zur Übergangspose. Bei ‚true' nur, wenn eine gültiger Griff vorhanden ist.

Ausgabe-Variablen

Name	Typ	Beschreibung
ist der Griff experimentell?	Bool	wird auf ‚true' gesetzt, falls der ausgeführte Griff ein experimenteller Griff war
Anzahl Funde	Integer	Anzahl der gefundenen Objekte
Anzahl greifbarer Funde	Integer	Anzahl der gefundenen Objekte, für die valide Griffe gefunden wurde
Anzahl greifbarer Funde (Bevorzugt)	Integer	Anzahl der gefundenen Objekte, für die valide Griffe der Greif-Kategorie „Bevorzugt" gefunden wurde
Anzahl greifbarer Funde (Standard)	Integer	Anzahl der gefundenen Objekte, für die valide Griffe der Greif-Kategorie „Standard" gefunden wurde
Anzahl greifbarer Funde (Experimentell)	Integer	Anzahl der gefundenen Objekte, für die valide Griffe der Greif-Kategorie „Experimentell" gefunden wurde
Fund-Name	String	Name des Fundes des verwendeten Griffs
Fund-Pose	Pose	Pose des Fundes des verwendeten Griffs
Kollisionsart	Integer	Kollisionsart des verwendeten Griffs (erlaubte Kollisionen: 2, keine Kollisionen: 0)
Greifer-Name	Greifer | String	Greifer des verwendeten Griffs (nützlich bei Suchen mit mehreren Greifern)
Greifer-ID	Integer	ID des Greifers des verwendeten Griffs (nützlich bei Suchen mit mehreren Greifern)
Griff-Kategorie	String	Kategorie des verwendeten Griffs („Bevorzugt" / „Standard" / „Experimentell")
Griff-Unterkategorie	Integer	Unterkategorie des verwendeten Griffs (0-10)
Griff-Name	String	Name des verwendeten Griffs
Griff-Offset	Pose	Offset des verwendeten Griffs relativ zum zugehörigen Fund
Griff-Tcp-Pose	Pose	Tcp-Pose des Greifpunkts des verwendeten Griffs
Griff-Flansch-Pose	Pose	Flansch-Pose des Greifpunkts des verwendeten Griffs
Start-Tcp-Pose	Pose	Tcp-Pose des Startpunktes des verwendeten Griffs
Start-Flansch-Pose	Pose	Flansch-Pose des Startpunktes des verwendeten Griffs
Anfahrt-Tcp-Pose	Pose	Tcp-Pose des Anfahrtspunktes des verwendeten Griffs
Anfahrt-Flansch-Pose	Pose	Flansch-Pose des Anfahrtspunktes des verwendeten Griffs
Rückfahrt-Tcp-Pose	Pose	Tcp-Pose des Rückfahrtspunktes des verwendeten Griffs
Rückfahrt-Flansch-Pose	Pose	Flansch-Pose des Rückfahrtspunktes des verwendeten Griffs
End-Tcp-Pose	Pose	Tcp-Pose des Endpunktes des verwendeten Griffs
End-Flansch-Pose	Pose	Flansch-Pose des Endpunktes des verwendeten Griffs
Gesamtkollisionstiefe	Double	Gesamtkollisionstiefe aller Schritte des verwendeten Griffs
Maximal-Kollisionstiefe	Double	Maximal-Kollisionstiefe aller Schritte des verwendeten Griffs
Durchschnitts-Kollisionstiefe	Double	Durchschnitts-Kollisionstiefe aller Schritte des verwendeten Griffs

Sequenzwechsel

Name	Bedingung	Anmerkung
Kiste nicht gefunden	Kiste nicht gefunden	Ist die verwendete Kiste nicht fix, sondern wird in der Punktwolke gesucht (s. *Szenenverwaltung* / *Hindernis*) so kann der Fall eintreten, dass der Fund der Kiste nicht zuverlässig genug war.
Kein Objekt gefunden	Kein Objekt gefunden	Falls in der Punktwolke kein zuverlässiger Fund ermittelt wurde.
Kein Griff gefunden	Kein Griff gefunden	Falls kein Griff gefunden wurde, der kollisionsfrei durchgeführt werden kann.
Griff gescheitert	Griff gescheitert	Falls der Griff nicht erfolgreich ausgeführt werden konnte. Beispiel: Griffsensor meldet Fehler oder eine Bewegung konnte nicht erfolgreich ausgeführt werden.

Verhalten bei erneutem Versuch

Dieser Task unterstützt die Möglichkeit, im Fehlerfall die Ausführung erneut zu versuchen.

Wurde der Griff nicht erfolgreich ausgeführt (z.B. Timeout bei Bewegung mit reduzierter Geschwindigkeit o.ä.), so sieht die erneute Ausführung des Tasks wiefolgt aus.

- Falls der Fehler auftritt, bevor ein Teil gegriffen wurde:

  Ein weiterer Versuch wird gestartet, den Griff auszuführen.

- Falls der Fehler auftritt, während oder nachdem ein Teil gegriffen wurde:
  - Der Greifer wird geöffnet und fällt somit zurück in die Kiste.
  - Eine Bewegung zurück zur Übergangspose wird geplant und ausgeführt.
  - Wurde die Bewegung zur Übergangspose erfolgreich ausgeführt, so wird der Mikado Programmablauf mit dem Sequenzwechsel ‚Griff gescheitert' fortgesetzt.

3.8.7 Griffsuchen-Statistik

Erhalte Informationen über die angegebene Griffsuche und den bisher besten gefundenen Griff in dieser Griffsuche.

Der Task stoppt nicht die Suche und legt sich auch nicht auf den Griff fest, sodass es später noch bessere Griffe geben kann.

Dieser Task ist nützlich, wenn eine Suche mit mehreren Greifern gestartet wurde. Dann kann man über diesen Task den Greifer des bislang besten Griffs auslesen und ggf. den Greifer wechseln. Im *Teil greifen* werden nur die Griffe beachtet, die zum dann aktuellen Greifer gehören.

Eingabe-Variablen

Name	Typ	Beschreibung
Produkt	Produkt | Integer	Name oder ID des Produkts, das gesucht wurde. Wird das Feld leer gelassen, so wird das aktive Produkt verwendet. Produktnamen werden in der *Produktverwaltung* definiert.
Kiste	Kiste | Integer	Name oder ID der Kiste, in der gesucht wurde. Wird das Feld leer gelassen, so wird die aktive Kiste verwendet. Kistennamen werden in der *Szenenverwaltung* definiert.
mögliche(r) Greifer	Greifer | StringListe | Integer | IntegerListe	Namen oder IDs der Greifer der Griffsuche, für die Informationen ausgelesen werden sollen. Statt einer Liste kann auch ein einzelner Name oder eine einzelne ID angegeben werden. Wird das Feld leer gelassen, so wird die zuletzt gestartete Griffsuche verwendet.

Ausgabe-Variablen

Name	Typ	Beschreibung
Anzahl Funde	Integer	Anzahl der gefundenen Objekte
Anzahl greifbarer Funde	Integer	Anzahl der gefundenen Objekte, für die valide Griffe gefunden wurde
Anzahl greifbarer Funde (Bevorzugt)	Integer	Anzahl der gefundenen Objekte, für die valide Griffe der Greif-Kategorie „Bevorzugt“ gefunden wurde
Anzahl greifbarer Funde (Standard)	Integer	Anzahl der gefundenen Objekte, für die valide Griffe der Greif-Kategorie „Standard“ gefunden wurde
Anzahl greifbarer Funde (Experimentell)	Integer	Anzahl der gefundenen Objekte, für die valide Griffe der Greif-Kategorie „Experimentell“ gefunden wurde
Fund-Name	String	Name des Fundes des aktuell besten Griffs
Fund-Pose	Pose	Pose des Fundes des aktuell besten Griffs
Kollisionsart	Integer	Kollisionsart des aktuell besten Griffs (erlaubte Kollisionen: 2, keine Kollisionen: 0)
Greifer-Name	Greifer | String	Greifer des aktuell besten Griffs (nützlich bei Suchen mit mehreren Greifern)
Greifer-ID	Integer	ID des Greifers des aktuell besten Griffs (nützlich bei Suchen mit mehreren Greifern)
Griff-Kategorie	String	Kategorie des aktuell besten Griffs („Bevorzugt“ / „Standard“ / „Experimentell“)
Griff-Unterkategorie	Integer	Unterkategorie des aktuell besten Griffs (0-10)
Griff-Name	String	Name des aktuell besten Griffs
Griff-Offset	Pose	Offset des aktuell besten Griffs relativ zum zugehörigen Fund
Griff-Tcp-Pose	Pose	Tcp-Pose des Greifpunkts des aktuell besten Griffs
Griff-Flansch-Pose	Pose	Flansch-Pose des Greifpunkts des aktuell besten Griffs
Start-Tcp-Pose	Pose	Tcp-Pose des Startpunktes des aktuell besten Griffs
Start-Flansch-Pose	Pose	Flansch-Pose des Startpunktes des aktuell besten Griffs
Anfahrt-Tcp-Pose	Pose	Tcp-Pose des Anfahrtspunktes des aktuell besten Griffs
Anfahrt-Flansch-Pose	Pose	Flansch-Pose des Anfahrtspunktes des aktuell besten Griffs
Rückfahrt-Tcp-Pose	Pose	Tcp-Pose des Rückfahrtspunktes des aktuell besten Griffs
Rückfahrt-Flansch-Pose	Pose	Flansch-Pose des Rückfahrtspunktes des aktuell besten Griffs
End-Tcp-Pose	Pose	Tcp-Pose des Endpunktes des aktuell besten Griffs
End-Flansch-Pose	Pose	Flansch-Pose des Endpunktes des aktuell besten Griffs
Gesamtkollisionstiefe	Double	Gesamtkollisionstiefe aller Schritte des aktuell besten Griffs
Maximal-Kollisionstiefe	Double	Maximal-Kollisionstiefe aller Schritte des aktuell besten Griffs
Durchschnitts-Kollisionstiefe	Double	Durchschnitts-Kollisionstiefe aller Schritte des aktuell besten Griffs

Sequenzwechsel

Name	Bedingung	Anmerkung
Keine Kiste gefunden	Keine Kiste gefunden	
Kein Objekt gefunden	Kein Objekt gefunden	
Kein Griff gefunden	Kein valider Griff gefunden	Keine greifbaren Objekte vorhanden

3.8.8 Teil ablegen

Öffnet den Greifer und entfernt das angehängte Objekt aus der Kollisionswelt.

Hinweis: Falls Sie ein Teil mit dem Task *Teil greifen* gegriffen haben, muss dieses auch mit dem Task ‚Teil ablegen' abgelegt werden. Wenn zum Loslassen eines gegriffenen Teils nur der Task *Hand öffnen/schließen* verwendet wird öffnet sich zwar der Greifer, das gegriffene Teil bleibt aber in der Kollisionswelt und wird sich weiter mit dem Roboter bewegen.

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.8.9 Lösche Funde und Punktwolke

Alle gefundenen Objekte und die Punktwolke (Octomap) aus der Kollisionswelt entfernen.

Beispielanwendung: Werden Kisten mit unterschiedlichen Produkten an derselben Position verwendet (z.B. durch manuellen Kistentausch), so müssen Sie beim Kistenwechsel die aktuellen Funde löschen. Ansonsten werden in Mikado Funde für die Kollisionsplanung verwendet, die in diesem Fall nicht mehr existieren.

Warnung: Dieser Task bricht alle laufenden Griffsuchen ab. Das heißt, bevor wieder etwas gegriffen werden kann, muss mit dem Task „*Griffsuche starten*" eine neue Suche gestartet werden. Andernfalls führt die Ausführung des „*Teil greifen*" Task zu einem Fehler.

3.8.10 Statisches Objekt hinzufügen

Fügt eine Instanz eines Produkts an einer beliebigen Position in die Welt ein. Dieses wird z.B. bei der Pfadplanung umfahren und dient zur Visualisierung. Das hinzugefügte Objekt kann anschließend an den TCP angehängt (*Statisches Objekt anhängen*) oder wieder entfernt werden (*Statisches Objekt entfernen*).

Beispiel: Ein Objekt wurde in bekannter Pose auf eine Zwischenablage abgelegt. Sobald ein Teil abgelegt wird (*Teil ablegen*), wird es von Mikado nicht mehr in der Pfadplanung berücksichtigt. Durch diesen Task können bei anschließenden Bewegungen Kollisionen des Roboters (und evtl. angehängter Teile) mit dem statischen Objekt vermieden werden.

Hinweis: Die Objekt ID muss eindeutig sein und darf noch nicht existieren, d.h. die Objekt ID darf nicht mit dem Namen eines Szeneobjekts (Kiste, Hindernis) oder eines Fundes („Produktname-NUMMER") übereinstimmen, um unerwünschtes Verhalten zu vermeiden.

Variablen

Name	Typ	Beschreibung
Produkt	Produkt, Integer	Name des Produkts das als Vorlage dient
Objekt ID	String	Bezeichner des neuen Objekts (für *Statisches Objekt entfernen*)
Objektpose	Pose	Position des erzeugten Objekts

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.8.11 Statisches Objekt entfernen

Entfernt ein zuvor durch einen Task *hinzugefügtes Objekt* aus der Szene. Es wird anschließend nicht mehr als Kollisionsobjekt berücksichtigt.

Hinweis: Ein statisches Objekt das mit dem Task *Statisches Objekt anhängen* an den TCP angehängt wurde kann nicht mit diesem Task entfernt werden. Es kann nur mit dem Task *Teil ablegen* abgelegt und gleichzeitig aus der Szene entfernt werden.

Variablen

Name	Typ	Beschreibung
Objekt ID	String	Bezeichner des zu entfernenden Objekts (aus *Statisches Objekt hinzufügen*)

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.8.12 Statisches Objekt anhängen

Hängt ein zuvor hinzugefügtes statisches Objekt (*Statisches Objekt hinzufügen*) an den TCP. Dieses wird z.B. bei der Pfadplanung und in der Visualisierung berücksichtigt und gemeinsam mit dem Roboter bewegt.

Das Objekt kann mit dem Task *Teil ablegen* wieder abgelegt werden. Es wird dann automatisch aus der Szene entfernt und steht nicht mehr als Szenenobjekt zur Verfügung.

Beispiel: Ein Objekt wurde in bekannter Pose auf eine Zwischenablage abgelegt und als statisches Objekt hinzugefügt um Kollisionen zu vermeiden. Das Objekt wird von der Zwischenablage blind gegriffen und durch diesen Task an den TCP angehängt. Dadurch können bei anschließenden Bewegungen Kollisionen des Objekts mit der Zelle vermieden werden, und Bewegungen können in Teilekoordinaten (*Bewege Teil nach*) durchgeführt werden.

Hinweis: Um ein Objekt an den Greifer anzuhängen darf nicht schon ein anderes Teil gegriffen sein. Die Objekt ID muss existieren. Falls kein statisches Objekt mit der entsprechenden Objekt ID existiert, schlägt der Task fehl.

Variablen

Name	Typ	Beschreibung
Objekt ID	String	Bezeichner des Objekts (aus *Statisches Objekt hinzufügen*)

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.8.13 Produktabmessungen auslesen

Liest die Abmessungen der Axis-aligned-Bounding-Box (AABB) des aktiven Produkts.

Hinweis: Auf die Abmessungen der AABB kann auch über die JavaScript API zugegriffen werden, siehe *getDimensions*.

Variablen

Name	Typ	Beschreibung
X	Double	Ausgabevariable (optional): Abmessung in X-Richtung des Produkt-Koordinatensystems [m]
Y	Double	Ausgabevariable (optional): Abmessung in Y-Richtung des Produkt-Koordinatensystems [m]
Z	Double	Ausgabevariable (optional): Abmessung in Z-Richtung des Produkt-Koordinatensystems [m]

Detailbeschreibung

Berechnet den kleinst möglichen umschließenden Quader für das aktive Produkt und gibt dessen Abmessungen zurück. Dabei ist die Orientierung des Quaders am Produkt-Koordinatensystem ausgerichtet (Axis-aligned-Bounding-Box AABB). In folgender Skizze ist die AABB durch einen gelb-durchsichtigen Quader angedeutet.

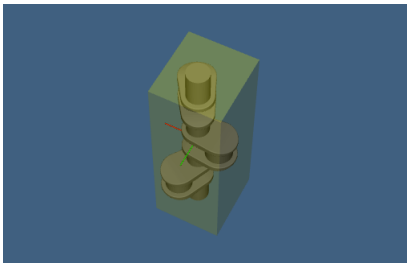


Abb. 10: Prinzipskizze des Tasks *Produktabmessungen auslesen*

3.8.14 Passe Objektpose an

Passt die Pose des angehängten (gegeriffenen) Teils an. Dieses wird in der Kollisionswelt beachtet und bei Fahrten mittels *Bewege Teil (kartesisch) nach* (*Bewege Teil nach*) zur Bahnplanung genutzt.

Beispiel: Ein Teil wird mit einem Magnetgreifer aus der Kiste entnommen und muss anschließend in eine Wendestation eingelegt werden. Beim herausziehen des Teils verschiebt dieses sich leicht, sodass es der Wendestation nicht mehr korrekt zugeführt werden kann. Nun kann das Teil unter eine Kamera gefahren werden, mittels *Finde Objekt* (*Finde Objekt*) die korrekte Pose bestimmt werden und mit *Passe Objektpose an* (*Passe Objektpose an*) nachjustiert werden. Anschließend kann das Teil Lage- und Orientierungsrichtig zugeführt werden.

Hinweis: Mithilfe der Eingabevariablen bezüglich Translations- und Rotationstoleranz kann ein Sequenzwechsel ausgeführt werden, falls sich die Pose des gefundenen Teils zu sehr von der erwarteten Pose unterscheidet.

Variablen

Name	Typ	Beschreibung
Zielpose	Pose	Pose, an welche das angehängte Teil verschoben wird
Toleranz Translation	Double	Zulässiger translatorischer Unterschied [m] zwischen der Pose des angehängten Teils und der Zielpose
Toleranz Rotation	Double	Zulässiger rotatorischer Unterschied [°] zwischen der Pose des angehängten Teils und der Zielpose

Sequenzwechsel

Name	Bedingung	Anmerkung
Toleranz überschritten	Abweichung Teilepose zu Zielpose > Toleranz	Falls die Abweichung der Translation oder Rotation der Pose des angehängten Teils und der Zielpose größer der zulässigen Toleranz ist, kann mit einer anderen Sequenz fortgefahren werden

---

Hinweis: Dieser Task löst einen Vorlaufstopp aus. D.h. sämtliche von vorherigen Tasks angesammelten Roboterbewegungen werden ausgeführt und müssen beendet sein, bevor dieser Task ausgeführt wird.

3.9 Transformationen

3.9.1 Pose auftrennen

Mit diesem Task kann eine `Pose` in ihre einzelnen Positions- und Orientierungskomponenten aufgeteilt werden. Das Ergebnis sind die xyz-Distanzen und Eulerwinkel, also Roll, Nick und Gier.

Variablen

Name	Typ	Beschreibung
Pose	Pose	–
x	Double	Distanz in Meter (opt)
y	Double	Distanz in Meter (opt)
z	Double	Distanz in Meter (opt)
Roll	Double	Winkel in Grad (opt)
Nick	Double	Winkel in Grad (opt)
Gier	Double	Winkel in Grad (opt)

Hinweis: Einzelne Ausgabewerte können weggelassen werden, falls sie nicht benötigt werden.

3.9.2 Pose erstellen

Mit diesem Task kann eine `Pose` aus einer Kombination verschiedener Eingabewerte erzeugt werden. Als Eingabewerte sind eine Eingangspose sowie einzelne Positions- und Orientierungskomponenten (x, y, z, Roll, Nick, Gier) zulässig. Alle Eingaben sind optional.

Wenn eine Eingabepose verwendet wird, dann bildet diese die Grundlage für die neue Pose. Alle zusätzlich gesetzten einzelnen Komponenten werden bei dieser Pose durch den gegebenen Wert ersetzt. Komponenten, die nicht gesetzt sind, bleiben von der Eingabepose erhalten.

Wenn keine Eingabepose verwendet wird, basiert die neue Pose auf einer Nullpose. Das bedeutet, dass alle nicht gesetzten optionalen Positions- und Orientierungskomponenten den Wert `0 m` bzw. `0°` erhalten.

Variablen

Name	Typ	Beschreibung
Eingabepose	Pose	optional
x	Double	Distanz in Meter (opt)
y	Double	Distanz in Meter (opt)
z	Double	Distanz in Meter (opt)
Roll	Double	Winkel in Grad (opt)
Nick	Double	Winkel in Grad (opt)
Gier	Double	Winkel in Grad (opt)
Ausgabepose	Pose	–

Hinweis: Die Drehrichtung ist nach der Rechter-Daumen-Regel festgelegt: Greift eine gedachte Hand um eine Achse des Koordinatensystems, so dass der Daumen vom Ursprung wegzeigt, dann geben die restlichen Finger zur Spitze hin die positive Drehrichtung vor.

Hinweis: Die Rotation um mehrere Differenzwinkel wird nach RPY Konvention und damit nacheinander für die Winkel Roll, Nick und Gier durchgeführt.

3.9.3 Rotiere Pose Global

Dieser Task dient dazu Posen in Eulerwinkel, also Roll, Nick und Gier, zu rotieren. Die Differenzwinkel beziehen sich dabei auf das globale Koordinatensystem des Roboters.

Variablen

Name	Typ	Beschreibung
Pose	Pose	Eingangs-Pose
Roll	Double	Winkel in Grad
Nick	Double	Winkel in Grad
Gier	Double	Winkel in Grad
Ergebnis	Pose	Ergebnis-Pose

Hinweis: Die Drehrichtung ist nach der Rechter-Daumen-Regel festgelegt: Greift eine gedachte Hand um eine Achse des Koordinatensystems, so dass der Daumen vom Ursprung wegzeigt, dann geben die restlichen Finger zur Spitze hin die positive Drehrichtung vor.

Hinweis: Die Rotation um mehrere Differenzwinkel wird nach RPY Konvention und damit nacheinander für die Winkel Roll, Nick und Gier durchgeführt.

Beispiel

In einem Beispiel wird eine Rotation des Gier-Winkels um 90° gezeigt. Veränderungen des Gier Winkels entsprechen Rotationen um die z-Achse (blau). Im Bild erkennt man sowohl die Achsen des globalen Koordinatensystems, das im Roboterursprung liegt, als auch die des lokalen Koordinatensystems der zu verändernden Pose, das hier im Roboterflansch liegt. Betrachtet man nun den Unterschied nach Ausführung des Tasks, so sieht man eine veränderte Orientierung des Posenkoordinatensystems und Roboterflansches. Die Pose wurde lediglich gedreht, die Position der Pose hat sich also nicht geändert. Die Gierdrehung bezog sich auf die globale z-Achse (blau in der Roboterbasis), wodurch die z-Achse der Eingangs-Pose im Roboterflansch verändert wurde.

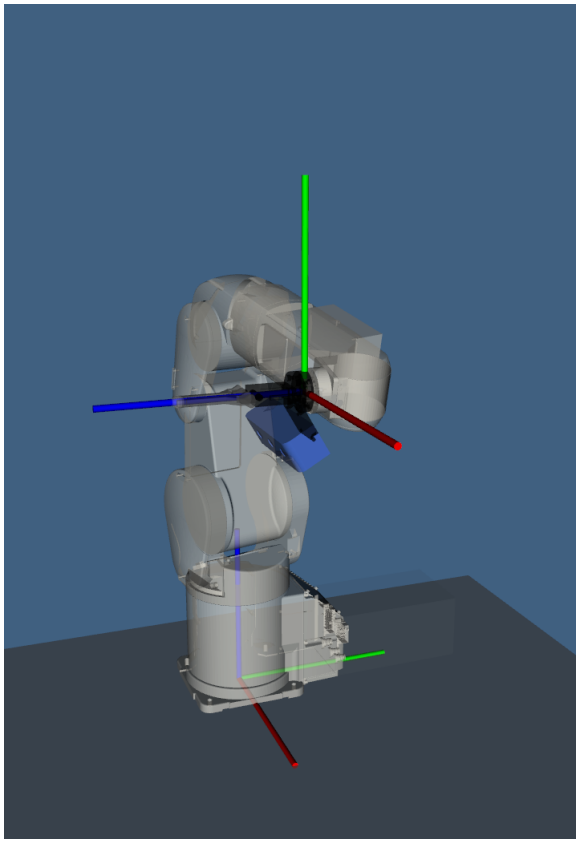


Abb. 11: Screenshot: initiale Pose

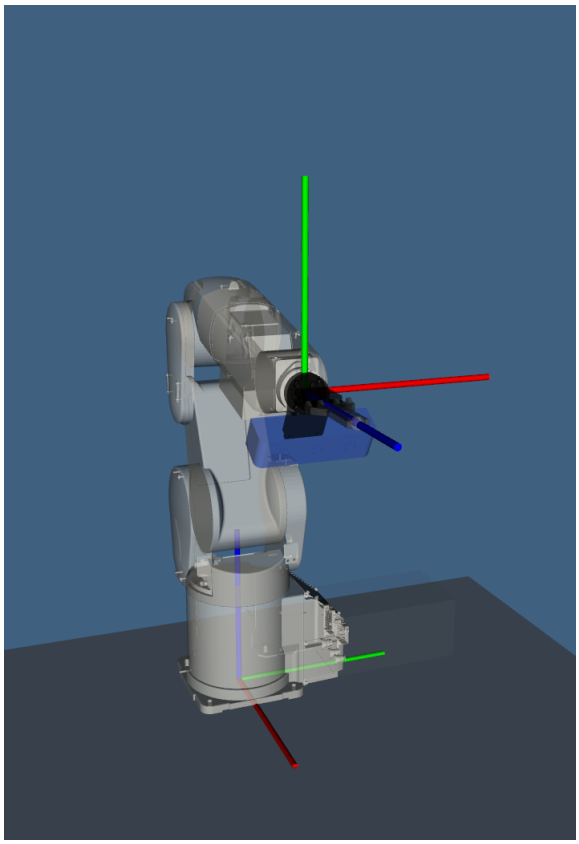


Abb. 12: Screenshot: Pose nach globaler Rotation um die z-Achse (Gier)

3.9.4 Rotiere Pose Lokal

Dieser Task dient dazu Posen in Eulerwinkel, also Roll, Nick und Gier, zu rotieren. Die Differenzwinkel beziehen sich dabei auf das lokale Koordinatensystem des Roboterflansches.

Variablen

Name	Typ	Beschreibung
Pose	Pose	Eingangs-Pose
Roll	Double	Winkel in Grad
Nick	Double	Winkel in Grad
Gier	Double	Winkel in Grad
Ergebnis	Pose	Ergebnis-Pose

Hinweis: Die Drehrichtung ist nach der Rechter-Daumen-Regel festgelegt: Greift eine gedachte Hand um eine Achse des Koordinatensystems, so dass der Daumen vom Ursprung wegzeigt, dann geben die restlichen Finger zur Spitze hin die positive Drehrichtung vor.

Hinweis: Die Rotation um mehrere Differenzwinkel wird nach RPY Konvention und damit nacheinander für die Winkel Roll, Nick und Gier durchgeführt.

Beispiel

In einem Beispiel wird eine Rotation des Gier-Winkels um 90° gezeigt. Veränderungen des Gier Winkels entsprechen Rotationen um die z-Achse (blau). Im Bild erkennt man sowohl die Achsen des globalen Koordinatensystems, das im Roboterursprung liegt, als auch die des lokalen Koordinatensystems der zu verändernden Pose, das hier im Roboterflansch liegt. Betrachtet man nun den Unterschied nach Ausführung des Tasks, so sieht man eine veränderte Orientierung des Roboterflansches und Posenkoordinatensystems. Die Pose wurde lediglich gedreht, die Position hat sich also nicht geändert. Die Gierdrehung bezog sich auf die lokale z-Achse (blau), die dadurch im Gegensatz zu den lokalen x- und y-Achsen unverändert blieb.

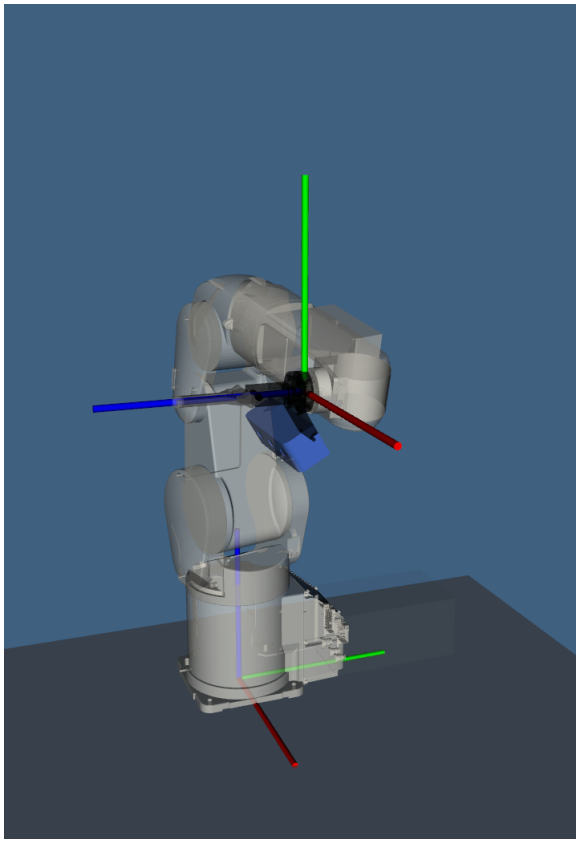


Abb. 13: Screenshot: initiale Pose

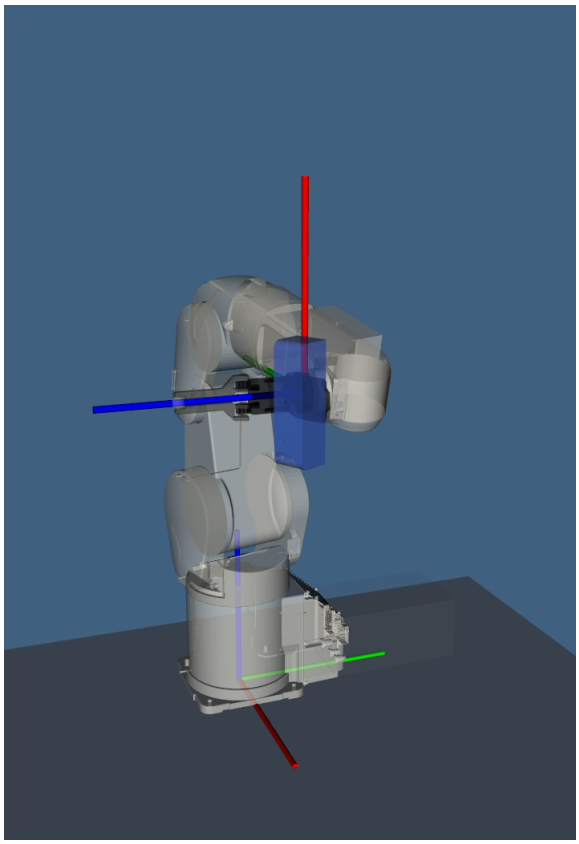


Abb. 14: Screenshot: Pose nach lokaler Rotation um die z-Achse (Gier)

3.9.5 Verschiebe Pose Global

Dieser Task dient dazu Posen zu verschieben, also eine Translation auf sie anzuwenden. Die Pose wird dabei um Differenzabstände als Eingabewerte dieses Tasks im globalen Koordinatensystem der Eingabepose verschoben.

Variablen

Name	Typ	Beschreibung
Pose	Pose	Eingangs-Pose
x	Double	Distanz in Meter
y	Double	Distanz in Meter
z	Double	Distanz in Meter
Ergebnis	Pose	Ergebnis-Pose

Beispiel

In einem Beispiel wird eine Translation in Z-Richtung um 0,1m global durchgeführt. Im Bild erkennt man sowohl die Achsen des globalen Koordinatensystems, das im Roboterursprung liegt. Als auch die des lokalen Koordinatensystems der zu verändernden Pose, das hier im Roboterflansch liegt. Betrachtet man nun den Unterschied nach Ausführung des Tasks, so sieht man eine veränderte Position des Posenkoordinatensystems und Roboterflansches. Die Pose wurde global in Richtung der blauen z-Achse (im Roboterursprung) verschoben.

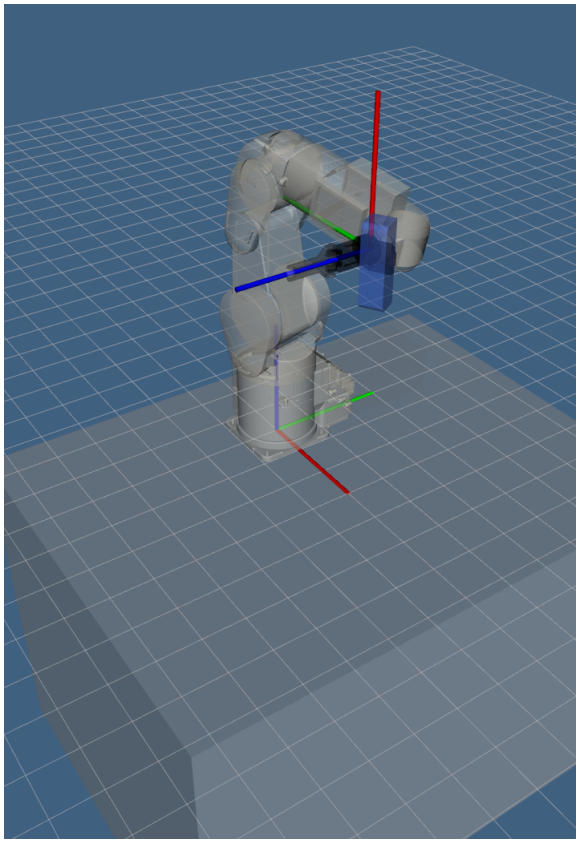


Abb. 15: Screenshot: initiale Pose

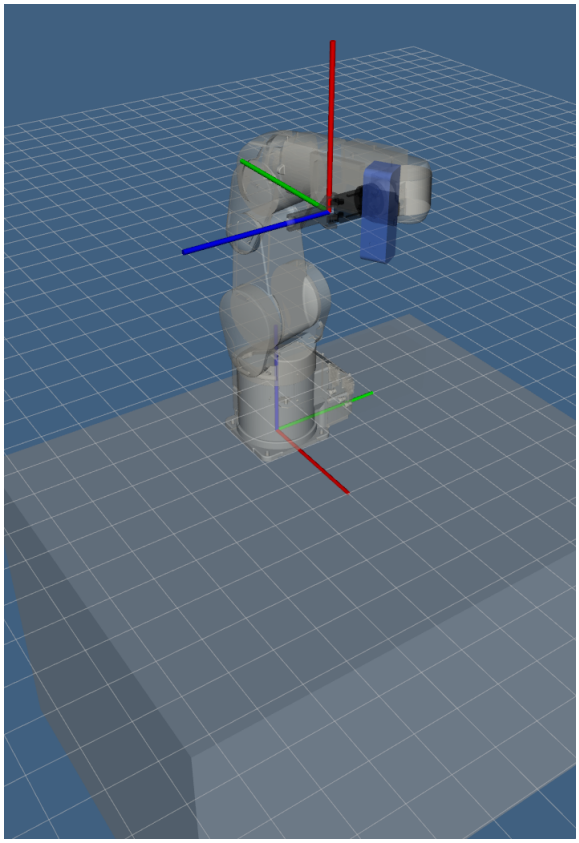


Abb. 16: Screenshot: Pose nach globaler Verschiebung

3.9.6 Verschiebe Pose Lokal

Dieser Task dient dazu Posen zu verschieben, also eine Translation auf sie anzuwenden. Die Pose wird dabei um Differenzabstände als Eingabewerte dieses Tasks im lokalen Koordinatensystem der Eingabepose verschoben.

Variablen

Name	Typ	Beschreibung
Pose	Pose	Eingangs-Pose
x	Double	Distanz in Meter
y	Double	Distanz in Meter
z	Double	Distanz in Meter
Ergebnis	Pose	Ergebnis-Pose

Beispiel

In einem Beispiel wird eine Translation in Z-Richtung um 0,1m lokal durchgeführt. Im Bild erkennt man sowohl die Achsen des globalen Koordinatensystems, das im Roboterursprung liegt. Als auch die des lokalen Koordinatensystems der zu verändernden Pose, das hier im Roboterflansch liegt. Betrachtet man nun den Unterschied nach Ausführung des Tasks, so sieht man eine veränderte Position des Posenkoordinatensystems und Roboterflansches. Die Pose wurde lokal in Richtung der blauen z-Achse (im Roboterflansch) verschoben.

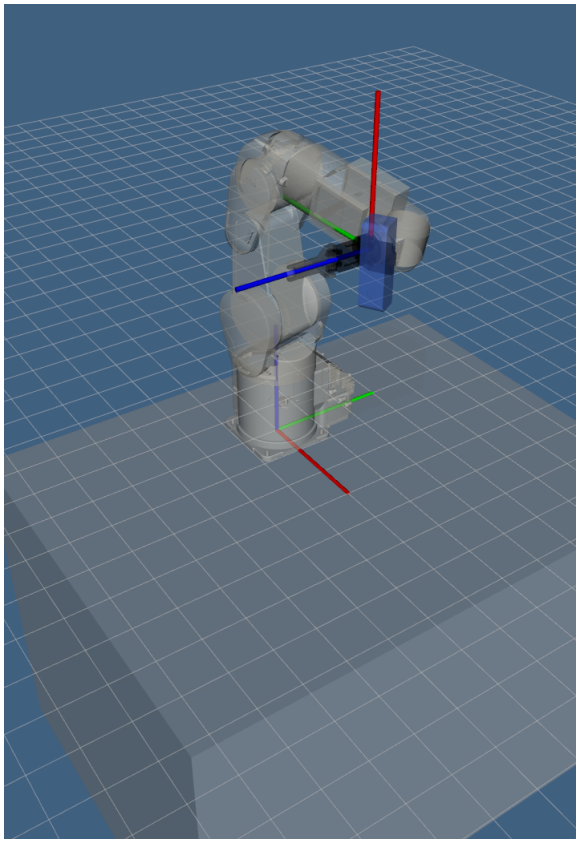


Abb. 17: Screenshot: initiale Pose

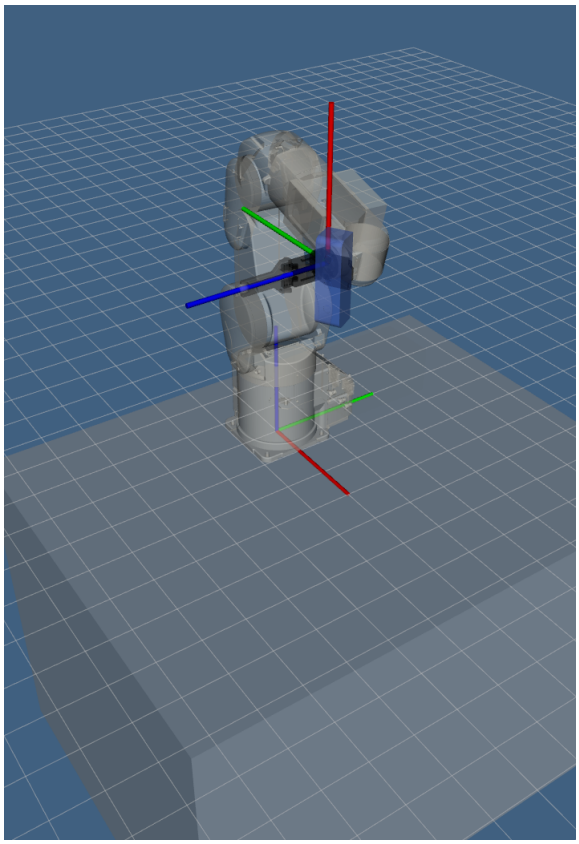


Abb. 18: Screenshot: Pose nach lokaler Verschiebung

3.9.7 Pose transformieren

Multipliziert zwei Posen/Transformationen, beispielsweise um einen Offset auf eine Aufnahmepose anzuwenden.

Hinweis: Die Verschiebung (und Rotation) der Transformations-Pose findet relativ zur Eingabe-Pose statt. Durch Vertauschung der Eingabewerte kann statt einer lokalen eine globale Transformation durchgeführt werden: Um z.B. eine Verschiebung in Z-Richtung des Weltkoordinatensystems zu erreichen, verwenden Sie die Z-Offset-Pose als Eingabe-Pose und die zu verschiebende Pose als Transformation.

Variablen

Name	Typ	Beschreibung
Pose	Pose	Eingabe-Pose
Transformation	Pose	Eingabe-Transformation als Pose
Ergebnis	Pose	Resultat der Transformation als Pose (= Pose * Transformation)

3.9.8 Inverse Transformation

Berechne die inverse Transformation einer gegebenen Transformation/Pose.

Variablen

Name	Typ	Beschreibung
Transformation	Pose	Eingabe-Transformation / -Pose
Inverses	Pose	Ausgabevariable: Inverse Transformation der Eingabe-Transformation

3.10 Erweiterungen

3.10.1 JavaScript

Führe benutzerdefinierten JavaScript-Code aus, um weitere Aktionen umzusetzen, die nicht als Task zur Verfügung stehen. Diese können z.B. mathematische Berechnungen, komplexe Logik oder Schnittstellen zu anderen Systemen sein. Es besteht Zugang zu allen Variablen. Weiterhin können Sequenzwechsel durchgeführt werden.

Weitere Informationen zu diesem Thema in Abschnitt *Mikado Control erweitern* / *JavaScript Task*.

Variablen

Name	Typ	Beschreibung
Code	JScript	Das auszuführende Code-Schnipsel als Quelltext

3.10.2 Halcon Script ausführen

Führe ein Halcon Script mithilfe der HDevEngine aus.

Sollen applikationsspezifische Bildverarbeitungsschritte mithilfe der Halcon Bibliothek realisiert werden, können native Halcon Scripte in den Mikado-Ablauf eingebunden werden. Hierzu muss das Halcon-Programm als .hdev-Datei im Ordner ‚scripts' des Projektverzeichnisses abgelegt werden.

Es besteht Zugang zu den letzten Kameraaufnahmen und zu allen Mikado-Variablen.

Weitere Informationen zu diesem Thema in Abschnitt *Mikado Control erweitern* / *Halcon Script Task*.

Variablen

Name	Typ	Beschreibung
Pfad zum .hdev	String	Name oder Pfad der .hdev-Datei (Absolutpfad oder Relativpfad zum Projektverzeichnis

KAPITEL 4

Tutorials

In diesem Kapitel sind Anleitungen und Beispiele gesammelt, um die Verwendung von Mikado zu veranschaulichen.

4.1 Ein erster Ablauf in zehn Minuten

Die folgenden Schritte erlauben es schnell eine minimale Mikado-Arbeitsumgebung zu schaffen. Dabei wird von einer eingerichteten Zelle, kalibrierten und konfigurierten Kamera ausgegangen.

4.1.1 Aufnahmepose einrichten

- Im Hauptmenü auf Tools und dann Roboter wechseln
- Per Jogging Roboter zu einer geeigneten Aufnahme Pose bewegen

4.1.2 Szene einrichten

- Kiste 1 anlegen, mit Parametrisierung: feste Kiste
- 3D Modell laden
- Punktwolke aufnehmen
- Die Kiste in Überdeckung mit der Punktwolke bringen
- ROI „Suchbereich“ für Objektsuche anlegen und verschieben, sodass der Kistenboden ausgeschnitten wird, zB. auf z=7 mm
- Kiste 2 anlegen, wie oben. An geeignete Ablageposition verschieben

4.1.3 Produkt einrichten

Nun wird ein Produkt eingerichtet, also das Modell geladen, ein einfacher Griff definiert, konfiguriert und erkannt. Ggfs. müssen Meshes vorverarbeitet werden, was *separat beschrieben* ist. Ein Produkt einzurichten und in der Punktwolke zu erkennen wird ausführlicher *in einem weiteren Tutorial* erklärt:

- Auf der Seite Edit -> Produkt kann ein neues Produkt angelegt und der Name angepasst werden.
- Dann werden die wichtigsten Parameter angepasst:

Name	Wert
Mesh	Pfad zur Meshdatei
min. Erkennungs-Score	Startwert niedrig, 20%, dann tunen
Anfahrtsdistanz	kartesische Anfahrt, um Objektkontur zu umfahren
Entnahmedistanz	~Teilhöhe, kartesisches „Rausziehen“, Kollisionen erlaubt
Symmetrie	entsprechend des Objektes wählen
Mindestabstand	~ halber Teildurchmesser, wie nah können zwei Objekte sein?

- Es muss ein geeigneter Außengriff/Innengriff definiert werden, dazu zum Beispiel den entsprechenden Standardgriff anwählen und per grafischem Marker anpassen
- der standardmäßig angelegte, nicht benötigte Innengriff/Außengriff kann entfernt werden
- Produktliste speichern
- Produkt speichern

4.1.4 Produkt speichern

- Zur Objekterkennungsseite wechseln
- Haken bei „Roi“ setzen
- Punktwolke aufnehmen + Anwenden und Objekte finden
- Minimal Score des Produkts anpassen

Hinweis: Bei Erkennungsproblemen bitte überprüfen, ob nach dem Klick auf Punktwolke aufnehmen eine Punktwolke erscheint und ob als aktives Produkt (untere Statusleiste) das neue Produkt ausgewählt ist. Ein Standard Tuneverfahren ist von einem niedrigen Erkennungsscore aus diesen iterativ höher zu stellen, bis keine Fehlerkennungen mehr gezeigt werden. Dies sollte für verschiedene Szenarien probiert werden.

4.1.5 Minimalprogramm erstellen

Es wird eine neue Sequenz „Ablauf“ erstellt, in die nach und nach die folgenden Tasks gezogen werden:

- Setze aktive ROI, Parametrisierung: neue String Variable erstellen, mit „Suchbereich“ befüllen und Initialwert setzen
- Bewege Roboter nach (neue Pose-Variable „Aufnahmepose“, aktuelle Roboterposition übernehmen, Initialwert setzen)
- Punktwolke aufnehmen
- Griffsuche starten
- Teil greifen
- Bewege Roboter nach (neue Pose-Variable „Abwurfpose“, noch nicht einlernen)
- Teil ablegen

4.1.6 Posen einlernen und Variablen anpassen

- Auf den Reiter Edit und Untermenüpunkt Variablen wechseln
- Abwurfpose: „Aktuelle Roboterpose übernehmen“, manuell Werte anpassen, Initialwert setzen

4.1.7 Laufen lassen und etwas variieren

- Zum Beispiel sollte bei Teil Greifen Task Fehlerbehandlung durch Sprünge zu entsprechenden weiteren Sequenzen erfolgen. Diese beinhalten dann im Mindestfall einen Fehler Task, der eine geeignete Anweisung an den Bediener gibt oder variieren im fortgeschrittenen Fall den Ablauf (andere Aufnahmepose oder Kiste)
- Es können mehr Griffe eingefügt werden. Im Griffinspektor kann der Nutzer sich anschauen, welche, warum nicht funktionieren (durch Programm steppen und nach „Griffsuche starten“ anhalten)
- Es kann zum Beispiel ein experimenteller Griff eingefügt werden. Zur Behandlung in Programm muss eine Bool Variable im Task „Teil glreifen“ eingefügt werden. Danach wird geprüft, ob diese auf true gesetzt wurde und eventuell zu einer Sequenz mit anderen Abwurfvariante gesprungen
- Eine Variation, die die Taktzeit optimiert ist die folgende: Nach dem Griff zu einer neuer Sequenz „Griffschleife“ springen, hier Punktwolke aufnehmen, Griffsuche starten, dann zu Abwurfpose fahren, Teil ablegen und Teil greifen. Dann zu „Griffschleife“ springen. Die Berechnung des neuen Griffs erfolgt so im Hintergrund.

4.2 Ein neues Produkt bis zur Objekterkennung

Innerhalb dieses Mikado-Tutorials erlernen Sie, wie Sie ein neues Produkt mithilfe der Mikado-Oberfläche einrichten. Sie benötigen hierfür lediglich ein geeignetes CAD-Modell im Dateiformat STL. Wie Sie Ihr CAD-Modell optimal vorbereiten können, erfahren Sie in einem *separaten Tutorial*.

Bemerkung: Dieses Tutorial bezieht sich auf den Suchalgorithmus Ensenso Partfinder.

4.2.1 Neues Produkt / Produktliste einrichten

- Auf die Seite „Edit“ → „Produkte“ wechseln

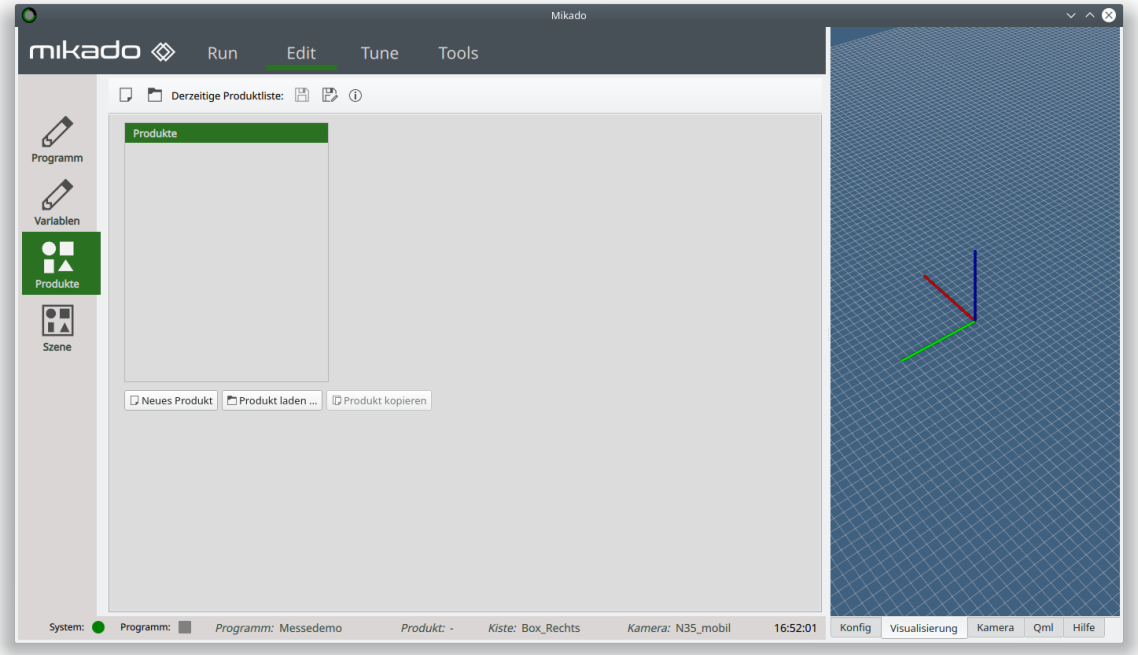


Abb. 1: Produktseite: Leere Ansicht, wenn noch keine Produkte angelegt wurden

- Der Button „Neues Produkt“ fügt der Produktliste ein neues Produkt mit dem Namen „Produkt1“ hinzu

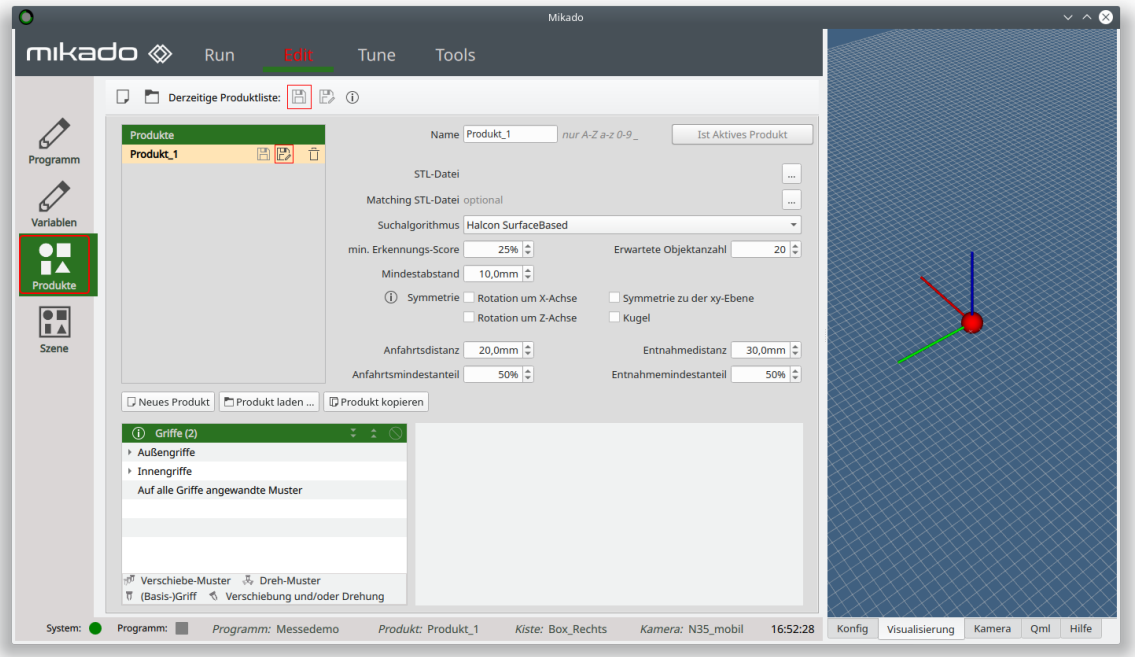


Abb. 2: Produktseite: Ansicht, wenn ein neues Produkt angelegt wurde

- Der Produktname wird im Feld „Name“ angegeben
- Das CAD-Modell im Format STL kann bei „STL-Datei“ angegeben werden. Hierzu den entsprechenden Auswahlbutton mit der Beschriftung „…“ drücken. Das geladene Produktmodell wird nun in der Visualisierung angezeigt.

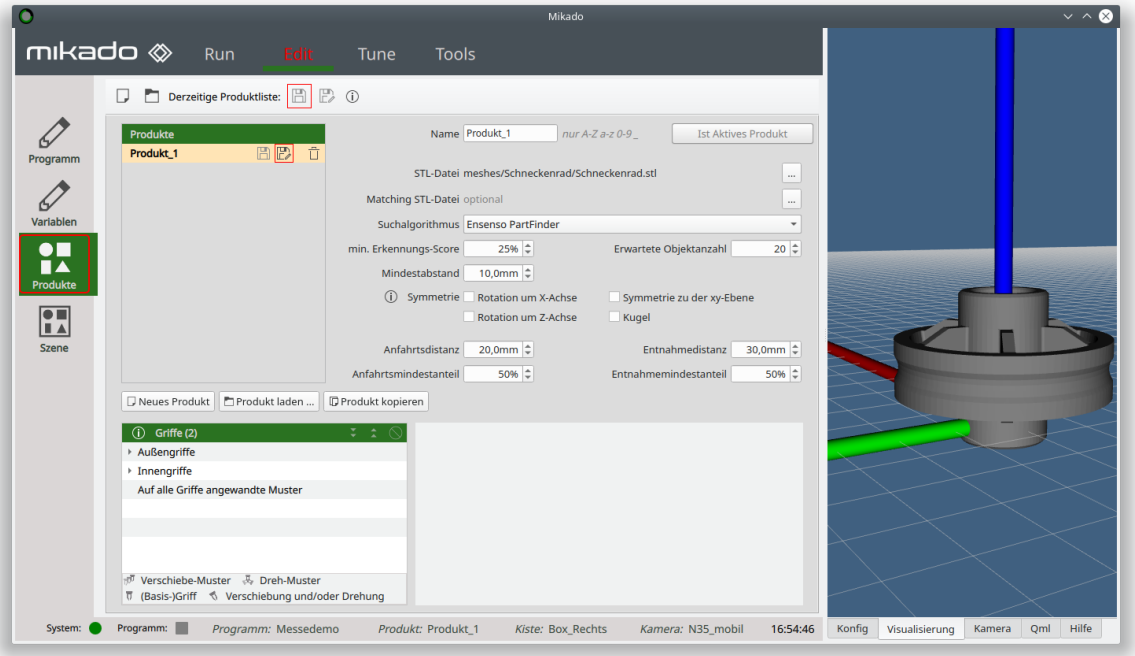


Abb. 3: Produktseite: Ansicht, wenn ein CAD-Modell ausgewählt wurde

- Passen Sie die Parameter an, die für die Objekterkennung relevant sind:

  - Min. Erkennungsscore: Wählen Sie vorerst einen sehr niedrigen Wert (ca. 10-25%). Ein geeigneter Wert wird in einem späteren Schritt mithilfe des Objekterkennungstools ermittelt.
  - Mindestabstand: Robustheit und Performance verbessern sich bei einem geeigneten Mindestabstand. Stellen Sie einen größtmöglichen Wert ein, bei dem nebeneinander liegende Teile weiterhin erkannt werden.
- Details zu allen Produkt-Parametern werden im Kapitel *Produktverwaltung* beschrieben.
- Speichern
  - Speichern Sie zuerst das Produkt (Diskettensymbol in der Produktliste) und wählen einen geeigneten Dateinamen
  - Nun können Sie auch die Produktliste abspeichern (oberes Diskettensymbol direkt unter der Haupt-Navigationsleiste)

4.2.2 Objekterkennung optimieren

- Aktives Produkt setzen
  - Auf die Seite „Edit“ → „Produkte“ wechseln
  - Gewünschtes Produkt in der Produkte-Liste auswählen
  - Button „Als aktives Produkt verwenden“ anklicken
  - In der *Statusleiste* können Sie prüfen, ob das korrekte Produkt nun aktiv ist
  - Auf die Seite „Tune“ → „Erkennung“ wechseln
- Mikado bietet Buttons für die Auswahl von Voreinstellungen für die Objekterkennungsparameter. Hier wird zwischen ‚Modellgenerierung‘ und ‚Suchparametern‘ unterschieden, wobei jeweils die Voreinstellungen ‚Fast‘, ‚Medium‘ und ‚Accurate‘ verfügbar sind. Es empfiehlt sich für den Anfang beide Parametersets auf ‚Fast‘ zu setzen.
- Ansichten beschränken. Bei der Modellgenerierung wird das Produkt aus verschiedenen Perspektiven betrachtet. Für jede Perspektive wird das Produkt in eine entsprechende Punktwolke umgewandelt, welche im Suchprozess mit der tatsächlich aufgenommen Punktwolke verglichen wird. Wenn ein Produkt Rotationssymmetrien aufweist oder nur aus einer bestimmten Richtung sichtbar ist, ist es sinnvoll die Ansichten entsprechend einzuschränken. Damit wird sowohl die Rechenzeit reduziert als auch Fehl-Matches vermieden.
- ROI anwenden → Gewünschte ROI auswählen, damit Hintergrunddaten vorgefiltert werden.

Hinweis: Eine ROI ist nicht zwingend notwendig, kann aber starken Einfluss auf Suchdauer und Zuverlässigkeit haben. Falls eine ROI noch nicht eingerichtet ist: Weitere Informationen in Kapitel *Region of Interest (ROI)*.

- „Punktwolke aufnehmen“ → „Anwenden und Objekte finden“
- Alle Funde größer dem eingestellten Mindesterkennungsscore werden werden visualisiert. Der Erkennungsscore liegt jeweils über einer erkannten Objektinstanz.

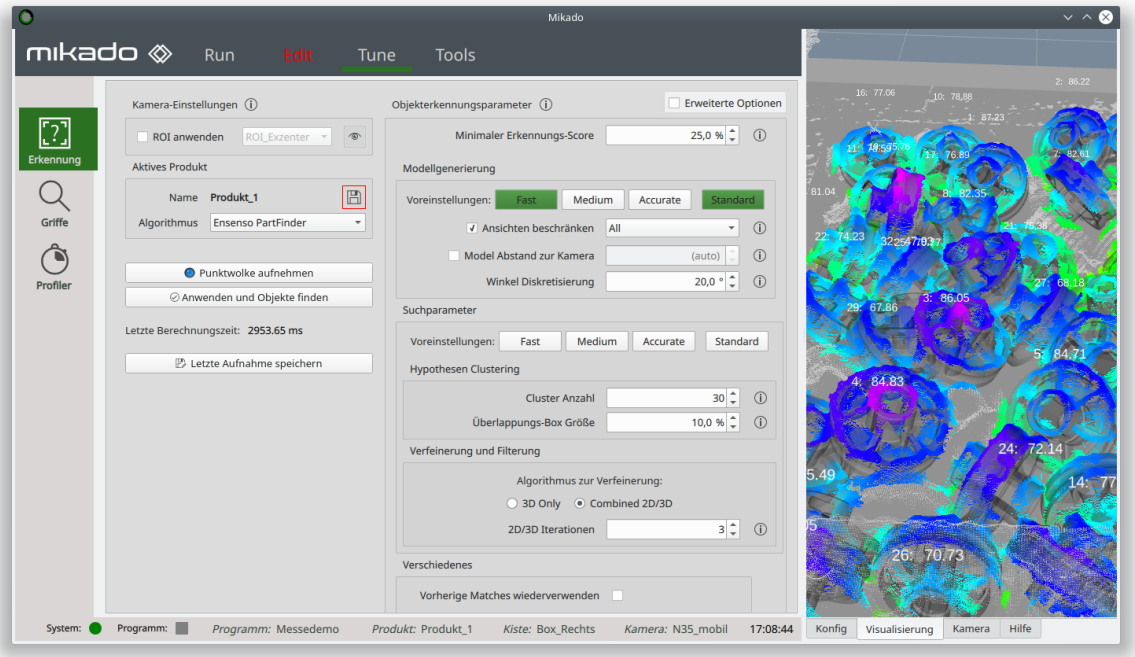


Abb. 4: Objekterkennung: Gefundene Teile werden grau/transparent eingezeichnet

- Sie können nun bewerten, welche Scores erreicht werden können und bei welchen Werten Ungenauigkeiten oder Fehlfunde auftreten. In unserem Beispiel mit den „Schneckenrädern“ liegen die Scores bei Teilen ohne Verdeckung bei über 75%. Daher ist dies hier ein geeigneter Mindestscore.
- Falls in Ihrer Anwendung Situationen auftreten bei denen kein Teil erkannt wird, kann der Score bei diesem Beispiel auch niedriger gewählt werden. Zu beachten ist jedoch, dass der Wert nicht zu gering gewählt wird.
- Weiterhin können ausgehend von den Voreinstellung die Parameter angepasst werden. Sollte die Qualität der Funde mit ‚Fast‘ zu ungenau sein, wählen sie erst ‚Medium‘ und dann ‚Accurate‘.
- Sie können die Parameter auch einzeln anpassen. Hinter jedem Parameter befindet sich in der Mikado-Oberfläche ein Info-Button, der zur entsprechenden Dokumentation verlinkt. Oben rechts befindet sich der Button ‚Erweiterte Optionen‘, durch dessen Aktivierung weitere Parameter erscheinen.
- Führen Sie nun weitere Tests durch, um Ihren neuen Mindestscore und Parameter zu testen. Verwenden Sie verschiedene Füllgrade und verschiedene Anordnungen der Teile, um die Robustheit der Einstellungen zu testen.
- Erhöhen Sie den Mindestscore, sobald Sie einen Fehlfund erhalten. Der neue Mindestscore sollte größer gewählt sein, als der Score eines Fehlfundes.

4.3 Meshlab

MeshLab (http://www.meshlab.net/) ist ein freies Werkzeug, welches auf einem Mikado-Controller installiert und verwendet werden kann. Mit MeshLab können mesh-basierte CAD-Dateien wie zum Beispiel das Dateiformat STL betrachtet und editiert werden. In den folgenden Tutorials werden Möglichkeiten der CAD-Bearbeitung erläutert, die ein optimales Ergebnis bei der Objekterkennung ermöglichen.

Sie können natürlich genauso ein Werkzeug Ihrerer Wahl verwenden, um die Modifikationen an Ihrem CAD-Modell umzusetzen.

4.3.1 Grundlagen: STL laden und speichern

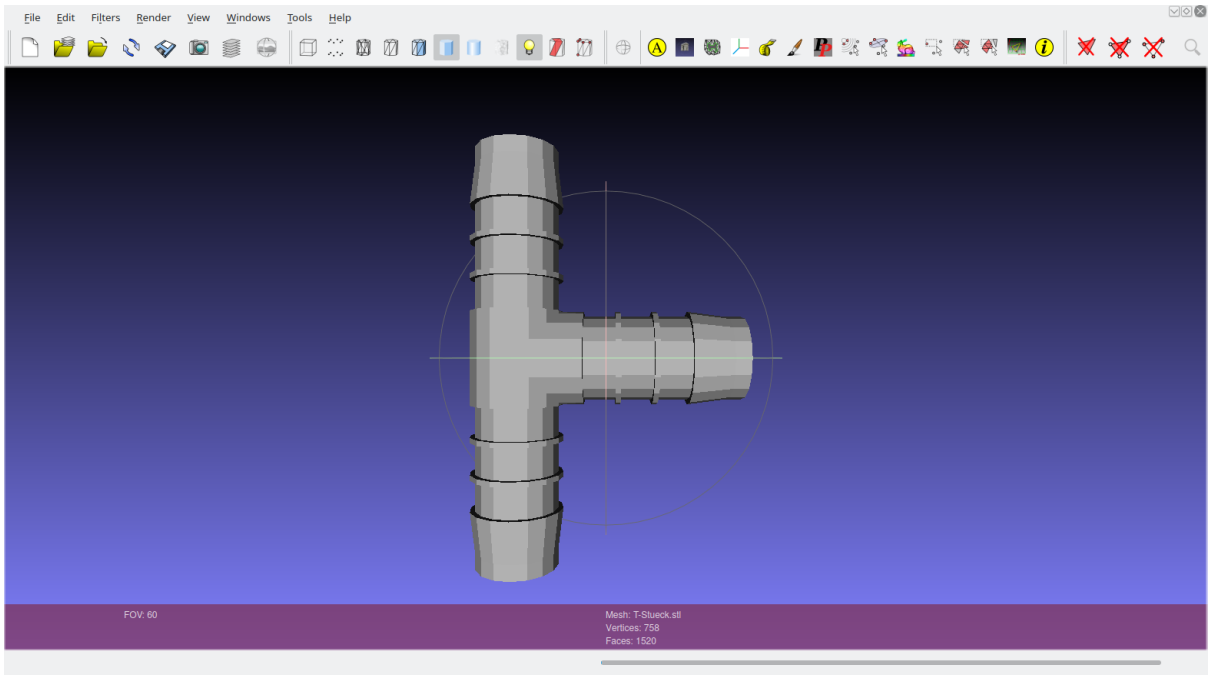


Abb. 5: MeshLab GUI

- MeshLab öffnen und das gewünschte Produkt laden
  - Hauptmenü → File → Import Mesh…
  - Gewünschte Datei auswählen (z.B. im Verzeichnis mikado/cells/<Zellname>/meshes/<Dateiname>.stl)
  - Meldung „Post-Open Processing“ mit OK bestätigen
- STL Datei exportieren:
  - File → Export Mesh As…
  - Neuer Dateiname eingeben
  - STL als Format auswählen → Save

Hinweis: Speichern Sie veränderte CAD Dateien unter neuem Namen, sodass Sie jederzeit alte Versionen wiederherstellen können.

4.3.2 Produktkoordinatensystem zentrieren

- Produktkoordinatensystem anzeigen
  - Render → Show axis

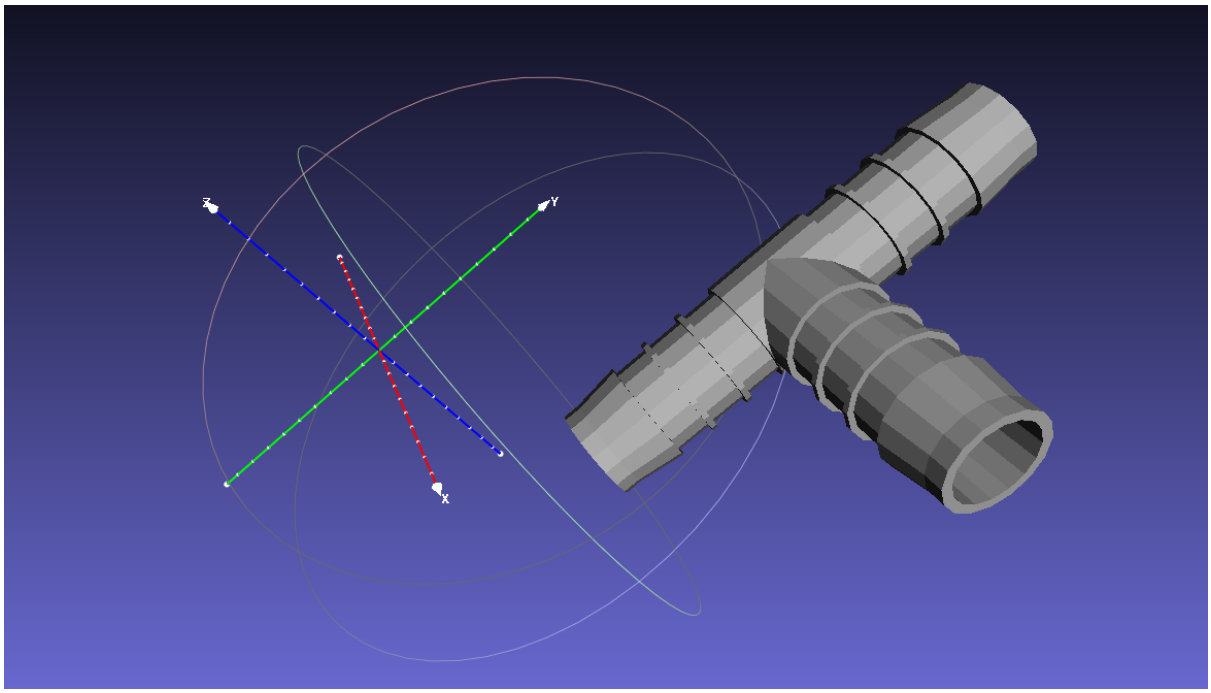


Abb. 6: MeshLab: Koordinatensystem anzeigen

- Koordinatensystem zentrieren
  - Filters → Normals, Curvatures and Orientation → Transform: Move, Translate, Center
  - „translate center of bbox to the origin“
  - Apply

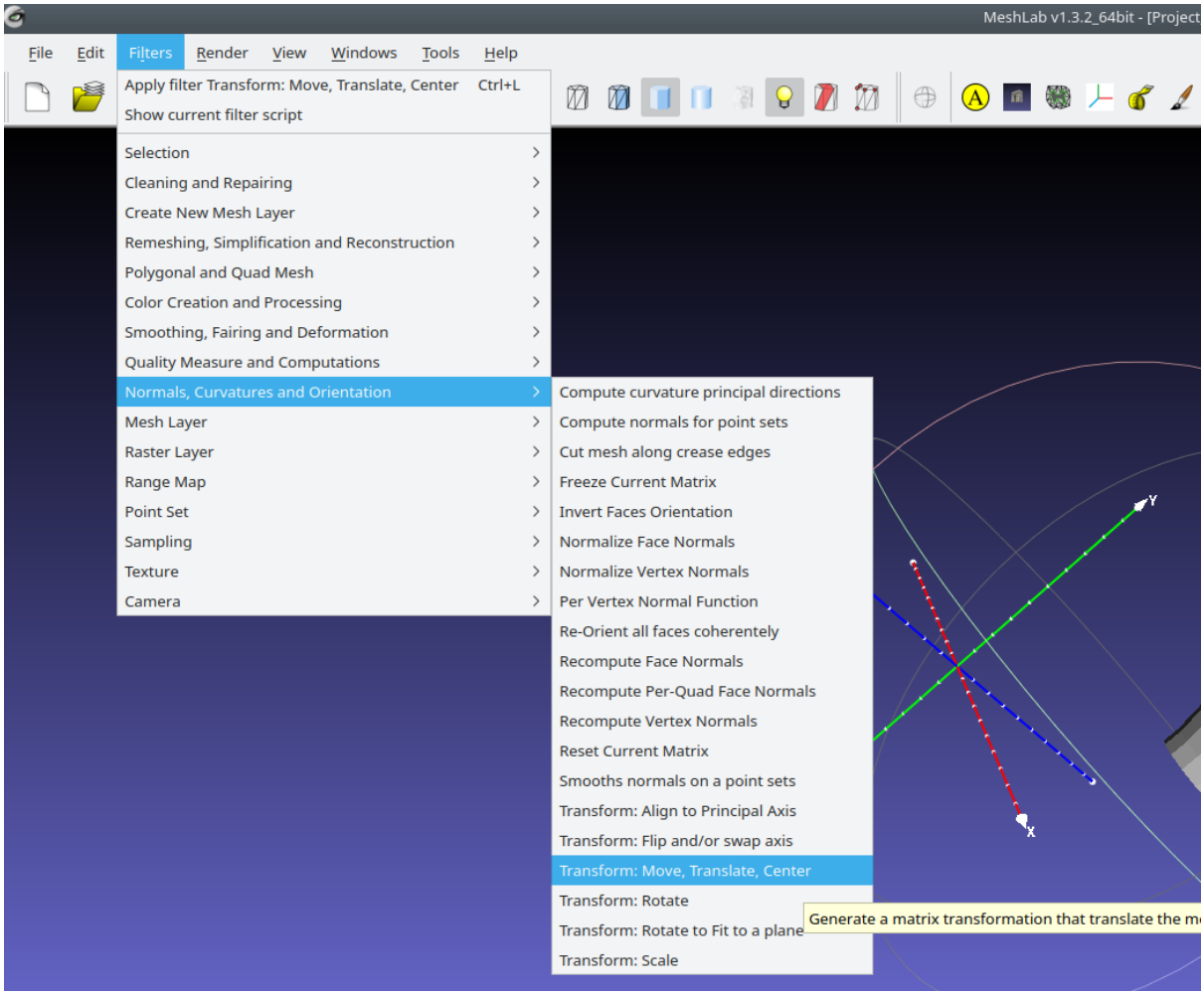


Abb. 7: MeshLab: Koordinatensystem transformieren

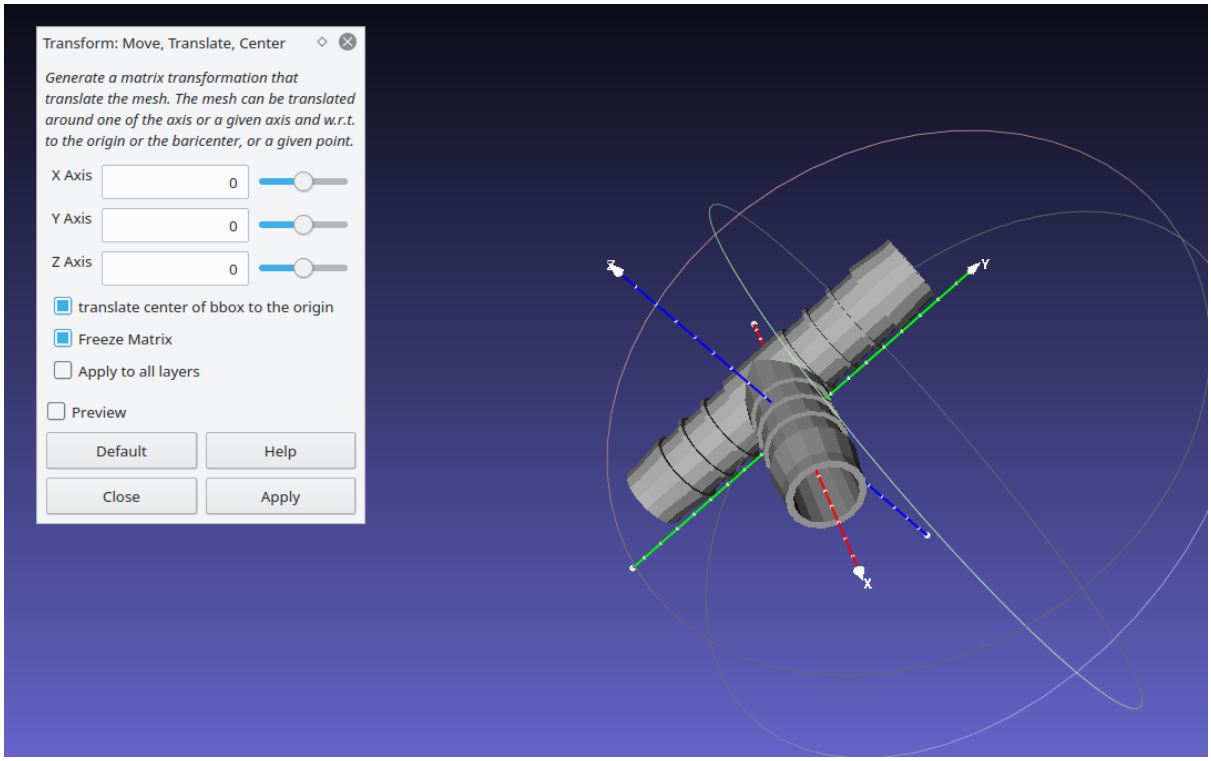


Abb. 8: MeshLab: Koordinatensystem zentriert

Hinweis:

- Mit den „Transform“ Filter kann das Koordinatensystem zudem manuell verschoben und rotiert werden. Liegt das STL in einer anderen Einheit als Millimeter vor, so kann dies auch hier mithilfe des Filters „Transform: Scale“ umgewandelt werden (s. *STL skalieren*).
- Eine Vorschau ist bei den meisten Filter durch das Aktivieren der Option „Preview“ möglich.

4.3.3 STL skalieren

Möchten Sie die Größe Ihres CADs verändern oder die Einheit verändern, so können Sie das STL mithilfe von Meshlab skalieren.

- CAD laden (s. *Grundlagen: STL laden und speichern*)
- CAD skalieren
  - Filters → Normals, Curvatures and Orientation → Transform: Scale

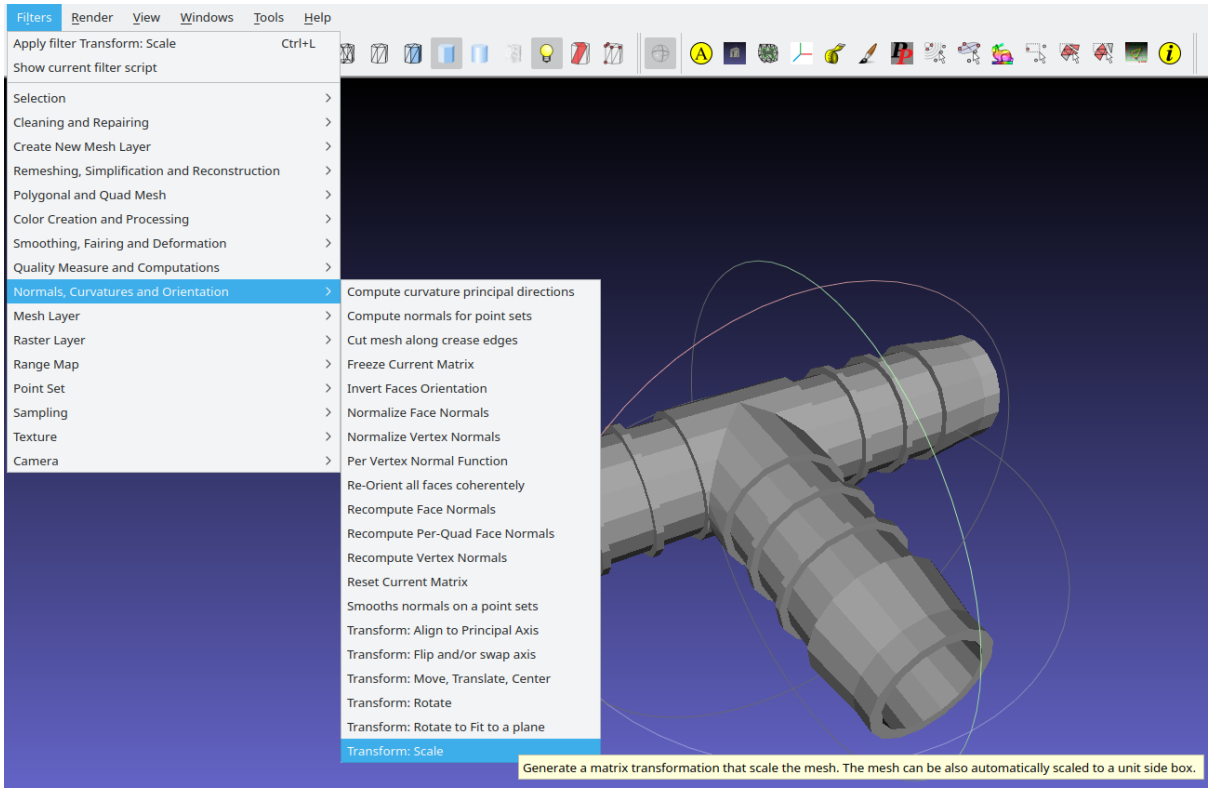


Abb. 9: MeshLab: Filter zum Skalieren

- „Uniform Scaling“ aktivieren
- „Preview“ aktivieren, damit Änderungen sofort dargestellt werden
- Faktor für „X axis“ anpassen (1.0 → 100%, 0.01 → 1%). Möchten Sie z.B. ein CAD, das in Meter definiert wurde zu Millimeter konvertieren, so wählen Sie als Wert 1000.0
- Apply

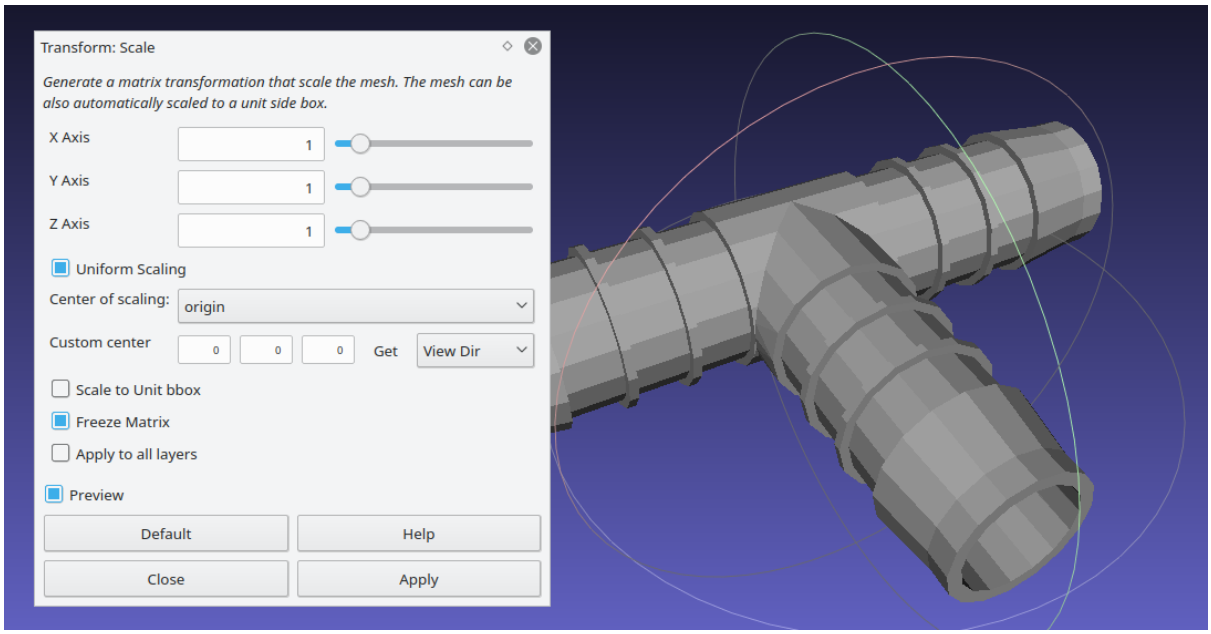


Abb. 10: MeshLab: Filter zum Skalieren - Einstellungen

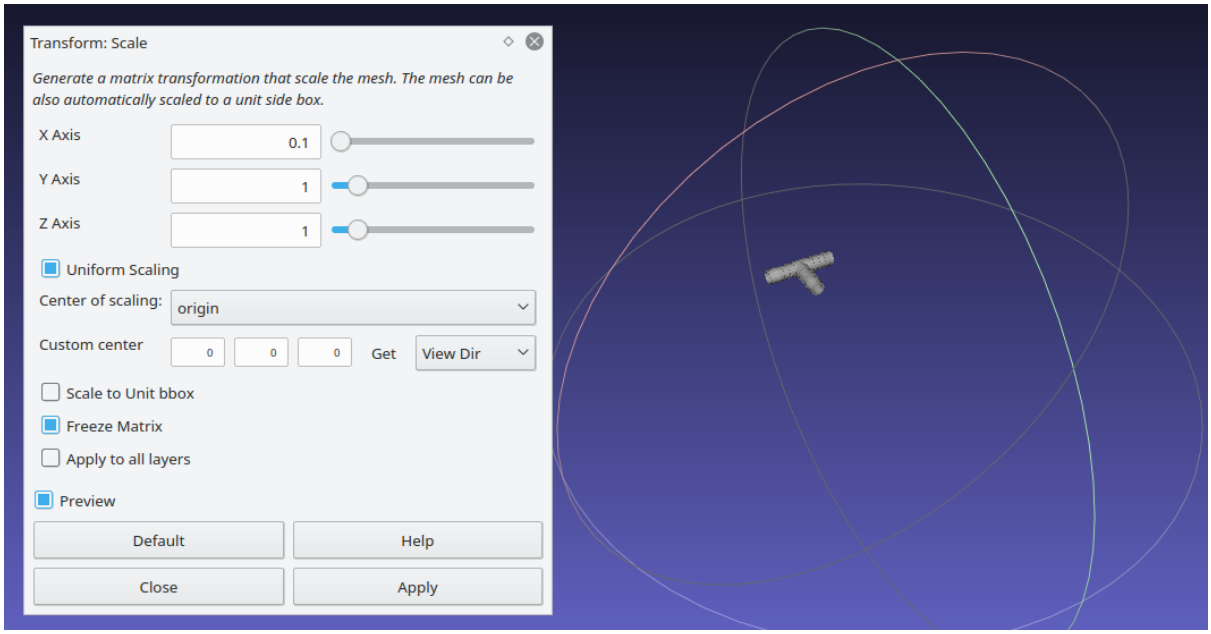


Abb. 11: MeshLab: Skaliertes STL

- STL exportieren (s. *Grundlagen: STL laden und speichern*)

4.3.4 STL Anzahl der Dreiecke reduzieren

Die Anzahl der Dreiecke eines Produkts nimmt Einfluss auf die Ladezeit eines Produkts und insbesondere auf die Kollisionsprüfungen bei der Bahnplanung. Es wird empfohlen, die Anzahl der Dreiecke soweit zu reduzieren, bis wichtige Erkennungsmerkmale und Strukturen des Produkts noch erkennbar sind.

- Momentane Anzahl an Dreiecken ist in der Statusanzeige am unteren Bereich aufgeführt
- Hauptmenü → Filters → Remeshing, Simplification and Reconstruction → Quadric Edge Collapse Decimation

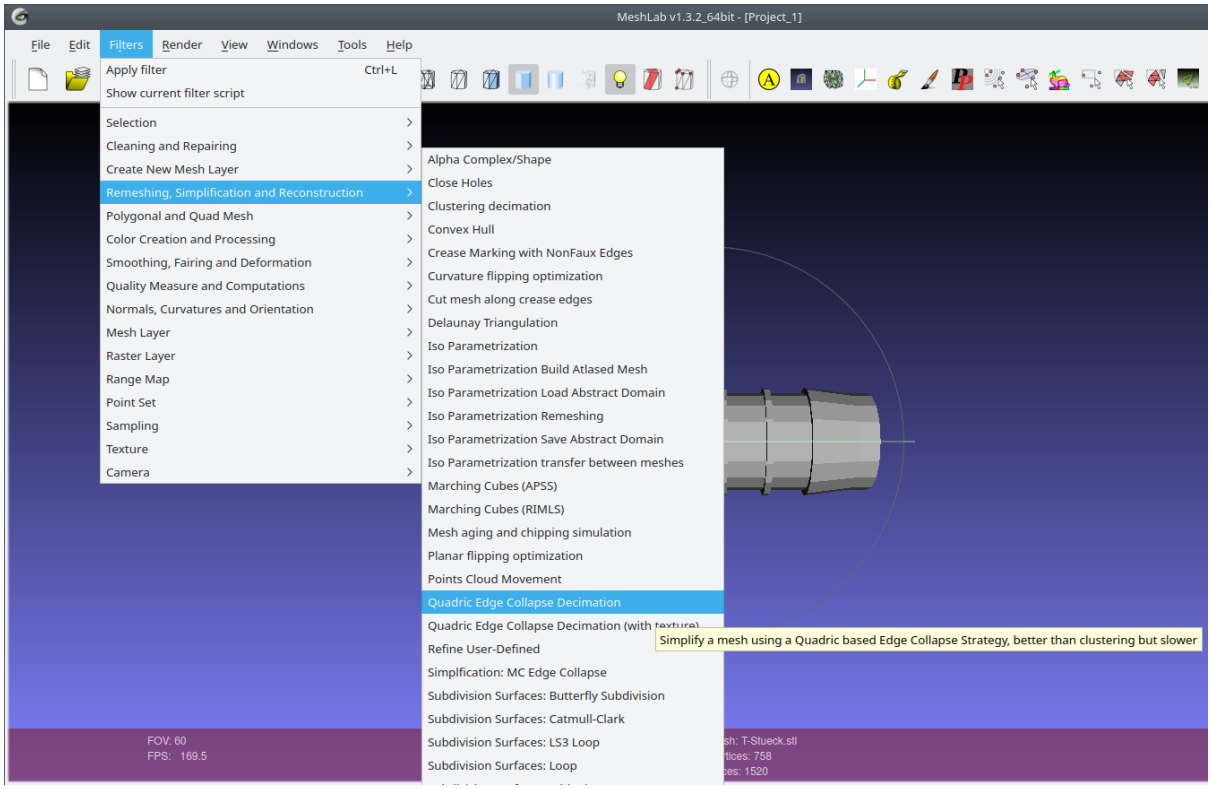


Abb. 12: MeshLab: Dezimierungsfilter

- Gewünschte Anzahl Dreiecke angeben → Apply
  - Bevor eine andere Parametrierung angewendet wird, zuerst Modell neu laden (Button „Reload“: )

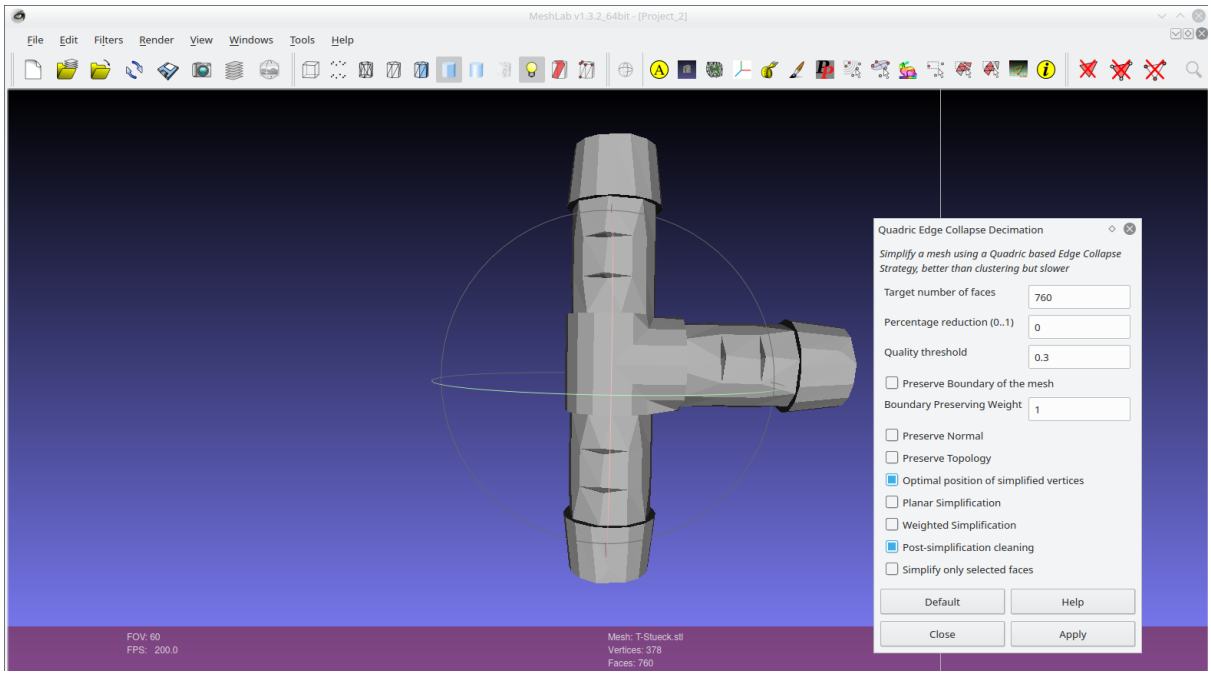


Abb. 13: MeshLab: Dezimiertes Modell des T-Stücks

4.3.5 STL Flächen entfernen

Viele CAD Konstruktionsdaten beinhalten einige Flächen und Strukturen, die aus Kamerasicht nie sichtbar sind. Hierzu zählen innenliegende Strukturen oder auch je nach Anwendungsfall die Unterseite eines Produkts. Auch bei symmetrischen Teilen kann diese Eigenschaft für eine Reduktion des CADs verwendet werden. Eine Kugel beispielsweise ist von einer einzelnen Kamera stets maximal als Halbkugel sichtbar. So kann in diesem Fall die untere Hälfte einer Kugel für eine Objektsuche entfernt werden. Der maximal zu erreichende Score steigt somit. Dies führt zu einer stabileren und schnelleren Objektlagen-Erkennung.

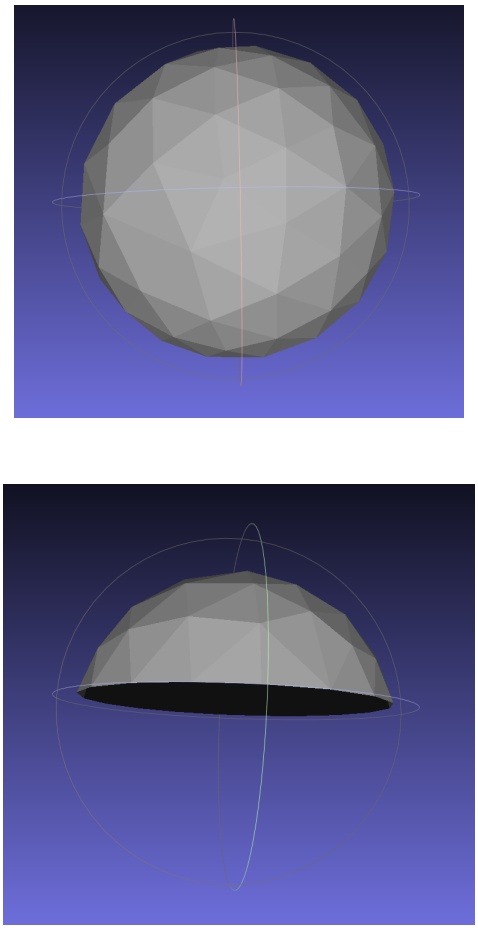


Abb. 14: Beispiel: Für ein Objektmatching einer Kugel führt eine Halbkugel zu höheren Scores. Das Kollisionsmodell muss allerdings weiterhin das komplette Produkt abbilden.

Hinweis: Im Kapitel *Produktverwaltung* wird beschrieben, wie unterschiedliche CADs für Objektsuche und Kollisionsprüfung eingerichtet werden können.

Im Folgenden sind Schritte beschrieben, wie die Anzahl der Dreiecke dezimiert werden kann.

1. **Produkt laden** (s. *Grundlagen: STL laden und speichern*)
2. **Flächen selektieren** (markierte Flächen werden rot eingefärbt)
   a) Option 1: **Rechteckselektion**
   Flächen in einer rechteckigen Region selektieren („Select Faces in a rectangular region“)
   - Selektionssymbol auswählen
   - Mit gedrückter Maustaste Bereich des CADs auswählen. Alle Dreiecke die in den Bereich ragen werden der Selektion hinzugefügt.

Hinweis: Hinweise zur Rechteckselektion:

- Halten Sie die Taste „Strg“ gedrückt, so wird der gewählte Bereich der existierenden Selektion hinzugefügt
- Halten Sie die Taste „Shift“ gedrückt, so wird der gewählte Bereich von der exisitierenden Selektion entfernt
- Mit der Taste „ESC“ kann zwischen Selektions- und Kameramodus gewechselt werden
- Zum Verlassen des Selektionsmodus, das gewählte Symbol erneut anklicken

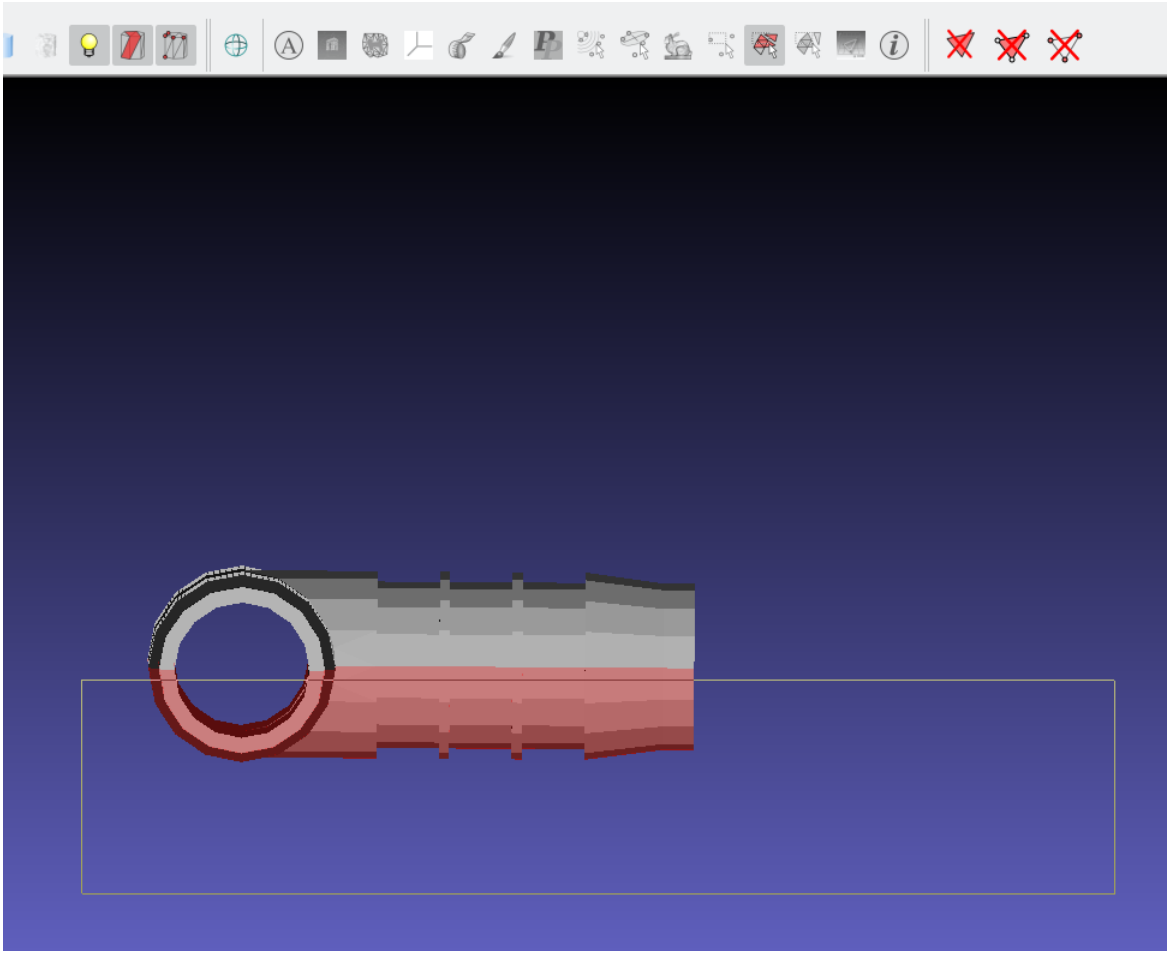


Abb. 15: MeshLab: Rechteckauswahl

b) Option 2: **Pinsel-Werkzeug**
Flächen mit dem Pinsel-Werkzeug selektieren („Z-Painting“)

- Pinsel Symbol anklicken
- Markierungspinsel wählen (roter Pinsel)
- Mit der Maus Flächen „anpinseln“

Hinweis: Hinweise zum Pinsel-Werkzeug:

- Linke Maustaste fügt der Selektion Flächen hinzu
- Rechte Maustaste entfernt der Selektion Flächen

- Pinselart und -größe können mithilfe der Pinseloptionen eingestellt werden
- Mit der Taste „ESC“ kann zwischen Pinsel- und Kameramodus gewechselt werden
- Zum Verlassen des Selektionsmodus, das gewählte Symbol erneut anklicken

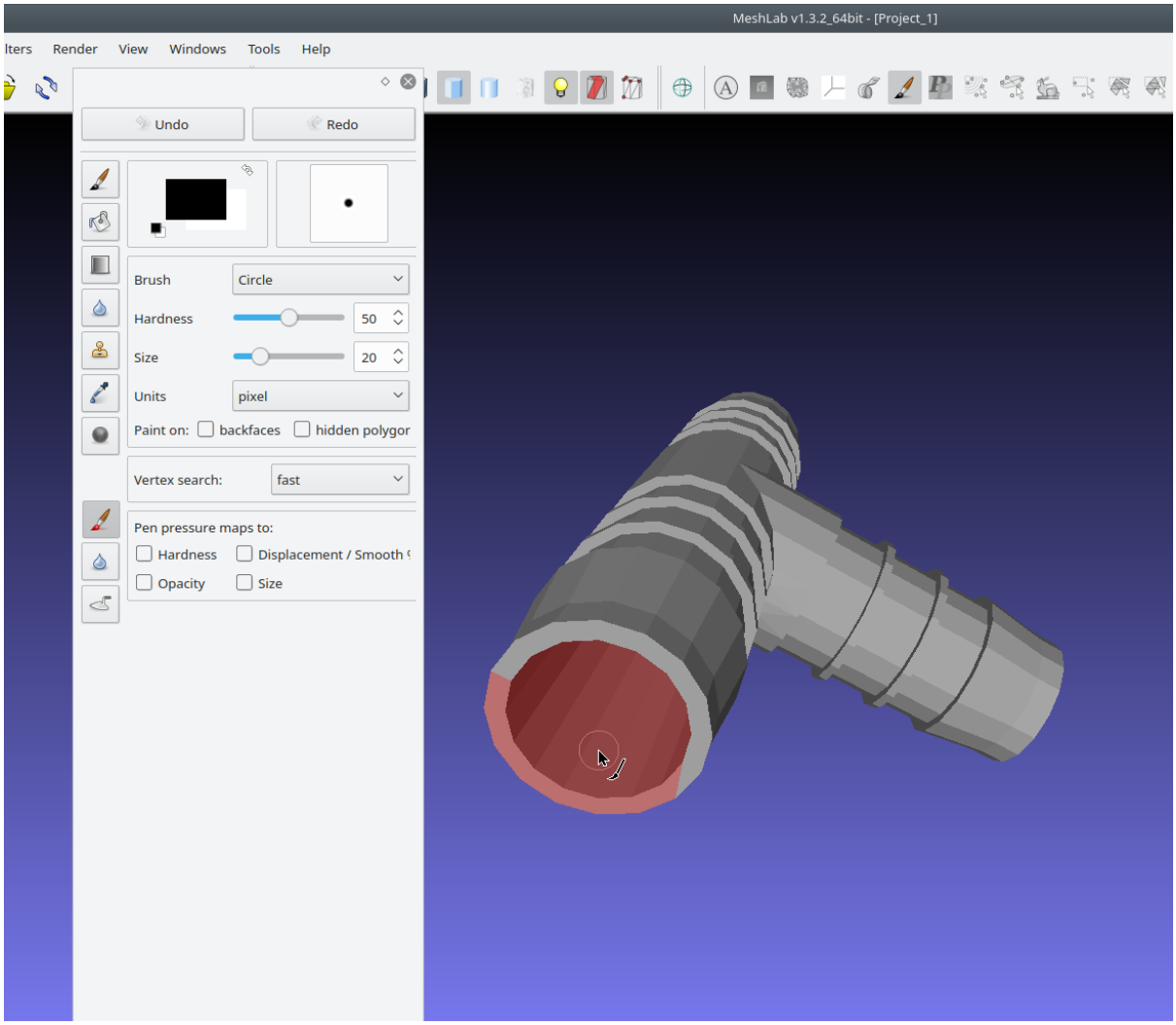


Abb. 16: MeshLab: Pinselauswahl

3. **Flächen entfernen**

   a) **Methode 1:**
   Am Produkt gewählte Strukturen entfernen (Selektierte Flächen entfernen)

   - Selektierte Dreiecke entfernen („Delete the current set of selected faces and all the vertices surrounded by that faces.“)

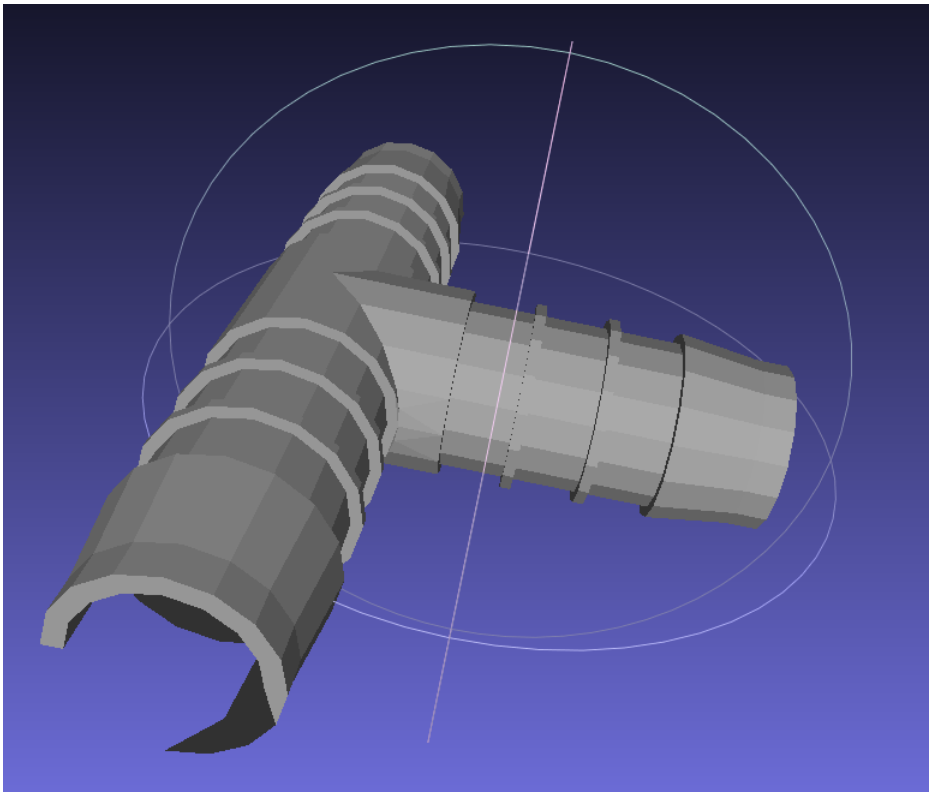


Abb. 17: MeshLab: Ausgewählte Dreiecke entfernt

b) **Methode 2:**
Am Produkt auswählen, was für die Kamera sichtbar ist (Alle Flächen entfernen, die nicht zur Selektion gehören)

- Mit dieser Methode wird am Produkt ausgewählt, was für die Kamera sichtbar ist (Alle Flächen entfernen, die nicht zur Selektion gehören)
- Selektion invertieren
  - Filters → Selection → Invert Selection
  - Apply

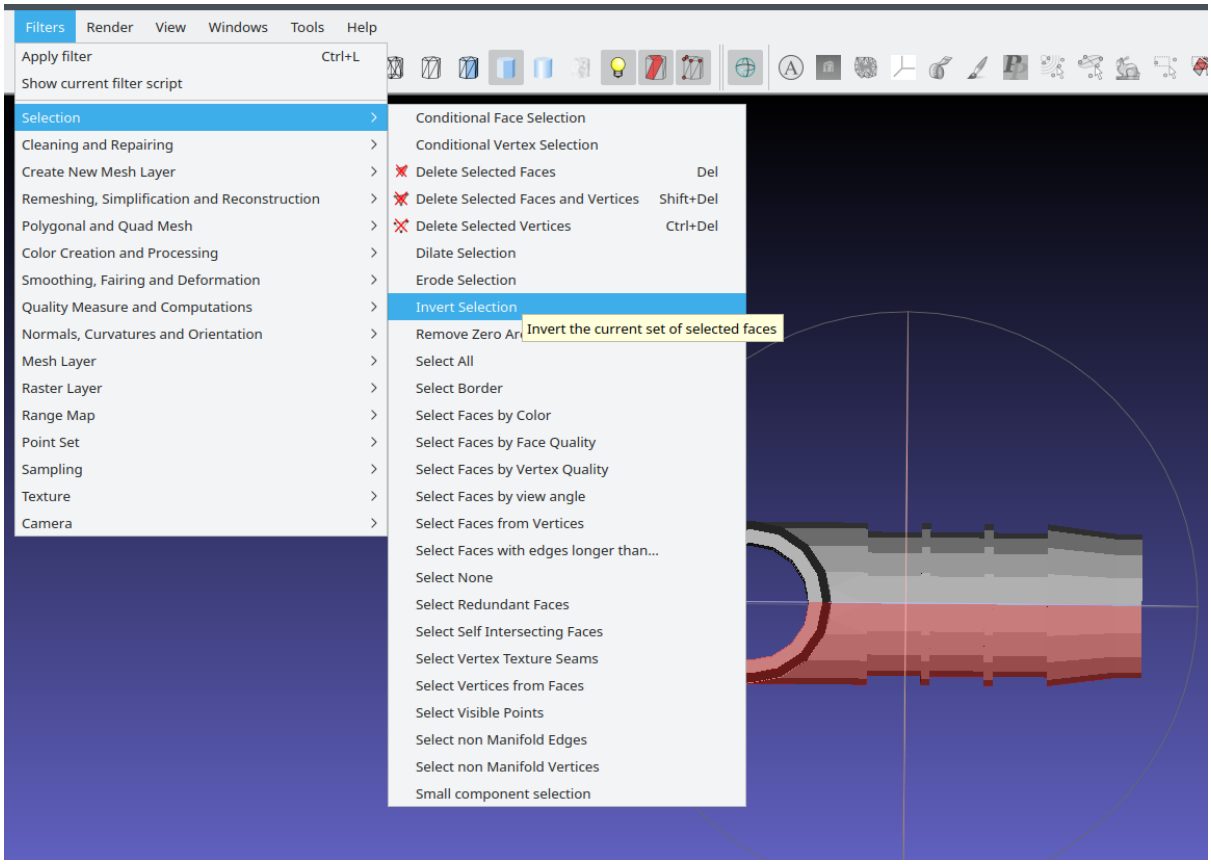


Abb. 18: MeshLab: Auswahl invertieren

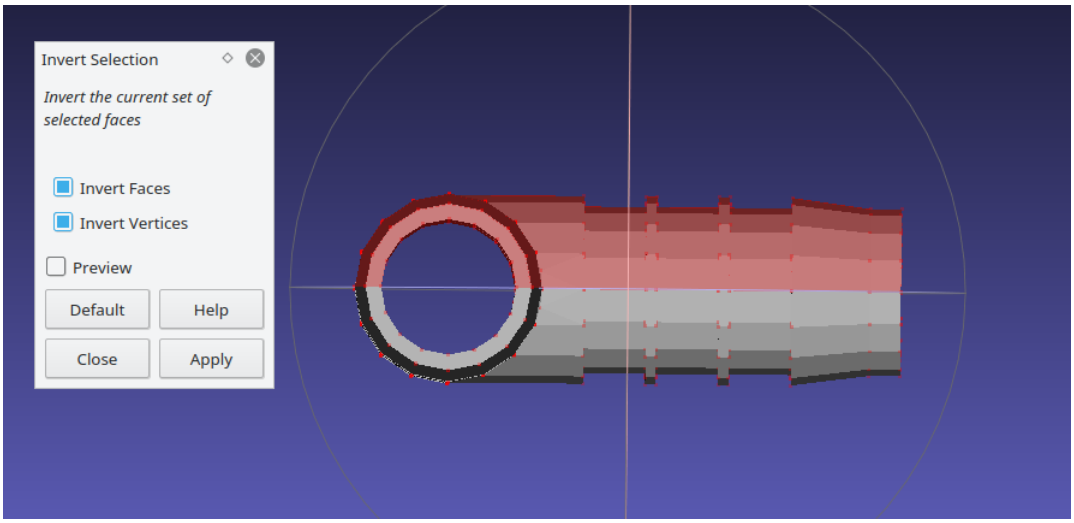


Abb. 19: MeshLab: Auswahl invertiert

- Selektierte Dreiecke entfernen („Delete the current set of selected faces and all the vertices surrounded by that faces.“)

4. **STL exportieren** (s. *Grundlagen: STL laden und speichern*)

Hinweis:

- Betrachten Sie die Selektion vor dem Entfernen von allen Seiten. Oft führen langgezogene oder unförmige Dreiecke zu ungewünschten Selektionen.

- Speichern Sie auch Zwischenversionen ab.
- Unter Umständen kann es hilfreich sein, den Field Of View (FOV) zu verändern. Hierzu während dem Scrollen mit dem Mausrad die Taste „Shift“ gedrückt halten.

4.3.6 STL Innenliegende Flächen automatisch entfernen

In Sektion *STL Flächen entfernen* wird beschrieben, wie Flächen / Dreiecke eines Produkts mit manuellen Werkzeugen selektiert und entfernt werden können. Möchten Sie die innenliegende und somit nicht sichtbaren Dreiecke Ihres Produkts entfernen, so können Sie dies auch MeshLab automatisch ausführen lassen. MeshLab bietet hierzu Algorithmen, die verdeckte Dreicke ermitteln.

Die folgende Anleitung erläutert die einzelnen Schritte anhand des Beispiel-Produkts ‚Rastriegel‘. Die CAD Daten des Herstellers beinhaltet ebenfalls einen innenliegenden Bolzen, der von außen jedoch nicht sichtbar ist. Außerdem sind tiefe Löcher ebenfalls nur selten Teil der Punktwolke, weshalb auch hier Dreiecke entfernt werden können. Diese Flächen werden in den folgenden Schritten mithilfe Meshlab entfernt.

1. **Produkt laden** (s. *Grundlagen: STL laden und speichern*)

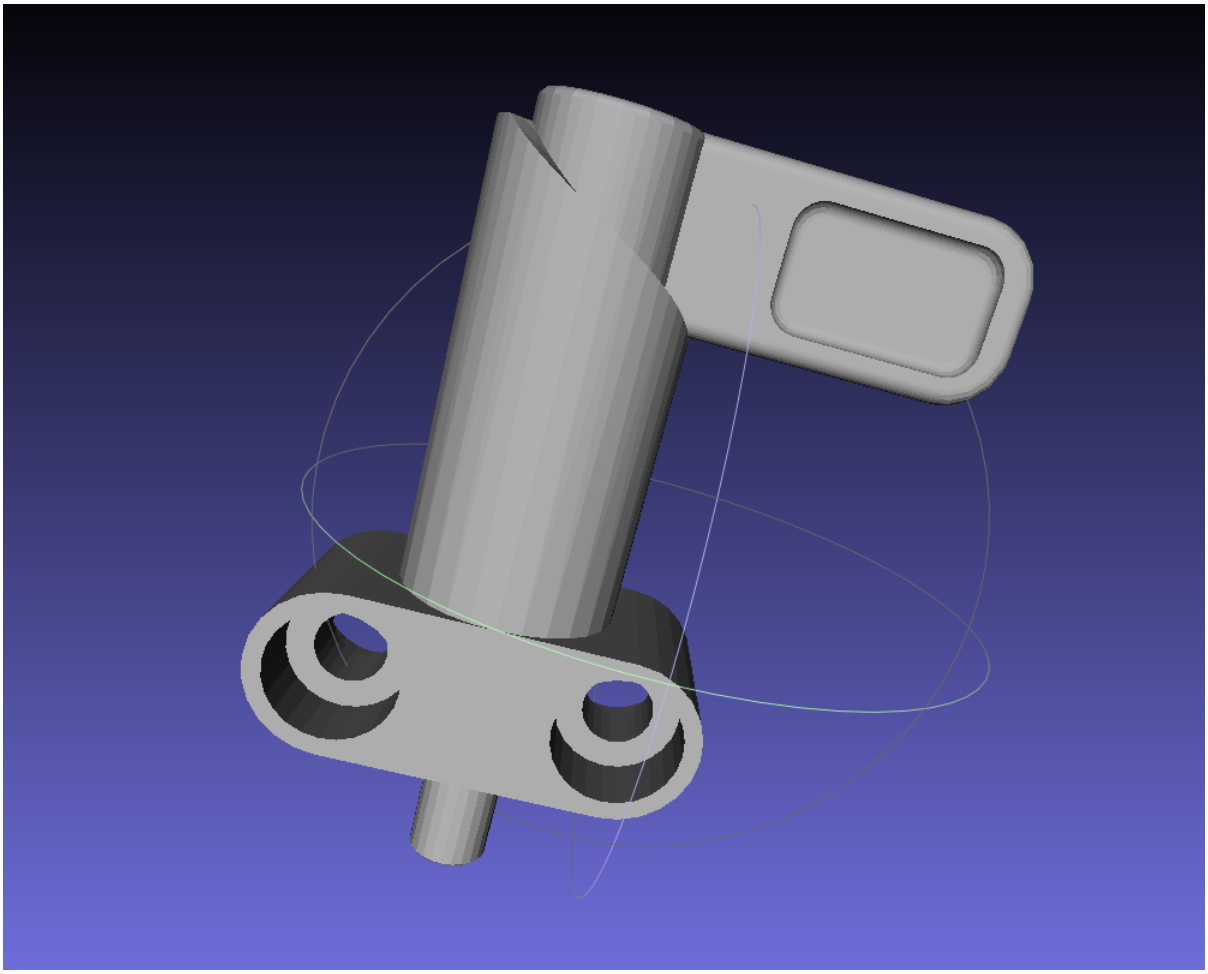


Abb. 20: MeshLab: Ein Beispiel-Produkt ‚Rastriegel CAD‘

2. **Ambiente Verdeckung berechnen**

   In diesem Schritt wird für jedes Dreieck eine Art „Sichtbarkeits-Bewertung“ berechnet. Dieser Wert dient im Folgeschritt als Grundlage zur Entscheidung, ob das Dreieck entfernt werden kann oder nicht.

   a. Öffnen Sie den Dialog für den Ambient Occlusion Filter unter

Filters → Color Creation and Processing → Ambient Occlusion Per Vertex

b. Aktivieren Sie ‚Use GPU acceleration', danach bestätigen Sie mit ‚Apply' Die Berechnung wird gestartet und jedem Vertex wird eine Sichtbarkeits-Bewertung als Qualitätsmerkmal hinterlegt. Die Flächen werden bei guter Sichtbarkeit nun hell und bei wenig Sichtbarkeit dunkel dargestellt.

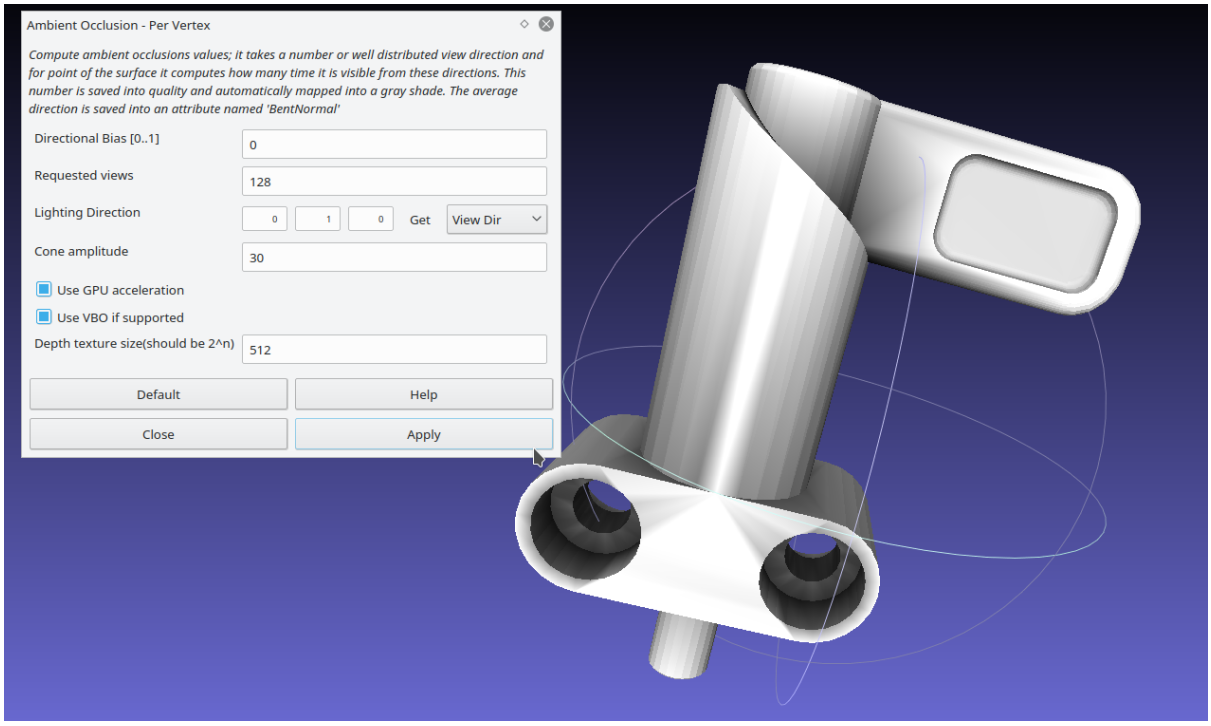


Abb. 21: MeshLab: „Sichtbarkeits-Bewertung" mithilfe des Ambient Occlusion Filters

3. **Flächen anhand der Sichtbarkeit selektieren**

Die Ergebnisse des vorigen Schritts kann nun als Grundlage für eine Selektion/Auswahl von Dreiecken verwendet werden.

a. Filters → Selection → Select Faces by Vertex Quality

b. Parameter ‚Min Quality' auf den niedrigsten Wert stellen.

c. „Preview" aktivieren

d. ‚Max Quality' anpassen, bis lediglich die gewünschten Flächen selektiert werden (rot eingefärbt)

e. ‚Apply'

Hinweis:

- Unter Umständen können teilverdeckte Flächen nicht klar von den zu entfernenden Flächen getrennt werden. In diesem Fall können Sie auch manuelle Werkzeuge verwenden um die Selektion nachträglich anzupassen (s. *STL Flächen entfernen*).
- Unter ‚Render' → ‚Render Mode' kann zwischen verschiedenen Arten der Visualisierung des CAD Modells gewechselt werden. Die ‚Wireframe'-Darstellung erlaubt es beispielsweise auch innenliegende Flächen als Dreiecksnetz sichtbar zu machen.

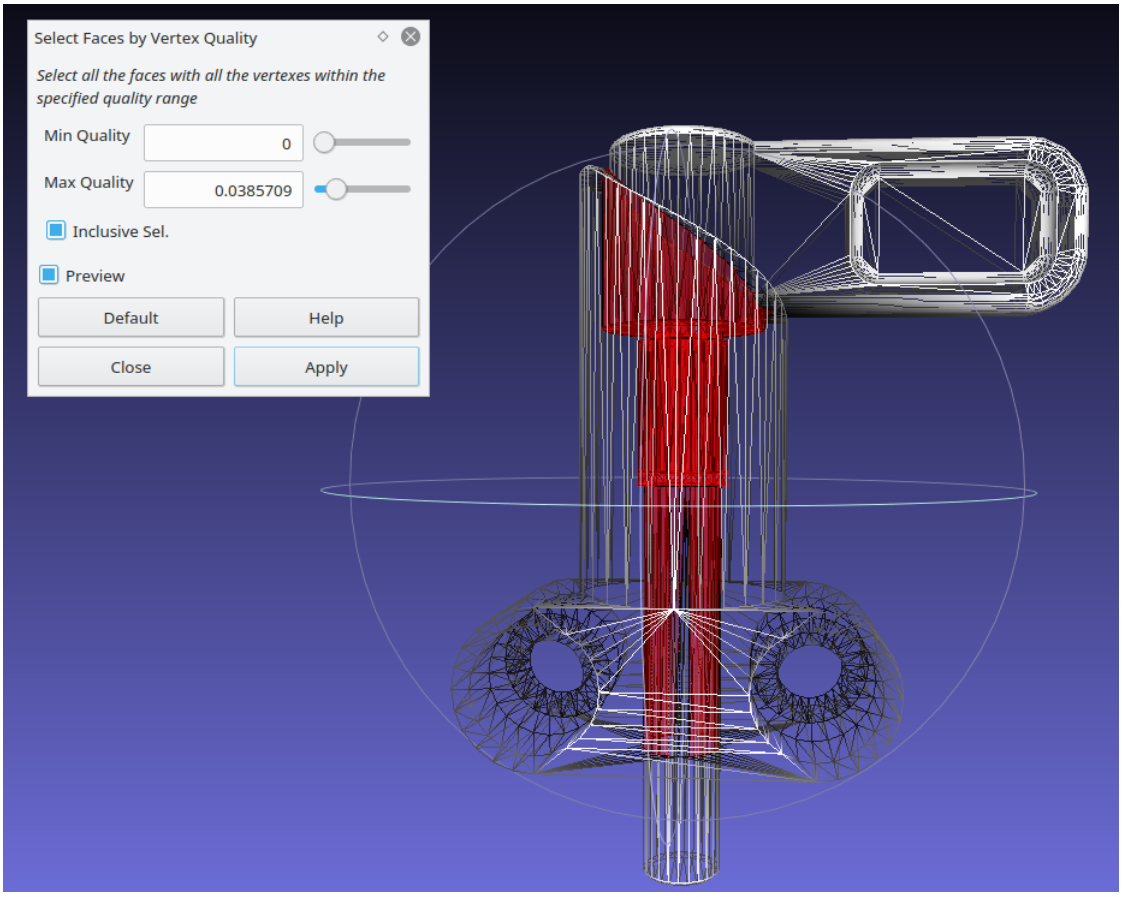


Abb. 22: MeshLab: Selektion anhand der Sichtbarkeit einer Fläche mit niedrigem Schwellwert

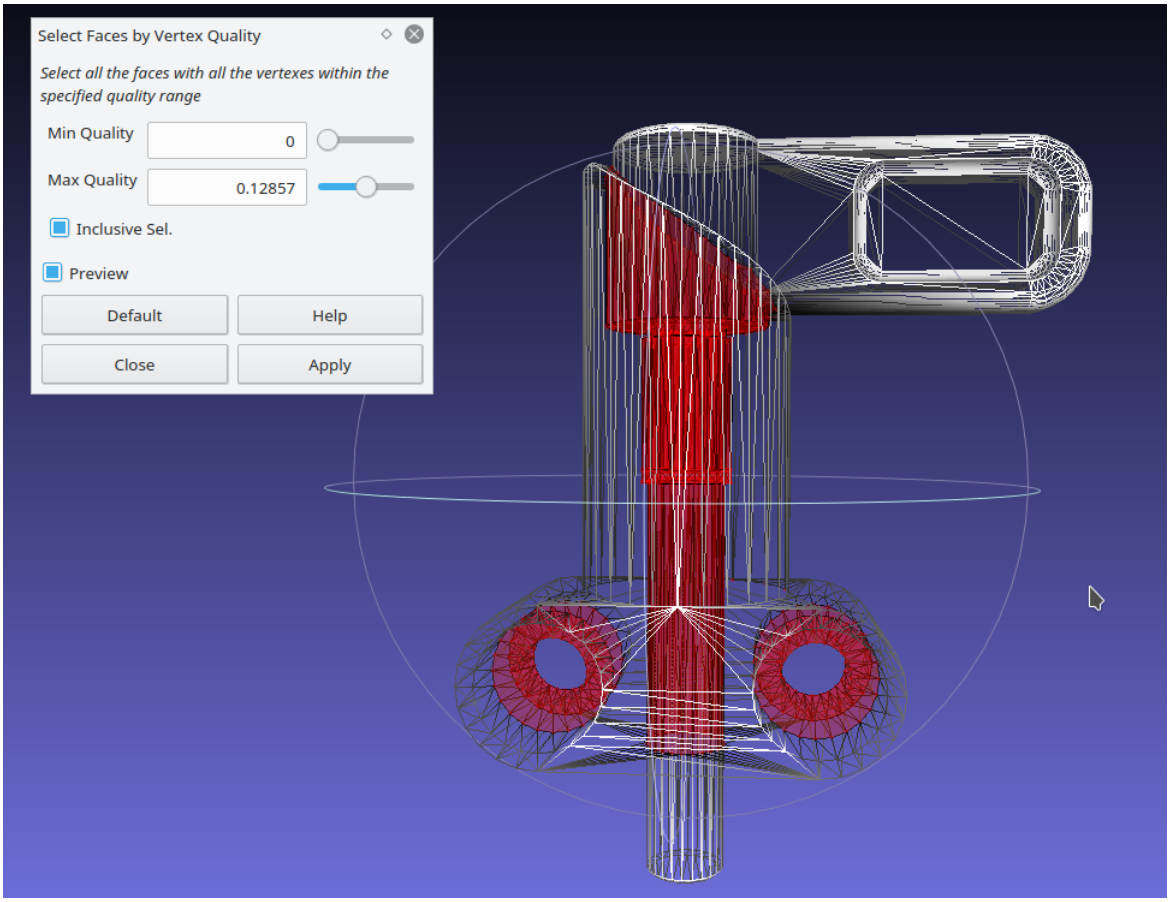


Abb. 23: MeshLab: Selektion anhand der Sichtbarkeit einer Fläche mit höherem Schwellwert: weitere innere Flächen werden selektiert. Im unteren Bereich kommen nun ebenfalls ausgewählte Flächen hinzu, da die Strukturen in einer Vertiefung liegen.

4. **Flächen entfernen**

   Selektierte Dreiecke entfernen („Delete the current set of selected faces and all the vertices surrounded by that faces.“)

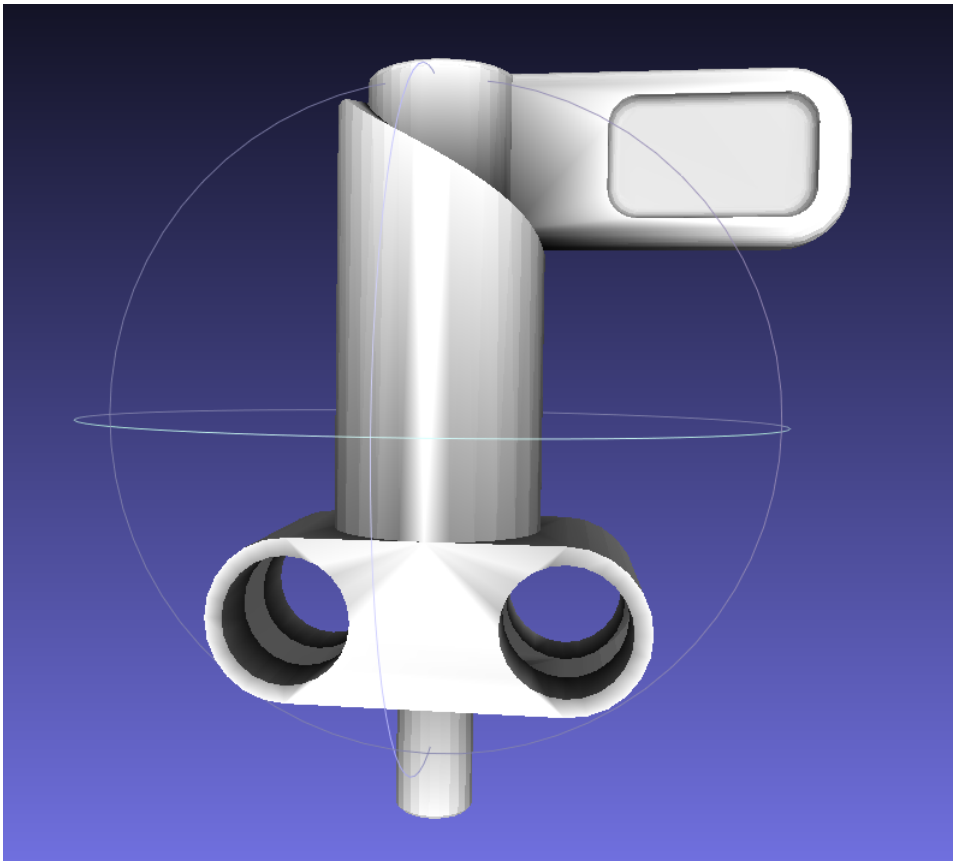


Abb. 24: MeshLab: Die bearbeitete STL hat nun kaum noch innere Strukturen und kann nun weiter manuell nachgearbeitet oder gespeichert werden. Da in diesem Beispiel ein hoher Schwellwert gewählt wurde, sind die Ringe innerhalb der unteren Löcher ebenfalls entfernt worden.

5. **STL exportieren** (s. *Grundlagen: STL laden und speichern*)

4.4 Halcon Script einbinden: Datamatrix Erkennung

Der Task *Halcon Script ausführen* ermöglicht die Einbindung eigener Halcon-Scripte (Hdev-Scripts) in einen Mikado-Programmablauf. Im folgenden Tutorial wird anhand eines Beispiels erläutert, welche Schritte nötig sind, um eine eigene Erkennung einer Datamatrix oder eines QR-Codes mittels der Halcon-Bibliothek umzusetzen. Die Bilddaten werden hierfür aus einer der beiden Mono-Kameras aus dem Stereobild der Ensenso-Kamera ausgelesen.

In einer praktischen Anwendung ist eine solche Funktion hilfreich um beispielsweise einen Produktwechsel über den Barcode anzustoßen. Folgende Barcodes wurden verwendet:



Abb. 25: Datamatrix Code für den Artikel '12345678'



Abb. 26: Datamatrix Code für den Artikel '98765432'



Abb. 27: QR Code Mikado (Inhalt ist ein Link auf die Mikado-Webseite)

4.4.1 Vorbereitung

Da mithilfe des Barcodes ein Produktwechsel stattfinden soll, benötigen Sie zuerst eine geeignete Produktliste. In diesem Beispiel beginnen die Produkte mit dem Präfix `Product_` gefolgt von der Artikelnummer, die durch den Barcode eingelesen wird. Das Einrichten von Produkten ist im Kapitel *Produktverwaltung* erläutert.

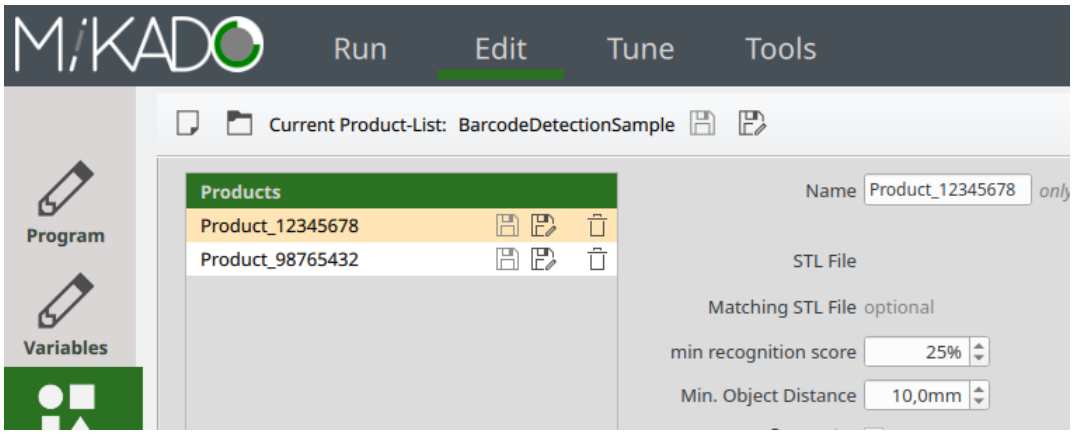


Abb. 28: Beispielhafte Produktliste

Weiterhin benötigen Sie eine Kamera-Konfiguration für Ihre Ensenso Kamera, die für die Aufnahme eines Monobildes geeignet ist (insbesondere sollten Sie den Projektor deaktivieren und Belichtungszeit für einen guten Kontrast anpassen). Speichern Sie die Konfigurationsdatei schließlich in das ‚config' Verzeichnis Ihrer Roboterzelle z.B. unter ‚barcodeDetection.json'.

Hinweis: Erläuterungen zu den Verzeichnissen sind unter *Verzeichnisstruktur* zu finden.

4.4.2 Programmablauf

Im ersten Schritt wird der Programmablauf wie abgebildet mithilfe des *Programm-Editors* erstellt.

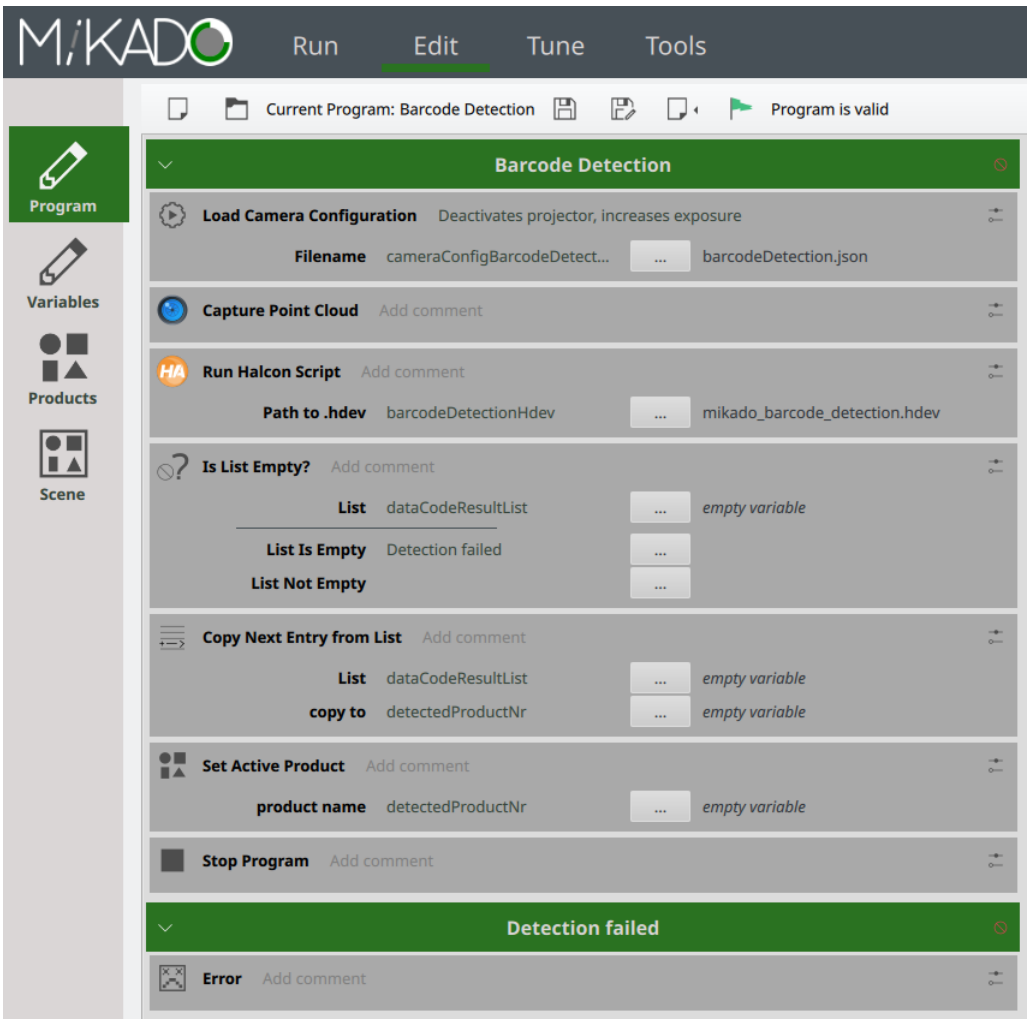


Abb. 29: Programm-Editor: Programmablauf für die Einbindung eines Hdev-Scripts

- *Startsequenz: 'Barcode Detection'*
  Wird bei Programmstart einmalig ausgeführt und hält am Ende der Sequenz an.
  - *Kamera Konfiguration laden*
    Wechselt die Konfiguration für die Ensenso Kamera. Die Konfiguration `barcodeDetection.json` ist hinsichtlich 2D Bildaufnahme optimiert. In diesem Fall wird der Projektor deaktiviert und die Belichtungszeit für gute Kontraste angepasst.
  - *Punktwolke aufnehmen*
    Triggert die Kamera für eine Aufnahme. In den folgenden Schritten wird keine 3D Punktwolke ausgewertet, dieser Task löst jedoch ebenso die Aufnahmen der Mono-Kameras aus.

- *Halcon Script ausführen*
  Die eigentliche Barcode-Erkennung findet in diesem Schritt statt. Das HDev-Script `mikado_barcode_detection.hdev` wird ausgeführt und kann auf Bilddaten der Ensenso zugreifen. Das Script selbst wird im nächsten Schritt erstellt.
- *Liste leer?*
  Das HDev-Script zur Barcode-Erkennung wird auf die Mikado-Variablen zugreifen und erkannte Barcodes als StringList in die Variable `dataCodeResultList` schreiben. Bleibt die Liste leer, so wurde kein Barcode erkannt. In diesem Fall springt der Ablauf zur zweiten Sequenz namens `Detection failed` und endet hier im Fehlerzustand. Enthält die Liste mindestens einen Eintrag, so findet kein Sequenzwechsel statt und der Ablauf fährt mit dem nächsten Task fort.
- *Kopiere nächsten Eintrag aus Liste*
  Kopiert den ersten Eintrag der Liste `dataCodeResultList` in die Variable `detectedProductNr`. Es wird also lediglich der erste erkannte Barcode ausgewertet.
- *Setze aktives Produkt*
  Führt den Produktwechsel durch. Hinweis: Existiert das Produkt nicht, wird an dieser Stelle ein Fehler ausgegeben und die Programmausführung bricht ab.
- *Programm beenden*
  Programm anhalten.

- *Sequenz: 'Detection failed'‘*
  Wird ausgeführt wenn die Barcode-Erkennung keine Barcodes findet.
  - *Fehler*
    Hält das Programm im Fehlerzustand an.

4.4.3 Hdev-Script für eine Erkennung von Datamatrizen / QR-Codes erstellen

Als nächstes widmen wir uns dem Erstellen des Halcon-Scripts, welches die Erkennung der verschiedenen Barcodes realisieren soll.

Hinweis: Alle Mikado-spezifischen Halcon-Operatoren werden in Kapitel *Halcon Script Task* erklärt.

Öffnen Sie hierzu HDevelop und erstellen ein neues HDev-Script. Speichern Sie die Datei unter dem Namen `mikado_barcode_detection.hdev` im Ordner `scripts` der Roboterzelle. Zu Beginn des Programms wird das rektifizierte Monobild der rechten Kamera der Ensenso eingelesen und in der Variable `Image2D` abgelegt (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')
```

Danach nutzen Sie den Operator `find_data_code_2d` um das Bild nach Datamatrizen zu durchsuchen (Z. 5 - 7). Die Standardparametrierung erfüllt hier ihren Zweck, kann aber natürlich auch durch Training optimiert oder werden. Zu diesem Thema finden Sie Informationen in der Halcon-Dokumentation.

Da die Produkte der Beispiel-Produktliste mit dem Präfix `Product_` beginnen, werden die dekodierten Artikelnummern entsprechend erweitert (Z. 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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
11 for Index := 0 to |decodedDataStrings|-1 by 1
12     decodedDataStrings[Index] := 'Product_' + decodedDataStrings[Index]
13 endfor
```

Da das Script ebenfalls QR Codes erkennen und dekodieren soll, starten wir die Suche erneut mit der entsprechenden Parametrierung (Z. 15 - 17).

```
15 create_data_code_2d_model('QR Code', [], [], DataCodeHandle)
16 find_data_code_2d(Image2D, qrXLDs, DataCodeHandle, [], [], resultHandles,␣
   ↪decodedQrStrings)
17 clear_data_code_2d_model(DataCodeHandle)
```

Nachdem die Ergebnisse zusammengefügt sind (Z. 19), werden die dekodierten Informationen als eine Liste von Strings an Mikado weitergegeben (Z. 20). Beim Erstellen des Mikado-Programmablaufs im obigen Schritt wurde die Variable ‚dataCodeResultList' für diesen Zweck erstellt.

```
19 decodedStrings := [decodedDataStrings, decodedQrStrings]
20 mikado_write_variable('dataCodeResultList', 'StringList', decodedStrings, [])
```

Als letzten Teil des Scripts können Sie noch ein Bild an Mikado weitergeben, das z.B. Ergebnisse oder Analyseinformationen anzeigen soll. In diesem Fall verwenden Sie eine Reihe an Operatoren, mit denen Sie die erkannten Umrisse der Barcodes farblich ins Bild einzeichnen (ab Z. 24). Datamatrizen werden grün, QR Codes werden orange umrahmt.

```
23 * Publish an image with results
24 get_image_size(Image2D, Width, Height)
25 open_window(0, 0, Width, Height, 0, 'buffer', '', WindowHandle)
26 disp_image(Image2D, WindowHandle)
27 set_line_width(WindowHandle, 5)
28 set_color(WindowHandle, 'green')
29 disp_xld(dataMatrixXLDs, WindowHandle)
30 set_color(WindowHandle, 'orange')
31 disp_xld(qrXLDs, WindowHandle)
32 dump_window_image(OutputImage, WindowHandle)
33 mikado_publish_2d_image(OutputImage)
```

Im Folgenden Screenshot nochmals das gesamte Script als Screenshot aus der HDevelop Umgebung.

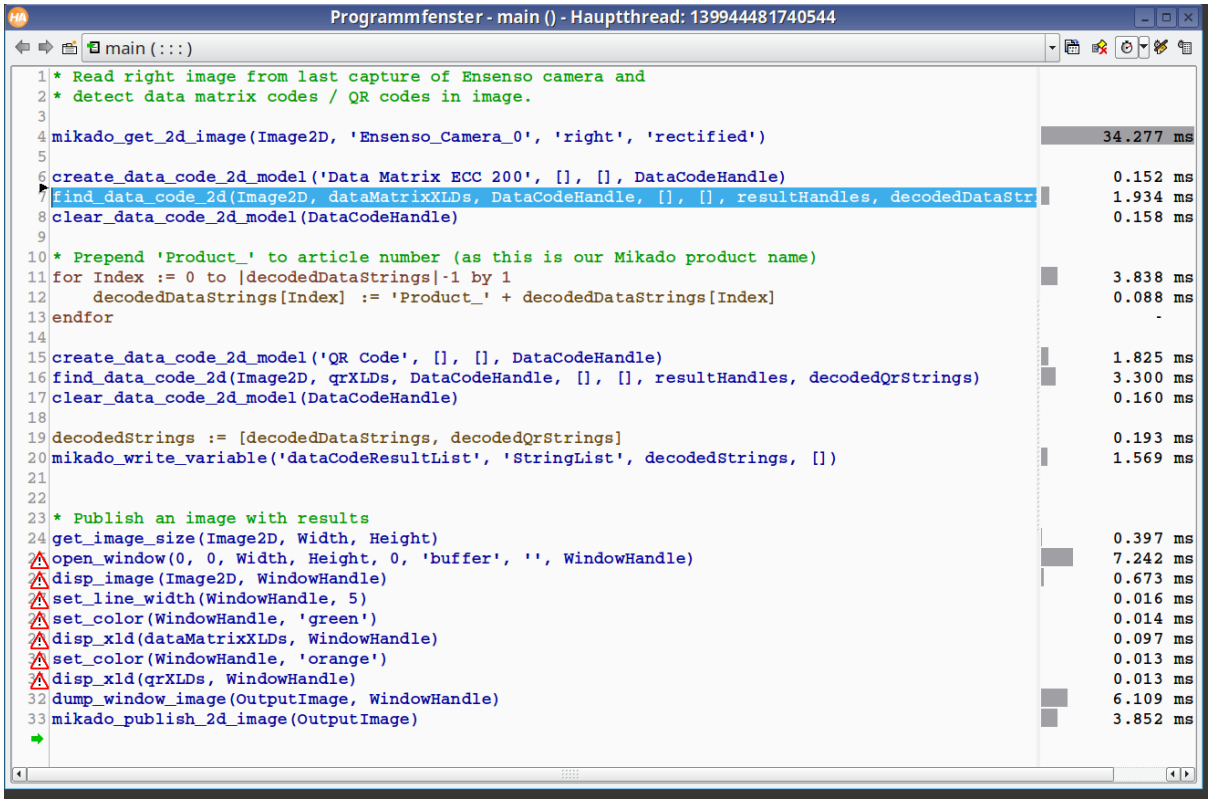


Abb. 30: Hdev-Script ‚mikado_barcode_detection.hdev‘

Hinweis: Einige Halcon Operatoren am Ende des Scripts sind mit einem roten Warnsymbol gekennzeichnet. Die Warnung betrifft die Ausführung des Scripts innerhalb der HDevelop-Umgebung. Da das Skript später jedoch in einer eigenen Hdev-Engine ausgeführt wird, können diese Warnungen ignoriert werden.

Tipp: Wenn Sie HDevelop direkt auf dem Mikado-Controller ausführen, so können Sie das Script auch direkt in HDevelop testen. Sie haben zudem die volle Unterstützung der Entwicklungsumgebung für die Mikado-Operatoren. Starten Sie hierfür Mikado einfach parallel zu HDevelop. Sowohl der Mikado-Programmablauf als auch die einzelnen Halcon-Operatoren können nun Schritt für Schritt ausgeführt werden. Dies hilft Ihnen bei der Entwicklung und beim Testen des HDev-Scripts.

4.4.4 Programm ausführen und testen

In den folgenden Screenshots ist das Ergebnis zu sehen. Wird der Ablauf ausgeführt während die Kamera auf eine der beiden Datamatrizen gerichtet ist, so wird das aktive Produkt erfolgreich auf die codierte Artikelnummer gewechselt.

Möchten Sie das zugehörige Bild mit eingezeichneten Resultaten anzeigen, so wechseln Sie vor dem Start des Programms unten rechts zum Reiter ‚Camera‘. Dort wählen Sie oben bei ‚Show Image:‘ in der Combobox den Eintrag ‚/mikado_hdevengine_node/halcon_picture‘ aus. Nun wird nach Ausführung Ihres Hdev-Scripts das aufgenommene Bild angezeigt und dekodierte Barcodes farblich eingerahmt.

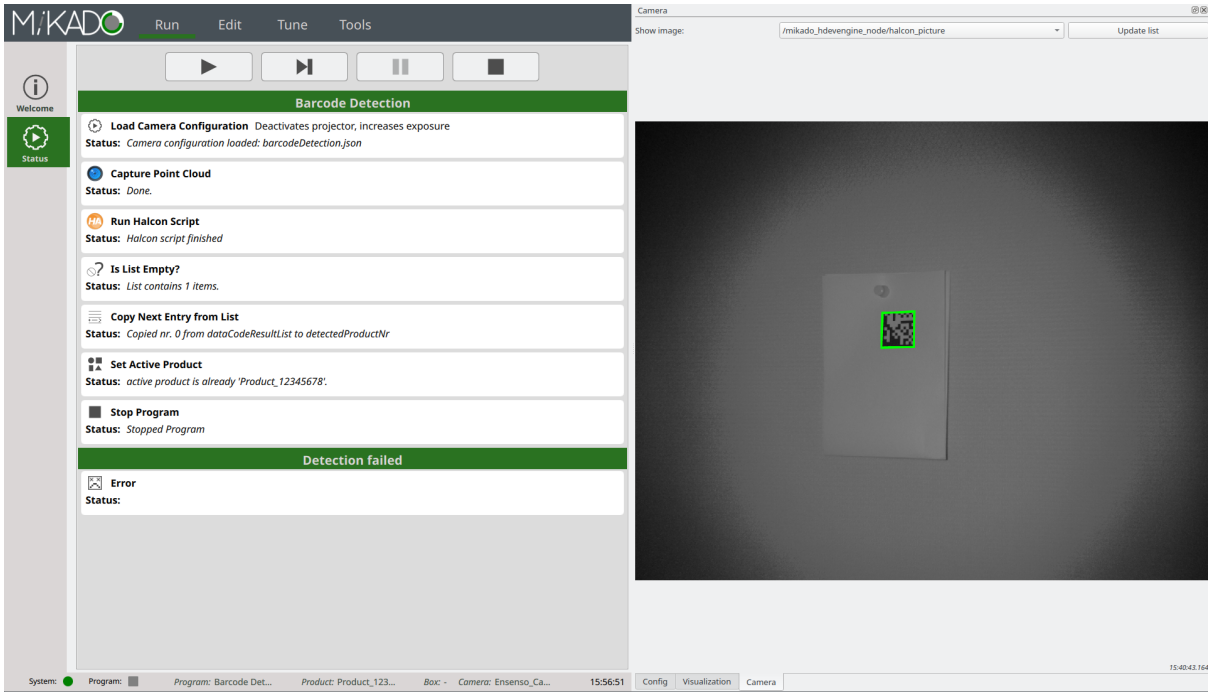


Abb. 31: Statusansicht bei Verwendung der Datamatrix des Artikels '12345678'

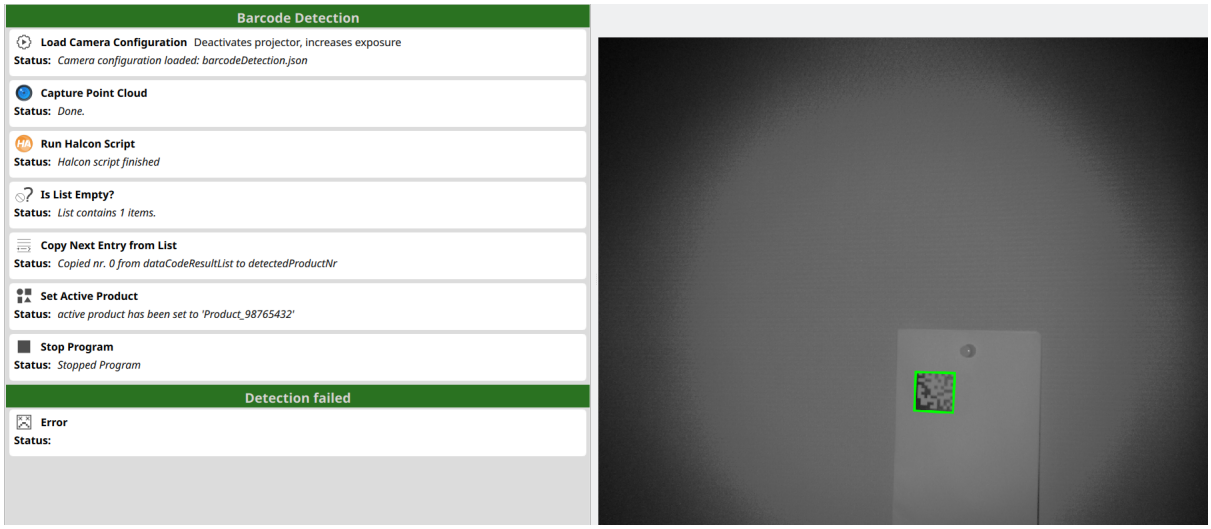


Abb. 32: Statusansicht bei Verwendung der Datamatrix des Artikels '98765432'

Ist die Kamera auf den QR Code gerichtet, enthält die dekodierte Information keine Artikelnummer, sondern eine URL zur Mikado Webseite. Da offensichtlich kein Produkt mit diesem Namen existiert, endet der Programmablauf wie im Screenshot dargestellt in einem Fehler.

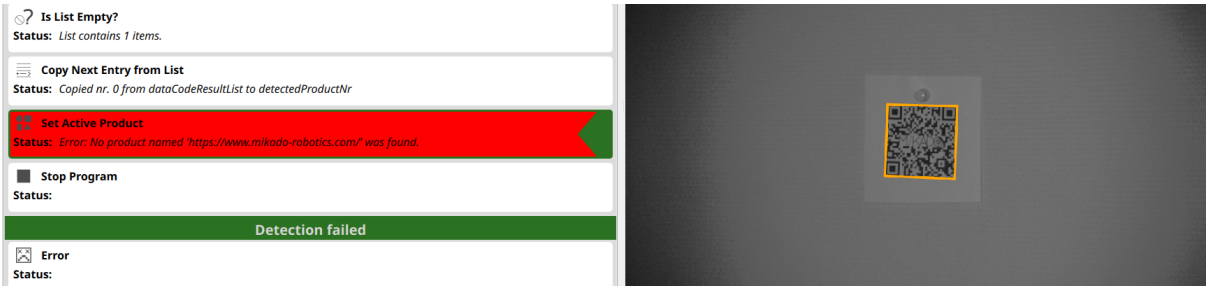


Abb. 33: Statusansicht bei Verwendung des Mikado QR Codes

4.5 Beispiel: Benutzerdefinierte Logdatei aus Mikado

Dieses Tutorial zeigt wie aus dem Mikado Programmablauf heraus Log Dateien im .csv Format geschrieben werden können. Anwendungsfälle können Tests zur statistischen Auswertung sein. Es wird der *JavaScript*-Task benutzt, dessen Schnittstelle auch im *Erweiterungs-Kapitel zum JavaScript Task* beschrieben ist.

Das Gesamtskript loggt zunächst eine Überschriftszeile und dann jeweils Datenzeilen. Dabei wird in jedem Programmdurchlauf einmal geloggt und an eine Logdatei angehängt. Die Logdatei lässt sich dann zum Beispiel mit herkömmlichen Tabelleneditoren importieren. Die einzelnen Skript Bausteine werden nachfolgend erklärt.

```
//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 Passendes Mikado Programm

Das Mikado Programm beinhaltet einen simplen Taskablauf innerhalb einer Sequenz:

- *Finde Objekte*: (Setzt fertig konfiguriertes Produkt voraus), zum Testen kann *fundPosen* auch eine statische, mit Dummy-Werten initialisierte Posenliste sein, dann wird dieser Task ausgelassen
- *Javascript*: Lädt und führt das besprochene Skript aus
- *Zahl erhöhen*: Die Variable *durchlaufN* erhöhen
- *Zahl größer als*: Vergleich der *durchlaufN* Variable mit der *maxN* Variable, Sprung zum Anfang, falls dies noch nicht erreicht ist, also im *Falsch*-Fall
- *Programm beenden* Task

4.5.2 Variablen des Mikado Programms

Im Mikado Programm müssen mehrere Variablen angelegt sein, auf denen das Skript basiert:

- *nameTestlauf* (String): Bezeichner dieses Durchlaufs, auch als Name der Zieldatei genutzt
- *ausgabeOrdner* (String): Absolutpfad zum Ausgabe Ordner, endet mit „/“, zB. „/home/mikado/“
- *durchlaufN* (Integer): Zähler der Logging Iteration
- *maxN* (Integer): Zähler Maximum, nach dem die Logging Iteration abgeschlossen werden soll
- *fundPosen* (PoseList): Liste, die Posen gefundener Objekte beinhaltet

4.5.3 Skript Details

Das Skript wird nun Schritt für Schritt erklärt. Gegliedert in initiale Aktionen, Operation und abschliessende Aktionen.

```
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 diesem initialen Block werden zunächst die Inhalte der Mikadovariablen gelesen. Daraus wird auch der Ausgabe-Dateipfad zusammengefügt.

```
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"
```

In diesem Block werden die „Rechenoperationen“ durchgeführt. Diese beinhalten in die Mikado Logs zu loggen, um Feedback über die Skriptausführung zu erhalten. Dann wird der String der Überschrift zusammengesetzt, per tertiärem Operator über *durchlaufN* aber nur im initialen Durchlauf. Unter dem Punkt „Auswertung“ wird die aktuelle Logzeile befüllt, die Werte jeweils durch Seperatoren getrennt. Der abschließende Zeilenumbruch mittels „n“ wird im nächsten Durchlauf weiteren Inhalt in die nachfolgende Zeile schreiben.

```
25 //Ausgabe in Datei
26 var append = true //überschreibt Datei, falls false
27 lab.writeStringToFile(ausgabeDateiFunde, ausgabeString, append)
```

Abschließend wird die Datei in das Dateisystem geschrieben. Die *append* Variable könnte in einer erweiterten Fassung auch beim ersten Mal *false* gesetzt werden, falls ein initiales Überschreiben der Logdatei gewünscht wäre.

4.6 Beispiel: Mikado PoseList sortieren

Dieses Beispiel zeigt wie eine Mikado Variable vom Typ *PoseList* nach bestimmten Kriterien sortiert werden kann. Ein möglicher Anwendungsfall ist beispielsweise das Ergebnis des *Finde Objekt* Task in Mikado nach der z-Koordinate zu sortieren, um oben liegende Teile bevorzugt zu greifen. Im hier dargestellte Beispiel wird deshalb exemplarisch eine Liste von Posen absteigend nach der z-Koordinate sortiert. Durch Modifikation des Beispiels können beliebige *PoseList* Variablen allerdings auch in anderer Form sortiert werden, z.B. aufsteigend statt absteigend oder nach der x- oder y-Koordinate sowie den Koordinatenwinkeln.

Achtung: Beim dargestellten Beispiel wird die originale unsortierte Mikado-Variable „ExamplePoseList“ mit dem sortierten Ergebnis überschrieben. Falls die originale Variable stattdessen beibehalten werden soll, ist ebenfalls eine Änderung notwendig.

```
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 }
```

Es wird die JavaScript-Methode Array *sort()* mit einer benutzerdefinierten Vergleichsfunktion zum Sortieren der Posenliste verwendet. Das Verhalten der Sortierung kann dementsprechend durch Modifikation der Vergleichsfunktion in Zeile 11 verändert werden. Zugriff auf die einzelnen Komponenten der Posen erhält man mit den Schlüsselwörtern `x`, `y`, `z`, `roll`, `pitch` und `yaw`. Im Folgenden sind exemplarisch einige alternative Möglichkeiten der Sortierung dargestellt:

- Sortieren nach der z-Koordinate aufsteigend

```
PoseList.sort((PoseA,PoseB) => PoseA.z - PoseB.z);
```

- Sortieren nach der x-Koordinate absteigend

```
PoseList.sort((PoseA,PoseB) => PoseB.x - PoseA.x);
```

- Sortieren nach dem Gier-Winkel absteigend

```
PoseList.sort((PoseA,PoseB) => PoseB.yaw - PoseA.yaw);
```

KAPITEL 5

Simulation

Nahezu alle Teilkomponenten von Mikado können entweder einzeln oder auch alle gleichzeitig in einen Simulationsmodus versetzt werden. So ist es z.B. möglich mit aufgezeichneten Bilder (Filecam) einen Ablauf mit einem simulierten Roboter im Büro zu optimieren. Die integrierte Physiksimulation ermöglicht die Befüllung von Behältern mit zu handhabenden Teilen auf Basis der Mesh-Modelle. Dadurch kann auf die vorherige Gewinnung von Kamerabildern vollständig verzichtet werden. Dies bietet sich beispielsweise an um verschiedene Greifervarianten zu evaluieren ohne die Greifer vorab fertigen zu lassen.

Hinweis: Aus den Ergebnissen der Simulation lassen sich nur sehr eingeschränkte Rückschlüsse auf die in der Realität erreichbare Taktzeiten ziehen. Die benötigten Beschleunigungswerte und Reaktionszeiten der Robotercontroller sind im Allgemeinen nicht zugänglich und auch von der spezifischen Konfiguration abhängig. Falls Sie eine Lizenz (oder Testversion) der Simulationssoftware ihres Roboterherstellers besitzen, kann Mikado natürlich auch damit betrieben werden, kontaktieren Sie uns für Details.

5.1 Roboter-Simulation

In der Konfigurationsdatei ‚config/manipulator.yaml' kann über die Variable ‚simulated_robot: true' in den Simulationsmodus gewechselt werden. Mikado wird nun keinen Kontakt mehr zum echten Roboter aufnehmen, sondern die Trajektorien offline abfahren.

5.2 Aufgezeichnete Bilddaten (Filecam)

Sie können sowohl manuell in der Anwendung Mikado, automatisiert aus einem Programmablauf heraus, oder auch in NxView Bilddaten abspeichern um sie später als Filecam wieder zu laden. Um solche Bilddaten zu betrachten starten Sie NxView, „Create…" → „Filecam". Wählen Sie dort den Ordner mit den abgespeicherten Bildern aus und notieren Sie sich den Namen der Kamera, automatisch ist dieser „File[Seriennummer]". Öffnen Sie ihre Zelle im Mikado Assistant und tragen Sie unter „Parameter" → „Kameras" den neuen Namen ein. Alternativ können Sie auch die Zeile „camera_serial_no: File[Seriennummer]" in der Datei „config/camera_0.yaml" anpassen. Die Filecam wird nun an Stelle der echten Kamera verwendet. Die Bilder werden bei jeder Aufnahme durch rotiert. Wenn die Bilder direkt von Mikado abgespeichert wurden, wird die Punktwolke auch an der richtigen Stelle im Raum angezeigt werden. Wenn die Filecam von NxView gespeichert wurde, dann funktioniert dies nur bei kalibrierten stationären Kameras.

Hinweis: Verwenden Sie bevorzugt die Schaltfläche „Letzte Aufnahme speichern“ unter „Tune“ → „Erkennung“ oder den *„Bild speichern“-Task* im Programmablauf. Nur so wird bei am Roboter montierten Kameras die aktuelle Position mit gespeichert. Sollten Sie dennoch Filecams in NxView erstellen, achten Sie bitte darauf diese nicht als ZIP zu speichern.

5.3 Simulation basierend auf Mesh-Dateien

Dieses Kapitel beschreibt das Zusatzprogramm *Mikado Simulator*, mit dem Sie Mikado synthetische Kamerabilder als Input geben können. Dazu wird in einer Physiksimulation eine Kiste mit Mesh-Dateien des Produkts gefüllt und anschließend werden mit einer sog. „virtuellen Kamera“ Bilder aufgenommen. Das Stereomatching und die Objekterkennung funktionieren analog zu echten Bilddaten.

Hinweis: Generell muss der Simulator nach Mikado gestartet werden. Sie finden dafür eine entsprechende separate Verknüpfung auf dem Desktop oder im Zellverzeichnis.

5.3.1 Simulator - Konfiguration

Diese Seite erreichen Sie über das Hauptmenü ‚Setup‘, Untermenü ‚General‘.

Den Simulator konfigurieren und einzelne Objekte hinzufügen und manipulieren.

Generelles Setup

Dieser Punkt erlaubt das Wiederherstellen eines vorherigen Simulator Zustandes. Dabei werden die folgenden Einstellungen geladen:

- Kistenobjekte mit Dropzone
- Produktobjekte mit individuellen Produkteigenschaften

Im jeweiligen Zellenordner können in der launch/simulator.launch folgende Parameter dauerhaft gesetzt werden:

- Kameraseriennummer und Eigenschaften
- Kamera fix oder beweglich
- Default Produktmeshdatei
- Automatikmodus: Default Parameter
- Automatikmodus: Default für „Starte sofort nach Simulatorstart“

Kamera Setup

Dieser Unterpunkt zeigt die derzeitig eingestellte Kameraseriennummer und ob diese fix oder beweglich ist. Der Wert für das Rauschlevel kann von 0 bis maximal 30 gewählt werden und wird nach Bestätigen angewendet. Während des Automatikmodus kann diese Einstellung nicht mehr angepasst werden.

Kistensetup

Bisher ist es möglich mit genau einer Kiste zu arbeiten.

Dieser Unterpunkt erlaubt die Parameter der aktiven Kiste anzupassen. Das bedeutet zum einen die Pose aus xyz Koordinaten und rpy Rotationswinkeln bestehend. Zum anderen kann für jede Kiste die Dimension und Höhe der Abwurfzone definiert werden. Deren Ausrichtung ist an die Orientierung der Kiste gebunden.

Produkt Setup

Dieser Unterpunkt erlaubt es das Produkt auszuwählen, das über der Kiste simuliert abgeworfen werden soll. Zur Definition des Produkts muss eine Meshdatei angegeben werden und ob es eine 180° Rotationssymmetrie um die x-Achse besitzt. Letzteres erlaubt es in der Auswertung auch um 180° verdrehte Erkennungen als korrekt zu klassifizieren. Mit der Eigenschaft ‚Reflektanz' kann als Materialeigenschaft simuliert werden, wie stark das Objekt das Projektorlicht der Kamera reflektieren soll. Um Teile hinzuzufügen muss eine entsprechende Anzahl gewählt werden. Mit ‚Hinzufügen' werden die Teile dann sowohl in Position als auch Rotation zufällig in der Abwurfzone platziert. Als Wahrscheinlichkeitsverteilung wird eine Gleichverteilung benutzt. Standardmäßig werden 20 Teile auf einmal simuliert. Werden mehr hinzugefügt erfolgt die Simulation in mehreren Tranchen.

Hinweis: Die Physiksimulation erfolgt über einen festgesetzten Zeitraum. Für große oder sehr geringe Abwurfhöhen kann der Parameter ‚simulate_total_duration' in der Zelldefinition unter launch/simulator.launch angepasst werden.

Hinweis: Die Anzahl der simulierten Teile sollte so gewählt sein, dass anfangs keine oder nur wenige Kollisionen beim zufälligen Positionieren in der Abwurfzone passieren.

Hinweis: Konvexe Teile können teilweise in der Physiksimulation zu Kollisionen von Objekten und Kiste führen.

Objekte

Diese Statusanzeige stellt alle derzeitig im System befindlichen Objekte dar. Erkannte Objekte haben einen fettgedruckten Namen, erfolgreich gegriffene Objekte haben einen durchgestrichenen Namen. Ein Klick auf ein einzelnes Objekt markiert dieses farblich in der Visualisierung und zeigt dessen Details, wie Position, Orientierung, Symmetrie- und Reflektanzeigenschaften. Die Pose des Objekts kann hier über die Eingabefelder angepasst und per Klick auf ‚Anwenden' übernommen werden. Solche Veränderungen der Pose werden nicht mit der Physiksimulation geprüft. Bei einer manuellen Anpassung einer Objektposition sollte darauf geachtet werden, dass im erstellten Szenario keine Überlappungen zwischen Objekten, Behälter oder Szenenobjekte entstanden sind.

5.3.2 Simulator - Automatischer Modus

Diese Seite erreichen Sie über das Hauptmenü ‚Run', Untermenü ‚Automatik'.

Der Simulator kann als Dauertester im automatischen Modus verwendet werden. Dabei werden mehrere simulierte Füllungen der Kiste durch Mikado abgearbeitet und darüber eine Statistik erzeugt.

Ablauf

Es werden mehrere Testdurchläufe, sog. *Füllungen*, durchgeführt. In jedem Durchlauf wird eine Kiste mit der angegebenen Teilanzahl gefüllt und Mikado zurückgesetzt und neu gestartet. Dazu müssen die folgenden Parameter gesetzt werden:

- ***Kistenfüllungen*:**
  Die Anzahl an Durchläufe, für die die Kiste jeweils neu gefüllt wird.
- ***Objekte pro Füllung*:**
  Die Anzahl an Objekten, die pro Füllung in die Kiste abgeworfen wird.
- ***Erlaubter Winkelfehler*:**
  Jede Objekterkennung von Mikado wird nur dann akzeptiert, wenn der absolute Winkelfehler unterhalb dieser Grenze liegt. Nicht akzeptierte Objekte können nicht gegriffen werden und gelten als Fehlgriff, falls Mikado versucht diese zu greifen.
- ***Erlaubter Distanzfehler*:**
  Wie *Erlaubter Winkelfehler*, nur beschreibt dieser Parameter die maximal erlaubte absolute Fehlerdistanz, um die die Objekterkennung daneben liegen darf.
- ***Erlaubte Fehlgriffe in Folge*:**
  Wie viele Objekte nacheinander nicht erfolgreich gegriffen werden dürfen, bis dieser Durchlauf abgebrochen und als gescheitert verbucht wird.
- ***Erlaubte Restmenge Objekte*:**
  Sollte Mikado mit einem Fehler stoppen, wird kontrolliert ob diese maximal erlaubte Restmenge an verbliebenen Teilen in der Kiste untertroffen ist. Falls ja, wird der Durchlauf beendet und als Erfolg verbucht.
- ***Pausieren bei Mikadofehler*:**
  Falls aktiv, unterbricht der Simulator den Dauerbetrieb, sollte in Mikado ein Fehlerzustand erreicht werden, bevor der gewünschte Entleerungsgrad erreicht ist. Der Dauerbetrieb kann per Klick auf das ‚Fortsetzen'-Symbol wieder aufgenommen werden. Diese Option erlaubt problematische Situationen in Mikado genauer zu analysieren. Ebenfalls können die Objektpositionen während der Unterbrechung gespeichert werden, um sie später zu reproduzieren.

Auswertung

Der Simulator gibt sowohl die Statistik über die derzeitig zu leerende Füllung, als auch über die Gesamtheit des derzeitigen Tests an. Die Gesamtstatistik bezieht sich dabei immer auf alle bisher beendeten Durchläufe.

Einzelne Werte werden im Folgenden genauer beschrieben:

- ***Entleerungsgrad*:**
  Beim Ende (Erfolgreich/Abbruch bei Fehler) eines jeden Durchlaufs wird der Anteil der entnommenen Teile zu der Gesamtzahl Teile pro Füllung gespeichert. Der *Entleerungsgrad* entspricht dem Durchschnitt über alle Durchläufe.
- ***Durchschnittliche Zeit pro Griff*:**
  Durchschnittszeit des Verhältnisses Gesamtzeit eines Durchlaufes durch die vom Roboter durchgeführten Picks während des Durchlaufes. Diese Größe ist unabhängig vom Erfolg der Einzelnen Griffe.
- ***Durchschnittliche Zeit pro Entnahme*:**
  Wie . . . *pro Griff*, allerdings gelten hier nur erfolgreiche Griffe. Diese Größe entspricht also einer Kombination aus Greif-/Erkennungssicherheit und durchschnittlicher Taktzeit.

5.3.3 Simulator - Anpassung der Kameraeinstellungen mit Szene aus Simulator

Ist noch kein Roboter und keine Kamera vorhanden und man möchte eine Simulation anfertigen, ist häufig eine Anpassung der Kameraeinstellungen nötig. Im folgenden Abschnitt werden die Schitte zur Erstellung der neuen Kameraeinstellungen aufgezeigt.

Vorgehensweise

1. Erstellung Zelle: Als Beispiel dient eine simulierte Zelle, bei der eine Kiste mit Kurbelwellen gefüllt ist.

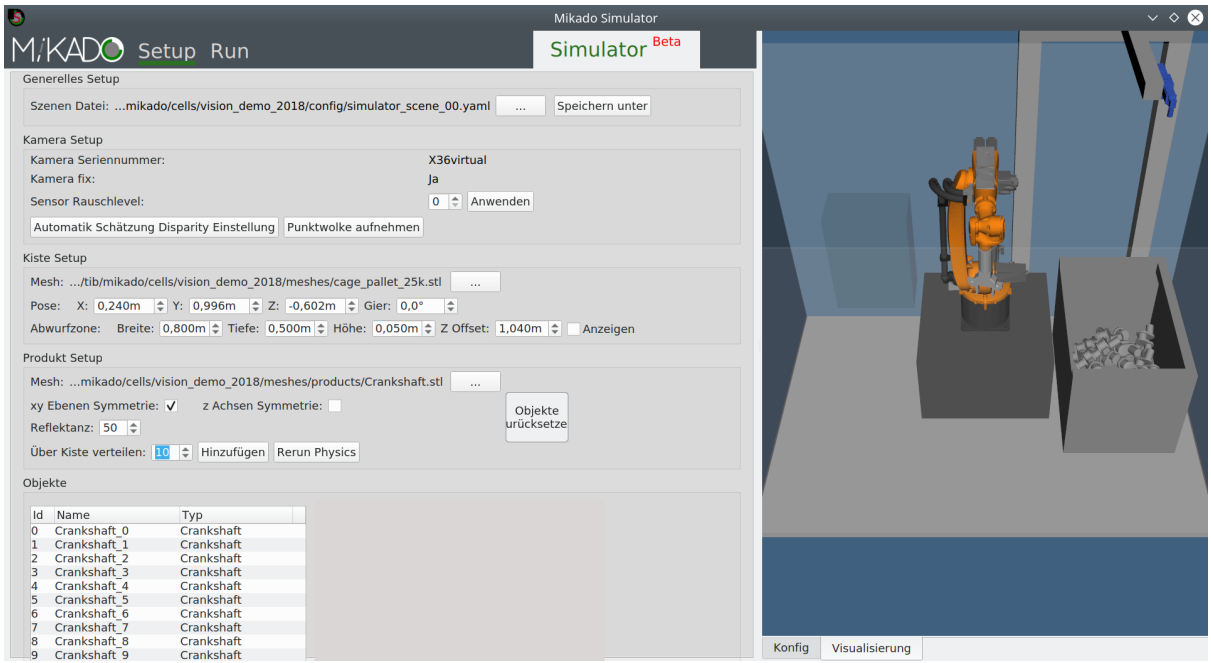


Abb. 1: Screenshot aus dem Mikado Simulator

2. Öffnen NxTree: Die Szene aus dem Simulator kann in NxTree geöffnet werden. Hierfür muss Mikado und der Simulator weiterhin geöffnet bleiben. Nach dem Starten von NxTree öffnet sich folgendes Fenster. Die Auswahl sollte wie im folgenden Fenster getroffen werden.

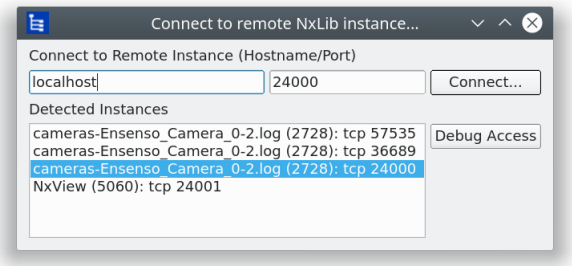


Abb. 2: Startfenster NxTree

3. Öffnen NxView: Im Anschluss muss die einzustellende Kamera in NxView geöffnet werden.

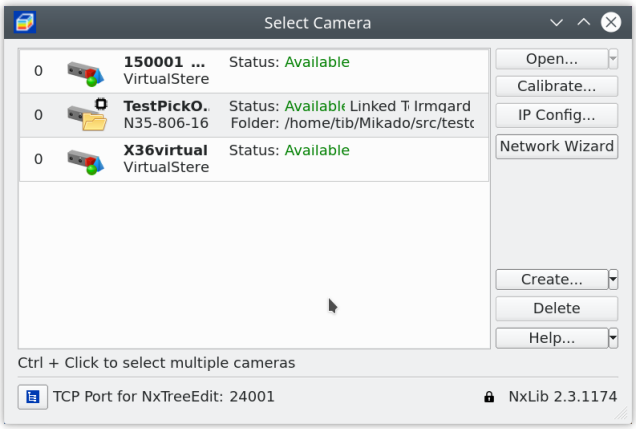


Abb. 3: Auswahl der einzustellenden Kamera in NxView

4. Öffnen NxTree in NxView: Hierbei wird eine beispielhaftes Szene in NxView dargestellt. Um die Szene des Simulators zu übertragen, muss in NxView NxTree gestartet werden. In der folgenden Abbildung ist dieser Schritt zu sehen.

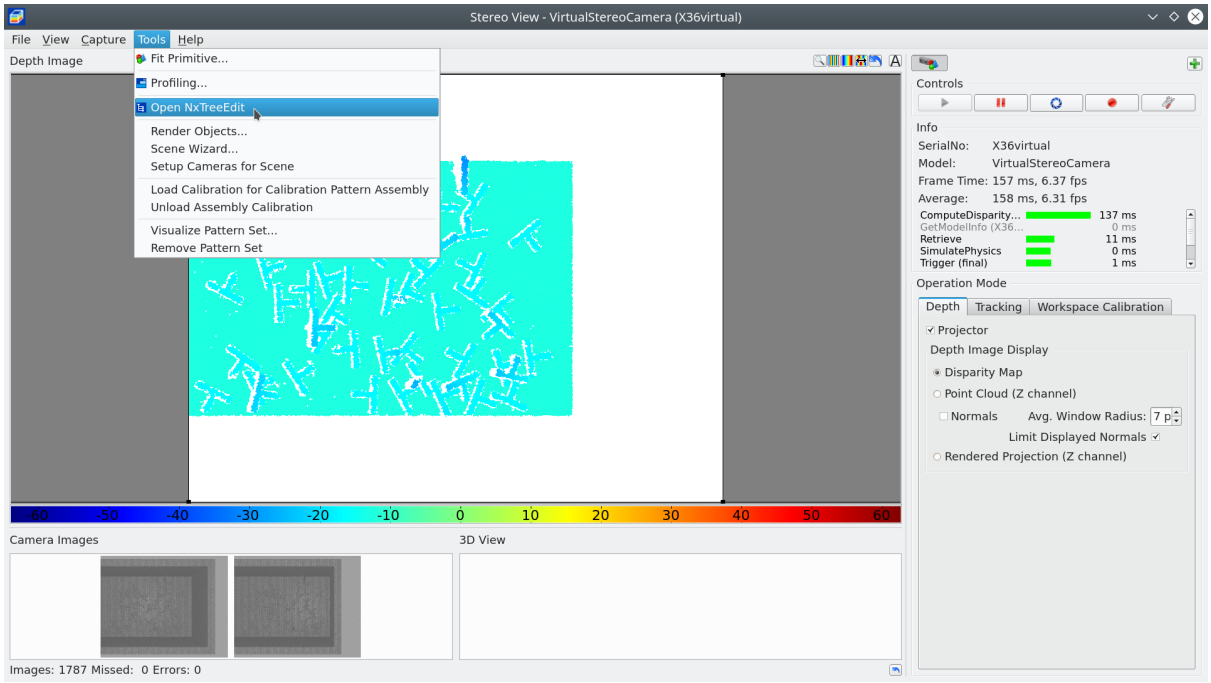
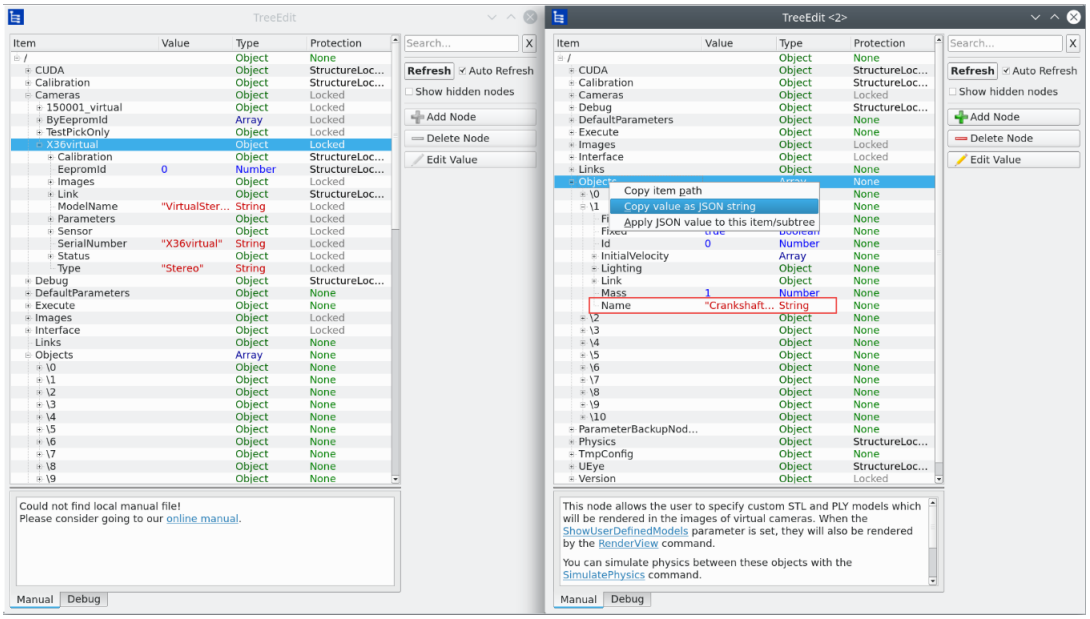
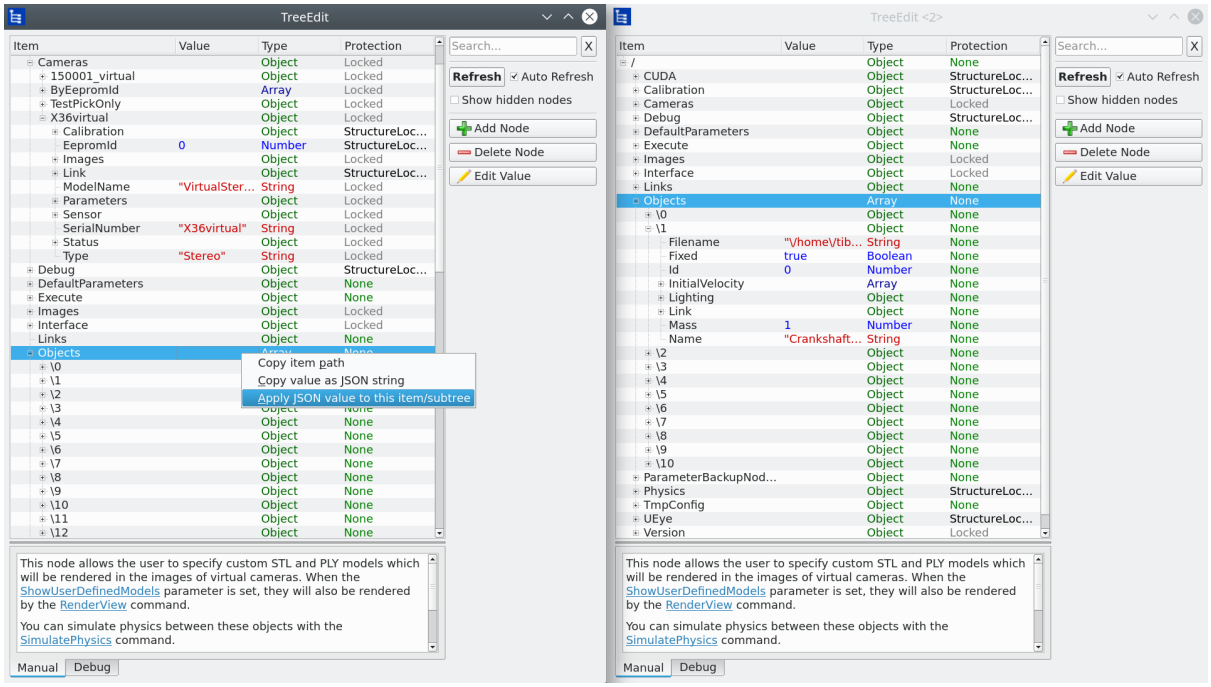


Abb. 4: Screenshot NxView

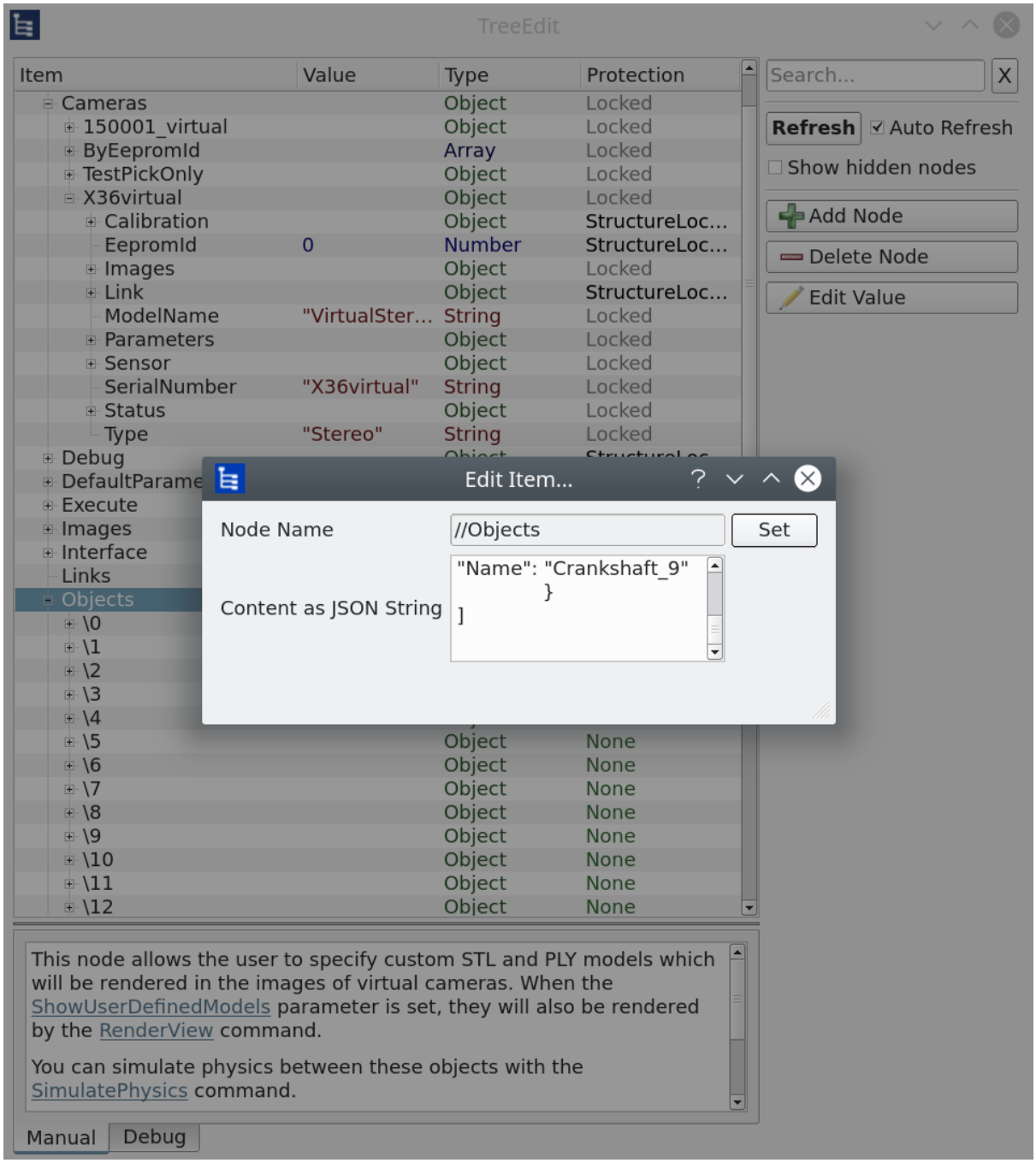
5. Kopieren der Elemente: In der folgenden Abbildungen ist NxTree der Szene in NxView (links) und der Szene im Simulator (rechts) zu sehen. Falls die Orientierung verloren geht, zu welcher Szene welches Fenster gehört, können beispielsweise die Objekte überprüft werden. In der Szene des Simulators (rechts) wurden Kurbelwellen (Englisch: Crankshaft) in die Kiste geworfen. Diese sind im rechten Fenster unter Objekte zu finden (siehe rotes Rechteck). Somit gehört dieses Fenster zum Simulator. Um diese Objekte zu übertragen müssen diese, wie in den folgenden Abbildungen dargestellt, kopiert und eingefügt werden. Der Befehl **Copy value as JSON string** erscheint, nachdem **Objects** mit der rechten Maustaste angeklickt wurde.



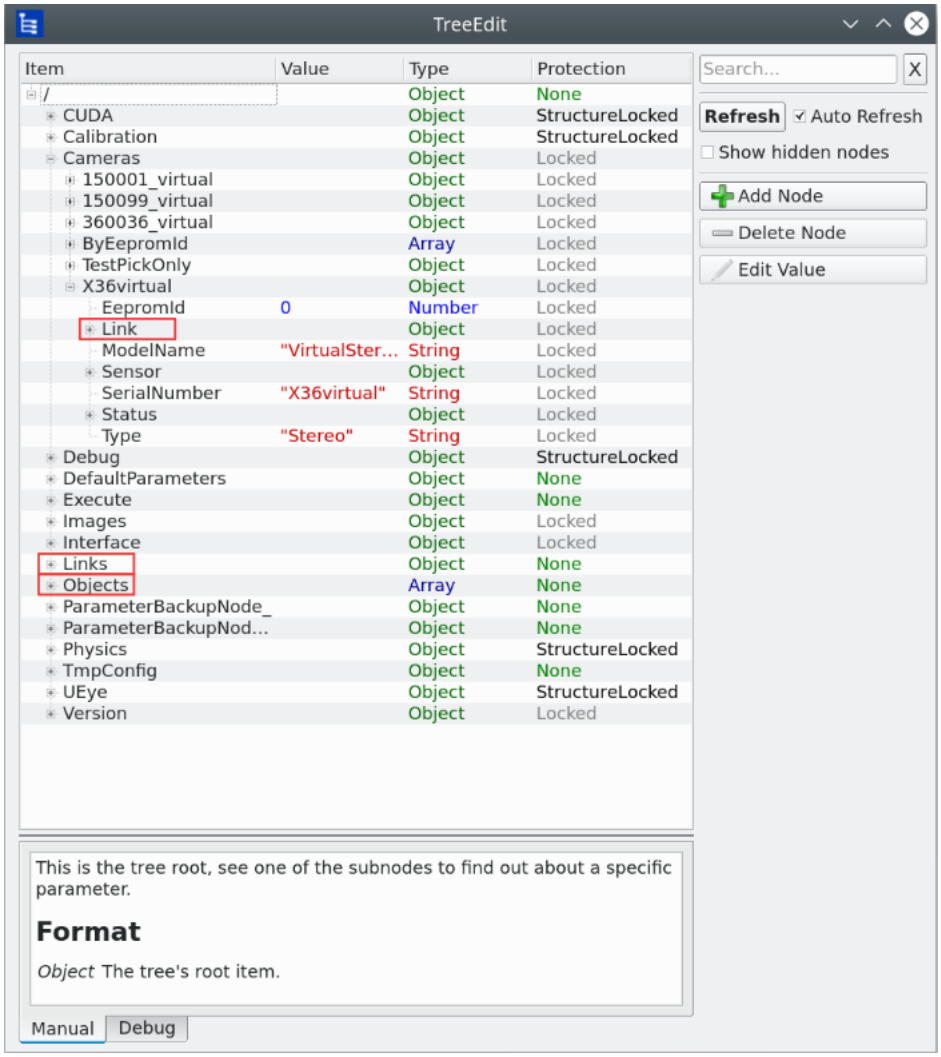
Die Objekte müssen anschließend in die zu NxView zugehörige Szene eingefügt werden (linkes Fenster). Hierfür muss mit der rechten Maustaste auf **Objects** geklickt und **Apply JSON value to this item/subtree** ausgewählt werden.



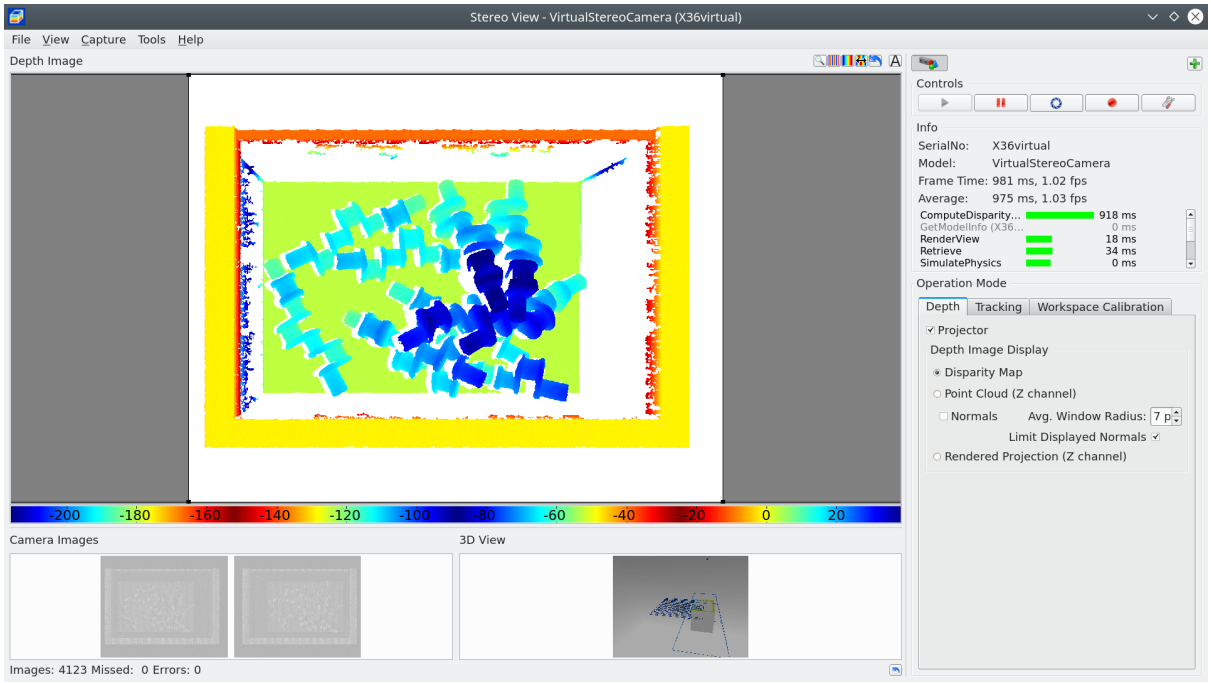
Im sich daraufhin öffnenden Fenster muss das Feld **Content as JSON String** befüllt werden. Hierfür auf das Feld mit der rechten Maustaste klicken und dann auf **Einfügen**. Anschließend ist der Inhalt im Feld zu sehen.



Dieses Vorgehen für **Objects** muss analog für **Links** und den **Link** der entsprechenden Kamera durchgeführt werden (siehe rote Rechtecke).



6. Anpassen der Kameraeinstellungen: Beim Schließen von NxTree ist nun die Szene aus dem Simulator zu sehen. Nun können die Parameter der Kamera so angepasst werden, dass die Kurbelwellen möglichst gut erkannt werden.


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

KAPITEL 6

Roboter-Modelle

Mikado erlaubt die Verwendung von Robotern vieler verschiedener Hersteller. In dieser Sektion ist aufgeführt wie die Robotercontroller der einzelnen Hersteller zu konfigurieren sind, damit diese problemlos mit Mikado kommunizieren.

Roboter verschiedener Hersteller unterscheiden sich in ihren Eigenschaften und Zusatzfunktionen. In den jeweiligen Abschnitten wird erläutert, welche Möglichkeiten die jeweiligen Roboter-Modelle und -Hersteller in Mikado bieten.

Parameter oder Steuerelemente, die modellabhängig sind und über die Oberfläche dargestellt werden, sind auf der Seite *Robotersteuerung* eingebunden.

6.1 ABB Controller einrichten

- *Systemanforderungen*
- *Netzwerkkonfiguration*
- *Installation der Programme*
- *Einrichtung der Programme*
- *Updaten der Roboterprogramme*
- *Roboter steuern*
- *Greifer und IOs einrichten mit Robot Web Services*
- *Kontrollabgabe*
- *Troubleshooting*

6.1.1 Systemanforderungen

Der roboterseitige Teil der ABB-Anbindung wurde in RAPID geschrieben und verwendet sowohl parallel laufende Tasks als auch Sockets. Es werden die folgenden Optionen benötigt:

- 623-1: Multitasking
- 616-1: PC Interface

Das **RobotWare** OS muss mindestens die Version 5.13 haben.

Zur Einrichtung wird **RobotStudio** benötigt. Es ist in einer kostenlosen Basic Variante erhältlich und wird in der folgenden Anleitung vorrausgesetzt.

Für die direkte Ansteuerung von Greifern und IOs über den Roboter wird Robot Web Services (RWS) benötigt.

RWS 1.0 ist in RobotWare OS 6.x enthalten.

RWS 2.0 ist ab RobotWare OS 7.0 installiert. Ältere Mikado Systeme vor Version 3.3 benötigen ggf. ein Update um über RWS 2.0 zu kommunizieren.

Alternativ können Greifer und IOs auch über Profinet oder andere Feldbusse gesteuert werden.

6.1.2 Netzwerkkonfiguration

Für die Kommunikation zwischen Roboter Controller und Mikado Controller wird eine Netzwerkverbindung zwischen den beiden Systemen benötigt. Verbinden Sie hierfür zunächst einen Ethernet Port des ABB Controllers mit einem Ethernet Port des Mikado Controllers.

Hinweis: Verwenden Sie auf ABB Seite unbedingt den WAN-Port, falls Ihre Steuerung über mehrere WAN-Ports verfügt, verwenden Sie WAN1.

Anschließend muss sowohl für den Mikado Controller als auch für den Roboter Controller eine feste IP-Adresse und eine zugehörige Subnetzmaske eingestellt werden. Die Netzwerkkonfiguration des Mikado Controllers wird in Sektion *Netzwerkadapter* näher erläutert. Auf dem ABB kann die IP-Adresse des öffentlichen Netzwerks unter `Settings` → `Network` → `Public Network` angepasst werden (default 192.168.3.2). Bitte entnehmen Sie weitere Hinweise zur Netzwerkkonfiguration des Controllers dem Benutzerhandbuch des Herstellers.

Hinweis: Beachten Sie, dass die IP-Adressen von Mikado Controller und ABB Controller im selben Adressraum liegen und die Subnetzmasken übereinstimmen.

Tragen Sie anschließend im **Mikado Assistant** die IP-Adresse des Roboter Controllers unter *Roboter* → *Verbindung* ein und speichern Sie die Einstellungen.

Eine Beschreibung des Verbindungstests findet sich in *Verbindungsprobleme*.

6.1.3 Installation der Programme

Damit der ABB Roboter von Mikado gesteuert werden kann müssen entsprechende Programme auf den Roboter Controller kopiert und dort eingerichtet werden. Sie finden die benötigten RAPID-Programme unter

`/opt/isys-vision/mikado/robot_programs/abb/rapid`

oder direkt über den entsprechenden Button im Mikado Assistant:

`Roboter` → `ABB-Controller einrichten` → `Verzeichnis öffnen`

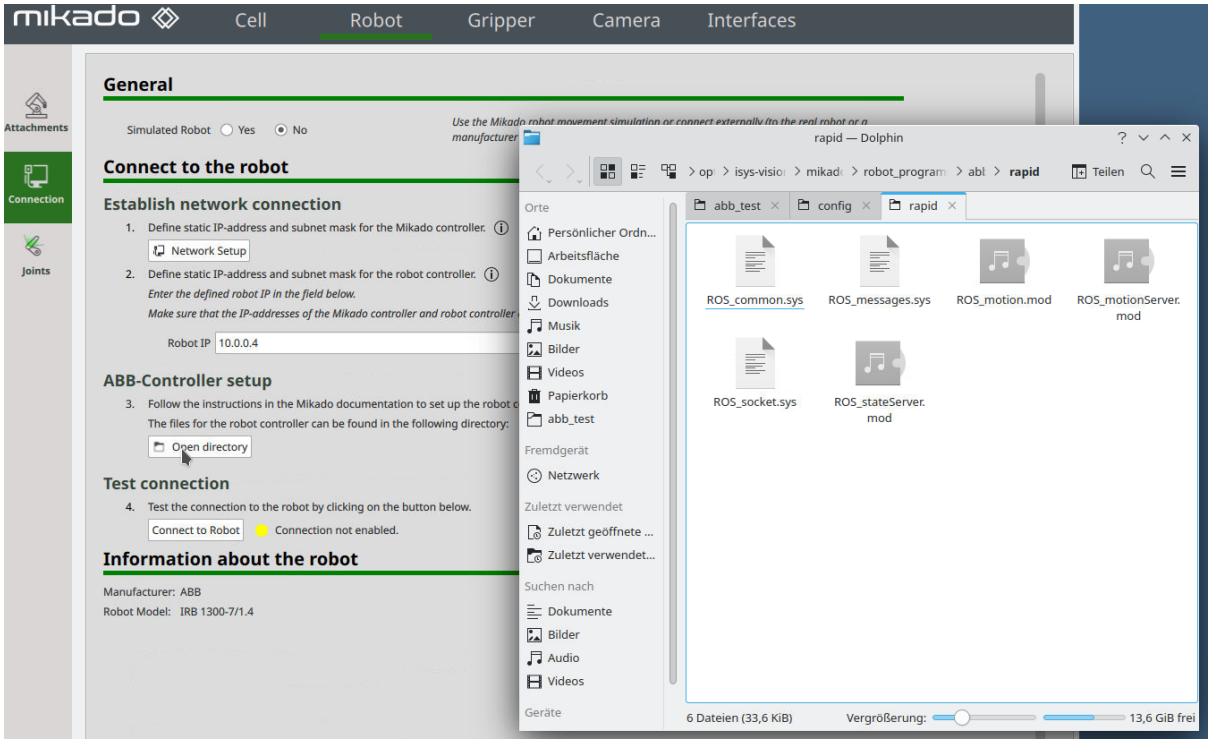


Abb. 1: Mikado Rapid Dateien auf dem Mikado Controller

Kopieren Sie die Dateien auf einen USB-Stick, stecken Sie diesen in den Controller beziehungsweise das Teach Pendant und legen Sie auf dem Roboter Controller einen neuen Ordner namens `ROS` an (z.B. unter `HOME`).

Kopieren Sie anschließend alle Dateien vom USB-Stick dort hin.

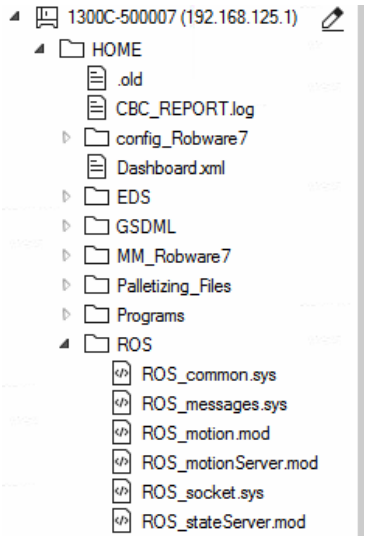


Abb. 2: Mikado Rapid Dateien in RobotStudio

Im folgenden Abschnitt wird die Einrichtung dieser Programme auf dem Roboter Controller genauer erläutert.

6.1.4 Einrichtung der Programme

Der Treiber besteht aus drei parallel laufenden Programmen:

- ROS_stateServer.mod Überträgt die momentane Gelenkstellung und den Zustand des Roboters
- ROS_motionServer.mod Empfängt Trajektorien von Mikado
- ROS_motion.mod Führt die empfangenen Trajektorien aus

Für diese drei Programme müssen nun eigene Tasks anlegt werden. Gehen Sie dazu auf `Controller` → `Configuration` → `Controller` → `Task`. Mit einem Rechtsklick und `New Task` können Sie nun die benötigten Tasks ergänzen:

Name	Type	TrustLevel	Motion Task
ROS_StateServer	Semistatic	NoSafety	No
ROS_MotionServer	Semistatic	SysStop	No
T_ROB1	Normal		Yes

Falls der Task `T_ROB1` bereits bei Ihnen angelegt ist und in diesem Bewegungskontrollmodule angelegt wurden, benennen Sie in `ROS_motion` die `main()`-Funktion in `ROS_main()` um, und ändern Sie den Einstiegspunkt zu `ROS_main()`.

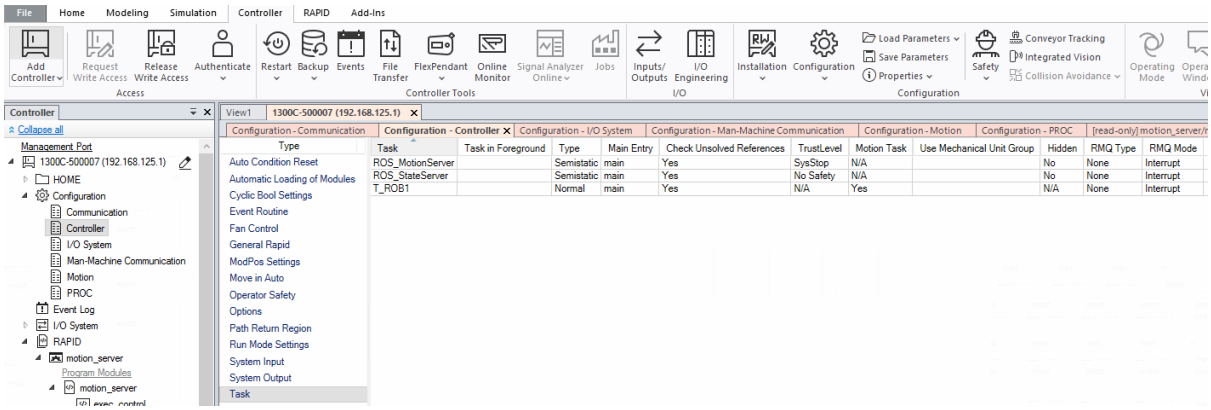


Abb. 3: Anzulegende Tasks in RobotStudio

Falls Sie in der Einrichtungsphase Änderungen am `ROS_StateServer` oder dem `ROS_MotionServer` durchführen möchten, stellen Sie diese zunächst auf den Task-Typen `Normal`. Für den Dauerbetrieb müssen beide Module aber auf `Semistatic` geschaltet werden.

Nun müssen einige Signale angelegt werden, gehen Sie dazu auf `I/O System` → `Signal` und ergänzen Sie durch Rechtsklick und `New Signal` die folgenden Signale:

Name	Type
signalExecutionError	Digital Output
signalMotionPossible	Digital Output
signalMotorOn	Digital Output
signalRobotActive	Digital Output
signalRobotEStop	Digital Output
signalRobotNotMoving	Digital Output
signalRosMotionTaskExecuting	Digital Output

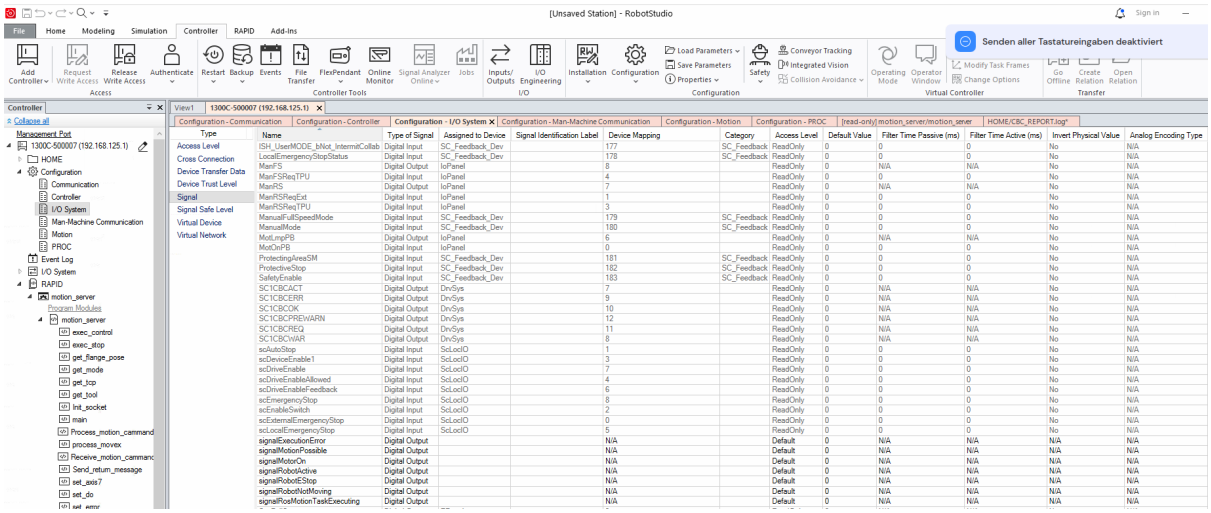


Abb. 4: Anzulegende Signale in RobotStudio

Diese müssen nun mit den Systemausgängen verbunden werden, gehen Sie dazu auf `Controller` → `System Output` und ergänzen Sie durch Rechtsklick und `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

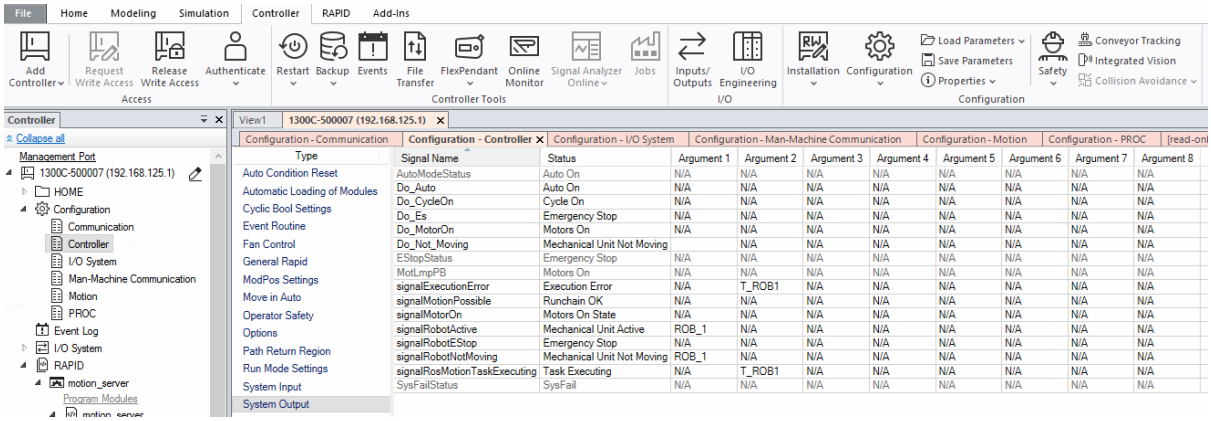


Abb. 5: Systemausgangssignale in RobotStudio

Nun muss noch der Autostart der Programme aktiviert werden, gehen Sie hierzu auf `Controller` → `Automatic Loading of Modules` und ergänzen Sie durch Rechtsklick und `New Automatic Loading of Modules`:

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

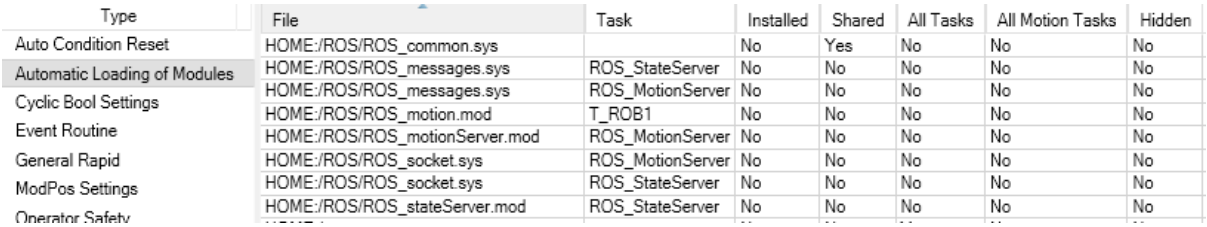


Abb. 6: Automatisches Laden der Module in RobotStudio

Nach einem Neustart des Systems (idealerweise P-Start) kann der Roboter nun von Mikado gesteuert werden.

6.1.5 Updaten der Roboterprogramme

Falls Sie die Roboterprogramme durch Programme einer anderen Version ersetzen möchten, laden Sie die Dateien der neuen Version wie in Punkt *Installation der Programme* beschrieben auf den Roboter und führen Sie einen P-Start durch.

`ABB` → `Restart` → `Advanced` → `P-Start` → `OK`

6.1.6 Roboter steuern

Um den Roboter mit Mikado zu steuern, auf dem Flex Pendant:

- Automatikmodus wählen
- Motoren einschalten
- Programmzeiger zurücksetzen
- Start

Der Roboter Status sollte in Mikado bei korrekter Installation wie folgt aussehen:

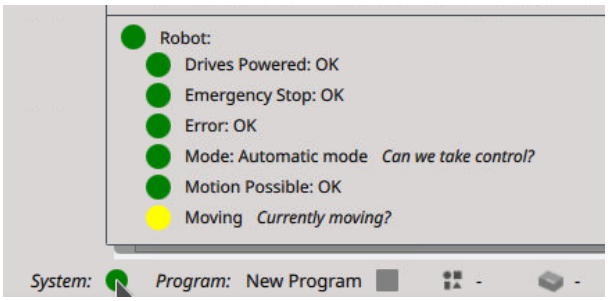


Abb. 7: Anzeige des Roboterstatus bei korrekter Installation

Bei Status `No Motion Possible` in Mikado: hat ein Rechner mit RobotStudio noch die Kontrolle über den Roboter? Diese muss abgegeben werden damit Mikado die Kontrolle übernehmen kann.

Bei Status `Timeout` in Mikado:

- Ist der Roboter pingbar?
  - `ping roboter-ip` in Konsole
- Laufen die drei Tasks ROS_StateServer, ROS_MotionServer und T_ROB1 auf dem Roboter?
  - Ggf. Motoren einschalten, Programmzeiger zurücksetzen und Start

6.1.7 Greifer und IOs einrichten mit Robot Web Services

Für das Ansteuern einfacher Greifer sowie für das Setzen von IOs kann ABBs Robot Web Service (RWS) verwendet werden.

Ab Mikado Assistant Version 3.3.1 kann die RWS Version bei der Konfiguration der Roboterverbindung ausgewählt werden.

Passen Sie bei Bedarf in der Datei *config/manipulator.yaml* ihrer Zelle die folgenden Daten an:

```
rws_username: Default User   #default
rws_password: robotics       #default
rws_digital_in_path: MIK_DI
rws_digital_out_path: MIK_DO
rws_version: 2
```

Wir empfehlen Benutzername und Passwort auf dem Controller zu ändern. Falls Benutzername und/oder Passwort geändert wurden, müssen die entsprechenden Angaben *rws_username* und *rws_password* entsprechend angepasst werden.

Mikado verwendet intern Zahlen für die einzelnen Ein- und Ausgänge, diese werden an den Präfix *rws_digital_in_path* angehängt, wenn also in Mikado Eingang *23* gelesen wird, wird auf dem ABB IRC in diesem Beispiel *MIK_DI23* ausgelesen. Analog für die Outputs: ein Mikado Schreibvorgang auf Ausgang 12 wird auf den ABB Output *MIK_DO12* abgebildet.

Wenn RWS (Robot Web Service) für den Zugriff auf Roboter IOs verwendet wird, muss das Sicherheitslevel in RobotStudio angepasst werden, da mit dem Sicherheitslevel `Remote Client in Auto Mode` standardmäßig auf IOs mit der Einstellung `Default` nur lesend zugegriffen werden kann.

Durch ein Mapping können wir einzelne IOs freischalten:

- Configuration > I/O System > Signal
- Rechtsklick > New Signal ...
- Neuen IO anlegen mit dem gewünschten Namen (Wichtig für Mikado: Prefix+Nummer, zb *MIK_DO1*)
- Type of Signal: Digital Output
- Assigned to Device: leer lassen
- Access Level: All (erlaubt Remotezugriff)

Dann kann eine Querverbindung angelegt werden:

- Configuration > I/O System > Cross Connection
- Rechtsklick > New cross connection ...
- Name nach Wahl
- Resultant: Name des „echten"/physikalischen IOs (z.B. *ABB_0_DO1* oder Profinet IOs)

Die IO-Nummern zur Greiferansteuerung können dann im Mikado Assistant eingetragen werden.

6.1.8 Kontrollabgabe

Es ist möglich, aus Mikado die Kontrolle über den Roboter abzugeben um andere Tasks/Bewegungen direkt auf dem Roboter durchzuführen und anschließend die Roboterkontrolle wieder an Mikado zu übergeben.

Dafür sind sowohl Anpassungen im Mikado-Programmablauf als auch im Roboterprogramm `ROS_motion` notwendig. Für die Übergabe wird außerdem ein Flag benötigt, das sowohl von Mikado als auch auf dem Roboter gelesen und geschrieben werden kann. In diesem Beispiel wird dafür der Ausgang *MIK_DO3* verwendet.

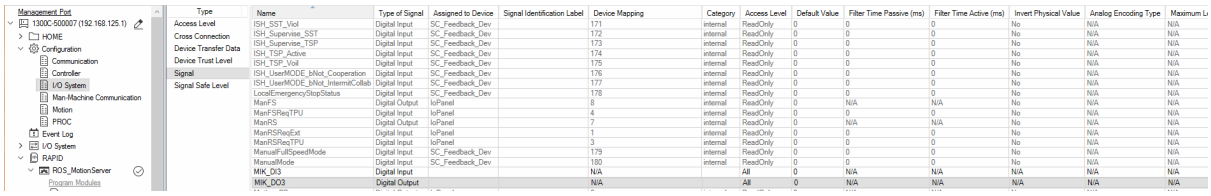


Abb. 8: Ansicht des im Beispiel verwendeten digitalen Ausgangs *MIK_DO3* in RobotStudio

In Ihrem Mikado-Programm müssen Sie nun die folgenden Änderungen vornehmen:

- Fügen Sie an der Stelle im Mikado-Programm, an der Sie die Kontrolle übergeben wollen, einen *Setze Roboter-Ausgangssignal*-Task ein. Geben Sie dem Task die dafür vorgesehene Ausgangsnummer und eine Variable mit dem Wert *true* als Argumente.
- Legen Sie eine neue Sequenz *Warten auf Roboterkontrolle* an.
- Fügen Sie unter dem *Setze Roboter-Ausgangssignal*-Task einen Sequenzwechsel ein, der zur gerade angelegten Sequenz wechselt.
- Die Sequenz *Warten auf Roboterkontrolle* sollte in einer Schleife immer wieder diesen Roboterausgang lesen. Fügen Sie dafür einen *Roboterausgang lesen*-Task ein, lesen Sie denselben Ausgang wie weiter oben und schreiben Sie das Ergebnis in eine Variable.
- Fügen Sie jetzt eine *IF*-Abfrage ein, die die Ergebnisvariable überprüft. Wenn sie *false* ist, bedeutet das, dass der Roboter die Kontrolle wieder an Mikado abgegeben hat. In diesem Fall kann mit einem Sequenzsprung mit dem eigentlichen Ablauf fortgefahren werden, nachdem die Roboterposition aktualisiert wurde (wie im Screenshot mittels *Roboterposition lesen*-Task). Falls der Ausgang noch auf *true* steht, warten wir einige Zeit und fragen dann erneut ab.

Logik, die von Mikado ausgeführt werden soll, während der Roboter die Kontrolle hat, kann nach dem Setzen des Ausgangs und vor dem Wechsel in die Wartesequenz eingefügt werden.

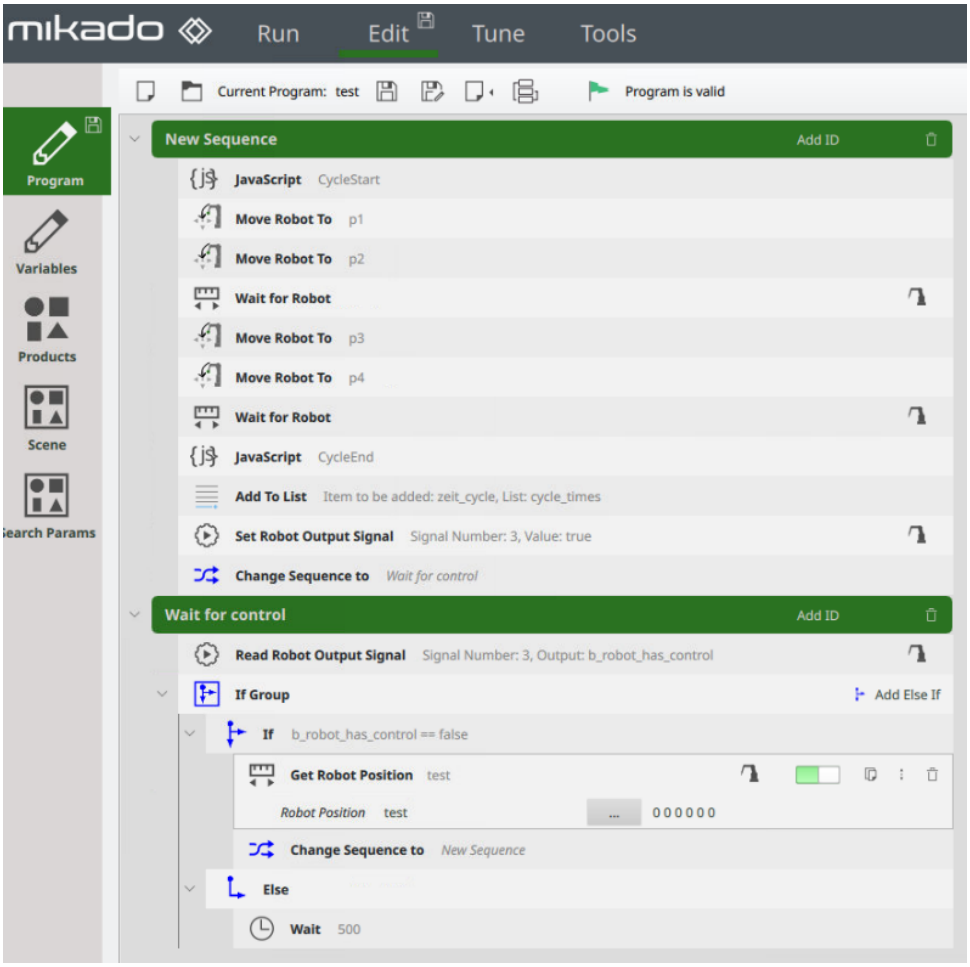


Abb. 9: Beispiel eines Mikado-Programms mit Kontrollabgabe an den ABB Roboter

Auf dem Roboter muss zusätzlich das Programm `ROS_motion` angepasst werden. Fügen Sie dazu in der *WHILE*-Schleife der *main*-Funktion die Abfrage des Ausgangs ein:

```
[...]

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
[...]
```

Sobald der Roboter seine Befehle ausgeführt hat, muss der dafür vorgesehene Ausgang wieder auf *false* gesetzt wer-

den, damit Mikado die Kontrolle wieder übernehmen kann. Im Beispiel geschieht das durch den Befehl `InvertDOMIK_DO3`.

6.1.9 Troubleshooting

Robotergeschwindigkeit erhöhen

Um die Ausführungsgeschwindigkeit der Roboterbewegungen anzupassen, öffnen Sie das Programm `ROS_motion` und passen Sie den Wert der Variablen `move_speed` an. Gehen Sie dabei vorsichtig vor, insbesondere, wenn Sie auch Anpassungen an der Variablen `stop_mode` vorgenommen haben.

Debug-Meldungen auf dem Teach Pendant

Zu Inbetriebnahmezwecken sind auf dem Roboter standardmäßig Debug-Meldungen für das Teach Pendant mittels *TPWrite* aktiviert (Updatefrequenz, Anzahl der übertragenen Trajektorienpunkte etc.). Wenn Sie diese deaktivieren möchten, können Sie dies im `ROS_motionServer` tun, indem Sie die persistente Variable `isDebug` auf false setzen. Meldungen, die nicht durch *TPWrite* als Nachrichten auf das Teach Pendant, sondern mittels *ErrWrite* \W in das Ereignislog geschrieben wurden, sind davon nicht betroffen.

RWS Debug-Meldungen

Um mehr Details zu RWS IO-Vorgängen zu erhalten kann das Logging-Level des RWS Adapters erhöht werden.

Dazu in *launch/rosconsole.config* folgende Zeile ergänzen:

```
log4j.logger.ros.abb_rws_mikado_adapter=DEBUG
```

Verbindungsprobleme

Im Fall von Verbindungsproblemen ist es ratsam, zunächst zu überprüfen, ob der Roboter Controller vom Mikado Controller aus erreicht werden kann. Öffnen Sie hierfür eine Konsole auf dem Mikado Controller (STRG + ALT + T) und geben Sie den folgenden Befehl in die Konsole ein (ersetzen Sie die IP durch die IP des Roboter Controllers):

```
ping 192.168.1.42
```

Das Pingen kann per *Strg + C* abgebrochen werden. Eine positive Antwort würde wie folgt aussehen:

```
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
```

Wenn der Ping erfolgreich ist, kann der Roboter Controller erreicht werden. Überprüfen Sie dann, ob auf dem Roboter beide Serverprogramme (State- und Motionserver) laufen und wo dort die Programmzeiger stehen. Falls in beiden Programmen auf die Verbindung eines Clients gewartet wird, versuchen Sie Mikado neu zu starten.

6.2 Doosan Controller einrichten

Es muss nichts auf den Doosan Roboter-Controller installiert oder kopiert werden.

6.2.1 Netzwerksetup

Für die Kommunikation zwischen Roboter Controller und Mikado Controller wird eine Netzwerkverbindung zwischen den beiden Systemen benötigt. Verbinden Sie hierfür zunächst einen Ethernet Port des Roboter Controllers mit einem Ethernet Port des Mikado Controllers.

- Unter `Settings` → `Network` → `Controller` als IP-Adresse des Doosan Controllers z.B. `192.168.127.100` wählen, und als Subnetzmaske `255.255.255.0`
- Nach dem Einrichten der IP Adresse muss der Doosan Roboter neu gestartet werden.
- Die mit dem Doosan-Controller verbundene Netzwerkkarte des Mikado Controllers muss eine IP im gleichen Subnetz erhalten, also z.B. `192.168.127.200`. Die Netzwerkkonfiguration des Mikado Controllers wird in der Sektion *Netzwerkadapter* näher erläutert.
- Die IP-Adressen des Mikado-Controllers und des Robotercontrollers dürfen nicht gleich sein.
- Kontrolle der Verbindung zum Roboter auf dem Mikado Controller in der Konsole: `ping <ROBOT_IP>`, also z.B. `ping 192.168.127.100`

Vorsicht: * Die IP-Adresse des Robotercontrollers darf nicht im Netz `10.0.0.x` sein, da sie intern verwendet wird. * Doosan-Controller `LAN-Buchse 4` **nicht** verwenden, da sie vom Roboter intern verwendet wird.

6.2.2 Allgemeine Hinweise

- Bevor der Roboter mit Mikado bewegt werden kann sollten die Servos bestromt sein. Man hört ein Klacken, wenn das passiert.
- Es empfiehlt sich den Energiesparmodus des Roboters auszuschalten: * Den Energiesparmodus unter `Settings` → `Idle Servo Off` komplett deaktivieren * Der Default-Tiemout für den Energiesparmodus ist 5 Minuten. Nach diesem Timeout wechselt der Roboter in den Zustand Safety Off.

6.2.3 Moduswechsel Automatik

Der Roboter startet zuerst im manuellen Modus und fährt dann nur langsam. Durch den Wechsel in den Automatikmodus kann die Höchstgeschwindigkeit erhöht werden.

```
# 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
```

Nur im manuellen Modus kann das Werkzeug-Gewicht eingestellt werden.

6.2.4 Name des aktuellen Tools abfragen

```
# 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 Name des aktuellen TCP abfragen

```
# 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 Quellen

- Doosan ROS Handbuch
- Netzwerkeinstellungen und Kontroll-Transfer
- Netzwerkeinstellungen
- Netzwerkanschlüsse
- Energiesparmodus

6.3 Fanuc Controller einrichten

Dieses Kapitel führt Sie durch Installation und Konfiguration eines Fanuc-Controllers zur Nutzung mit einem Mikado-System.

6.3.1 Voraussetzungen

- Robotercontroller:

  Unterstützt werden Fanuc-Controller zur Robotersteuerung wie der R-30iB mit Softwareversion v7 oder höher. Kontaktieren Sie uns, falls Sie abweichende Hardware oder Software verwenden möchten.

- Reset:

  Um Wechselwirkungen mit anderen Programmen oder Konfigurationen zu vermeiden, sollten Sie vor der Anbindung an Mikado den Controller auf Werkseinstellungen zurücksetzen, sofern dieser sich nicht mehr im Lieferzustand befindet. Ein Backup der Controller-Konfiguration wird ebenfalls empfohlen (s. Handbuch Ihres Controllers).

- Teach-Panel:

  Für die einmalige Einrichtung und für das manuelle Fahren wird ein Teach-Panel (z.B. das *iPendant*) benötigt.

- Installieren Sie auf dem Fanuc-Controller folgende Optionen:
  - R648 - User Socket Messaging

  Die installierten Optionen können auf dem Teach-Panel unter *Menu* → *NEXT* → *STATUS* → *Version ID*, dort auf *CONFIG* (oder F3) eingesehen werden.

- Einstellung `$Karel_ENB` setzen:
  - *Menu* → *System* → *Variables*: Prüfen Sie, dass `$Karel_ENB = 1` gesetzt ist

- Register, Variablen, Server Tags: Folgende Slots müssen auf dem Controller freigehalten werden:

  - Server Tags: 3 - 5
  - Digitale Ausgänge: 145 - 146
  - Integer Register: 71 - 76
  - Positionsregister: 61 - 90
- Kollisionsgrenzen / Dual Check Safety (DCS):

  Falls Sie im Fanuc-Controller Kollisionsgrenzen definiert haben oder Dual Check Safety verwenden, ist zu beachten, dass diese Grenzen bei der Kollisions- und Bahnplanung von Mikado nicht berücksichtigt werden. Wählen Sie also immer eine Konfiguration, bei der die Kollisionsgrenzen des Fanuc-Controllers weniger restriktiv sind als die in der Mikado-Konfiguration.
- USB-Stick: Als Installationsmedium verwenden Sie am besten einen USB-Stick. Alternativ können Sie Dateien auch über das Netzwerk (FTP) übermitteln (s. Handbuch Fanuc).

6.3.2 Netzwerkkonfiguration

Zunächst muss eine Ethernet Verbindung hergestellt werden. Dazu verbinden Sie den Fanuc Robotercontroller von dessen Ethernet Port 1 mit dem entsprechenden Mikado-Controller Ethernet Port. Im Mikado Controller sollte nun in der Menüleiste unten rechts eine neue Verbindung angezeigt werden. Für den Fall der direkten Verbindung muss nun diese auf eine feste IP und Subnetzmaske eingestellt werden. Wie ein Mikado-Controller eine feste IP erhält, wird in Sektion *Netzwerkadapter* erläutert.

Auf Seiten des Roboter Controllers muss die IP-Konfiguration nun ebenfalls angepasst werden.

- *Menu* → *Setup* → *Host Comm*
- *TCP/IP* → *F3 Detail*
- *Port#1 IP Addr*: IP Adresse ggf. passend zu IP der Roboter-Netzwerkverbindung des Mikado Controllers wählen. Sie darf aber nicht komplett identisch mit dessen Adresse sein.

Hinweis: Beachten Sie, dass die IP-Adressen von Mikado Controller und Fanuc Controller im selben Adressraum liegen und die Subnetzmasken übereinstimmen.

Tragen Sie anschließend im **Mikado Assistant** die IP-Adresse des Roboter Controllers unter *Roboter* → *Verbindung* ein und speichern Sie die Einstellungen.

Eine Beschreibung des Verbindungstests findet sich in *Troubleshooting: Verbindung testen*.

6.3.3 Mikado-Schnittstelle installieren

Die Versionskompatibilität ist mit Version v7.70/v8.30/v9.30 getestet, die Installationsdatei ist zu höheren Versionen womöglich ebenfalls kompatibel.

- Kopieren Sie zunächst alle benötigten Installationsdateien in das Root-Verzeichnis eines USB Sticks (FAT32 formatiert). Sie finden die Dateien auf dem Mikado Controller in dem Verzeichnis `/opt/isys-vision/mikado/robot_programs/fanuc/fanuc_install_packages/fanuc_mikado_driver_VX.XX/` oder mit Hilfe des entsprechenden Buttons im Mikado Assistant („Parameter“ -> „Roboter“ -> „Verzeichnis öffnen“)

Hinweis: Es ist wichtig, dass die Dateien in das Root-Verzeichnis des USB Sticks kopiert werden, da das Installationsskript sonst die Dateien nicht finden kann.

- Stecken Sie den USB Stick am Teach-Panel des Roboters ein und navigieren Sie zum Inhalt
  - *Menu* → *File* → *File* → *All*

  - *F5 UTIL → Set Device → UT1:* (USB Port am Teach Pendant)
- Selektieren Sie die Installationsdatei `install.cm` und führen Sie die Installation durch das Betätigen der *Enter*-Taste aus
- Falls eine bestehende Treiberinstallation aktualisiert werden soll, muss zuerst die vorherige Installation entfernt werden:
  - *FNCTN → Abort ALL (2x)*
  - Erst `uninstall.cm` ausführen, dann `install.cm`
- Starten Sie den Roboter nach erfolgreicher Ausführung des Installskripts neu

6.3.4 Mikado-Schnittstelle parametrieren

- Flags und Register:

  Wie in Abschnitt *Voraussetzungen* erwähnt, werden zur Kommunikation einige Flags und Register verwendet. Bitte prüfen Sie, dass diese nicht in Benutzung sind und kommentieren Sie ggf. die Flag- und Registerbeschreibungen.
- HTTP Zugriff und KCL freischalten. Dies ermöglicht den automatischen Start der Fanuc-Programme mit Mikado und wird benötigt, damit native Programme ausgeführt werden können (s. *Optional: Natives Roboterprogramm ausführen*).
  - Voraussetzung: HTTP Zugang ist in den Einstellungen altiviert
  - Mithilfe des Teach-Panels nach *Menu → SETUP → Host Comm → HTTP* navigieren
  - Zeile wählen, in der bei Resource: `KCL:\*` eingetragen ist
  - KCL Freischalten, indem *Unprotect* ausgeführt wird
  - (wenn möglich) Folgende Zugangsdaten in den entsprechenden Feldern einrichten: Nutzer: `ROS`, Passwort: `ROS`

Hinweis: Unter *Optional: Natives Roboterprogramm ausführen* wird erläutert, was Sie zusätzlich konfigurieren müssen, damit native Roboterprogramme in Ihren Mikado-Ablauf eingebunden werden können

---

6.3.5 System starten

Das Programm ‚MIKADO' startet alle nötigen Unterprogramme und baut somit die Verbindung zu Mikado auf.

Hinweis: Möchten Sie das laufende Fanuc-Programm stoppen, so führen Sie **zwei Mal** *ABORT ALL* aus (unter *Functions* zu finden).

6.3.6 Optional: Greifer-Ansteuerung über I/Os

Möchten Sie Ihren Greifer über Inputs und Outputs ansteuern, können Sie dies standardmäßig über digitale Ein-/ und Ausgänge umsetzen (DI und DO). In Sektion *gripper.yaml* sind die Konfigurationsmöglichkeiten hierfür aufgeführt.

Sofern Sie jedoch die Roboter I/Os (RI und RO) verwenden möchten, müssen Sie ein Mapping auf DIs und DOs wiefolgt einrichten:

- Neues Background Logic Programm erstellen (BG Logic)
- Beispiel Pseudocode:

  RO[1] = DO[101] RO[2] = DO[102] DI[101] = RI[1]

- In der Mikado-Konfiguration wird der Greifer in diesem Beispiel über die Output-Signale 101 und 102 angesteuert. Signal-Input 101 wird von Roboter-Input 1 gelesen.

6.3.7 Optional: Natives Roboterprogramm ausführen

Mithilfe des Tasks „*Roboterprogramm ausführen*“ lassen sich native Programme auf dem Controller ausführen.

Hierzu müssen Sie lediglich den Programmname des Fanucprogramms im Mikado-Task angeben. Falls im Mikado-Task Variablen parametriert wurden, so werden diese entsprechend zum Fanuc-Programm durchgereicht. Einen Rückgabewert können Sie bei einem Fanuc Roboter leider nicht setzen, hier wird standardmäßig eine 0 übermittelt.

Um diesen Task in Benutzung eines Fanuc Roboters zu verwenden, müssen folgende Vorkehrungen getroffen werden:

- Voraussetzung: HTTP Zugang ist in den Einstellungen aktiviert
- Unter *Menu* → *System* → *Setup* → *Remote/Local Setup*: von `local` auf `remote` stellen
- Benötigte Kommandos über den HTTP Zugang freischalten
  - *Menu* → *System* → *Variables*: Prüfen Sie, dass `$Karel_ENB = 1` gesetzt ist
  - *Select*: Das Programm `HTTPKCL.VR` auswählen
  - *Data* → *[SHOW]* → *KAREL Vars* → *CMDS*: Die nun angezeigte Liste beinhaltet alle blockierten Kommandos
  - `RUN`, `CONTINUE` und `ABORT` aus der Liste entfernen
- *Menu* → *System* → *Variables*: Den Wert `$RMT_MASTER = 1` einstellen (Achtung: diese Einstellung verhindert das Starten von Programmen via UOP)
- (Optional) Falls Ihre nativen Programme längere Zeit aktiv sind, sollten Sie die Timeout-Zeit anpassen (Default: maximal 200 Sekunden). Diese finden Sie in Ihrem Zellverzeichnis unter `launch/manipulator.launch.xml`: Suchen Sie folgende Zeile in der Datei und ändern den Wert von `value` (in Sekunden):

  ```
  <param name="timeout_running_programs" value="200" />
  ```

Hinweis: Sie können die Konfiguration auf Korrektheit prüfen, indem Sie auf Ihrem Mikado-Controller einen Browser öffnen und ein Kommando über den Aufruf einer URL ausführen:

`http://<ROBOTER_IP>/KCL/run%20<PROGRAMMNAME>` (z.B. http://192.168.2.42/KCL/run%20ROS_STATE)

Erhalten Sie keine Fehlermeldung, war der Aufruf erfolgreich.

Hinweis: Wird der Task „*Roboterprogramm ausführen*“ in einem Mikado Ablauf aufgerufen, passiert folgendes:

1. Mikado beendet das Programm auf dem Controller, das für Mikado Bewegungen ausführt
2. Variablenwerte werden für das aufzurufende Programm übermittelt
3. Natives Programm wird gestartet
4. Mikado wartet, bis das native Programm beendet wurde
5. Mikado startet das Programm erneut, welches Bewegungen für Mikado ausführt

---

6.3.8 Troubleshooting: Verbindung testen

Fehlerbehebung, falls nach der Konfiguration und Neustart keine Verbindung zum Roboter besteht. Im Bereich der *Statusleiste* lässt sich erkennen, ob die Verbindung zum Roboter erfolgreich aufgebaut wurde.

- Prüfen: Robotercontroller und Mikado-Controller haben die gleiche Subnetzmaske und unterscheidliche feste IPs im gleichen Adressraum
- Prüfen: Die Status LED des Netzwerkports zeigt keinen Fehler an
- Prüfen: Ist der Roboter unter der eingetragenen IP per *ping* erreichbar

Das nötige Handwerkszeug zum Ausführen des Ping Kommandos vom Mikado-Controller:

Auf dem Mikado-Controller muss eine Konsole gestartet werden. Der einzugebene Ping Befehl lautet dann zum Beispiel:

```
ping 192.168.1.42
```

Das Pingen kann per *Strg + C* abgebrochen werden. Eine positive Antwort würde wiefolgt aussehen:

```
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 KUKA Controller einrichten

Im Folgenden wird beschrieben, wie die roboterseitige Vorbereitung für Mikado funktioniert, insbesondere die Verwendung der verschiedenen KRL-Skripte. Auf dem Roboter laufen zwei Interpreter: Der Submit-Interpreter (`C:\KRC\ROBOTER\KRC\R1\System\mik_service_sps.sub` und `mik_status_sps.sub`) und der Roboter-Interpreter (`C:\KRC\ROBOTER\KRC\R1\Program\mikado.src`). Der Submit-Interpreter wird üblicherweise automatisch gestartet und läuft dann im Hintergrund, das Roboter-Program muss ausgewählt werden. Diese kommunizieren beide per EthernetKRL mit dem Mikado Controller.

6.4.1 Systemanforderungen

Mikado kommuniziert über EthernetKRL mit dem KUKA Controller und verwendet Splines für Fahrbefehle. Es muss daher mindestens KSS Version 8.3 installiert sein sowie das Zusatzpaket EthernetKRL ab Version 2.2. Da aus Performancegründen zwei Submit Programme laufen, muss auf Controllern mit KSS 8.3 oder 8.4 noch die MultiSubmit Erweiterung installiert werden, ab KSS 8.5 ist diese Standard.

6.4.2 Netzwerkkonfiguration

Für die Kommunikation zwischen dem Mikado-Controller und der Robotersteuerung muss eine Netzwerkverbindung hergestellt werden. Standardmäßig ist hierfür die **KLI** Schnittstelle (**KUKA Line Interface**) der Robotersteuerung mit den folgenden Anschlüssen vorgesehen:

- KR C4: `X66`
- KR C5: `XF5`

Warnung: Ohne zusätzliche Optionspakete müssen unbedingt diese Netzwerkanschlüsse verwendet werden. Andere RJ45-Buchsen der Robotersteuerung funktionieren gar nicht für die Kommunikation mit Mikado (z.B. KSI) oder sie müssen mittels zusätzlicher Optionspakete freigeschaltet werden. Alternative Anschlussmöglichkeiten, die mittels Optionspaketen verfügbar sind, werden weiter unten näher erläutert.

Definieren Sie sowohl für den Mikado-Controller als auch für den Roboter-Controller eine feste IP-Adresse und eine zugehörige Subnetzmaske. Die Netzwerkkonfiguration des Mikado Controllers wird in Sektion *Netzwerkadapter* näher erläutert.

Hinweise zur Netzwerkkonfiguration des KUKA-Controllers finden Sie im Benutzerhandbuch des Herstellers.

Die IP-Adresse des Roboter-Controllers muss anschließend in der Mikado-Zelle hinterlegt werden. Navigieren Sie dafür im Mikado Assistant zu `Parameter` → `Roboter` und tragen dort die IP-Adresse des Roboter-Controllers in das entsprechende Feld ein.

Hinweis: Beachten Sie, dass die IP-Adressen von Mikado-Controller und KUKA-Controller im selben Adressraum liegen und die Subnetzmasken übereinstimmen. Falls wie empfohlen eine separate Netzwerkverbindung für die Kamera verwendet wird, muss diese in einem anderen Addressraum liegen, um einen uneindeutigen Paketversand zu vermeiden.

Die IP-Adresse der KLI Schnittstelle des Robotercontrollers darf **nicht** innerhalb der folgenden von KUKA reservierten Bereiche liegen:

- 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 Anschlussmöglichkeiten:

Wenn die oben genannten Standard-Netzwerkanschlüsse für die KLI Schnittstelle anderweitig belegt sind (z. B. durch die Profinet-Kommunikation des Roboters), können folgende Ausweichmöglichkeiten in Betracht gezogen werden:

- Es kann ein externer Ethernet-Switch an `XF5` (KR C5) oder `X66` (KR C4) verwendet werden, um die Anzahl verfügbarer Anschlüsse für die KLI Schnittstelle zu erhöhen.
- KR C5 Steuerungen haben einen internen Netzwerkswitch für die KLI-Schnittstelle verbaut, dessen Anschlüsse mittels Optionspaketen freigeschaltet werden können.
  - **KUKA.EtherNetSwitch Dual-Port**: zusätzlich zu `XF5` kann auch `XF6` verwendet werden
  - **KUKA.EtherNetSwitch Quad-Port**: zusätzlich zu `XF5` können auch `XF3`, `XF4` und `XF6` verwendet werden
- KR C4 Steuerungen haben je nach Ausstattung gegebenenfalls auch einen Switch für die KLI-Schnittstelle verbaut. Falls vorhanden, handelt es sich hierbei um die Anschlüsse `X67.1`, `X67.2` und `X67.3`. Diese Anschlüsse können je nach Ausstattungsvariante des Controllers allerdings auch für andere optionale Interfaces verwendet werden und sind nicht immer eindeutig der KLI Schnittstelle zugeordnet.

  Um Aufschluss darüber zu erhalten, ob diese Anschlüsse für KLI verwendbar sind, konsultieren Sie gegebenenfalls das Handbuch Ihrer konkreten Ausstattungsvariante.
- Mit der Option **KUKA.OptionalNetworkInterface** kann bei KR C5 Steuerungen die **KONI** Schnittstelle (`XF7`) freigeschaltet werden. Diese kann auch für die Kommunikation via Ethernet KRL verwendet werden. Im Gegensatz zu den *KUKA.EtherNetSwitch* Optionen für die KLI Schnittstelle stellt KONI ein zusätzliches, komplett eigenständiges Netzwerkinterfac bereit, das unabhängig konfigurierbar ist (eigene IP-Adresse, Subnetz etc.). Dadurch ist es beispielsweise möglich, für die Ethernet KRL Kommunikation und die Profinet-Kommunikation des Roboters verschiedene Netzwerkinterfaces zu verwenden. Dies kann Kommunikationsprobleme bei sehr hohem Profinet-Kommunikationsaufkommen vermeiden, siehe *Troubleshooting*.

Konfiguration von KONI: Die Ports 54600, 54601 und 54602 dürfen unter `Netzwerkkonfiguration` → `KONI` → `Erweitert` → `NAT` **nicht** in der Liste der Ports aufgeführt sein. Diese Ports werden für die Kommunikation mit Mikado benötigt und dürfen deshalb nicht anderweitig belegt werden.

- KRC4 Steuerungen haben je nach Ausstattung gegebenenfalls auch eine **KONI** Schnittstelle zur Konfiguration eines eigenständigen, von der KLI Schnittstelle getrennten Netzwerkinterfaces. Falls vorhanden, handelt es sich hierbei um die Anschlüsse `X64.1`, `X64.2` und `X64.3`.

  Um Aufschluss darüber zu erhalten, ob KONI vorhanden ist, konsultieren Sie gegebenenfalls das Handbuch Ihrer konkreten Ausstattungsvariante.

6.4.3 KRC vorbereiten

Um den KUKA-Roboter mit Mikado Control steuern zu können, müssen die entsprechenden Roboterprogramme auf die Robotersteuerung kopiert, konfiguriert und gestartet werden. Im Folgenden werden die hierfür notwendigen Schritte näher erläutert.

Kopieren der Dateien

Zunächst müssen die benötigten Roboterprogramme auf die Robotersteuerung kopiert werden. Zu finden sind diese Programme auf dem Mikado Controller unter:

`/opt/isys-vision/mikado/robot_programs/kuka/`

oder mit Hilfe des Buttons im Mikado Assistant (`Roboter` → `Verbindung` → `Verzeichnis öffnen`). Innerhalb des Verzeichnisses gibt es drei Unterordner, die jeweils einem Ordner auf dem Roboter Controller entsprechen:

- `EthernetKRL/*` → `C:\KRC\ROBOTER\Config\User\Common\EthernetKRL\*`
- `Program/*` → `C:\KRC\ROBOTER\KRC\R1\Program\*` oder `KRC:\R1\Program\*`
- `System/*` → `C:\KRC\ROBOTER\KRC\R1\System\*` oder `KRC:\R1\System\*`

Im Ordner EthernetKRL finden Sie die Netzwerkeinstellungen. Da ein KUKA-Controller mehrere Netzwerkanschlüsse haben kann, muss die IP des für die Kommunikation mit Mikado verwendeten Anschlusses in den Konfigurationsdateien hinterlegt werden (Die IP-Adresse der Robotersteuerung, nicht die des Mikado-Controllers!). Dazu müssen Sie in den Dateien `MikadoXmlService.xml`, `MikadoXmlStatus.xml` und `MikadoXmlTraj.xml` jeweils in Zeile 12 die IP-Adresse des KUKA Controllers angeben. Dies geht beispielsweise recht bequem mit dem Texteditor Kate, nachdem Sie die Dateien aus oben genanntem Verzeichnis auf einen USB-Stick kopiert haben (der Ordner /opt und alle darin enthaltenen Dateien sind schreibgeschützt, weshalb die Bearbeitung erst nach dem Kopieren möglich ist). Etwas weniger bequem können die Dateien auch direkt auf dem KUKA-Controller bearbeitet werden. Nach Änderungen an den EthernetKRL XML-Dateien ist ein Neustart des KUKA Controllers notwendig.

Der System-Ordner enthält Programme, die vom Submit-Interpreter ausgeführt werden (siehe unten).

Der Program-Ordner enthält Programme, die vom Roboter-Interpreter ausgeführt werden. Insbesondere ist dies das Programm „mikado.src“, welches wiederum andere .src-Dateien aufruft.

Um Dateien vom USB-Stick mit dem SmartPAD auf den KUKA-Controller zu kopieren: `Anzeige` → `Fenster` → `Navigator`. Im Navigator können Dateien markiert und dann über `Bearbeiten` → `Kopieren` und `Bearbeiten` → `Einfügen` kopiert werden. Eine USB-Maus am SmartPAD vereinfacht die Bedienung. Bei ausreichenden Benutzerrechten kann die HMI-Oberfläche auch minimiert und der Windows Explorer verwendet werden.

Hinweis: Aus Sicherheitsgründen sind USB-Sticks für manche Benutzergruppen eventuell nicht sichtbar.

Wechseln Sie mindestens zur Benutzergruppe `Experte`, wenn der USB-Stick auf dem SmartPAD nicht als Datenträger angezeigt wird. Wenn er auch dann nicht angezeigt wird, verwenden Sie einen anderen USB-Anschluss (z. B. direkt am Roboter-Controller statt am SmartPad) oder einen anderen USB-Stick.

Submit-Interpreter [S]

Wählen Sie im Submit-Interpreter die Datei `mik_status_sps.sub` für den Slot EX1 an. Wechseln Sie dafür in den Modus T1 und anschließend auf das Benutzerlevel Experte. Dann über `S` → `Stoppen`, dann `Anzeigen/Zuweisen` und das Programm auswählen. Im Mikado Control sollte der Roboter nun in der richtigen Stellung angezeigt werden. Laden Sie zusätzlich `mik_service_sps.sub` in Slot EX2, dieses steuert zB die IOs.

Damit die Module auch bei einem Neustart der Robotersteuerung automatisch angewählt und gestartet werden, müssen sie in der Kaltstart-Konfiguration angewählt werden. Schalten Sie hierzu vom Reiter `Aktuelle Anzeige/Zuweisung` um auf `Kaltstart-Konfiguration` und wählen Sie auch dort die beiden Module für die Slots EX1 und EX2 an.

Roboter-Interpreter [R]

Wählen Sie nun das Hauptprogramm `mikado.src` für den Roboter-Interpreter an und starten Sie dieses. Navigieren Sie dazu auf dem SmartPAD in den Ordner „Program“ (siehe oben), wählen das Programm aus und drücken am unteren Bildschirmrand die `Anwählen` Taste. Das Programm ist nun zur Ausführung vorgewählt und muss anschließend je nach Betriebsmodus wie folgt gestartet werden:

- Manuelle Modi **T1 / T2**: Zustimmtaster auf der Rückseite des SmartPADs drücken und halten. Anschließend die `Start` Taste drücken und ebenfalls halten. (Ggf. muss die `Start` Taste mehrfach gedrückt werden, bis das Programm erfolgreich gestartet ist). Beide Tasten müssen in den manuellen Modi permanent gedrückt gehalten werden, damit das Roboterprogramm ausgeführt wird und Bewegungen von Mikado ausgeführt werden können. Diese Betriebsart eignet sich somit z.B. für die Inbetriebnahme (Durchführung der Kalibrierung, Einlernen von Posen, etc.), da der Roboter nur durch loslassen einer der beiden Tasten sofort gestoppt werden kann.
- Automatik Modi **Auto / Extern**: `Start` Taste drücken (ggf. mehrfach). Halten des Zustimmtasters auf der Rückseite des SmartPADs ist bei den Automatik-Modi nicht erforderlich. Das Programm wird gestartet und dann ohne zusätzliche Benutzerinteraktion automatisch weiter ausgeführt (d.h. es muss im Gegensatz zu den manuellen Modi keine Taste permanent gedrückt gehalten werden).

Bei erfolgreich gestartetem Roboterprogramm wird das Symbol für den Roboter-Interpreter `R` am oberen Bildschirmrand des SmartPADs grün angezeigt. Hat das Symbol eine andere Farbe (z.B. orange oder rot), so ist das Roboterprogramm noch nicht erfolgreich gestartet.

Wenn Sie nun im Mikado auf der Roboterseite z.B. eine kleine Fahrt von Achse 6 vornehmen, sollte diese vom KUKA-Roboter ausgeführt werden.

6.4.4 Optional: Natives Roboterprogramm ausführen

Mithilfe des Tasks „*Roboterprogramm ausführen*“ lassen sich native Programme auf dem Controller ausführen.

Soll ein neues Programm namens „XYZ“ hinzugefügt werden, dann sollte das Programm in der Datei „KRC/R1/Program/prog_<NAME>.src“ mit gleichnamigem Funktionsnamen („prog_<NAME>“) gespeichert werden. Anschließend muss in der Datei „C:/KRC/ROBOTER/KRC/R1/Program/run_program.src“ das Programm im Bereich „run_program_impl“ hinzugefügt werden. Dort aus dem Kommentarblock entweder Variante A für Funktionen mit Argumenten oder Variante B für Prozeduren ohne Argumente kopieren und anpassen.

Hinweis: Die Vorlage führt standardmässig dazu, dass ein Scheitern zurück gemeldet wird. Ändern Sie die Zeile `result.success = false` auf `true` um ein erfolgreiches Ausführen an Mikado zu melden. Mit einem Semikolon `;` können Sie zusätzlich die Fehlermeldung `result.errorMsg` auskommentieren, oder Sie löschen die Zeile komplett.

Hinweis: Wenn Sie im nativen Programm eine Fahrt ausführen, muss das Program mit einem Vorlaufstopp beendet werden. Dieser kann bei KUKA durch ein `wait for true` ausgelöst werden. Nur so kann Mikado bei darauf folgenden Fahrten mit der richtigen Ausgangsposition planen.

6.4.5 Roboterkontrolle übergeben

In manchen Anwendungen ist es erwünscht, dass Roboter-Bewegungen teilweise durch Mikado und teilweise über ein Roboterprogramm kontrolliert werden. Das heißt, dass Mikado nur einen Teil des gesamten Bewegungsablaufs des Roboters ausführt und anschließend die Kontrolle abgibt, damit andere Abläufe auf dem Roboter ausgeführt werden können, zum Beispiel um auf dem Roboter geteachte Positionen anzufahren.

Der Roboter muss bei der Kontrollübergabe jeweils stillstehen!

Kontrolle an Mikado übergeben

Damit Fahrbefehle von Mikado ausgeführt werden können, muss die Funktion `mikado.src/mikado()` im Roboterinterpreter des KRC ausgeführt werden. Diese Funktion startet eine Endlosschleife auf dem Robotercontroller, in der auf neue Fahrbefehle von Mikado gewartet und diese ausgeführt werden.

Mikado hat während der Ausführung dieser Funktion die Kontrolle über den Roboter.

Kontrolle an den Roboter übergeben

Um die Kontrolle von Mikado an den Robotercontroller abzugeben, kann an der gewünschten Stelle im Mikado-Ablauf die folgende Funktion mittels eines *JavaScript* Tasks aufgerufen werden.

```
robot.triggerRobotService("/give_up_robot_control")
```

Durch diesen Aufruf wird die oben genannte Endlosschleife innerhalb von `mikado.src/mikado()` unterbrochen und die Funktion wird verlassen. Anschließend kann auf dem Roboter beliebig fortgefahren werden.

6.4.6 Geschwindigkeit und Fine-Tuning

Die Geschwindigkeit mit der Fahrten ausgeführt werden, kann über einige Parameter eingestellt werden. Zusätzlich ist auf dem KUKA-Controller/Teach Pendant noch die Geschwindigkeit einstellbar, diese wird dann mit den Werten von Mikado multipliziert.

Allgemein für alle Fahrten können die folgenden Parameter in `config/manipulator.yaml` eingestellt werden:

`velocity_scaling` skaliert die im URDF angegebene Maximalgeschwindigkeit. Wenn diese in Mikado einstellbar ist, legt `max_velocity_scaling` die Obergrenze für den Benutzer fest. Der Wert liegt zwischen 0 und 1, was der vollen Geschwindigkeit oder 100%% entspricht. `acceleration_scaling` setzt den entsprechenden Faktoren für die Beschleunigung.

Für den kartesischen (linearen) Teil eines Griffes können in den einzelnen `.product` Dateien im Ordner `products` noch separat langsamere Fahrten festgelegt werden. Diese fungieren ebenfalls als Faktoren und wirken also in Kombination mit velocity_scaling. `ApproachVelocityScaling: 0.5` gibt den Geschwindigkeitsfaktor vor dem Greifen und `RetractVelocityScaling: 0.3` nach dem Greifen beim Hochheben an. Damit Mikado die Änderungen nicht überschreibt, sollte es am besten beendet werden, bevor man die Werte in diesen `.product`-Dateien anpasst.

Basierend auf der Maximalgeschwindigkeit im URDF bzw `joints_limits.yaml`, den obigen Faktoren und den möglichen Beschleunigungen rechnet Mikado nun aus wie lange die einzelnen Abschnitte der Fahrt benötigen werden. Aus dieser Dauer wird für den KUKA ein Prozentsatz der Maximalgeschwindigkeit zurückberechnet. Leider führt diese Berechnung im Allgemeinen zu niedrigeren Geschwindigkeiten als erwartet, daher kann mit `velocity_scaling_factor` das Ergebnis vergrößert werden, `max_velocity_clipping` setzt eine obere Grenze, `min_velocity_clipping` eine untere.

6.4.7 Troubleshooting

Timeouts und Verbindungsabbrüche bei hoher Netzwerkauslastung

Bei hoher Auslastung der mit Mikado verbundenen Netzwerk-Schnittstelle durch weitere Kommunikationsvorgänge kann es zu Timeouts oder Verbinundungsabbrüchen aufgrund fehlender Antwort-Nachrichten des Roboters kommen.

Die Profinet-Kommunkation hat als Echtzeit-Protokoll grundsätzlich eine höhere Priorität auf dem Robotercontroller als die Ethernet KRL Schnittstelle, die für die Kommunikation mit Mikado verwendet wird. Bei den meisten Anwendungen stellt diese Tatsache kein Problem dar.

Wenn sowohl Profinet als auch Mikado (Ethernet KRL) über die KLI Schnittstelle kommunizieren, kann ein sehr hohes Profinet-Kommunikationsaufkommen jedoch letztendlich dazu führen, dass die Ethernet KRL Schnittstelle aufgrund ihrer niedrigeren Priorität gegebenenfalls Nachrichten nicht mehr wie erwartet senden oder empfangen kann.

Wenn dieser Fall eintritt, führt das wiederum auf Seiten von Mikado zu Timeouts und Verbindungsabbrüchen. Diese äußern sich üblicherweise in Form von Fehlermeldungen folgender Art.:

```
[kuka-driver][communication][...][read] reading timed out after 300 ms!
[kuka-driver][keep_alive][...] reading timed out!
```

Diese Problematik scheint besonders bei sehr kurzen Zykluszeiten (Sendetakt) für das Profinet-Protokoll in Erscheinung zu treten.

Beispiel: Ein Profinet-Sendetakt von 2 ms kann bei einer KR C5 Steuerung die Netzwerk-Schnittstelle des Roboters so stark belasten, dass das beschriebene Problem auftritt.

Um diese Situation zu vermeiden, gibt es zwei Lösungsansätze:

- KONI Schnittstelle für die Ethernet KRL Kommunikation verwenden, um Profinet und Ethernet KRL komplett voneinander zu trennen.
- Profinet-Kommunikation soweit möglich reduzieren, um die Netzwerk-Schnittstelle des Roboters zu entlasten (z.B. durch langsameren Profinet-Sendezyklus).

6.5 Mitsubishi Controller einrichten

Getestet mit Controller CR75x-D, Software Version S7 17-02-28.

6.5.1 Netzwerkkonfiguration

Zunächst muss eine Ethernet Verbindung hergestellt werden. Dazu verbinden Sie den Mitsubishi Robotercontroller von dessen Ethernet Port mit einem Mikado-Controller Ethernet Port. Im Mikado Controller sollte nun in der Menüleiste unten rechts eine neue Verbindung angezeigt werden. Für den Fall der direkten Verbindung muss nun diese auf eine feste IP und Subnetzmaske eingestellt werden. Wie ein Mikado-Controller eine feste IP erhält, wird in Sektion *Netzwerkadapter* erläutert.

Auf Seiten des Roboter Controllers muss die IP-Konfiguration nun ebenfalls angepasst werden. Dazu öffnen Sie auf dem Teach-Panel den Unterpunkt *Parameter* des Hauptmenüs.

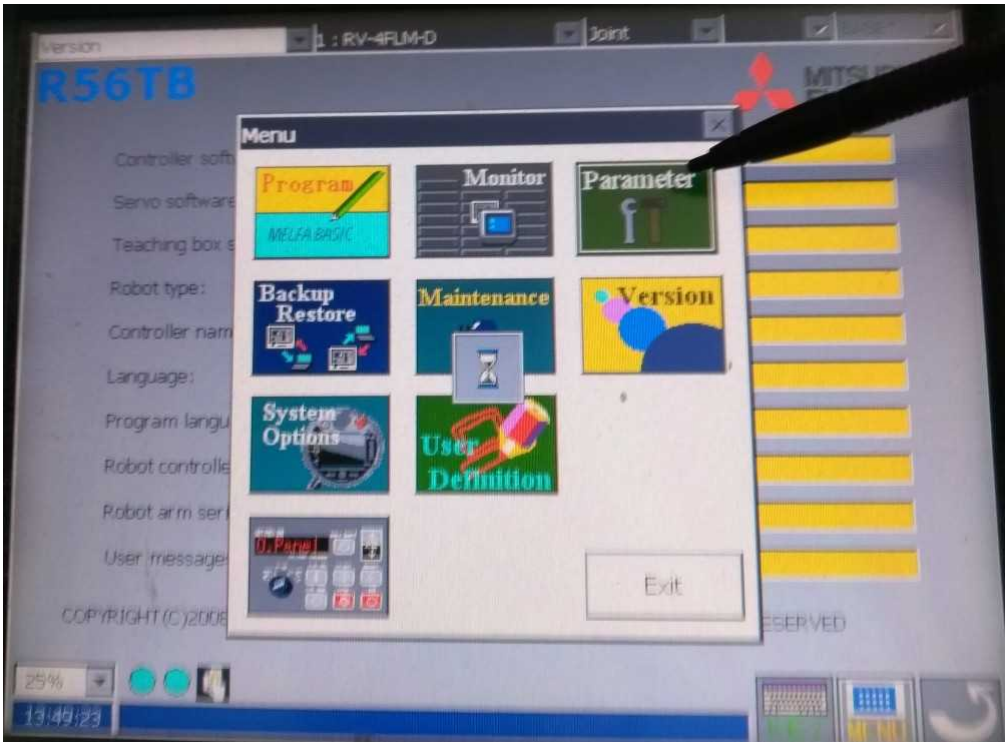


Abb. 10: Bild Mitsubishi Parameter Seite öffnen

Hier wählen Sie das Ethernet-Menü.

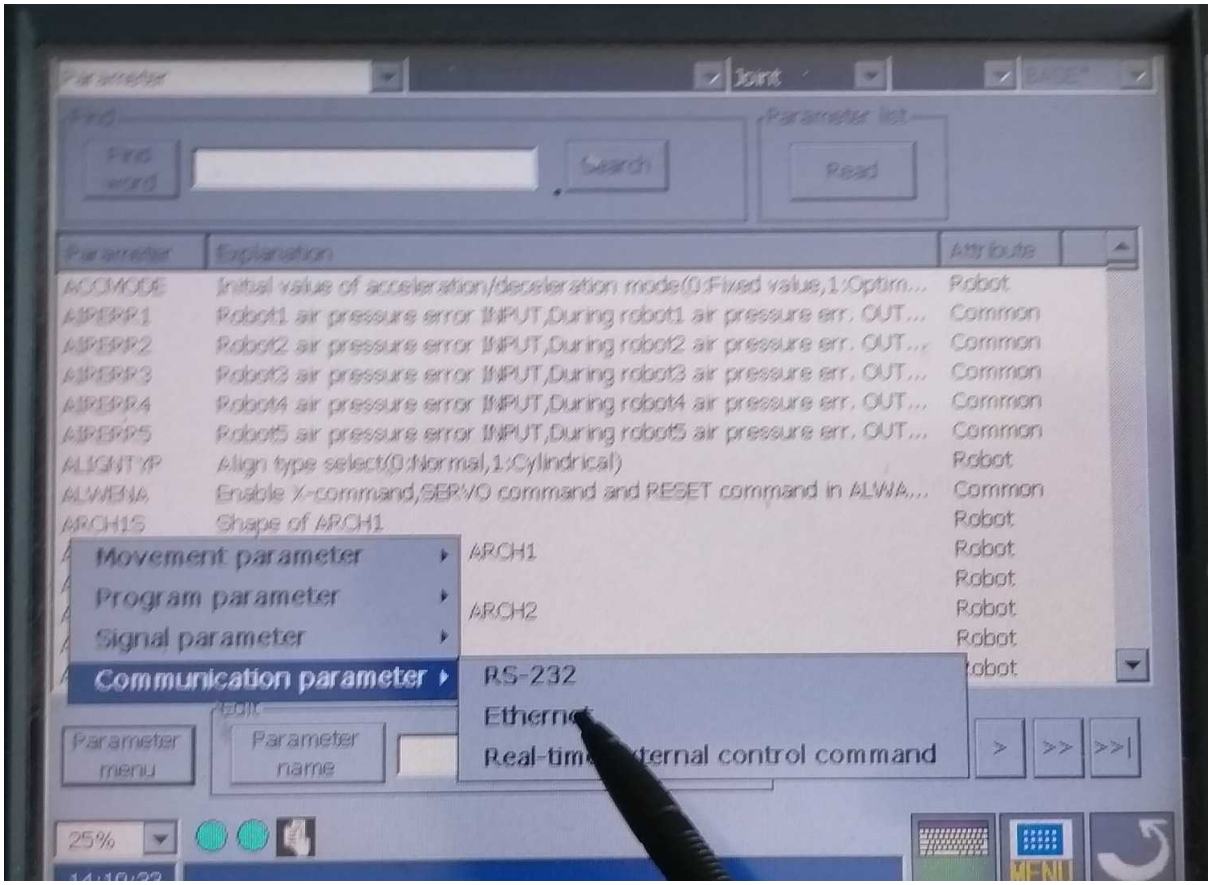


Abb. 11: Bild Mitsubishi TP Ethernet

In der nun erscheinenden Ansicht lässt sich die IP-Adresse und die Subnetzmaske einstellen.

Hinweis: Für den Roboter Controller muss eine IP-Adresse im selben Adressraum wie der Mikado Controller gewählt werden und die Subnetzmasken müssen identisch sein.

Außerdem müssen die Ports wie nachfolgend beschrieben eingestellt werden, so dass eine Datenverbindung wie erwartet hergestellt werden kann.

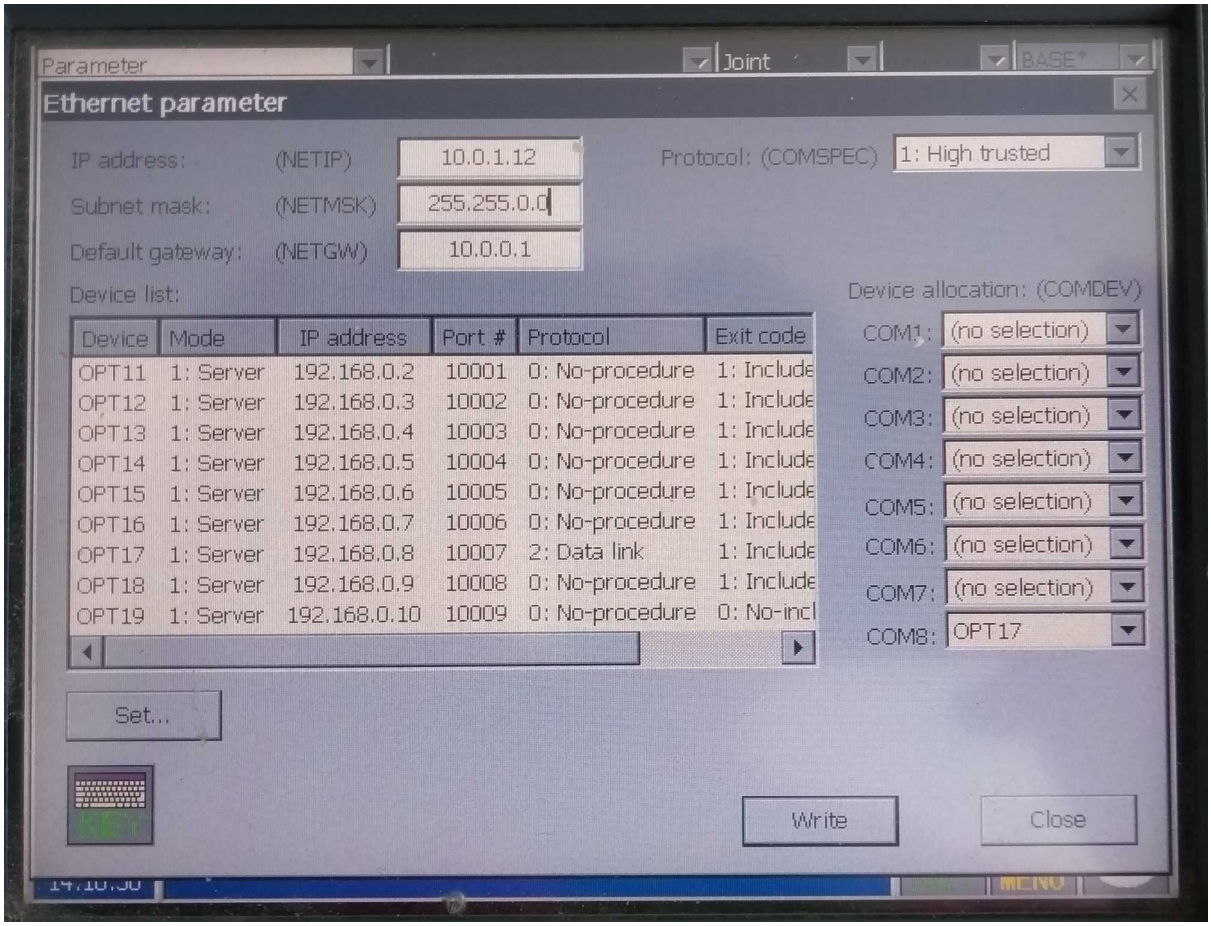


Abb. 12: Bild Mitsubishi Beispiel IP Konfiguration

Wählen Sie das Device OPT17, bzw. den Port 10007 an und drücken den **Set…** Knopf. Es erscheint ein Auswahlmenü, für das Sie die folgenden Werte übernehmen:

Name	**Wert**
Mode	Server
Port #	10007
Protocol	Data link
Packet type	CR
Allocation	COM8

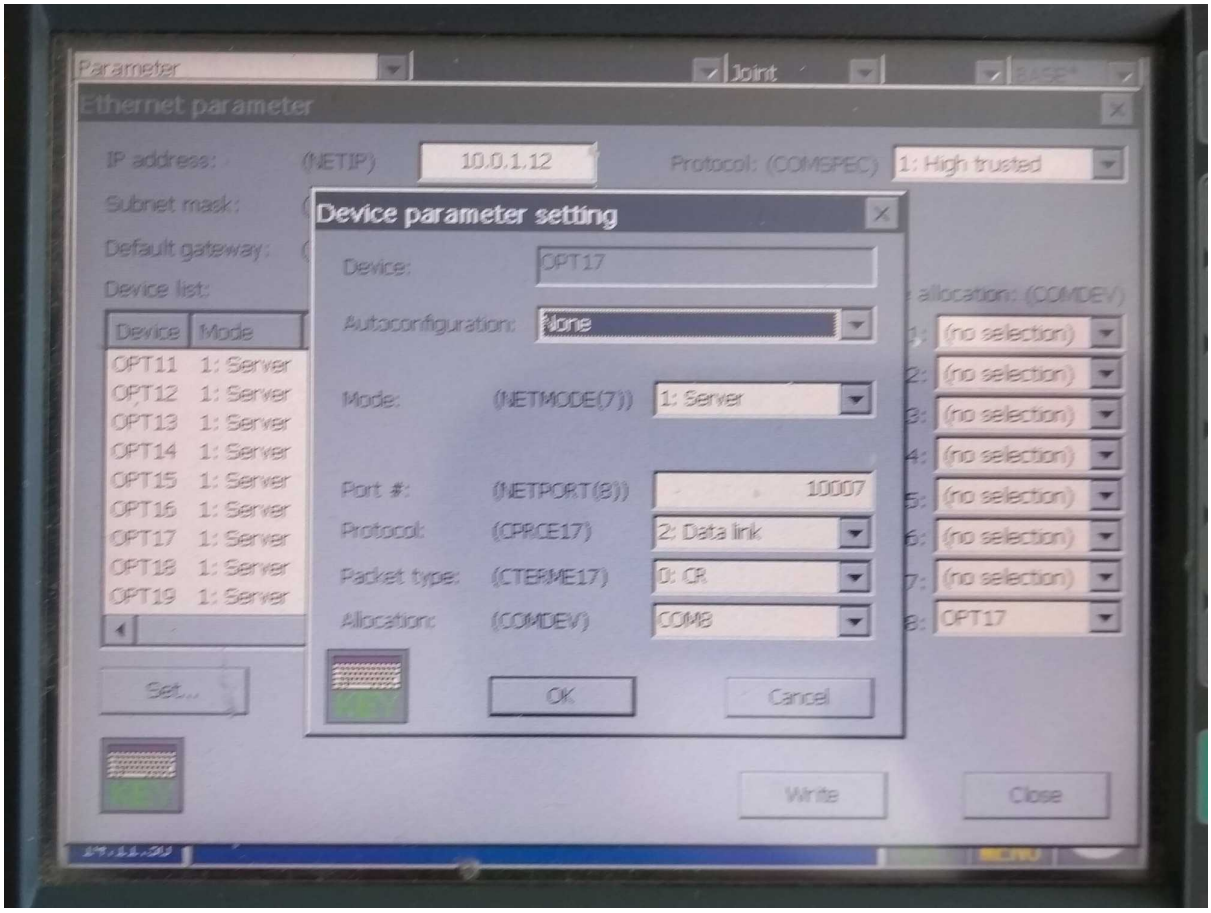


Abb. 13: Bild Mitsubishi Beispiel Port Konfiguration

Für alle anderen Ports ist keine Rollenzuweisung nötig, bzw. vorhandene sollten abgeschaltet werden durch setzen des *Protocol* Parameters auf den Wert *No procedure*:

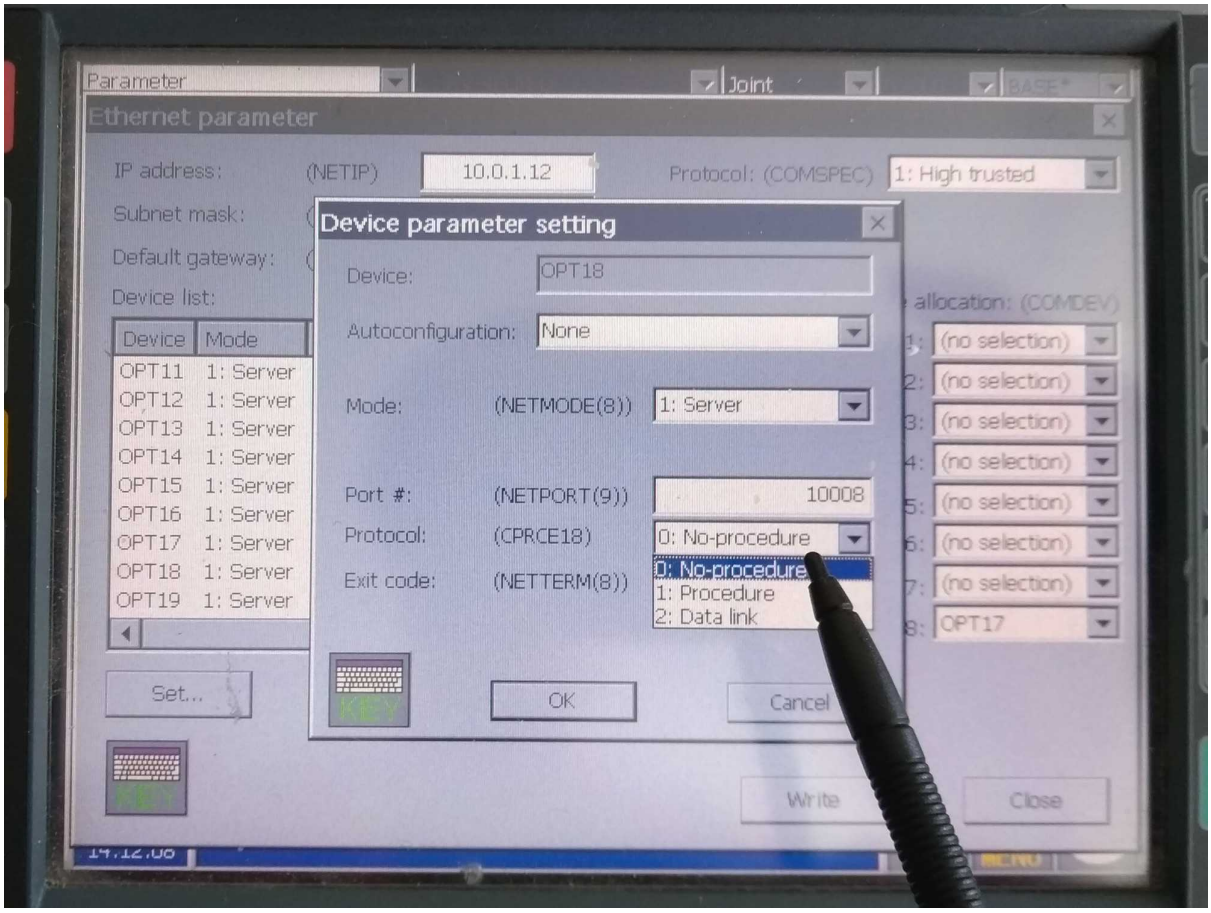


Abb. 14: Bild Mitsubishi Beispiel Port Neutral setzen

Die Änderungen müssen nun vom Teach Panel auf den Robotercontroller überschrieben werden.

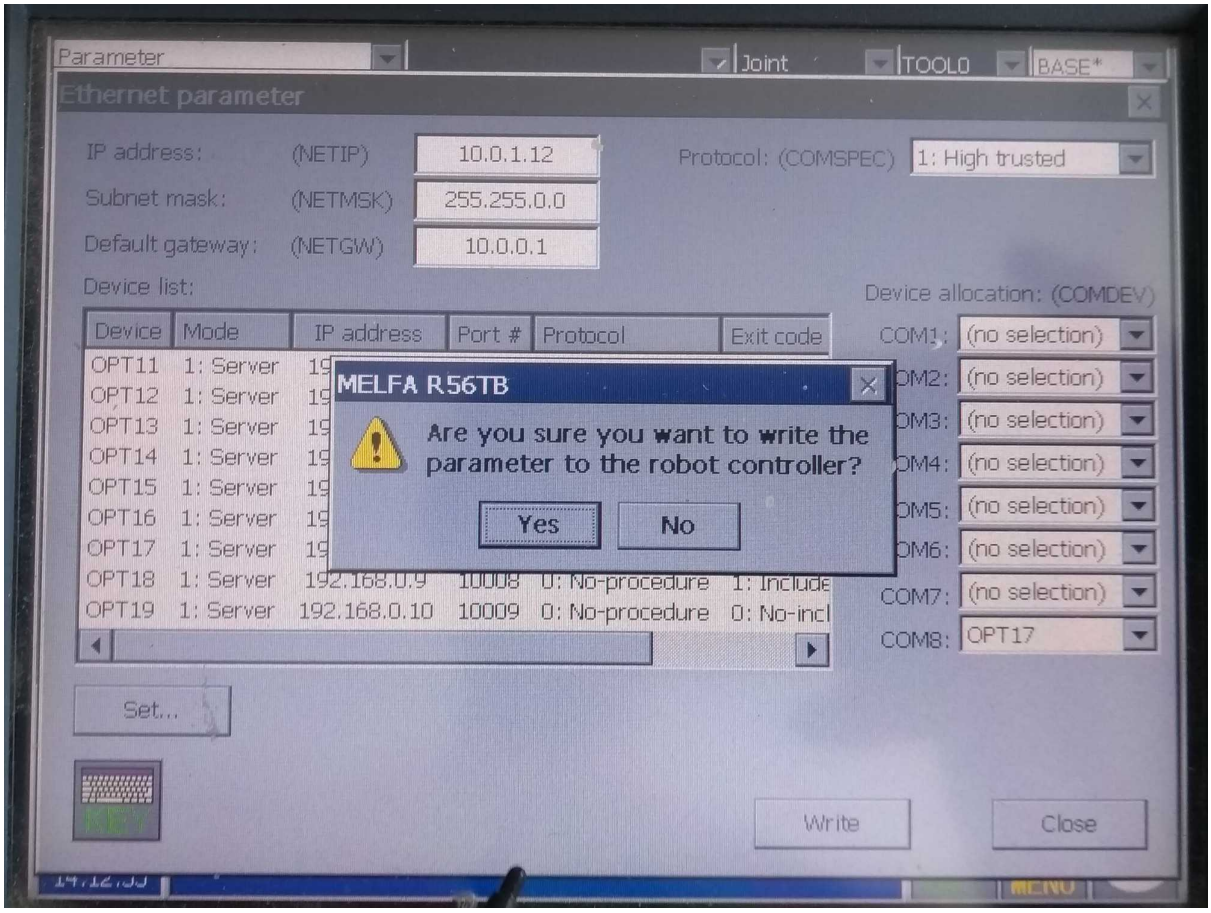


Abb. 15: Bild Mitsubishi Parameter Schreiben

Danach muss der Controller neu gestartet werden.

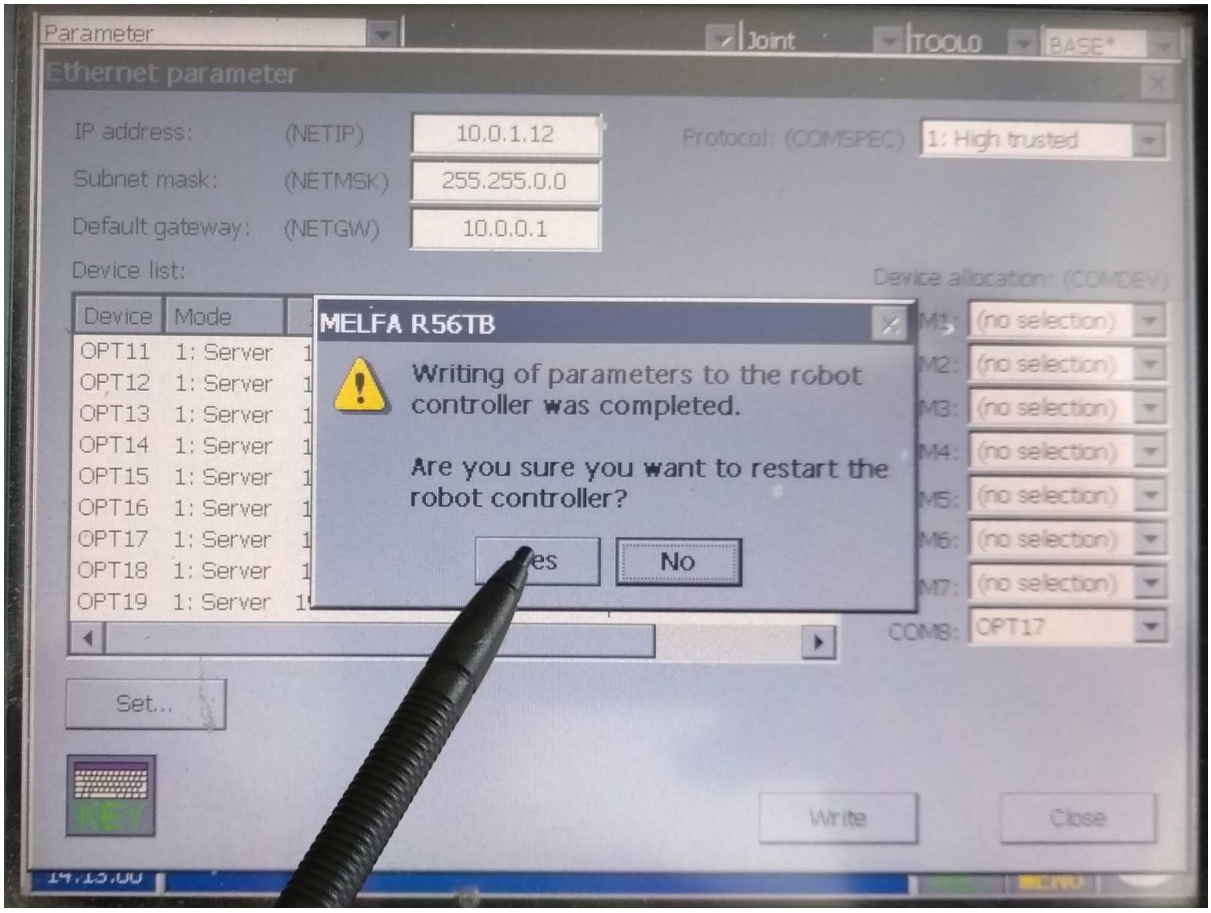


Abb. 16: Bild Mitsubishi Controller Neustart

Tragen Sie die gewählte IP-Adresse des Roboter Controllers nun im Mikado Assistant unter „Parameter“ -> „Roboter“ in das entsprechende Feld ein und speichern Sie die Einstellungen.

6.5.2 Verbindung konfigurieren

Der Mikado-Controller erwartet, dass die explizite Zeilenterminierung eingeschaltet ist. Diese überprüfen Sie, bzw. setzen Sie über das Parameter Menü. Hier geben Sie den Namen *NETTERM* ein und wählen *Read*.

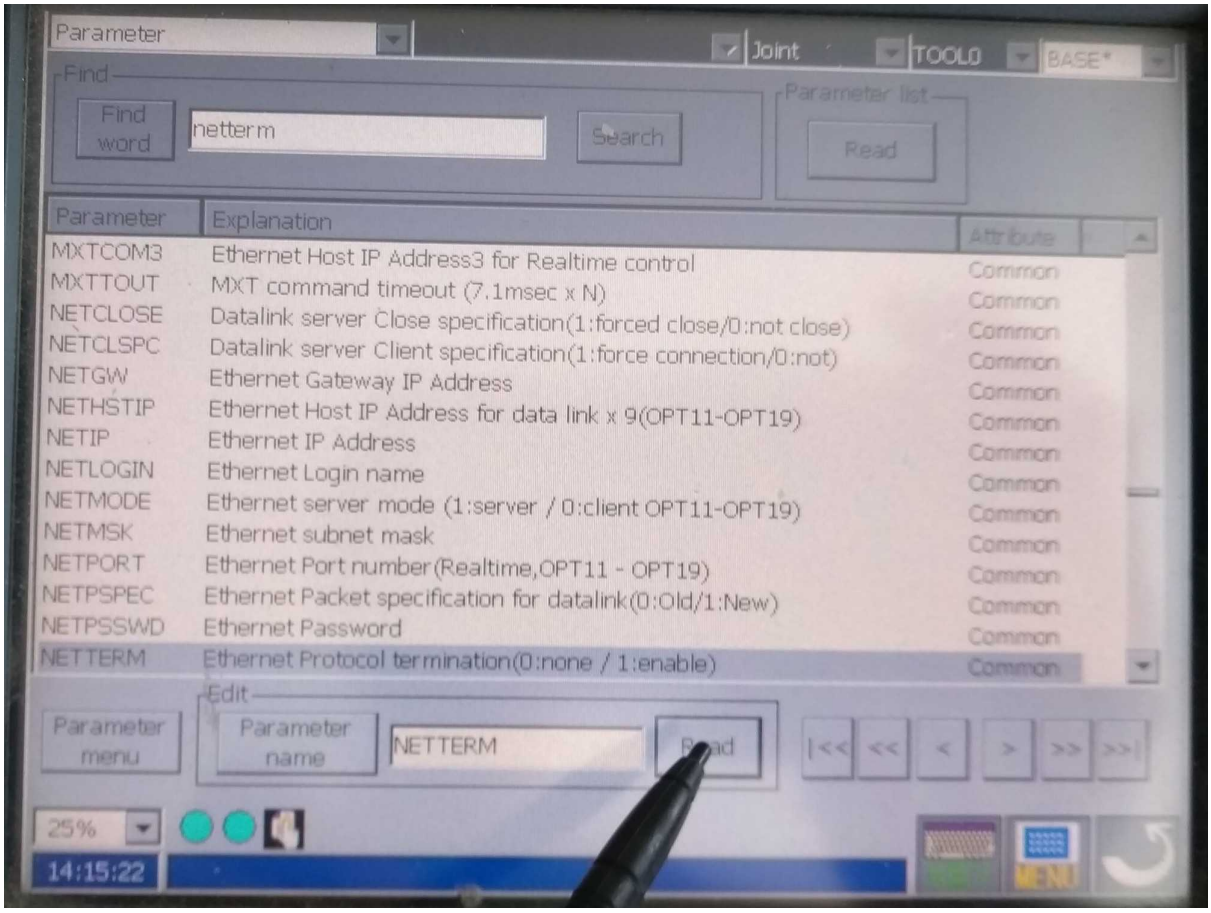


Abb. 17: Bild Mitsubishi Netterm Parameter finden

Im *Parameter edit* Dialog sehen Sie dann die derzeitige Einstellung. Die Bits 1-8 sollten auf 1 eingestellt sein. **Andernfalls müssen die Werte manuell auf 1 gesetzt und auf den Controller geschrieben werden.**

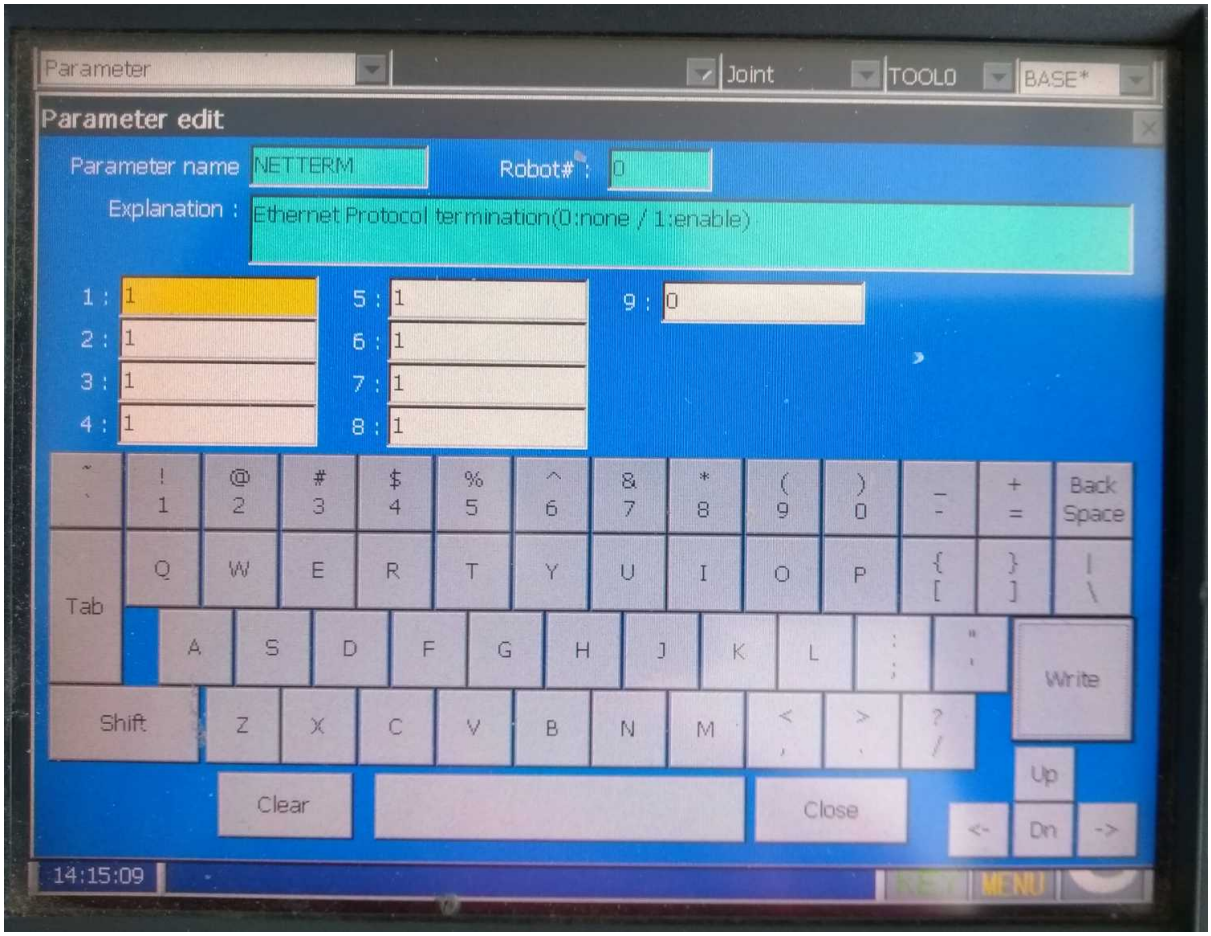


Abb. 18: Bild Mitsubishi Netterm Parameter editieren

6.5.3 Programme konfigurieren

Der Mikado-Controller bespielt den Robotercontroller während der Laufzeit mit einem Programm, das die Steuerung übernimmt. Dieses darf nicht durch andere ab Start laufende Programme blockiert werden.

Öffnen Sie aus dem Parameter Menü heraus den Dialog *Slot table parameter* .

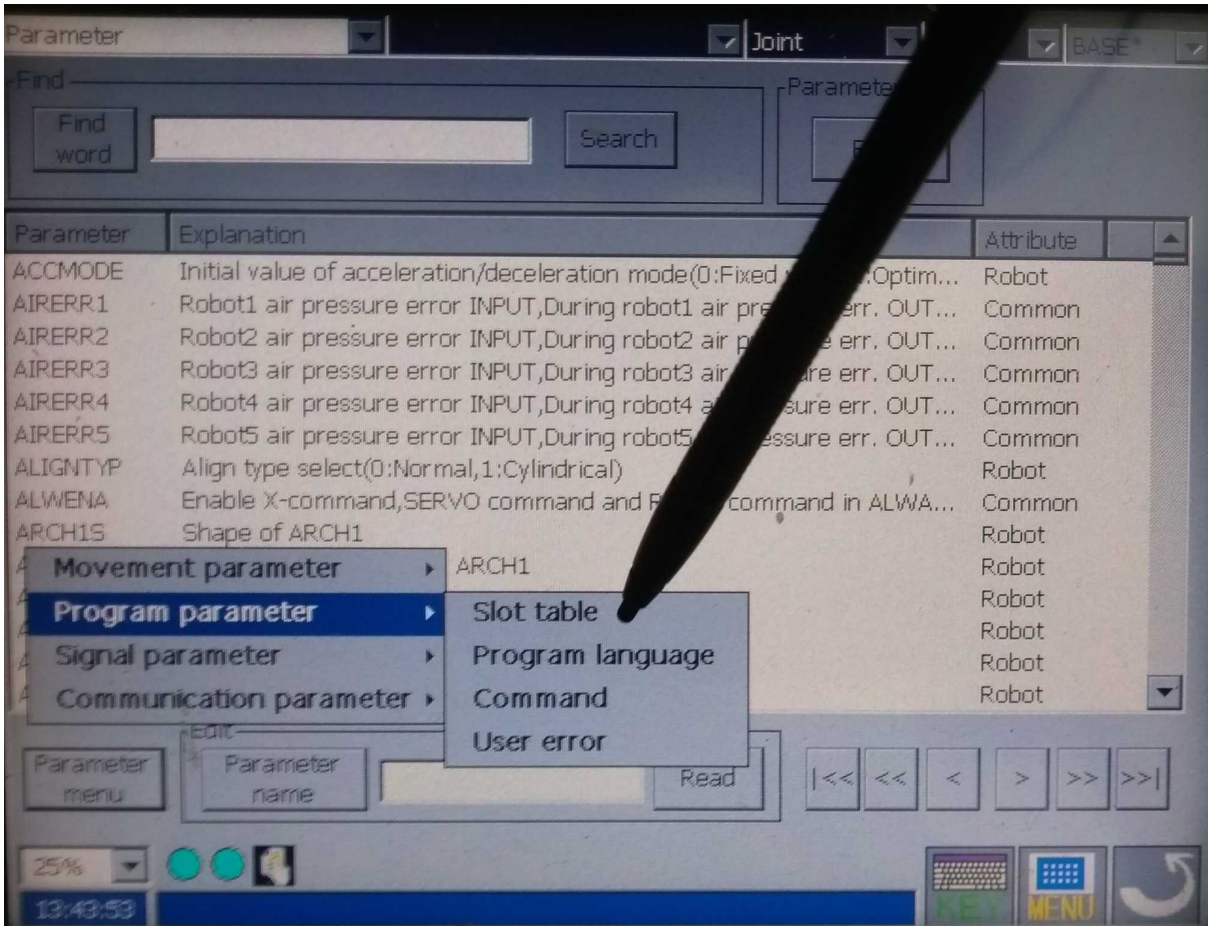


Abb. 19: Bild Mitsubishi Program Slot Dialog öffnen

Entfernen Sie nun gegebenenfalls Programme aus dem automatischen Start und starten Sie den Controller neu.

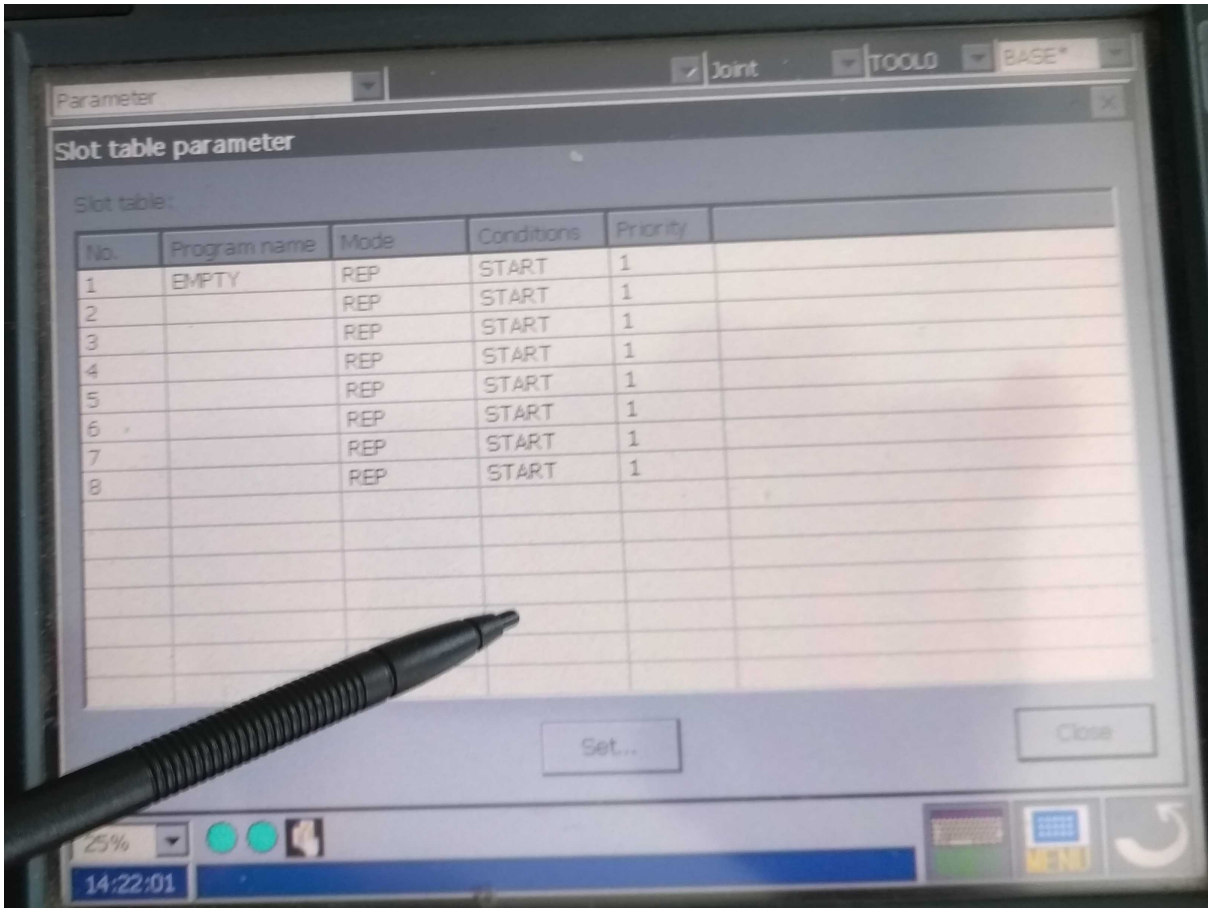


Abb. 20: Bild Mitsubishi Controller Slot table parameter

Der Mikado-Controller benötigt ausreichend freien Speicherplatz auf dem Robotercontroller. Dazu wählen Sie aus dem Auswahlmenü den Punkt *Program list*.

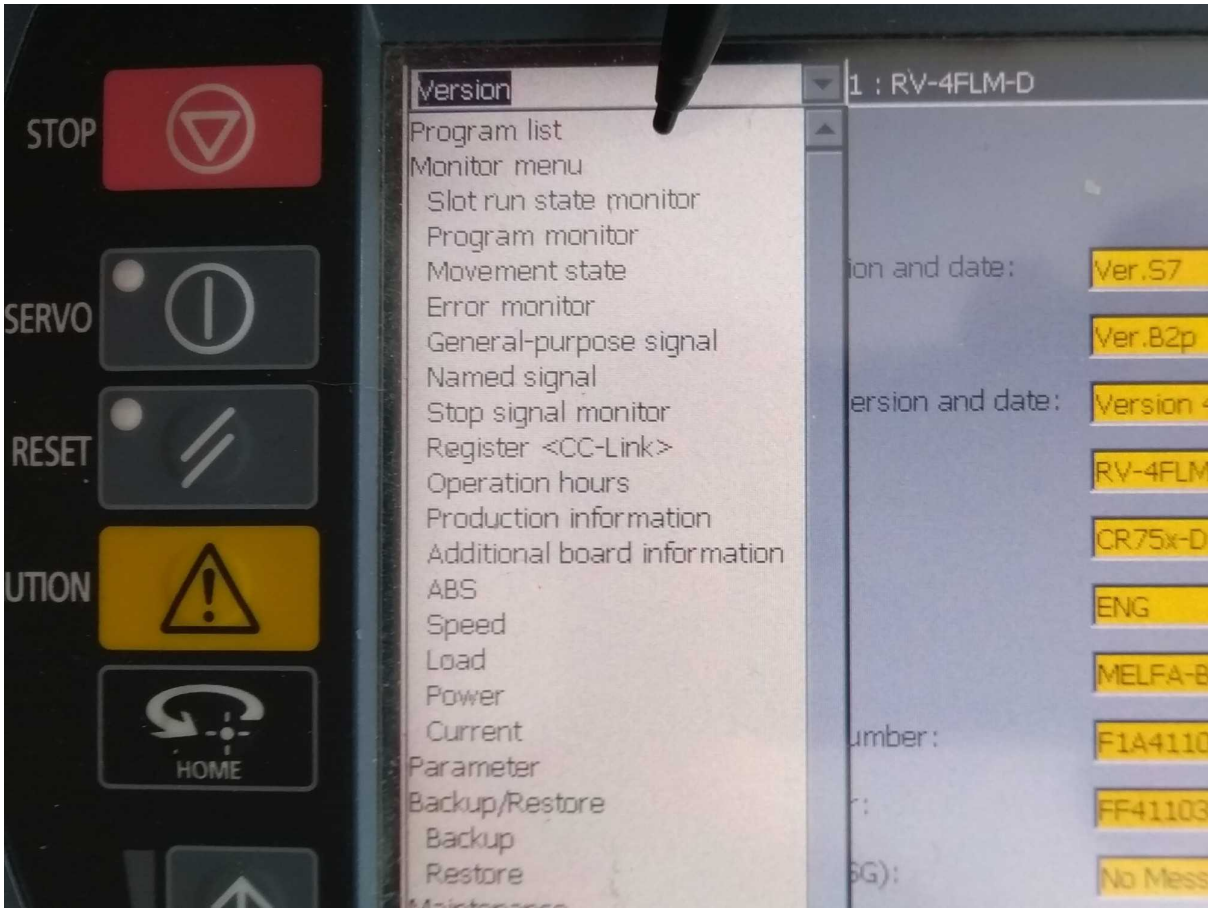


Abb. 21: Bild Mitsubishi Controller Programm Liste wählen

Es öffnet sich eine Programmübersicht, die die derzeit vorhandenen Programme und den freien Speicherplatz anzeigt.

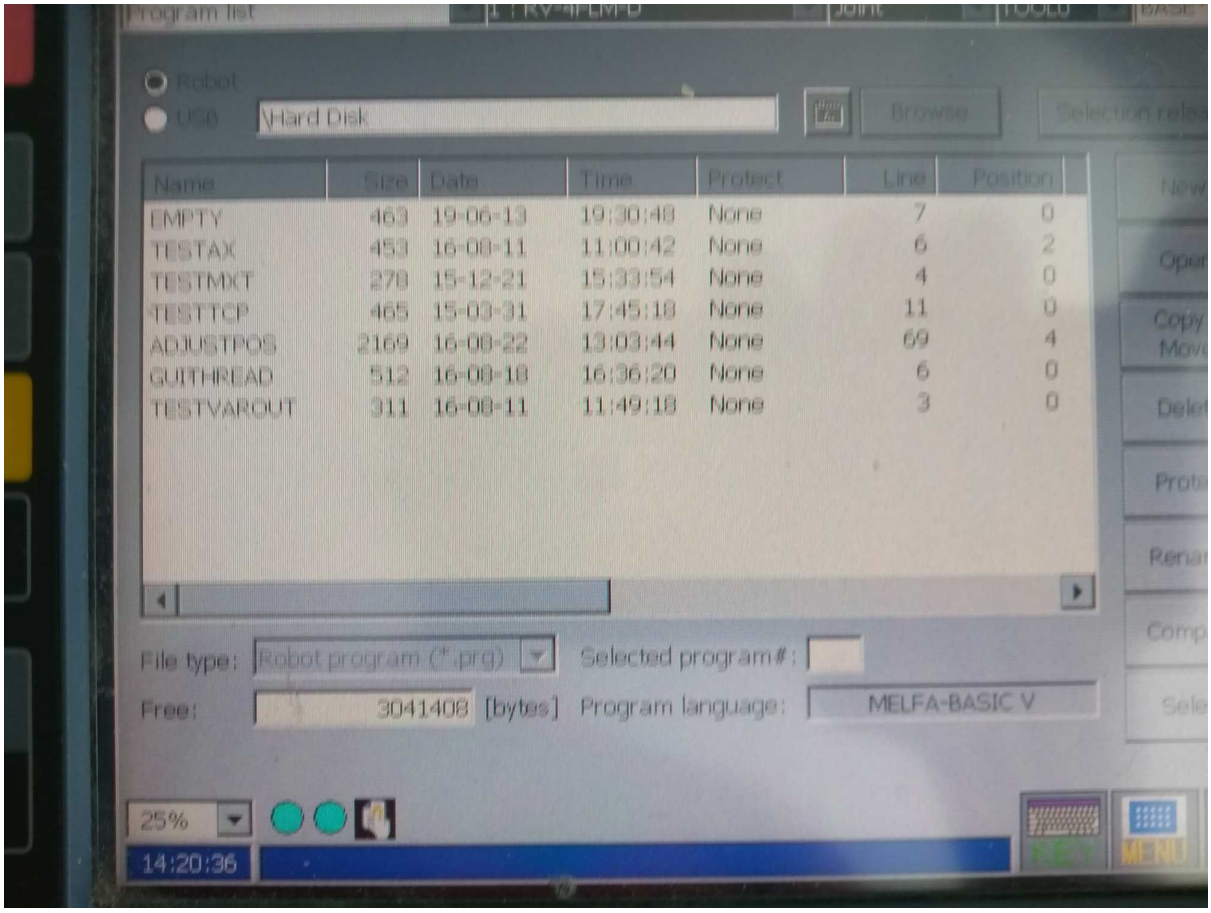


Abb. 22: Bild Mitsubishi Controller Speicherplatz

6.5.4 Troubleshooting: Verbindung testen

Fehlerbehebung, falls nach der Konfiguration und Neustart keine Verbindung zum Roboter besteht. Im Bereich der *Statusleiste* lässt sich erkennen, ob die Verbindung zum Roboter erfolgreich aufgebaut wurde.

- Prüfen: Robotercontroller und Mikado-Controller haben die gleiche Subnetzmaske und unterscheidliche feste IPs im gleichen Adressraum
- Prüfen: Die Status LED des Netzwerkports zeigt keinen Fehler an
- Prüfen: Ist der Roboter unter der eingetragenen IP per *ping* erreichbar
- Prüfen: Ist der Roboter per R3 Protokoll kommunikationsfähig
- Prüfen: Wird ein Carriage Return bei R3 Kommunikation mitgesendet

Das nötige Handwerkszeug zum Ausführen des Ping Kommandos und Senden von R3 Befehlen vom Mikado-Controller:

Auf dem Mikado-Controller muss eine Konsole gestartet werden. Der einzugebene Ping Befehl lautet dann zum Beispiel:

```
ping 10.0.0.4
```

Das Pingen kann per *Strg + C* abgebrochen werden. Eine positive Antwort würde wiefolgt aussehen:

```
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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=0.448 ms
^C
--- 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
```

Zum Versenden von R3 Kommandos öffnen Sie eine *telnet* Verbindung. Ein Verbindungstest läuft über Port 10002 und sollte als Antwort *QOk* zurückgeben:

```
telnet [Roboter IP] 10002
```

Über den Port 10003 können dann Kommandos direkt an den Roboter versendet werden:

```
telnet [Roboter IP] 10003
```

Zum Beispiel könnte der Geschwindigkeitsfaktor gelesen werden mit:

```
1;1;OVRD
```

Zum Beispiel könnte die Roboter-Hand / der Greifer mit den folgenden Befehlen gesteuert werden:

```
1;1;CNTLON
1;1;HNDON1
```

Die Antworten des Controllers müssen jeweils mit einem Carriage Return abgeschlossen sein, sprich der Cursor an den Anfang der neuen Zeile wandern!

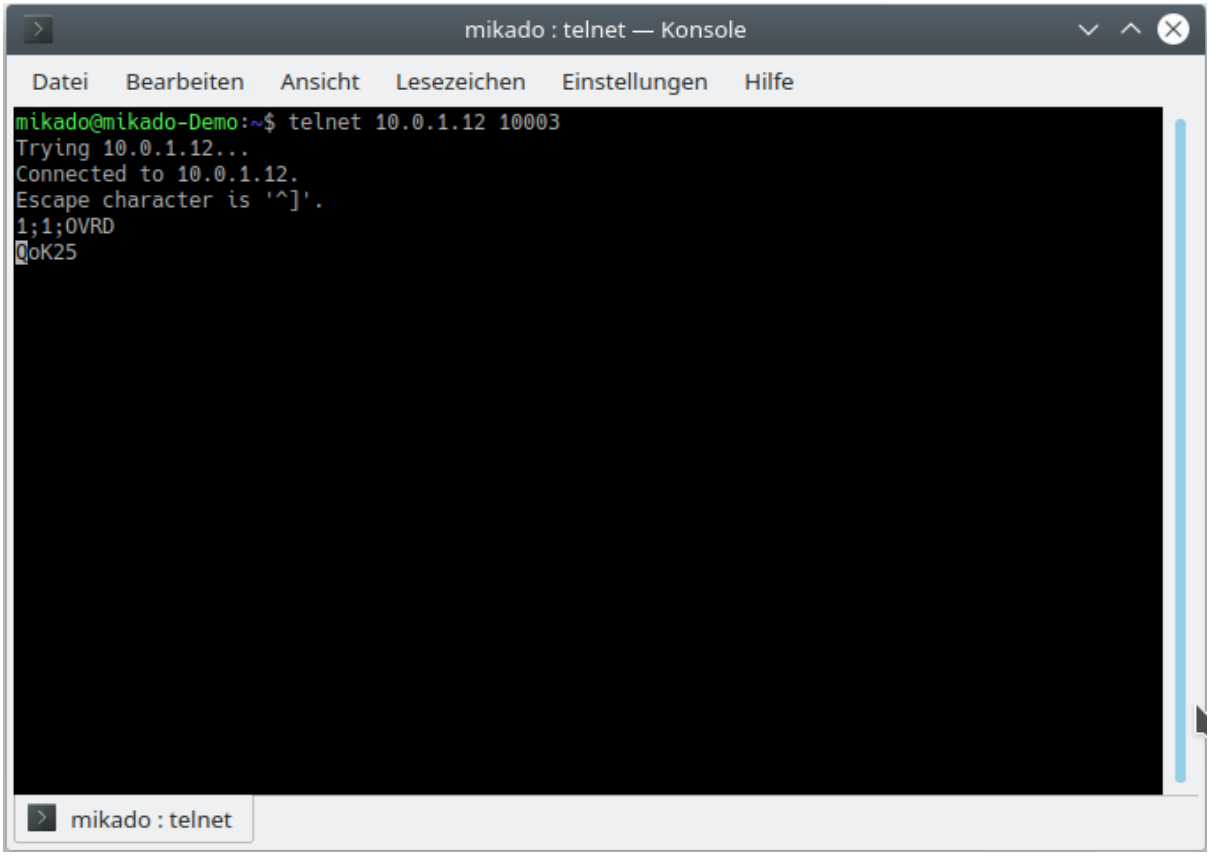


Abb. 23: Bild Mitsubishi Telnet OVRD Parameter lesen

6.5.5 Optional: Handansteuerung

Soll die Ansteuerung des Greifers per Roboter I/O erfolgen, so muss diese angepasst, bzw. getestet werden. Die Hand I/Os können hierzu verwendet werden. Haben Sie Ihren Greifer beispielsweise an Hand I/O **Port 1** angeschlossen, tragen Sie in der Greiferkonfiguration (*gripper.yaml*) die entsprechenden I/O Nummern ein. Bei Mitsubishi liegen die Hand-I/O-Nummern im Bereich 900-907, sowohl für Inputs als auch für Outputs. Alternativ können Sie natürlich genauso die General I/Os des Mitsubishi-Controllers verwenden (diese beginnen i.d.R. bei Index 0, s. Dokumentation des verwendeten Robotercontrollers).

Auszug der Beispielkonfiguration für einen Greifer an ‚Hand Port 1':

```
open_hand_signal_number: 900
close_hand_signal_number: 901
is_closed_signal_number: 900
is_open_signal_number: 901
```

Auf dem Teachpanel kann nun zunächst überprüft werden, ob die Greiferaktionen ausgelöst werden. Dazu öffnen Sie das Hand-Menü (eigener Knopf oben rechts des Teachpanels) und betätigen die Open und Close, also **+** und **-** Tasten. Stellen Sie sicher, dass die Funktion nicht invertiert ist, also Open tatsächlich ein Öffnen auslöst und kein Schließen.

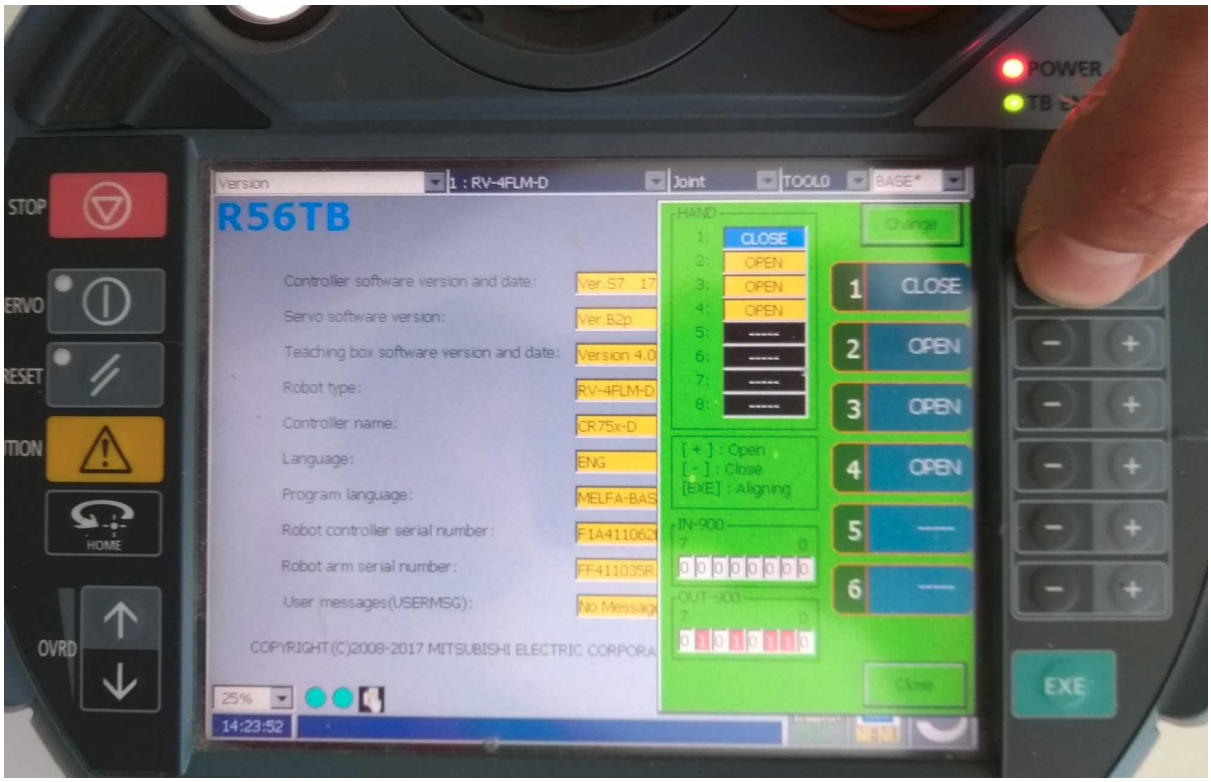


Abb. 24: Bild Mitsubishi Hand überprüfen

6.5.6 Optional: Basiskoordinatensystem anpassen

Mikado bietet beim *Erstellen einer neuen Zelle* die Möglichkeit das Zellkoordinatensystem unabhängig von dem des Roboters zu setzen. Dies ist zum Beispiel für die Überkopfmontage sinnvoll, um ein intuitives Weltkoordinatensystem zu erhalten. Um auf dem Teachpanel nun Posen im gleichen Koordinatensystem dargestellt zu bekommen, muss auch hier der Ursprung des Roboters entsprechend angepasst werden. Dies ist nicht zwingend erforderlich, erleichtert aber die Integration und das Teachen von Posen. Beim Mitsubishi Robotercontroller passen sie das Roboterkoordinatensystem wie folgt an. Wählen sie den Punkt *Movement Parameter* und dann den Unterpunkt *Tool* an.

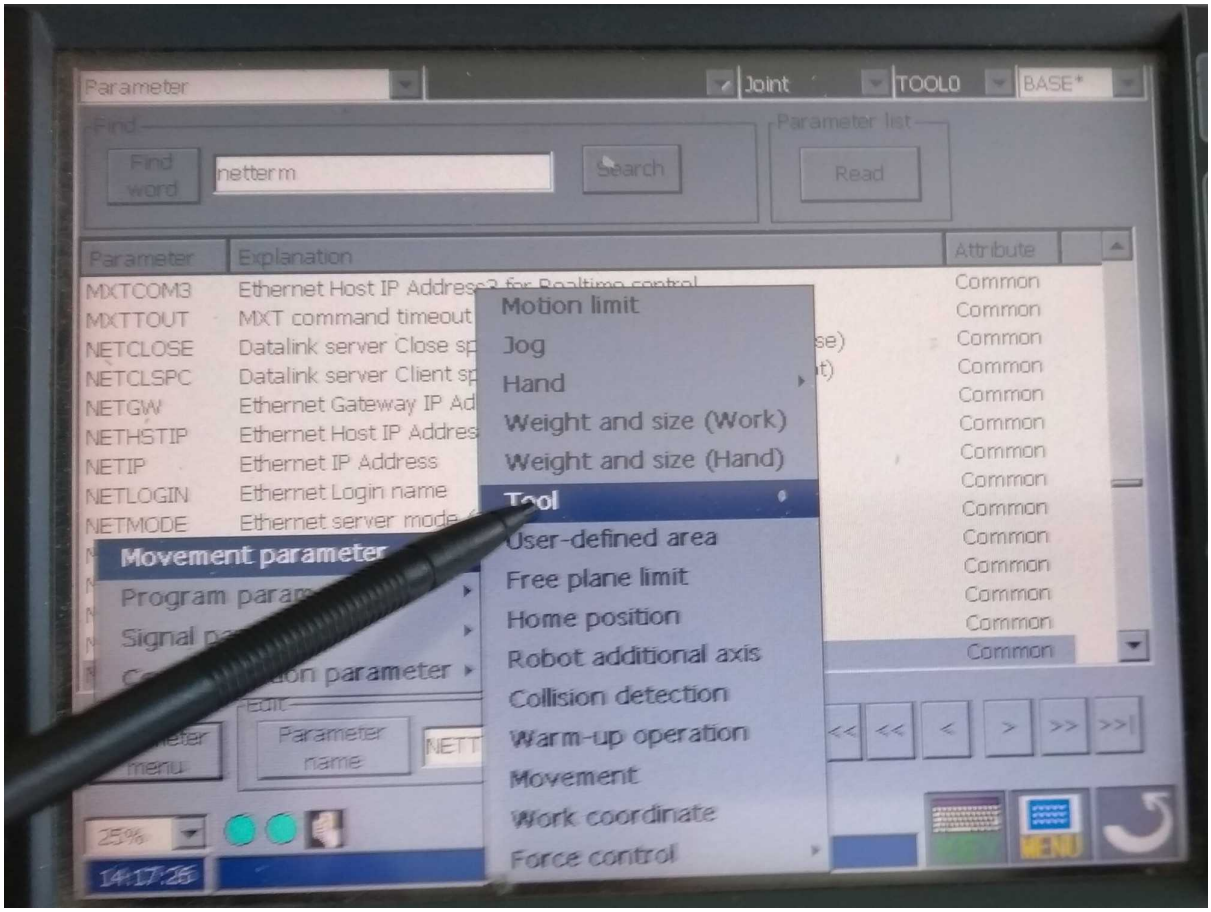


Abb. 25: Bild Mitsubishi Menü Koordinatensystem

In dem folgenden Dialog können Sie dann den Ursprung des Roboters entsprechend verändern. In dem gezeigten Beispiel ist die Basis um 90° um die z-Achse verdreht angebracht.

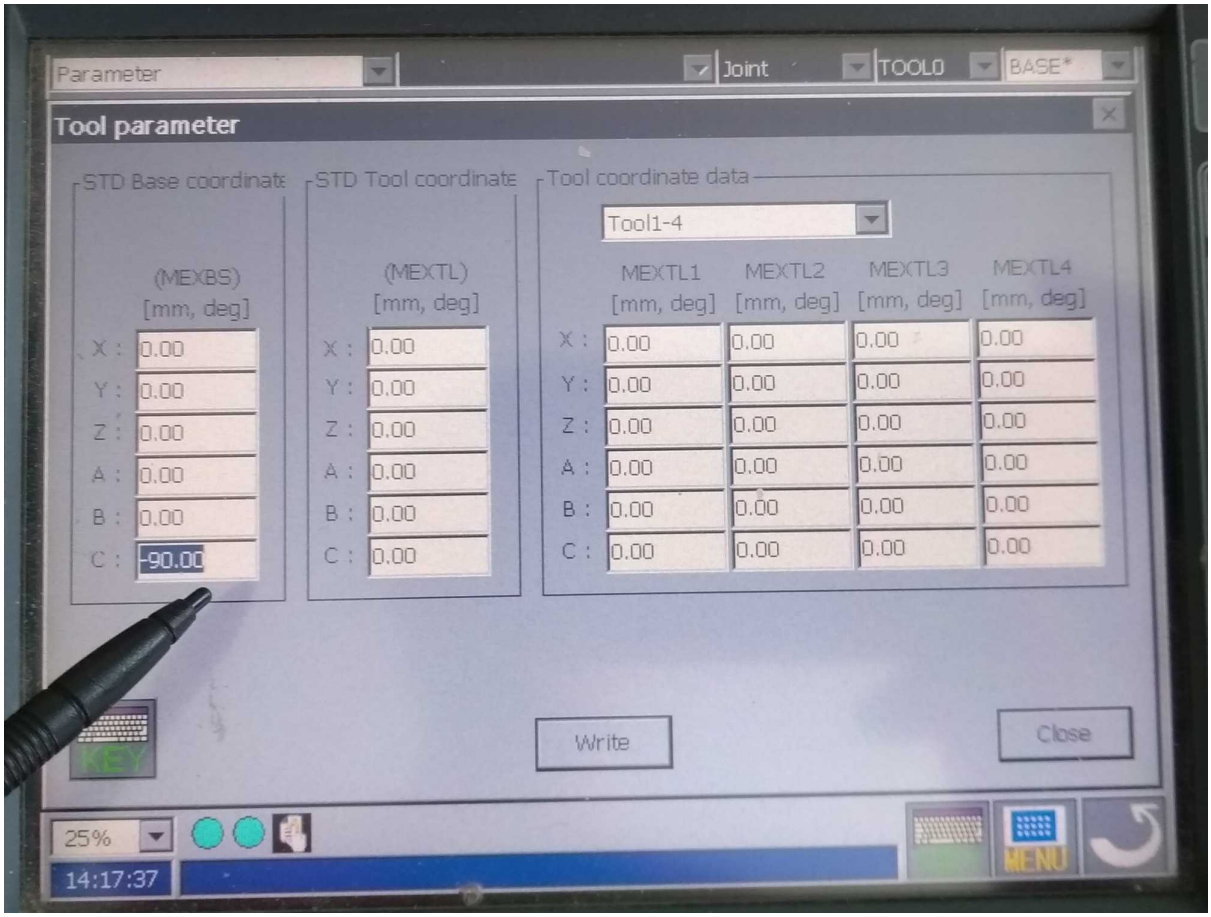


Abb. 26: Bild Mitsubishi Dialog Koordinatensystem anpassen

Die Änderungen werden per *Write* Befehl übernommen. Die Richtigkeit kann leicht auf der Mikado Roboterseite überprüft werden, die immer die derzeit ausgelesene Roboterpose anzeigt. Sie sollte nun mit der im Teachpanel angezeigten übereinstimmen.

6.6 Mitsubishi Roboter verwenden

6.6.1 Parameter und Steuerelemente

- **Geschwindigkeitsfaktor**:

  Geschwindigkeit für Roboterbewegungen setzen. Ein Klick auf *Übernehmen* setzt den eingestellten Wert auf dem Robotercontroller. Beachten Sie, dass in der *Systemkonfiguration* Höchstwerte hinterlegt sind, sodass Sie eventuell einen Grenzwert nicht überschreiten können.

- **Fordere Roboterkontrolle zurück**:

  Die Roboterkontrolle kann Mikado Control unter gewissen Umständen entzogen werden (z.B. wenn Sie am Roboter in den manuellen Modus wechseln). Drücken Sie diesen Button um die Kontrolle wieder an Mikado Control zu übergeben.

- **An Roboter senden**:

  Mitsubshi-spezifische R3-Kommandos können in das Textfeld neben diesem Button eingegeben und an den Roboter gesendet werden. Die Antwortnachricht wird danach in Rohform dargestellt.

  Mit Hilfe dieses Steuerelements können Sie direkt über die Oberfläche beliebige Befehle an den Roboter senden. So können Sie z.B. die Geschwindigkeit des Roboters steuern oder auch bestimmte Roboter-IOs

ansprechen. Weitere Informationen zu R3 und den einzelnen Kommandos finden Sie in Ihrer Roboterdokumentation.

6.6.2 Roboter einschalten

- Mikado und Roboter können gleichzeitig gestartet werden.
- Teach Pendant muss deaktiviert sein und der Controller muss im Modus „Automatic“ operieren

6.6.3 Sonstige Hinweise

- Position Teachen:
  - In den manuellen Modus wechseln.
  - Teach Pendant aktivieren und Servos einschalten.
  - Position im Jog-Modus anfahren.
  - In Mikado „Aktuelle Roboterpose übernehmen“ anklicken.
- Werte der aktuellen Roboterpose:
  - Roboterpose des Controllers kann von der Pose abweichen, die in der Mikado-Oberfläche dargestellt wird. Dies ist jedoch nur relevant, falls Sie native Roboterprogramme in Kombination mit Mikado verwenden wollen.
  - Achten Sie darauf, dass das Basis-Koordinatensystem korrekt definiert ist (Mitsubishi Teach Pendant: Parameters → Movement Parameters → Tool). Stellen Sie hier die Transformation von Weltkoordinaten zum Roboter-Koordinatensystem ein.
  - x-/y-/z-Koordinate können aufgrund der Werkskalibrierung einen leichten Offset aufweisen.

6.7 Nachi Controller einrichten

6.7.1 Netzwerkkonfiguration

Für die Kommunikation zwischen Roboter Controller und Mikado Controller wird eine Netzwerkverbindung zwischen den beiden Systemen benötigt. Verbinden Sie hierfür zunächst einen Ethernet Port des Nachi Controllers mit einem Ethernet Port des Mikado Controllers. Anschließend muss sowohl für den Mikado Controller als auch für den Roboter Controller eine feste IP-Adresse und eine zugehörige Subnetzmaske eingestellt werden. Die Netzwerkkonfiguration des Mikado Controllers wird in Sektion *Netzwerkadapter* näher erläutert. Wir arbeiten an einer detaillierten Beschreibung zur Netzwerkkonfiguration des Nachi Controllers. Vorerst sind Hinweise zur Netzwerkkonfiguration des Controllers im Benutzerhandbuch des Herstellers nachzulesen.

Hinweis: Beachten Sie, dass die IP-Adressen von Mikado Controller und Nachi Controller im selben Adressraum liegen und die Subnetzmasken übereinstimmen.

Tragen Sie die IP-Adresse des Roboter Controllers anschließend im Mikado Assistant unter „Parameter“ -> „Roboter“ in das entsprechende Feld ein und speichern Sie die Einstellungen.

6.7.2 Vorbereitung

Diese Beschreibung wurde bei einem Nachi Controller Ver. `4 63 05-30-2019` erstellt. Bei abweichenden Versionen kann die Anleitung unvollständig sein. Bei Problemen wenden Sie sich bitte an den Mikado Support.

Die Vorbereitung zur Übertragung der Programme und Usertasks wird so beschrieben, dass sie mit Hilfe des Teach Panels umgesetzt werden kann. Zuallererst empfiehlt es sich die Taste `R` zu drücken und dann `314` im sich öffnenden Feld einzugeben. Dann fragt der Controller nach dem Passwort für die Berechtigung `Specialist`, die man für Teile der Einrichtung des Nachis benötigt. Als Passwort muss hierbei `12345` eingegeben und anschließend die `Enter` Taste gedrückt werden. Auf dem Teach Panel kommt man durch das Drücken der `R` Taste im Menü einen Schritt zurück. Die meisten Tasten auf dem Teach Panel sind aus Platzgründen doppelt belegt. Wird `Enable` und eine weitere Taste gleichzeitig gedrückt, wird die grüne Funktion auf der Taste ausgeführt. Wird beispielweise die `0` Taste zusammen mit `Shift` gedrückt ist das Ergebnis ein + (Pluszeichen) und ohne eine `0` (Skalar).



Abb. 27: Foto eines Nachi Teach Panels in dem die Doppelbelegung der Tasten zu sehen ist

6.7.3 Kopieren der Programme/Usertasks

Damit der Nachi Roboter von Mikado gesteuert werden kann, müssen entsprechende Programme auf den Roboter Controller kopiert und dort kompiliert werden. Sie finden die benötigten Programme auf dem Mikado Controller unter

`/opt/isys-vision/mikado/robot_programs/nachi/`

oder direkt über den entsprechenden Button im Mikado Assistant („Parameter“ -> „Roboter“ -> „Verzeichnis öffnen“). Das Kompilieren der Programme funktioniert nur, wenn der Name des Roboters Teil des Programmnamens ist. Im hier dargestellten Beispiel handelt es sich um einen `MZ07L-01`. Sollten Sie einen anderen Roboter besitzen passen Sie bitte den Namen der Roboterprogramme an. Kopieren Sie nun zunächst die Programme aus dem oben genannten Verzeichnis zur Übertragung auf den Nachi auf einen USB-Stick. Am Nachi gibt es 2 USB Schnittstellen, an denen Daten übertragen werden können. Eine am Teach Panel und eine am Controller selbst. Die Erfahrung hat gezeigt, dass der Port am Teach Panel unzuverlässig funktioniert, weshalb die Übertragung am Controller empfohlen wird.

Zum Kopieren der Dateien klicken Sie dann bitte auf dem Teach Panel auf:

`Service Utilities` → `7 File Manager` → `1 File Copy` → wählen Sie unter `Device (src)` Ihren USB-Stick aus und klicken auf den Pfeil nach rechts auf dem Teach Panel → markieren Sie die Programme auf der rechten Seite, indem Sie auf `Enter` klicken → `Execute`.

6.7.4 Kompilieren der Programme/Usertasks

Der Treiber besteht aus 2 Roboterprogrammen die nacheinander aufgerufen werden sowie zwei parallel laufenden Usertasks:

- `MZ07L-01-A.008` → Roboterprogramm zum Zurücksetzen der Variablen
- `MZ07L-01-A.012` → Roboterprogramm zum Steuern des Roboters
- `USERTASK-A.000` → Usertask zum Empfangen der Daten von Mikado
- `USERTASK-A.001` → Usertask zum Senden der Daten zu Mikado
- `VAR.INC` → Liste von benötigten Variablen

Die Liste mit zusätzlichen Variablen muss nur kopiert, aber nicht kompiliert werden. Zum Kompilieren der anderen Dateien bitte wie folgt vorgehen:

`Service Utilities` → `15 ASCII File Edit` → `Datei auswählen/markieren` und mit `Execute` öffnen → `Complete` → Die Frage ob die Datei kompiliert werden soll bestätigen. Erscheint die Abfrage nach dem Bestätigen nicht, muss überprüft werden, ob Name der Datei stimmt (stimmt der Name des Roboters, stimmt die Endung, . . .).

6.7.5 Einstellen der Priorität der Usertasks

Die Usertasks laufen parallel zueinander. Durch das Verändern der Priorität kann man dem Empfangen der Daten eine höhere Wichtigkeit einräumen als dem Senden der Daten. Da Trajektorien in Mikado häufig über 100 Joint Posen beinhalten, muss dem Empfangen von Daten eine höhere Priorität zugewiesen werden. Dies funktioniert wie folgt:

`Service Utilities` → `4 Monitor 2` → `37 User Task` → `1 User Task Monitor`.

Hier muss die Priorität der User Tasks verändert werden. Hierfür die Priorität von `Prog 0` auf `6` und die von `Prog 1` auf `4` stellen. Dazu muss die Zahl im Feld `Priority` markiert sein und kann nun durch das Klicken der Taste mit dem Blatt und Stift (neben der `FN` Taste) auf dem Teach Panel geändert werden. Nach Änderung der Prioritäten den Bearbeitungsmodus durch erneutes Klicken auf den Button mit dem Blatt und Stift beenden.

Möchte man einen User Task stoppen, so müssen die `Enable` Taste und `Off` (liegt auf der Ziffer `2`) gedrückt werden, da die `Off` Funktion grün hinterlegt ist. Nach Starten des Controllers werden Usertasks von selbst gestartet.

6.7.6 Starten des Roboterprogramms

Nach dem Umlegen des Hebels am Teach Panel und Controller auf `Playback` Betrieb kann das Programm gestartet werden. Durch das Klicken des Programm Buttons auf dem Teach Panel kann das richtige Programm angewählt werden. Hiermit kann das Programm auch nach einem Stopp durch beispielweise einen Not-Aus, zurückgesetzt werden. Der Roboter lässt sich durch das Klicken von `Enable` und `Motor ON` einschalten. Ist das richtige Programm angewählt und der Roboter bestromt kann das Programm durch das gleichzeitige Drücken von `Enable` + `Shift` + `GO` gestartet werden. Nun kann der Nachi von Mikado gesteuert werden.

Die Steuerung des Greifers (oder Saugers) erfolgt über Outputs, diese stellen Sie im Mikado-Zellverzeichnis in der Datei `config/gripper.yaml` ein:

```
open_hand_signal_number: 23
close_hand_signal_number: 24
```

6.8 Stäubli Controller einrichten

Der Treiber kann mit einem 6-Achsroboter von Stäubli und der CS8 oder CS9 Steuerung verwendet werden. Es wird mindestens VAL 3 Version s7.7.2 benötigt.

6.8.1 Netzwerkkonfiguration

Für die Kommunikation zwischen Roboter Controller und Mikado Controller wird eine Netzwerkverbindung zwischen den beiden Systemen benötigt. Verbinden Sie hierfür zunächst einen Ethernet Port des Stäubli Controllers mit einem Ethernet Port des Mikado Controllers. Anschließend muss sowohl für den Mikado Controller als auch für den Roboter Controller eine feste IP-Adresse und eine zugehörige Subnetzmaske eingestellt werden. Die Netzwerkkonfiguration des Mikado Controllers wird in Sektion *Netzwerkadapter* näher erläutert. Wir arbeiten an einer detaillierten Beschreibung zur Netzwerkkonfiguration des Stäubli Controllers. Vorerst sind Hinweise zur Netzwerkkonfiguration des Controllers im Benutzerhandbuch des Herstellers nachzulesen.

Hinweis: Beachten Sie, dass die IP-Adressen von Mikado Controller und Stäubli Controller im selben Adressraum liegen und die Subnetzmasken übereinstimmen.

Tragen Sie die IP-Adresse des Roboter Controllers anschließend im Mikado Assistant unter „Parameter“ -> „Roboter“ in das entsprechende Feld ein und speichern Sie die Einstellungen.

6.8.2 Installation

Damit der Stäubli Roboter von Mikado gesteuert werden kann müssen zwei Ordner mit VAL3 Programmen auf den Stäubli Controller kopiert werden, Sie finden diese Programme auf dem Mikado Controller unter

`/opt/isys-vision/mikado/robot_programs/staubli`

oder mit Hilfe des entsprechenden Button im Mikado Assistant („Parameter“ -> „Roboter“ -> „Verzeichnis öffnen“). Zum Kopieren kann entweder ein USB-Stick (< 2GB bei CS8), eine FTP-Verbindung oder der Transfermanager im Stäubli Robotics Studio verwendet werden.

6.8.3 Serversockets einrichten

Nun müssen noch zwei Sockets eingerichtet werden, das Vorgehen unterscheidet sich leicht zwischen CS8 und CS9:

Bei *CS8* wählen Sie ausgehend vom Hauptmenü Control panel –> I/O –> Socket –> TCP Servers

Bei *CS9* wählen Sie ausgehend von 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 Starten

Nun kann der Treiber gestartet werden, auch hier unterscheidet sich das Vorgehen leicht zwischen CS8 und CS9:

Bei CS8 den *Start/Run* Knopf drücken, *ros_server* auswählen und mit F8 (Ok) bestätigen.

Bei CS9 drücken Sie VAL# –> Memory –> *ros_server* auswählen –> Play

6.8.5 Greifer und IOs einrichten

Greifer und IOs können bei Stäubli über Modbus/TCP oder über eine Feldbusverbindung (zB ProfiNet) geschrieben werden, bitte konsultieren Sie die Stäubli Dokumentation für die Einrichtung auf dem Stäubli und kontaktieren Sie uns für Details für die Einrichtung auf Seiten von Mikado.

6.9 Techman Controller einrichten

Der Techman Controller ist in wenigen Schritten eingerichtet. Im Betrieb gilt es zum Beispiel für native Programme einige Details zu beachten. Getestet wurde die nachfolgende Konfiguration mit den Controller TMflow Software-Versionen 1.72, 1.76 und 2.14.

6.9.1 Netzwerkkonfiguration

Zunächst muss eine Ethernet Verbindung zwischen Roboter Controller und Mikado Controller hergestellt werden. Dies muss beim Techman Controller *zwingend auf Port 1* geschehen, da dieser alleinig mit dem Listen Node funktioniert. Es muss dann sowohl für den Mikado Controller als auch für den Roboter Controller eine feste IP-Adresse und eine zugehörige Subnetzmaske eingestellt werden. Die Netzwerkkonfiguration des Mikado Controllers wird in Sektion *Netzwerkadapter* näher erläutert. Für den Techman Controller können Sie unter dem Menüpunkt *System*, dann *Network* die TCP/IP Einstellungen anpassen. Weitere Hinweise zur Netzwerkkonfiguration des Controllers sind im Benutzerhandbuch des Herstellers nachzulesen.

Hinweis: Beachten Sie, dass die IP-Adressen von Mikado Controller und Techman Controller im selben Adressraum liegen und die Subnetzmasken übereinstimmen.

Hinweis: Der Port 1 entspricht der *Local Area Connection 9* auf der Network Seite. Der Robotercontroller sollte bei erfolgreicher Kommunikation vom Mikadocontroller aus per *ping* erreichbar sein.

6.9.2 Modbus Verbindung herstellen

Unter dem Menüpunkt *Settings*, dann *Connection* kann der derzeitige Modbus-Zustand abgerufen und eingestellt werden. Es muss die IP-Adresse des Mikado-Controllers auf dem Techman-Controller zugelassen werden und der Modbus-Zustand muss aktiviert sein. Dafür sind folgende Schritte notwendig:

- Zunächst muss der Modbus-Zustand deaktiviert werden.
- Dann unter *IP-Filter* die IP-Adresse des Mikado-Controllers mit aktivierter *writer-Permission* hinzufügen.
- Und schließlich den Modbus-Zustand wieder aktivieren. Hinter *Status* sollte *Enable* stehen.

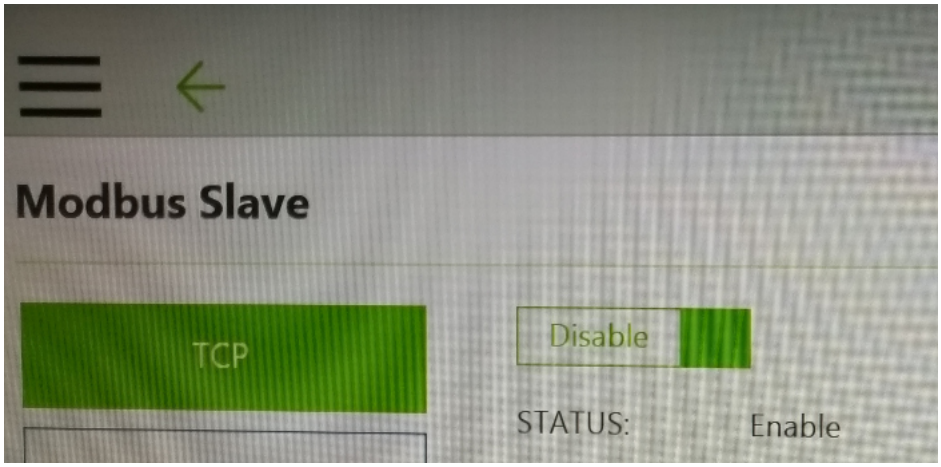


Abb. 28: Bild: Einstellung Modbus Enable

Hinweis: Oben rechts in der Ecke lässt sich die Konsole öffnen. Deaktiviert und aktiviert man den Modbus Server wieder, so wird auch dessen derzeitige IP dort hinein geschrieben. Nach Ändern der TCP/IP Verbindungseinstellungen muss auch der Modbus-Server neu gestartet werden.

6.9.3 Projekt erstellen

Mikado benötigt einen Projekt Flow mit Listen Node, um zur Laufzeit kommunizieren zu können. Dieser muss aktiv sein, damit Mikado Roboterfahrten ausführen kann. Eigene Programme können aber ebenfalls ausgeführt werden (s.u.). Ein Minimalablauf bestünde aus *Start*, *Listen* und *Stop* Knoten. Ein etwas komplexerer Ablauf ist im Folgenden dargestellt.

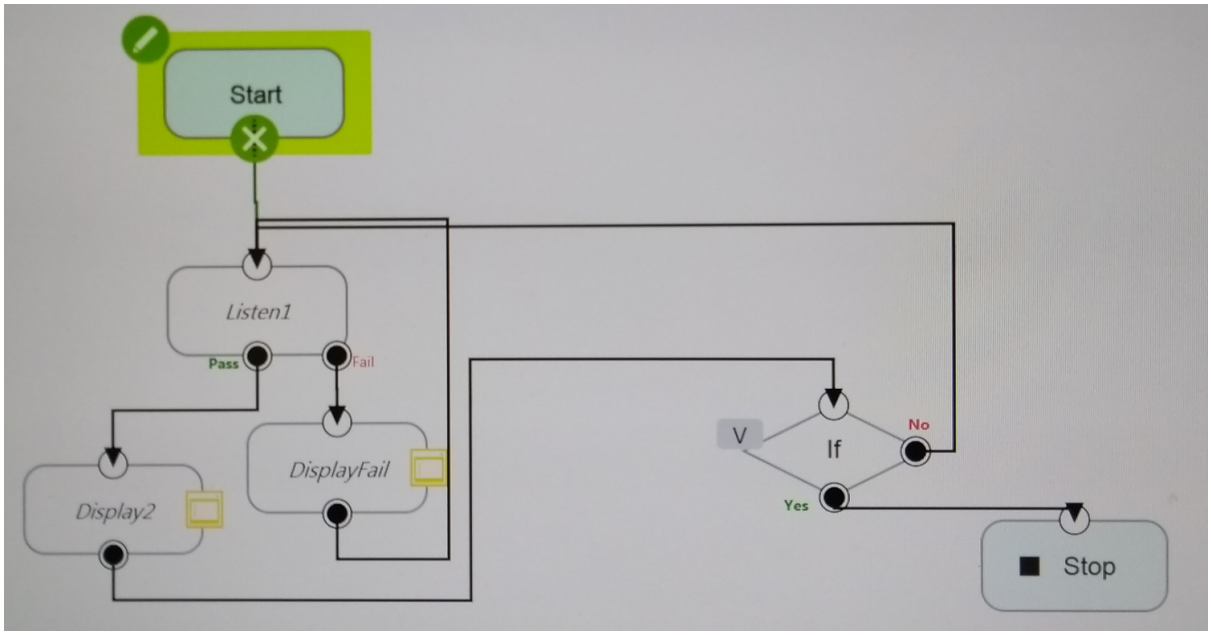


Abb. 29: Listen Node mit Debug Display und Möglichkeit zurückzukehren

Der Ablauf erlaubt per Display Knoten Debug nachrichten für den Nutzer darzustellen. Wichtiger ist aber noch der Verlauf über einen *if* Knoten. Dieser testet den Zustand der *var_stop* Variable und entscheidet wohin das Programm weiter führt. Die Variable kann aus dem Mikado Programm heraus geschrieben werden, siehe *Roboterprogramm ausführen* und so der Techman Flow gesteuert werden.

6.9.4 Projekt testen

Per *Diagnosis*, dann *Stepping* lässt sich das Programm/Flow testweise starten und Schritt für Schritt durchlaufen. In der Konsole sollte auch die TCP/IP Verbindung mit Adresse angezeigt werden, sobald der Listen Node erreicht wird. Ab diesem Punkt sollte auch die Verbindung zu Mikado funktionieren. D.h. startet man nun eine Mikadozelle oder läuft diese bereits, so sollte die Status Ampel für den Roboter auf grün springen.

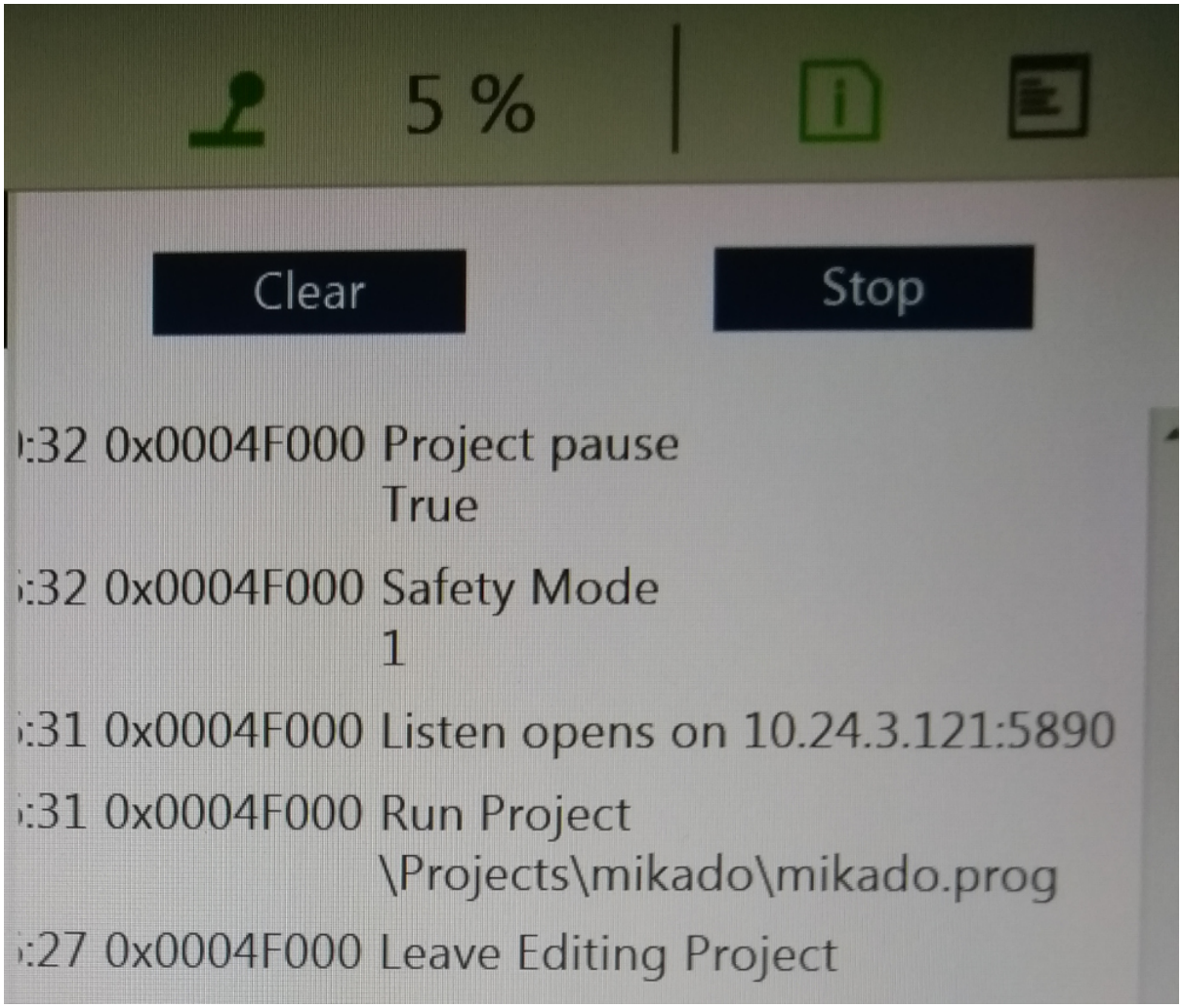


Abb. 30: Listen Node in der Konsole

6.9.5 Projektgeschwindigkeit einstellen und Automatik-Modus aktivieren

Auf der Seite *Run Settings* können alle ausführbaren Projekte eingesehen werden. Die Anzeige **current** bei einem Projekt bedeutet, dass dieses Projekt gerade ausgeführt wird, die Anzeige **tested** bedeutet, dass dieses Projekt im Automatik-Modus ausgeführt werden darf.

Damit das Projekt im Automatik-Modus ausgeführt werden kann, muss es den Testlauf bestehen. Der Testlauf wird im manuellen Modus durchgeführt. Hierfür sind folgende Schritte notwendig:

- Auf der Seite *Project* das gewünschte Projekt öffnen
- Auf der geöffneten Projekt-Seite den Play-Button auf dem Teachpanel drücken.
- Geschwindigkeit am Teachpanel mit **+/-** auf den gewünschten Wert einstellen.
- Nachdem Sie Ihre bevorzugte Geschwindigkeit eingestellt haben, halten Sie die M/A-Modus-Schalttaste gedrückt (halten Sie das Projekt nicht an).
- Wenn die beiden Anzeigen am Teachpanel zu blinken beginnen, drücken Sie die Tasten **+/-** in dieser Reihenfolge: **+ - + + -**.
- Jetzt kann das Projekt mit **Stop** angehalten werden
- Der Roboter speichert die Geschwindigkeit zu diesem Zeitpunkt als Projektgeschwindigkeit.
- Prüfen Sie auf der *Run Settings*, dass jetzt beim Projekt die neue Geschwindigkeit angezeigt wird

- Wechseln sie jetzt in den Auto-Modus: **M/A + - + + -**

Hinweis: Denken Sie daran, dass die Projektgeschwindigkeit wieder auf 10 % zurückgesetzt wird, wenn Sie den Probelauf erneut im manuellen Modus durchführen. Sie müssen diese Schritte dann erneut ausführen, um die Projektgeschwindigkeit einzustellen.

6.9.6 Troubleshooting

Da das notwendige Roboterprogramm für die Kommunikation mit Mikado eine Dauerschleife beinhaltet, wird der Testlauf niemals enden. Um den **tested**-Status dennoch zu erlangen und somit das Programm im Automatik-Modus starten zu können kann das neue Programm vorerst aus den Bausteinen

Start -> Wait -> Stop

bestehen. Dann werden folgende Schritte ausgeführt:

- Der Testlauf wird gestartet (Play-Button auf dem Teachpanel bei geöffneten Projekt-Seite drücken).
- Während der Wait-Block aktiv ist, die Projektgeschwindigkeit einstellen und mit **M/A + - + + -** bestätigen.
- Nach Ablauf der Zeit, ist das Projekt **tested**.

Da der **tested**-Status durch Änderungen nicht verloren geht, kann nun der Wait-Block durch einen Listen-Block (oder durch mehrere Blöcke wie in der beispielhaften Abbildung *Listen Node mit Debug Display und Möglichkeit zurückzukehren*) ersetzt werden.

6.9.7 Techman Variablen schreiben und Programme ausführen

Zum Variablen schreiben und Programme ausführen benutzt man im Mikado Programm den *Roboterprogramm ausführen* Task. Native Roboterprogramme müssen durch Techmans sequentielle Flow Programmierung etwas anders als bei anderen Robotern ausgeführt werden. Mikado schreibt die Variablen und verlässt dann den Listen Node, wodurch die Kontrolle abgegeben wird. Mithilfe der geschriebenen Variablen kann das Techman Programm/Flow dann zB. in einem Subflow weitere Fahrten ausführen. Schließlich muss auf Techman Seite wieder in einen Listen Node gegangen werden.

6.10 Universal Robots Controller einrichten

In diesem Abschnitt wird beschrieben, welche Vorbereitungen und Einstellungen notwendig sind, um einen UR-Roboter mit Mikado zu steuern.

6.10.1 Systemanforderungen

Die Mikado-Anbindung für Universal Robots ist kompatibel mit UR-Robotern der CB3-Serie ab V3.7 und der e-Serie ab V5.1.

6.10.2 Netzwerkkonfiguration

Für die Kommunikation zwischen Roboter Controller und Mikado Controller wird eine Netzwerkverbindung zwischen den beiden Systemen benötigt. Verbinden Sie hierfür zunächst einen Ethernet Port des Universal Robots Controllers mit einem Ethernet Port des Mikado Controllers. Anschließend muss sowohl für den Mikado Controller als auch für den Roboter Controller eine feste IP-Adresse und eine zugehörige Subnetzmaske eingestellt werden. Die Netzwerkkonfiguration des Mikado Controllers wird in Sektion *Netzwerkadapter* näher erläutert.

Die IP-Adresse des UR-Controllers kann über das Menü *Einstellungen* (Listen-Icon rechts oben) der Polyscope Oberfläche erreicht werden. Dort kann unter der Rubrik *System* → *Netzwerk* → *Statische IP-Adresse* eine IP-Adresse vergeben werden.

Beachten Sie, dass die IP-Adressen von Mikado Controller und Universal Robots Controller im selben Adressraum liegen und die Subnetzmasken übereinstimmen. Die IP-Adressen der Geräte dürfen nicht identisch sein.

> **Warnung:** Im Bereich *Installation* darf die Schnittstelle *Feldbus* → *Ethernet/IP* nicht aktiviert werden, sie stört die Netzwerkverbindung zum Mikado Controller.

Tragen Sie die IP-Adresse des Roboter-Controllers anschließend im Mikado Assistant unter „Roboter“ -> „Verbindung“ in das entsprechende Feld ein und speichern Sie die Einstellungen.

6.10.3 Kommunikationsprogramm einrichten

Damit der Roboter von Mikado gesteuert werden kann muss ein Kommunikationsprogramm auf den Universal Robots Controller kopiert und dort eingerichtet werden. Führen Sie hierfür die folgenden Schritte durch:

- Kopieren Sie das Programm URCap „ExternalControl“ auf einen USB Stick. Sie finden das Programm über den Button im Mikado Assistant („Parameter“ -> „Roboter“ -> „Verzeichnis öffnen“) oder direkt in folgendem Verzeichnis: `/opt/isys-vision/mikado/share/ur_robot_driver/resources/externalcontrol-X.X.X.urcap`
- URCap auf dem UR-Controller installieren:
  - CB3: *Setup Robot* → *URCaps* → +-Button → externalcontrol-X.X.X.urcap auswählen → Neustart
  - e-Serie: Oben rechts auf das Menüsymbol → *Settings* → *System* → *URCaps* → +-Button → externalcontrol-X.X.X.urcap auswählen → Neustart
- URCap konfigurieren
  - *Program Robot* → *Installation* → *External Control*
  - IP des Mikado-Controllers eintragen
- Neues UR Programm anlegen
  - Knoten „*External Control*“ (unter *Structure* gelistet) einfügen
  - Programm speichern (z.b. unter dem Programm-Namen *mikado*)
- PolyScope 5.x: *Remote Control Mode* aktivieren
  - Oben rechts auf das Menüsymbol → *Settings* → *System* → *Remote Control* → *Enable*

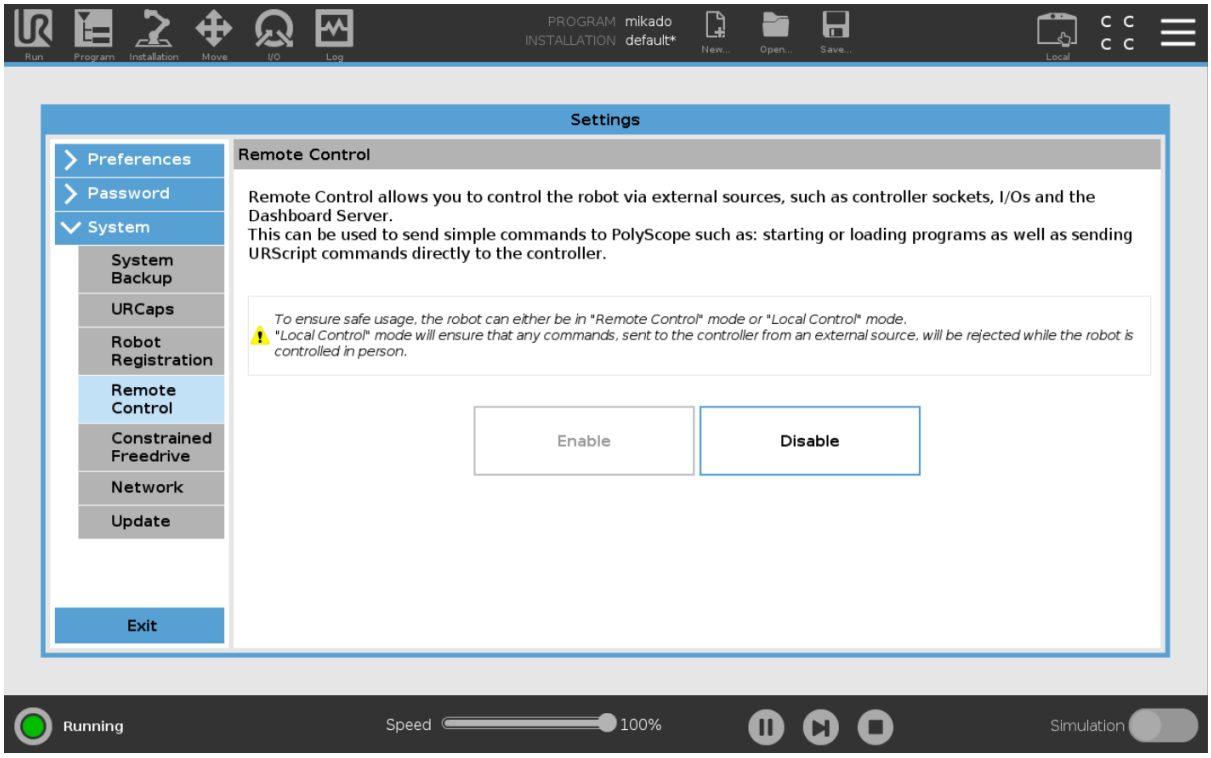


Abb. 31: Einstellungen für Remote Control Mode

– *Exit* → Oben rechts Modus von *Local* auf *Remote Control* wechseln

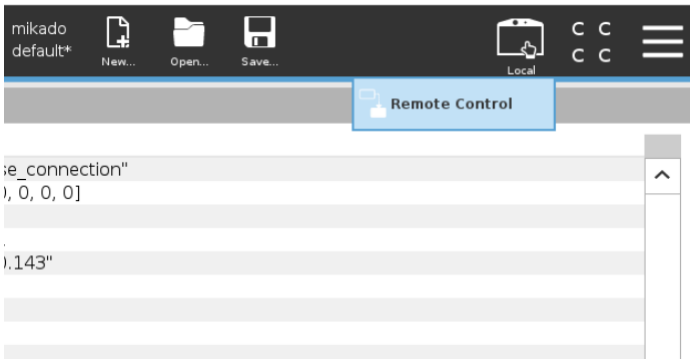


Abb. 32: Wechsel zwischen *Local* und *Remote Control* Modus

- **Ab PolyScope 5.10**: Services aktivieren
  - In neueren PolyScope Versionen sind im Auslieferungszustand für die Verbindung mit Mikado notwendige Dienste des UR Controllers deaktiviert
  - Um diese wieder zu aktivieren, damit der Roboterzustand in Mikado angezeigt werden kann:
    1. Hamburger Menü (rechts oben) → *Settings*
    2. *Security* → *Services*
    3. Menü mit dem Admin Passwort entsperren. (default: „easybot“, dieses muss vor Änderung der Einstellungen evtl. unter *Password* → *Admin* geändert werden)
    4. **Dashboard Server**, **Primary Client Interface**, **Secondary Client Interface** und **Real-Time Data Exchange (RTDE)** aktivieren
    5. Menü wieder sperren und *Exit*

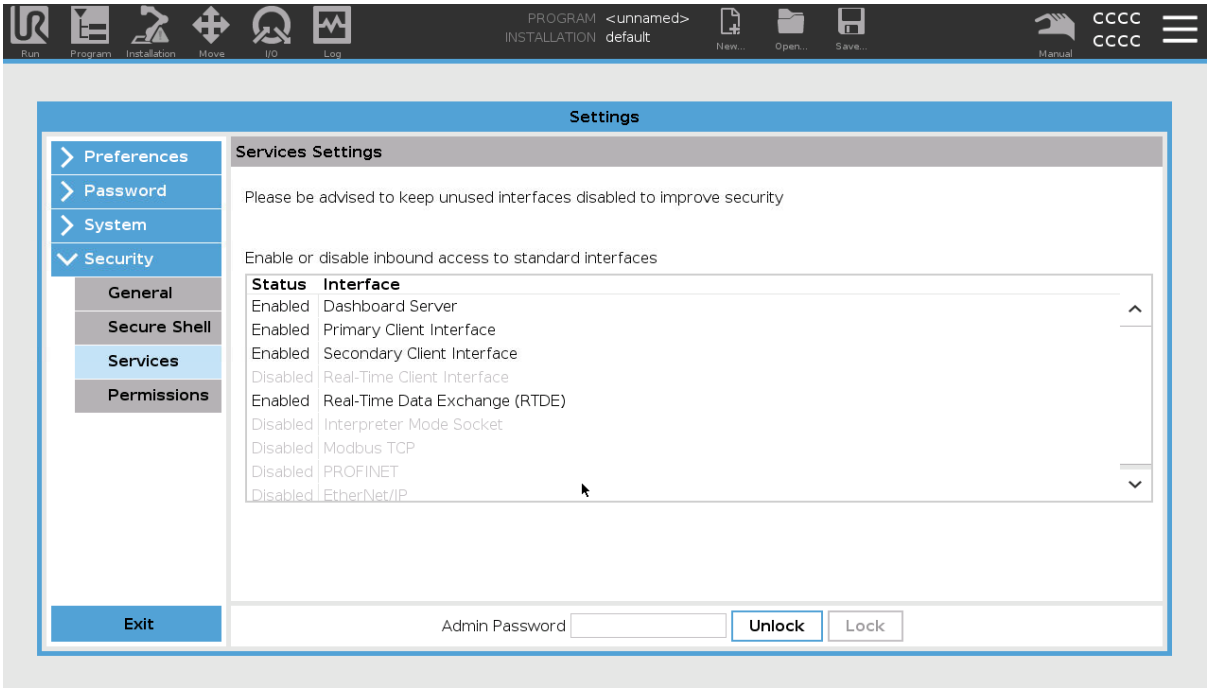


Abb. 33: PolyScope 5.23 Services

6.10.4 Autostart einrichten

Damit Mikado direkt bei Applikationsstart die Möglichkeit hat, den Roboter anzusteuern und zu bewegen, kann der Autostart aktiviert werden. Hierbei triggert Mikado den UR, sodass das Kommunikationsprogramm beim Freigeben der Roboterachsen mitgestartet wird.

Autostart einrichten und konfigurieren:

- Am UR Controller unter *Installation* → *Startup* → Haken bei *Load default program* setzen → Kommunikationsprogramm auswählen (z.B. *mikado.urp*)
- Installation Speichern
- Das Autostart-Programm öffnen. Dies wird bei zukünftigen Neustarts automatisch ausgeführt.
- Remote Modus aktivieren
- Gegebenenfalls am Mikado System in der entsprechenden Zelle in der Datei `launch/manipulator.launch.xml` folgende Parameter anpassen:

```
<param name="enable_autostart" value="true"/>
<param name="autostart_ur_program_name" value="mikado.urp"/>
```

  - *enable_autostart*: De-/Aktivieren der Autostart Funktion. Mikado wartet bei Programmstart bis ein Roboterstatus gelesen werden konnte und triggert erst dann das Kommunikationsprogramm. *(Standardwert: 10s)*
  - *autostart_ur_program_name*: Programmname des Autostart-Programms (in der Regel Name des Kommunikationsprogramms inklusive .urp-Endung) *(Standardwert: mikado.urp)*
- Nun wird beim Start von Mikado das passende UR-Programm automatisch gestartet.

Hinweis: UR bietet die Möglichkeit, über Digitale Inputs den Roboter direkt nach dem Controller-Start zu bestromen und freizuschalten. Nutzen Sie diese Funktion (zu finden unter *Installation* → *Startup*), um das System in einen betriebsbereiten Zustand hochzufahren.

6.10.5 Gelenkwinkelgrenzen

Die Mikado Bahnplanung unterstützt Gelenkwinkelbereiche bis 360°. Da ein UR durchdrehende Gelenke hat, muss der Wertebereich im Roboter-Controller (Polyscope) reduziert werden. In den Roboter Sicherheitseinstellungen können Sie Limits für die *Joints* definieren. Verwenden Sie hier dieselben Wertebereiche, wie Sie sie in der Konfiguration (*joint_limits.yaml*) von Mikado eingestellt haben. So vermeiden Sie, dass Sie versehentlich Roboterposen einteachen, die außerhalb der Grenzwerte liegen.

Beachten Sie: Geben Sie den Gelenkgrenzen im Controller etwas mehr Spielraum (+/- 4°), da ein Sicherheitsstopp schon früher ausgelöst werden kann und der Freedrive bereits vor den Grenzen gegensteuert. Beispiel: Konfiguration Mikado: von -180° bis 180° → Konfiguration Polyscope: von -184° bis 184°

Hinweis: Möchten Sie für bestimmte Anwendungen über den Gelenkwinkel-Wertebereich hinaus, können Sie den Wertebereich auch verschieben. Sie müssen lediglich darauf achten, dass die Gelenkwinkelgrenzen auf Controller und der Mikado-Konfiguration übereinstimmen.

Weiterhin besteht die Möglichkeit, ein natives Roboterprogramm zu verwenden, um Drehungen über die Grenzen hinaus zu ermöglichen. So können Sie z.B. für bestimmte Bewegungen in Ihrerer Applikation den Task „*Roboterprogramm ausführen*“ nutzen, und die Kontrolle danach an Mikado zurückgeben.

6.10.6 Geschwindigkeit und Beschleunigung

Achten Sie darauf, dass Sie in Mikado stets eine ausreichend niedrige Geschwindigkeit/Beschleunigung einstellen, sodass der Roboter nicht die Maximalgeschwindigkeit/-beschleunigung überschreiten kann, die im UR-Controller eingestellt sind. Falls der UR-Controller die Geschwindigkeit begrenzen muss, kann dies zur Fehlermeldung „Goal distance too high“ führen und der Roboter bleibt eventuell stehen, bevor er das geplante Ziel erreicht. In diesem Fall müssen Sie entweder die Begrenzungen im UR-Controller erhöhen oder die Werte in Mikado auf der Seite *Robotersteuerung* verringern.

6.10.7 Roboter-Kalibrierdaten auslesen und verwenden

Alle UR-Roboter werden vor Auslieferung im Werk kalibriert, um Ungenauigkeiten in der Roboterkinematik zu kompensieren. Diese Werkskalibrierung kann extrahiert und von Mikado zur Berechnung der inversen Kinematik verwendet werden. Auf diese Weise werden Ungenauigkeiten auch dann kompensiert, wenn der Roboter durch Mikado gesteuert wird.

Die Verwendung der Roboter-Kalibrierdaten für Mikado ist nicht zwingend erforderlich, wird aber dringend empfohlen, da die Position des Endeffektors sonst um mehrere Zentimeter abweichen kann.

Folgende Schritte müssen hierfür durchgeführt werden:

1. UR-Roboter starten und sicherstellen, dass eine Netzwerkverbindung zum Mikado Controller besteht (siehe *Netzwerkkonfiguration*)
2. Konsolen-Fenster auf dem Mikado Controller öffnen und den nachfolgenden Befehl ausführen. Hierbei müssen <ROBOTER IP> und <ZELLNAME> entsprechend durch die IP-Adresse des Roboters sowie den Namen der zu verwendenden Mikado-Zelle ersetzt werden. Mittels dieses Befehls wird die Kalibrierung des Roboters ausgelesen und in der angegebenen Datei abgespeichert:

   ```
   roslaunch ur_calibration calibration_correction.launch robot_ip:=<ROBOTER IP>␣
   ↪target_filename:="${HOME}/mikado/cells/<ZELLNAME>/config/ur_calib_data.yaml"
   ```

   Der Name der Zieldatei kann bei Bedarf frei gewählt werden. Dies ist dann jedoch in den nachfolgenden Schritten entsprechend zu berücksichtigen.
3. Die extrahierten Kalibrierdaten müssen nun mit der Roboterzelle verknüpft werden. Öffnen Sie hierfür die Zelle im Mikado-Assistant und wechseln zur Seite *Zelle → Zellmodell*. Wählen Sie dort *Urdf anzeigen* und dann die Datei `world.xacro` aus. Ersetzen Sie nun den Pfad zur Kalibrierdatei (*kinematics_parameters_file*)

wie in folgendem Beispiel dargestellt. <ZELLNAME> muss auch hier durch den Namen Ihrer Mikado-Zelle ersetzt werden:

```
<xacro:ur10e_robot kinematics_parameters_file="$(find ur_description)/config/
↪ur10e/default_kinematics.yaml" prefix="${arm_prefix}"/>
```

wird ersetzt durch:

```
<xacro:ur10e_robot kinematics_parameters_file="$(find <ZELLNAME>)/config/ur_
↪calib_data.yaml" prefix="${arm_prefix}"/>
```

4. Öffnen Sie außerdem die Datei `launch/manipulator.launch.xml` im Zellverzeichnis und ersetzen analog die Zeile:

```
<arg name="kinematics_config" default="$(find ur_description)/config/ur10e/
↪default_kinematics.yaml" ... />
```

durch:

```
<arg name="kinematics_config" default="$(arg project_dir)/config/ur_calib_data.
↪yaml" ... />
```

5. Prüfen Sie nun, ob die Änderungen erfolgreich waren, indem Sie Mikado starten. Sind die Kalibrierdaten korrekt eingebunden, so wird beim Start von Mikado folgende Nachricht in den Logs erscheinen: `Calibration checked successfully.`

   Im Fehlerfall erscheint die Nachricht `The calibration parameters of the connected robot don't match the ones from the given kinematics config file [...]`.

Hinweis: Nach Übernahme der Kalibrierdaten muss eine neue Hand-Auge-Kalibrierung (HAK) durchgeführt werden, da die Kalibrierdaten die Roboterkinematik verändern. Mit den Kalibrierdaten kann die Kameraposition genauer berechnet werden.

Warnung: Achten sie darauf, stets in beiden Dateien die neuen Kalibrierdaten einzubinden:

- `launch/manipulator.launch.xml`
- `urdf/world.xacro`

6.10.8 Optional: Natives Roboterprogramm ausführen

Mithilfe des Tasks „*Roboterprogramm ausführen*“ lassen sich native Programme auf dem Controller ausführen.

Folgende Schritte sind hierzu nötig:

- Mithilfe des Teach Pendants ein neues UR-Programm anlegen
- Möchten Sie Variablen an das Programm übertragen, so legen Sie hierfür entsprechende Installationsvariablen an:
  - Diese finden Sie unter *Program Robot* → *Installation* → *General* → *Variables*
  - Vergessen Sie nicht, die Installationskonfiguration mit den Installationsvariablen ebenfalls zu speichern

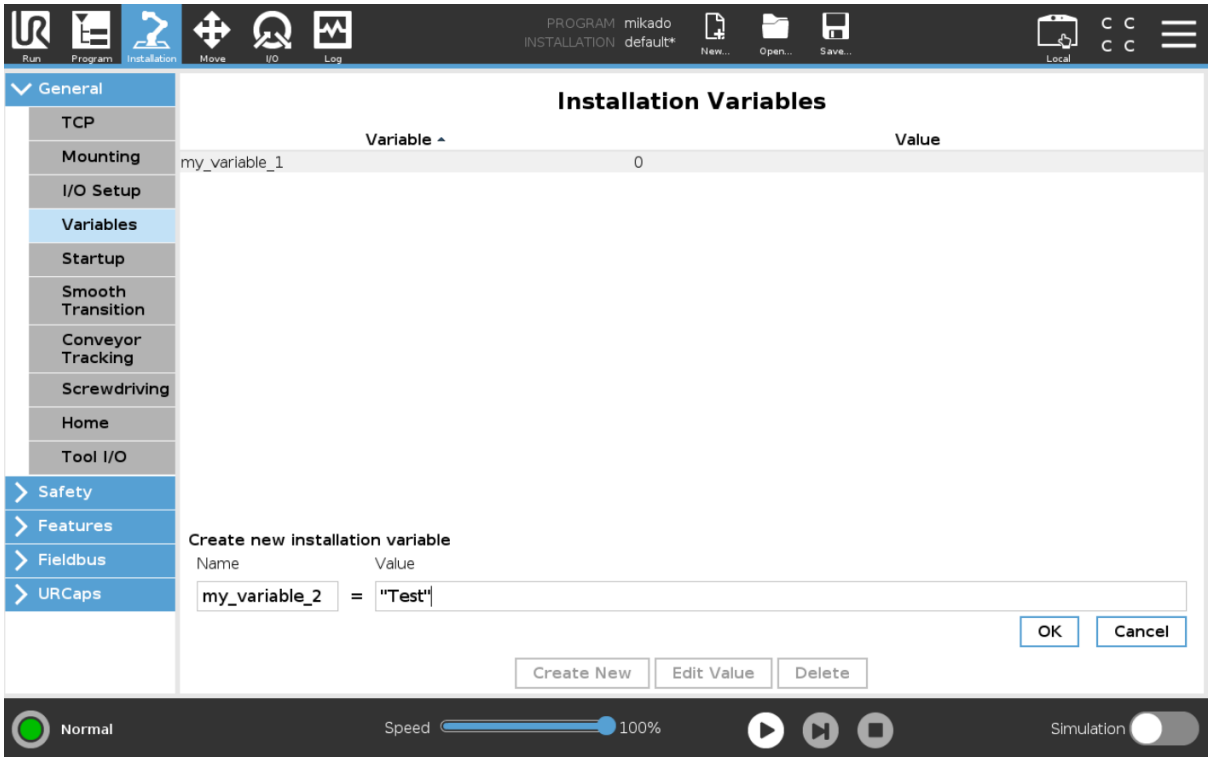


Abb. 34: UR Installationsvariablen

- Fügen Sie am Ende Ihres UR-Programms das UR-Kommando **Halt** ein. An dieser Stelle stoppt das UR-Programm und gibt die Kontrolle zurück an Mikado.

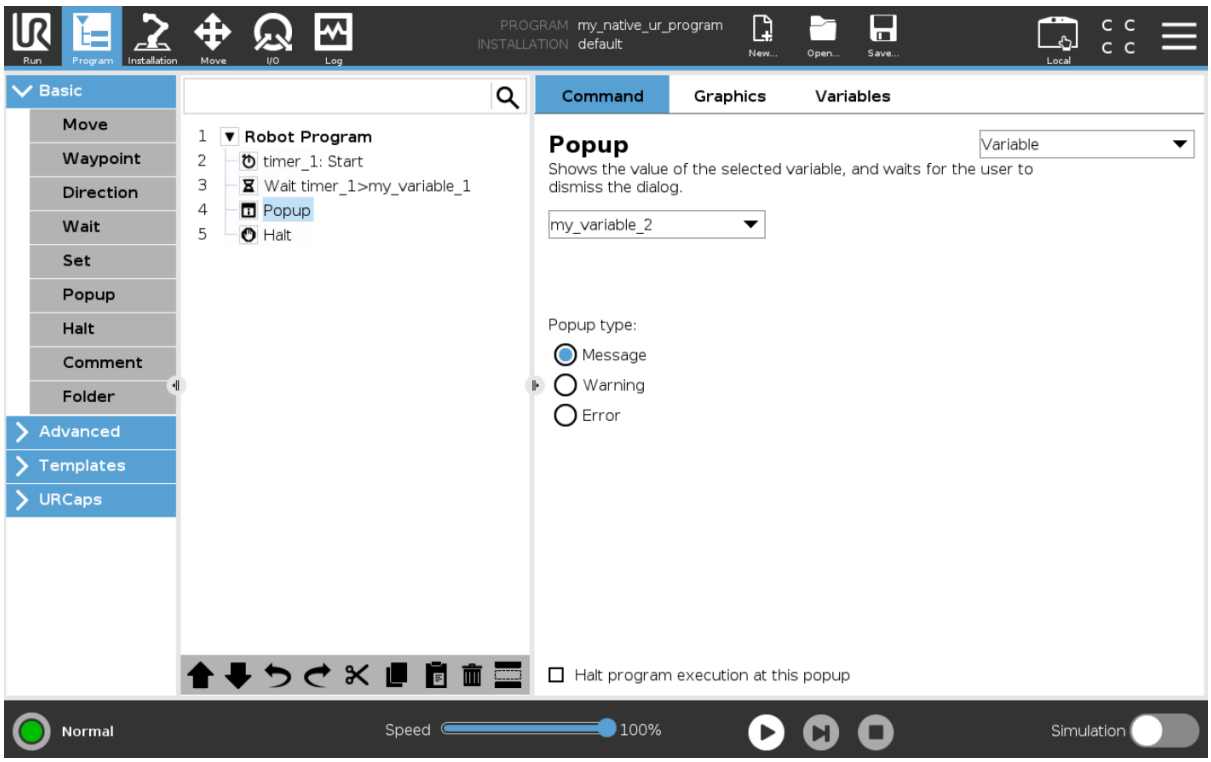


Abb. 35: Beispiel für ein natives UR-Programm

- Nachdem Sie Ihr UR-Programm fertiggestellt und getestet haben, laden und starten Sie wieder das Kontroll-

programm

- Nun können Sie den Task „*Roboterprogramm ausführen*“ in Mikado verwenden, um Ihr natives UR-Programm in den Ablauf einzubinden

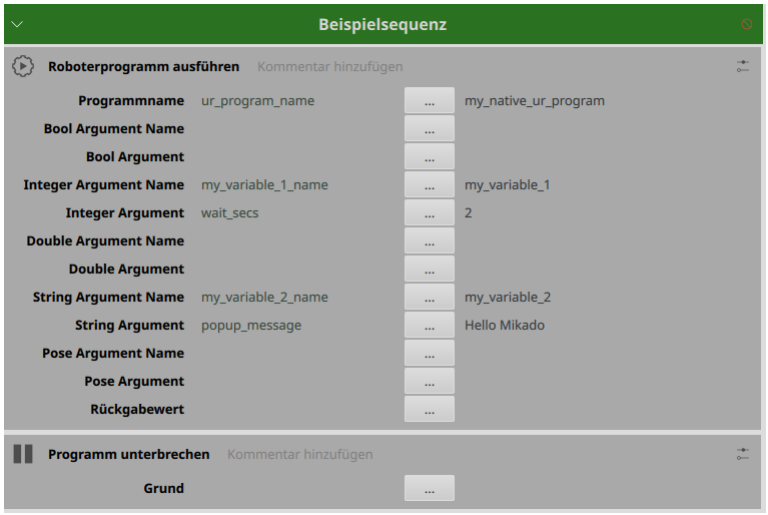


Abb. 36: Beispiel Mikado Programm mit Task „*Roboterprogramm ausführen*“

Hinweis: In der aktuell verfügbaren Mikado-Version haben Sie keine Möglichkeit einen Rückgabewert an Mikado zurückzugeben. Sie werden standardmäßig den Wert *0* als Rückgabewert des Tasks erhalten.

6.11 UR Roboter verwenden

6.11.1 Roboter einschalten

- Mikado und Roboter einschalten
- Roboter bestromen und freischalten
- Stellen Sie sicher, dass auf dem Teach Pendant **zum Startzeitpunkt von Mikado** der Modus *Remote Control* ausgewählt ist

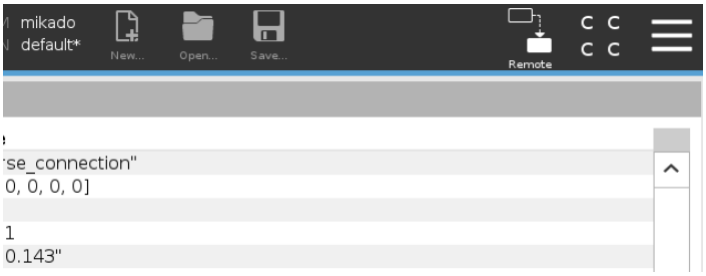


Abb. 37: Oben rechts am UR Teach Pendant lässt sich der *Remote Control Mode* auswählen

- Programm *Mikado* auf dem UR Controller laden und starten
  - Eine Verbindung zu Mikado wird aufgebaut. Ist dies nicht erfolgreich, erscheint nun eine Fehlermeldung. Prüfen Sie im Fehlerfall die Netzwerkverbindung und starten Sie die Mikado-Oberfläche.

– Die Auführung von Bewegungen kann nun auf dem Teach Pendant gestoppt oder pausiert werden. Dies kann jedoch dazu führen, dass in Mikado Zeitüberschreitungen stattfinden und somit Tasks aufgrund des Timeouts abbrechen.

Hinweis:

- Beachten Sie, dass das Mikado-Kommunikationsprogramm beendet wird, sobald Sie den Roboter manuell bewegen. Starten Sie das Programm erneut, sobald Sie die Kontrolle an Mikado zurückgeben möchten.
- Sie können wahlweise einen Autostart für das UR-Programm verwenden (s. *Autostart einrichten*).

6.11.2 URCap Gripper verwenden

Zur Nutzung weiterer URCaps z.B. zur Greifersteuerung:

- das **ExternalControl** URCap muss permanent ausgeführt werden, damit Mikado Kontrolle über den Roboter hat
- um zusätzlich z.B. über andere URCaps einen **Greifer** zu steuern, kann das Programm nebenläufig über Events oder in einem zusätzlichen Thread abhängig von Roboter-Outputs passende Funktionen ausführen.

Mit Events lässt sich das beispielsweise so umsetzen:

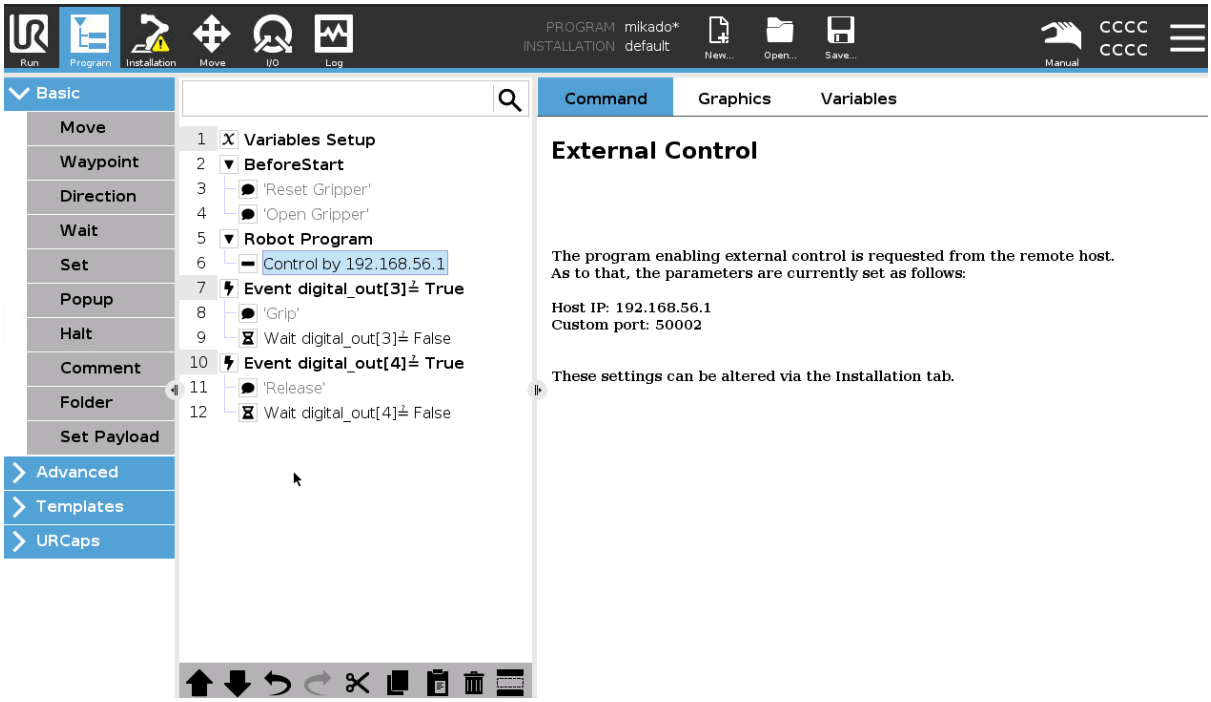


Abb. 38: URCap Gripper Events für einen Greifer

Es ist auch möglich verschiedene Greifer anzusprechen, wenn der aktuell verwendete Greifer entsprechend mit einem Digital Output angezeigt wird. Im folgenden Beispiel werden dazu exemplarisch die Digital Outputs 5 und 6 verwendet, um den jeweiligen Greifer zu aktivieren.

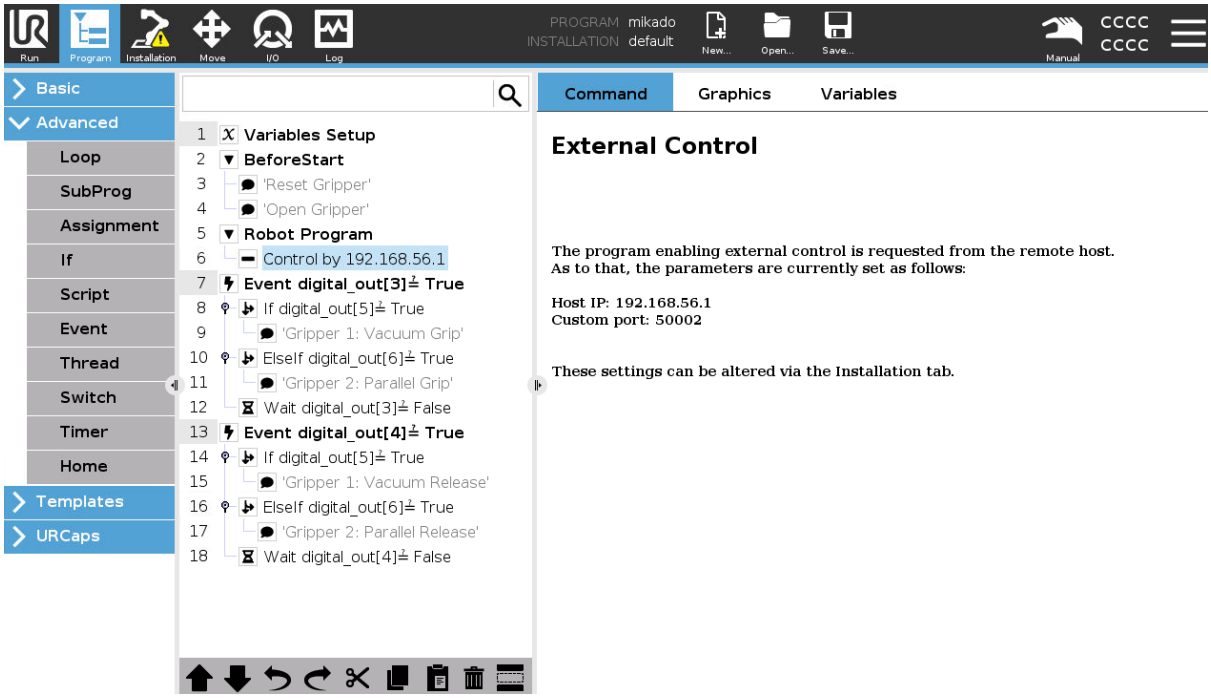


Abb. 39: URCap Gripper Events für mehrere Greifer

Bei nur zwei Greifern lässt sich das auch vereinfachen, indem nur die beiden Zustände (True/False) eines einzigen Digital Outputs verwendet werden, um zwischen den beiden Greifern zu wählen.

Mit einem Thread anstelle von Events könnte ein Programm z.B. so aussehen:

```
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 Parameter und Steuerelemente

- **Geschwindigkeit - Beschleunigung**:

  Zwei Eingabefelder bieten Ihnen die Möglichkeit, den Geschwindigkeits- und Beschleunigungsfaktor für Roboterbewegungen einzustellen. Beachten Sie, dass in der *Systemkonfiguration* Höchstwerte hinterlegt sind, sodass Sie eventuell einen Grenzwert nicht überschreiten können.

6.11.4 Sonstige Hinweise

- Position Teachen:
  - Sie können Positionen einteachen, indem Sie entweder Schaltflächen auf dem Teach Pendant (Polyscope) des Roboters oder den Freedrive Modus verwenden.
  - In Mikado „Aktuelle Roboterpose übernehmen“ anklicken.
  - Beachten Sie, dass Sie nach manuellen Roboterbewegungen das UR-Kommunikationsprogramm wieder starten müssen, damit Mikado wieder Befehle an den Roboter senden kann.

6.11.5 Troubleshooting Universal Robot

- Der Roboter lässt sich über Mikado nicht bewegen
  - Prüfen Sie, ob eine Verbindung zum Roboter besteht
  - Sind die Achsen bestromt?
  - Läuft das Kommunikationsprogramm auf dem Teach Pendant?
  - Ist der UR im Remote Control Mode?
  - Hat das Kommunikationsprogramm als IP die Adresse des Mikado-Controllers eingetragen?
- Das UR Kommunikationsprogramm lässt sich nicht starten und zeigt die Fehlermeldung `The connection to the remote PC could not be established`
  - Hat das Kommunikationsprogramm als IP die Adresse des Mikado-Controllers eingetragen?
  - Sind die IP Adressen von Roboter und Mikado-Controller korrekt konfiguriert und im selben Subnetz?
- Der Mikado-Systemstatus für den Roboter ist rot
  - Prüfen Sie, ob die Achsen bestromt sind und das Kommunikationsprogramm erfolgreich gestartet ist
- Ein natives Programm lässt sich nicht starten.
  - Prüfen Sie Programmname, Variablennamen und die Existenz sowie Datentyp der Installationsvariablen.
  - Wichtig: Während des Startvorgangs der Mikado-Oberfläche verbindet sich das System mit dem UR Controller. Zu diesem Zeitpunkt muss der Modus *Remote Control Mode* auf dem Teach Pendant ausgewählt sein.
  - Soll im nativen Programm eine Roboterbewegung ausgeführt werden, so muss der erste Wegpunkt mit der aktuellen Roboterposition übereinstimmen. Ansonsten erhalten Sie einen Fehler.
- Autostart des Mikado-Kommunikationsprogramms funktioniert nicht
  - Während des Startvorgangs der Mikado-Oberfläche verbindet sich das System mit dem UR Controller. Zu diesem Zeitpunkt muss der Modus *Remote Control Mode* auf dem Teach Pendant ausgewählt, und das Autostart-Programm geladen sein
- Keine Verbindung zum Roboter möglich und die Mikado-Konsole meldet: `Variable 'tool_digital_output_mask' is currently controlled by another RTDE client.`

  - Stellen Sie sicher, dass neben dem URCap „ExternalControl“ kein anderes UR-Programm RTDE (Real-time Data Exchange) nutzt.
  - Wenn der Konflikt durch ein benötigtes URCap ausgelöst wird (bspw. für einen OnRobot Controller), kann der Mikado UR Treiber angepasst werden, um die Kontrolle über die doppelt beanspruchten Ressourcen abzugeben:
    * die Datei rtde_input_recipe.txt muss zuerst schreibbar gemacht und dann angepasst werden:

```
roscd ur_robot_driver/resources
sudo chmod 777 ./rtde_input_recipe.txt
kate ./rtde_input_recipe.txt
```

    * in der Datei rtde_input_recipe.txt sollten die Zeilen entfernt werden, die auch von einem anderen URCap beansprucht werden:

```
tool_digital_output_mask
tool_digital_output
```

    * Siehe auch https://github.com/UniversalRobots/Universal_Robots_ROS_Driver/issues/239

- Keine Verbindung zum Roboter möglich und die Mikado-Konsole meldet: `Variable 'speed_slider_mask' is currently controlled by another RTDE client.`
  - Stellen Sie sicher, dass im Bereich *Installation* die Schnittstelle *Feldbus* → *Ethernet/IP* nicht aktiviert ist, sie stört die Netzwerkverbindung zum Mikado Controller.
    * Siehe auch https://github.com/UniversalRobots/Universal_Robots_ROS_Driver/issues/204
- Ein zuvor installiertes OnRobot URCap funktioniert nicht mehr, z.B. mit einer Meldung „Waiting for valid … timed out. Program halted“
  - Ggf. müssen unter *Installation* → *URCaps* → *OnRobot* die verwendeten RTDE Register mit einem geeigneten Offset verlegt werden, um den Resourcenkonflikt der URCaps aufzulösen.
- Die Mikado-Zelle startet nach einem **Update** des `mikado-robots-ur` Pakets nicht mehr und zeigt in der Konsole eine Fehlermeldung wie `resource not found: ur_e_description`. Diese Situation tritt auf, wenn die Mikado-Zelle ursprünglich mit einer älteren Version des UR-Treibers erstellt wurde und nun nicht mehr mit der neu installierten Treiberversion kompatibel ist. Es sind die unten aufgeführten manuellen Anpassungen notwendig, um die Mikado-Zelle mit dem aktuellen Treiber funktionsfähig zu machen.

  Achtung: die folgenden Änderungen sind alle exemplarisch anhand eines `ur10e` dargestellt. Falls ein anderer Roboter in der Zelle verwendet wird, müssen die Beispiele entsprechend des tatsächlich verwendeten Modells angepasst werden.

  - **urdf/world.xacro**

    Es müssen die folgenden Zeilen:

```
<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}"/>
```

    durch diese Zeilen ersetzt werden:

```
<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**

    Es muss die folgende Zeile:

```
<rosparam unless="$(arg simulated_robot)" file="$(find ur10_e_moveit_config)/
↪config/controllers.yaml"/>
```

durch diese Zeile ersetzt werden:

```
<rosparam unless="$(arg simulated_robot)" file="$(find ur10e_moveit_config)/
↪config/ros_controllers.yaml"/>
```

– **launch/manipulator.launch.xml**

Es muss der folgende Abschnitt:

```
<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>
```

durch diesen Abschnitt ersetzt werden:

```
<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"/>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
  <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)
↪"/>
      <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**

  Es muss die folgende Zeile:

```
<rosparam command="load" file="$(find ur10_e_moveit_config)/config/
↪kinematics.yaml"/>
```

  durch diese Zeile ersetzt werden:

```
<rosparam command="load" file="$(find ur10e_moveit_config)/config/kinematics.
↪yaml"/>
```

6.12 Yaskawa Motoman Controller einrichten

6.12.1 Vorbereitung

Diese Beschreibung wurde für die Yaskawa Motoman Controller Modelle DX200, FS100, YRC1000 und YRC1000micro (auch YRC1000u) erstellt. Bei Problemen wenden Sie sich bitte an den Mikado Support. Die Einrichtungsprozedur unterscheidet sich abhängig davon ob ein **Standard Pendant** (kleiner Bildschirm oben, Tastenfeld unten) oder ein **Smart Pendant** (großer Touchscreen, Tasten rechs im Rahmen) verwendet wird.

Hinweis zum Smart Pendant: Seit Softwarestand 2.1 verfügt das Smart Pendant über eine `Classic Interface` Oberfläche welche das Standard Pendant emuliert. Darüber lassen sich Anleitungen für den Standard Pendant auch auf den Smart Pendant anwenden. Das `Classic Interface` ist unter `Menu` → `System Settings` → `Classic Interface` → `Start Classic Interface` zu finden.

Software Pendant: Alle Anleitungen für das **Standard Pendant** können alternativ auch mit der Windows-Software **Software Pendant** umgesetzt werden.

6.12.2 Netzwerkkonfiguration

Für die Kommunikation zwischen Roboter Controller und Mikado Controller wird eine Netzwerkverbindung zwischen den beiden Systemen benötigt. Verbinden Sie hierfür zunächst einen Ethernet Port des Yaskawa Controllers mit einem Ethernet Port des Mikado Controllers (**YRC1000**: LAN2 oder LAN3). Anschließend muss sowohl für den Mikado Controller als auch für den Roboter Controller eine feste IP-Adresse und eine zugehörige Subnetzmaske eingestellt werden. Die Netzwerkkonfiguration des Mikado Controllers wird in Sektion *Netzwerkadapter* näher erläutert. Netzwerkeinstellungen für den Yaskawa Controller können wie folgt vorgenommen werden:

Für diesen Schritt muss der Controller im `Maintenance Mode` gestartet werden.

Standard Pendant: Um den Controller im `Maintenance Mode` zu starten muss die `Main Menu` Taste durchgehend gehalten werden während der Controller frisch eingeschaltet wird und hochfährt. Die Taste kann losgelassen werden wenn das Logo auf dem Pendant Bildschirm erscheint. Alternativ kann über `System Info` → `CPU Reset` ein Neustart direkt in den Maintenance Mode ausgelöst werden.

Smart Pendant: Ab Version 2.1 kann das Smart Pendant über `Menu` → `System Settings` → `Classic Interface` die Oberfläche das Standard Pendant emulieren. Indem statt `Connect` über `Special Mode` → `Maintenance Mode` ausgewählt wird, kann diese emulierte Oberfläche im **Maintenance Mode** gestartet werden.

Im Maintenance Mode:

Standard Pendant: Security Level auf `Management` (bzw. `Erweitert`) setzen (`System` → `Security`, Standardpasswort: 16 mal die Ziffer 9). Anschließend kann die IP-Adresse des Controllers unter `System` → `Setup` → `Option Function` → `Lan Interface Setting` eingerichtet werden.

Smart Pendant: Unter `Menu` → `System Settings` → `Controller` → `IP Address` kann die aktuelle IP-Adresse des Motoman Controllers ausgelesen und ggf. geändert werden.

Hinweis: Beachten Sie, dass die IP-Adressen von Mikado Controller und Yaskawa Controller im selben Adressraum liegen und die Subnetzmasken übereinstimmen.

Um eine Verbindungsaufnahme durch Mikado zu ermöglichen muss die IP-Adresse des Yaskawa Motoman Controllers bekannt sein. Tragen Sie hierfür die IP-Adresse des Roboter Controllers im Mikado Assistant während des Erstellungsvorgangs der Zelle oder in der Sektion Roboter einer existierenden Zelle ein (`Roboter` → `Verbindung`).

6.12.3 Einrichtung des Roboter Controllers

Übersicht

Die Controller-Seite des Yaskawa Motoman ROS-Treibers heißt **MotoRos** und ist eine MotoPlus Anwendung. Sie verwendet ein Socket Interface und mehrere parallele Anwendungen. Das Standardsetup für Mikado verwendet nur einen 6-Gelenk-Manipulator ohne weitere Achsen. Die MotoRos-Anwendung fungiert auf dem Motoman-Controller als Server für Clients auf dem Mikado-PC.

Kopieren der Dateien

Die für die Einrichtung benötigten Dateien befinden sich im Ordner: `/opt/isys-vision/mikado/robot_programs/motoman/motoman_files`.

Das Verzeichnis lässt sich auch über den Mikado Assistant mit Hilfe des entsprechenden Buttons öffnen (`Roboter` → `Verbindung` → `Verzeichnis öffnen`).

- Das MotoRos-Programm liegt im MotoRos Unterordner, für den YRC1000 Controller bspw. in `MotoRos/YRC1000/MotoRosYRC1_/MotoRosYRC1_19122_200hz.out`.
- Die Job-Datei `INIT_ROS.JBI` und `IONAME.DAT` für gewöhnliche Ein-Arm-Setups liegen im Unterordner `Inform/DX200, FS100, YRC1000/single_arm/`.

Diese Dateien sollten auf einen USB-Datenträger kopiert werden, der dann in den Pendant eingesteckt wird. Die nächste Abschnitte beschreiben, wie diese Dateien auf den Roboter-Controller kopiert werden können.

MotoRos einspielen

FS100, DX200, YRC:

Für diesen Schritt muss der Controller im `Maintenance Mode` gestartet werden.

Standard Pendant: Um den Controller im `Maintenance Mode` zu starten muss die `Main Menu` Taste durchgehend gehalten werden während der Controller frisch eingeschaltet wird und hochfährt. Die Taste kann losgelassen werden wenn das Logo auf dem Pendant Bildschirm erscheint.

Smart Pendant: Ab Version 2.1 kann das Smart Pendant über `Menu` → `System Settings` → `Classic Interface` die Oberfläche das Standard Pendant emulieren. Indem statt `Connect` über `Special Mode` → `Maintenance Mode` ausgewählt wird, kann diese emulierte Oberfläche im **Maintenance Mode** gestartet werden.

Im Maintenance Mode:

1. Sicherheitslevel auf `Management` (bzw. `Erweitert`) erhöhen (`System` → `Security`, Standardpasswort: 16 mal die Ziffer `9`)
2. go to `System` → `Setup`, select `Option Function`
3. `MotoPlus Func` auf `USED` setzen
4. `MOTOMAN DRIVER` auf `USED` setzen
5. unter `Motoplus APL` → `Device` den USB-Speicher auswählen (Der Port `USB1:Controller` befindet sich am Controller, `USB:Pendant` bzw. `USB:PHG` befinden sich auf der Rückseite des Teach Pendants)
6. unter `Motoplus APL` → `Load (User App)` die Datei `MotoRosXXXXX.out` mit `Select` auswählen und mit `Enter` bestätigen
7. unter `Motoplus APL` → `File List` verifizieren dass MotoROS installiert wurde und keine weiteren MotoPlus Anwendungen auf den Controller geladen wurden

Nur bei DX200 und YRC:

1. Schlüsselschalter auf `TEACH`-Modus stellen
2. Maintenance Mode beenden
    1. **Standard Pendant**: Controller neu starten um wieder ohne Maintenance Mode zu starten
    2. **Classic Interface**: Maintenance Mode beenden über den Power-Button rechts oben um das Classic Interface zu verlassen

Innerhalb von 30 Sekunden nach dem Neustart sollte der Alarm `8001 [10] Speed FB enabled, reboot now` auftauchen. Nach einem erneuten Neustart sollten keine Alarme mehr erscheinen. Falls weiter Alarme auftauchen kann die MotoRos Troubleshooting Seite weiterhelfen.

INFORM Code einrichten

Die MotoROS Anwendung verwendet den `INIT_ROS INFORM`-Job. Die im Abschnitt *Kopieren der Dateien* erwähnte Datei `INIT_ROS.JBI` muss nun zusammen mit der `IONAME.DAT` Datei auf den Controller kopiert werden. Die `IONAME.DAT` Datei enthält die Namen von MotoRos-Ausgangssignalen.

Für die nächsten Schritte sollten sie einen USB-Speicher mit den beiden Dateien in das Teachpendant einstecken und den Controller normal (nicht im `Mainenance Mode`) starten. Beide Dateien sollten im gleichen Ordner liegen.

Standard Pendant oder **Software Pendant** oder **Classic Interface**:

1. Sicherheitslevel auf `Management` (bzw. `Erweitert`) erhöhen (`System Info` → `Security`, Standardpasswort: 16 mal die Ziffer `9`)
2. unter `Ext Memory` → `Device` den USB-Speicher auswählen

3. unter `Ext Memory` → `Load` → `Job` den `INIT_ROS.JBI` Job auswählen und laden
4. unter `Job` → `Select Job` verifizieren dass `INIT_ROS` korrekt installiert wurde

Smart Pendant:

1. zu `Menu` → `Utility` → `File Transfer` navigieren
2. `To Controller` auswählen
3. falls die Datei nicht im Hauptverzeichnis des USB-Speichers liegt mittels `Change Folder` in den entsprechenden Ordner navigieren
4. In der Job-Liste das Häkchen vor dem `INIT_ROS.JBI` Job setzen und `Copy Files to Controller` auswählen
5. Den Dialog zur Dateiübertragung mit `Yes` beantworten

Controller Konfiguration

Jetzt kann verifiziert werden ob der Roboter-Controller im **Remote**-Modus Befehle empfangen kann:

Standard Pendant oder **Classic Interface**:

1. am Pendant über den Schlüsselschalter den `Teach`-Modus auswählen
2. Sicherheitslevel auf `Management` (bzw. `Erweitert`) erhöhen (`System Info` → `Security`, Standardpasswort: 16 mal die Ziffer `9`)
3. am Pendant `In/Out` → `Pseudo Input Sig` auswählen
4. Den Pseudo Input `CMD REMOTE SEL` aktivieren. Die Signalnummer ist `#87015` bei **YRC** und `#82015` bei **FS100** und **DX200** Controllern
    1. dazu den Cursor zum Kreis neben dem passenen Input bewegen
    2. `Interlock + Select` zusammen drücken um den Input zu aktivieren
5. `Setup` → `Operate Cond` auswählen. Cursor zu den `Cycle Switch in xxx Mode` Einträgen bewegen. Alle diese Einträge auf `Cycle` stellen.
6. am Pendant über den Schlüsselschalter den `Remote`-Modus auswählen

Smart Pendant:

1. über `Menu` → `System Settings` → `Classic Interface` das **Classic Interface** starten und `Connect` auswählen
2. am Pendant über den Schlüsselschalter den `Teach`-Modus auswählen
3. `Setup` → `Operate Cond` auswählen. Cursor zu den `Cycle Switch in xxx Mode` Einträgen bewegen. Alle diese Einträge auf `Cycle` stellen.
4. den Power-Button rechts oben auswählen und mit `Disconnect` zur Smart Pendant Oberfläche zurück kehren
5. am Pendant über den Schlüsselschalter den `Remote`-Modus auswählen

`CMD REMOTE SEL` ist beim Smart Pendant immer aktiv und muss nicht explizit aktiviert werden.

Verbindungstest mit Mikado

Der Robotercontroller sollte vollständig hochgefahren sein bevor Mikado gestartet wird. Nach dem Start von Mikado sollte der Roboter hörbar die Servomotoren aktivieren und links unten in der Statusanzeige einen grünen Status anzeigen, wenn am Pendant über den Schlüsselschalter der `Remote`-Modus ausgewählt wurde.

Falls Mikado nach Abschluss der Konfiguration keine Verbindung aufbauen kann muss der Robotercontroller noch einmal neu gestartet werden.

6.12.4 Weiteres & Troubleshooting

Fehlercode-Beschreibung: http://wiki.ros.org/motoman_driver/Troubleshooting

Eine Mikado-unabhängige Anleitung zur Einrichtung von MotoRos auf dem Controller: ROS Wiki Tutorial

Yaskawa Motoman Dokumentation: https://www.motoman.com/en-us/service-training/product-documentation

Yaskawa Motoman Downloadbereich für das Software Pendant: https://software.yaskawa.eu.com/programs

Yaskawa Motoman Smart Pendant Updates um Version 2.1 oder neuer zu installieren: https://yaskawapartners.com/download/smart-pendant-software/

Servomotoren

In der Standardkonfigurationd es Controllers werden die Servomotoren nach einer gewissen Zeit ohne Aktivität in einen Stromsparmodus versetzt. Dieser wird automatisch beendet sobald der Roboter eine neue Bewegungsaufgabe erhält.

I/O Adressbereiche

Die Mikado-Anwendung verwendet für E/A-Zugriffe ausschließlich `General Input`- und `General Output`-Adressen.

Die erste gültige Mikado E/A-Adressen für Motoman Roboter hat den Index 1. Die Motoman Teach Pendants verwenden allerdings die 1-basierte Notation `Out#001` und `In#001` für General Purpose E/A-Signale. Diese Notation verwendet kontinuierliche Indizes, nicht zu verwechseln mit den globalen Motoman-Adressen vom Format `#87015`.

Beispiel: Der Mikado Output mit Index 1 wird auf die Motoman-Adresse `#10010` abgebildet welche auf dem Pendant auch als `Out#001` bezeichnet wird.

Yaskawa Motoman Adressen werden dezimal notiert, die letzte Ziffer beschreibt allerdings ein 8-Bit Byte weshalb hier nur die Ziffern 0-7 gültig sind. Der Adressbereich 10010 - 15127 enthält daher beispielsweise 512 Byte und 4096 gültige Bitadressen.

Mikado E/A-Adressen werden kontinuierlich auf gültige Motoman-Adressen abgebildet, daher entspricht Mikado Eingang 9 den Motoman Adressen `In#009` und `` `#00020` ``.

Tab. 5: **Adressbereiche verschiedener Controller**

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 verwendet einige fixe Outputs, diese werden in der `IONAME.DAT` benannt:

```
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 Beschleunigungslimits in joint_limits.yaml eintragen

Die Beschleunigungslimits der Motoman Manipulatoren sind applikationsabhängig und nicht frei öffentlich zugänglich, daher verwendet Mikado initial ein Beschleunigungslimit von $1\ rad/s^2$. Gerade für kleine und schnelle Manipulatoren empfiehlt es sich in der `joint_limits.yaml` explizit höhere Beschleunigungslimits für die Gelenke anzugeben, indem `has_acceleration_limits` auf `true` und `max_acceleration` auf das gewünschte Limit in rad/s^2 gesetzt wird.

```
joint_1_s:
  min_position: -2.967
  max_position: 2.967
  has_acceleration_limits: true
  max_acceleration: 8.11
```

Vorsicht: Zu hoch gewählte Beschleunigungslimits können zu Problemen wie falsch erkannten Kollisionen und Bewegungsabbrüchen sowie Alarmmeldungen auf dem Robotercontroller führen.

In der Datei `joint_limits.yaml` kann zudem der Bewegungsspielraum des Robots durch Verringerung der `min`- und `max`-Stellung einzelner Gelenke eingeschränkt werden, um unerwünschte Roboterkonfigurationen bei Bewegungen zu kartesischen Posen zu verhindern.

6.12.6 Fehlercodes / Alarme

Nachdem ein Alarm aufgetreten ist können auf dem Bedienpanel Details zum Fehler angezeigt werden. Dort findet sich auch ein Fehler-Subcode, Informationen zum betroffenen Gelenk und mögliche Fehlerursachen. Informationen zu früheren Alarmen finden sich im Standard Pendant unter `System Info` → `Alarm History`. Dort kann zwischen der Anzeige von Hauptalarmen und untergeordneten Alarmen umgeschaltet werden.

Wenn ein untergeordneter Alarm aufgetreten ist (**Fehlercode 4000 oder größer**) muss dieser vom Anwender am Teach Pendant quittiert werden bevor Mikado den Roboter wieder bewegen kann.

Hinweis: Nachdem ein Haupt-Alarm (**Fehlercode 0000-3999**) aufgetreten ist, muss der Yaskawa Controller neu gestartet werden (CPU Reset oder Hauptschalter Off&On).

`Fehler 4315: Kollision erkannt` und `Fehler 1303: Arithmetikfehler` weisen auf unerwartete Kollisionen bzw. unerwartet hohe Widerstände hin. Wenn diese Meldungen auftreten ohne dass der Roboter seine Umgebung berührt, sollten die Schwellwerte der Kollisionserkennung (Smart Pendant: `Roboter` → `Kollisionserkennung` → `Modus 8` (Standard)) überprüft und eventuell angepasst werden. Durch eine Anpassung der Tool-Eigenschaften Masse und Schwerpunkt an die Realität kann die Zuverlässigkeit der Kollisionserkennung erhöht und können die Schwellwerte gesenkt werden.

Bei zu hoch gewählten *Beschleunigungslimits* in der Datei `joint_limits.yaml` kann neben diesen beiden Fehlern auch der Fehler `4414: Excessive Segment` auftreten. Der Fehler-Subcode weist auf das betroffene Robotergelenk mit zu hoch gesetztem Beschleunigungslimit hin.

Bei Abweichungen der an den Roboter gesendeten Startposition von der aktuellen Position des Roboters können verschiedene Fehlermeldungen auftreten:

- `Fehler 3011: Trajectory start position doesn't match current robot position`.
  - Falls dieser Fehler ohne erkennbaren externen Einfluss entsteht (z. B. aufgrund minimaler Positionsänderungen am Ende einer schnellen Bewegung), ist es möglich, dem Robotertreiber bei Bedarf zusätzliche Wiederholungsversuche für die Ausführung der Bewegung zu erlauben. Falls sich der Roboterarm noch „einschwingt", besteht dadurch die Möglichkeit, dass er bei der Wiederholung nahe genug an der erwarteten Startposition zur Ruhe gekommen ist und die Bewegung dann erfolgreich ausführen kann.

    Dafür kann in der Datei `launch/manipulator.launch.xml` eine Einstellung vorgenommen werden:

    * Der Wert für `retry_traj_on_invalid_start_pos` muss auf einen positiven Wert gesetzt werden, z.B. `"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>
```

    * Bei älteren Zellen, die mit `mikado-robots-motoman` Paket-Versionen vor *3.2.0* erstellt wurden, existiert die Zeile `<arg name="retry_traj_on_invalid_start_pos" value="0" />` noch nicht.

      Sie muss direkt nach der Zeile `<include...$(motoman_driver)/launch/robot_interface_streaming...` eingefügt werden.

> **Warnung:** Der Wert sollte nicht höher als 10 gesetzt werden, da der Roboter sonst unter gewissen Umständen unerwartete Positionsabweichungen ignoriert und aufgrund dieser Einstellung dennoch ein Fahrbefehl an den Roboter gesendet wird, obwohl sich dieser an einer unerwarteten Startposition befindet.

- Bei `Fehler 5012` findet sich im Log die Beschreibung `Incremental move rejected on MotoROS side (5012)`. In einer verbundenen Telnet-Konsole wären zudem die Meldungen `mpExRcsIncrementMove returned: -1` und `ERROR: Controller is not ready (code: 5012). Can't process ROS_MSG_JOINT_TRAJ_PT_FULL` sichtbar.
- Diese Fehler können insbesondere bei einer Doppelpassung des vom Greifer gehaltenen Werkstücks mit einem anderen Teil der Roboterzelle aufteten: Die Doppelpassung kann zu einer passiven Bewegung des Roboterarms führen, sowie zu einer entgegengesetzten Bewegung sobald sich der Greifer wieder öffnet. Da auf die Greiferöffnung oft direkt ein Roboterbewegungsbefehl folgt kann es zu einer Abweichung von der zuletzt empfangenen Roboterposition und den erwähnten Fehlerbildern kommen.

6.12.7 Kontrolle über den Roboter vorübergehend abgeben

Nach Installation des MotoROS Programms und des INIT_ROS Jobs auf dem Roboter wird automatisch INIT_ROS ausgeführt. Normalerweise übernimmt Mikado dann die Kontrolle über die Bewegung des Roboters.

Um diese Kontrolle während der Laufzeit von Mikado abzugeben und wieder zu erlangen, kann der Task *Roboterprogramm ausführen* verwendet werden.

- Durch Angabe des Programmnamens `"robot_disable"` kann die Kontrolle über den Roboter temporär abgegeben werden, um bspw. sicherzustellen, dass nur bspw. eine SPS oder ein Job den Roboter bewegen kann.
- Dies kann durch Ausführung des Tasks mit dem Programmnamen `"robot_enable"` rückgängig gemacht werden. Falls dieser Vorgang scheitert, wird ein Fehler im Roboterstatus angezeigt.

Um zu verhindern, dass Mikado beim Start die Kontrolle über den Roboter übernimmt, kann außerdem in `config/manipulator.yaml` der Wert **control_on_startup** auf **false** gesetzt werden.

6.12.8 Andere Jobs auf dem Roboter ausführen

Es ist möglich die Ausführung des Mikado-INFORM-Jobs auf dem Yaskawa Robotercontroller vorübergehend zu unterbrechen und andere INFORM Jobs auszuführen.

Dafür kann beispielsweise die alternative Version der Datei `INIT_ROS.JBI` aus dem Ordner **Inform/DX200, FS100, YRC1000/single_arm_call_job_main** verwendet werden, statt der Datei aus dem **single_arm** Ordner. Mehr Informationen zur Installation der Job-Datei ist unter *Kopieren der Dateien* zu finden.

Dieser beispielhafte **INIT_ROS.JBI** Job prüft beim Start zwei Signale um daraufhin entweder Mikado die Kontrolle über den Roboter zu geben, oder um stattdessen den Job **MAIN** aufzurufen:

```
IF ( IN#(1871)=ON AND IN#(1872)=OFF ) THEN
  CALL JOB:MAIN
ENDIF
```

Der Wechsel in einen anderen Job ist nur möglich, nachdem zuvor wie unter *Kontrolle über den Roboter vorübergehend abgeben* die Funktion `robot_disable` ausgeführt wurde.

6.12.9 Yaskawa 16-Bit Register lesen und schreiben

Mit dem Task *Roboterprogramm ausführen* können weitere Sonderfunktionen des Yaskawa Roboters verwendet werden:

- Durch Angabe des Programmnamens `"read_mreg"` kann ein Register-Wert vom Robotercontroller ausgelesen werden
  - Der Index des Registers kann als Integer Argument angegeben werden
  - Der gelesene Register-Inhalt wird in den Output-Parameter **Rückgabewert** geschrieben
- Durch Angabe des Programmnamens `"write_mreg"` kann ein Register-Wert auf den Robotercontroller geschrieben werden
  - Der Index des Registers kann als Integer Argument angegeben werden
  - Der zu schreibende Wert wird als Double Argument übergeben

KAPITEL 7

Installation

In diesem Kapitel wird erläutert wie sie Ihre Roboterzelle in Mikado Control einrichten und ein lauffähiges System erhalten. Sie finden hier auch Hinweise wie die Eigenschaften des Systems später an ihre Bedürfnisse angepasst werden können.

7.1 Übersicht

In dieser Übersicht werden die unterschiedlichen Komponenten aufgeführt, die die Mikado Umgebung aus Hard- und aus Softwaresicht darstellen. Mithilfe eines Diagramms sind zudem die Verknüpfungen und Schnittstellen zwischen den Komponenten schematisch aufgeführt.

7.1.1 Hardware-Übersicht

Das Hardware-Setup eines typischen Mikado-Systems besteht aus den folgenden Hardware-Bausteinen. Damit alle Systeme mit dem Mikado-Controller erfolgreich kommunizieren können, müssen diese entsprechend auf dem Controller eingerichtet sein. Zur Einrichtung der einzelnen Komponenten stehen der *Assistant* und die *Konfigurationsdateien* zur Verfügung.

- Mikado-Controller
  - Zentrale Komponente: Übernimmt Roboter-, Greifer- und Kamerasteuerung
  - Objekterkennung, Bahnplanung, Inverse Kinematik, Ablaufsteuerung, Feldbus-Anbindung
- Roboter
  - TCP/IP Verbindung zu Mikado-Controller
  - Mit Zelle wird herstellerabhängig das entsprechende Kommunikationsmodul geladen
  - Robotercontroller muss herstellerabhängig konfiguriert werden (z.B. wie in Sektion *Mitsubishi Controller einrichten* beschrieben)
- Greifer
  - CAD-Modell der Form (für Kollisionsprüfungen)
  - Parametrierung der Öffnung, IO-Ansteuerung oder separater Treiber
  - Entweder über Roboter-IOs, TCP/IP oder externe IOs (bzw. SPS) angebunden

- Kamera
  - CAD-Modell (für Kollisionsprüfungen)
  - Hand-Auge-Kalibrierung über Mikado-Oberfläche
  - Auswahl: Real, File-Kamera oder simuliert
  - TCP/IP Verbindung zu Mikado-Controller
- SPS / Leitsystem
  - Schnittstelle zu SPS oder externe Systeme
  - Feldbus-Anbindung zu Mikado-Controller wie z.B. via Profinet (benötigt ggf. optionales Schnittstellen-Modul im Mikado-Controller)
  - Steuerung von außen (Sequenzen starten)
  - Datenaustausch / Ergebnisse

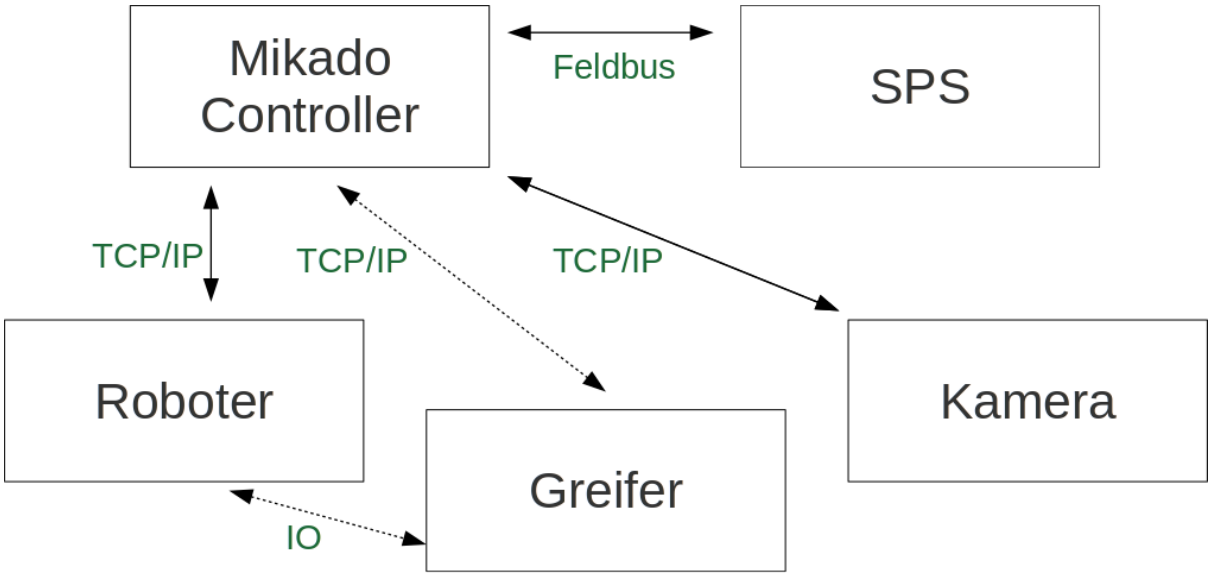


Abb. 1: Übersicht der Hardware-Komponenten mit Schnittstellen

7.1.2 Software-Übersicht

Das Mikado-System lässt sich in fünf verschiedene Softwaremodule aufteilen. Auch diese Komponenten wirken im Zusammenspiel, was in der folgenden Grafik dargestellt wird.

- Mikado (*Verwendung*)
  - Bediener- und Konfigurationsoberfläche
- Mikado Simulator (*Simulation*)
  - Reale Kamera durch Filekamera / Virtuelle Kamera ersetzen
  - Kisten füllen mit Physik-Simulation
  - Greiferkonzept testen
  - Simuliert Stereo-Bilder gefüllter Kisten
- Mikado Assistant (*Erstellen einer neuen Zelle*)
  - Roboterzelle erstellen
  - Greifer modellieren

- Scripts und eigene Erweiterungen (*Mikado Control erweitern*)
  - Eigene Halcon Scripts
  - Adapter für Feldbus-Anbindung
  - Weitere Möglichkeiten wie JavaScript und Python
- Ensenso NxView (Ensenso Manual)
  - 3D Kamera konfigurieren und testen
  - Filekameras erstellen
  - Konfiguration für Mikado exportieren

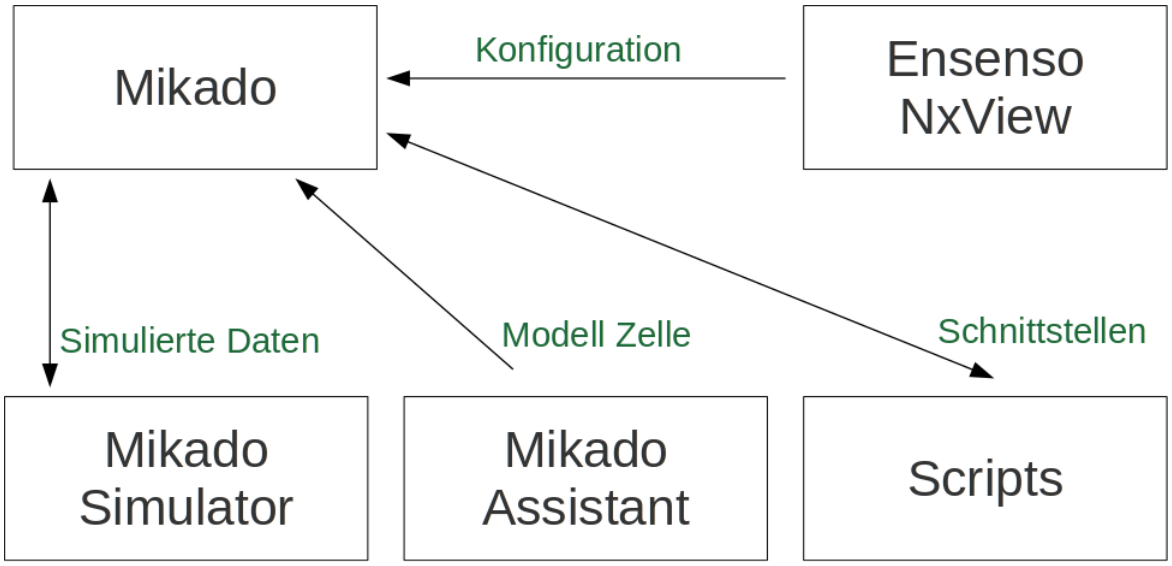


Abb. 2: Übersicht der Software-Komponenten mit Schnittstellen

7.1.3 Integrationsablauf (Schritt-für-Schritt Anleitung)

Eine kompakte Schritt-für-Schritt Anleitung, die als Leitfaden bei Neu-Integrationen dienen kann, ist im Folgenden aufgeführt. In der Spalte ‚Werkzeuge' sind Hinweise und Verlinkungen zu den Sektionen des Handbuchs, die für diesen Schritt am wichtigsten sind.

Schritt	Bezeichnung	Werkzeuge
1	Netzwerkadapter einrichten	*Netzwerkmanager*
2	Zelle erstellen und anpassen	*Mikado Assistant Zellerstellung* und *URDF Editier-Hilfe*
3	Greifer einbinden	*Mikado Assistant Zellerstellung* und *CAD Greifer*
4	Roboter und Kamera einbinden	*Roboter-Controller*
5	Kamerakonfiguration(en) erstellen	NxView Manual
6	Hand-Auge-Kalibrierung	*Mikado Kalibrierung* und Kalibrierplatte
7	TCP-Offset des Greifers ermitteln	*TCP kalibrieren* und Kalibrierplatte
8	Szene, Produkt, Griffe und Programmablauf definieren	*Mikado GUI*
9	Optimierung und Test	*Mikado GUI* und ggf. *Mikado Simulator*
10	Optional: Schnittstelle einrichten	*Mikado Control erweitern*

7.2 Konfigurations-Backup

Ein Backup auf Dateisystemebene umfasst den Ordner */home/mikado/mikado/cells* und enthält alle Anlagen-spezifischen Einstellungen.

Warnung: Das Passwort für die Zugangskontrolle (*Passwortschutz* bzw. *access_control.yaml*) ist ebenfalls Teil dieses Backups und dort lesbar.

7.3 Erstellen einer neuen Zelle

Zum Erstellen einer neuen Zelle gibt es den *Mikado Assistant*. Um ihn zu starten öffnen Sie das Programmmenü (K unten links), bzw. betätigen die Super/Windows Taste, tippen Sie *Assist...* und klicken auf den gefundenen Starter *Mikado Assistant*. Der Assistant erlaubt dann sowohl das Öffnen existierender Zellen, als auch neue Zellen zu erstellen. Die erstellten Zellen beinhalten dann:

- Fertig verknüpfte Schnittstellen für einen gewählten Roboter-, Greifer- und Kameratyp
- Modell zur Kollisionsprüfung
- Voreinstellungen für Mikado-Parameter

7.3.1 Neue Mikado Control Zelle Erstellen

Klicken Sie nach Starten des Assistants auf *Neue Control Zelle*. In dem folgenden Auswahlmenü lässt sich die Zelle in wenigen Schritten grob vormodellieren und die wichtigen Einstellungen zu Roboter- und Greifertyp treffen. Nach dem Erstellen kann im Assistant dann per Editor das Modells genau an Kundenanforderungen angepasst werden. Bei der Parameterwahl ist zu beachten, dass sowohl Zellname, Roboter- und Greifertyp im Nachhinein nicht mehr geändert werden können.

Hinweis: Die Posen werden im Mikado Programm immer im Welt-Koordinatensystem gerechnet. Das Welt-Koordinatensystem wird per Mikado Standard auf das des Roboters gesetzt. Dies hat den Vorteil, dass sowohl im Teach Panel, als auch im Mikado Programm mit den gleichen Koordinaten gearbeitet werden kann. Möchten Sie eine unabhängiges Koordinatensystem, zB. für die Roboter Montage Überkopf, so müssen sie in der *Erweiterten Modus* wechseln (Im Visualisierungsreiter *Konfiguration*, *Erweiterter Modus*). Es erscheint dann die Einstellmöglichkeit *Welt- identisch mit Roboterkoordinatensystem*, die auf false gesetzt werden kann.

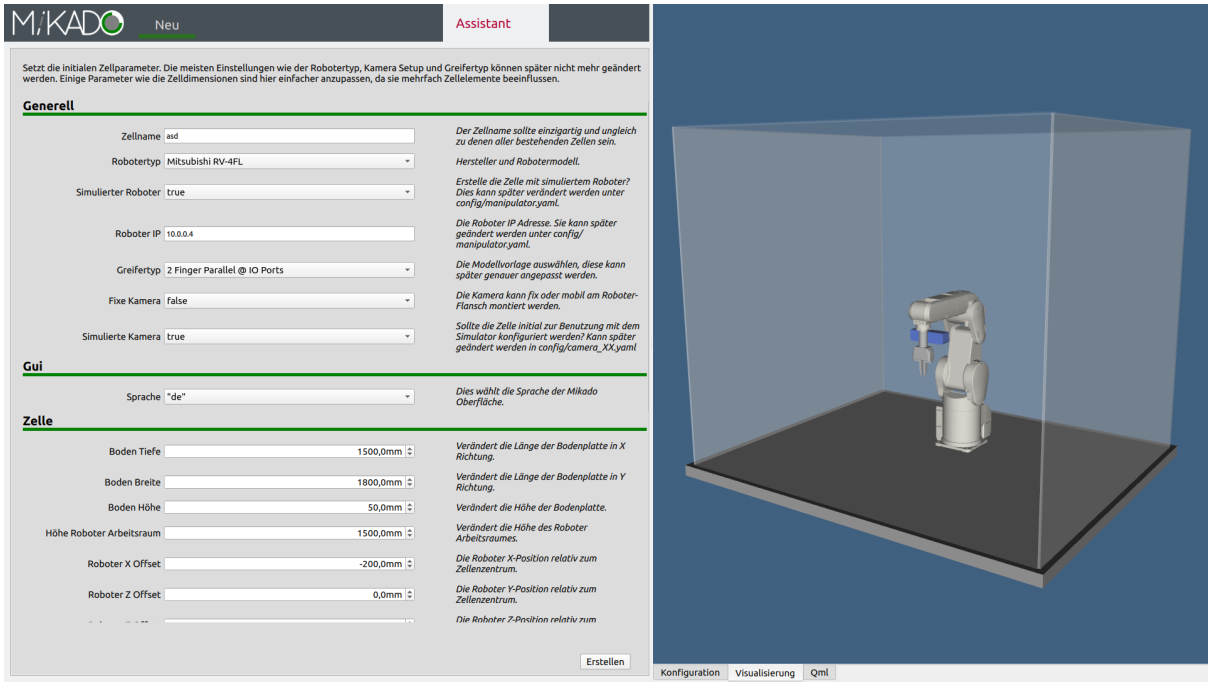


Abb. 3: Screenshot Assistant Übersicht Zellerstellung

7.4 Anpassen einer vorhandenen Zelle

Eine bereits existierende Zelle besitzt eine Vielzahl globaler Parameter, die sie konfigurierbar macht.

7.4.1 Geschwindigkeitseinstellungen

Für jedes Gelenk des Roboters muss definiert werden, wie weit es bewegt werden darf, wie schnell es bewegt werden darf und wie stark es beschleunigt werden darf. Dafür werden für jedes Gelenk vier Grenzwerte definiert:

- max_position: maximale Position des Gelenks in rad
- min_position: minimal Position des Gelenks in rad
- max_velocity: maximale Verfahrgeschwindigkeit des Gelenks in rad/s
- max_acceleration: maximale Beschleunigung des Gelenks in rad/s^2

Die Bedeutung der beiden zuletzt genannten Werte ist in unten stehender Abbildung dargestellt.

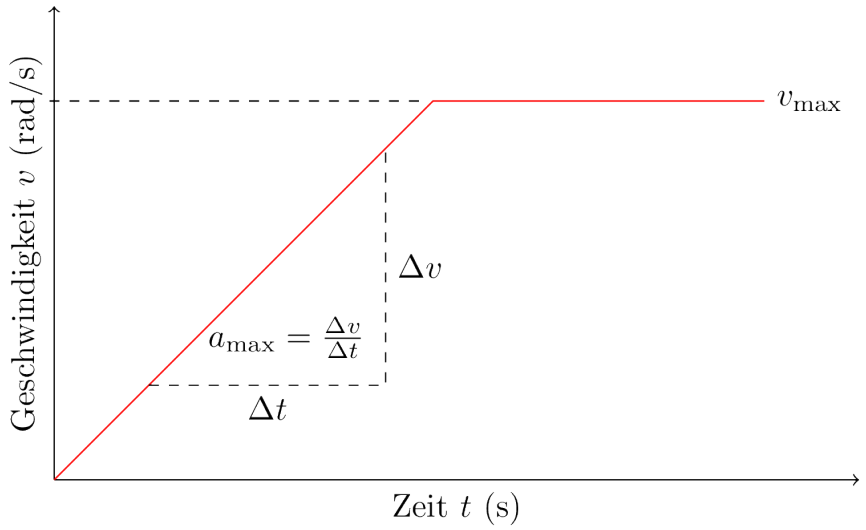


Abb. 4: Bedeutung von Maximal-Geschwindigkeit und Beschleunigung

Die Verfahrgeschwindigkeit des Roboters bei einer Steuerung durch Mikado hängt von unterschiedlichen Faktoren ab.

Absolutwerte

Zunächst besitzt jedes Robotergelenk physikalische Grenzen, die nicht überschritten werden dürfen. Die Grenzen bezüglich Position und Geschwindigkeit werden für gewöhnlich vom Hersteller angeben. Diese Herstellerangaben sind im Urdf des verwendeten Roboters hinterlegt, welches im Installationsverzeichnis zu finden ist:

```
/opt/isys-vision/mikado/share/<robot_model>_support/urdf/<robot_model>_macro.xacro
```

Dort sind in den Joints (siehe *Joints*) eines jeden Roboter-Links die jeweiligen Grenzwerte für Position und Geschwindigkeit in folgender Form definiert:

```
<limit lower="${radians(-120)}" upper="${radians(120)}" effort="0" velocity="$
↪{radians(623)}" />
```

Es fällt auf, dass es hier keinen Wert für die Steigung der Beschleunigungsrampe gibt. Dies liegt daran, dass die maximal zulässige Beschleunigung stark abhängig vom Gewicht des verwendeten Greifers und des gegriffenen Teils ist. Deshalb gibt es hierzu keinerlei universal gültigen Herstellerangaben. Für die Berechnung der Trajektorienpunkte benötigt Mikado jedoch diesen Wert, weshalb standardmäßig eine maximale Beschleunigung von 1 rad/s^2 angenommen wird. Dieser Wert ist relativ niedrig und muss inkrementell erhöht werden, bis die gewünschte Taktzeit erreicht ist oder der Roboter einen Fehler wirft.

Um die vier Grenzwerte bequem einstellen zu können, bietet Mikado die Konfigurationsdatei *joint_limits.yaml*.

Hier können die Absolutwerte für max_position, min_position, max_velocity und max_acceleration gesetzt werden.

Bemerkung: Die in `joint_limits.yaml` definierten Werte für min_position, max_position und max_velocity überschreiben die entsprechenden Werte im URDF; unabhängig davon ob sie größer oder kleiner sind.

Relativwerte

Typischerweise wird die Geschwindigkeit eines Roboters am Teach-Panel über einen Skalierungsfaktor eingestellt. Dieser Skalierungsfaktor taucht auch in Mikado auf und kann in der Konfigurationsdatei *manipulator.yaml* eingestellt werden:

- `velocity_scaling` wird hierbei mit der geltenden, absoluten max_velocity multipliziert.
- `acceleration_scaling` wird hierbei mit der geltenden, absoluten max_acceleration multipliziert.

Diese beiden Skalierungen werden permanent in `manipulator.yaml` festgelegt, können aber auch temporär zur Laufzeit von Mikado geändert werden. Für manche Roboter (bspw. Mitsubishi) ist dies unter Tools/Roboter möglich. Immer ist eine temporäre Änderung möglich mittels eines JavaScript-Tasks unter Verwendung von *setRobotVelocityScaling* bzw. *setRobotAccelerationScaling*.

Unterschiedliches Verhalten bei unterschiedlichen Robotern

Erhält ein Roboter-Controller eine Abfolge von Trajektorienpunkten (bspw. eine von Mikado geplante Bahn), wird diese stückweise bearbeitet. Einige Roboter-Controller (bspw. Fanuc) besitzen jedoch einen begrenzten Planungshorizont. Dementsprechend wird zum Anfahren des nächsten Punktes nur eine begrenzte Anzahl der nachfolgenden Trajektorienpunkte betrachtet. Dabei wird die sicherheitsrelevante Annahme getroffen, dass der zuletzt betrachtete Punkt dem Endpunkt der Bahn entspricht. Typischerweise muss der Roboter am Endpunkt der Bahn zum Stillstand kommen. Dies führt bei Bahnen mit dicht beieinander liegenden Trajektorienpunkten dazu, dass niemals auf die maximale Geschwindigkeit hoch geregelt wird.

Der freie Bahnplaner von Mikado erzeugt sehr viele Trajektorienpunkte um sicher zu gehen, dass keine Kollisionen auftreten. In Kombination mit einem Controller, der bei jedem angefahrenen Punkt auch anhalten könnte, führt dies zu einer sehr langsamen Fahrt. Aufgrund dieser Ursache lässt sich hier auch nichts über die Skalierungsfaktoren ändern. Durch das Hinzufügen von geeigneten Zwischenposen lässt sich der Einsatz des freien Bahnplaners jedoch vermeiden. Dabei muss der direkte Weg (im Gelenkraum) zwischen den einzelnen Zwischenposen kollisionsfrei abfahrbar sein. Andernfalls wird der Bahnplaner wiederum bemüht. Folgende Abbildung zeigt beispielhaft eine eingefügte Zwischenpose „intermediate“ zwischen der Start- und Endpose.

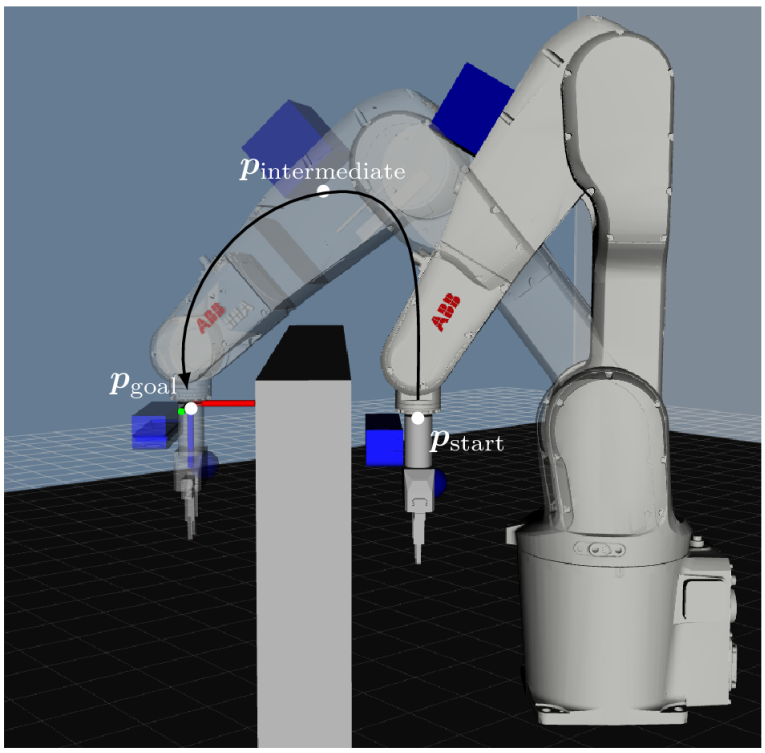


Abb. 5: Beispielhaft eingefügte Zwischenpose, um den Einsatz des freien Bahnplaner zu vermeiden

Bemerkung: Wenn sich der Roboter bei frei-geplanten Fahrten sehr langsam bewegt, können Sie durch das Hinzufügen von geeigneten Zwischenposen die Aktivierung des freien Bahnplaners vermeiden.

Folgende Links verweisen auf Roboter-spezifisches Geschwindigkeits-Tuning:

Tab. 2: **Links zu Roboter-spezifischem Geschwindigkeits-Tuning**

Kuka Roboter:	*Geschwindigkeit und Fine-Tuning*
Universal Robots:	*Geschwindigkeit und Beschleunigung*

7.4.2 Konfigurationsdateien

Im Ordner „config/“ finden sich einige Dateien zum Einstellen von Eigenschaften der Oberfläche, aber auch zum Parametrisieren der Objektsuche und Pfadplanung. Von besonderem Interesse dürften *access_control.yaml*, *mikado.yaml* sowie *status.yaml* sein.

access_control.yaml

- *enabled*: (true/false) legt fest ob die einfache Zugriffskontrolle aktiviert sein soll. Für alle Seiten der Oberfläche auf denen Änderungen vorgenommen werden können, ist dann die Eingabe eines Passworts notwendig. Der Effekt wird auch beschrieben im Kapitel *Passwortschutz*.
- *password*: (string) Legt das Passwort fest. Die Zugangskontrolle sollte ausschließlich als Schutz vor versehentlichem Zugriff verstanden werden und nicht als Sicherheitsmechanismus gegen bewusste Manipulation. Das Passwort wird im Klartext in der Konfigurationsdatei *access_control.yaml* gespeichert und kann einfach ausgelesen werden. Daher sollte kein Passwort eines anderen Systems wiederverwendet werden.

Im Mikado Assistant können diese Einstellungen unter Zelle > Anwendung > Zugangskontrolle vorgenommen werden.

calibration.yaml

Einige Parameter für die automatische Kalibrierung, bzw. für die Generierung neuer Aufnahmeposen können in dieser Datei eingestellt werden:

- *targetLink*: Bezeichnung des Links, der für die Generierung der Roboterposen verwendet wird. (Optional, Default: Handgelenk des Roboters)
- *max_translation_before_precalib*: Maximale Verschiebung der generierten Posen zur Ausgangspose bevor eine Vorkalibrierung durchgeführt wurde [in Meter]
- *max_translation_after_precalib*: Maximale Verschiebung der generierten Posen zur Ausgangspose nachdem eine Vorkalibrierung durchgeführt wurde [in Meter]
- *rotation_before_precalib*: Rotation generierter Posen zur Ausgangspose bevor eine Vorkalibrierung durchgeführt wurde [im Bogenmaß]
- *rotation_after_precalib*: Rotation generierter Posen zur Ausgangspose nachdem eine Vorkalibrierung durchgeführt wurde [im Bogenmaß]
- *check_pose_visibility*: (true/false) Hiermit kann der Sichtbarkeitstest nach der Vorkalibrierung explizit deaktiviert werden (Optional, Default: true)
- *max_failed_moves*: Eine zufällig generierte Pose kann gelegentlich vom Roboter nicht erreicht werden (z.B. bei Kollision oder keine Inverse Kinematik gefunden). Dieser Grenzwert definiert die maximale Anzahl an Versuchen bei der der Kalibriervorgang abgebrochen wird.

- *max_failed_visible*: Nach einer erfolgreichen Vorkalibrierung wird bei jeder generierten Pose geprüft, ob das Kalibriermuster im Sichtfeld liegen wird. Dieser Grenzwert definiert, wieviel Versuche die Kalibrierung maximal pro Pose starten darf. Wird danach immer noch keine Pose gefunden, bricht die Kalibrierung ab.

Hinweis: Die Werte nach der Vorkalibrierung sollten mehr Spielraum zulassen um ein besseres Kaliberungsendergebnis zu erhalten.

---

camera_X.yaml

Für jede Kamera wird eine Parameterdatei angelegt. Diese beinhaltet:

- *camera_serial*: Die Kameraseriennummer, die zum Etablieren der Kommunikation benutzt wird.
- *fixed_camera*: Legt fest, ob die Kamera als fest oder beweglich am Roboterarm behandelt wird
- *camera_config*: Pfad zu der beim Start geladenen Konfigurationsdatei (siehe *Kamerakonfiguration*), zum Beispiel `cellname:config/150001.ensparam`. Filekameras verwenden diese nicht.
- *mono_cameras* (optional): Möchten Sie eine zusätzliche monokulare Kamera verwenden, kann diese hier eingebunden werden (z.B. zur Verwendung mit *Halcon Scripts*)
  - *serial_no*: Seriennummer der monokularen Kamera (z.B. `4101230815`)
  - *config*: Pfad zu der beim Start geladenen Konfigurationsdatei für diese monokularen Kamera z.B. `cellname:/config/4101230815.ensparam`

Warnung: Filekameras die in einem *.zip liegen werden nicht unterstützt und müssen in einen Ordner entpackt werden.

gripper.yaml

In der Konfigurationsdatei *gripper.yaml* wird an oberster Stelle mit dem Parameter *type* der Greifer Typ spezifiziert. Dabei gibt es die folgenden Kategorien:

- *type*:
  - *None:* Ein unbeweglicher „Greifer“ ohne weitere Besonderheiten
  - *Magnetic:* ein magnetischer Greifer
  - *Vacuum:* ein Vacuumgreifer, hier wird das Vacuum schon während der Anrückfahrt geschaltet um ein Verrücken durch den Vakuumsauger (insbesondere bei Balgsaugern) zu vermeiden.
  - *ParallelTwoStates:* ein einfacher Pneumatikgreifer mit *Offen* und *Geschlossen* Zuständen. Kann sowohl für Innen- als auch für Außengriffe verwendet werden. Funktioniert natürlich auch mit einfachen Elektrogreifern.
  - *ParallelPositionControl:* ein Elektrogreifer der auf beliebige Positionen geregelt werden kann.
  - *ParallelForceControl:* ein Elektrogreifer mit Kraftmessung und Kraftregelung.

Die übrigen Parameter sind in nachfolgender Tabelle aufgeführt. Einige Parameter sind je nach Greifertyp optional und daher eventuell nicht Teil Ihrer Konfigurationsdatei. In der Spalte *Robotertyp* ist für jeden Parameter angebenen ob er für einen speziellen Robotertyp existiert. Die Spalte *Greifertyp* gibt an, ob ein Parameter für einen speziellen Greifertyp exisitiert. Dabei bezeichnet *IO* mittels IOs angesteuerte Greifer, *Finger* bezeichnet Greifer mit mindestens zwei Fingern und *WSG* bezeichnet Greifer der WSG-Serie der Fa. Weiss.

Tab. 3: **Übersicht der Greifer-Parameter**

	Robotertyp	Greifertyp

Fortsetzung auf der nächsten Seite

Tab. 3 – Fortsetzung der vorherigen Seite

Ausgänge			
open_hand_signal_number	Signalnummer (IO-Nummer) mit der der Greifer in den Offen-Zustand geschaltet werden kann	alle	IO
open_low_active	Ist der Offen-Zustand des Greifers low-active, diesen Parameter auf *true* setzen	Mitsubishi, Techman	IO
close_hand_signal_number	Signalnummer (IO-Nummer) mit dem der Greifer in den Geschlossen-Zustand geschaltet werden kann	alle	IO
close_low_active	Ist der Geschlossen-Zustand des Greifers low-active, diesen Parameter auf *true* setzen	Mitsubishi, Techman	IO
Eingänge			
is_open_signal_number	Signalnummer (IO-Nummer) an der der Offen-Zustand des Greifers signalisiert wird	Mitsubishi, Techman, UR	IO
is_open_low_active	Ist der verwendete Sensor for den Offen-Zustand low-active, diesen Parameter auf *true* setzen	Mitsubishi, Techman, UR	IO
is_closed_signal_number	Signalnummer (IO-Nummer) an der der Geschlossen-Zustand des Greifers signalisiert wird	Mitsubishi, Techman, UR	IO
is_closed_low_active	Ist der verwendete Sensor for den Geschlossen-Zustand low-active, diesen Parameter auf *true* setzen	Mitsubishi, Techman, UR	IO
Finger Konfiguration			
joint_finger_name	Name des Gelenks des führenden Greiferfingers (z.B. ‚finger_left'). Kann auf ‚' gesetzt werden, wenn Greifer keine beweglichen Gelenke hat.	alle	Finger
joint_value_open	[1] Wert des Gelenks in Meter, wenn der Greifer geschlossen ist (z.B. 0.0). Dieser Wert wird verwendet um z.B. elektrische Greifer mit Linearachsen anzusteuern.	alle	Finger
joint_value_closed	Seite 278, [1] Wert des Gelenks in Meter, wenn der Greifer geöffnet ist (z.B. 0.07).	alle	Finger
Zeitplan			
wait_time_before_pick	Wartezeit in Sekunden an der Übergangspose eines Greifvorgangs, falls der Greiferzustand für den Griff geändert werden muss	alle	alle
wait_time_pick	Wartezeit in Sekunden direkt nachdem der Greifer ein Teil gegriffen hat	alle	alle
finger_state_publish_rate	Update-Rate des Finger- (oder Greifer-)Gelenks in Hz	alle außer UR, Techman	alle
WSG			
speed	Geschwindigkeit der Greifer-Achse im m/s	alle	WSG
acceleration	Beschleunigung der Greifer-Achse im m/s^2	alle	WSG
force	Greifkraft der Greifer-Achse im Newton	alle	WSG

[1] Die Werte für *joint_value_closed* und *joint_value_open* müssen ebenfalls im *URDF* des Greifers als Joint Limits (‚lower' und ‚upper') eingetragen werden! Nutzen Sie den Mikado-Assistant, um das URDF des Greifers anzupassen. Details zur URDF-Parametrierung von Gelenken finden Sie in Sektion *Joints*.

joint_limits.yaml

Diese Datei legt die Grenzen für die einzelnen Gelenke fest. Es können folgende Parameter angeben werden:

- *min_position* minimale Position
- *max_position* maximale Position (beide in Radiant also π (pi) bzw 3.141 entspricht 180°)
- *has_velocity_limits* (true/false) sind Geschwindigkeitsbegrenzungen vorhanden
- *max_velocity* maximale Geschwindigkeit (Rad/Sekunde)
- *has_acceleration_limits* (true/false) sind Beschleunigungsgrenzen vorhanden
- *max_acceleration* maximale Beschleunigung (Rad/Sekunde^2)

Hierbei gelten leider die folgenden Einschränkungen:

- momentan verwenden wir für einige Robotertypen eine Steuerung die rein auf Trajektorien (also keine Geschwindigkeits- oder Zeitkomponenten enthält) beruht. Die maximale Geschwindigkeit und Beschleunigung ergibt sich dann aus den Einstellungen des Roboter-Controllers (bzw. dessen Leistungsfähigkeit).
- Auch die Planer berücksichtigen zur Zeit z.B. lange Beschleunigungsrampen nicht. Es wird rein auf den kürzesten Weg optimiert.

Die hier angegebenen Werte überschreiben die theoretischen Grenzen der Hardware wie sie im URDF/XACRO im Roboterpaket angegeben wurden. Wir empfehlen auch auf dem Robotercontroller Gelenkgrenzen zu setzen und im *joint_limits.yaml* geringfügig engere Grenzen zu setzen.

manipulator.yaml

- *simulated_robot:* true/false Verwende simulierten Roboter anstatt eine Verbindung zum realen Roboter herzustellen
- *robot_ip:* z.B. „192.168.2.1“ Tragen Sie hier die eingestellte IP des Roboters ein
- *manufacturer:* z.B. „Universal Robots“ Hersteller des Roboters, schaltet ggf vorhandene Spezialfunktionen in Mikado Control frei.
- *model:* z.B. „UR5“ bisher nur zu Dokumentationszwecken
- *velocity_scaling:* (0.01-1.0) Skalierungsfaktor auf die Maximalgeschwindigkeit die im URDF festgelegt wurde. Kann temporär in der Oberfläche unter Tools/Roboter verstellt werden.
- *max_velocity_scaling:* (0.01-1.0) Obergrenze für den temporären Wert in der Oberfläche
- *acceleration_scaling:* (0.01-1.0) Skalierungsfaktor auf die maximale Beschleunigung die im URDF festgelegt wurde. Kann ebenfalls temporär in der Oberfläche verstellt werden.
- *max_acceleration_scaling:* (0.01-1.0) Obergrenze für den temporären Wert in der Oberfläche

mikado.yaml

Diese Datei legt einige Eigenschaften der Oberfläche fest:

- *separate_windows*: false Visualisierung und Bedienoberfläche in separaten Fenstern, bei mehreren Monitoren verteilt auf diese
- *program_file_name*: „project:programs/Motek.yaml“ ein Programmablauf der direkt geladen werden soll (optional)
- *product_list_file_name*: „project:products/T-Stueck_productList.yaml“ eine Produktliste die direkt geladen wird (optional)
- *scene_objects_file_name*: „project:scenes/EnvObjects.yaml“ eine Datei mit den Szenenobjekten (optional)

- *debug_qml*: true/false Im Qml-Debug Modus können die Programmoberfläche und insbesondere die anlagenspezifischen Seiten ohne Neustart des Programms neu geladen werden. Hierzu erscheint ein neuer Reiter „Qml“ unterhalb der Visualisierung.

Dabei bedeutet der Prefix *project:* das die Angaben relativ zum Projektverzeichnis, also zum Verzeichnis der Roboterzelle sind.

object_recognition.yaml

In dieser Datei werden Parameter für die Objekterkennung angegeben die für alle Produkte gelten. Einige Parameter lassen sich nicht über die Mikado-Oberfläche beeinflussen, da diese beim Systemstart bereits festgelegt sein müssen. Die folgenden Parameter müssen eingestellt werden:

- ***voxel_size***
  Voxelgröße. Ein 3D-Punkt wird bei der Kollisionsplanung als „Voxel“ betrachtet, also als kleiner Würfel. Hier kann die Seitenlänge eines solchen Würfels festgelegt werden. [mm] *Default: 2.0*

  Bemerkung: Die in der Ansicht dargestellte Voxelgröße (Octomap) kann in *sensor_manager.launch.xml* verändert werden.

- ***dilation_radius***
  Dilationsradius Objektsuche. Wird ein Objekt gefunden, so werden alle 3D-Punkte um dieses Objekt für die Objektsuche ausgeschnitten. Hier kann der maximale Abstand eingestellt werden, innerhalb dessen die Punkte entfernt werden. [Anzahl Voxel] *Default: 0*

- ***dilation_radius_object_cutting***
  Dilationsradius beim Ausschneiden von Matches. Wird ein Objekt gefunden, so werden alle Punkte um dieses Objekt für die Kollisionsplanung ausgeschnitten. Hier kann der maximale Abstand eingestellt werden, innerhalb dessen die Punkte entfernt werden. Sollte größer als der Dilationsradius der Objektsuche gewählt werden. [Anzahl Voxel] *Default: 2*

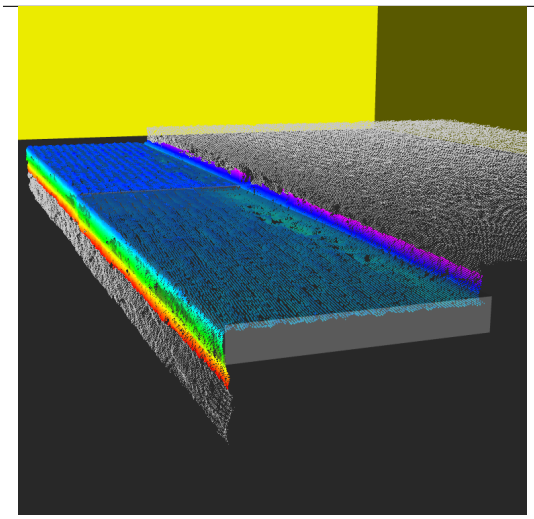


Abb. 6: *dilation_radius_object_cutting: 4*. Für eine Voxelgröße von 3 mm ergibt sich ein Ausschnitt von 4*3 mm = 12 mm.

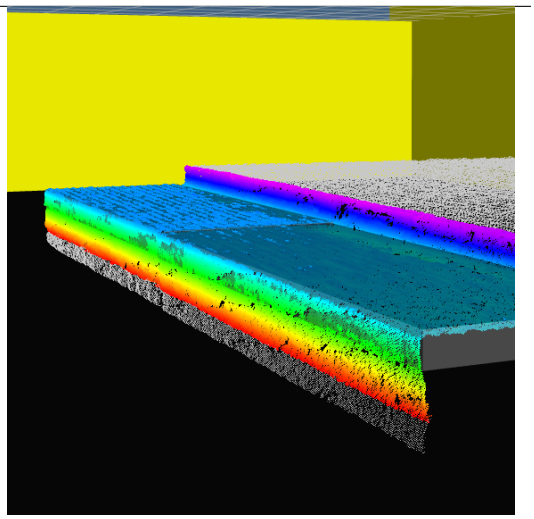


Abb. 7: *dilation_radius_object_cutting: 8*. Für eine Voxelgröße von 3 mm ergibt sich ein Ausschnitt von 8*3 mm = 24 mm.

- ***cuda_enabled***
  CUDA verwenden. Falls der Mikado-Controller mit einer NVIDIA Grafikkarte ausgerüstet ist, kann mithilfe von CUDA das Stereo-Matching beschleunigt werden. [bool] *Default: true*

- ***useFloat16***
  Beschleunigt die SCM- und SGM-Berechnungen (Matching Method/ComputeDisparityMap), sofern die GPU halbe Arithmetik unterstützt. [bool] *Default: true*

- ***halcon_available***
  Halcon verfügbar. Falls keine Halcon Lizenz verfügbar ist, können hiermit alle Halcon-abhängige Features deaktiviert werden. [bool] *Default: false*

Halcon spezifische Parameter:

- ***relative_sampling_distance***
  Relative Sampling-Distanz (Produkte). Abtastdistanz des Oberflächenmodells (Produkts) relativ zu dessen Durchmesser. [0.0 - 1.0] *Default 0.0375*
- ***pose_ref_rel_sampling_distance***
  Relative Sampling-Distanz für Verfeinerung (Produkte). Abtastdistanz des Oberflächenmodells (Produkts) relativ zu dessen Durchmesser für den Verfeinerungsschritt der Objektsuche. Sollte größer gewählt sein als der Parameter *Relative Sampling-Distanz (Produkte)* [0.0 - 1.0] *Default 0.015*

ompl_planning.yaml

Hier können Parameter für die verfügbaren Pfadplanungsalgorithmen angepasst werden. Mikado Control verwendet momentan standardmässig „RRTConnect“ und dies ist auch nicht umschaltbar.

pick_search.yaml

Einige der folgenden Begriffe sind in Sektion *Griff-Inspektion* näher erläutert.

- ***allow_OO_collisions_during_extraction:***
  Objekt/Octomap-Kollisionen aktivieren/deaktivieren
- ***allow_max_depth_collisions:***
  Gegriffenes Objekt darf mit anderen Objekten während der Entnahme- (Extract-) Bewegung kollidieren
- ***max_allowed_collision_depth:***
  Maximale „Tiefe“ einer erlaubten Kollision
- ***specific_max_allowed_collision_depths:***
  Maximale „Tiefe“ einer erlaubten Kollision für zwei spezifische Produkte/Objekte
- ***pick_selection_order:***
  Priorisierung der Griffe kann hier durch eine Änderung der Reihenfolge der Listenelemente beeinflusst werden
  - *score_bin_size:* Größe der Gruppierungsintervalle nach Score
  - *z_bin_size:* Größe der Gruppierungsintervalle nach z-Wert in *world*-Koordinaten
- ***search_fn_blacklist:***
  Einzelne Entnahme- (Extract-) oder Rückfahr- (Retract-)Funktionen abschalten
- ***allow_skipping_new_object_search:***
  3D Matching nach neuen Objekten überspringen, sofern durch Wiederverwenden bisheriger Funde genug mögliche Griffe vorhanden sind. Folgende Kriterien müssen erfüllt sein, damit die Objektsuche nicht ausgelöst wird:
  - *min_valid_picks:* Mindestanzahl der validen Griffe insgesamt
  - *min_valid_picks_per_object:* Anzahl der Griffe, die bei einem Objekt gefunden werden müssen, damit das Objekt als „greifbar“ eingestuft wird
  - *min_valid_objects:* Mindestanzahl der „greifbaren“ Objekte
- ***object_recognition_max_translation_diff:***
  Wiederverwenden von Funden: Maximaler Abstand des vorherigen Fundes, damit dieser als derselbe Fund klassifiziert wird

- ***object_recognition_max_rotation_diff:***
  Wiederverwenden von Funden: Maximale Verdrehung zum vorherigen Fundes, damit dieser als derselbe Fund klassifiziert wird
- ***interpolation_max_translation_diff:***
  Maximale Schrittweite zweier Roboterzustände in interpolierten Fahrten (Distanz)
- ***interpolation_max_rotation_diff:***
  Maximale Schrittweite zweier Roboterzustände in interpolierten Fahrten (Rotation)
- ***pick_search_timeout_in_seconds:***
  Wird ein Task im Mikado-Ablauf ausgeführt, der auf das Ergebnis der Griffsuche warten muss (z.B. der Task ‚*Teil greifen*'), so gilt die hier festgelegte maximale Rechenzeit. Wird diese Zeit überschritten, ohne dass ein valider Griff gefunden werden konnte, so bricht der Ablauf im Fehlerzustand ab.

sensors_ensenso.yaml

Hier wird konfiguriert, wie die von der Kamera aufgenommene Punktwolke für die Pfadplanung berücksichtigt werden soll. In der von uns verwendeten Standardeinstellung wird immer nur die letzte Aufnahme berücksichtigt und bekannte Geometrien wie z.B. der Roboterarm („Selfie") werden *nicht* rausgefiltert. Die gefundenen Produkte werden hingegen gefiltert, sodass sich dieser Modus z.B. gut für die Verwendung in einer Kiste eignet. Wird für *sensor_plugin* stattdessen *occupancy_map_monitor/PointCloudOctomapUpdater* verwendet, wird jede Punktwolke als „Update" der bisherigen Szene interpretiert. Über mehrere Aufnahmen mit einer beweglichen Kamera entsteht so eine Kollisionsgeometrie der Zelle. (Diese kann bisher nicht gespeichert werden und ist daher hier auch nur als nette Spielerei erwähnt.)

status.yaml

Diese Datei legt fest aus welchen Komponenten (zusätzlich zu einigen fest eingebauten für Kameras und die Mikado Control Lizenz) sich der Systemstatus ergibt. Typischerweise wird sich hier ein Eintrag für den Roboter, ein oder mehrere für die Kameras und ggf ein weiterer für Feldbus-Komponenten wie ProfiNet finden. Hier ein Beispiel für einen Roboter und eine ProfiNet Komponente:

```
status:
  - name: Robot
    type: RobotStatus
    topic: robot_status
  - name: Profinet
    type: RosDiagnosticStatus
    topic: profinet/status
    enabled_if_rosparam: /profinet
```

Das System kann einfach um eigene Komponenten für z.B. spezifische Hard- oder Software ergänzt werden. Sie müssen dazu nur regelmäßig (< 10s) eine Nachricht vom Typ *RosDiagnosticStatus* auf ein Topic ihrer Wahl publizieren (googlen sie nach ros publisher oder fragen Sie uns gerne nach einem Beispielprogramm).

view_settings.yaml

Eine vollständige Standard-Konfiguration der Ansichten jeder 3D-Visualisierung in Mikado. Hier werden z.B. Position und Ausrichtung der Szenenkamera gespeichert. Im Kapitel *Navigation* sind Informationen zur Nutzung der Speicherfunktion für die Ansichtseinstellungen zu finden.

world.srdf

Diese Datei beinhaltet einige Zusatzinformationen zum Aufbau der Zelle die nicht in der *Zellgeometrie* angegeben werden können. Hier sind unter Anderem alle Ausnahmen von der Kollisionsüberwachung aufgeführt. Standardmäßig erlaubt Mikado Kollisionen für folgende Kategorien:

- alle an *tool0* angehängten Objekte dürfen miteinander kollidieren,
- alle an *workcell_base* angehängten Objekte dürfen miteinander kollidieren,
- allen benachbarten Gliedern des Roboters dürfen paarweise kollidieren und
- die Gruppe aller an *tool0* angehängten Objekte darf mit Link_5 und Link_6 des Roboters kollidieren.

Dies beschleunigt das System z.B. bei stationären Objekten, kann aber auch notwendig sein wenn sich beispielsweise die Kollisionsgeometrien zweier benachbarter Abschnitte des Roboters ständig oder zeitweise minimal überschneiden. Weitere Ausnahmen können über das Feld „Neue Ausnahme hinzufügen“ zugelassen werden. Um eine benutzerdefinierte Ausnahme zu löschen, wählen Sie den entsprechenden Eintrag in der Liste aus und drücken Sie die Schaltfläche „Löschen“. Es können nur benutzerdefinierte Ausnahmen gelöscht werden.

Bemerkung: Wird ein neues Objekt direkt an einem Roboter-Link angeheftet, erzeugt Mikado keine automatische Ausnahme der Kollisionsprüfung, da keine der oben genannten Kategorien diesen Fall abdeckt. Sollte sich das neue Objekt und Eltern-(Roboter)-Link überschneiden, lässt sich der Roboter von Mikado nicht mehr bewegen, da eine Kollision erkannt wurde (Eltern-Objekt und das neue Objekt). Für diesen Fall muss die Ausnahme manuell angelegt werden.

Warnung: Die Kollisionsprüfung zwischen den hier aufgeführten Teilen wird komplett deaktiviert. Deaktivieren Sie die Kollisionsprüfung deshalb nur für Teile, die in der Realität nicht kollidieren können. Eine unüberlegte Deaktivierung kann schwere Materialschäden nach sich ziehen. Stellen Sie deshalb sicher, beispielsweise über maximale Achswinkel (siehe *URDF*), dass deaktivierte Kollisionen nicht auftreten können!

Hier zwei Beispiele:

```
<disable_collisions link1="base_link" link2="link_1" reason="Adjacent" />
<disable_collisions link1="base_link" link2="pane_back" reason="Never" />
```

Es wird die Kollision zwischen „base_link“ (dem Sockel des Roboters) und „link_1“ (dem ersten Segment des Roboters) deaktiviert, da diese sich leicht berühren und auch nur durch eine senkrechte Rotationsachse verbunden sind. Sie können also unabhängig von Roboterbewegungen nie in eine tatsächliche Kollision gebracht werden. Die Kollision zwischen „base_link“ und „pane_back“ (Rückwand der Zelle) wird deaktiviert, da beide fest montiert sind.

Außerdem sind die Einträge zur *gripper_group* sowie die *disable_collisions* Tabelle interessant. Die *gripper_group* legt fest, welche Teile des Greifers im Greifpunkteditor eingeblendet werden. Hierfür ein Beispiel aus dem 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

Die einzelnen Softwarekomponenten eines Mikado-Systems werden über sogenannte Launchfiles gestartet. Dabei handelt es sich um XML-Dateien mit einer relativ gut lesbaren Struktur. Wenn Sie den Mikado Assistant verwendet haben, sollten keine Änderungen an den Launchfiles nötig sein. Eine Ausnahme ist beispielsweise das nachträgliche Ändern des Robotertyps. Jede Zelle verfügt über eine Datei *launch/main.launch* die den eigentlichen Startvorgang des Systems beschreibt. Dieser können optional Argumente übergeben werden, zB „sim:=true“ um Mikado Control im Simulationsmodus zu starten. *main.launch* wird Typischerweise über das Bash-Skript „zellname.bash“ aufgerufen, welches wiederrum über die „zellname.desktop“ gestartet wird.

Die folgende Beschreibung der möglichen Einträge in einem Launchfile ist bewusst kurz gefasst, für weitere Informationen sei auf das ROS-Wiki (http://wiki.ros.org/roslaunch/XML) verwiesen (in Englisch).

- <arg> ein Parameter *name* innerhalb des Launchfiles der beim Aufruf des Launchfiles angegeben werden kann, wird er nicht angegeben wird *default* verwendet. Wird später dann per „$(arg name)“ entweder in Fallunterscheidungen (if/unless) oder als Wert verwendet.
- <group> kann verwendet werden um <node>s und Parameter zu gruppieren, entweder um einen kompletten Block per if/unless optional zu machen oder um die nodes und Parameter in einen Namespace zu verschieben.
- <include> lädt ein weiteres Launchfile, dabei können dessen Parameter ebenfalls per <arg> gesetzt werden
- <node> startet ein Programm unter der eindeutigen Bezeichnung *name*, es entstammt dabei dem Paket *pkg* und hat den Dateinamen *type*. Sollte sich das Programm geplant oder durch einen Absturz beenden, kann es per *respawn*=“*true*“ erneut gestartet werden. Falls hingegen *required*=“*true*“ angegeben wurde, wird das gesamte Launchfile in diesem Fall beendet.
- <param> setzt einen einzelnen Parameter auf dem zentralen Parameterbaum.
- <rosparam> wird meist verwendet um eine gesamte Konfigurationsdatei (.yaml) auf den Parameterbaum zu laden. Mittels *ns* kann auch in einen „Unterordner“ des Baumes geschrieben werden.

sensor_manager.launch.xml

Hier kann die Auflösung der dargestellten Octomap eingestellt werden, was insbesondere für Systeme ohne performante Grafikkarte relevant sein kann. Die dargestellte Octomap-Auflösung kann in folgender Zeile angepasst werden:

```
<param type="double" name="octomap_resolution" value="0.001"/>
```

Folgende Screenshots zeigen den Unterschied zwischen einer 1 mm-Auflösung und einer 4 mm-Auflösung.

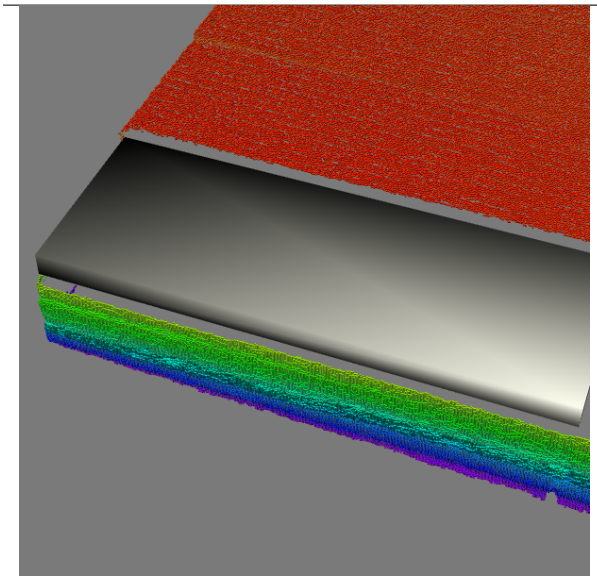


Abb. 8: *octomap_resolution = 0.001*

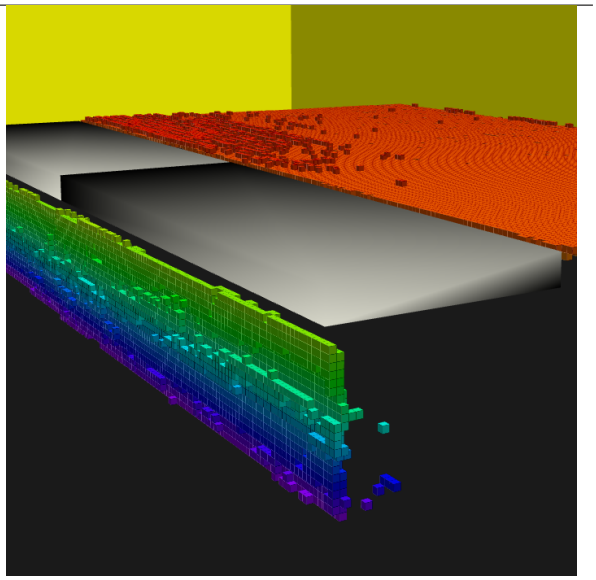


Abb. 9: *octomap_resolution = 0.004*

Bemerkung: Die Auflösung der Octomap hat nichts mit der Voxelgröße gemein, die in *object_recognition.yaml* definiert wird.

7.4.4 Zellgeometrie

Die programmunabhängige Geometrie (und Kinematik) der Roboterzelle wird über eine Datei im Unified Robot Description Format (*URDF*) beschrieben. Dieses Format ist ein XML Dialekt und kann mit einem beliebigen Texteditor bearbeitet werden.

Mikado organisiert URDF-Beschreibung der Roboterzelle in insgesamt fünf verschiedenen Dateien: *Global*, *Zelle*, *Greifer*, *Roboter* und *Kamera*, wobei Änderungen nur in den drei erstgenannten Dateien möglich sind. Die veränderbaren URDF Dateien sind im Unterverzeichnis *urdf* des Projektverzeichnisses abgelegt. Diese Unterteilung hilft dabei die einzelnen Module einer Roboterzelle übersichtlich und einfach austauschbar zu halten.

Die unten stehende Abbildung zeigt den hierarchischen Aufbau der Dateistruktur. Die *Global*-Datei bindet die anderen vier Dateien ein. Das *J* (für Joint/Gelenk) zeigt an, in welcher Datei die geometrische Verbindung zwischen den entsprechenden Modulen definiert ist.

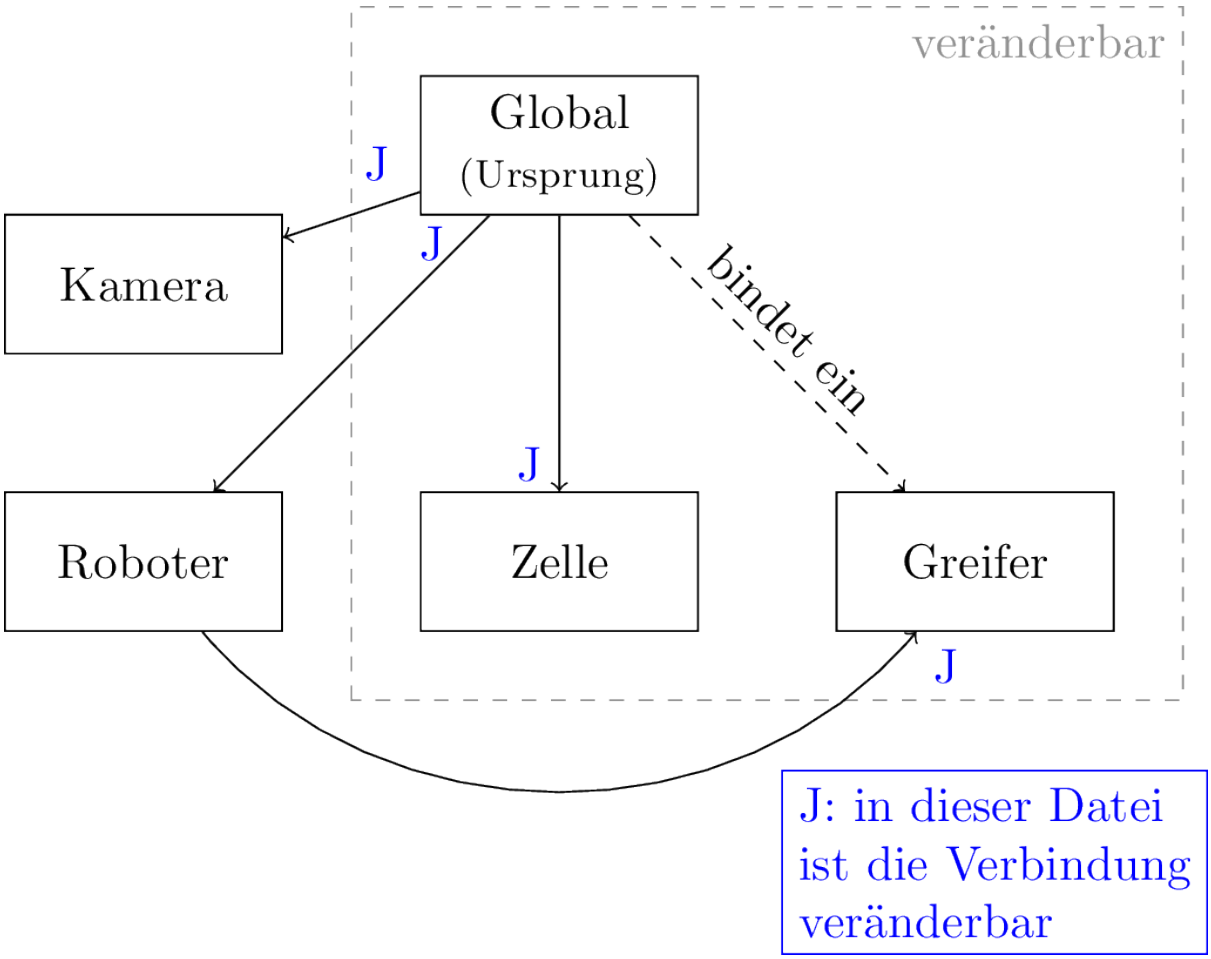


Abb. 10: Organisation der URDF Dateien

Folgende Tabelle gibt einen kurzen Überblick über den vorgesehenen Inhalt der einzelnen Dateien:

Tab. 4: **Übersicht der einzelnen URDF Dateien**

Global	Der globale Bereich verbindet den Roboter, die Zelle und den Greifer. Er kann benutzt werden um weitere Komponenten zum Roboter hinzuzufügen.
Zelle	Im Abschnitt Arbeitszelle werden alle Objekte definiert, die sich auf die statische Arbeitszelle beziehen. Fügen Sie hier ausschließlich stationäre Objekte hinzu, die nicht mit dem Roboter verbunden sind.
Greifer	Im Abschnitt Greifer werden alle Objekte definiert, die mit dem Greifer in Verbindung stehen. Fügen Sie hier nur Objekte hinzu, die direkt oder indirekt am Flansch des Roboters montiert sind. Der Werkzeugmittelpunkt (gripper_tcp), der für die Griffdefinition entscheidend ist, wird hier ebenfalls definiert.
Roboter	Hier wird die Geometrie des Roboters definiert. Der Inhalt dieser Datei ist vom Hersteller festgelegt und ist nicht veränderbar.
Kamera	Hier wird die Geometrie der Kamera definiert. Der Inhalt dieser Datei ist von Optonic festgelegt und ist nicht veränderbar.

Mikado bietet mit dem Assistant verschiedene Tools um die Zellgeometrie möglichst nutzerfreundlich zu modifizieren. Im Folgendem wird zunächst eine kurze Einführung in *URDF* gegeben und die Möglichkeiten aufgezeigt dieses mit *XACRO* zu strukturieren. Weiterhin werden die *URDF Einfüge-Hilfe* und die *URDF Editier-Hilfe* vorgestellt.

URDF

Die Unified Robot Description Format (URDF) Beschreibung einer Roboterzelle beginnt mit einem umschließenden <robot> Tag. Dieser hat ein Attribut *name*, welches mit dem Tag im zugehörigen *world.srdf* übereinstimmen muss. Darauf folgen eine Reihe von <link> und <joint> Tags. Ein <link> stellt dabei einen Knoten, ein Objekt oder weniger Abstrakt zB die Tisch/Bodenplatte der Zelle dar. <link>s werden über <joint>s miteinander verbunden. Diese können entweder starr oder beweglich sein und bilden eine Baumstruktur. Wir beginnen diesen immer mit dem Wurzelobjekt <link name="world_frame"/>. Dieser muss immer existieren und darf nicht umbenannt werden! Die untenstehende Abbildung zeigt eine mit URDF beschriebene Beispielzelle. Die Koordinatensysteme der einzelnen Links sind durch die dreifarbigen Koordinatenkreuze mit entsprechendem weißem Label dargestellt. Die durch gelbe Pfeile dargestellten Joints zeigen an, wie die einzelnen Links miteinander verbunden sind.

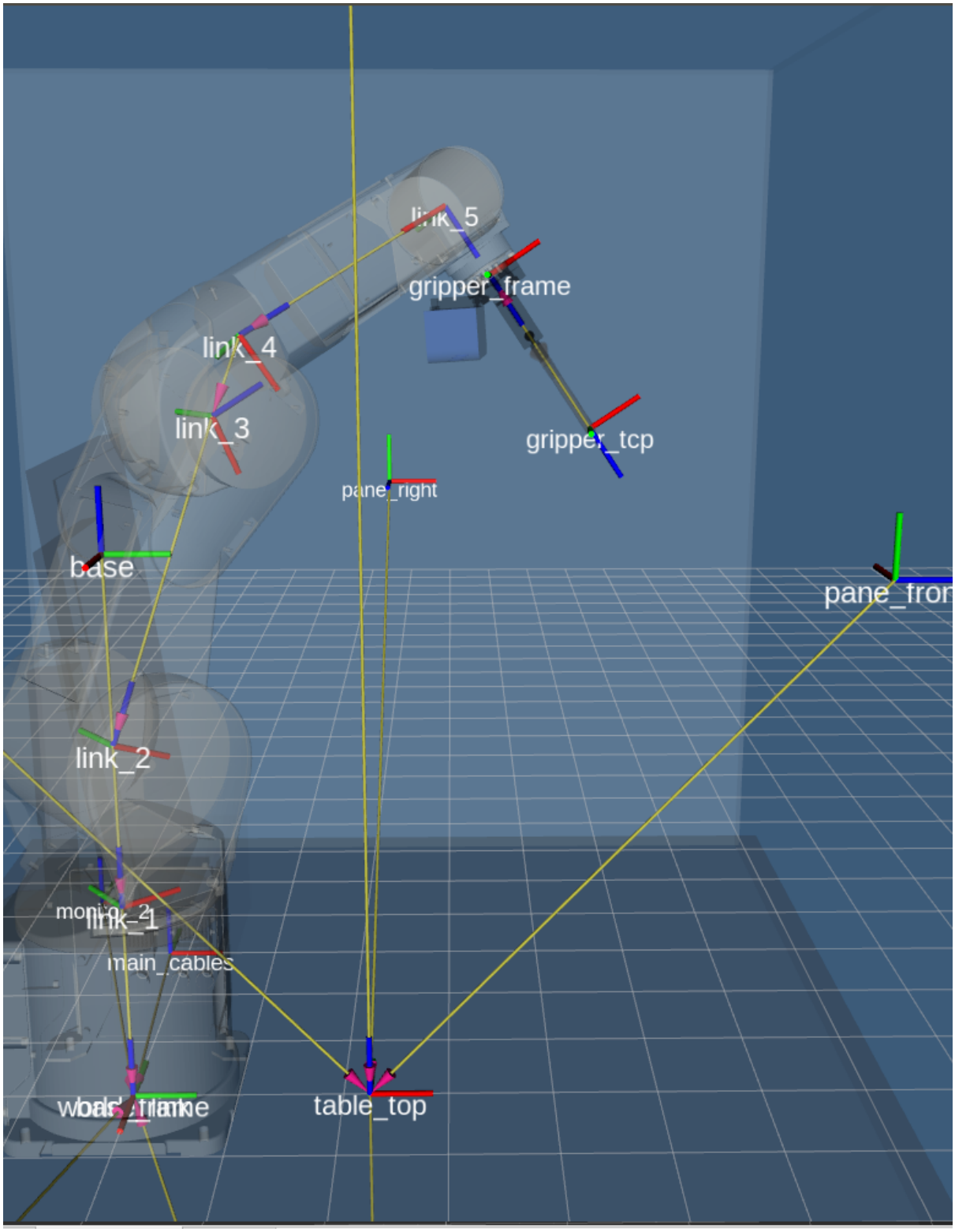


Abb. 11: Visualisierung einer mit URDF beschriebenen Roboterzelle

Links

Ein kleines Beispiel erläutert den Aufbau eines Links:

```
<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>
```

Der Knoten „mein_link“ besteht aus einem CAD/Mesh-Datei, welches die visuelle Erscheinung beschreibt und in Cyan/Türkis (der Name ist frei wählbar, entscheidend sind die RGBA Werte) eingefärbt wurde. Mesh-Dateien müssen entweder als STL (einfache Dreiecke, keine Farbinformationen) oder als Collada (.dae, Texturen etc) und in Metern vorliegen. Die visuelle Geometrie kann über <origin xyz=“0 0 0“ rpy=“0 0 0“ /> verschoben werden, allerdings ist übersichtlicher die CAD Datei passend zu verschieben (zB in Meshlab, SolidWorks, Blender, ...). Für die Kollisionsgeometrie sollte eine möglichst einfache Beschreibung verwendet werden, die aber immer gleich groß oder größer sein muss als die tatsächliche physikalische Geometrie. Hier wurde einfach ein Zylinder mit einem Radius von einem Meter und einer Höhe von einem halben Meter verwendet. Der Ursprung liegt in der Mitte des Zylinders. Wird der Ursprung eines URDF-Primitives verschoben, ist zu beachten, dass diese Verschiebung stets im ursprünglichen Koordinatensystem des Primitives geschieht. Folgendes Bild veranschaulicht die Verschiebung des Ursprung eines URDF-Primitives.

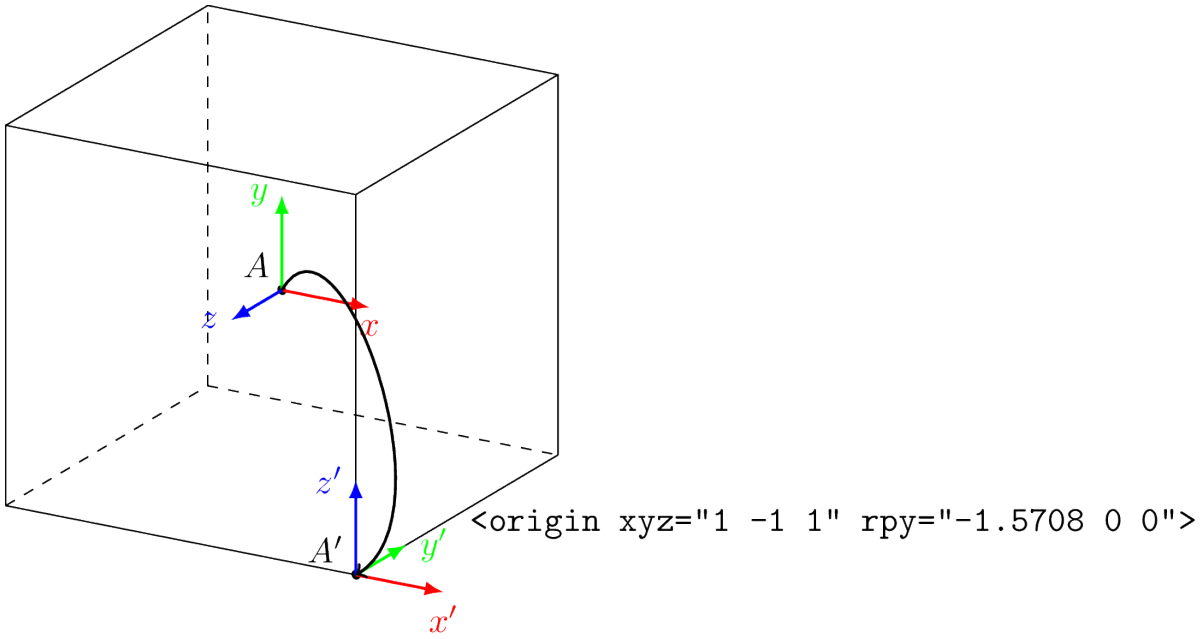


Abb. 12: Die Verschiebung des Ursprung eines URDF-Primitive

Sowohl für die visuelle als auch für die Kollisionsgeometrie können die Primitiven <box>, <cylinder>, <sphere> sowie CAD Modelle <mesh> verwendet werden:

```
<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" />
```

Es ist durchaus möglich mehrere geometrische Objekte in einem Link zu verwenden, allerdings sollte dies vermieden werden da es nur unzureichend getestet wird und daher nicht offiziell unterstützt wird. Dies gilt sowohl für Kollisionsgeometrien als auch für visuelle Geometrien.

Joints

Die einzelnen Links werden über Gelenke (Joints) miteinander verbunden. Im einfachsten Fall handelt es sich um eine starre Verbindung:

```
<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>
```

Hier wird festgelegt das ein Link mit dem Namen „table_base“ um 30cm in x-Richtung und um -50cm in z-Richtung verschoben wird. Dabei ist zu beachten, dass die in „<origin>“ Node spezifizierte Verschiebung stets im Koordinatensystem des Elternobjekts interpretiert wird. Folgendes Bild veranschaulicht die Verschiebung in einem Gelenk.

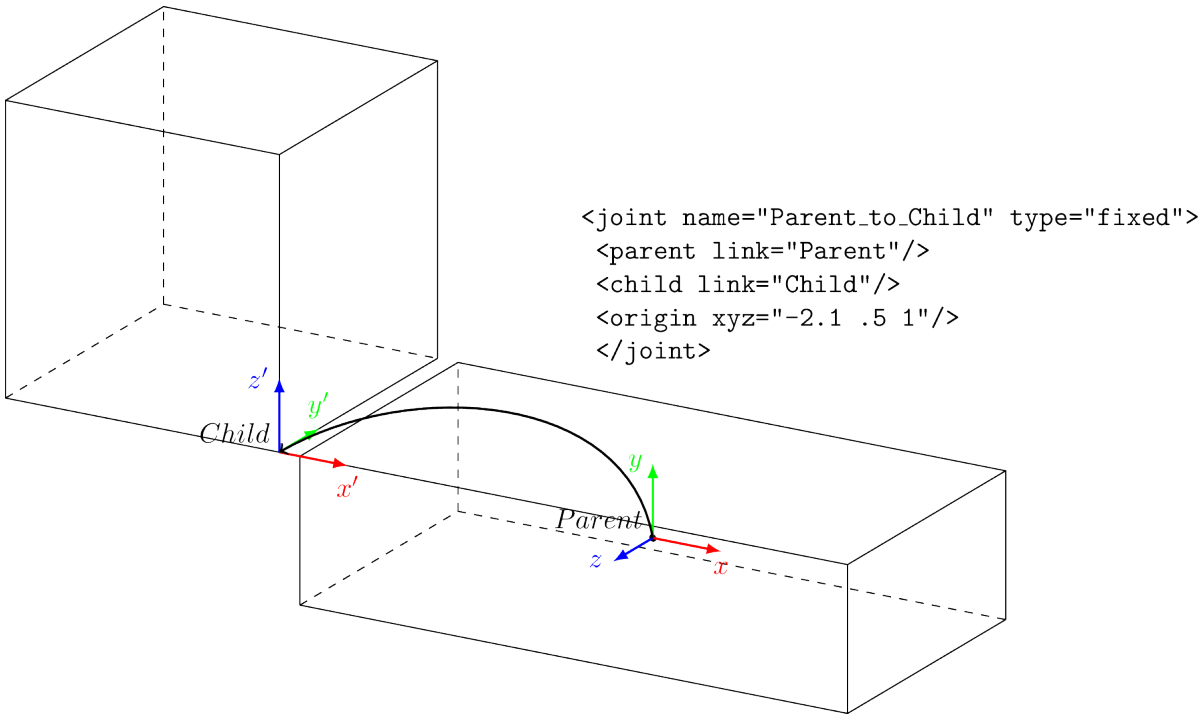


Abb. 13: Die Bedeutung der Verschiebung innerhalb eines Gelenks

Industrielle Roboter haben typischerweise 6 Rotationsachsen, solch eine Achse wäre dann beschrieben durch:

```
<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_
↪rad}" effort="0" />
 </joint>
```

Der Typ „revolute“ gibt hier an, dass es sich um ein Rotationsgelenk handelt. Dessen Ursprung wurde um 31cm von link2 verschoben und es wird um die y-Achse rotiert. Da die Roboter an sich typischerweise von Optonic modelliert werden, ist hier vorallem der letzte Eintrag <limit> von Interesse. *lower* gibt den minimalen Wert an, den das Gelenk annehmen kann, *upper* den maximalen. Diese Werte müssen in Radiant/Bogenmaß angegeben werden. Um die Verständlichkeit etwas zu verbessern wurde hier gleich etwas XACRO verwendet um von Grad umzurechnen. Die Grenzen müssen nicht symmetrisch sein und sind es oft auch nicht.

Warnung: Die Grenzen sollten immer auch im Controller eingestellt werden da Mikado Control nicht sicherheitsrelevant ist. Dabei ist darauf zu achten, die Grenzen im Controller ca 1° größer zu setzen als in Mikado Control. (Bei UR 3°). Versucht Mikado Control eine Position anzufahren die im Controller nicht erlaubt ist, gibt es meist sehr unschöne Notbremsungen die manuell quitiert werden müssen (aus Sicherheitsgründen). Werden die Grenzen identisch gesetzt kann dies durch Rundungsfehler dennoch auftreten.

velocity gibt die maximal erreichbare Geschwindigkeit des Gelenks in Radiant pro Sekunde an. Auch diese sollte etwas niedriger gewählt werden als die im Controller konfigurierte Maximalgeschwindigkeit. *effort* gibt im Prinzip die maximale Kraft des Motors an, wird aber in Mikado Control nicht verwendet. Die maximale Beschleunigung kann über die Datei *joint_limits.yaml* konfiguriert werden. Um Verwirrung zu vermeiden sollten dort die selben Werte angegeben werden.

Insbesondere in Greifern kommen auch lineare Gelenke zum Einsatz, hier ein Beispiel:

```
<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>
```

Hier wird durch „prismatic“ eine lineare Verschiebung angegeben, in diesem Fall entlang der x-Achse. Die Grenzen geben nun Meter anstelle von Radiant an. Gerade bei einem Parallelgreifer möchte man beide Finger gemeinsam bewegen. Dazu gibt es sog „mimic“-Joints (Pantomimegelenke). Der rechte Finger liese sich dann so an den linken koppeln:

```
<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>
```

Das Gelenk ist also identisch zum linken Finger bis auf eine zusätzliche Rotation von 180° um die z-Achse. Der <mimic> Tag bewirkt das die Stellung des Gelenks an die von „finger_left“ gekoppelt ist. Optional könnte auch noch ein *multiplier* und ein *offset* auf den Wert angewandt werden.

Die offizielle Dokumentation zu URDF finden Sie unter http://wiki.ros.org/urdf/XML/model Allerdings werden nicht alle dort erwähnten Tags und Gelenktypen offiziell von Mikado Control unterstützt.

XACRO

Öffnet man den urdf-Ordner in Dolphin (Dateimanager) fällt auf, dass dort nur Dateien mit der Endung *.xacro* liegen. XACRO steht für **X**ml M**acro** und ist ein Hilfsmittel um flexible und kompakte XML erstellen zu können, die modular sind und sich nicht ständig wiederholen. Schreibt man die komplette Zellbeschreibung inklusive Roboter in eine URDF-Datei wird diese recht groß und es ist umständlich zB die Größe der Zelle zu verändern. Daher verwenden wir eine Makrosprache die URDF um die Möglichkeit erweitert Variablen zu definieren, Werte auszurechnen (zB Grad in Radiant) und zusätzliche Dateien einzubinden. Im Launchfile wird dann automatisch eine reine URDF Datei aus dieser XACRO Datei erzeugt. Um manuell eine URDF Datei zu erzeugen verwenden Sie den Befehl:

```
xacro --inorder datei.xacro > datei.urdf
```

Dies kann insbesondere zum Testen einer Änderung hilfreich sein. Nun zu einigen der Funktionen von XACRO:

Berechnungen

Variablen und die Grundrechenarten können verwendet werden:

```
<xacro:property name="diameter" value="4.3" />
<sphere radius="${0.5 * diameter}" />
```

Hier wird eine Kugel mit einem Durchmesser von 4.3m erzeugt indem der Durchmesser halbiert und als Radius verwendet wird. Einige Konstanten wie *pi* sind bereits definiert und es kann sogar beliebiger Python Code verwendet werden. Ein weiteres Beispiel (es gibt allerdings keinen 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)}" />
```

Es bietet sich an immer eine Variable *deg_to_rad* zu definieren und dann die deutlich intuitiveren Gradangaben verwenden zu können:

```
<xacro:property name="deg_to_rad" value="${1.0/180*pi}" />
```

URDF Einfüge-Hilfe

Um das Einfügen von Urdf-Primitiven zu erleichtern bietet der Mikado Assistant eine Einfüge-Hilfe. Diese kann für die drei Urdf-Seiten *Zelle*, *Greifer* und *Global* mittels der CheckBox „Einfüge-Hilfe“ am unteren linken Fensterrand eingeblendet werden. Folgender Screenshot zeigt eine ausgefüllte Urdf Eingabe-Hilfe:

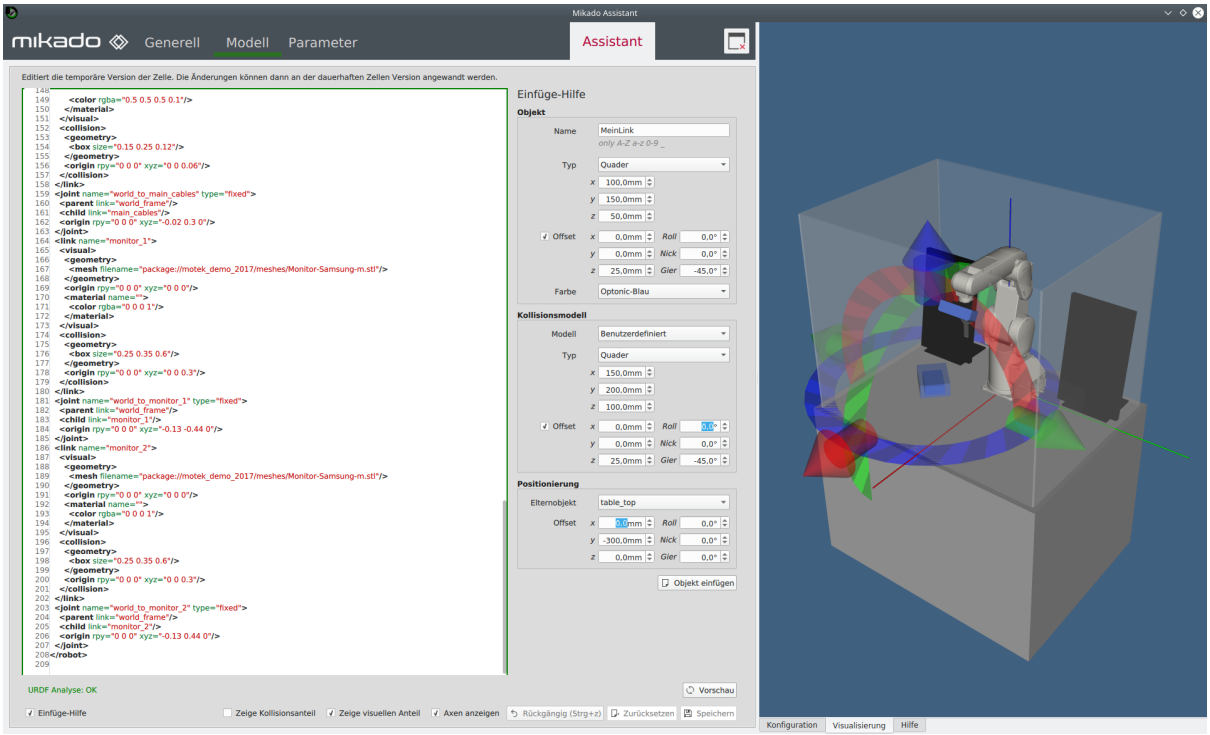


Abb. 14: Screenshot Assistant URDF Einfüge-Hilfe

Im gezeigten Beispiel soll ein Quader mit den Abmessungen 100x150x50 mm auf dem Boden der Zelle eingefügt werden. Hierzu wird der z-Wert des Ursprungs mit der Schaltfläche „Offset“ im „Objekt“-Kasten auf 25 mm gesetzt. Da der Quader in diesem Beispiel schräg in der Zelle steht, wird das „Gier“-Eingabefeld auf -45° gesetzt.

Als Kollisionsmodell soll ebenfalls ein Quader, jedoch mit etwas größeren Abmessungen verwendet werden. Dies kann im „Kollisionsmodell“-Kasten analog zu den „Objekt“-Einstellungen durchgeführt werden.

Im „Positionierungs“-Kasten lässt sich einstellen an welchem Elternobjekt das neue Objekt aufgehängt werden soll. Über das Dropdownmenü werden alle momentan in der Zelle enthaltenen Objekte zur Auswahl angeboten. In diesem Beispiel wird „base“ ausgewählt, da der neue Quader auf dem Boden stehen soll. Über die „Offset“-Schlatflächen lassen sich die relative Verschiebung und Verdrehung der Ursprünge von Elternobjekt und neuem Objekt einstellen.

Die Visualisierung auf der rechten Seite zeigt eine Vorschau des neuen Objekts an. Betätigt der Benutzer den ‚Objekt einfügen‘-Button, so wird für unser Beispiel folgender Codeblock erzeugt und in den Urdf-Editor auf der linken Seite eingefügt:

```
<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>
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
</link>

<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 Editier-Hilfe

Die URDF Editier-Hilfe ermöglicht eine ausschließlich graphisch basierte Modifizierung der Zellgeometrie. Die linke Seite des untenstehenden Screenshots zeigt die Baumstruktur der URDF-Objekte. Hier lässt sich ein URDF-Objekt auswählen. Das angewählte Objekt lässt sich anschließend mit der Maske auf der rechten Seite modifizieren. Analog zur *URDF Einfüge-Hilfe* lassen sich das visuelle und das Kollisionsmodell, sowie die Positionierung des Objekts über die entsprechenden Schaltflächen einstellen. Außerdem kann ein Objekt auch umbenannt oder dessen Farbe geändert werden.

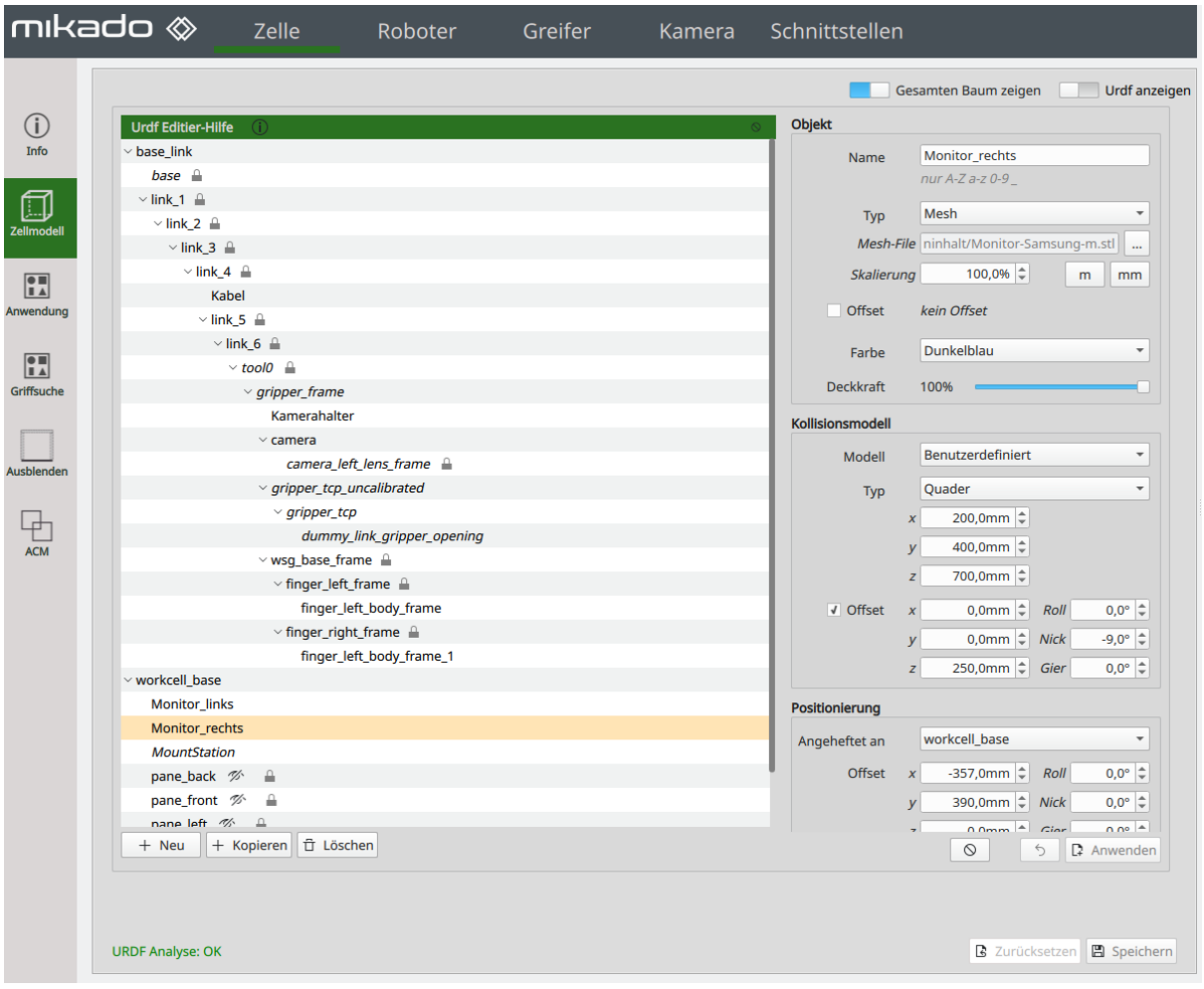


Abb. 15: Screenshot Assistant Editier-Hilfe

Änderungen an einem URDF Objekt müssen durch Betätigen des „Anwenden“-Knopfs bestätigt werden. Wird der Knopf nicht betätigt, gehen die Änderung der aktuellen Eingabemaske mit der Auswahl eines anderen Objekts oder dem Verlassen der Seite verloren. Um die Änderungen permanent zu speichern, drücken Sie den „Speichern“-Knopf am unteren rechten Rand.

Im gezeigten Beispiel weisen die unterschiedlichen Objekte verschiedene Schriftlagen und Symbole auf. Folgende Tabelle gibt eine kurze Übersicht über deren Bedeutung.

Tab. 5: **Legende zur Editier-Hilfe**

Aufrechte Schrift	Das Objekt besitzt eine Kollisionsgeometrie.
Schräggestellte Schrift	Das Objekt ist virtuell. Es besitzt keine Kollisionsgeometrie. Beachten Sie, dass es dennoch eine visuelle Geometrie besitzen kann.
	Das Objekt lässt sich nicht editieren. Dies kann unterschiedliche Gründe haben: - 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.
	Das Objekt ist in der vereinfachten Ansicht ausgeblendet. Die Konfiguration der vereinfachten Ansicht ist im Menü unter *Ausblenden* zu finden.

Mikado Key-Links

Folgende Urdf-Objekte werden von Mikado zwingend benötigt:

- `gripper_tcp` Griffpunkt des Greifers. Wird für die Griffdefinition und den Greifprozess verwendet. Eine Griffpose wird immer in positive z-Richtung des Koordinatensystems dieses Objekts angefahren.
- `tool0` Roboterflansch. Wird im jeweiligen Roboterpaket gesetzt. Dieses Objekt kann nicht geändert werden.
- `camera_left_lens_frame` Position der linken Monokamera. Wird nur vom Mikado Simulator verwendet.

7.5 Verzeichnisstruktur

Generell besteht ein Mikado-System aus einem allgemeinen Anlagen-unspezifischem Teil der bereits vorinstalliert ist (als Debian Paket) sowie der spezifischen Beschreibung ihrer Zelle. Dabei können auf einem PC durchaus mehrere Zellen nebeneinander eingerichtet werden, es sollte aber immer nur eine Applikation gleichzeitig gestartet werden. Zusätzliche Zellen, auch mit z.B. anderen Robotertypen, können beispielsweise zur Simulation einer geplanten neuen Anlage verwendet werden.

Alle vorhandenen Zellen finden sich im Ordner `mikado/cells` im Ordner `/home/mikado/mikado/cells` als Unterordner wieder. Eine Zelle mit dem Namen `beispielzelle` hat dabei folgende Struktur:

- `config/` enthält Konfigurationsdateien sowohl zur Oberfläche als auch zu Parametern der Objektsuche und Pfadplanung
- `launch/` enthält Informationen aus welchen Softwarekomponenten das System zusammengesetzt ist und wie diese zu starten sind
- `masks/` enthält Masken (Bilder) z.B. für die Suche von Szenenobjekten über ihre Kontur
- `meshes/` enthält 3D-Modeldaten der verwendeten Produkte, Kisten, Hindernisse etc.
- `products/` enthält die Produktbeschreibungen sowie Produktlisten
- `programs/` enthält den eigentlichen Programmablauf wie er im Programmeditor erstellt wurde
- `scenes/` enthält Beschreibungen der beweglichen Objekte in der Zelle, also Kisten, Hindernisse und ROIs
- `urdf/` enthält die Beschreibung der unveränderlichen Bestandteile der Zelle, also Roboter, Greifer, Schutzzaun, ...

- `package.xml` definiert den Namen der Zelle, der Einfachheit halber sollte er mit dem Ordnername übereinstimmen

Diese Struktur sollte eingehalten werden, um die Lauffähigkeit Mikados zu garantieren.

7.6 Mikado Select / Control Montageanleitung

7.6.1 Mikado Controller Hardware

Der Mikado Control Controller ist in einer kompakten und in einer größeren leistungsstärkeren 19-Zoll-Variante erhältlich. Beide Controller sollten in einer geschützten Umgebung z.B. in einem Schaltschrank montiert werden.

19-Zoll-Variante

Die leistungsstarke Variante des Mikado Controllers wird als 19-Zoll Einschub mit 4 HE (ca. 18 cm Höhe) geliefert. Auf der Vorderseite finden sich hinter einer verschließbaren Klappe der Anschalter sowie zwei USB Ports. Alle weiteren Anschlüsse befinden sich auf der Rückseite. Die Stromversorgung erfolgt auch hier über 110-240 V Kaltgerätekabel. Maus und Tastatur können über USB angeschlossen werden. Für Bildschirme stehen 3x DisplayPort, sowie je ein HDMI und DVI Anschlüsse bereit. Kameras und Roboter können über vier Gigabit-Netzwerkports einer Steckkarte angeschlossen werden. Für Remotezugriff oder Integration ins Gesamtsystem steht ein weiterer Onboard-Gigabit-Port zur Verfügung. Für die Kommunikation mit der SPS etc. kann auch hier die Hilscher cifX-Karte verwendet werden.

Kompakt-Variante

Die Ausmaße des Controller sind ca 250 x 160 x 160 mm, nach vorne sollten aber noch ca 5 cm Platz für die Kabel verbleiben. Eine Befestigung ist über mitgelieferte Bleche mit dem Lochraster 270 x 135 mm möglich. Die Stromversorgung erfolgt über ein 110-240 V Kaltgerätekabel, Tastatur, Maus und Bildschirme (DVI/HDMI) können über die üblichen Anschlüsse angeschlossen werden. Für die Verbindung zur Kamera und Roboter stehen zwei Gigabit-Ethernet Anschlüsse zur Verfügung. Bei der Steckkarte handelt es um eine Hilscher cifX-Karte mit der z.B. eine ProfiNet-Integration erfolgen kann. Alternativ kann dort auch eine 4-fach Netzwerkkarte verbaut werden.

7.6.2 Anschluss der Ensenso Kameras

Hier kann nur eine kurze Zusammenfassung gegeben werden, weitere Details finden Sie im Ensenso Manual.

N-Serie

Die kompakten Kameras der N-Serie können sowohl über PoE (Power over Ethernet) als auch über eine separate Stromversorgung (24 V) angeschlossen werden. Die Datenübertragung erfolgt in beiden Fällen über Gigabit Ethernet. Dafür ist ein ausreichend geschirmtes Kabel notwendig, mindestens Cat 5e. In vielen Robotern ist leider nur ein 100 MBit Kabel ohne ausreichende Schirmung verlegt. Mit Übertragungszeiten im Bereich von Sekunden ist damit dann keine minimale Taktzeit mehr zu erreichen.

Die Befestigung sollte mit drei M4 Schrauben in einer dreieckigen Anordnung erfolgen um eine Verspannung des Kamerakörpers zu vermeiden. Das Bohrbild und die Dimensionen der Kamera finden Sie hier im Ensenso Manual.

X-Serie

Bei der X-Serie erfolgt die Stromversorgung immer über ein externes 24 V Netzteil. Eine Stromversorgung über PoE ist hier nicht möglich. Da der Projektor bei der Aufnahme eine relativ hohe Leistung verbraucht, sollte für eventuelle Verlängerungen ein Kabel mit größerem Leitungsdurchmesser verwendet werden. Die Monokameras werden am Projektor angeschlossen und von diesem mit 12 V versorgt.

Zur Datenübertragung gibt es bei der X-Serie zwei Möglichkeiten:

- Es können die beiden Kameras mit dem integrierten Switch des Projektors verbunden werden. Der Switch wiederum wird direkt mit dem Mikado-Controller verbunden.
- Alternativ kann nur eine der beiden Mono-Kameras mit dem Switch verbunden werden. Die andere Kamera und der Switch werden dann direkt mit dem Mikado-Controller verbunden.

Die zweite Variante verdoppelt die Datenübertragungsrate und die Aufnahmezeit kann entsprechend verringert werden. Der Projektor wird mit vier M4 Schrauben befestigt, das Bohrbild und die Dimensionen finden Sie hier im Ensenso Manual.

7.6.3 Befestigung der Kalibrierplatte

Für die Hand-Auge-Kalibrierung (HAK) muss im Falle einer stationären Kamera ein Ensenso-Kalibrierpattern am Flansch oder Greifer befestigt werden. Bei einer beweglichen Kamera kann dieses hingegen einfach in die Zelle gelegt werden.

Stationäre Kamera

Grundsätzlich sollte die Stellung und Achswinkel des Roboters während der Kalibrierung möglichst nah an einer typischen Greifposition sein. Wie in der unteren Abbildung zu sehen sollte dabei Achse 6, also auch der Flansch, in negative Z-Richtung und die Kalibrierplatte (Pattern) in Richtung Kamera ausgerichtet sein. Weiterhin sollte die Befestigung der Kalibrierplatte hinreichend stabil sein und sich nicht bewegen, die genaue Position ist jedoch irrelevant. Der Ausleger für die Kamera muss dabei so lang sein, sodass die Kamera eine gute Sicht auf das gesamte Pattern oder zumindest auf den Großteil davon hat. Zu guter Letzt ist ein unkomplizierter Umbau hilfreich, damit ein erneutes Kalibrieren jederzeit einfach und schnell möglich ist.

Der einfachste Aufbau ist eine längliche Platte mit Bohrlöchern für den Roboterflansch und einer aufgeklebten oder geklemmten Kalibrierplatte neben dem Flansch. Nachteil ist hier die notwendige Demontage des Greifers. In das Pattern darf nicht gebohrt werden, da dann die Gefahr besteht, dass es sich verzieht. Das Pattern ist ein Messinstrument und sollte auch nach dem Gebrauch vor Kratzern geschützt werden.

Wenn ein stabiler Formschluss des Greifers gegeben ist, kann auch ein Teil mit angeklebtem Pattern gegriffen werden. Auch das seitliche Befestigen eines Hutschienenabschnitts am Greifer und das Ankleben eines entsprechenden Gegenstücks am Pattern haben sich bewährt.

Hinweis: Bei den Keramikpattern eine Hintersicherung vorsehen, falls der Greifer unbeabsichtigt geöffnet wird.



Abb. 16: Beispiel für die Befestigung einer Kalibrierplatte am Roboterflansch bei stationärer Kamera

Bewegliche Kamera

Bei einer mobilen am Roboterflansch bzw. am Greifer befestigten Kamera kann das Pattern frei im Arbeitsraum platziert werden. Günstig ist eine mittige Platzierung im späteren Aufnahmebereich, z.B. auf oder in der Teilekiste. Das Pattern muss nicht fixiert werden, es muss lediglich während der Roboterbewegungen im Laufe des Kalibriervorgangs rutschsicher ruhen.

7.7 Greifer-Integration

In diesem Kapitel der Greifer-Integration wird aufgezeigt, welche Möglichkeiten Mikado Control zur Einbindung Ihres Greifer bietet. Erläutert werden unterstützte Greifertypen und wie Sie das CAD-Modell des Greifers einbinden und konfigurieren können.

7.7.1 Greifertypen

Mikado Control unterstützt mehrere Greifertypen wie z.B. Parallel-, Vakuum-, Magnet oder Elektro-Greifer. Eine Auflistung und Beschreibung der Greiferkategorien ist in Sektion *gripper.yaml* aufgeführt.

7.7.2 Greifer einbinden

Sobald Sie einen Greifer am Roboter angebracht haben, können Sie diesen in das Robotermodell aufnehmen. Sie benötigen hierzu ein passendes CAD-Modell des Greifers.

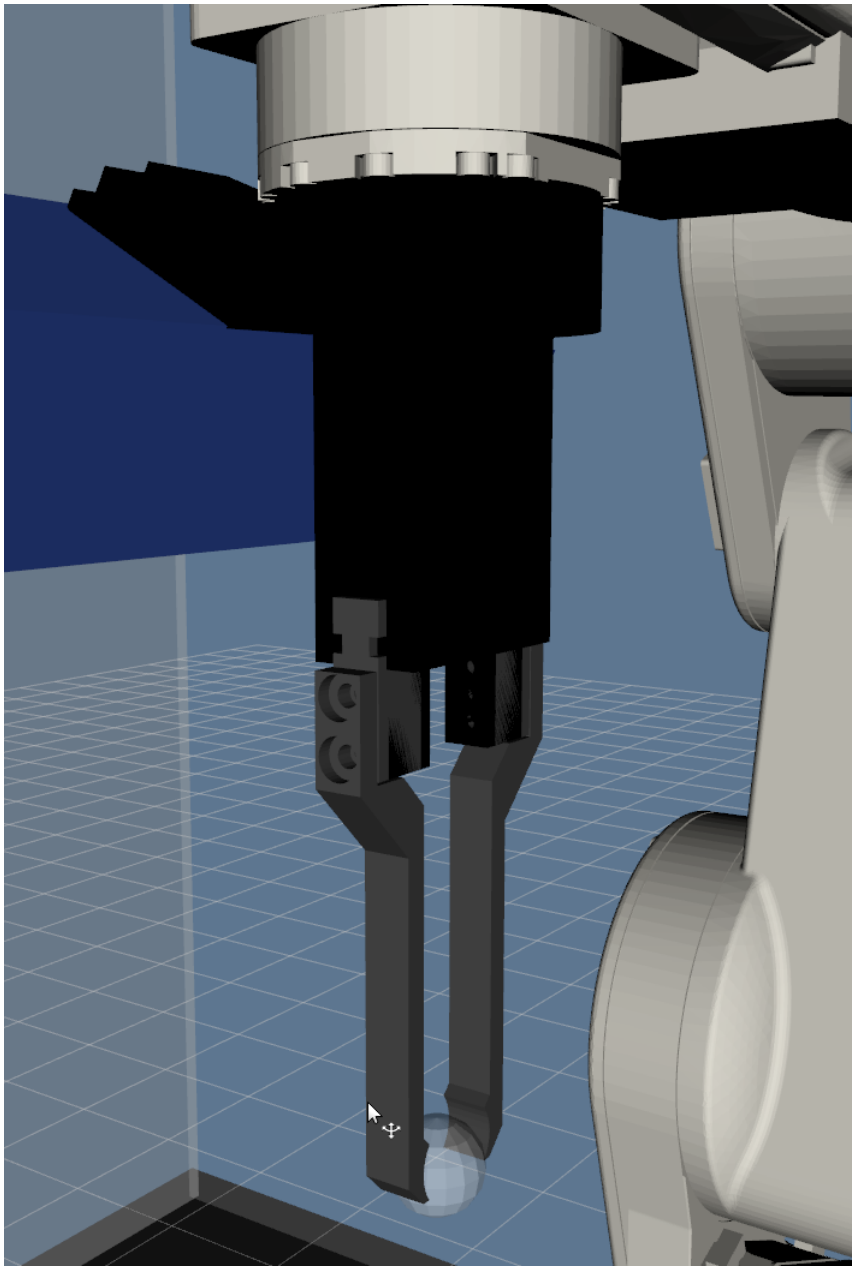


Abb. 17: Integrierter Greifer in der Mikado-Visualisierung (Beispiel)

Greifer modifizieren

Möchten Sie lediglich kleinere Veränderungen am Greifermodell eines bereits integrierten Systems vornehmen, können Sie einfach die eingebundene STL-Datei mit einem CAD-Tool Ihrer Wahl modifizieren. Über den *Mikado Assistant* haben Sie Zugriff auf die URDF-Datei, die Ihren Greifer definiert. Dort können Sie einsehen, welche STL-Dateien verwendet werden und die Modelle anpassen.

Hinweis: Modifizieren Sie vorhandene CAD-Modelle anstatt komplett neue zu erstellen, so müssen Sie sich nicht mehr um Skalierung oder Positionierung des Greifers kümmern.

CAD vorbereiten

Damit Mikado Control Greif- und Bahnplanungen ausführen kann, wird ein Modell des Greifers benötigt. Daher benötigen Sie ein CAD-Modell des Greifers im Dateiformat STL. Achten Sie darauf, dass Sie die Anzahl der Dreiecke auf das nötige Minimum reduzieren. Eine extrem hohe Anzahl an Dreiecken kann die Kollisionsprüfungen stark verzögern und führt somit zu langen Ausführungszeiten. Reduzieren Sie das Modell soweit, dass alle Strukturen innerhalb des Modells liegen, aber trotzdem präzise genug für die Greifvorgänge sind. Meist ist es sinnvoll, den Basiskörper stärker zu approximieren als zum Beispiel die Greifbacken.

Verwenden Sie einen Greifertyp mit beweglichen Elementen (beispielsweise ein Parallelgreifer), so benötigen Sie die Einzelteile als STL-Dateien. Ein Parallelgreifer besteht also aus drei STL-Dateien: Basiseinheit, linker Greiffinger (oder -backe) und rechter Greiffinger. Hat der Greifer keine beweglichen Elemente (Vakuum oder Magnetgreifer), genügt eine STL-Datei die den kompletten Greifer beinhaltet.

Speichern Sie nun die Dateien im *meshes/* Verzeichnis Ihres Zellen-Projektverzeichnisses ab.

Hinweis: Sind die Greiffinger symmetrisch, können Sie dasselbe Modell für beide Greiffinger verwenden. Sie unterscheiden sich dann lediglich in der Positionierung am Roboter.

Im obigen Beispielbild der Mikado-Visualisierung ist ein Parallelgreifer zu sehen der aus folgenden Komponenten besteht:

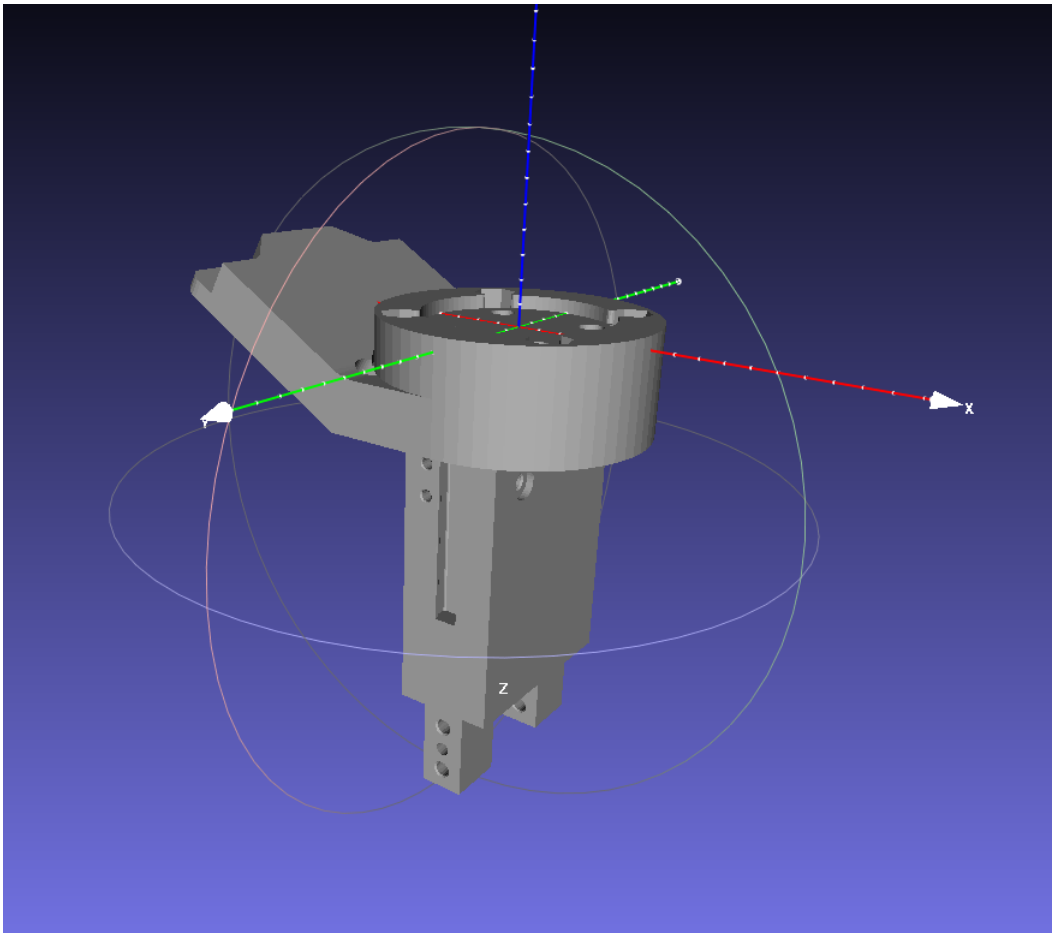


Abb. 18: Basisgeometrie des Beispielgreifers (oben schwarz dargestellt)

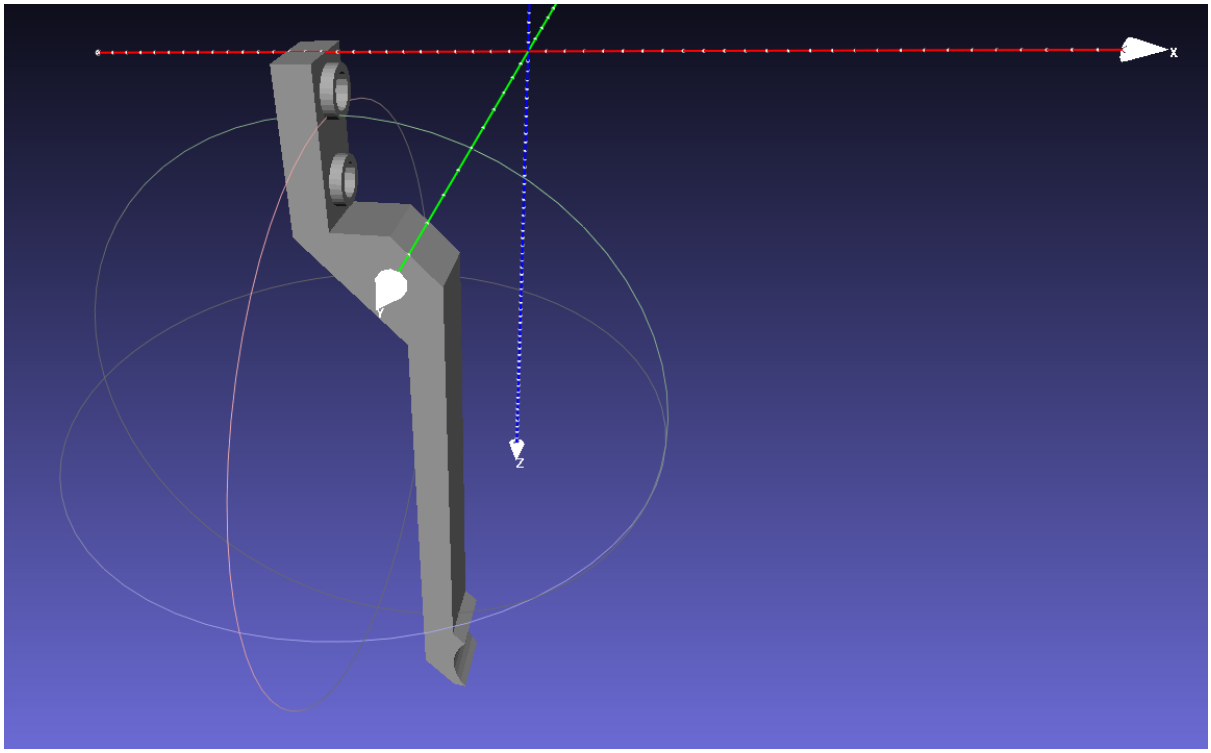


Abb. 19: Fingergeometrie des Beispielgreifers (oben grau dargestellt): Koordinaten-Ursprung liegt auf der z-Achse des Roboter-Werkzeugflanschs. Dies erleichtert die Positionierung.

CAD einbinden

Nachdem alle benötigten CAD-Dateien vorliegen, können Sie diese mithilfe des Mikado-Assistant in Ihre Zelle einbinden.

Sobald Sie Ihre Zelle über den Mikado-Assistant erstellt haben, wird bereits eine Vorlage eines Greifers eingebunden. Um Ihre eigenes Greifermodell zu verwenden, können Sie nun die Dateipfade anpassen, die in der *URDF-Datei* des Greifermodells definiert sind:

```
<mesh filename="package://mein_projekt/meshes/greifer_dateiname.stl" />
```

Hinweis: Außer STL-Dateien können Sie auch Greifermodelle im DAE-Format einbinden.

Greifermodell positionieren

Sie werden feststellen, dass der Greifer oft eine falsche Skalierung aufweist oder nicht am gewünschten Ort positioniert ist.

- **Skalierung anpassen**

  Eigenschaft ‚scale' setzen, wenn die STL Datei nicht die Einheit Meter verwendet. In diesem Beispiel wurde im STL die Einheit mm verwendet, daher der Faktor 0.001:

  ```
  <mesh filename="package://mein_projekt/meshes/greifer_in_mm.stl" scale="0.001" />
  ```

- **Greiferposition anpassen**

  Genau wie bei der Zellgeometrie werden einzelne Körper/Teile mithilfe von Gelenken (Joints) miteinander verbunden und positioniert. Die Position des Greifers und der daran hängenden Greiferteile lassen sich daher auf dieselbe Weise verändern. Weitere Details sind ausführlich in Sektion *URDF* beschrieben.

Vergessen Sie dabei nicht, auch den Joint <joint name="gripper_to_tcp" type="fixed"> anzupassen. Dieser Joint definiert die Position des Greifpunktes am Werkzeug und wird in Mikado als transparente Kugel visualisiert.

7.7.3 Mikado-Konfiguration

Damit Ihr Greifer korrekt angesteuert wird, definieren Sie Eigenschaften eines Greifers in der Konfigurationsdatei, die in Sektion *gripper.yaml* erläutert wird.

7.7.4 Greifer Anschlussvarianten

Einen Großteil der Greifervarianten lässt sich über Roboter-IOs oder einen Feldbus einbinden. Welche IO-Belegung Sie für die Öffnen- und Schließen-Aktionen verwenden, tragen Sie ebenfalls in die Mikado-Konfiguration ein (s. *gripper.yaml*).

Verwenden Sie einen Greifer der über TCP/IP angesteuert wird, müssen Sie dem Greifer über die Hersteller-Software eine IP zuweisen, die im Netz des Mikado-Controllers liegt. Die IP tragen Sie nun in der Datei *main.launch* des *launch/*-Verzeichnisses ein. Innerhalb der Datei finden Sie folgende Zeile, in der Sie die neue IP-Adresse eintragen müssen.

```
<include file="...">
    <arg name="ip" value="10.0.1.13" />
```

7.7.5 TCP kalibrieren

Bei der TCP-Kalibrierung wird die Transformation der Greifer-Baugruppe zum Roboterflansch bestimmt.

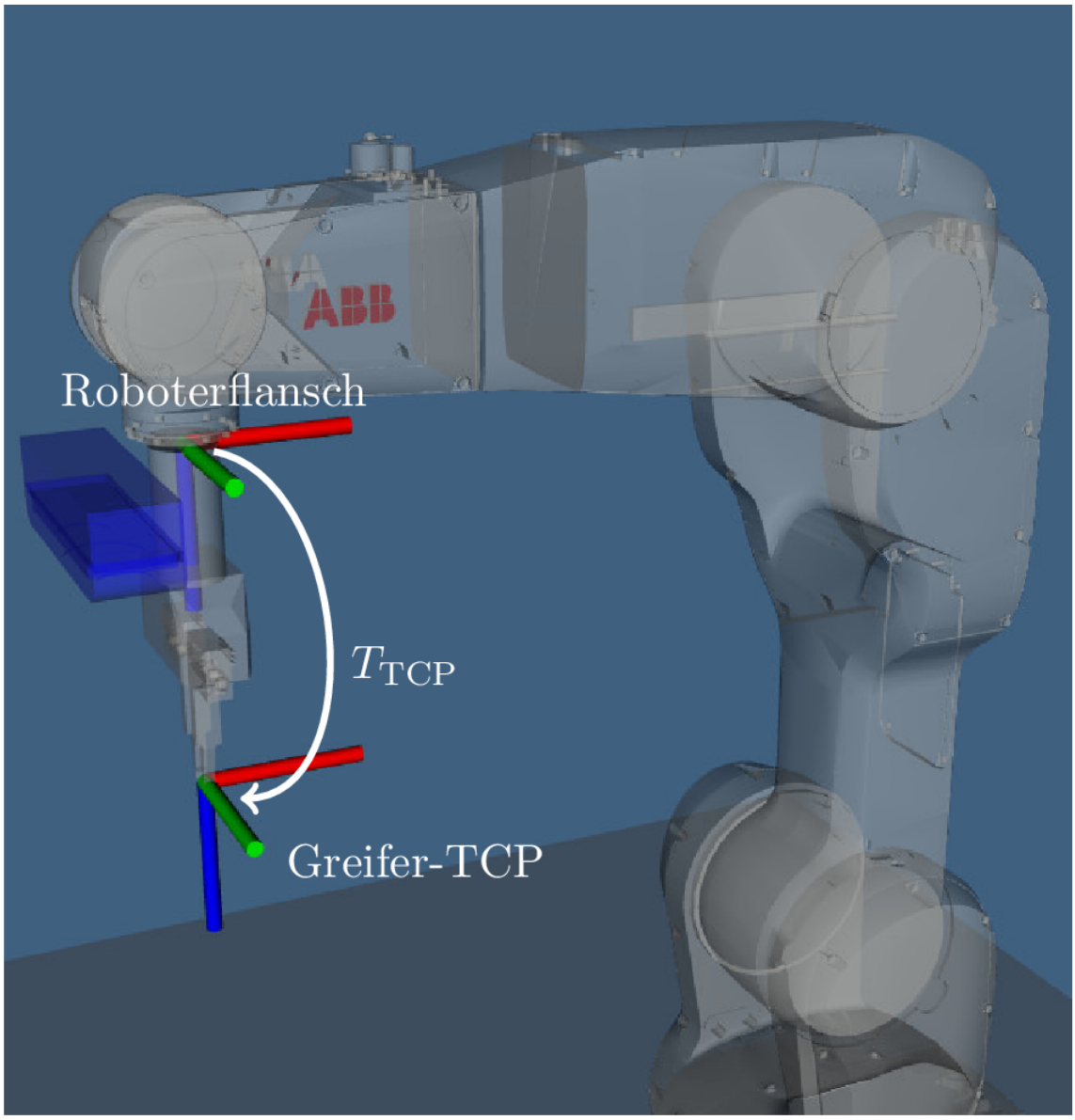


Abb. 20: Durch die TCP-Kalibrierung bestimmte Transformation.

Eine Überprüfung der TCP Offsets nachdem ein Greifer eingerichtet oder neu montiert wurde wird unbedingt empfohlen. Jede Abweichung der Montage zu den eingetragenen Werten in der Greiferposition ist beim Greifen oder Anfahren von Teilen zu erkennen.

Ein einfacher Mikado-Ablauf hilft Ihnen dabei visuell zu prüfen, ob die eingetragene Greiferposition korrekt ist.

- Starten Sie die Mikado Control-Oberfläche nachdem Sie Ihren Greifer eingerichtet haben
- Legen Sie die Kalibrierplatte vor den Roboter ins Sichtfeld der Kamera bei der Aufnahmepose
- Laden Sie das folgende Mikado-Programm aus *programs/examples/tcp_calibration_example.program*, oder erstellen Sie es manuell
  - Bewege Roboter zu einer geeigneten Aufnahmepose
  - Patternpose bestimmen
  - Patternpose umdrehen (180° rollen)
  - Patternpose in z-Richtung verschieben (z.B. um +10 mm)

  - Bewege TCP des Roboters zur ermittelten Pose

- Nun sollte der Roboter mit dem TCP des Greifers zentral über dem mittleren Punkt der Kalibrierplatte stehen und den festgelegten Abstand zur Oberfläche der Platte aufweisen (z.B. 10 mm)
- Messen Sie die Abweichungen von der erwarteten zentralen Position und wenden Sie die Abweichungen im Mikado Assistant Greifer-Editor auf den Offset von *gripper_tcp* an
- Der Offset von *gripper_tcp_uncalibrated* zu *gripper_tcp* gibt den Versatz zwischen der im CAD erwarteten und der tatsächlich gemessenen TCP-Position an
- Nach dem Speichern der Änderungen im Assistant muss die Mikado-GUI neu gestartet werden um die neuen Werte einzulesen
- Wiederholen Sie den Vorgang um sicherzustellen, dass die neuen Offset Werte zu einer Verbesserung der Positioniergenauigkeit geführt haben

7.8 Hardware einrichten

In diesem Bereich sind Anleitungen aufgeführt, um spezifische Hardware mit Mikado Control zu verbinden.

7.8.1 Übersicht Anschlüsse/Verkabelung

Der Mikado Controller muss mit der Kamera verbunden sein, doch je nach Kamera-Modell gibt es hier unterschiedliche Dinge zu beachten. Die kompakten Kameras der N-Serie können alternativ über PoE (Power over Ethernet) oder über eine separate Stromversorgung (24V) angeschlossen werden. Die Datenübertragung erfolgt über Gigabit Ethernet. Dafür ist ein ausreichend geschirmtes Kabel notwendig, mindestens Cat 5e.

Im Folgenden ist die Verkabelung einer Zelle dargestellt, welche eine X36 Kamera beinhaltet. Hierbei ist insbesondere die Minimalspannung von > 20 V bei einem Strom von 5 A an der Kamera zu beachten. Bei der Berechnung des Spannungsabfalls ist vor allem die Kabellänge zu beachten. Hierbei gibt es einen Hin- und Rückweg, da es einen Plus- und Minuspol gibt. Der Spannungsabfall ist außerdem vom Strom, dem Querschnitt des Kabels sowie dem spezifischen elektrischen Widerstand von Kupfer abhängig. Die Netzwerkkabel oder den Lichtwellenleiter vom Mikado Controller zur Kamera oder vom Mikado Controller zum Roboter sind nicht Teil des Lieferumfangs, da die Länge vom Anlagenbau abhängig ist.

$$\Delta V = l_{Kabel} * I * \rho_{Kupfer} * \frac{1}{A_{Kabel}}$$

$$\Delta V = (l_{Hin} + l_{Rück}) * 5A * \frac{0{,}017\Omega * mm^2}{m} * \frac{1}{0{,}5mm^2}$$

$$\Delta V = (10m + 10m) * 5A * \frac{0{,}017\Omega * mm^2}{m} * \frac{1}{0{,}5mm^2}$$

$$\Delta V = 3{,}4\,V$$

Abb. 21: Beispiel für Berechnung des Spannungsabfalls in Abhängigkeit des Querschnitts und der Kabellänge

Da Kabel mit einem Querschnitt > 0,5 mm^2 nur unter großem Aufwand in einem M8-Stecker unterzubringen sind, bietet sich die Wahl eines M12-Kabels an, an dessen Ende ein M12- zu M8-Adapter angeschlossen ist.

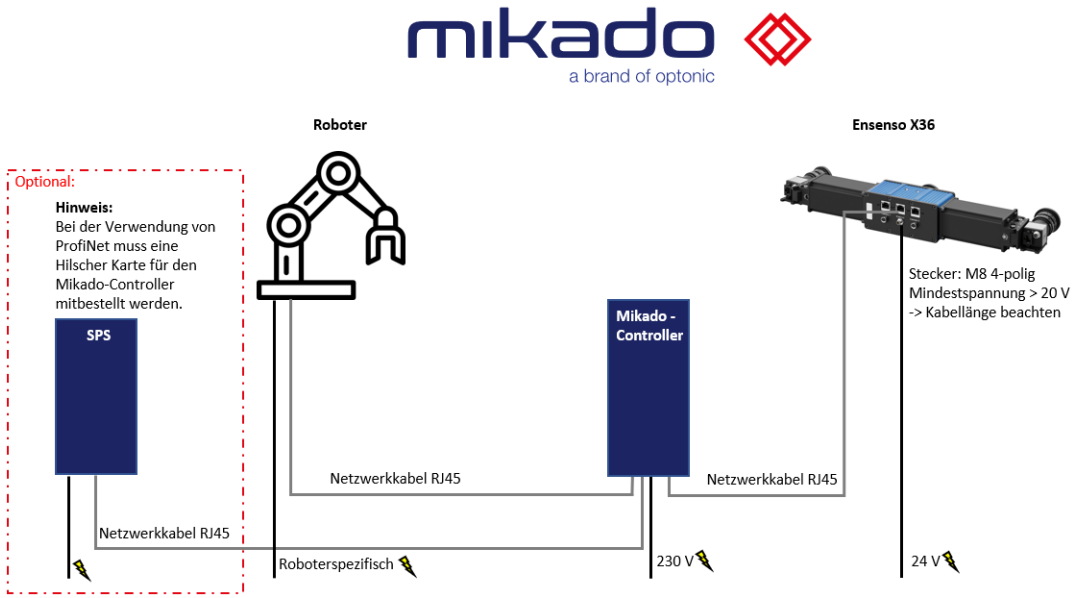


Abb. 22: Anschlussübersicht bei Verwendung einer Ensenso X36

Während in einer Ensenso X36 ein Netzwerk-Switch integriert ist, gibt es beim Modell XQ einen externen Switch, der jedoch direkt mit der XQ-Kamera verdrahtet ausgeliefert wird. Es wird hierbei kein M8-Stecker benötigt, sondern drei Kabel. Am Switch befinden sich Steckverbindungen, in die die Kabel und deren Aderendhülsen gesteckt werden. Die Übertragung der Bilder vom Switch zum Mikado Controller erfolgt über einen Lichtwellenleiter, um hohe Übertragungsgeschwindigkeiten sicherzustellen. Die dafür benötigte 10-GBit Netzwerkkarte und den Adapter für den Mikado Controller sind Teil des Lieferumfangs.

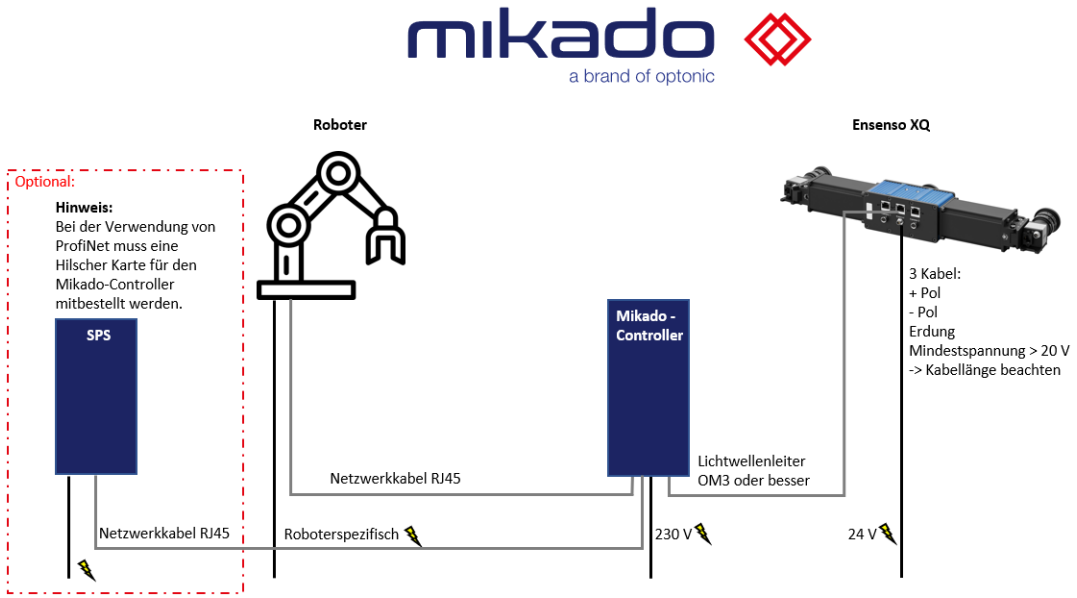


Abb. 23: Anschlussübersicht bei Verwendung einer Ensenso XQ (Foto stellt eine X36 dar, keine XQ)

Falls Mikado via Feldbus wie z.B. ProfiNet mit Ihrer SPS kommunizieren soll, sprechen Sie uns bei der Bestellung darauf an, sodass eine entsprechende Schnittstellen-Karte miteingeplant werden kann. Sollten Sie beispielsweise via OPC UA kommunizieren wollen, ist in der Regel keine zusätzliche Hardware nötig.

7.8.2 Netzwerkadapter

Der Mikadocontroller muss mit Komponenten wie der Ensenso Kamera und Robotern über eine TCP/IP Schnittstelle verbunden sein. Es wird nachfolgend erklärt, wie man diese korrekt einrichtet.

Feste IP

In dieser Konfiguration wird jede Komponente an einen dedizierten Netzwerkport des Mikadocontrollers angeschlossen. Dem Controller und den Komponenten wird dann jeweils eine feste IP zugeordnet. Auf der Mikadocontroller Seite funktioniert dies, indem man zunächst in die Netzwerkeinstellungen wechselt, zum Beispiel per Klick auf das Netzwerksymbol unten rechts in der Taskleiste und einem weiteren Klick auf das Einstellungssymbol im Dialog:



Abb. 24: Screenshot Netzwerkeinstellungen öffnen

Zum Einstellen einer festen IP folgt man den fünf Schritten:

1. Auswahl des entsprechenden Adapters
2. Auswahl des IP v4 Reiters
3. Auswahl Methode: Manuell
4. In der Liste per + *Hinzufügen* einen neuen Eintrag produzieren und eine IP und Subnetzmaske eingeben.
5. *Anwenden* und *OK*

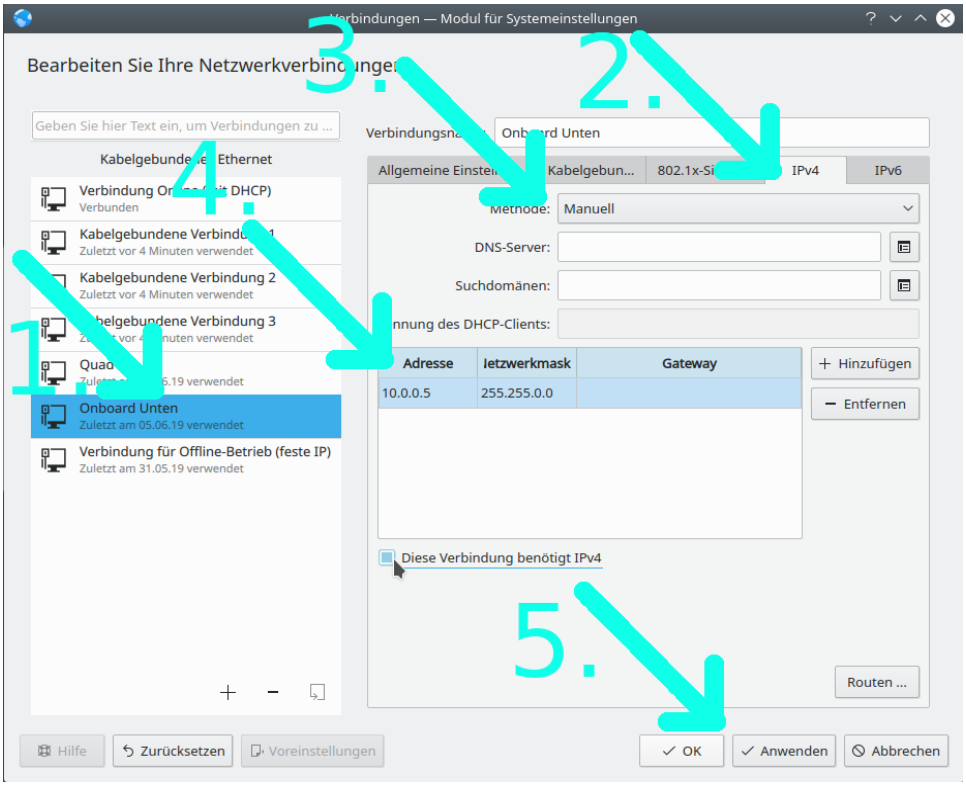


Abb. 25: Screenshot Netzwerkeinstellungen feste IP konfigurieren

Warnung: Die Netzwerkadapter des Mikado Controllers sollten in unterschiedlichen Adressräumen liegen, da sonst der Paketversand uneindeutig ist. Konfigurieren Sie zum Beispiel einen Port auf die 192.168.5.1 und einen weiteren auf die 192.168.2.1.

7.8.3 Verbindungstest

Im Fall von Verbindungsproblemen ist es ratsam, zunächst zu überprüfen, ob das jeweilige Netzwerkgerät vom Mikado Controller aus erreicht werden kann. Öffnen Sie hierfür eine Konsole auf dem Mikado Controller (STRG + ALT + T) und geben Sie den folgenden Befehl in die Konsole ein (ersetzen Sie die IP durch die IP des verbundenen Netzwerkgeräts):

```
ping 192.168.1.42
```

Das Pingen kann per *Strg + C* abgebrochen werden. Eine positive Antwort würde wie folgt aussehen:

```
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
```

Wenn die Pings jeweils innerhalb weniger Millisekunden erfolgreich sind, ist das Gerät mit der angegebenen IP-Adresse ansprechbar.

Wenn die eingegebene IP-Adresse nicht erreichbar ist, gibt es mehrere mögliche Ursachen:

- Das angesprochene Gerät ist nicht betriebsbereit.
- Es liegt keine Kabelverbindung zwischen den Geräten vor. Eventuell sitzt ein Netzwerkstecker nicht vollständig in der RJ-45 Buchse.
- Die IP-Adresse des Mikado Controllers und des verbundenen Netzwerkgeräts liegen nicht im selben Subnetz-Adressraum.

7.8.4 Ensenso-Kameras einrichten

Die Einrichtung einer Kamera teilt sich grundsätzlich in drei Schritte auf.

1. Einrichtung der Netzwerkschnittstelle und damit Kommunikation zur Kamera
2. Optimierung der Kamerakonfiguration für das Szenario, insbesondere Belichtungsdauer, FlexView und Parameter fürs Stereomatching
3. Hand-Auge Kalibrierung für die Mikadozelle

Kamera Netzwerkadapter einrichten

Im ersten Schritt sollte eine Verbindung zur Kamera hergestellt werden können. Dazu muss dieser eine IP im Subnetzraum des Mikado Netzwerkadapters zugewiesen werden. Nachdem der Netzwerkadapter auf die gewünschte IP eingestellt wurde, kann dies per Network Wizard in NxView geschehen.

Falls keine Kameras in NxView angezeigt werden, bitte folgende Punkte prüfen:

- LED an der Ensenso Kamera blinken und zeigen somit an, dass die Kamera bestromt wird
- Ensenso N-Serie: Über die Konsole `sudo idscameramanager` starten. Dies ist ein eigenständiges Tool um den Kameratreiber zu konfigurieren. Sieht man die Kamera auch nach wenigen Sekunden nicht, so muss unter „Daemon Control“ für den verwendeten Netzwerkadapter der „ETH-ConfigurationMode“ aktiviert werden. Die so gefundene Kamera kann per „Manual ETH configuration“ mit einer festen IP im gleichen Adressraum versehen werden.
- Verwenden Sie die aktuelle Version des Ensenso Manual, um weitere Informationen über die korrekte Konfiguration und Anschlüsse Ihres Systems nachzuschlagen.

Werden in Ihrer Anwendung mehrere Kameras verwendet, müssen Sie beide Geräte an separate Netzwerkadapter Ihres Mikado-Controllers anschließen. Beide Adapter müssen in einem eigenen Subnetz liegen.

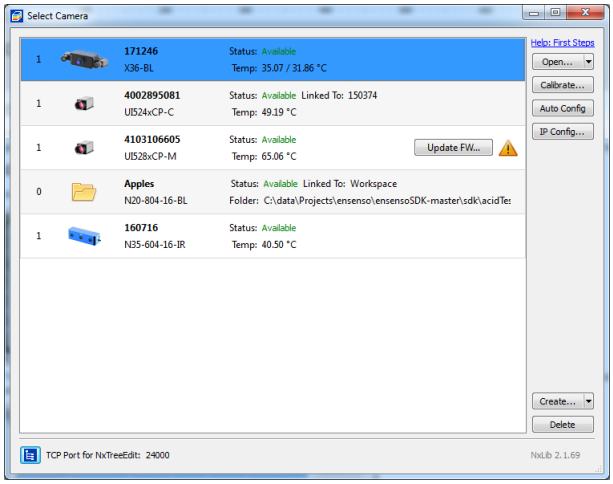
Hinweis: Ist die feste IP eines Netzwerkadapters zum Beispiel die 192.168.2.42 (mit Subnetzmaske 255.255.255.0), so bedeutet gleicher Adressraum nur einen erlaubten Unterschied im letzten Zahlenblock. Eine geeignete IP für die Kamera wäre dann zum Beispiel die 192.168.2.3.

Warnung: Keine zwei Netzwerkadapter sollten im gleichen Adressraum liegen, da dies den Paketversand uneindeutig macht. Konfigurieren Sie zum Beispiel einen Port auf die 192.168.5.1 und einen weiteren auf die 192.168.2.1.

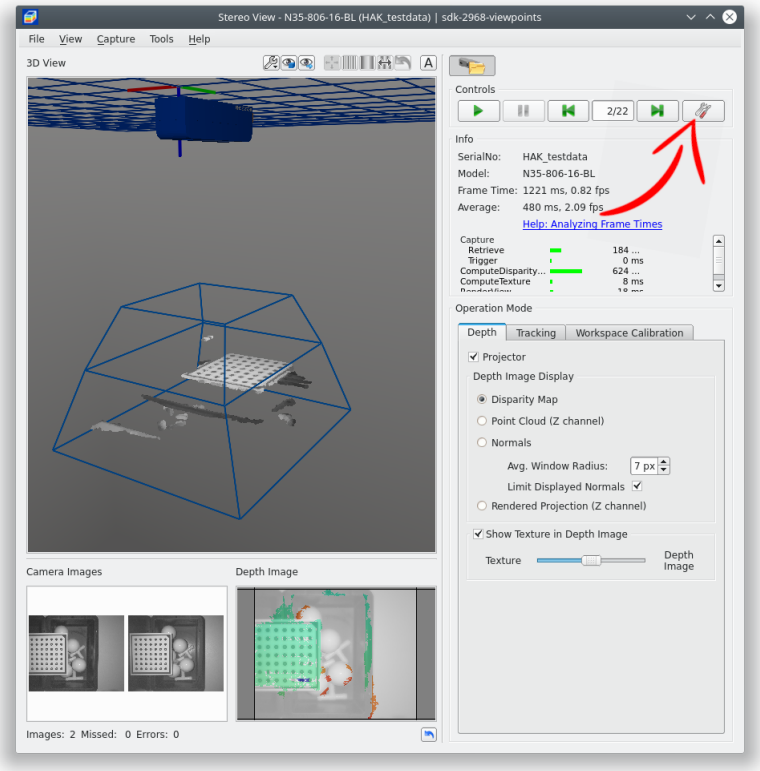
Kamerakonfiguration

Unter der Kamerakonfiguration wird das interne Parameterset des genutzten 3D-Sensors verstanden. Um die Kamera zu konfigurieren wird die Software NxView verwendet. Zum Erstellen einer Kamerakonfiguration sind folgende Schritte notwendig:

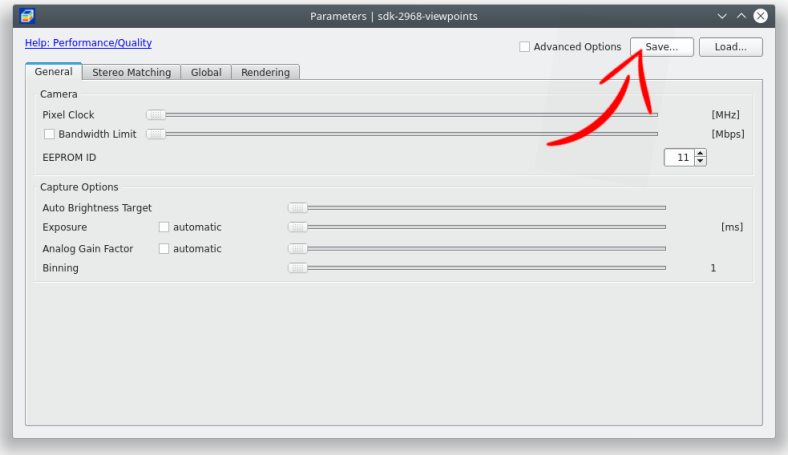
1. Öffnen Sie NxView und wählen Sie die gewünschte Kamera aus.



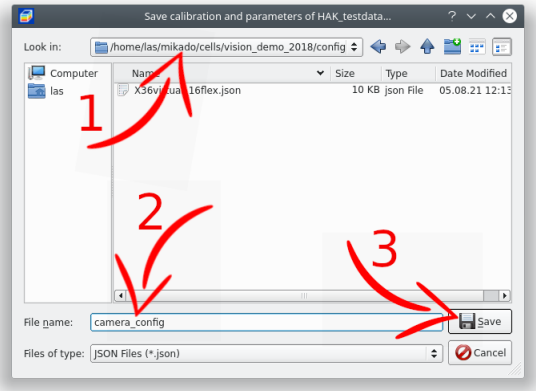
2. Sobald die Kamera geöffnet ist, klicken Sie auf das Parameter-Icon. Es öffnet sich ein neues Fenster.



3. Im Parameterfenster lassen sich alle Kamera-Parameter einstellen. Sobald Sie ein Parameterset gefunden haben, welches zur geforderten Datenqualität führt, speichern Sie das Parameterset. Hierfür klicken Sie auf den Save-Button. Ein geeignet Name kann die Seriennummer der Kamera sein, also z.B. `229020.ensparam`.



4. Es öffnet sich ein Dialogfenster, indem Sie den Dateipfad Ihrer Mikado-Zelle auswählen (1), einen sprechenden Namen für die Kamerakonfigurationsdatei wählen (2) und schließlich auf Speichern klicken (3).



Öffnen Sie nun die Zelle im Mikado-Assistant und wählen Sie die neu erstelle Konfiguration im Menüpunkt ‚Parameter → Kameras → Konfig Datei‘ aus.

Weitere, detaillierte Informationen finden Sie im NxView Manual.

Bemerkung: Bei einer Ensenso X-Serie ist eventuell noch eine intrinsische Kalibrierung notwendig.

Hand-Auge Kalibrierung

Ein Start der Hand-Auge-Kalibrierung aus Bedienersicht ist im Kapitel *Hand-Auge-Kalibrierung* näher beschrieben. Damit diese funktioniert müssen noch die Bewegungsparameter während der Kalibrierung festgelegt werden. Diese sind im Kapitel *calibration.yaml* aufgeführt.

Filekamera

Filekameras bieten sowohl gute Optimierungsmöglichkeiten für neu eingerichtete Zellen, als auch Fehler-Report-Möglichkeiten. Ein Sammeln der nötigen Rohdaten ist sowohl aus Mikado heraus, als auch mittels NxView möglich. Es wird vor allem für Setups mit beweglicher Kamera (Kameras-am-Roboterarm) empfohlen die Daten aus Mikado heraus zu sammeln, entweder mit dem entsprechenden Task im Programmablauf (siehe Task ‚*Bild speichern*') oder auf der *Erkennungsseite* per Letzte-Aufnahme-Speichern Knopf. Mikado speichert bei pro Aufnahme zusätzlich eine Link-Datei, die bei beweglichen Kameras die derzeitige Roboterposition enthält. Unkalibrierte Filekameras stellen die Punktwolke in der Visualisierung so dar, als würde die Kamera im Koordinatenursprung mit Blick nach oben positioniert sein. Dennoch kann auch mit unkalibrierter Kamera eine Objekterkennung, zum Beispiel im Rahmen einer Voruntersuchung, problemlos durchgeführt werden.

Für jede Aufnahme werden folgende Rohdaten geschrieben:

Bilder	*.png	Anzahl FlexView x (links,rechts)
Kameraparameter	*_calib.json	1x
Roboterposition	Links*.txt	1x

Eine Filekamera aus aufgezeichneten Daten zu erstellen kann über NxView geschehen. Wählen Sie in der Oberfläche ‚Create' → ‚File Camera' und den Pfad des Ordners der Rohdaten. Danach lässt sich noch die Seriennummer der Kamera anpassen, die standardmäßig auf ‚File[Seriennummer]' gesetzt wird.

Nun muss noch die Seriennummer der Kamera in Mikado angepasst werden, siehe dazu Kapitel *camera_X.yaml*. Achten Sie in dieser Konfigurationsdatei ebenfalls auf den Parameter `fixed_camera`, da dieser auch bei Filekameras angibt, ob die Kamera beweglich oder fix montiert war. Mikado verwendet dann die Aufnahmen der Filekamera in der Reihenfolge, in der sie aufgenommen wurden. Ist die letzte Aufnahme erreicht, beginnt die Filekamera wieder von vorne. Es werden die Kameraparameter benutzt, die mit der Filekamera abgelegt sind, *nicht die beim Mikadostart oder im Programmablauf geladenen*.

Warnung: Filekameras die in einem *.zip liegen können zwar im NxView erstellt werden, werden aber von Mikado nicht unterstützt. Diese müssen entpackt und die Filekamera mit den entpackten Dateien erstellt werden.

Multi-Kamera Setup

Möchten Sie in Ihrer Anwendung eine weitere Kamera einbinden, sind für die Installation folgende Schritte durchzuführen:

1. Kamera an den Controller anschließen und Kommunikation einrichten wie in *Kamera Netzwerkadapter einrichten* beschrieben.
2. Zweite Kamerakonfiguration erstellen und ebenfalls ins Verzeichnis `config` ablegen.
3. Im Verzeichnis `config` die Datei `camera_0.yaml` kopieren, die Kopie umbenennen (z.B. in `camera_1.yaml`, `camera_2.yaml`, etc.) und öffnen. Dort nun die Eingaben anpassen (s. Sektion *camera_X.yaml*).
4. Im Verzeichnis `launch` die Datei `main.launch` öffnen und unter der Zeile `<!-- Cameras -->` folgenden Abschnitt einfügen. Achten Sie darauf, dass Sie der Kamera einen neuen eindeutigen Namen geben und die korrekte Konfigurationsdatei eintragen.

```
<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. Mikado starten und im *Systemstatus* prüfen, ob die Kamera korrekt geöffnet werden konnte.

Die Kamera ist nun eingerichtet. Sie können nun mit der Hand-Auge-Kalibrierung fortfahren (s. *Hand-Auge Kalibrierung*). Die Mikado-Oberfläche wird nun an entsprechenden Stellen ein Auswahldialog für die zu verwendende Kamera anzeigen. In Ihrem Mikado-Ablauf kann die aktive Kamera via dem Task ‚*Setze aktive Kamera*' angepasst werden. Die momentan aktive Kamera ist in der *Systemleiste* unten zu sehen.

7.8.5 Roboter-Controller einrichten

Die Einrichtung der Roboter-Controller wird im Kapitel *Roboter-Modelle* beschrieben.

7.9 Feldbus-Schnittstelle

Mikado kann über Feldbusse z.B. mit einer SPS verbunden werden.

7.9.1 Kurzbeschreibung der Schnittstelle

Über die Feldbus-Schnittstelle können die Kernfunktionen von Mikado aufgerufen werden. Dabei wird unterschieden zwischen allgemeinen Funktionen und Funktionen zur Steuerung eines geladenen Mikado-Programms.

Allgemeine Funktionen

- Durchführen der Hand-Auge-Kalibrierung.

Funktionen zur Steuerung eines Mikado-Programms

- Starten und Stoppen des Mikado-Programms bzw. einer einzelnen Sequenz innerhalb des Mikado-Programms.
- Die Synchronisation von Mikado-Variablen mit der SPS.

7.9.2 Genereller Ablauf der Kommunikation

Die Kommunikation wird realisiert mittels Kommandos (siehe *Kommandos*) die von SPS an Mikado gesendet werden. Der Status der Kommunikation wird mittels Status-Bits beschrieben (siehe *Status der Kommunikation*). Die folgende Ablaufbeschreibung gilt für alle Kommandos. Bedingung für die Kommunikation ist die vollständige Betriebsbereitschaft des Mikado-Systems. Das wird durch `OutReadyBit = 1` signalisiert und ist in der Mikado-Oberfläche anhand des **Systemstatus** abzulesen.

Tab. 6: Genereller Ablauf der Kommunikation

Zeit	**SPS**	**Mikado**
$< t_1$	Warte auf `OutReadyBit = 1` Warte auf `OutBusyBit = 0` Parameter setzen	Warten auf Kommando
t_1	`InCmd` setzen Warte auf `OutBusyBit = 1`	

t_2		`OutErrorCode = 0`
		`OutBusyBit = 1` (Feldbus-Adapter verarbeitet Befehl)
		Nur beim Start des Programms / einer Sequenz:
		`OutRunningBit = 1`

t_3	`InCmd` zurücksetzen auf `0` Warte auf `OutBusyBit = 0`	
t_4		`OutErrorCode` schreiben `OutBusyBit = 0` (Feldbus-Adapter hat Befehl verarbeitet)
t_5	Lesen von `OutErrorCode` ***Nur nach Start des Programms / einer Sequenz:*** Warte auf `OutRunningBit = 0`	
t_6	***Nur nach Start des Programms / einer Sequenz:*** Lesen von `OutErrorCode`	***Nur wenn zuvor ein Programms / eine Sequenz gestartet wurde:*** `OutRunningBit = 0` (MIK-Programm endet)
ab hier	ggf. Lesen von Variablen	

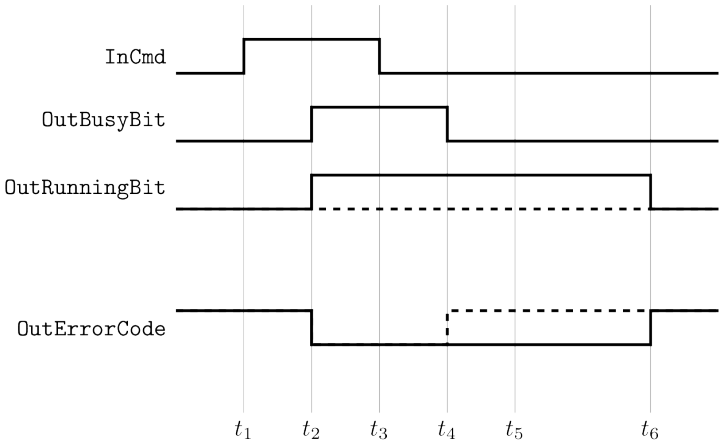


Abb. 26: Signalverlauf allgemein, alternative Verläufe gestrichelt

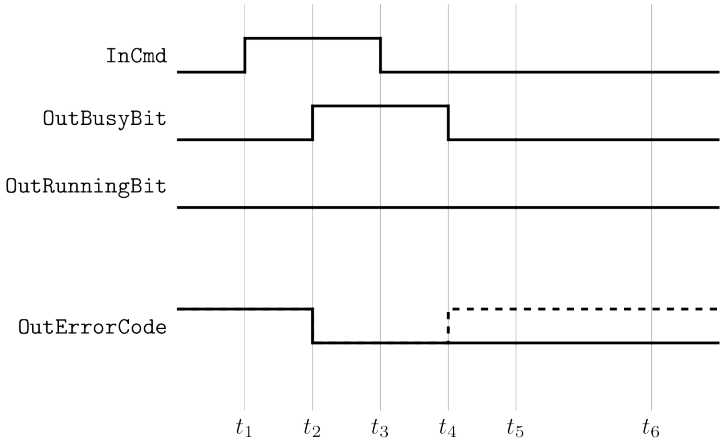


Abb. 27: Signalverlauf für Kommandos wie StartCalib, CaptureCalib, FinishCalib

- Wird ein Kommando gesetzt (`InCmd` $\neq$ `0`), antwortet Mikado mit `OutBusyBit = 1` und setzt zunächst den `OutErrorCode` zurück. Auf `OutBusyBit = 1` antwortet die SPS mit `InCmd = 0` und wartet dann auf `OutBusyBit = 0`.
- Die SPS wertet den `OutErrorCode` aus. Wenn dieser den Wert 0 hat, ist das Kommando erfolgreich ausgeführt worden.
- Nur bei Kommando `RunMikadoProg` und `StartSequence` wird `OutRunningBit = 1`. Das `OutRunningBit` signalisiert, dass Mikado das aktuelle Programm abarbeitet. Sobald `OutRunningBit = 0` wird, können der Fehlercode `OutErrorCode` und ggf. weitere Output-Variablen ausgelesen werden.

7.9.3 Status der Kommunikation

LifeBit

Um den Status der Kommunikation zwischen Mikado und SPS zu überprüfen wird ein LifeBit verwendet. Dabei liest die SPS den Wert von `OutLifeBit` und schreibt diesen unverändert auf `InLifeBit`. Der Feldbusadapter liest den Wert von `InLifeBit` und schreibt den negierten Wert auf `OutLifeBit`. Verändert sich der Wert von `OutLifeBit` nicht regelmäßig, gilt die Kommunikation als unterbrochen und ein entsprechender Fehler wird im Mikado-Status angezeigt.

Mikado-Status

Der Status von Mikado wird mittels drei verschiedener Bits an die SPS übertragen. Folgende Tabelle fasst die Bedeutung der einzelnen Bits zusammen.

Tab. 7: Bedeutung der einzelnen Status-Bits

Name	Erläuterung	Wert
`OutBusyBit`	Status des Feldbus-Adapters.	True = Kommando wird verarbeitet. False = Adapter ist bereit neues Kommando entgegenzunehmen
`OutReadyBit`	System-Status von Mikado.	True = Mikado betriebsbereit False = Mikado nicht bereit
`OutRunningBit`	Programm-Status von Mikado	True = Mikado-Programm läuft (entspricht in Mikado: Grün oder Gelb) False = Programm läuft nicht (entspricht in Mikado: Orange, Rot oder Grau)

Bemerkung: Neue Kommandos werden nur entgegengenommen, wenn `OutBusyBit` = False

7.9.4 Synchronisieren von SPS-Variablen und Mikado-Variablen

SPS-Variablen können auf Mikado-Variablen abgebildet werden und umgekehrt. Das Mapping zwischen SPS-Variablen und Mikado-Variablen kann im Mikado-Assistant unter „Schnittstellen“ angepasst und erweitert werden. Für die verschiedenen, unterstützten Kommunikationsprotokolle (ADS, EtherCAT, EtherNet/IP, Profinet) stehen entsprechende Einrichtungsseiten zu Verfügung. Die standardmäßige Feldbus-Konfiguration neu erstellter Zellen enthält alle notwendigen Einträge für die Ausführung der Kommandos.

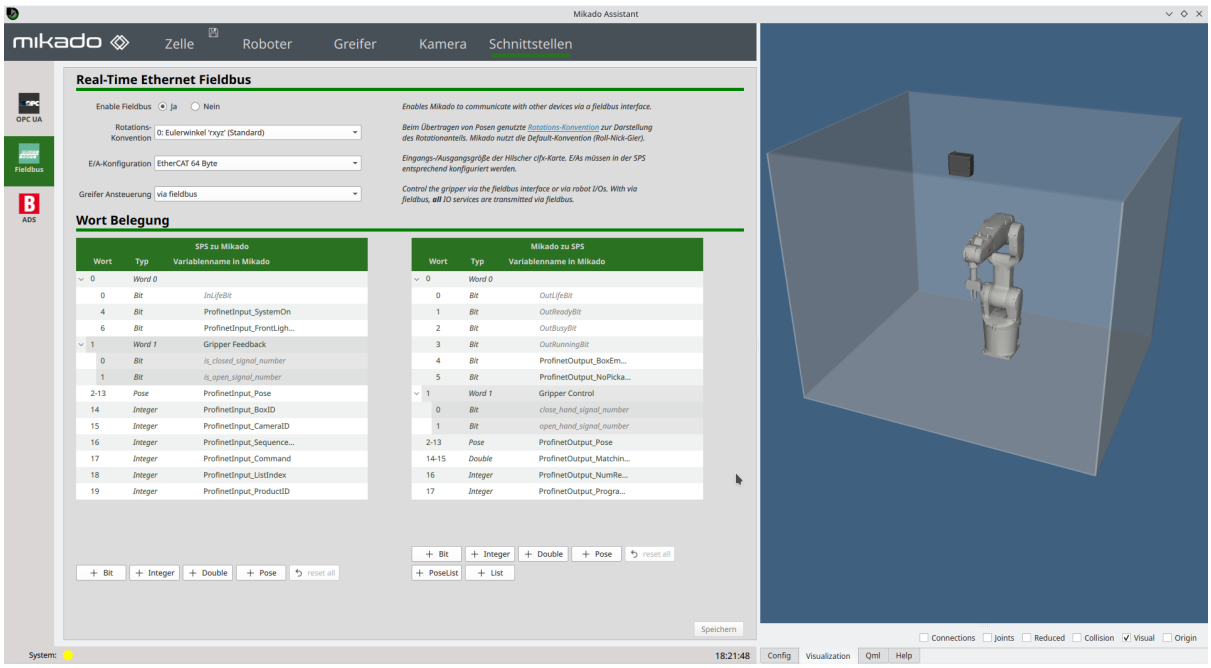


Abb. 28: Assistant zum Anpassen der Wortbelegung und synchronisierter Variablen via Feldbus

Die Feldbus-Konfiguration wird in der Konfigurationsdatei `config/profinet.yaml` bzw. `config/ads.yaml` gespeichert. Die Datei ist folgendermaßen aufgebaut:

```
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
    - ...
```

Alle in dieser Dokumentation erwähnten Namen von synchronisierten Variablen beziehen sich auf `<VariableID>`. Ist beispielsweise vom Einlesen der Variable `InSequenceID` die Rede, wird davon ausgegangen, dass die verwendete Konfigurationsdatei einen Eintrag `name: InSequenceID` enthält.

7.9.5 Rotations-Konvention

Die Rotations-Konvention legt fest, wie die Ausrichtung von Posen auf die Schnittstelle übertragen bzw. von der Schnittstelle gelesen wird. Die Ausrichtung wird dabei immer in Form von Eulerwinkeln angegeben, welche jedoch je nach Konvention auf unterschiedliche Weise interpretiert werden können (siehe unten).

Die Rotations-Konvention wird jeweils angegeben als eine Kombination aus vier Buchstaben: **r** (**r**elativ, intrinsisch, gedrehtes Koordinatensystem) oder **s** (**s**tatisch ,extrinsisch, feststehendes Koordinatensystem) gefolgt von der Reihenfolge der drei Rotationsachsen **x/y/z**. Es ergeben sich somit theoretisch 24 verschiedene Kombinationen, in denen die Ausrichtung von Posen in Eulerwinkeln dargestellt werden kann.

In der nachfolgenden Tabelle sind die Rotations-Konventionen für einige Roboterhersteller aufgeführt. Die Konvention muss passend zum verwendeten Roboter im Mikado Assistant unter dem Reiter „Schnittstellen“ eingestellt werden, damit die Ausrichtung von Posen korrekt interpretiert werden kann.

Mikado verwendet als Standard die Konvention *rxyz* (Roll-Nick-Gier / RPY). Bei Auswahl dieser Konvention werden die Mikado-Posen unverändert an die Schnittstelle weitergegeben. Bei Auswahl einer anderen Einstellung wird der Rotationsanteil einer Pose von der Schnittstelle in das entsprechende Format konvertiert.

Eine besondere Stellung nimmt die Rotations-Konvention *uvw* ein, welche von Universal Robots nativ verwendet wird. Anstatt diese Roboterseitig in Eulerwinkel umzurechnen, kann diese durch das Wählen der entsprechenden Konvention direkt verwendet werden. Die Parameter **u**, **v** und **w** stellen dabei einen Vektor dar, um welchen gedreht wird. Die Art der Rotationsbeschreibung entspricht der *Axis–Angle Representation*.

Tab. 8: Unterstützte Konventionen

ID	Konvention	Roboterhersteller	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 Fehler-Codes

Das Ausgangs-Wort `OutErrorCode` kodiert Fehlermeldungen. Die folgende Tabelle fasst alle möglichen Fehlercodes zusammen und erläutert jeweils ihre Bedeutung sowie empfohlene Maßnahmen zur Behebung.

Tab. 9: Mögliche Fehlermeldungen

Code	Fehlername	Beschreibung	Empfohlene Maßnahme
0	`NoError`	Ausgangszustand. Es liegt kein Fehler vor	-
>0	Fehler im Mikado-Programm	Projektspezifischer Fehler im Mikado-Programmablauf	Details siehe Mikado-Programm bzw. projektspezifische Dokumentation
-1	`ServiceCallFailed`	Aufruf eines Kommandos ist durch einen internen Fehler gescheitert.	Mikado Support kontaktieren
-4	`NoReplyValue`	Keinen Rückgabe-Wert erhalten (interner Fehler)	Mikado Support kontaktieren
-5	`TooManyReplyValues`	Anzahl der Rückgabewerte höher als erwartet (interner Fehler)	Mikado Support kontaktieren
-6	`FatalErrorMikadoProgram`	Mikado-Programm hat einen fatalen Fehler. Die Ausführung des Programms kann von dieser Stelle aus nicht fortgesetzt werden.	Mikado-Programm zurücksetzen und Programm / Sequenz erneut starten
-8	`UnknownCommand`	Unbekanntes Kommando aufgerufen	`InCmd` überprüfen. Gültige Kommandos siehe *Kommandos*.
-9	`LifebitClientMissing`	Lifebit des Clients wurde nicht erkannt.	Feldbus-Verbindung zwischen Mikado und dem Kommunikationspartner und Konfiguration des `LifeBit` überprüfen
-11	`TriggeringFailed`	Aufruf einer Mikado-Sequenz fehlgeschlagen. Dies kann beispielsweise auftreten, wenn Mikado nicht betriebsbereit ist oder bereits ein Programm ausgeführt wird.	Sicherstellen, dass Mikado betriebsbereit ist (`ReadyBit = true`) und dass nicht bereits ein Programm ausgeführt wird (`RunningBit = false`)
-12	`ReadingVariableFailed`	Interner Fehler beim Lesen von Daten der Feldbus-Schnittstelle	Mikado Support kontaktieren
-13	`WritingVariableFailed`	Interner Fehler beim Schreiben von Daten zur Feldbus-Schnittstelle	Mikado Support kontaktieren

Tab. 10: Mögliche Fehlermeldungen, Fortsetzung

Code	Fehlername	Beschreibung	Empfohlene Maßnahme
-15	`CancelCalibrationFailed`	Abbrechen einer bereits gestarteten Kalibrierung fehlgeschlagen	Sicherstellen, dass Kalibrierung zuvor gestartet wurde (*StartCalib*) und kein Programm ausgeführt wird (`RunningBit = false`)
-16	`CaptureForCalibrationFailed`	Kommando *CaptureCalib* ist fehlgeschlagen. Aufnahme für laufende Kalibrierung nicht erfolgreich	Sicherstellen, dass Kalibrierung zuvor gestartet wurde (*StartCalib*) und kein Programm ausgeführt wird (`RunningBit = false`)
-17	`FinishCalibrationFailed`	Kommando *FinishCalib* ist fehlgeschlagen. Kalibrierung berechnen und speichern war nicht erfolgreich	Sicherstellen, dass Kalibrierung zuvor gestartet wurde (*StartCalib*) und kein Programm ausgeführt wird (`RunningBit = false`)
-18	`StartCalibrationFailed`	Kommando *StartCalib* ist fehlgeschlagen. Kalibrierung konnte nicht gestartet werden	Sicherstellen, dass nicht bereits eine Kalibrierung läuft und kein Programm ausgeführt wird (`RunningBit = false`)
-19	`SequenceDoesNotExist`	Kommando *StartSequence* ist fehlgeschlagen. Mikado Sequenz mit der übergebenen ID existiert nicht	`SequenceID` überprüfen und Kommando *StartSequence* mit einer gültigen ID aufrufen
-20	`MissingConfigurationItem`	Ungültige Feldbus-Konfiguration. Ein erforderlicher Eintrag ist nicht vorhanden.	Mikado Support kontaktieren
-60	`RetryErrorMikadoProgram`	Mikado-Programm hat einen Fehler aber die Ausführung kann wiederholt werden. Dies kann auftreten, wenn die Verbindung zum Roboter oder zur Kamera kurzzeitig unterbrochen wird, während Mikado gerade eine Aktion ausführen möchte.	Kommando *RunMikadoProg* ausführen. In vielen Fällen ist die Ursache des Fehlers bis zum Wiederholversuch bereits automatisch behoben worden und der Ablauf kann erfolgreich fortgesetzt werden. Falls nicht, wird erneut ein Fehler ausgelöst.
-61	`RetryActionErrorMikadoProgram`	Roboterbewegung fehlgeschlagen. Ein Fahrbefehl von Mikado konnte nicht erfolgreich vom Roboter ausgeführt werden. Dies kann auftreten, wenn die Verbindung zum Roboter verloren geht oder dieser unerwartet gestoppt wird, beispielsweise durch einen Fehler oder einen Sicherheits-Stopp (Not-Aus)	Falls vorhanden, zunächst die Ursache des unerwarteten Stopps beheben und dann Kommando *RunMikadoProg* ausführen. Der Ablauf wird fortgesetzt. Wenn die Bewegung weiterhin nicht erfolgreich ausgeführt werden kann, wird erneut ein Fehler ausgelöst.
-100	`UnexpectedError`	Unerwarteter Fehler aufgetreten	Mikado Support kontaktieren

7.9.7 Kommandos

Alle Kommandos haben mindestens `OutErrorCode` als Rückgabewert. Einige Kommandos haben zusätzliche Eingabe- und Rückgabewerte. In der Detailbeschreibung der Kommandos sind die zusätzlichen Eingabe- und Rückgabewerte aufgeführt, sofern existent.

Bemerkung: Die Eingabewerte müssen **vor** dem Setzen des Kommandos am Feldbus anliegen damit sie berücksichtigt werden können. Die Rückgabewerte liegen am Feldbus an, sobald `OutBusyBit == 0` ist und die Konfigurationsdatei die entsprechenden Einträge enthält (siehe *Synchronisieren von SPS-Variablen und Mikado-Variablen*).

Folgende Tabelle listet alle verfügbaren Kommandos auf:

Tab. 11: Verfügbare Kommandos

`InCmd`	**Kommandoname**	**Beschreibung**
0	`Idle`	Wartezustand
Programm spezifische Kommandos		
1	*RunMikadoProg*	Mikado-Programm ab dem aktuellen Task starten/fortsetzen
2	*StartSequence*	Eine bestimmte Sequenz des Mikado-Programms starten. Hierfür muss SequenceID zuvor gesetzt sein
3	*PauseMikadoProg*	Ablauf nach Ausführung des aktuellen Tasks anhalten
4	*StopMikadoProg*	Mikado-Programm stoppen
5	*ResetMikadoProg*	Mikado-Programm stoppen und alle Variablen und Zustände im Mikado zurücksetzen
Allgemeine Kommandos		
8	*StartCalib*	Hand-Auge-Kalibrierung starten
9	*CaptureCalib*	Löst eine Bildaufnahme und Messung des Kalibriermusters aus
10	*FinishCalib*	Hand-Auge-Kalibrierung berechnen und speichern
11	*readPoseAndPublishTool0*	Setzt die Pose des Roboterflanschs in Mikado
14	*CancelCalib*	Hand-Auge-Kalibrierung abbrechen

Programm spezifische Kommandos

Programm spezifische Kommandos beziehen sich auf das aktuelle Mikado-Programm und können nur ausgeführt werden, wenn ein Programm geladen ist. Diese Kommandos haben keine zusätzlichen Rückgabewerte. Die gewünschten Ergebnisse hängen von der Implementierung im Mikado-Programm ab und werden individuell über die synchronisierten Variablen kommuniziert (siehe hierzu *Synchronisieren von SPS-Variablen und Mikado-Variablen*).

Andersherum kann das Verhalten des Mikado-Programms über den Feldbus beeinflusst werden, indem synchronisierte Variablen als Inputs für Tasks im Mikado-Programm verwendet werden. Dabei ist zu beachten, dass das Auslesen einer im Mikado-Programmablauf als Task-Input verwendeten synchronisierten Variable zum Zeitpunkt der Task-Ausführung geschieht. Deshalb sollten synchronisierte Variablen auf SPS-Seite nur geändert werden, wenn kein Mikado-Programm läuft.

Bemerkung: Um Fehlverhalten im Programmablauf zu vermeiden sollten synchronisierte Variablen auf SPS-Seite nur geändert werden, wenn kein Mikado-Programm läuft (`OutRunningBit = 0`).

RunMikadoProg

Mit diesem Kommando wird das aktuelle Mikado-Programm gestartet oder fortgesetzt. Nach erfolgreicher Ausführung des Kommandos wechselt das `RunningBit = True` und signalisiert das laufende Mikado-Programm. Sobald dieses von Mikado zurückgesetzt wird, können die synchronisierten Variablen ausgewertet werden.

StartSequence

Mit diesem Kommando wird die Mikado-Sequenz mit der in Wort `SequenceID` hinterlegten ID ausgeführt. Die ID einer Sequenz lässt sich ab Programm-Version 3 in dem Textfeld rechts neben dem Sequenz-Namen einstellen. In früheren Programm-Versionen muss der Sequenz-Name gleich der gesendeten ID sein, beispielsweise die Ganzzahl ‚42'.

Nach erfolgreicher Ausführung des Kommandos wechselt das `RunningBit = 1` und signalisiert das laufende Mikado-Programm. Sobald dieses von Mikado zurückgesetzt wird, können die synchronisierten Variablen ausgewertet werden.

Tab. 12: Zusätzliche Eingabe- und Rückgabewerte

Eingabewert		Rückgabewert
`InSequenceID`	ID der zu startenden Sequenz	*keine*

PauseMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms pausiert.

StopMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms gestoppt.

ResetMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms gestoppt und der Zustand des Programms und aller Variablen werden zurückgesetzt.

Allgemeine Kommandos

Allgemeine Kommandos können unabhängig vom Mikado-Programm ausgeführt werden. Die Rückgabewerte liegen auf der in der Konfigurationsdatei definierten Adresse an, sobald `OutBusyBit == 0`.

StartCalib

Mit diesem Kommando wird die Hand-Augen-Kalibrierung gestartet.

Tab. 13: Zusätzliche Eingabe- und Rückgabewerte

Eingabewert		Rückgabewert
`InCameraID`	ID der zu verwendenden Kamera	*keine*
`InFrontLightOn`	Licht während der Kalibrierung anschalten?	

CaptureCalib

Mit diesem Kommando werden Posen für die Hand-Augen-Kalibrierung gesammelt. Der Kalibrierungsprozess muss vorher durch das Kommando `StartCalib` gestartet worden sein. Es sollten zwischen 10 und 20 Posen gesammelt werden. Dieses Kommando muss dementsprechend häufig ausgeführt werden.

Tab. 14: Zusätzliche Eingabe- und Rückgabewerte

Eingabewert		**Rückgabewert**
`InPose`	Derzeitige Tool0-Pose	*keine*

Beispielablauf:

- Roboter fährt eine Pose an
- SPS schreibt die aktuelle Flanschpose (tool0) auf `InPose`
- Das Kommando `CaptureCalib` wird ausgelöst
- Mikado liest die aktuelle Roboterpose von `InPose` und führt eine Bildaufnahme durch

FinishCalib

Mit diesem Kommando wird die Hand-Augen-Kalibrierung berechnet. Dies ist der letzte Schritt im Prozess der Hand-Augen-Kalibrierung. Dies erfolgt nach einer ausreichenden Anzahl von gesammelten Patternposen mit Hilfe von `CaptureCalib`.

Tab. 15: Zusätzliche Eingabe- und Rückgabewerte

Eingabewert	**Rückgabewert**	
keine	`OutCalibMaxDist`	Maximaler Posen Fehler Distanz
	`OutCalibMeanDist`	Mittlerer Posen Fehler Distanz
	`OutCalibMeanX`	Mittlerer Posen Fehler x
	`OutCalibMeanY`	Mittlerer Posen Fehler y
	`OutCalibMeanZ`	Mittlerer Posen Fehler z
	`OutCalibMeanAngle`	Mittlerer Posen Fehler Winkel

CancelCalib

Mit diesem Kommando wird die Hand-Augen-Kalibrierung abgebrochen.

readPoseAndPublishTool0

Bemerkung: Dieses Kommando findet in der Regel nur bei Select Zellen Anwendung.

Mit diesem Kommando wird eine von der SPS übermittelte Pose eingelesen und in Mikado als aktuelle Pose des Roboterflanschs (tool0) gesetzt. Für den Mikado-Programmablauf muss die Position des Roboters bei einer Bildaufnahme bekannt sein, wobei Mikado Select selbst keine Informationen über den Roboterzustand hat. Bei der Hand-Auge-Kalibrierung ist die Benutzung dieses Kommandos nicht notwendig, da das Kommando `CaptureCalib` im Hintergrund bereits dieselben Funktionalitäten ausführt.

Tab. 16: Zusätzliche Eingabe- und Rückgabewerte

Eingabewert		**Rückgabewert**
`InPose`	Derzeitige Tool0-Pose	*keine*

7.9.8 Beispielablauf für Select-Zellen

Anhand des Beispiel-Programms `select_findObjects.program` wird im Folgenden gezeigt, wie sich ein Mikado-Programm über den Feldbus steuern lässt. Das Beispiel-Programm findet sich ab Mikado-Version 2.1 in der Programm-Bibliothek.

Voraussetzungen:

- Es muss eine Produktliste geladen sein, die ein Produkt mit dem Namen `P_0` enthält.
- Es muss eine Szene geladen sein, die eine ROI mit dem Namen `ROI_0` enthält.

Tab. 17: Beispiel-Ablauf für eine Select-Zelle

#	SPS	Mikado
Nur bei mobiler Kamera: Setze Roboterpose (bei statischer Kamera direkt weiter mit #3)		
1	Setze `InPose` auf die aktuelle Pose des Roboterflansch (tool0) Setze `InCmd = 11`	
2		Übernimmt die Werte in `InPose` als aktuelle Roboterpose
Finde Objekte		
3	Setze `InSequenceID = 1` Setze `InBoxID= 0` Setze `InProductID = 0` Setze `InCmd = 2` (`RunSequence`)	
4		Starte Sequenz mit ID = 1 (*Finde Objekte*) Setze `OutNumResults` Ende der Sequenz: `OutRunningBit = 0`
Kommuniziere Ergebnisse		
5	Lese `OutNumResults` Setze `InListIndex = 0` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
6		Starte Sequenz mit ID = 2 (*get Result*) Setze `OutPose` w.r.t. `InListIndex` Ende der Sequenz: `OutRunningBit = 0`
7	Lese `OutPose` Setze `InListIndex = 1` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
8		*wiederhole Schritt 6*
9	*wiederhole Schritt 5 mit* `InListIndex++`	
10	Lese `OutPose` Setze `InListIndex = OutNumResults` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
11		*wiederhole Schritt 6*
12	Lese `OutPose`	

7.9.9 Aktualisierungen der Feldbus-Schnittstelle

Version 3.1.0

(mikado-profinet v3.1.0, mikado_io_beckhoff v3.1.0)

Bemerkung: SPS-seitige Wertänderungen während der Programm-Laufzeit (`RunningBit = 1`) können ab dieser Version zu unerwartetem Verhalten führen, da sie sofort für das Programm sichtbar werden.

Die neue Feldbus-Adapter Version bringt die wesentliche Neuerung, dass synchronisierte Variablen ständig aktualisiert werden. Bei vorherigen Versionen mussten Variablen mit dem Kommando `syncAll` aktualisiert werden. Aus dieser Änderung resultieren folgende Konsequenzen:

- Die Nutzer-definierte Live-Eigenschaft von Variablen entfällt, weil nun alle Variablen „live“ sind.
- Das Kommando `syncAll` ist überflüssig und deshalb auch nicht mehr in dieser Dokumentation aufgeführt. Das Kommando kann weiterhin gesendet werden. Der zurückgegebene `OutErrorCode` ist in diesem Fall `NoError`.
- Erfolgen SPS-seitige Wertänderungen zur Programm-Laufzeit kann es zu unerwartetem Verhalten kommen.

Nachstehende Tabelle zeigt beispielhaft wie SPS-seitige Änderungen zu einem ungünstigen Zeitpunkt zu einem unerwartetem Verhalten führen können. Angenommen in Mikado ist ein Programm geladen, das unter Anderem den Task `Finde Szenenobjekt` aufruft. Als Task-Input wird die synchronisierte Variable `InBoxID` verwendet.

Tab. 18: Beispiel für SPS-seitige Änderungen zu einem ungünstigen Zeitpunkt

Zeit	SPS	Mikado	Kommentar
t_1	Setze `InBoxID = 42` Setze `InCmd = 1` (Starte Programm)		SPS setzt die Variable `InBoxID = 42` und sendet Start-Kommando

t_2		`OutBusyBit = 1`
		`OutRunningBit = 1`
		`OutBusyBit = 0`
			Adapter führt Kommando aus
			Mikado startet Programm
			Kommando-Ausführung beendet

t_3	`InBoxID = 0` (**falsch**) `InCmd = 0`		**Dies ist ein ungünstiger Zeitpunkt.** Im Programmablauf verwendete Variablen sollten nicht zur Laufzeit geändert werden.
t_4		Programm ruft auf `FindSceneObject (InBoxID)`	Mikado ist mittlerweile beim Aufruf des Tasks angekommen. Es wird die momentan anliegenden `InBoxID = 0` verwendet.
t_5		`OutRunningBit = 0`	Programm-Ende erreicht
t_6	`InBoxID = 0` (**richtig**)		**Der richtige Zeitpunkt um Variablen zurückzusetzen.**

Version 3.1.2

Ab *mikado-profinet* Version 3.1.2 unterstützt das Paket neben Profinet auch die Protokolle EtherCAT und EtherNet/IP.

Das gewünschte Protokoll kann im Mikado Assistant auf der Seite `Feldbus-Schnittstelle` konfiguriert werden.

7.9.10 Standard-Feldbus-Schnittstelle

Die verschiedenen Feldbusschnittstellen haben viele Gemeinsamkeiten, daher folgt hier eine Beschreibung der vorhandenen Datentypen und die *Standard-Wortbelegung*.

Diese Schnittstellenbeschreibung mit der Abbildung von Werten auf Worte / Bytes / Bits zeigt die Umsetzung für die Protokolle Profinet, EtherCAT und EtherNet/IP. Andere Protokolle weichen wie im entsprechenden Abschnitt beschrieben in Teilen davon ab, verwenden beispielsweise wie bei Beckhoff ADS eine Abbildung auf Namen anstelle von Speicheradressen.

Ein Überblick zur Dokumentation der Feldbus-Schnittstelle findet sich *hier*.

Byte-Reihenfolge

Die Byte-Reihenfolge entspricht der Siemens Konvention Big-Endian. Bei der Interpretation der Wörter muss deshalb auf der SPS je nach Konfiguration des Master-Systems ein Byte-Swap erfolgen.

Feldbus-Datentypen

Daten werden prinzipiell wortweise in 2-Byte-Paketen übertragen. Wie oben erwähnt, ist bei der Interpretation der Daten die Byte-Reihenfolge Big-Endian zu beachten. Folgende Datentypen sind möglich:

- `Bits`: Das Wort wird in 16 einzelne Bits zerlegt, wobei jedes Bit als Bool interpretiert wird.
- `Integer`: Das Wort wird als Ganzzahl interpretiert.
- `Double`: Zwei Wörter (32 Bit) werden gemeinsam als eine Gleitkommazahl interpretiert.
- `Pose`: 12 aufeinanderfolgende Wörter werden als sechs Double-Werte einer Pose interpretiert (X, Y, Z, R1, R2, R3).
- `PoseList` (nur Mikado an SPS): `12 x` <`Listenlänge`> aufeinanderfolgende Wörter, die als Liste von Posen interpretiert werden. Für eine Liste der Länge 5 werden z.B. 60 aufeinanderfolgende Wörter benötigt.
- `StringList` (nur Mikado an SPS): `2 x` <`Listenlänge`> aufeinanderfolgende Wörter werden als eine Liste von Double-Werten interpretiert. Strings werden dabei automatisch zu Double konvertiert. Falls ein String nicht zu einem Double konvertierbar ist, kann er nicht über diese Schnittstelle übetragen werden.

Standard-Wortbelegung

Die meisten verwendeten Wörter werden direkt auf Mikado-Variablen abgebildet.

Davon ausgenommen sind die in der Abbildung *Übersicht der Standard-Belegung von Input und Output Byte-Arrays* grau gekennzeichneten Wörter, deren Werte unabhängig vom Mikado-Programm durch den Feldbus-Adapter gesetzt werden.

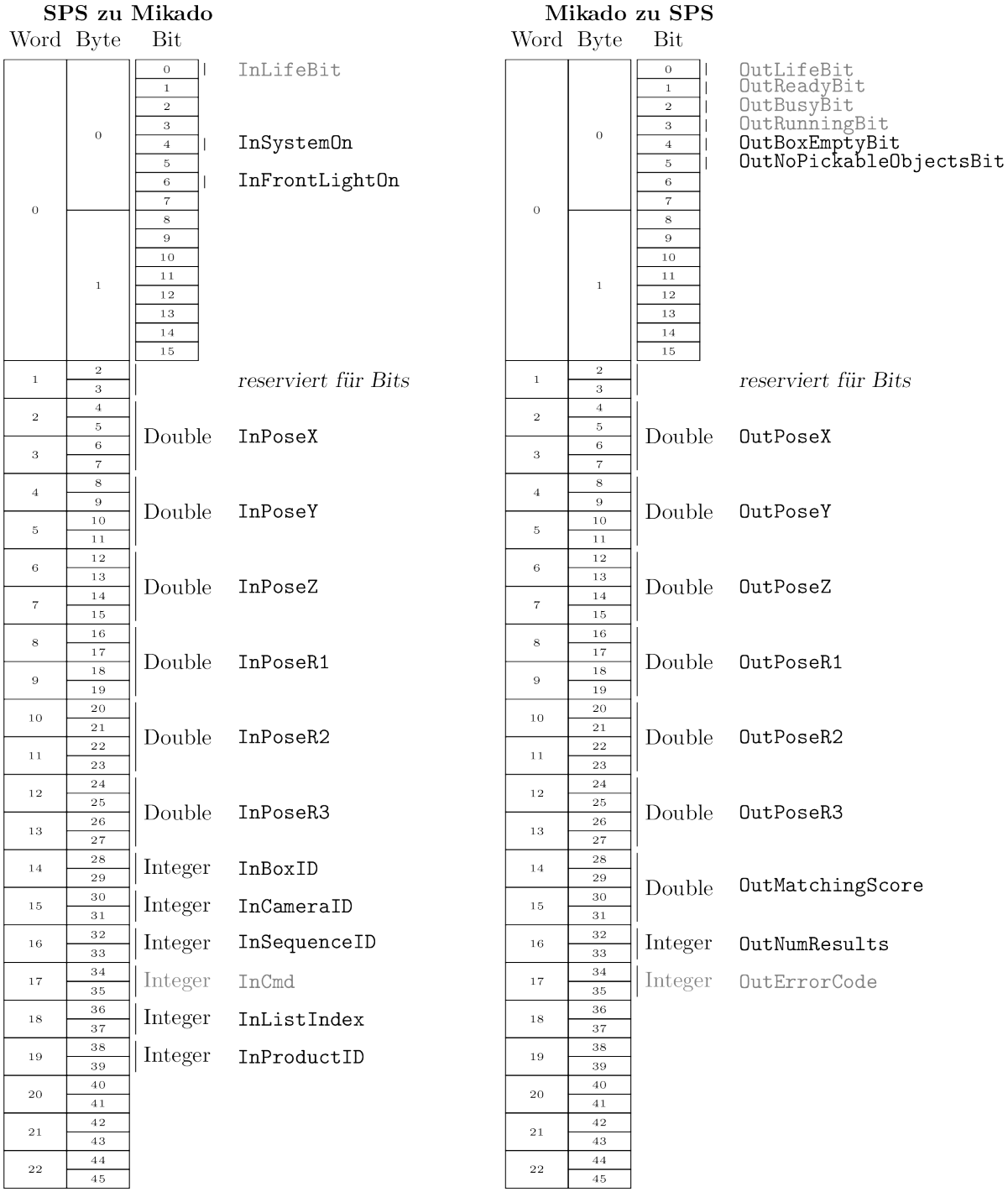


Abb. 29: Übersicht der Standard-Belegung von Input und Output Byte-Arrays

7.9.11 Spezifische Konfiguration für Profinet

Dieser Abschnitt beschreibt die notwendigen Schritte zur Einrichtung der Hardware für die Kommunikation via Profinet.

Ein Überblick zur Dokumentation der Feldbus-Schnittstelle findet sich *hier*.

GSDML

Die Schnittstellenbeschreibung bei ProfiNet-Anbindung ist über die GSDML-Datei festgelegt. Nach der Installation des Debian-Pakets `mikado-profinet` findet sich diese Datei unter folgendem Pfad:

`/opt/cifx/gsdml/GSDML-V2.35-HILSCHER-CIFX RE PNS-20190108.xml`

Folgendes Profinet-IO Gerät der GSDML ist hierbei am Master einzubinden:

`CIFX RE/PNS V3.5 - V3.5.x`

Der Mikado-Controller stellt je nach ausgewählter Konfiguration folgende Ein-/Ausgangslayouts zur Verfügung. Dabei sind Ein- und Ausgänge aus Sicht des Profinet-Masters definiert:

- 64 Byte Ein- und Ausgang
- 128 Byte Ein- und Ausgang
- 256 Byte Ein- und Ausgang
- 512 Byte Ein- und Ausgang

Diese Ein- und Ausgänge werden entsprechend der konfigurierten Wortbelegung geschrieben bzw. gelesen, siehe *Standard-Wortbelegung*.

Standard-Feldbus-Schnittstelle

Die Byte-Reihenfolge, die verfügbaren Datentypen und die Standard-Wortbelegung entsprechen der *Standard-Feldbus-Schnittstelle*.

7.9.12 Spezifische Konfiguration für EtherCAT

Dieser Abschnitt beschreibt die notwendigen Schritte zur Einrichtung der Hardware für die Kommunikation via EtherCAT.

Ein Überblick zur Dokumentation der Feldbus-Schnittstelle findet sich *hier*.

Schnittstellenbeschreibung

Die Schnittstellenbeschreibung bei EtherCAT-Anbindung ist über XML-Datei festgelegt. Nach der Installation des Debian-Pakets `mikado-profinet` findet sich diese Datei unter folgendem Pfad:

`/opt/cifx/ethercat/Hilscher CIFX RE ECS V4.6.X.{BYTE_GRÖßE}.xml`

Folgendes EtherCAT Gerät ist hierbei am Master einzubinden: `Hilscher CIFX RE ECS`

Der Mikado-Controller stellt je nach ausgewählter Konfiguration folgende Ein-/Ausgangslayouts zur Verfügung. Dabei sind Ein- und Ausgänge aus Sicht des EtherCAT-Masters definiert:

- 64 Byte Ein- und Ausgang
- 128 Byte Ein- und Ausgang
- 256 Byte Ein- und Ausgang

Diese Ein- und Ausgänge werden entsprechend der konfigurierten Wortbelegung geschrieben bzw. gelesen, siehe *Standard-Wortbelegung*.

Standard-Feldbus-Schnittstelle

Die Byte-Reihenfolge, die verfügbaren Datentypen und die Standard-Wortbelegung entsprechen der *Standard-Feldbus-Schnittstelle*.

7.9.13 Spezifische Konfiguration für EtherNet/IP

Dieser Abschnitt beschreibt die notwendigen Schritte zur Einrichtung der Hardware für die Kommunikation via EtherNet/IP.

Ein Überblick zur Dokumentation der Feldbus-Schnittstelle findet sich *hier*.

Schnittstellenbeschreibung

Die Schnittstellenbeschreibung bei EtherNet/IP-Anbindung ist in einer EDS-Datei festgelegt. Nach der Installation des Debian-Pakets `mikado-profinet` findet sich diese Datei unter folgendem Pfad:

`/opt/cifx/ethernet_ip/HILSCHER CIFX-RE EIS V1.1.EDS`

Folgendes EtherNet/IP Gerät ist hierbei am Master einzubinden:

`Hilscher CIFX 50-RE EIS`

Der Mikado-Controller stellt je nach ausgewählter Konfiguration folgende Ein-/Ausgangslayouts zur Verfügung. Dabei sind Ein- und Ausgänge aus Sicht des EtherNet/IP-Masters definiert:

- 64 Byte Ein- und Ausgang
- 128 Byte Ein- und Ausgang
- 256 Byte Ein- und Ausgang

Diese Ein- und Ausgänge werden entsprechend der konfigurierten Wortbelegung geschrieben bzw. gelesen, siehe *Standard-Wortbelegung*.

Warnung: Die EtherNet/IP-Schnittstelle von Mikado ist standardmäßig für die Zuweisung einer IP-Adresse über `DHCP` konfiguriert. Diese Einstellung kann derzeit nicht vom Anwender geändert werden. Sollten Sie für Ihre Anwendung eine statische IP-Adresse für die Schnittstelle benötigen, wenden Sie sich bitte an den Mikado Support.

Standard-Feldbus-Schnittstelle

Die Byte-Reihenfolge, die verfügbaren Datentypen und die Standard-Wortbelegung entsprechen der *Standard-Feldbus-Schnittstelle*.

7.9.14 Spezifische Konfiguration für Beckhoff ADS-Schnittstelle

Dieser Abschnitt beschreibt die notwendigen Schritte zur Einrichtung der Kommunikation via Beckhoff ADS.

Ein Überblick zur Dokumentation der Feldbus-Schnittstelle findet sich *hier*.

Datentypen

Datentyp Mikado	Datentyp SPS	Bemerkung
`Bool`	`BOOL`	
`Integer`	`INT`	
`Double`	`REAL`	
`Pose`	`ARRAY [1..6] OF REAL`	Die sechs Werte repräsentieren X, Y, Z, R1, R2, R3 der Mikado-Pose
`PoseList`	`ARRAY [1..6*n] OF REAL`	Die Länge des Arrays muss ein Vielfaches von 6 sein. Jeweils 6 benachbarte Werte repräsentieren eine Pose.
`String`	`STRING`	
`StringList`	`ARRAY [1..n] OF REAL`	**Nur Mikado an SPS**. Die Länge des Arrays unterliegt keinen Beschränkungen. Die Werte innerhalb der Mikado-StringList werden zu Doubles konvertiert.

Bemerkung: Ist eine PoseList oder StringList in Mikado länger als der SPS-seitige verfügbare Platz, wird die Mikado-seitige Liste auf die verfügbare Länge abgeschnitten.

Konfiguration

In der Konfigurationsdatei `ads.yaml` lassen sich zusätzlich zu den Feldbus-Adapter-Parametern folgende ADS-spezifischen Parameter einstellen:

```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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
output_array_name: GVL.outputBytes # IO-Name on PLC

# 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 Informationen zum Betriebssystem

Mikado Select/Control wird vorinstalliert auf einem Mikado Controller geliefert und verwendet ein angepasstes Kubuntu 22.04 als Grundlage. Es erfolgt eine automatische Anmeldung des Benutzers *mikado*. Das Standardpasswort lautet ebenfalls *mikado*. Dieses Passwort wird benötigt um beispielsweise Updates zu installieren oder tiefergehende Änderungen am System vorzunehmen.

Warnung: Wir empfehlen das Passwort im Rahmen der Inbetriebnahme zu ersetzen. Bitte beachten Sie dabei gegebenenfalls relevante Unternehmens- oder Konzernrichtlinien. Das Passwort kann mittels Systemeinstellungen > Benutzer > Passwort ändern oder über den Kommandozeilenbefehl *passwd* geändert werden.

KAPITEL 8

Mikado Control erweitern

8.1 JavaScript Task

Der *JavaScript Task* erlaubt es eigenen JavaScript/ECMAScript Code als Task auszuführen. Zusätzlich zu allen Funktionalitäten von JavaScript erlaubt es dieser Task über vordefinierte Module direkt auf einige Mikado-Variablen zu zugreifen. Folgende Tabelle gibt eine Übersicht über die durch die Module bereitgestellten Schnittstellen.

Tab. 1: **Übersicht der Schnittstellen im JavaScript Task**

task	Erlaubt Zugriff auf die Visualisierung des Tasks und das Setzen von Folge-Sequenzen.
dataStore	Erlaubt Lese- und Schreib-Zugriff auf alle Mikado-Variablen.
picksearch	Erlaubt Zugriff auf die Ergebnisse der Griffsuche.
kinematics	Modul für Kinematik, Kollisionsprüfung und Erreichbarkeitsüberlegungen.
product	Erlaubt Zugriff auf Produkt-spezifische Eigenschaften.
robot	Zugriff auf den Zustand des Roboters
scene	Erlaubt den Zugriff auf Szenenobjekte wie Kisten, ROIs und Hindernisse
file	Erlaubt Zugriff auf Dateien
program	Erlaubt Zugriff auf Mikado Programme
lab	Erlaubt Zugriff auf Mikado-spezifische, selten genutzte, aber dennoch nützliche Funktionen.
pointCloud	Erlaubt Zugriff auf Attribute der Punktewolke

Ready-to-use Vorlagen für bestimmte Anwendungsfälle sind hier aufgelistet:

- *Beispiel: Benutzerdefinierte Logdatei aus Mikado*.
- *Beispiel: Mikado PoseList sortieren*.

8.1.1 Allgemeines

Log-Einträge können innerhalb eines JavaScript Tasks wie folgt geschrieben werden:

```
console.log("My log message")
```

Weiterführende Links zu JavaScript im Allgemeinen:

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

Auf das Task-Modul kann über die Variable `task` zugegriffen werden. Der Statustext kann über `task.status` bearbeitet werden. Wenn `task.setNextSequence` gesetzt wird, wird die Sequenz mit diesem Namen als nächstes ausgeführt, bei einem leeren String wird die Aktuelle weiter fortgesetzt:

```
task.status = "Running"
task.setNextSequence("Run to goal")
```

Tritt ein Fehler auf, kann der Programmablauf mit `task.setError("Beschreibung")` beendet werden oder mit `task.setErrorMightRetry("Beschreibung")` pausiert und zum erneuten Versuch angeboten werden.

8.1.3 dataStore

Auf Variablen kann über das `dataStore` Modul zugegriffen werden. Man verwendet dazu `dataStore.data(name)` bzw. `dataStore.setData(name, value)` Beispielsweise:

```
var a = dataStore.data("a")
a = a + 1
var success = dataStore.setData("a", a)
```

Hierbei liefert die Funktion `dataStore.setData(name, value)` als Rückgabewert `true`, wenn der Wert erfolgreich geschrieben wurde. Im Fehlerfall ist der Rückgabewert `false` und im Log sind Details zum entsprechenden Fehler zu finden (z.B. bei einem Formatierungsfehler).

Bei einigen Variablentypen (Pose, JointPosition und PoseList) liefert die Funktion `dataStore.data(name)` jedoch interne Datentypen zurück. Hier bietet sich stattdessen die Verwendung der Funktion `dataStore.getJsObject(name)` an. Diese liefert ein JavaScript-typisches Array oder Object zurück. Es kann dann direkt bearbeitet und ggf. mit `dataStore.setJsObject(name, wert)` zurück geschrieben werden:

```
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])
```

Mit der Funktion `dataStore.addVariable(name, type)` lassen sich zudem neue Variablen anlegen. `dataStore.setVariableInitialValue(name)` führt dazu, dass der aktuelle Wert der Variable als Initialwert übernommen wird.

8.1.4 picksearch

Das `picksearch` Modul bietet Zugriff auf Ergebnisse der Griff- und Objektsuche. Folgende Funktionen stehen zu Verfügung:

- *getPickedObjectPose*
- *getBestPick*
- *getFoundObjectPoses*
- *setUserBins*
- *getSucceededPicks*
- *getLastUsedGrasp*
- *rerunPickSearch*
- *awaitBestPickPerObject*

getPickedObjectPose

Pose des momentan gegriffenen Objekts im Weltkoordinatensystem auslesen. Zuvor muss ein *Teil greifen* Task erfolgreich ein Objekt gegriffen haben. Wurde kein Objekt gegriffen oder wurde das Objekt bereits abgelegt, gibt die Funktion `null` zurück.

Verwendung: `Pose = picksearch.getPickedObjectPose()`

```
var pickedObjectPose = picksearch.getPickedObjectPose()
if (pickedObjectPose != null) {
 dataStore.setJsObject("pickedObjectPose", pickedObjectPose)
}
```

getBestPick

Informationen über den besten zu diesem Zeitpunkt bekannten Griff auslesen. Zuvor muss ein *Griffsuche starten* Task ausgeführt worden sein.

Akzeptiert optional den Namen eines Produktes (`productName`), den Namen einer Kiste (`boxName`) sowie den Namen eines oder mehrerer Greifer (`endEffectorName(s)`) als Argumente, um eine bestimmte Griffsuche auszuwählen. Wird die Funktion ohne Produkt- und Kisten-Namen aufgerufen, wird die zuletzt gestartete Suche verwendet. Wird kein Greifer angegeben, wird die zu dieser Suche gehörende zuletzt gestartete Griffsuche verwendet.

Gibt ein JavaScript Objekt mit folgenden Eigenschaften zurück:

- Alle Informationen, die auch in einem `Grasp` enthalten sind, siehe z.B. *getLastUsedGrasp*
- `objectName` (`String`): Name des gegriffenen Objekts
- `objectPose` (`Pose`): Pose des gegriffenen Objekts im Weltkoordinatensystem
- `graspTcpPose` (`Pose`): TCP Pose des Griffs (Griffpose) im Weltkoordinatensystem
- `graspTool0Pose` (`Pose`): Tool0 Pose (Flanschpose) des Griffs im Weltkoordinatensystem
- `startTcpPose` (`Pose`): Ausgangspose des Griffs (siehe *Greifprozess*)
- `approachTcpPose` (`Pose`): Anfahrtspose des Griffs (siehe *Greifprozess*)
- `retractTcpPose` (`Pose`): Entnahmepose des Griffs (siehe *Greifprozess*)
- `finalTcpPose` (`Pose`): Endpose des Griffs (siehe *Greifprozess*)
- `distanceJointSpace` (`Double`): Trajektoriendistanz im Gelenkraum

Verwendung:

- `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"]

 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

Gibt eine Liste der Positionen aller gefundenen Objekte zurück.

Akzeptiert optional den Namen eines Produktes (`productName`) und den Namen einer Kiste (`boxName`) als Argumente, um eine bestimmte Griffsuche auszuwählen. Wird die Funktion ohne Argumente aufgerufen, wird die zuletzt gestartete Griffsuche verwendet.

Verwendung ohne Argumente: `PoseList = picksearch.getFoundObjectPoses()`

```
// eine Liste aller gefundenen Objekte auslesen und in eine Mikado
// Variable speichern
var foundPoses = picksearch.getFoundObjectPoses()
dataStore.setJsObject("foundPoses", foundPoses)
```

Verwendung mit Argumenten: `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

Mit dieser Funktion kann für jedes Objekt ein Bin gesetzt werden, der mit dem Kriterium `user_binning` verwendet werden kann (siehe *Priorisierung*). Die Anzahl der Einträge in der übergebenen Liste (`bins`) muss exakt mit der Anzahl der gefundenen Objekte übereinstimmen.

Akzeptiert optional den Namen eines Produktes (`productName`), den Namen einer Kiste (`boxName`) sowie den Namen eines oder mehrerer Greifer (`endEffectorName(s)`) als Argumente, um eine bestimmte Griffsuche auszuwählen. Wird die Funktion ohne Produkt- und Kisten-Namen aufgerufen, wird die zuletzt gestartete Suche verwendet. Wird kein Greifer angegeben, wird die zu dieser Suche gehörende zuletzt gestartete Griffsuche verwendet.

Verwendung ohne Argumente: `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)
```

Verwendung mit Argumenten:

- `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

Gibt eine Liste alle bisher erfolgreichen Griffe zurück. Die Sortierung folgt den normalen Kriterien der PickSearch.

Akzeptiert optional den Namen eines Produktes (`productName`) und den Namen einer Kiste (`boxName`) als Argumente, um eine bestimmte Suche auszuwählen. Akzeptiert optional außerden den Namen eines oder mehrerer Greifer (`endEffectorName(s)`) um eine bestimmte Griffsuche auszuwählen. Wird die Funktion ohne Produkt- und Kisten-Namen aufgerufen, wird die zuletzt gestartete Suche verwendet. Wird kein Greifer angegeben, wird die zu dieser Suche gehörende zuletzt gestartete Griffsuche verwendet.

Verwendung ohne Argumente: `picks = picksearch.getSucceededPicks()`

```
// Liste aller erfolgreichen Griffe auf der Konsole ausgeben
var picks = picksearch.getSucceededPicks()

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);
}
```

Verwendung mit Argumenten:

- `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

Informationen über den aktuellen (zuletzt verwendeten) Griff auslesen. Zuvor muss ein Objekt mit dem *Teil greifen* Task gegriffen worden sein.

Gibt ein JavaScript Objekt mit folgenden Eigenschaften zurück:

- `graspName` (`String`): Name des verwendeten Griffs
- `category` (`String`): Kategorie des Griffs („Preferred“ / „Standard“ / „Experimental“)
- `objectToTcpTransform` (`Pose`): TCP Pose des Griffs (Griffpose) im lokalen Koordinatensystem des Teils
- `objectToTool0Transform` (`Pose`): Tool0 Pose (Flanschpose) des Griffs im lokalen Koordinatensystem des Teils
- `graspOffset` (`Pose`): Transformation von gegriffenem Objekt zum Greifer-TCP
- `isInwards` (`Bool`): Innengriff (true) oder Außengriff (false)
- `gripper` (`String`): Name des für diesen Griff verwendeten Greifers
- `tcpName` (`String`): Name des für diesen Griff verwendeten TCP

Verwendung: `grasp = picksearch.getLastUsedGrasp()`

```
// Informationen über den aktuellen Griff ausgegeben
var grasp = picksearch.getLastUsedGrasp()

task.status = "Verwendeter Griff: "+grasp["graspName"]+"\n"
+ "Kategorie: "+grasp["category"]+"\n"
```

rerunPickSearch

Wiederholung einer zuvor bereits durchgeführten Griffsuche ohne erneute Objektsuche. Diese Funktion dient hauptsächlich dazu, bei Anwendungen mit mehreren Greifern (oder TCPs) nacheinander Griffe für verschiedene Greifer berechnen zu lassen, ohne die Objektsuche jedes Mal wiederholen zu müssen.

Akzeptiert optional den Namen eines Produktes (`productName`) und den Namen einer Kiste (`boxName`) als Argumente, um eine bestimmte Suche auszuwählen. Akzeptiert optional außerden den Namen eines oder mehrerer Greifer (`endEffectorName(s)`) um eine bestimmte Griffsuche auszuwählen. Wird die Funktion ohne Produkt- und Kisten-Namen aufgerufen, wird die zuletzt gestartete Suche verwendet. Wird kein Greifer angegeben, wird die zu dieser Suche gehörende zuletzt gestartete Griffsuche verwendet.

Verwendung ohne Argumente:

- `rerunPickSearch()`

```
var success = picksearch.rerunPickSearch()
```

- `rerunPickSearch(String searchParams)`

```
var success = picksearch.rerunPickSearch("Suchparameter mit pick_search.yaml")
```

Verwendung mit Argumenten:

- `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

Warten, bis die Griffsuche beendet ist und eine Liste des jeweils besten gültigen Griffs pro Objekt zurückgeben. Die zurückgegebene Liste beinhaltet JavaScript Objekte mit den Eigenschaften eines `Pick`, siehe *getBestPick*.

Akzeptiert optional den Namen eines Produktes (`productName`) und den Namen einer Kiste (`boxName`) als Argumente, um eine bestimmte Griffsuche auszuwählen. Wird die Funktion ohne Argumente aufgerufen, wird die zuletzt gestartete Griffsuche verwendet.

Verwendung ohne Argumente: `GraspList = awaitBestPickPerObject()`

```
var bestPickPerObjectList = picksearch.awaitBestPickPerObject()
```

Verwendung mit Argumenten:

- `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

Das `kinematics`-Modul erlaubt Zugriff auf Funktionen zur Berechnung der Vorwärts- sowie der Rückwärts-Kinematik. Außerdem stellt es Funktionen bereit zur Überprüfung der Erreichbarkeit einer Zielpose zu einer gegebenen Startpose. Folgende Funktionen stehen zu Verfügung:

- *forwardKinematics*
- *inverseKinematics*

- *isMovePossible*
- *isMovePossibleCartesian*

forwardKinematics

Berechnet die Pose eines Target-Links bei gegebenen Gelenkwinkeln. Standardmäßig wird die Pose von „tool0“ berechnet.

Gibt ein Objekt mit folgenden Keys zurück:

- `success` (`Bool`): Vorwärts-Kinematik konnte gelöst werden?
- `result` (`Pose`): Pose des Target-Links. Wird nur bei erfolgreicher Durchführung gesetzt.

Verwendung: `Var = forwardKinematics(JointPose, String = "tool0")`

```
// Berechne TCP-Pose zu gegebenen Gelenkwinkeln
var jointPose = dataStore.getJsObject('EineJointPose')
var map = kinematics.forwardKinematics(jointPose, "gripper_tcp")
if (map.success) {
    dataStore.setJsObject('Pose_TCP', map.result)
}
```

inverseKinematics

Berechnet die Gelenkwinkel des Roboters, sodass ein Target-Link die gegebene Pose einnimmt. Standardmäßig wird „tool0“ als Target-Link verwendet. Optional kann ein Initialwert für die Gelenkwinkel Optimierung gesetzt werden.

Gibt ein Objekt mit folgenden Keys zurück:

- `success` (`Bool`): Sind Fehler aufgetreten? Folgende Rückgabewerte werden nur bei erfolgreicher Durchführung gesetzt.
- `ikCollisionFree` (`Bool`): Kollisionsfreie Lösung gefunden?
- `ikSolutionFound` (`Bool`): Lösung gefunden?
- `result` (`JointPose`): Gelenkwinkel des Roboters.

Verwendung: `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

Überprüft ob eine kollisionsfreie Bahn von Start- zu Zielpose gefunden werden kann. Standardmäßig wird als Startpose die momentane Roboterpose verwendet. Sowohl Start- als auch Zielpose müssen als JointPose übergeben werden. Verwenden Sie ggf. *inverseKinematics* um eine Pose in eine JointPose zu überführen.

Gibt ein Objekt mit folgenden Keys zurück:

- `success` (`Bool`): Sind Fehler aufgetreten? Folgende Rückgabewerte werden nur bei erfolgreicher Durchführung gesetzt.
- `isPossible` (`Bool`): Kollisionsfreie Bahn gefunden?
- `reason` (`String`): Information warum das Ziel ggf. nicht erreichbar ist.

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

Überprüft ob eine kartesische, kollisionsfreie Bahn von Start- zu Zielpose gefunden werden kann. Eine kartesische Bahn ist die direkte Linie zwischen Start- und Zielpose im kartesischem Raum. Die Zielpose muss als Pose übergeben werden.

Die Startpose muss als JointPose übergeben werden. Standardmäßig wird als Startpose die momentane Roboterpose verwendet. Verwenden Sie ggf. *inverseKinematics* um eine Pose in eine JointPose zu überführen.

Optional kann ein Target-Link übergeben werden, welcher kartesisch auf gegebene Pose gefahren werden soll. Standardmäßig wird als Target-Link „tool0“ verwendet.

Gibt ein Objekt mit folgenden Keys zurück:

- `success` (`Bool`): Sind Fehler aufgetreten? Folgende Rückgabewerte werden nur bei erfolgreicher Durchführung gesetzt.
- `isPossible` (`Bool`): Kollisionsfreie Bahn gefunden?
- `goalJointPosition` (`JointPosition`): last point of trajectory (goal) but only given if isPossible = true (type: JointPosition)
- `reason` (`String`): Information warum das Ziel ggf. nicht erreichbar ist.

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)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
console.log("move possible: " + map.isPossible)
console.log("reason: " + map.reason)
```

8.1.6 product

Das `product`-Modul bietet Zugriff auf Produkt-spezifische Eigenschaften. Folgende Funktionen stehen zu Verfügung:

- *activeProduct*
- *getUsedEndEffectorNames*
- *getGraspOffset*
- *getDimensions*
- *getAllProductNames*

activeProduct

Liest oder setzt das aktuelle Produkt. Das Setzen des aktuellen Produkts kann auch über den Task *Setze aktives Produkt* geschehen.

Verwendung: `String = activeProduct` oder `activeProduct = String`

```
// Setzen des aktiven Produkts
product.activeProduct = "P_1"

// Lesen des aktiven Produkts
task.status = "active product is " + product.activeProduct
```

getUsedEndEffectorNames

Bei Anwendungen mit mehreren Greifern kann mit dieser Funktion eine Liste aller Greifer abgefragt werden, welche für das aktive Produkt konfiguriert sind. Die Konfiguration eines Greifers für ein Produkt geschieht auf der Produkt-Seite.

Bemerkung: Für Anwendung mit nur einem Greifer hat diese Funktion keine Bewandtnis.

Verwendung: `StringList = getUsedEndEffectorNames()`

```
// gibt Liste an angelegten Greifern von aktivem Produkt zurück
var greiferListe = product.getUsedEndEffectorNames()
```

getGraspOffset

Lesen der Griff-Pose (bzw. der Position des Teils im Greifer) eines bestimmten Griffs für das aktive Produkt.

Verwendung: `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)
```

Im obigen obige Beispielcode wird die Pose des „Basisgriff“ gelesen. Die zum Beispielcode passende Griffdefinition ist im untenstehendem Screenshot dargestellt. Die Griff-Pose wird anschließend dazu verwendet die Pose des momentan angehängten Teils zu überschreiben. Dies kann bspw. dann nützlich sein, wenn sich das Teil durch Formschluss im Greifer zentriert. Siehe auch das Beispiel zu *setCurrentGraspOffset*.

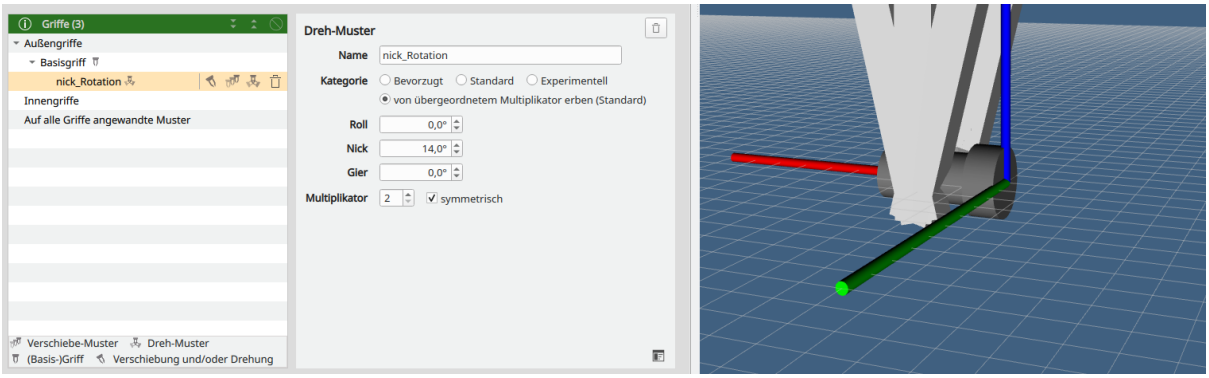


Abb. 1: Beispielhafte Einträge im Griffeditor

getDimensions

Berechnet den kleinst möglichen umschließenden Quader für das spezifizierte Produkt und gibt dessen Abmessungen zurück. Dabei ist die Orientierung am Produkt-Koordinatensystem ausgerichtet (Axis-align Bounding Box AABB). Ist kein Produkt spezifiziert werden die Abmessungen des aktiven Produkts berechnet.

Gibt ein Objekt mit folgenden Keys zurück:

- `x` (`Double`): Abmessung in X-Richtung des Produkt-Koordinatensystems [m]
- `y` (`Double`): Abmessung in Y-Richtung des Produkt-Koordinatensystems [m]
- `z` (`Double`): Abmessung in Z-Richtung des Produkt-Koordinatensystems [m]

Verwendung: `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
```

Hinweis: Auf die Produktabmessungen kann auch mit dem Task *Produktabmessungen auslesen* zugegriffen werden.

getAllProductNames

Liste mit allen Produktnamen der aktiven Produktliste auslesen.

Verwendung: `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

Folgende Funktionen stehen zu Verfügung:

- *getLinkTransform*
- *readInput*
- *readOutput*
- *writeOutput*
- *getCurrentTipLink*
- *getCurrentEndEffectorGroupName*
- *isObjectAttached*
- *runRobotProgram*

getLinkTransform

Gibt die absolute Transformation eines Links relativ zum World-Frame basierend auf der Soll-Position nach Ausführung aller vorherigen Tasks aus. Um die Ist-Position ohne Vorlauf auszulesen kann *getTF* verwendet werden.

Verwendung: `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

Liest den Wert eines Robotereingangs oder je nach Konfiguration eines Feldbus-Bits.

Verwendung: `value = robot.readInput(Number)`:

```
var input = robot.readInput(23)
```

readOutput

Liest den Wert eines Roboterausgangs oder je nach Konfiguration eines Feldbus-Bits.

Verwendung: `value = robot.readOutput(Number)`:

```
var output = robot.readOutput(23)
```

writeOutput

Schreibt den Wert eines Roboterausgangs oder je nach Konfiguration eines Feldbus-Bits.

Verwendung: `value = robot.writeOutput(Number, Value)`:

```
var output = robot.writeOutput(23, true)
```

getCurrentTipLink

Name des gerade aktiven TCP auslesen. Besonders nützlich bei Roboterzellen mit Wechselgreifer, bei denen es mehrere Greifer und somit mehrere TCPs gibt, von denen aber immer nur einer aktiv ist.

Verwendung: `tipLinkName = robot.getCurrentTipLink()`:

```
var tipLinkName = robot.getCurrentTipLink()
task.status = "Aktiver TCP: " + tipLinkName
```

getCurrentEndEffectorGroupName

Name des gerade aktiven Greifers auslesen. Besonders nützlich bei Roboterzellen mit Wechselgreifer, bei denen es mehrere Greifer gibt, von denen aber immer nur einer aktiv ist.

Verwendung: `endEffectorName = robot.getCurrentEndEffectorGroupName()`:

```
var endEffectorName = robot.getCurrentEndEffectorGroupName()
task.status = "Aktiver Greifer: " + endEffectorName
```

isObjectAttached

Auslesen, ob in Mikado gerade ein Objekt am Greifer hängt, d.h. es wurde ein Objekt gegriffen. Gibt `true` zurück, wenn ein Objekt angehängt ist, andernfalls `false`.

Verwendung: `objectAttached = robot.isObjectAttached()`:

```
var objectAttached = robot.isObjectAttached()

if(objectAttached) {
    task.status = "Objekt am Greifer angehängt"
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
} else {
    task.status = "Kein Objekt am Greifer angehängt"
}
```

runRobotProgram

Ein natives Roboterprogramm ausführen und auf das Ergebnis warten. Im Gegensatz zum *Roboterprogramm ausführen* Task kann diese Funktion eine beliebige Anzahl von Argumenten anstelle von nur einem einzigen Wert pro Datentyp übergeben. Außerdem erzwingt die Funktion im Gegensatz zum genannten Task keinen Stillstand des Roboters. Es kann deshalb nicht garantiert werden, dass von Mikado keine Roboterbewegungen parallel ausgeführt werden. Ein Stillstand des Roboters muss bei Bedarf ggf. durch vorherigen Aufruf des *Warten bis zum Stillstand* Task sichergestellt werden. Alle übertragenen Posen beziehen sich auf das roboterfeste Koordinatensystem an der Basis des Roboters.

Verwendung: `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},
    type: "pose"
},
{
    name: "testabcd",
    value: dataStore.getJsObject("upperRight"),
    type: "pose"
}
]
var result = robot.runRobotProgram("test", args)
task.status = result
```

8.1.8 scene

Folgende Funktionen stehen zu Verfügung:

- *getSceneObjectDimensions*
- *setSceneObjectDimensions*
- *getSceneObjectPose*
- *setSceneObjectPose*
- *roiContainsPoint*

getSceneObjectDimensions

Gibt die Größe einer ROI oder die Größe des Platzhalters (Defaultgröße) bei einem Hindernis oder einer Kiste zurück. Die Größe eines Meshes/CADs wird nicht berücksichtigt.

Verwendung: `size = scene.getSceneObjectDimensions(Name)`

```
var size = scene.getSceneObjectDimensions("ROI1")
console.log("ROI1 ist "+size.x+" * "+size.y+" * "+size.z+" [m] groß")
```

setSceneObjectDimensions

Ändert die Größe einer ROI oder die Größe des Platzhalters (Defaultgröße) bei einem Hindernis oder einer Kiste zurück. Die Größe/Skalierung eines Meshes/CADs wird nicht verändert.

Verwendung: `scene.setSceneObjectDimensions(Name, valueX, valueY, valueZ)`:

```
var size = scene.getSceneObjectDimensions("ROI1")
scene.setSceneObjectDimensions("ROI1", size.x, size.y, 0.23)
```

getSceneObjectPose

Gibt die Pose eines Szenenobjekts zurück (in Meter und Radian).

Verwendung: `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

Setzt die Position eines Szenenobjekts.

Verwendung: `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

Prüfen, ob ein Punkt im dreidimensionalen Raum innerhalb einer bestimmten ROI liegt. `name` ist der Name der zu prüfenden ROI. Die Koordinaten `x`, `y` und `z` müssen in Metern als Einheit übergeben werden. Gibt `true` zurück, wenn der Punkt innerhalb der ROI liegt, andernfalls `false`.

Verwendung: `scene.roiContainsPoint(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

Das JavaScript-File Modul bietet Zugriff auf Textdateien. Folgende Funktionen stehen zur Verfügung:

- *readCsvAsDict*

readCsvAsDict

Die Funktion liest eine Textdatei im CSV-Format und erstellt ein Dictionary (key/values). Die erste Spalte wird als Key verwendet, die restliche Zeile ist als Liste verfügbar.

Die erste Zeile wird ignoriert da sie typischerweise Überschriften enthält.

Ein relativer Dateiname bezieht sich auf das Zellverzeichnis.

Verwendung: `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]
```

Im Beispiel wird eine Datei `test.csv` in einem konfigurierbaren Ordner (Mikado Variable `ausgabe_ordner`) gelesen. Der erste Parameter nach dem Namen wird in den Task Status ausgegeben.

8.1.10 program

Das JavaScript `program` Modul bietet Zugriff auf Mikado Programme. Folgende Funktionen stehen zur Verfügung:

- *save*

save

Das aktuelle Mikado-Programm speichern. Diese Funktion kann z.B. verwendet werden, um Änderungen an Ausgangswerten von Variablen während der Programmausführung automatisch zu speichern, so dass sie auch nach einem Neustart der Anwendung erhalten bleiben. Ein Fehler wird ausgelöst, wenn das Speichern fehlschlägt.

Verwendung: `program.save()`

8.1.11 lab

Das JavaScript-Lab bietet Zugriff auf selten genutzte, aber dennoch nützliche Funktionen. Folgende Funktionen stehen zu Verfügung:

- *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

Invertiert eine Pose.

Verwendung: `Pose = invertPose(Pose)`

```
var pose = dataStore.getJsObject('einePose')
var pose_inv = lab.invertPose(pose)
```

getPositionAverage

Berechnet den Mittelwert der x-y-z-Koordinaten aller Posen einer PoseList.

Verwendung: `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

Berechnet den euklidischen Abstand zwischen den Positionen zweier Posen.

Verwendung: `Double = getPoseDistanceTrans(Pose, Pose)`

```
var pose1 = dataStore.getJsObject('einePose')
var pose2 = dataStore.getJsObject('anderePose')
var Abstand = lab.getPoseDistanceTrans(pose1, pose2)
```

getPoseDistanceRot

Berechnet die relative Verdrehung zwischen den Orientierungen zweier Posen.

Verwendung: `Double = getPoseDistanceRot(Pose, Pose)`

```
var pose1 = dataStore.getJsObject('einePose')
var pose2 = dataStore.getJsObject('anderePose')
var Verdrehung = lab.getPoseDistanceRot(pose1, pose2)
```

getPoseDifference

Berechnet die Differenz zweier Posen nach folgendem Schema, wobei R_i jeweils den Rotationsanteil und t_i den Translationsanteil der Pose i beschreibt:

$$T = \begin{bmatrix} R_1 R_2^{-1} & t_1 - t_2 \\ 0 & 1 \end{bmatrix}.$$

Verwendung: `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

Berechnet die Winkelabweichung zwischen der z-Achse einer Pose mit dem Rotationsanteil R und der z-Achse des globalen Koordinatensystems:

$$\alpha = \cos^{-1}(Rz \cdot z) \, .$$

Standardmäßig wird die Winkelabweichung zur z-Achse berechnet. Durch den Zusatz eines zweiten Parameters kann der Operator auch dazu verwendet werden die Abweichung zur x- bzw. zur y-Achse zu berechnen.

Verwendung: `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

Schreibt einen String in eine Datei.

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

Gibt die aktuelle Zeit zurück.

Verwendung: `String = getCurrentDateTime()`

```
var dateTime = lab.getCurrentDateTime() // dateTime = <YYYY>-<MM>-<DD>T<hh>:<mm>:<ss>
```

getPoseAsJson

Konvertiert die gegebene Pose in das Json-Format, sodass sie beispielsweise in NxTreeEdit weiterverwendet werden kann.

Verwendung: `String = getPoseAsJson(Pose)`

```
var pose1 = dataStore.getJsObject('einePose')
var poseAsJson = lab.getPoseAsJson(pose1)
console.log("Json Darstellung:\n" + poseAsJson)
```

getTF

Berechnet die Transformation zwischen zwei Urdf-Objekten. Urdf-Objekten werden im *Zellgeometrie* definiert.

Verwendung: `Pose = getTF(String originFrameName, String goalFrameName)`

```
var tool0_zu_gripperTcp = lab.getTF('tool0', 'gripper_tcp')
dataStore.setJsObject("tool0_zu_gripperTcp", tool0_zu_gripperTcp)
```

sendRobotCommand

Sendet ein R3-Kommando an den angeschlossenen Roboter.

> **Warnung:** Funktioniert nur mit Mitsubishi Robotern.

Verwendung: `String = sendRobotCommand(String)`

```
var response = lab.sendRobotCommand('1;1;OVRD=25') // setze Geschwindigkeit auf 25%
```

getRobotVelocityScaling

Liest die aktuell verwendete Geschwindigkeits-Skalierung für die Bahnplanung des Roboters aus.

Verwendung: `Double = getRobotVelocityScaling()`

```
var velocityScaling = lab.getRobotVelocityScaling()
```

setRobotVelocityScaling

Setzt die Geschwindigkeits-Skalierung für die Bahnplanung des Roboters, siehe *Geschwindigkeitseinstellungen* für Details.

Verwendung: `setRobotVelocityScaling(Double)`

```
lab.setRobotVelocityScaling(0.4) // setze Geschwindigkeit-Skalierung auf 40%
```

getRobotAccelerationScaling

Liest die aktuell verwendete Beschleunigungs-Skalierung für die Bahnplanung des Roboters aus.

Verwendung: `Double = getRobotAccelerationScaling()`

```
var accScaling = lab.getRobotAccelerationScaling()
```

setRobotAccelerationScaling

Setzt die Beschleunigungs-Skalierung für die Bahnplanung des Roboters, siehe *Geschwindigkeitseinstellungen* für Details.

Verwendung: `setRobotAccelerationScaling(Double)`

```
lab.setRobotAccelerationScaling(0.4) // setze Beschleunigungs-Skalierung auf 40%
```

getCurrentGraspName

Liest den Namen des zuletzt verwendeten Griffs. Die Namen der Griffe werden in der *Griff-Inspektion* angezeigt und ergeben sich aus den *Greifposen* eines Produkts. Bei sprechender Benamung der Greifposen, lassen sich über den Griff-Namen Rückschlüsse über die Lage des Teils im Greifer ziehen.

Verwendung: `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

Liest die Posen-Repräsentation der Transformationsmatrix von angehängtem Teil zu Griffpunkt (siehe *Details* für Beschreibung der Lage des Griffpunkts) des zuletzt verwendeten Griffs.

Siehe auch *getCurrentGraspPieceToToolTransform* für die Transformation vom angehängten Teil zum Tool0.

Im folgenden Beispiel soll die Lage eines Teils im Greifer analysiert werden. Betrachtet wird die in untenstehender Abbildung dargestellte Situation. Eingezeichnet ist jeweils das Koordinatensystem (rot, grün, blau entspricht x, y, z) von Greifer-TCP und Teil. Für den Ablage-Prozess des Teils interessiert uns die Orientierung des Teils im Greifer. Im gezeigten Fall kann die z-Achse des Teils entweder in positive oder negative y-Richtung des Greifer-TCP zeigen. Dies ergibt sich aus der Definition des ‚gripper_tcp' (siehe *Mikado Key-Links*) und der für das Produkt (Teil) angelegten Griffe. In unserem Beispiel zeigen beide Achsen in entgegengesetzte Richtungen. Der folgende Beispielcode liefert das gewünschte Ergebnis.

Verwendung: `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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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
```

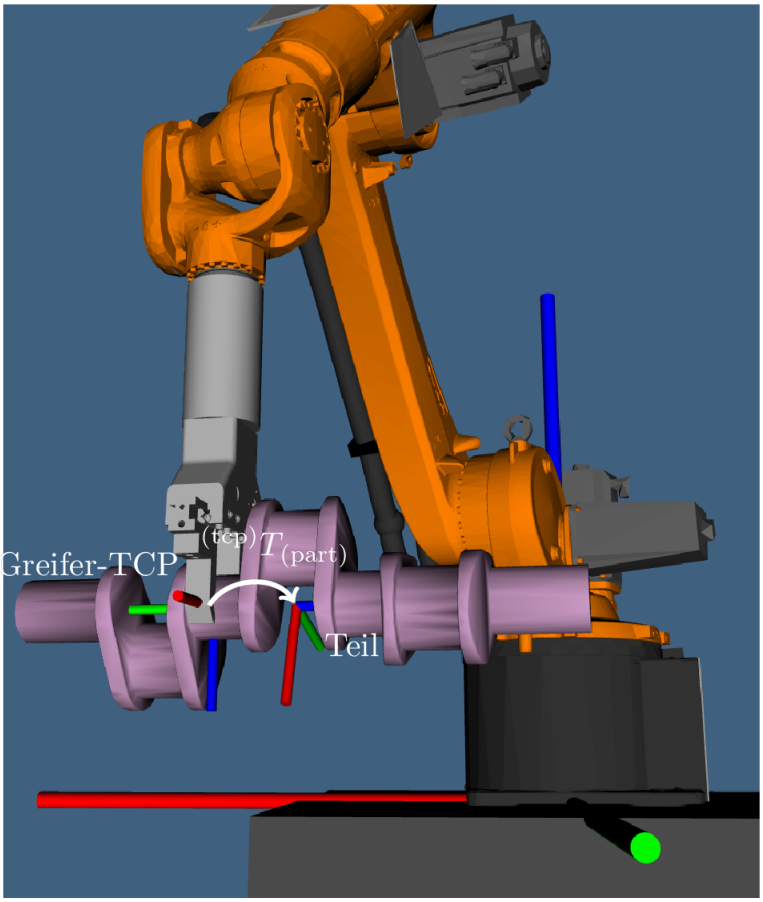


Abb. 2: Veranschaulichung des Beispiels zur Analyse des momentan gegriffenen Teils

Details

Folgende Skizze verdeutlicht die Lage des Griffpunkts:

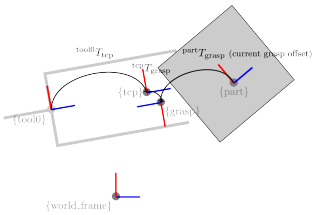


Abb. 3: Skizze zur Lage des Griffpunkts

Die Griffpunkt-Transformation ${}^{\text{part}}T_{\text{grasp}}$ beschreibt die relative Lage des angehängten Teils zu Griffpunkt. Hierbei ist zu beachten, dass zwischen Bezugssystem TCP und Bezugssystem grasp ein y-Flip ${}^{\text{tcp}}T_{\text{grasp}}$ existiert:

$$T = \begin{bmatrix} R(180°, y) & 0 \\ 0 & 1 \end{bmatrix}.$$

setCurrentGraspOffset

Überschreibt die Transformationsmatrix von angehängtem Teil zu Griffpunkt (siehe *Details*).

Dies kann nützlich sein, wenn die physikalische Position des Teils im Greifer von der modellierten Position abweicht. Diese Abweichung kann bspw. durch eine integrierte Zentrier-Vorrichtung oder Formschluss des verwendeten Greifers entstehen. Oder auch durch Ungenauigkeiten in der Objekterkennung und demzufolge im Greifprozess. Die tatsächliche Position des Teils im Greifer kann bspw. durch eine zusätzliche Kamera-Aufnahme bestimmt werden („Griffcheck“).

Mit Hilfe dieses Operators kann die Modelldarstellung aktualisiert werden und nachfolgende Tasks wie beispielsweise *Bewege Teil nach* ändern ihr Verhalten entsprechend.

Siehe auch *getCurrentGraspOffset* für die entsprechende Lese-Funktion.

Bemerkung: Dieser Operator hat nur einen Effekt, wenn sich gerade ein Teil im Greifer befindet.

Bemerkung: Die Kollisionsprüfung und Bahnplanung verwendet erst ab diesem JavaScriptTask die neue Position des Teils im Greifer. Befinden sich zum Zeitpunkt der Ausführung dieses Tasks noch Aufträge in der Roboter-Action-Queue, so wurden diese mit der alten Teilposition geplant. Die korrigierte Teilposition wird nicht rückwirkend auf bereits geplante Fahrten berücksichtigt.

Verwendung: `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 unten stehender Abbildung besitzt der Greifer eine Zentrierung durch Formschluss, d.h. es kann angenommen werden, dass sich jedes gegriffene Teil an den Greiferfingern ausrichtet (linkes Bild). Da mit dem durchgeführten Griff das Teil jedoch leicht verkippt angefahren wurde, ist diese Verkippung auch in der Modelldarstellung berücksichtigt (mittleres Bild). Dies führt zu einer nennenswerten Abweichung zwischen Modell und tatsächlicher Szene. Um diese Abweichung zu korrigieren, kann durch Anwendung des obigen Beispiel-Codes die Modelldarstellung aktualisiert werden (siehe rechtes Bild).

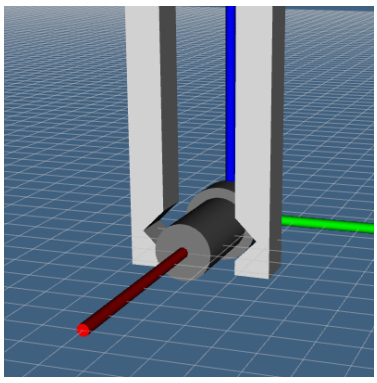


Abb. 4: Formschluss ergibt sich aus Greifer- und Teilgeometrie

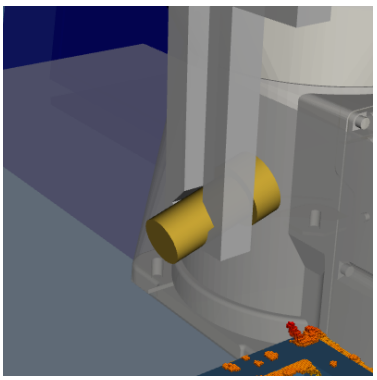


Abb. 5: Ausgangssituation

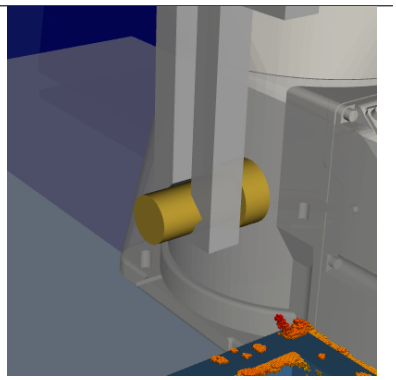


Abb. 6: Situation nach Anwendung des Operators `setCurrentGraspOffset`

Ist die Position des gegriffenen Teils im Weltkoordinatensystem beispielsweise durch die Verwendung des Tasks *Finde Objekt* bekannt, kann die Position des angehängten Teils im Greifer mittels folgender Transformation nachjustiert werden:

$$ {}^{\text{match}}T_{\text{grasp}} = \left(\underbrace{{}^{\text{tcp}}T_{\text{world}}}_{\text{urdf}} \quad \underbrace{{}^{\text{world}}T_{\text{match}}}_{\text{findObject}} \right)^{(-1)} \quad \underbrace{{}^{\text{tcp}}T_{\text{grasp}}}_{180^\circ \text{ flip}} . $$

Folgendes JavaScript Beispiel demonstriert das Nachjustieren des gegriffenen Teils anhand der Rückgabe des *Finde Objekt* Tasks:

```
// 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

Berechnet die Transformation zwischen dem momentan gegriffenen Teil und tool0.

Siehe auch *getCurrentGraspOffset* für die Transformation vom angehängten Teil zum TCP.

Verwendung: `Pose = getCurrentGraspPieceToToolTransform()`

```
var pieceToToolTF = lab.getCurrentGraspPieceToToolTransform()
// Pose könnte per Feldbus-Schnittstelle an SPS übertragen werden
```

mult

Multipliziert die Transformationsdarstellung einer Pose mit der Transformationsdarstellung einer Pose oder einem 3D-Vector.

Verwendung: `[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)
```

Folgende Abbildung zeigt ein Beispiel zur Analyse eines gefunden Objekts. Eingezeichnet ist jeweils das Koordinatensystem (rot, grün, blau entspricht x, y, z) von Welt und Teil. Der folgende Beispielcode gibt an, ob z-Achse des Teils und x-Achse der Welt in die gleiche Richtung zeigen (im gezeigten Beispiel, zeigen beide Achsen in die selbe Richtung).

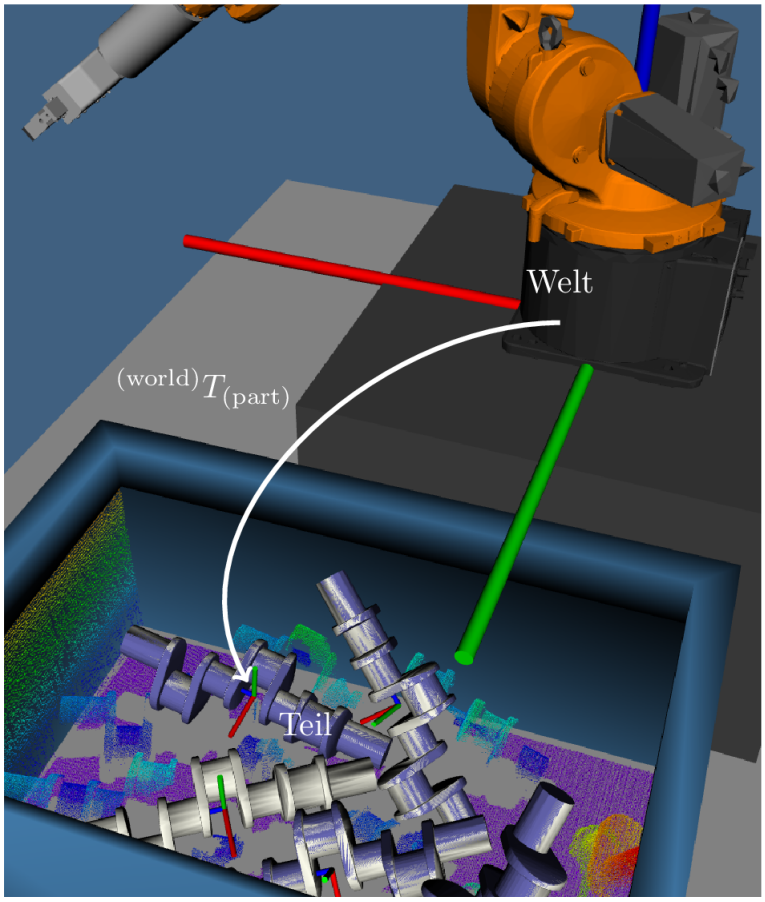


Abb. 7: Veranschaulichung des Beispiels zur Analyse eines gefundenen Teils

Beispiel: Teile-Pose untersuchen

```
// 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

Löscht alle gefundenen Objekte aus der Kollisionswelt. Nach dem Aufruf dieses Operators wird trotz aktivierter „Vorherige Matches wiederverwenden“-Flag (siehe auch *Halcon-spezifische Objekterkennungsparameter*) eine komplett neue Suche gestartet.

Verwendung: `Bool = clearAllRecognizedObjects()`

```
var success = lab.clearAllRecognizedObjects()
```

updateOctomap

Berechnet und veröffentlicht die Octomap aus der aktuellen Punktwolke. Sie kann damit z.B. für die Kollisionsprüfung bei der Ablage verwendet werden. Der optionale Parameter `waitUntilUpdateReceived` gibt an, ob der Aufruf blockieren soll.

Verwendung: `Bool = updateOctomap(Bool waitUntilUpdateReceived=True)`

```
var success = lab.updateOctomap()
```

setSceneObjectDimension

Setzt die Default-Größe eines Szenen-Objekts. Szenen-Objekte können *Hindernis*, *Region of Interest (ROI)* oder *Kiste* sein.

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

Es kann ein ROS Service Call zu einem Service vom Typ `mikado_extension_msgs::RunExtensionBasic` aufgerufen werden und es wird auf das Ergebnis gewartet. Der Service kann beispielsweise in Python implementiert werden und projektspezifische Sonderfunktionen implementieren.

Verwendung: `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
```

Eine Beispielvorlage für die Implementierung eines projektspezifischen ROS Services ist im Kapitel *Eigenen ROS Service erstellen* zu finden.

setGetJsonFromCamera

Mit dieser Funktion kann direkt in den Variablenbaum der nxLib (nxTreeEdit) einer Ensenso-Kamera geschrieben und/oder gelesen werden. Sie kann z.B. dazu verwendet werden Parameter auszulesen, zu verändern oder über den `/command` Knoten auch direkt Aktionen der NxLib auszuführen.

Äquivalent zur Prozedur im Halcon-Task *ensenso_set_get_json*.

Verwendung: `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

Folgende Funktionen stehen zu Verfügung:

- *getXyzMinMax*

getXyzMinMax

Diese Funktion erlaubt den Zugriff auf Eigenschaften der Punktewolke. Sie liefert die minimalen und maximalen X-, Y- und Z-Werte sowie die Anzahl der Punkte. Ist eine ROI aktiv, bezieht sich der Ausgabe auf die angewählte ROI.

Die Funktion kann unter anderem dazu genutzt werden, um zu kontrollieren ob eine Kiste vollständig entleert wurde. Ein anderer Anwendungsfall wäre, Teile abhängig vom Befüllungsgrad zurück in die Kiste zu legen, indem man eine Abwurfhöhe abhängig von der ermittelten, maximalen Höhe errechnet.

Gibt ein Objekt mit folgenden Keys zurück:

- `x_min` (`Double`): Minimale X-Koordinate der Punktewolke im Welt-Koordinatensystem [m]
- `x_max` (`Double`): Maximale X-Koordinate der Punktewolke im Welt-Koordinatensystem [m]
- `y_min` (`Double`): Minimale Y-Koordinate der Punktewolke im Welt-Koordinatensystem [m]
- `y_max` (`Double`): Maximale Y-Koordinate der Punktewolke im Welt-Koordinatensystem [m]
- `z_min` (`Double`): Minimale Z-Koordinate der Punktewolke im Welt-Koordinatensystem [m]
- `z_max` (`Double`): Maximale Z-Koordinate der Punktewolke im Welt-Koordinatensystem [m]
- `number_points` (`Integer`): Anzahl der Punkte der Punktwolke (ggf. in der aktiven ROI)

Verwendung: `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
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
if (punkteInROI < 100) {          // Durch Rauschen kann es vorkommen, dass sich␣
↪vereinzelt Punkte in der ROI befinden.
    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)
```

Das obige Beispiel überprüft ob sich Punkte in der aktiven ROI befinden. Falls ja, soll die Ablageposition als belegt erachtet werden, falls nein, ist die Ablageposition frei.

8.2 Halcon Script Task

Die Halcon Bildverarbeitungsbibliothek der Firma MVTec Software GmbH ermöglicht es, eigene Bildverarbeitungsalgorithmen in einer Halcon-spezifischen Script-Sprache zu entwickeln. Soll eine Applikation um benutzerdefinierte Halcon Scripte erweitert werden, kann eine einfache Integration mithilfe des Tasks *Halcon Script ausführen* erfolgen. Hierzu entwickeln Sie Ihre eigene Algorithmen in der Entwicklungsumgebung von Halcon, HDevelop. Eine Erweiterung bietet hier denselben Zugriff auf das Mikado System wie später im Task. Zum Entwickeln benötigen Sie eine separate Entwicklerlizenz für HDevelop, momentan in Version 13. Funktioniert das so entwickelte Programm zufriedenstellend, kann es nun über den Task *Halcon Script ausführen* in einen Mikado Ablauf integriert werden.

Möchten Sie eigene Algorithmen für die Objekterkennung mittels Halcon umsetzen, lesen Sie hierzu Sektion *Benutzerdefinierte Objekterkennung mit Halcon*.

Folgende HDevelop-Operatoren werden von der Mikado-Extension für HDevelop zur Verfügung gestellt:

8.2.1 mikado_get_2d_image

Auslesen der letzten Aufnahme einer in Mikado eingebundenen Ensenso Kamera. Verwenden Sie Tasks wie ‚*Kamera Konfiguration laden*' und ‚*Punktwolke aufnehmen*' um die Bildaufnahme auszuführen und anschließend diesen Halcon-Operator, um auf die Aufnahme zuzugreifen.

Ist eine monokulare Kamera an Mikado angeschlossen, kann über diesen Befehl ein Bild eingezogen werden. Beachten Sie, dass der Funktionsaufruf lediglich bei monokularen Kameras auch gleichzeitig eine neue Aufnahme auslöst. Verwenden Sie den Parameter ‚CameraTarget', um die Seriennummer der Kamera zu übermitteln und somit das rohe oder rektifizierte Bild anzufordern.

Die folgende Tabelle beschreibt die einzelnen Parameter des Operators.

Parameter	Parameter Name	Beschreibung	Beispiel
Ausgabe	Image2D	In dieser Variablen wird das Bild nach Aufruf des Operators abgelegt	OutputImage
Eingabe	CameraName	Name der Kamera	'Ensenso_Camera_0'
	CameraTarget	Rechte oder linke Kamera (Ensenso). Seriennummer für Monokameras.	'left', 'right', '4101230815'
	ImageType	Bildtyp	'raw', 'rectified', 'withOverlay'

Hinweis: Beispiele zur Verwendung des HDevelop-Operators:

- `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')`

Warnung: Dieser Operator löst keine Aufnahme aus. Sie müssen die Aufnahme vorher z.B. mit dem Task *Punktwolke aufnehmen* auslösen.

8.2.2 mikado_publish_2d_image

Veröffentlicht ein 2D Bild zur Anzeige in Mikado. Das Bild, ob Grauwert oder RGB(A), kann dann in der Mikado Oberfläche in der rechten Hälfte bei der Live-Kameraansicht betrachtet werden indem es aus der Liste ausgewählt wird.

Die folgende Tabelle beschreibt die einzelnen Parameter des Operators.

Parameter	Parameter Name	Beschreibung	Beispiel
Eingabe	Image2D	Anzuzeigendes Bild	Image2D

Hinweis: Beispiele zur Verwendung des HDevelop-Operators:

- `mikado_publish_2d_image (Image2D)`

8.2.3 mikado_get_point_cloud

Auslesen der letzten Punktwolke einer in Mikado eingebundenen Ensenso Kamera. Verwenden Sie Tasks wie ‚*Kamera Konfiguration laden*' und ‚*Punktwolke aufnehmen*' um die Bildaufnahme auszuführen und anschließend diesen Halcon-Operator, um auf die Aufnahme zuzugreifen. Da Halcon nicht gut mit NaN Werten zurecht kommt, werden diese auf Null gesetzt. Mit der zusätzlichen Ausgabe Region können diese gefiltert werden.

Die folgende Tabelle beschreibt die einzelnen Parameter des Operators.

Parameter	Parameter Name	Beschreibung	Beispiel
Ausgabe	ImageX	Die X-Komponente der Punktwolke	OutputX
	ImageY	Die Y-Komponente der Punktwolke	OutputY
	ImageZ	Die Z-Komponente der Punktwolke	OutputZ
	Region	Gültige Punkte der Punktwolke	ValidRegion
Eingabe	CameraName	Name der Kamera	'Ensenso_Camera_0'

Hinweis: Beispiele zur Verwendung des HDevelop-Operators:

- `mikado_get_point_cloud (ImageX, ImageY, ImageZ, ValidRegion, 'Ensenso_Camera_0')`

Warnung: Dieser Operator löst keine Aufnahme aus. Sie müssen die Aufnahme vorher, z.B. mittels des Tasks *Punktwolke aufnehmen* auslösen.

8.2.4 mikado_write_variable

Den Wert einer Mikado-Variablen setzen.

Die folgende Tabelle beschreibt die einzelnen Parameter des Operators.

Parameter	Parameter Name	Beschreibung
Eingabe	VariableName	Name der Variable, die verändert werden soll
	VariableValueType	Typ der Variablen
	VariableStringValue	Wert der Variablen, falls der Variablentyp String oder StringList ist
	VariableNumericValue	Wert der Variablen, falls der Variablentyp ein Integer, Bool oder Double ist

Alle Mikado-Variablentypen werden in der folgenden Tabelle aufgelistet. Die Zuordnung, welcher Wert bei welchem Variablentyp verwendet wird, ist in dieser Tabelle zusammen mit Beispielen aufgeführt.

Variablentyp	Schreiben möglich	Eingabe über Variable-StringValue / Variable-NumericValue	Beispiele
Integer	✓	Numeric	1 42
Double	✓	Numeric	1.1 3.14
String	✓	String	'meinWert'
StringList	✓	String	['StringListWert1', 'StringListWert2']
JointPosition	–		
Pose	✓	Numeric	Halcon Pose, wird intern umgerechnet in Radiant und Meter
PoseList	✓	Numeric	Tuple von Halcon Posen
Bool	✓	Numeric	0 1
JScript	–		

Hinweis: Beispiele zur Verwendung des HDevelop-Operators:

- `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

Den Wert einer Mikado-Variablen auslesen.

Die folgende Tabelle beschreibt die einzelnen Parameter des Operators.

Parameter	Parameter Name	Beschreibung	Beispiele
Eingabe	VariableName	Name der Mikado Variable, die gelesen werden soll	'VarName'
Ausgabe	VariableValueType	Typ der gelesenen Variable	'String', 'Integer', …
	VariableStringValue	Wert der Variablen als String	'meinWert', '42', ['StringListWert1', 'StringListWert2']

Fortsetzung auf der nächsten Seite

Tab. 7 – Fortsetzung der vorherigen Seite

Parameter	Parameter Name	Beschreibung	Beispiele
	VariableNumericValue	Wert der Variable, sofern die Mikado Variable einen numerischen Typ hatte	42, 3.14, 0

Alle Mikado-Variablentypen werden in der folgenden Tabelle aufgelistet. Die Zuordnung, welcher Wert bei welchem Variablentyp beschrieben wird, ist in dieser Tabelle zusammen mit Beispielen aufgeführt.

Variablentyp	Ausgabe über VariableStringValue / VariableNumericValue	Beispiele
Integer	Numeric	1 42
Double	Numeric	1.1 3.14
String	String	'meinWert'
StringList	String	['StringListWert_1', 'StringListWert2']
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 1.54\nxyz: 0.10 0.20 0.30 rpy: 3.14 0.00 0.00'
Bool	Numeric	0 1

Hinweis: Beispiele zur Verwendung des HDevelop-Operators:

- `mikado_read_variable ('MeineMikadoVariable', VariableValueType, VariableStringValue, VariableNumericValue)`
- `mikado_read_variable ('ProduktName', VariableValueType, VariableStringValue, VariableNumericValue)`

8.2.6 ensenso_set_get_json

Mit dieser Prozedur kann direkt in den Variablenbaum der nxLib einer Ensenso-Kamera geschrieben und/oder gelesen werden. Sie kann zB dazu verwendet werden Parameter auszulesen, zu verändern oder über den `/command` Knoten auch direkt Aktionen der NxLib auszuführen. Diese Prozedur ist äquivalent zum Funktionsaufruf im JavaScript-Task *setGetJsonFromCamera*.

Die folgende Tabelle beschreibt die einzelnen Parameter des Operators.

Parameter	Parameter Name	Beschreibung	Beispiele
Eingabe	CameraName	Name der Kamera	'Ensenso_Camera_0'
	SetJson	Json Teilbaum der geschrieben werden soll. Ggf leer.	'{}', '{„Execute“:{„Aufnahme“: {„Command“: „Capture“}}}'

Fortsetzung auf der nächsten Seite

Tab. 9 – Fortsetzung der vorherigen Seite

Parameter	Parameter Name	Beschreibung	Beispiele
	GetJsonPath	Pfad dessen Json gelesen gelesen werden soll	'/Version‘, '', '/Execute/Aufnahme/Result'
Ausgabe	SetSuccess	War das Schreiben/Setzen erfolgreich?	'true', 'false'
	SetError	Fehlermeldung beim Setzen	'', 'CouldNotInterpretJsonText', …
	GetError	Fehlermeldung beim Lesen	'', 'ItemInexistent'

Hinweis: Beispiele zur Verwendung des HDevelop-Operators:

- `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)`

Quellcode 1: Beispiel: Auslesen der Kameraparameter für N30, N35, N40, N45 und 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 Benutzerdefinierte Objekterkennung mit Halcon

Mikado bietet eine Schnittstelle zu Halcon (Informationen zu Halcon s. *Halcon Script Task*), um eigene Algorithmen zur Objekterkennung einzubinden. Um Ihre Skripte einzubinden müssen Sie

- Ein .hdev Skript erstellen, welches entsprechende Prozeduren beinhaltet
- Die object_recognition.yaml (Informationen zu Konfigurationsdateien in Sektion *Konfigurationsdateien*) erweitern, um Mikado mitzuteilen für welche Produkte welches Skript aufgerufen werden soll

8.3.1 Hdev-Skript

Voraussetzung für Ihre eigene Objekterkennung ist zuallererst ein passendes Hdev-Skript, das in Mikado eingebunden werden kann. Eine Installation von HDevelop direkt auf dem Mikado-Controller ist empfehlenswert, um direkt auf dem Controller zu testen und die Mikado-Extension verwenden zu können.

Definieren Sie mit der Entwicklungsumgebung für Halcon ein neues Programm mit folgenden Prozeduren:

- `load_model`
  - Eingabe-Steuerparameter (Produkteigenschaften)
  - *ModelName*: Produktname des zu ladenden Produkts.
  - *MeshFilename*: Dateipfad der CAD-Datei des Produkts.
  - *RecognitionMeshFilename* (optional): Dateipfad der CAD-Datei des Produkts, die für die Objekterkennung verwendet werden soll. Wert ist leer, wenn nicht angegeben.
  - *ParamNames* (optional): Liste an Parameternamen, die in der object_recognition.yaml für dieses Produkt angegeben sind.
  - *ParamValues* (optional): Liste an Parameterwerten, die in der object_recognition.yaml für dieses Produkt angegeben sind.
  - Minimal-Beispiel:

```
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 ()
```

Hinweis: Mithilfe von `global tuple` können Tupel deklariert werden, die in anderen Prozeduren wie z.B. der `find_objects`-Prozedur ebenfalls verfügbar sind.

- `find_objects`
  - Eingabe-Steuerparameter (Produkteigenschaften)
  - *ModelNames*: Produktname des zu erkennenden Produkts.
  - *MinimalScores*: Mindest-Score des Produkts.
  - *MaxTotalMatches*: Erwartete Anzahl an Funden (Summe aller Produktfunde).

- Ausgabe-Steuerparameter (Rückgabewerte an Mikado)
- *RecognizedModels*: Produktnamen der Funde. Bleibt diese Liste leer, so wird standardmäßig das erste Produkt für alle Fundposen verwendet.
- *Poses*: Erkannte Objektposen
- *Scores*: Score-Werte der einzelnen Objekte
- Minimal-Beispiel:

  Im folgenden Beispiel wird die Funktion *mikado_get_point_cloud* verwendet, um die Punktewolke der letzten Aufnahme abzurufen und für die Objektsuche zu verwenden.

```
* 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 Anpassen der Konfigurationsdatei

Die Konfigurationsdatei object_recognition.yaml enthält die Parametrierung für die Objekterkennung. An dieser Stelle können Sie auch Ihr benutzerdefiniertes Hdev-Skript einbinden.

1. Beenden Sie die Mikado-Oberfläche.
2. Die Datei ‚object_recognition.yaml' im Verzeichnis ‚config' öffnen
3. Suchen Sie die Sektion `models:` und erstellen Sie für die Produkte mit eigener Objekterkennung Einträge in folgender 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. Mikado starten.

Hinweis: Sie können prüfen, ob Ihre Konfiguration korrekt ist, indem Sie Ihr Produkt in der Mikado-Oberfläche laden und danach in die Logs sehen. Dort finden Sie einen Info-Eintrag: „[HDevEngine node] Request received to load model for object recognition via halcon script: [..]“.

8.4 Eigene Seiten mit QML

Die Benutzeroberfläche von Mikado ist in QML programmiert und kann leicht um zwei kundenspezifische Seiten erweitert werden. Beispielsweise wird auf der ersten Seite dem Bediener ein anlagenspezifischer Status angezeigt. Auf einer weiteren Seite können detailiertere Informationen ergänzt oder auch Parameter eingestellt werden. Diese zweite Seite ist, falls aktiviert, durch das Passwort geschützt.

8.4.1 Konfiguration

Im Ordner `qml` finden Sie die Datei `pages.yaml` in der eingerichtet werden kann, welche zwei Dateien als Seiten angezeigt werden sollen. Nach einem Neustart von Mikado tauchen diese nun zwischen *Willkommen* und *Status* im *Run*-Modus auf. Sie können diese Dateien mit einem beliebigen Texteditor bearbeiten, vorinstalliert ist auf dem Controller *kate*. Wenn Sie in der Datei `config/mikado.yaml` das Parameter `debug_qml: true` setzen, erscheint in der rechten Bildhälfte eine weitere Seite auf der Sie ihren Code live neu laden können. So können, idealerweise auf einem zweiten Monitor, bequem und schnell Anpassungen an den Seiten vorgenommen werden.

8.4.2 Einführung in QML

QML ist eine deklarative Sprache ganz ähnlich zu HTML, analog zu diesem kann auch JavaScript für dynamische Aspekte verwendet werden. Beginnen wir mit einem einfachen Beispiel:

```
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"
        }
    }
}
```

Hier wird zuerst das Grundpaket für QtQuick geladen. Es stellt einige Basics wie Rechtecke (`Rectangle`), Textanzeigen (`Text`) sowie Bilder (`Image`) zur Verfügung. Anschließend beginnt die Objekthierarchie mit einem Minimalelement (`Item`), dieses wird in der Mikado-Oberfläche automatisch an das Fenster angepasst und benötigt daher keine Größenangabe. In diesem befindet sich ein blaues Rechteck mit abgerundeten Ecken (`radius: 20`). Interessant ist hier die relative Positionierung in der Mitte der Seite über einen `anchor`. In diesem Rechteck befindet sich wiederum ein weißer Text.

Etwas spannender wird es wenn wir eine Schaltfläche (Button) ergänzen und beim Anklicken eine Sequenz auslösen:

```
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
   }
}
```

Hier wird nach einem Klick (`onClicked`) die JavaScript-Funktion `startSequence` ausgeführt. Diese greift über das `core` Objekt auf Mikado zu und versucht das Programm auf die Sequenz namens `Test` zu wechseln. Existiert diese nicht, wird ein Hinweis in die Konsole (und die Logs) ausgegeben. Ansonsten wird der Mikado Programmablauf aktiviert.

Einige weitere vordefinierte Elemente finden Sie in den folgenden Auszügen.

Mikado-Variable anzeigen:

```
VariableLabel {
  variable: "TeilZaehler"
}
```

Mikado-Variable im Textformat anpassen:

```
VariableTextField {
  id: varTextField
  variable: "Zeit"
  setAsInitial: true
}
```

Ansonsten können auch alle weiteren Bausteine aus zB Qt Quick Controls 2 verwendet werden.

8.4.3 Weiterführende Informationen

Eine sehr umfangreiche und verständliche Einführung in QML bietet das QML-Buch ‚Qt5 Cadaques' unter https://qmlbook.github.io sowie die offizielle Dokumentation zu QML unter https://doc.qt.io/qt-5/qmlreference.html . Wir verwenden die Qt Quick Controls 2 deren Dokumentation man unter https://doc.qt.io/qt-5/qtquickcontrols-index.html findet.

8.5 Externe Ansteuerung mit ROS/Python

Sowohl für die externe Ansteuerung durch eine SPS oder ein Leitsystem als auch für die Integration spezieller Hardwarekomponenten wie individuelle Greifer, Schwenkeinheiten, … bietet Mikado eine ROS-Schnittstelle. Auf diese API kann mit verschiedenen Sprachen zugegriffen werden, im folgenden wird Python verwendet da es im Allgemeinen kompakten und verständlichen Quelltext ergibt. Falls Sie lieber mit C, C++, C# oder node.js arbeiten wollen, fragen Sie uns bitte direkt nach einer Anleitung und Beispielen.

8.5.1 Beispielprogramm

Beginnen wir mit einem einfachen Beispielprogramm:

```
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 diesem Beispiel wird zuerst die API für die Kommunikation mit Mikado Control geladen (1). Anschließend wird in einem Beispiel erläutert wie über einen *ServiceCall*, also einen Funktionsaufruf, eine Variable aus Mikado ausgelesen wird. Dazu wird zuerst ein Funktionsobjekt definiert (5) um dann dessen Funktion aufzurufen (6). Der Rückgabewert wird in `max_ansicht` gespeichert. Da Variablen in Mikado verschiedene Typen haben können, ist dieser im Format eines Sammelobjekts. Über die Eigenschaft `value_ints` kann man nun beispielsweise auf ein Array aller Integer Rückgabewerte zugreifen.

Im zweiten Abschnitt starten wir eine Mikado Sequenz namens `Init` (ein Name der im Mikado Programm vergeben wurde). Dazu wird wieder ein Funktionsobjekt erzeugt (10) und dieses anschließend aufgerufen. Dabei kann entweder die ausführliche Variante (11) mit benannten Argumenten oder die kompakte Variante (14) verwendet werden.

8.5.2 Mikado ROS Services

Bei einem laufenden Mikado kann die ROS-Service-Schnittstelle bequem auf der Kommandozeile inspiziert und getestet werden. `rosservice list` gibt die verfügbaren Services aus, `rosservice info /run` gibt beispielsweise die Informationen für die Funktion zum Starten des Mikado-Ablaufs aus und mit `rosservice call /run ...` kann diese direkt aufgerufen werden. Drücken Sie doppelt die Tab (Tabulator) Taste um die Liste der Argumente vervollständigt zu bekommen. Im Folgenden sind die wichtigsten Befehle aufgeführt:

Mikado

`mikado_msgs.`**`pause`**`()`

Pausiert das momentan laufende Mikado Programm nach Beendigung des momentan laufenden Tasks. Die Funktion ist asynchron, d.h. nach Beenden der Funktion kann sich zB der Roboter noch bewegen. Um sicherzustellen, dass Mikado pausiert wurde, verwenden Sie bitte das `/engine_state` Topic.

`mikado_msgs.`**`read_variable`**`()`

Liest eine Mikado Variable und gibt deren Wert zurück. Aufruf in Code: rospy.ServiceProxy(‚read_variable‘, ReadVariable)

Parameter

`name` (*`string`*) – Name der Variable

Rückgabe

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)

Parameter

`step` (*`bool`*) – Nach einem Schritt pausieren (Einzelschritt)

Startet die Ausführung am Anfang des Mikado Programms oder an der momentanen Position des Programmzeigers. Das Argument `step` gibt an, ob nur der nächste Task ausgeführt werden soll oder der Automatikmodus gestartet werden soll.

`mikado_msgs.`**`run_sequence`**`()`

Aufruf in Code: rospy.ServiceProxy(‚run_sequence‘, RunSequence)

Parameter

- **`name`** (*`string`*) – Name der auszuführenden Sequenz
- **`check_existence_only`** (*`bool`*) – die Funktion endet mit `success = true` falls die Sequenz existiert
- **`stop_after_first`** (*`bool`*) – Führt nur den ersten Task in der gewünschten Sequenz aus, kann mit `run(step=true)` dann in Einzelschritten weiter ausgeführt werden

Startet die Programmausführung mit einer gewünschten Sequenz `name`.

mikado_msgs.**stop**()

Aufruf in Code: rospy.ServiceProxy(‚stop‘, Stop)

Beendet die Programmausführung, setzt alle Variablen auf ihren Initialwert zurück und entfernt dynamische Daten. Dies beinhaltet Punktwolken, Objektfunde, Griffe, Positionen von Szenenobjekten . . . Aufruf in Code: rospy.ServiceProxy(‚stop‘, Stop)

mikado_msgs.**write_variable**()

Aufruf in Code: rospy.ServiceProxy(‚write_variable‘, WriteVariable)

Parameter

- **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[]*) – Setzt den Wert einer Mikado Variable, die den angegebenen `value_type` haben muss. Der Wert muss in den entsprechenden Feldern stehen. `JointPosition` werden aus `value_floats` gelesen, `JScript` (JavaScript) aus `value_strings`.

Ensenso

isys_object_recognition_msgs.**calibrate**()

/cameras/Ensenso_Camera_0/calibrate

Parameter

- **store** (*bool*) – Speichern der Kalibrierung in der Kamera
- **calibration_setup** (*string*) – `Moving` oder `Fixed`, ist die Kamera beweglich (am Roboter befestigt) oder an einer festen Position in der Zelle

Führt die Kamerakalibrierung mit den Aufnahmen, die mit `capture_for_calibration` aufgenommen wurden, durch.

isys_object_recognition_msgs.**calibrate_workspace**()

/cameras/Ensenso_Camera_0/calibrate_workspace

Parameter
defined_pose_transform (*Pose*) –

Setzt das Koordinatensystem der Kamera basierend auf einer Kalibrierplatte, so dass sich das Pattern an der Pose `defined_pose_transform` befindet. (Nur für Mikado Select sinnvoll, ersetzt eine existierende Hand-Auge-Kalibrierung)

isys_object_recognition_msgs.**capture_for_calibration**()

/cameras/Ensenso_Camera_0/capture_for_calibration

Parameter

- **reset** (*bool*) – Bisherige Aufnahmen verwerfen und von Vorne beginnen
- **tool0_pose** (*Pose*) – Position des Roboterflansches (`tool0`)

Macht eine Aufnahme und speichert die Positionen von Pattern und Roboter für die anschließende Kalibrierung mit `calibrate`.

isys_object_recognition_msgs.**estimate_pattern_pose**()

/cameras/Ensenso_Camera_0/estimate_pattern_pose

Rückgabe

success (bool): war ein Pattern sichtbar und konnte seine Position bestimmt werden

pattern_pose (geometry_msgs.Pose):

Nimmt ein Kamerabild auf, sucht darin nach einem Ensenso-Kalibrierpattern und gibt dessen Position als `pattern_pose` zurück.

isys_object_recognition_msgs.**get_captured_images**()

/cameras/Ensenso_Camera_0/get_captured_images

Parameter

- **left_image_type** (*int*) –
- **right_image_type** (*int*) –

Rückgabe

success (bool)

left_image, right_image (sensor_msgs.Image)

Gibt ein Bild von der letzten Aufnahme zurück, `type` kann dabei 0 (`NO_IMAGE`), 1 (`RAW_IMAGE`), 2 (`RECTIFIED_IMAGE`) oder 3 (`OVERLAY_IMAGE`) sein und bezieht sich auf die analogen Stufen der Ensenso Bildverarbeitung.

isys_object_recognition_msgs.**get_point_cloud**()

/cameras/Ensenso_Camera_0/get_point_cloud

Rückgabe

point_clouds (sensor_msgs.PointCloud2) Liste von Punktwolken

Gibt die letzte aufgenommene Punktwolke zurück. Bei Mehrfachaufnahmen werden diese als einzelne Punktwolken zurück gegeben.

isys_object_recognition_msgs.**is_calibration_pattern_visible**()

/cameras/Ensenso_Camera_0/is_calibration_pattern_visible

Parameter

- **calibration_setup** (*string*) – `Moving` oder `Fixed`, ist die Kamera beweglich (am Roboter befestigt) oder an einer festen Position in der Zelle
- **tool0_pose** (*Pose*) – Pose des Roboterflansches

Überprüft ob bei gegebener Pose das Pattern im Sichtfeld der Kamera läge. Dazu muss mindestens eine Vorkalibrierung mittels `run_precalibration` oder eine vollständige Kalibrierung erfolgt sein.

isys_object_recognition_msgs.**load_camera_config**()

/cameras/Ensenso_Camera_0/load_camera_config

Parameter

- **filename** (*string*) –
- **load_capture_parameters** (*bool*) –
- **load_link** (*bool*) –

Lädt eine in NxView gespeicherte Kamerakonfiguration im JSON-Format in die aktuelle Kamera. Es können nur die Aufnahmeparameter (`load_capture_parameters`), nur die Transformation/Link (`load_link`) oder beide geladen werden.

isys_object_recognition_msgs.**recalibrate**()

/cameras/Ensenso_Camera_0/recalibrate

Führt eine Schätzung und Neukalibrierung des Vergenzwinkels durch. Dazu muss ein Kalibrierpattern sichtbar sein.

isys_object_recognition_msgs.**run_precalibration**()

/cameras/Ensenso_Camera_0/run_precalibration

Parameter

calibration_setup (*string*) – `Moving` oder `Fixed`, ist die Kamera beweglich (am Roboter befestigt) oder an einer festen Position in der Zelle

Führt eine erste grobe Hand-Auge-Kalibrierung durch. Nach dieser kann `is_calibration_pattern_visible` verwendet werden um beispielsweise bei zufälligen Kalibrierpositionen die Sichtbarkeit zu überprüfen.

isys_object_recognition_msgs.**save_camera_config**()

/cameras/Ensenso_Camera_0/save_camera_config

Parameter

filename (*string*) –

Schreibt die momentane Kamerakonfiguration in eine JSON-Datei.

isys_object_recognition_msgs.**save_raw_data**()

/cameras/Ensenso_Camera_0/save_raw_data

Parameter

- **path** (*string*) – Zielordner, relativ zum `Home` Verzeichnis (Persönlicher Ordner in Dolphin)
- **prefix** (*string*) – Dem Dateinamen wird ein Prefix voran gestellt
- **rectified** (*bool*) – Falls das Bild in rektifizierter Version gespeichert werden soll, diesen Parameter auf true setzen

Speichert die aktuelle Aufnahme, der entstehende Satz Bilder kann beispielsweise für eine Filecam-Kamera verwendet werden.

isys_object_recognition_msgs.**set_capture_mode**()

/cameras/Ensenso_Camera_0/set_capture_mode

Parameter

- **front_light** (*bool*) –
- **projector** (*bool*) –

Für 2D Aufnahmen mit der Ensenso-Kamera kann der Projektor aus und ggf das Frontlight angeschaltet werden. Muss für 3D-Aufnahmen zurück gesetzt werden.

isys_object_recognition_msgs.**set_get_json**()

/cameras/Ensenso_Camera_0/set_get_json

Parameter

- **set_json** (*string*) –
- **wait_for_null_json** (*string*) –
- **get_json** (*string*) –

Rückgabe

bool set_success: Schreiben erfolgreich

string get_result:

string set_error:

string get_error:

Diese Funktion erlaubt es bequem mit der NxLib zu interagieren. Dazu können mit `set_json` Parameter geschrieben, aber auch Kommandos gestartet werden. Dazu schreibt man einen `Execute`-Knoten und gibt bei `wait_for_null_json` den Pfad zu dessen `Command`-Knoten an. Wenn der Befehl ausgeführt wurde, wird dieser Knoten dann von der NxLib auf `Null` gesetzt. Mittels `get_json` kann entweder ein Parameter gelesen werden oder auch der `Result`-Knoten eines Befehls abgefragt werden.

`isys_object_recognition_msgs.`**`set_roi_rotated_box`**`()`

/cameras/Ensenso_Camera_0/set_roi_rotated_box

Parameter

- **box_pose** (*geometry_msgs.Pose*) –
- **box_size** (*geometry_msgs.Vector3*) –

Setzt eine Region of Interest (ROI) auf die die Punktwolke beschnitten wird.

`isys_object_recognition_msgs.`**`start_capture`**`()`

/cameras/Ensenso_Camera_0/start_capture

Parameter
left_camera (*bool*) – Soll die linke oder rechte Kamera verwendet werden

Rückgabe
success: true

Startet eine Daueraufnahme (2D), das linke oder rechte 2D Rohbild wird auf das Topic `/cameras/Ensenso_Camera_0/camera_image` publiziert und in Mikado auf der Kameraseite angezeigt.

`isys_object_recognition_msgs.`**`stop_capture`**`()`

/cameras/Ensenso_Camera_0/stop_capture

Rückgabe
success: true

Beendet den Dauerlauf der Bildaufnahme

Roboter/Hardware

`extra_industrial_msgs.`**`control_hand`**`(`*bool open*`)`

Aufruf in Code: rospy.ServiceProxy(‚control_hand‘, ControlHand)

Parameter
open (*bool*) –

Schaltet den Greifer.

`extra_industrial_msgs.`**`read_input_signal`**`(`*int signal_number*`)`

Parameter
signal_number (*int*) –

Rückgabe

returnValue: bool

success: bool

Codebeispiel
read_input_signal = rospy.ServiceProxy(‚read_input_signal‘, ReadSignal)

res = read_input_signal(signal_number=23)

if res.success:

print res.returnValue

Liest den Wert eines digitalen Eingangs.

extra_industrial_msgs.**read_output_signal**(*int signal_number*)

Aufruf in Code: rospy.ServiceProxy(‚run_output_signal‘, ReadSignal)

Parameter

signal_number (*int*) –

Rückgabe

returnValue (bool):

success (bool):

Codebeispiel

read_output_signal = rospy.ServiceProxy(‚read_output_signal‘, ReadSignal)

res = read_output_signal(signal_number=23)

if res.success:

print res.returnValue

Liest den Wert eines digitalen Ausgangs.

extra_industrial_msgs.**set_output_signal**(*int signal_number*, *bool value*)

Aufruf in Code: rospy.ServiceProxy(‚set_output_signal‘, SetSignal)

Parameter

- **signal_number** (*int*) –
- **value** (*bool*) –

Rückgabe

success (bool):

Setzt einen digitalen Ausgang.

extra_industrial_msgs.**run_program**()

Parameter

- **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]

Rückgabe

returnValue (int): Ein frei belegbarer Rückgabewert

success (bool): War die Ausführung erfolgreich

Führt ein klassisches Roboterprogramm aus, das auf der Robotersteuerung gespeichert ist.

industrial_msgs.**industrial_io_read**()

Aufruf in Code: rospy.ServiceProxy(‚industrial_io_read‘, IORead)

Parameter

items (*industrial_msgs.IOReadRequestItem[]*) – Ein Array von [int type, int index]

Rückgabe
items (industrial_msgs.IOReadReplyItem[]): Ein Array von [int type, int index, int result, int value]

Erlaubt es mehrere Ein- oder Ausgänge gleichzeitig zu lesen, dies ist insbesondere interessant um ganze Datenblöcke bei einem Feldbus zu erfassen. `type` gibt den Typ des Feldes an, mögliche Werte sind

DigitalIn = 1 (Ein Eingangsbit lesen)

DigitalOut = 2 (Ein Ausgangsbit lesen)

AnalogueIn = 3

AnalogueOut = 4

GroupedIn = 5 (Ein Eingangswort, 16 Bit, lesen)

GroupedOut = 6 (Ein Ausgangswort, 16 Bit, lesen)

Dabei unterstützt nicht jedes Endgerät alle Typen, für Feldbusse (Hilscher) sind beispielsweise nur die digitalen Ein/Ausgänge sowie das Word-weise lesen/schreiben verfügbar.

`industrial_msgs.`**`industrial_io_write`**`()`

Parameter
`items` (*`industrial_msgs.IOWriteRequestItem[]`*) – Ein Array von [int type, int index, int value]

Rückgabe
items (industrial_msgs.IOWriteReplyItem[]): Ein Array von [int type, int index, int result]

Schreiben von mehreren Ausgängen oder Feldern in einem Durchlauf, insbesondere interessant für Feldbusschnittstellen. Setzt ein oder mehrere Felder mit dem Typ `type`, der Adresse `index` auf den Wert `value`. Der Rückgabewert enthält für jedes Feld die Information ob dieses geschrieben werden konnte (result = 1) oder nicht (result = 0). Welcher `type` zu Ein- oder Ausgang gehört kann in der Auflistung bei industrial_io_read entnommen werden.

8.5.3 Mikado ROS Actions

Neben den ROS-Services existieren als externe Schnittstelle auch ROS-Actions. Diese lassen sich mittels Action-Clients nutzen und erlauben asynchrone Befehle und Rückmeldungen. Genaue Informationen über Nutzung von ROS-Actions entnehmen Sie bitte der offizellen ROS Dokumentation.

Im Folgenden sind die ROS-Actions aufgeführt, die Mikado anbietet.

`mikado_msgs.calibration.`**`cancel`**`()`

Hand-Auge-Kalibrierung abbrechen

Rückgabe
error_code

EC_SUCCESS	0
EC_ERROR	1
EC_ACTION_RUNNING	2
EC_DISABLED	3

`mikado_msgs.calibration.`**`capture`**`()`

Aufnahme für die Hand-Auge-Kalibrierung auslösen

Parameter
`current_pose` (*`geometry_msgs.Pose`*) – Aktuelle Roboterpose

Rückgabe
error_code

EC_SUCCESS	0
EC_ERROR	1
EC_ACTION_RUNNING	2
EC_DISABLED	3
EC_NO_CALIBRATION_STARTED	4

mikado_msgs.calibration.**finish**()

Hand-Auge-Kalibrierung berechnen und im Erfolgsfall auf der Kamera speichern.

Die Rückgabewerte beinhalten neben der Fehlernummer Werte zum Kalibrierergebnis (durchschnittliche Abweichungen).

Rückgabe
error_code

EC_SUCCESS	0
EC_ERROR	1
EC_ACTION_RUNNING	2
EC_DISABLED	3
EC_NO_CALIBRATION_STARTED	4

Rückgabe
meanDist [mm]

Rückgabe
meanX [mm]

Rückgabe
meanY [mm]

Rückgabe
meanZ [mm]

Rückgabe
meanAngle [radian]

Rückgabe
maxDist [mm]

mikado_msgs.calibration.**set_origin**()

Ursprung setzen mithilfe einer Kalibrierplatte

Parameter

- **pose_of_measured_object** (*geometry_msgs.Pose*) – Pose der Kalibrierplatte zum Zeitpunkt der Aufnahme relativ zum gewünschten Ursprung
- **camera_name** (*string*) – Kameraname (z.B. „Ensenso_Camera_0“)
- **frontlight** (*bool*) – Frontlicht der Kamera für die Aufnahme aktivieren (falls vorhanden)

Rückgabe
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**()

Kalibriervorgang initialisieren (manueller Modus) oder Automatik starten (automatischer Modus).

Parameter
calibration_mode –

CALIB_MODE_MANUAL	0
CALIB_MODE_AUTO	1

Parameter

- **pose_list_name** (*string*) – (optional) Variablenname der PoseList, die als Liste der Kalibrierposen verwendet werden soll
- **camera_name** (*string*) – Kameraname (z.B. „Ensenso_Camera_0“)
- **frontlight** (*bool*) – Frontlicht der Kamera für die Aufnahme aktivieren (falls vorhanden)

Rückgabe
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

Ein Skript kann eigene Informationen zum Systemstatus in Mikado beitragen, ein Fehlerzustand führt dann z.B. zu einer Nachfrage vor der Programmausführung und wird auch in der Liste am linken Rand der Statusleiste angezeigt. Dazu muss zum Einen ein Eintrag in der `config/status.yaml` ergänzt werden:

```
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
```

Auf dem neu ergänzen Topic muss nun mindestens alle 10s eine Nachricht publiziert werden. Hier ein minimales Beispiel

```
1 #!/usr/bin/env python
2 import rospy
3 from diagnostic_msgs.msg import DiagnosticArray, DiagnosticStatus
4
5 pub = rospy.Publisher('sps_status', DiagnosticArray, queue_size=10)
6 rospy.init_node('sps_script', anonymous=True)
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
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()
```

Im Gegenzug kann das Skript auch auf dem Topic `/system_status` eine Zusammenfassung des aktuellen Systemstatus lesen.

8.5.5 Eigenen ROS Service erstellen

Der folgende Beispielcode kann als Grundlage verwendet werden, um eine eigene ROS-Node mit beliebiger projektspezifischer Funktionalität zu erstellen. Der Node bietet einen ROS-Service vom Typ `mikado_extension_msgs::RunExtensionBasic` an, welcher wiederum beispielsweise mittels der `lab.callExtensionService()` Funktion im Mikado-Programmablauf aufgerufen werden kann (siehe Kapitel *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"
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:
```

(Fortsetzung auf der nächsten Seite)

(Fortsetzung der vorherigen Seite)

```
        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()
```

Erklärung: Die Node im Beispiel trägt den Namen „custom_node“ und stellt einen ROS Service `/custom_node/berechne_produkt_id_plus_100` zur Verfügung. Beim Aufruf dieses Services wird die Logik der Funktion `custom_calculation()` aufgerufen. Diese liest im dargestellten Beispiel die Mikado Variable „ProfinetInput_ProductID“ aus, addiert 100 und schreibt das Ergebnis in die Mikado Variable „i_ergebnisBerechnung“.

Aufruf (exemplarisch):

```
lab.callExtensionService("/custom_node/berechne_produkt_id_plus_100")
```

Um eigene Logik zu implementieren, kann der Code der Funktion `custom_calculation(...)` mit eigener Logik ergänzt werden. Um die Node automatisch mit der Zelle zu starten fahren Sie mit der Anleitung im Kapitel *Launchfile*.

8.5.6 Launchfile

Um ein Skript automatisch mit Mikado zu starten und zu beenden, speichert man es im Ordner `Scripts` der Zelle ab (ggf den Ordner anlegen). Nun öffnet man die Datei `launch/main.launch` und ergänzt den folgenden Textblock:

```
<node name="custom_script" pkg="meine_zelle" type="Custom.py" output="screen" />
```

Hier wird unter dem frei wählbaren Namen[1] `custom_script` unser Script mit dem Dateinamen `Custom.py` aus der Zelle `meine_zelle` gestartet.

Hinweis: Damit die Node gestartet werden kann, muss die Datei ausführbar sein. Um eine Datei ausführbar zu machen, öffnen Sie die Konsole im Unterordner `Scripts` (Rechtsklick -> Konsole öffnen) und geben Sie den folgenden Befehl ein:

```
chmod +x ./Custom.py
```

---

[1] zulässig sind Kleinbuchstaben, Zahlen und Unterstriche. Der Namen dient aber auch nur dazu, das Programm ggf zu erkennen.

KAPITEL 9

Aktualisierungen

In diesem Kapitel werden Änderungen in Mikado-Versionen detailliert beschrieben. Die Liste der Änderungen an allen Mikado Komponenten finden Sie auch immer aktuell auf unserer Webseite unter https://optonic.com/support.

9.1 Mikado

Untenstehend finden Sie die Änderungen für alle Mikado Version.

9.1.1 Version 3.4.0

Mikado Version 3.4.0 erweitert die Griffsuche, ergänzt neue Programmiermöglichkeiten und verbessert die Bedienung beim Starten, Sichern und Einrichten von Programmen.

Erweiterte Griffsuche

Die Tasks *Griffe auslesen*, *Teil greifen* und *Griffsuchen-Statistik* stellen zusätzliche Ausgabevariablen bereit. Dadurch können Programme direkt auf Informationen wie Fundanzahl, Anzahl greifbarer Funde, Fundpose, Griff, Greifer und Kollisionsart zugreifen.

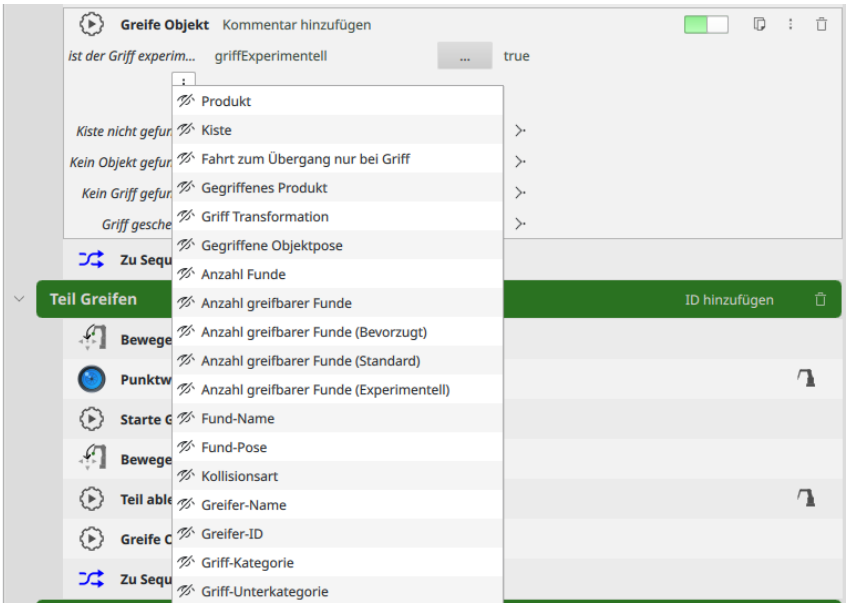


Abb. 1: Zusätzliche Ausgabevariablen im Task `Greife Objekt`.

Für die Griffsuche kann nun eine feste Z-Höhe relativ zur Kiste vorgegeben werden. Die kartesischen Bewegungen des Greifens und Entnehmens werden dadurch bis zu dieser Höhe geplant, was bei Gestänge, engen Kisten oder höheren Kanten die Pfadplanung besser kontrollierbar macht.

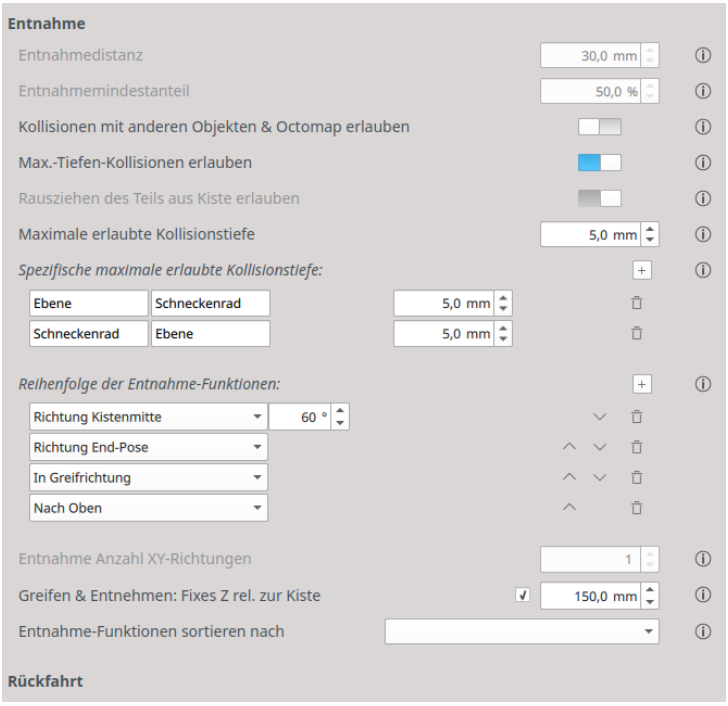


Abb. 2: Einstellung einer festen Z-Höhe relativ zur Kiste für Greif- und Entnahmepfade.

Außerdem können Grifftrajektorien geglättet werden. Dadurch entstehen weichere Übergänge zwischen Anfahrt, Griff, Entnahme und Rückzug.

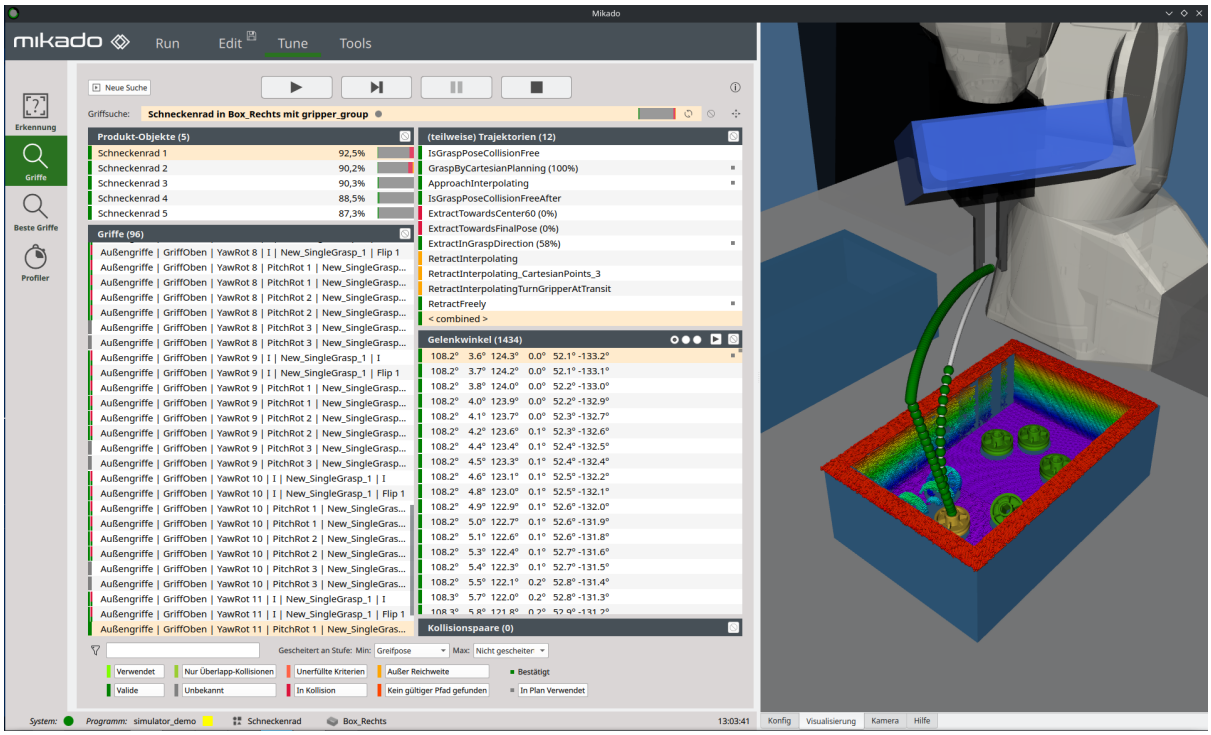


Abb. 3: Geglättete Grifftrajektorie in der Griffinspektion.

Neue Tasks und Schnittstellen

- **Roboterbewegungen:** Neue Tasks ermöglichen relative Bewegungen des Roboters, TCPs oder gegriffenen Teils.
- **Robotersignale:** Der neue Task *Auf Roboterausgangssignal warten* wartet auf Ausgangssignale des Roboters.
- **JavaScript:** Die Pose des gegriffenen Objekts kann über *picksearch.getPickedObjectPose()* abgefragt werden.
- **FANUC:** Die FANUC-spezifische Seite kann das Roboter-Webinterface anzeigen und im Browser öffnen.

Bessere Bedienung

- **Welcome-Seite:** Startdateien für Programme, Produkte und Szenenobjekte können direkt ausgewählt werden.
- **Systemseite:** Die aktuelle Zelle kann als ZIP-Archiv gesichert werden.
- **Programmeinstellungen:** Sprachauswahl, Fenstergröße, Fensterzustand und Vollbildmodus werden nach Änderungen im Mikado auf dem jeweiligen Controller gespeichert. Dadurch kann z.B. bei Offline-Inbetriebnahmen oder Fehlersuche bequem mit der eigenen Sprache gearbeitet werden, ohne die Zelle vom Kunden umstellen zu müssen.
- **Task-Editor:** Optionale Variablen ohne Wert werden zunächst ausgeblendet und können bei Bedarf über ein Menü hinzugefügt werden.

9.1.2 Version 3.3.0

Mikado Version 3.3.0 führt verschiedene neue Tasks, eine neue Feldbusseite sowie neue Fehlermeldungsfunktionalitäten ein und enthält zahlreiche Usability- und Stabilitätsverbesserungen.

Höhere Flexibilität bei Vakuumgreifern

Für Vakuumgreifer kann ab Mikado 3.3.0 ein Bereich definiert werden, der bei der Griffplanung nicht als Kollisionsobjekt berücksichtigt wird. Wie in dem unten gezeigten Beispiel, wird dabei eine Ebene relativ zum TCP definiert (mit einem festlegbaren Z-Offset). Diese Funktionalität erlaubt es unter anderem, flexible Saugnäpfe von Vakuumgreifern bei der Griffplanung von Kollisionschecks auszuschließen. Einstellbar ist die neue Funktion auf der Greiferseite des Mikado Assistant.

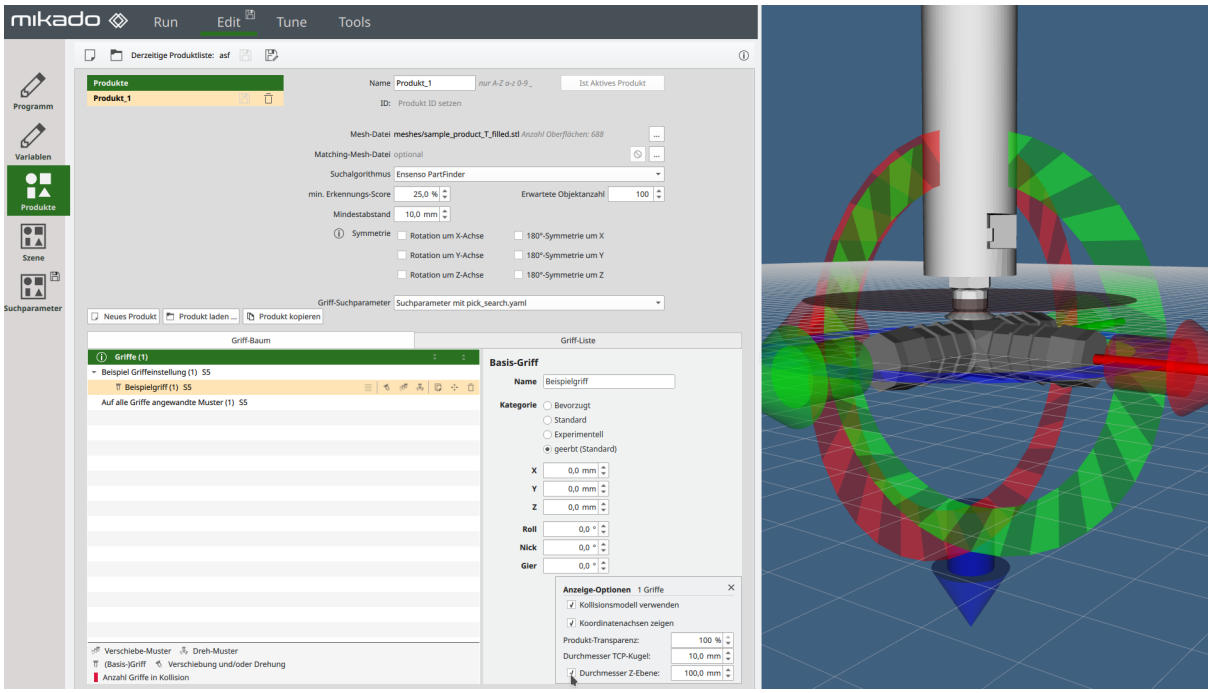


Abb. 4: Visualisierung des während der Griffsuche nicht kollisionsgeprüften Bereichs am Greifer. Alles unterhalb der dunkelgrauen Ebene wird bei der Griffsuche nicht als Kollisionsobjekt behandelt.

Darüber hinaus ist es nun bei verschiedenen Roboterherstellern möglich, den Vakuumgreifer nicht bereits an der Transitpose anzuschalten, sondern dies während der Fahrt bei Erreichen eines festlegbaren Abstands zu tun. Dadurch kann ungewolltes Ansaugen oder Verschieben von Teilen vor Erreichen der eigentlichen Griffpose minimiert werden.



Abb. 5: Einstellungsmöglichkeit der entfernungsbasierten Aktivierung eines Vakuumgreifers und des komprimierbaren Greiferbereichs im Mikado Assistant.

Zusätzliche Filtermöglichkeiten

Um bei komplexen Programmen die Handhabung zu erleichtern, wurden an verschiedenen Stellen Filtermöglichkeiten hinzugefügt. Durch Eingabe in die entsprechenden Textfelder können Variablen/Sequenzen einfacher gefunden werden.

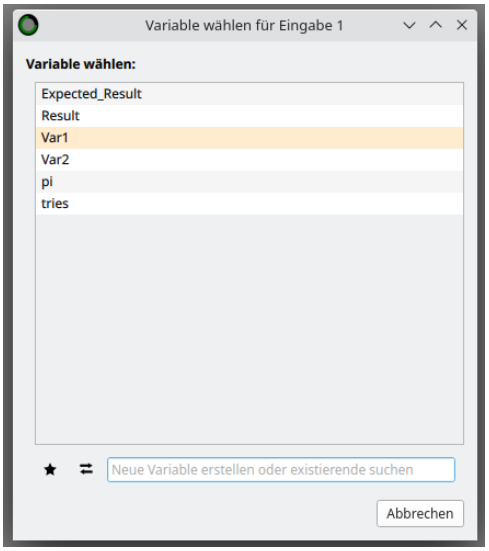


Abb. 6: Neuer Filter in der Variablenauswahl

Neue Filteroptionen gibt es an den folgenden Stellen:

- **Variablenauswahl:** Filtern der Kisten, ROIs, Produkte und anderer Variablen
- **Variableneditor:** Filtern nach Typ
- **Sequenzauswahl:** Filtern nach Namen

- **Programmeditor:** Filtern beim Springen zu Sequenzen
- **Beste Griffe:** Neuer Filter um ausschließlich Griffe anzuzeigen, die zum Zeitpunkt der Griffauswahl vorhanden waren
- **Griffinspektion:** Filtern nach Namen des Griffs

Neue und überarbeitete Tasks

In der Version 3.3.0 stehen drei neue Rechentasks zur Verfügung: *Arithmetische Operation*, *Logische Operation* und *Vergleichsoperation*. Der Task *Arithmetische Operation* ermöglicht die Durchführung einfacher Rechenoperationen mit zwei Variablen (Integer oder Double), mit Hilfe des Tasks *Logische Operation* können boolsche Verknüpfungen realisiert werden, und *Vergleichsoperation* ermöglicht das Vergleichen von zwei Variablen.



Abb. 7: Der neue *Arithmetische Operation* Task

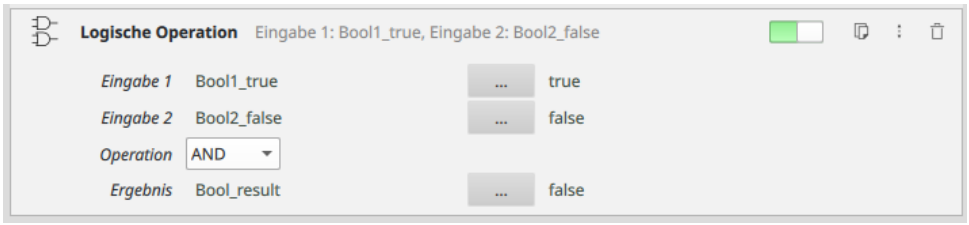


Abb. 8: Der neue *Logische Operation* Task

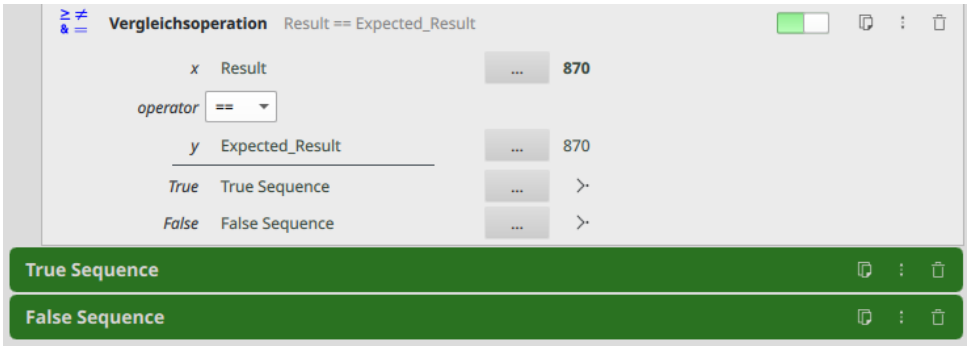


Abb. 9: Der neue *Vergleichsoperation* Task

Als weitere Neuerung steht jetzt der *Feldbusausgangssignal asynchron setzen* Task bereit, der einen Feldbusausgang setzt, sobald der Roboter eine festgelegte ROI verlassen hat. Das ermöglicht eine komfortable und schnelle Rückmeldung an die SPS, wenn der Roboter einen bestimmten Bereich der Anlage verlassen hat.

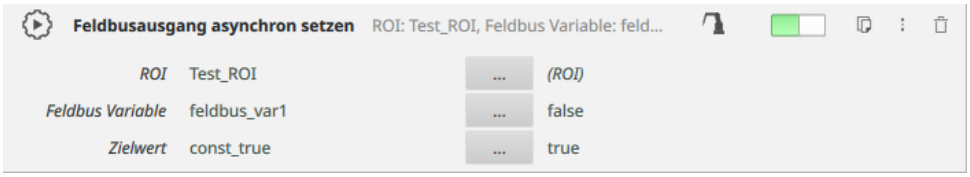


Abb. 10: Der neue *Feldbusausgangssignal asynchron setzen* Task

Weitere Änderungen von Tasks:

- *Pose erstellen* kann nun auch zum Überschreiben einer einzelnen Komponente (z.B. des Z-Werts) verwendet werden
- *Finde Objekt*, *Griffsuche starten* und *Griffe auslesen* nehmen keine Änderungen am aktiven Produkt mehr vor
- Im *Teil greifen* Task kann über einen optionalen Parameter eingestellt werden, dass nur bei einem gefundenen Griff die Fahrt zur Transitpose erfolgen soll

Erweiterte Funktionalität für JavaScript

Um komplexe Funktionalitäten noch besser umsetzen zu können, wurde der JavaScript-Task erweitert und folgende Verbesserungen hinzugefügt:

- Beim Umbenennen einer Mikado-Variable wird diese auch in *JavaScript* Tasks umbenannt
- XMLHttpRequests können optional erlaubt werden
- Bei Griffen aus der Griffsuche sind neben den TCP-Posen auch die Tool0-Posen zugänglich
- `lab.callService` erlaubt die Verwendung von beliebigen ROS Services
- native Roboterprogramme können aus JavaScript aufgerufen werden

Verbesserungen in der Griffsuche

Die Griffsuche wurde um verschiedene Parameter und Sortierkriterien erweitert. Außerdem wurden die Rückmeldungen an den Nutzer verbessert:

- *Griffsuche starten* akzeptiert nun einen Parameter für den (zukünftigen) Greifer
- **Positionsgleichheit**: Separate Parameter für Toleranz bei Wiedererkennung von erfolglos gegriffenen und wiedererkannten Objekten: `Teilepose Max. Translationsunterschied`, `Teilepose Max. Rotationsunterschied`, `Griff Max. Translationsunterschied` und `Griff Max. Rotationsunterschied`
- **Sortierkriterien:** Neue Sortierkriterien `Teile-Winkel zur Vertikalen` und `Anzahl Griffversuche`
- **Griffinspektion:** Neuer Button um Visualisierung auf Kistenmitte zu zentrieren
- **Griffinspektion:** Anzeigen wenn Transitpose nicht erreichbar ist oder andere Probleme bestehen
- **Griffsuche:** Verbesserte Berücksichtigung des User-Binnings

Verbesserte Nachvollziehbarkeit für Feldbus Kommandos

Um über einen Feldbus empfangene Kommandos besser nachvollziehen zu können, werden diese auf der neuen `Feldbus` Seite protokolliert. Diese kann in Mikado unter dem Menüpunkt `Run` gefunden werden und hält unter anderem Informationen über die Art der empfangenen Befehle, Startzeit und Dauer sowie den Befehlsstatus bereit.

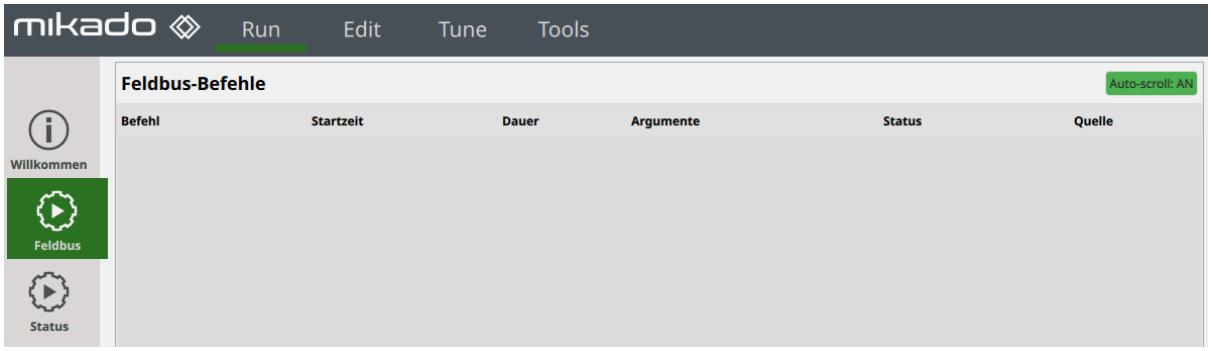


Abb. 11: Auf der neuen Feldbusseite werden alle eingehenden Kommandos erfasst und protokoliert.

Weitere Feldbus Neuerungen:

- **Feldbus Kommandos:** Neue Kommandos zum Starten und Auswerten einer Griffsuche direkt über den Feldbus
- **Mikado Status:** Verbesserte Statusmeldungen bezüglich des Ladens von Programmen, Produkten und Szenen

Objekterkennung

Mit der Version 3.3.0 wurde die Benennung der Symmetrien überarbeitet und weitere Symmetrien ergänzt. Alle Symmetrien werden bei der Objekterkennung und dem Task `Bewege Teil nach` berücksichtigt. Auf der Erkennungsseite kann im Modellgenerierungstab durch die neue Visualisierung der vom PartFinder berücksichtigten Ansichten je nach gewählter Symmetrie, die Auswirkungen der Symmetriewahl besser nachvollzogen werden.

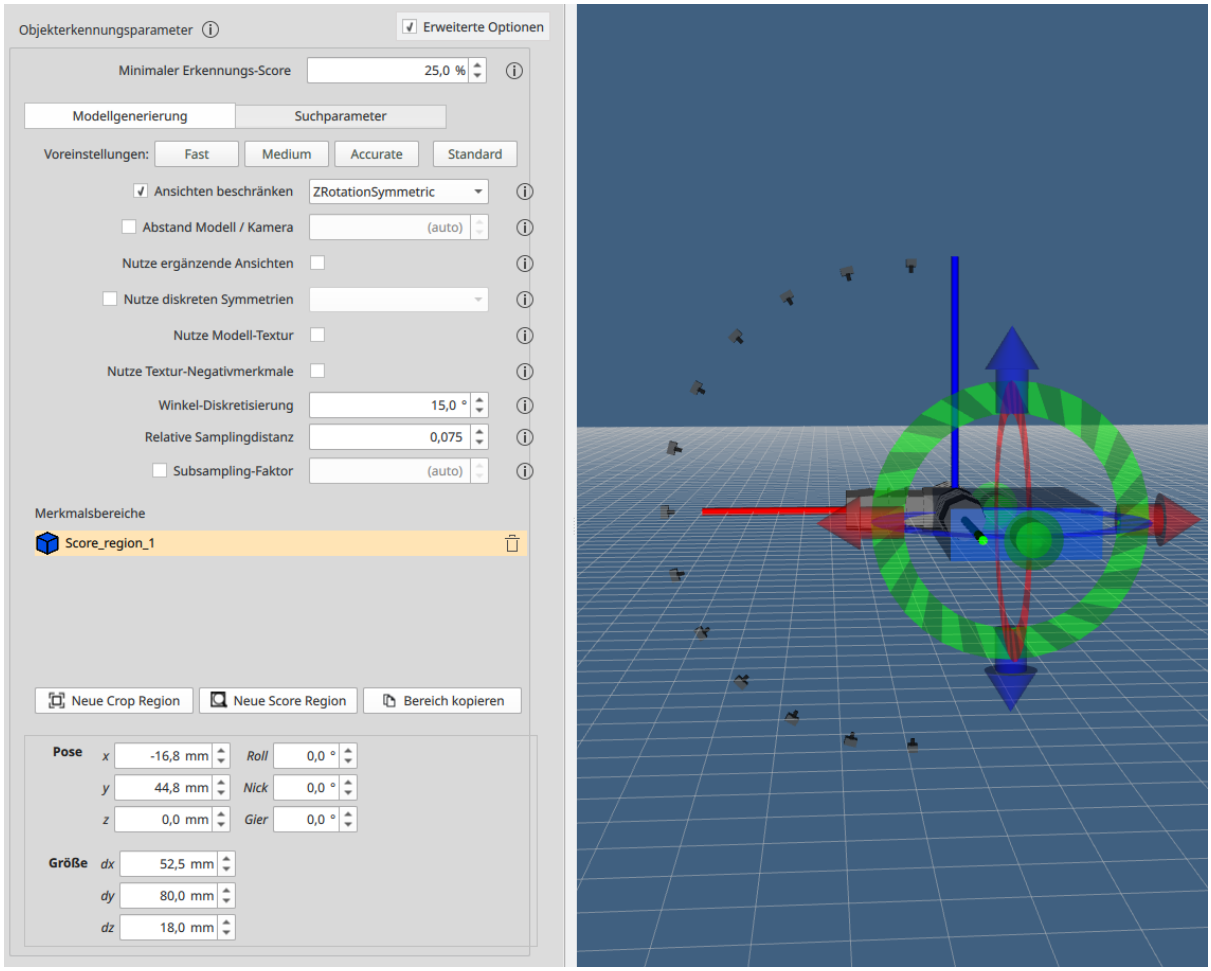


Abb. 12: Neue Visualisierung der Anzeige der bei der Objekterkennung berücksichtigten Ansichten und Beispiel einer `Score Region`

Außerdem wurden an Mikado Anpassungen vorgenommen, um die Ensenso SDK-Version 4.2 zu unterstützen, wodurch eine deutliche Verbesserung der Objekterkennung erreicht wird.

Zusätzlich wurden folgende Punkte verbessert:

- **Bild speichern:** Der aktuelle Produktname kann beim Speichern von Filecams als Prefix verwendet werden
- **Modellgenerierung:** Es können nun Bereiche des Modells definiert werden (`Score Region`), die priorisiert in die Berechnung des Erkennungs-Scores eingehen
- **Produkt Mesh:** OBJ-Meshes können jetzt direkt als Produkt Mesh ausgewählt werden
- **Hypothesen Clustering:** bis zu 4096 Cluster erlaubt (wie im Partfinder)

Debugging und Error Reporting

Um im Fehlerfall die Analyse zu vereinfachen, wurde auf der Seite Tools->Tools ein Button hinzugefügt, um einen Fehlerbericht zu erstellen. Nach einem Klick auf den Button werden Sie dazu aufgefordert, den beobachteten Fehler zu beschreiben. Nach einem Klick auf „OK“ wird automatisch eine Archivdatei mit relevanten Debugdateien erstellt, die die Untersuchung des Fehlers deutlich erleichtert.

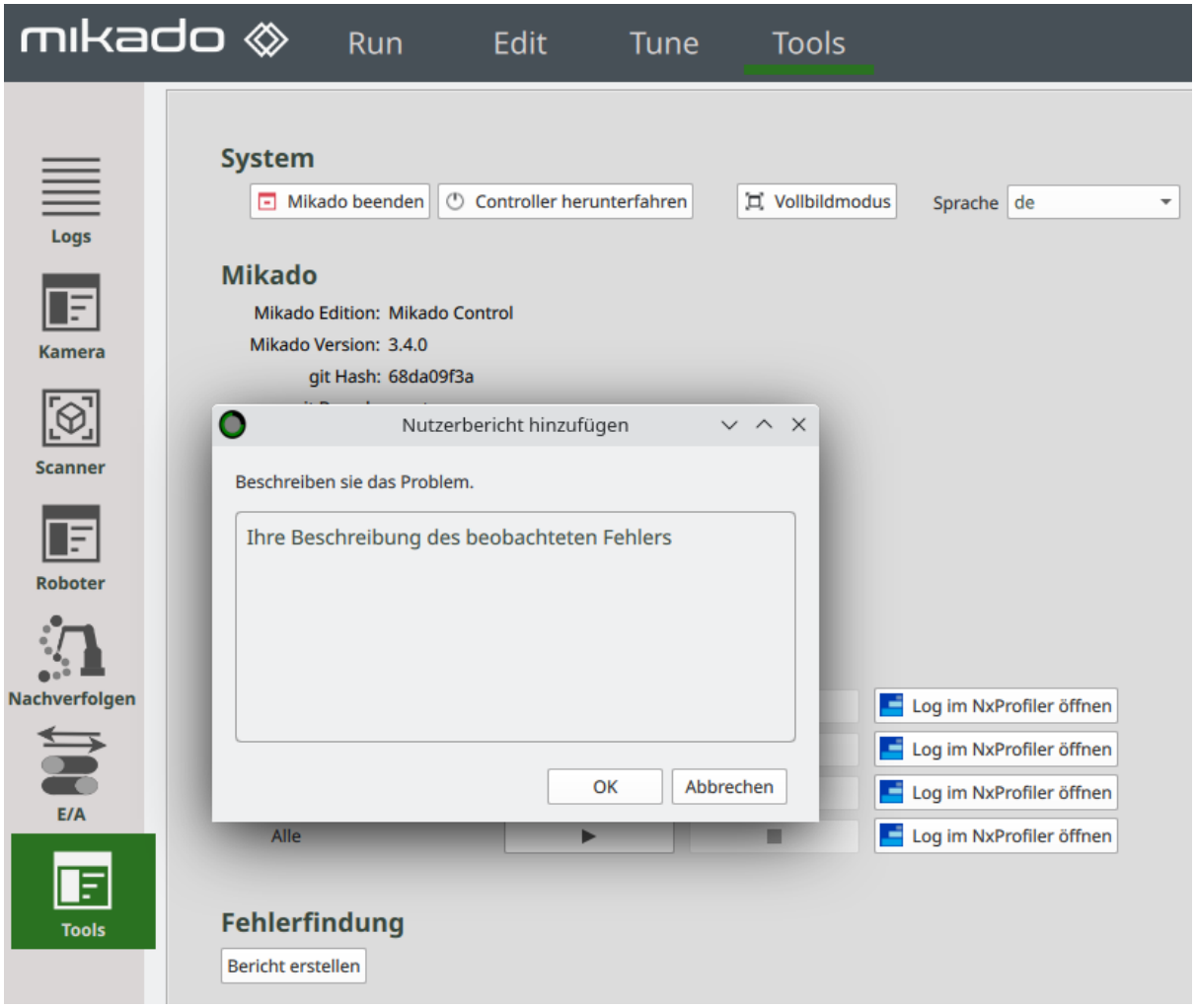


Abb. 13: Erstellung eines Fehlerberichts auf der Seite Tools->Tools

Die Log-Anzeige unter Tools->Logs wurde modernisiert. Die eigenständige Log-GUI kann nun auch gespeicherte Logs zur Analyse laden.

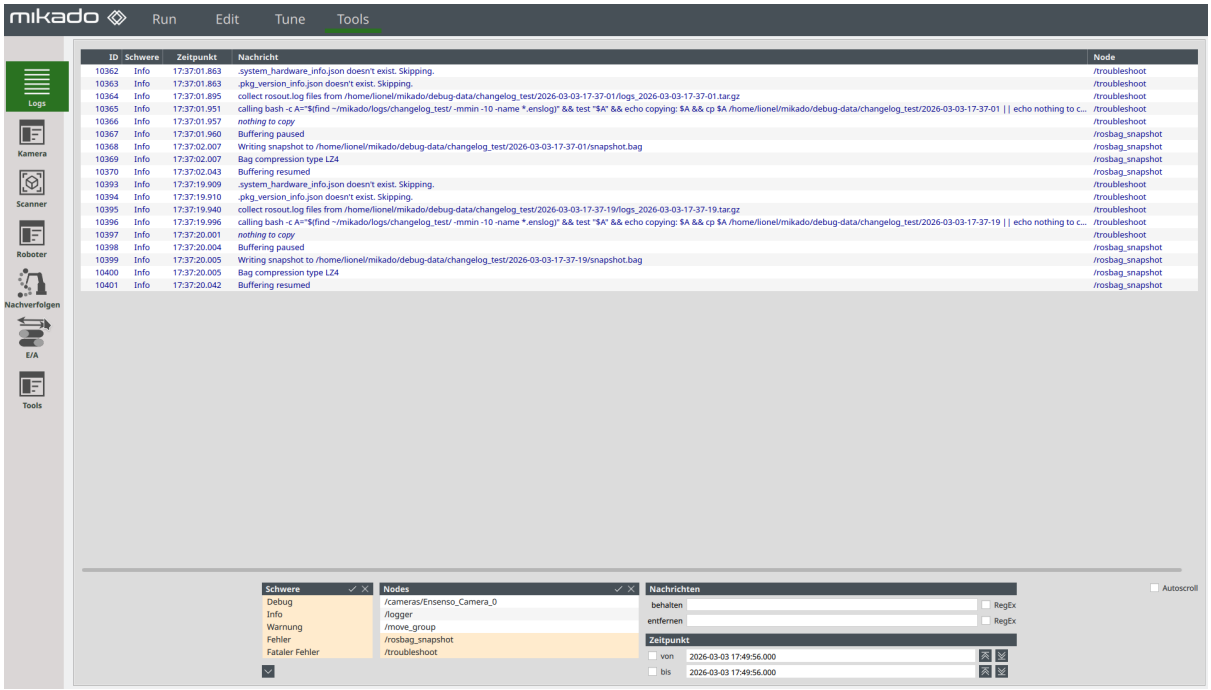


Abb. 14: Ansicht der modernisierten Seite Tools->Logs

Bessere Usability

- **Variablen:** Kameras und Greifer als Variablentyp
- **Griffeditor:** Vereinfachtes Einrichten durch optional transparente Produkte und an Produktgröße angepasste Marker
- **Kioskmodus:** Das Verlassen des Kioskmodus mit Alt+Tab etc. wird unterdrückt. Über eine KWin-Fensterregel kann auch die Windows-/Meta-Taste unterdrückt werden. Der Status und Logins im Kioskmodus werden geloggt.
- **Roboterseite:** Der aktive Wechselgreifer wird anzeigt und kann geändert werden
- **Inbetriebnahmemodus:** Neuer Einrichtmodus, in dem Handbetrieb nicht als Fehler im Roboterstatus kommuniziert wird
- **Szenenobjekte:** Kameraauswahl bei Aufnahmebutton

Verschiedenes

- **Protokollierung:** Im Ordner einer Mikado-Zelle werden nun neben den Softwareversionen auch die Hardware und IP-Adressen protokolliert
- **Roboterseite:** Anzeige von Informationen zum Robotermodell und Hersteller
- **Statusbar:** Techman-Fehlercodes werden passend zum Handbuch in Hex angezeigt
- **Suchstrategien:** Schrittweite der kartesischen Bewegungen ist einstellbar
- **Kuka:** Bei Robotern von Kuka sind Parameter zur Beeinflussung der Geschwindigkeit auf der Roboterseite sichtbar
- **MeshScanner:** Meshs nicht immer im OBJ-Format speichern
- **Mesh-Namen:** Mehr Sonderzeichen in Mesh-Dateinamen erlaubt
- **Programmeditor:** Warnung wenn Programm wegen voller Festplatte nicht geschrieben werden kann
- **Szenenobjekte:** Verbesserte Rückmeldung wenn Kistensuche scheitert

- **Optimierungen:** Optimierungen beim Laden und Validieren von Programmen sowie bei der Berechnung von Griffen
- **Bugfixes:** Diverse Bugfixes

9.1.3 Version 3.2.0

Griffsuche graphisch einstellbar

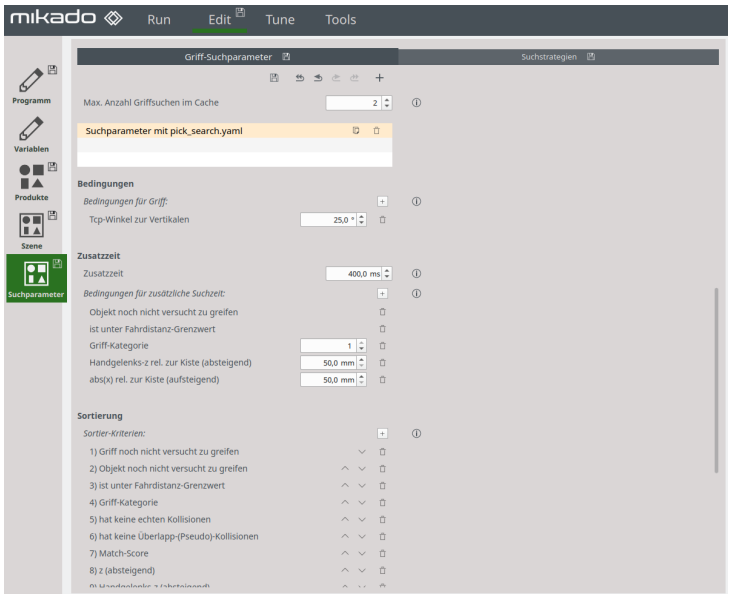


Abb. 15: Einstellung der Parameter einer Griffsuche

Die Parameter der Griffsuche können nun vollständig direkt in Mikado eingestellt werden und es können auch je nach Produkt oder Situation verschiedene Parametersätze verwendet werden. Damit kann beispielsweise bequem beeinflusst werden ob Teile strikt in einer gewissen Reihenfolge entnommen werden müssen oder ob beim Greifen bestimmte Strategien bevorzugt werden sollen. Die bisherigen, rudimentären Einstellmöglichkeiten im Mikado Assistant wurden mit dieser Version entfernt.

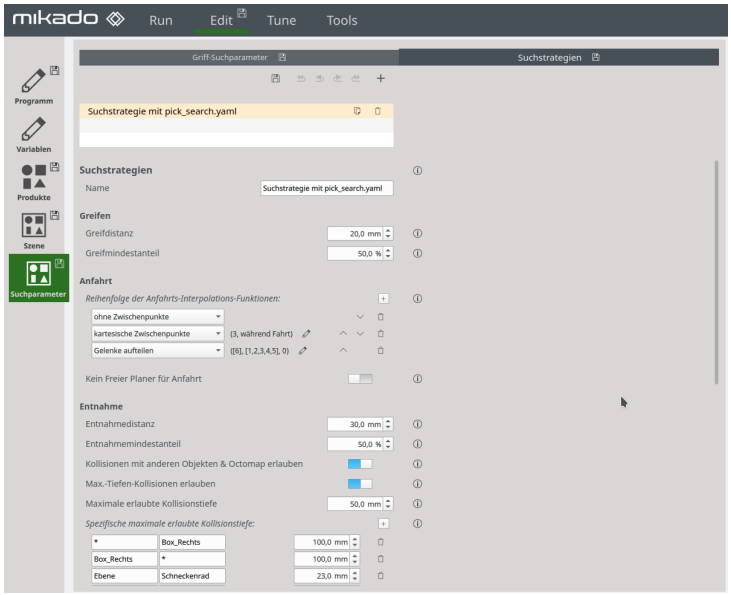


Abb. 16: Einstellung der Entnahmestrategien

Zusätzlich sind einige neue Extraktionsvarianten hinzu gekommen, insbesondere das seitliche Herausziehen im Objektkoordinatensystem, das sich für chaotisch geschichtete Blechteile bewährt hat.

Wechsel zwischen mehreren Griffsuchen

Es kann nun zwischen mehreren Griffsuchen gewechselt werden und zum Beispiel effizient abwechselnd aus mehreren Kisten gegriffen werden. Die Griffsuchen können dabei alle in der Griffinspektion bzw. in der Liste der besten Griffe untersucht und optimiert werden. Es können auch verschiedene Greifer durchgerechnet werden, ohne eine neue Objekterkennung starten zu müssen. Dies erfolgt automatisch bei einem Starte Griffsuche ohne neue Aufnahme oder über JavaScript (siehe *rerunPickSearch*).

Numerische IDs für Greifer, Kameras, Produkte und Szenenobjekte

Für eine vereinfachte Steuerung des Prozessablaufs über eine SPS können nun numerische IDs zusätzlich zu den Namen verwendet werden, um Greifer, Kameras, Produkte oder Szenenobjekte zu wechseln.

Gelenkwinkelgrenzen besser verständlich

Es wird nun klarer kommuniziert, wenn der Roboter sich außerhalb der Gelenkgrenzen befinden. Sei es beim Programmstart, beim Jogging oder bei Gelenkpositions-Variablen im Editor.

Neue PartFinder Features

Es wurden einige neue Parameter der PartFinder Objekterkennung des Ensenso SDKs 4.1 zugänglich gemacht, dazu zählen die TexturePenalty oder das ComplementViewpoints. Weiterhin ist es nun möglich die Objekterkennung auf Ausschnitte des Models zu begrenzen ohne dazu das CAD Modell künstlerisch bearbeiten zu müssen. Dies erfolgt über sogenannte Merkmalsbereiche, die in der Benutzeroberfläche visualisiert werden.

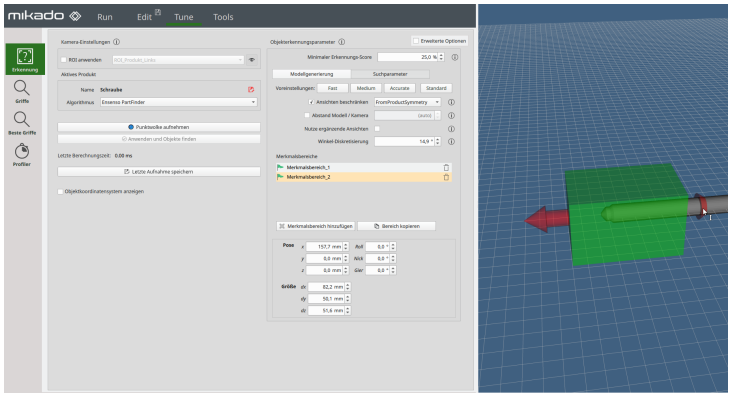


Abb. 17: Beispiel für Merkmalsbereiche: Nur die Spitze einer Schraube soll beachtet werden

Für eine vereinfachte Interpretation der Objekterkennungsergebnisse ist es nun möglich, die Koordinatensysteme der Funde einzublenden.

Griffe auslesen in Mikado Select

Die kollisionsgeprüfte Griffsuche in Mikado Select wird nun wie im Mikado Control über Starte Griffsuche und den neuen Task Griffe auslesen asynchron durchgeführt. Der bisherige Finde Objekte Task wird automatisch beim Upgrade durch die beiden Tasks ersetzt wenn die Griffsuche bisher verwendet wurde.

Kleinere Verbesserungen

- Programm: Tasks können deaktiviert bzw auskommentiert werden und werden dann übersprungen
- Jogging: Suchen nach Variablenname
- Filtern von Variablen im Auswahldialog
- JS: IOs lesen und schreiben (siehe *robot*)
- JS: Slot um Task auf ErrorMightRetry zu setzen (siehe *task*)
- JS: lab.callExtensionService für Erweiterungen (siehe *callExtensionService*)
- WaitForFieldbusInputSignalTask: Task zum Warten auf eine Wertänderung in einer Feldbusvariable (siehe *Auf Feldbuseingangssignal warten*)
- SetInitialValueTask: optionales Speichern des Programms (siehe *Ausgangswert setzen*)
- Auch Variablen von „Addons“ anzeigen
- Status: Springen zum aktiven Task
- Yaskawa-spezifische Funktionen und Fehlermeldungen
- Beim FindObjectTask wird eine ggf. parallel laufende Griffsuche nicht mehr unnötig beendet
- Anzahl Kontaktpunkte bei Kollisionsprüfung einstellbar
- Anzahl Kontaktpunkte bei Kollisionsprüfung erhöht für genauere Eindringtiefenberechnung
- PickInspektor: Bei herausgefilterten Griffen Begründung detailierter
- PickInspektor: Bessere Erläuterungen wenn Griff per Definition mit Objekt kollidiert
- Verwendung von Variablen in JavaScript Tasks auf Variablenseite anzeigen
- Anzahl Mesh-Dreiecke in Szenenobjekten anzeigen
- Bei Bewegungsfehler asynchrone Aufnahme sofort abbrechen
- Neues Profiler-Format verwenden
- separates Paket mikado-robots-common für Helferprogramme
- Usability Verbesserungen am Griffeditor
- Filecams werden unter mikado/filecams gespeichert und die Größe limitiert
- Neuer Task: Szenenobjekt-Position zurücksetzen

9.1.4 Version 3.1.2

Referenzen auf Szenenobjekte und Produkte

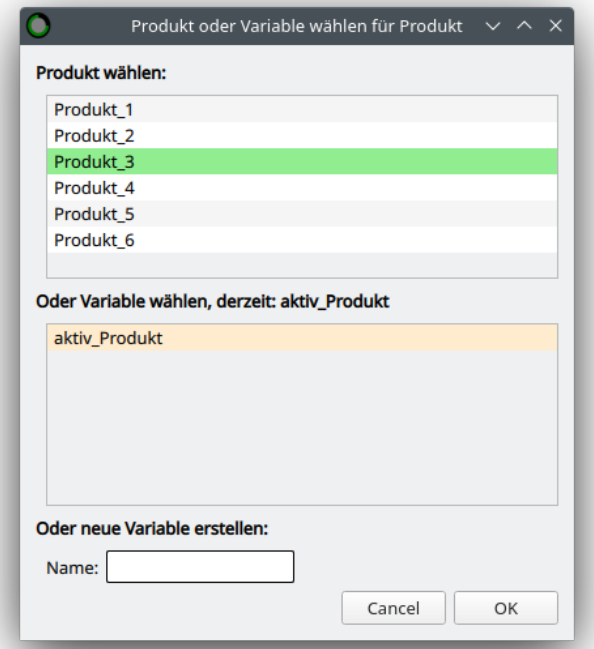


Abb. 18: Auswahl einer Programm-Variable oder einer Produkt-Referenz

Produkte und Szenenobjekte können in Tasks direkt referenziert werden. Dieses Feature kommt insbesondere dann zur Geltung, wenn innerhalb eines Tasks stets die selbe ROI genutzt oder das selbe Produkt gesucht werden soll. Das Setzen des Wertes via Programm-Variable ist weiterhin möglich.

Überarbeitung Roboter Nachverfolgen

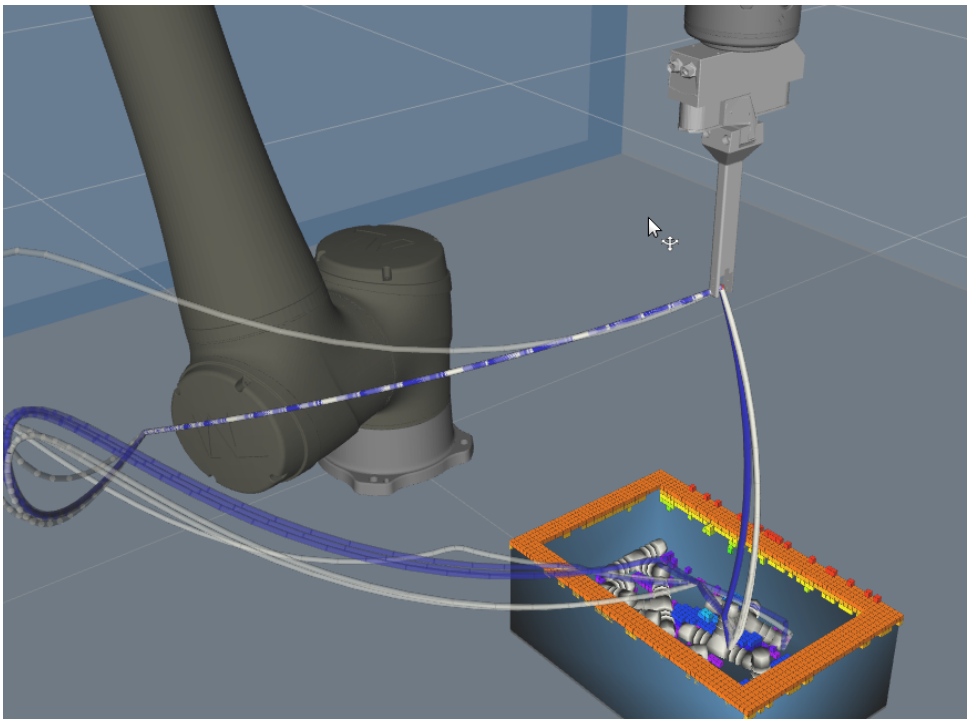


Abb. 19: Visualisierte Bahn des aktiven TCPs

Auf der *Roboter Nachverfolgen*-Seite wird die Bahn des aktiven ToolCenterPoints (TCP) als Linie visualisiert. Des Weiteren unterstützt die Seite die Verwendung von Wechselgreifern.

Greifer mit Positionskontrolle

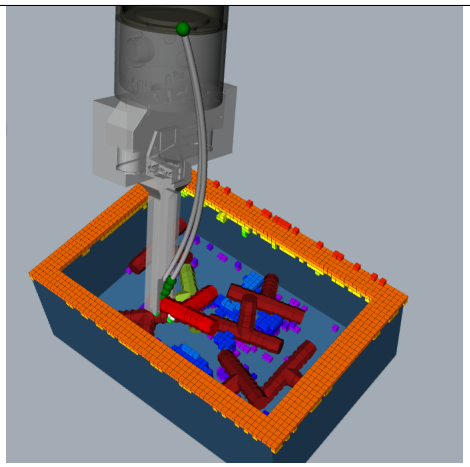


Abb. 20: Griff mit geringer Öffnungsweite

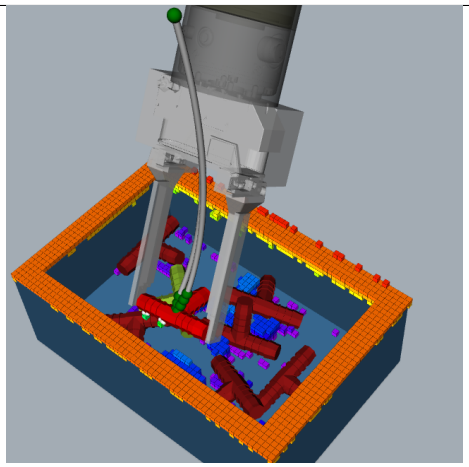


Abb. 21: Griff mit hoher Öffnungsweite

Für Greifer mit Positionskontrolle lässt sich die Öffnungsweite der Greiferfinger für jede Griff-Kategorie einstellen. Hierfür muss in der entsprechenden `gripper.yaml` der Typ des Greifers auf `type: ParallelPositionControl` gesetzt sein.

Mesh-Scanner

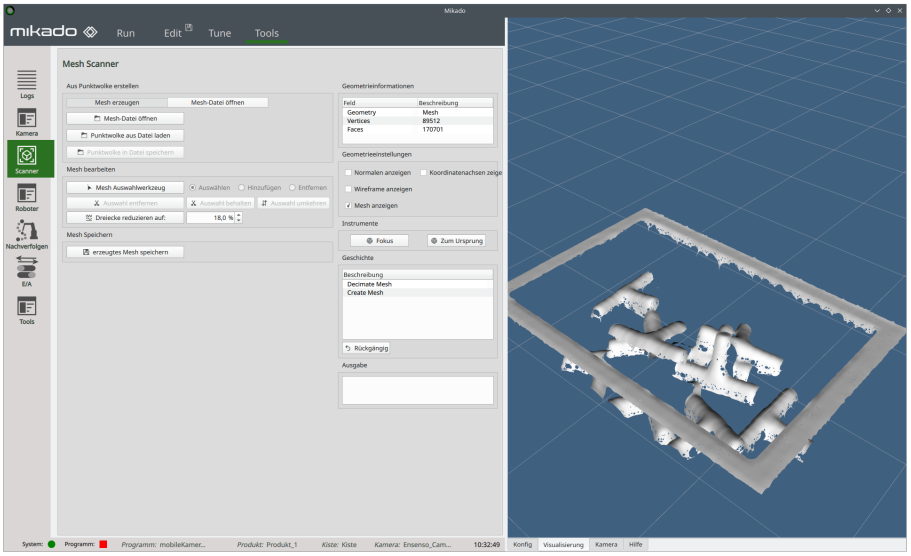


Abb. 22: Screenshot der Scanner-Seite

Die Scanner-Seite bietet die Möglichkeit ein Mesh aus einer Punktwolke zu erstellen und das erstellte Mesh nachzubereiten. Das Mesh kann anschließend in der Produktsuche verwendet werden. Der Scanner kann:

- Punktwolke erfassen oder Punktwolke/Mesh laden
- Mesh aus Punktwolken erstellen
- Mesh vereinfachen
- Teile des Meshes ausschneiden
- Mesh speichern

Kollisionsprüfung der Greifposen im Griff-Editor

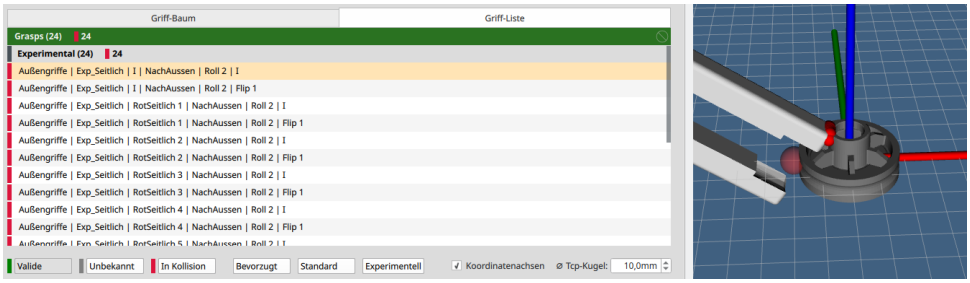


Abb. 23: Griff-Liste und Visualisierung der Kollisionspunkte

Neu erstellte Griffe werden beim Anlegen auf Kollision mit dem Teil geprüft (siehe *Kollisionsprüfung der Greifposen*). Neben dem Griff-Baum gibt es den Reiter Griff-Liste. Diese Liste enthält alle durch den Griff-Baum generierten Griffe. Wählt man in der Griff-Liste einen Griff aus, wird dieser in der Visualisierung angezeigt. Bei einem kollidierenden Griff werden die erkannten Kollisionspunkte zwischen Greifer und Produkt durch kleine rote Kugeln markiert.

Kleinere Optimierungen

- Deutlich schnellere Profinet-Schnittstelle durch event-basiertes Backend. Details siehe: *Version 3.1.0*
- GUI-Sprache auch ohne Mikado Neustart änderbar
- Roboterstatusanzeige aufgeräumt, mehr Fehlerdetails
- Status und Programmeditor: informativere Platzhalter-Texte
- Update Octomap jetzt auch aus JS möglich, ohne Griffsuche / FindeObjekte (siehe *updateOctomap*)
- Griff-Inspektor: Griffsuche kann manuell gestartet werden
- Griffe nach *Kollisionstiefe* sortierbar
- JavaScript: CSV Datei in Dict einlesen, zB für Produktparameter (siehe *file*)
- Warnung falls ein Produkt sehr große Meshes verwendet
- Fehlerhafte Variablen werden auf Variablenseite angezeigt, z.B. Schreiben in konstante Variable oder Variablen, die bei der Verwendung einen falschen Typ haben
- JavaScript: Tcp Koordinaten für Start, Approach, Retract, Final für Griffe (siehe *picksearch*)
- QML: es ist jetzt möglich beliebige Prozesse zu starten, zB Okular für eine projektspezifische Anleitung
- Schnittstelle um die Objekterkennung mit einem projektspezifischen Algorithmus umzusetzen
- PickSearch/JS: Zusätzliches frei nutzbares Bewertungskriterium für Objekte (User Binning)
- Position eines gegriffenen Teils anpassen
- Bewege Roboter nach: eventuelle Kollisionen direkt im Status des Tasks
- Griff-Editor: Kopieren und Verschieben von Griffen
- X > Y? und ähnliche Tasks mit blauen Symbolen für Logik/Sprünge
- Task um die Länge einer Liste auszulesen
- FindeObjekte: optionaler Sequenzwechsel falls Kiste gesucht und nicht gefunden wurde
- JS: Mehr Informationen über den letzten Griff
- Griffe nach Kollisionstiefe sortierbar

9.1.5 Version 3.0.5

- Fehlerkorrekturen zur Behandlung neu erstellter Variablen

9.1.6 Version 3.0.4

- Cuda kann für die Produktsuche via PartFinder aktiviert werden
- Sehr kurze kartesischen Fahrten werden nicht mehr auf 10 Punkte erweitert

9.1.7 Version 3.0.3

- Für Entwickler ist „Finde Objekte“ auch in Mikado Control sichtbar
- Finde Objekte hat nun auch einen optionalen Parameter „Box“
- PartFinder: maximale Anzahl der Cluster auf 200 erhöht
- neuer Task: Tool Center Point (TCP) kartesisch bewegen, optional mit Toleranz zum Ziel.
- Informationen über die aktuell verwendete FileCam anzeigen auf der Objekterkennungsseite
- GriffEditor: Verschiebungen können nun auch symmetrisch angelegt werden

9.1.8 Version 3.0.2

- Griffe: Gesamte Trajektorie in Vorschau visualisierbar
- „read-only“ Variablen
- Variablenseite: Favorisierte Variablen

9.1.9 Version 3.0.1

- Detaillierte Anzeige der Objektsuche / Griffsuche Pipeline
- ADS Schnittstellen-Support
- NxLogs Gui für Profiling

9.1.10 Version 3.0.0

Kollisionsprüfung des Greifers für Mikado Select

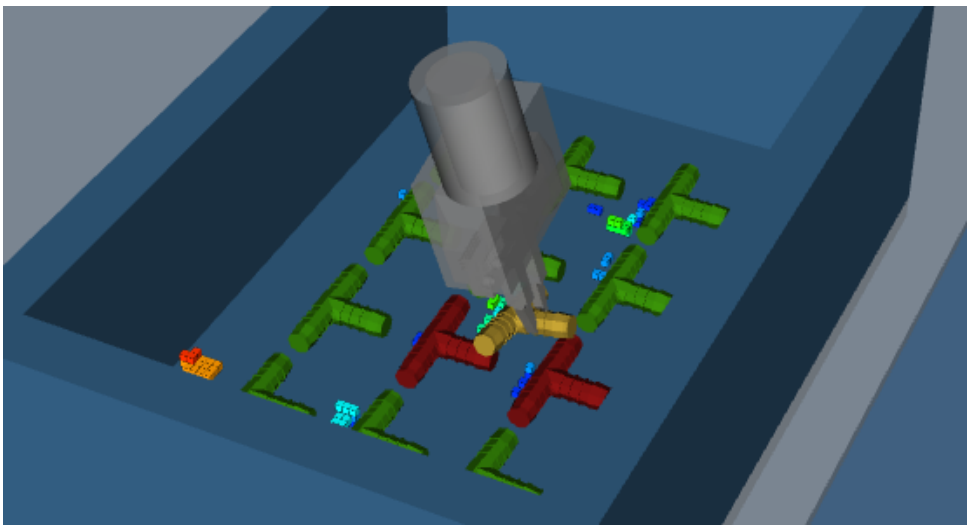


Abb. 24: Kollisionsprüfung des Greifers für Mikado Select

Mikado Select unterstützt das Finden von kollisions-geprüften Griffen. Der Task *Finde Objekt* besitzt hierfür einen neuen Rückgabewert `Valide Griffe`. Diese Posen-Liste enthält den besten Griff pro gefundenem und greifbarem Objekt. Die Liste ist sortiert nach benutzerspezifischen Kriterien (siehe *Beste Griffe*).

Visualisierung der Griff-Priorisierung

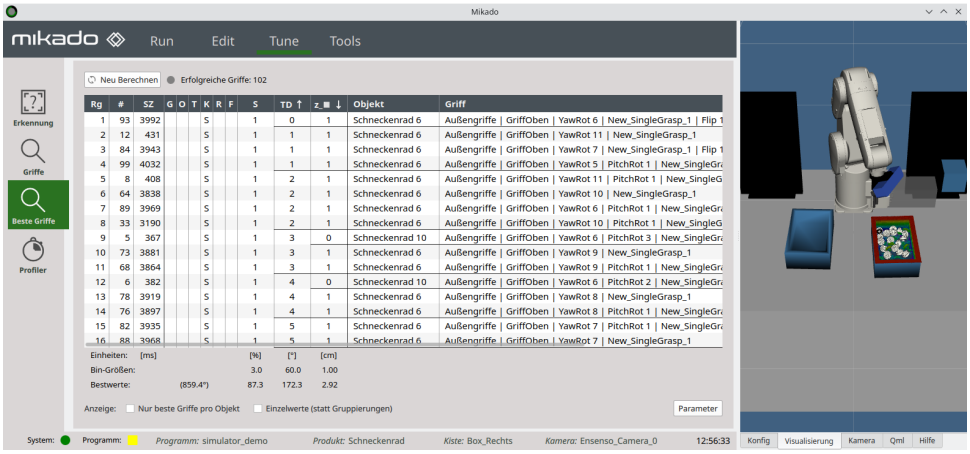


Abb. 25: Screenshot Beste-Griffe

Sortier- und Ausschluss-Kriterien für Griffe können global (für alle Produkte) in der `pick_search.yaml` definiert werden. Um alle Griffe bezüglich der verschiedenen Kriterien vergleichen zu können wird Binning verwendet. Die neue Seite *Beste Griffe* bietet ein Analyse-Tool für die Sortier-Kriterien.

Unterstützung für Wechselgreifer

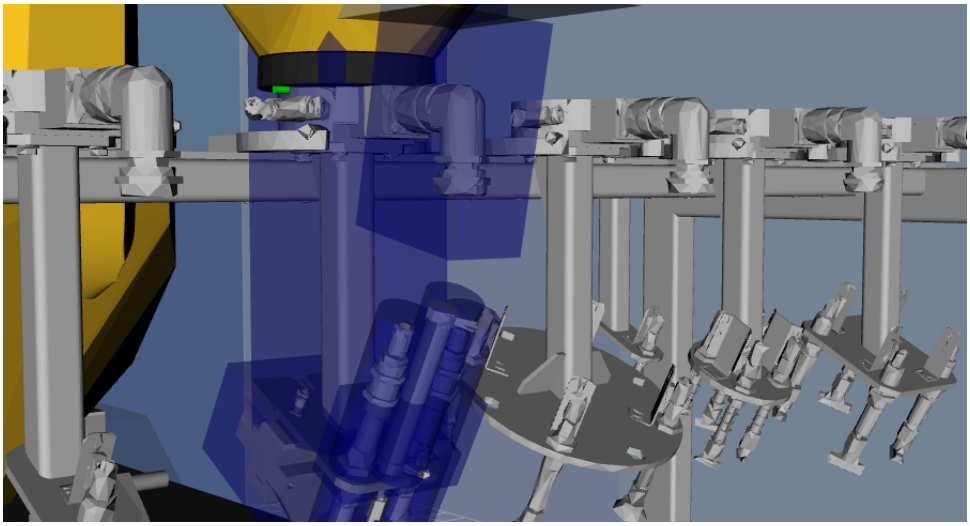


Abb. 26: Unterstützung für Wechselgreifer

Mikado Control unterstützt das Wechseln von Greifern zur Laufzeit einer Roboter-Zelle. Mit dem neuen Task *Setze aktiven Greifer* kann innerhalb des Roboterprogramms zwischen den definierten Greifer-Geometrien gewechselt werden. Produkte haben nun Greifer-abhängige Griffdefinitionen, siehe *Einrichten von Griffen bei Wechselgreifern*. Die Griffsuche prüft jeweils nur Griffe die für den aktiv gesetzten Greifer definiert wurden.

Das Einrichten einer Zelle mit mehreren Greifern wird vorerst vom Mikado-Support übernommen. Das Hinzufügen von neuen Greifern in Mikado Assistant ist in Arbeit und für das nächste Release angestrebt. Das Editieren von bereits eingerichteten Zellen in Mikado Assistant wird voll unterstützt.

Zusatzgelenke auch bei Parallelgreifern

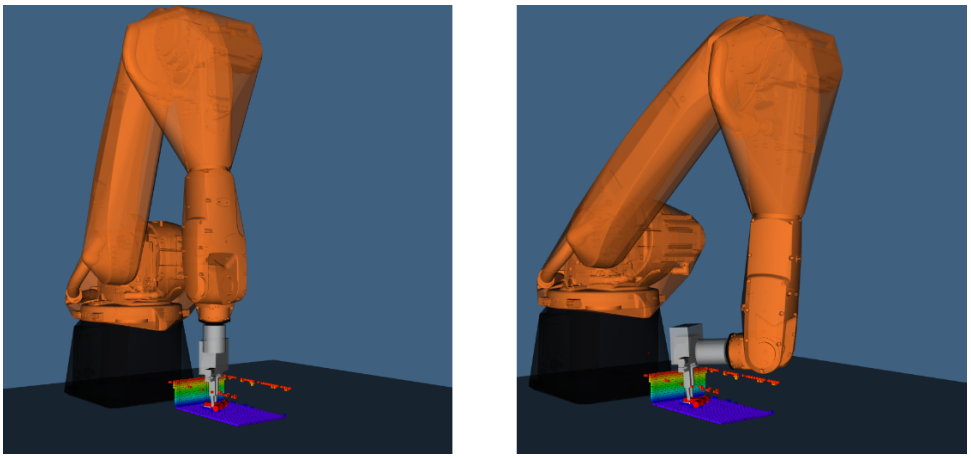


Abb. 27: Kippgelenk für Parallelgreifer

Zusätzliche Gelenke am Greifer werden auch für Parallelgreifer unterstützt. Zuvor waren nur Magnet- und Vakuumgreifer möglich (Geometrie ändert sich nicht beim Öffnen/Schließen). Weitere Details sind in der Dokumentation des *Greiferzustand einstellen*-Task zu finden.

Bibliothek mit Beispielprogrammen

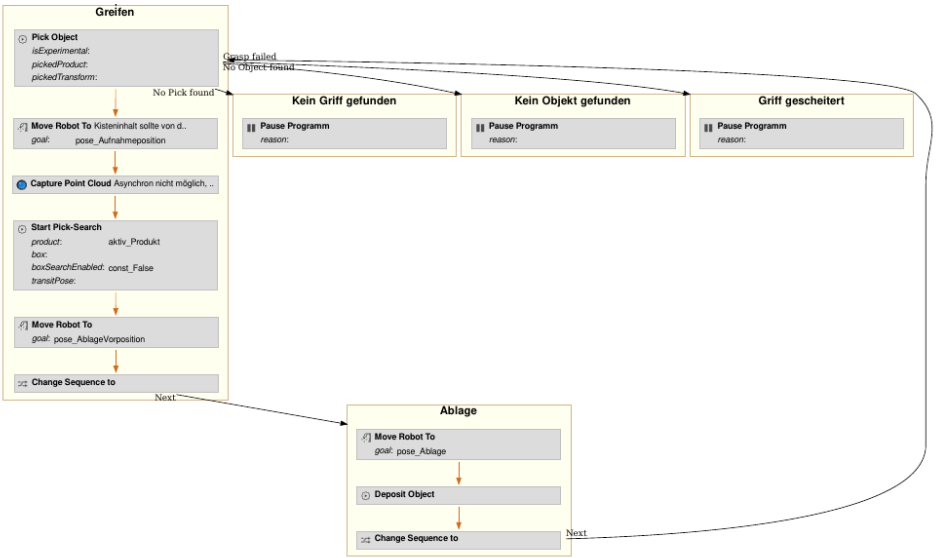


Abb. 28: Beispielprogramm für mobile Kamera

Die Programm-Bibliothek befindet sich auf der Willkommen-Seite und lässt sich mit einem Klick auf das Ordner-Symbol links neben der Überschrift ‚Programme' öffnen. Sie bietet Beispielprogramme die als Ausgangspunkt für Projekt-spezifische Programme geeignet sind. Darunter sind:

- **Select**
  - TCP-Kalibrierung Programm
  - Objekte finden und Übertragen der Ergebnisse via Profinet-Schnittstelle
- **Control**

- TCP-Kalibrierung Programm
- Asynchrone Bildaufnahme mit stationärer Kamera
- Synchrone Bildaufnahme mit mobiler Kamera

If-Else Tasks für bedingtes Ausführen von Teilsequenzen

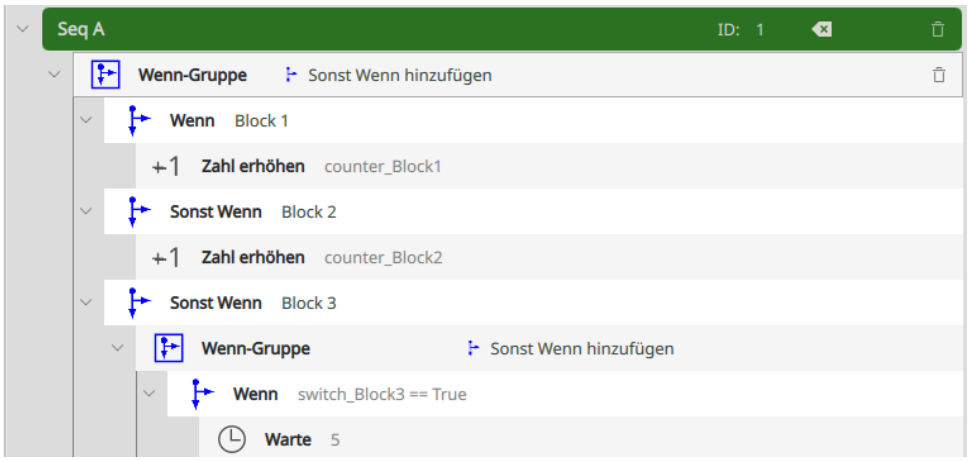


Abb. 29: Verschachtelte Wenn-Gruppen im Programm-Editor

Der neue *Wenn-Gruppe*-Task ermöglicht das bedingte Ausführen von Teilsequenzen. Dieses Feature ermöglicht unter Anderem auch:

- Das Kopieren von Sequenzen
- Das Untergliedern von Sequenzen in Teilsequenzen
- Das parallele Löschen von mehreren Tasks
- Re-Design des Programm-Editors

Rotation auf Jogging Seite

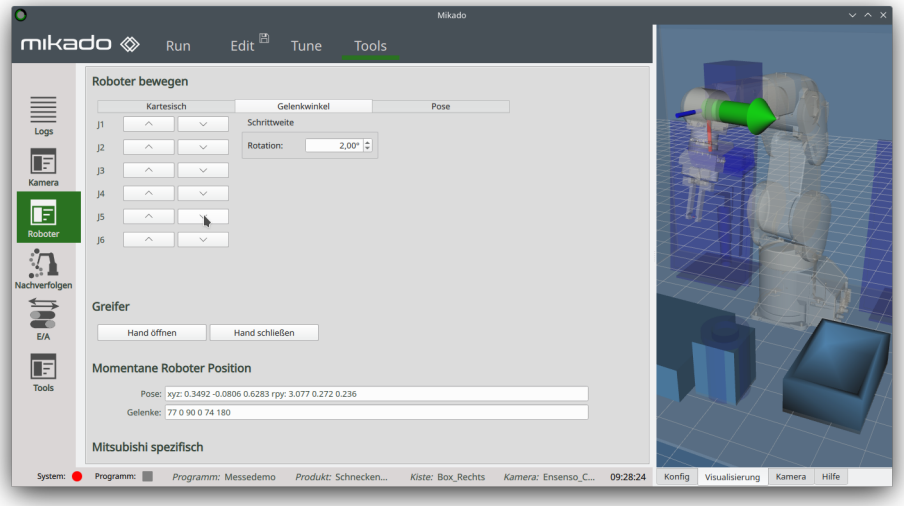


Abb. 30: Jogging Seite: Vorschau der Drehachse durch grünen Pfeil

Zusätzlich zu kartesischen Fahrten in x-, y- und z-Richtung, lässt sich der Roboter auf der Roboter-Seite (siehe *Robotersteuerung*) stückweise rotieren. Dabei kann manuell ausgewählt werden in welchem Koordinatensystem rotiert wird. Zu Verfügung stehen hierbei folgende Koordinatensysteme:

- Welt - Koordinatensystem
- Roboter - Koordinatensystem
- Tool0 (bzw. Roboter-Handgelenk) - Koordinatensystem

Weitere Neuerungen:

- Die Bewegung wird auf der rechten Seite als Vorschau angezeigt
- Re-Design der Roboter-Seite

Depalettierungs-Regeln

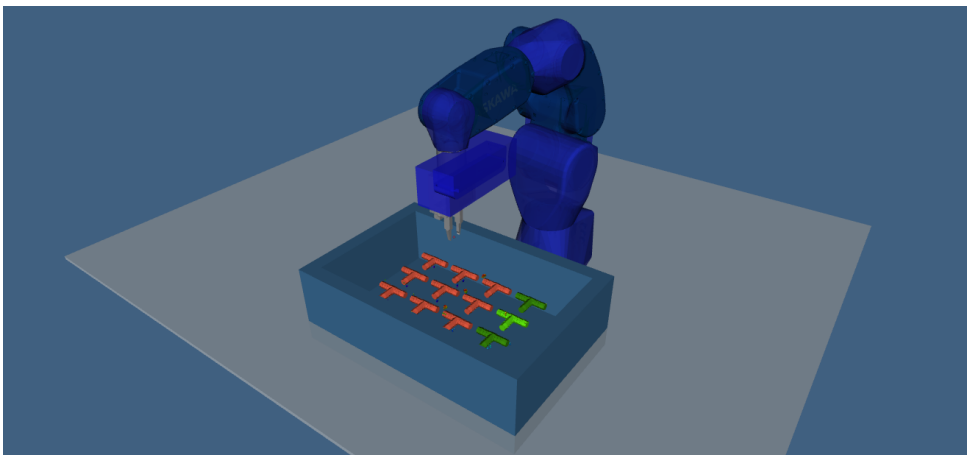


Abb. 31: Nur Griffe mit kleinstem x-Wert sind erlaubt

Der neue Parameter `max_bins_required` ermöglicht das Ausschließen von bestimmte Funden (siehe *Bedingungen*). So können beispielsweise Produkte mit hohem x-Wert relativ zu anderen Produkten ausgeschlossen werden. Dies ermöglicht das Depalettieren von links nach rechts.

Mehr Informationen zu gescheiterten Griffen

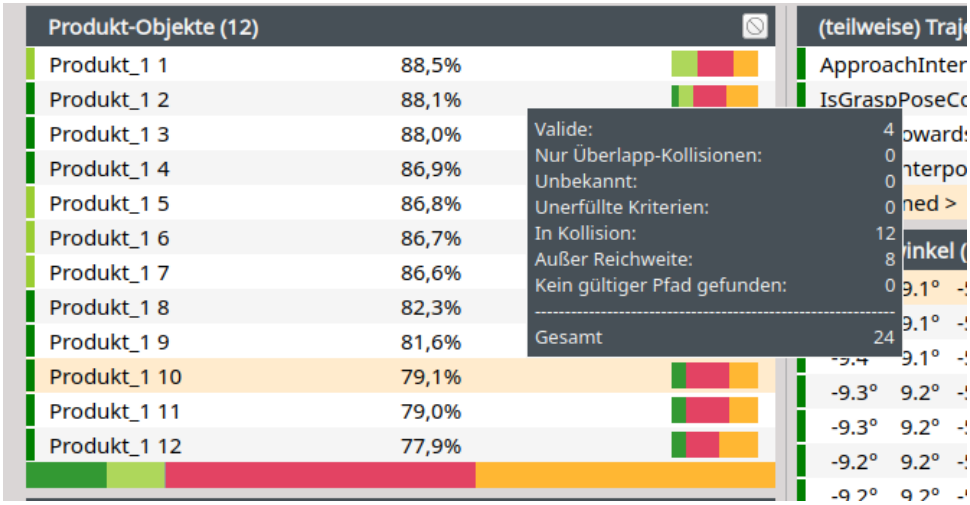


Abb. 32: Re-Design der Griffe-Seite

Die Griffe Seite wurde überarbeitet und bietet jetzt folgende zusätzliche Feature:

- Kollisionspaare haben teilweise zusätzlichen Erklärungstext
- Balken-Diagramme zeigen aktuellen den Zustand der Griffsuche insgesamt und den Produkt-spezifischen Zustand

JavaScript: API für Szenenobjekte

```
var size = scene.getSceneObjectDimensions("ROI1")
scene.setSceneObjectDimensions("ROI1", size.x, size.y, 0.23)
```

Abb. 33: Setzen des z-Wertes einer ROI auf einen fixen Wert

Über die JavaScript-API können nun Größe und Position von Szenen-Objekten gelesen und geschrieben werden (siehe *scene*). Folgende neue Funktionen stehen zur Verfügung:

- `getSceneObjectDimensions`
- `setSceneObjectDimensions`
- `getSceneObjectPose`
- `setSceneObjectPose`

9.1.11 Version 2.1.0

- Integration der Ensenso PartFinder Objekterkennung
- Objekterkennungsparameter werden produktspezifisch abgespeichert und eingestellt
- Gesamtanzahl Griffe wird beim Produkt angezeigt
- Es können sowohl einzelne Griffmuster als auch alle resultierenden Griffe angezeigt werden

Griffinspektion:

- Objekte werden je nach Greifbarkeit eingefärbt und können direkt angeklickt werden
- Griffe können nach Erfolg gefiltert werden

Erweiterungen:

- Die Ergebnisse der Griffsuche können nun per JavaScript abgerufen werden
- JavaScript Kinematik API um zB die Erreichbarkeit von mehreren Posen oder Pfaden zu prüfen
- Die Dimensionen der Punktwolke innerhalb einer ROI kann per JavaScript abgefragt werden
- Kalibrierungen über eine ROS-Schnittstelle (und damit zB über OpcUA oder ProfiNet)

9.2 Szenen-Objekte

9.2.1 Version 5

Warnung: Wenn Ihr Programm einen *Szenenobjekt-Position auslesen* oder *Szenenobjekt-Position setzen* verwendet, kann sich das Verhalten des Programms ändern. Bitte überprüfen Sie Ihr Programm entsprechend.

Die Bedeutung der `Default-Pose` eines Szenenobjekts wurde geändert. In Version kleiner 5 beschreibt die Default-Pose den Mittelpunkt des Bodens eines Szenenobjekt. Für die Ermittlung des Bodens wurde der Parameter `Default-Größe` verwendet. Somit war die tatsächliche Pose des Szenenobjekts abhängig von der Default-Pose und der Default-Größe. Die Abhängigkeit zur Default-Größe wurde entfernt. Ab Version 5 beschreibt die Default-Pose bei Szenenobjekten mit Quader-Geometrie (wie bspw. ROIs) die Lage des Volumenschwerpunkts, statt wie zuvor den Mittelpunkt des Bodens. Bei Szenenobjekten mit Mesh-Geometrie (wie bspw. Kisten) beschreibt die Default-Pose nun direkt die Lage des Ursprungs eines Meshes im Welt-Koordinatensystem. Zusätzlich wurde der neue Parameter `Mesh-Offset` eingeführt, um den Ursprung eines Meshes aus Mikado verschieben zu können.

Mit dem Version-Update werden die verschiedenen Posen-Parameter für Szenen-Objekte automatisch auf die neue Bedeutung umgestellt. Folgende Operation werden durchgeführt:

1. Verschieben der `Default-Pose` um die Hälfte der z-Komponente der Default-Größe d_z in positive z-Richtung des lokalen Koordinatensystems,
2. Verschieben der `Übergangspose` um den gleichen Betrag und in entgegengesetzte Richtung und
3. Setzen der z-Komponente des `Mesh-Offset` auf $-\frac{1}{2}d_z$ (gleicher Betrag wie bei 1 und 2).

Die neue Bedeutung der Default-Pose ist in nachfolgender Abbildung dargestellt.

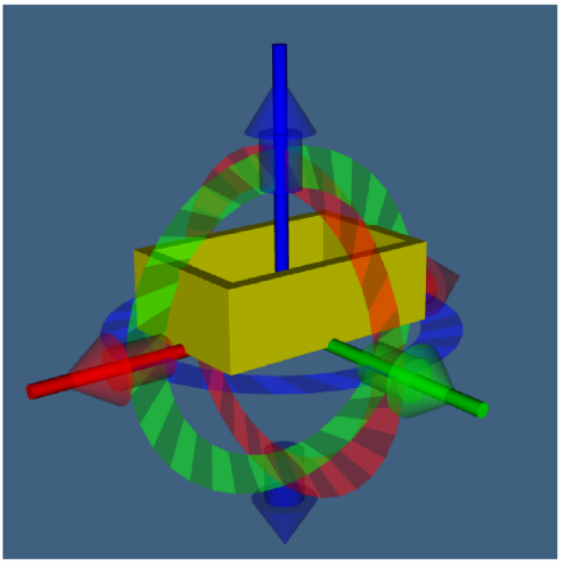


Abb. 34: Alte Bedeutung der `Default-Pose`

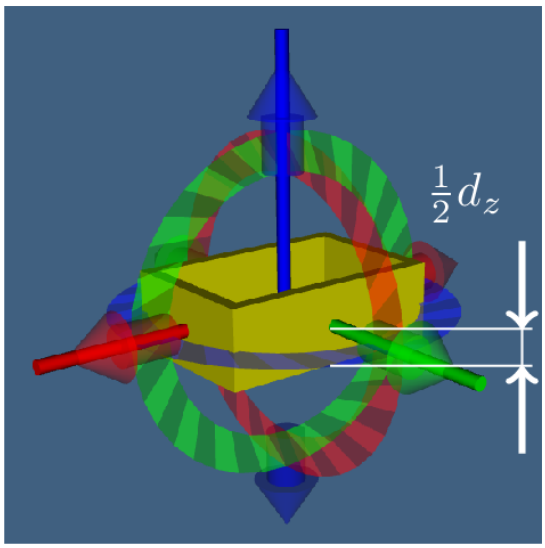


Abb. 35: Neue Bedeutung der `Default-Pose`

Die Default-Größe wird wie in folgenden Screenshots gezeigt in Mikado eingestellt.

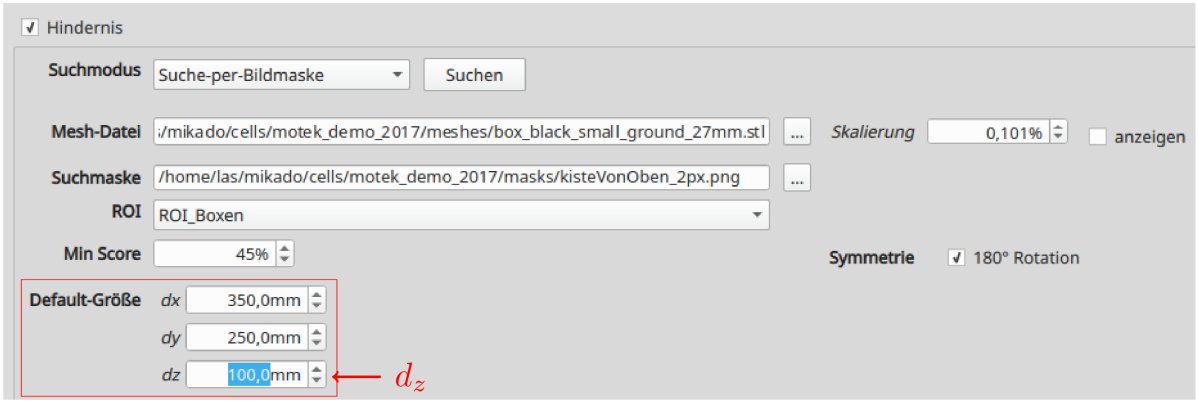


Abb. 36: Einstellung der Default-Größe einer Kiste

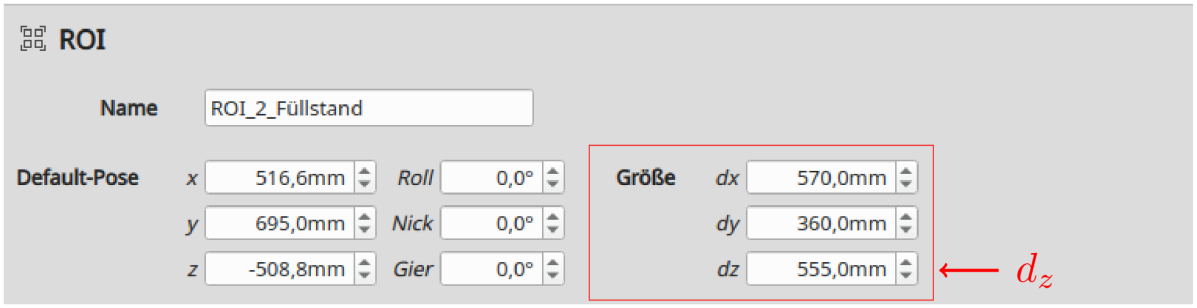


Abb. 37: Einstellung der Default-Größe einer ROI

9.2.2 Version 4

Neue Funktion `ROI` hinzugefügt, die bei der Suche nach Szenenobjekten benötigt wird. Bitte wählen Sie ein ROI-Objekt für jedes Szenenobjekt, das `Search-By-Image` verwendet.

9.2.3 Version 3

Neue Funktion `Symmetrien suchen` hinzugefügt, die bei der Suche nach Szenenobjekten verwendet wird. Standardmäßig deaktiviert.

9.2.4 Version 2

Die Standard-Dateiendung für Szenenobjekte hat sich geändert. Bitte speichern Sie Ihre Mikado-Szenenobjekt-Dateien ab sofort mit der Endung `*.scene`. Die Pfade für das Startprogramm in config/mikado.yaml müssen manuell bearbeitet werden.

KAPITEL 10

Änderungshistorie

Die hier beschriebenen Versionsnummern folgen den Releases des Mikado Handbuchs im Paket **mikado-docs-user-<language>**.

Tab. 1: Liste alle Änderungen

Datum	Version	Details
18.12.2017	1.0.1	Initiale Version
18.01.2018	1.2.1	Mikado Version 1.2
23.02.2018	1.2.2	Mikado Version 1.2 Überarbeitung
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 Überarbeitung
18.02.2019	1.5.1	Mikado Version 1.5
05.03.2019	1.5.2	Mikado Version 1.5 Überarbeitung
27.06.2019	1.6.1	Mikado Version 1.6
12.07.2019	1.6.2	Mikado Version 1.6 Überarbeitung
20.07.2019	1.6.3	Integration erweitert
07.10.2019	1.6.4	Mikado Version 1.6 Überarbeitung und Montageanleitung
09.12.2019	1.7.1	Mikado Version 1.7
23.06.2020	1.7.2	Mikado Version 1.7 Überarbeitung
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	Detailkorrekturen Assistant
24.05.2022	2.1.3	JavaScript API dokumentiert, Hinweise zu Robotergeschwindigkeit
04.08.2022	2.1.4	Yaskawa Motoman ergänzt, andere Roboter detaillierter
05.10.2022	2.1.5	Griffsuche detaillierter
26.10.2022	2.1.6	Kleinere Korrekturen
16.03.2023	2.1.7	PartFinder, Montageanleitung und TCP Kalibrierung
12.04.2023	2.1-8	Alle Tasks dokumentiert
01.07.2023	3.0-0	Mikado Version 3.0
06.02.2024	3.0.1	Roboter Setup wird direkt unter *Roboter-Modelle* beschrieben. Dokumentation ergänzt zu: Doosan Setup, CSV einlesen mit JavaScript, Task „Bewege TCP kartesisch“, „Roboter Nachverfolgen“, Feldbus Rotationskonventionen, ADS Datentypen und Read-only Variablen.
12.03.2024	3.0.2	Übersetzung des Handbuchs auf Englisch, Beschreibung von getCurrentGraspOffset und Verlinkung des „Auf Roboter Warten Task“ aus der Mikado GUI korrigiert.
24.05.2024	3.1.0	Griffpriorisierung nach *Kollisionstiefe*, *Kollisionsprüfung der Greifposen*, Neuheiten in Mikado *Version 3.1.2*

KAPITEL 11

Copyright

© Copyright Optonic GmbH, 2015-2025 – Alle Rechte vorbehalten. – Nachdruck, auch auszugsweise, in jedweder Form sowie mechanische oder elektronische Vervielfältigung nur mit ausdrücklicher schriftlicher Genehmigung der Optonic GmbH. Optonic, ist ein Markenzeichen oder eingetragenes Markenzeichen von Optonic GmbH. Andere hier genannte Markennamen sind Eigentum der jeweiligen Markeninhaber.

Mikado Control verwendet die folgenden freien Bibliotheken unter den Bedingungen der LGPL V. 3.0:

- Qt: http://www.qt.io
- breeze icon theme: https://github.com/KDE/breeze-icons

Optonic stellt Ihnen innerhalb einer Frist von drei Jahren nach Erhalt dieser Software den Quelltext dieser Komponenten in der verwendeten Form auf Anfrage zur Verfügung. Den genauen Wortlaut der LGPL V.3 finden Sie nach Abschluss dieser Übersicht.

Mikado Control verwendet die folgenden freien Bibliotheken unter den Bedingungen der 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

Die Liste der angegebenen ROS Pakete ist nicht als abschliesend zu betrachten, genaue Hinweises finden sich bei den installierten Paketen und werden hier gerne ergänzt.

Im folgenden der genaue Wortlaut der LPGL V.3 sowie der BSD-Lizenz:

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.