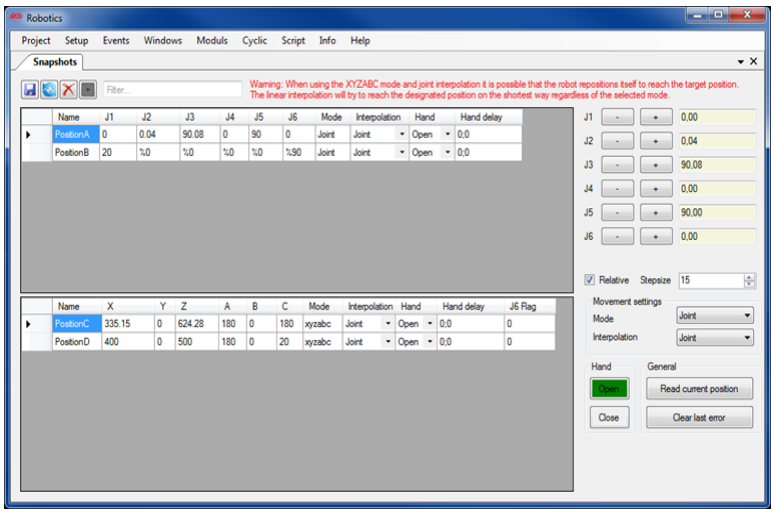
Bedienungsanleitung

Toolmonitor Robotics



GET IN touch
WITH SENSITIVE TESTING

Softline

Modline

Conline

Boardline

Avidline

Pixline

Applikation

MCD Elektronik GmbH
Hoheneichstr. 52
75217 Birkenfeld
Telefon 0 72 31/78 405-0
Telefax 0 72 31/78 405-10
info@mcd-elektronik.de
www.mcd-elektronik.de
Sitz: Birkenfeld
Geschäftsführer: Bruno Hörter
Registergericht Mannheim
HRB 505692

Vorlagenversion: 5.4 / 2018-04-09
V1.5 2018-09-03 BM (JJ)

Inhalt

1. ALLGEMEIN 4

2. INSTALLATION VON SOFTWARE, TREIBERN 4

2.1. SOFTWARE UND TREIBER INSTALLIEREN 4

2.2. STARTEN DER INSTALLIERTEN SOFTWARE 4

3. EINFÜHRUNG IN DIE BEDIENUNG 5

3.1. KURZEINFÜHRUNG 5

4. HANDBUCH SOFTWARE 7

4.1. STEUERUNGS - OBERFLÄCHE 7

4.1.1. Homing 7

4.1.2. Abbrechen 7

4.2. AKTUALISIERUNG DER OBERFLÄCHE 8

4.3. MODULE 8

4.3.1. Positionierung 8

4.3.2. Kugelgelenk 10

4.3.3. Rotationsgelenk 11

4.3.4. Torsionsgelenk 12

4.3.5. Lineargelenk 13

4.3.6. Plattform 14

4.3.7. Endeffektor / Greifarm 15

4.4. SNAPSHOT - OBERFLÄCHE 15

4.4.1. Variante 1 15

4.4.2. Variante 2 16

4.5. SETZEN UND AUSLESEN VON SIGNALEN 16

4.5.1. Setzen von Signalen 17

4.5.2. Auslesen von Signalen 17

4.5.3. Zurücksetzen von Notaus - Meldungen 17

5. EXIT - /STATUS - CODES 18

5.1. ENDEFFEKTOR / GREIFER STATUS - CODES 19

6. ROBOTER-TYPEN 20

6.1. NANOTEC PLATTFORM 20

6.2. MITSUBISHI MELFA RV 4FLM D 21

6.2.1. Compliance - Modus 22

6.2.2. Speed 23

6.2.3. Sicherheitswarnungen 24

6.2.4. Inbetriebnahme 25

6.2.5. Arbeiten mit Snapshots 28

6.2.6. Input / Output Signale 31

6.2.7. Kollisions - Erkennung 31

6.2.8. Verwendung eines Greifers 32

6.3. RWDRHANDLER 36

7. OPTIONEN 37

7.1. CONTROLLING - OPTIONEN 37

7.2. MODUL - OPTIONEN 37

1. Allgemein

Beim MCD Toolmonitor Robotics handelt es sich um eine Software, die es ermöglicht verschiedene Roboter-Typen zu steuern. Dabei können Achsen manuell über die Oberfläche bedient werden, sowie Positionen und Zustände gespeichert und wieder angefahren werden.

Der Toolmonitor unterstützt eine Vielzahl verschiedenster Elemente:

- Kugelgelenke
- Rotationsgelenke
- Torsionsgelenke
- Lineargelenke
- Plattformen
- Greifarme

Bestellnummer: # 150225

2. Installation von Software, Treibern

2.1. Software und Treiber installieren

Zum aktuellen Zeitpunkt ist kein Installer verfügbar. Das Ablegen der Anwendung in einen beliebigen Ordner ist daher ausreichend. Treiber sind nur dann notwendig, wenn ein entsprechend angeschlossener Roboter diese benötigt.

2.2. Starten der installierten Software

Die Software kann durch Ausführen der *RoboticsMonitor.exe* gestartet werden.

3. Einführung in die Bedienung

3.1. Kurzeinführung

Wird die Anwendung das erste Mal gestartet, ist zunächst kein Roboter - Typ eingestellt.

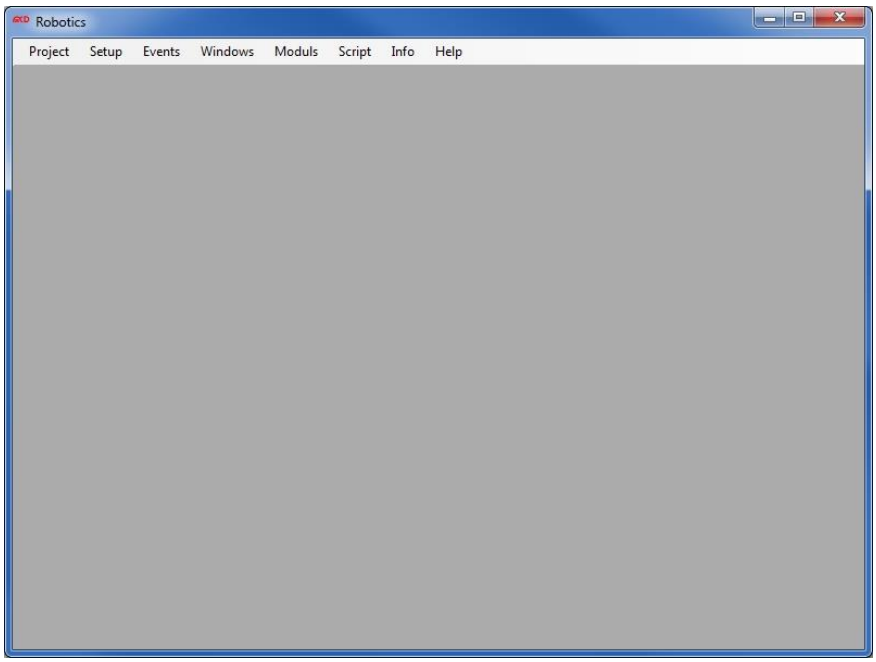


Abbildung 1: Start - Menü des Toolmonitor Robotics

Über das Options - Menü *Setup → Options* kann bspw. der *ExampleRobot* eingestellt werden. Dieser simuliert verschiedene Funktionen des Toolmonitors.

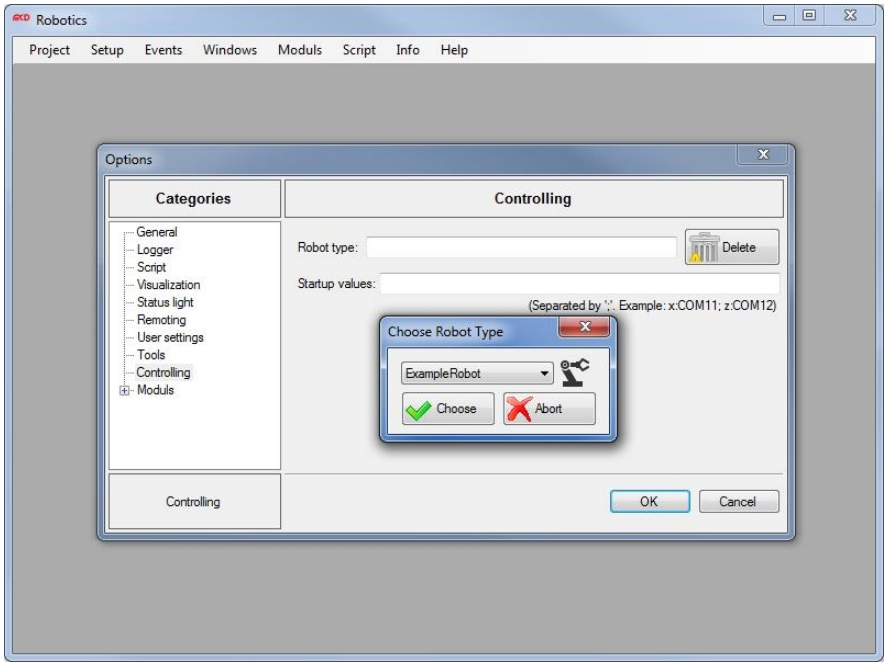


Abbildung 2: Auswahl des Roboter - Typs

Nachdem der Roboter - Typ gewählt wurde, können nun über den Menü - Punkt „*Windows*“ die Steuerungs - und Snapshot - Oberfläche angezeigt werden.

Abbildung 3: Anzeigen der Steuerungs - und Snapshot - Oberfläche

Über „*Moduls*“ kann auf die einzelnen Module des Roboters zugegriffen werden. Hier sind alle Gelenke, Plattformen und Endeffektoren aufgelistet, sofern diese vorhanden sind.

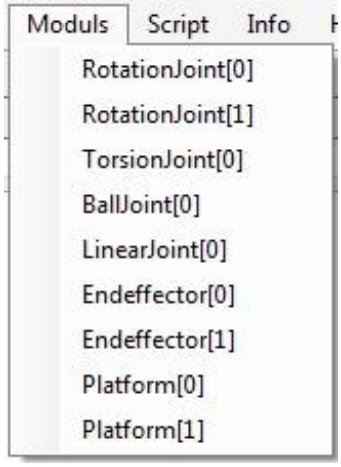


Abbildung 4: Module des Roboters

Beispielsweise kann das erste Rotationsgelenk angezeigt werden (*RotationJoint[0]*).

Im Beispielroboter ist es möglich bei diesem Gelenk den Winkel zu setzen und Positionen manuell „anzufahren“. Dies geschieht hierbei nur virtuell, da keine Hardware angeschlossen ist.

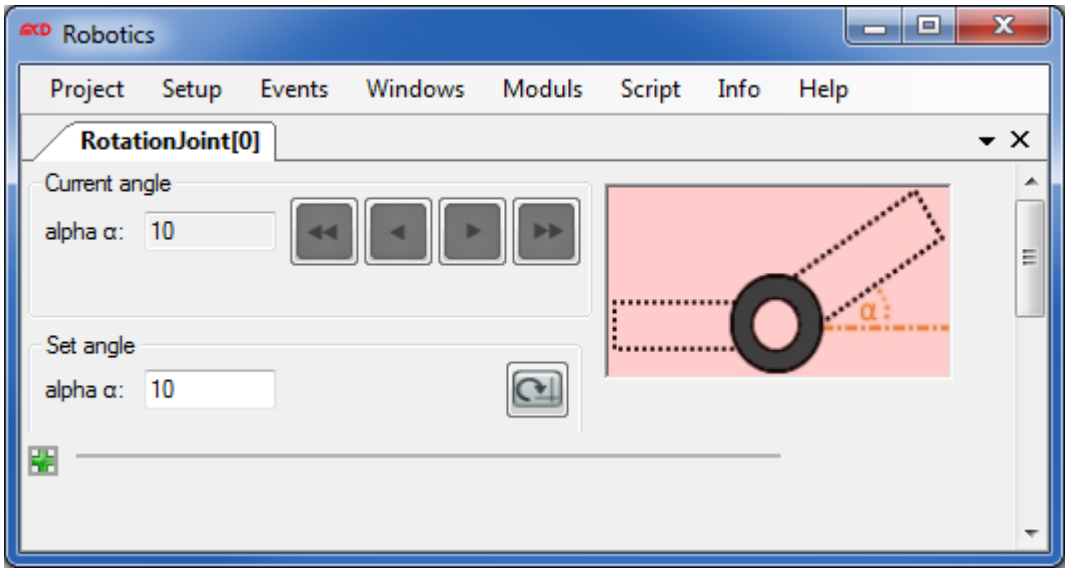


Abbildung 5: Auswahl des ersten Rotationsgelenks

4. Handbuch Software

Der MCD Toolmonitor Robotics ermöglicht es verschiedene Robotertypen zu steuern. Dabei wird die Oberfläche bei jedem eingestellten Roboter gleich aufgebaut. Gesteuert werden können alle Gelenke, Plattformen und Endeffektoren, die für einen Robotertypen vorgesehen wurden.

4.1. Steuerungs - Oberfläche

Über die Steuerungs - Oberfläche sind grundsätzliche Funktionen möglich. Darunter das Homing, Abbrechen von Aktionen und Einstellen des Aktualisierungs - Intervalls der Oberfläche. Weiterhin zeigt diese Oberfläche Informationen über den ausgewählten Roboter an.

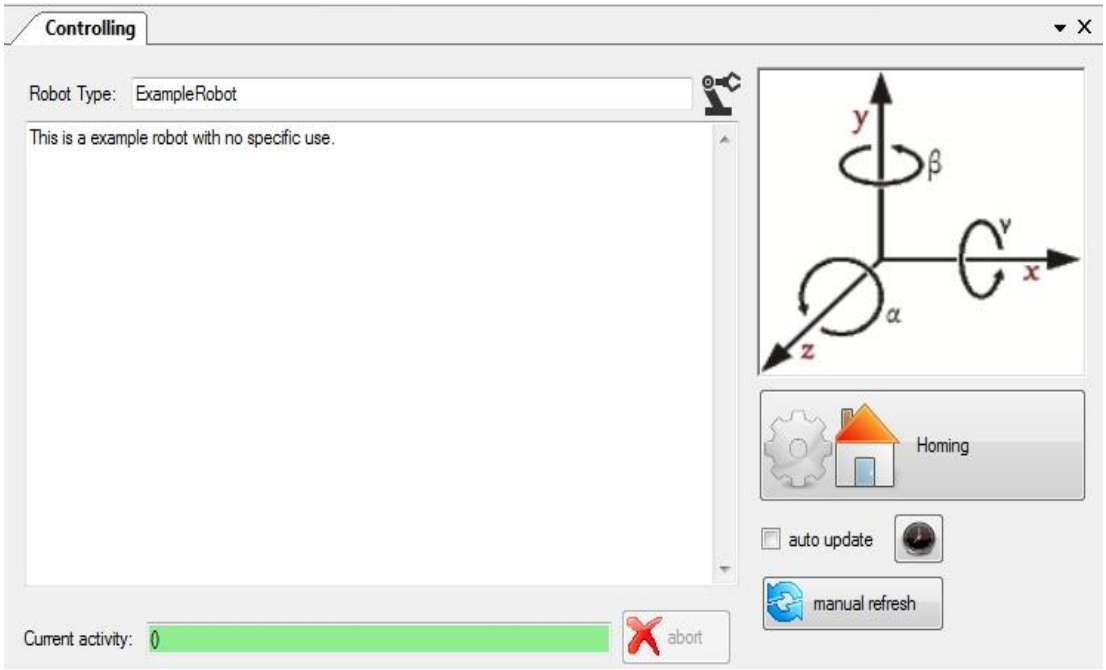


Abbildung 6: Steuerungs - Oberfläche

4.1.1. Homing

Das „Homing“ stellt für einige Roboter eine Voraussetzung für weitere Operationen dar. Beim Homing findet eine Referenzierung der Position statt. Dies geschieht, indem der Roboter im Regelfall durch die Mithilfe von Endschaltern in seinen Startzustand gefahren wird.

Abbildung 7: Homing - Button

4.1.2. Abbrechen

Über den Button *„Abort“* kann die aktuell ausgeführte Operation des Roboters abgebrochen werden. Wann und wie der Roboter die Aktion abbricht, hängt von der jeweiligen Implementierung des Roboters ab. Daher sollte diese Taste nicht als Ersatz zu einem Notaus - Schalter dienen. Der Button ist nur dann aktiv, wenn der Roboter eine Aktion durchführt.

ACHTUNG: Der Abort - Button stellt keine gleichwertige Alternative zu einem Notaus - Schalter dar!

4.2. Aktualisierung der Oberfläche

Über den Menü - Punkt „Cyclic“ können die Angaben der Oberfläche aktualisiert werden. Ist das automatische Aktualisieren nicht ausgewählt, werden die Positionsveränderungen der Gelenke nicht live angezeigt. Das Aktualisierungsintervall kann in den Optionen des Toolmonitors eingestellt werden.

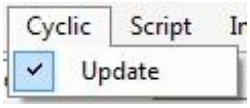


Abbildung 8: Aktualisierung der Oberfläche

ACHTUNG: Wird das automatische Aktualisieren deaktiviert, können die angezeigten Positionen in den einzelnen Modulen einen alten Zustand zeigen!

4.3. Module

Ein Roboter besteht jeweils aus einer Menge von Modulen. Welche Module eines Roboters gesteuert werden können, hängt von der jeweiligen Implementierung des Roboter - Typs ab.

4.3.1. Positionierung

Die Positionierung stellt kein eigenes Modul dar. Allerdings ist es Teil von nahezu allen verfügbaren Modulen.

Über diese Oberfläche können Positionen angefahren und gespeichert werden. Abgespeicherte Positionen können wieder angefahren werden.

Außerdem ist es möglich Positionen manuell anzufahren. Voraussetzung hierfür ist, dass der Roboter seine aktuelle Position ermitteln kann.

Grundsätzlich ist über die Positionierungs - Oberfläche das Einstellen der Position in X -, Y - und Z - Richtung möglich. Je nach Implementierung der einzelnen Module, werden einzelne Achsen ausgeblendet.

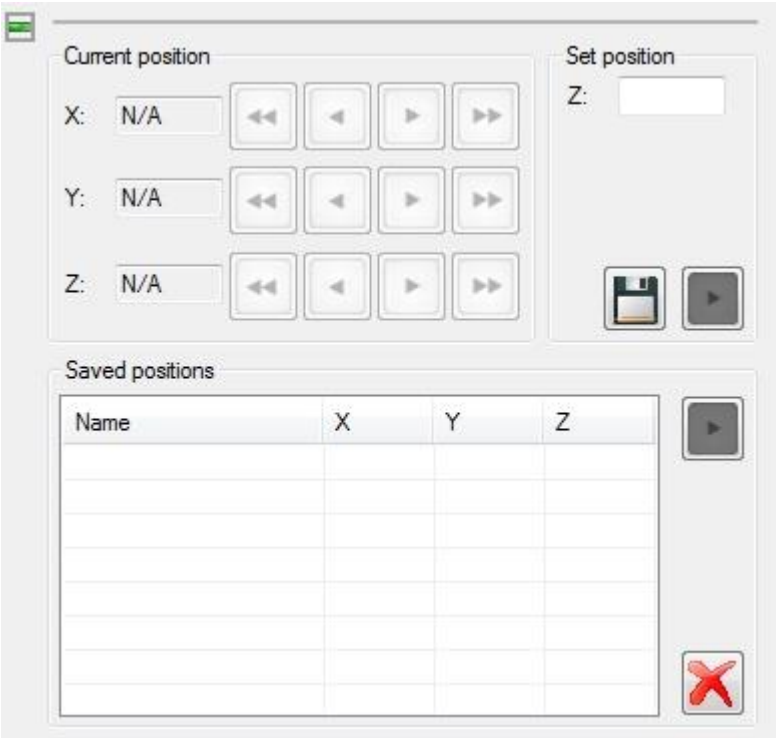


Abbildung 9: Positionierung - Oberfläche

Weiterhin ist es möglich beim Speichern spezielle Positionen einzulernen. Diese haben je nach Implementierung des Roboter - Typs unterschiedliche Bedeutungen. Diese Positionen sind in der Liste der gespeicherten Positionen gelb hinterlegt.

Folgende spezielle Positionen sind einlernbar:

- MiminumX: Legt i.d.R. den kleinsten anfahrbaren X - Wert fest
- MaximumX: Legt i.d.R. den größten anfahrbaren X - Wert fest
- MinimumY: Legt i.d.R. den kleinsten anfahrbaren Y - Wert fest
- MaximumY: Legt i.d.R. den größten anfahrbaren Y - Wert fest
- MinimumZ: Legt i.d.R. den kleinsten anfahrbaren Z - Wert fest
- MaximumZ: Legt i.d.R. den größten anfahrbaren Z - Wert fest
- SpecialPosition: Die Interpretation dieses Punktes hängt von der jeweiligen Implementierung ab.

HINWEIS: Die Maximal - und Minimalpositionen greifen nur, wenn der ausgewählte Roboter diese unterstützt!

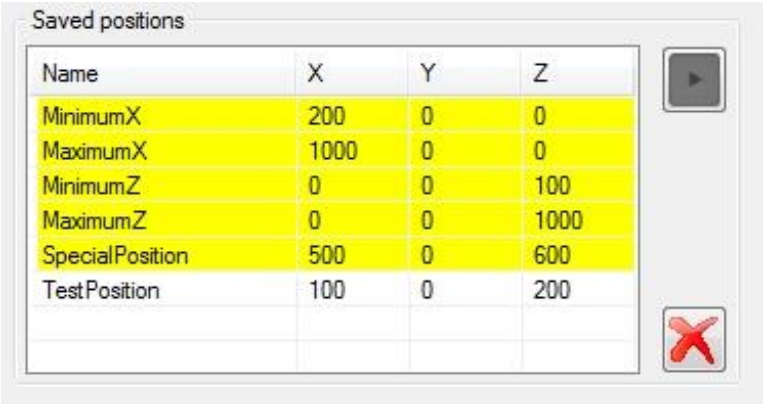


Abbildung 10: Gespeicherte Positionen

Die Schrittweiten der Vorwärts - / Rückwärts - Taster sind über das Options - Menü änderbar.

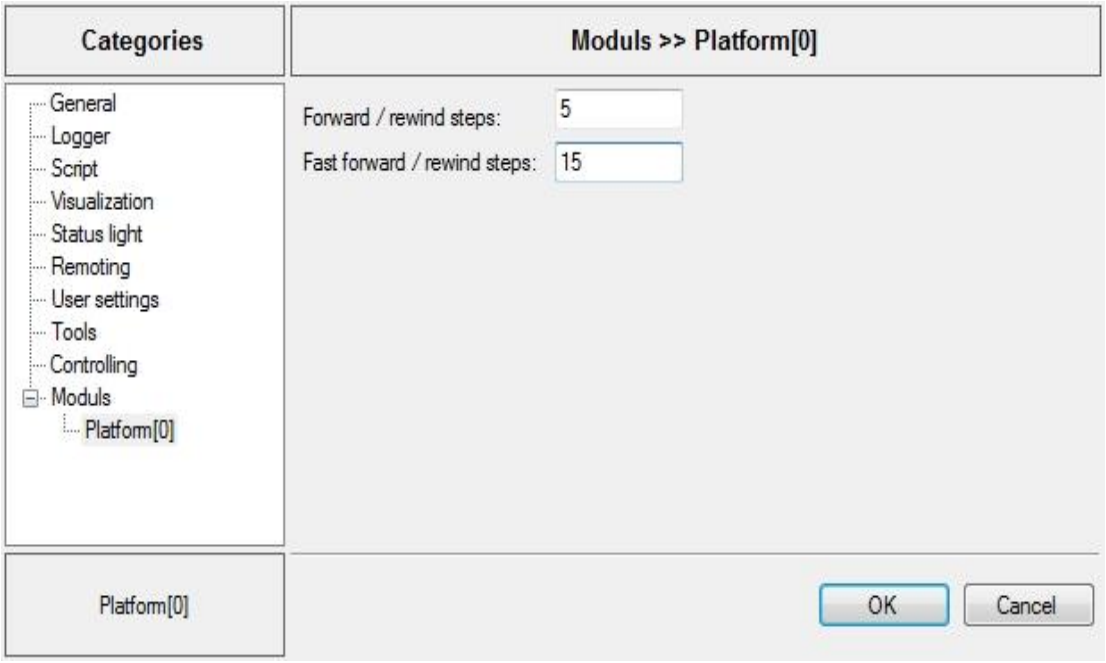


Abbildung 11: Einstellung der Schrittweite

4.3.2. Kugelgelenk

Das Kugelgelenk ist durch zwei Winkel verstellbar. Diese können direkt gesetzt oder manuell gesteuert werden. Weiterhin bildet die bereits beschriebene Positionierungs - Oberfläche einen Bestandteil dieses Moduls.

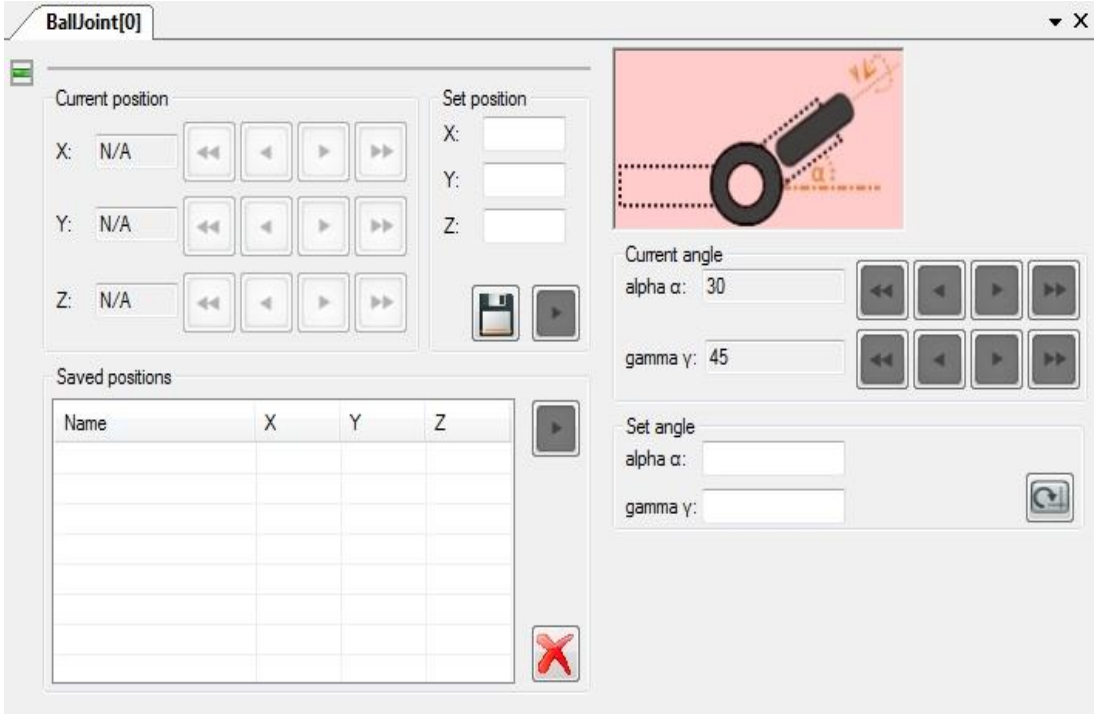


Abbildung 12: Positionierungs - Oberfläche mit Kugelgelenk

4.3.3. Rotationsgelenk

Das Rotationsgelenk ist nur durch einen Winkel verstellbar. Dieser ist manuell anfahrbar oder kann direkt gesetzt werden. Teil des Moduls ist außerdem die Positionierungs - Oberfläche.

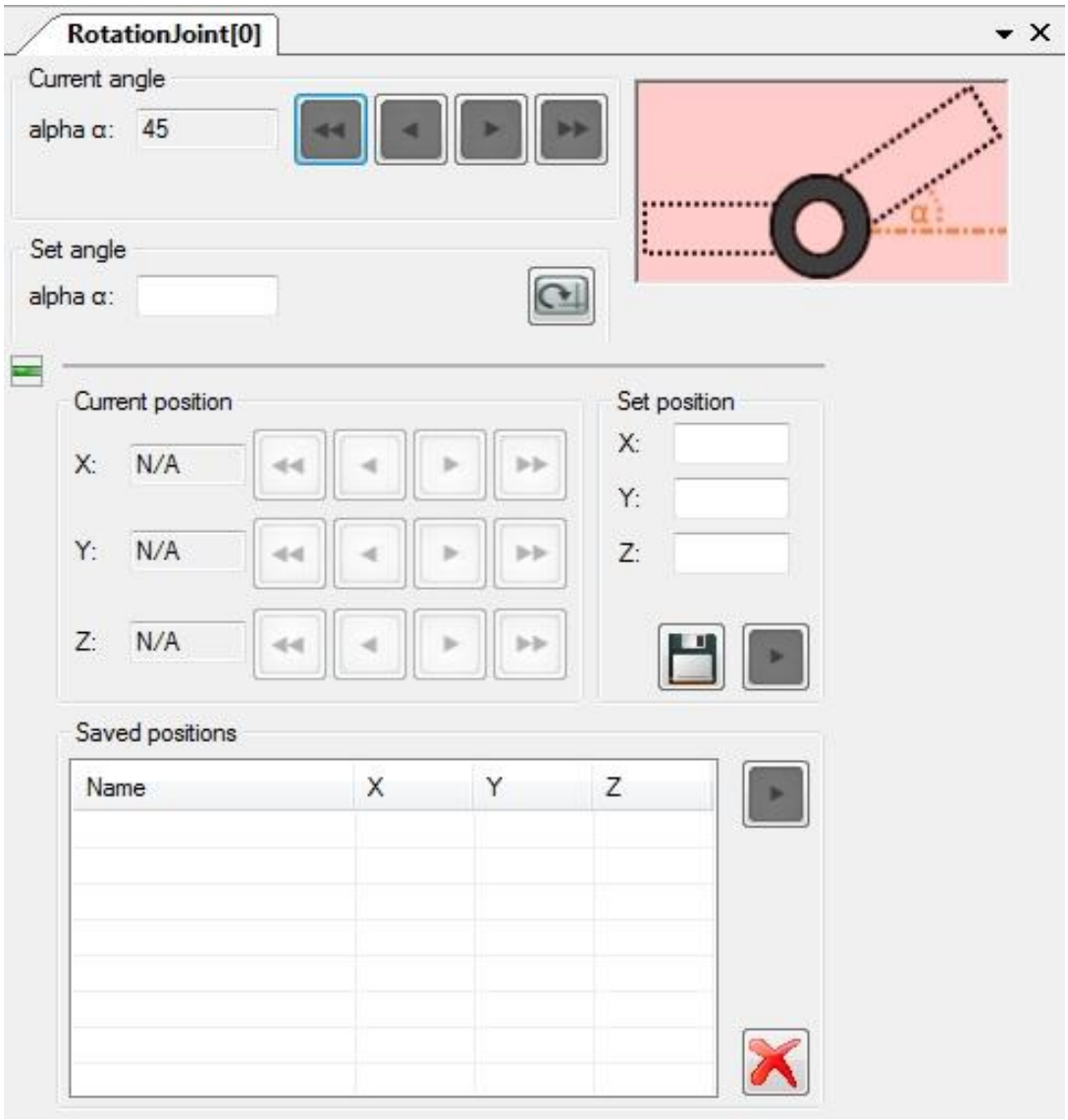


Abbildung 13: Positionierungs - Oberfläche mit Rotationsgelenk

4.3.4. Torsionsgelenk

Auch beim Torsionsgelenk kann nur ein Winkel eingestellt werden. Wie bereits bei den bisherigen Modulen beschrieben, ist auch hier die Positionierungs - Oberfläche Teil des Moduls.

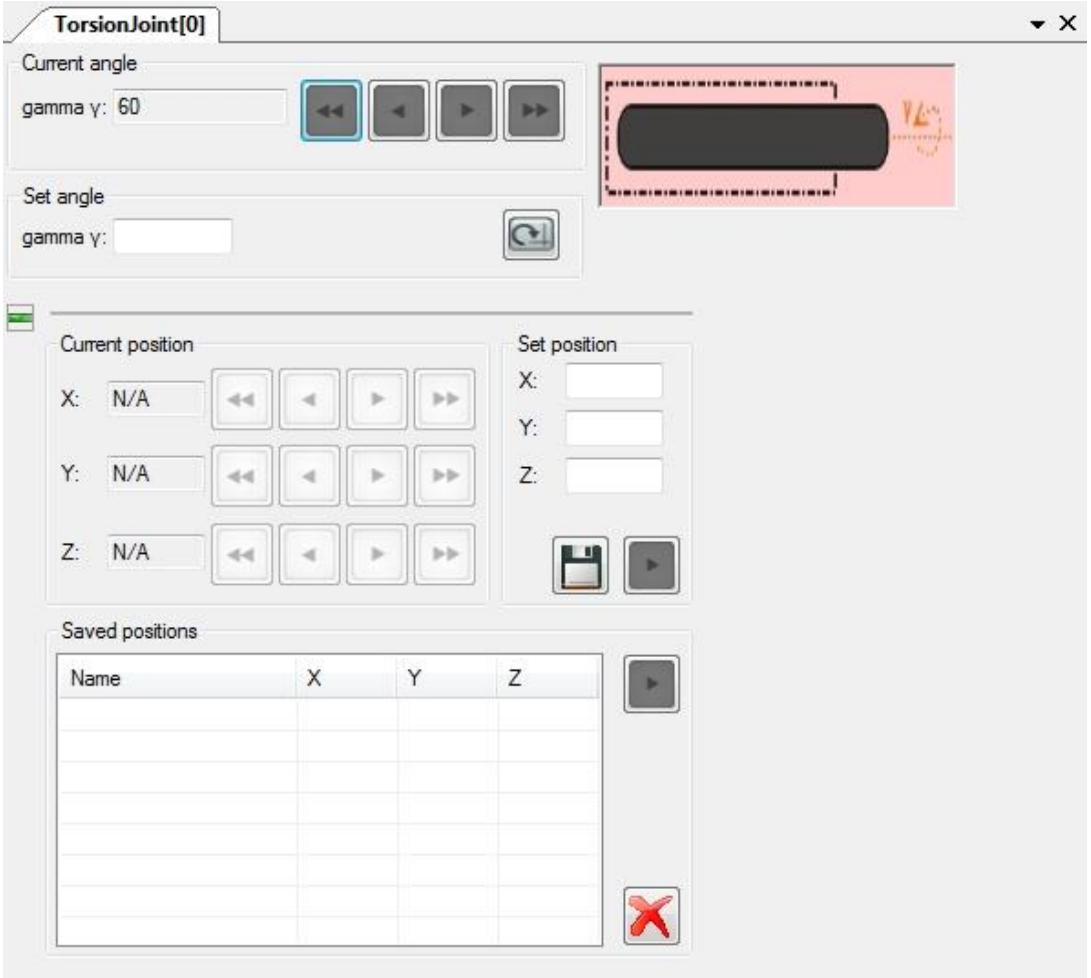


Abbildung 14: Positionierungs – Oberfläche mit Torsionsgelenk

4.3.5. Lineargelenk

Das Lineargelenk verfügt im Gegensatz zu den bisherigen Modulen nicht über einen Winkel. Stattdessen kann das Gelenk ausgefahren werden. Dabei ändert sich die lineare Position bzw. die Länge des Moduls.

Auch hier bildet die Positionierungs - Oberfläche wieder einen Teil des Moduls.

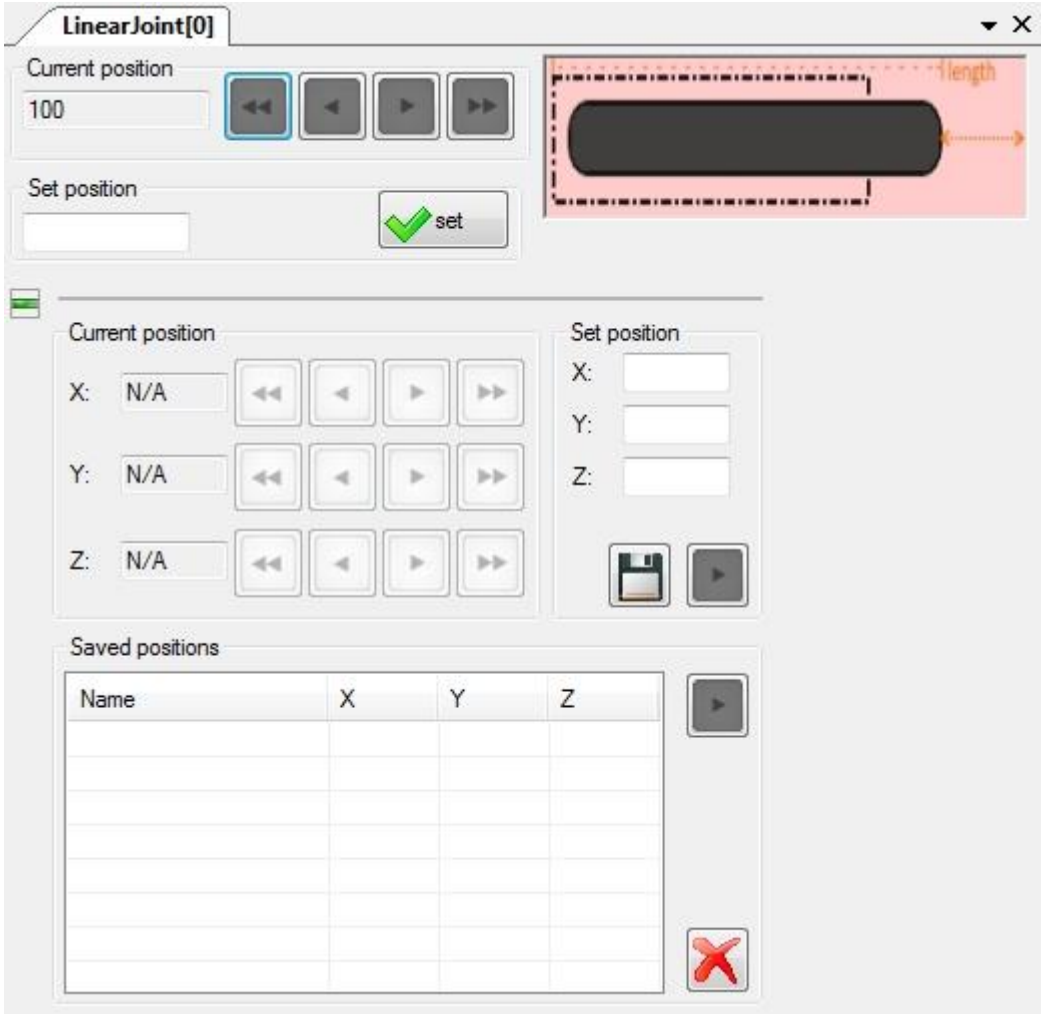


Abbildung 15: Positionierungs - Oberfläche mit Lineargelenk

4.3.6. Plattform

Die Plattform besteht ausschließlich aus der Positionierungs - Oberfläche. Welche Achsen - Positionen gesetzt werden können, hängt von der entsprechenden Implementierung des Roboter - Typs ab. Im hier abgebildeten Fall, kann die Plattform nur in Z - Richtung verfahren. Liefert der Roboter keine Information zur aktuellen Position zurück, ist das manuelle Verfahren aus Sicherheitsgründen deaktiviert.

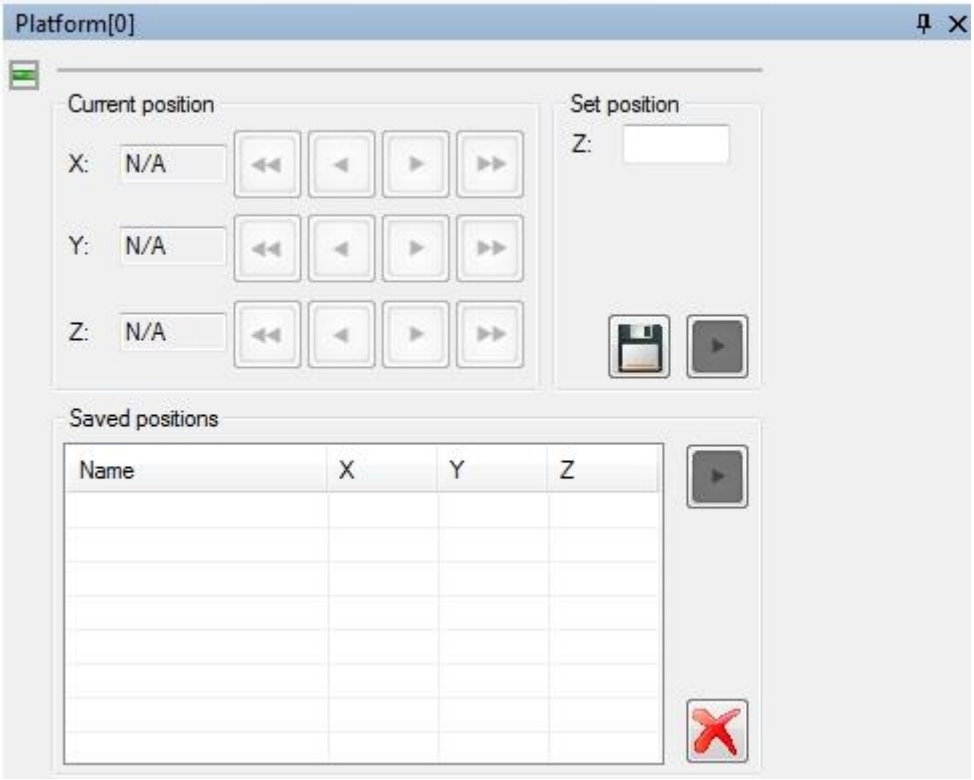


Abbildung 16: Plattform - Modul

4.3.7. Endeffektor / Greifarm

Der Endeffektor bildet i.d.R. das letzte Element einer kinematischen Kette. Der Toolmonitor Robotics implementiert hierbei einen Greifarm, der geöffnet und geschlossen werden kann. Wie schnell und wie weit sich der Greifer schließt, kann eingestellt werden, sofern der Roboter - Typ dies unterstützt. Weiterhin kann das Modul über die Positionierungs - Oberfläche gesteuert werden.

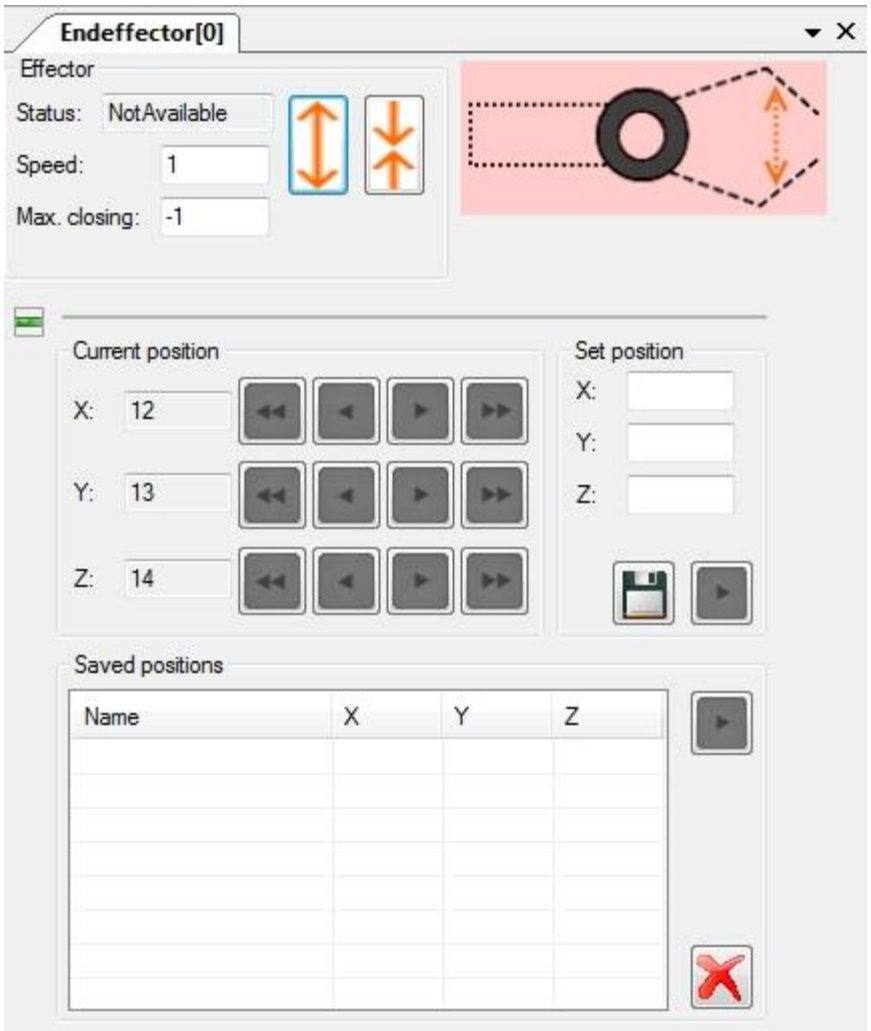


Abbildung 17: Positionierungs - Oberfläche mit Endeffektor / Greifarm

4.4. Snapshot - Oberfläche

Die Snapshot - Oberfläche ist in zwei Varianten im Toolmonitor enthalten. Bei 6 - Achs - Robotern ist in aller Regel die Variante 2 zu finden.

4.4.1. Variante 1

Über die Positionierungs - Oberflächen können für die einzelnen Module jeweils nur die Positionen gespeichert werden. Die Snapshot - Oberfläche ermöglicht darüber hinaus das Speichern der Positionen, Winkelstellungen, Zustände und Einstellungen aller Module. Gespeichert wird also der Zustand des gesamten Roboters und nicht nur einzelner Elemente. Dieser Zustand kann gespeichert und zu einem späteren Zeitpunkt wieder angefahren werden.

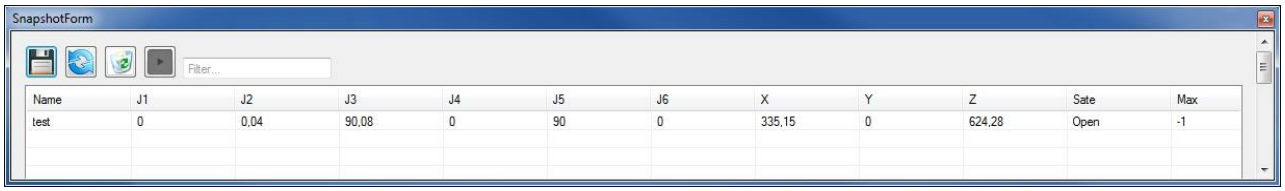


Abbildung 18: Snapshot Form (Variante 1)

4.4.2. Variante 2

Variante 2, auch Advanced Snapshots genannt, beinhaltet neben der Funktionalität zum Speichern und Anfahren von eingespeicherten Position zusätzlich die Möglichkeit die Achsen direkt zu verfahren, sowie die Hand zu Öffnen und zu Schließen.

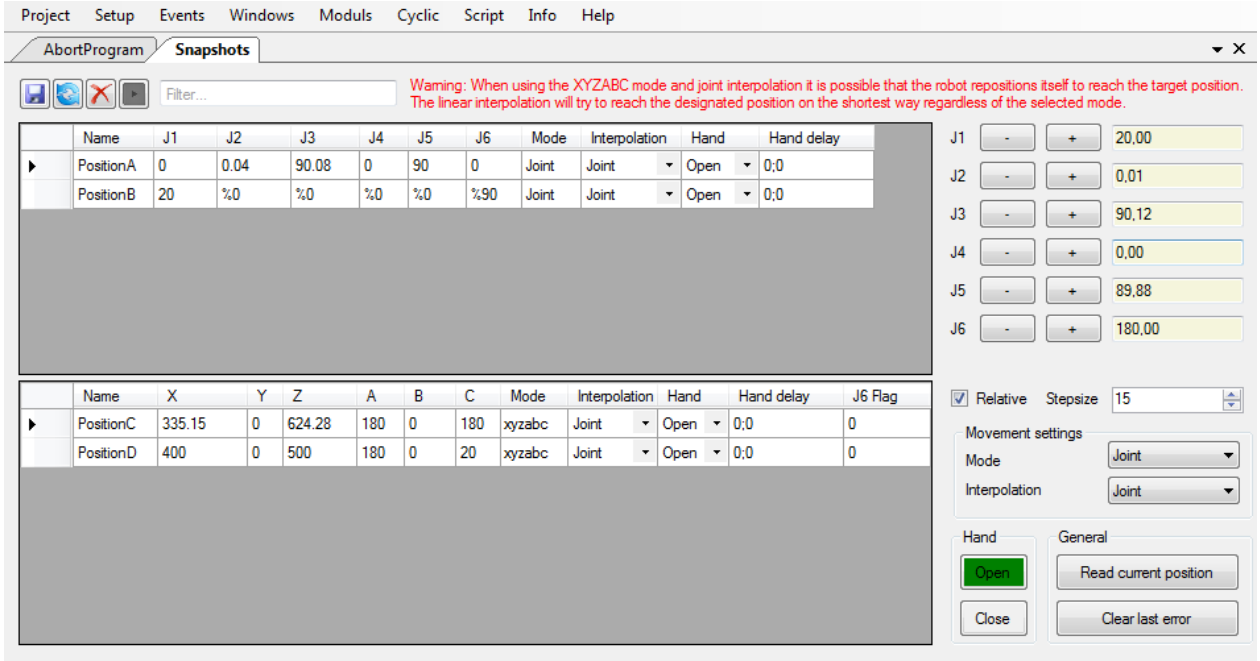


Abbildung 19: Advanced Snapshots (Variante 2)

Zusätzlich ermöglicht diese Snapshot - Variante die Möglichkeit die Art der Interpolation festzulegen. Außerdem ist es möglich festzulegen, ob die Positionen als XYZABC - Koordinaten gespeichert werden sollen oder die einzelnen Gelenkstellungen. Diese Einstellungen sind für jeden Snapshot getrennt möglich.

Werte von gespeicherten Snapshots können nachträglich durch Klicken des zu ändernden Feldes angepasst werden.

4.5. Setzen und Auslesen von Signalen

Je nach Roboter - Typ ist es möglich, Signale an den Roboter zu senden oder abzufragen. Dies geschieht über das „*Virtuelle Interface*“. Wie der Befehl interpretiert wird, hängt vom jeweiligen Roboter - Typ ab.

4.5.1. Setzen von Signalen

Gesetzt werden kann ein Signal über „*SetSignal*“.

Beispiel:

```
SetValue("RobotControl.SetSignal", 255);
```

4.5.2. Auslesen von Signalen

Um ein Signal auszulesen, wird der Befehl „*GetInputSignal*“ verwendet.

Beispiel:

```
int signal = GetValue("RobotControl.GetInputSignal");
```

Alternativ kann ein gezielter Pin abgefragt werden, falls dies vom Roboter - Typ unterstützt wird.

Beispiel:

```
int pin0 = GetValue("RobotControl.GetInputSignal.0");
int pin1 = GetValue("RobotControl.GetInputSignal.1");
```

4.5.3. Zurücksetzen von Notaus - Meldungen

Wurde der Notaus - Schalter betätigt, während der Toolmonitor eine Aktion ausgeführt hat, wird er bei manchen Robotern zunächst blockiert, bis er explizit wieder freigegeben wird.

Diese Freigabe erfolgt über die Funktion „*ResetEmergencyStop*“ im *Virtuellen Interface*.

```
SetEvent("RobotControl.ResetEmergencyStop");
```

5. Exit - /Status - Codes

Oftmals wird zu einer „*GetString*“ - Methode auch eine „*GetValue*“ - Methode angeboten, welche einen Status - Code zurückgibt.

So zum Beispiel die Funktionen:

- SnapshotControl.ProgramStatusCode
- Endeffector[0].EndeffectorStatusCode
- RobotControl.CurrentRobotStatusCode

Die Codes beziehen sich dabei auf folgende Meldungen:

Code	Name	Beschreibung
-1	NotAvailable	Die aufgerufene Funktion ist für den ausgewählten Roboter - Typ nicht verfügbar.
0	Successful	Die (letzte) Aktion wurde erfolgreich ausgeführt. **Oder**: Die Ausführung ist beendet.
1	Error	Es ist ein genereller, nicht weiter spezifizierbarer Fehler aufgetreten.
2	ConnectionFailed	Der Verbindungsaufbau zum Roboter ist fehlgeschlagen.
3	UNDEFINED	Es ist ein unerwarteter Fehler aufgetreten.
4	EmergencyStop	Der Notaus - Schalter wurde betätigt.
5	Aborted	Die (letzte) Aktion wurde abgebrochen.
6	MotorNotReady	Die Motoren des Roboters sind noch nicht initialisiert oder in einem Zustand, in welchem sie nicht gestartet werden können.
7	Homing_Required	Die Aktion kann erst ausgeführt werden, wenn zunächst ein Homing stattgefunden hat.
8	ExternProgramIsMissing	Ein externes Programm fehlt (bei Melfa bspw. ROBOTCONTROL auf Roboter - Controller).
9	CurrentPositionNotAvailable	Die Abfrage der aktuellen Position des Roboters ist fehlgeschlagen.
10	PositionOutOfReach	Die gewünschte Position kann nicht erreicht werden.
11	CAUTION_Could_not_set_speed	Die Geschwindigkeit des Roboters konnte nicht gesetzt werden. Hierbei gilt es vorsichtig zu sein, da ggf. die volle Geschwindigkeit gesetzt ist.
12	Timeout	Ein Timeout bei der Ausführung ist aufgetreten.
13	NotImplemented	Die Funktion wurde nicht implementiert (nur für Debugging - Zwecke bei Entwicklung).
14	AutomaticModeNotEnabled	Der Roboter befindet sich nicht im Automatic - Mode (Melfa).
15	EmergencyStop_or_DoorOpen	Der Roboter ist gesperrt. Dies kann u.a. durch Drücken des Notaus - Schalters oder durch Öffnen einer sicherheitsrelevanten Tür erfolgen.

16	Running	Der Roboter führt derzeit ein Snapshot - Programm aus.
17	WaitPosition	Der Roboter führt derzeit ein Snapshot - Programm aus und befindet sich in einer Warte - Position.
18	OpeningHandFailed	Die Hand des Roboters konnte nicht geöffnet werden.
19	ClosingHandFailed	Die Hand des Roboters konnte nicht geschlossen werden.
20	ExternInterrupt	Ein externer Interrupt ist aufgetreten.
21	InvalidInput	Für einen übergebenen Parameter wurden falsche Werte angegeben oder der erlaubte Wertebereich überschritten.

5.1. Endeffektor / Greifer Status - Codes

Wird der Status - Code eines Endeffektors angefragt, sind folgende Rückgabe - Codes möglich:

Code	Name	Beschreibung
-2	Undefined	Die Hand befindet sich in einem unbekannten Zustand. Dies ist bei manchen Robotern immer nach dem Starten der Fall.
-1	NotAvailable	Bei diesem Roboter kann der Status des ausgewählten Endeffektors nicht abgefragt werden.
0	Open	Der Greifer ist geöffnet.
1	Closed	Der Greifer ist geschlossen.
2 3 4 5	Closed0 Closed1 Closed2 Closed3	Roboter - spezifische Information darüber, wie weit die Hand geschlossen ist. Weitere Informationen hierzu finden sich in der Beschreibung des jeweiligen Roboters, falls diese Funktionalität implementiert sein sollte.

6. Roboter-Typen

Der Toolmonitor Robotics implementiert bereits verschiedene Roboter - Typen. Falls zu diesen weiterführende Erklärungen nötig sind, sind diese im Folgenden beschrieben.

6.1. NanoTec Plattform

Der Motor in Z - Richtung befindet sich an COM12. Der Motor in X - Richtung, durch welchen die Kamera an das Testobjekt herangefahren wird, an COM11.

Da sich diese Ports ändern könnten, ist es nötig, die COM - Schnittstellen über den Menüpunkt „*Options*“ anzugeben, wie hier im Bild gezeigt.

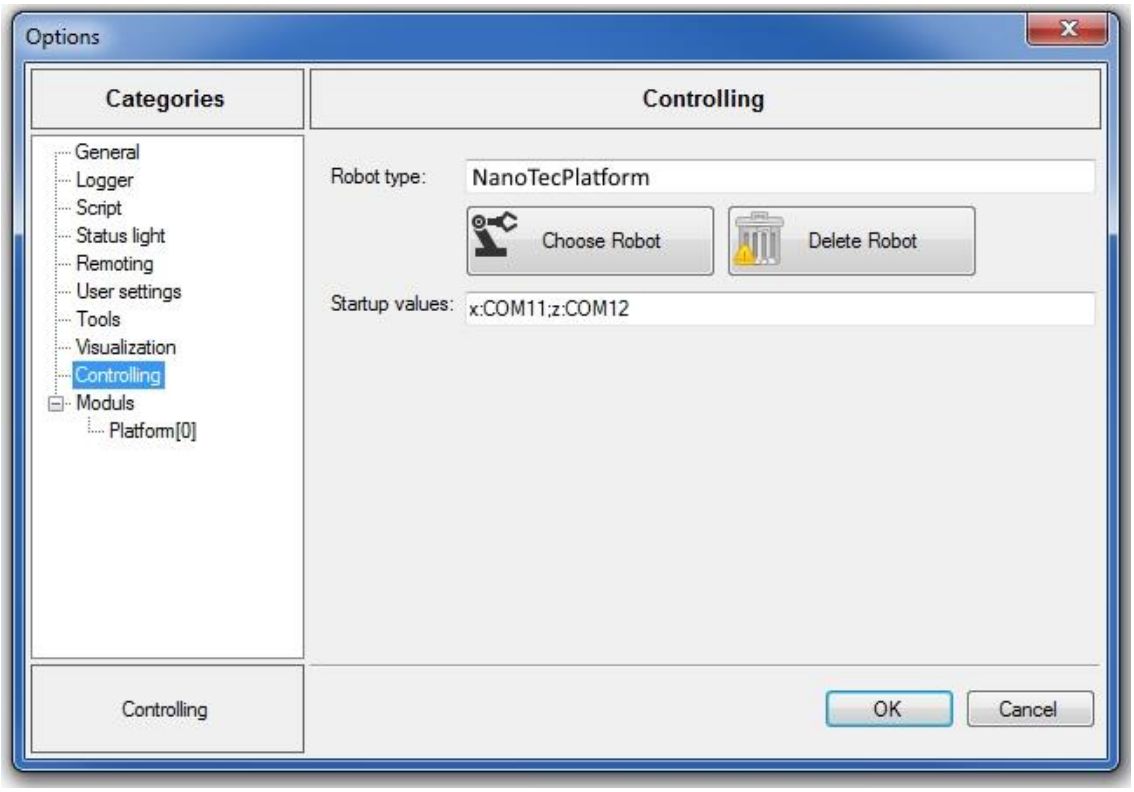


Abbildung 20: COM - Schnittstellen festlegen

6.2. Mitsubishi Melfa RV 4FLM D

Bei dem Mitsubishi - Roboter Melfa RV-4FLM-D handelt es sich um einen 6 – achsigen - Roboter.

Abbildung 21: Mitsubishi - Roboter Melfa RV 4FLM D

Das Steuern dieser Achsen, sowie angeschlossener Greifhände, wird durch den Toolmonitor Robotics ermöglicht. Dabei ist es möglich jede Achse einzeln anzusprechen, Snapshots einzulernen und diese anschließend wieder anzufahren.

Hierbei kann auch eine Sequenz von bis zu 10 Snapshots über das *Virtuelle Interface* angesteuert werden oder ein Snapshot - Programm eingespeichert werden. Dies wird im Kapitel 6.2.5 Arbeiten mit Snapshots genauer erklärt.

Neben Snapshots ist es auch möglich Endpositionen über die X -, Y - und Z - Achse zu speichern. Die Funktionalität zum Abspeichern von Endpositionen lassen sich über den Menü - Punkt „*Setup*“ deaktivieren.

ACHTUNG:

Wird eine Position angefahren, bleibt die Ausrichtung des Gelenks J5 gleich!

Wird eine Position angefahren, gespeichert und zu einem späteren Zeitpunkt wieder angefahren, muss das Gelenk J5 zunächst wieder in diese Ausrichtung gebracht werden!

Empfohlen wird daher die Verwendung von Snapshots.

WARNUNG:

Der Roboter merkt nicht, wenn er auf Widerstand stößt!

Vor dem Anfahren einer Position oder eines Snapshots muss sichergestellt werden, dass sich kein Gegenstand und keine Person im Weg befindet!

Auch im Compliance - Modus besteht hier Verletzungsgefahr und die Gefahr der Beschädigung von Bauteilen!

Über die Funktionen „*GetCurrentRobotStatus*“ und „*GetCurrentRobotStatusCode*“ kann der aktuelle Zustand des Roboters ausgelesen werden.

Letztere der beiden Funktionen gibt dabei einen Code zurück. Eine Auflistung von möglichen Codes findet sich unter „*Exit/Status Codes*“.

```
string robotStatus = GetString("RobotControl.CurrentRobotStatus");

int robotStatusCode = GetValue("GetValue.CurrentRobotStatusCode");
```

6.2.1. Compliance - Modus

Durch den Compliance - Modus werden die Gelenke des Roboters weich und passen sich so bei der Positionierung der Umgebung etwas an.

ACHTUNG: Hierbei sollte zunächst sichergestellt werden, ob der Gegenstand bei Gegendruck nicht beschädigt wird.

Der Compliance - Modus kann über das *Virtuelle Interface* an - und ausgeschaltet werden.

HINWEIS: Der Compliance - Mode wird aktiv / inaktiv bei der nächsten Positionierung.

```
// Compliance-Modus anschalten

 SetValue("RobotControl.SetValues", new []{"Compliance=1"});


// Compliance-Modus ausschalten

SetValue("RobotControl.SetValues", new []{"Compliance=0"});
```

6.2.2. Speed

Die Geschwindigkeit des Roboters kann über zwei Wege eingestellt werden. Zum einen kann die Geschwindigkeit über das *Setup* festgelegt werden. Diese eingestellte Geschwindigkeit ist solange gültig, bis der Wert durch das *Virtuelle Interface* geändert wird.

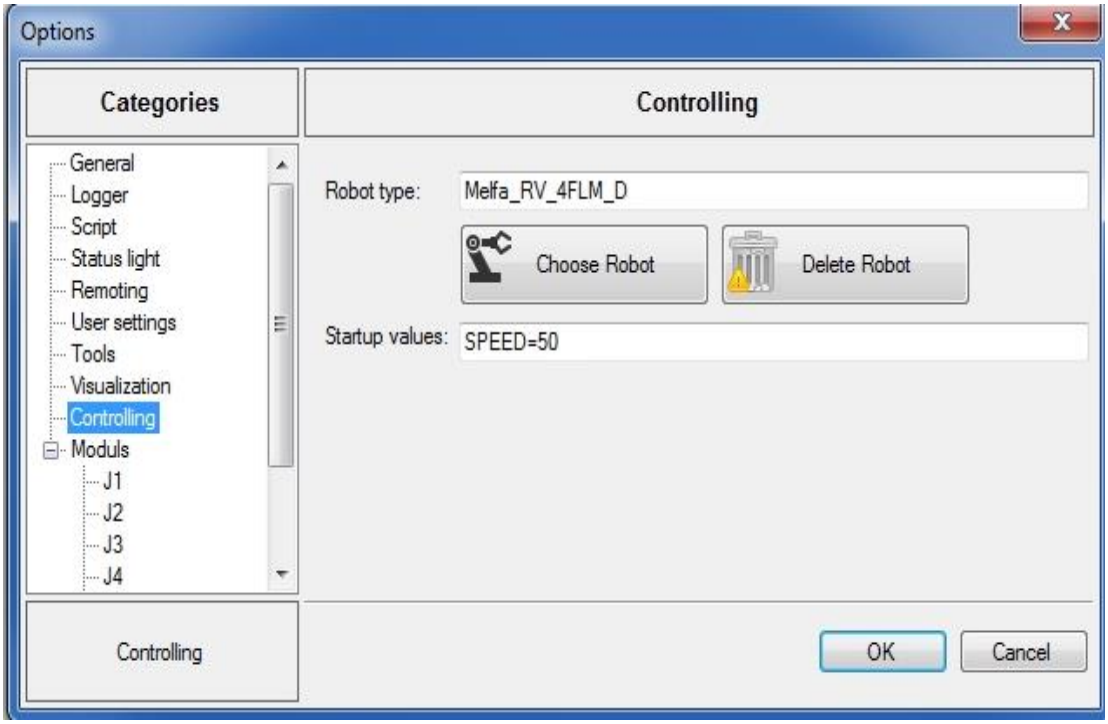


Abbildung 22: Anpassung der Geschwindigkeit

Die Geschwindigkeit über das *Virtuelle Interface* einzustellen stellt die zweite Möglichkeit dar, wie im folgenden Beispiel gezeigt wird:

```
// Setzt die Geschwindigkeit auf 10%
SetValue("RobotControl.SetValues", new []{"Speed=10"});

// Setzt die Geschwindigkeit auf 5%
SetValue("RobotControl.SetValues", new []{"Speed=5"});
```

ACHTUNG:
Stellen Sie im Einlern - und Testbetrieb die Geschwindigkeit niemals auf > 10%, um die Sicherheit von Personen und Gegenständen nicht zu gefährden!

Es ist möglich die Geschwindigkeit auf 100% zu stellen. Prüfen Sie daher vor dem Ausführen jeder Aktion, wie die Geschwindigkeit des Roboters eingestellt ist.

Stellen Sie sicher, dass sich beim Steuern des Roboters keine Personen in Reichweite des Roboters befinden und achten Sie auf umliegende Gegenstände!

6.2.3. Sicherheitswarnungen

Gefährdung für Personen

Stellen Sie zu jedem Zeitpunkt während des Betriebs des Roboters sicher, dass keine Personen zu Schaden kommen. Es dürfen sich keine Personen in Reichweite des Roboters aufhalten, wenn dieser im Betrieb ist.

Sollten Wartungsarbeiten am Roboter nötig sein, schalten Sie diesen ab.

Es muss sichergestellt werden, dass keine Personen im laufenden Betrieb die Zelle des Roboters betreten können oder durch Unwissenheit in Gefahr geraten.

Gefährdung für Gegenstände

Stellen Sie sicher, dass während dem Betrieb keine Gegenstände durch den Roboter zu Schaden kommen oder durch den Raum geworfen werden können.

Überprüfen Sie vor dem Anfahren einer Position oder eines Snapshots, dass sich im Fahrweg keine Gegenstände befinden, welche bei der Umpositionierung gestriffen oder getroffen und damit beschädigt werden können.

Gefährdung im Compliance - Mode

Testen Sie den Compliance - Mode ausgiebig, bevor Sie ihn im laufenden Betrieb einsetzen.

Stellen Sie sicher, dass Gegenstände auch trotz der Weichstellung der Gelenke nicht zerdrückt werden und so zu Schaden kommen.

Die Gelenke sind im Compliance - Mode zwar weicher, geben aber erst bei Gegendruck nach. Gelenke lassen sich im Compliance - Mode nicht beliebig verschieben. Weicht die Position mehr als 2 cm von der ursprünglichen Position ab, stoppt der Roboter.

Gefährdung durch Geschwindigkeit

Die Geschwindigkeit des Roboters kann durch den Toolmonitor auf 100% gestellt werden. Stellen Sie bei jeder Verwendung des Roboters sicher, dass die korrekte Geschwindigkeit eingestellt ist.

Im Test - und Einlernbetrieb sollte die Geschwindigkeit nicht größer als 10% sein!

Es sollte sichergestellt werden, dass der Roboter gut befestigt ist und auch bei hohen Geschwindigkeiten in seiner Verankerung bleibt.

Weitere Sicherheitshinweise und Richtlinien

Dies ist keine vollständige Liste von Sicherheitshinweisen und Sicherheitsrichtlinien.

Weitere Informationen können den Unterlagen entnommen werden, welche dem Roboter beiliegen oder auf der Website von Mitsubishi angefragt werden.

6.2.4. Inbetriebnahme

Es wird empfohlen zunächst die Kollisions - Erkennung des Roboters zu aktivieren, wenn diese verwendet werden soll. Beschreibungen hierzu finden sich im Kapitel 6.2.7. Kollisions - Erkennung, weitere Informationen können in der Anleitung des Roboters gefunden werden.

Weiterhin wird empfohlen, virtuelle Wände zu definieren, welche der Roboter nicht durchfahren kann.

RT Toolbox 2

Um den Toolmonitor Robotics zusammen mit einem Mitsubishi Roboter benutzen zu können, muss die RT Toolbox 2 installiert sein. Diese Mitsubishi Software ermöglicht es Programme auf den Roboter zu übertragen, diesen zu warten und manuell zu steuern.

Weiterhin werden durch die Installation die nötigen Bibliotheken bereitgestellt, durch welche der Toolmonitor mit dem Roboter kommuniziert.

Netzwerkkonfiguration

Um mit dem Roboter kommunizieren zu können, muss dieser mit dem Computer verbunden werden, auf dem der Toolmonitor installiert ist. Die Verbindung wird dabei über TCP/IP hergestellt.

Um eine Kommunikation aufzubauen, müssen hierbei die IP - Optionen des jeweiligen Ethernet - Ports am PC auf folgende Einstellungen geändert werden.

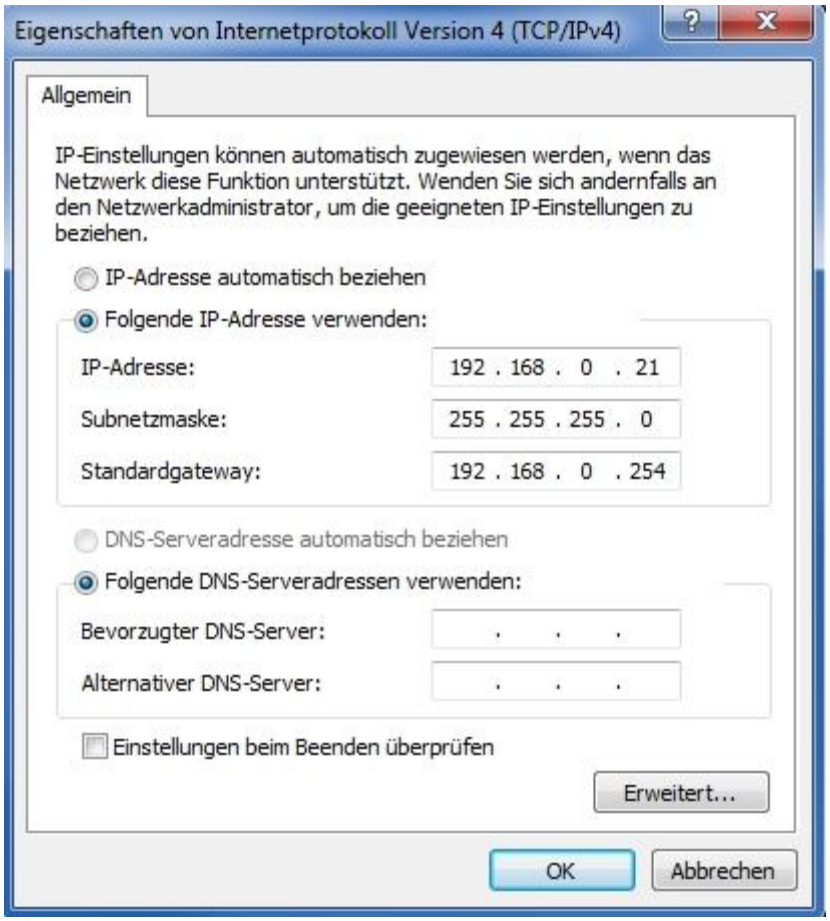


Abbildung 23: Netzwerkkonfiguration

Anschließend muss die Anwendung *RoboCom.exe*, welche zusammen mit dem Toolmonitor geöffnet wird, eingestellt werden.

Hierbei muss die Kommunikationsart zunächst auf TCP/IP gestellt werden und anschließend die IP *192.168.0.20* des Roboters angegeben werden.

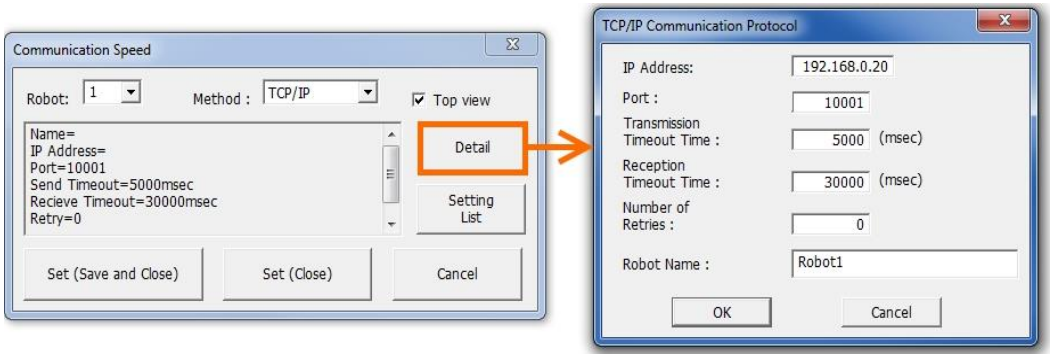


Abbildung 24: Einstellen der Kommunikationsart

Automatic Mode aktivieren

Damit der Toolmonitor schreibenden Zugriff auf den Controller des Roboters erhält, muss sich dieser im Automatic - Mode befinden.

Dieser Modus kann durch einen Schlüssel an der Front des Controllers an - und ausgeschaltet werden.

ACHTUNG:
Wurde im manuellen Teaching - Betrieb die Geschwindigkeit auf 100% gestellt, bleibt diese im Automatic - Mode erhalten, bis die Geschwindigkeit durch den Toolmonitor oder den Anwender wieder geändert wird.

Steuerprogramm auf Roboter

Damit der Toolmonitor mit dem Roboter interagieren kann, muss auf den Roboter vor der Inbetriebnahme ein Programm übertragen werden. Dies geschieht, indem über die Software RT Toolbox 2 eine Verbindung mit dem Roboter hergestellt und das Programm auf den Roboter aufgespielt wird.

Damit der Toolmonitor auf das Programm zugreifen kann, muss diesen mit dem Namen **ROBOTCONTROL** gespeichert werden.

Der Programmcode wird im Folgenden angefügt.

```
If M_Err Then Reset Err
 Servo On
 If P2.X <> 0.0 Then
     CmpG 0.2, 0.2, 0.2, 0.2, 0.2, 0.2,,
     Cmp Pos, &B111111
 Else
     Cmp Off
 EndIf

 If P1.X <> 0.0 Or P1.Y <> 0.0 Or P1.Z <> 0.0 Then
     P4.X = P1.X - P_Curr.X
```

```
    P4.Y = P1.Y - P_Curr.Y
    P4.Z = P1.Z - P_Curr.Z

    P4.A = 0
    P4.B = 0
    P4.C = 0

    P3 = P_Curr + P4

    If P3.A = -180 Then
        P3.C = 180
    ElseIf P3.A = 180 Then
        P3.C = 0
    EndIf

    Fine 0.02, P
    Mov P3 Type 0,0

Else
    Fine 0.02, J
    Mov JP1
    Mov JP2
    Mov JP3
    Mov JP4
    Mov JP5
    Mov JP6
    Mov JP7
    Mov JP8
    Mov JP9
    Mov JP10
EndIf

End
```

Weiterhin ist es nötig, dass folgendes Abbruch - Programm mit dem Namen **ABORTPRGM** auf dem Controller vorhanden ist.

```
IF M_Err THEN Reset Err
Servo On
Fine 0.001, J
HOpen 1
HOpen 2
HOpen 3
HOpen 4
PC = P_Curr
PC.Z = PC.Z + 30
MOV PC
MOV JP1
```

6.2.5. Arbeiten mit Snapshots

Snapshots können beim Melfa RV-4FLM-D Roboter in verschiedenen Varianten verwendet werden.

So ist es möglich:

- einen einfachen Snapshot anzufahren
- eine Sequenz von Snapshots anzufahren (bis zu 10 Snapshots)
- ein Snapshot - Programm einzuspeichern (Anzahl nur durch Programmspeicher begrenzt)

Wird eine Sequenz von Snapshots verwendet, anstatt jeden Snapshot einzeln nacheinander anzufahren, entfällt die Zeit zum Umladen des Programmspeichers zwischen den Snapshots. Hierdurch kann viel Zeit eingespart werden.

Wird mit einem Snapshot - Programm gearbeitet, ist es möglich Pausen zwischen Snapshots festzulegen, in denen auf ein externes Signal gewartet wird. Die einzelnen Varianten werden im Nachfolgenden genauer erklärt.

Einen einfachen Snapshot anfahren

Einzelne Snapshots können über die Benutzeroberfläche der Snapshots, sowie über das *Virtuelle Interface* angefahren werden. Über das *Virtuelle Interface* sieht ein solcher Befehl bspw. wie folgt aus:

```
SetValue("SnapshotControl.SetSnapshot", "NameDesSnapshots");
```

Sollen mehrere Snapshots nacheinander angefahren werden, müsste jeder einzeln aufgerufen werden. Da vor dem Anfahren eines Snapshots dieser in den Programmspeicher des Roboters geladen werden muss, geht hier allerdings Zeit verloren. Hierfür bietet es sich dann an eine Sequenz von Snapshots anzufahren.

Sequenz von Snapshots anfahren

Um mehrere Snapshots nacheinander anfahren zu können, ohne durch das Umladen des Programmspeichers Zeit zu verlieren, bietet es sich an mit Snapshot - Sequenzen zu arbeiten. Dabei ist es möglich bis zu 10 Snapshots direkt zu übergeben, welche dann ohne Pause direkt abgefahren werden.

Beispiel:

```
SetValue("SnapshotControl.SetSnapshots", "Snapshot1;Snapshot2;Snapshot3;Snapshot4;");
```

Snapshot - Programm einspeichern und ausführen

Wie die Snapshot - Sequenz, bietet auch das Snapshot - Programm die Möglichkeit die Ladezeit des Programmspeichers zwischen Snapshots zu entfernen. Weiterhin bieten es die Möglichkeit auf ein externes Signal zu warten. Dadurch ist es möglich Sequenzen von Snapshots einzulernen, welches durch ein externes Signal an Pin 6 getaktet werden können.

Beispiel:

```
SetValue("SnapshotControl.WriteProgram",
"Snapshot1;Snapshot2;Snapshot3;STOP;Snapshot4;Snapshot5;STOP;Snapshot6");
SetValue("SnapshotControl.StartProgram");
```

Dabei wird zunächst ein Programm mit „*WriteProgram*“ eingespeichert. Dies muss nur einmalig durchgeführt werden, bis das Programm entsprechend geändert werden soll.

Mit „*STOP*“ werden die Stellen markiert, an denen der Controller auf ein externes Signal warten soll. Er führt bezogen auf das Beispiel also Snapshot 1 bis 3 durch und wartet dann auf ein externes Signal, worauf er Snapshot 4 und 5 durchführt.

Durch „*StartProgram*“ kann das Programm, nachdem es übertragen wurde, jederzeit gestartet werden. Wird dieser Befehl aufgerufen, bleiben zwar die Tasten zur weiteren Kontrolle des Roboters blockiert bis das Programm abgearbeitet wurde, allerdings wird das *Virtuelle Interface* direkt wieder freigegeben. Dadurch kann das aufrufende Programm direkt weiterarbeiten. Dies bedeutet aber auch, dass es keinen direkten Rückgabe - Wert gibt, welcher mitteilt, ob das Programm noch arbeitet oder ob Fehler aufgetreten sind.

Dies wird durch die Funktion „*SnapshotControl.ProgramStatus*“ ermöglicht, wie das folgende Beispiel zeigt:

```
string programStatus = GetString("SnapshotControl.ProgramStatus");
```

Eine Beispielhafte Anwendung wäre folgende:

```
SetValue("SnapshotControl.WriteProgram",
"Snapshot1;Snapshot2;Snapshot3;STOP;Snapshot4;Snapshot5;STOP;Snapshot6");
SetValue("SnapshotControl.StartProgram");


while(GetString("SnapshotControl.ProgramStatus") == "Running")
{
  // Warte bis das Programm beendet oder abgebrochen wurde.
}
 Debug("Operation wurde mit folgendem Code beendet: " +
GetString("SnapshotControl.ProgramStatus"));
```

Hierbei wird ein Programm eingespeichert und ausgeführt. Anschließend wird in einer „*while - Schleife*“ gewartet, bis das Programm beendet wurde. Ist das Programm beendet und sind keine Fehler aufgetreten, liefert der Programstatus „*Successful“* zurück.

HINWEIS: Hat das Programm einen Punkt erreicht, an dem es auf ein externes Signal an Pin 6 wartet, wird „*WaitPosition“* zurückgegeben. Alternativ kann der Programmstatus auch als Code ausgelesen werden. Welche Codes dabei möglich sind, ist im Kapitel 5 Exit / Status Codes genauer beschrieben.

```
SetValue("SnapshotControl.WriteProgram", "Snapshot1;SPEED=10;Snapshot2;SPEED=STD;Snapshot3");
```

Soll sich die Geschwindigkeit während eines Snapshot - Programms ändern, kann dies durch das Schlüsselwort „*SPEED*“ angegeben werden. Folgendes Beispiel zeigt, wie nach einer Bewegung das Tempo verändert und anschließend wieder auf den Standard - Wert zurückgesetzt werden kann.

Als Standard - Wert gilt der in den Optionen eingestellte Wert für die Geschwindigkeit, falls dieser über das *Virtuelle Interface* nicht nachträglich verändert wurde.

HINWEIS: Es kann hierbei nur eine Geschwindigkeit kleiner gleich der Standard - Geschwindigkeit angegeben werden.

```
SetValue("SnapshotControl.ProgramStatusCode");
```

Soll das Programm warten, bis eine Warte - / Stopp - Position erreicht wurde, kann mit dem Befehl „*WaitForWaitPosition“* gearbeitet werden. Diesem Befehl kann auch ein Timeout übergeben werden.

Die Funktion liefert „*1“* zurück, wenn die Position erreicht wurde und „*0“*, wenn ein Fehler aufgetreten ist, die Funktion nicht verfügbar ist oder das Timeout überschritten wurde.

```
// Ohne Timeout
int waitPositionReached1 = GetValue("SnapshotControl.WaitForWaitPosition");
// Mit Timeout von 2000 ms
int waitPositionReached2 = GetValue("SnapshotControl.WaitForWaitPosition.2000");
```

Über den Befehl „*StopProgram“* kann das Programm im laufenden Betrieb gestoppt werden. Denkbar wäre eine anschließende Ausführung des Abort- Programmes. Bei diesem bewegt sich der Roboter zunächst in Z- Richtung nach oben und begibt sich dann zum übergebenen Snapshot.

```
SetEvent("SnapshotControl.StopProgram");
SetValue("SnapshotControl.StartAbortProgram", "Snapshot1");
```

Alternativ zu einem externen Signal, kann auch mit der Funktion „*SendImpuls“* gearbeitet werden.Befindet sich das Programm an einer Warte -/ Stop - Position, kann durch diese Funktion ein Impuls gesendet werden, wodurch der Roboter weiter arbeitet.

Beispiel:

```
SetEvent("SnapshotControl.SendImpuls");
```

6.2.6. Input / Output Signale

Über die beiden Funktionen des *Virtuellen Interfaces* „*GetSignal*“ und „*SetSignal*“ können Signale an der Slot - Karte des Controllers gesetzt und abgefragt werden.

Setzen von Signalen

Durch „*SetSignal*“ werden die Output Signale gesetzt. Zu beachten ist hierbei, dass manche der Pins bereits vom Controller reserviert sind und nicht gesetzt werden können.

Das Setzen der Signale erfolgt über einen dezimalen 16 - bit Wert.

Beispiel:

```
SetValue("RobotControl.SetSignal", 255);
```

In diesem Beispiel würden die ersten 8 Bit gesetzt werden, allerdings sind vom Controller einige Pins bereits reserviert.

Auslesen von Input Signalen

Über „*GetSignal*“ können die Input Signale des Roboters abgefragt werden. Dabei ist es möglich das gesamte Signal auszulesen oder einzelne Pins abzufragen.

```
int signal = GetValue("RobotControl.GetSignal");

int pin0 = GetValue("RobotControl.GetSignal.0");
```

Setzen von Pseudo Input Signalen für Debugging - Zwecke

Wird mit einer Simulation gearbeitet, können die ersten 15 Input - Signale für Testzwecke gesetzt werden. Dies ist keine direkte Funktionalität des Roboters und erfolgt daher über die Funktion „*RobotControl.SetValues*“.

Beispiel:

```
SetStringStream("RobotControl.SetValues", new [] {"PseudoInput=64"});
```

Dieses Beispiel setzt den Pin 6 auf 1.

6.2.7. Kollisions - Erkennung

Damit der Roboter stoppt, wenn er auf Gegenstände oder Hindernisse trifft, muss die Kollisions - Erkennung aktiviert und eingestellt werden.

ACHTUNG: Auch bei aktivierter Kollisions - Erkennung können Verletzungen oder Beschädigungen von Gegenständen beim ersten Kontakt nicht ausgeschlossen werden!

Ist die Sensitivität zu niedrig eingestellt, stoppt der Roboter nicht!

Einschalten der Kollisions - Erkennung

Aktiviert werden kann die Kollisions - Erkennung über die Teaching - Box oder RT Toolbox 2. Der entsprechende Parameter des Controllers ist „**COL“**.

Dieser Parameter besteht aus drei Elementen:

- Element 1: 1=Enable, 0=Disable
- Element 2: Aktivierung bei Initialisierung; 1=Enable, 0=Disable
- Element 3: Aktivierung bei JOG - Operationen; 1=Enable, 0=Disable

Einstellen der Sensitivität der Kollisions - Erkennung

Die Sensitivität der Kollisions - Erkennung kann durch den Parameter „**COLLVL“** eingestellt werden. Hierbei ist es möglich für jedes Gelenk einen Wert zwischen 1 und 500% anzugeben. Je kleiner der Wert ist, desto höher ist das Erkennungs - Level.

Einstellen der Sensitivität der Kollisions - Erkennung im JOG - Modus

Die Sensitivität kann auch für den JOG - Betrieb eingestellt werden. Dies geschieht über den Parameter „**COLLVLJG**“.

Auch hierbei können Werte zwischen 1 und 500% angegeben werden. Je kleiner der Wert ist, desto höher ist das Erkennungs - Level.

Weitere Informationen finden sich in der Anleitung des Roboters.

6.2.8. Verwendung eines Greifers

Die Verwendung eines Greifers, welcher an das J6 Gelenk angeschlossen werden kann, ist standardmäßig deaktiviert. Aktiviert werden kann die Funktionalität über das *Setup* durch den Startup Value „*EnableHand*“.

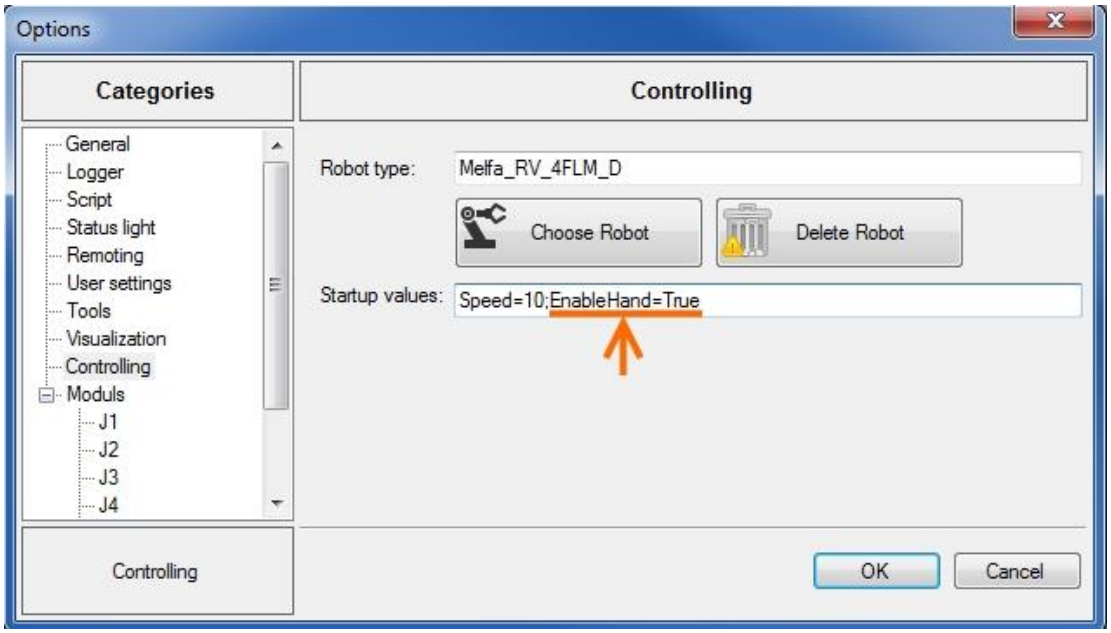


Abbildung 25: Verwendung eines Greifers

Ist die Funktionalität aktiviert, kann der Greifer über die Endeffektor - Oberfläche geöffnet und in 4 Stufen geschlossen werden.

Öffnen des Greifers

Der Greifer kann über die Endeffektor - Oberfläche geöffnet werden (Button hier rot markiert). Dabei werden alle 4 Stufen des Greifers zurückgesetzt.

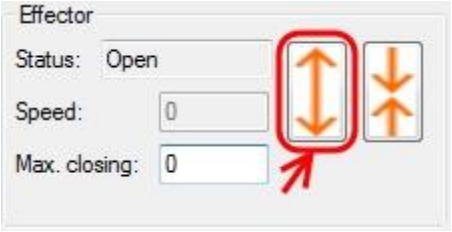


Abbildung 26: Öffnen des Greifers per Endeffektor - Oberfläche

Im *Virtuellen Interface* erfolgt das Öffnen des Greifers über den Befehl *„Open“* im entsprechenden Endeffektor.

Beispiel:

```
SetEvent("Endeffector[0].Open"); // Für einen Endeffektor mit dem Namen Endeffektor[0]
SetEvent("J6End.Open"); // Für einen Endeffektor mit dem Namen J6End
```

Schließen des Greifers

Geschlossen werden kann der Greifer in 4 Stufen: 0, 1, 2 und 3.

Die Stufe beschreibt, wie weit der Greifer geschlossen werden soll:

0-2: Nur teilweise schließen

3: Komplett schließen

Die vier Hand - Signale verhalten sich dabei so:

- **Stufe 0:** 1:Close; 2:Open; 3:Open; 4:Open
- **Stufe 1:** 1:Close; 2:Close; 3:Open; 4:Open
- **Stufe 2**: 1:Close; 2:Close; 3:Close; 4:Open
- **Stufe 3**: 1:Close; 2:Close; 3:Close; 4:Close

Die Stufe wird über das Feld *„Max. closing“* eingestellt. Das Tempo des Schließens ist für den Melfa - Roboter nicht implementiert.

Abbildung 27: Schließen des Greifers

Über das *Virtuelle Interface* gibt es zwei Möglichkeiten die Hand zu schließen:

Beispiel:

```
// Variante 1 (Stufe/Max wird getrennt gesetzt):
SetValue("Endeffector[0].Max", 3); // Setze die Stufe auf 3
SetEvent("Endeffector[0].Close"); // Schließe die Hand

// Variante 2:
SetValue("Endeffector[0].Close", 3); // Schließe die Hand auf Stufe 3.
```

Abfragen des Greiferzustandes

Der Greiferzustand wird zum einen auf der Benutzeroberfläche angezeigt und kann zum anderen über das *Virtuelle Interface* abgefragt werden. Dies erfolgt über den Befehl „*Status*“ des Endeffektors. Der Name des Endeffektors wird über das *Setup* festgelegt (default: *Endeffector[index]*).

Beispiel:

```
// Status des Endeffektors mit dem Namen Greifer
string status = GetString("Greifer.Status");

// Status des Endeffektors mit dem Namen Endeffektor[0]
string status = GetString("Endeffektor[0].Status");

// Status des Endeffektors mit dem Namen J6End
string status = GetString("J6End.Status");

...
```

Bei geöffnetem Greifer, wird „*Open*“ zurückgeliefert. Ist er teilweise oder ganz geschlossen, wird die Stufe zurückgegeben: *Closed0*, *Closed1*, *Closed2* oder *Closed3*.

Alternativ kann mit der Methode „*StatusCode*“ gearbeitet werden. Eine Liste der möglichen Status - Codes findet sich im Kapitel *5* Exit / Status Codes.

Beispiel:

```
// Status des Endeffektors mit dem Namen Greifer
int statusCode = GetString("Greifer.StatusCode");

// Status des Endeffektors mit dem Namen Endeffektor[0]
int statusCode = GetString("Endeffektor[0].StatusCode");

// Status des Endeffektors mit dem Namen J6End
int statusCode = GetString("J6End.StatusCode");
```

Auswerten der Input-Signale eines Greifers

Die bisher vorgestellten Methoden ermöglichen es die 4 Stufen des Greifers zu steuern und abzufragen. Oftmals kann es sein, dass nur die erste Stufe verwendet wird. In diesem Falle erfolgt die Abfrage, wie weit die Hand geschlossen ist, über externe Signale.

Allgemein (also auch, wenn alle 4 Stufen verwendet werden), können diese Signale wie folgt abgefragt werden:

Zunächst ist es nötig, dass im Controller des Roboters der Parameter **HIOTYPE** auf 0 (Source) gesetzt wird. Die externen Signale können nun an den Ports 900 - 915 abgefragt werden.

Dies geschieht über den Standard - Befehl für Signale.

Zum Beispiel:

```
int signal0 = GetValue("RobotControl.GetSignal.900");
int signal1 = GetValue("RobotControl.GetSignal.901");
int signal2 = GetValue("RobotControl.GetSignal.902");
int signal3 = GetValue("RobotControl.GetSignal.903");
int signal4 = GetValue("RobotControl.GetSignal.904");
int signal5 = GetValue("RobotControl.GetSignal.905");
int signal6 = GetValue("RobotControl.GetSignal.906");
int signal7 = GetValue("RobotControl.GetSignal.907");
// ...
```

Weiterhin ist es möglich 16 Signale gleichzeitig auszulesen. Angegeben wird dies im „*GetSignal*“ - Befehl durch ein vorausgehendes Minus - Zeichen. Um das gesamte Signal des Greifers abzufragen, kann also folgender Befehl verwendet werden:

```
int signal = GetValue("RobotControl.GetSignal.-900");
```

Zurückgeliefert wird eine 16 - bit Dezimalzahl.

6.3. RWDRHandler

Der Radialwellendichtring - Handler verfügt über zwei Linear - Achsen, welche in Z - Richtung verfahren. Die Achsen besitzen je einen Motor, welcher über die serielle Schnittstelle angeschlossen wird. Hierbei handelt es sich um ISEL - Motoren.

Da sich die Ports ändern könnten, ist es nötig, die COM - Schnittstellen über die Optionen anzugeben. Dies ist im folgenden Bild dargestellt.

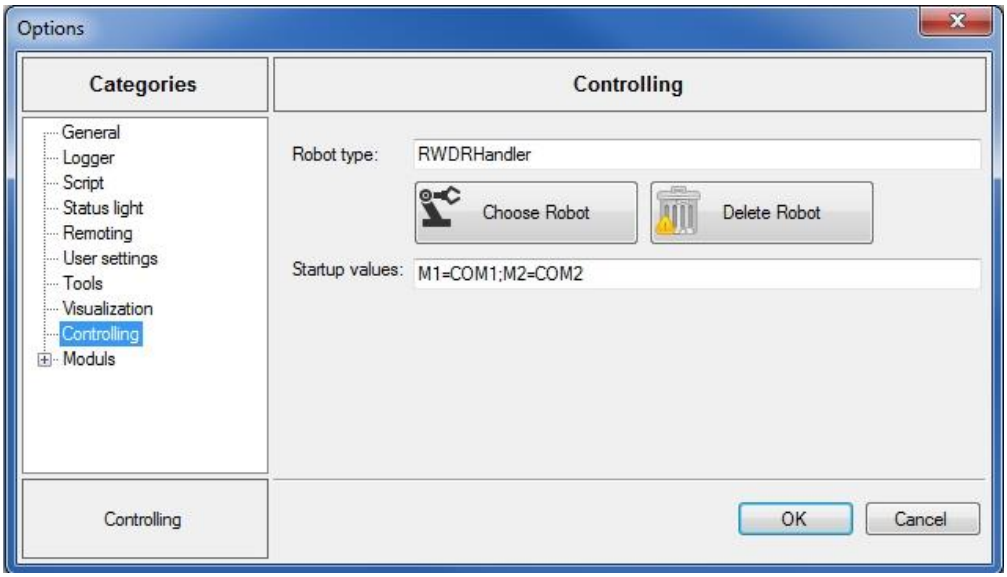


Abbildung 28: Angabe der COM- Schnittstelle

Weiterhin können in den Optionen folgende Parameter eingestellt werden:

- Geschwindigkeit **SPEED = [INTEGER > 0]**
- Beschleunigung **ACCEL = [INTEGER > 0]**
- Port - Timeout **PORTTIMEOUT = [INTEGER > 0]**
- Positionierungs - Timeout **POSTMEOUT = [INTEGER > 0]**

Sind diese Parameter nicht angegeben, werden Standard - Werte verwendet. Für das Homing werden immer die Standard - Werte des Motors verwendet.

ACHTUNG: Werden die Motoren über das *Virtuelle Interface* gesteuert, werden beim Homing und beim Anfahren von Positionen **keine** Exceptions geworfen. Fehlermeldungen werden hierbei in den Roboter - Status umgeleitet.

Dieser kann über die folgenden Befehle abgefragt werden:

```
// Als Zahlencode (siehe Exit-Codes)
int fehlercode = GetString("RobotControl.CurrentRobotStatusCode");
// Als String (siehe Exit-Codes)
string fehlername = GetString("RobotControl.CurrentRobotStatusCode ");
```

7. Optionen

7.1. Controlling - Optionen

Über die Controlling - Optionen wird der Roboter - Typ ausgewählt.

Weiterhin ist es möglich einen Value - String zu definieren, welcher dem Roboter beim Starten übergeben wird. Welche Werte übergeben werden können, kann aus den einzelnen Beschreibungen der Roboter - Typen entnommen werden.

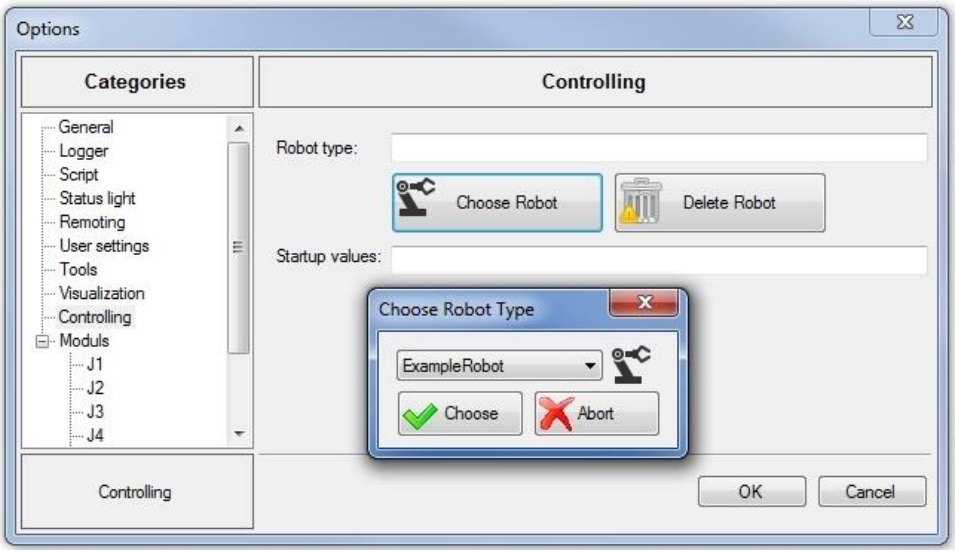


Abbildung 29: Controlling - Optionen

7.2. Modul - Optionen

Über die Modul - Optionen können die Name eines Moduls eingestellt und die Schrittweiten angepasst werden. Weiterhin ist es hier möglich Module ganz auszublenden oder nur deren Control zum Speichern von Positionen zu deaktivieren.

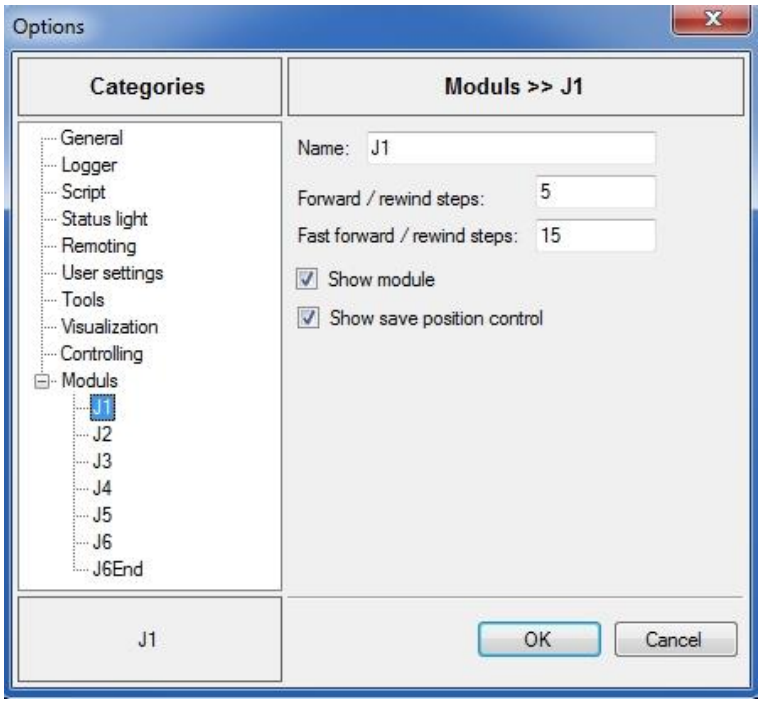


Abbildung 30: Modul - Optionen