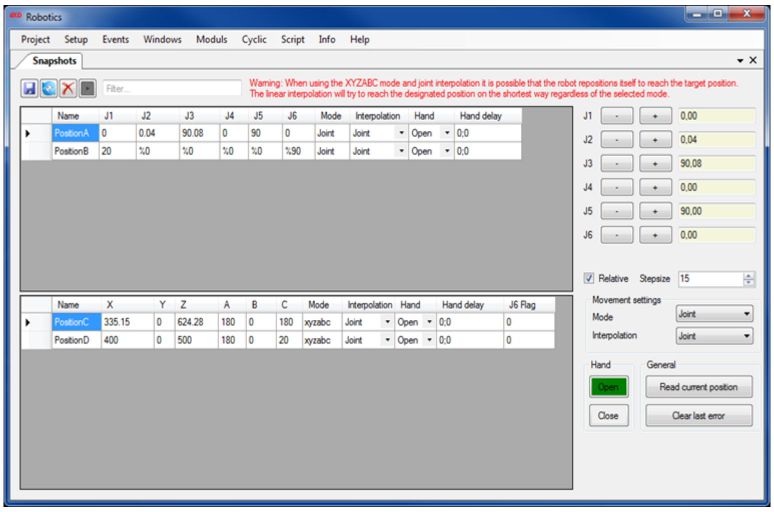
GET IN touch WITH SENSITIVE TESTING

Manual

Toolmonitor Robotics



Warning: When using the XYZABC mode and joint interpolation it is possible that the robot repositions itself to reach the target position. The linear interpolation will try to reach the designated position on the shortest way regardless of the selected mode.

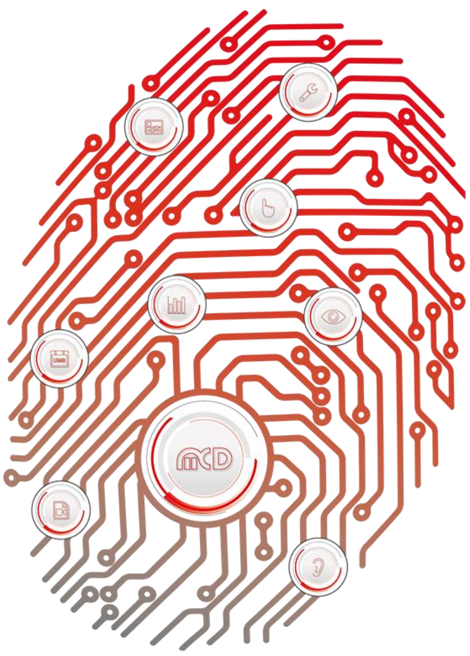
Name	J1	J2	J3	J4	J5	J6	Mode	Interpolation	Hand	Hand delay
PositionA	0	0.04	90.08	0	90	0	Joint	Joint	Open	0;0
PositionB	20	%0	%0	%0	%0	%90	Joint	Joint	Open	0;0

Name	X	Y	Z	A	B	C	Mode	Interpolation	Hand	Hand delay	J6 Flag
PositionC	335.15	0	624.28	180	0	180	xyzabc	Joint	Open	0;0	0
PositionD	400	0	500	180	0	20	xyzabc	Joint	Open	0;0	0

J1 0,00
J2 0,04
J3 90,08
J4 0,00
J5 90,00
J6 0,00

[✓] Relative Stepsize 15

Movement settings
Mode: Joint
Interpolation: Joint



Softline

Modline

Conline

Boardline

Avidline

Pixline

Application

MCD Elektronik GmbH
Hoheneichstr. 52
75217 Birkenfeld
Tel. +49 (0) 72 31/78 405-0
Fax +49 (0) 72 31/78 405-10
info@mcd-elektronik.de
www.mcd-elektronik.com
HQ: Birkenfeld
Managing CEO: Bruno Hörter
Register Court Mannheim
HRB 505692

Template version: 5.4 / 2018-04-09
V1.4 2018-09-03 BM (JJ)

Table of Contents

1. General

The MCD Toolmonitor Robotics is a software for controlling different robot types. Axis can be controlled manually via the interface, as well as positions and states can be stored and approached again.

The Toolmonitor supports a variety of different elements:

- Ball joints
- Rotation joints
- Torsion joints
- Linear joints
- Platforms
- Grippers

Order number: # 150225

2. Software, Driver Installation

2.1. Installing Software and Driver

Currently no installer is available. The application can be simply saved in a user - defined directory. Drivers are only required, if they are required by a respectively connected robot.

2.2. Starting the Installed Software

The software can be started by executing *RoboticsMonitor.exe*.

3. Introduction into Operation

3.1. Brief Introduction

No robot type is set on first start - up of the application.

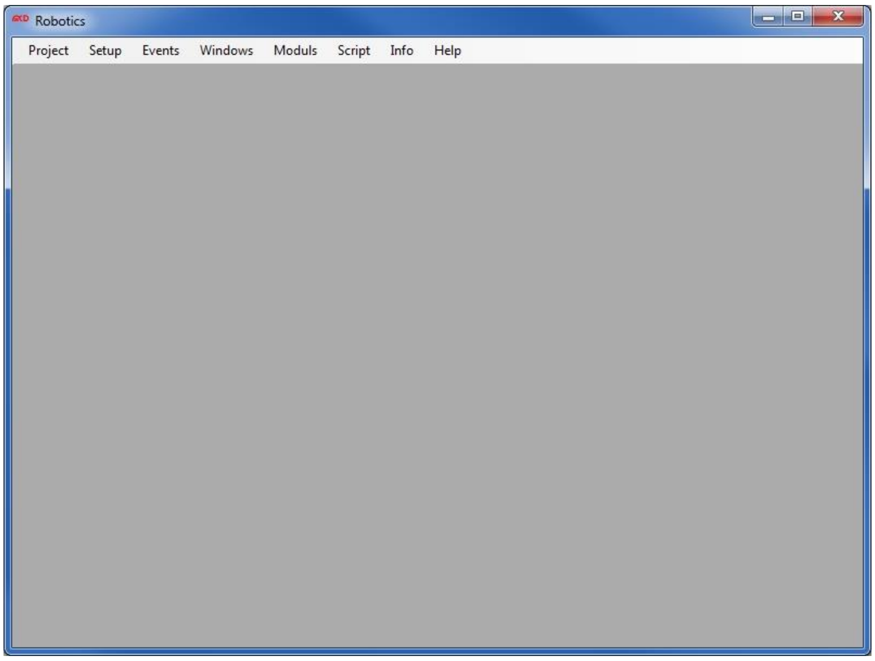
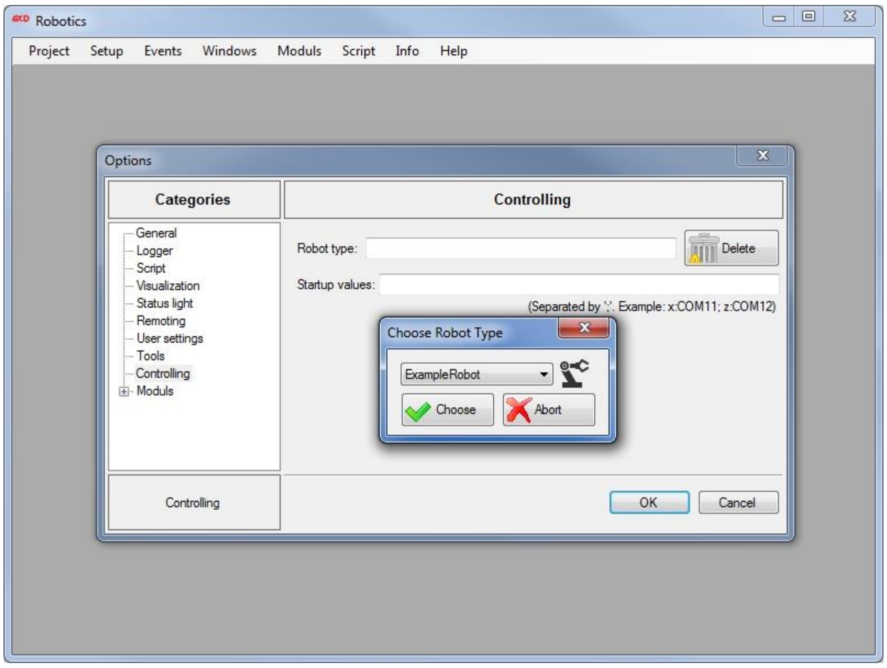


Figure 1: Start - Menu of the Toolmonitor Robotics

Using the option menu *Setup → Options*, the ExampleRobot can for example be adjusted. This robot simulates different functions of the Toolmonitor.



Robot type:

Startup values:

(Separated by ';'. Example: x:COM11; z:COM12)

ExampleRobot

Figure 2: Selection of the Robot Type

After the robot type has been selected, the controlling and snapshot interface can be displayed via the “*Windows”* menu item.

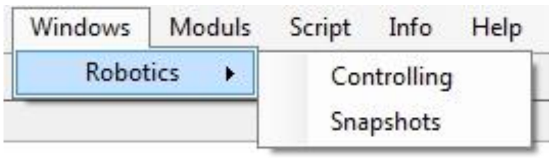


Figure 3: Display of the Controlling and Snapshot Interface

The individual robot modules can be accessed via “M*oduls*“. Here, all available joints, platforms, and end effectors are listed.

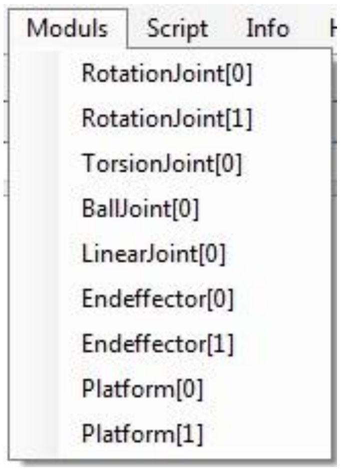
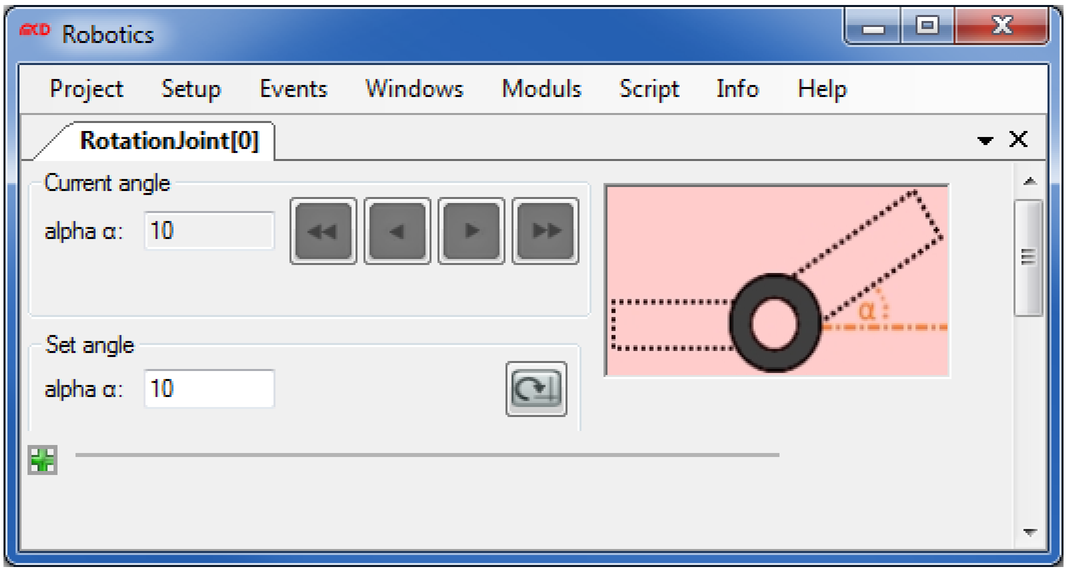


Figure 4: Robot Modules

For example, the first rotation joint can be displayed *(RotationJoint[0]).*

In the example robot, the angle can be set for this joint and the position can be manually “approached”. However, this is a virtual approach only, as no hardware is connected.



Current angle
alpha α: 10

Set angle
alpha α: 10

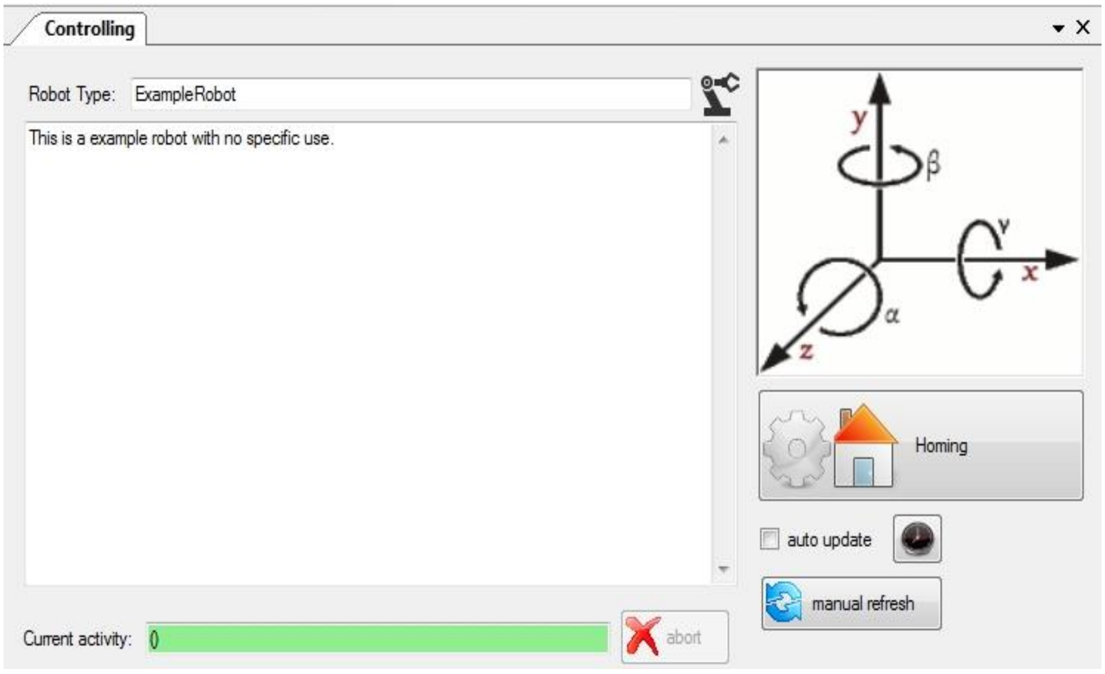
Figure 5: Selection of the First Robot Joint

4. Software Manual

The MCD Toolmonitor Robotics enables controlling of different robot types. The interface is structured the same for every adjusted robot. All joints, platforms, and end effectors designed for a robot type can be controlled.

4.1. Controlling Interface

Basic functions are possible via the controlling interface. This includes homing, action cancellation, and the adjustment of the interface refresh rate. Furthermore, this interface shows information about the selected robot.



Robot Type: ExampleRobot

This is a example robot with no specific use.

Current activity: 0

Figure 6: Controlling Interface

4.1.1. Homing

For some robots, *„homing"* is a requirement for any further operations. During homing, the position is referenced. For this purpose, the robot is moved into its initial state. This is mostly realized using limit switches.

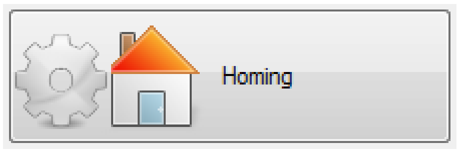


Figure 7: Homing Button

4.1.2. Canceling

Using the "*Abort*" button, the currently executed robot operation can be canceled. When and how the robot cancels the action depends on the respective robot implementation. For this reason, this button should not be used instead of the emergency off switch. The button is only active if the robot executed an action.

CAUTION: The abort button is not an equivalent alternative to an emergency off switch!

4.2. Refreshing the Interface

Using the menu point “*cyclic*”, the interface data can be updated. If auto update is not selected, the position changes of the joints are not displayed live. When activating the automatic update, a time interval (refresh rate) can be set. This interval can be changed in the menu - item “*Options*”

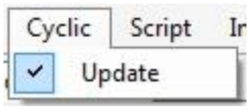


Figure 8: Refreshing the Interface

CAUTION: If the auto update is deactivated, the displayed positions in the individual modules may show an old state!

4.3. Modules

A robot consists of a set of modules. The respective implementation of the robot type decides, which robot modules can be controlled.

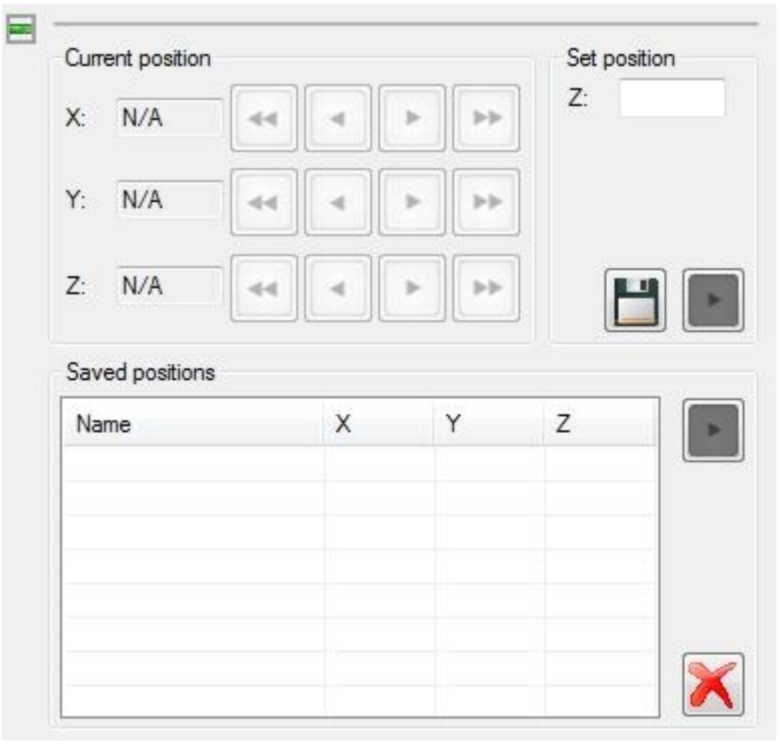
4.3.1. Positioning

Positioning is not a separate module. However, it is part of almost all available modules.

Positions can be approached and saved using this interface. Saved positions can be approached again.

In addition, positions can also be approached manually. For this purpose, the robot must be able to determine its current position.

The position in X, Y, and Z direction can be set using the positioning interface. Depending on the implementation of the individual modules, individual axes are hidden.



Current position

X: N/A

Y: N/A

Z: N/A

Set position

Z:

Saved positions

Name	X	Y	Z

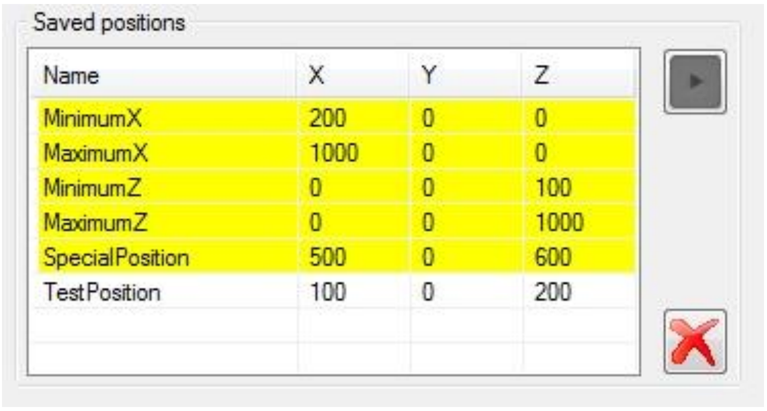
Figure 9: Positioning Interface

Furthermore, special positions can be taught during saving. Depending on the robot type, they have different meanings. These positions are highlighted in yellow in the list of saved positions.

The following special positions can be taught:

- MinimumX: Defines usually the smallest approachable X - value
- MaximumX: Defines usually the largest approachable X - value
- MinimumY: Defines usually the largest approachable Y - value
- MaximumY: Defines usually the largest approachable Y - value
- MinimumZ: Defines usually the largest approachable Z - value
- MaximumZ: Defines usually the largest approachable Z - value
- SpecialPosition: Interpretation of this point depends on the respective implementation.

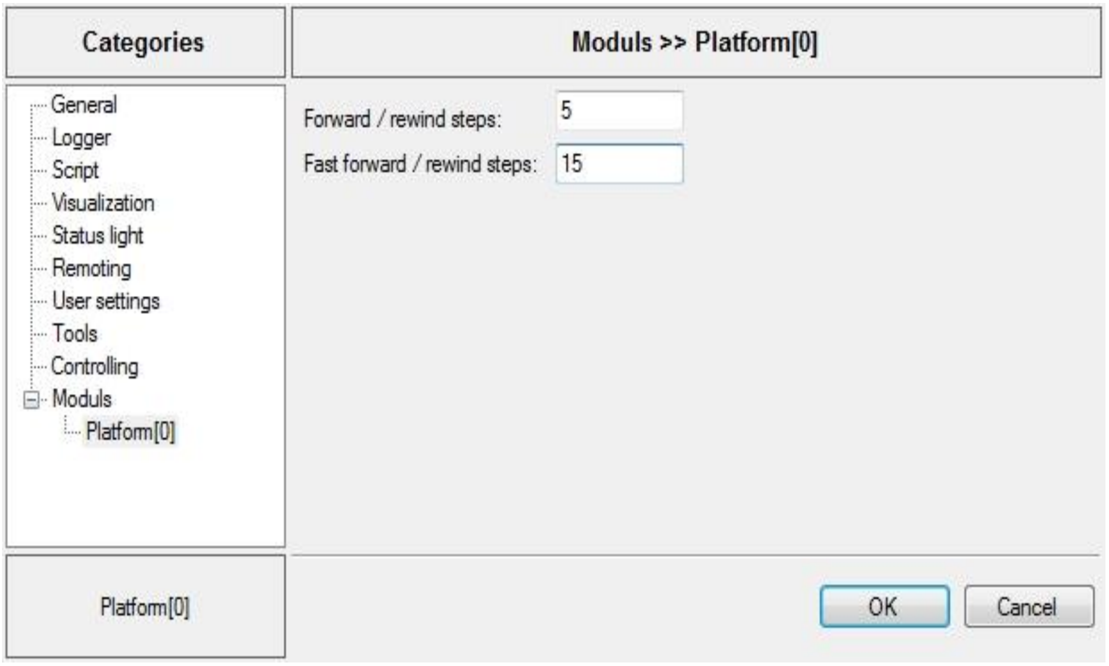
NOTE: The maximum and minimum positions are only effective if the selected robot supports them!



Name	X	Y	Z
MinimumX	200	0	0
MaximumX	1000	0	0
MinimumZ	0	0	100
MaximumZ	0	0	1000
SpecialPosition	500	0	600
TestPosition	100	0	200

Figure 10: Saved Positions

The increments of the forward / backward buttons can be changed via the options menu.



Forward / rewind steps: 5

Fast forward / rewind steps: 15

Figure 11: Adjustment of the Increments

4.3.2. Ball Joint

The ball joint can be adjusted using two angles. These can be directly set or manually controlled.

Furthermore, the already described positioning interface in included in this module.

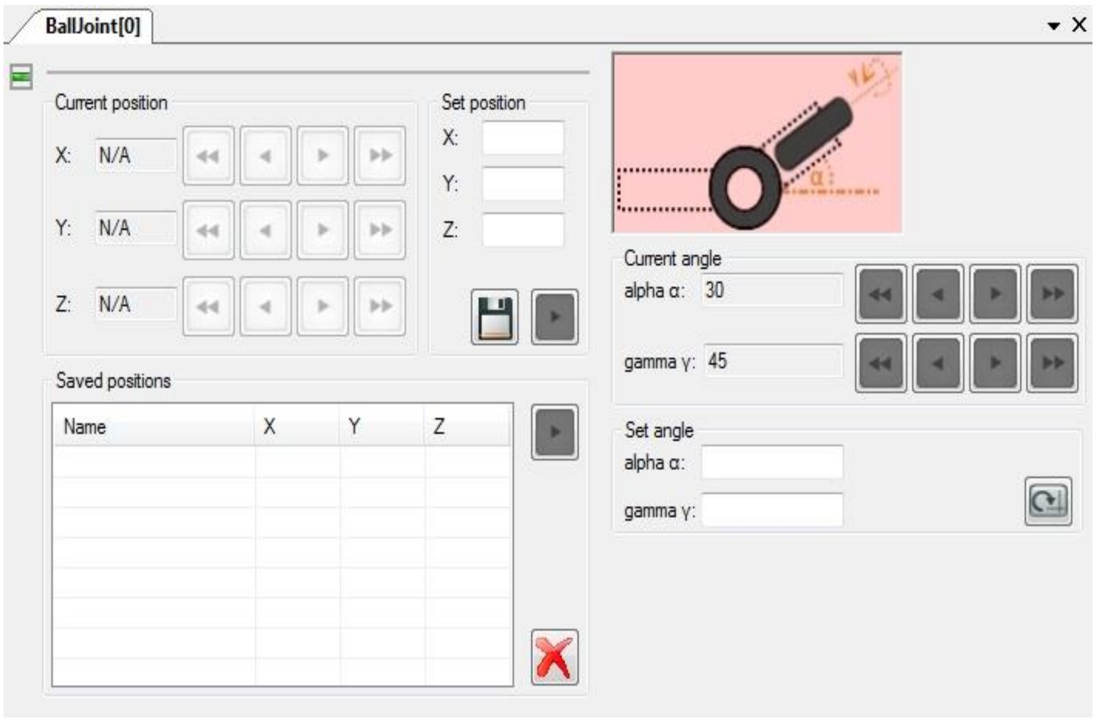


Figure 12: Positioning Interface with Ball Joint

4.3.3. Rotation Joint

The rotation joint can be adjusted using only one angle. This angle can be manually approached or directly set. The positioning interface is also included in this module.

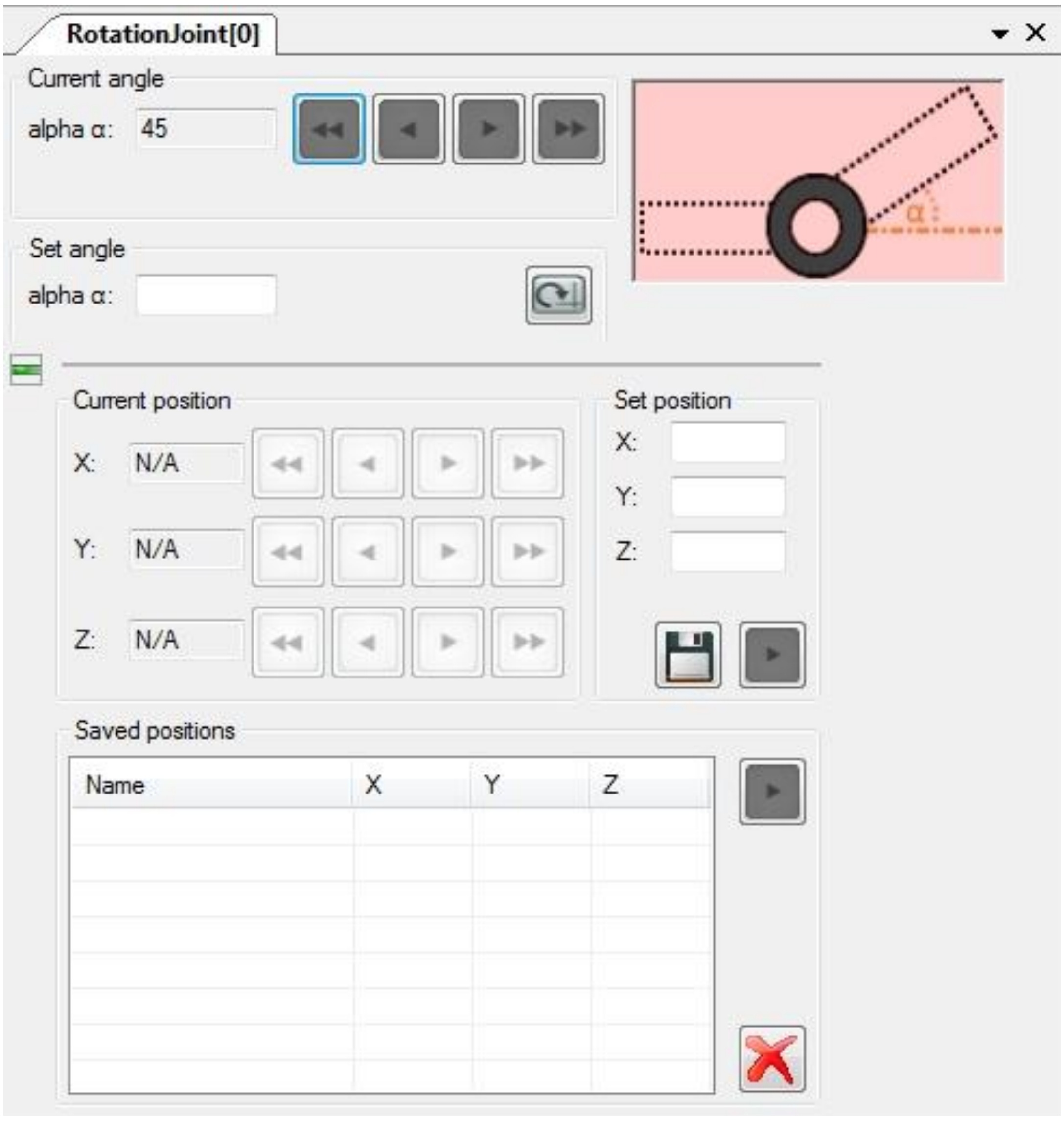


Figure 13: Positioning Interface with Rotation Joint

4.3.4. Torsion Joint

Only one angle can be set for the torsion joint. As already described for the previous modules, the positioning interface is included in the module.

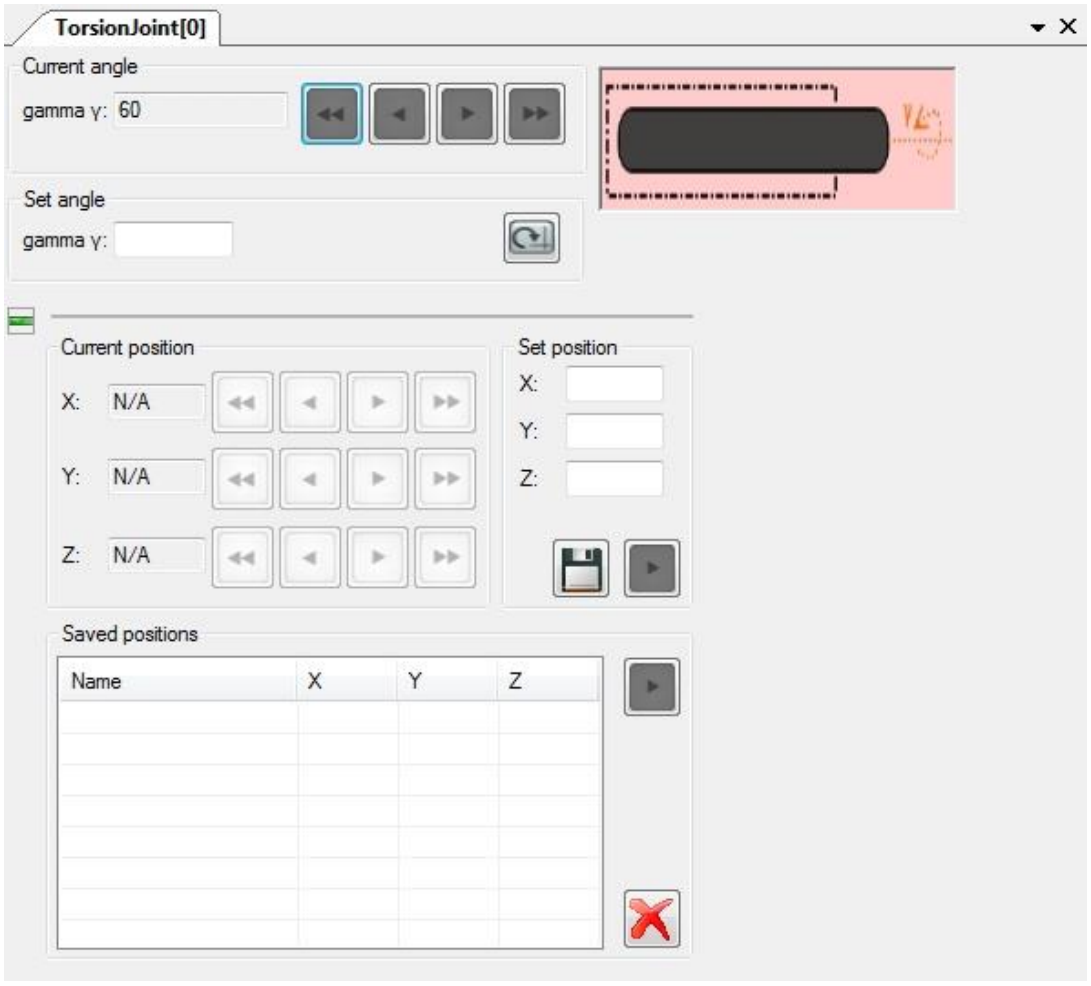


Figure 14: Positioning Interface with Torsion Joint

4.3.5. Linear Joint

Differently from the previous modules, the linear joint does not feature an angle. The joint can be extended instead. Here, the linear position, and / or length of the module changes.

The positioning interface is also included in the module.

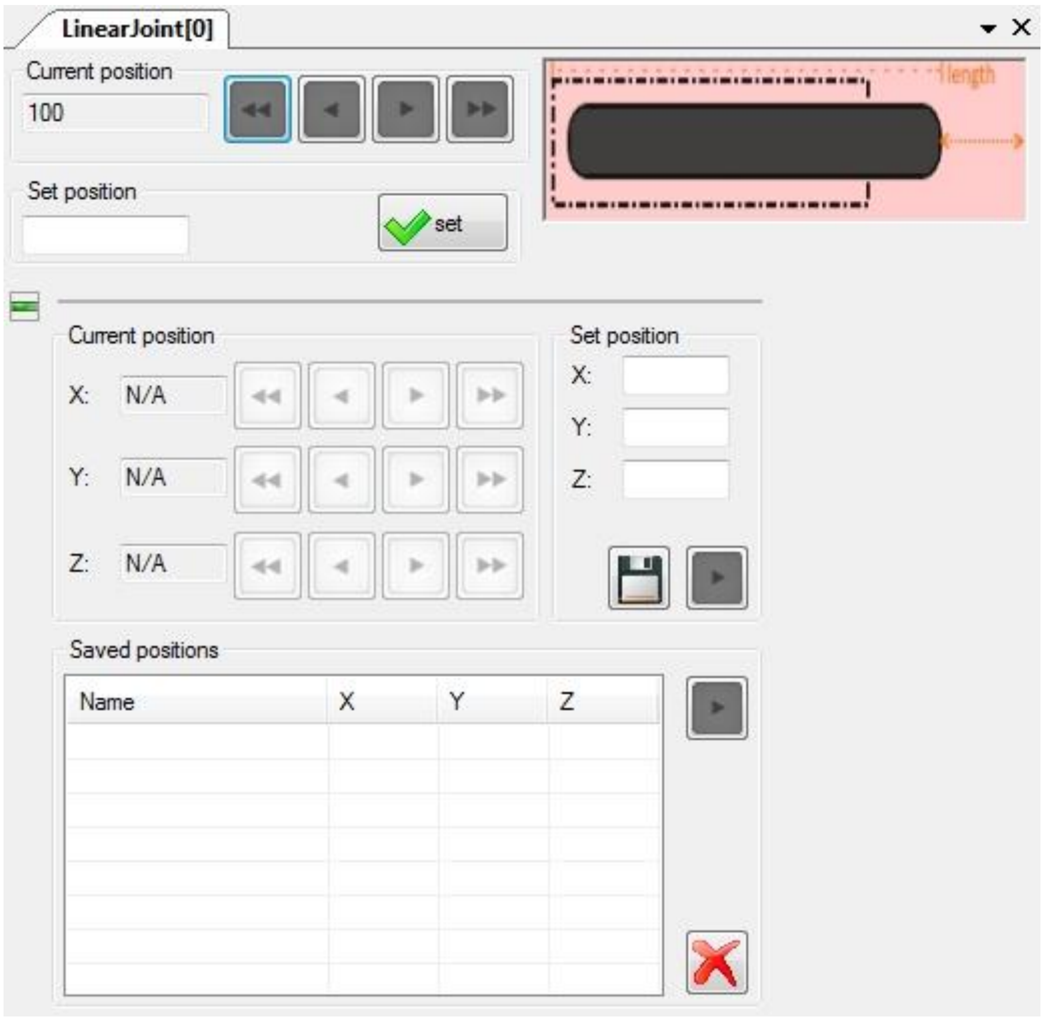


Figure 15: Positioning Interface with Linear Joint

4.3.6. Platform

The platform consists of the positioning interface only. The respective implementation of the robot type decides which axes positions can be set. In the case shown here the platform can only travel in Z - direction. If the robot does not return information regarding the current position, manual traveling is deactivated for safety reasons.

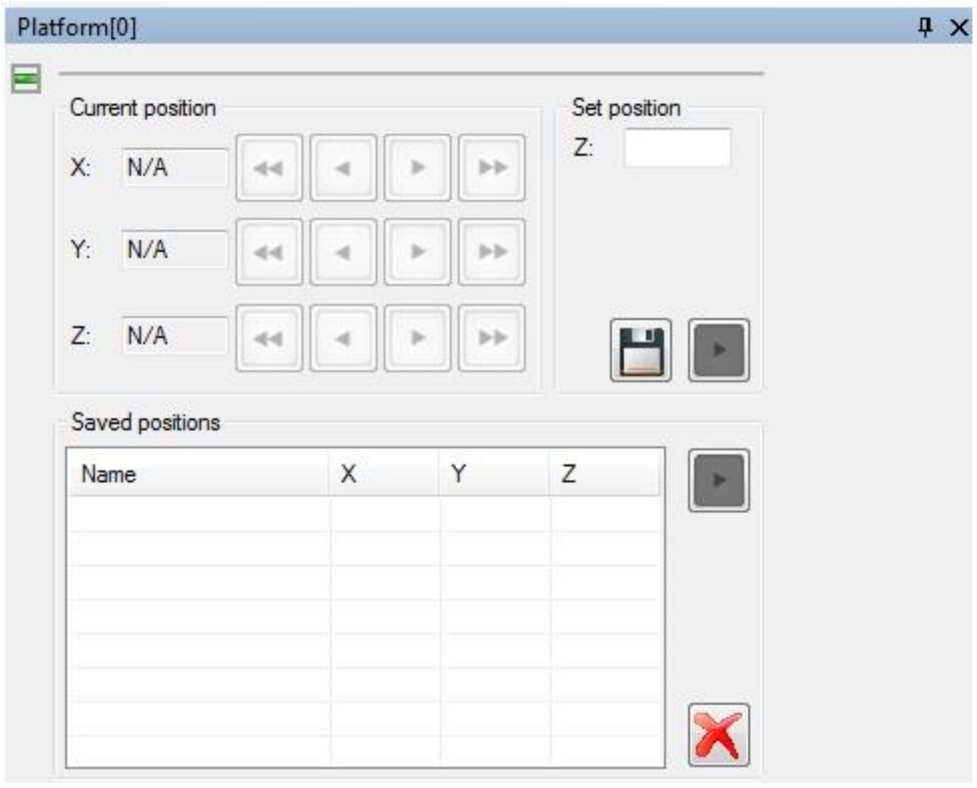
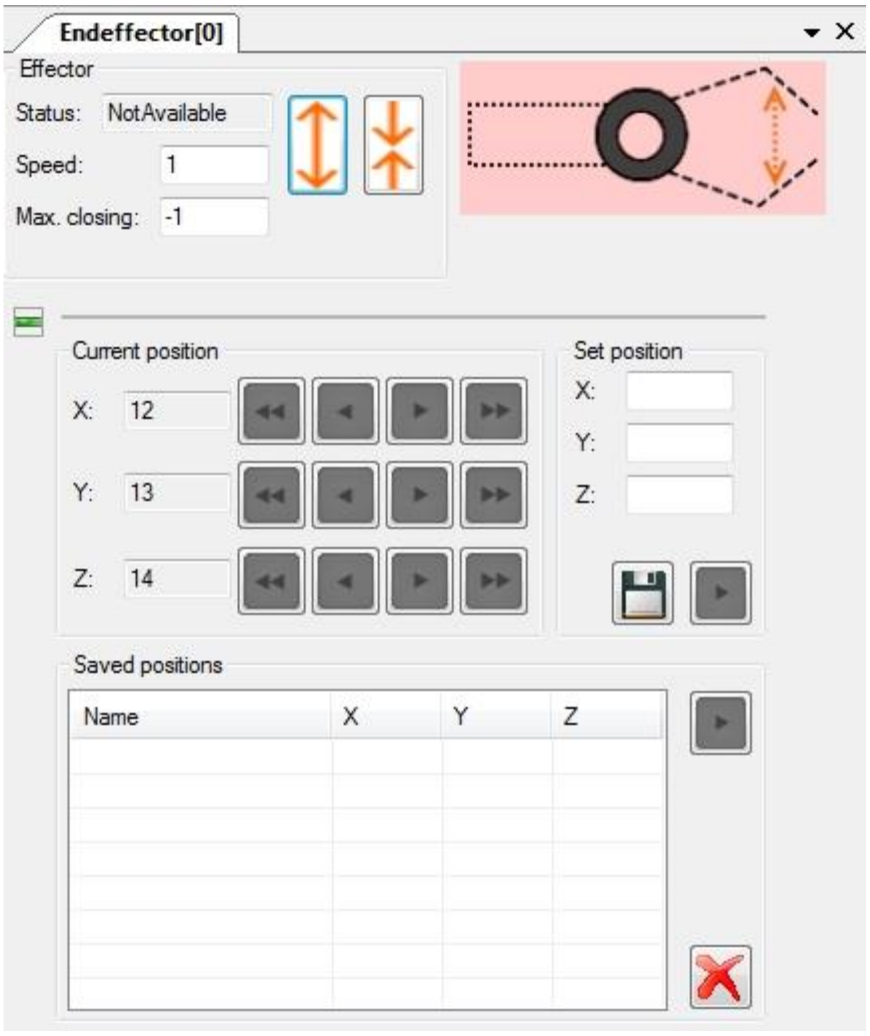


Figure 16: Platform Module

4.3.7. End Effector / Gripper

The end effector is usually the last element of a kinematic chain. Here, the Toolmonitor Robotics implements a gripper that can be opened and closed. If supported by the robot type, closing speed and distance of the gripper can be adjusted. The modules can also be controlled via the positioning interface.



Status: NotAvailable
Speed: 1
Max. closing: -1

Current position: X: 12, Y: 13, Z: 14

Name	X	Y	Z

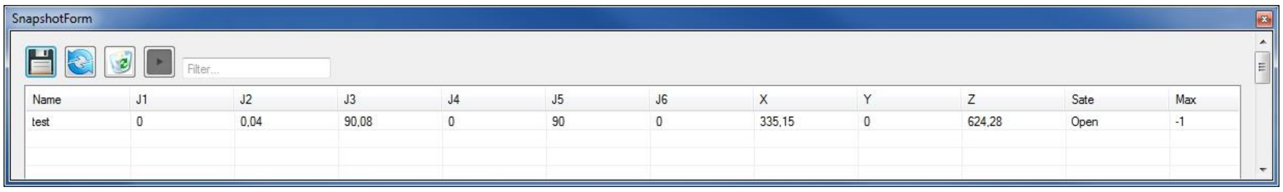
Figure 17: Positioning Interface with End Effector / Gripper

4.4. Snapshot Interface

The Toolmonitor Robotics includes two variants of the snapshot interface. For 6 - axes robots it is normally the variant 2.

4.4.1. Variant 1

Using the positioning interface, positions can be saved only for the individual modules. The snapshot interface supports saving of positions, angle settings, states, and settings of all modules. Thus, the state of the entire robot can be saved, not only of individual elements. This state can be saved and approached again at a later point in time.


SnapshotForm

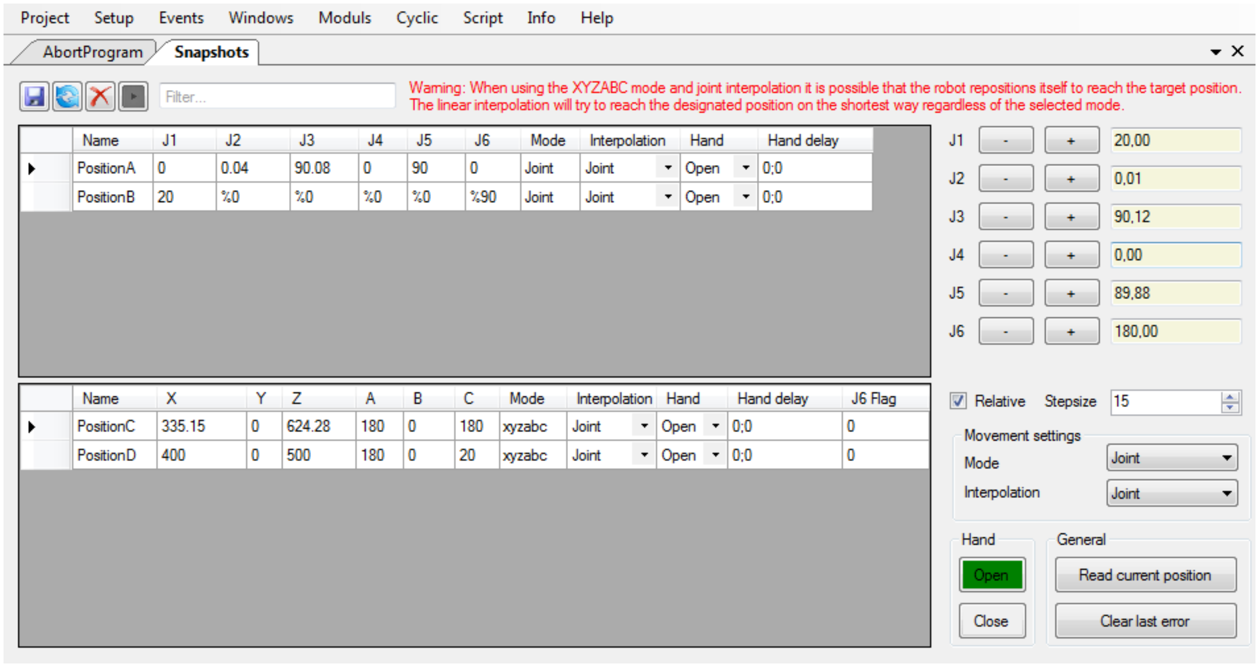
Filter...

Name	J1	J2	J3	J4	J5	J6	X	Y	Z	Sate	Max
test	0	0,04	90,08	0	90	0	335,15	0	624,28	Open	-1

Figure 18: Snapshot Form (Variant 1)

4.4.2. Variant 2

Variant 2, also called Advanced Snapshot, includes the functionality to save and approach saved postions as well. Additionally variant 2 has the option to process the axes directly and open or close the gripper.



Warning: When using the XYZABC mode and joint interpolation it is possible that the robot repositions itself to reach the target position. The linear interpolation will try to reach the designated position on the shortest way regardless of the selected mode.

Name	J1	J2	J3	J4	J5	J6	Mode	Interpolation	Hand	Hand delay
PositionA	0	0.04	90.08	0	90	0	Joint	Joint	Open	0;0
PositionB	20	%0	%0	%0	%0	%90	Joint	Joint	Open	0;0

Name	X	Y	Z	A	B	C	Mode	Interpolation	Hand	Hand delay	J6 Flag
PositionC	335.15	0	624.28	180	0	180	xyzabc	Joint	Open	0;0	0
PositionD	400	0	500	180	0	20	xyzabc	Joint	Open	0;0	0

J1	-	+	20,00
J2	-	+	0,01
J3	-	+	90,12
J4	-	+	0,00
J5	-	+	89,88
J6	-	+	180,00

✓ Relative Stepsize 15

Movement settings
Mode: Joint
Interpolation: Joint

Figure 19: Advanced Snapshot (Varaint 2)

In addition, this snapshot variant allows to specify the type of interpolation. It is also possible to specify, wether the positions should be stored as XYZABC coordinates or the individual joint positions. These settings are possible for each snapshot separately.

Subsequently the values of stored snapshots can be adjusted by a click on the field, which should be changed.

4.5. Setting and Reading Out of Signals

Depending on the robot type, signals can be sent to the robot or inquired. For this purpose, the “*Virtual Interface*“ is used. The interpretation of the command depends on the respective robot type.

4.5.1. Setting Signals

A signal can be set via *“SetSignal“*.

Example:

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

4.5.2. Reading Out Signals

The “*GetInputSignal“* command is used to read out a signal.

Example:

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

Alternatively, a specific pin can be inquired, assuming that is supported by the robot type.

Example:

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

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

4.5.3. Restoring Emergency - Stop Signals

In case the emergency stop switch was pressed while the Toolmonitor was working, some robots disallow the execution of commands until the emergency stop was restored.

The approval is carried by the “*ResetEmergencyStop”* function in the *Virtual Interface*.

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

5. Exit / Status Code

In addition to a "*GetString*" method, a "*GetValue*" method is often provided that returns a status code.

This is the case for the following functions:

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

The codes refer to the following messages:

Code	Name	Description
-1	NotAvailable	The called function is not available for the selected robot type.
0	Successful	The (last) action was executed successfully. **Or:** Execution finished.
1	Error	A general, not further specifiable error occurred.
2	ConnectionFailed	No connection could be established to the robot.
3	UNDEFINED	An unexpected error occurred.
4	EmergencyStop	The emergency off button was operated.
5	Aborted	The (last) action was canceled.
6	MotorNotReady	The robot motors have not been initialized yet or they are in a state, in which they cannot be started.
7	Homing_Required	The action can only be executed after a homing was performed.
8	ExternProgramIsMissing	An external program is missing (e.g., for Melfa ROBOTCONTROL on the robot controller).
9	CurrentPositionNotAvailable	The inquiry of the current robot position failed.
10	PositionOutOfReach	The desired position cannot be reached.
11	CAUTION_Could_not_set_speed	The robot speed could not be set. Be careful, as maximum speed may have been set.
12	Timeout	A timeout occurred during execution.
13	NotImplemented	The function is not implemented (for debugging purposes only during development).
14	AutomaticModeNotEnabled	The robot is not in automatic mode (Melfa).
15	EmergencyStop_or_DoorOpen	The robot is blocked. This may be caused by pressing the emergency off button or by opening a safety - relevant door.

16	Running	The robot is currently executing a snapshot program.
17	WaitPosition	The robot is currently executing a snapshot program and is in a wait position.
18	OpeningHandFailed	The robot hand could not be opened.
19	ClosingHandFailed	The robot hand could not be closed.
20	ExternInterrupt	An extern interrupt has occurred.
21	InvalidInput	The specified value for a parameter is invalid or the value range was exceeded.

5.1. End Effector / Gripper Status Codes

If the status code of an end effector is inquired, the following return codes are possible:

Code	**Name**	**Description**
-2	Undefined	The hand is in an unknown state. For some robots, this is always the case after start - up.
-1	NotAvailable	The status of the selected end effector cannot be inquired for this robot.
0	Open	Gripper is open.
1	Closed	Gripper is closed.
2 3 4 5	Closed0 Closed1 Closed2 Closed3	Robot - specific information, how far the hand is closed. Further information can be found in the description of the respective robot, assuming that this functionality is supported.

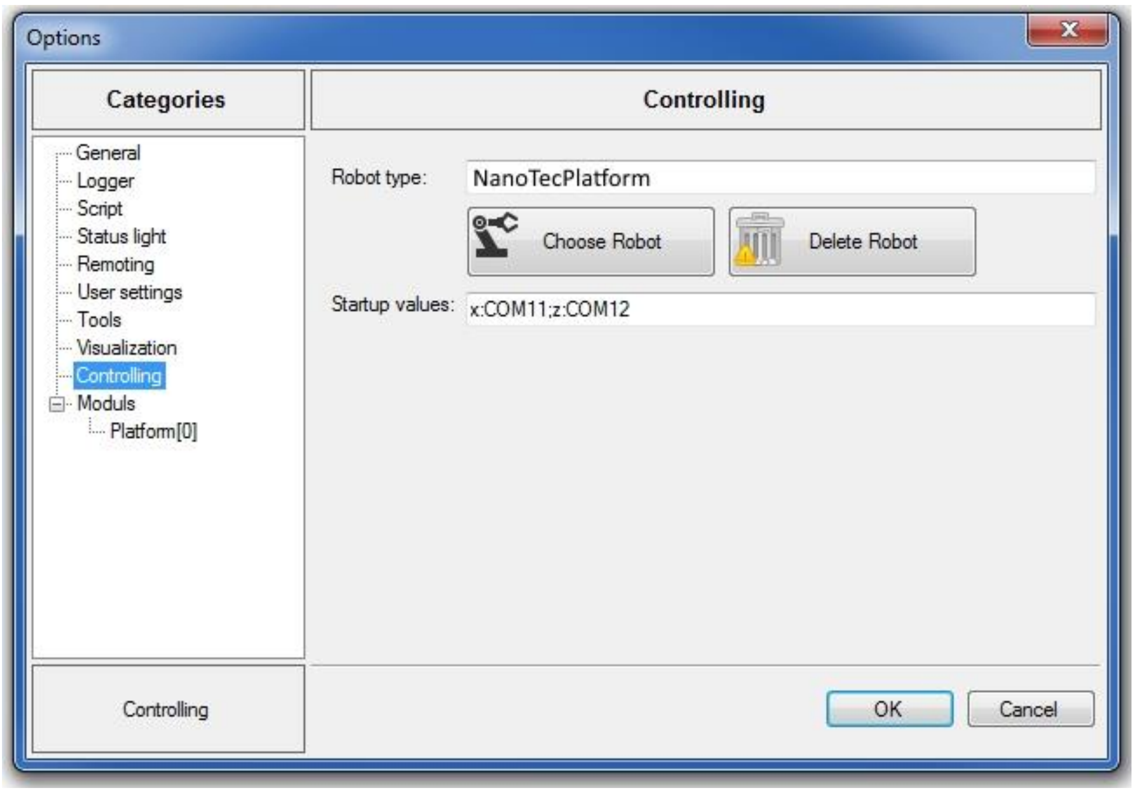
6. Robot Types

The Toolmonitor Robotics already implements different robot types. If further explanations are required, they can be found in the following sections.

6.1. NanoTec Platform

The motor in Z - direction is at COM12. The motor in X - direction used for approaching the test object using the camera is at COM11.

As these ports may change, the COM interfaces must be entered in the menu - item “*Options*” as shown in the figure.



Robot type: NanoTecPlatform

Startup values: x:COM11;z:COM12

Figure 20: Determine the COM Interfaces

6.2. Mitsubishi Melfa RV 4FLM D

The Mitsubishi robot Melfa RV-4FLM-D is a six-axes robot.

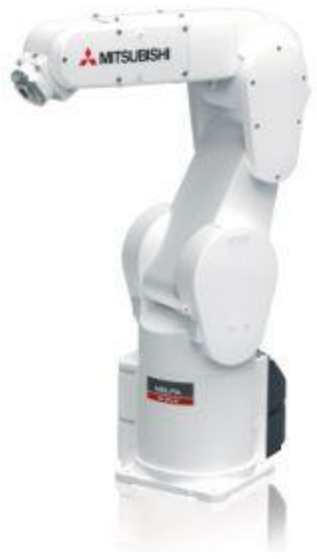


Figure 21: Mitsubishi Robot Melfa RV 4FLM D

These axes as well as the connected grippers can be controlled using the Toolmonitor Robotics. Every axis can be individually addressed. Snapshots can be taught and approached again later.

Here, a sequence of up to 10 snapshots can be controlled via the *Virtual Interface*. A snapshot program can be saved as well. Please refer to chapter 6.2.5 Working with Snapshots for explanations that are more detailed.

In addition to snapshots, end positions via the X, Y, and Z - axes can be saved as well. The functionality of saving end positions can be deactivated via the menu-item “*Setup*“.

CAUTION!

If a position is approached, the orientation of joint J5 remains the same!

If a position is approached, saved, and approached again at a later point in time, joint J5 must be moved into this orientation first!

It is recommended to use snapshots instead.

WARINING!

The robot does not realize when it meets resistance.

It must be ensured that no object or person are in the path prior to approaching a position or snapshot!

Even in Compliance mode, there is a risk of injury and damage to components!

The current robot state can be read out using the *GetCurrentRobotStatus* and *GetCurrentRobotStatusCode* functions.

The latter of the two functions returns a code. A list of possible codes can be found under Exit/Status Codes.

```
 string robotStatus = GetString("RobotControl.CurrentRobotStatus");
 int robotStatusCode = GetValue("GetValue.CurrentRobotStatusCode");
```

6.2.1. Compliance Mode

The Compliance Mode softens the joints of the robot and slightly adjust to the environment during positioning.

CAUTION: First it must be ensured that the object will not be damaged, when counter - pressure is applied.

The Compliance Mode can be activated and deactivated via the *Virtual Interface*.

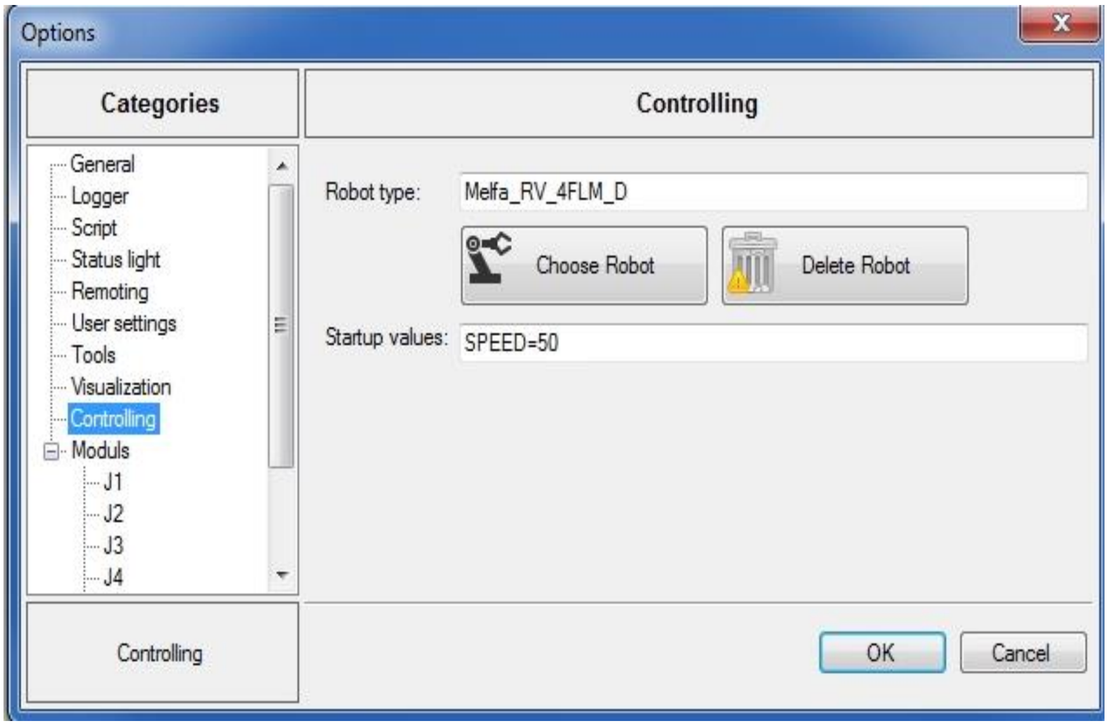
NOTE: Compliance Mode becomes active / inactive during next positioning.

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

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

6.2.2. Speed

Two different options are available for adjusting the robot speed. On the one hand, the speed can be set during *setup*. This set speed applies, until the value is changed via the *Virtual Interface.*



Robot type: Melfa_RV_4FLM_D

Startup values: SPEED=50

Figure 22: Speed Adjustment

Setting the speed via the *Virtual Interface* is the second option. Here is an example:

```
// Setting the speed to 10 %
 SetValue("RobotControl.SetValues", new []{"Speed=10"});

 // Setting the speed to 5%
 SetValue("RobotControl.SetValues", new []{"Speed=5"});
```

CAUTION:
Never set the speed > 10 % during teaching and test operation to ensure the safety of persons and objects!

The speed can be set to 100 %. For this reason, check what speed is adjusted for the robot prior to executing any action.

Make sure that no persons are within the range of the robot when controlling the robot! Pay attention to objects in the surrounding!

6.2.3. Safetey Warnings

Danger to Persons

Make sure that no personal injuries occur at all times when operating with the robot. Persons must not be within the robot range during robot operation.

If the robot requires maintenance, switch the robot off.

Make sure that nobody can access the robot cell during running robot operation. Nobody must be exposed to danger due to ignorance.

Danger of Objects

Make sure that no objects are damaged by the robot or thrown in the area during robot operation.

When approaching a position or snapshot, make sure that no objects are on the travel path that could be brushed or hit and thus damaged during repositioning.

Danger in Compliance Mode

Test the Compliance Mode extensively prior to using it during running operation.

Make sure that object cannot be crushed or damaged due to joint softening.

Although the joints are softer in Compliance Mode, they only yield in the case of counter - pressure. Joints cannot be randomly moved in Compliance Mode. If the position deviates by more than 2 cm from the original position, the robot stops.

Danger caused by speed

The robot speed can be set to 100 % using the Toolmonitor. Make sure that the correct speed is adjusted during every robot use.

The speed in testing and teaching mode should not be greater than 10 %!

It should be ensured that the robot is well secured and remains in its anchorage at high speeds.

Further Safety Information and Guidelines

This is not a complete list of safety information and guidelines.

Further information can be found in the robot documentation or inquired on the Mitsubishi website.

6.2.4. Commissioning

It is recommended to activate the collision detection of the robot, assuming that this function should be used. Descriptions can be found under chapter 6.2.7. Collision Detection. Further information are available in the robot instructions.

It is also recommended to define virtual walls, which the robot cannot cross.

RT Toolbox 2

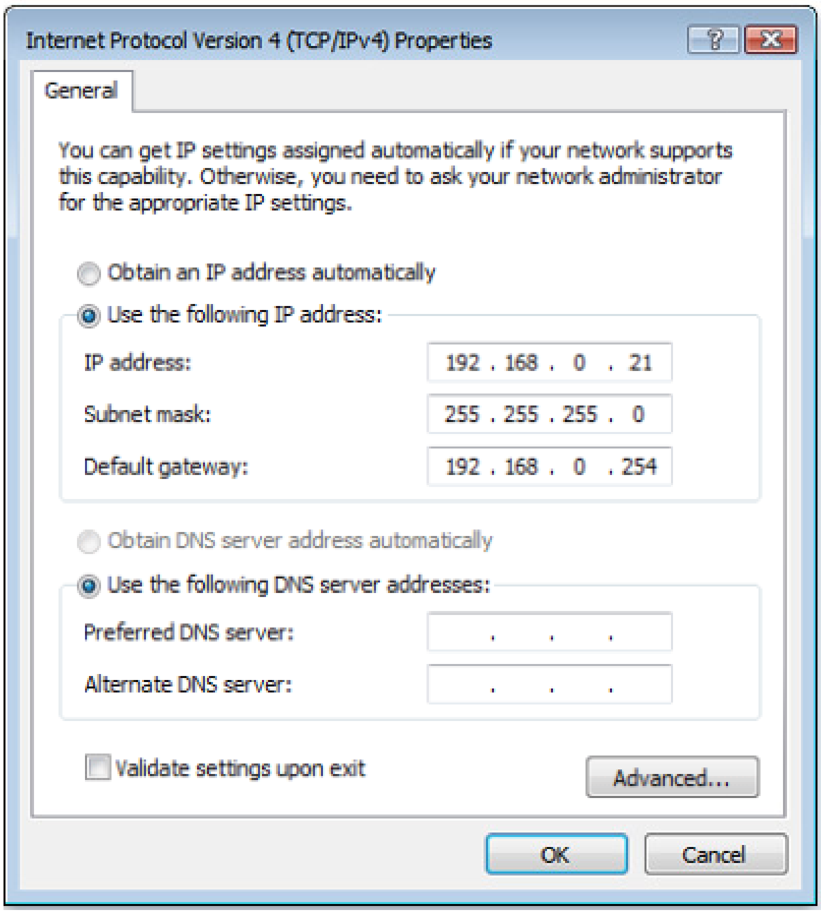
RT Toolbox 2 must be installed to use the Toolmonitor Robotics together with a Mitsubishi robot. Using this Mitsubishi software, programs can be transferred to the robot; the robot can be maintained and manually controlled.

Furthermore, the installation provides the libraries required by the Toolmonitor for robot communication.

Network Configuration

To communicate with the robot, the robot must be connected to the computer hosting the Toolmonitor. The connection is established via TCP/IP.

The IP options of the respective ethernet port on the PC must be changed to the following settings to establish communication.


Internet Protocol Version 4 (TCP/IPv4) Properties

General

You can get IP settings assigned automatically if your network supports this capability. Otherwise, you need to ask your network administrator for the appropriate IP settings.

( ) Obtain an IP address automatically
[✓ Use the following IP address:]

IP address:	192 . 168 . 0 . 21
Subnet mask:	255 . 255 . 255 . 0
Default gateway:	192 . 168 . 0 . 254

( ) Obtain DNS server address automatically
[✓ Use the following DNS server addresses:]

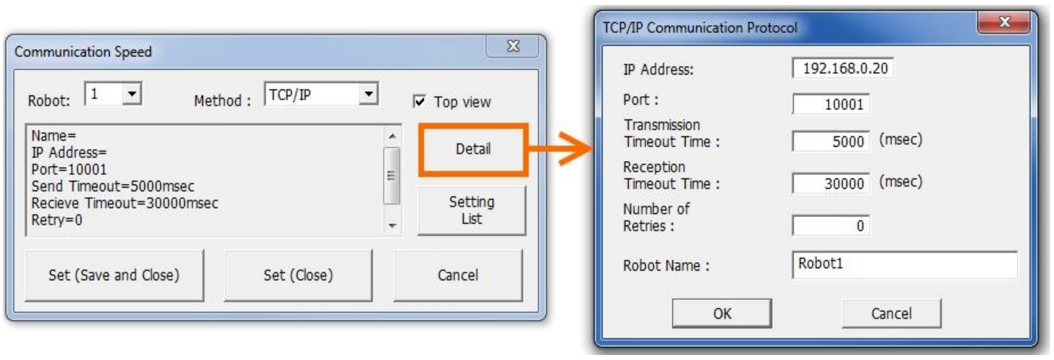
Preferred DNS server:	. . .
Alternate DNS server:	. . .

☐ Validate settings upon exit

Figure 23: Network Configuration

Next, *RoboCom.exe* must be loaded. This application is opened together with the Toolmonitor.

Here, the communication type must be set to TCP/IP first. Next, the robot IP 192.168.0.20 must be entered.



Robot: 1
Method : TCP/IP
☑ Top view

```
Name=
IP Address=
Port=10001
Send Timeout=5000msec
Recieve Timeout=30000msec
Retry=0
```

IP Address:	192.168.0.20
Port :	10001
Transmission Timeout Time :	5000 (msec)
Reception Timeout Time :	30000 (msec)
Number of Retries :	0
Robot Name :	Robot1

Figure 24: Select Communication Type

Activating Automatic Mode

The robot must be in the automatic mode in order to allow the Toolmonitor writing access to the robot controller.

This mode can be activated and deactivated using a key on the front of the controller.

CAUTION:
If the speed was set to 100 % in manual teaching mode, this speed will still be set in automatic mode, until the speed is changed via the Toolmonitor or the user.

Control Program on Robot

A program must be transferred to the robot prior to commissioning to enable Toolmonitor interaction with the robot.

For this purpose, a connection is established to the robot using RT Toolbox 2 and the software is loaded to the robot.

To allow Toolmonitor access to the program, it must be saved under name **ROBOTCONTROL**.

The program code is attached below.

```
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
```

In addition, the following cancellation program with the name **ABORTPRGM** must exist on the controller.

```
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. Working with Snapshots

In the case of the Melfa RV-4FLM-D robot, snapshots can be used in different variants.

It is possible to:

- Approach a single snapshot
- Approach a snapshot sequence (up to 10 snapshots)
- Save a snapshot program (quantity limited by program memory)

If a snapshot sequence is used instead of individually approaching every snapshot in a row, the time for reloading the program memory between the snapshots is omitted. This can save a lot of time.

If a snapshot program is used, pauses can be defined between snapshots, during which an external signal is awaited. The individual variants are explained in the following in more detail.

Approaching a Single Snapshot

Individual snapshots can be approached via the snapshots user interface and the *Virtual Interface*. When the *Virtual Interface* is used, the command looks e.g. as follows:

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

If several snapshots should be approached in a row, each must be called individually. As a snapshot must be loaded into the program memory of the robot prior to approaching it. So time will be lost here. In this case, a snapshot sequence makes more sense.

Approaching a Snapshot Sequence

To approach several snapshots in a row without losing time while reloading the program memory, a snapshot sequence can be used. It is possible to hand over up to 10 snapshots directly, which can then be approached directly without a break.

Example:

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

Saving and Executing a Snapshot Program

Similar to the snapshot sequence, the snapshot program offers the possibility to remove the reloading time of the program memory between snapshots. In addition, an external signal can be awaited. This way, snapshot sequences can be taught that can be cycled using an external signal to pin 6.

Example:

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

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

For this purpose, a program is saved first with “*WriteProgram*“. This must be performed once only, until the program should be changed.

Using *“STOP*“, all points are marked, where the controller should wait for an external signal. Thus, in the example, snapshots 1 to 3 are executed. Then, the controller waits for an external signal. Once it receives the signal, snapshots 4 and 5 are executed.

Using “*StartProgram*“, the program can be started at any time, once it is transferred. If the command is called, the keys for further robot control remain are locked, until the program is finished. However, the *Virtual Interface* is directly released again. This way, the calling program can continue processing. This also means that there is no direct return value indicating, whether the program is still running or an error occurred.

For this purpose, the “*SnapshotControl.ProgramStatus*“ function can be used as shown in the following example:

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

The following would be an application example:

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

 SetValue("SnapshotControl.StartProgram");


 while(GetString("SnapshotControl.ProgramStatus") == "Running")

 {

   // Wait until the program has finised or canceled.

 }

  Debug("Operation was finished with the following code: " +
GetString("SnapshotControl.ProgramStatus"));
```

Here, a program is saved and executed. Next, the program end is awaited in a “*while loop*“. If the program finished and no error occurred, the program status is returned as “*successful*“.

NOTE: If the program reached a point, where it waits for an external signal at pin 6, *“WaitPosition“* is returned. Alternatively, the program status can also be read out as code. The possible codes are described in more detail under chapter 5 Exit / Status Codes.

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

If it is required to change the speed during a snapshot program, the keyword “*SPEED*“ can be used.The following example shows how the speed is first changed and then changed back again to the standard value.

The standard value is defined as the speed value that is set in the options, if it wasn’t changed afterwards via the *Virtual Interface*.

NOTE: Only one speed less than the standard speed can be specified.

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

If the program should wait, until a wait / stop position was reached, the *“WaitForWaitPosition“* command can be used. A timeout can be added to this command.

The function returns “1“, if the position was reached, and “0“, if an error occurred, the function is not available, or the timeout was exceeded.

```
// Without Timeout
 int waitPositionReached1 = GetValue("SnapshotControl.WaitForWaitPosition");
 // With Timeout von 2000 ms
 int waitPositionReached2 = GetValue("SnapshotControl.WaitForWaitPosition.2000");
```

Using the *“StopProgram“* command, the program can be stopped while it is running. The Abort program could be executed next. In this case, the robot first moves up in Z direction and then to the handed-over snapshot.

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

Alternatively, instead of an external signal, the ”*SendImpuls”* function can be used.

If the program is at a wait / stop position, this function can send an impulse asking the robot to continue processing.

Example:

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

6.2.6. Input / Output Signals

Using the two *Virtual Interface* functions “*GetSignal”* and “*SetSignal”*, signals can be set and inquired at the slot card of the controller.

Setting Signals

Using “*SetSignal”*, the output signals are set. Here, it must be observed that some of the pins are already reserved by the controller and cannot be set.

The signals are set via a decimal 16 - bit value.

Example:

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

In this example, the first 8 bits would be set. However, some pins are already reserved by the controller.

Reading Out Input Signals

Using “*GetSignal“*, the input signals of the robot can be inquired. The entire signal or individual pins can be inquired.

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

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

Setting Pseudo Input Signal for Debugging Purposes

If a simulation is used, the first 15 input signals can be set for test purposes. This is not a direct robot functionality. Thus, it is realized via “*RobotControl.SetValues“.*

Example:

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

This example sets pin 6 to 1.

6.2.7. Collision Detection

Collision detection must be activated and adjusted so that the robot stops if it meets objects or obstacles.

CAUTION: Even if collision detection is activated, injuries and damage to objects can occur during initial contact!

If the sensitivity is set too low, the robot will not stop!

Activating Collision Detection

Collision detection can be activated via the Teaching Box or RT Toolbox 2. The respective controller parameter is "**COL**".

This parameter consists of three elements:

- Element 1: 1=Enable, 0=Disable
- Element 2: Activation during initialization; 1=Enable, 0=Disable
- Element 3: Activation during JOG operations; 1=Enable, 0=Disable

Adjsuting Sensitivity of Collision Detection

The collision detection sensitivity can be adjusted using the "**COLLVL**" parameter. Here, a value between 1 and 500 % can be set for every joint. The smaller the value, the higher the detection level.

Adjusting Collision Detection Sensitivity in JOG Mode

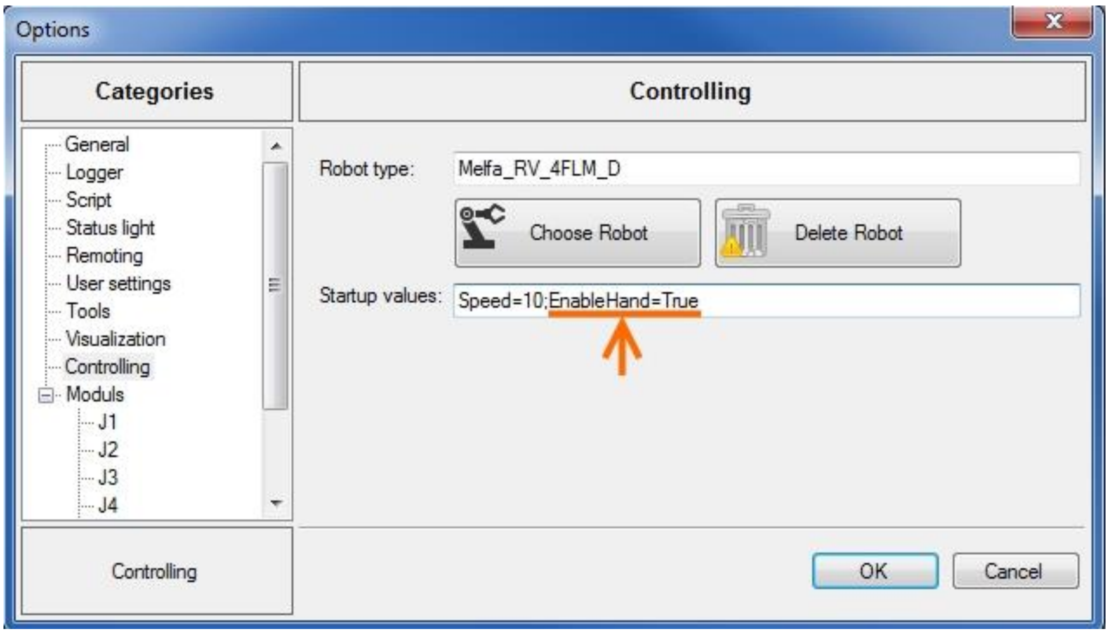
The sensitivity can be adjusted for the JOG Mode. This happens by using the parameter "**COLLVLJG**".

Values between 1 and 500 % can also be set here. The smaller the value, the higher the detection level.

Further information can be found in the robot instructions.

6.2.8. Using a Gripper

Using a gripper installed at the J6 joint is deactivated by default. The functionality can be activated via the *setup* using startup value "*EnableHand*".



Robot type: Melfa_RV_4FLM_D

Startup values: Speed=10;EnableHand=True

Figure 25: Using a Gripper

If the functionality is activated, the gripper can be opened and closed in 4 stages via the end effector interface.

Opening the Gripper

The gripper can be opened via the end effector interface (buttons marked in red here). Hereby, all 4 stages of the gripper are reset.

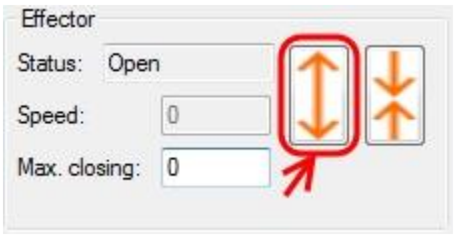


Figure 26: Open the Gripper via End Effector Interface

In the *Virtual Interface*, the gripper is opened using the *"Open"* command in the respective end effector.

Example:

```
SetEvent("Endeffector[0].Open"); // For an end effector with name Endeffektor[0]

SetEvent("J6End.Open"); // For an end effector with name J6End
```

Closing the Gripper

The gripper can be closed in 4 stages: 0, 1, 2, and 3.

The stage describes, how far the gripper should be closed:

0-2: Close only partly

3: Close completely

The four hand signals behave as follows:

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

The stage is adjusted via the "*Max. closing*" field. Closing speed is not implemented for the Melfa robot.

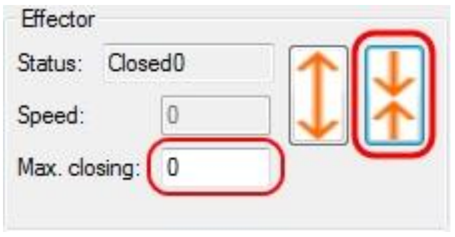


Figure 27: Closing the Gripper

In the *Virtual Interface*, the hand can be closed in two variants:

Example:

```
// Variante 1 (Stage/Max are set separately):
SetValue("Endeffector[0].Max", 3); // Set Stage to 3
SetEvent("Endeffector[0].Close"); // Close hand

// Variante 2:
SetValue("Endeffector[0].Close", 3); // Close hand to stage 3.
```

Inquiring the Gripper State

The gripper state is displayed on the user interface. It can also be inquired via the *Virtual Interface*. The Status command of the end effector is used for this purpose. The name of the end effector is defined via *setup* (default: *Endeffector[index]*).

Example:

```
// Status of the end effector with name Gripper
string status = GetString("Gripper.Status");

// Status of the end effector with name Endeffektor[0]
string status = GetString("Endeffektor[0].Status");

// Status of the end effector with name J6End
string status = GetString("J6End.Status");

...
```

If the gripper is open, “Open“ is returned. If the gripper is closed partly or completely, the stage is returned: *Closed0, Closed1, Closed2, or Closed3*.

Alternatively, the StatusCode method can be used. A list of possible status codes can be found under chapter 5 Exit / Status Codes.

Example:

```
// Status of the end effector with name Gripper
int statusCode = GetString("Gripper.StatusCode");

// Status of the end effector with name Endeffektors[0]
int statusCode = GetString("Endeffektor[0].StatusCode");

// Status of the end effector with name J6End
int statusCode = GetString("J6End.StatusCode");
```

Analysis of Gripper Input Signals

Using the presented methods, the 4 stages of the gripper can be controlled and inquired. The case is often that the first stage is used only. In this case, the inquiry, how far the hand is closed, is realized via external signals.

In general (also if all 4 stages are used), these signals can be inquired as follows:

First, parameter **HIOTYPE** in the robot controller must be set to 0 (source). Now, external signals can be inquired at ports 900 to 915.

The standard command for signals is used for this purpose.

Example:

```
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");
// ...
```

It is also possible to read out 16 signals at the same time. This is indicated by a leading minus sign in the "*GetSignal*" command. To inquire the entire gripper signal, the following command can be used:

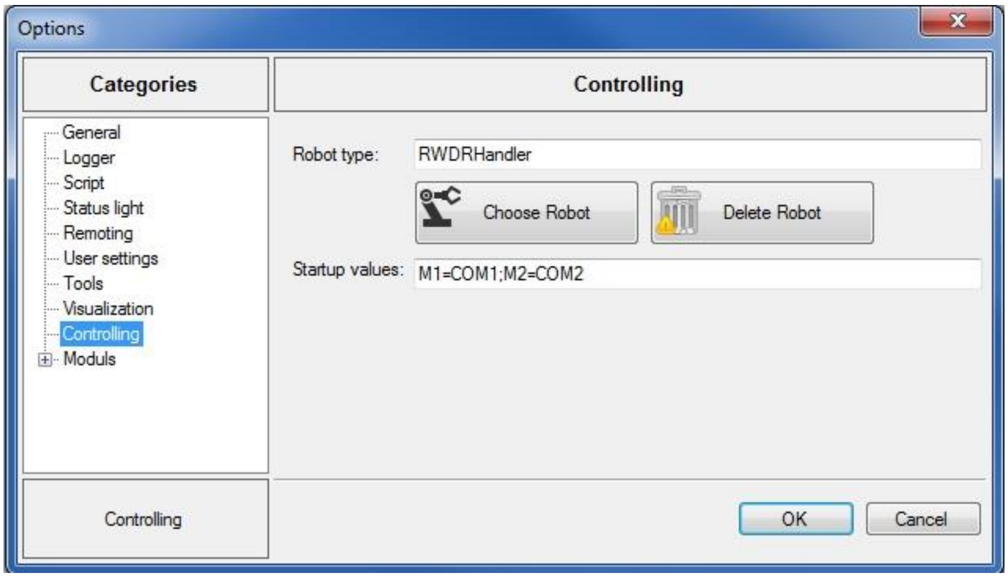
```
int signal = GetValue("RobotControl.GetSignal.-900");
```

A 16-bit decimal number is returned.

6.3. RWDRHandler

The Rotary-Shaft-Sealing-Handler of two linear axes that are moving in Z - direction. Each axis has its own motor, which is connected to the computer via serial port. The motors are ISEL motors.

Since it is possible that the serial ports of the motors change, it is necessary to specify them in the options menu. This is shown in the following image:



Controlling

Robot type: RWDRHandler

Startup values: M1=COM1;M2=COM2

Figure 28: Specify COM Interface

Furthermore, it is possible to adjust the following parameters:

- Motor speed **SPEED = [INTEGER > 0]**
- Acceleration **ACCEL = [INTEGER > 0]**
- Serial port timeout **PORTTIMEOUT = [INTEGER > 0]**
- Positioning timeout **POSTMEOUT = [INTEGER > 0]**

If these parameters are not specified, standard values are used. The standard values of the motor are always used for homing.

CAUTION: When controlling the motors by using the *Virtual Interface* of the Toolmonitor, **no** exceptions are shown while using the homing function or approaching positions. The error messages are redirected to the robot status which can be requested, using the following command:

```
// Integer-Code (see Exit-Codes)
int errorcode = GetString("RobotControl.CurrentRobotStatusCode");
// String-Code (see Exit-Codes)
string errorname = GetString("RobotControl.CurrentRobotStatusCode ");
```

7. Options

7.1. Controlling Options

The robot type is selected using the controlling options.

In addition, a value string can be defined, which is handed over to the robot during startup. Which values is handed can be taken from the individual descriptions of the robot types.

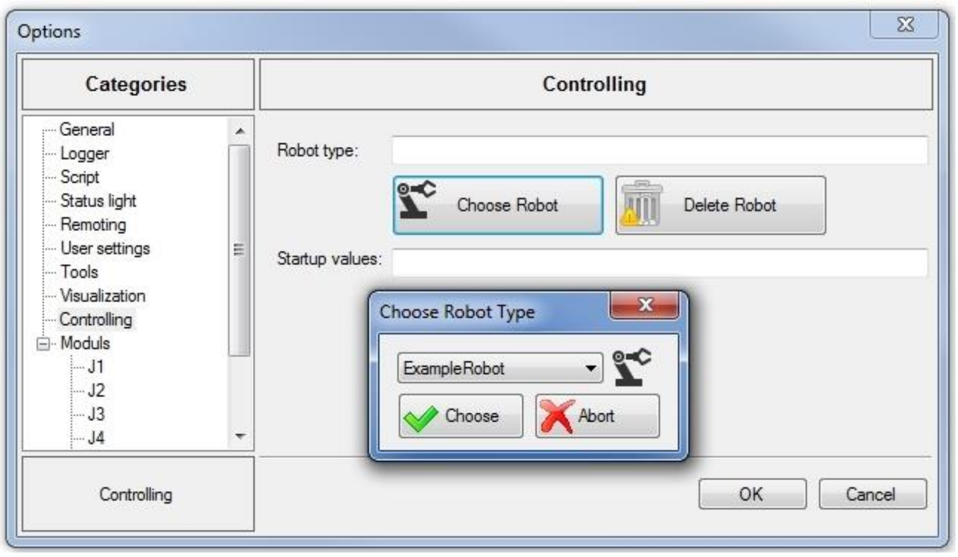
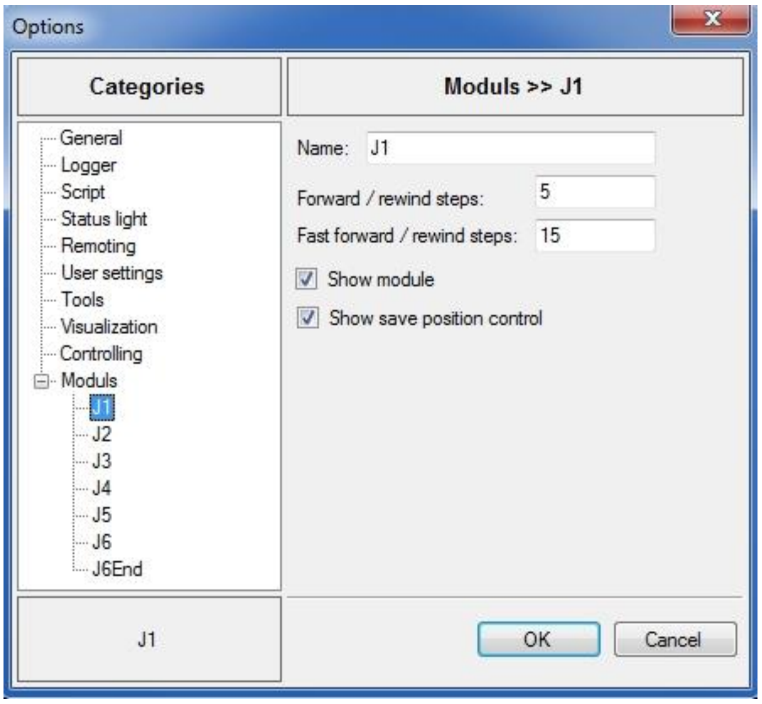


Figure 29: Controlling Options

7.2. Module Options

Using the Module Options, the name of a module can be set and the increment adjusted. It is also possible to hide complete modules or to deactivate their control for position saving.



Name: J1

Forward / rewind steps: 5

Fast forward / rewind steps: 15

☑ Show module

☑ Show save position control

Figure 30: Module Options