isys vision

Handbuch



Adaptive Robot Control

Version 1.7

Ein Produkt von
isys vision
www.mikado-robotics.de

isys vision GmbH | Zollhallenstr. 11 | 79106 Freiburg | Deutschland/Germany
Tel.: +49 761–79095–0 | E-Mail: info@isys-vision.de
© Copyright isys vision GmbH, 2015-2020

Änderungshistorie

Datum	Dokumenten-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

Inhaltsverzeichnis

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

2 Verwendung **3**
- 2.1 Navigation . . . 3
- 2.2 Startseite . . . 5
- 2.3 Bediener-Seite . . . 5
- 2.4 Einrichtungs-Seite . . . 5
- 2.5 Statusseite . . . 5
  - 2.5.1 Steuerbuttons . . . 6
  - 2.5.2 Programmablauf . . . 6
- 2.6 Statusleiste . . . 7
  - 2.6.1 Systemstatus . . . 8
- 2.7 Passwortschutz . . . 9
- 2.8 Programm-Editor . . . 9
  - 2.8.1 Mikado-Programme . . . 9
  - 2.8.2 Grafische Oberfläche . . . 11
  - 2.8.3 Toolbar . . . 12
  - 2.8.4 Programm erstellen . . . 13
  - 2.8.5 Programm-Validator . . . 14
- 2.9 Variablen-Editor . . . 14
  - 2.9.1 Variablentabelle . . . 15
  - 2.9.2 Bearbeitung von Variablen . . . 15
  - 2.9.3 Neue Variable anlegen . . . 17
  - 2.9.4 Variablen aus Datei importieren . . . 17
  - 2.9.5 Visualisierung der Roboter-Pose . . . 18
- 2.10 Produktverwaltung . . . 21
  - 2.10.1 Überblick . . . 21
  - 2.10.2 Name . . . 22
  - 2.10.3 Aktives Produkt . . . 22
  - 2.10.4 Objektsuche . . . 22

2.10.5 Greifprozess . . . 24
2.10.6 Greifposen . . . 26
2.11 Szenenverwaltung . . . 29
2.11.1 Überblick . . . 29
2.12 Objekterkennungs-Tool . . . 33
2.12.1 Erste Schritte . . . 34
2.12.2 Kamera-Auswahl . . . 35
2.12.3 Funktionsweise der Objekterkennung . . . 35
2.12.4 Objekterkennungsparameter . . . 35
2.12.5 Ansicht . . . 37
2.12.6 Letzte Aufnahme speichern . . . 39
2.12.7 Parameter übernehmen . . . 40
2.13 Griff-Inspektion . . . 41
2.13.1 Überblick . . . 41
2.13.2 Greifbewegungssuche . . . 41
2.13.3 Inspektion der Greifbewegungssuche . . . 45
2.14 Profiler . . . 49
2.15 Logs . . . 50
2.15.1 Überblick . . . 50
2.16 Kamera-Einstellungen . . . 50
2.16.1 Livebild 2D . . . 51
2.16.2 Kamera-Tab . . . 52
2.16.3 Kalibrierung . . . 53
2.17 Robotersteuerung . . . 56
2.17.1 Roboterkontrolle . . . 56
2.17.2 Roboterparameter . . . 57
2.18 System-Tools . . . 58

3 Tasks **59**
3.1 Ablauf . . . 59
3.1.1 Zu Sequenz wechseln . . . 59
3.1.2 Programm beenden . . . 60
3.1.3 Fehler . . . 60
3.1.4 Warten . . . 60
3.1.5 Programm unterbrechen . . . 60
3.2 Zahlen . . . 61
3.2.1 Zahl erhöhen . . . 61
3.2.2 Zahl verringern . . . 61
3.2.3 x kleiner als y? . . . 61
3.2.4 x größer als y? . . . 62
3.2.5 x gleich y? . . . 63
3.3 Grundlegendes . . . 64
3.3.1 Kopieren . . . 64
3.3.2 Werte tauschen . . . 64
3.3.3 Variable zurücksetzen . . . 65
3.3.4 Ausgangswert setzen . . . 65
3.3.5 Negiere Boolsche Variable . . . 65
3.3.6 Kopiere Eintrag aus Liste . . . 65

3.3.7 Kopiere nächsten Eintrag aus Liste . . . 66
3.3.8 Zur Liste hinzufügen . . . 67
3.3.9 Setze Listenelement . . . 67
3.3.10 Lösche Element aus Liste . . . 67
3.3.11 Liste leeren . . . 68
3.3.12 Liste leer? . . . 68
3.4 Roboterkontrolle . . . 69
3.4.1 Bewege Roboter nach . . . 69
3.4.2 Bewege Tcp nach . . . 70
3.4.3 Bewege Roboter kartesisch nach . . . 71
3.4.4 Bewege Teil nach . . . 72
3.4.5 Bewege Teil kartesisch nach . . . 73
3.4.6 Roboterposition auslesen . . . 74
3.4.7 Teileposition auslesen . . . 74
3.4.8 Ist Roboter Position mit Teil erreichbar? . . . 75
3.4.9 Hand öffnen/schließen . . . 75
3.4.10 Roboterprogramm ausführen . . . 76
3.4.11 Roboterausgangssignal setzen . . . 77
3.4.12 Roboterausgangssignal lesen . . . 78
3.4.13 Robotereingangssignal lesen . . . 78
3.4.14 Auf Robotereingangssignal warten . . . 79
3.4.15 Teil übergeben . . . 80
3.5 Bildverarbeitung . . . 80
3.5.1 Setze aktive Kamera . . . 80
3.5.2 Kamera Konfiguration laden . . . 81
3.5.3 Punktwolke aufnehmen . . . 81
3.5.4 Punktwolke asynchron aufnehmen . . . 82
3.5.5 Mehrfachaufnahme . . . 82
3.5.6 Kamerapose nachjustieren . . . 83
3.5.7 Anzahl der Punkte ermitteln . . . 85
3.5.8 Bild speichern . . . 85
3.5.9 ROI setzen . . . 86
3.5.10 Patternpose bestimmen . . . 86
3.6 Szenenobjekt . . . 87
3.6.1 Setze aktives Szenenobjekt . . . 87
3.6.2 Suche Szenenobjekt . . . 88
3.6.3 Szenenobjekt-Position auslesen . . . 88
3.6.4 Szenenobjekt-Position setzen . . . 89
3.7 Objekt-Handhabung . . . 90
3.7.1 Setze aktives Produkt . . . 90
3.7.2 Griffsuche starten . . . 90
3.7.3 Griffsuche abbrechen . . . 91
3.7.4 Finde Objekt . . . 91
3.7.5 Teil greifen . . . 92
3.7.6 Teil ablegen . . . 94
3.7.7 Funde löschen . . . 95
3.7.8 Statisches Objekt hinzufügen . . . 95
3.7.9 Statisches Objekt entfernen . . . 95

3.8 Transformationen . . . 96
3.8.1 Pose Auftrennen . . . 96
3.8.2 Pose Erstellen . . . 96
3.8.3 Rotiere Pose Global . . . 97
3.8.4 Rotiere Pose Lokal . . . 99
3.8.5 Verschiebe Pose Global . . . 101
3.8.6 Verschiebe Pose Lokal . . . 103
3.8.7 Pose transformieren . . . 105
3.8.8 Inverse Transformation . . . 105
3.9 Erweiterungen . . . 105
3.9.1 JavaScript . . . 105
3.9.2 Halcon Script ausführen . . . 106

4 Tutorials **107**
4.1 Ein erster Ablauf in zehn Minuten . . . 107
4.1.1 Aufnahmepose einrichten . . . 107
4.1.2 Szene einrichten . . . 107
4.1.3 Produkt einrichten . . . 108
4.1.4 Produkt speichern . . . 108
4.1.5 Minimalprogramm erstellen . . . 109
4.1.6 Posen einlernen und Variablen anpassen . . . 109
4.1.7 Laufen lassen und etwas variieren . . . 109
4.2 Ein neues Produkt bis zur Objekterkennung . . . 110
4.2.1 Neues Produkt / Produktliste einrichten . . . 110
4.2.2 Objekterkennung optimieren . . . 111
4.3 Meshlab . . . 114
4.3.1 Grundlagen: STL laden und speichern . . . 114
4.3.2 Produktkoordinatensystem zentrieren . . . 115
4.3.3 STL skalieren . . . 117
4.3.4 STL Anzahl der Dreiecke reduzieren . . . 119
4.3.5 STL Flächen entfernen . . . 121
4.3.6 STL Innenliegende Flächen automatisch entfernen . . . 127
4.4 Halcon Script einbinden: Datamatrix Erkennung . . . 133
4.4.1 Vorbereitung . . . 133
4.4.2 Programmablauf . . . 134
4.4.3 Hdev-Script für eine Erkennung von Datamatrizen / QR-Codes erstellen 136
4.4.4 Programm ausführen und testen . . . 139
4.5 Javascript: Benutzerdefinierter Dateilog aus Mikado . . . 141
4.5.1 Passendes Mikado Programm . . . 142
4.5.2 Variablen des Mikado Programms . . . 142
4.5.3 Skript Details . . . 142

5 Simulation **145**
5.1 Roboter-Simulation . . . 145
5.2 Aufgezeichnete Bilddaten (Filecam) . . . 146
5.3 Simulation basierend auf Mesh-Dateien . . . 146
5.3.1 Simulator - Konfiguration . . . 146
5.3.2 Simulator - Automatischer Modus . . . 148

5.3.3 Simulator - Anpassung der Kameraeinstellungen mit Szene aus Simulator . . . 149

6 Roboter-Modelle **157**
6.1 Mitsubishi . . . 157
6.1.1 Parameter und Steuerelemente . . . 157
6.1.2 Roboter einschalten . . . 158
6.1.3 Sonstige Hinweise . . . 158
6.2 Universal Robots . . . 158
6.2.1 Parameter und Steuerelemente . . . 158
6.2.2 Roboter einschalten . . . 159
6.2.3 Sonstige Hinweise . . . 159
6.3 Weitere Roboter . . . 159

7 Installation **161**
7.1 Übersicht . . . 161
7.1.1 Hardware-Übersicht . . . 161
7.1.2 Software-Übersicht . . . 163
7.1.3 Integrationsablauf (Schritt-für-Schritt Anleitung) . . . 164
7.2 Konfigurations-Backup . . . 165
7.3 Erstellen einer neuen Zelle . . . 165
7.3.1 Neue ARC Zelle Erstellen . . . 165
7.4 Anpassen einer vorhandenen Zelle . . . 166
7.4.1 Konfigurationsdateien . . . 166
7.4.2 Launchfiles . . . 174
7.4.3 Zellgeometrie . . . 174
7.5 Verzeichnisstruktur . . . 179
7.6 Greifer-Integration . . . 179
7.6.1 Greifertypen . . . 179
7.6.2 Greifer einbinden . . . 180
7.6.3 Mikado-Konfiguration . . . 184
7.6.4 Greifer Anschlussvarianten . . . 184
7.6.5 TCP kalibrieren . . . 184
7.7 Hardware einrichten . . . 185
7.7.1 Netzwerkadapter . . . 185
7.7.2 Ensenso . . . 187
7.7.3 Roboter-Controller einrichten . . . 189
7.7.4 ProfiNet mit Hilscher . . . 222
7.7.5 Wago/Beckoff IO . . . 223

8 Mikado ARC Erweitern **225**
8.1 JavaScript Task . . . 225
8.2 Halcon Script Task . . . 227
8.3 Benutzerdefinierte Objekterkennung mit Halcon . . . 232
8.3.1 Hdev-Skript . . . 233
8.3.2 Anpassen der Konfigurationsdatei . . . 234
8.4 Eigene Seiten mit QML . . . 235
8.4.1 Konfiguration . . . 235
8.4.2 Einführung in QML . . . 236

8.4.3 Weiterführende Informationen 237
8.5 Externe Ansteuerung mit ROS/Python 238
8.5.1 Beispielprogramm 238
8.5.2 Schnittstellenbeschreibung 238
8.5.3 Launchfile 246

9 Copyright **249**
9.1 GNU LESSER GENERAL PUBLIC LICENSE 250
9.2 BSD 3-Clause License 252

KAPITEL 1

Einleitung

Mikado Adaptive Robot Control (kurz Mikado ARC) ist eine grafische Software für den Griff in die Kiste (Bin Picking). Sie bündelt Lösungen für alle Teilprobleme von Bildaufnahme und Objekterkennung über Pfadplannung und Robotersteuerung bis hin zur Kommunikation mit externen Steuerungen. Trotz dieser teils komplexen Aufgabengebiete ermöglicht Ihnen Mikado ARC 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 3D' 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* mit vertiefendem Wissen im Bereich *Roboter-Modelle*. Sektion *Mikado ARC Erweitern* zeigt die verschiedenen Schnittstellen und Erweiterungsmöglichkeiten 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 zum Kern Mikado Programm, z.B. „motek_demo", 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 Zelldefinition werden diese Verknüpfungen automatisch angelegt. Geht ein Controller in den Dauerbetrieb kann auch 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 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 unter ensenso.com.

KAPITEL 2

Verwendung

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

2.1 Navigation

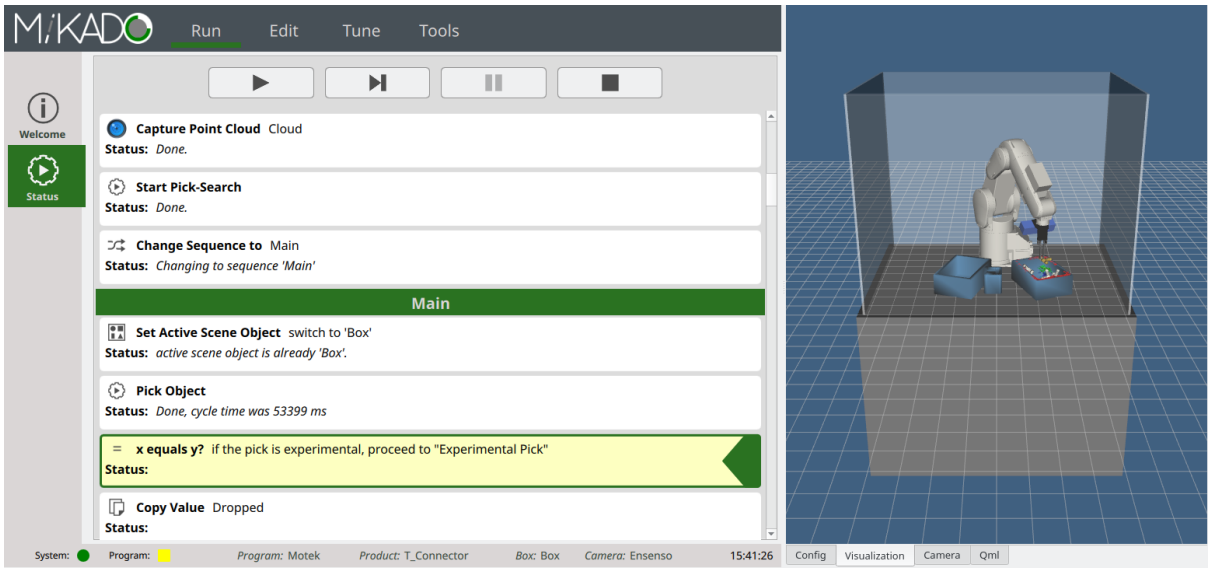


Abb. 1: Mikado-Oberfläche mit Statusseite

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 ARC 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 ARC 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 ARC 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.

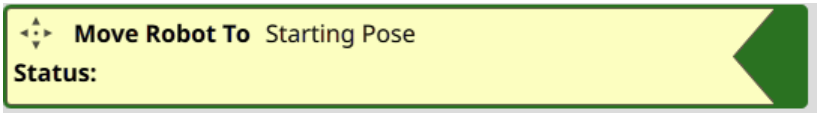


Abb. 3: Programmcursor

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.

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.

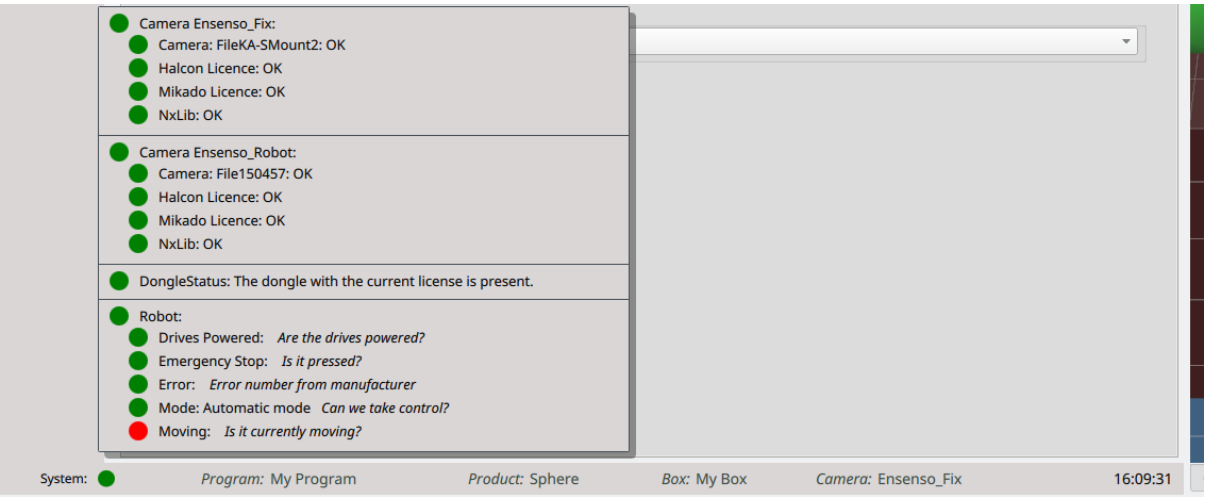


Abb. 4: Statusleiste und Systemstatus

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

- **Systemstatus** *s. unten*
- **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 (s. *Aktives Produkt*)
- **Kiste** Aktive Kiste. Die aktive Kiste kann sich während der Programmausführung verändern (s. *Setze aktives Szenenobjekt*). In diesem Bereich der Statusleiste sehen Sie die aktuell aktive Kiste.
- **Kamera** Aktive Kamera. Die aktive Kamera kann sich während der Programmausführung verändern (s. *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 schneller Indikator, ob alle Hardware-Komponenten Ihres Systems korrekt arbeiten. Der farbige Kreis in der Statusleiste links zeigt an, ob ein Problem vorliegt. Funktionieren alle konfigurierten Komponenten wie z.B. Roboter, Kamerasysteme, Lizenzdongles oder Feldbussysteme, so färbt sich dieser Kreis ‚grün'. Erkennt Mikado ARC 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 Kreis. Nun erscheint eine Übersicht der einzelnen Komponenten mit jeweils einem Statusindikator und Anmerkung.

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.

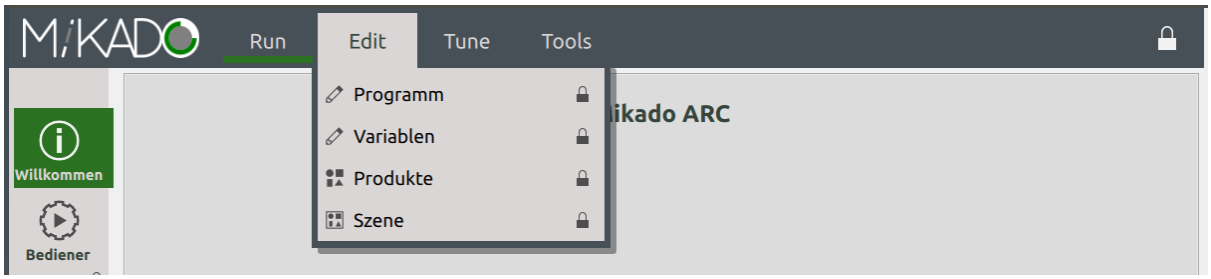


Abb. 5: Screenshot Menü mit aktiviertem Passwortschutz

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.

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

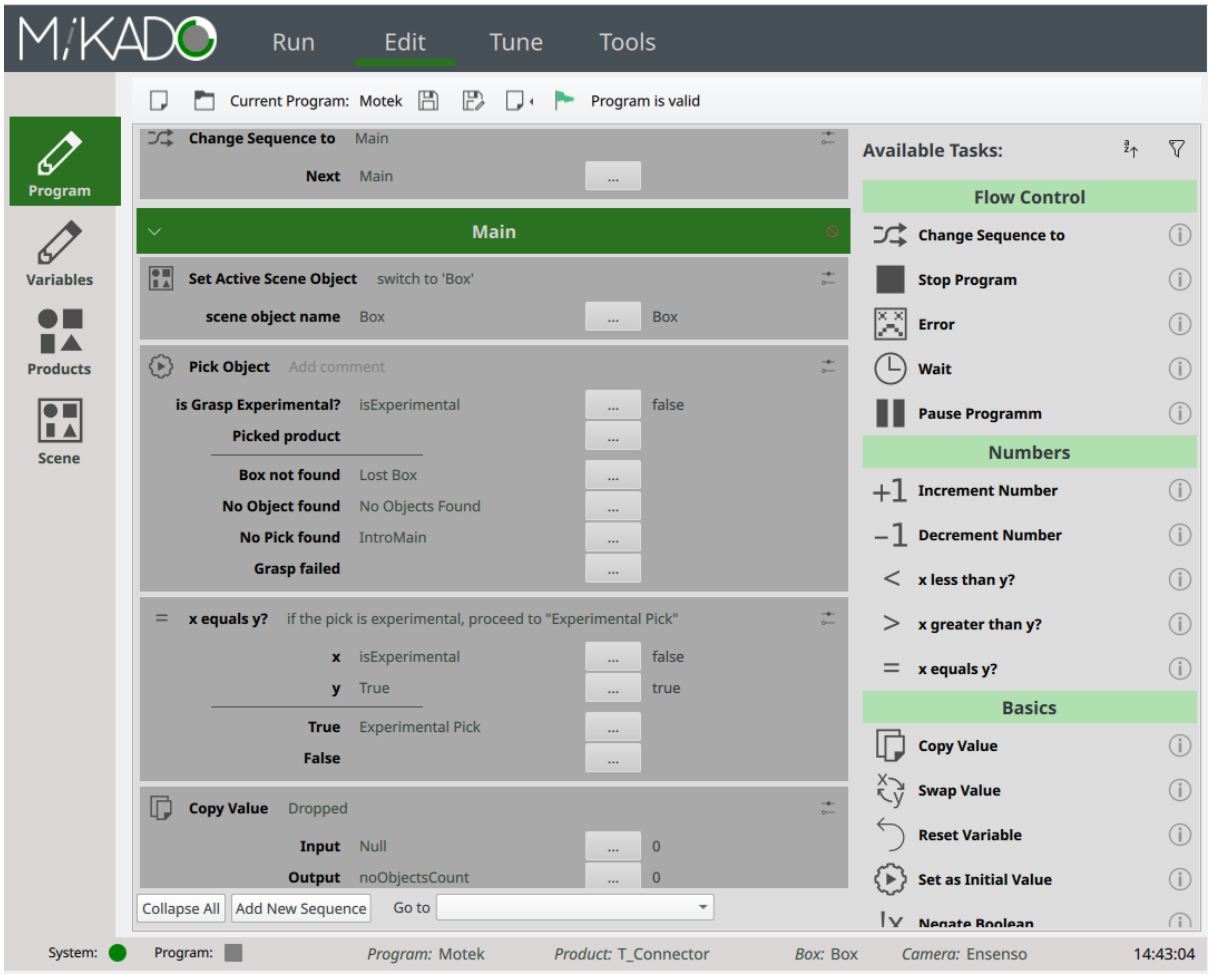


Abb. 6: Programm-Editor

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.

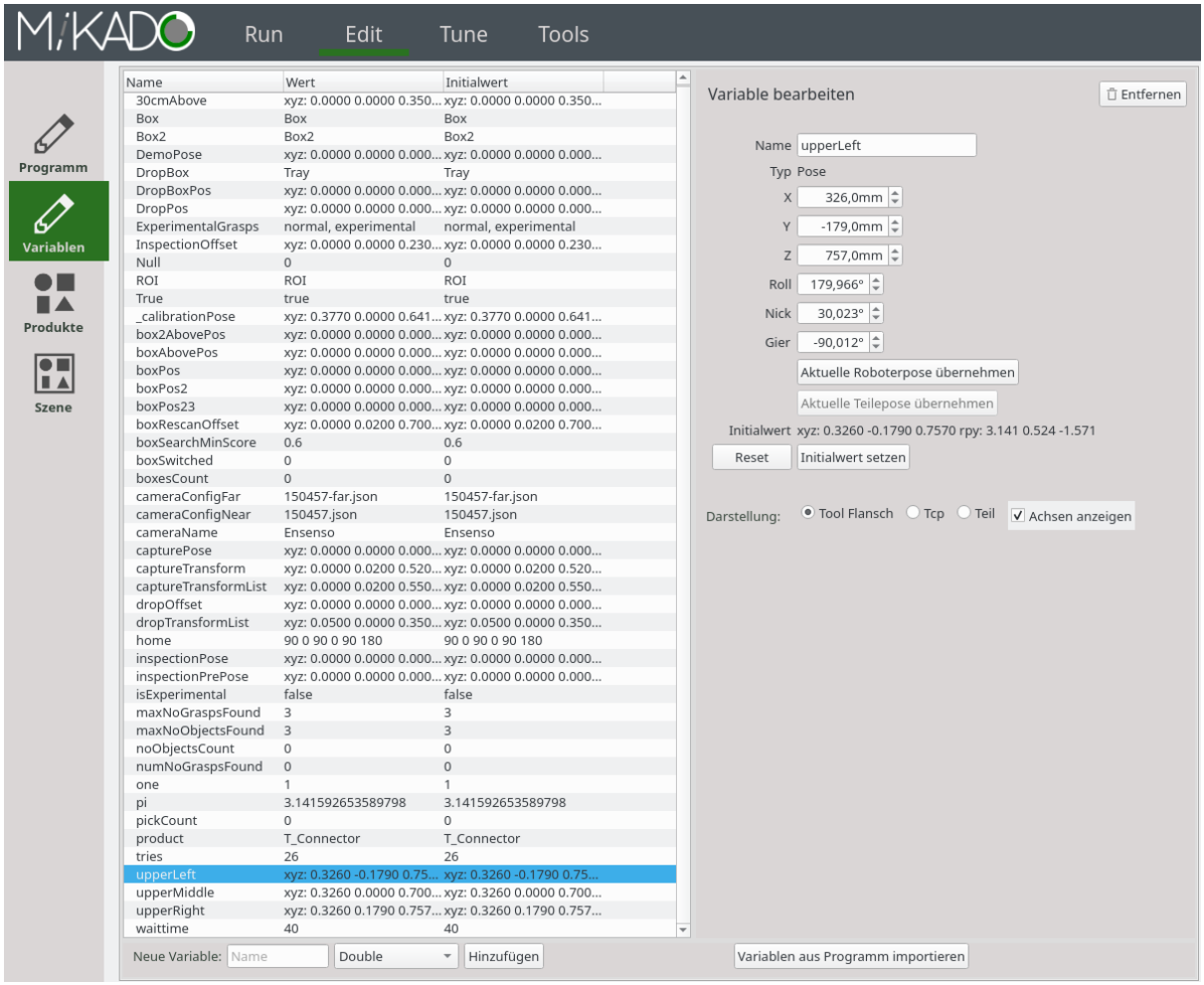


Abb. 8: Variablen-Editor

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:

- ***Name*** Variablenname
- ***Typ*** Variablentyp

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.
- ***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 Neue Variable anlegen

Unter der Variablentabelle sind Eingabefelder für das Anlegen einer neuen Variable zu finden. Geben Sie hier einen noch nicht verwendeten Variablennamen ein und wählen den gewünschten Variablentyp aus. Nach einem Klick auf ‚Hinzufügen' erscheint die neue Variable in der Tabelle und kann wie gewohnt editiert 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.

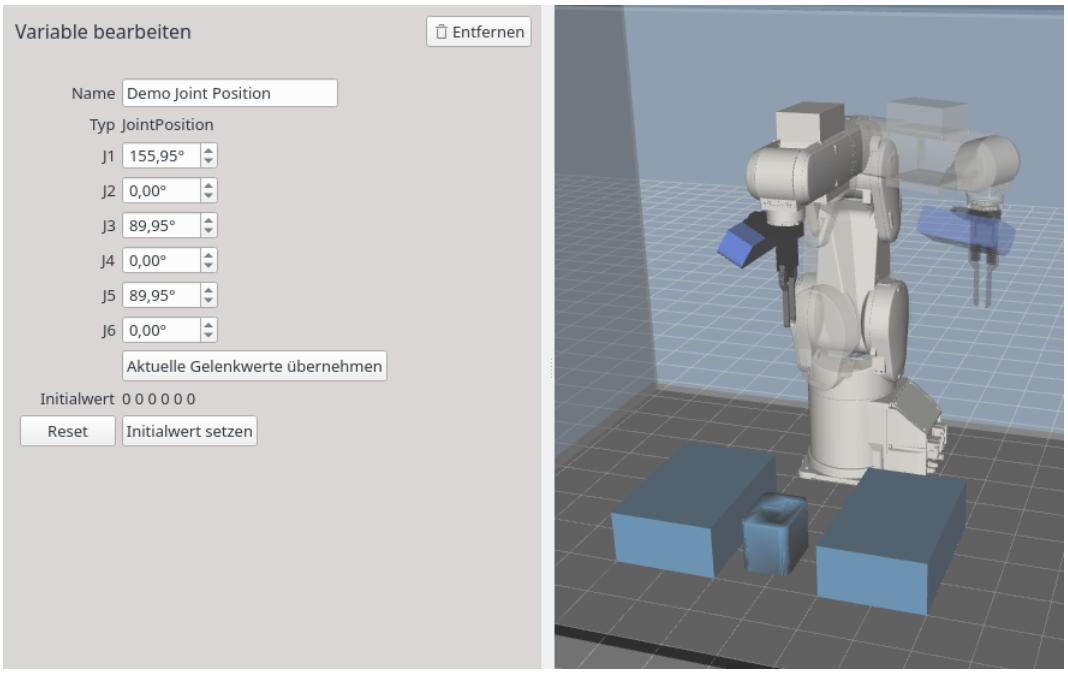


Abb. 9: Visualisierung einer JointPosition. Links die tatsächliche momentane Roboterposition, rechts das transparente Modell als Vorschau der ausgewählten JointPosition.

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.

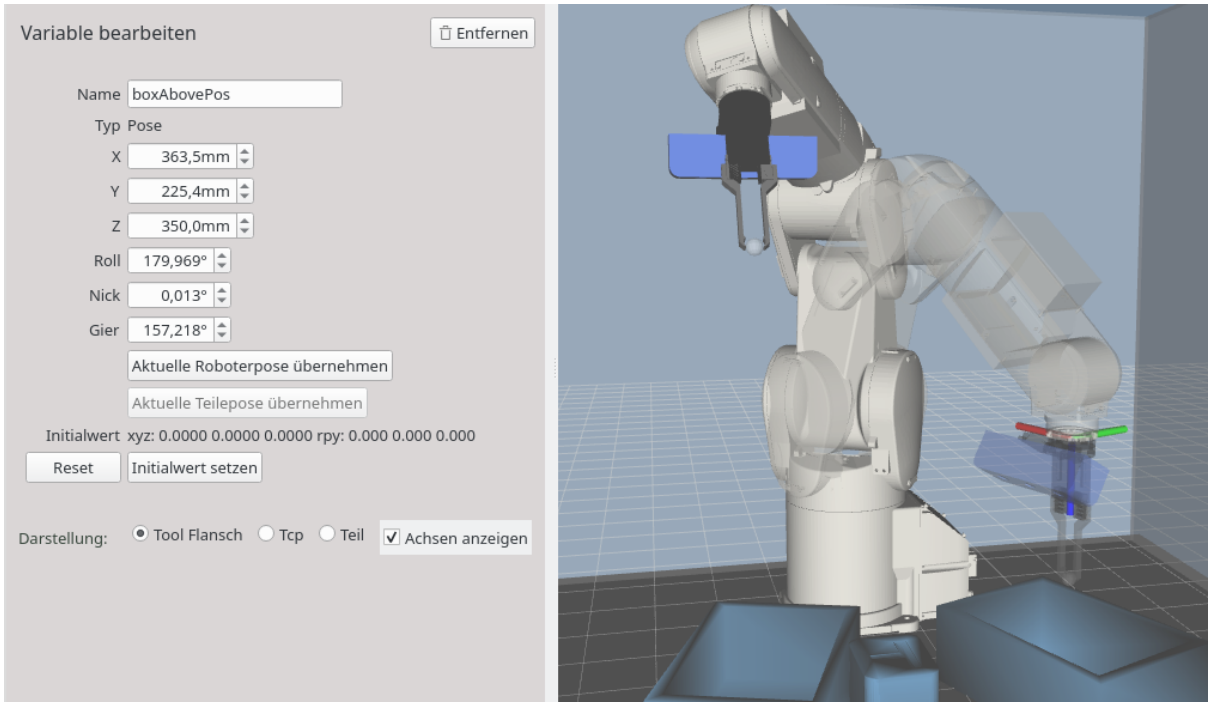


Abb. 10: Visualisierung einer Roboterpose im ‚Handgelenk' Modus

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.

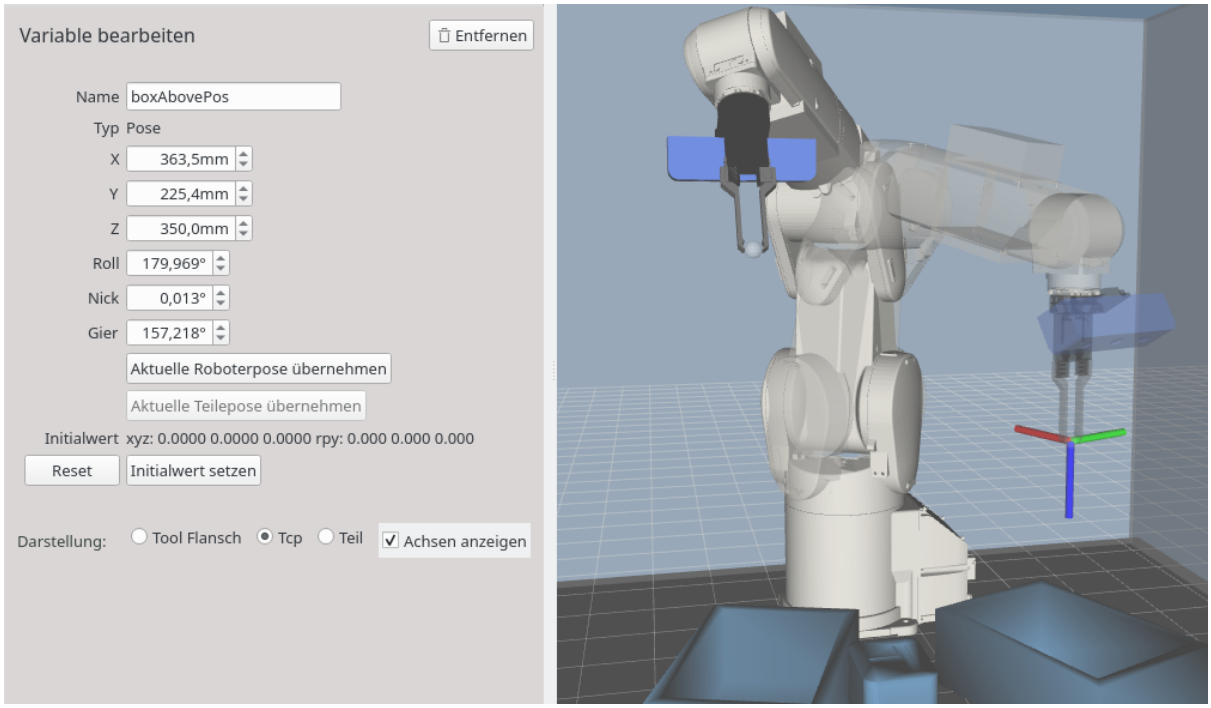


Abb. 11: Visualisierung einer Roboterpose im ‚TCP' Modus

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.

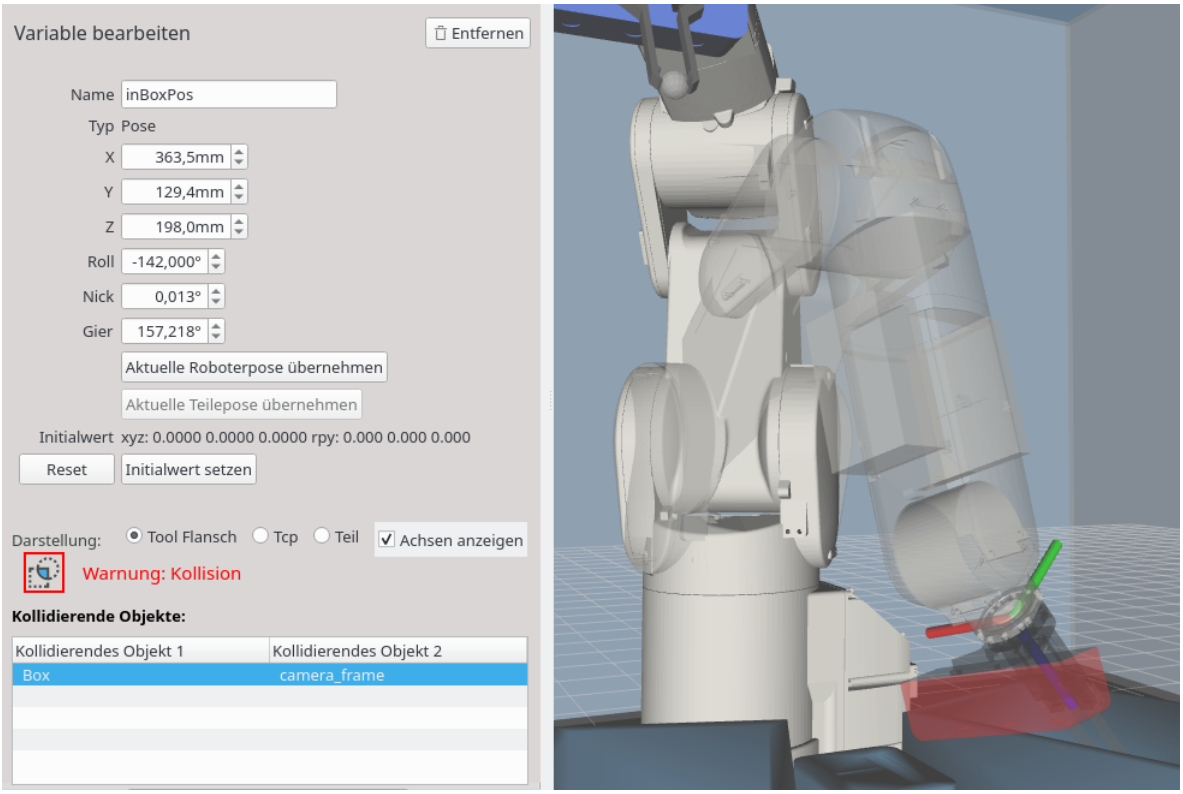


Abb. 12: Visualisierung mit Kollisionen am Greifer

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).

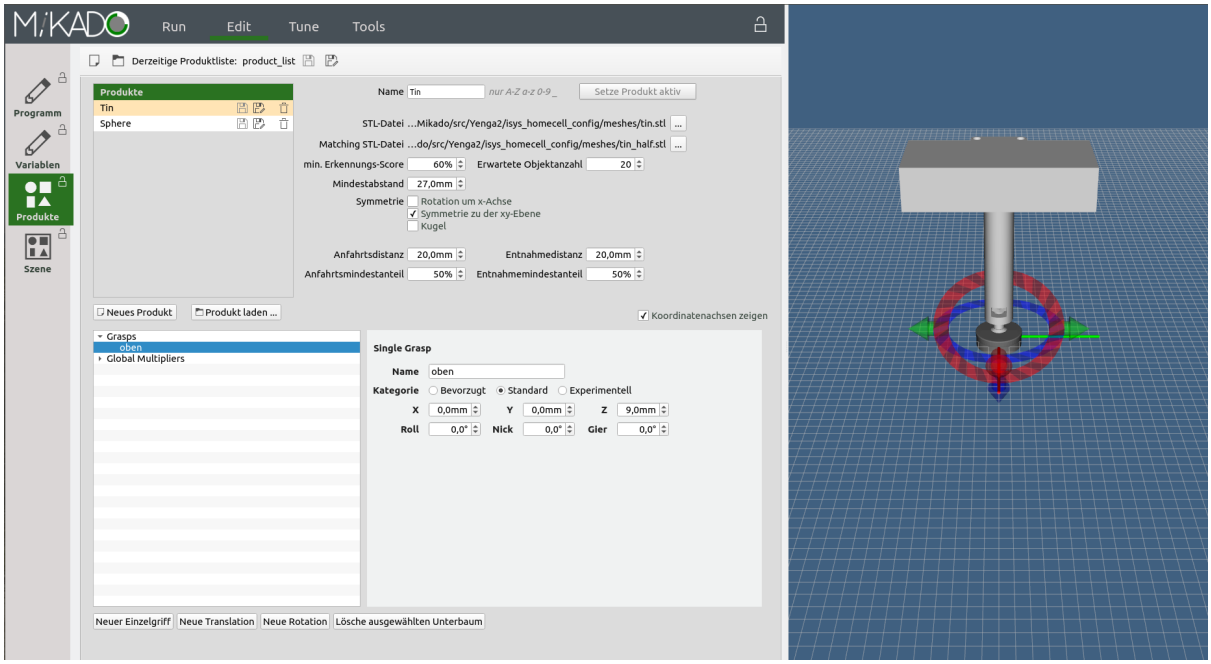


Abb. 13: Screenshot Produktverwaltung mit Einzelgriff

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.

„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.

„Ist kugelförmig?“ - „symmetrisch zur xy-Ebene“ - „rotationssymmetrisch um die x-Achse“

Hier werden Symmetrien des Produktes angegeben. Diese haben Auswirkungen auf die Orientierung der Pose eines erkannten Objektes. Ist ein Objekt bspw. kugelförmig, so ist die Orientierung eines gefundenen Objektes immer so, dass dessen Achsen parallel zu den entsprechenden Achsen des globalen Koordinatensystems sind (x parallel zu x, y parallel zu y, z parallel zu z). Ist das Produkt symmetrisch zur xy-Ebene oder rotationssymmetrisch um die x-Achse , so wird ein gefundenenes Objekt stets so ausgerichtet, dass dessen z-Achse „nach oben“ zeigt. (Das Skalarprodukt zwischen der z-Achse des Objekts und der z-Achse der Szene ist nichtnegativ.)

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

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

2.10.5 Greifprozess

Für jedes Produkt gibt es mehrere potenzielle Greifposen (definiert im Produkt-Koordinatensystem). Diese werden für jedes gefundene Objekt auf die Objektlage multipliziert. Sind also N mögliche Greifposen für ein Produkt definiert und es werden bei einer Objektsuche M Instanzen dieses Produkts gefunden, so gibt es „N x M“ mögliche Greifposen im Raum, für die geprüft wird, ob ein kollisionsfreies Greifen möglich ist. Das Definieren der Greifposen wird im Absatz „Greifposen“ erläutert.

Der Greifprozess wiederum ist grundsätzlich in die folgenden Schritte unterteilt:

Anfahren Ausgangspose → Anfahrtspose

Greifen Anfahrtspose → Greifpose

Entnehmen Greifpose → Entnahmepose

Zurückfahren Entnahmepose → Endpose

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

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

Die Anfahrts- und Entnahmepose werden in Abhängigkeit der Greifpose bestimmt.

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 (üblicherweise die Richtung, in die der Greifer „zeigt“). Die Distanz zwischen Anfahrts- und Greifpose wird im Parameter „Anfahrtsdistanz“ eingestellt.

Zur Berechnung der Entnahmepose wird wiederum eine bestimmte Distanz von der Greifpose aus „rückwärts“ gerechnet. Hierbei gibt es drei verschiedene Varianten von „rückwärts“:

a) in z-Richtung im Koordinatensystem der Greifpose (üblicherweise die Richtung, in die der Greifer „zeigt“)

b) in Richtung der Endpose (im globalen Koordinatensystem)

c) in z-Richtung im globalen Koordinatensystem („nach oben“)

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 im Parameter „Entnahmedistanz“ eingestellt.

Beim Überprüfen des Greifens und des Entnehmens (s.o.) wird jeweils von der Greifpose aus die Strecke hin zur Anfahrts- bzw. Entnahmepose auf Kollisionen überprüft. Dabei kann eingestellt werden, dass nicht die gesamte Strecke zurückgelegt werden muss, sondern nur ein bestimmter Prozenzsatz. Ist beispielsweise eingestellt, dass nur 70 % bis zur Entnahmepose gefahren werden müssen und nach 80 % würde eine Kollision auftreten, so wird die tatsächliche Entnahmepose auf die Pose nach 79 % gesetzt und von dieser aus wird frei (zur Endpose hin) geplant. Dies ermöglicht es, Anfahrts- und Entnahmedistanz für den Idealfall zu definieren und andererseits jeweils eine Mindestdistanz zu definieren, die in gerader Richtung abgefahren werden muss. Die Prozenzsätze (Mindestdistanz/Idealdistanz) werden in den Parametern „Anfahrtsmindestanteil“ und „Entnahmemindestanzeil“ eingestellt.

Im Allgemeinen verhindert die Bahnplanung jegliche Kollisionen zwischen Roboter, Zelle, Szenenobjekten, Produktobjekten und 3D-Punktwolke. Während des Entnehmens (s.o. Greifpose $\rightarrow$ 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.

„Anfahrtsdistanz“ - „Entnahmedistanz“

Hier wird die Distanz zwischen Greif- und Anfahrts- bzw. zwischen Greif- und Entnahmepose festgelegt.

„Anfahrtsmindestanteil“ - „Entnahmemindestanzeil“

Hier wird die Mindestdistanz beim Greifen und Entnehmen als Prozenzsatz der Anfahrts- und Entnahmedistanz festgelegt.

2.10.6 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)
```

(continues on next page)

(continued from previous page)

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

Globale Multiplikatoren

Hier können Multiplikatoren definiert werden, die an jedes Blatt des Greifmultiplikator-Baumes 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
```

2.11 Szenenverwaltung

2.11.1 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 optionale 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- und y-Koordinaten der Default-Pose die Zentrums-Koordinaten der Region sind, während die z-Koordinate die Höhe des Bodens der Region angibt. Somit steht eine Kiste, deren Default-Pose-Einträge alle den Wert 0 haben, auf der (z=0)-Ebene.

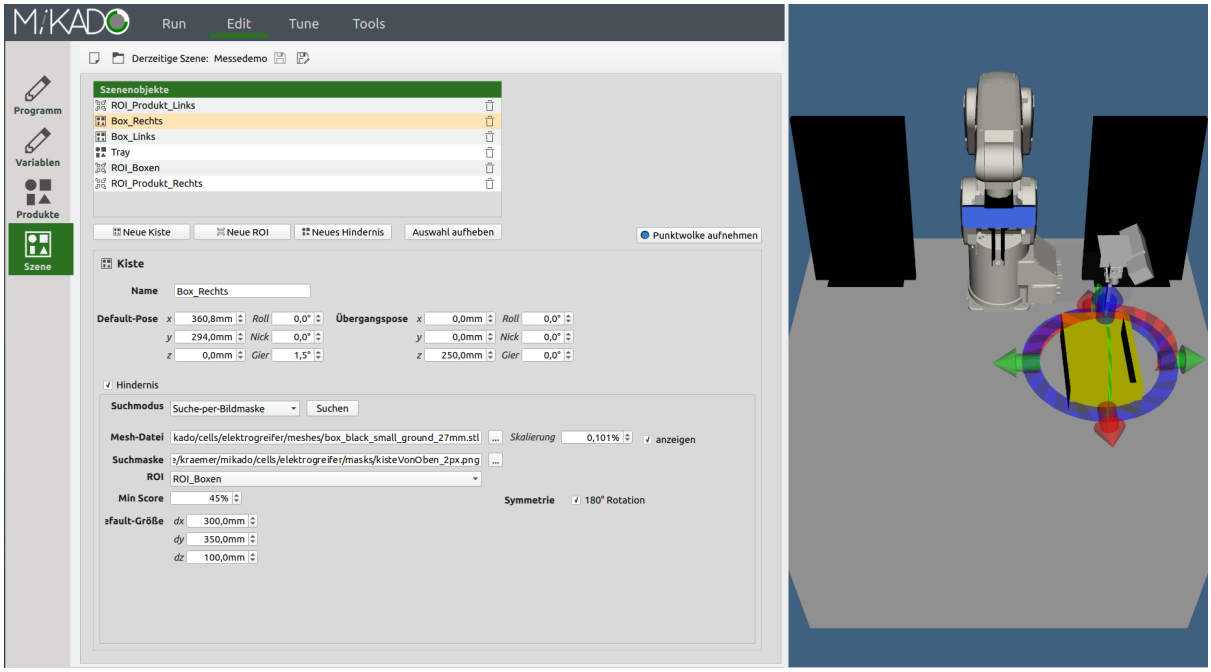


Abb. 14: Screenshot Szenenverwaltung (Beispiel einer Aufnahmekiste)

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 (teiltransparenten) Objekt verdeckt werden.

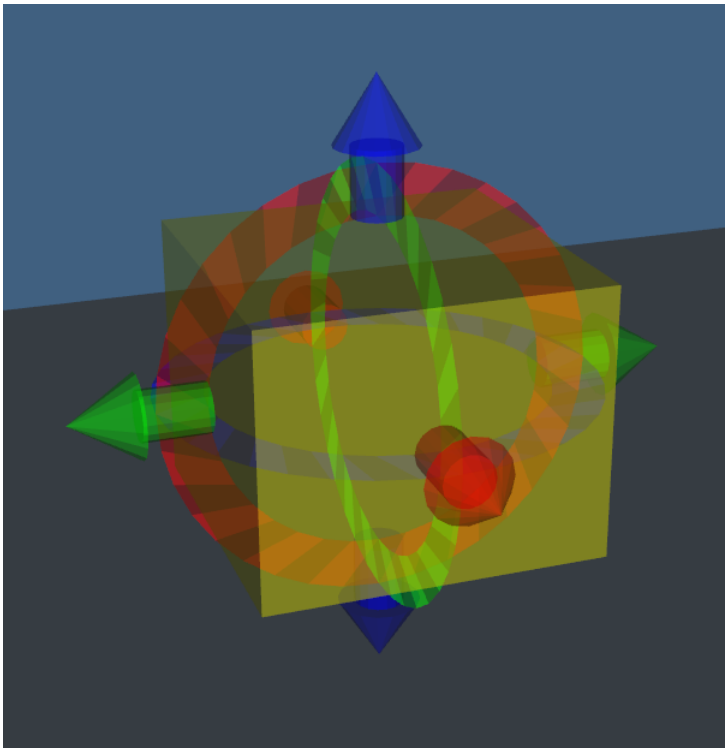


Abb. 15: Interaktiver Marker für Szenenobjekte (Steuerelemente in rot, grün und blau)

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.

Hindernis

Ein *Hindernis* ist ein physikalisches Objekt in der Szene. Es kann als fix betrachtet oder per 2D-Suche gesucht werden.

Kiste

Eine Kiste wird in der Greifsuche benötigt. Sie enthält die sogenannte *Übergangspose*, welche als Start- und Zielpose für die Greifplanung fungiert. Bei der *Übergangspose* gilt es zu beachten, dass diese relativ zur Default-Pose der Kiste angegeben wird, nicht relativ zum Roboter.

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

Zu beachten ist, dass für Kisten mit Hindernis, die für die Griffsuche verwendet werden sollen, unbedingt eine Mesh-Datei für die Kollisionsvermeidung angegeben werden muss.

Für Kisten, die nicht immer an exakt die gleiche Stelle gestellt oder fixiert werden können, gibt es eine Lagenachführung per 2D Suche. Für die 2D-Suche muss eine 2D-Maske angegeben werden. Diese kann aus Mikado heraus erstellt werden:

1. *ROI anlegen* Diese sollte in z nach Möglichkeit nur den oberen Kistenrand erhält und in x und y den Bereich abdeckt in dem die Kiste verschoben werden könnte. Sie wird auch nach dem Anlegen im laufenden Programm verwendet werden. Bei der Masken-Erstellungsaufnahme sollte die Kiste möglichst orthogonal ausgerichtet im Bild stehen. Die ROI unter Suchparameter auswählen.
2. *Visualisieren* Den Reiter Kamera auswählen und im Auswahlmenü „. . . projected_heightmap“ auswählen. Dann auf Suchen klicken.
3. *Speichern* Per Rechtsklick auf das Vorschaubild kann entweder die Rohversion gespeichert werden oder eine zugeschnittene und freigestellte Variante.
4. *Nachbearbeiten* Die Kiste muss exakt rechtwinklig im Bild stehen. Variables Rauschen und damit fehlende Konturen sollten nachbearbeitet werden. Bei der Suche von Kisten mit variabler Höhe oder wo sonst Teile in den Bereich der Such-ROI ragen, können diese Regionen per alpha-Channel ausmaskiert werden. Sie gehen dann nicht in die Suchbewertung mit ein. Die zugeschnittene Variante maskiert standardmäßig alle nicht-Randbereiche aus.

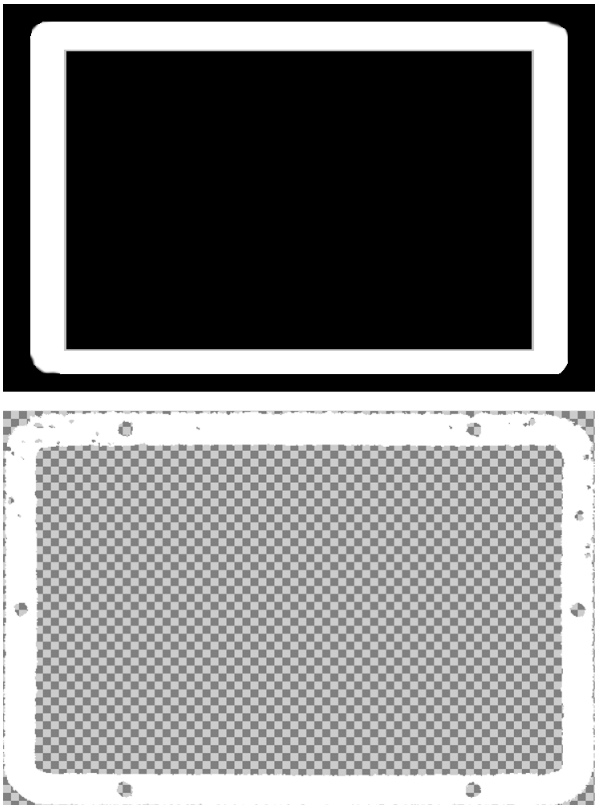


Abb. 16: Zwei Beispiele einer 2D Suchmaske für eine Kiste, oben, der Hintergrund wird als schwarz angenommen und geht in die Bewertung mit ein. Unten, der transprarente Hintergrund bedeutet Dont-Care, an diesen Stellen können beliebige Pixel/Höhenwerte stehen.

In der hier abgebildeten Beispielmaske, 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 Ursprung des Objekts darstellt.

Um ein Hindernis in der Kollisionsvermeidung einzuplanen, gibt es die Möglichkeit eine Mesh-Datei anzugeben, die das Objekt beschreibt. Der Ursprung einer Mesh-Datei muss zentral im Boden des Objekts liegen.

Es ist ebenso möglich eine Default-Größe anzugeben, die dem Hindernis eine quaderförmige Region zuordnet. Ist das Hindernis per 2D-Suche zu suchen, wurde aber im Laufe des Programmablaufs noch nicht gesucht (oder nicht gefunden), so wird zunächst die Default-Größe für die Kollisionsvermeidung verwendet. Dieser Quader sollte also den 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, der angibt wie sicher die Maske zu den Kameradaten passen muss. Die Ausgabe erfolgt unter *Logs* oder in der Konsole.

- **180° Symmetrie**

  Weist das Objekt eine Rotationssymmetrie um die z-Achse auf, so kann diese Option aktiviert werden. Mithilfe dieser Eigenschaft können trotz Symmetrie eindeutige Übergangsposen definiert werden. Beispielsweise für rechteckige Kisten können Sie diese Eigenschaft aktivieren.

Region of Interest (ROI)

Eine ROI ist eine Region im Raum – 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. Ausblendung des Untergrunds). Zusätzlich zur Default-Pose hat sie lediglich Größeneigenschaften ((Länge, Breite, Höhe) bzw. (dx, dy, dz)). Wird eine ROI angewählt, so hat sie direkt Auswirkung auf die Aufnahme einer Punktwolke. So kann die Größe der ROI interaktiv angepasst werden.

2.12 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 kön-

nen Sie direkt inspizieren. Zudem bietet dieses Werkzeug verschiedene Ansichten, die es ermöglichen die Ergebnisse zu verstehen und Maßnahmen zur Erzielung besserer Resultate einzuleiten.

2.12.1 Erste Schritte

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').

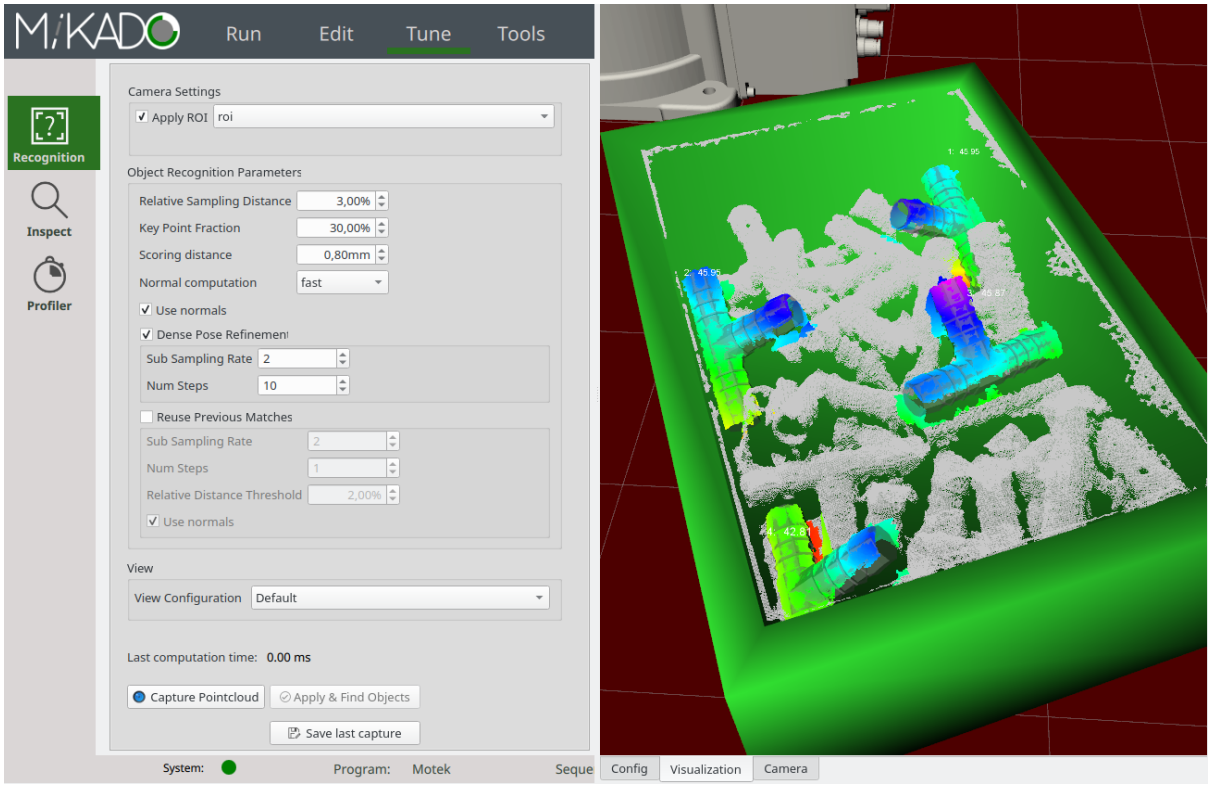


Abb. 17: Screenshot Objekterkennungs-Tool

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

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 über den Buttons angezeigt (*Letzte Berechnungszeit*).

Hinweis: Bevor Sie die Parameter für die Objektsuche optimieren, sollten Sie sicherstellen, dass Sie möglichst gute grundlegende 3D-Daten erhalten. Der Bereich *Kamera-Einstellungen* bietet sich an, um zunächst zu kontrollieren, ob das Hardware-Setup den Anforderungen entsprechend eingerichtet ist.

2.12.2 Kamera-Auswahl

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

2.12.3 Funktionsweise der Objekterkennung

Die Objektsuche besteht in der Regel aus 3 Schritten:

1. ***Näherungsweises Matching*** Die CAD-Abbildung des Produkts wird in eine Repräsentation durch Punkte umgewandelt und die Daten der 3D-Kamera werden auf eine Menge von Schlüsselpunkten reduziert, um eine schnelle Suche zu ermöglichen, die die grobe Lage der Objekte erkennt.
2. ***Verfeinerung*** Nun wird die CAD-Abbildung des Produkts erneut in eine Punktrepräsentation gebracht, diesmal allerdings detailgetreuer. Die Ergebnisse aus Schritt 1 werden dann durch aufwändigere Suchverfahren verfeinert.
3. ***Wiederverwendung vorheriger Matches*** Sofern diese Funktion aktiviert ist, werden die Ergebnisse zwischengespeichert und unterstützen in der nächsten Objektsuche. Ein Fund/Match wird in der folgenden Suche nur daraufhin überprüft, ob er noch vorhanden ist und dann aus der regulären Suche entfernt. So kann die folgende Objektsuche schneller Objekte finden, die sich nicht bewegt haben, was auch die Suche nach neuen Objekten beschleunigt.

2.12.4 Objekterkennungsparameter

In diesem Bereich können Sie die Parametrierung der Objekterkennung optimieren. Die Initialwerte der Parameter entsprechen der *Konfiguration* des Projekts (‚config/object_recognition.yaml‘, siehe *Verzeichnisstruktur*). Beachten Sie, dass hier globale Parameter spezifiziert werden, die nicht produktspezifisch sind, sondern für alle Produkte gemeinsam eingestellt werden. Produktspezifische Parameter können Sie mithilfe der *Produktverwaltung* anpassen und mit dem Objekterkennungs-Tool testen. Manche davon haben einen großen Einfluss auf das Suchergebnis und sollten ebenfalls als Objekterkennungsparameter betrachtet werden (s. *Produktverwaltung*).

Warnung: Alle Objekterkennungsparameter dieser Seite sind sofort aktiv, auch für den normalen Programmablauf! Ein Neustart von Mikado ARC setzt jedoch alle Werte wieder auf den Initialzustand zurück, der in der *Konfiguration* festgelegt ist. Abschnitt *Parameter übernehmen* beschreibt die Vorgehensweise, wie Parameter dauerhaft übernommen werden können.

- ***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

Weitere Parameter

Einige Parameter lassen sich nicht über die Mikado-Oberfläche beeinflussen, da diese beim Systemstart bereits festgelegt sein müssen. Im Abschnitt *Parameter übernehmen* wird erläutert, wie diese Parameter verändert werden können.

- ***Voxelgröße*:** Ein 3D-Punkt wird bei der Kollisionplanung als „Voxel“ betrachtet, also als kleiner Würfel. Hier kann die Seitenlänge eines solchen Würfels festgelegt werden. [mm]
- ***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]
- ***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 Dilationsradious der Objektsuche gewählt werden. [Anzahl Voxel]
- ***Relative Sampling-Distanz (Produkte)*:** Abtastdistanz des Oberflächenmodells (Produkts) relativ zu dessen Durchmesser. [0.0 - 1.0]
- ***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]
- ***CUDA verwenden*:** Falls der Mikado-Controller mit einer NVIDIA Grafikkarte ausgerüstet ist, kann mithilfe von CUDA das Stereo-Matching beschleunigt werden. [true, false]

2.12.5 Ansicht

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.

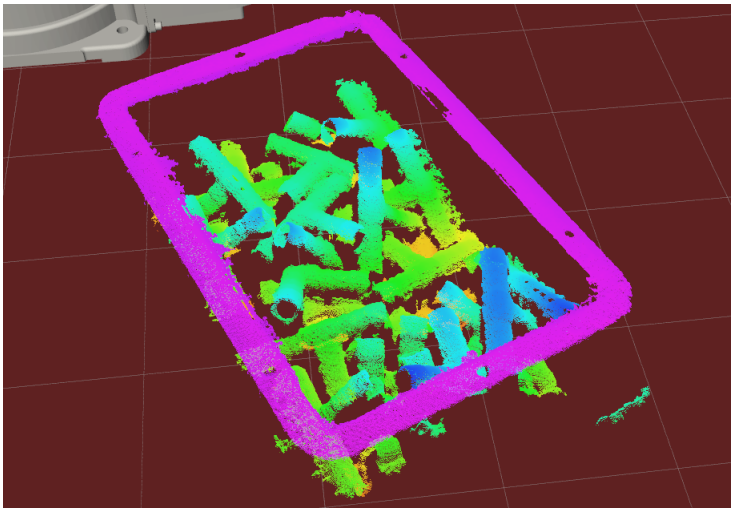


Abb. 18: Screenshot Objekterkennungs-Tool Punktwolke nach Bildaufnahme

Wurde eine Objektsuche durchgeführt, verändert sich die Darstellung: Gefundene Objekte werden je nach Ansichtseinstellung abgebildet und die dazugehörigen Scores eingeblendet:

- ***Standard*-Ansicht:** Funde werden in der Visualisierung/Punktwolke als CAD-Modell grau eingezeichnet. Punkte, die nahe an einem Fund liegen, sind dabei farblich dargestellt (ebenfalls nach ihrem Z-Wert). Alle weiteren Punkte werden grau dargestellt.

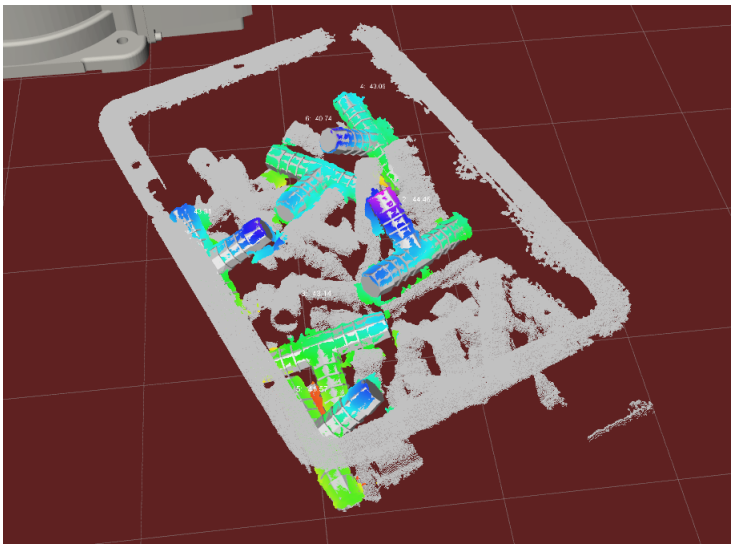


Abb. 19: Default-Ansicht des Objekterkennungs-Tools

- ***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.

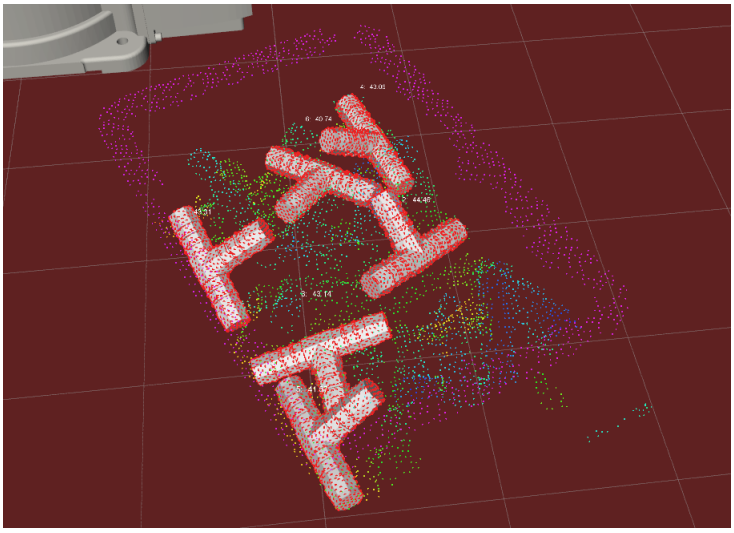


Abb. 20: Schlüsselpunkte-Ansicht des Objekterkennungs-Tools

- ***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.

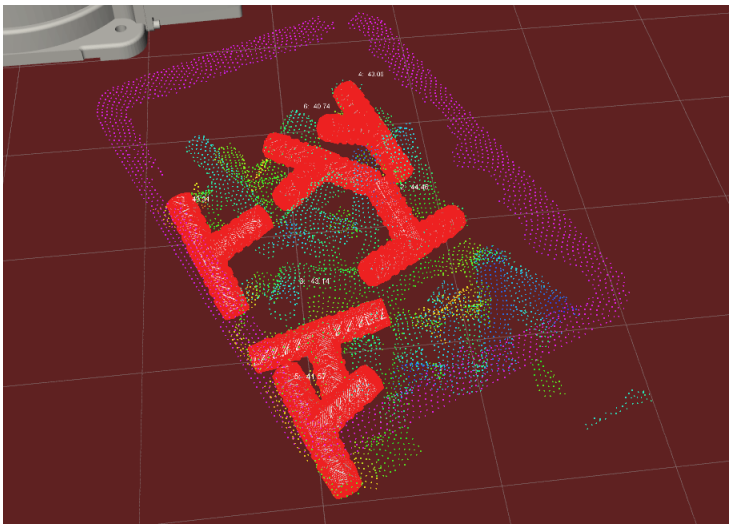


Abb. 21: Verfeinerungsansicht des Objekterkennungs-Tools

2.12.6 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.

2.12.7 Parameter übernehmen

Sobald Sie eine geeignete Parameterierung für Ihr System gefunden haben, müssen Sie für eine dauerhafte Übernahme die Werte in die *Konfiguration* (‚config/object_recognition.yaml‘, siehe *Verzeichnisstruktur*) übertragen.

Der folgenden Tabelle ist zu entnehmen, für welchen Parameter welche Bezeichnung in der Konfigurationsdatei verwendet wird.

Parameter		Konfigurationsbezeichnung
Relative Sampling-Distanz		hv_SamplingDistance
Anteil entscheidender Punkte		hv_KeypointFraction
Scoring Abstand		pose_ref_scoring_dist_abs
Verwende Normaleninformation		pose_ref_use_scene_normals
Berechnung der Szenennormalen		scene_normal_computation
Exakte Posenverfeinerung	Aktivieren/Deaktivieren	dense_pose_refinement
	Sub-Sampling-Rate	pose_ref_sub_sampling
	Anzahl der Schritte	pose_ref_num_steps
Vorherige Matches wiederverwenden	Aktivieren/Deaktivieren	reuse_matches
	Sub-Sampling-Rate	check_pose_ref_sub_sampling
	Anzahl der Schritte	check_pose_ref_num_steps
	Schwellwert der relativen Distanz	check_pose_ref_dist_threshold_rel
	Verwende Szenen-Normalen	check_pose_ref_use_scene_normals

Weitere Parameter (nicht über GUI modifizierbar)	Konfigurationsbezeichnung
Voxelgröße	voxel_size
Dilationsradius Objektsuche	dilation_radius
Dilationsradius beim Ausschneiden von Matches	dilation_radius_object_cutting
Relative Sampling-Distanz (Produkte)	relative_sampling_distance
Relative Sampling-Distanz für Verfeinerung (Produkte)	pose_ref_rel_sampling_distance
CUDA verwenden	cuda_enabled

2.13 Griff-Inspektion

2.13.1 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 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.

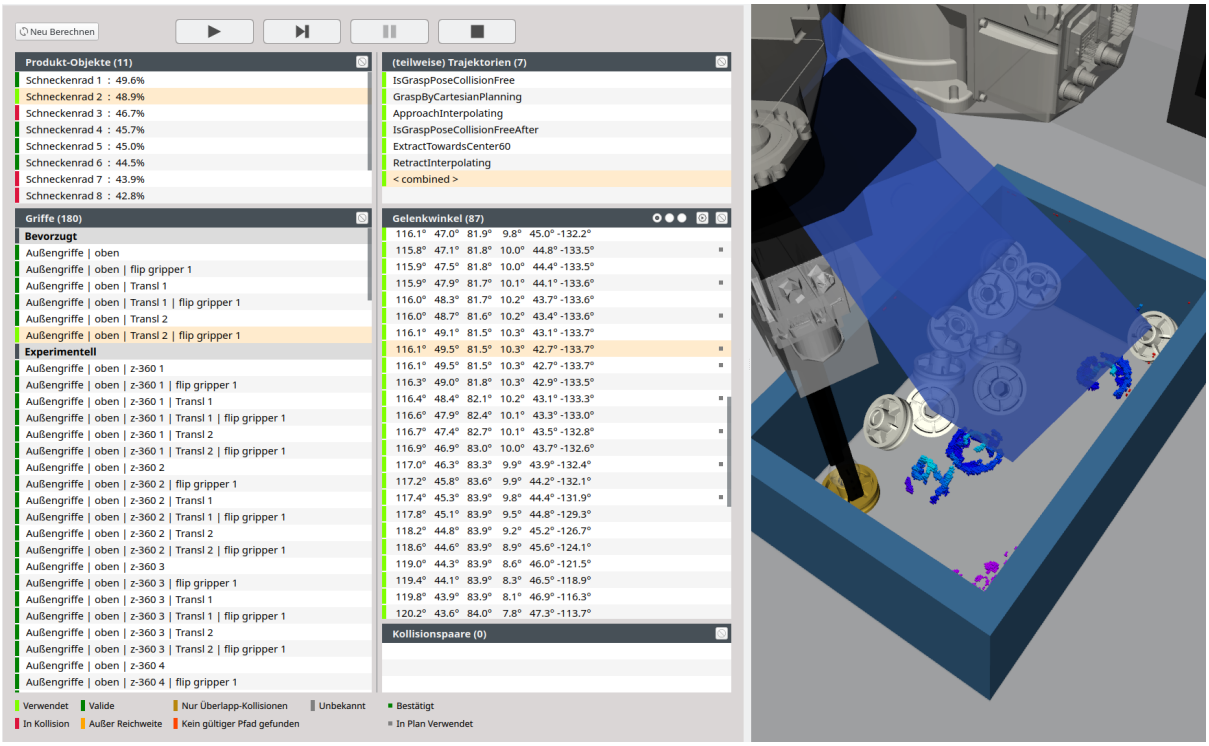


Abb. 22: Screenshot Griff-Inspektion

2.13.2 Greifbewegungssuche

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

Die Gesamttrajektorie besteht somit 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. Für die Definition der Anfahrts- und Entnahmepose siehe *Greifprozess*.

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

    (1) in Greif-Richtung (*ExtractInGraspDirection*)

    (2) nach oben (*ExtractUpwards*)

    (3) in Richtung der Endpose (*ExtractTowardsFinalPose*)

    (4) in Richtung Kistenmitte im 10°-Winkel zur (x,y)-Ebene (*ExtractTowardsCenter10*)

    (5) in Richtung Kistenmitte im 40°-Winkel zur (x,y)-Ebene (*ExtractTowardsCenter40*)

    (6) in Richtung Kistenmitte im 60°-Winkel zur (x,y)-Ebene (*ExtractTowardsCenter60*)

  (ii) Objekt/Octomap-Kollisionen (siehe unten) sind erlaubt.

    (1) in Greif-Richtung (*ExtractInGraspDirection_AllowOOCollisions*)

(2) nach oben (*ExtractUpwards_AllowOOCollisions*)

(3) in Richtung der Endpose (*ExtractTowardsFinalPose_AllowOOCollisions*)

(4) in Richtung Kistenmitte im 10°-Winkel zur (x,y)-Ebene (*ExtractTowardsCenter10_AllowOOCollisions*)

(5) in Richtung Kistenmitte im 40°-Winkel zur (x,y)-Ebene (*ExtractTowardsCenter40_AllowOOCollisions*)

(6) in Richtung Kistenmitte im 60°-Winkel zur (x,y)-Ebene (*ExtractTowardsCenter60_AllowOOCollisions*)

(iii) Überlappkollisionen (siehe unten) sind erlaubt.

(1) in Greif-Richtung (*ExtractInGraspDirection_AttObjObjColl*)

(2) nach oben (*ExtractUpwards_AttObjObjColl*)

(3) in Richtung der Endpose (*ExtractTowardsFinalPose_AttObjObjColl*)

(4) in Richtung Kistenmitte im 10°-Winkel zur (x,y)-Ebene (*ExtractTowardsCenter10_AttObjObjColl*)

(5) in Richtung Kistenmitte im 40°-Winkel zur (x,y)-Ebene (*ExtractTowardsCenter40_AttObjObjColl*)

(6) in Richtung Kistenmitte im 60°-Winkel zur (x,y)-Ebene (*ExtractTowardsCenter60_AttObjObjColl*)

(f) Zurückfahren

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

(ii) per Freier Bahnplanung (*RetractFreely*)

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* in der *pick_search.yaml* erlaubt werden.

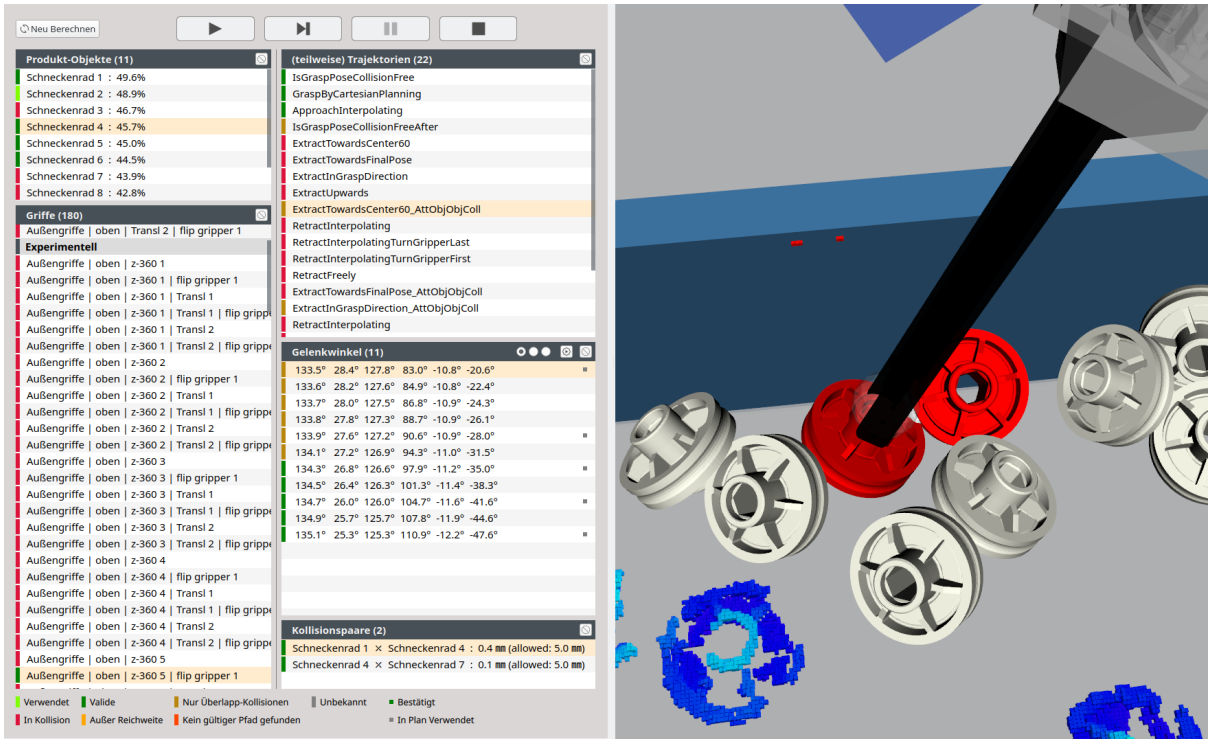


Abb. 23: Screenshot Beispiel für Überlappkollisionen

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 den Parameter *max_allowed_collision_depth* (allgemein) bzw. über *specific_max_allowed_collision_depths* (Produkt-spezifisch) eingestellen. 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.

Es ist auch möglich, einzelne Entnahme- (Extract-) oder Rückfahr- (Retract-)Funktionen abzuschalten. Hierzu dient der Parameter *search_fn_blacklist*. 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*).

Wenn mehrere valide Griffe gefunden wurden, muss entschieden werden, welcher Griff präferiert werden soll. Dies geschieht nach den folgenden Kriterien:

(1) Griff-Kategorie (bevorzugte Griffe vor Standardgriffen vor experimentellen Griffen)

(2) kollisionsfreie Griffe vor Griffen mit erlaubten Kollisionen

(3) Objekte, deren Fahrtlänge im Gelenkwinkelraum deutlich größer als die kleinste Fahrtlänge ist, werden schlechter gestellt. Die „kleinste“ Fahrtlänge ist dabei die kürzeste Fahrtlänge aller bis dahin gefundenen validen Griffe. Als „deutlich größer“ gilt ein Griff, wenn seine Fahrtlänge mehr als anderthalb Standardabweichungen über der mittleren Fahrtlänge liegt, wobei als Datengrundlage auch hier alle bis dahin gefundenen validen Griffe dienen.

(4) (*konfigurierbar*)

   (a) Gruppierung nach Höhe des Objektmittelpunktes

   (b) Gruppierung nach Objekterkennungsqualität

(5) Länge der Fahrt im Gelenkwinkelraum

Gruppierung nach Höhe des Objektmittelpunktes Vom höchstgelegenen Objekt aus wird der Höhen-Wertebereich in gleich-hohe Segmente unterteilt, bspw. von 1 cm. Die Höhe (z-Wert im Welt-Koordinatensystem) ist über den Parameter *z_bin_size* einstellbar (Default-Wert: 0.05 (= 5 cm)). Objekte in einem höheren Segment werden gegenüber Objekten in einem tieferen Segment bevorzugt, aber Objekte innerhalb eines Segments werden gleich behandelt. Bsp: Objekte haben folgende Höhenwerte: {45,4 cm, 44,6 cm, 43,8 cm, 43,3 cm} und die Segmentgröße ist 1 cm. Dann liegen die ersten beiden Objekte in Gruppe 1 (zwischen 44,4 cm und 45,4 cm), das dritte in Gruppe 2 (zwischen 43,4 cm und 44,4 cm) und das vierte in Gruppe 3 (zwischen 42,4 cm und 43,4 cm).

Gruppierung nach Qualität der Objekterkennung Ausgehend vom besten Erkennungs-Score wird der Score-Wertebereich unterteilt in gleich große Segmente, bspw. jeweils 0,05 (= 5 %). Die Segmentgröße ist einstellbar über den Parameter *score_bin_size* (Default-Wert: 0.1 (= 10 %)). Objekte werden dann entsprechend ihres Erkennungs-Scores einem Segment zugeordnet. Objekte in höheren Segmenten werden gegenüber Objekten in niedrigeren Segmenten bevorzugt. Bsp.: siehe „Gruppierung nach Höhe des Objektmittelpunktes“.

Ob die Objekthöhe und/oder -erkennungsqualität berücksichtigt werden sollen, kann ebenfalls über die *pick_search.yaml* parametrisiert werden. Hierzu dient der Parameter *pick_selection_order*, welcher eine Liste erwartet, deren Elemente die folgenden Namen haben dürfen: „score-binning“, „z-binning“. Ersterer schaltet die Gruppierung nach Erkennungsqualität ein, letzterer die Gruppierung nach Objekthöhe. Eine leere Liste überspringt beide Schritte. Der Default-Wert für die *pick_selection_order* ist [„score-binning“, „z-binning“].

2.13.3 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.

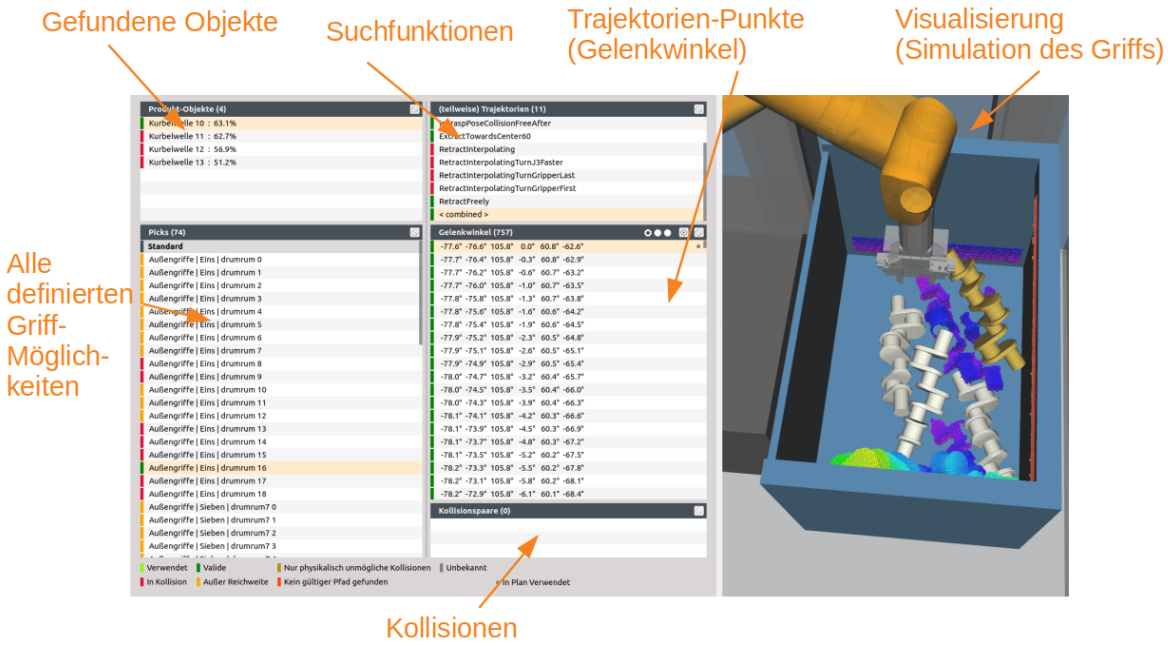


Abb. 24: Screenshot Griff-Inspektion mit Beschreibung

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.

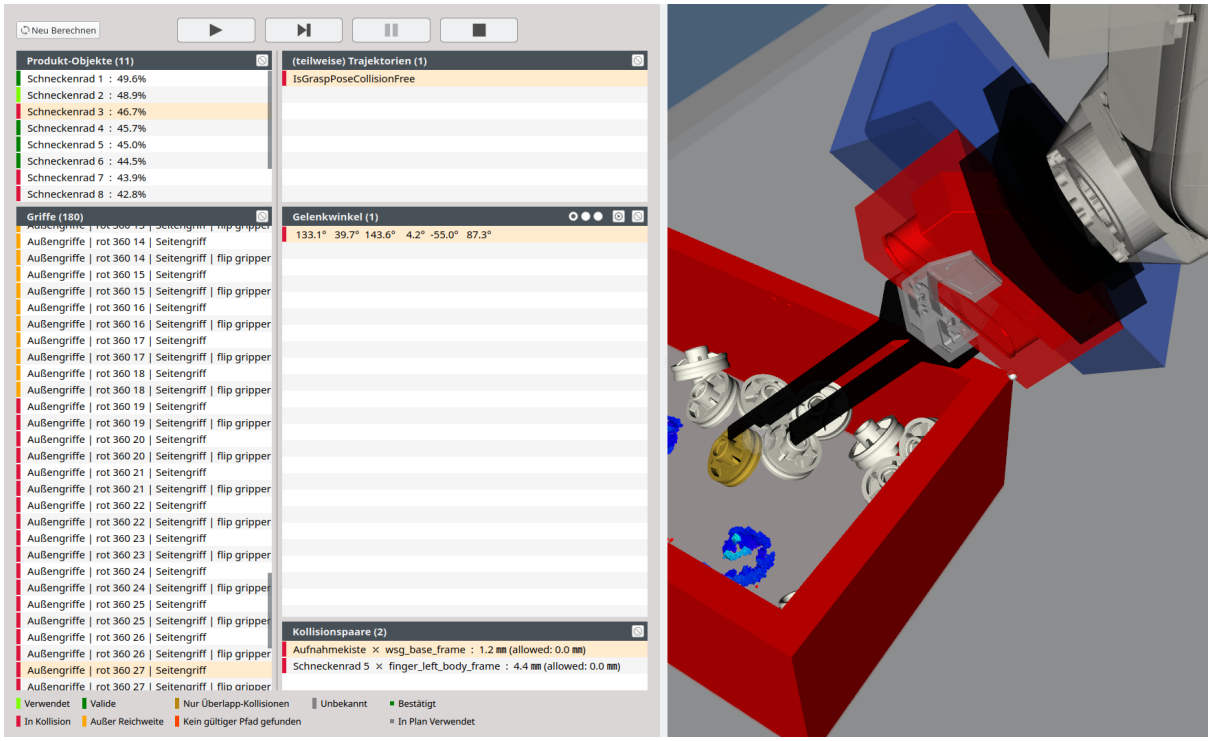


Abb. 25: Screenshot Kollisionsanzeige in der Griff-Inspektion

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.

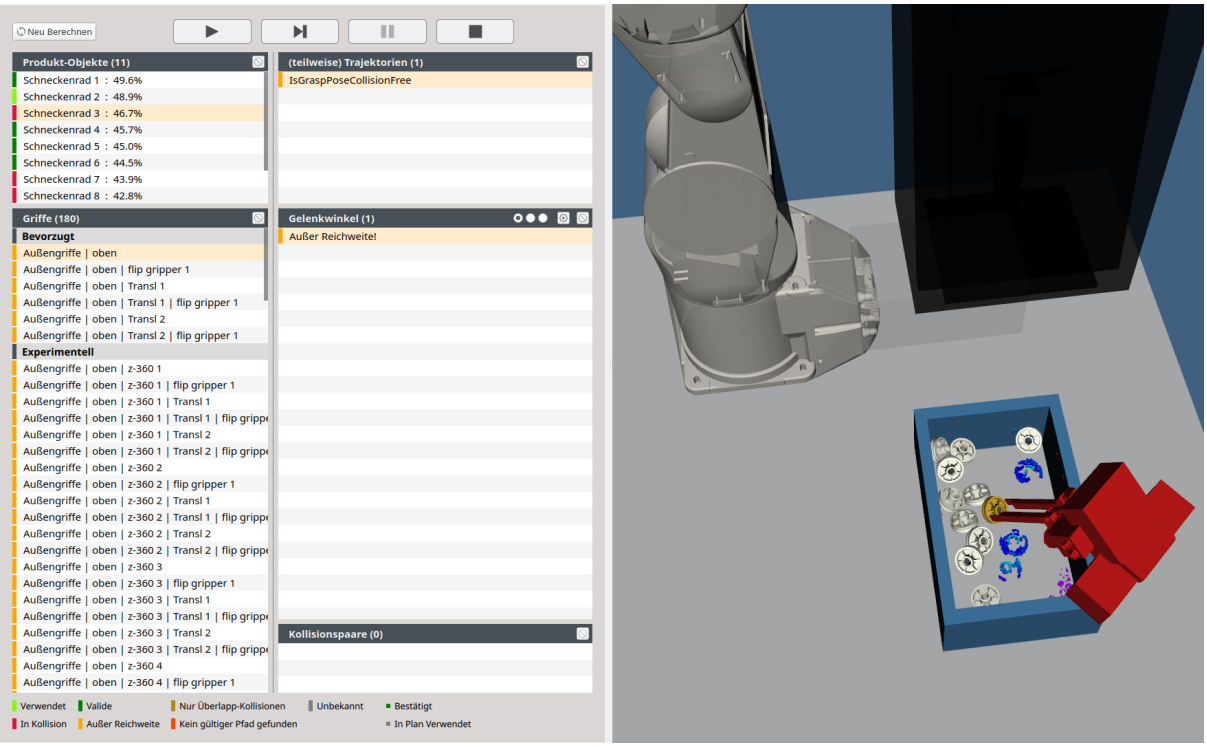


Abb. 26: Screenshot Darstellung unerreichbarer Griffe in der Griff-Inspektion

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.

2.14 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.

2.15 Logs

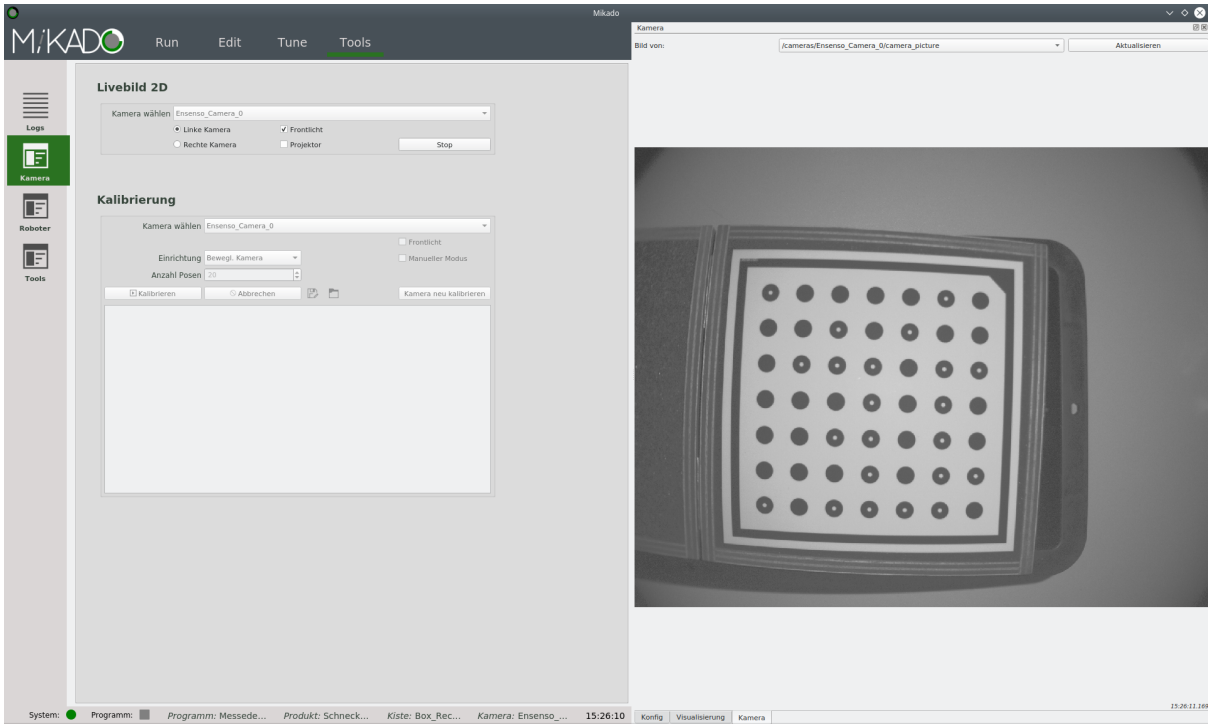
2.15.1 Überblick

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

2.16 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. So stellen Sie sicher, dass korrekte 3D-Daten mithilfe Ihres 3D-Sensors erfasst werden. Mikado ARC kann dann verwendet werden, um die Aufnahmen zu überprüfen oder um zwischen verschiedenen Konfigurationen zu wechseln.

2.16.1 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.

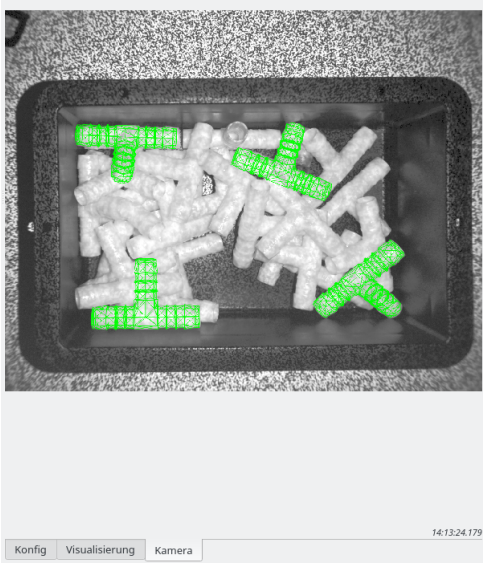
2.16.2 Kamera-Tab

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 ange

- ***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 Kamera-

bild 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

2.16.3 Kalibrierung

Mikado ARC unterstützt verschiedene Kalibrierungsvarianten, 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.

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

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.
- ***Kalibrieren*:** Dieser Button startet die Kalibrierung
- ***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*:** In diesem Modus wird eine ‚Hand-Auge-Kalibrierung' durchgeführt. Die Kamera befindet sich am Roboterarm und ist daher beweglich. 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.

  Bemerkung: Nach wenigen Aufnahmen wird bereits eine Vorkalibrierung durchgeführt, damit die Posen möglichst effektiv generiert werden.

- ***Fixe 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.

  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 Mo-

dus 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.

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.

Hinweis: Wechseln Sie während der Kalibrierung einer beweglichen Kamera in den ‚Visualisierungs'-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.

Manueller Modus

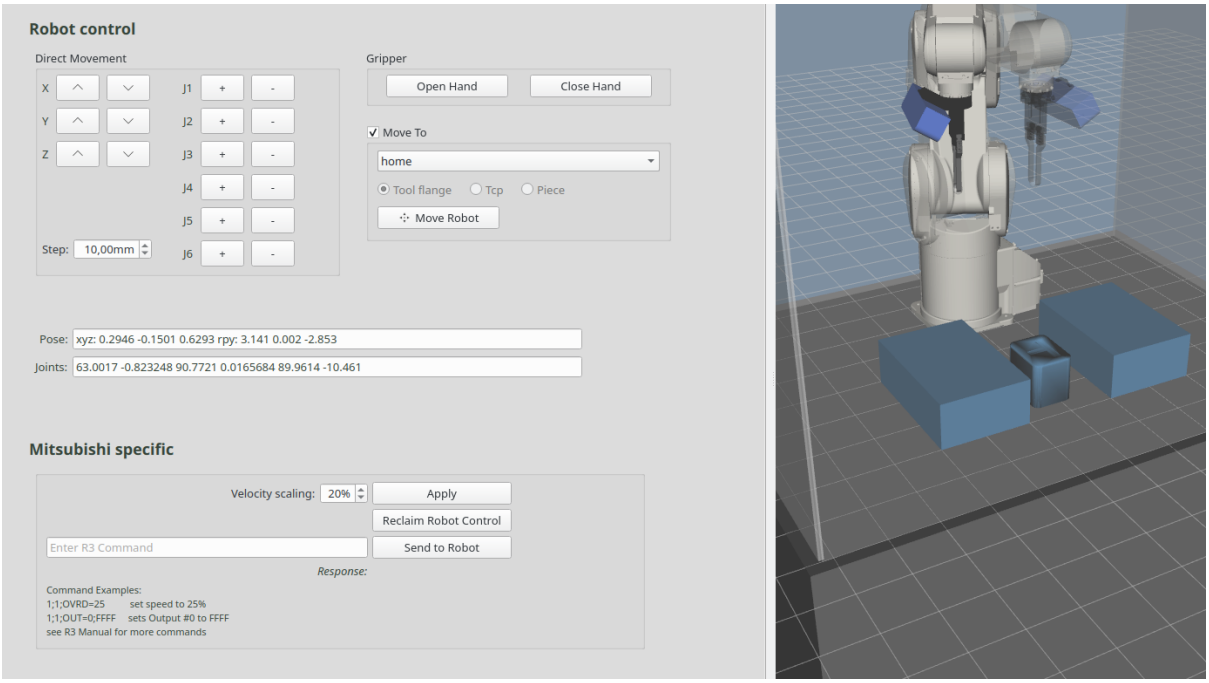
Im Manuellen Modus können Sie die einzelnen Kalibrierungsschritte selbst steuern. Diese Option steht Ihnen bei den Setup-Typen ‚Bewegliche Kamera' und ‚Fixe Kamera' zur Verfügung. Der Manuelle Modus erlaubt Ihnen zum Beispiel, den Roboter für die einzelnen Aufnahmen manuell in eine geeignete Position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

2.17 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.



2.17.1 Roboterkontrolle

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.

- **X / Y / Z**:

  Ein Klick auf einen Pfeil erhöht/verringert den Wert auf der jeweiligen Achse um die angegebene Schrittweite (Standardwert: 10 mm). Dabei fährt der Roboter in kartesischen Koordinaten mit dem Handgelenk des Roboterarms als Referenz des Koordinatensystems.

- **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.

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).

- **Öffne Hand**:

  Öffnet den Greifer des Roboters

- **Schließe Hand**:

  Schließt den Greifer des Roboters

- **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 ARC.

- **Gelenke**:

  Positionswerte des Roboters als Gelenkwinkel in Grad.

2.17.2 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*.

2.18 System-Tools

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

- Buttons um Mikado ARC zu beenden oder den Mikado-Controller komplett herunterzufahren.
- Informationen über die Software-Version von Mikado ARC.
- Eine Checkbox, um die Mikado-Oberfläche in den Entwicklermodus zu schalten. Dieser Modus ist für den internen Gebrauch bestimmt und wird z.B. für Profiling- und Test-Zwecke der Software verwendet.

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 Programm beenden

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

3.1.3 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 ARC zu, so ist es unter Umständen wichtig zu wissen, ob ein Programm erfolgreich beendet wurde oder aufgrund eines Fehlers gestoppt wurde.

3.1.4 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

3.1.5 Programm unterbrechen

Das Programm anhalten, durch ‚Start' oder ‚Nächster Task' fortsetzen.

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.

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 x kleiner als y?

Ermöglicht einen bedingten Sprung zu einer anderen Sequenz. Ist der Wert von Variable x kleiner als Variable y, so springt der Task zur Sequenz des Sequenzwechsels ‚Wahr'. Entsprechend

fährt die Programmausführung mit der Sequenz des Sequenzwechsels ‚Falsch' fort wenn x nicht kleiner als y ist.

Weiterhin kann ein Sequenzwechsel auch leer gelassen werden. In diesem Fall fährt die Programmausführung mit dem nächsten Task der aktiven Sequenz fort, sofern die entsprechende Bedingung erfüllt wird.

Beispiel: x = 1, y = 5, Wahr: leer, Falsch: Sequenz ‚Meine Sequenz'

In dieser Situation würde die aktive Sequenz einfach weiter abgearbeitet, da die Bedingung x < y erfüllt wird und dem Sequenzwechsel ‚Wahr' keine Sequenz zugeordnet wurde.

Hinweis: Variablentypen von x und y müssen übereinstimmen

Variablen

Name	Typ	Beschreibung
x	Integer, Double	–
y	Integer, Double	–

Sequenzwechsel

Name	Bedingung	Anmerkung
Wahr	x < y	–
Falsch	x >= y	–

3.2.4 x größer als y?

Ermöglicht einen bedingten Sprung zu einer anderen Sequenz. Ist der Wert von Variable x größer als Variable y, so springt der Task zur Sequenz des Sequenzwechsels ‚Wahr'. Entsprechend fährt die Programmausführung mit der Sequenz des Sequenzwechsels ‚Falsch' fort wenn x nicht größer als y ist.

Weiterhin kann ein Sequenzwechsel auch leer gelassen werden. In diesem Fall fährt die Programmausführung mit dem nächsten Task der aktiven Sequenz fort, sofern die entsprechende Bedingung erfüllt wird.

Beispiel: x = 1, y = 5, Wahr: leer, Falsch: Sequenz ‚Meine Sequenz'

In dieser Situation würde ein Sequenzwechsel zur Sequenz ‚Meine Sequenz' folgen, da die Bedingung x > y erfüllt wird und dem Sequenzwechsel ‚Wahr' diese Sequenz zugeordnet wurde.

Hinweis: Variablentypen von x und y müssen übereinstimmen

Variablen

Name	Typ	Beschreibung
x	Integer, Double	–
y	Integer, Double	–

Sequenzwechsel

Name	Bedingung	Anmerkung
Wahr	x > y	–
Falsch	x <= y	–

3.2.5 x gleich y?

Ermöglicht einen bedingten Sprung zu einer anderen Sequenz. Stimmt der Wert von Variable x mit dem Wert der Variable y überein, so springt der Task zur Sequenz des Sequenzwechsels ‚Wahr'. Entsprechend fährt die Programmausführung mit der Sequenz des Sequenzwechsel ‚Falsch' fort wenn x ungleich y ist.

Weiterhin kann ein Sequenzwechsels auch leer gelassen werden. In diesem Fall fährt die Programmausführung mit dem nächsten Task der aktiven Sequenz fort, sofern die entsprechende Bedingung erfüllt wird.

Beispiel: x = 1, y = 5, Wahr: leer, Falsch: Sequenz ‚Meine Sequenz'

In dieser Situation würde die aktive Sequenz einfach weiter abgearbeitet, da die Bedingung ‚x = y' nicht erfüllt wird und dem Sequenzwechsel ‚Wahr' keine Sequenz zugeordnet wurde.

Hinweis: Variablentypen von x und y müssen übereinstimmen

Variablen

Name	Typ	Beschreibung
x	Integer, Double	–
y	Integer, Double	–

Sequenzwechsel

Name	Bedingung	Anmerkung
Wahr	x = y	–
Falsch	x <> y	–

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 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

3.3.5 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	

3.3.6 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.

3.3.7 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

3.3.8 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

3.3.9 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

3.3.10 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.

3.3.11 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

3.3.12 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	–

3.4 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 fortgesetzt, 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 Tcp nach

Bewege den Roboter mittels freier Bewegungsplanung, sodass der Tool Center Point (TCP oder Endeffektor) 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.

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 fortgesetzt, 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.

3.4.3 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 fortgesetzt, 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.

3.4.4 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*‘.

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 fortgesetzt, 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.

3.4.5 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*‘.

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 fortgesetzt, 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.

3.4.6 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

3.4.7 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
Roboterposition	Pose	Ausgabevariable: Momentane Position des Handgelenks

Sequenzwechsel

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

3.4.8 Ist Roboter Position mit Teil erreichbar?

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

Variablen

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

Sequenzwechsel

Name	Bedingung	Anmerkung
Nicht Erreichbar	Zielpose nicht erreichbar	–

3.4.9 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)

3.4.10 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.

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 ARC geben. In der hier angegebene Variable wird der Rückgabewert nach Ausführung des Tasks hinterlegt.

3.4.11 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.

3.4.12 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‘)

3.4.13 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‘)

3.4.14 Auf Robotereingangssignal warten

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

Warnung: Es wird stark empfohlen, einen Timeout-Wert anzugeben. Andernfalls kann es vorkommen, dass der Task die Ausführung des Programms blockiert.

Variablen

Name	Typ	Beschreibung
Signalnummer	Integer	Ausgangssignale eines Roboter-Controllers sind eindeutig durchnummeriert.
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	

3.4.15 Teil übergeben

Ein Speziallfall von *Roboterprogramm ausführen*, der die Verschiebung auf der x-Achse und die Rotation eines angehängten Teils bestimmt. Diese werden als Parameter beim Aufruf des Programms übergeben.

Variablen

Name	Typ	Beschreibung
Programmnummer	Integer	Programmnummer des nativen Roboterprogramms für die Teilübergabe.
Rückgabewert	Integer	Rückgabewert des nativen Roboterprogramms für die Teilübergabe.

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
Kameraname	String	Name 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.

Der Pfad bezieht sich relativ zum Verzeichnis ‚config' des Projekts. Wird auf die momentan aktive Kamera angewendet.

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: ‚cameraconfig.json'*

3.5.3 Punktwolke aufnehmen

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

Axel FC Bayern

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.

3.5.4 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.

3.5.5 Mehrfachaufnahme

Nimmt mehrere Punktwolken auf und kombiniert sie, *Punktwolke aufnehmen* verwenden um eine Serie abzuschließen.

Nutzen Sie diesen Task um mehrere Aufnahmen zu kombinieren (z.B. aus verschiedenen Aufnahmeperspektiven). Bis auf die letzte Aufnahme müssen Sie diesen Task ‚Mehrfachaufnahme' verwenden. Indem Sie den Task *Punktwolke aufnehmen* verwenden, werden alle vorigen Mehrfachaufnahmen zusammengeführt und stehen für den nächsten Schritt (z.B. Objektsuche) zur Verfügung.

Hinweis: Mehrfachaufnahmen kombinieren alle entstehenden Punktwolken und erhöhen dadurch die Anzahl der Punkte stark. Insbesondere überlappende Bereiche der Aufnahmen weisen in diesem Fall die doppelte Anzahl an Punkten auf. Beachten Sie, dass dies eine Objektsuche zwar verbessern, aber auch stark verzögern kann.

3.5.6 Kamerapose nachjustieren

Macht eine Aufnahme, sucht das Referenzobjekt und berechnet den Kamera-Offset.

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

Die Orientierung eines gefundenen Referenzobjekts fließt in die Berechnung nicht mit 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.

3.5.7 Anzahl der Punkte ermitteln

Anzahl der Punkte der letzten Aufnahme ermitteln.

Hinweis: Um Punkte einer bestimmten Region zu zählen, vorher den *ROI setzen* Task benutzen. Der *x größer als y?* 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 gefundenen Punktes ausgegeben.

Variablen

Name	Typ	Beschreibung
Anzahl	Double	Anzahl der 3D-Punkte in der aktiven ROI der aktiven Kamera wird in diese Variable geschrieben

3.5.8 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	Verzeichnispfad relativ zum Persönlichen Ordner, in dem die Bilder abgelegt werden sollen (z.B. ‚data/filecam1‘)
Präfix	String	Präfix der Dateinamen für die Bilder und Konfigurationsdateien (z.B. ‚capture_before_pick‘)

Hinweis: Der Task benötigt zum einen viel Zeit, da die Daten zeitsynchron auf die Festplatte geschrieben, zum anderen muss auf dem Controller ein Volllaufen der Festplatte verhindert werden. Daher sollte der Task nur sparsam eingesetzt werden, z.B. um nach Fehlerfällen die letzte Punktwolke zu speichern.

3.5.9 ROI setzen

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-Name	String	Name des ROI-Objekts, das ab sofort verwendet werden soll. Die Verwaltung der ROI-Objekte findet im Bereich *Szenenverwaltung* statt.

3.5.10 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

3.6 Szenenobjekt

3.6.1 Setze aktives Szenenobjekt

Falls mehr als ein Szenenobjekt in der Szenenobjekte-Liste geladen ist, kann hiermit das 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 dieses automatisch das aktive Objekt. In diesem Fall ist es nicht nötig, diesen Task in Ihren Programmablauf einzubauen.
Das aktive Szenenobjekt wird z.B. standardmäßig bei der Griffsuche verwendet (s. *Griffsuche starten*).

Variablen

Name	Typ	Beschreibung
Szenenobjektname	String	Name des Szenenobjekt, das nun aktiv sein soll

3.6.2 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. Darauf wird eine 2D-Bildsuche ausgelöst, weitere Informationen zu diesem Suchverfahren und wie diese eingerichtet werden kann s. Abschnitt *Szenenverwaltung* / *Hindernis*.

Hinweis: Die aktuelle Position eines Szenenobjekts kann mithilfe des Tasks *Szenenobjekt-Position auslesen* ausgelesen werden.

Variablen

Name	Typ	Beschreibung
Objektname	String	Der Name des Szenenobjekts, das gesucht werden soll

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.

3.6.3 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
Objektname	String	Der Name des Szenenobjekts
Position	Pose	Ausgabevariable: Die Position, bzw. Pose des Objekts wird vom Task in diese Variable geschrieben.

3.6.4 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
Objektname	String	Der Name des Szenenobjekts
Position	Pose	Die Position, bzw. Pose, die für das Szenenobjekt verwendet werden soll.

3.7 Objekt-Handhabung

3.7.1 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
Produktname	String	Das aktiv zu setzende Produkt. Produktnamen werden in der *Produktverwaltung* definiert.

3.7.2 Griffsuche starten

Starte Suche nach bestmöglichem Griff. 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 wiefolgt:

- 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 → Gefundene Objekte und restliche Punktwolke werden in der Visualisierung angezeigt
3. Griffsuche → Alle Griffe 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.

Variablen

Name	Typ	Beschreibung
Produkt	String	Name 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.
Kiste	String	Name der Kiste (Szenenobjekt), in der das Produkt zu finden ist. Wird das Feld leer gelassen, so wird die aktive Kiste verwendet. Kistennamen werden in der *Szenenverwaltung* definiert.
Kistensuche	Bool	Gibt an, ob die Kiste vor der Griff-/Objektsuche in der aktuellen Aufnahme gesucht werden soll (optional, Defaultwert: ‚true')

3.7.3 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.

3.7.4 Finde Objekt

Gibt eine Liste aller Posen der gefundenen Produkte sowie aller möglichen Griffe zurück.

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	String	Name 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.
Liste der gefundenen Objekte	PoseList	Liste an Posen der gefundenen Objekte. Die Posen beziehen sich auf den Ursprung des Produktkoordinatensystems.
Liste der möglichen Griffe	PoseList	Liste an Posen der möglichen Griffe. Die Posen beziehen sich dabei auf das ‚tool0', also das Handgelenk des Roboters beim Griff.

Sequenzwechsel

Name	Bedingung	Anmerkung
Kein Objekt gefunden	Kein Objekt gefunden	

3.7.5 Teil greifen

Greife das Produkt aus der Kiste. Der Roboter wird zur Transit-Pose der verwendeten Kiste gefahren, wo er auf die Ergebnisse der Greifplanung wartet. Wurde ein geeigneter Greifpfad gefunden (kürzere Pfade werden präferiert), wird das Teil gegriffen.

Hinweis: Für eine höhere Durchsatzgeschwindigkeit sollte die Punktwolke so früh wie möglich aufgenommen werden und die Greifplanung mit dem Task *Griffsuche starten* gestartet werden.

Bei Bedarf blockiert dieser Task, bis ein valider Griff gefunden wurde. Bei Zeitüberschreitung beendet der Task den Ablauf im Fehlerzustand (weiteres zum Timeout-Parameter in Sektion *pick_search.yaml*).

Weitere Informationen zum Teile greifen sind bei dem Task *Griffsuche starten* zu finden.

Hinweis: Um das Teil wieder abzulegen, müssen Sie den Task *Teil ablegen* nutzen!

Variablen

Name	Typ	Beschreibung
ist der Griff experimentell?	Bool	Ausgabevariable: wird auf ‚true' gesetzt, falls der ausgeführte Griff ein experimenteller Griff war
Gegriffenes Produkt	String	Ausgabevariable: der Name des Produkts, welches gegriffen wurde
Grifftransformation	String	Ausgabevariable: Die Transformation vom gegriffenen Teil zum TCP (Enthält den 180Grad Flip vom Teil in das TCP Koordinatensystem)

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 sicherer Fahrt o.ä.), so wird beim erneuten Versuch

- Der Greifer geöffnet, falls bereits ein Teil gegriffen wurde. Das Teil fällt also zurück in die Kiste.
- Eine Bewegung zurück zur Übergangspose geplant und ausgeführt.
- Wurde die Bewegung zur Übergangspose erfolgreich ausgeführt, so wird der Mikado Programmablauf mit dem Sequenzwechsel ‚Griff gescheitert' fortgesetzt.

3.7.6 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. Verwenden Sie auch zum Loslassen eines gegriffenen Teils auf keinen Fall den Task *Hand öffnen/schließen*, sonst öffnet sich zwar der Greifer, in der Kollisionswelt ist das Teil jedoch noch vorhanden.

3.7.7 Funde löschen

Alle Funde von 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.

3.7.8 Statisches Objekt hinzufügen

Füge eine Instanz eines Produkts an einer beliebigen Position in die Welt ein. Dieses wird zB bei der Pfadplannung umfahren und dient zur Visualisierung. Es kann in der aktuellen Version noch nicht gegriffen oder andersweitig bewegt werden.

Hinweis: Die Objekt ID muss eindeutig sein. Momentan wird auch noch nicht überprüft ob der Namen mit dem eines Szeneobjekts (Kiste etc) oder eines Fundes übereinstimmt („Produktname-NUMMER“). Diese sollten vermieden werden, es kann sonst zu seltsamem Verhalten führen!

Variablen

Name	Typ	Beschreibung
Produktname	String	Bezeichnung des Produkts
Objekt ID	String	Bezeichner des Objekts (zB zum Entfernen)
Objektpose	Pose	An welcher Position soll das Objekt erzeugt werden

3.7.9 Statisches Objekt entfernen

Entfernt eine Instanz eines Produkts aus der Welt.

Variablen

Name	Typ	Beschreibung
Objekt ID	String	Bezeichner des Objekts (zB zum Entfernen)

3.8 Transformationen

3.8.1 Pose Auftrennen

Dieser Task dient dazu Posen in ihre Positions- und Rotationskomponenten aufzuteilen. 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.

3.8.2 Pose Erstellen

Dieser Task dient dazu Posen aus Positions- und Rotationskomponenten zu erstellen. Eingabewerte sind die xyz-Distanzen und Eulerwinkel, also Roll, Nick und Gier.

Variablen

Name	Typ	Beschreibung
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: Einzelne Eingabewerte können weggelassen werden und dann auf 0.0m bzw. 0.0° gesetzt.

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.

3.8.3 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	–
Roll	Double	Winkel in Grad
Nick	Double	Winkel in Grad
Gier	Double	Winkel in Grad
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 (lokal) gedreht, die Position der Pose hat sich also nicht geändert. Die Gierdrehung bezog sich auf die blaue, lokale z-Achse.

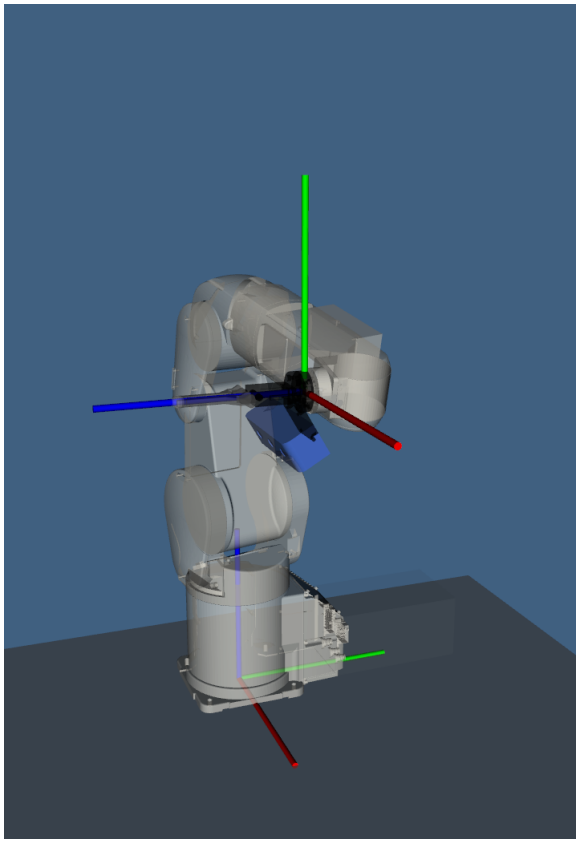


Abb. 1: Screenshot Pose initial

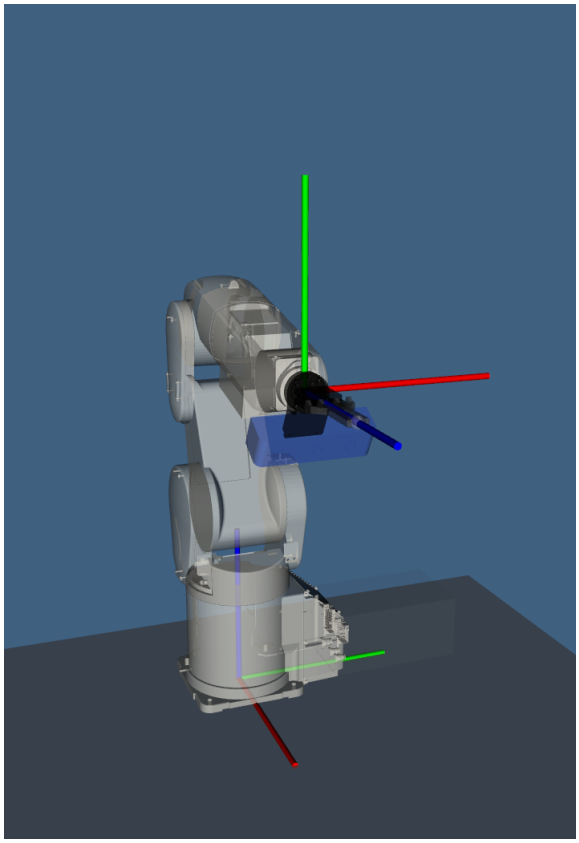


Abb. 2: Screenshot Pose nach globaler Rotation

3.8.4 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	–
Roll	Integer, Double	Winkel in Grad
Nick	Integer, Double	Winkel in Grad
Gier	Integer, Double	Winkel in Grad
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 lokal gedreht, die Position hat sich also nicht geändert. Die Gierdrehung bezog sich auf die lokale z-Achse (blau), die sich dadurch auch nicht verändert hat.

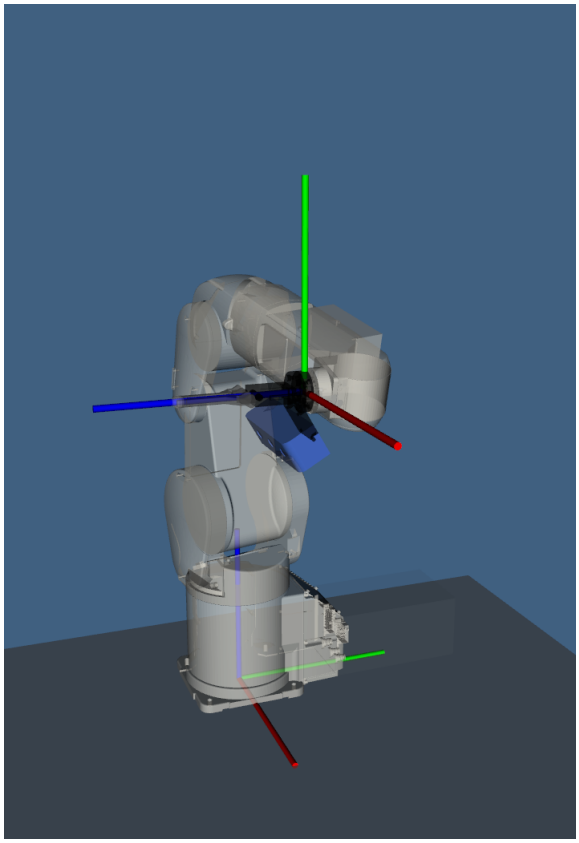


Abb. 3: Screenshot Pose initial

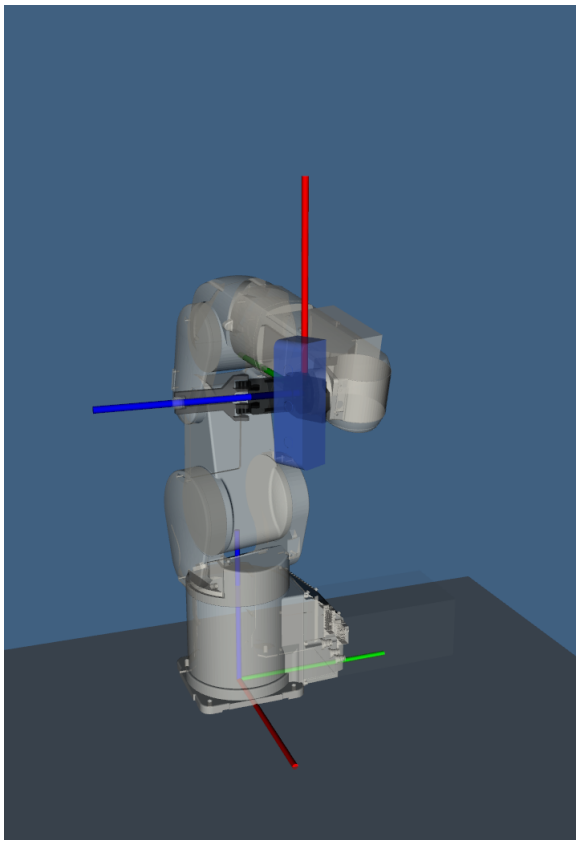


Abb. 4: Screenshot Pose nach lokaler Rotation

3.8.5 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	–
x	Double	Distanz in Meter
y	Double	Distanz in Meter
z	Double	Distanz in Meter
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.

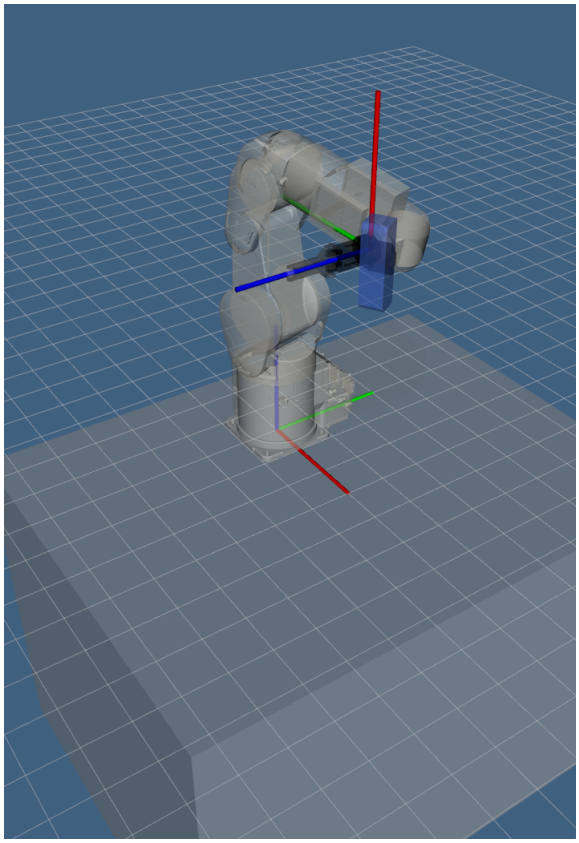


Abb. 5: Screenshot Pose initial

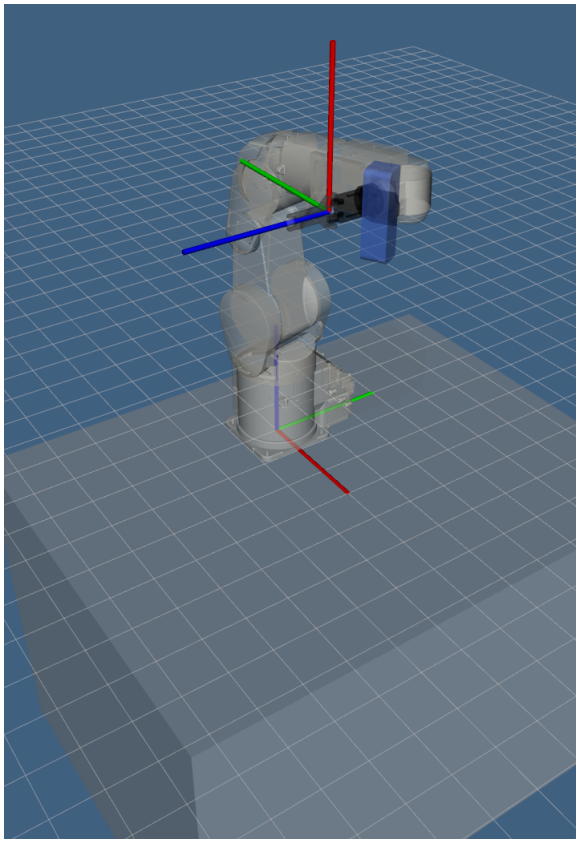


Abb. 6: Screenshot Pose nach globaler Verschiebung

3.8.6 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	–
x	Double	Distanz in Meter
y	Double	Distanz in Meter
z	Double	Distanz in Meter
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.

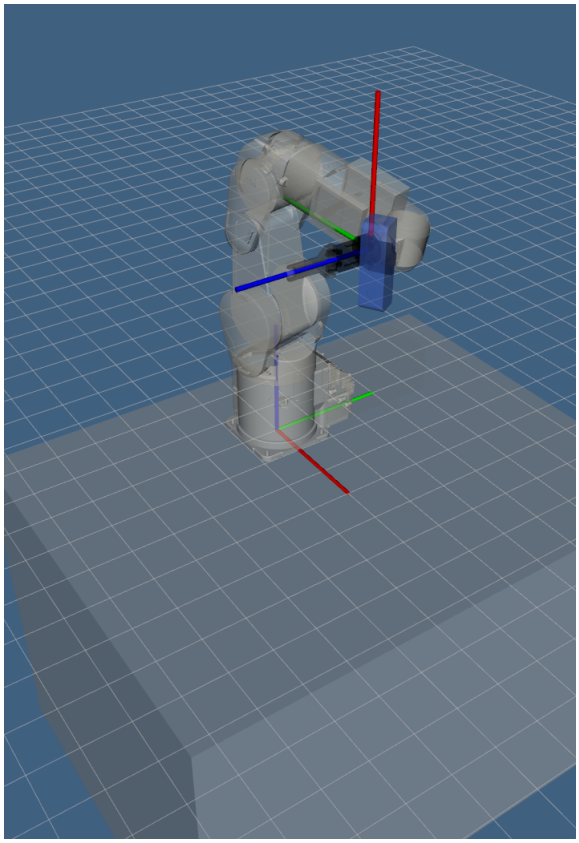


Abb. 7: Screenshot Pose initial

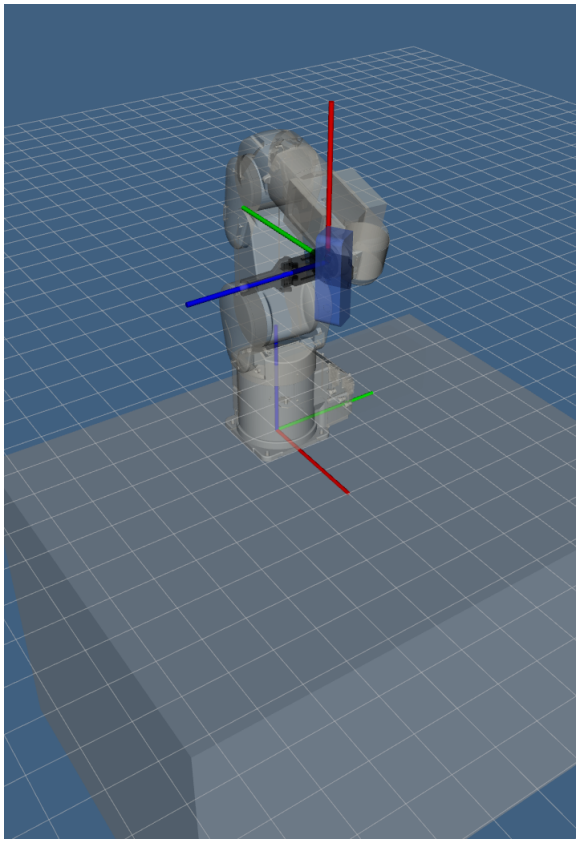


Abb. 8: Screenshot Pose nach lokaler Verschiebung

3.8.7 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. Um z.B. eine Verschiebung in Z-Richtung des Weltkoordinatensystems zu erreichen, verwenden Sie die Offset-Pose als Eingabe-Pose und die zu verschiebende Pose dann 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)

3.8.8 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

3.9 Erweiterungen

3.9.1 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 ARC Erweitern* / *JavaScript Task*.

Variablen

Name	Typ	Beschreibung
Code	JScript	Das auszuführende Code-Schnipsel als Quelltext

3.9.2 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 ARC 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=7mm

- 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 komplexe Mesh Dateien vorverarbeitet werden, 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: * ROI setzen, 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*.

4.2.1 Neues Produkt / Produktliste einrichten

- Auf die Seite „Edit“ → „Produkte“ wechseln

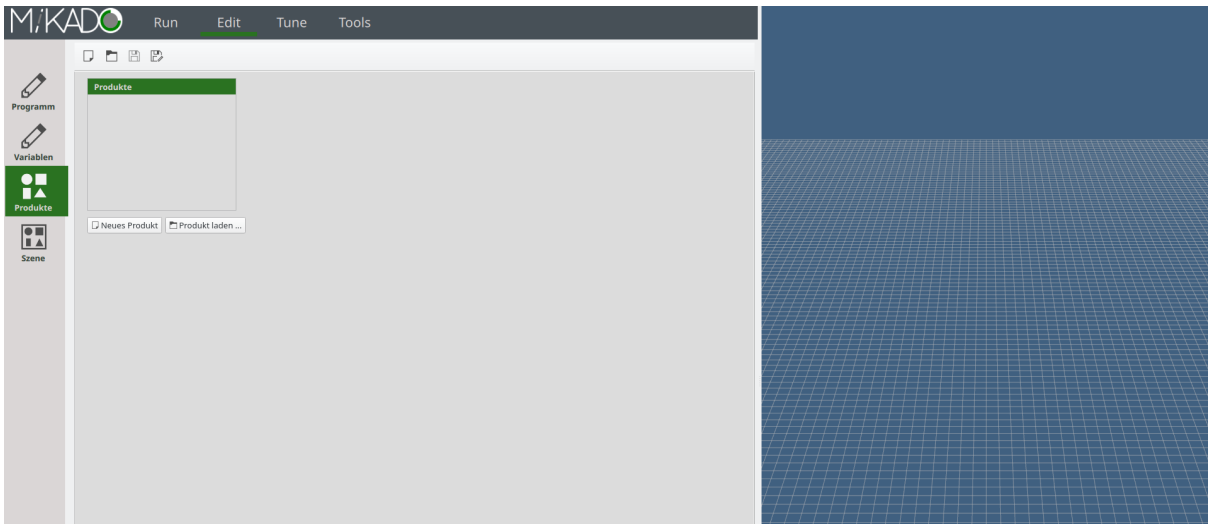


Abb. 1: Produktseite: Leere Ansicht, wenn noch keine Produkte angelegt wurden

- Der Button „Neues Produkt“ fügt der Produktliste ein neues Produkt mit dem Namen „Produkt1“ hinzu

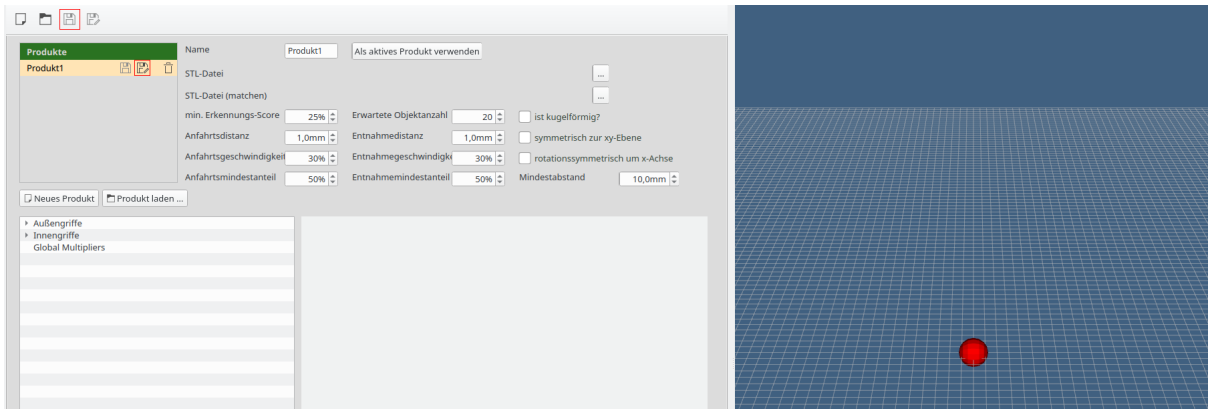


Abb. 2: Produktseite: Ansicht, wenn ein neues Produkt angelegt wurde

- 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.

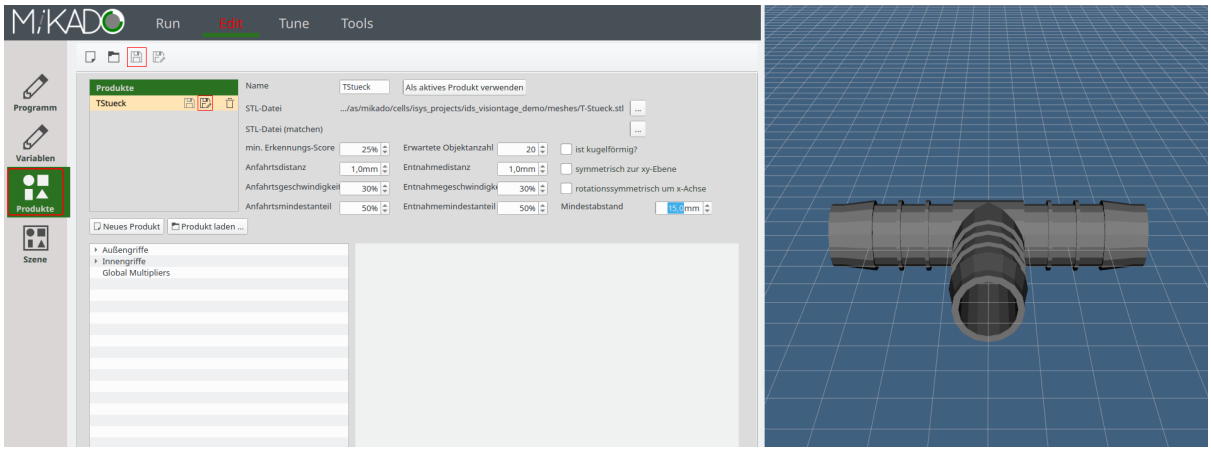


Abb. 3: Produktseite: Ansicht, wenn ein CAD-Modell ausgewählt wurde

- 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 diesen Produkt-Parametern und weitere Parameter 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
- ROI aktivieren → 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 *Szenenverwaltung*.

- „Punktwolke aufnehmen“ → „Anwenden und Objekte finden“
- Alle Funde größer dem eingestellten Mindesterkennungsscore werden werden visualisiert. Der Erkennungsscore liegt jeweils über einer erkannten Objektinstanz.

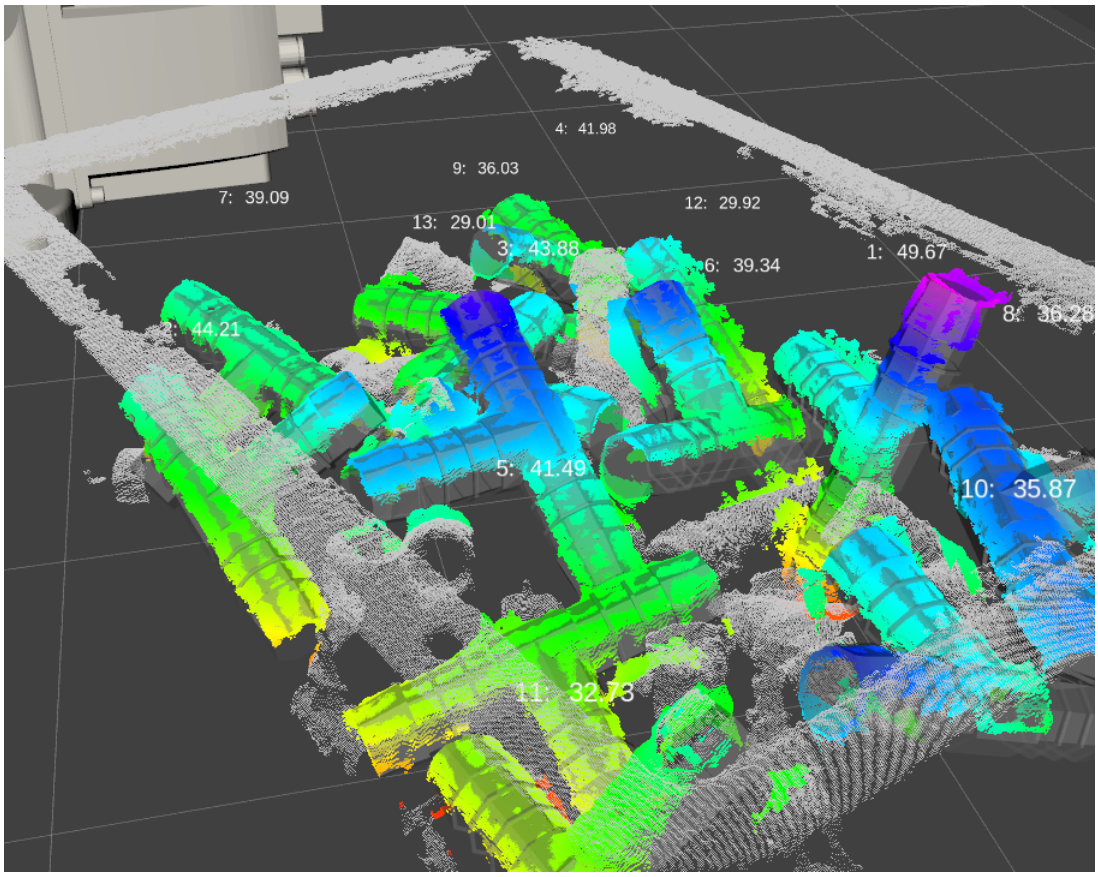


Abb. 4: Objekterkennung: Gefundene Teile werden grau/transparent eingezeichnet

- Sie können nun bewerten, welche Scores erreicht werden können und bei welchen Werten Ungenauigkeiten oder Fehlfunde auftreten. In unserem Beispiel mit den „T-Stücken“ liegen die Scores bei Teilen ohne Verdeckung bei über 40%. Daher ist dies hier ein geeigneter Mindestscore. Diesen können Sie nun auf der Produktseite („Edit“ → „Produkte“) anpassen und speichern.
- 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. Auf dem folgenden Bild ist ein Fund zu sehen, bei dem die Ausrichtung nicht korrekt erkannt werden kann. Der untere Teil des T-Stücks ist nicht in der Punktwolke zu sehen und konnte daher nicht korrekt erkannt werden (rot eingekreist). Daraus können Sie schließen, dass ein Mindestscore für das T-Stück bei mindestens 25% liegen muss.

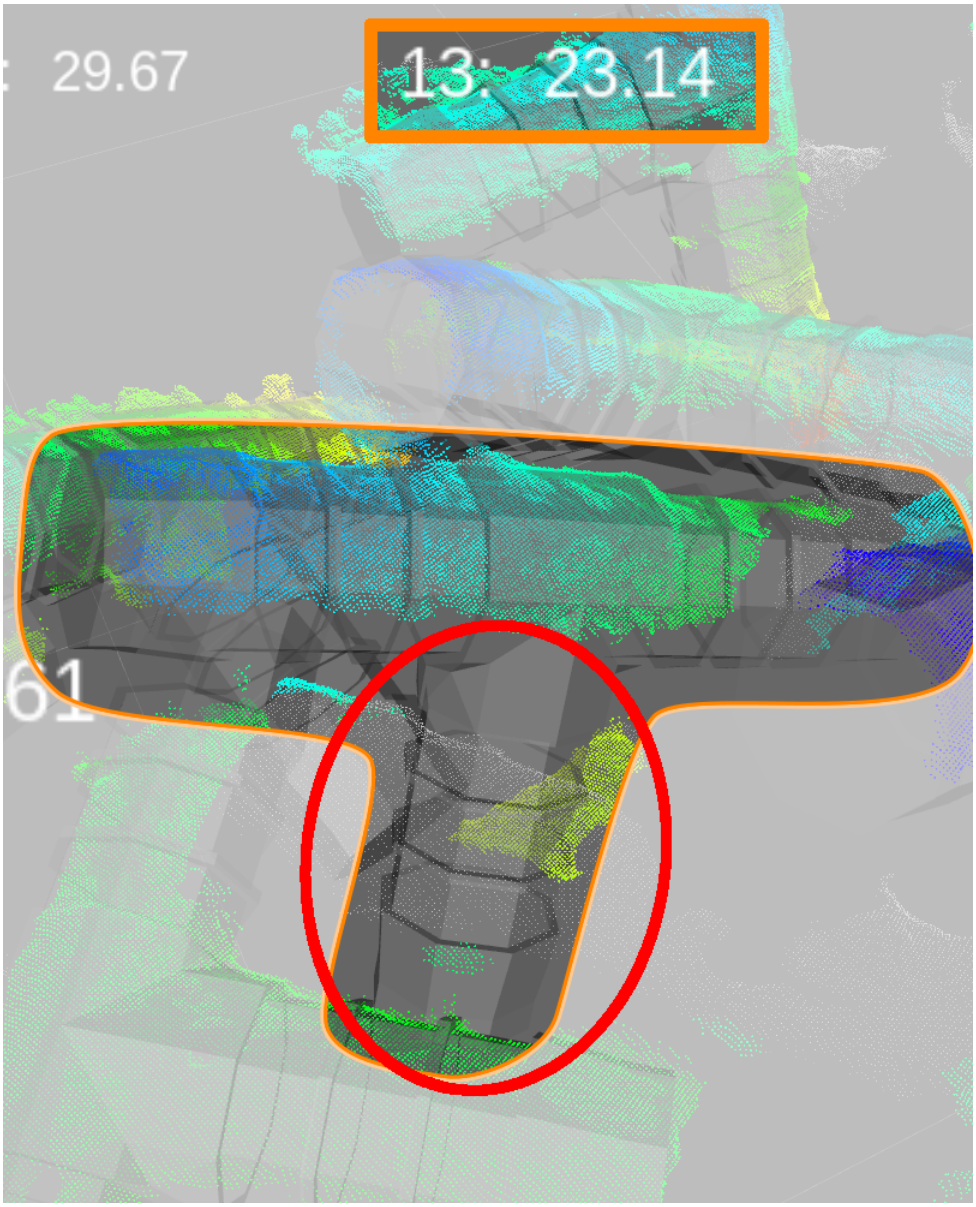


Abb. 5: Beispiel eines schlechten Fundes mit niedrigem Score von 23,14%. Die Ausrichtung konnte nicht korrekt erkannt werden (rot eingekreist).

- Führen Sie nun weitere Tests durch, um Ihren neuen Mindestscore 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.

Hinweis: Vertiefende Informationen zur Parametrierung und Nutzung der Objekterkennung sind unter *Objekterkennungs-Tool* zu finden.

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

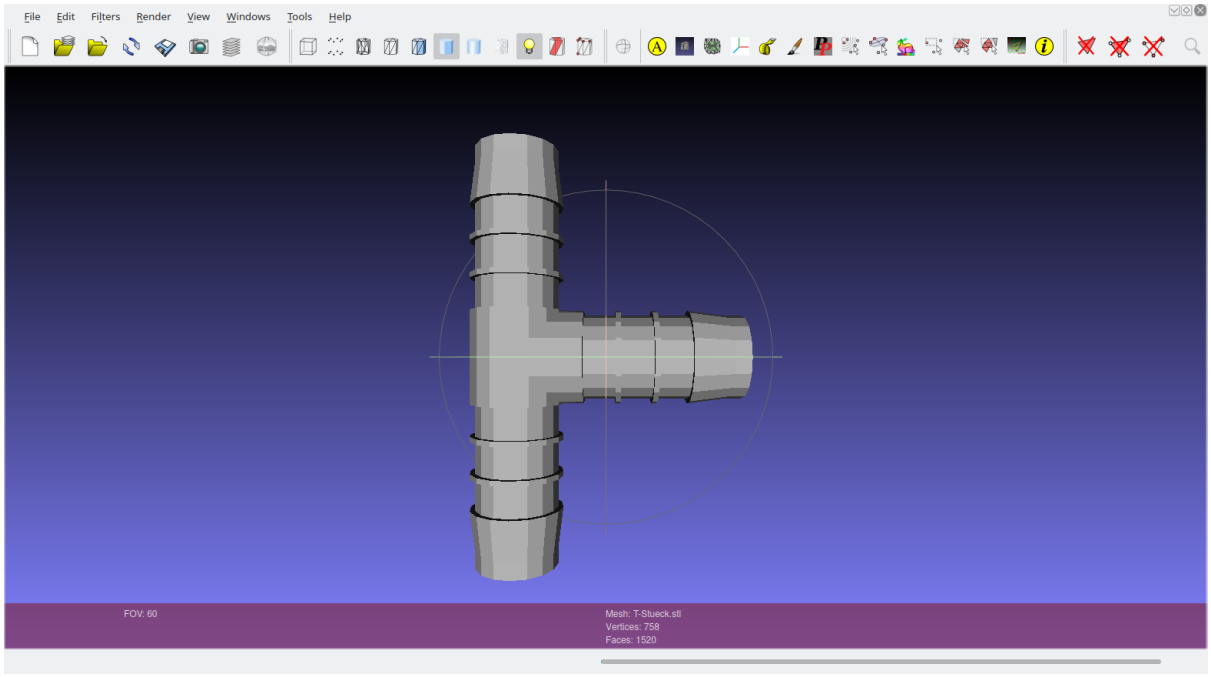


Abb. 6: MeshLab GUI

- 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

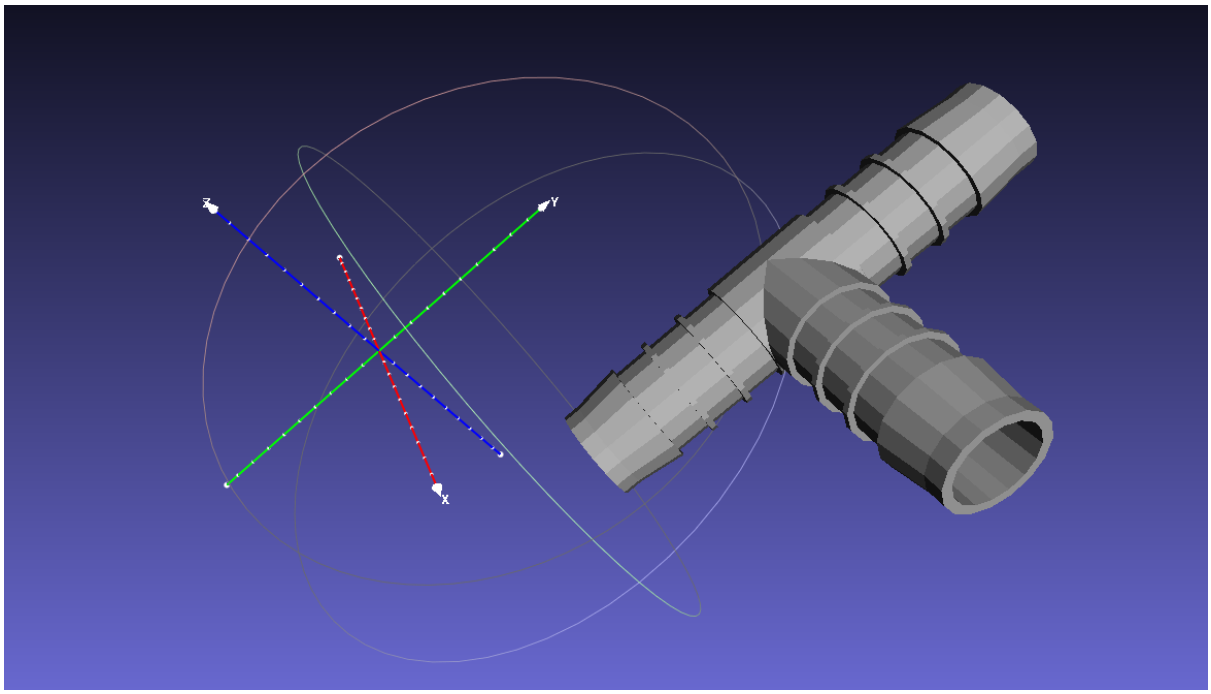


Abb. 7: MeshLab: Koordinatensystem anzeigen

- Koordinatensystem zentrieren
  - Filters → Normals, Curvatures and Orientation → Transform: Move, Translate, Center
  - „translate center of bbox to the origin“
  - Apply

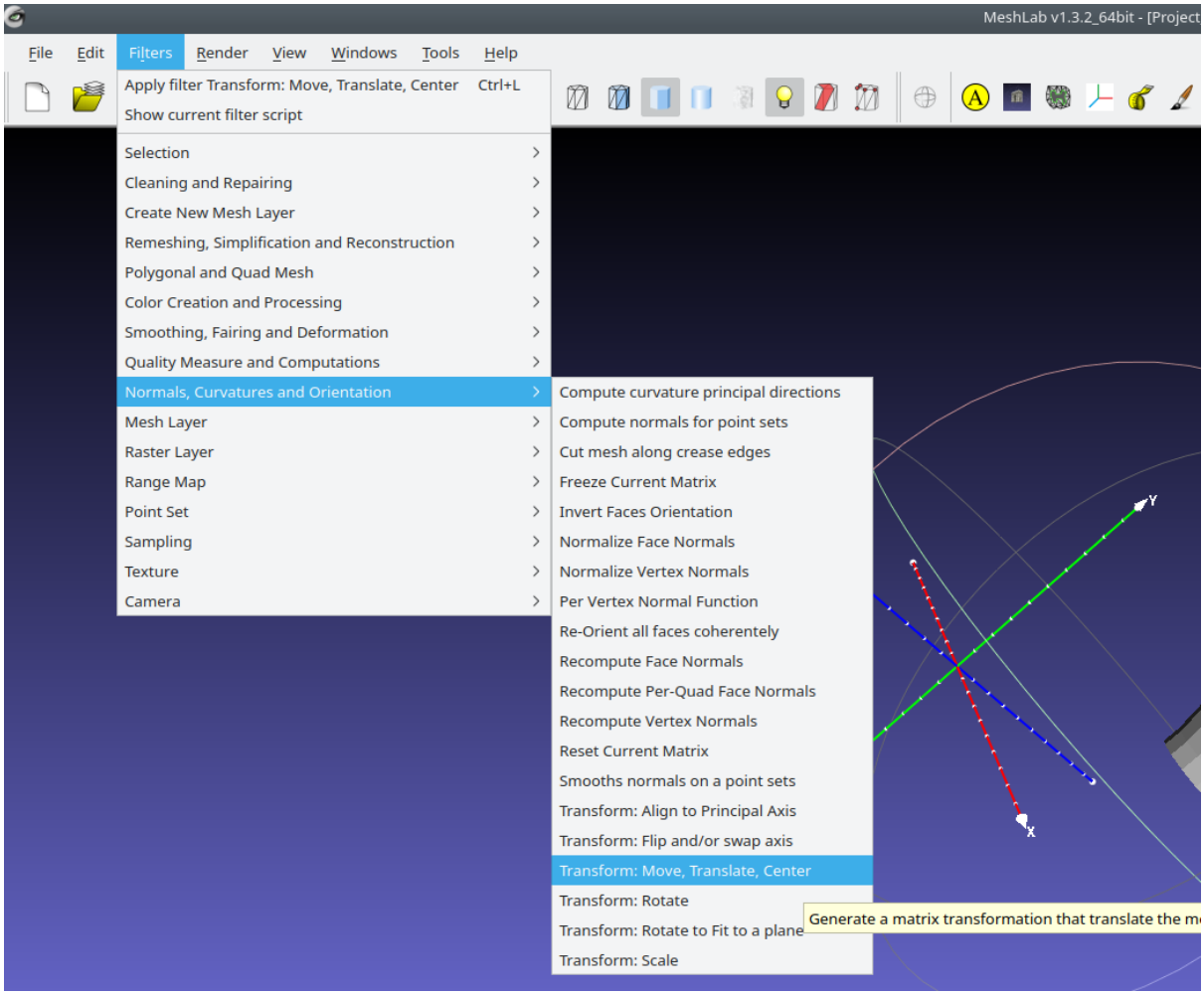


Abb. 8: MeshLab: Koordinatensystem transformieren

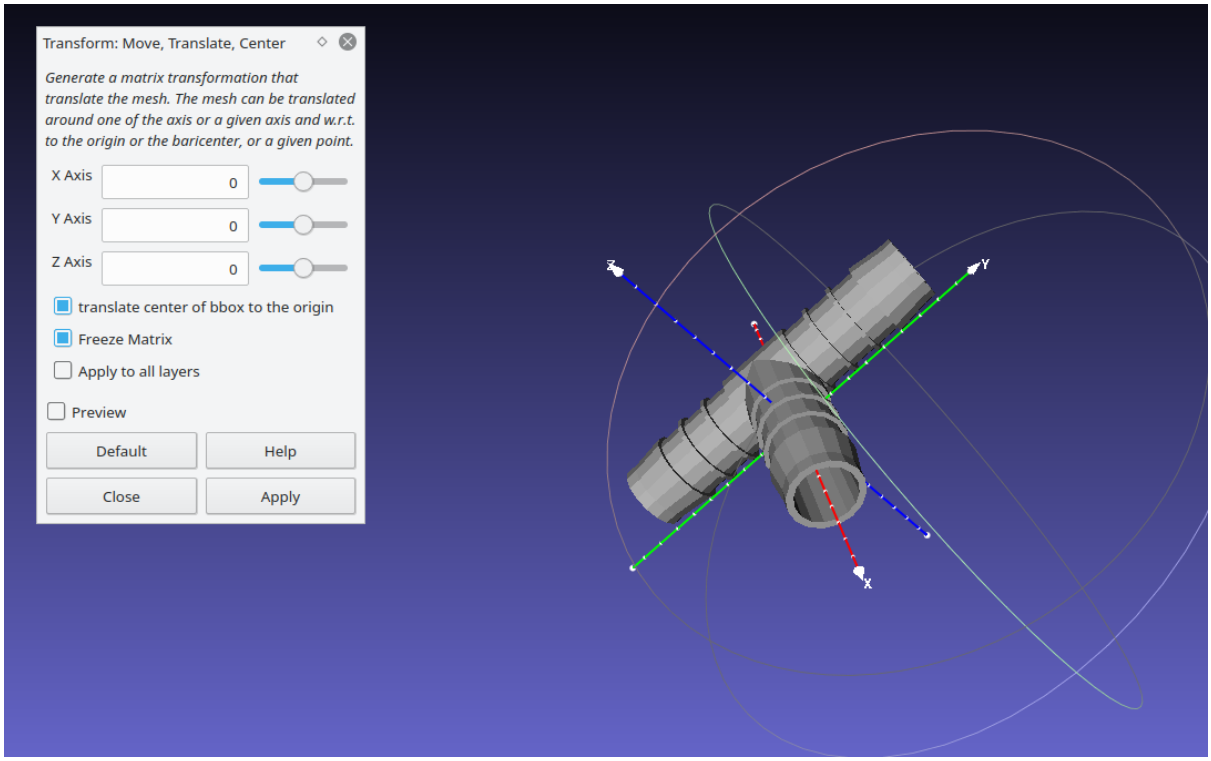


Abb. 9: MeshLab: Koordinatensystem zentriert

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

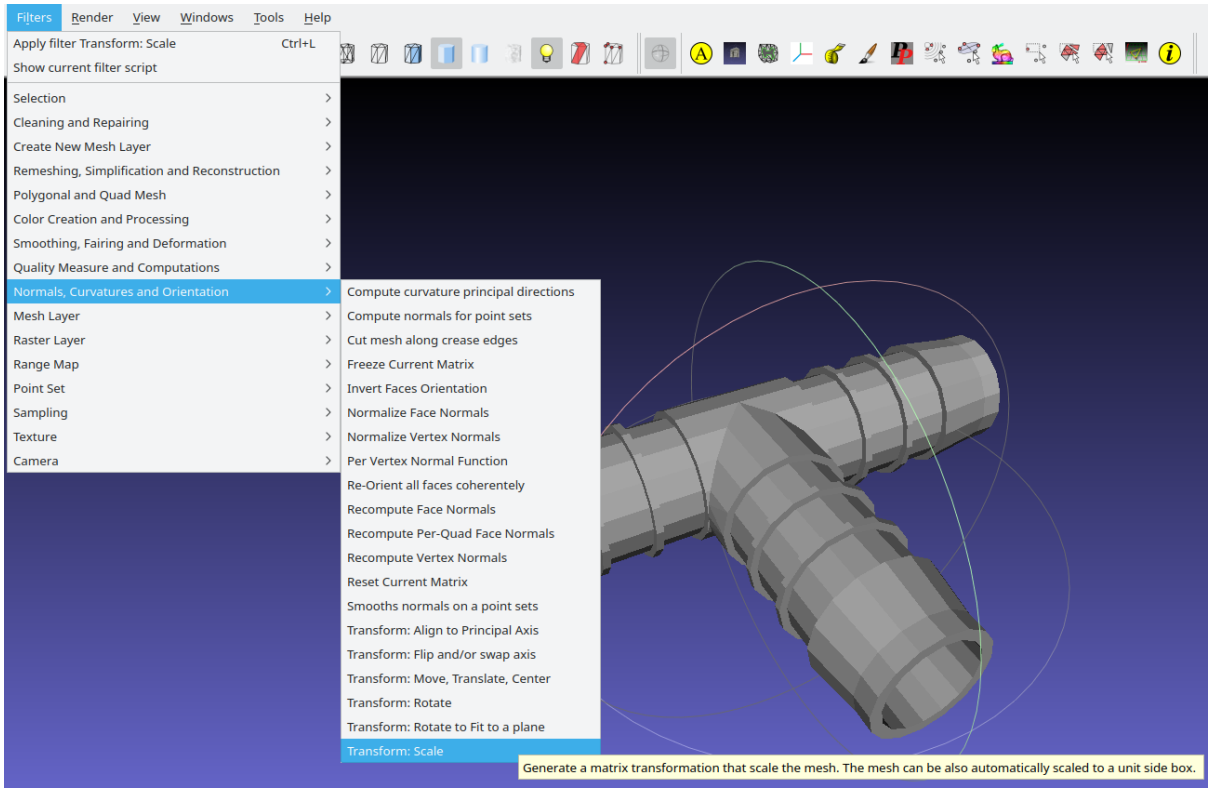


Abb. 10: MeshLab: Filter zum Skalieren

- „Uniform Scaling“ aktivieren
- „Preview“ aktivieren, damit Änderungen sofort dargestellt werden
- Faktor für „X axis“ anpassen (1.0 $\rightarrow$ 100%, 0.01 $\rightarrow$ 1%). Möchten Sie z.B. ein CAD, das in Meter definiert wurde zu Millimeter konvertieren, so wählen Sie als Wert 1000.0
- Apply

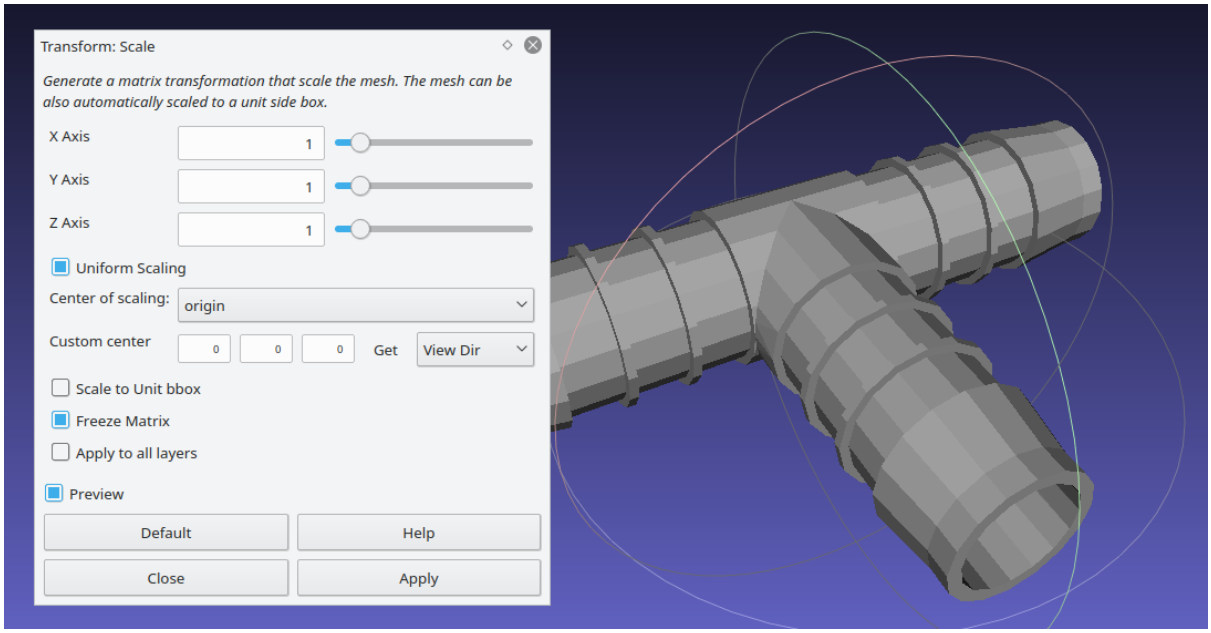


Abb. 11: MeshLab: Filter zum Skalieren - Einstellungen

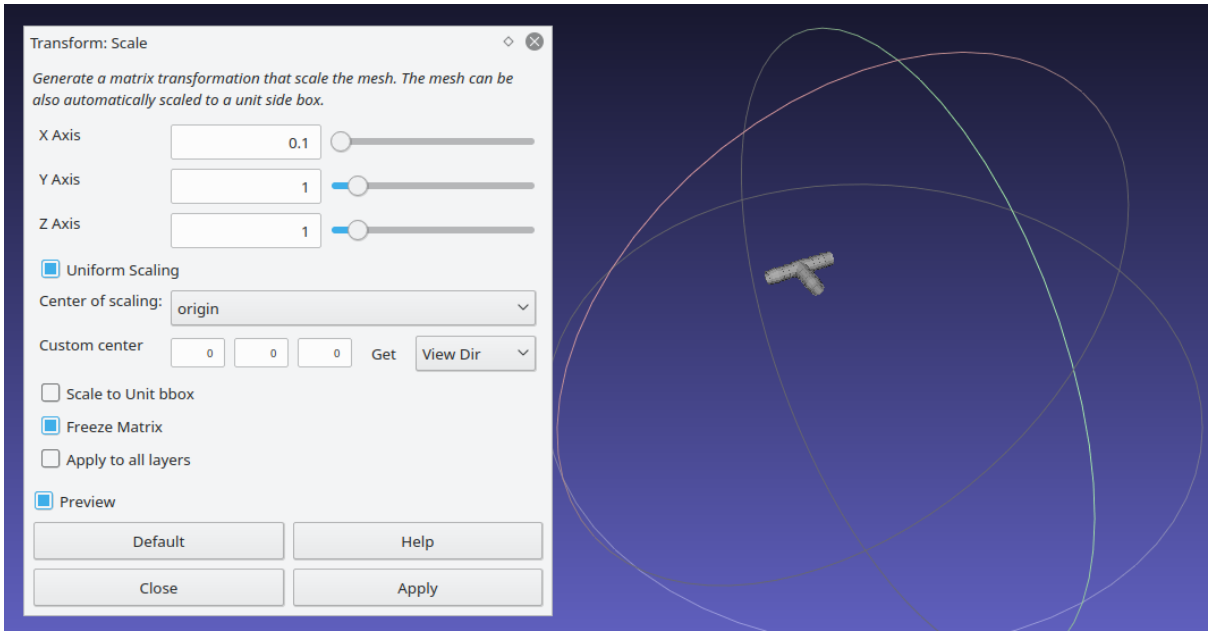


Abb. 12: MeshLab: Skaliertes STL

- 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

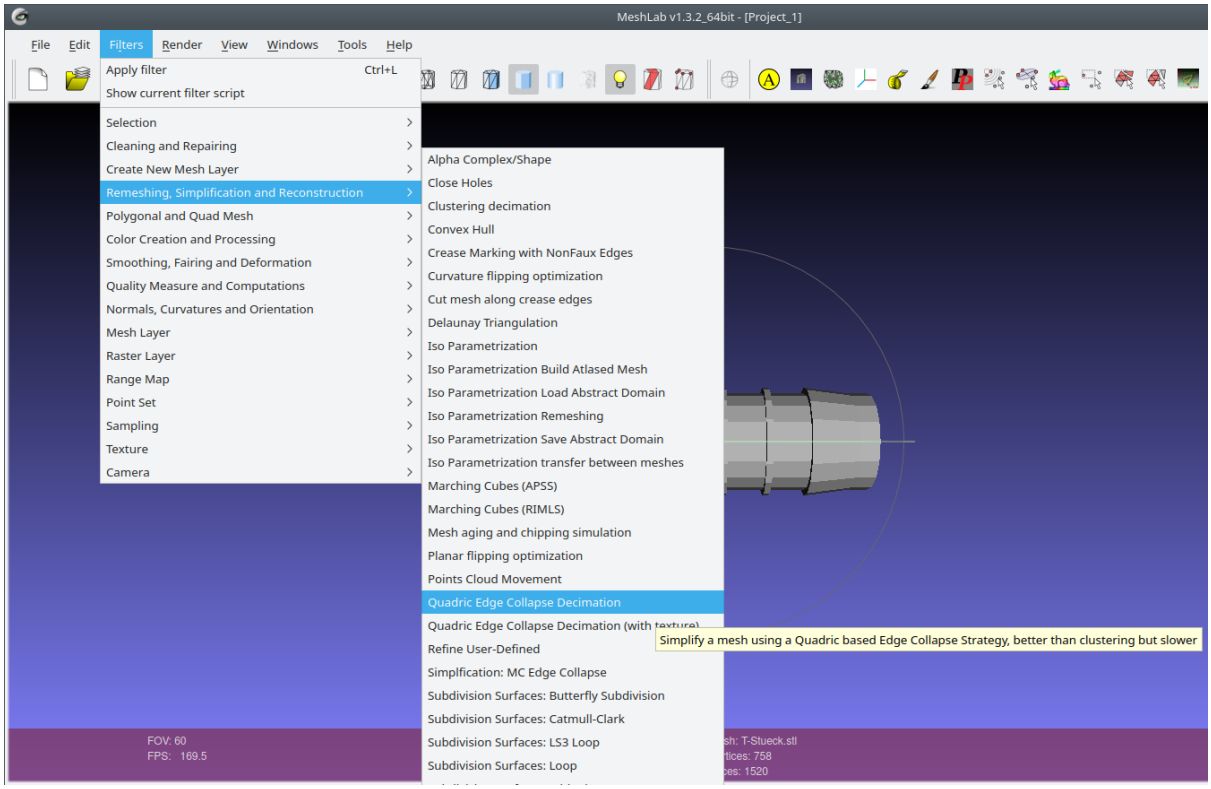


Abb. 13: MeshLab: Dezimierungsfilter

- Gewünschte Anzahl Dreiecke angeben → Apply
  - Bevor eine andere Parametrierung angewendet wird, zuerst Modell neu laden (Button „Reload“: )

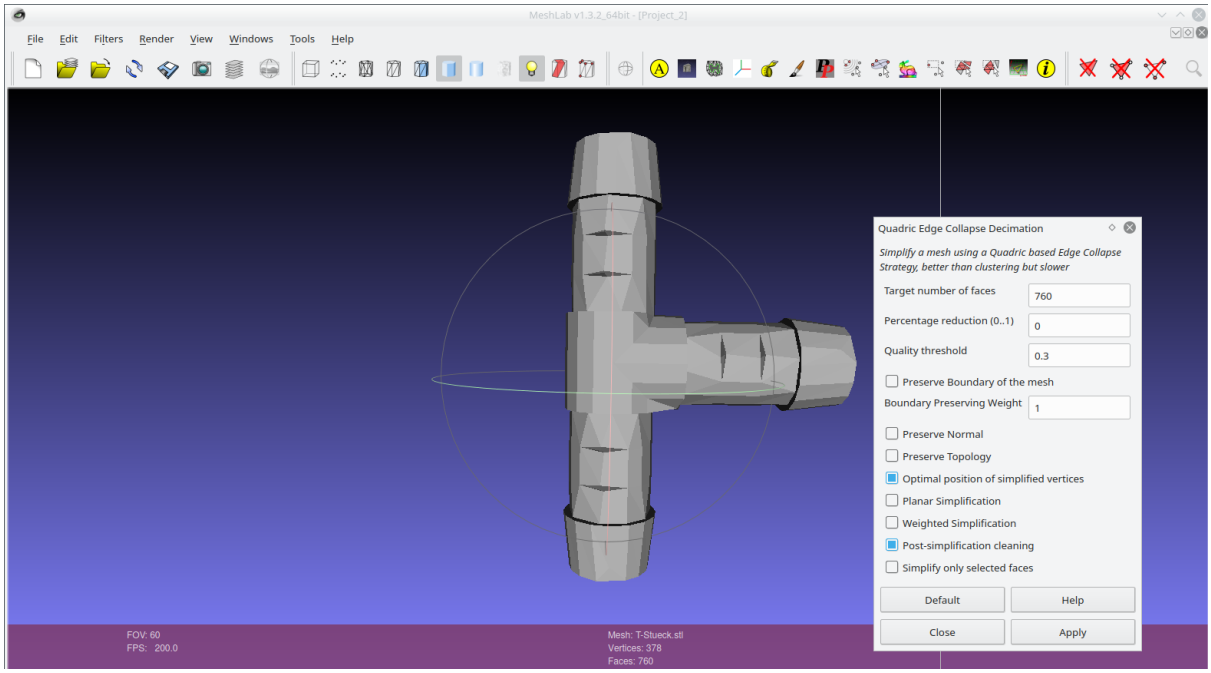
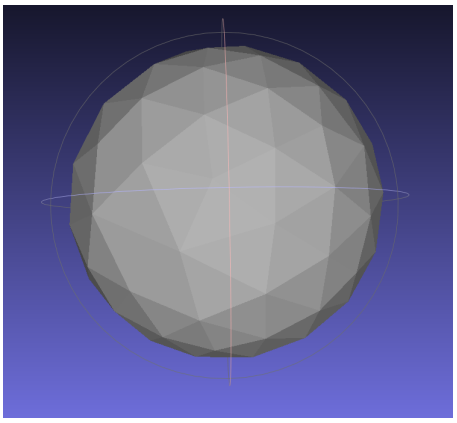


Abb. 14: MeshLab: Dezimiertes Modell des T-Stücks

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.



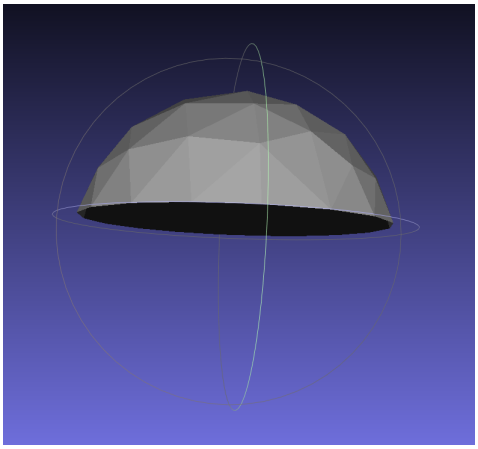


Abb. 15: Beispiel: Für ein Objektmatching einer Kugel führt eine Halbkugel zu höheren Scores. Das Kollisionsmodell muss allerdings weiterhin das komplette Produkt abbilden.

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

    ---

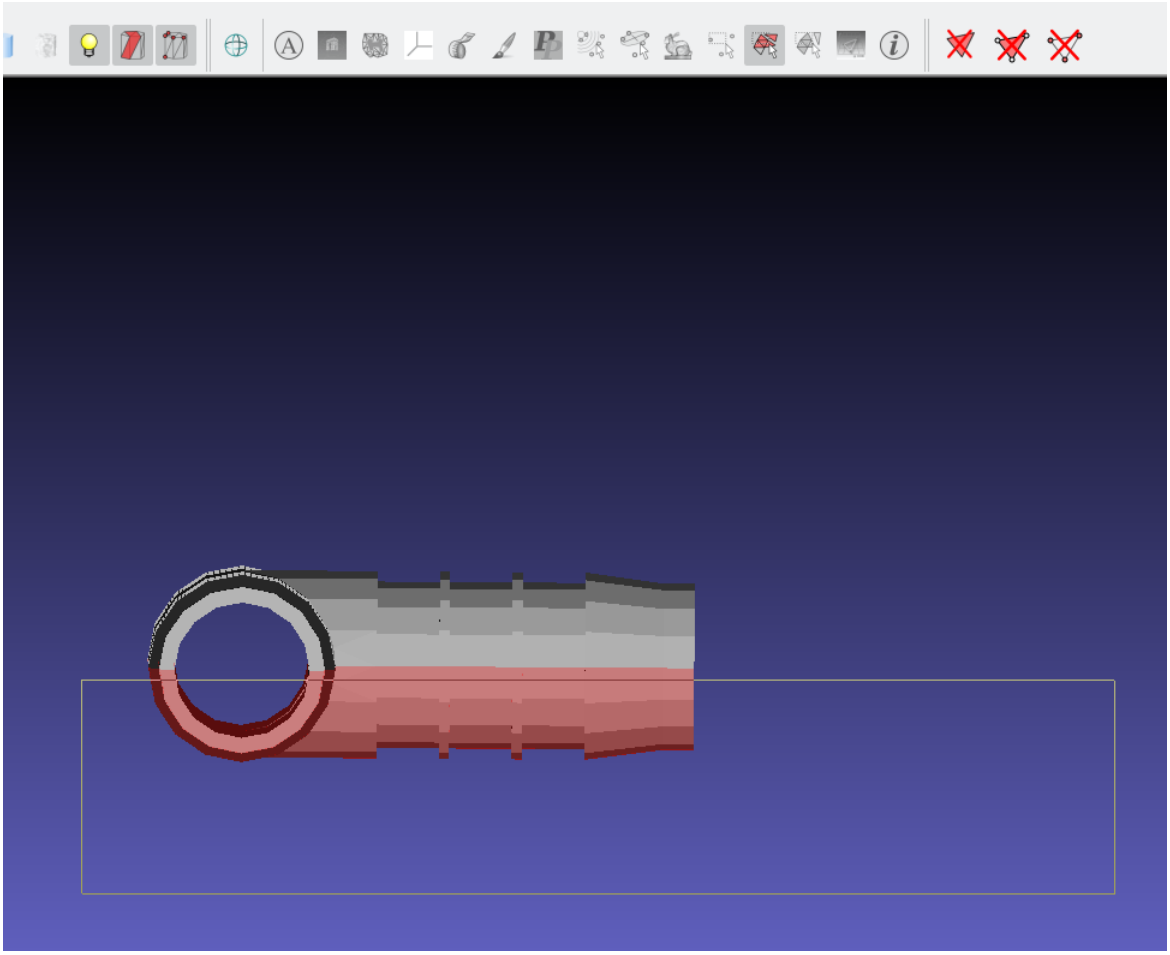


Abb. 16: MeshLab: Rechteckauswahl

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

---

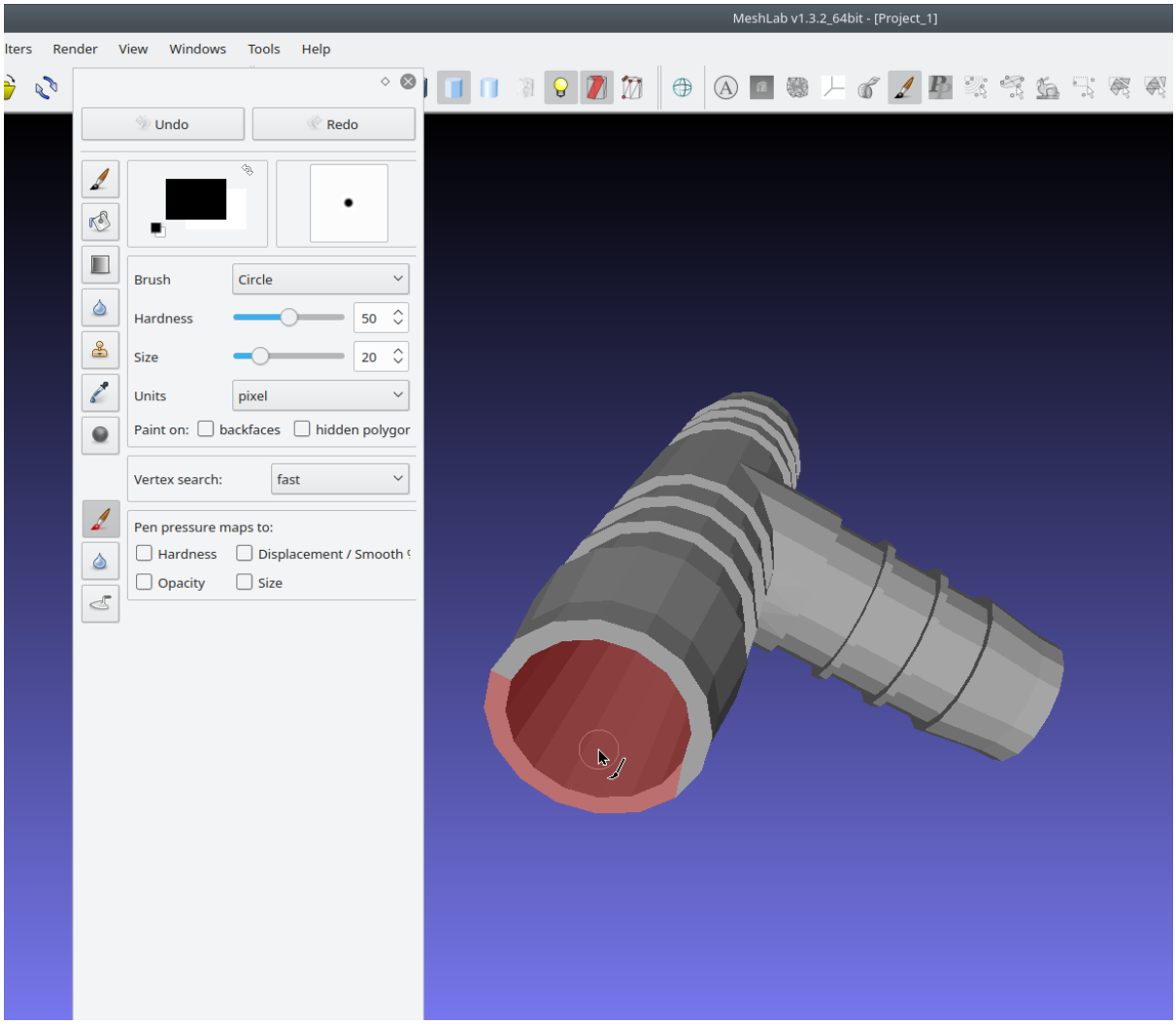


Abb. 17: MeshLab: Pinselauswahl

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.“)

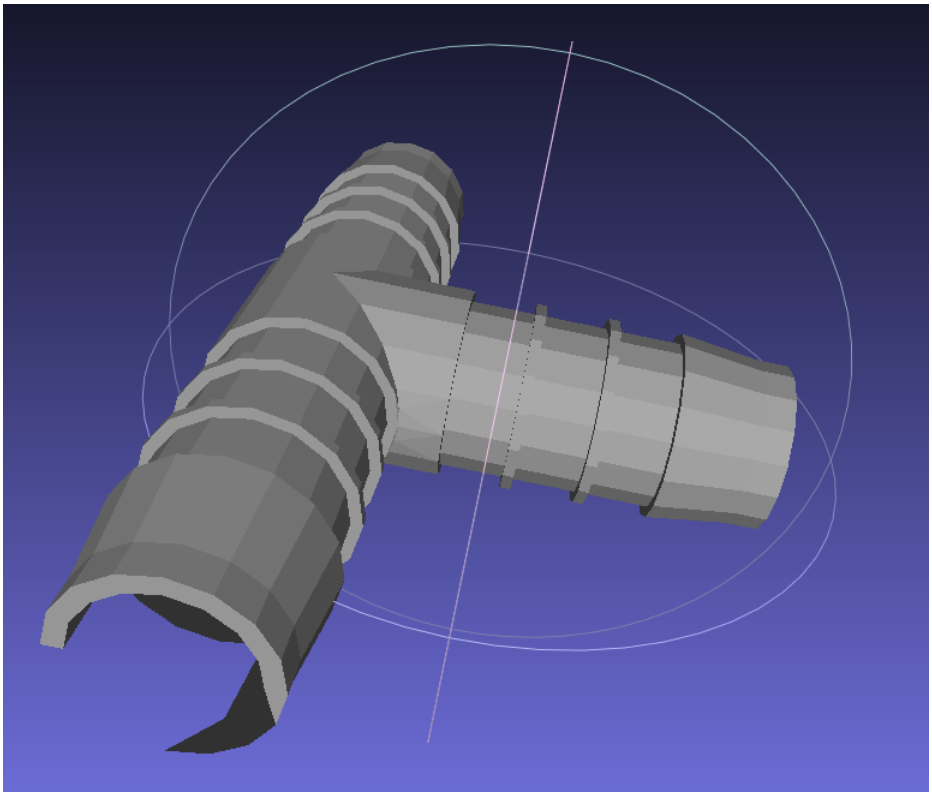


Abb. 18: MeshLab: Ausgewählte Dreiecke entfernt

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

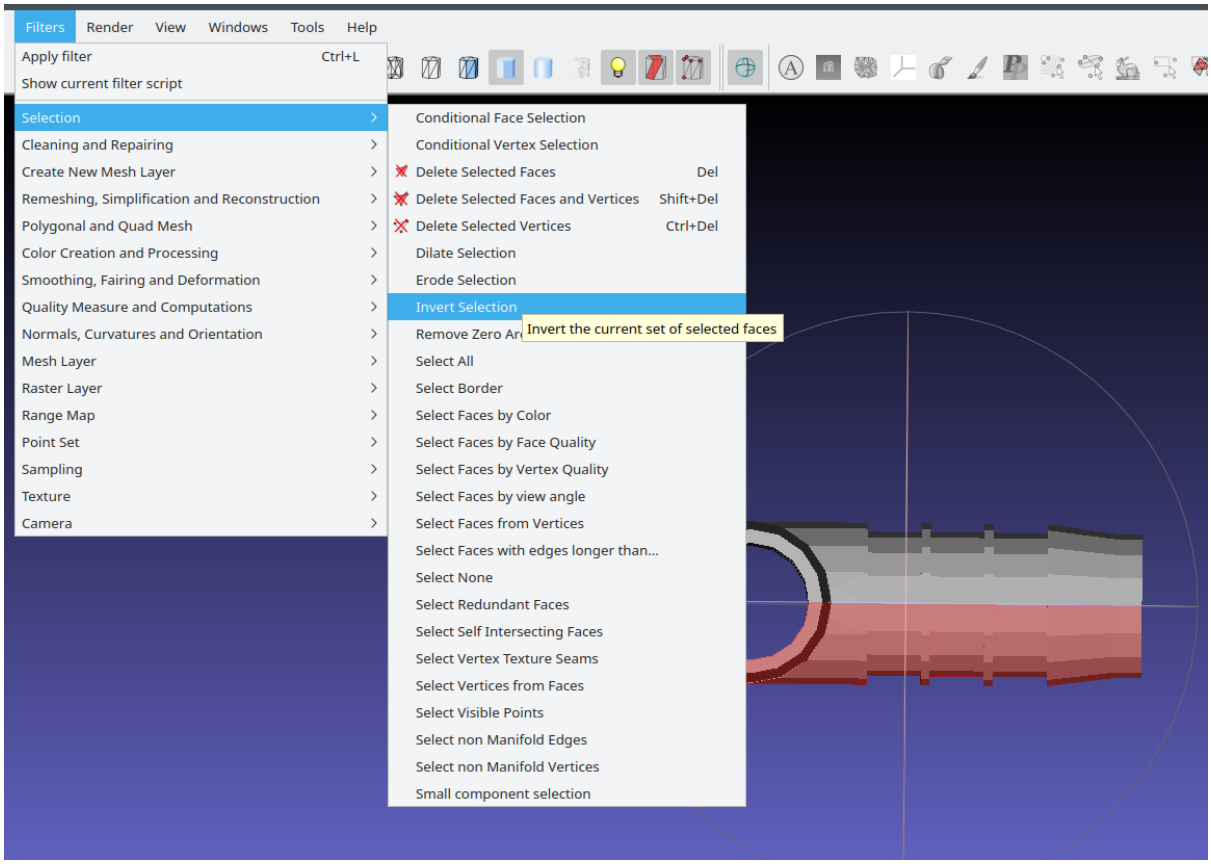


Abb. 19: MeshLab: Auswahl invertieren

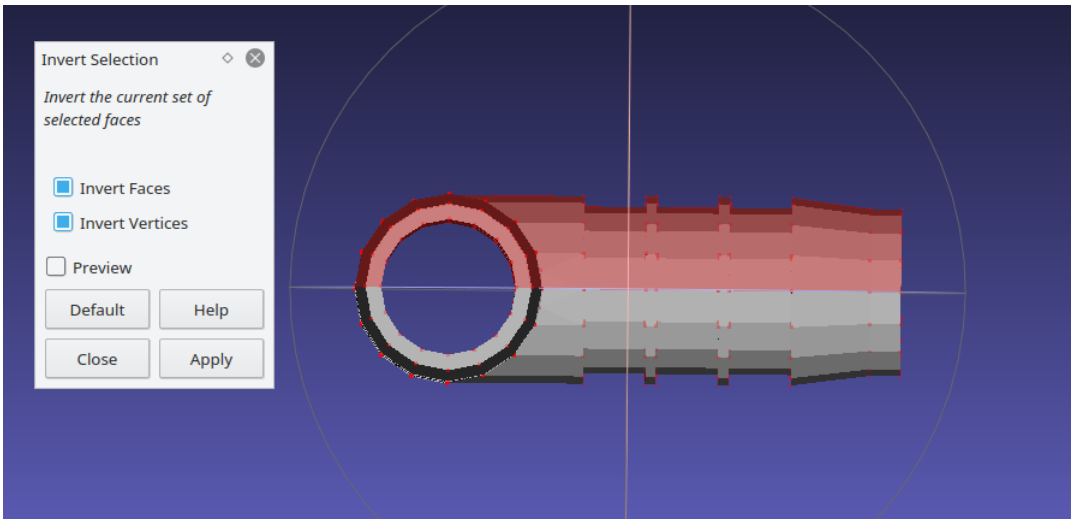


Abb. 20: MeshLab: Auswahl invertiert

- 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*)

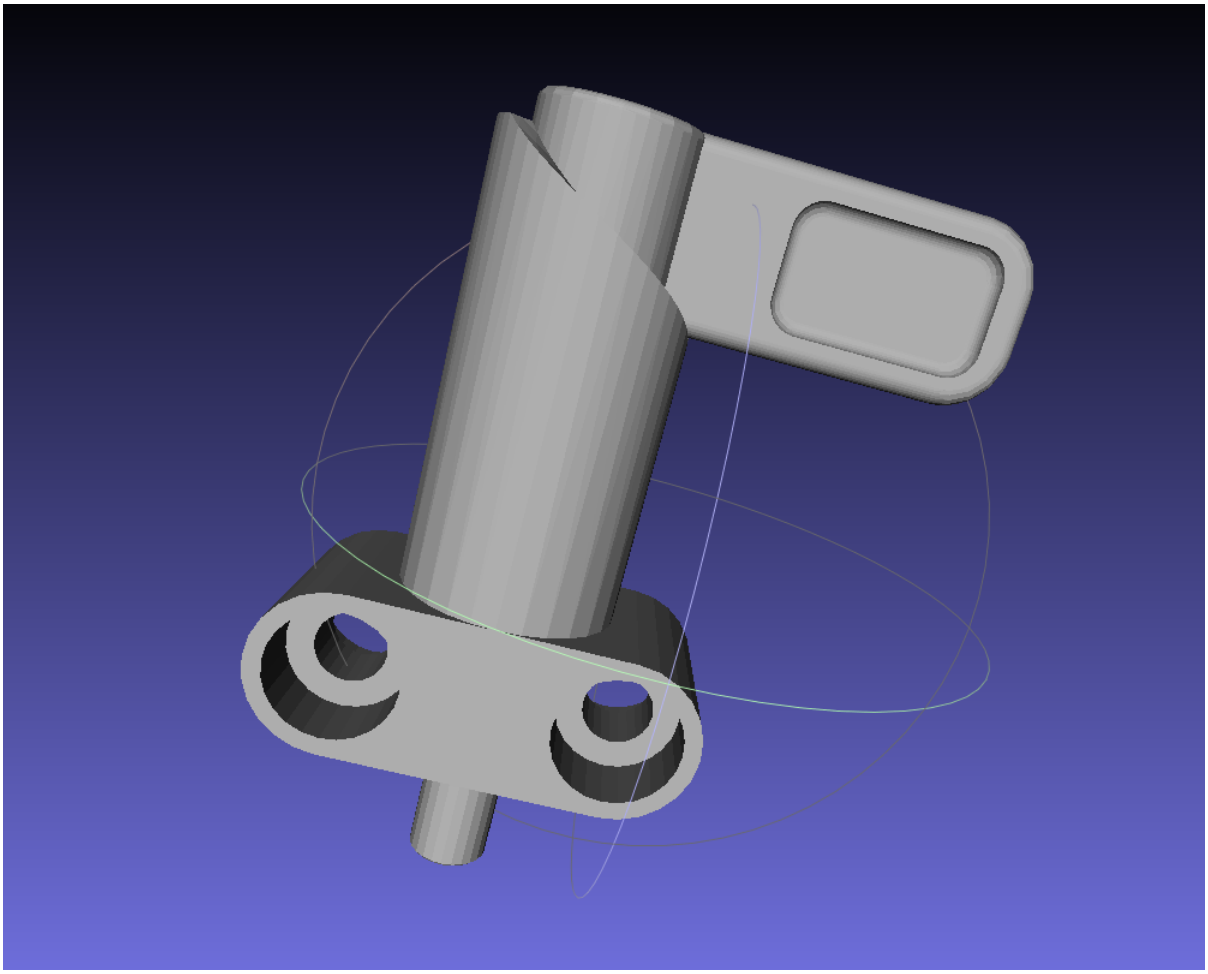


Abb. 21: MeshLab: Ein Beispiel-Produkt ‚Rastriegel CAD‘

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.

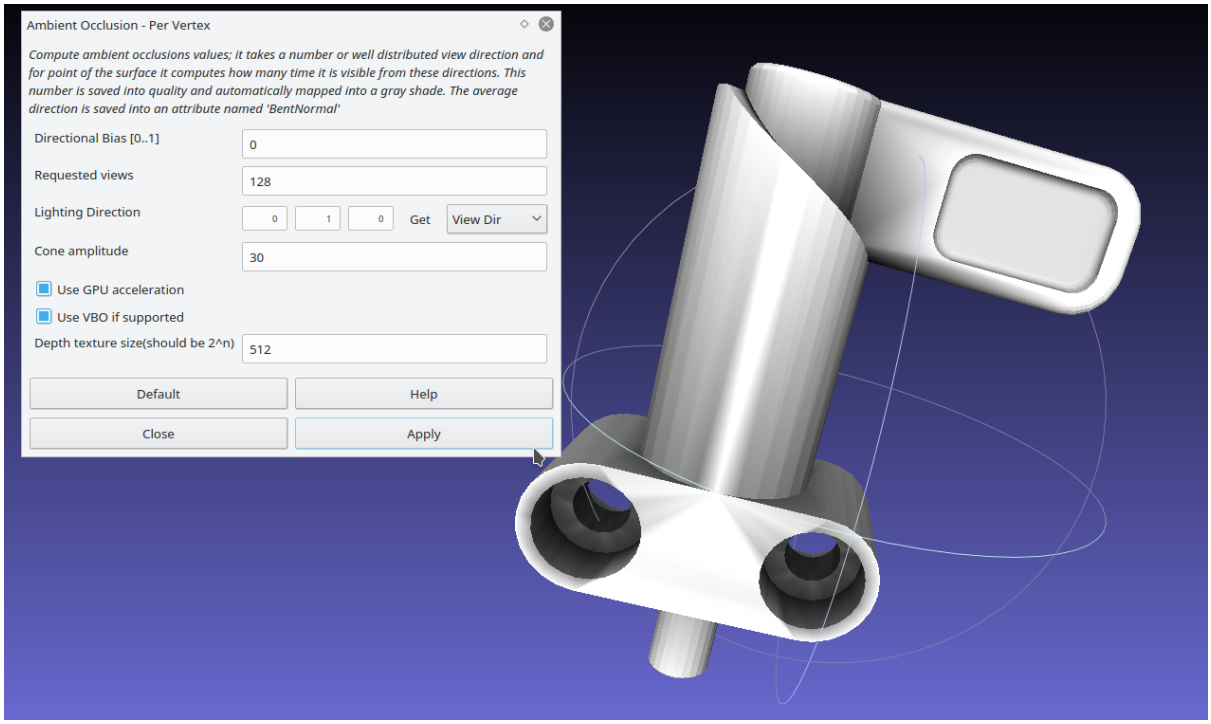


Abb. 22: MeshLab: „Sichtbarkeits-Bewertung“ mithilfe des Ambient Occlusion Filters

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.

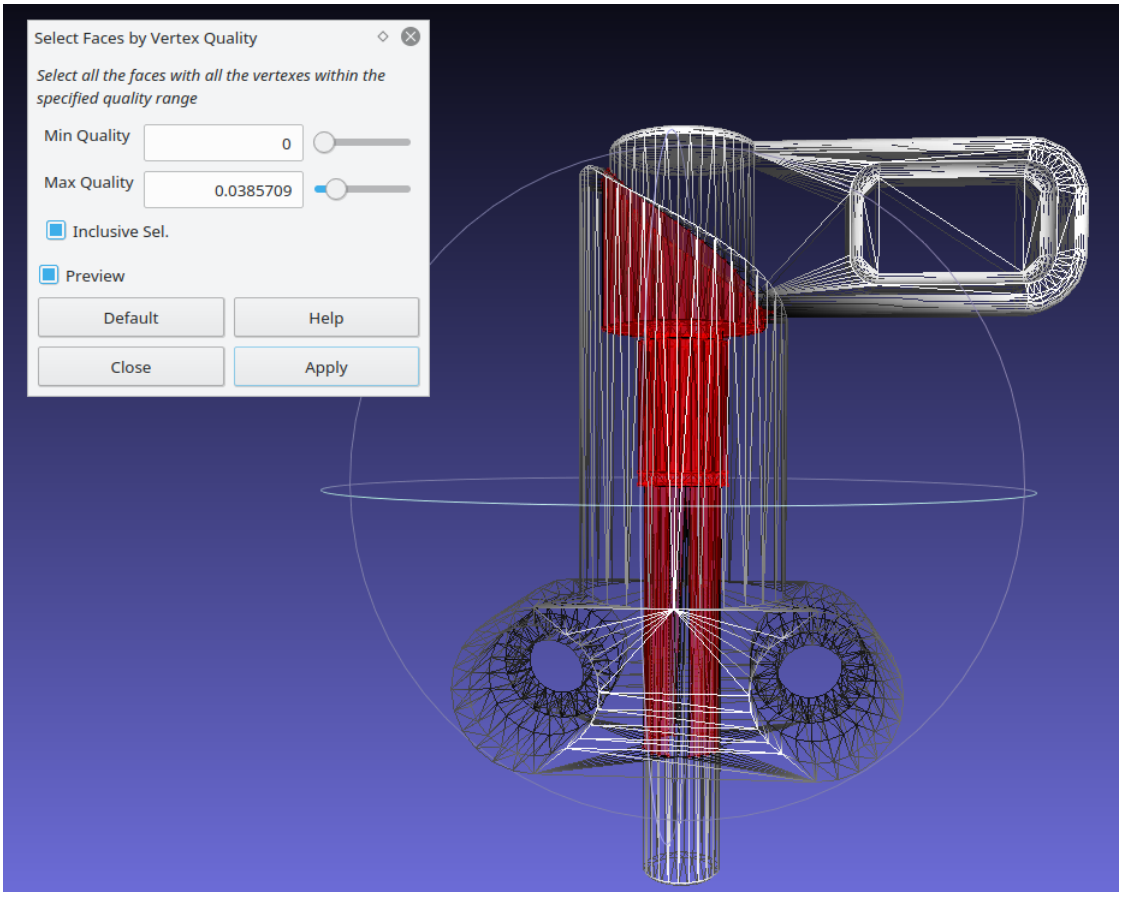


Abb. 23: MeshLab: Selektion anhand der Sichtbarkeit einer Fläche mit niedrigem Schwellwert

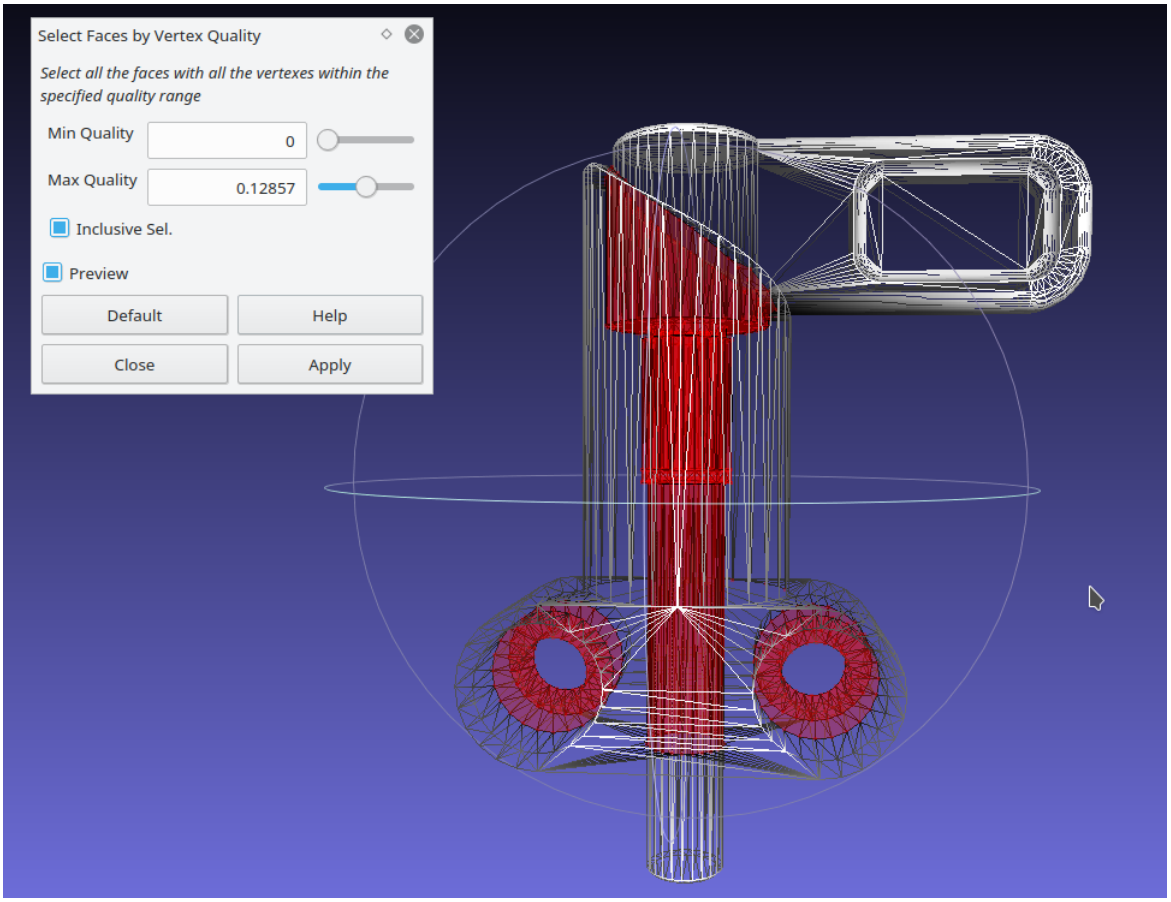


Abb. 24: MeshLab: Selektion anhand der Sichtbarkeit einer Fläche mit höherem Schwellwert: weitere innere Flächen werden selektiert. Im unteren Bereich kommen nun ebenfalls ausgewählte Flächen hinzu, da die Strukturen in einer Vertiefung liegen.

4. **Flächen entfernen**

   Selektierte Dreiecke entfernen („Delete the current set of selected faces and all the vertices surrounded by that faces.“)

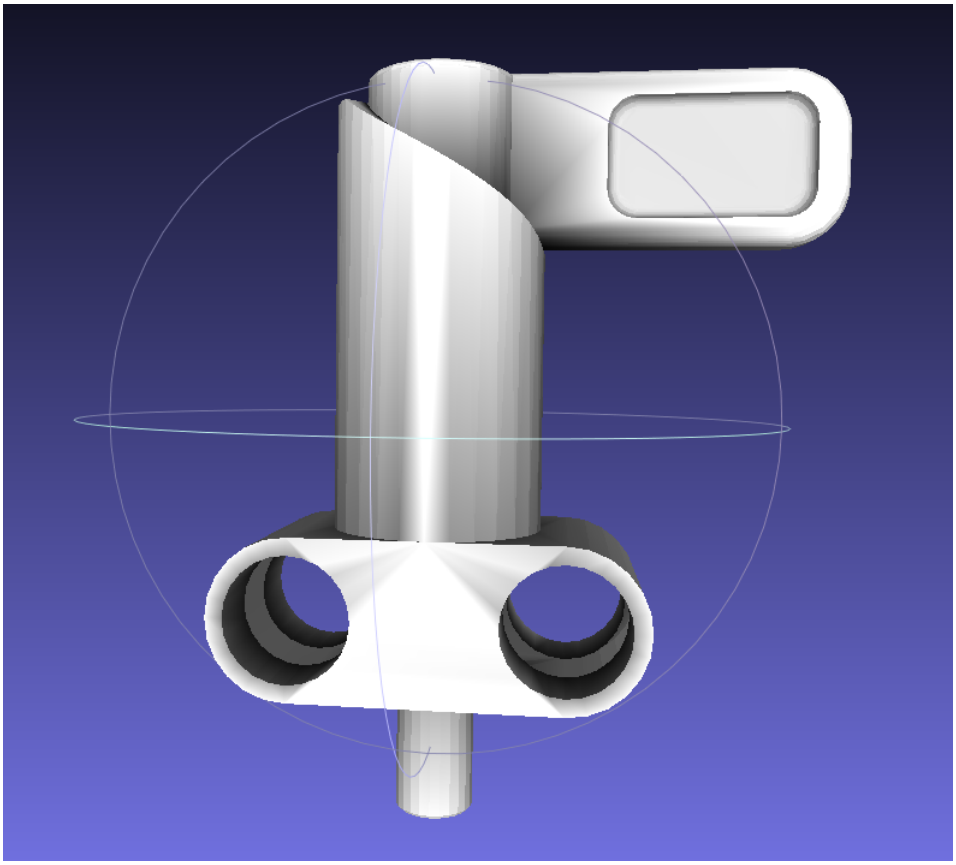


Abb. 25: MeshLab: Die bearbeitete STL hat nun kaum noch innere Strukturen und kann nun weiter manuell nachgearbeitet oder gespeichert werden. Da in diesem Beispiel ein hoher Schwellwert gewählt wurde, sind die Ringe innerhalb der unteren Löcher ebenfalls entfernt worden.

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. 26: Datamatrix Code für den Artikel ‚12345678'



Abb. 27: Datamatrix Code für den Artikel ‚98765432'



Abb. 28: 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.

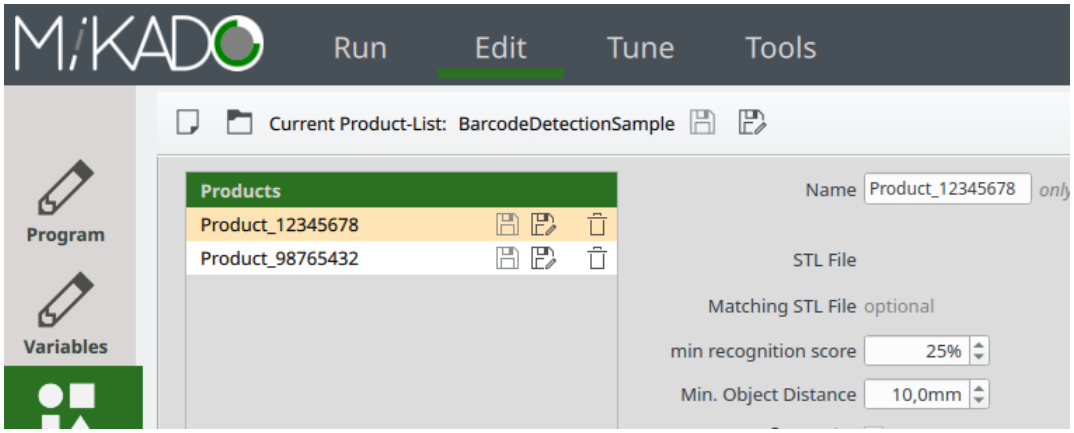


Abb. 29: Beispielhafte Produktliste

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.

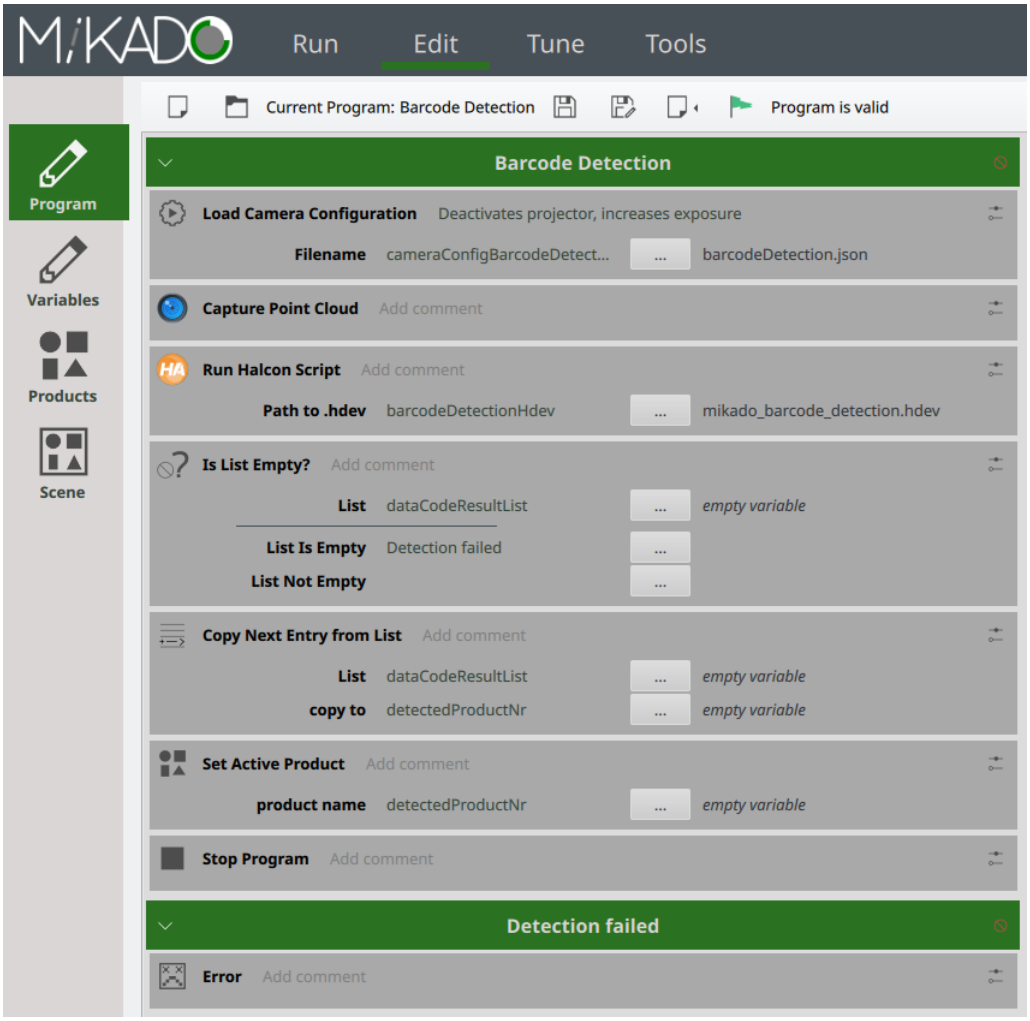


Abb. 30: Programm-Editor: Programmablauf für die Einbindung eines Hdev-Scripts

- *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 natür-

lich 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)
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)
```

(continues on next page)

(continued from previous page)

```
set_color(WindowHandle, 'orange')
disp_xld(qrXLDs, WindowHandle)
dump_window_image(OutputImage, WindowHandle)
mikado_publish_2d_image(OutputImage)
```

Im Folgenden Screenshot nochmals das gesamte Script als Screenshot aus der HDevelop Umgebung.

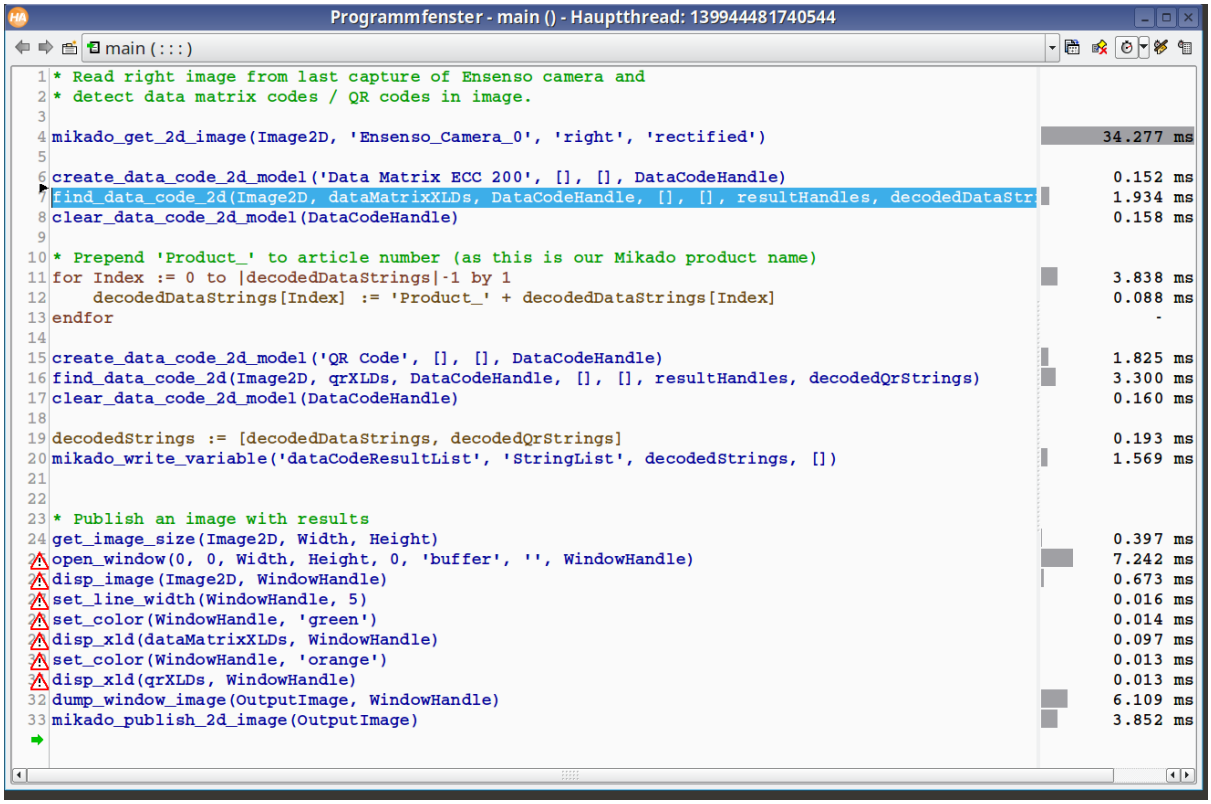


Abb. 31: Hdev-Script ‚mikado_barcode_detection.hdev‘

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.

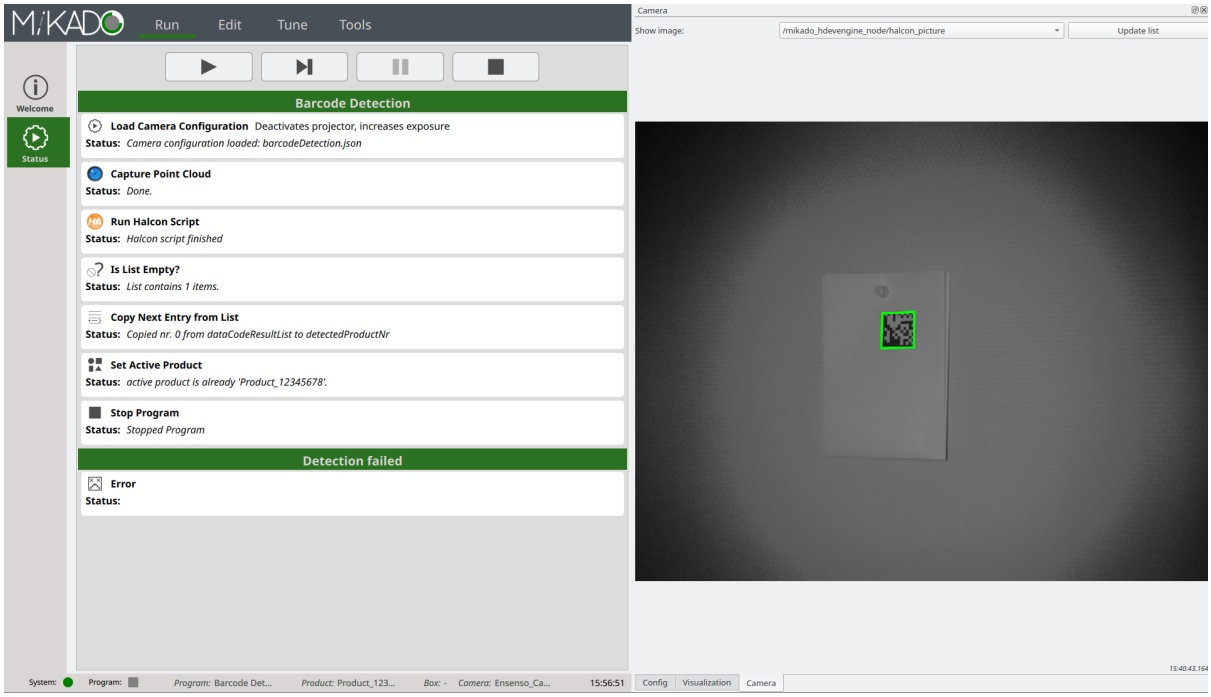


Abb. 32: Statusansicht bei Verwendung der Datamatrix des Artikels ‚12345678'

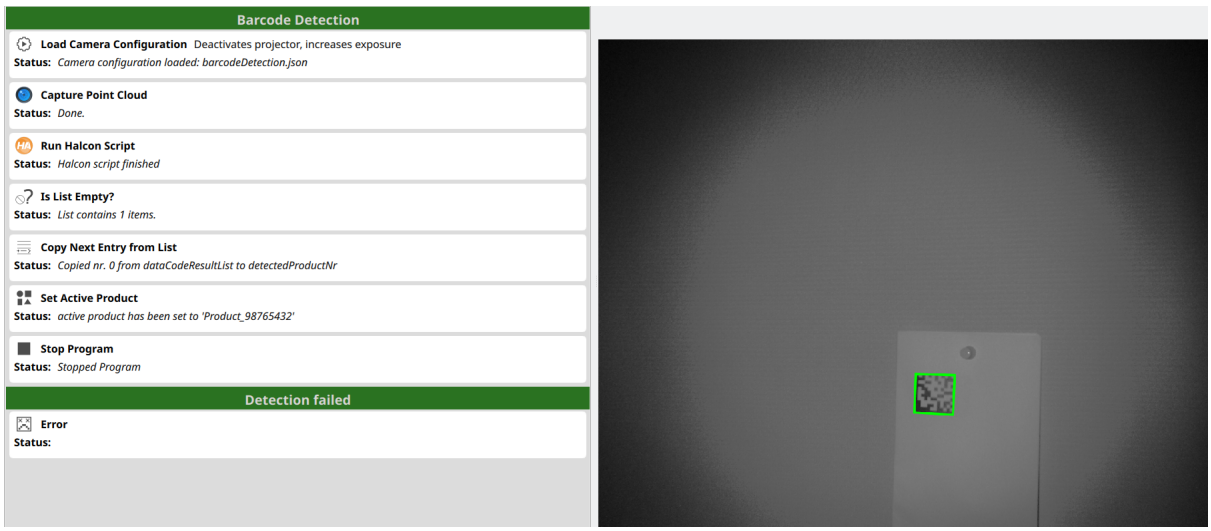


Abb. 33: Statusansicht bei Verwendung der Datamatrix des Artikels ‚98765432'

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.

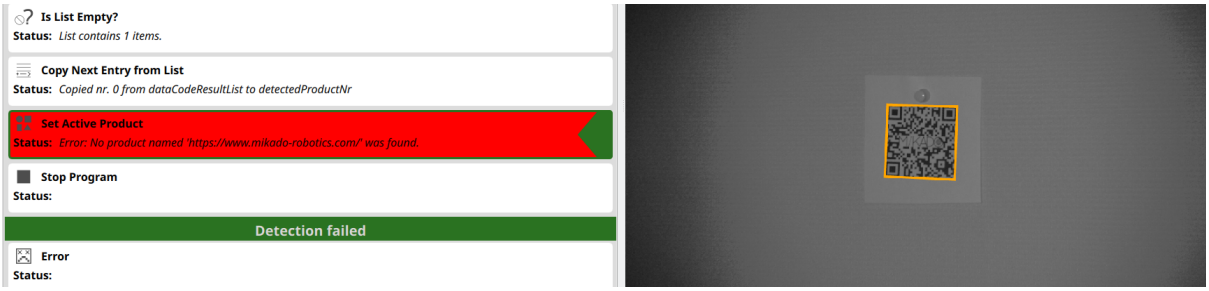


Abb. 34: Statusansicht bei Verwendung des Mikado QR Codes

4.5 Javascript: Benutzerdefinierter Dateilog 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 in Erweiterungs-Kapitel *JavaScript Task* beschrieben ist.

Das Gesamtskript loggt zunächst eine Überschriftszeile und dann jeweils Datenzeilen. Dabei wird in jedem Programmdurchlauf einmal geloggt und an eine die Logdatei angehängt. Die Logdatei lässt sich dann zum Beispiel mit herkömmlichen Tabelleneditoren importieren. Die einzelnen Skript Bausteine werden nachfolgend erklärt.

```
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("objektListe")
6
7  // Konstanten, .csv Spalten Seperator, Zielpfad
8  var sep = ";"
9  ausgabeDateiFunde = ausgabeDateiFunde + testlauf + ".csv"
10
11
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"
23
24
25 //Ausgabe in Datei
26 var append = true //überschreibt Datei, falls false
27 lab.writeStringToFile(ausgabeDateiFunde, ausgabeString,␣
    ↪append)
28
29 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 die das Skript basiert:

- *nameTestlauf*: Bezeichner dieses Durchlaufs, auch als Name der Zieldatei genutzt
- *ausgabeOrdner*: Absolutpfad zum Augabe Ordner, endet mit „/“, zB. „/home/mikado/“
- *durchlaufN*: Zähler der Logging Iteration
- *maxN*: Zähler Maximum, nach dem die Logging Iteration abgeschlossen werden soll
- *fundPosen*: 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.

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)

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 „FileSERIAL“. Ergänzen Sie „camera_serial_no:=FileSERIAL“ als Argument oder in der .bash Datei für Ihre Zelle. Die Filecam wird nun an Stelle der echten Kamera verwendet. Die Bilder werden bei jeder Aufnahme durchrotiert. Wenn die Bilder direkt in Mikado abgespeichert wurden wird die Punktwolke auch an der richtigen Stelle im Raum angezeigt werden. Stammt die Filecam aus nxView funktioniert dies nur bei kalibrierten Stationären Kameras. In einer zukünftigen Version wird eine nachträgliche manuelle „Hand-Auge-Kalibrierung“ möglich sein.

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 in der BETA-Phase 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 in der BETA-Phase geladen:

- Kistenobjekte mit Dropzone
- Produktobjekte mit individuellen Produkteigenschaften

Im jeweiligen Zellenordner können in der launch/simulator.launch in der BETA-Phase 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.

Kiste Setup

In der Beta-Phase 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 in der BETA-Phase 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.

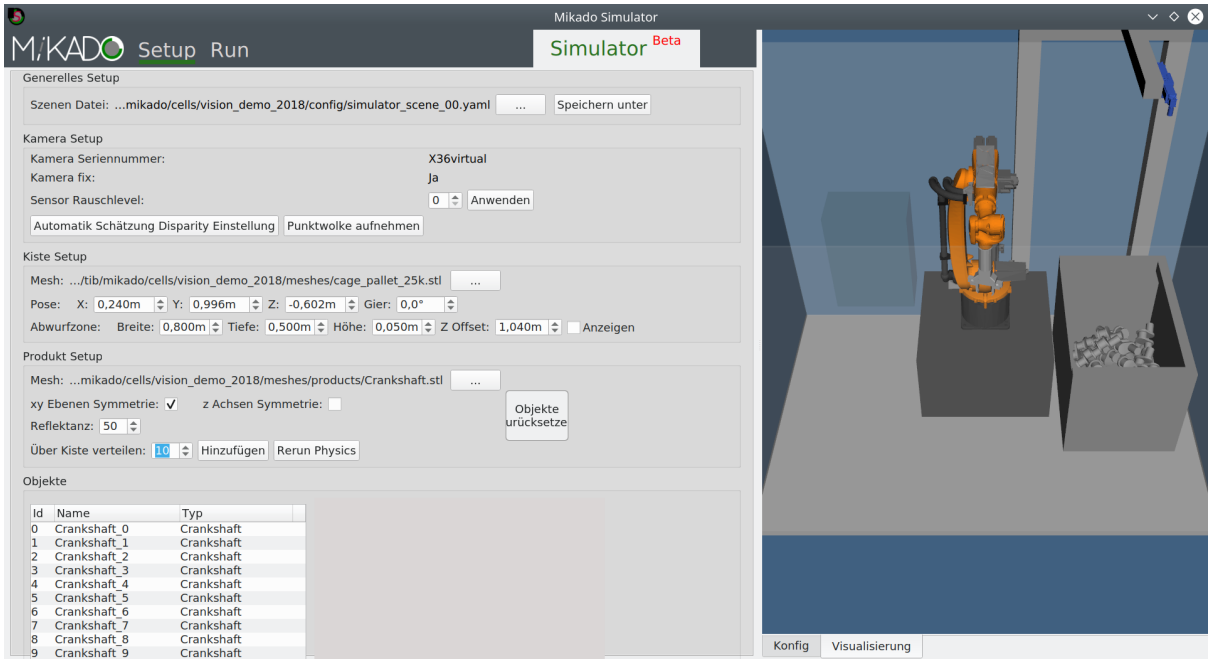


Abb. 1: Screenshot aus dem Mikado Simulator

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.

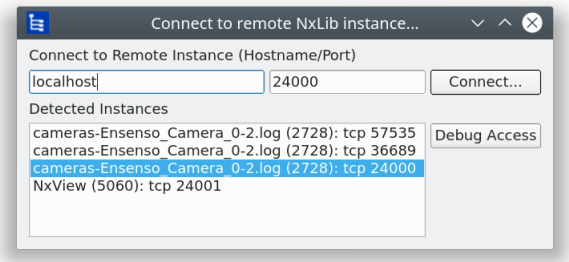


Abb. 2: Startfenster NxTree

3. Öffnen NxView: Im Anschluss muss die einzustellende Kamera in NxView geöffnet werden.

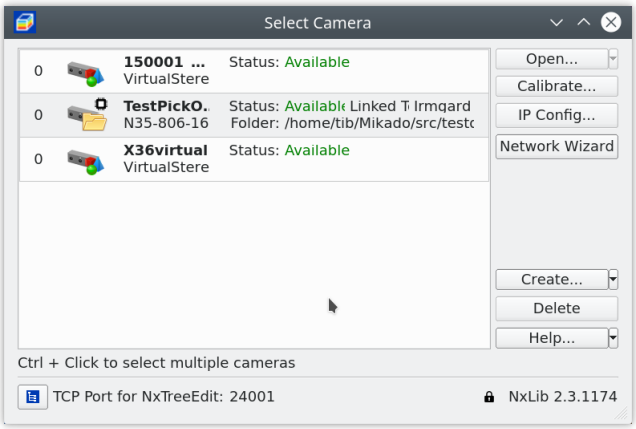


Abb. 3: Auswahl der einzustellenden Kamera in NxView

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.

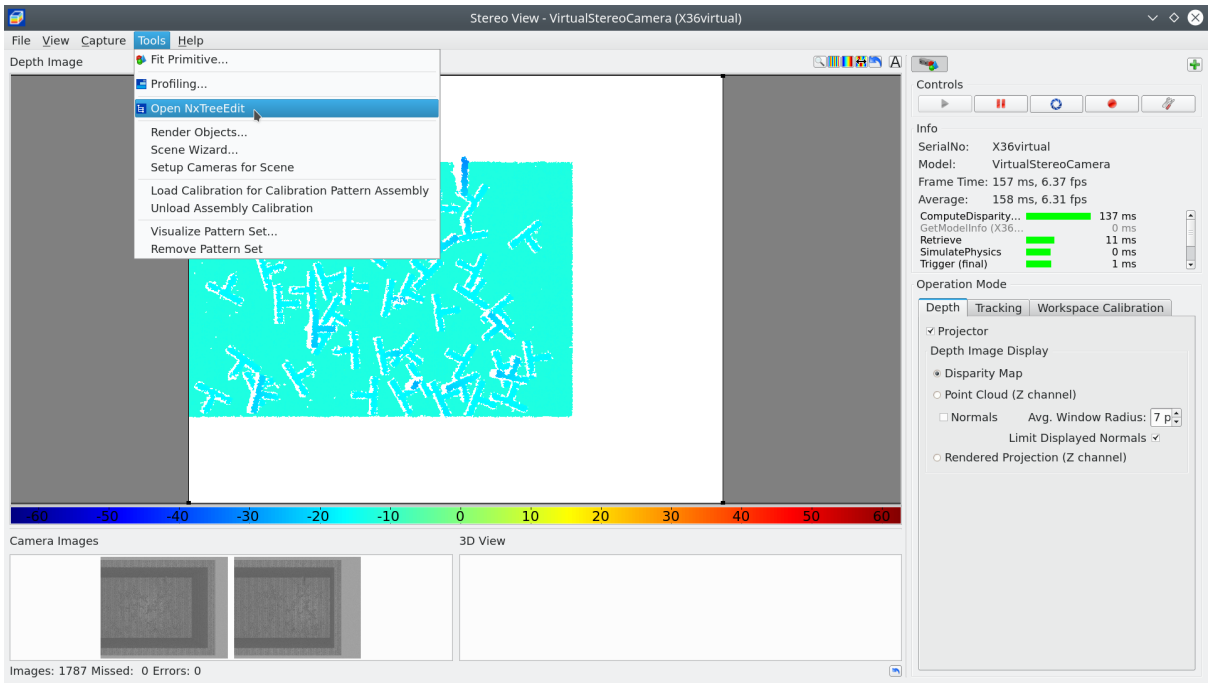
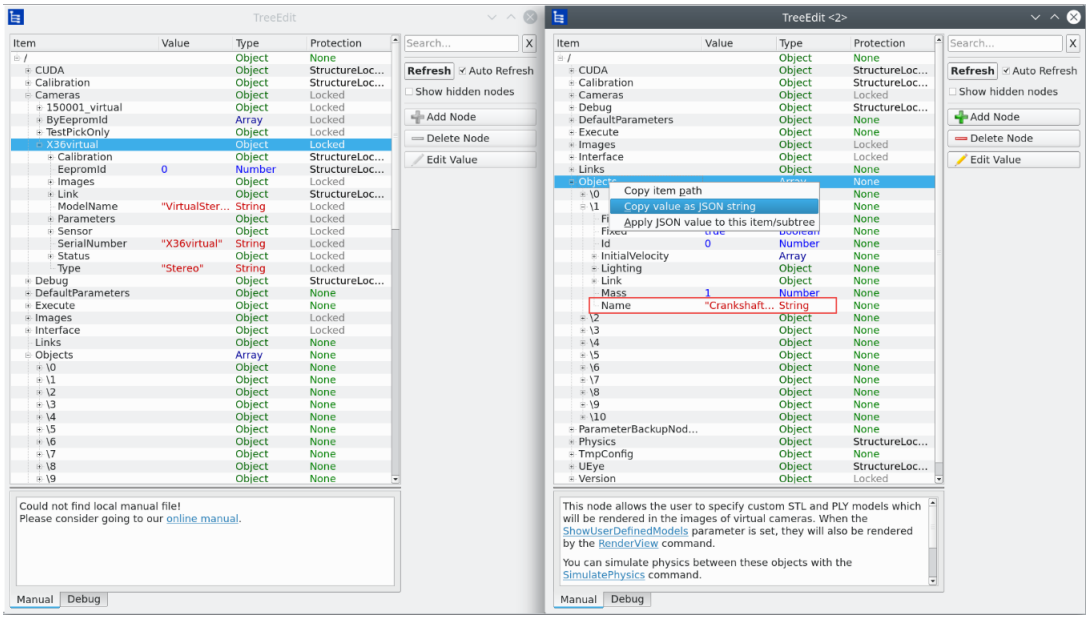
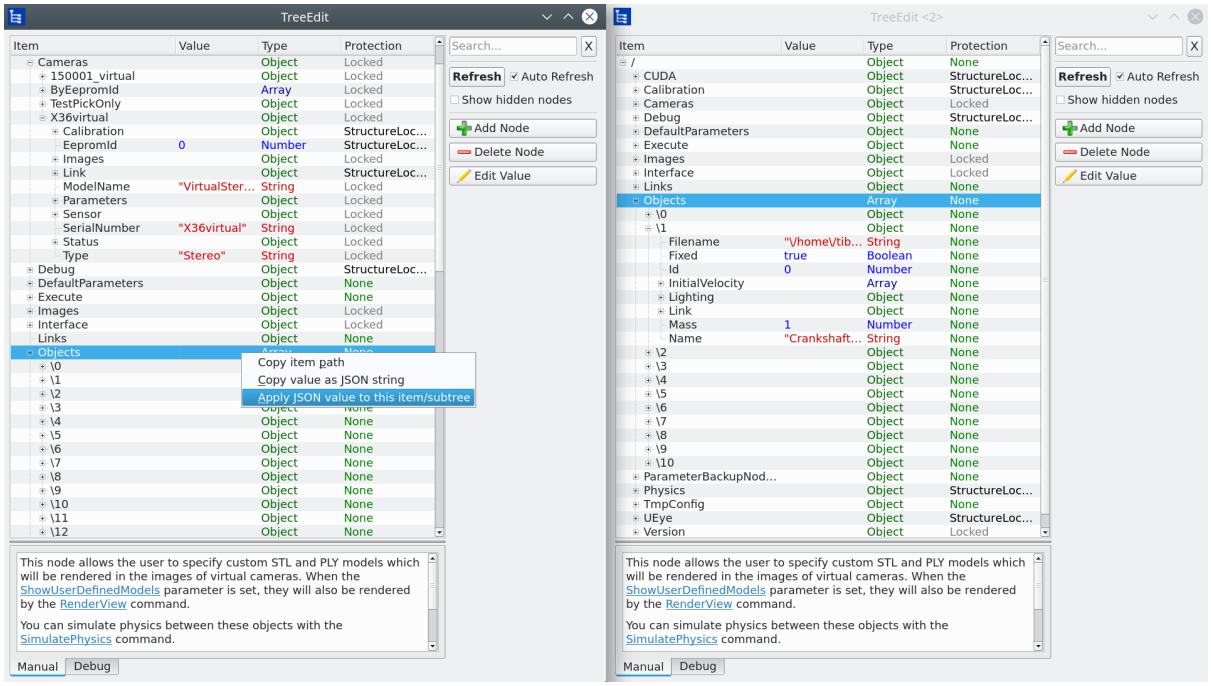


Abb. 4: Screenshot NxView

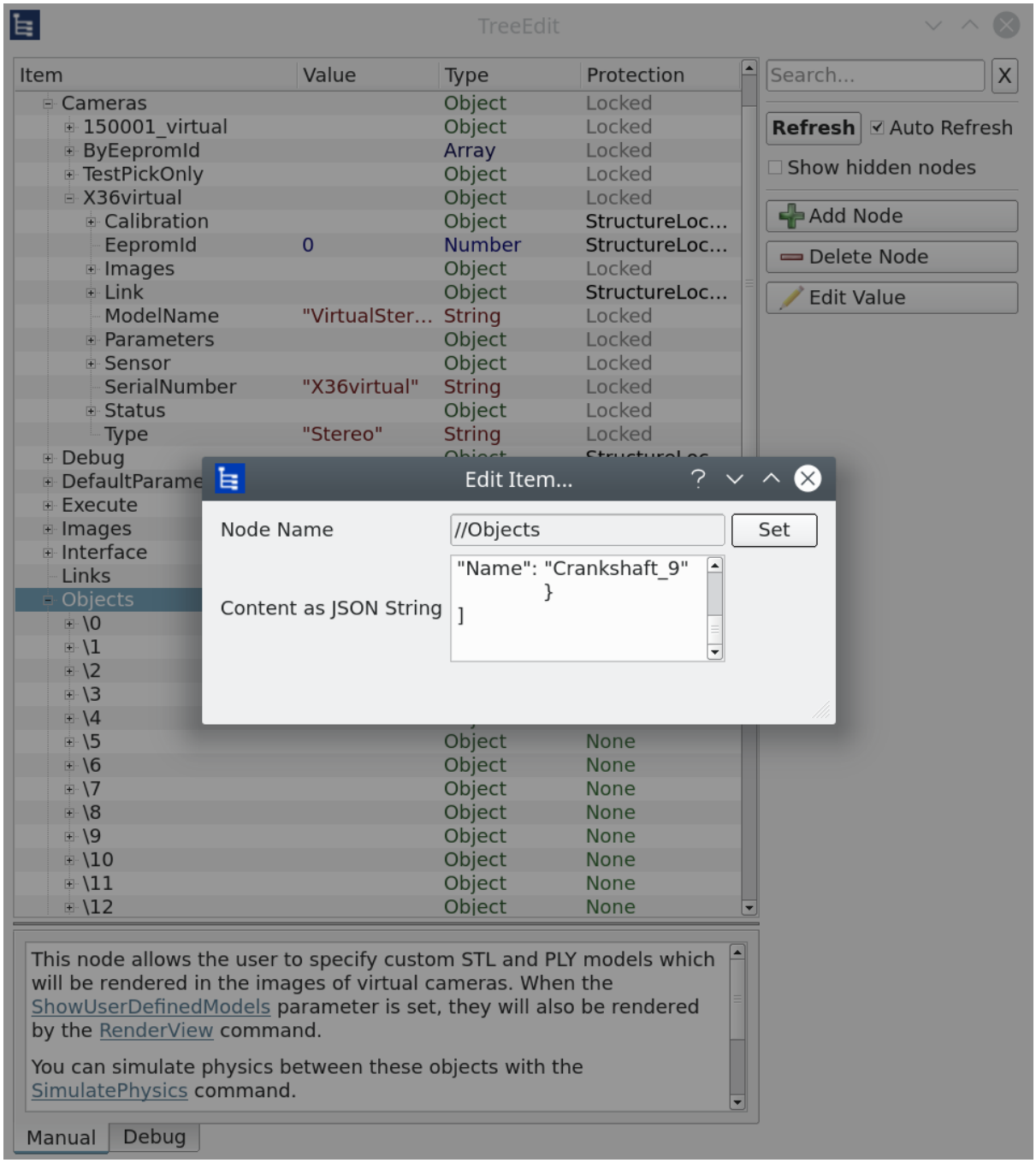
5. Kopieren der Elemente: In der folgenden Abbildungen ist NxTree der Szene in NxView (links) und der Szene im Simulator (rechts) zu sehen. Falls die Orientierung verloren geht, zu welcher Szene welches Fenster gehört, können beispielsweise die Objekte überprüft werden. In der Szene des Simulators (rechts) 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.



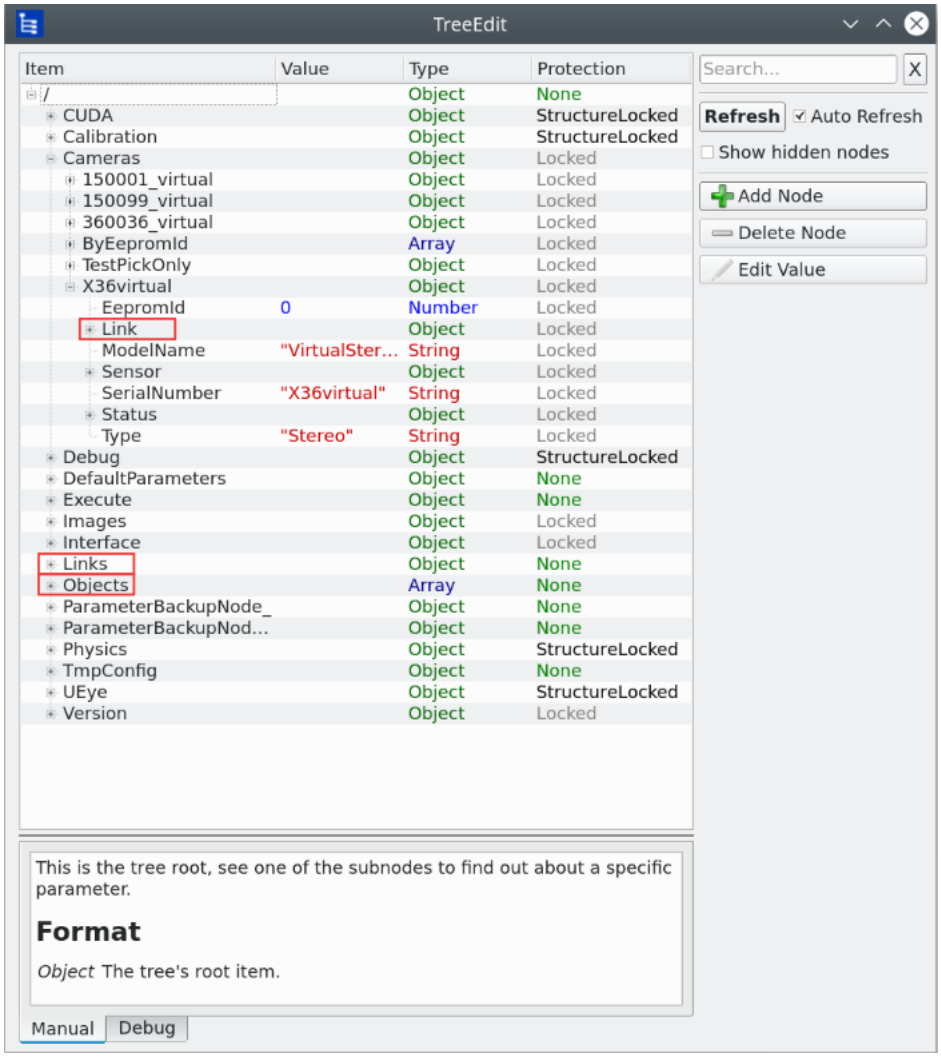
Die Objekte müssen anschließend in die zu NxView zugehörige Szene eingefügt werden (linkes Fenster). Hierfür muss mit der rechten Maustaste auf **Objects** geklickt und **Apply JSON value to this item/subtree** ausgewählt werden.



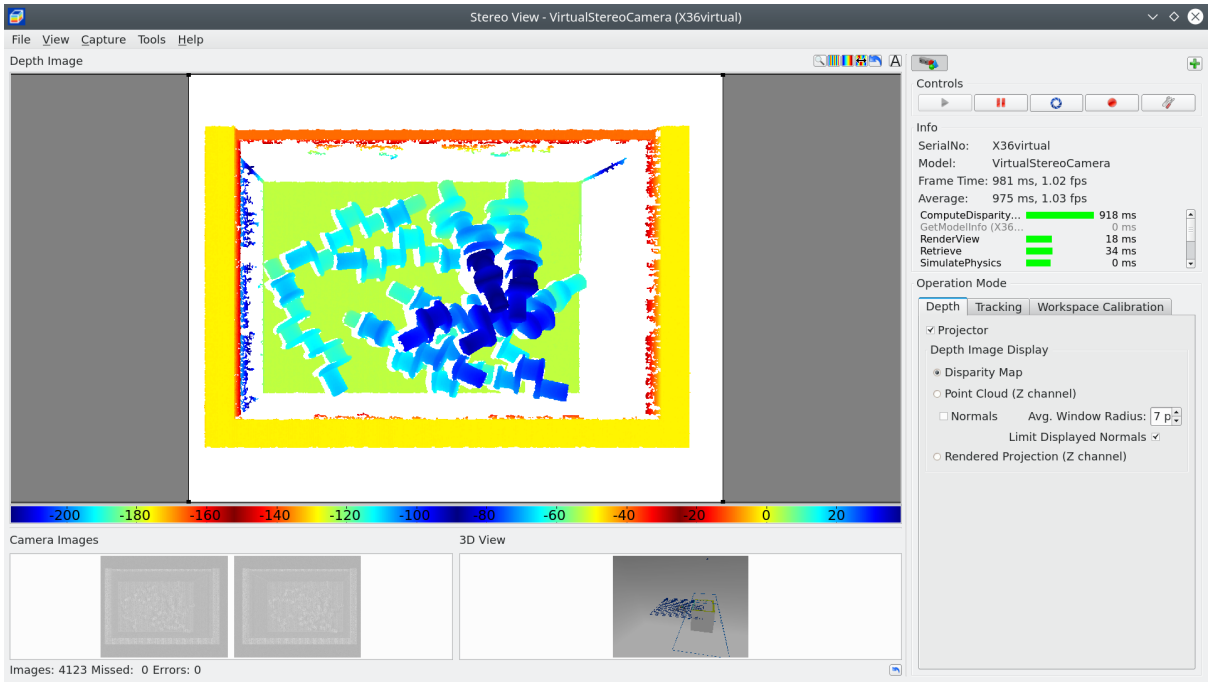
Im sich daraufhin öffnenden Fenster muss das Feld **Content as JSON String** befüllt werden. Hierfür auf das Feld mit der rechten Maustaste klicken und dann auf **Einfügen**. Anschließend ist der Inhalt im Feld zu sehen.



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

Unterschiedliche Roboter haben unterschiedliche Eigenschaften. In diesem Kapitel wird erläutert, welche Möglichkeiten die verschiedenen Roboter-Modelle und -Hersteller bieten. Parameter oder Steuerelemente, die modellabhängig sind und über die Oberfläche dargestellt werden, sind auf der Seite *Robotersteuerung* eingebunden.

Möchten Sie die Controller der Roboter einrichten, finden Sie entsprechende Informationen und Anleitungen im Kapitel *Roboter-Controller einrichten*.

6.1 Mitsubishi

6.1.1 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 ARC 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 ARC 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.

6.1.2 Roboter einschalten

- Mikado und Roboter können gleichzeitig gestartet werden.
- Teach Pendant muss deaktiviert sein und der Controller muss im Modus „Automatic“ operieren

6.1.3 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.

Hinweis: Die Konfiguration ist in Kapitel ‚*Mitsubishi Controller einrichten*‘ beschrieben.

6.2 Universal Robots

6.2.1 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.

6.2.2 Roboter einschalten

- Mikado und Roboter können gleichzeitig gestartet werden. Der Roboter muss zuerst bestromt und freigegeben werden.
- UR Polyscope: grüner Kreis oben links anklicken.
- Ein/Start drücken.

6.2.3 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.

Hinweis: Die Konfiguration ist in Kapitel ‚*Universal Robots Controller einrichten*‘ beschrieben.

6.3 Weitere Roboter

Die Dokumentation für die Verwendung weiterer Roboterhersteller aus Anwendersicht ist noch nicht ganz fertig gestellt, bitte kontaktieren Sie uns für eine Vorabversion.

KAPITEL 7

Installation

In diesem Kapitel wird erläutert wie sie Ihre Roboterzelle in Mikado ARC 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

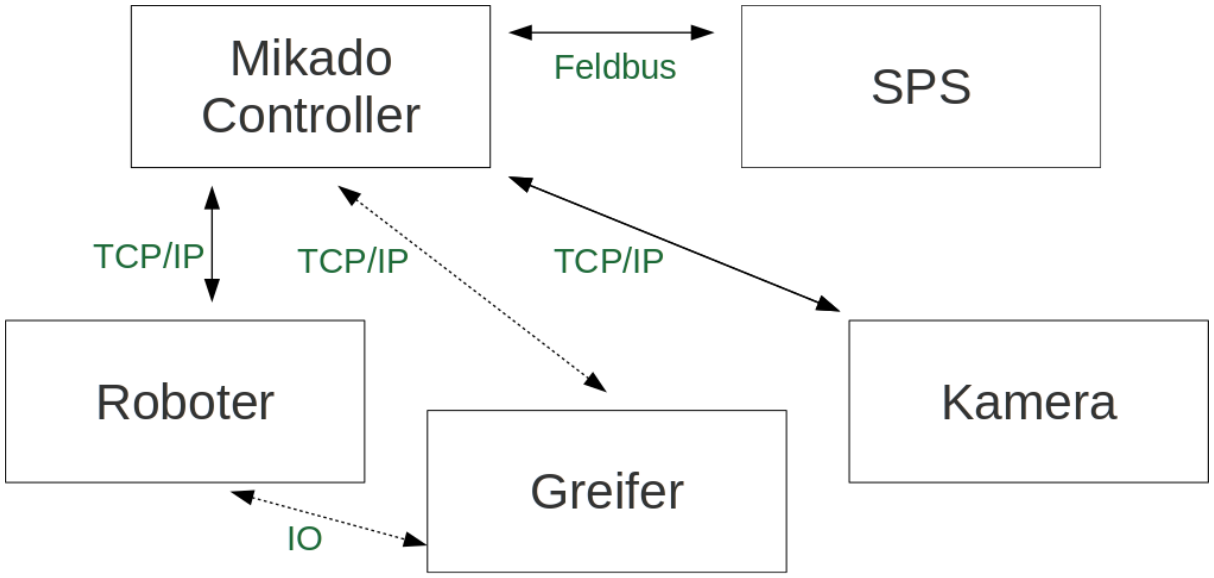


Abb. 1: Übersicht der Hardware-Komponenten mit Schnittstellen

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 ARC Erweitern*)
  - Eigene Halcon Scripts
  - Adapter für Feldbus-Anbindung
  - Weitere Möglichkeiten wie JavaScript und Python
- Ensenso NxView (Offizielles Handbuch)
  - 3D Kamera konfigurieren und testen
  - Filekameras erstellen
  - Konfiguration für Mikado exportieren

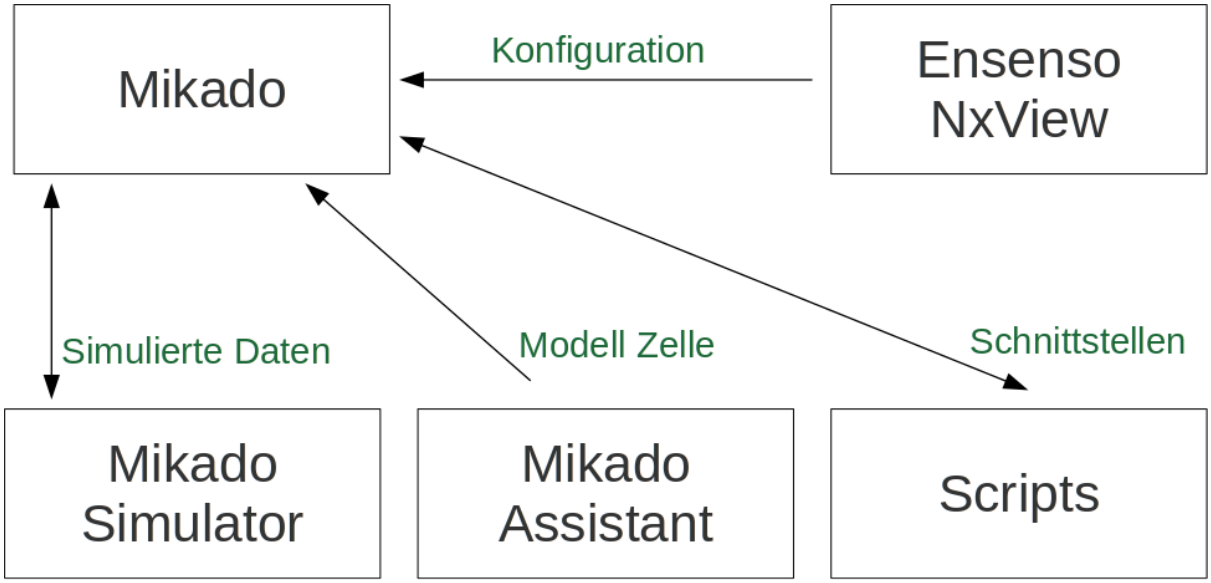


Abb. 2: Übersicht der Software-Komponenten mit Schnittstellen

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* + *URDF*
3	Greifer einbinden	*Mikado Assistant* + *URDF* + *CAD Greifer*
4	Roboter und Kamera einbinden	*Roboter-Controller* + *manipulator.yaml* + *camera_X.yaml*
5	Kamerakonfiguration(en) erstellen	Ensenso NxView
6	(Hand-Auge-) Kalibrierung	*calibration.yaml*, *Mikado Kalibrierung* (+ Ensenso Kalibrierplatte)
7	Offset Greifer ermitteln	Mikado (z.B. mithilfe der Kalibrierplatte) + *URDF*
8	Szene, Produkt + Griffdefinitionen, Programmablauf	*Mikado GUI*
9	Optimierungen + Test	*Mikado GUI* (+ *Simulation*)
10	(Optional: Schnittstelle einrichten)	*Mikado ARC Erweitern*

7.2 Konfigurations-Backup

Ein minimales Backup auf Dateisystemebene umfasst den Ordner */home/mikado/mikado* und enthält alle Anlagen-spezifischen Einstellungen (der enthaltene Ordner */logs* ist nicht erforderlich).

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 ARC Zelle Erstellen

Klicken Sie nach Starten des Assistants auf *Neue ARC 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.

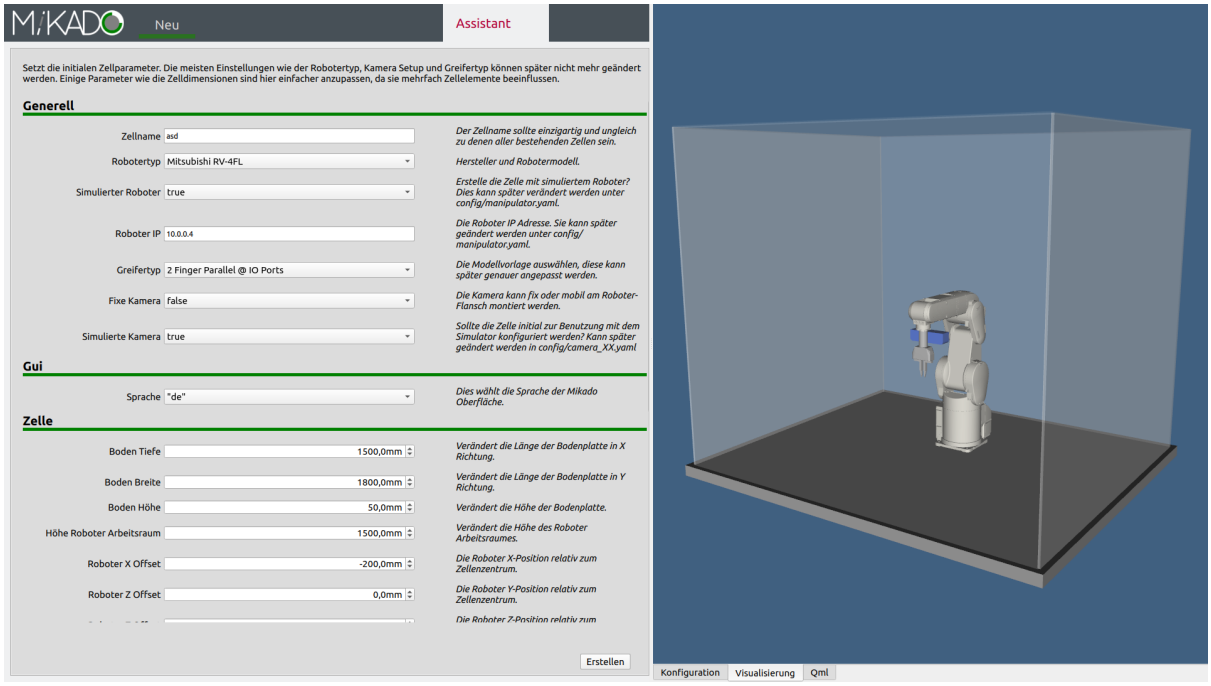


Abb. 3: Screenshot Assistant Übersicht Zellerstellung

7.4 Anpassen einer vorhandenen Zelle

Eine bereits existierende Zelle besitzt eine Vielzahl globaler Parameter, die sie konfigurierbar macht.

7.4.1 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. Offensichtlich muss man technisch nicht besonders versiert sein um dieses Passwort heraus zu finden. Es sollte also zum Einen kein „wertvolles“ Passwort verwendet werden und zum Anderen immer bedacht werden, dass es sich eher um einen Hinweis handelt als um wirkliche Sicherheit.

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, zum Beispiel `zellname:config/150001.json`. 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. `zellname:/config/4101230815.json`

Warnung: Filekameras, die in einem *.zip liegen, werden nicht unterstützt und müssen in einen Ordner entpackt werden.

gripper.yaml

Einige Parameter sind je nach Greifertyp optional und daher eventuell nicht Teil Ihrer Konfigurationsdatei.

- *type:* legt den Typ des Greifers fest. Dabei gibt es momentan die folgenden Kategorien:
  - *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 Balgensaugern) 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.
- *joint_finger_name:* Name des Gelenks des führenden Greiferfingers (z.B. ‚finger_left‘). Kann auf ‚‘ gesetzt werden, wenn Greifer keine beweglichen Gelenke hat.
- *joint_value_closed:*[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.
- *joint_value_open:*[1] Wert des Gelenks in Meter, wenn der Greifer offen ist (z.B. 0.007). Dieser Wert wird verwendet um z.B. elektrische Greifer mit Linearachsen anzusteuern.
- *open_hand_signal_number:* Signalnummer (IO-Nummer) mit dem der Greifer in den Offen-Zustand geschaltet werden kann
- *closed_signal_number:* Signalnummer (IO-Nummer) mit dem der Greifer in den Geschlossen-Zustand geschaltet werden kann
- *is_closed_signal_number:* Signalnummer (IO-Nummer) an dem der Geschlossen-Zustand des Greifers signalisiert wird

[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*.

- *is_open_signal_number:* Signalnummer (IO-Nummer) an dem der Offen-Zustand des Greifers signalisiert wird
- *is_closed_low_active:* Ist der verwendete Sensor for den Geschlossen-Zustand low-active, diesen Parameter auf *true* setzen
- *is_open_low_active:* Ist der verwendete Sensor for den Offen-Zustand low-active, diesen Parameter auf *true* setzen
- *finger_state_publish_rate:* Update-Rate des Finger- (oder Greifer-)Gelenks in Hz
- *wait_time_pick:* Wartezeit in Sekunden direkt nachdem der Greifer ein Teil gegriffen hat
- *wait_time_before_pick :* Wartezeit in Sekunden an der Übergangspose eines Greifvorgangs, falls der Greiferzustand für den Griff geändert werden muss
- *speed:* Geschwindigkeit der Greifer-Achse im m/s (z.B. für Linearachse)
- *acceleration:* Beschleunigung der Greifer-Achse im m/s^2 (z.B. für Linearachse)
- *force:* Greifkraft der Greifer-Achse im Newton (z.B. für Linearachse)

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

- *manufacturer:* z.B. „Universal Robots“ Hersteller des Roboters, schaltet ggf vorhandene Spezialfunktionen in Mikado ARC 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:

- fullscreen: true/false Vollbildmodus aktiv
- 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)
- language: „en“ Die Sprache der Oberfläche. Momentan „de“ oder „en“. Wird keine Sprache angegeben wird die Systemsprache verwendet.
- 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. Die Bedeutung der einzelnen Parameter ist im Kapitel *Objekterkennungs-Tool* näher erläutert. Die folgenden Parameter müssen eingestellt werden (angegebene Werte sind Beispielswerte):

- voxel_size: 2.0
- dilation_radius: 1
- dilation_radius_object_cutting: 5
- relative_sampling_distance: 0.06
- pose_ref_rel_sampling_distance: 0.03
- hv_SamplingDistance: 0.03
- hv_KeypointFraction: 0.3
- pose_ref_use_scene_normals: true
- check_pose_ref_num_steps: 1
- check_pose_ref_use_scene_normals: true
- check_pose_ref_dist_threshold_rel: 0.02

ompl_planning.yaml

Hier können Parameter für die verfügbaren Pfadplannungsalgorithmen angepasst werden. Mikado ARC verwendet momentan standardmässig „RRTConnect“ und dies ist auch nicht umschaltbar. Es ist unwahrscheinlich, dass Sie an diesen Einstellungen etwas ändern wollen.

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 beeinflusst werden
  - *score_bin_size:* Gruppierungsintervalle nach Score
  - *z_bin_size:* Gruppierungsintervalle nach z-Wert
- *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 Funde: Maximaler Abstand des vorherigen Fundes, damit dieser als derselbe Fund klassifiziert wird
- *object_recognition_max_rotation_diff:* Wiederverwenden von Funde: 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 das 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, so das sich dieser Modus z.B. gut für die Verwendung in einer Kiste eignet. Wird für *sensor_plugin* statt dessen *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 ARC 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
```

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 Geometrie/Kinematikbeschreibung in der URDF/XACRO angegeben werden können. Ausgehend vom generierten Beispiel sind hier vor allem 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>
```

Über Einträge in die *disable_collisions*-Tabelle können Sie festlegen, das Kollisionen zwischen zwei Körpern ignoriert werden. Dies beschleunigt das System z.B. bei stationären Gegenständen, kann aber auch hilfreich sein wenn z.B. zwei benachbarte und bündige Abschnitte des Roboters sich ständig oder zeitweise minimal überlappen

Warnung: Die Kollisionsprüfung zwischen den hier aufgeführten Teilen ist komplett deaktiviert. Falls die beiden Teile in der Realität kollidieren können, wird sich irgendwann sicher eine Situation ergeben wo sie dies auch tun. Inklusive dem damit verbundenen Materialschaden. Stellen Sie also z.B. über maximale Achswinkel (siehe URDF) etc sicher das deaktivierte Kollisionen auch wirklich 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 egal wie der Roboter verfahren wird 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.

7.4.2 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 das Tool zur Einrichtung verwendet haben, sollten eigentlich keine Änderungen an den Launchfiles nötig sein, außer Sie wollen den Typ des Roboters ändern oder zusätzliche Software wie z.B. einen Feldbuskopler einrichten. 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 ARC 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 grupieren, 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 geplannt oder durch einen Absturz beenden, kann es per *respawn=“true“* erneut gestarted 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.

7.4.3 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. Auf dem Mikado Kontroller ist

zu diesem Zweck *kate* installiert. Es gibt auch eine Live-Vorschau die später erläutert wird. Die URDF Dateien sind im Unterverzeichnis *urdf* des Projektverzeichnises abgelegt. Öffnet man den Ordner in Dolphin (Dateimanager) fällt auf das dort nur Dateien mit der Endung *.xacro* liegen. XACRO steht für Xml Macro und ist ein Hilfsmittel um flexible und knappe XML erstellen zu können die modular sind und sich nicht ständig wiederholen. Im folgendem wird zuerst kurz URDF an sich erläutert, anschließend werden die Möglichkeiten gezeigt dieses mit XACRO zu strukturieren und zu guter Letzt wird die von uns empfohlene Organisation der Dateien erläutert.

URDF

Die Unified Robot Description Format (URDF) Beschreibung einer Roboterzelle beginnt mit einem umschließenden <robot> Tag. Dieser hat ein Attribut *name* das mit dem Tag im zugehörigen 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 star 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!

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 also aus einem CAD/Mesh-Datei die die visuelle Erscheinung beschreibt und in Cyan/Türkis (der Name ist frei wählbar, entscheidend sind die RGBA Werte)

eingefärbt wurde. Meshdatein müssen entweder als STL (einfache Dreiecke, keine Farbinformationen) oder als Collada (.dae, Texturen etc) und in Metern vorliegen. Die visuelle Geometrie kann theoretisch über <origin xyz="0 0 0" rpy="0 0 0" /> verschoben werden, allerdings wird dies leicht verwirrend und es 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.

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.

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 isys vision 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 ARC nicht sicherheitsrelevant ist. Dabei ist darauf zu achten, die Grenzen im Controller ca 1° größer zu setzen als in Mikado ARC. (Bei UR 3°). Versucht Mikado ARC 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 ARC 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 weitere mögliche Tags und Gelenktypen die dort erwähnt werden nicht offiziell von Mikado ARC unterstützt.

XACRO

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}" />
```

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“ in „Persönlicher 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 Pfadplannung
- 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 diese Datei 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 Greifer-Integration

In diesem Kapitel der Greifer-Integration wird aufgezeigt, welche Möglichkeiten Mikado ARC zur Einbindung Ihres Greifer bietet. Erläutert werden unterstützte Greifertypen und wie Sie das CAD-Modell des Greifers einbinden und konfigurieren können.

7.6.1 Greifertypen

Mikado ARC unterstützt mehrere Greifertypen wie z.B. Parallel-, Vakuum-, Magnet oder Elektro-Greifer. Eine Auflistung und Beschreibung der Greiferkategorien ist in Sektion *gripper.yaml* aufgeführt.

7.6.2 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.

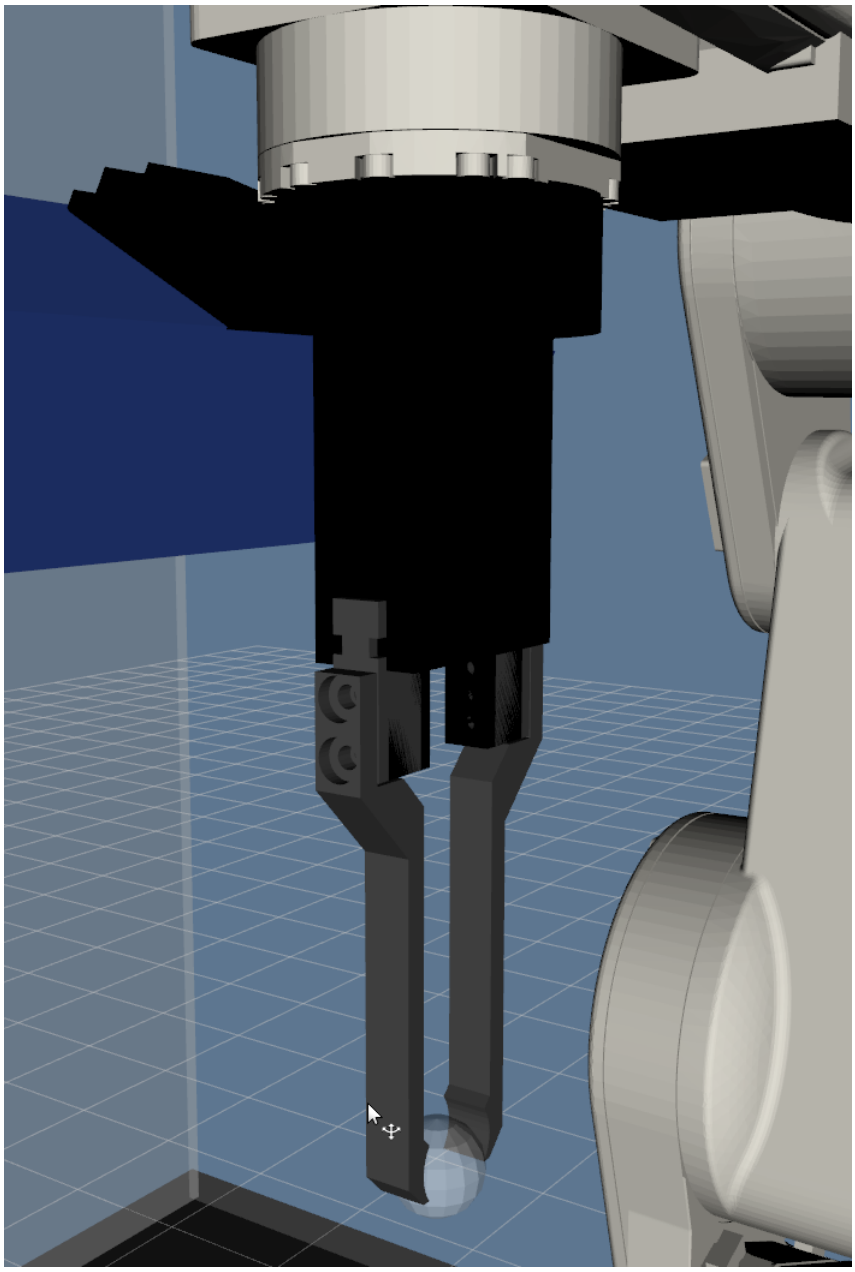


Abb. 4: Integrierter Greifer in der Mikado-Visualisierung (Beispiel)

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 ARC 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 Aufführungszeiten. 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 ein 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 *mesh/* 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:

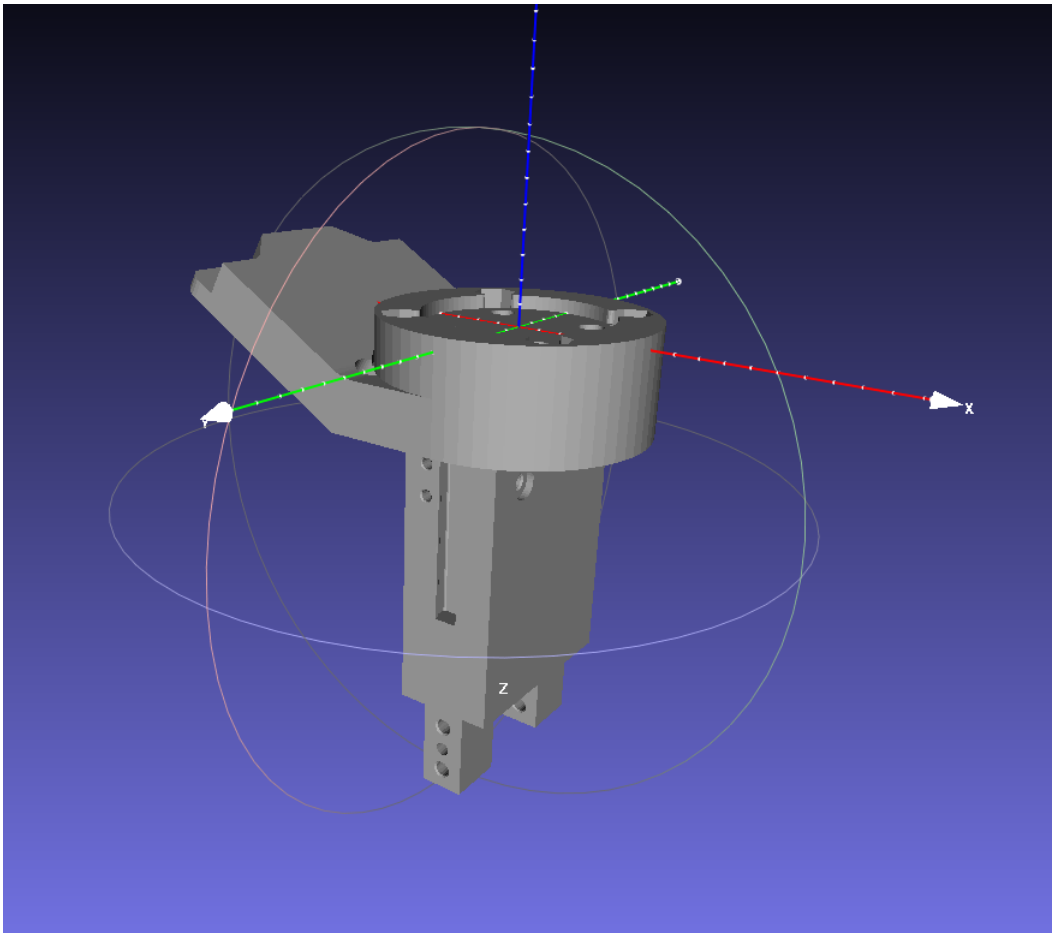


Abb. 5: Basisgeometrie des Beispielgreifers (oben schwarz dargestellt)

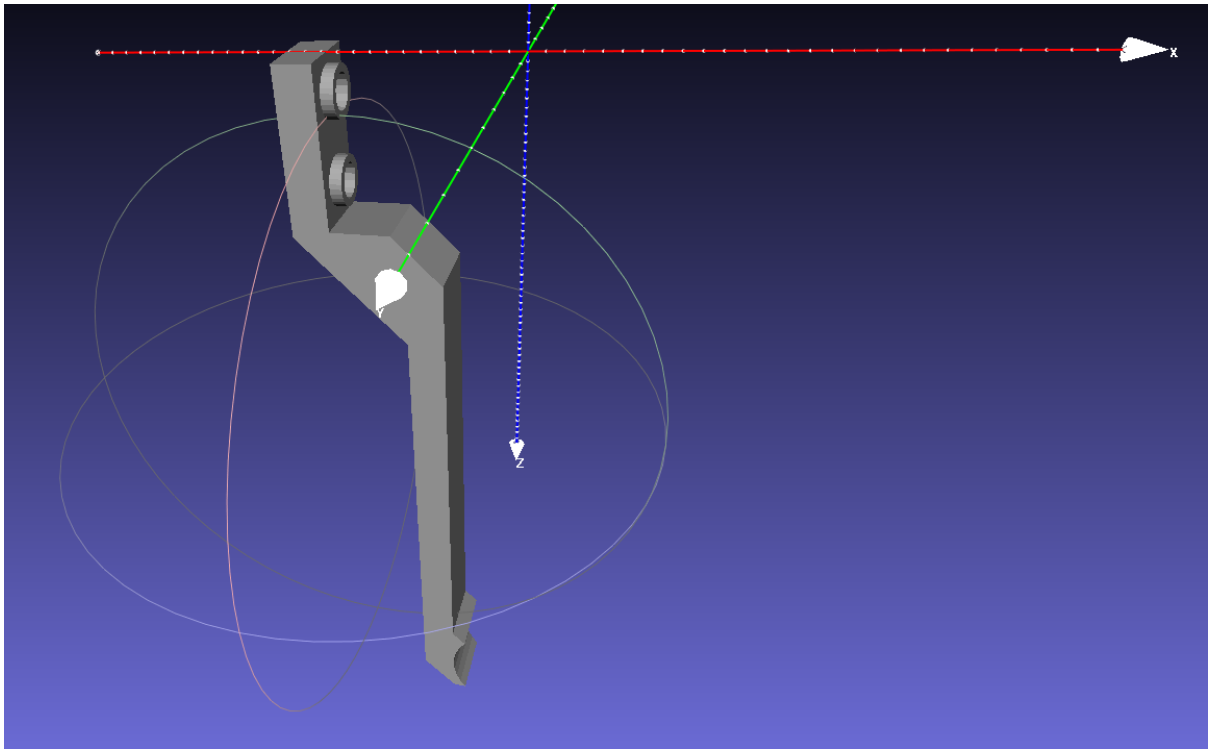


Abb. 6: Fingergeometrie des Beispielgreifers (oben grau dargestellt): Koordinaten-Ursprung liegt auf der z-Achse des Roboter-Werkzeugflanschs. Dies erleichtert die Positionierung.

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 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 (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.

7.6.3 Mikado-Konfiguration

Damit Ihr Greifer korrekt angesteuert wird, definieren Sie Eigenschaften eines Greifers in der Konfigurationsdatei, die in Sektion *gripper.yaml* erläutert wird.

7.6.4 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 jedoch 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" />
```

7.6.5 TCP kalibrieren

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 ARC-Oberfläche nachdem Sie Ihren Greifer eingerichtet haben
- Legen Sie die Kalibrierplatte vor den Roboter ins Sichtfeld der Kamera bei der Aufnahmepose
- Erstellen Sie folgendes Mikado-Programm

  - Bewege Roboter zu einer geeigneten Aufnahmepose
  - Patternpose bestimmen
  - Patternpose drehen (180° drehen)
  - 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 verwendeten Abstand zur Oberfläche der Platte aufweisen (z.B. 10 mm)
- Nun können die Abweichungen in die Positionierung Ihres Greifers einfließen und die Abweichungen können in die URDF des Greifers eingetragen / verrechnet werden.
- Wiederholen Sie den Vorgang um sicherzustellen, dass die neuen Offset Werte zu einer Verbesserung der Positioniergenauigkeit geführt haben

7.7 Hardware einrichten

In diesem Bereich sind Anleitungen aufgeführt, um spezifische Hardware mit Mikado ARC zu verbinden.

7.7.1 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:

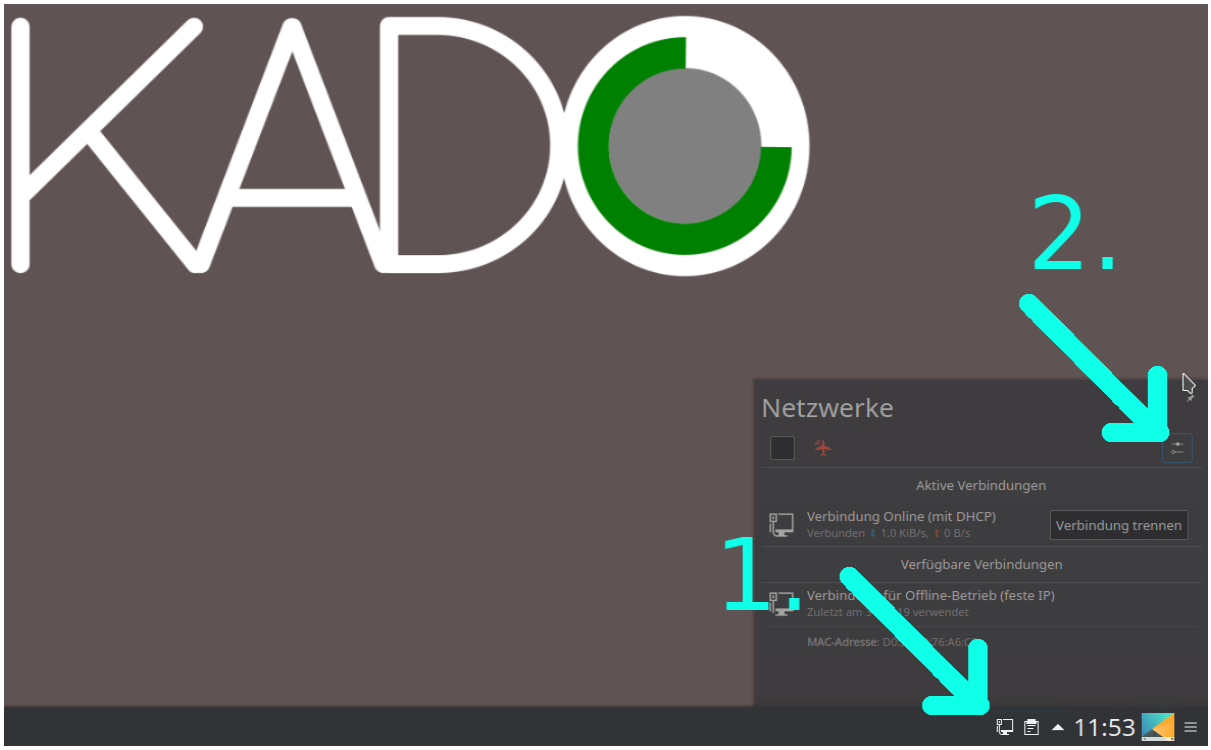


Abb. 7: Screenshot Netzwerkeinstellungen öffnen

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*

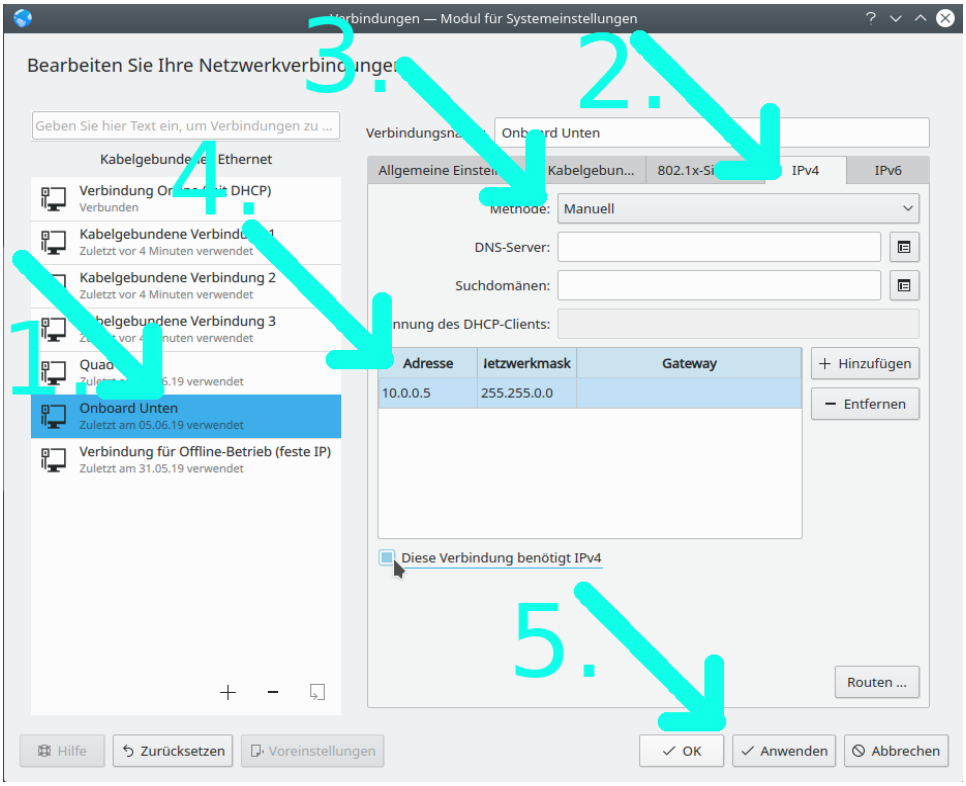


Abb. 8: Screenshot Netzwerkeinstellungen feste IP konfigurieren

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.

7.7.2 Ensenso

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 entweder per Network Wizard in NxView geschehen oder man startet in der Konsole per `sudo idscameramanager` ein eigenständiges Tool. Sieht man die Kamera auch nach wenigen Sekunden nicht, so muss unter „Daemon Control“ für den verwendeten Netzwerkadapter der „ETH-ConfigurationMode“

angeschaltet werden. Die so gefundene Kamera kann per „Manual ETH configuration“ mit einer festen IP im gleichen Adressraum versehen werden.

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

Die Kamerakonfiguration ist in den Schulungsunterlagen genauer dokumentiert. Die Hersteller-Bedienungsanleitung für die Kamera finden Sie unter ensenso.com.

Hand-Auge Kalibrierung

Ein Start der Hand-Auge-Kalibrierung aus Bedienersicht ist im Kapitel *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.

7.7.3 Roboter-Controller einrichten

In dieser Sektion ist aufgeführt wie die Robotercontroller der einzelnen Hersteller zu konfigurieren sind, damit diese problemlos mit Mikado kommunizieren. Die herstellerabhängigen Bedienungseigenschaften und weitere Informationen finden Sie in Kapitel *Roboter-Modelle*.

ABB Controller einrichten

Benötigte Optionen

Der roboterseitige Teil der ABB-Anbindung wurde in RAPID geschrieben und verwendet sowohl parallel laufende Tasks als auch Sockets. Es werden daher die folgenden Optionen benötigt:

623-1: Multitasking

672-1: Socket Messaging (in neueren RobotWare-Versionen ist dies Teil von 616-1: PC Interface)

Es wird mindestens RobotWare OS version 5.13 benötigt für einige Socket-Optionen. Zum Installieren empfiehlt es sich RobotStudio zu verwenden, es ist in einer kostenlosen Basic Variante erhältlich und wird in der folgenden Anleitung vorrausgesetzt.

Installation der Programme

Sie finden die RAPID-Programme unter

/opt/isys-vision/mikado/robot_programms/abb/rapid

Legen Sie auf dem Controller einen neuen Ordner namens ROS an und kopieren Sie alle Dateien aus dem rapid Ordner dort hin.

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

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

Diese müssen nun mit den Systemausgängen verbunden werden, gehen Sie dazu auf „I/O System“ -> „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

Fortsetzung auf der nächsten Seite

Tab. 3 – Fortsetzung der vorherigen Seite

Signal Name	Status	Arg 1	Arg 2
signalRobotEStop	Emergency Stop	N/A	N/A
signalRobotNotMoving	Mechanical Unit Not Moving	ROB_1	N/A
signalRosMotionTaskExecuting	Task Executing	N/A	T_ROB1

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
HOME:/ROS/ROS_common.sys	•	NO	YES	NO
HOME:/ROS/ROS_socket.sys	•	NO	YES	NO
HOME:/ROS/ROS_messages.sys	•	NO	YES	NO
HOME:/ROS/ROS_stateServer.mod	ROS_StateServer	NO	NO	NO
HOME:/ROS/ROS_motionServer.mod	ROS_MotionServer	NO	NO	NO
HOME:/ROS/ROS_motion.mod	T_ROB1	NO	NO	NO

Nach einem Neustart, idealerweise P-Start, des Systems kann der Roboter nun von Mikado gesteuert werden.

Greifer und IOs einrichten

Für das Ansteuern einfacher Greifer sowie für IOs im Allgemeinen wird ABBs Robot Web Service (RWS) verwendet. Tragen Sie in der Datei *config/manipulator.yaml* ihrer Zelle die folgenden Daten ein:

```
rws_username: Default User
rws_passwort: robotics
rws_port: 80
rws_digital_in_path: din
rws_digital_out_path: dout
```

Wir empfehlen Benutzername und Passwort auf dem Controller zu ändern!

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 *din23* ausgelesen.

Einfache Greifer können nun über die Datei *config/gripper.yaml* eingerichtet werden.

Mitsubishi Controller einrichten

Getestet mit Controller CR75x-D, Software Version S7 17-02-28.

Verbindung herstellen

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.

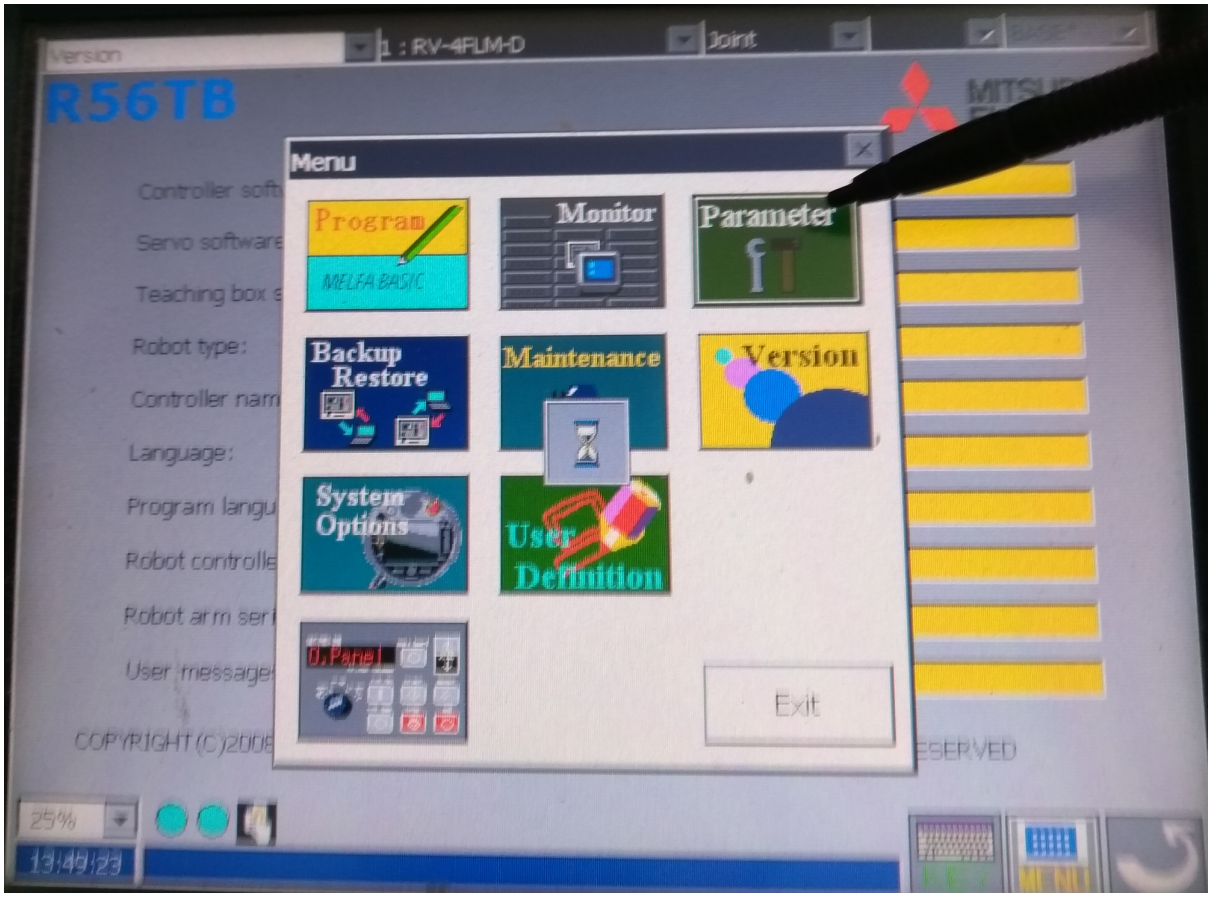


Abb. 9: Bild Mitsubishi Parameter Seite öffnen

Hier wählen Sie das Ethernet-Menü.

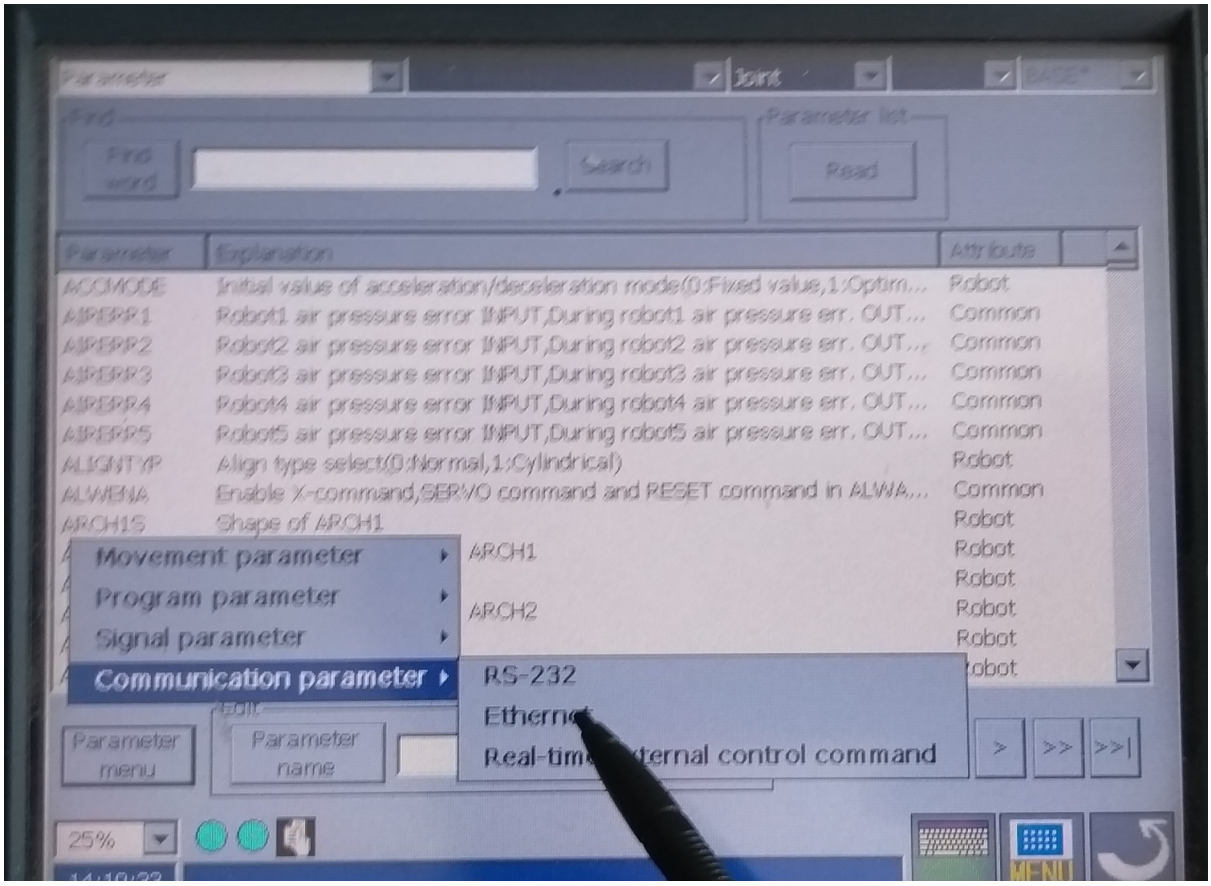


Abb. 10: Bild Mitsubishi TP Ethernet

In der nun erscheinenden Ansicht wählen Sie eine IP im gleichen Addressraum und die gleiche Subnetzmaske wie auf Ihrem Mikado-Controller. Außerdem müssen die Ports wie nachfolgend beschrieben werden, so dass eine Datenverbindung wie erwartet hergestellt werden kann.

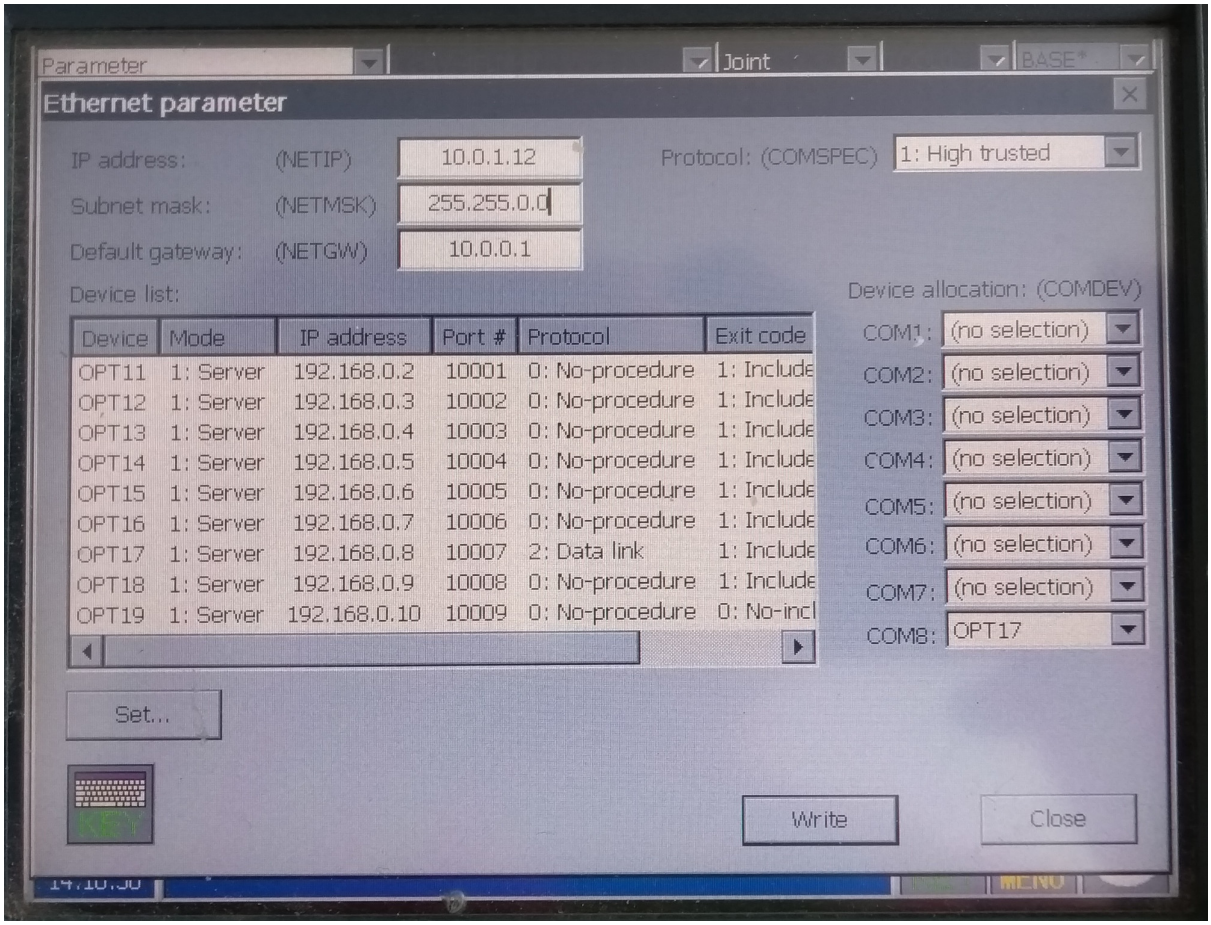


Abb. 11: Bild Mitsubishi Beispiel IP Konfiguration

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

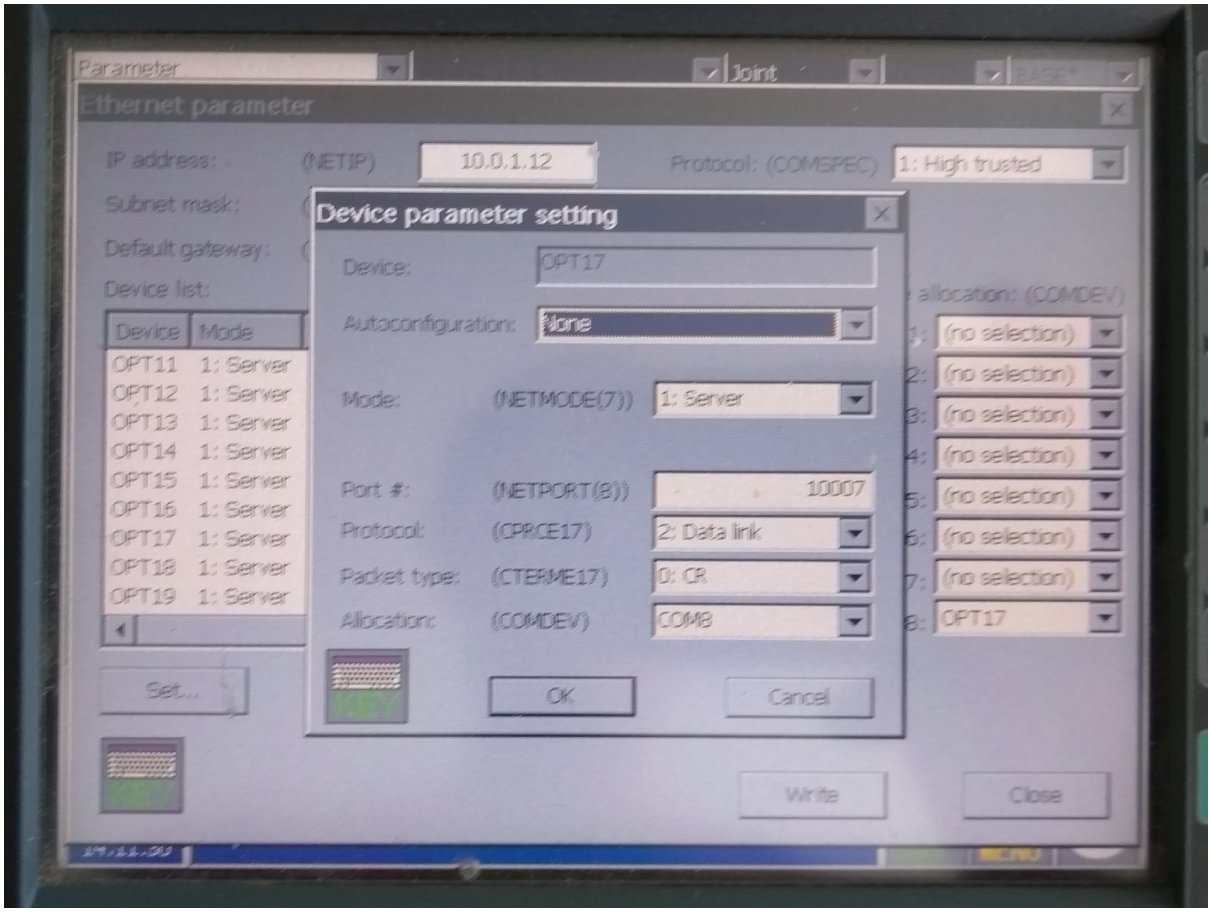


Abb. 12: Bild Mitsubishi Beispiel Port Konfiguration

Für alle anderen Ports ist keine Rollenzuweisung nötig, bzw. vorhandene sollten abgeschaltet werden durch setzen des *Protocol* Parameters auf den Wert *No procedure*:

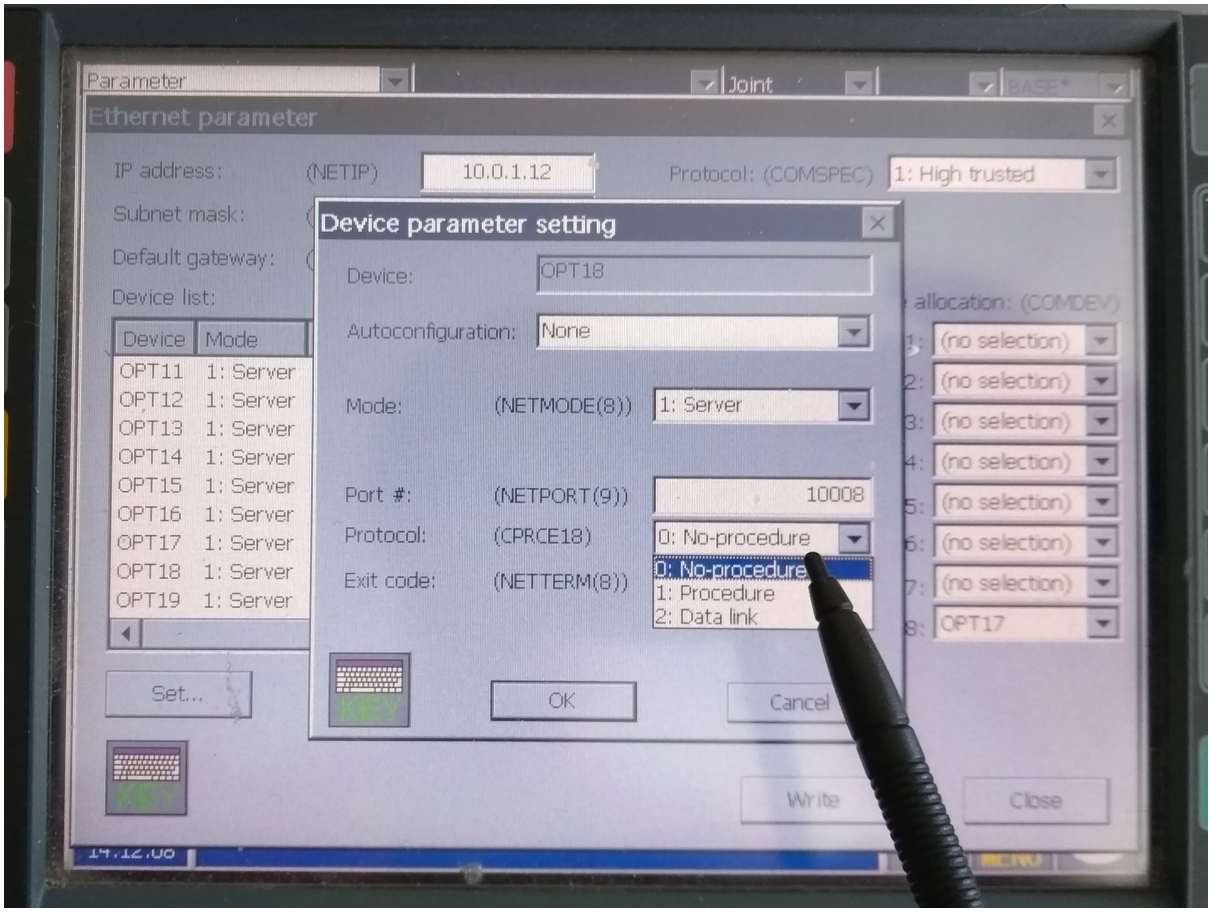


Abb. 13: Bild Mitsubishi Beispiel Port Neutral setzen

Die Änderungen müssen nun vom Teach Panel auf den Robotercontroller überschrieben werden.

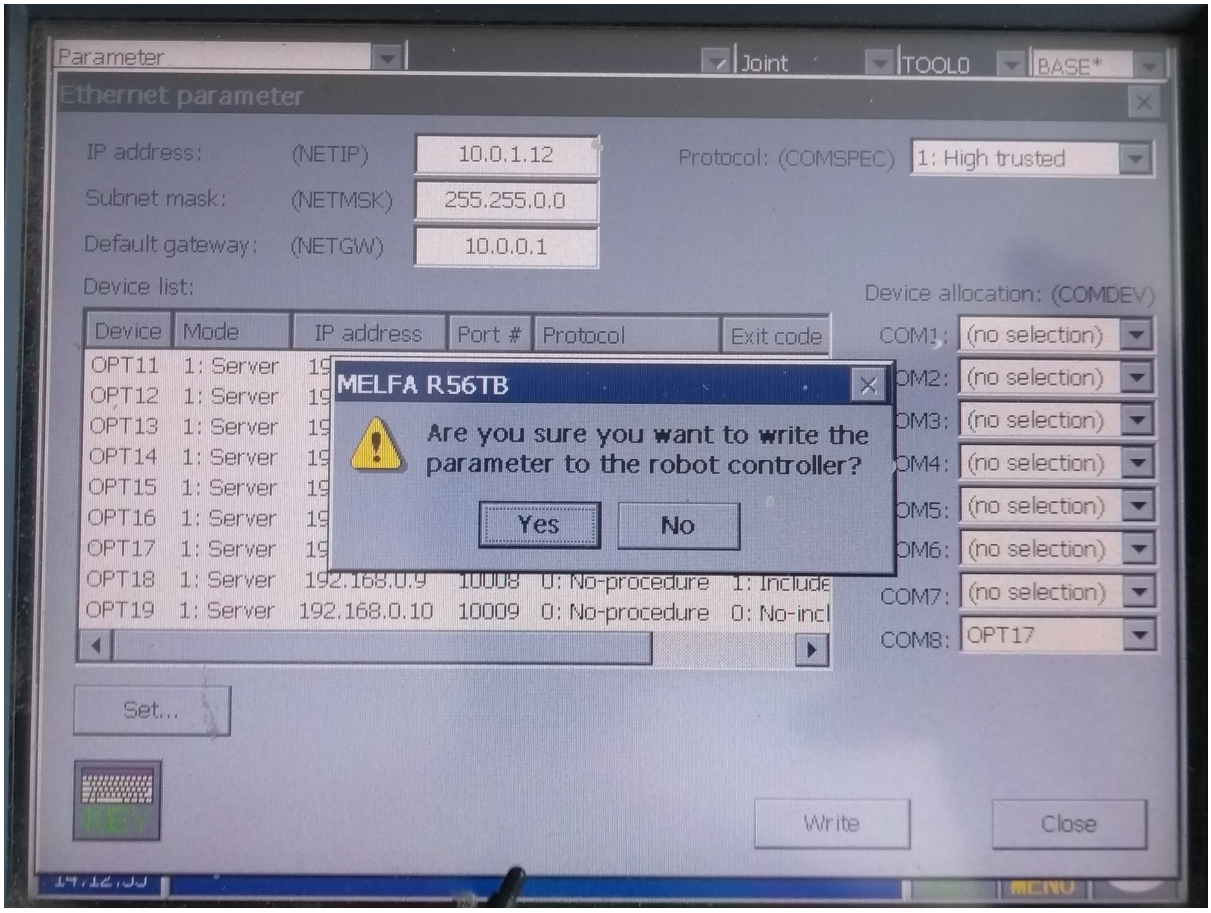


Abb. 14: Bild Mitsubishi Parameter Schreiben

Danach muss der Controller neu gestartet werden.

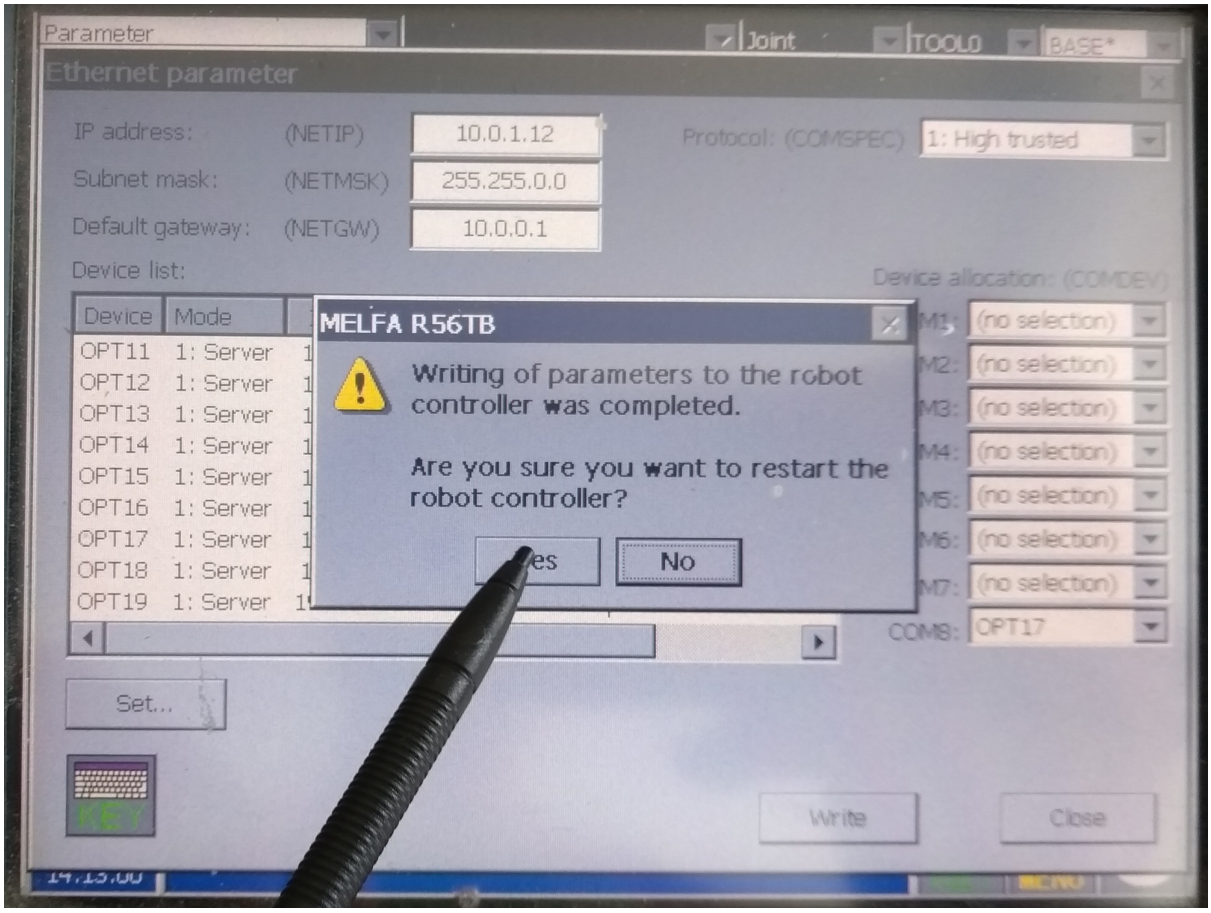


Abb. 15: Bild Mitsubishi Controller Neustart

In der Mikado Zellen-Konfiguration muss die entsprechende IP in *config* in der *manipulator.yaml* unter dem Parameter *robot_ip* eingestellt werden.

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*.

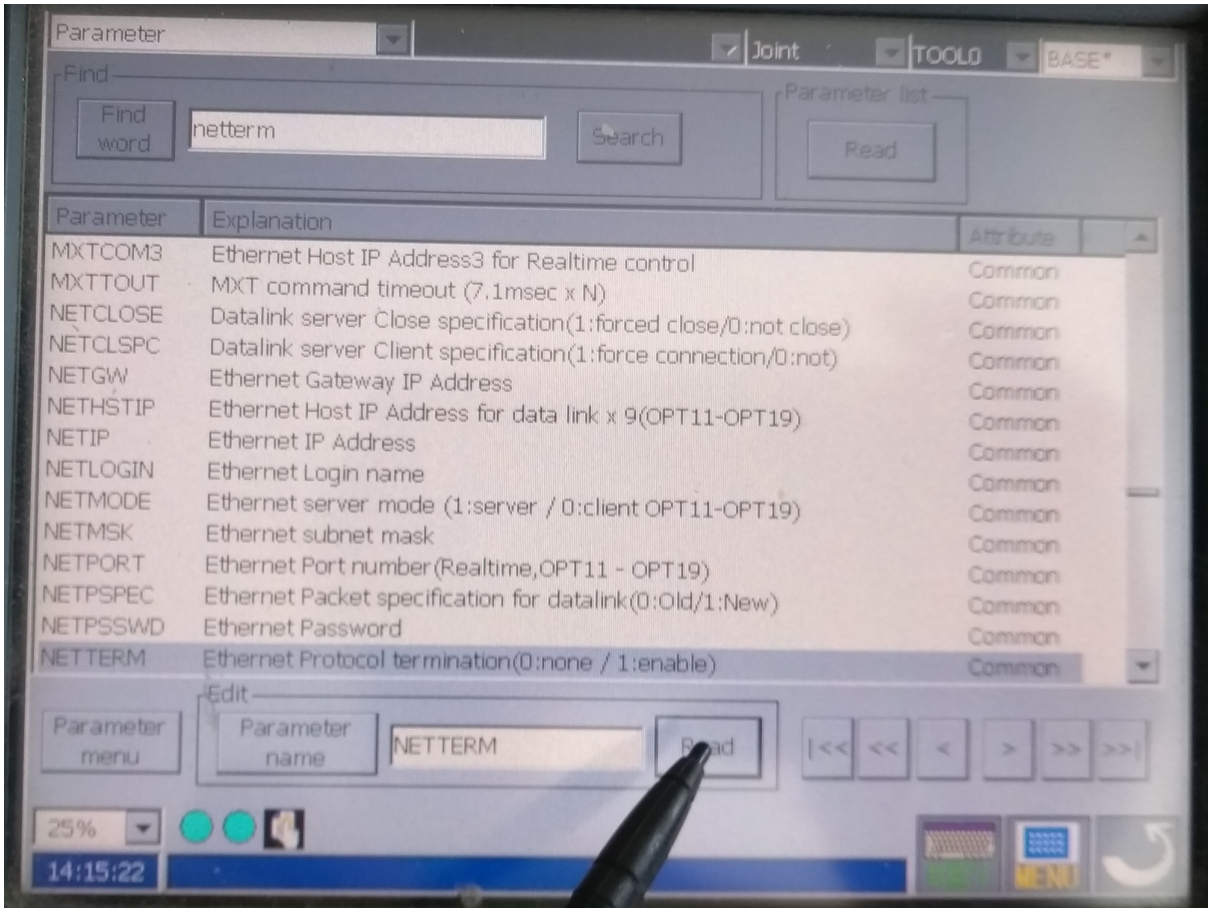


Abb. 16: Bild Mitsubishi Netterm Parameter finden

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.**

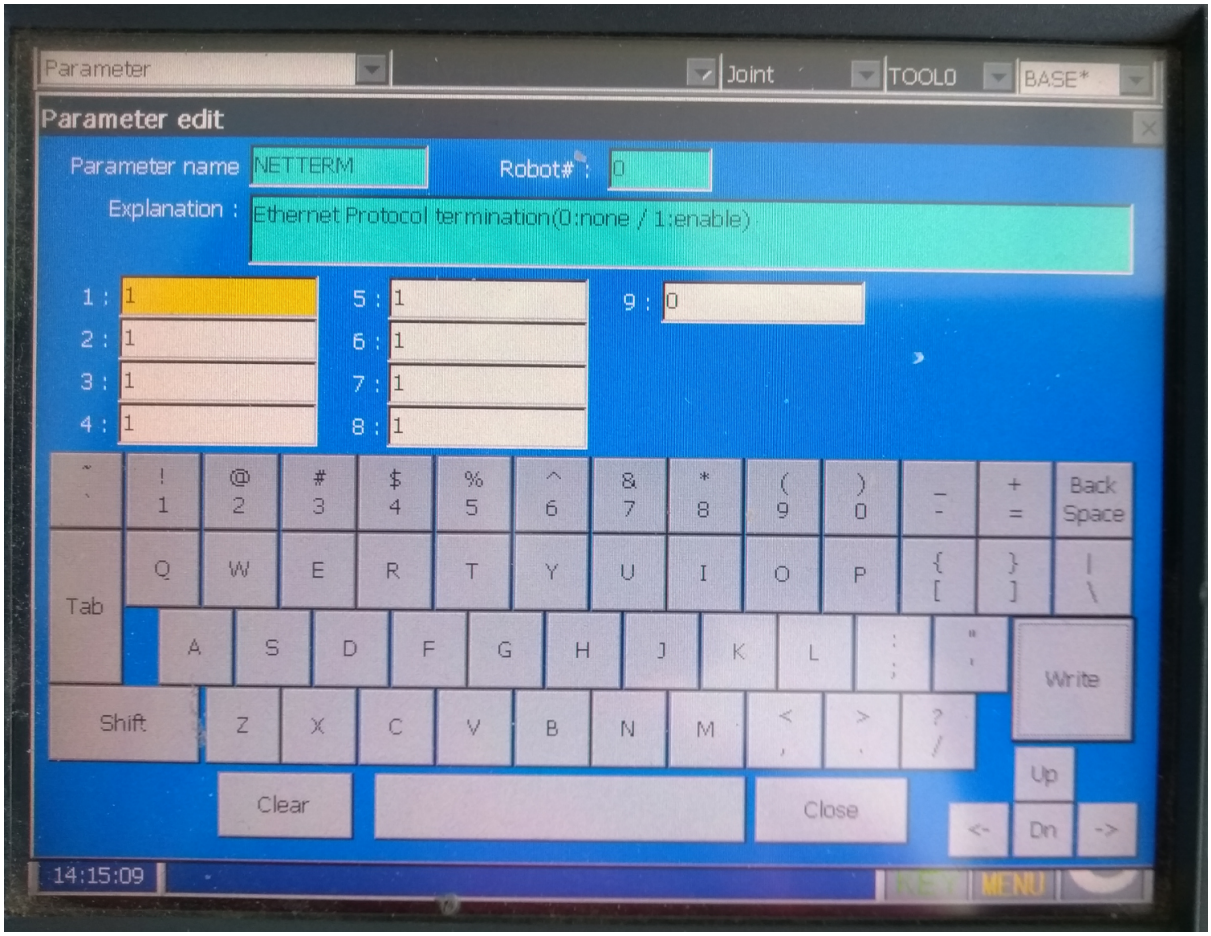


Abb. 17: Bild Mitsubishi Netterm Parameter editieren

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* .

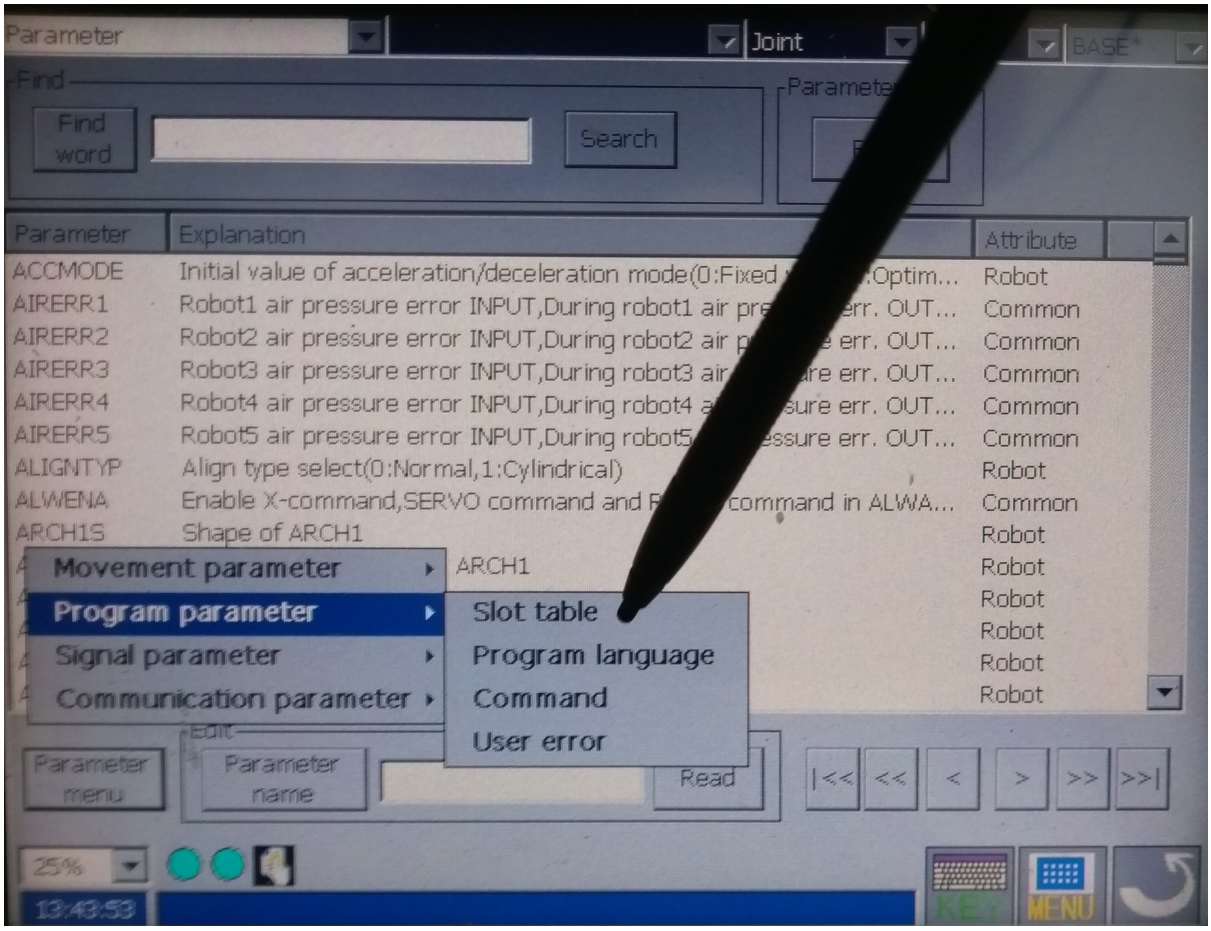


Abb. 18: Bild Mitsubishi Program Slot Dialog öffnen

Entfernen Sie nun gegebenenfalls Programme aus dem automatischen Start und starten Sie den Controller neu.

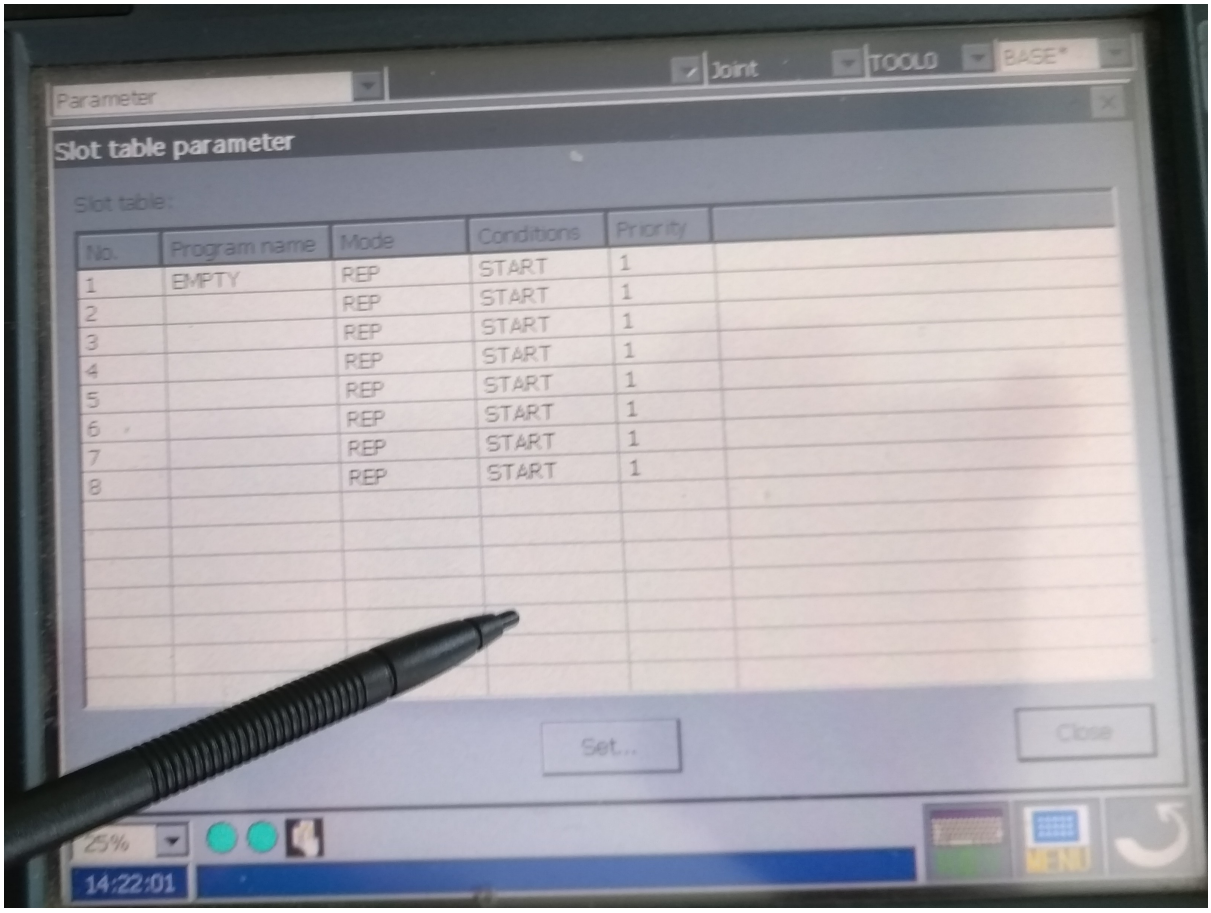


Abb. 19: Bild Mitsubishi Controller Slot table parameter

Der Mikado-Controller benötigt ausreichend freien Speicherplatz auf dem Robotercontroller. Dazu wählen Sie aus dem Auswahlmenü den Punkt *Program list*.

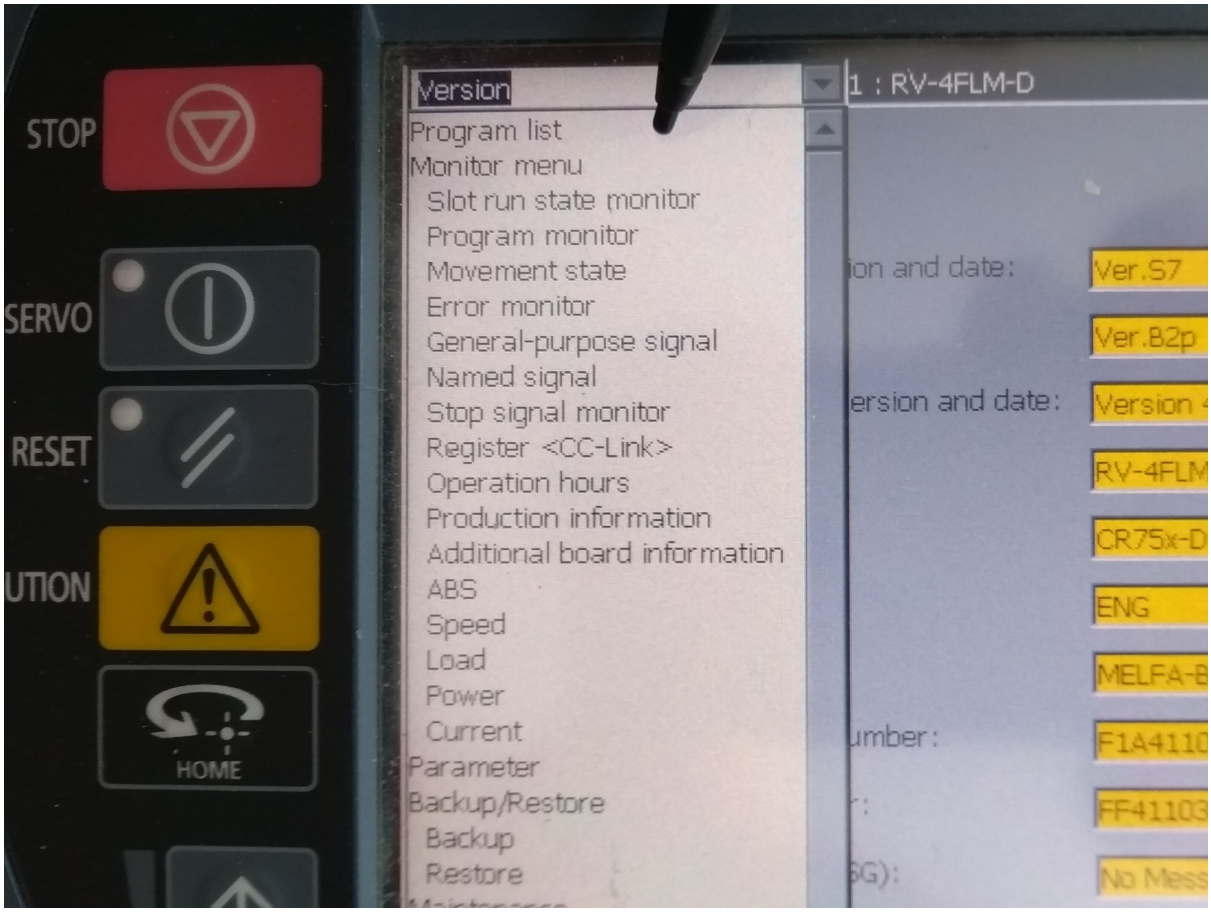


Abb. 20: Bild Mitsubishi Controller Programm Liste wählen

Es öffnet sich eine Programmübersicht, die die derzeit vorhandenen Programme und den freien Speicherplatz anzeigt.

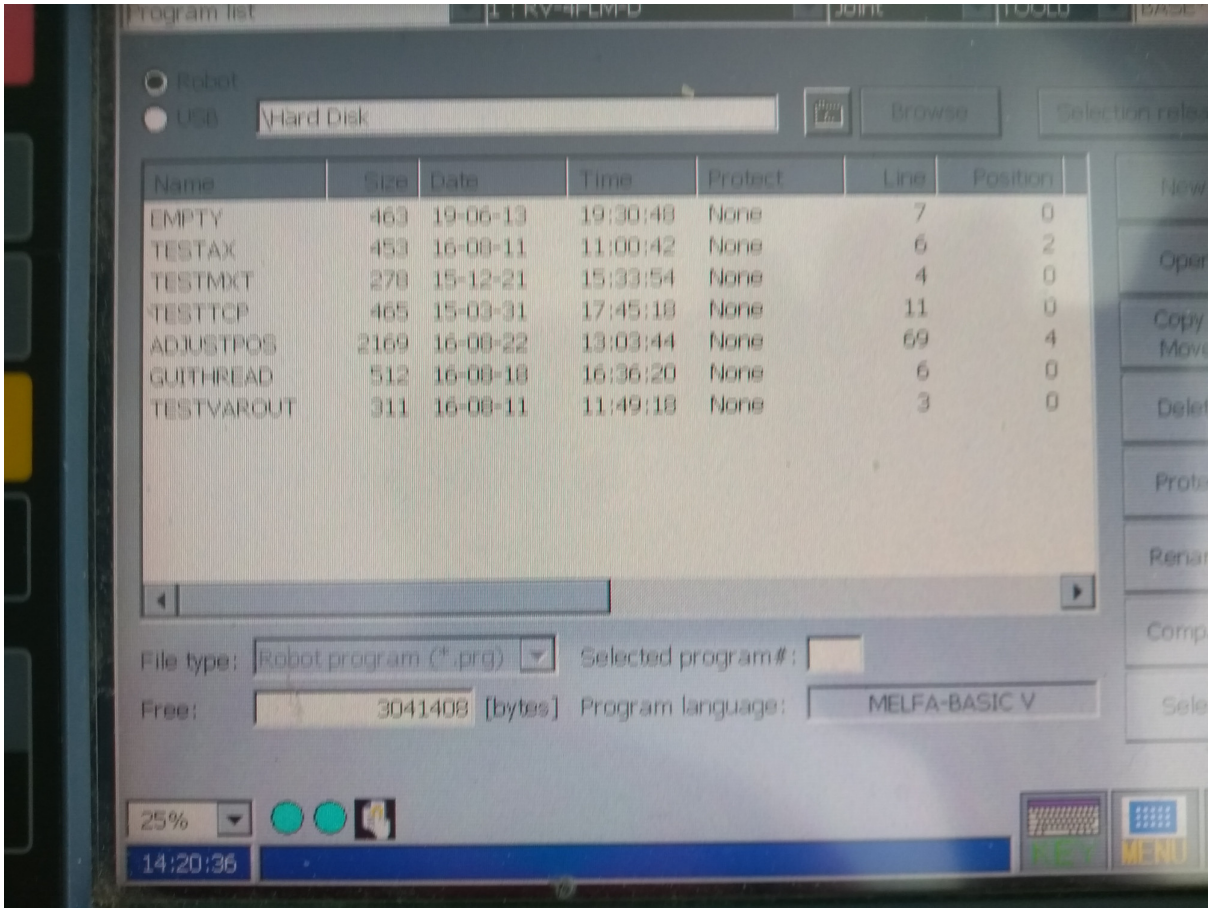


Abb. 21: Bild Mitsubishi Controller Speicherplatz

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 Addressraum
- 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
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!

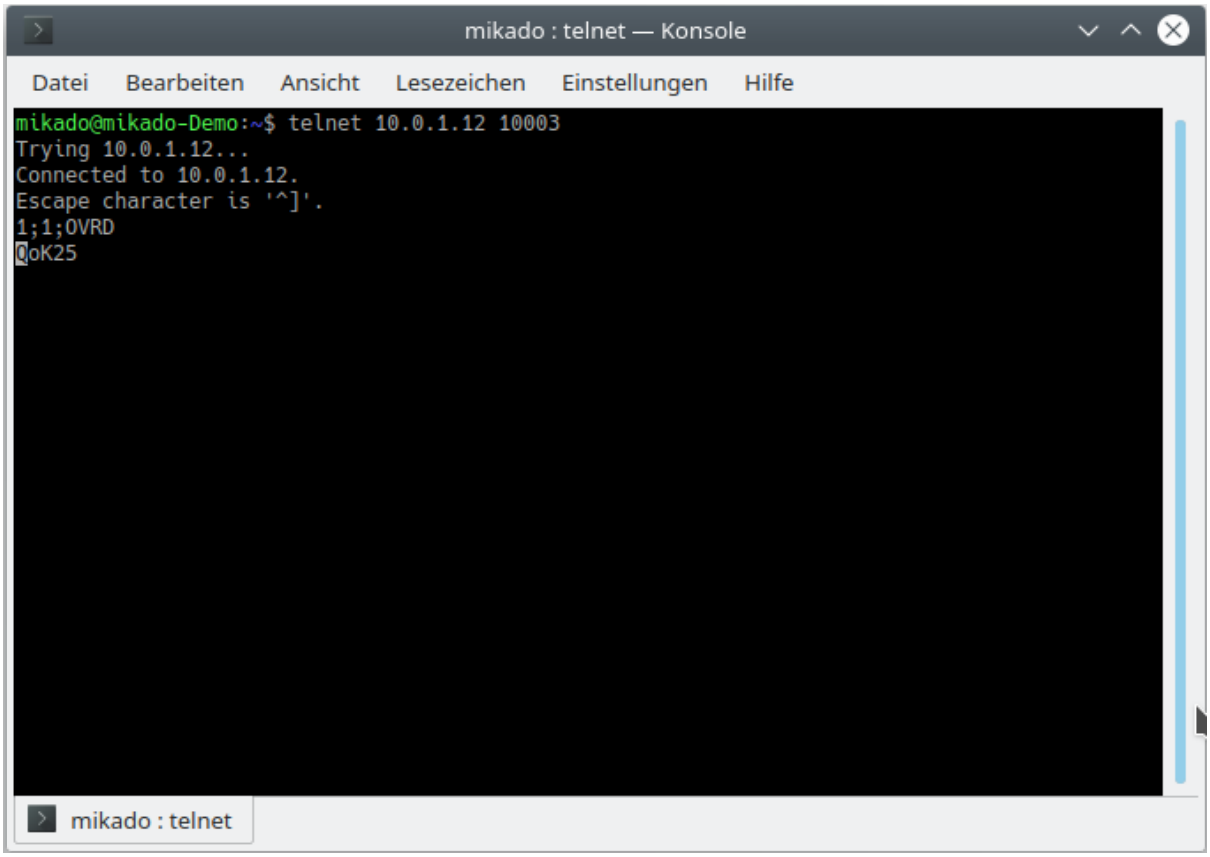


Abb. 22: Bild Mitsubishi Telnet OVRD Parameter lesen

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_signal_number: 900
closed_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.

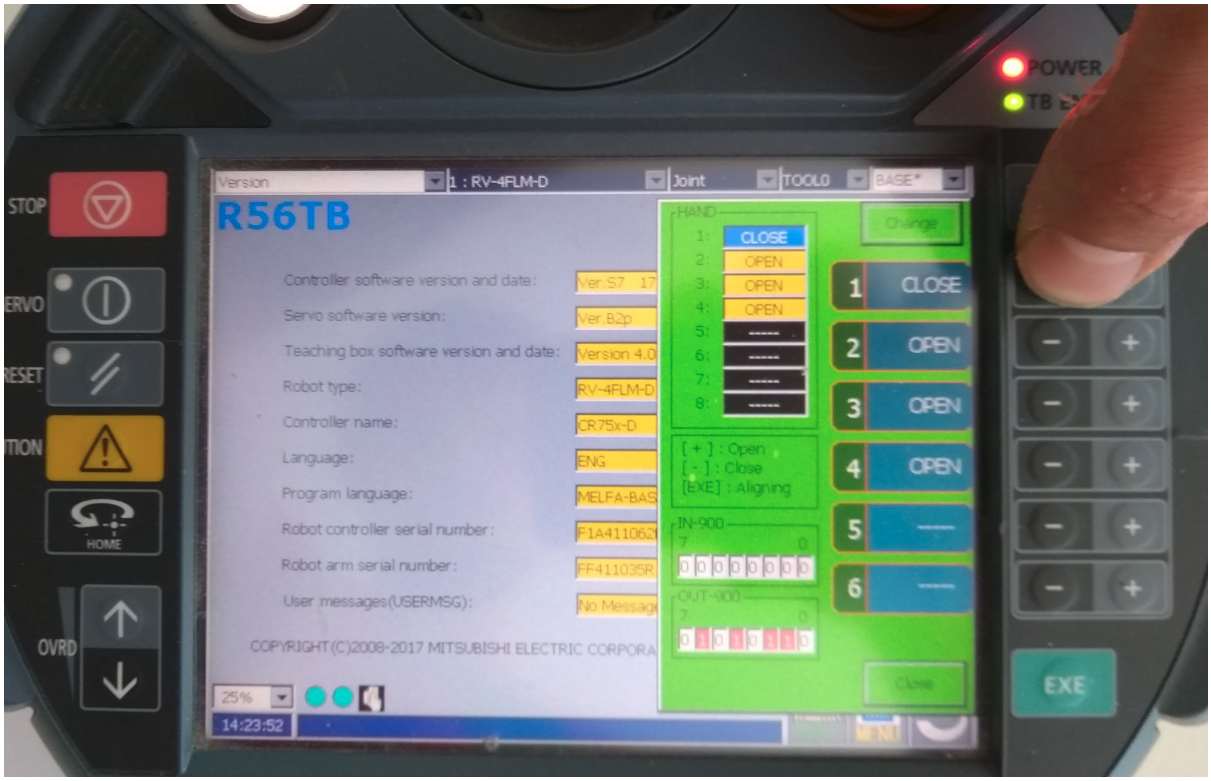


Abb. 23: Bild Mitsubishi Hand überprüfen

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.

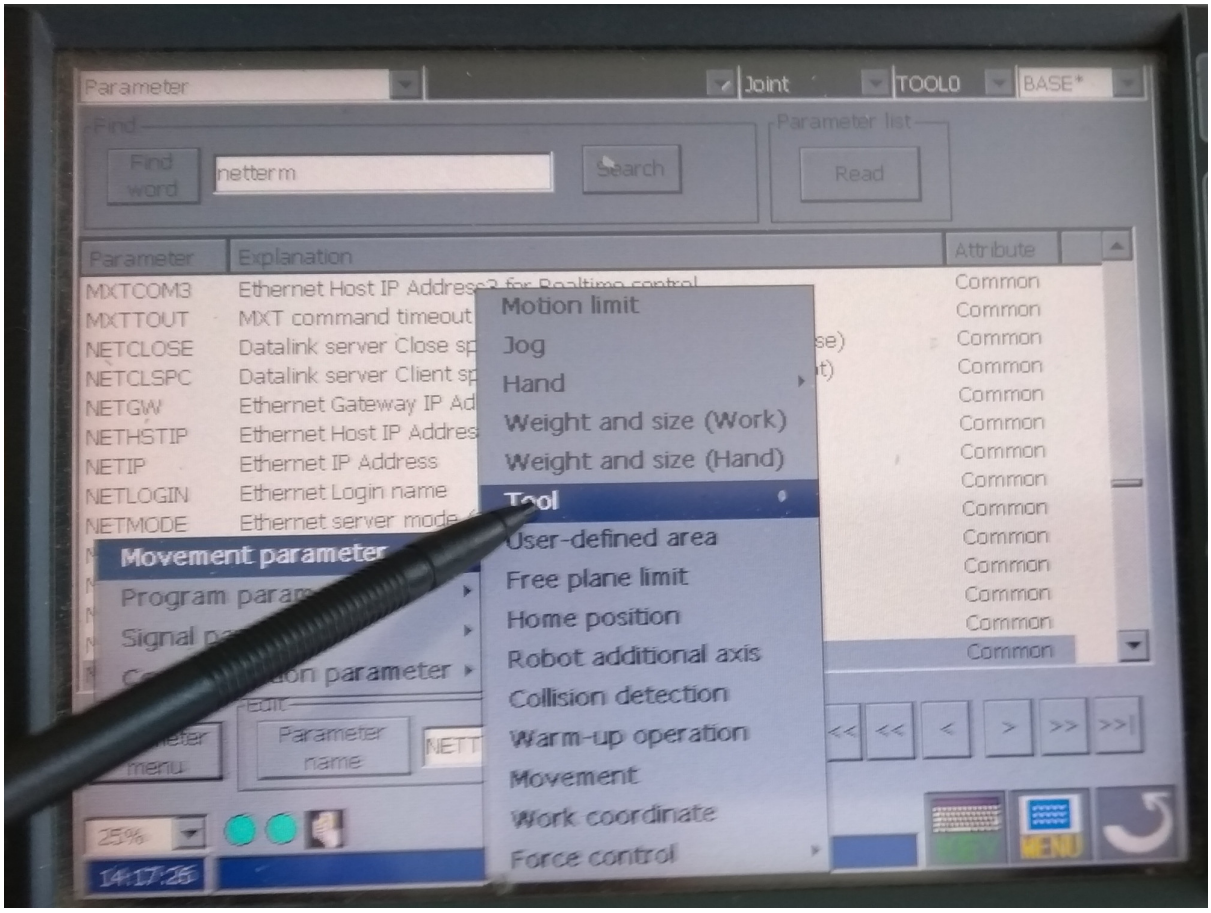


Abb. 24: Bild Mitsubishi Menü Koordinatensystem

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.

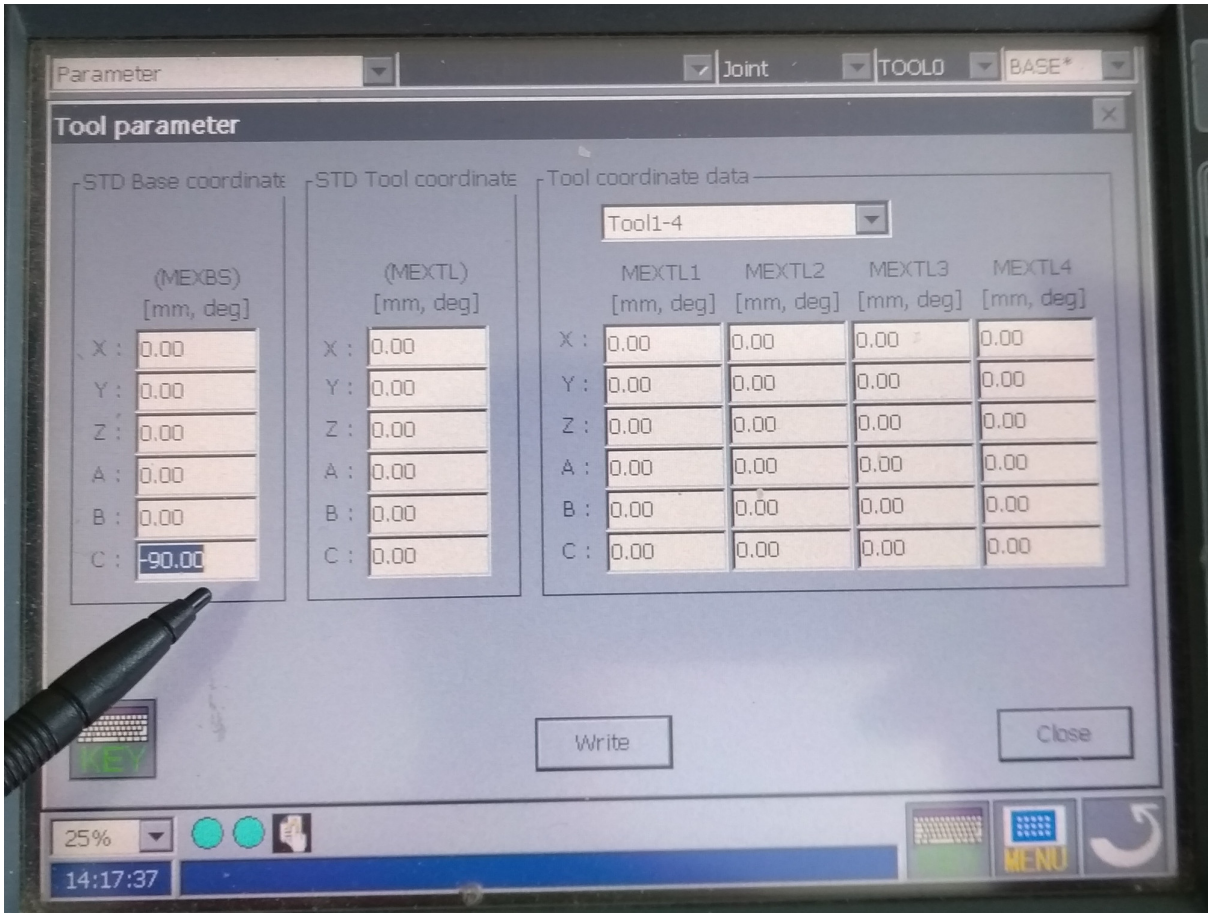


Abb. 25: Bild Mitsubishi Dialog Koordinatensystem anpassen

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.

Universal Robots Controller einrichten

Getestet mit UR Software V3.5.1, V3.9.1.64192.

IP-Adresse konfigurieren

Stellen Sie dieselbe IP-Adresse ein, die im *Launchfile* für den Roboter konfiguriert ist. Konsultieren Sie die Betriebsanleitung vom Hersteller um herauszufinden, wie Sie die IP an Ihrem Controller ändern können.

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 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.

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.

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/mikado_sps.sub“) und der Roboter-Interpreter („C:/KRC/ROBOTER/KRC/R1/Program/mikado.src“). Der Submit-Interpreter wird üblicherweise automatisch gestartet, das Roboter-Program muss ausgewählt werden. Diese kommunizieren beide per EthernetKRL mit dem Mikado Controller.

Systemanforderungen

Mikado kommuniziert über EthernetKRL mit dem Kuka KRC4-Controller und verwendet Splines für Fahrbefehle. Es muss daher mindestens Version 8.3 installiert sein sowie das Zusatzpaket EthernetKRL ab Version 2.3.

Kommunikation mit dem Controller

Für die Kommunikation mit dem Mikado Controller muss eine Netzwerkverbindung zum KRC4 existieren. Laut EthernetKRL Handbuch können dafür die Anschlüsse X66 oder X67.1-X67.3 verwendet werden. Stellen Sie sicher das der KRC4 und der Mikado Controller im selben IP-Bereich liegen (falls wie empfohlen eine separate Netzwerkverbindung für die Kamera verwendet wird, müssen sich diese in verschiedenen IP-Bereichen befinden!).

Einrichten

Um den Kuka-Roboter mit Mikado ARC steuern zu können, müssen zunächst die entsprechenden Programme auf den Controller kopiert werden. Zu finden sind sie unter /opt/isys-vision/mikado/robot_programs/kuka/

Dort gibt es drei Unterordner, die jeweils einem Ordner auf dem Controller entsprechen:

- EthernetKRL/* -> C:\KRC\ROBOTER\Config\User\Common\EthernetKRL*
- Program/* -> C:\KRC\ROBOTER\KRC\R1\Program*
- System/* -> C:\KRC\ROBOTER\KRC\R1\System*

Im Ordner EthernetKRL finden Sie die Netzwerkeinstellungen. Aus Sicherheitsgründen muss auf dem Roboter die IP des Mikado-Controllers hinterlegt werden. Dazu müssen Sie in den beiden Dateien `XmlReceiveSub.xml` und `XmlTransmitSub.xml` jeweils in Zeile 10 bzw 11 die IP-Adresse des Mikado Controllers angeben. Dies geht beispielsweise recht bequem mit dem Texteditor Kate nachdem Sie die Dateien auf einen USB-Stick kopiert haben (der Ordner /opt und alle enthaltenen Dateien sind schreibgeschützt). Oder etwas weniger bequem direkt auf dem Controller. Nach Änderrungen an der IP ist ein Neustart des Kuka Controllers notwendig.

Der System-Ordner enthält Programme, die vom Submit-Interpreter ausgeführt werden können, insbesondere die Datei „mikado_sps.sub“.

Der Program-Ordner enthält Programme, die vom Roboter-Interpreter ausgeführt werden können, insbesondere die Datei „mikado.src“ (welche wiederum andere .src-Dateien aufruft).

Wählen Sie im Submit-Interpreter die Datei `mikado_sps.sub` an. Diese muss einmalig angewählt werden und startet dann automatisch (sofern kein Fehler vorliegt). Im Mikado ARC sollte der Roboter nun in der richtigen Stellung angezeigt werden.

Wählen Sie nun das Hauptprogramm `mikado.src` an, dies kann in allen Betriebsmodi (T1, T2, Auto oder Extern) erfolgen. Wenn Sie nun im Mikado auf der Roboterseite z.B. eine kleine Fahrt von Achse 6 vornehmen, sollte diese vom Kuka ausgeführt werden.

Um den Greifer einzurichten passen Sie in der Datei `mikado.dat` die folgenden Zeilen an:

```
global signal gripper_open   $OUT[1]
global signal gripper_closed $OUT[2]
```

Falls Sie einen etwas komplizierteren Ablauf zum Steuern des Greifers benötigen, können Sie auch direkt die Datei `handle_req_gripper.src` bearbeiten.

Submit-Interpreter

Der Submit-Interpreter läuft dauerhaft im Hintergrund und schickt Informationen zum Status raus: Die jeweils aktuelle Roboter-Position sowie einige Zusatzinfos. Die Details sind in „C:/KRC/ROBOTER/Config/User/Common/EthernetKRL/XmlTransmitSub.xml“ festgelegt. Folgende Daten werden ausgegeben:

- `robot/axis/@j1`
- `robot/axis/@j2`
- `robot/axis/@j3`
- `robot/axis/@j4`
- `robot/axis/@j5`
- `robot/axis/@j6`
- `robot/status/@drives_powered`
- `robot/status/@e_stopped`
- `robot/status/@error_code`
- `robot/status/@in_error`
- `robot/status/@in_motion`
- `robot/status/@mode`
- `robot/status/@motion_possible`

Roboter-Interpreter

Der Roboter-Interpreter wiederum empfängt Befehle und entweder (a) führt sie aus und gibt eine entsprechende Erfolgs-Antwort zurück, oder (b) gibt eine Fehler-Antwort zurück. Die Befehle sind in „C:/KRC/ROBOTER/Config/User/Common/EthernetKRL/XmlReceiveSub.xml“ festgelegt. Folgende Befehle gibt es bisher:

- default_vel: Setzen der Zielgeschwindigkeit
  - `pc/default_vel/@vel` (real): Zielgeschwindigkeit in Prozent
- read_input: Eingangssignal lesen
  - `pc/read_input/@which` (int): Index des zu lesenden Eingangs
- read_output: Ausgangssignal lesen
  - `pc/read_output/@which` (int): Index des zu lesenden Ausgangs
- write_output: Ausgangssignal lesen
  - `pc/write_output/@which` (int): Index des zu setzenden Ausgangs
  - `pc/write_output/@value` (bool): zu setzender Wert

- gripper: Greifer öffnen / schließen
  - `pc/gripper/@open` (bool): zu setzender Wert des open-Ausgangs (Ausgang #1)
  - `pc/gripper/@closed` (bool): zu setzender Wert des closed-Ausgangs (Ausgang #2)
- traj: Trajektorie abfahren
  - `pc/traj/@j` (real): Array von Gelenkwerten (muss Multiples von 6 sein, da 6 Gelenke)
- traj_vel: Trajektorie mit Geschwindigkeiten abfahren
  - `pc/traj_vel/@j` (real): Array von Gelenkwerten (muss Multiples von 6 sein, da 6 Gelenke)
  - `pc/traj_vel/@v` (real): Array von Gelenkgeschwindigkeitswerten (muss genauso viele Werte wie @j enthalten)
- run_program: Programm ausführen
  - `pc/run_program/@name` (string): Name des auszuführenden Programms (der auszuführenden Funktion)
  - `pc/run_program/boolArg` (bool): Array von bool'schen Argumenten
  - `pc/run_program/intArg` (int): Array von int-Argumenten
  - `pc/run_program/realArg` (real): Array von float-Argumenten
  - `pc/run_program/strArg` (string): Array von string-Argumenten
  - `pc/run_program/poseArg` (frame): Array von Pose-Argumenten –> wird aktuell nicht verarbeitet, aufgrund eines Bugs in EthernetKRL

Hinzufügen von eigenen Programmen

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/handle_req_run_program.src“ das Programm im Bereich „call program“ (ganz unten) hinzugefügt werden. (Beispiel-Code fürs Hinzufügen steht direkt unter dem „call program“-Kommentar. Einkommentieren nicht vergessen.)

N.B.: Für die KRL-Syntax gibt es ein Language-Server-Plugin. Dieses kann bspw. mit Visual Studio Code verwendet werden, um Syntax-Highlighting zu bekommen.

Standard-Antwortformat

Alle Requests außer „run_program“, „read_input“ und „read_output“ führen zu einer Antwort in folgendem Format:

- `robot/command/@accept` (bool): Erfolg: ja / nein
- `robot/command/@error_msg` (string): Fehlernachricht (im Fehlerfall) oder " " (im Erfolgsfall)
- `robot/command/@runtime_ms` (int): optional (d.h.: kann leer sein). Gibt die Laufzeit der Befehlsausführung in Millisekunden an

N.B.: Derzeit wird „runtime_ms" nur bei den Requests „traj" & „traj_vel" gemessen und übertragen.

run_program-Antwortformat

Requests der Art „run_program" führen zu einer Antwort in folgendem Format:

- `robot/result/@success` (bool): Erfolg: ja / nein
- `robot/result/@error_msg` (string): Fehlernachricht (im Fehlerfall) oder " " (im Erfolgsfall)
- `robot/result/@boolRes` (bool): optionaler bool-Rückgabewert
- `robot/result/@intRes` (int): optionaler int-Rückgabewert
- `robot/result/@realRes` (real): optionaler float-Rückgabewert
- `robot/result/@strRes` (string): optionaler string-Rückgabewert

N.B.: optionale Rückgabewerte können leer sein, z.Bsp.: <robot><result success="1" error_msg=" " boolRes="" intRes="5" realRes="" strRes="" ></robot>

read_input-/read_output-Antwortformat

Requests der Art „read_input" bzw „read_output" führen zu einer Antwort in folgendem Format:

- `robot/io/@is_input` (bool): Gibt an, ob sich der Rückgabewert auf einen Eingang oder einen Ausgang bezieht
- `robot/io/@which` (int): Index des ausgelesenen Signals; ist im Fehlerfall -1
- `robot/io/@value` (bool): Wert des ausgelesenen Signals
- `robot/io/@error_msg` (string): Fehlernachricht (im Fehlerfall) oder " " (im Erfolgsfall)

Roboterfahrten

Roboterfahrten werden derzeit als Spline-Blöcke mit SPTP-Schritten ausgeführt. Es gibt Spline-Blöcke der Größe 1, 2, 3, 4, 5, 6, 7, 8, 16, 24, 32, die kombiniert werden können um Trajektorien beliebiger Größe abzufahren. Um Spline-Blöcke zu interpolieren, muss an das „endspline" ein „c_spl" angehängt werden. Daher gibt es für die Blöcke der Größen 8, 16,

24, 32 jeweils zwei Funktionen, einmal mit „c_spl“ („run_spline_32_cont“) und einmal ohne („run_spline_32“). Der letzte Block muss immer ein Block ohne „c_spl“ sein, sonst kommt eine Warnung (was Zeit kostet). Da nicht klar ist, ob das Überschleifen zwischen den Spline-Blöcken ein normales PTP-Überschleifen ist oder in die Spline-Berechnung mit einfließt, wurde versucht, für jede Trajektorien-Größe möglichst wenige verschiedene Blöcke zu verwenden. Alle Größen <= 40 verwenden nun maximal 2 Blöcke. Danach kommt alle 32 Schritte ein weiterer Block hinzu. (> 40 Schritte: 2-3 Blöcke, > 72 Schritte: 3-4 Blöcke, > 104 Schritte: 4-5 Blöcke, etc.pp.)

Die derzeitige maximale Blockgröße von 32 lässt sich bei Bedarf beliebig erweitern. Ist bspw. 64 gewünscht, bräuchte es „run_spline_<N>“- & „run_spline_<N>_cont“-Funktionen für N = 40, 48, 56, 64, sowie eine entsprechende Erweiterung des switch-statements in der „run_robot“-Funktion. Alternativ könnte man „run_spline_<N>“-Funktionen für N = 9-15 hinzufügen und dafür die beiden Funktionen für 24, 40, 56 streichen (übrig blieben dann: 16, 32, 48, 64).

Das wird alles nur gebraucht, wenn die Sampling-Dichte der Trajektorie sehr hoch ist. Dies kann bspw. benötigt werden, um Cartesische Fahrten (Linearfahrten, Kreisförmige Fahrten) zu implementieren, oder aber um möglichst eng um Hindernisse herumzufahren ohne aufgrund von Abkürzungen zu kollidieren.

Fanuc Controller einrichten

Dieses Kapitel führt Sie durch Installation und Konfiguration eines Fanuc-Controllers zur Nutzung mit einem Mikado-System.

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:
  - R632 - KAREL
  - 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.

- Register, Variablen, Server Tags: Folgende Slots müssen auf dem Controller freigehalten werden:
  - Server Tags: 3 - 5
  - Flags: 1 - 2
  - Integer Register: 1 - 2
  - Positionsregister: 1 - 2
- USB-Stick: Als Installationsmedium verwenden Sie am besten einen USB-Stick. Alternativ können Sie Dateien auch über das Netzwerk (FTP) übermitteln (s. Handbuch Fanuc).

Verbindung herstellen

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. Verwenden Sie bitte das Benutzerhandbuch Ihres Robotercontrollers, um Informationen zum Einrichten der TCP/IP Verbindung zu erhalten.

In der Mikado Zellen-Konfiguration muss die entsprechende IP in *config* in der *manipulator.yaml* unter dem Parameter *robot_ip* eingestellt werden.

Mikado-Schnittstelle installieren

Die Versionskompatibilität ist mit Version v8.20 getestet, die Installationsdatei ist zu höheren Versionen womöglich ebenfalls kompatibel.

- Kopieren Sie die Installationsdatei aus dem Verzeichnis `/opt/isys-vision/mikado/robot_programs/fanuc` auf einen USB Stick (FAT formatiert)
- Stecken Sie den USB Stick am Teach-Panel ein und navigieren zum Inhalt (*Menu* → *File* → *File* → *All*)
- Öffnen / Starten Sie die Installationsdatei und führen Sie die Installation durch

Mikado-Schnittstelle parametrieren

- Server Tags:

  - Mithilfe des Tech-Panels nach *Menu* → *SETUP* → *Host Comm* → *[SHOW]* → *Servers* navigieren
  - Server Tags für Tag 1 bis 3 einrichten (mit ENTER Server Tag auswählen):
  - Gehen Sie zurück zur Serverliste und prüfen Sie den momentanen Status der drei eingerichteten Tags. Dieser sollte auf ‚STARTED' stehen. Falls dies nicht der Fall ist, mittels *[ACTION]* → *DEFINE* und *[ACTION]* → *START* starten.

- 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.

- KAREL Programme konfigurieren:
  - *Select* → *[TYPE]* → *KAREL Progs*, dort das KAREL Programm ROS_STATE auswählen
  - *Data* → *[TYPE]* → *KAREL Vars*
  - Nun sollte die `cfg_` Variable zu sehen sein → *ENTER*
  - *checked* mit ENTER auswählen und auf TRUE setzen

  Nun dasselbe wiederholen für die beiden weiteren Programme ROS_TRAJ und ROS_IO.

  Sie können in der Konfiguration weitere Anpassungen vornehmen. Interessante Parameter des Programms *ROS_TRAJ* sind z.B.
  - `move_cnt`: CNT für die Bewegungen (prozentualer Wert für das Überschleifen)
  - `move_speed`: Geschwindigkeitsfaktor für die Bewegungen (wird mit dem eingestellten Wert am Tech-Panel multipliziert)

  Parameter des Programms *ROS_TRAJ* müssen wiefolgt eingestellt werden:
  - `s_tcp_nr: 11003`
  - `s_tag_nr: 5`

Hinweis: Unter Sektion „*Optional: Natives Roboterprogramm ausführen*" wird erläutert, was Sie zusätzlich konfigurieren müssen, damit native Roboterprogramme in Ihren Mikado-Ablauf eingebunden werden können

---

System starten

Das Programm ‚ROS_WITH_IO' 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).

Optional: Natives Roboterprogramm ausführen

Mithilfe des Tasks „*Roboterprogramm ausführen*" lassen sich native Programme auf dem Controller ausführen.

Um diesen Task in Benutzung eines Fanuc Roboters zu verwenden, müssen folgende Vorkehrungen getroffen werden:

- HTTP Zugang ist in den Einstellungen aktiviert und
  - Zugangsdaten sind einrichten. Nutzer: `ROS`, Passwort: `ROS`
  - Folgende Kommandos über den HTTP Zugang sind freigeschaltet: `RUN` und `ABORT`
- Unter *Menu* → *System* → *Setup* → *Remote/Local Setup*: von `local` auf `remote` stellen
- *Menu* → *System* → *Variables*: Den Wert `$RMT_MASTER = 1` einstellen
- (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: 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

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 Addressraum

- 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
```

Stäubli Controller einrichten

Die Dokumentation für die Einrichtung eines Stäubli Controllers ist noch nicht ganz fertig gestellt, bitte kontaktieren Sie uns für eine Vorabversion.

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 dem Controller Software Stand 1.72.

TCP/IP Verbindung herstellen

Zunächst muss eine Ethernet Verbindung hergestellt werden. Dies muss *zwingend auf Port 1* geschehen, da dieser alleinig mit dem Listen Node funktioniert. Unter dem Menüpunkt *System*, dann *Network* können die TCP/IP Einstellungen angepasst werden.

Hinweis: Der Port 1 entspricht der *Local Area Connection 9* auf der Network Seite. Der Robotercontroller sollte vom Mikadocontroller aus per *ping* erreichbar sein.

Modbus Verbindung herstellen

Unter dem Menüpunkt *Setting*, dann *Modbus* kann der derzeitige Modbus Zustand abgerufen werden, der aktiviert sein sollte. Hinter *Status* sollte *Enable* stehen.

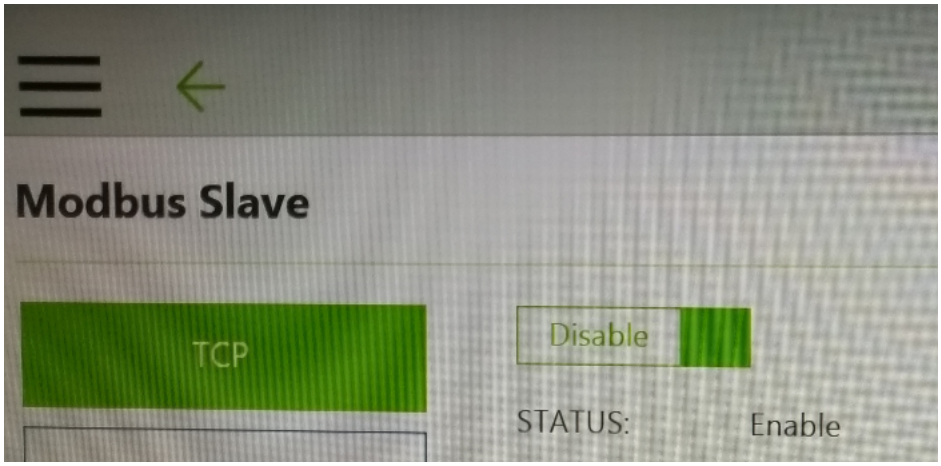


Abb. 26: Bild: Einstellung Modbus Enable

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.

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.

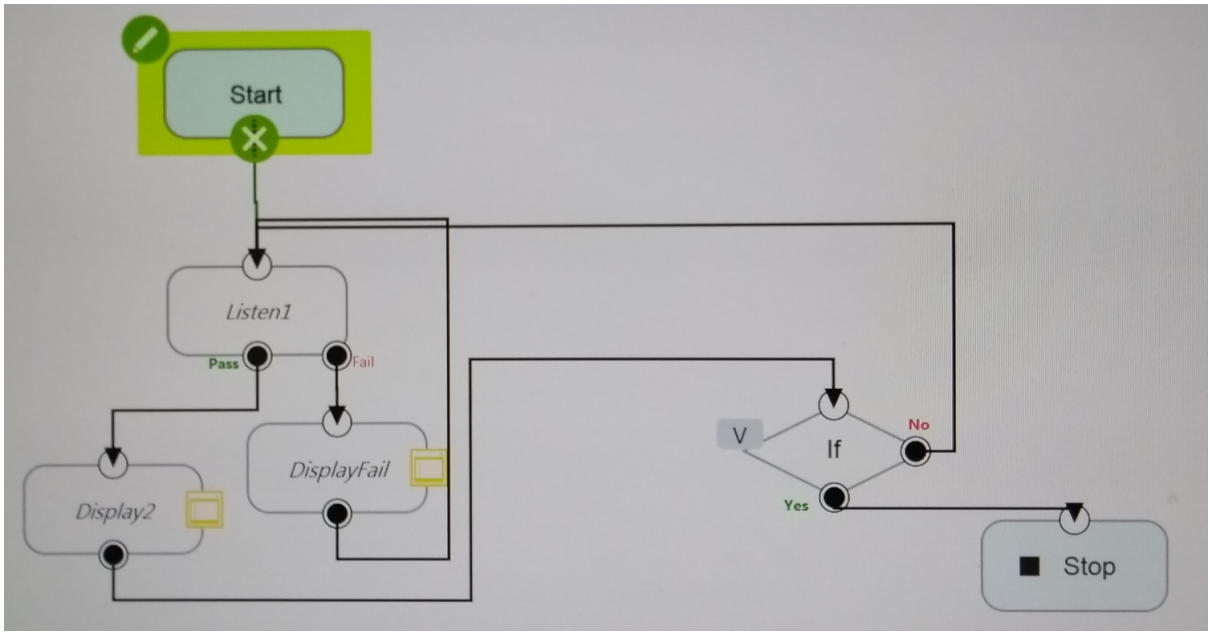


Abb. 27: Listen Node mit Debug Display und Möglichkeit zurückzukehren

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.

Projekt testen

Per *Diagnosis*, dann *Stepping* lässt sich das Programm/Flow testweise starten und durchsteppen. In der Konsole sollte auch die TCP/IP Verbindung mit Adresse geschrieben 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.

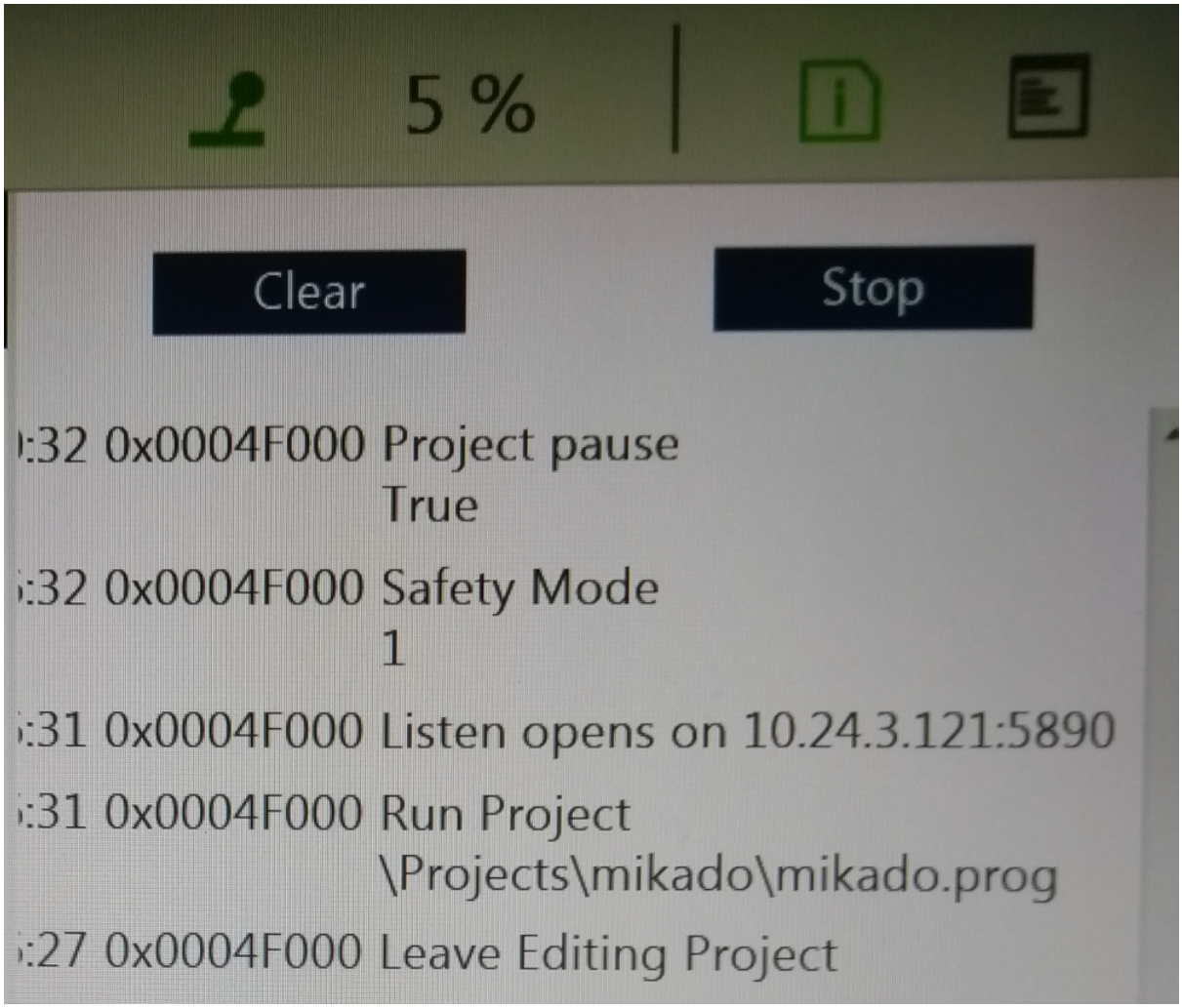


Abb. 28: Listen Node in der Konsole

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 und gibt so die Kontrolle ab. 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.

7.7.4 ProfiNet mit Hilscher

Bitte kontaktieren Sie uns für eine Vorabversion dieses Dokuments. Teilen Sie uns dabei gerne direkt mit welchen Feldbus sie genau benötigen und ob Mikado als Slave oder Master (experimentell) fungieren soll.

7.7.5 Wago/Beckoff IO

Die Dokumentation für das Verbinden zu einem IO-Koppler von Wago, Beckoff oder auch einem anderen Modbus/TCP Server ist noch nicht ganz fertig gestellt, bitte kontaktieren Sie uns für eine Vorabversion.

KAPITEL 8

Mikado ARC Erweitern

8.1 JavaScript Task

Der ‚*JavaScript Task*' erlaubt es eigenen JavaScript/ECMAScript Code als Task auszuführen. Auf das Task-Objekt 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")
```

Auf Variablen kann über das `dataStore` Objekt 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 Rückggabewert `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.

Log-Einträge können ebenfalls geschrieben werden:

```
console.log("My log message")
```

Weitere Links:

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

8.2 Halcon Script Task

Die Halcon Bildverarbeitungsbibliothek der Firma MVTec Software GmbH ermöglicht es, eigene Bildverarbeitungsalgorithmen in einer Halcon-spezifischen Scriptsprache 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 ‚*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:

- **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 rektivizierte 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). Seriennumer 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. mittels des Tasks ‚*Punktwolke aufnehmen*‘ auslösen.

- **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)`

- **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

Fortsetzung auf der nächsten Seite

Tab. 3 – Fortsetzung der vorherigen Seite

Parameter	Parameter Name	Beschreibung	Beispiel
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.

- **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 VariableString-Value / VariableNumericValue	Beispiele
Integer	✓	Numeric	1 42
Double	✓	Numeric	1.1 3.14
String	✓	String	‚meinWert‘

Fortsetzung auf der nächsten Seite

Tab. 5 – Fortsetzung der vorherigen Seite

Variablentyp	Schreiben möglich	Eingabe über VariableString-Value / Variable-NumericValue	Beispiele
StringList	✓	String	[‚StringListWert1‘, ‚String-ListWert2‘]
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'], [])`

- **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‘, ‚StringList-Wert2‘]

Fortsetzung auf der nächsten Seite

Tab. 6 – 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)`

- **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.

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“}}}‘
	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)`

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:

```
1  global tuple SurfaceModelID
2
3  if (RecognitionMeshFilename == '')
4      RecognitionMeshFilename := MeshFilename
5  endif
6
7  read_object_model_3d(RecognitionMeshFilename, 1.0,␣
   ↪ParamNames, ParamValues, ObjModel3D, Status)
8
9  create_surface_model(ObjModel3D, 0.03, [], [],␣
   ↪SurfaceModelID)
10
11 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
```

(continues on next page)

(continued from previous page)

```
//...

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, ... biete 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 ARC 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 Schnittstellenbeschreibung

Bei einem laufenden Mikado kann die ROS-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`**`(`*name*`)`

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. Es wird so verschoben, dass sich das Pattern an der Pose `defined_pose_transform` befindet. (Nur für Mikado 3D 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 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="adapter_script" pkg="meine_zelle" type=
↪"ProfinetMikadoAdapter.py" output="screen" />
```

Hier wird unter dem frei wählbaren Namen[1] `adapter_script` unser Script mit dem Dateinamen `ProfinetMikadoAdapter.py` aus der Zelle `meine_zelle` gestartet.

[1] zulässig sind Kleinbuchstaben, Zahlen und Unterstriche. Der Namen dient aber auch nur dazu, das Programm ggf zu erkennen.

KAPITEL 9

Copyright

© Copyright isys vision GmbH, 2015-2020 – Alle Rechte vorbehalten. – Nachdruck, auch auszugsweise, in jedweder Form sowie mechanische oder elektronische Vervielfältigung nur mit ausdrücklicher schriftlicher Genehmigung der isys vision GmbH. isys vision, ist ein Markenzeichen oder eingetragenes Markenzeichen von isys vision GmbH. Andere hier genannte Markennamen sind Eigentum der jeweiligen Markeninhaber.

Mikado ARC 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

isys vision 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 ARC 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:

9.1 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.

9.2 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 (IN-

CLUDING, 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.