KUKA

Training

KUKA Roboter GmbH

Robot Programming 1

KUKA System Software 8

Training Documentation

Issued: 15.01.2015

Version: P1KSS8 robot programming 1 (R2) V4

KUKA Roboter GmbH
Zugspitzstraße 140
D-86165 Augsburg
Germany

Other functions not described in this documentation may be operable in the controller. The user has no claims to these functions, however, in the case of a replacement or service work.

We have checked the content of this documentation for conformity with the hardware and software described. Nevertheless, discrepancies cannot be precluded, for which reason we are not able to guarantee total conformity. The information in this documentation is checked on a regular basis, however, and necessary corrections will be incorporated in the subsequent edition.

Subject to technical alterations without an effect on the function.

Translation of the original documentation

KIM-PS5-DOC

Publication:	Pub COLLEGE P1KSS8 Roboterprogrammierung 1 (R2) (PDF-COL) en
Book structure:	P1KSS8 Roboterprogrammierung 1 (R2) V3.1
Version:	P1KSS8 robot programming 1 (R2) V4

Contents

1 Structure and function of a KUKA robot system 7

1.1 Overview 7
1.2 Robot basics 7
1.3 Robot arm of a KUKA robot 8
1.4 (V)KR C4 robot controller 10
1.5 The KUKA smartPAD 12
1.6 Overview of smartPAD 14
1.7 Robot programming 15
1.8 Robot safety 17

2 Moving the robot 19

2.1 Overview 19
2.2 Reading and interpreting robot controller messages 19
2.3 Selecting and setting the operating mode 21
2.4 Disconnecting the smartPAD 24
2.5 Moving individual robot axes 28
2.5.1 Exercise: Operator control and axis-specific jogging 34
2.6 Coordinate systems in conjunction with robots 35
2.7 Moving the robot in the world coordinate system 36
2.7.1 Exercise: Operator control and jogging in the world coordinate system 41

3 Starting up the robot 43

3.1 Overview 43
3.2 Start-up mode 43
3.3 Mastering principle 44
3.4 Mastering the robot 47
3.4.1 Exercise: Robot mastering 55
3.5 Loads on the robot 57
3.6 Tool load data 57
3.7 Monitoring tool load data 58
3.8 Supplementary loads on the robot 59
3.9 Moving the robot in the tool coordinate system 61
3.9.1 Exercise: Jogging in the tool coordinate system 65
3.10 Tool calibration 66
3.10.1 Exercise: Tool calibration – pen 75
3.10.2 Exercise: Tool calibration – gripper, 2-point method 78
3.11 Moving the robot in the base coordinate system 82
3.11.1 Exercise: Jogging in the base coordinate system 86
3.12 Base calibration 87
3.12.1 Exercise: Base calibration – table, 3-point method 92
3.13 Displaying the current robot position 94

4 Executing robot programs 97

4.1 Overview 97
4.2 Performing an initialization run 97
4.3 Selecting and starting robot programs 98
4.4 Exercise: Executing robot programs 105

5 Working with program files 107

5.1 Overview 107
5.2 Creating program modules 107
5.3 Editing program modules 108
5.4 Archiving and restoring robot programs 109
5.5 Tracking program modifications and changes of state by means of the logbook 110

6 Creating and modifying programmed motions 115

6.1 Overview 115
6.2 Correction of existing motion points 115
6.3 Creating cycle-time optimized motion (axis motion) 117
6.4 Exercise: Dummy program – program handling and SPTP motions 123
6.5 Creating CP motions 125
6.5.1 SCIRC: Orientation behavior – example: auxiliary point 137
6.5.2 SCIRC: Orientation behavior – example: end point 139
6.5.3 Restrictions for $CIRC_MODE 140
6.6 Modifying motion commands 141
6.7 Exercise: CP motion and approximate positioning 146

7 Programming spline motions 149

7.1 Overview 149
7.2 Programming spline blocks with inline forms 149
7.3 Velocity profile for spline motions 149
7.4 Modifications to spline blocks 152
7.5 Block selection with spline motions 155
7.6 Approximation of spline motions 156
7.7 Replacing an approximated CP motion with a spline block 157
7.7.1 SLIN-SPL-SLIN transition 159
7.8 Programming CP SPLINE blocks via inline forms 160
7.9 Programming PTP SPLINE blocks via inline form 163
7.10 Exercise: Path contour with spline block 166

8 Using logic functions in the robot program 169

8.1 Overview 169
8.2 Introduction to logic programming 169
8.3 Programming wait functions 170
8.4 Programming simple switching functions 174
8.5 Logic programming with SPLINE 177
8.5.1 Programming the spline trigger 178
8.5.1.1 Reference point for approximate positioning – overview 182
8.5.1.2 Reference point for homogenous approximate positioning 182
8.5.1.3 Reference point for mixed approximate positioning (spline) 184
8.5.1.4 Reference point for mixed approximate positioning (LIN/CIRC/PTP) 185
8.5.1.5 Constraints for functions in the trigger 185
8.5.2 Programming a conditional stop 185
8.5.3 Programming the constant velocity range 188
8.5.3.1 Block selection to the constant velocity range 190
8.5.3.2 Maximum limits 191
8.6 Exercise: Motion programming with spline 193

9 Using technology packages 195
9.1 Overview 195
9.2 Gripper operation with KUKA.GripperTech 195
9.3 Gripper programming with KUKA.GripperTech 196
9.4 KUKA.GripperTech configuration 199
9.5 Exercise: Gripper programming – plastic panel 202
10 Configuration and programming of external tools 205
10.1 Overview 205
10.2 Moving the robot 205
10.2.1 Jogging with a fixed tool 205
10.2.2 Exercise: Jogging with a fixed tool 207
10.3 Starting up the robot 208
10.3.1 Calibration of a fixed tool 208
10.3.2 Calibration of a robot-guided workpiece 210
10.3.3 Exercise: Calibrating an external tool and robot-guided workpiece 212
10.4 Creating and modifying a programmed motion 217
10.4.1 Motion programming with external TCP 217
10.4.2 Exercise: Motion programming with external TCP 218
11 Introduction to Expert level 219
11.1 Overview 219
11.2 Using Expert level 219
11.3 Structuring robot programs 221
11.4 Linking robot programs 224
11.5 Exercise: Programming a subprogram call 227
12 Variables and declarations 229
12.1 Overview 229
12.2 Data management in KRL 229
12.3 Working with simple data types 232
12.3.1 Declaration of variables 232
12.3.2 Initialization of variables with simple data types 234
12.3.3 Manipulation of variable values of simple data types with KRL 236
12.4 Displaying variables 239
12.5 Exercise: Simple data types 241
13 Using program execution control functions 243
13.1 Overview 243
13.2 Programming loops 243
13.2.1 Programming an endless loop 243
13.2.2 Programming a counting loop 245
13.2.3 Programming a rejecting loop 247
13.2.4 Programming a non-rejecting loop 248
13.3 Programming conditional statements or branches 250
13.4 Programming a switch statement (SWITCH – CASE) 252
13.5 Programming a jump command 254
13.6 Programming wait functions in KRL 255
13.6.1 Time-dependent wait function 256

13.6.2 Signal-dependent wait function 256

14 Working with a higher-level controller 259

14.1 Overview 259
14.2 Preparation for program start from PLC 259
14.3 Adapting the PLC interface (Cell.src) 261
14.4 Questions: Working on a higher-level controller 263

15 Appendix 265

15.1 Programming motions with collision detection 265
15.2 Programming time-distance functions 270
15.3 Configuring and implementing Automatic External 277
15.4 Abbreviations 285
15.5 Additional exercises 286
15.5.1 Exercise: Gripper programming – pen 287
15.5.2 Exercise: Use of loops 289
15.5.3 Exercise: Constant velocity range and conditional stop 290
15.5.4 Exercise: Automatic External 291

Index 293

1 Structure and function of a KUKA robot system

1.1 Overview

The following contents are explained in this training module:

- Introduction to robotics
- Robot arm of a KUKA robot
- KR C4 robot controller
- KUKA smartPAD
- Robot programming
- Robot safety

1.2 Robot basics

What is a robot?

- The term *robot* comes from the Slavic word *robota*, meaning *hard work*.
- According to the official definition of an industrial robot: “A robot is a freely programmable, program-controlled handling device”.
- The robot system thus also includes the controller and the operator control device, together with the connecting cables and software.

Fig. 1-1: Industrial robot

1 Controller ((V)KR C4 control cabinet)
2 Manipulator (robot arm)
3 Teach pendant (KUKA smartPAD)

Everything outside the system limits of the industrial robot is referred to as the *periphery*:

- Tooling (end effector/tool)

- Safety equipment
- Conveyor belts
- Sensors
- Machines
- etc.

1.3 Robot arm of a KUKA robot

What is a manipulator?

The manipulator is the actual robot arm. It consists of a number of moving links (axes) that are linked together to form a “kinematic chain”.

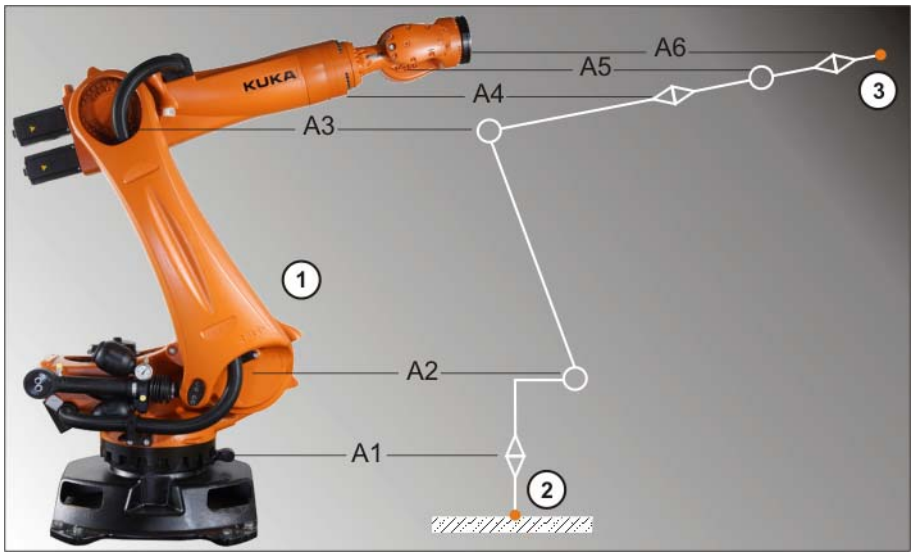


Fig. 1-2: Manipulator

1	Manipulator (robot arm)
2	Start of the kinematic chain: base of the robot (ROBROOT)
3	Free end of the kinematic chain: flange (FLANGE)
A1...A6	Robot axes 1 to 6

The individual axes are moved by means of targeted actuation of servomotors. These are linked to the individual components of the manipulator via reduction gears.

Fig. 1-3: Overview of manipulator components

1	Base frame	4	Link arm
2	Rotating column	5	Arm
3	Counterbalancing system	6	Wrist

The components of a robot arm consist primarily of cast aluminum and steel. In isolated cases, carbon-fiber components are also used.

The individual axes are numbered from bottom (robot base) to top (robot flange):

Fig. 1-4: Axes of a KUKA robot

Excerpt from the technical data of manipulators from the KUKA product range:

- **Number of axes:**
  - *4 axes* for KR 40 PA
  - *6 axes* for standard vertical jointed-arm robots
  - *7 axes* for lightweight robots
- **Reach:** from 0.7 m (KR 6 R700) to 3.9 m (KR 120 R3900 ultra K)
- **Weight:** from 23 kg (LBR iiwa 7 R800) to 4700 kg (KR 1000 Titan).
- **Accuracy:** 0.03 mm to 0.2 mm repeatability.

The axis ranges of main axes A1 to A3 and wrist axis A5 of the robot are limited by means of mechanical end stops with a buffer.

Axis 1	Axis 2	Axis 3
		

Additional mechanical end stops can be installed on the external axes.

NOTICE If the robot or an external axis hits an obstruction or a buffer on the mechanical end stop or axis range limitation, this can result in material damage to the robot system. KUKA Roboter GmbH must be consulted before the robot system is put back into operation. The affected buffer must be replaced with a new one before robot operation is resumed. If a robot (or external axis) collides with a buffer at more than 250 mm/s, the robot (or external axis) must be exchanged or recommissioning must be carried out by the KUKA Roboter GmbH.

1.4 (V)KR C4 robot controller

Who controls motion?

The manipulator is moved by means of servomotors controlled by the (V)KR C4 controller.

Fig. 1-5: (V)KR C4 control cabinet

Properties of the (V)KR C4 controller

- Robot control (path planning): control of six robot axes plus up to two external axes with the
  (V)KR C4 controller

Fig. 1-6: (V)KR C4 axis control

- Robot control (path planning): control of six robot axes plus up to six external axes with the
  (V)KR C4 extended controller

Fig. 1-7: (V)KR C4 extended axis control

- Optional sequence control: integrated Soft PLC KUKA.PLC in accordance with IEC61131
- Safety controller
- Motion control
- Communication options via bus systems (e.g. ProfiNet, Ethernet IP, Interbus):
  - Programmable logic controllers (PLC)
  - Additional controllers
  - Sensors and actuators
- Communication options via network:
  - Host computer
  - Additional controllers
  - Service notebook

Fig. 1-8: (V)KR C4 communication options

1.5 The KUKA smartPAD

How is a KUKA robot operated?

The KUKA robot is operated by means of the KUKA smartPAD teach pendant.

Fig. 1-9

Features of the KUKA smartPAD:

- Touch screen (touch-sensitive user interface) for operation by hand or using the integrated stylus
- Large display in portrait format
- KUKA menu key
- +/- keys that can be used flexibly (e.g. as jog keys)
- Keys for operator control of the technology packages
- Program execution keys (Stop/Backwards/Forwards)
- Key for displaying the keypad
- Keyswitch for changing the operating mode
- EMERGENCY STOP button
- Space Mouse
- Detachable
- USB connection

1.6 Overview of smartPAD

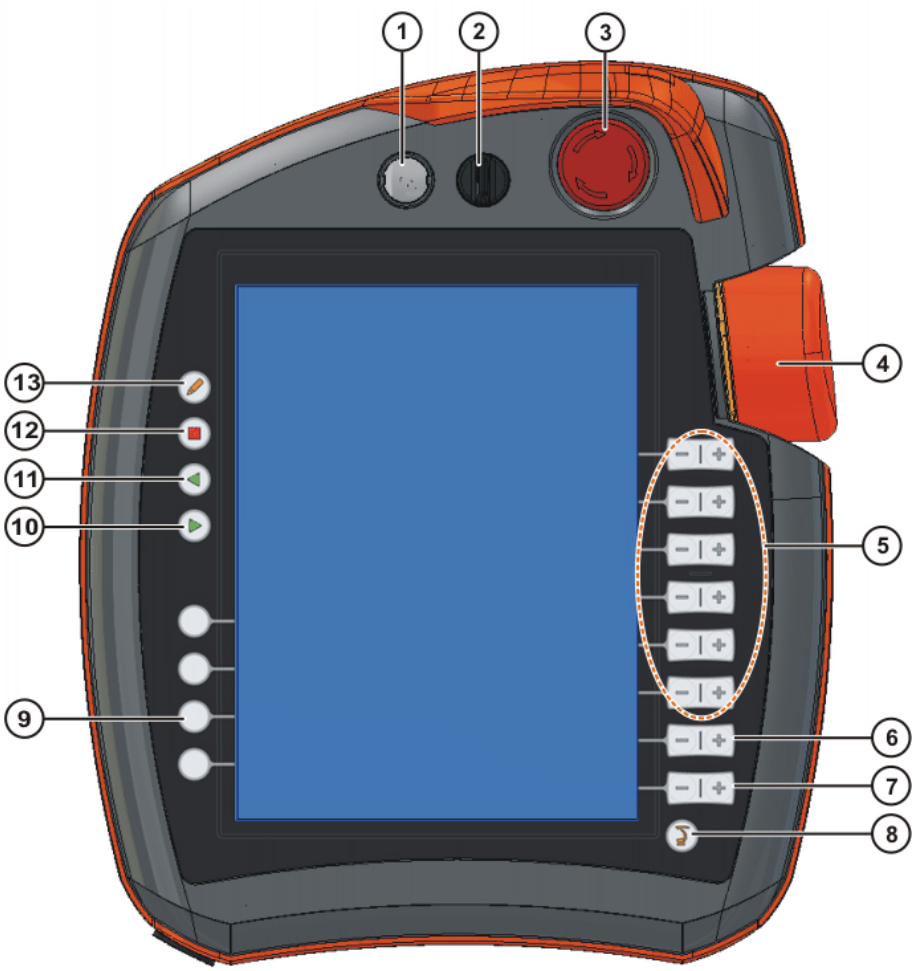


Fig. 1-10

Item	Description
1	Button for disconnecting the smartPAD
2	Keyswitch for calling the connection manager. The switch can only be turned if the key is inserted. The operating mode can be changed by using the connection manager.
3	EMERGENCY STOP button. Stops the robot in hazardous situations. The EMERGENCY STOP button locks itself in place when it is pressed.
4	Space Mouse: For moving the robot manually.
5	Jog keys: For moving the robot manually.
6	Key for setting the program override
7	Key for setting the jog override

Item	Description
8	Main menu key: Shows the menu items on the smartHMI
9	Status keys. The status keys are used primarily for setting parameters in technology packages. Their exact function depends on the technology packages installed.
10	Start key: The Start key is used to start a program.
11	Start backwards key: The Start backwards key is used to start a program backwards. The program is executed step by step.
12	STOP key: The STOP key is used to stop a program that is running.
13	Keyboard key Displays the keyboard. It is generally not necessary to press this key to display the keyboard, as the smartHMI detects when keyboard input is required and displays the keyboard automatically.

1.7 Robot programming

A robot is programmed so that motion sequences and processes can be executed automatically and repeatedly. For this, the controller requires a large amount of information:

- Current robot position = position of the current tool (Tool) in the current space (Base)
- Type of motion
- Velocity / acceleration
- Signal information for wait conditions, branches, dependencies, etc.

What language does the controller speak?

The programming language is **KRL** - **K**UKA **R**obot **L**anguage

Simple programs are created using predefined forms (inline forms). KRL is used for loops.

Example program:

```
LOOP
SPTP P1 Vel=100% PDAT1 Tool[2] Base[4]
SPTP P2 Vel=100% PDAT2 Tool[2] Base[4]
SPTP P3 Vel=100% PDAT3 Tool[2] Base[4]
SLIN P4 Vel=2m/s CPDAT4 Tool[2] Base[4]
...
ENDLOOP
```

How is a KUKA robot programmed?

Various programming methods can be used for programming a KUKA robot:

- **Online programming** with the teaching method.

Fig. 1-11: Robot programming with the KUKA smartPAD

- **Offline programming**
  - **Interactive, graphics-based programming:** simulation of the robot process

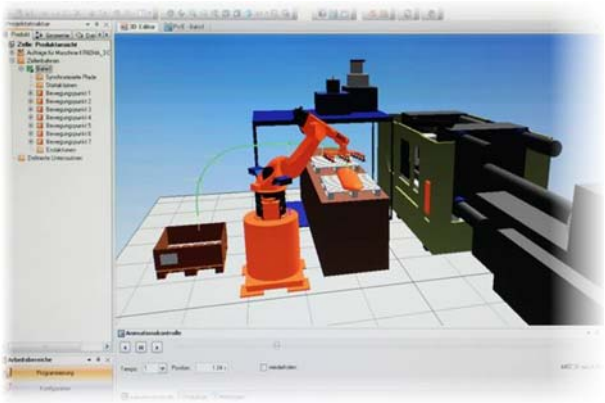


Fig. 1-12: Simulation with KUKA Sim

  - **Text-based programming:** programming with the aid of the smartPAD user interface display on a higher-level control PC (also for diagnosis, online adaptation of programs that are already running)

Fig. 1-13: Robot programming with KUKA OfficeLite

1.8 Robot safety

A robot system must always have suitable safety features. These include, for example, physical safeguards (fences, gates, etc.), EMERGENCY STOP buttons, dead-man switches, axis range limitations, etc.

Example: College training cell

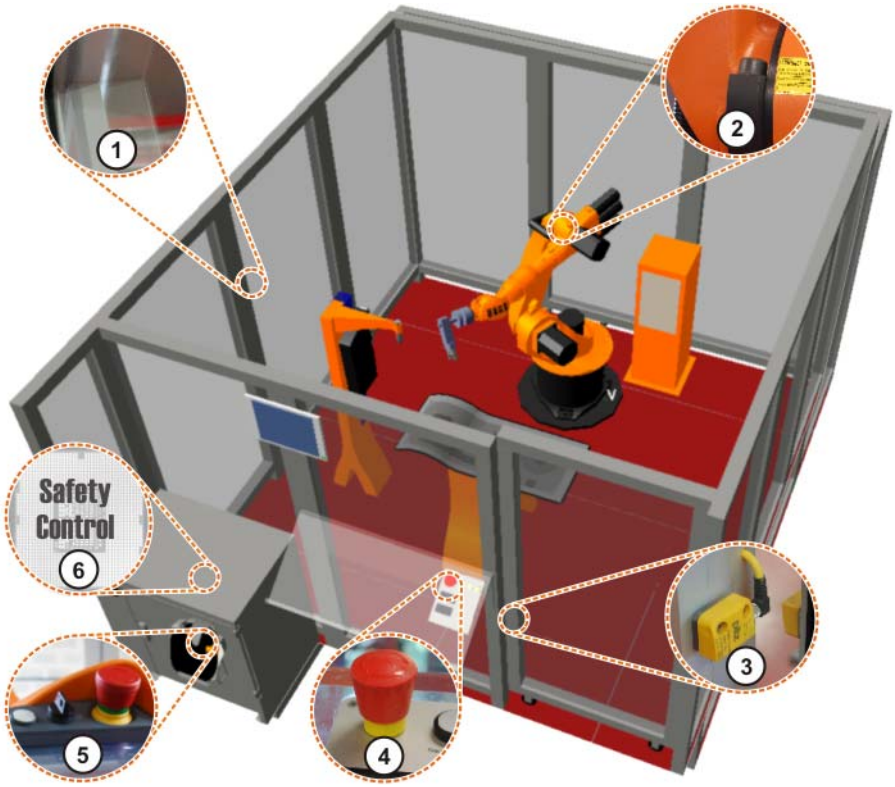


Fig. 1-14: Training cell

1 Safety fence
2 Mechanical end stops or axis range limitation for axes 1, 2 and 3
3 Safety gate with contact for monitoring the closing function
4 EMERGENCY STOP button (external)
5 EMERGENCY STOP button, enabling switch, keyswitch for calling the connection manager
6 Integrated (V)KR C4 safety controller

⚠ DANGER In the absence of functional safety equipment and safeguards, the robot system can cause personal injury or material damage. If safety equipment or safeguards are dismantled or deactivated, the robot system may not be operated.

EMERGENCY STOP device

The EMERGENCY STOP device for the industrial robot is the EMERGENCY STOP button on the KCP. The button must be pressed in the event of a hazardous situation or emergency.

Reactions of the industrial robot if the EMERGENCY STOP button is pressed:

- The manipulator and any external axes (optional) are stopped with a safety stop 1.

Before operation can be resumed, the EMERGENCY STOP button must be turned to release it and the ensuing stop message must be acknowledged.

⚠ WARNING Tools and other equipment connected to the manipulator must be integrated into the EMERGENCY STOP circuit on the system side if they could constitute a potential hazard.
Failure to observe this precaution may result in death, severe injuries or considerable damage to property.

There must always be at least one external EMERGENCY STOP device installed. This ensures that an EMERGENCY STOP device is available even when the KCP is disconnected.

External E-STOP

There must be EMERGENCY STOP devices available at every operator station that can initiate a robot motion or other potentially hazardous situation. The system integrator is responsible for ensuring this.

There must always be at least one external EMERGENCY STOP device installed. This ensures that an EMERGENCY STOP device is available even when the KCP is disconnected.

External EMERGENCY STOP devices are connected via the customer interface. External EMERGENCY STOP devices are not included in the scope of supply of the industrial robot.

Operator safety

The **operator safety** signal is used for interlocking physical safeguards, e.g. safety gates. Automatic operation is not possible without this signal. In the event of a loss of signal during automatic operation (e.g. safety gate is opened), the manipulator stops with a safety stop 1.

Operator safety is not active in the test modes T1 (Manual Reduced Velocity) and T2 (Manual High Velocity).

⚠ WARNING Following a loss of signal, automatic operation may only be resumed when the safeguard has been closed and when the closing has been acknowledged. This acknowledgement is to prevent automatic operation from being resumed inadvertently while there are still persons in the danger zone, e.g. due to the safety gate closing accidentally.
The acknowledgement must be designed in such a way that an actual check of the danger zone can be carried out first. Other acknowledgement functions (e.g. an acknowlegement which is automatically triggered by closure of the safeguard) are not permitted.
The system integrator is responsible for ensuring that these criteria are met. Failure to met them may result in death, severe injuries or considerable damage to property.

Safe operational stop

The safe operational stop can be triggered via an input on the customer interface. The state is maintained as long as the external signal is FALSE. If the external signal is TRUE, the manipulator can be moved again. No acknowledgement is required.

External safety stop 1 and external safety stop 2

Safety stop 1 and safety stop 2 can be triggered via an input on the customer interface. The state is maintained as long as the external signal is FALSE. If the external signal is TRUE, the manipulator can be moved again. No acknowledgement is required.

2 Moving the robot

2.1 Overview

The following contents are explained in this training module:

- Reading and interpreting messages
- Selecting and setting operating modes
- Moving individual robot axes
- Moving the robot in the world coordinate system
- Disconnecting the smartPAD
- Moving the robot in the tool coordinate system
- Moving the robot in the base coordinate system

2.2 Reading and interpreting robot controller messages

Overview of messages

Fig. 2-1: Message window and message counter

1 Message window: the current message is displayed.
2 Message counter: number of messages of each message type.

The controller communicates with the operator via the message window. It has five different message types:

Overview of message types:

Icon	Type
	**Acknowledgement message** ■ Displays states that require confirmation by the operator before program execution is resumed (e.g. “Ackn. EMERGENCY STOP”). ■ An acknowledgement message always causes the robot to stop or not to start.
	**Status message** ■ Status messages signal current controller states (e.g. “EMERGENCY STOP”). ■ Status messages cannot be acknowledged while the status is active.

Icon	Type
	**Notification message** ■ Notification messages provide information for correct operator control of the robot (e.g. “Start key required”). ■ Notification messages can be acknowledged. They do not need to be acknowledged, however, as they do not stop the controller.
	**Wait message** ■ Wait messages indicate the event the controller is waiting for (status, signal or time). ■ Wait messages can be canceled manually by pressing the “Simulate” button.

NOTICE The command “Simulate” may only be used if there is no risk of a collision or other hazards!

	**Dialog message** ■ Dialog messages are used for direct communication with the operator, e.g. to ask the operator for information. ■ A message window with buttons appears, offering various possible responses.

An acknowledgeable message can be acknowledged with **OK**. All acknowledgeable messages can be acknowledged at once with **All OK**.

Influence of messages

Messages influence the functionality of the robot. An acknowledgement message always causes the robot to stop or not to start. The message must be acknowledged before the robot can be moved.

The command **OK** (acknowledge) represents a prompt to the operator, forcing a conscious response.

Tips for dealing with messages:

- Read attentively!
- Read older messages first. A newer message could simply be a follow-up to an older one.
- Do not simply press “All OK”.
- Particularly after booting: look through the messages. Display all messages. Touching the message window expands the message list.

Dealing with messages

Messages are always displayed with the date and time in order to be able to trace the exact time of the event.

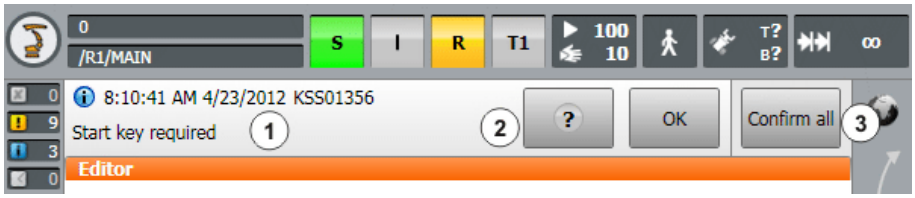


Fig. 2-2: Acknowledging messages

Procedure for viewing and acknowledging messages:

1. Touch the message window to expand the message list.
2. Acknowledge:

- Acknowledge individual messages with **OK**.
- Alternatively: acknowledge all messages with **All OK**.

3. Touching the top message again or the “X” at the left-hand edge of the screen closes the message list.

2.3 Selecting and setting the operating mode

Operating modes of a KUKA robot

- T1 (Manual Reduced Velocity)
  - For test operation, programming and teaching
  - Velocity in program mode max. 250 mm/s
  - Velocity in jog mode max. 250 mm/s
- T2 (Manual High Velocity)
  - For test operation
  - Velocity in program mode corresponds to the programmed velocity!
  - Jog mode: not possible
- AUT (Automatic)
  - For industrial robots without higher-level controllers
  - Velocity in program mode corresponds to the programmed velocity!
  - Jogging with the jog keys or Space Mouse is not possible.
- AUT EXT (Automatic External)
  - For industrial robots with higher-level controllers (PLC)
  - Velocity in program mode corresponds to the programmed velocity!
  - Jog mode: not possible

Safety instructions – operating modes

Manual mode T1 and T2
Manual mode is the mode for setup work. Setup work is all the tasks that have to be carried out on the robot system to enable automatic operation. This includes:

- Teaching/programming
- Executing a program in jog mode (testing/verification)

New or modified programs must always be tested first in **Manual Reduced Velocity mode (T1)**.

In **Manual Reduced Velocity mode (T1)**:

- Operator safety (safety gate) is not monitored.
- The number of people inside the safeguarded space must be reduced to a minimum.

  If it is necessary for there to be several persons inside the safeguarded area, the following must be observed:
  - All persons must have an unimpeded view of the robot system.
  - Eye-contact between all persons must be possible at all times.
- The operator must be so positioned that he can see into the danger area and get out of harm’s way.

In **Manual High Velocity mode (T2)**:

- Operator safety (safety gate) is not monitored.

Unlike in standard systems, however, the safety gate in KUKA College training cells is monitored and must be closed.

- This mode may only be used if the application requires a test at a velocity higher than manual reduced velocity.

- Teaching is not possible in this operating mode.
- Before commencing the test, the operator must ensure that the enabling devices are operational.
- The programmed velocity is reached when programs are executed in T2 mode.
- The operator and other persons must be positioned outside the danger zone.

Operating modes **Automatic** and **Automatic External**

- Safety equipment and safeguards must be present and fully operational.
- All persons are outside the safeguarded area.

Stop reactions

Stop reactions of the industrial robot are triggered in response to operator actions or as a reaction to monitoring functions and error messages. The following tables show the different stop reactions according to the operating mode that has been set.

Trigger	T1, T2	AUT, AUT EXT
Start key released	STOP 2	-
STOP key pressed	STOP 2	
Drives OFF	STOP 1	
“Motion enable” input drops out	STOP 2	
Robot controller switched off (power failure)	STOP 0	
Internal error in non-safety-oriented part of the robot controller	STOP 0 or STOP 1 (dependent on the cause of the error)	
Operating mode changed during operation	Safety stop 2	
Safety gate opened (operator safety)	-	Safety stop 1
Enabling switch released	Safety stop 2	-
Enabling switch pressed fully down or error	Safety stop 1	-
E-STOP pressed	Safety stop 1	
Error in safety controller or periphery of the safety controller	Safety stop 0	

Term	Description
Safe operational stop	The safe operational stop is a standstill monitoring function. It does not stop the robot motion, but monitors whether the robot axes are stationary. If these are moved during the safe operational stop, a safety stop STOP 0 is triggered. The safe operational stop can also be triggered externally. If a safe operational stop is triggered, the robot controller signals a safe output (X13, SIB extended) or alternatively via a safe field bus protocol to a higher-level controller. The output is set even if not all the axes were stationary at the time of triggering, thereby causing a safety stop STOP 0 to be triggered.
Safety STOP 0	A stop that is triggered and executed by the safety controller. The safety controller immediately switches off the drives and the power supply to the brakes. **Note:** This stop is called safety STOP 0 in this document.
Safety STOP 1	A stop that is triggered and monitored by the safety controller. The braking process is performed by the non-safety-oriented part of the robot controller and monitored by the safety controller. As soon as the manipulator is at a standstill, the safety controller switches off the drives and the power supply to the brakes. When a safety STOP 1 is triggered, the robot controller sets an output via the field bus. The safety STOP 1 can also be triggered externally. **Note:** This stop is called safety STOP 1 in this document.
Safety STOP 2	A stop that is triggered and monitored by the safety controller. The braking process is performed by the non-safety-oriented part of the robot controller and monitored by the safety controller. The drives remain activated and the brakes released. As soon as the manipulator is at a standstill, a safe operational stop is triggered. When a safety STOP 2 is triggered, the robot controller sets an output via the field bus. The safety STOP 2 can also be triggered externally. **Note:** This stop is called safety STOP 2 in this document.
Stop category 0	The drives are deactivated immediately and the brakes are applied. The manipulator and any external axes (optional) perform path-oriented braking. **Note:** This stop category is called STOP 0 in this document.
Stop category 1	The drives are deactivated after 1 s and the brakes are applied. The manipulator and any external axes (optional) perform path-maintaining braking. **Note:** This stop category is called STOP 1 in this document.
Stop category 2	The drives are not deactivated and the brakes are not applied. The manipulator and any external axes (optional) are braked with a path-maintaining braking ramp. **Note:** This stop category is called STOP 2 in this document.

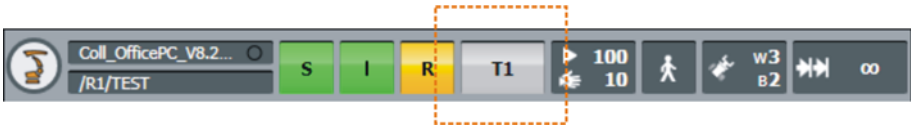
Procedure

If the operating mode is changed during operation, the drives are immediately switched off. The industrial robot stops with a safety stop 2.

1. On the KCP, turn the switch for the connection manager. The connection manager is displayed.

2. Select the operating mode.

3. Return the switch for the connection manager to its original position.
   The selected operating mode is displayed in the status bar of the smartPAD.



2.4 Disconnecting the smartPAD

Description of smartPAD disconnection

- The smartPAD can be disconnected while the robot controller is running.
- A smartPAD can be connected at any time.
- The connected smartPAD assumes the current operating mode of the robot controller.
- The smartPAD variant (firmware version) connected is irrelevant, as it is automatically upgraded or downgraded.
- The EMERGENCY STOP and enabling switches are not operational again until 30 s after connection.
- The smartHMI (user interface) is automatically displayed again (this takes no longer than 15 s).

smartPAD disconnection function

-

⚠ WARNING If the smartPAD is disconnected, the system can no longer be switched off by means of the EMERGENCY STOP button on the smartPAD. For this reason, an external EMERGENCY STOP must be connected to the robot controller.

-

⚠ WARNING The operator must ensure that disconnected smartPADs are immediately removed from the system and stored out of sight and reach of personnel working on the industrial robot. This serves to prevent operational and non-operational EMERGENCY STOP devices from becoming interchanged.

-

⚠ WARNING Failure to observe these precautions may result in death to persons, severe physical injuries or considerable damage to property.

-

⚠ WARNING The user connecting a smartPAD to the robot controller must subsequently stay with the smartPAD for at least 30 s, i.e. until the EMERGENCY STOP and enabling switches are operational once again. This prevents another user from trying to activate a non-operational EMERGENCY STOP in an emergency situation, for example.

Procedure for disconnecting a smartPAD

Disconnection:

1. Press the disconnect button on the smartPAD.
   A message and a counter are displayed on the smartHMI. The counter runs for 25 s. During this time, the smartPAD can be disconnected from the robot controller.

Fig. 2-3: Button for disconnecting the smartPAD

NOTICE If the smartPAD is disconnected without the counter running, this triggers an EMERGENCY STOP. The EMERGENCY STOP can only be canceled by plugging the smartPAD back in.

2. Open the door of the (V)KR C4 control cabinet.
3. Disconnect the smartPAD from the robot controller.

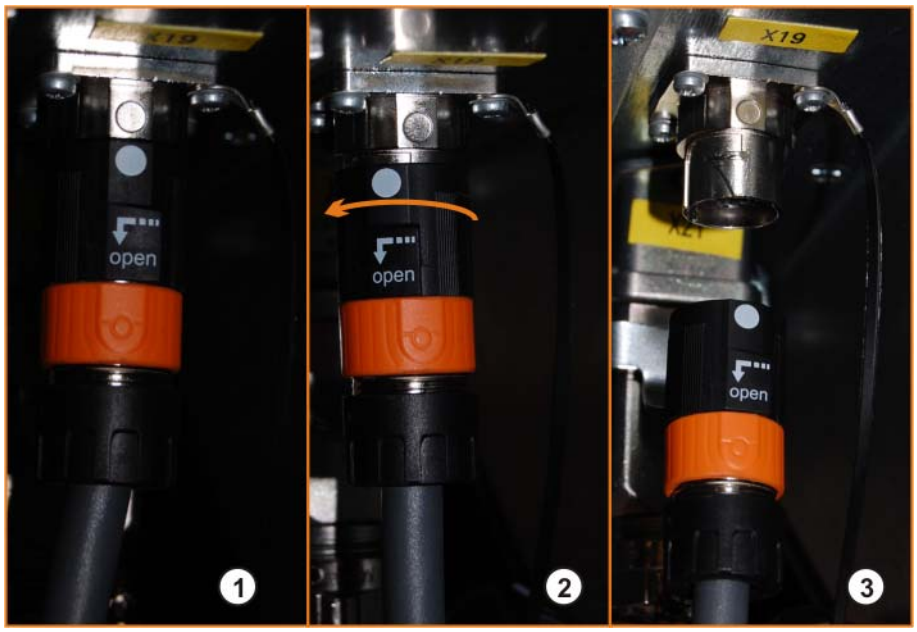


Fig. 2-4: Disconnecting the smartPAD

1	Connector connected
2	Turn the upper black part approx. 25° in the direction indicated by the arrow.
3	Pull the connector downwards.

4. Close the door of the (V)KR C4 control cabinet.

If the counter expires without the smartPAD having been disconnected, this has no effect. The disconnect button can be pressed as often as required to display the counter.

Connection:

1. Ensure that the same smartPAD variant is used again.
2. Open the door of the (V)KR C4 control cabinet.
3. Connect the smartPAD connector.

NOTICE Pay heed to the markings on the female connector and the smartPAD connector.

Fig. 2-5: Connecting the smartPAD

1	Connector disconnected (observe marking)
2	Push connector upwards. The upper black part automatically turns approx. 25° while it is being pushed up.
3	The connector automatically locks in place, i.e. the markings are aligned.

⚠ WARNING The user connecting a smartPAD to the robot controller must subsequently stay with the smartPAD for at least 30 s until the EMERGENCY STOP and enabling switches are operational once again. This prevents another user from trying to activate a non-operational EMERGENCY STOP in an emergency situation, for example.

4. Close the door of the (V)KR C4 control cabinet.

2.5 Moving individual robot axes

Description: Axis-specific motion

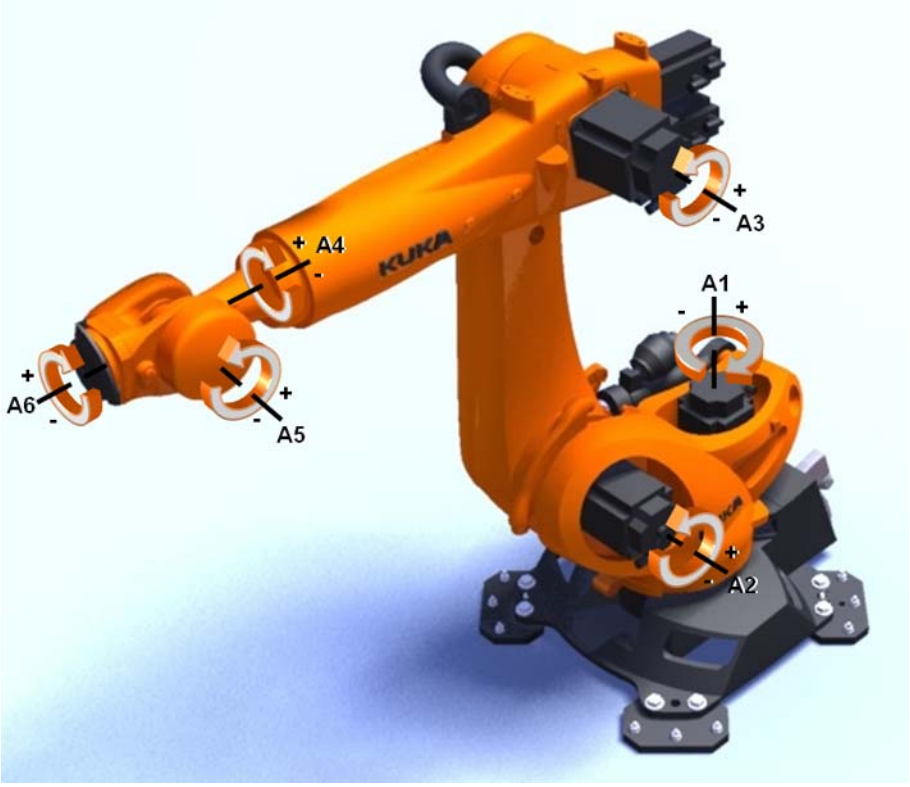


Fig. 2-6: Axes of a KUKA robot

Moving robot axes

- Jog each axis individually in the plus and minus direction.
- The jog keys or Space Mouse of the KUKA smartPAD are used for this.
- The velocity can be modified (jog override: HOV).
- Jogging is only possible in T1 mode.
- The enabling switch must be pressed.

Principle

- The drives are activated by pressing the enabling switch. If the drives are enabled, the labeling of the jog keys turns green. As soon as a jog key or the Space Mouse is pressed, servo control of the robot axes starts and the desired motion is executed.
- Continuous motion and incremental motion are possible. The size of the increment must be selected in the status bar. (>>> "Procedure" Page 30)

The following messages influence manual operation:

Message	Cause	Remedy
“Active commands inhibited”	A (STOP) message or state is present which inhibits active commands (e.g. EMERGENCY STOP pressed or drives not yet ready).	Release EMERGENCY STOP and/or acknowledge messages in the message window. As soon as an enabling switch is pressed, the drives are available immediately.
“Software limit switch A5”	The robot has moved to the software limit switch of the axis indicated (e.g. A5) in the direction indicated (+ or -).	Move the indicated axis in the opposite direction.

Safety instructions relating to axis-specific jogging

Operating mode

Manual operation of the robot is only possible in T1 mode (Manual Reduced Velocity). The maximum jog velocity in T1 is 250 mm/s. The operating mode is set via the connection manager.

Enabling switches

In order to be able to jog the robot, an enabling switch must be pressed. There are three enabling switches installed on the smartPAD. The enabling switches have three positions:

- Not pressed
- Center position
- Panic position

Software limit switches

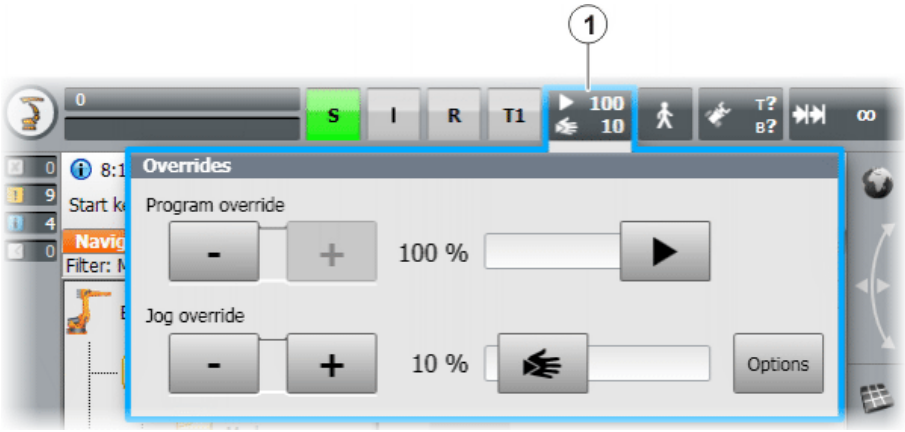
The motion of the robot is also limited in axis-specific jogging by means of the maximum positive and negative values of the software limit switches.

NOTICE If the message “Perform mastering” appears in the message window, these limits can be exceeded. This can result in damage to the robot system!

Procedure: Executing an axis-specific motion

1. Select **Axis** as the option for the jog keys.

2. Set jog override.



3. Press the enabling switch into the center position and hold it down.

4. Axes A1 to A6 are displayed next to the jog keys.
   Press the Plus or Minus jog key to move an axis in the positive or negative direction.

Incremental jogging

Description

Incremental jogging makes it possible to move the robot a defined distance, e.g. 10 mm or 3°. The robot then stops by itself.

Incremental jogging can be activated for jogging with the jog keys. Incremental jogging is not possible in the case of jogging with the Space Mouse.

Areas of application:

- Positioning of equidistant points
- Moving a defined distance away from a position, e.g. in the event of a fault
- Mastering with the dial gauge

Precondition

- The jog mode “Jog keys” is active.
- Operating mode T1

Procedure

1. Select the size of the increment in the status bar:

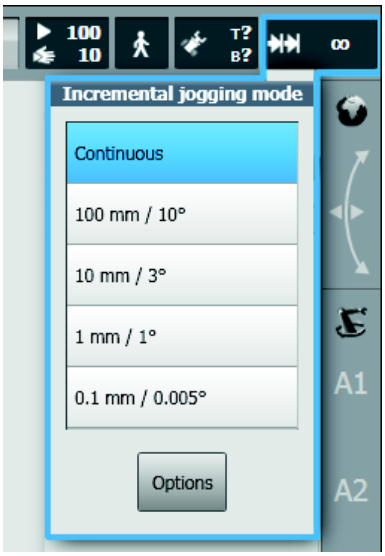


Fig. 2-7: Incremental jogging

2. Jog the robot using the jog keys. Jogging can be Cartesian or axis-specific.

   Once the set increment has been reached, the robot stops.

If the robot motion is interrupted, e.g. by releasing the enabling switch, the interrupted increment is not resumed with the next motion; a new increment is started instead.

The following options are available:

Setting	Description
Continuous	Incremental jogging is deactivated.
100 mm / 10°	1 increment = 100 mm or 10°
10 mm / 3°	1 increment = 10 mm or 3°
1 mm / 1°	1 increment = 1 mm or 1°
0.1 mm / 0.005°	1 increment = 0.1 mm or 0.005°

Increments in mm:

- Valid for Cartesian jogging in the X, Y or Z direction.

Increments in degrees:

- Valid for Cartesian jogging in the A, B or C direction.
- Valid for axis-specific jogging.

Moving the robot in emergencies without the controller

Fig. 2-8: Release device

Description

- The release device can be used to move the robot mechanically after an accident or malfunction.
- The release device can be used for the main axis drive motors and, depending on the robot variant, also for the wrist axis drive motors.
- It is **only** for use in exceptional circumstances and emergencies (e.g. for freeing people).

- If the release device is used, it must be ensured that the brakes are working flawlessly.
  - A brake test must be performed for this. If the test fails, the motors must be exchanged.
  - If the brake test is not available on the controller or if it cannot be performed, the affected motors must be exchanged.

⚠ WARNING The motors reach temperatures during operation which can cause burns to the skin. Contact must be avoided. Appropriate safety precautions must be taken, e.g. protective gloves must be worn.

Procedure

1. Switch off the robot controller and secure it (e.g. with a padlock) to prevent unauthorized persons from switching it on again.
2. Remove the protective cap from the motor.
3. Push the release device onto the corresponding motor and move the axis in the desired direction.

   Labeling of the directions with arrows on the motors can be ordered as an option. It is necessary to overcome the resistance of the mechanical motor brake and any other loads acting on the axis.

Example for the motor of axis 2:

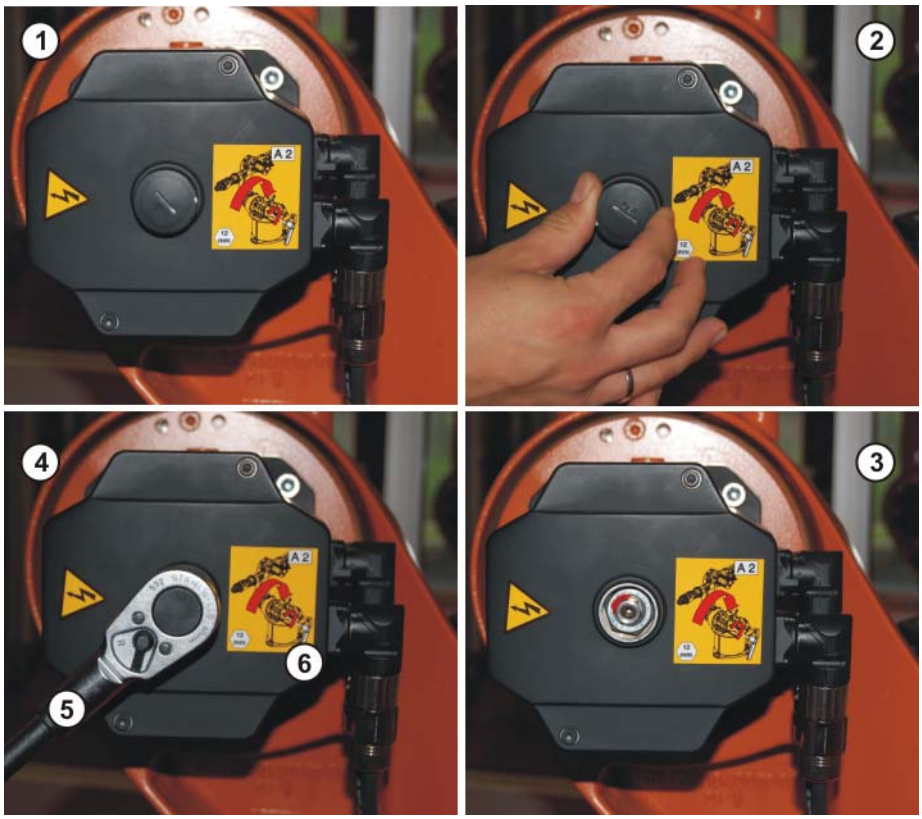


Fig. 2-9: Procedure for using release device

Item	Description
1	Motor A2 with protective cap fitted
2	Removing the protective cap from motor A2
3	Motor A2 with protective cap removed
4	Mounting the release device on motor A2

Item	Description
5	Release device
6	Sign (optional) indicating the direction of rotation

⚠ CAUTION Moving an axis with the release device can damage the motor brake. This can result in personal injury and material damage. After using the release device, the affected motor must be exchanged.

Further information is contained in the robot operating or assembly instructions.

2.5.1 Exercise: Operator control and axis-specific jogging

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Switch the robot controller on and off.
- Basic operator control of the robot using the smartPAD
- Axis-specific jogging of the robot by means of the jog keys and Space Mouse
- Interpret and reset first simple system messages

Preconditions

The following are preconditions for successful completion of this exercise:

- Completion of safety instruction

> **Note!**
> Safety instruction must be completed and documented before commencing this exercise!

- Theoretical knowledge of the general operator control of a KUKA industrial robot system
- Theoretical knowledge of axis-specific jogging

Task description

Carry out the following tasks:

1. Switch the control cabinet on and wait for the system to boot.
2. Release and acknowledge the Emergency Stop.
3. Ensure that T1 mode is set.
4. Activate axis-specific jogging.
5. Perform axis-specific jogging of the robot with various different jog override (HOV) settings using the jog keys and Space Mouse.
6. Explore the motion range of the individual axes, being careful to avoid any obstacles present, such as a table or cube magazine with fixed tool (accessibility investigation).
7. On reaching the software limit switches, observe the message window.

2.6 Coordinate systems in conjunction with robots

During the operator control, programming and start-up of industrial robots, the coordinate systems are of major significance. The following coordinate systems are defined in the robot controller:

- **ROBROOT**
  Robot base coordinate system
- **WORLD**
  World coordinate system
- **BASE**
  Base coordinate system
- **FLANGE**
  Flange coordinate system
- **TOOL**
  Tool coordinate system

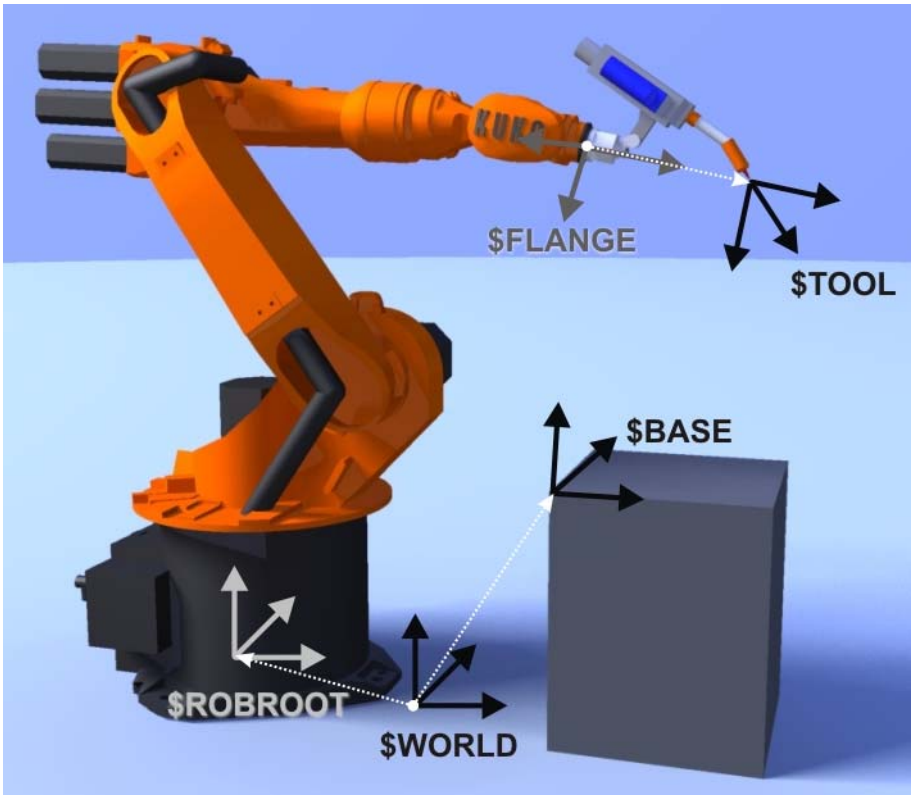


Fig. 2-10: Coordinate systems on the KUKA robot

- **ROBROOT**
  - Fixed in the robot base.
  - Origin of the robot.
  - Reference point for the WORLD coordinate system.
- **WORLD**
  - Identical to the ROBROOT coordinate system on delivery.
  - It can be “pushed out” of the robot base.
  - Defines the position of the WORLD coordinate system relative to the ROBROOT coordinate system.
  - Used, for example, with wall- and ceiling-mounted robot systems.
- **BASE**
  - Freely definable, customer-specific coordinate system.

- Defines the position of the base relative to WORLD.
- Used for calibrating workpieces and fixtures.

- **FLANGE**
  - The FLANGE coordinate system is fixed in the robot flange.
  - The origin is the center of the robot flange.
  - Reference point for the TOOL coordinate system.
- **TOOL**
  - Freely definable, customer-specific coordinate system.
  - The origin of the TOOL coordinate system is called the “**TCP**” (Tool Center Point).
  - Used for calibrating tools.

2.7 Moving the robot in the world coordinate system

Motion in the world coordinate system

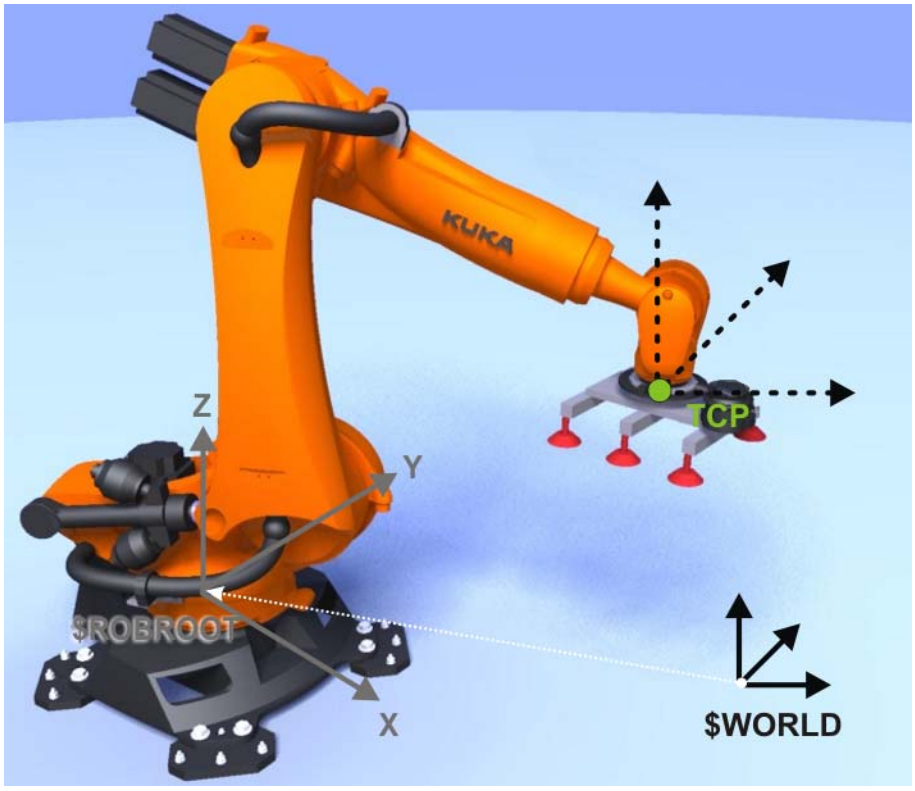


Fig. 2-11: Principle of jogging in the world coordinate system

- The robot tool can be moved with reference to the coordinate axes of the world coordinate system.

  In this case, **all** robot axes move.
- The jog keys or Space Mouse of the KUKA smartPAD are used for this.
- By default, the world coordinate system is located in the base of the robot (Robroot).
- The velocity can be modified (jog override: HOV).
- Jogging is only possible in T1 mode.
- The enabling switch must be pressed.

Space Mouse

- The Space Mouse allows intuitive motion of the robot and is the ideal choice for jogging in the world coordinate system.
- The mouse position and degrees of freedom can be modified.

Principle of jogging in the world coordinate system

A robot can be moved in a coordinate system in two different ways:

- Translational (in a straight line) along the orientation directions of the coordinate system: X, Y, Z
- Rotational (turning/pivoting) about the orientation directions of the coordinate system: angles A, B and C

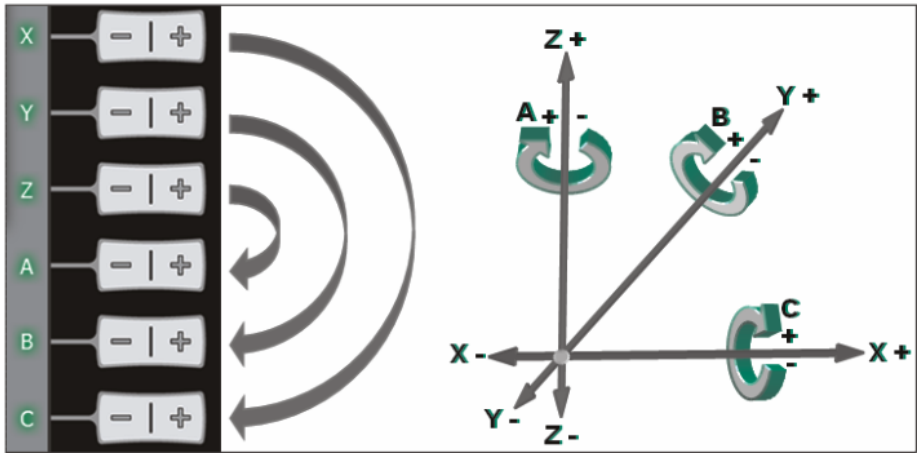


Fig. 2-12: Cartesian coordinate system

In the case of a motion command (e.g. jog key pressed), the controller first calculates a path. The starting point of the path is the tool center point (TCP). The direction of the path is specified by the world coordinate system. The controller then controls the axes to guide the tool along this path (translation) or about it (rotation).

Advantages of using the world coordinate system:

- The motion of the robot is always predictable.
- The motions of the TCP in space are always unambiguous, as the origin and coordinate axes are always known.
- The world coordinate system can always be used with a mastered robot.
- The Space Mouse allows intuitive operator control.

Using the Space Mouse

- All motion types are possible with the Space Mouse:
  - Translational: by pushing and pulling the Space Mouse

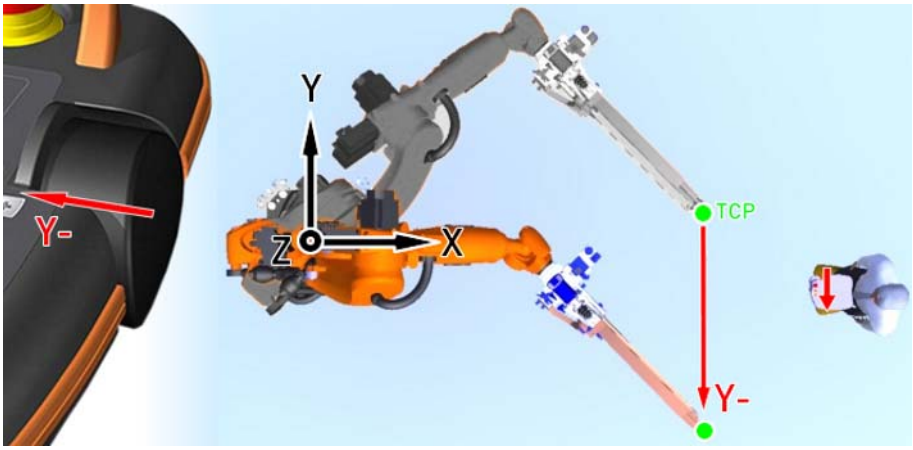


Fig. 2-13: Example: motion to the left

  - Rotational: by turning the Space Mouse

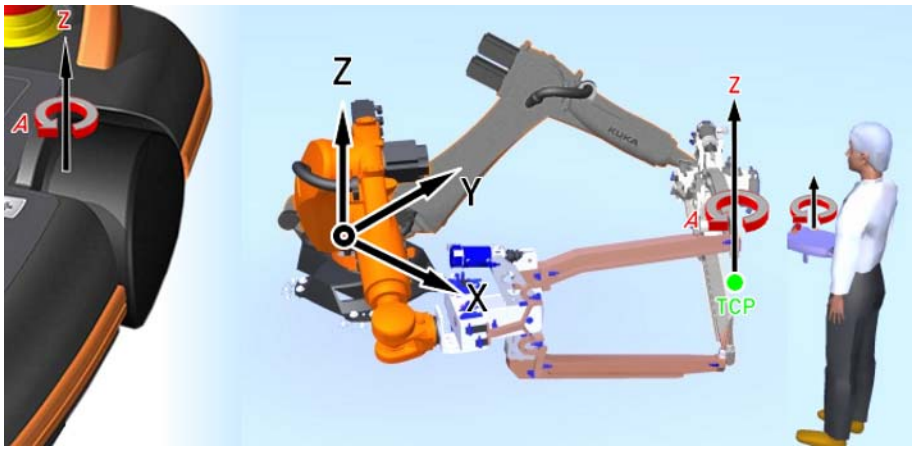


Fig. 2-14: Example: rotational motion about Z – angle A

- The Space Mouse position can be adapted to the position of the operator relative to the robot.

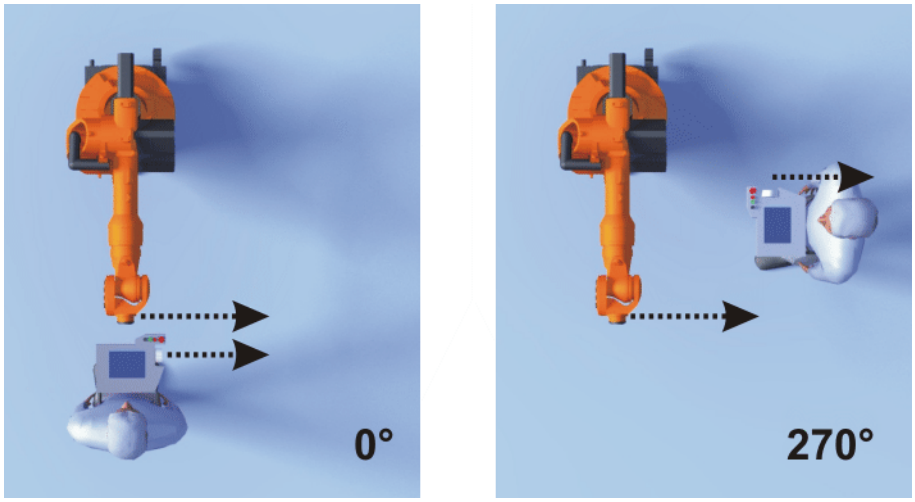
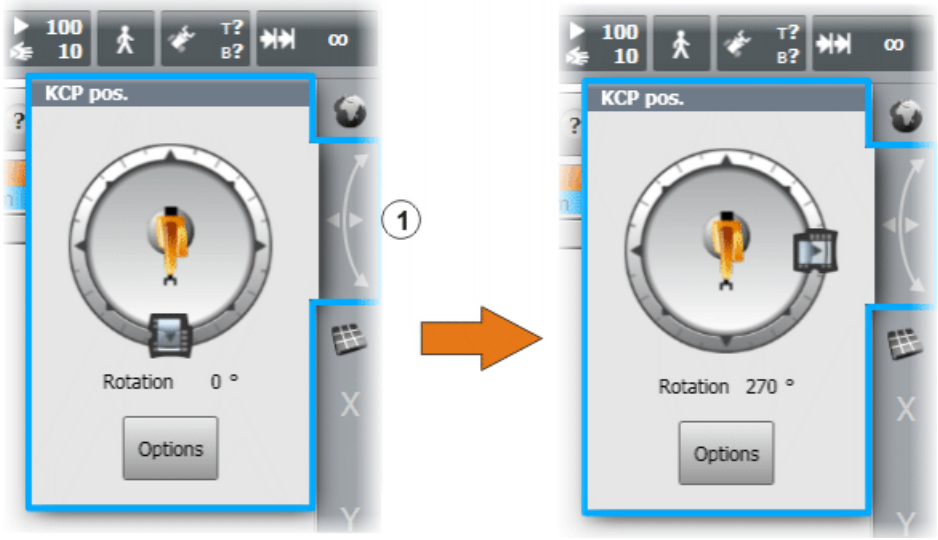


Fig. 2-15: Space Mouse: 0° and 270°

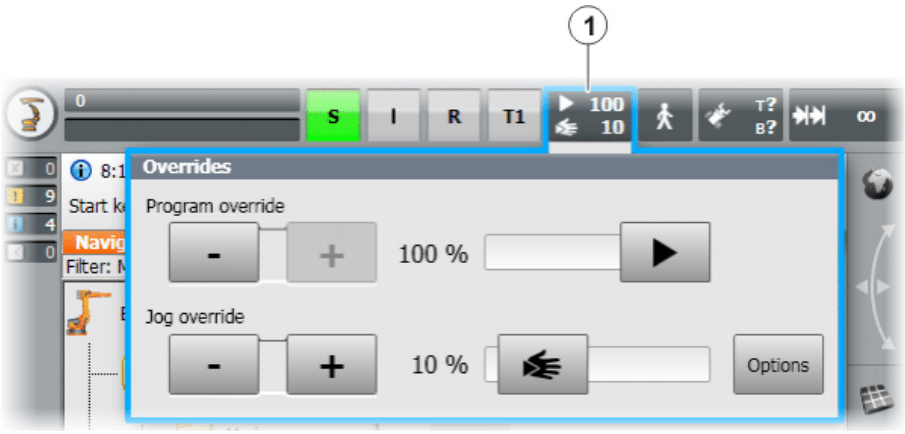
Executing a translational motion (world)

1. Set the KCP position by moving the slider control (1).



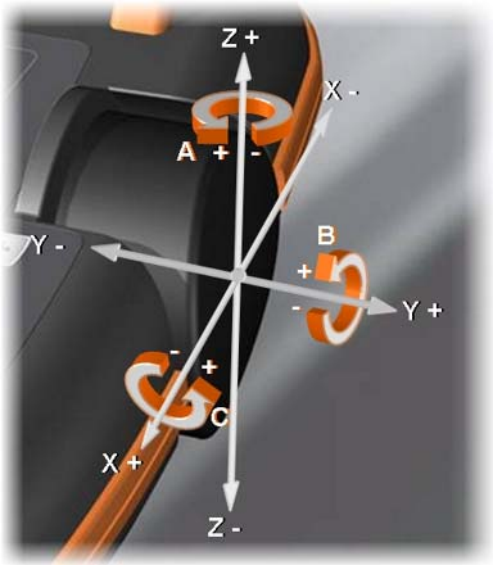
2. Select **World** as the option for the Space Mouse.

3. Set jog override.



4. Press the enabling switch into the center position and hold it down.

5. Jog in the corresponding direction using the Space Mouse.



6. Alternatively, the jog keys can be used.

2.7.1 Exercise: Operator control and jogging in the world coordinate system

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Switch the robot controller on and off.
- Basic operator control of the robot using the smartPAD
- Jog the robot, in the world coordinate system, by means of the jog keys and Space Mouse
- Interpret and reset first simple system messages

Preconditions

The following are preconditions for successful completion of this exercise:

- Completion of safety instruction

> **Note!**
> Safety instruction must be completed and documented before commencing this exercise!

- Theoretical knowledge of the general operator control of a KUKA industrial robot system
- Theoretical knowledge of jogging in the world coordinate system

Task description

1. Release and acknowledge the Emergency Stop.
2. Ensure that T1 mode is set.
3. Activate the world coordinate system.
4. Remove a cube from the cube magazine and position it on the table.
5. Attempt to align the robot gripper over the cube.
6. Close the gripper. The cube must not move when the gripper closes.
7. Move away from the table plate with the cube and set the cube down at any other point.
8. Switch between jog key and mouse operation.

What you should now know:

1. How can messages be acknowledged?

. . . . . .

. . . . . .

2. Which icon represents the world coordinate system?

a) b) c) d)




3. What is the name of the velocity setting for jog mode?

. . . . . .

4. What operating modes are there?

. . . . . .

. . . . . .

3 Starting up the robot

3.1 Overview

The following contents are explained in this training module:

- Start-up mode
- Robot mastering
- Tool calibration
- Base calibration

3.2 Start-up mode

Description

- If, in the case of initial commissioning, for example, no safety periphery is yet present (e.g. external E-STOP), the robot cannot be moved.
- This can be remedied by using *Start-up mode* which allows robot motion at the reduced velocity T1.
- This enables start-up tasks, such as mastering the robot.
- The industrial robot can be set to Start-up mode via a menu item of the smartHMI user interface.
- Start-up mode is terminated or prevented if there is an active connection to a safety system.

Use of Start-up mode disables all external safeguards.

Possible hazards and risks involved in using Start-up mode:

- A person walks into the manipulator's danger zone.
- An unauthorized person moves the manipulator.
- In a hazardous situation, a disabled external EMERGENCY STOP device is actuated and the manipulator is not shut down.

Additional measures for avoiding risks in Start-up mode:

- Cover disabled EMERGENCY STOP devices or attach a warning sign indicating that the EMERGENCY STOP device is out of operation.
- If there is no safety fence, other measures must be taken to prevent persons from entering the manipulator's danger zone, e.g. use of warning tape.
- Use of Start-up mode must be minimized – and avoided where possible – by means of organizational measures.
- The service personnel are responsible for ensuring that there is no-one in or near the danger zone of the manipulator as long as the safeguards are disabled.

Intended use of Start-up mode:

- Only service personnel who have received safety instruction may use Start-up mode.
- Fault localization (periphery fault).
- Start-up.

Misuse

- Any use or application deviating from the intended use is deemed to be impermissible misuse. This includes, for example, use by any other personnel.
- KUKA Roboter GmbH accepts no liability for damage or injury caused thereby. The risk lies entirely with the user.

Precondition

Precondition for PROFIsafe configuration:

- PROFIsafe is activated via the safety configuration.
- The user group “Expert” is required.
  Menu path: *KUKA key > Configuration > User group*
- The robot controller prevents or terminates Start-up mode if a connection to a higher-level safety PLC exists or is established.

Precondition for KSS configuration: X11 / SIB, or VSS: XS2 + XS5 / SIB

- The user group “Expert” is required.
  Menu path:*KUKA key > Configuration > User group*
- KR C4: X311 must be connected to the CCU as a jumper plug.
  KR C4 compact: Not required, jumper plug not present.
- **System Software 8.2 or earlier:**
  Start-up mode is always possible if all input signals at the safety interface KSS: X11, or VSS: XS2 + XS5
  have the state “logical zero”. If this is not the case, the robot controller prevents or terminates Start-up mode.
- **System Software 8.3:**
  Start-up mode is always possible. This also means that it is independent of the state of the inputs at safety interface X11.

Procedure

- Activation of Start-up mode:
  Menu path: *KUKA key > Start-up > Service > Start-up mode*

Menu	Description
✓ Start-up mode	Start-up mode is active. Touching the menu item deactivates the mode.
Start-up mode	Start-up mode is not active. Touching the menu item activates the mode.

- Active Start-up mode is signaled by a flashing yellow **IBN** indicator on the HMI.
  The message *Start-up mode active, EMERGENCY STOP has LOCAL effect ONLY* is displayed.

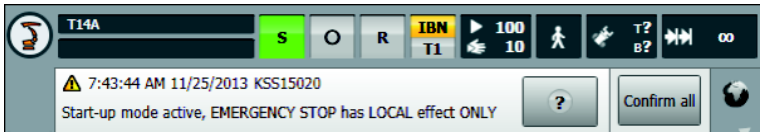


Fig. 3-1: Start-up mode

3.3 Mastering principle

Why is mastering carried out?

An industrial robot can only be used optimally if it is also completely and correctly mastered. Only then can it exploit its pose accuracy and path accuracy to the full, or be moved using programmed motions at all.

During mastering, a reference value is assigned to every axis. In this way, the robot controller knows where this axis is located.

A complete mastering operation includes the mastering of every single axis. With the aid of a technical tool (**EMD** – *Electronic Mastering Device*), a reference value (e.g. 0°) is assigned to every axis in its **mechanical zero position**. Since, in this way, the mechanical and electrical positions of the axis are

matched, every axis receives an unambiguous angle value. For mastering of the Agilus small robot, the **MEMD** (*microEMD*) is used.

The mastering position is similar, but not identical, for all robots. The exact positions may even vary between individual robots of a single robot type.

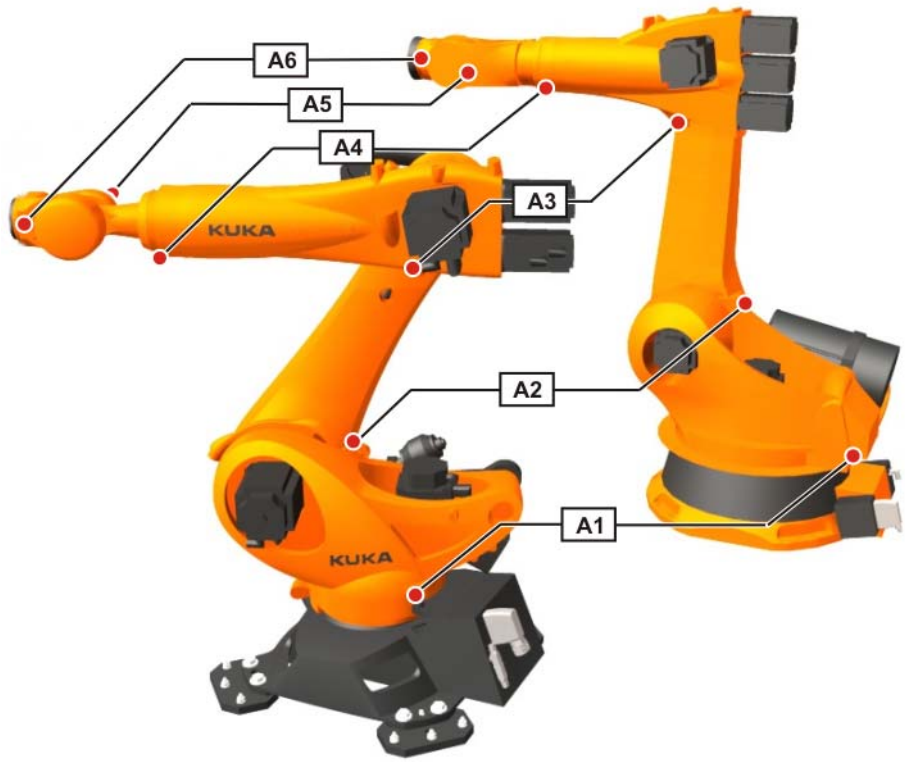


Fig. 3-2: Positions of the mastering cartridges

Angle values of the mechanical zero position (= reference values)

Axis	"Quantec" robot generation	Other robot types (e.g. Series 2000, KR 16, etc.)
A1	-20°	0°
A2	-120°	-90°
A3	+110°	+90°
A4	0°	0°
A5	0°	0°
A6	0°	0°

When is mastering carried out?

A robot must always be mastered. Mastering must be carried out in the following cases:

- During commissioning
- Following maintenance work to components that are involved in the acquisition of position values (e.g. motor with resolver or RDC)
- If robot axes are moved without the controller (e.g. by means of a release device)
- Following mechanical repairs/problems, the robot must first be unmastered before mastering can be carried out:
  - After exchanging a gear unit
  - After an impact with an end stop at more than 250 mm/s
  - After a collision

Before carrying out maintenance work, it is generally a good idea to check the current mastering.

Safety instructions for mastering

The functionality of the robot is severely restricted if robot axes are not mastered:

- Program mode is not possible: programmed points cannot be executed.
- No Cartesian jogging: motions in the coordinate systems are not possible.
- Software limit switches are deactivated.

NOTICE The software limit switches of an unmastered robot are deactivated. The robot can hit the end stop buffers, thus damaging the robot and making it necessary to exchange the buffers. An unmastered robot should not be jogged, if at all avoidable. If it must be jogged, the jog override must be reduced as far as possible.

Carrying out mastering

Fig. 3-3: EMD in operation

Mastering is carried out by determining the mechanical zero point of the axis. The axis is moved until the mechanical zero point is reached. This is the case when the gauge pin has reached the lowest point in the reference notch. Every axis is thus equipped with a mastering cartridge and a mastering mark.

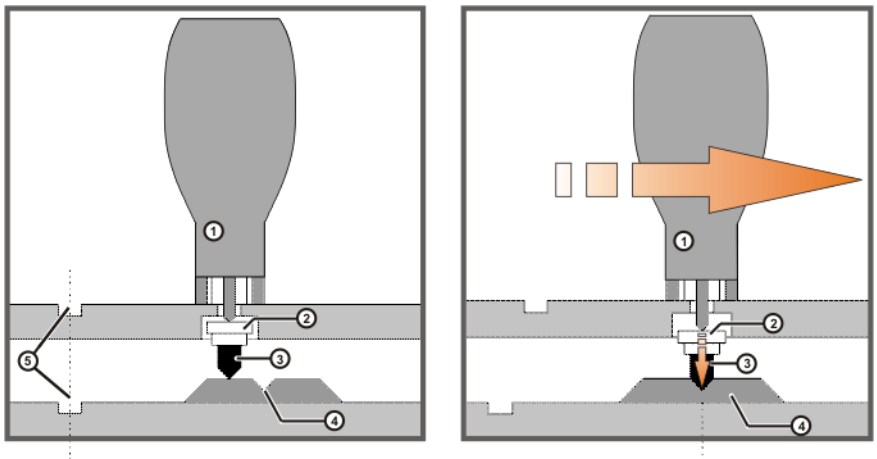


Fig. 3-4: EMD mastering sequence

1 EMD (Electronic Mastering Device)
2 Gauge cartridge
3 Gauge pin
4 Reference notch
5 Premastering mark

3.4 Mastering the robot

Robot mastering options

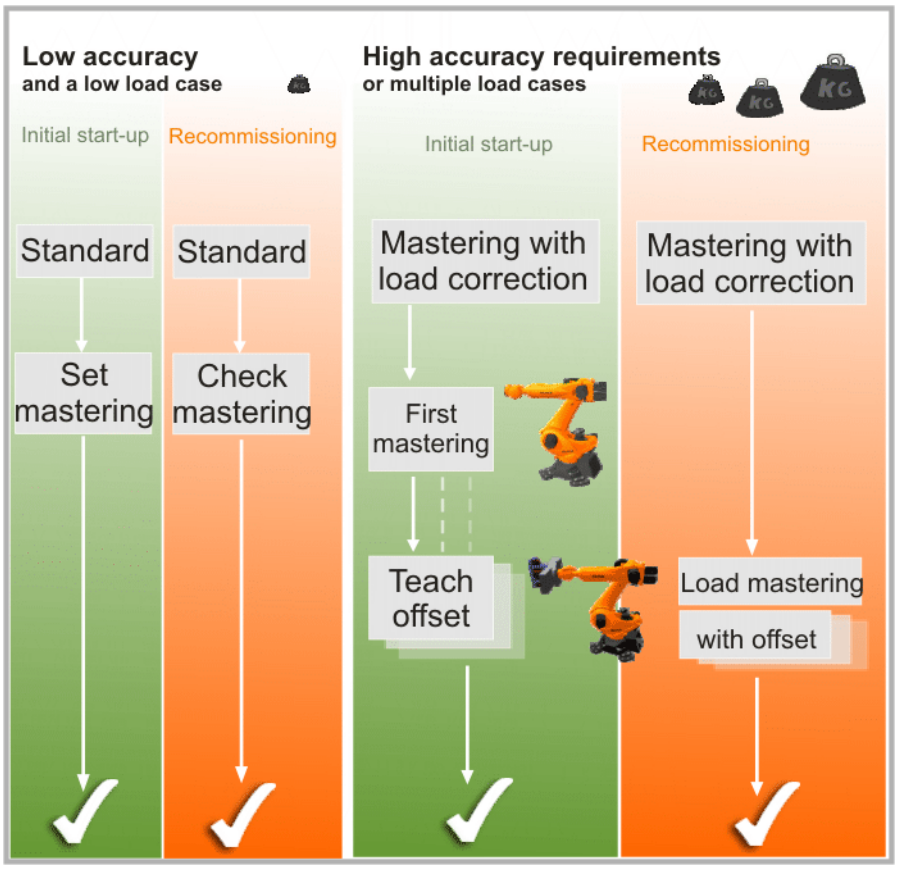


Fig. 3-5: Mastering options

- **Standard mastering**
  Column in diagram: *Low accuracy and a low load case*
  This mastering type is used if:
  - the robot in its application environment guides a permanently installed tool with a constant weight, e.g. an adhesive nozzle
  - or the application only requires low accuracy, e.g. the palletizing of packages
- **Mastering with load correction**
  Column in diagram: *High accuracy requirements or multiple load cases*
  This mastering type is used if:
  - and simultaneously high accuracy is required, e.g. laser welding
  - the robot in its application environment works with changing loads, e.g. gripper with and without load

Why teach the offset?

Due to the weight of the tool mounted on the flange, the robot is subjected to a static load. Material-related elasticity of the components and gear units can result in the robot positions being different for a loaded robot and an unloaded robot. These differences of just a few increments affect the accuracy of the robot.

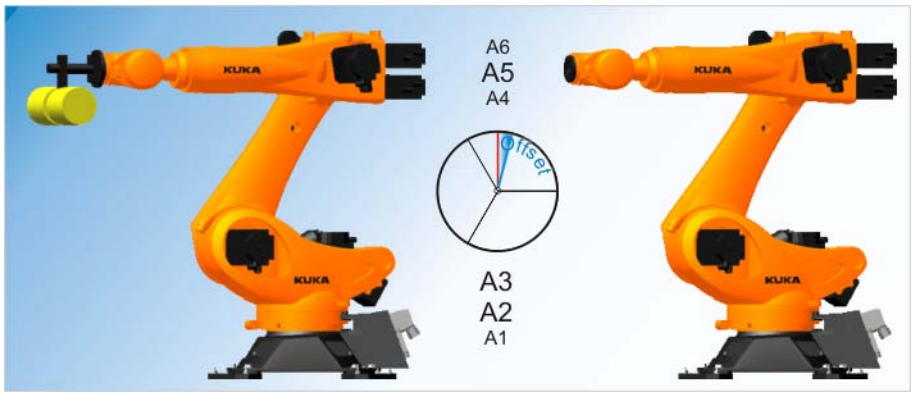


Fig. 3-6: Teach offset

"Teach offset" is carried out with a load. The difference from the first mastering (without a load) is saved.

If the robot is operated with different loads, the function "Teach offset" must be carried out for every load. In the case of grippers used for picking up heavy workpieces, "Teach offset" must be carried out for the gripper both with and without the workpiece.

Only a robot mastered with load correction has the required accuracy. For this reason, an offset must be taught for every load case! A precondition is that the geometric calibration of the tool has already been carried out and that the tool has thus been assigned a tool number.

Mastering log files

- When the robot is mastered, specific mastering data are logged in a log file.
- The calculated offsets are saved in degrees in the file *C:\KRC\ROBOTER\LOG\Mastery.log*.
- The following specific mastering data are saved in the file **Mastery.log**:
  - Time stamp (date, time)
  - Axis
  - Serial number of the robot
  - Mastering value (*FirstEncoderValue*)
  - Tool number
  - Offset value (*Encoder Difference*) in degrees on the motor
- **Sample Mastery.log:**

```
Date: 01.09.11  Time: 13:41:07  Axis 1  Serialno.: 864585  First
Mastering  (FirstEncoderValue: 1.138909)
Date: 01.09.11 Time: 13:42:07 Axis 2 Serialno.: 864585
First Mastering (FirstEncoderValue: 0.644334)
Date: 01.09.11 Time: 13:42:56 Axis 3 Serialno.: 864585
First Mastering (FirstEncoderValue: 0.745757)
Date: 01.09.11 Time: 13:43:29 Axis 4 Serialno.: 864585
First Mastering (FirstEncoderValue: 1.450234)
Date: 01.09.11 Time: 13:44:03 Axis 5 Serialno.: 864585
First Mastering (FirstEncoderValue: 0.686983)
Date: 01.09.11 Time: 13:44:30 Axis 6 Serialno.: 864585
First Mastering (FirstEncoderValue: 0.901439)
...
Date: 01.09.11  Time: 14:07:10
Axis 1 Serialno.: 864585
Tool Teaching for Tool No 1 (Encoder Difference: -0.001209)
Date: 01.09.11  Time: 14:08:44
Axis 2 Serialno.: 864585
```

```
Tool Teaching for Tool No 1 (Encoder Difference: 0.005954)
...
```

Procedure for first mastering

NOTICE First mastering may only be carried out if the robot is without a load. There must be no tool or supplementary load mounted.

1. Move robot to the pre-mastering position.

Fig. 3-7: Examples of the pre-mastering position

2. Select **Start-up** > **Master** > **EMD** > **With load correction** > **First mastering** in the main menu.

   A window opens. All axes to be mastered are displayed. The axis with the lowest number is highlighted.
3. Remove the protective cap of the gauge cartridge on the axis highlighted in the window. Turned around, the EMD can be used as a screwdriver. Screw the EMD onto the gauge cartridge.

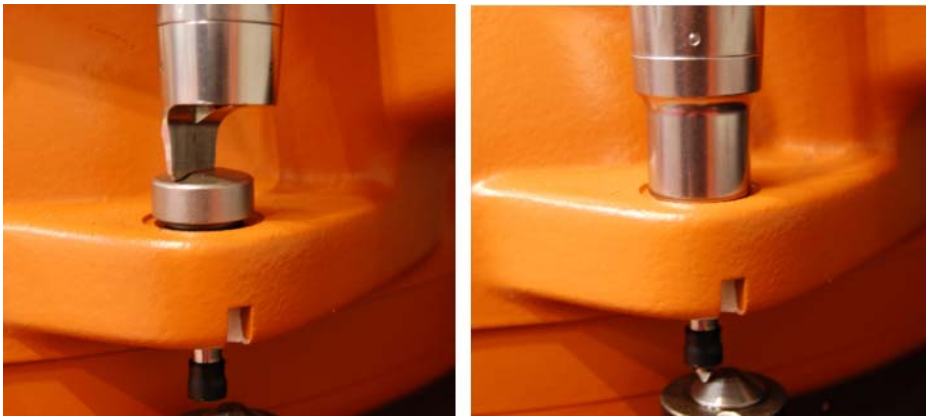


Fig. 3-8: EMD screwed onto gauge cartridge

4. Then attach the signal cable to the EMD and plug into connector X32 on the robot junction box.

Fig. 3-9: EMD cable, connected

⚠ CAUTION The EMD must always be screwed onto the gauge cartridge without the signal cable attached. Only then may the signal cable be attached to the EMD. Otherwise, the signal cable could be damaged.
Similarly, when removing the EMD, the signal cable must always be removed from the EMD first. Only then may the EMD be removed from the gauge cartridge.
After mastering, remove the signal cable from connection X32. Failure to do so may result in interference signals or damage.

5. Press **Master**.
6. Press the enabling switch into the center position and hold it down, and press and hold down the Start key.

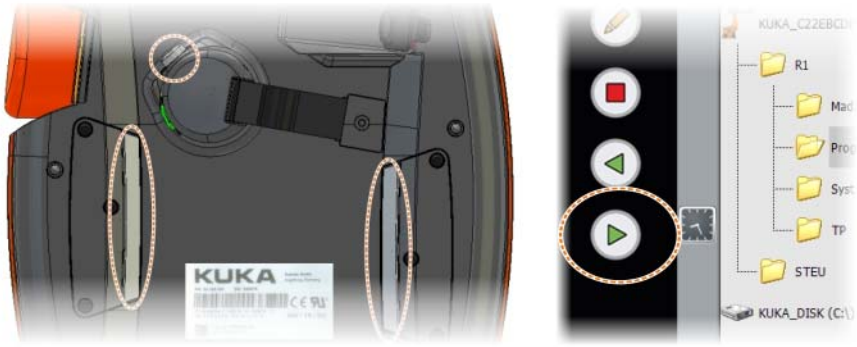


Fig. 3-10: Enabling switch and start key

When the EMD has passed through the lowest point of the reference notch, the mastering position is reached. The robot stops automatically. The values are saved. The axis is no longer displayed in the window.

7. Remove the signal cable from the EMD. Then remove the EMD from the gauge cartridge and replace the protective cap.
8. Repeat steps 2 to 5 for all axes to be mastered.
9. Close the window.
10. Remove signal cable from connection X32.

Procedure for teaching an offset

"Teach offset" is carried out with a load. The difference from the first mastering is saved.

1. Move robot to the pre-mastering position.
2. Select **Start-up** > **Master** > **EMD** > **With load correction** > **Teach offset** in the main menu.
3. Enter tool number. Confirm with **Tool OK**.
   A window opens. All axes for which the tool has not yet been taught are displayed. The axis with the lowest number is highlighted.
4. Remove the protective cap of the gauge cartridge on the axis highlighted in the window. Screw the EMD onto the gauge cartridge. Then attach signal cable to EMD and plug into connector X32 on the base frame junction box.
5. Press **Learn**.
6. Press an enabling switch and the Start key.
   When the EMD detects the lowest point of the reference notch, the mastering position is reached. The robot stops automatically. A window opens. The deviation of this axis from the first mastering is indicated in degrees and increments.
7. Confirm with **OK**. The axis is no longer displayed in the window.

8. Remove the signal cable from the EMD. Then remove the EMD from the gauge cartridge and replace the protective cap.
9. Repeat steps 3 to 7 for all axes to be mastered.
10. Remove signal cable from connection X32.
11. Exit the window by means of **Close**.

Procedure for set/check load mastering with offset

Load mastering with offset is carried out with a load. The first mastering is calculated.

1. Move robot to the pre-mastering position.
2. In the main menu, select **Start-up > Master > EMD > With load correction > Load mastering > With offset**.
3. Enter tool number. Confirm with **Tool OK**.
4. Remove the cover from connection X32 and connect the signal cable.
5. Remove the protective cap of the gauge cartridge on the axis highlighted in the window. (Turned around, the EMD can be used as a screwdriver.)
6. Screw the EMD onto the gauge cartridge.
7. Attach the signal cable to the EMD, aligning the red dot on the connector with the groove in the EMD.
8. Press **Check**.
9. Hold down an enabling switch and press the Start key.
10. If required, press **Save** to save the values. The old mastering values are deleted. To restore a lost first mastering, always save the values.
11. Remove the signal cable from the EMD. Then remove the EMD from the gauge cartridge and replace the protective cap.
12. Repeat steps 4 to 10 for all axes to be mastered.
13. Close the window.
14. Remove signal cable from connection X32.

The mastering type to be used (standard or with load correction) must correspond, in the case of existing systems, to that used during start-up. If an incorrect mastering type is used, this can result in an incorrectly mastered robot.

SEMD/MEMD mastering kit

Description

- The SEMD/MEMD mastering kit replaces the following mastering kits:
  - **EMT** – *Electronic Measuring Tool* for KR C1 and KR C2 technology
    The **option for KR C2 technology** (adapter) is required for this.
  - **EMD** – *Electronic Mastering Device* for KR C4 technology

  The mastering kit is also available separately as the SEMD or MEMD mastering kit.
- The SEMD/EMD mastering kit can be used to master all robot variants with a **large** or **small** mastering cartridge.
  - **MEMD** – *Micro Electronic Mastering Device*
    for small mastering cartridges
  - **SEMD** – *Standard Electronic Mastering Device*
    for large mastering cartridges

Fig. 3-11: SEMD/MEMD mastering kit

Item	Description	Item	Description
1	Universal mastering box	4	SEMD sensor for large mastering cartridges
2	Screwdriver	5	Cables
3	MEMD sensor for small mastering cartridges		

Available mastering kits

Mastering kit	Article number	Description
SEMD/MEMD	00-228-936	Mastering device for all gauge cartridges (M8/M20 fine thread). Suitable for all robot types
SEMD	00-228-934	Mastering device for gauge cartridges with M20 fine thread. Suitable for Quantec series, for example.
MEMD	00-208-642	Mastering device for gauge cartridges with M8 fine thread. Suitable for AGILUS, for example.
KR C2 option	00-228-327	Adapter cable for using the SEMD as KTL (mastering kit for KR C2 systems).

Mastering the AGILUS small robot

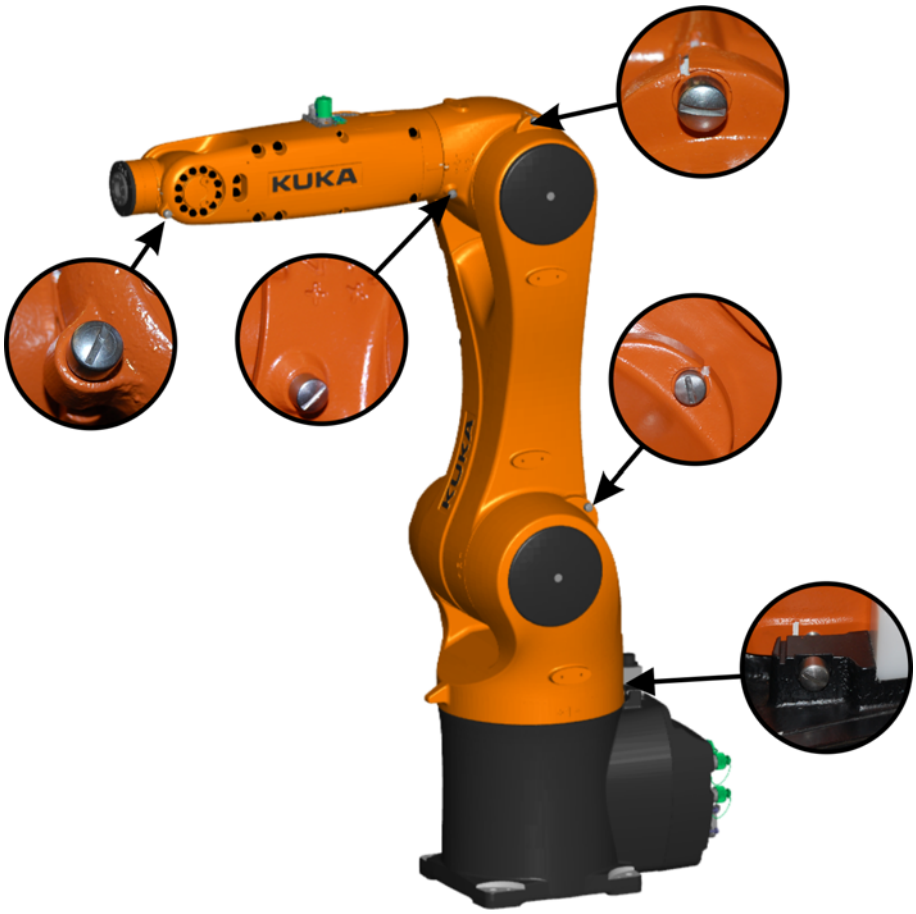


Fig. 3-12: Mastering position for KR AGILUS

Angle values of the mechanical zero position (= reference values)

Axis	KR AGILUS
A1	0°
A2	-90°
A3	+90°
A4	0°
A5	0°
A6	0°

- Axes 1 - 5 are mastered using the MEMD sensor for small mastering cartridges.
- A6 is mastered without the MEMD.
- Before mastering, A6 must be moved to its mastering position. (This means before the overall mastering process, not directly before mastering A6 itself). For this purpose, A6 has fine marks in the metal.
- To move A6 to the mastering position, these marks must be aligned exactly.

When moving to the mastering position, it is important to look at the fixed mark in a straight line from in front. If the mark is observed from the side, the movable mark cannot be aligned accurately enough. This results in incorrect mastering.

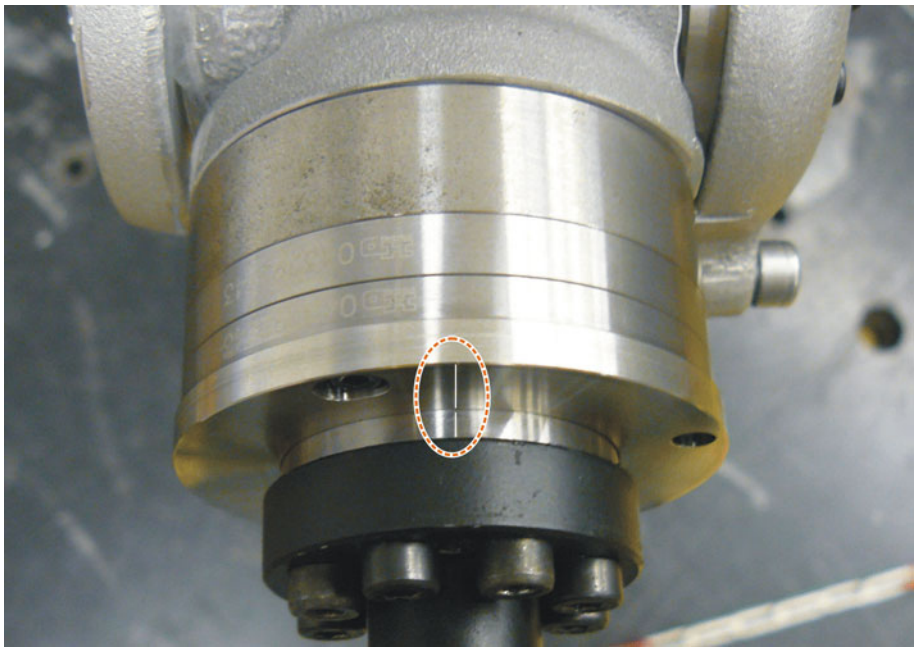


Fig. 3-13: Mastering position A6 – view from above

- In the main menu, select **Start-up** > **Master** > **Reference**.
  The option window **Reference mastering** is opened. A6 is displayed and is selected.
- Press **Master**. A6 is mastered and removed from the option window.
- Close the window.
- Disconnect the EtherCAT cable from X32 and the MEMD box.

3.4.1 Exercise: Robot mastering

Aim of the exercise

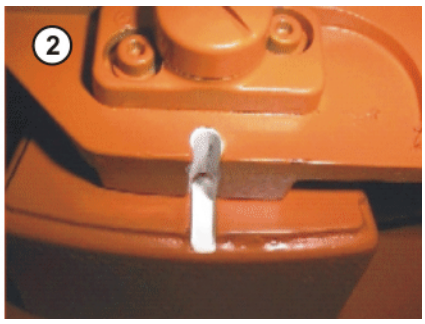
On successful completion of this exercise, you will be able to carry out the following activities:

- Move to pre-mastering position
- Select the correct mastering type
- Work with the “Electronic Mastering Device” (EMD)
- Master all axes using the EMD

Preconditions

The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of the general procedure for mastering
- Theoretical knowledge of the location of the pre-mastering position



1 Axis not in pre-mastering position

2 Axis in pre-mastering position

- Correct connection of the EMD to the robot
- Mastering via the Start-up menu

Task description

Carry out the following tasks:

1. Unmaster all robot axes.
2. Move all robot axes to the pre-mastering position in joint mode.
3. Perform load mastering with offset for all axes using the EMD/MEMD.
4. Use tool 14 for this.
5. Display the actual position in “Joint” mode.

What you should now know:

1. Why is mastering carried out?

. . . . . .

. . . . . .

2. Specify the angles of all 6 axes in the mechanical zero position.

A1: ..... A2: .....

A3: ..... A4: .....

A5: ..... A6: .....

3. What must be taken into consideration with an unmastered robot?

. . . . . .

. . . . . .

. . . . . .

4. Which mastering tool should be used for preference?

. . . . . .

. . . . . .

5. What is the danger of moving the robot with the EMD (dial gauge) screwed in place?

. . . . . .

. . . . . .

3.5 Loads on the robot

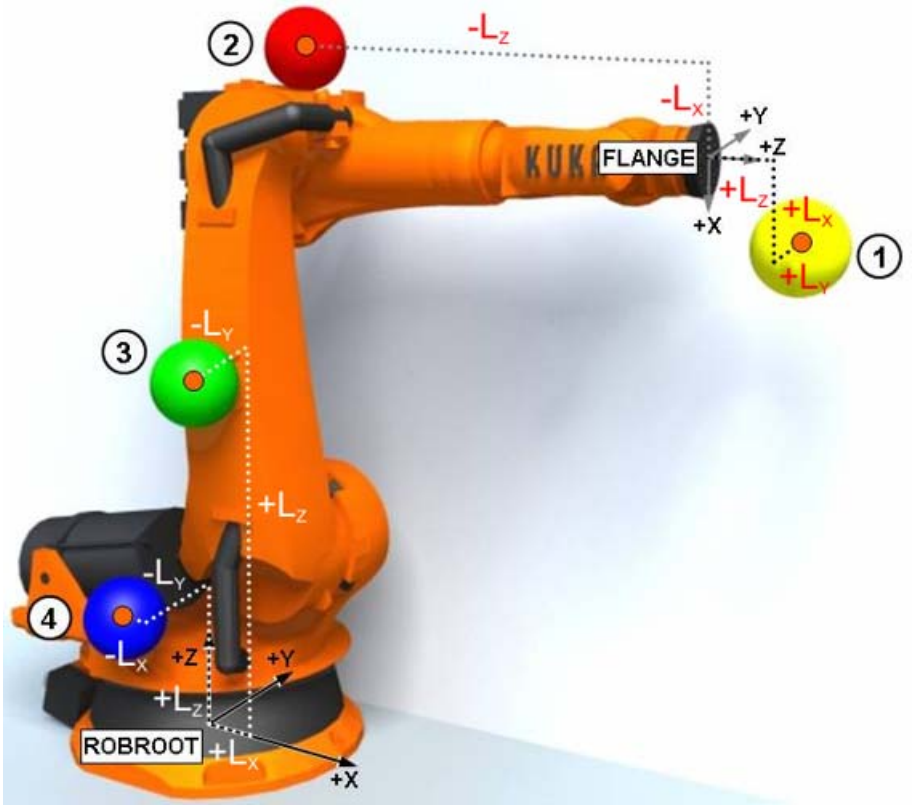


Fig. 3-14: Loads on the robot

1	Payload	3	Supplementary load on axis 2
2	Supplementary load on axis 3	4	Supplementary load on axis 1

3.6 Tool load data

What are tool load data?

Tool load data refer to all the loads mounted on the robot flange. They form an additional mass mounted on the robot which must also be moved together with the robot.

The values to enter are the mass, the position of the center of gravity (point on which the mass acts) and the mass moments of inertia with the corresponding principal inertia axes.

The payload data **must** be entered in the robot controller and assigned to the correct tool.

Exception: If the payload data have already been transferred to the robot controller by KUKA.LoadDataDetermination, no manual entry is required.

Tool load data can be obtained from the following sources:

- Software option KUKA.LoadDetermination (only for payloads)
- Manufacturer information
- Manual calculation
- CAD programs

Effects of the load data

The entered load data affect a wide range of controller processes. These include, e.g.:

- Control algorithms (calculation of acceleration)
- Velocity and acceleration monitoring

- Torque monitoring
- Collision detection
- Energy monitoring
- and many more.

It is thus very important that the load data are entered correctly. If the robot executes its motions with correctly entered load data...

- one can profit from its great accuracy.
- motion sequences with optimal cycle times are possible.
- the robot has a long service life (due to reduced wear).

Procedure

1. In the main menu, select **Start-up** > **Calibrate** > **Tool** > **Tool load data**.
2. Enter the number of the tool in the box **Tool no.**. Confirm with **Continue**.
3. Enter the payload data:
   - Box **M**: Mass
   - Boxes **X**, **Y**, **Z**: Position of the center of gravity relative to the flange
   - Boxes **A**, **B**, **C**: Orientation of the principal inertia axes relative to the flange
   - Boxes **JX**, **JY**, **JZ**: Mass moments of inertia

     (JX is the inertia about the X axis of the coordinate system that is rotated relative to the flange by A, B and C. JY and JZ are the analogous inertia about the Y and Z axes.)
4. Confirm with **Continue**.
5. Press **Save**.

3.7 Monitoring tool load data

Description

For many robot types, the robot controller monitors whether or not there is an overload or underload during operation. This monitoring is called “Online load data check” (OLDC).

If the OLDC detects an underload, for example, the robot controller reacts, e.g. by displaying a message. The reactions can be configured.

The results of the check can be polled using the system variable $LDC_RESULT.

OLDC is available for those robot types for which KUKA.LoadDataDetermination can also be used. Whether or not OLDC is available for the current robot type can be checked by means of $LDC_LOADED (TRUE = yes).

Overload	There is an overload if the actual load is greater than the configured load.
Underload	There is an underload if the actual load is less than the configured load.

Activation and configuration

OLDC can be configured as follows:

- During manual entry of the tool data
- During separate entry of the payload data

The following boxes are displayed in the same window in which the payload data are also entered:

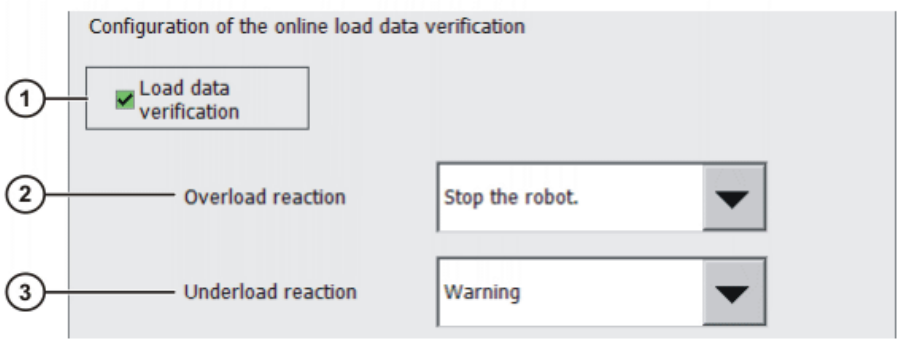


Fig. 3-15: Online load data check

Item	Description
1	**TRUE**: OLDC is activated for the tool displayed in the same window. The defined reactions are carried out in the case of an overload or underload. **FALSE**: OLDC is deactivated for the tool displayed in the same window. There is no reaction in the case of an overload or underload.
2	The overload reaction can be defined here. ■ **None**: No reaction. ■ **Warning**: The robot controller generates the following status message: *Check of robot load (Tool {No.}) calculated overload.* ■ **Stop robot**: The robot controller generates an acknowledgement message with the same content as that generated under **Warning**. The robot stops with a STOP 2.
3	The underload reaction can be defined here. The possible reactions are analogous to those for an overload.

NULLFRAME

OLDC cannot be configured for motions to which the tool **NULLFRAME** has been assigned. The reactions are defined for this case and cannot be influenced by the user.

- Overload reaction: **Stop robot**

  The following acknowledgement message is generated: *Overload calculated when checking robot load (no tool defined) and the set load data.* The robot stops with a STOP 2.
- Underload reaction: **Warning**

  The following status message is generated: *Underload calculated when checking robot load (no tool defined) and the set load data.*

3.8 Supplementary loads on the robot

Supplementary loads on the robot

Supplementary loads are additional components mounted on the base frame, link arm or arm, e.g.:

- Energy supply system
- Valves
- Materials feeder
- Materials supply

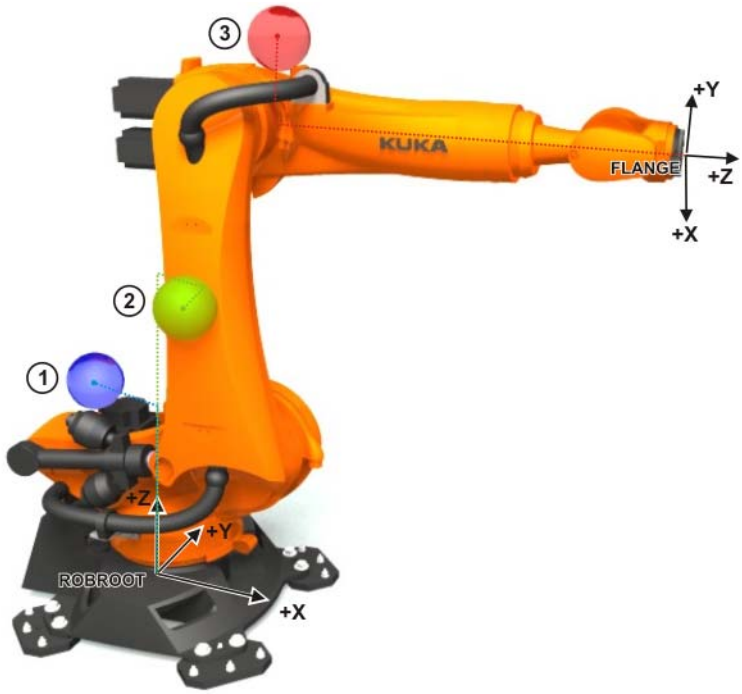


Fig. 3-16: Supplementary loads on the robot

The supplementary load data must be entered in the robot controller. Required specifications include:

- Mass (m) in kg
- Distance from center of mass to the reference system (X, Y and Z) in mm
- Orientation of the principal inertia axes relative to the reference system (A, B and C) in degrees (°)
- Mass moments of inertia about the inertia axes (Jx, Jy and Jz) in kgm²

Reference systems of the X, Y and Z values for each supplementary load:

Load	Reference system
Supplementary load A1	ROBROOT coordinate system A1 = 0°
Supplementary load A2	ROBROOT coordinate system A2 = -90°
Supplementary load A3	FLANGE coordinate system A4 = 0°, A5 = 0°, A6 = 0°

Load data can be obtained from the following sources:

- Software option KUKA.LoadDetect (only for payloads)
- Manufacturer information
- Manual calculation
- CAD programs

Influence of the supplementary loads on the robot motion

Specification of the load data influences the robot motion in various ways:

- Path planning
- Accelerations
- Cycle time
- Wear

⚠ WARNING If a robot is operated with incorrect load data or an unsuitable load, this can result in danger to life and limb and/or substantial material damage.

Procedure

1. In the main menu, select **Start-up** > **Calibrate** > **Supplementary load data**.
2. Enter the number of the axis on which the supplementary load is to be mounted. Confirm with **Continue**.
3. Enter the load data. Confirm with **Continue**.
4. Press **Save**.

3.9 Moving the robot in the tool coordinate system

Jogging in the tool coordinate system

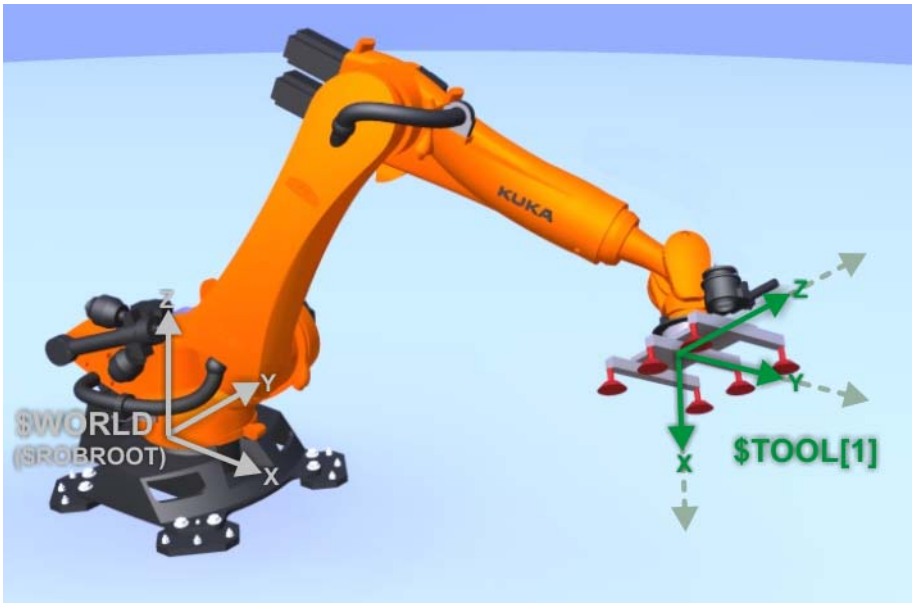


Fig. 3-17: Robot tool coordinate system

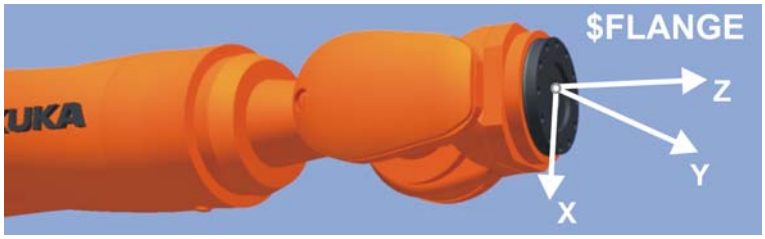
- In the case of jogging in the tool coordinate system, the robot can be moved relative to the coordinate axes of a previously calibrated tool.

  The coordinate system is thus not fixed (cf. world/base coordinate system), but guided by the robot.

  In this case, **all** required robot axes move. Which axes these are is determined by the system and depends on the motion.

  The origin of the tool coordinate system is called the **TCP** (*Tool Center Point*) and corresponds to the working point of the tool, e.g. the tip of an adhesive nozzle.
- The jog keys or Space Mouse of the KUKA smartPAD are used for this.
- There are 16 tool coordinate systems to choose from.
- The velocity can be modified (jog override: HOV).
- Jogging is only possible in T1 mode.
- The enabling switch must be pressed.

In the case of jogging, uncalibrated tool coordinate systems always correspond to the flange coordinate system.



Principle of jogging – tool

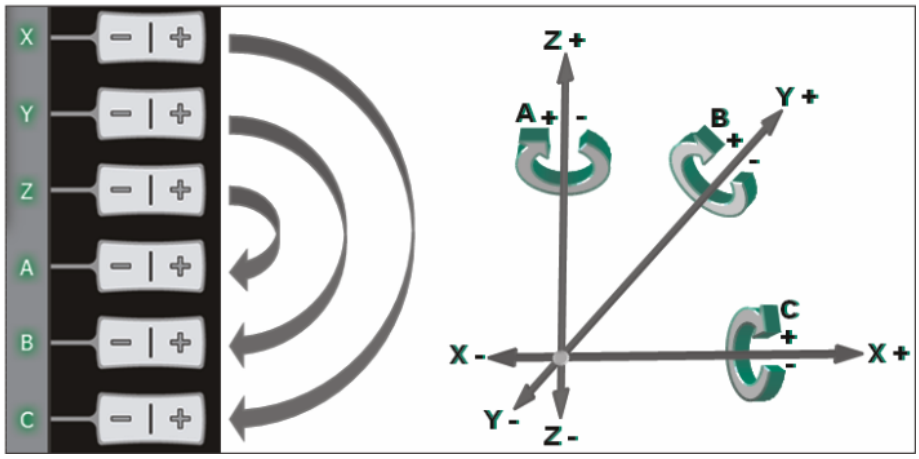


Fig. 3-18: Cartesian coordinate system

A robot can be moved in a coordinate system in two different ways:

- Translational (in a straight line) along the orientation directions of the coordinate system: X, Y, Z
- Rotational (turning/pivoting) about the orientation directions of the coordinate system: angles A, B and C

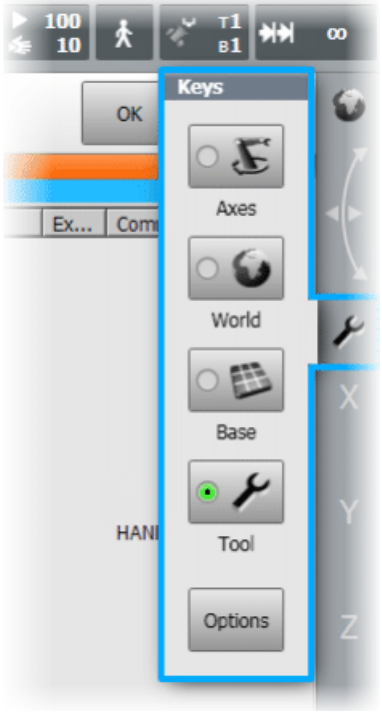
Advantages of using the tool coordinate system:

- The motion of the robot is always predictable as soon as the tool coordinate system is known.
- It is possible to move in the tool direction or to orient about the TCP.

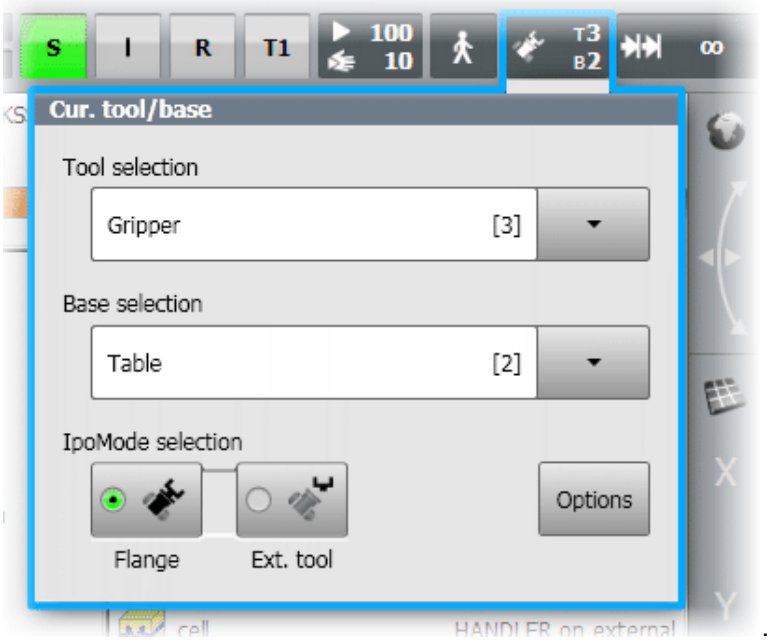
  The *tool direction* is the working or process direction of the tool, e.g. the direction in which adhesive is dispensed from an adhesive nozzle, the direction of gripping when gripping a workpiece, etc.

Procedure

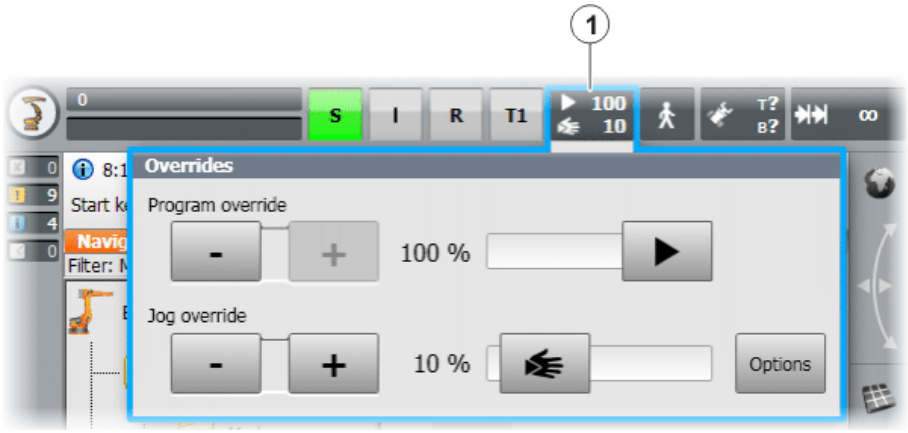
1. Select **Tool** as the coordinate system to be used.



2. Select the tool number.



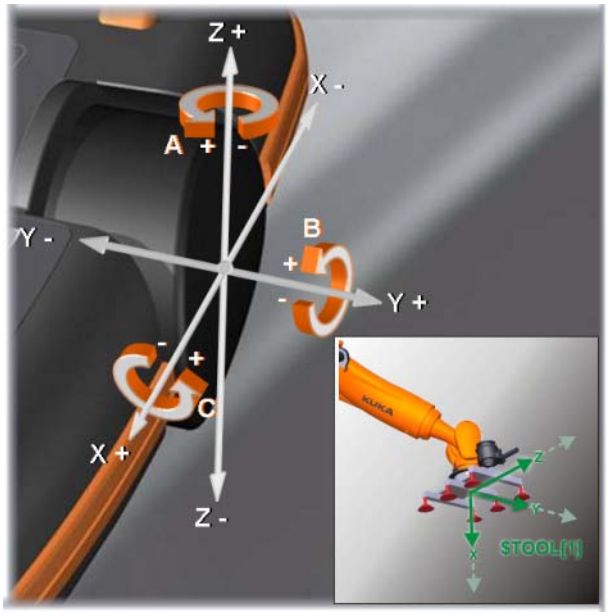
3. Set jog override.



4. Press the enabling switch into the center position and hold it down.

5. Move the robot using the jog keys.

6. Alternatively: Jog in the corresponding direction using the Space Mouse.



3.9.1 Exercise: Jogging in the tool coordinate system

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Jog the robot, in the tool coordinate system, by means of the jog keys and Space Mouse
- Jog the robot in the working direction of the tool

Preconditions

The following are preconditions for successful completion of this exercise:

- Completion of safety instruction

> **Note!**
> Safety instruction must be completed and documented before commencing this exercise!

- Theoretical knowledge of jogging in the tool coordinate system

Task description

Carry out the following tasks:

1. Release and acknowledge the EMERGENCY STOP.
2. Ensure that T1 mode is set.
3. Activate the tool coordinate system.
4. Jog the robot in the tool coordinate system with various different jog override (HOV) settings using the jog keys and Space Mouse. Test motion in the working direction of the tool and re-orientation about the TCP.
5. Fetch the pen from the holder using the tool “Gripper”.

3.10 Tool calibration

Description

Calibration of the tool (**TOOL** coordinate system) informs the robot controller where the tip of the tool (**TCP** - *Tool Center Point*) is located relative to the flange center point and how it is oriented.

Tool calibration thus consists of calibration...

- of the TCP (origin of the coordinate system).
- of the alignment of the coordinate system.

Fig. 3-19: Examples of calibrated tools

A maximum of 16 TOOL coordinate systems can be saved. (Variable: TOOL_DATA[1…16]).

During calibration, the distance between the tool coordinate system (in X, Y and Z) and the flange coordinate system, and the rotation of this coordinate system (angles A, B and C) are saved.

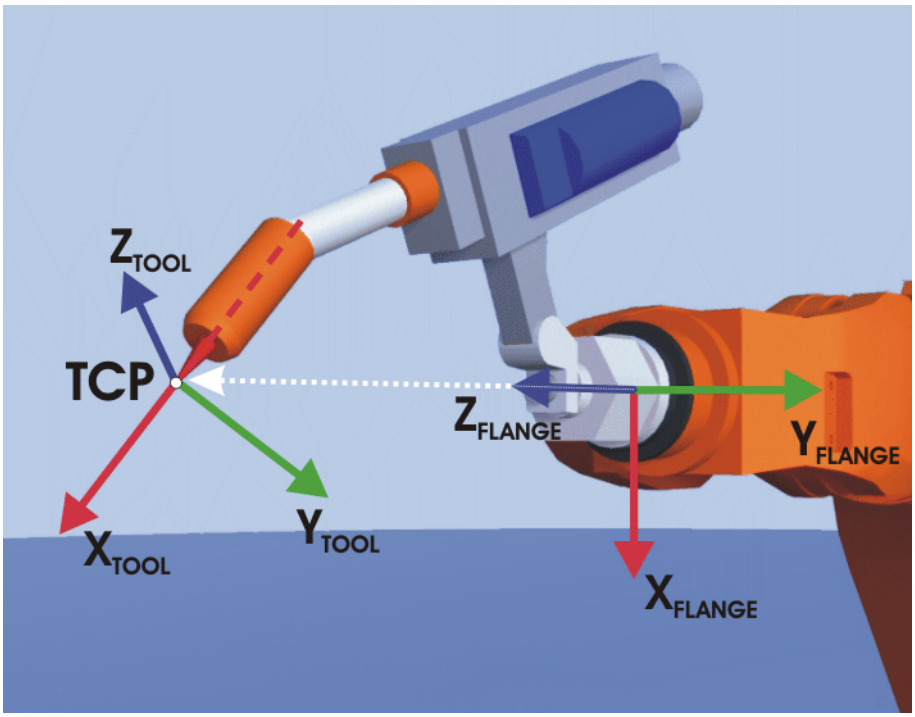


Fig. 3-20: TCP calibration principle

Advantages

If a tool has been calibrated precisely, this has the following practical advantages for the operating and programming personnel:

- Improved jogging
  - Reorientation about the TCP (e.g. tool tip) is possible.

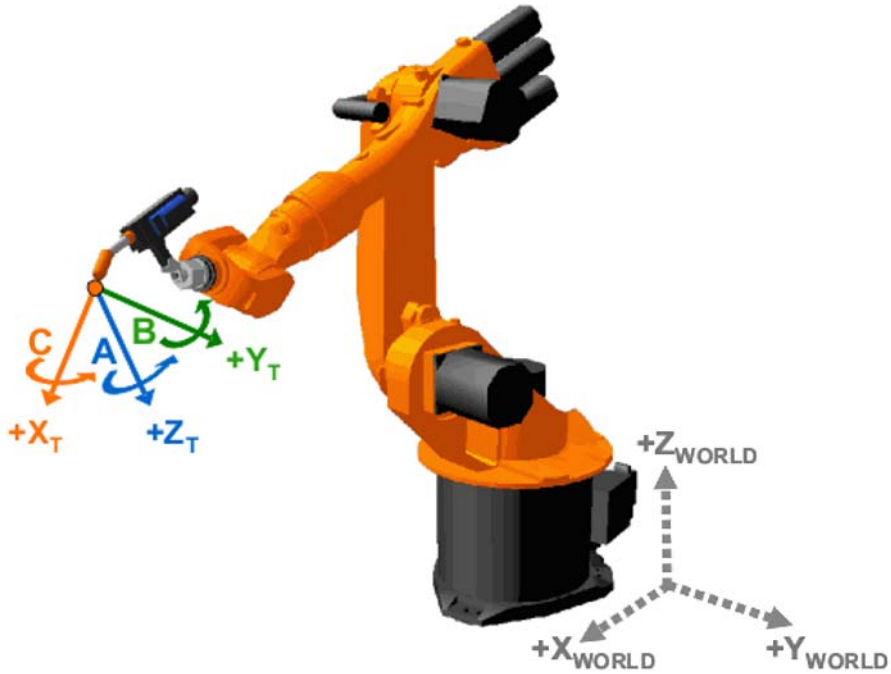


Fig. 3-21: Reorientation about the TCP

  - Moving the robot in the tool direction

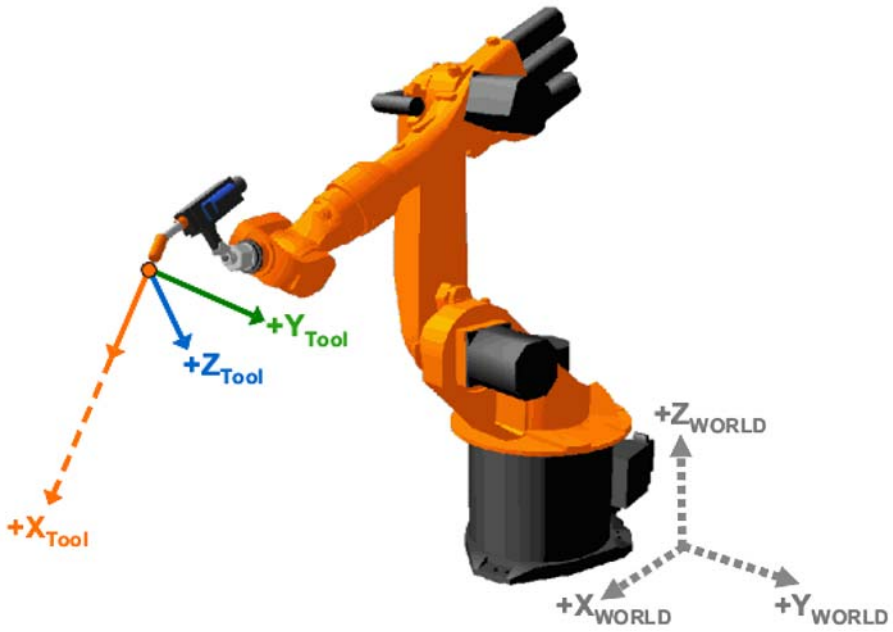


Fig. 3-22: Tool working direction

- Advantages during CP motion programming (linear or circular motion)
  - The programmed velocity is maintained at the TCP along the path.

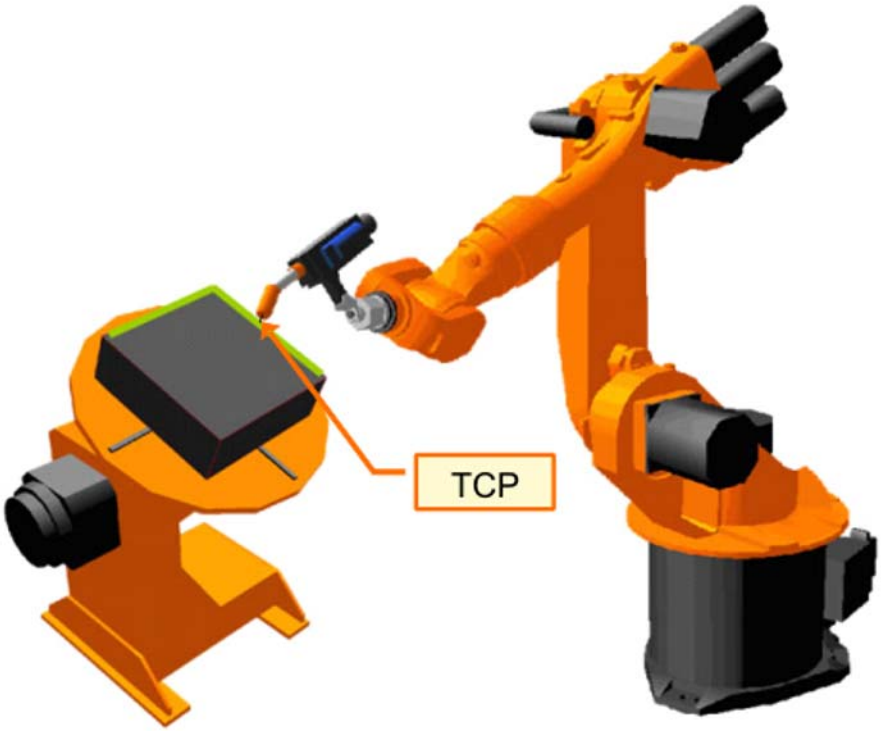


Fig. 3-23: Program mode with TCP

  - Furthermore, defined orientation control along the path is possible.

Tool calibration options

Tool calibration consists of 2 steps:

Step	Description
1	**Definition of the origin of the TOOL coordinate system** The following methods are available: ■ *XYZ 4-point* ■ *XYZ Reference*
2	**Definition of the orientation of the TOOL coordinate system** The following methods are available: ■ *ABC World* ■ *ABC 2-point*
Alternatively:	Direct entry of the values for the distance from the flange center point (X, Y, Z) and the rotation (A, B, C). ■ *Numeric input*

TCP calibration: XYZ 4-point method

The TCP of the tool to be calibrated can be moved to a reference point from 4 different directions. The reference point can be freely selected. The robot controller calculates the TCP from the different flange positions.

The 4 flange positions at the reference point must be sufficiently different from one another.

Procedure for XYZ 4-point method:

1. Select the menu **Start-up** > **Calibrate** > **Tool** > **XYZ 4-point**.
2. Assign a number and a name for the tool to be calibrated. Click on **OK** to confirm.

Numbers 1 to 16 are available.

3. Move the TCP to a reference point. Click on **OK** to confirm.
4. Move the TCP to the reference point from a different direction. Click on **OK** to confirm.

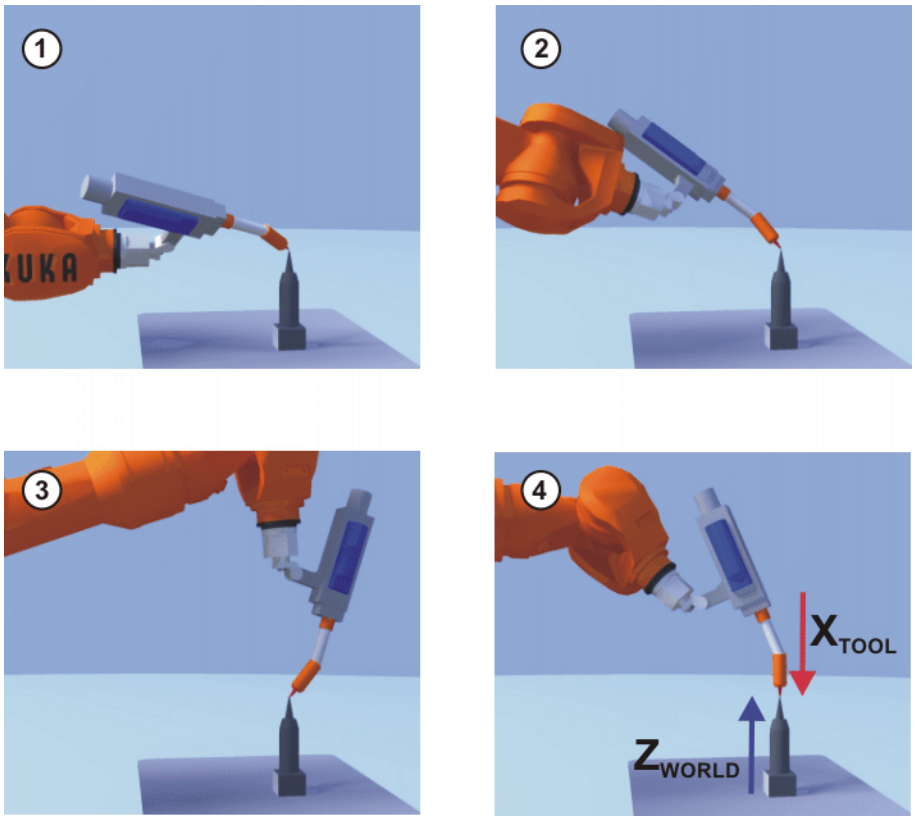


Fig. 3-24: XYZ 4-Point method

5. Repeat step 4 twice.
6. Press **Save**.

TCP calibration: XYZ Reference method

In the case of the *XYZ Reference method*, a new tool is calibrated with a tool that has already been calibrated. The robot controller compares the flange positions and calculates the TCP of the new tool.

The *XYZ Reference method* is used to teach multiple tools of the same type with similar geometry in the controller. The number of calibration runs required here can be reduced to two, unlike with the *XYZ 4-point method*.

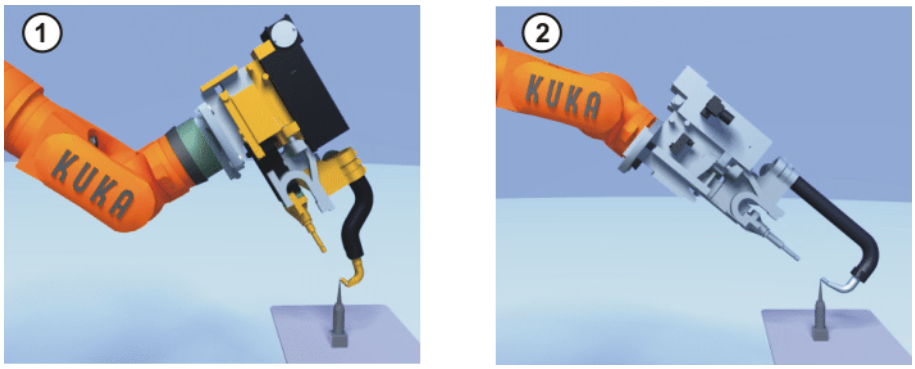


Fig. 3-25: XYZ Reference method

Procedure

1. A precondition for this is that a calibrated tool is mounted on the flange and that the data of the TCP are known.
2. In the main menu, select **Start-up** > **Calibrate** > **Tool** > **XYZ Reference**.
3. Assign a number and a name for the new tool. Confirm with **Next**.
4. Enter the TCP data of the calibrated tool. Confirm with **Next**.
5. Move the TCP to a reference point. Press **Calibrate**. Confirm with **Next**.

6. Move the tool away and remove it. Mount the new tool.
7. Move the TCP of the new tool to the reference point. Press **Calibrate**. Confirm with **Next**.
8. Press **Save**. The data are saved and the window is closed.

Or press **Load data**. The data are saved and a window is opened in which the payload data can be entered.

Orientation calibration: ABC World method

The axes of the TOOL coordinate system are aligned parallel to the axes of the WORLD coordinate system. This communicates the orientation of the TOOL coordinate system to the robot controller.

There are 2 variants of this method:

- **5D**: Only the tool direction is communicated to the robot controller. By default, the tool direction is the X axis. The directions of the other axes are defined by the system and cannot be detected easily by the user.
  Area of application: e.g. MIG/MAG welding, laser cutting or waterjet cutting
- **6D**: The directions of all 3 axes are communicated to the robot controller.
  Area of application: e.g. for weld guns, grippers or adhesive nozzles

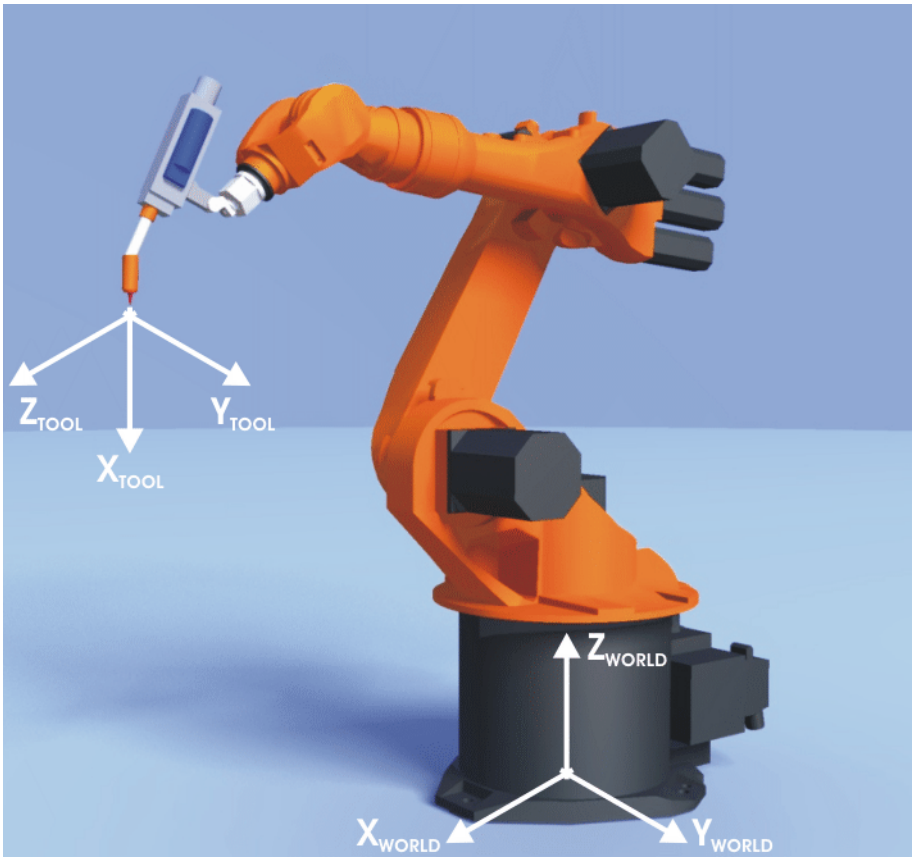


Fig. 3-26: ABC World method

Procedure – ABC World 5D method

a. Select the menu **Start-up** > **Calibrate** > **Tool** > **ABC World**.
b. Enter the number of the tool. Confirm with **OK**.
c. Select a variant in the box **5D/6D**. Confirm with **OK**.
d. If **5D** is selected:
   Align $+X_{TOOL}$ parallel to $-Z_{WORLD}$. ($+X_{TOOL}$ = tool direction)
   If **6D** is selected:
   Align the axes of the TOOL coordinate system as follows.
   - $+X_{TOOL}$ parallel to $-Z_{WORLD}$. ($+X_{TOOL}$ = tool direction)

- $+Y_{TOOL}$ parallel to $+Y_{WORLD}$
- $+Z_{TOOL}$ parallel to $+X_{WORLD}$

e. Confirm with **OK**.

f. Press **Save**.

Orientation calibration: ABC 2-point method

The axes of the TOOL coordinate system are communicated to the robot controller by moving to a point on the X axis and a point in the XY plane.

This method is used if it is necessary to define the axis directions with particular precision.

> The following procedure applies if the tool direction is the default tool direction (= X axis). If the tool direction has been changed to Y or Z, the procedure must also be changed accordingly.

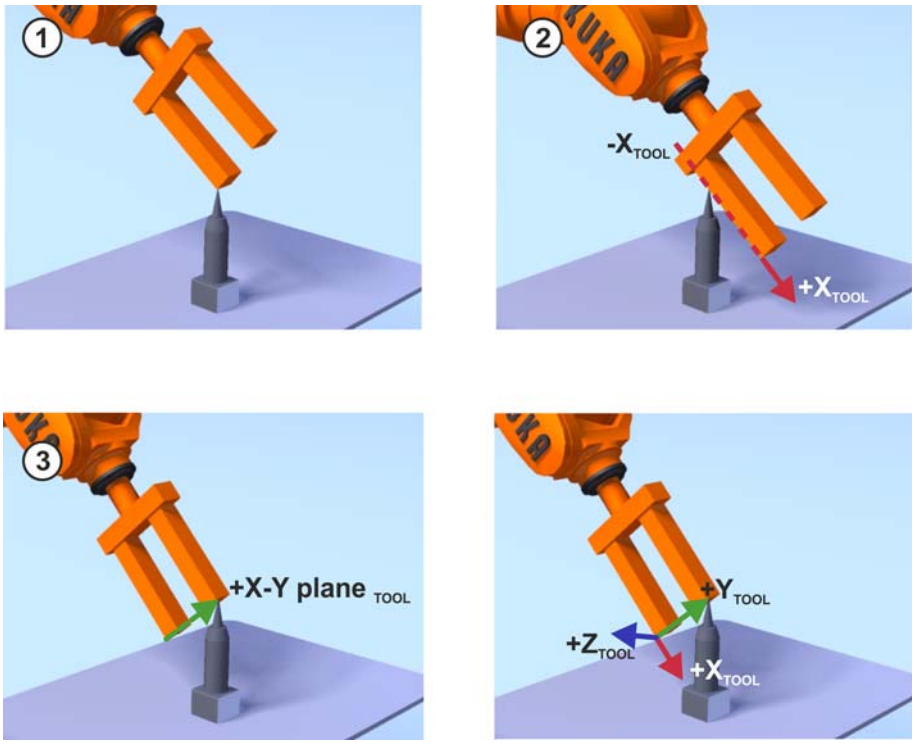


Fig. 3-27: ABC 2-point method

1. A precondition is that the TCP has already been calibrated by means of the XYZ method.
2. In the main menu, select **Start-up** > **Calibrate** > **Tool** > **ABC 2-point**.
3. Enter the number of the mounted tool. Confirm with **Next**.
4. Move the TCP to any reference point. Press **Calibrate**. Confirm with **Next**.
5. Move the tool so that the reference point on the X axis has a negative X value (i.e. move against the tool direction). Press **Calibrate**. Confirm with **Next**.
6. Move the tool so that the reference point in the XY plane has a negative Y value. Press **Calibrate**. Confirm with **Next**.
7. Either press **Save**. The data are saved and the window is closed.

   Or press **Load data**. The data are saved and a window is opened in which the payload data can be entered.

Gripper safety instructions for training mode

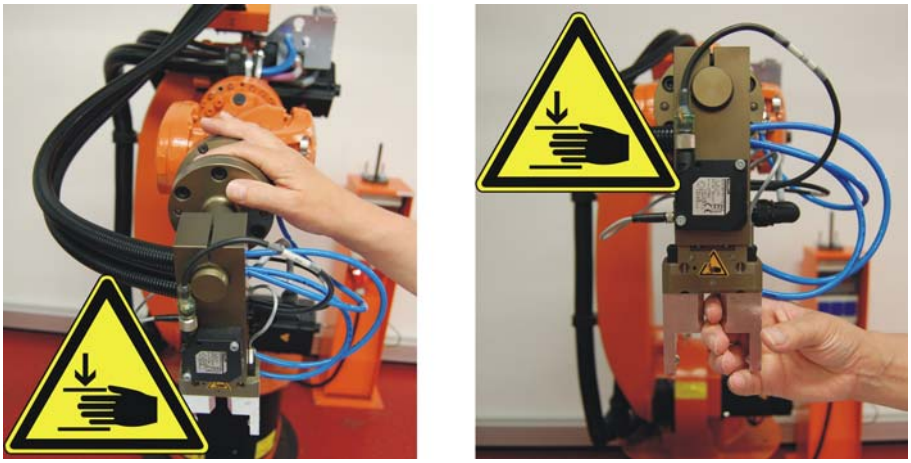


Fig. 3-28: Crushing hazards on the training gripper

⚠ WARNING When using the gripper system there is a risk of crushing and cutting. Anyone using the gripper must ensure that no part of the body can be crushed by the gripper.

Great care must be taken when clamping workpieces (cube, pen).

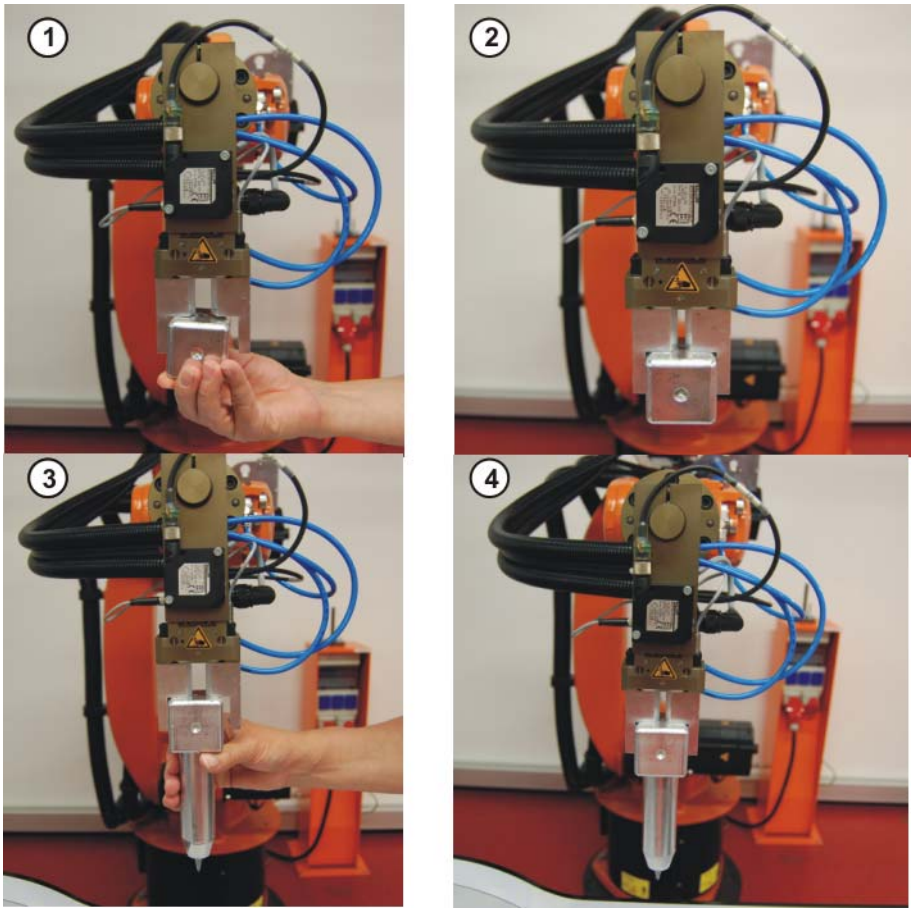


Fig. 3-29: Clamping objects in the training gripper

Item	Comments
1	Clamping the cube
2	Clamped cube

Item	Comments
3	Clamping a pen
4	Clamped pen

The collision protection device is triggered in the event of a collision.

The robot can be moved clear after the collision protection device has been triggered during a collision. One course participant presses the button (1) and keeps all parts of his/her body well away from the robot, the collision protection device and the gripper. Before moving the robot clear, the second participant ensures that no-one will be put at risk by the motion of the robot.

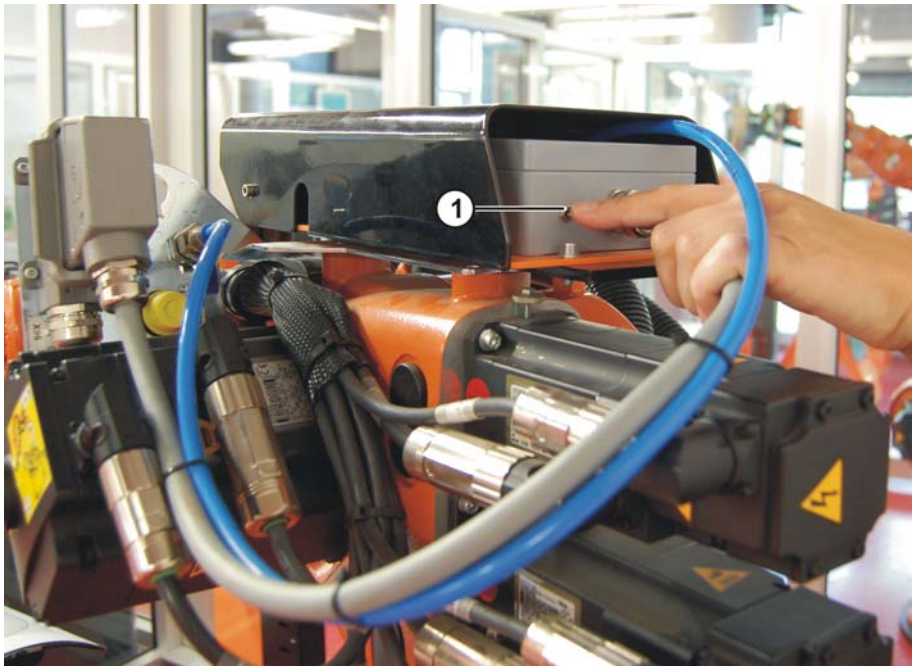


Fig. 3-30: Button for clearing the collision protection device, variant A

Fig. 3-31: Button for clearing the collision protection device, variant B

3.10.1 Exercise: Tool calibration – pen

Aim of the exercise

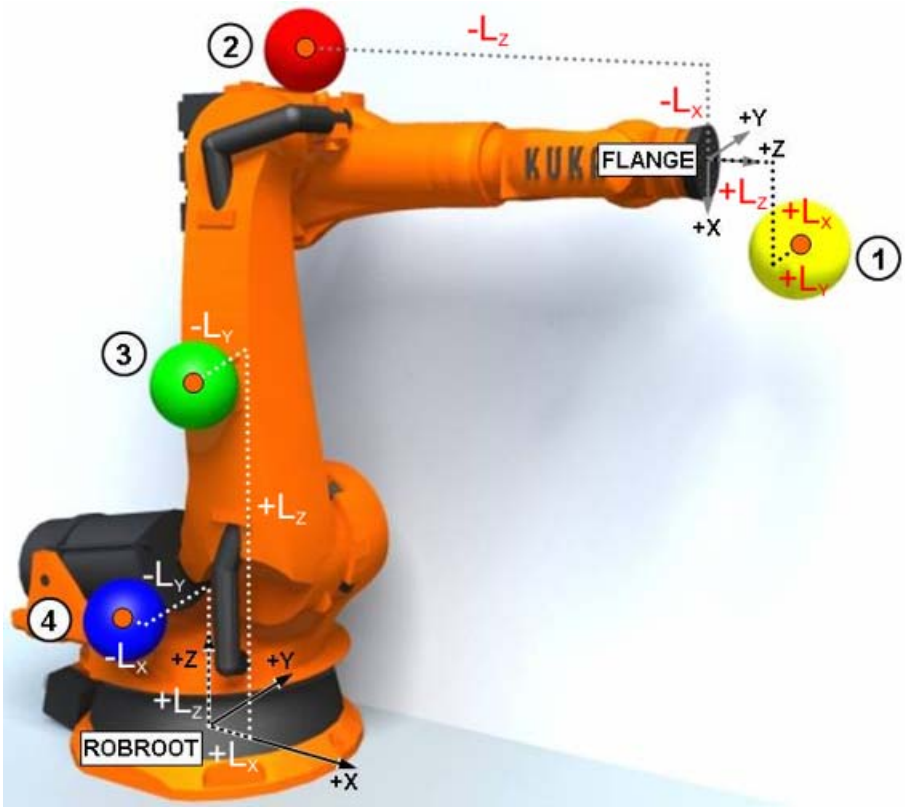
On successful completion of this exercise, you will be able to carry out the following activities:

- Calibration of a tool using the XYZ 4-point and ABC World methods
- Activation of a calibrated tool
- Moving the robot in the tool coordinate system
- Moving the robot in the tool direction
- Reorientation of the tool about the Tool Center Point (TCP)

Preconditions

The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of the various TCP calibration methods, especially the XYZ 4-point method
- Theoretical knowledge of the various tool orientation calibration methods, especially the ABC World method
- Theoretical knowledge of robot load data and how to enter them



1	Payload	3	Supplementary load on axis 2
2	Supplementary load on axis 3	4	Supplementary load on axis 1

Task description

Carry out the following tasks: Pen calibration

1. Calibrate the TCP of the pen using the XYZ 4-point method. Use the black metal tip as the reference point. Remove the uppermost pen from the pen magazine and clamp it in the gripper. Use the **tool number 2** and assign the name **Pen1**.
2. Save the tool data.
3. Calibrate the tool orientation using the ABC World 5D method.
4. The tolerance should not exceed 0.95 mm. In practice, this value is not sufficient. It is better to achieve tolerances of 0.5 mm or even 0.3 mm.
5. Enter the load data for the gripper with pen as tool number 2.
   Use the correct gripper with pen for this (see below).

6. Save the TOOL data and test the robot motion with the pen in the TOOL coordinate system

Tool load data for gripper with pen

- Training gripper for KR 16

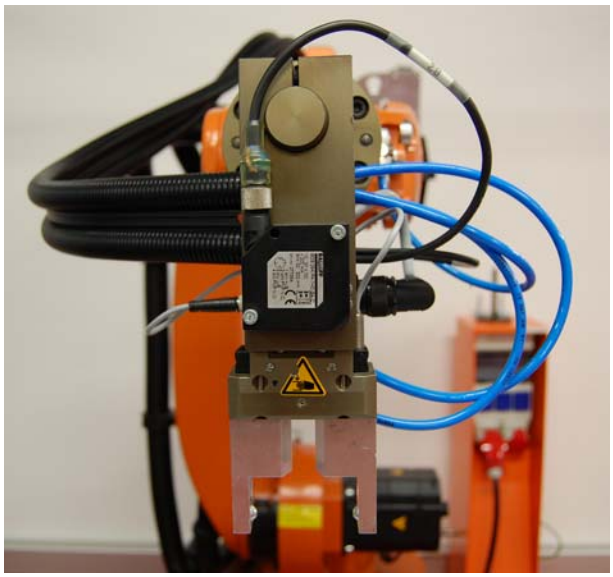


Fig. 3-32: Training gripper for KR 16

Mass:		
M = 4.9 kg		
Center of mass:		
X = 53 mm	Y = 49 mm	Z = 65 mm
Orientation:		
A = 0°	B = 0°	C = 0°
Moments of inertia:		
J_X = 0.02 kgm^2	J_Y = 0.03 kgm^2	J_Z = 0.15 kgm^2

- Training gripper for modular cell

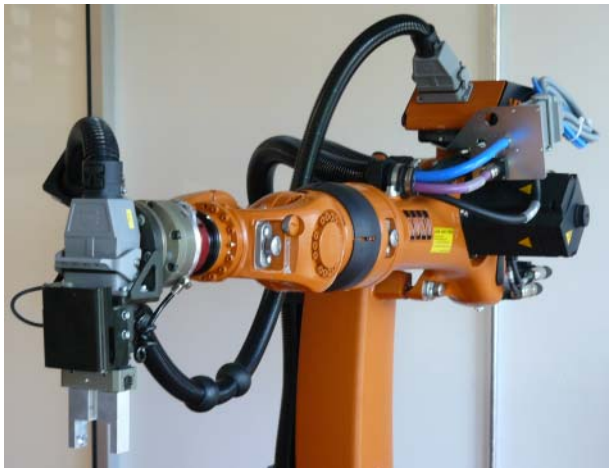


Fig. 3-33: Training gripper for modular cell

Mass:		
M = 5 kg		
Center of mass:		
X = 23 mm	Y = -38 mm	Z = 84 mm
Orientation:		

A = 0°	B = 0°	C = 0°
Moments of inertia:		
J_X = 0.02 kgm^2	J_Y = 0.07 kgm^2	J_Z = 0.14 kgm^2

- Training gripper for mobile cell

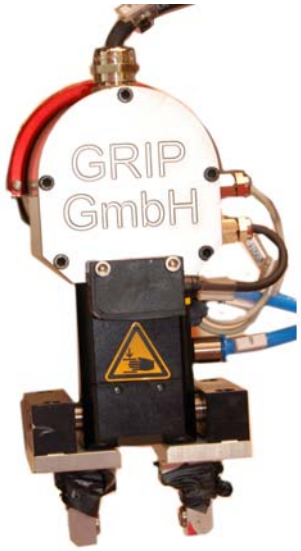


Fig. 3-34: Training gripper for mobile cell

Mass:		
M = 2.14 kg		
Center of mass:		
X = 30 mm	Y = 0 mm	Z = 64 mm
Orientation:		
A = 0°	B = 0°	C = 0°
Moments of inertia:		
J_X = 0.002 kgm^2	J_Y = 0.004 kgm^2	J_Z = 0.003 kgm^2

What you should know after the exercise:

1. What are the advantages of tool calibration?

. . . . . .

. . . . . .

. . . . . .

2. What tool calibration methods are there?

. . . . . .

. . . . . .

. . . . . .

3. What is calculated using the XYZ 4-point method?

. . . . . .

. . . . . .

3.10.2 Exercise: Tool calibration – gripper, 2-point method

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Tool calibration using the XYZ 4-point and ABC 2-point methods
- Activation of a calibrated tool
- Moving the robot in the tool coordinate system
- Moving the robot in the tool direction
- Reorientation of the tool about the Tool Center Point (TCP)

Preconditions

The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of the various TCP calibration methods, especially the 2-point method
- Theoretical knowledge of robot load data and how to enter them

Task description

Carry out the following tasks:

1. Use the name “Gripper_new” and the tool number “3” for tool calibration of the gripper.
2. Calibrate the TCP of the gripper using the XYZ 4-point method as illustrated:

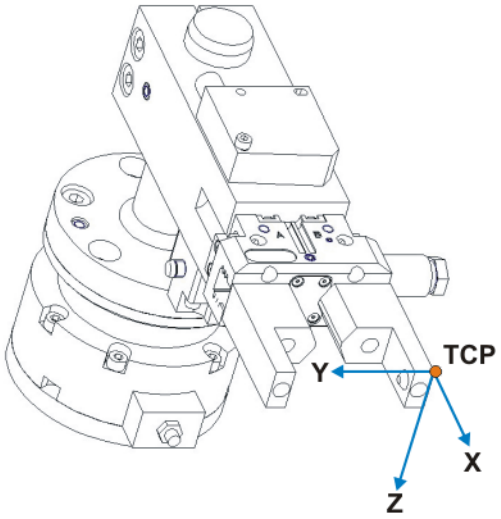


Fig. 3-35: College gripper: position of the TCP

3. Calibrate the orientation of the gripper coordinate system using the ABC 2-point method.
4. Enter the load data for the gripper as tool number 3.
   Use the correct gripper for this (see below).
5. Save the TOOL data and test jogging with the gripper in the TOOL coordinate system.

Tool load data for gripper

- Training gripper for KR 16

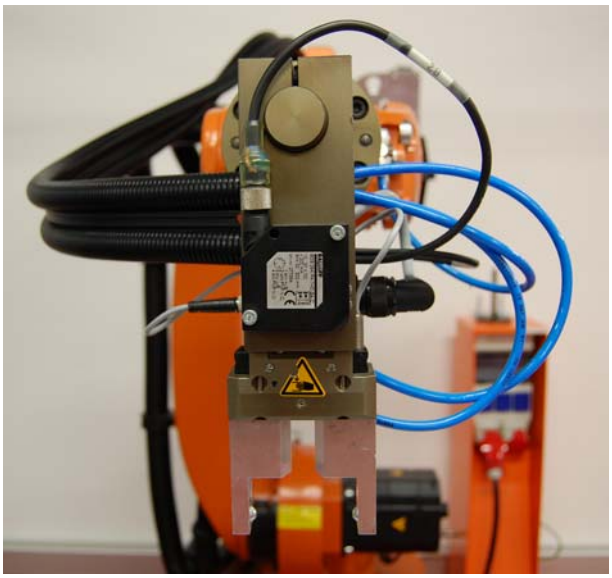


Fig. 3-36: Training gripper for KR 16

Mass:		
M = 4.3 kg		
Center of mass:		
X = 41 mm	Y = 35 mm	Z = 62 mm
Orientation:		
A = 0°	B = 0°	C = 0°
Moments of inertia:		
J_X = 0.03 kgm^2	J_Y = 0.04 kgm^2	J_Z = 0.12 kgm^2

- Training gripper for modular cell

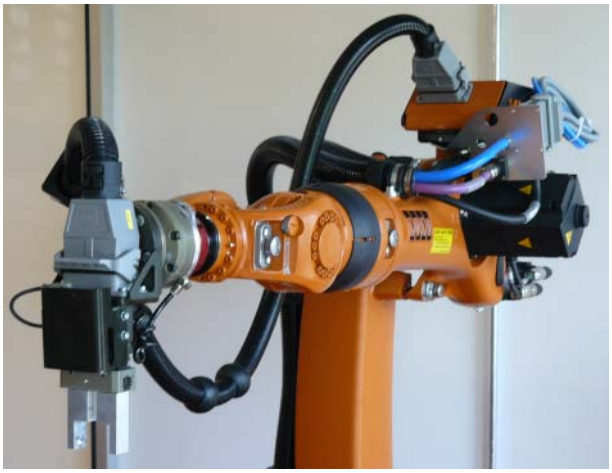


Fig. 3-37: Training gripper for modular cell

Mass:		
M = 4.4 kg		
Center of mass:		
X = 7 mm	Y = -24 mm	Z = 80 mm
Orientation:		
A = 0°	B = 0°	C = 0°
Moments of inertia:		
J_X = 0.02 kgm^2	J_Y = 0.05 kgm^2	J_Z = 0.11 kgm^2

- Training gripper for mobile cell

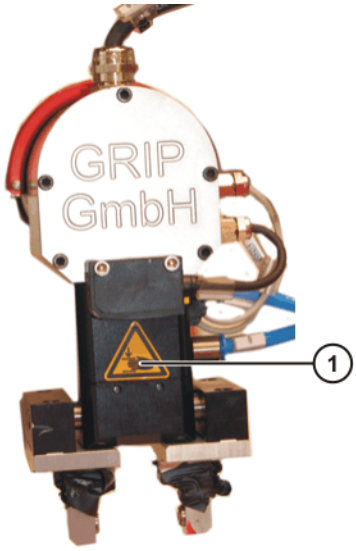


Fig. 3-38: Plate on gripper

Mass:		
M = 2 kg		
Center of mass:		
X = 23 mm	Y = 0 mm	Z = 61 mm
Orientation:		
A = 0°	B = 0°	C = 0°
Moments of inertia:		
J_X = 0.002 kgm^2	J_Y = 0.004 kgm^2	J_Z = 0.003 kgm^2

Alternative task description

Alternatively, the gripper can also be calibrated by means of numeric input:

Carry out the following tasks:

1. Use the name “Gripper_new” and the tool number “3” for tool calibration of the gripper.
2. Calibrate the TCP of the gripper using “numeric input”:

Training gripper for KR 16					
X	Y	Z	A	B	C
122.64 mm	177.73 mm	172.49 mm	45°	0°	180°

Training gripper for modular cell					
X	Y	Z	A	B	C
175.38 mm	-123.97 mm	172.71 mm	-45°	0°	-180°

Training gripper for mobile cell					
X	Y	Z	A	B	C
12.0 mm	0 mm	77.0 mm	0°	0°	0°

What you should now know:

1. Which icon represents the tool coordinate system?

a) b) c) d)




2. What is the maximum number of tools that the controller can manage?

. . . . . .

3. What does the value -1 in the tool load data mean?

. . . . . .

. . . . . .

3.11 Moving the robot in the base coordinate system

Motion in the base coordinate system

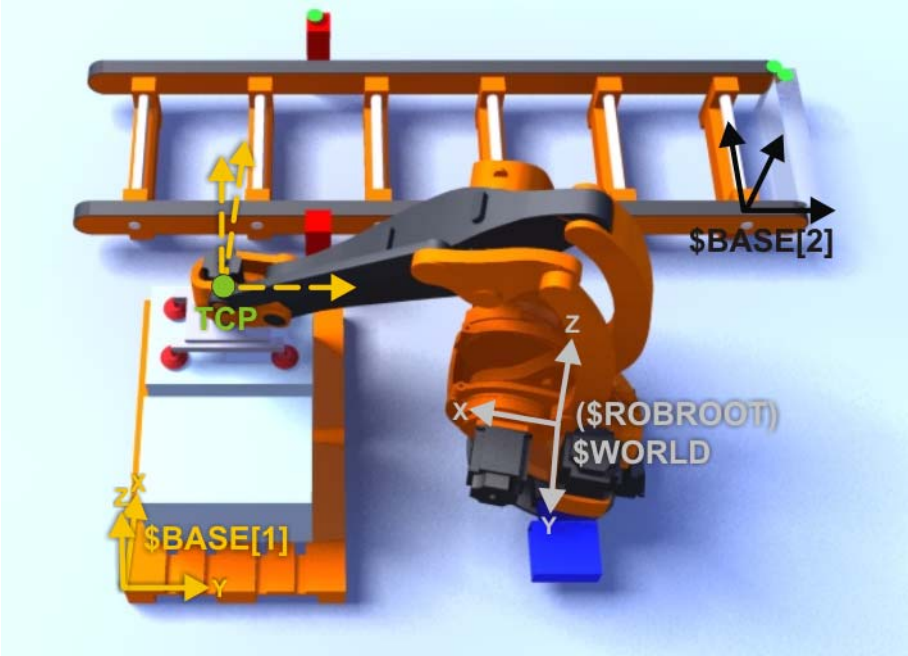


Fig. 3-39: Jogging in the base coordinate system

Description of bases

- The robot tool can be moved with reference to the coordinate axes of the base coordinate system. Base coordinate systems can be calibrated individually and are often oriented along the edges of workpieces, workpiece locations or pallets. This allows convenient jogging!

  In this case, **all** required robot axes move. Which axes these are is determined by the system and depends on the motion.
- The jog keys or Space Mouse of the KUKA smartPAD are used for this.
- There are 32 base coordinate systems to choose from.
- The velocity can be modified (jog override: HOV).
- Jogging is only possible in T1 mode.
- The enabling switch must be pressed.

Principle of jogging – base

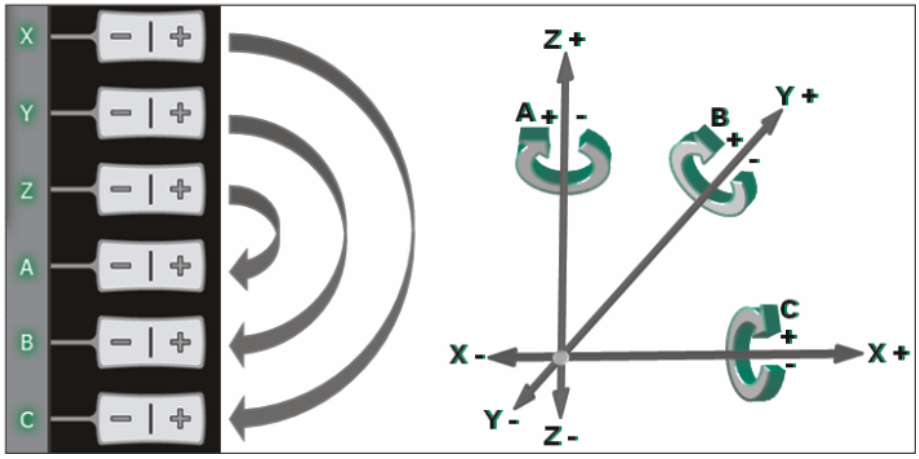


Fig. 3-40: Cartesian coordinate system

A robot can be moved in a coordinate system in two different ways:

- Translational (in a straight line) along the orientation directions of the coordinate system: X, Y, Z

- Rotational (turning/pivoting) about the orientation directions of the coordinate system: angles A, B and C

In the case of a motion command (e.g. jog key pressed), the controller first calculates a path. The starting point of the path is the tool center point (TCP). The direction of the path is specified by the world coordinate system. The controller then controls the axes to guide the tool along this path (translation) or about it (rotation).

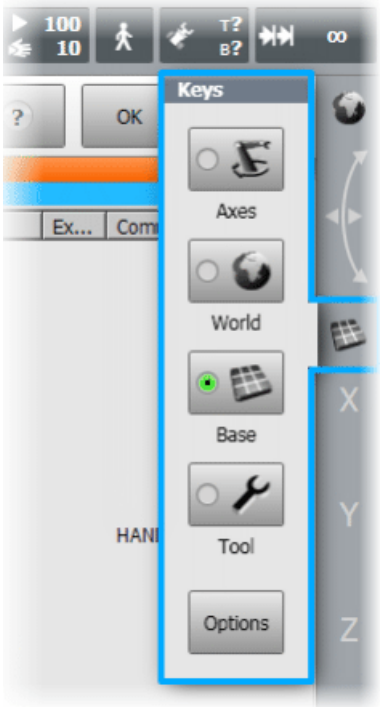
Use of the base coordinate system:

- The motion of the robot is always predictable as soon as the base coordinate system is known.
- Here also, the Space Mouse allows intuitive operator control. A precondition is that the operator is standing correctly relative to the robot or the base coordinate system. If necessary, select the X axis as the reference axis.
- Enables parallel motion, e.g. parallel to a workpiece or the edge of a pallet.

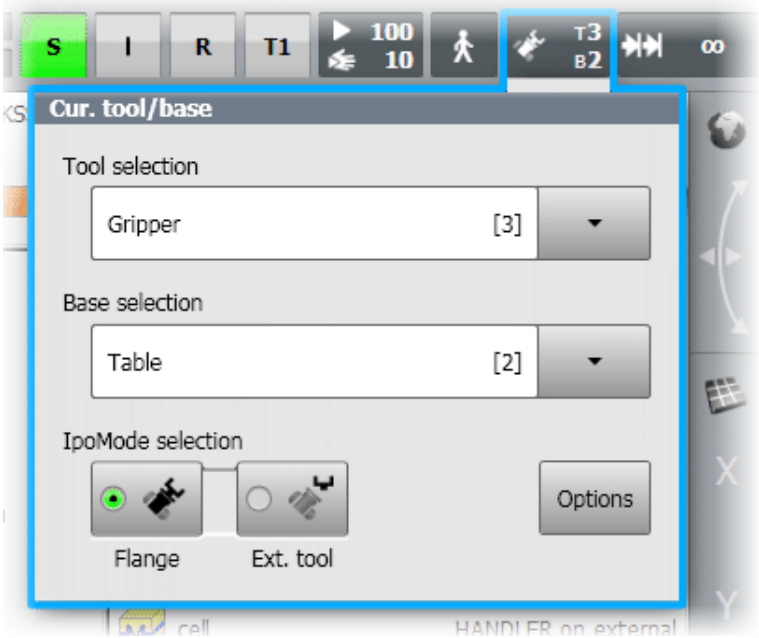
If the correct tool coordinate system is also set, re-orientation about the TCP is possible in the base coordinate system.

Procedure

1. Select **Base** as the option for the jog keys.



2. Select tool and base.

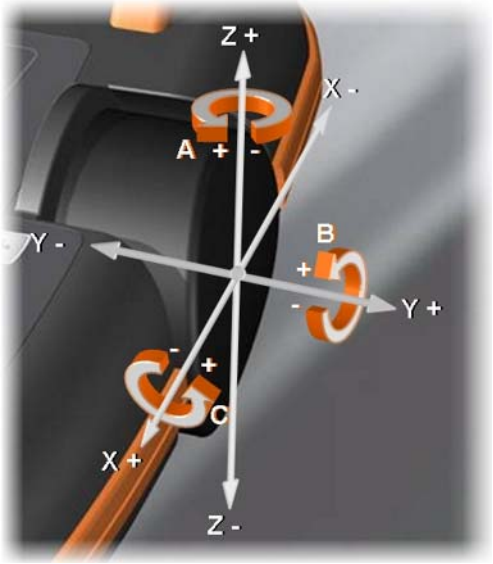


3. Set jog override.

4. Press the enabling switch into the center position and hold it down.

5. Move in the desired direction using the jog keys.

6. Alternatively, jogging can be carried out using the Space Mouse.



3.11.1 Exercise: Jogging in the base coordinate system

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Jog the robot, in the workpiece coordinate system, by means of the jog keys and Space Mouse
- Jog along predefined workpiece edges

Preconditions

The following are preconditions for successful completion of this exercise:

- Completion of safety instruction

> **Note!**
> Safety instruction must be completed and documented before commencing this exercise!

- Theoretical knowledge of jogging in the workpiece coordinate system

Task description

Carry out the following tasks:

1. Release and acknowledge the EMERGENCY STOP.
2. Ensure that T1 mode is set.
3. Activate the base coordinate system *D_Red Base Straight*.
4. Clamp the pen in the gripper and select the tool coordinate system *D_Pen165*.
5. Jog the robot in the base coordinate system with various different jog override (HOV) settings using the jog keys and Space Mouse.
6. Move the pen along the outer contour on your worktable.

3.12 Base calibration

Description

Calibration of a base means the creation of a coordinate system at a specific point in the robot environment, relative to the world coordinate system. The objective is for the jog motions and the programmed robot positions to refer to this coordinate system. For this reason, defined edges of workpiece locations, shelves, pallets or machines, for example, are useful as reference points for a base coordinate system.

Base calibration is carried out in two steps:

1. Determination of the coordinate origin
2. Definition of the coordinate axes

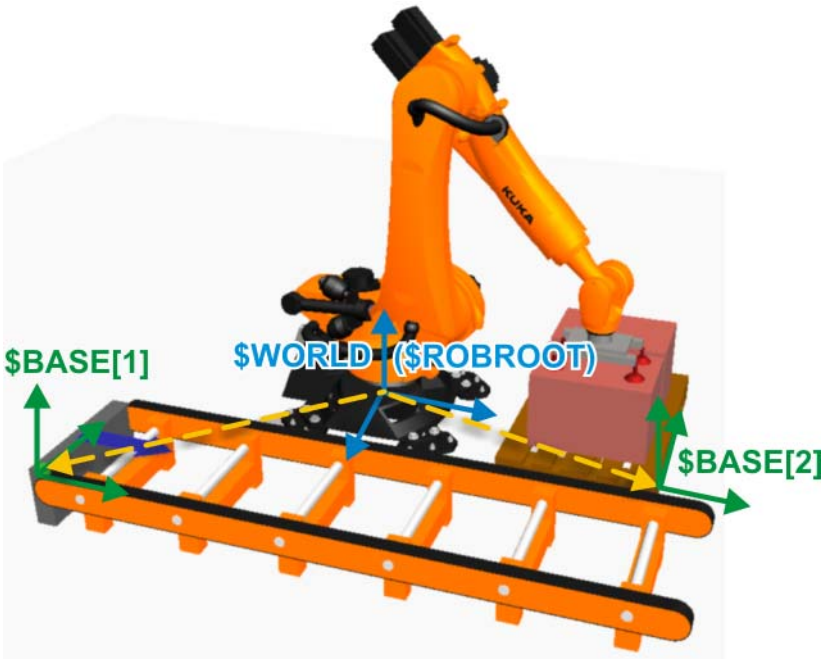


Fig. 3-41: Base calibration

Advantages

Base calibration has the following advantages:

- Motion along the edges of the workpiece:
  The TCP can be jogged along the edges of the work surface of the workpiece.

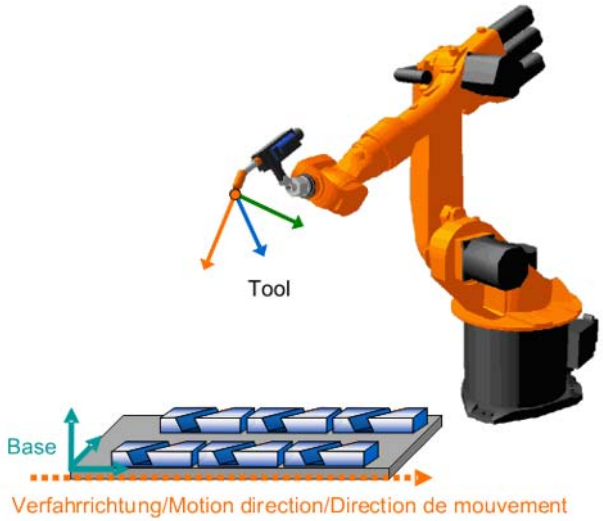


Fig. 3-42: Advantages of base calibration: motion direction

- Reference coordinate system:
  Taught points refer to the selected coordinate system.

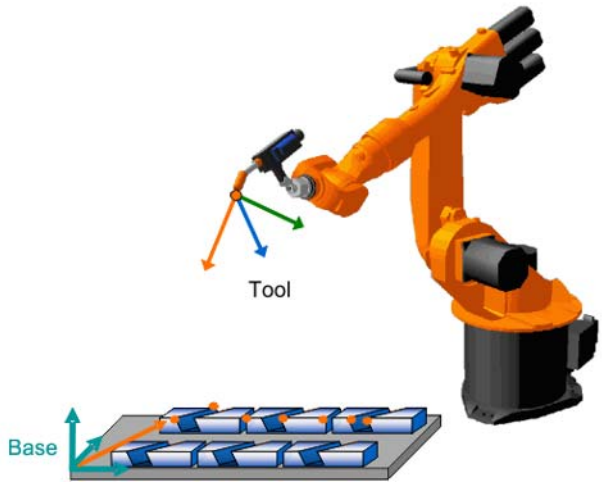


Fig. 3-43: Advantages of base calibration: reference to the desired coordinate system

- Correction / offset of the coordinate system:
  Points can be taught relative to the base. If it is necessary to offset the base, e.g. because the work surface has been offset, the points move with it and do not need to be retaught.

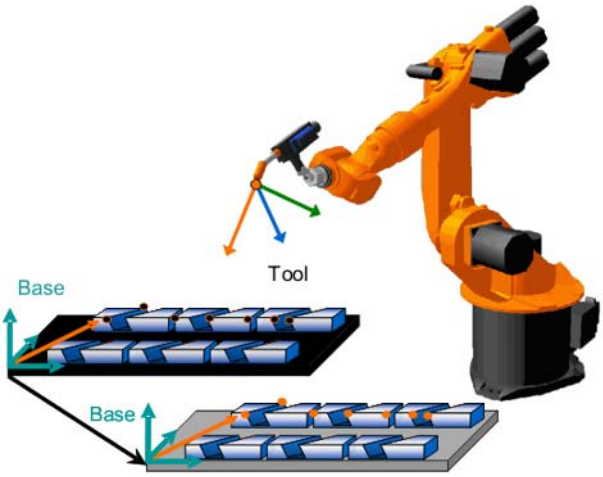


Fig. 3-44: Advantages of base calibration: offset of the base coordinate system

- Using multiple base coordinate systems:
  Up to 32 different coordinate systems can be created and used depending on the specific program step.

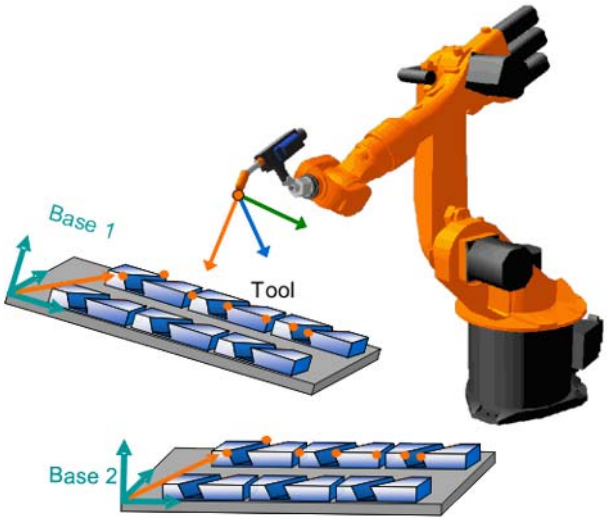


Fig. 3-45: Advantages of base calibration: use of multiple base coordinate systems

Base calibration options

The following base calibration methods are available:

Methods	Description
3-point method	1. Definition of the origin 2. Definition of the positive X axis 3. Definition of the positive Y axis (XY plane)
Indirect method	The indirect method is used if it is not possible to move to the origin of the base, e.g. because it is inside a workpiece or outside the workspace of the robot. The TCP is moved to 4 points whose positions relative to the base that is to be calibrated are known and the coordinates of which must be known (CAD data). The robot controller calculates the base from these points.
Numeric input	Direct entry of the values for the distance from the world coordinate system (X, Y, Z) and the rotation (A, B, C).

Procedure for 3-point method

Base calibration can only be carried out with a previously calibrated tool (TCP must be known).

1. In the main menu, select **Start-up** > **Calibrate** > **Base** > **3-point**.
2. Assign a number and a name for the base. Confirm with **Next**.
3. Enter the number of the tool whose TCP is to be used for for base calibration. Confirm with **Next**.
4. Move the TCP to the origin of the new base. Press the **Calibrate** softkey and confirm the position with **Yes**.

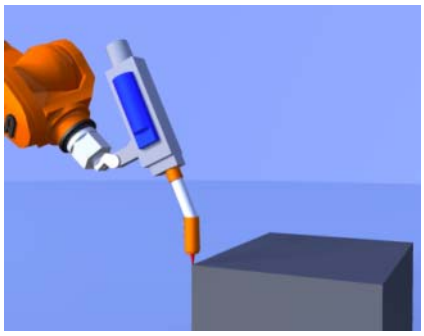


Fig. 3-46: First point: origin

5. Move the TCP to a point on the positive X axis of the new base. Press **Calibrate** and confirm the position with **Yes**.

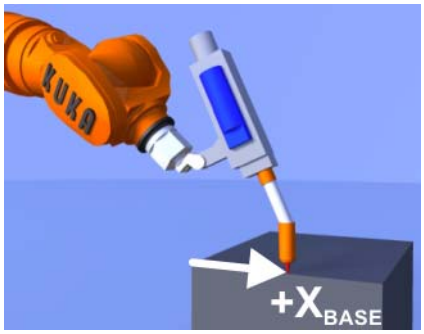


Fig. 3-47: Second point: X axis

6. Move the TCP to a point in the XY plane with a positive Y value. Press **Calibrate** and confirm the position with **Yes**.

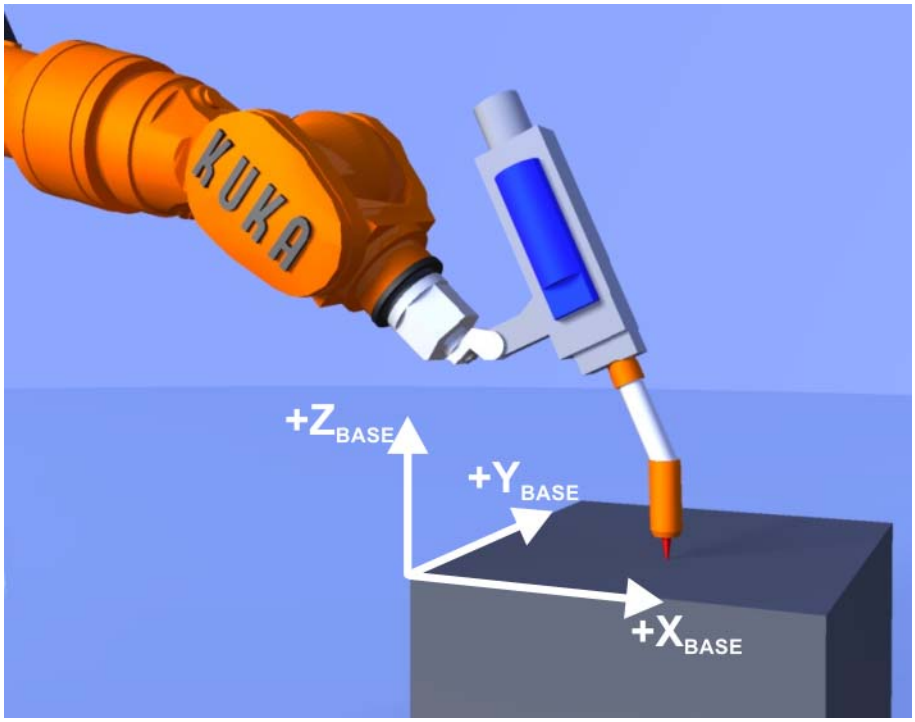


Fig. 3-48: Third point: XY plane

7. Press **Save**.
8. Close the menu.

The three calibration points must not be in a straight line. A minimum angle (default setting 2.5°) between the points must be maintained.

3.12.1 Exercise: Base calibration – table, 3-point method

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Define any base
- Calibrate a base
- Activate a calibrated base for jogging
- Move the robot in the base coordinate system

Preconditions

The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of the base calibration methods, especially the 3-point method

Task description

Carry out the following tasks:

1. Calibrate the blue base on the table using the 3-point method. Assign the **base number 1** and the name **blue**. Use pen1 which has already been calibrated (tool number 2) as the calibration tool.
2. Save the data of the calibrated base.
3. Calibrate the red base on the table using the 3-point method. Assign the **base number 2** and the name **red**. Use pen1 which has already been calibrated (tool number 2) as the calibration tool.
4. Save the data of the calibrated base.
5. Move the tool to the origin of the blue coordinate system, thus causing the actual position to be displayed in “Cartesian” mode.

X	Y	Z	A	B	C
.....	.....	.....	.....	.....	.....

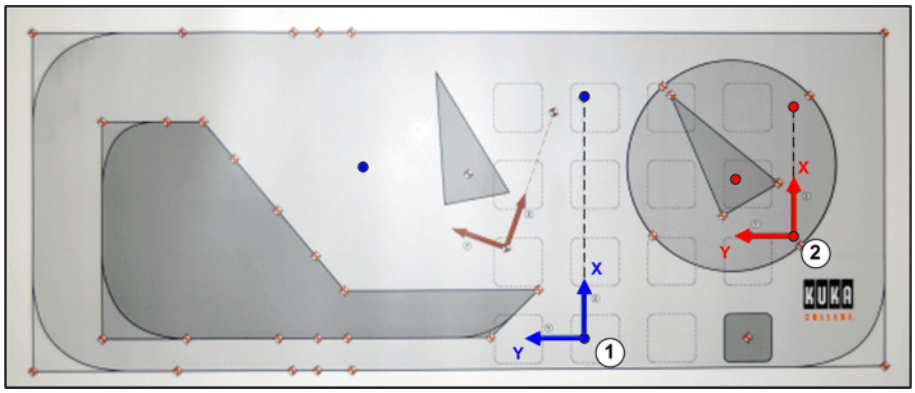


Fig. 3-49: Base calibration on the table

What you should know after the exercise:

1. Why should a base be calibrated?

. . . . . .

. . . . . .

. . . . . .

2. Which icon represents the base coordinate system?

. . . . . .

a) b) c) d)

3. What base calibration methods are there?

. . . . . .

. . . . . .

. . . . . .

4. What is the maximum number of base systems that the controller can manage?

. . . . . .

. . . . . .

. . . . . .

5. Describe calibration using the 3-point method.

. . . . . .

. . . . . .

. . . . . .

3.13 Displaying the current robot position

Options for displaying robot positions

The current robot position can be displayed in two different ways:

- **Axis-specific:**

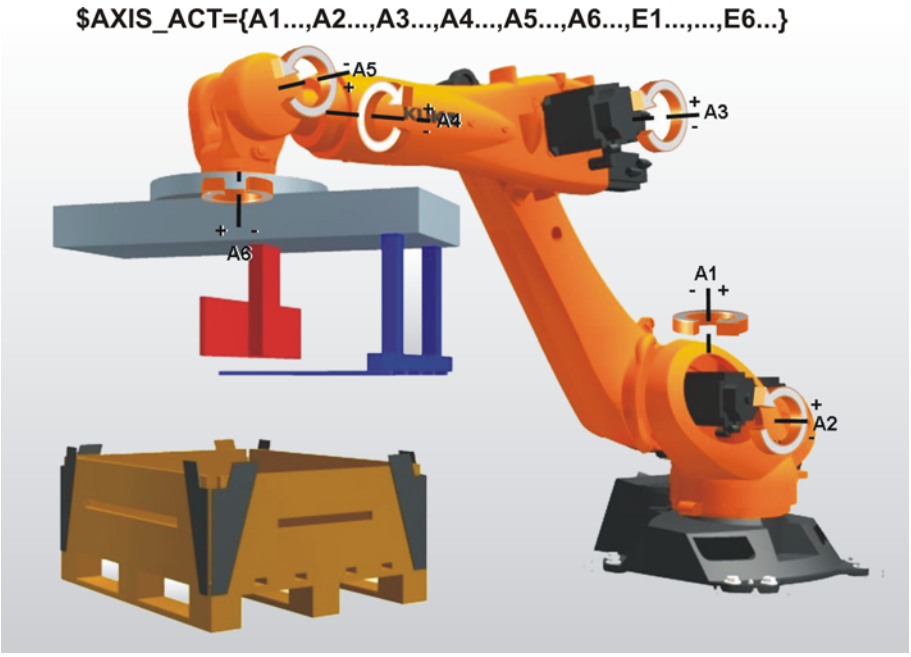


Fig. 3-50: Axis-specific robot position

  The current axis angle is displayed for every axis: this corresponds to the absolute axis angle relative to the zero position of the axis.

- **Cartesian:**

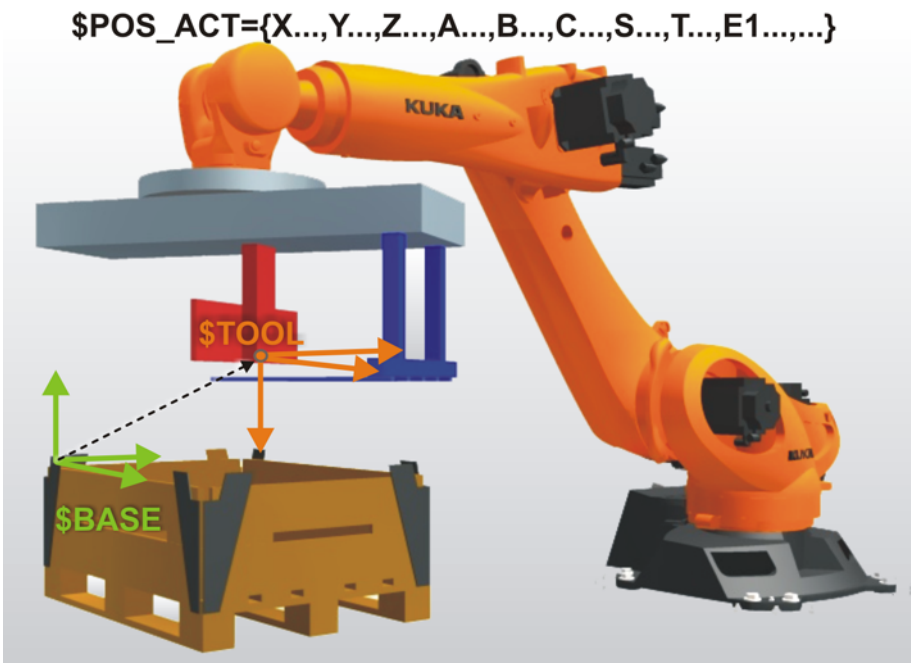


Fig. 3-51: Cartesian position

The current position of the current TCP (tool coordinate system) is displayed relative to the currently selected base coordinate system.

If no tool coordinate system is selected, the flange coordinate system applies!

If no base coordinate system is selected, the world coordinate system applies!

Cartesian position with different base coordinate systems

Looking at the Figure below, it is immediately apparent that the robot is in the same position in all three instances. The position display contains different values in each of the three cases, however:

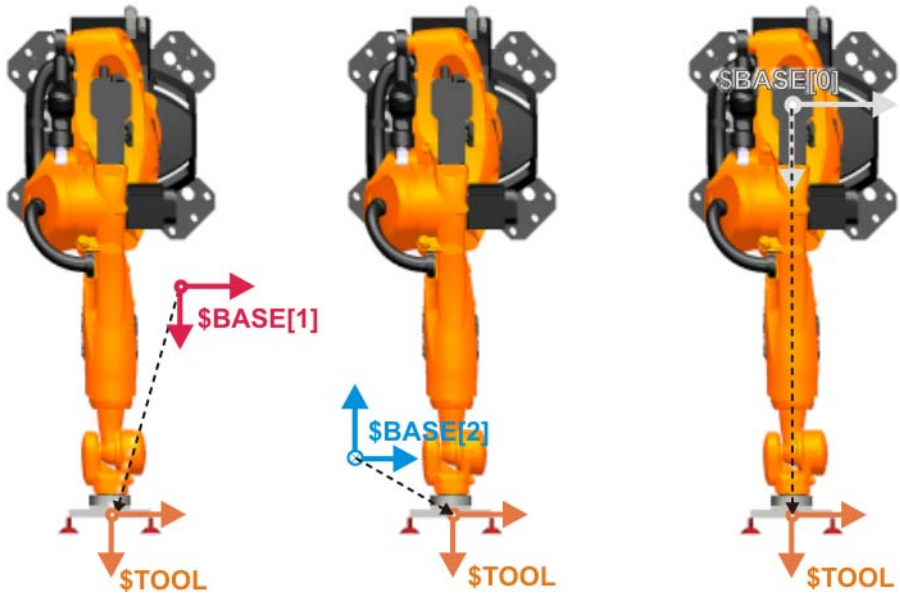


Fig. 3-52: Three tool positions – one robot pose!

The position of the tool coordinate system/TCP is displayed in the respective base coordinate system:

- for base **1**
- for base **2**

- for base **0**: this corresponds to the robot root point coordinate system (in most cases identical to the world coordinate system)!

Fig. 3-53: Selecting tool and base

The Cartesian actual value display only provides sensible values if the correct base and tool are selected!

Displaying the robot position

Procedure:

- Select **Display** > **Actual position** in the menu. The Cartesian actual position is displayed.
- To display the axis-specific actual position, press **Axis spec.**.
- To display the Cartesian actual position again, press **Cartesian**.

4 Executing robot programs

4.1 Overview

The following contents are explained in this training module:

- Performing an initialization run
- Selecting and starting robot programs

4.2 Performing an initialization run

BCO run

The initialization run of a KUKA robot is called a BCO run.

BCO stands for **B**lock **CO**incidence. Coincidence means “coming together” of events in time/space.

A BCO run is carried out in the following cases:

- Program selection
- Program reset
- Jogging in program mode
- Change in program
- Block selection

Examples for the performance of a BCO run:

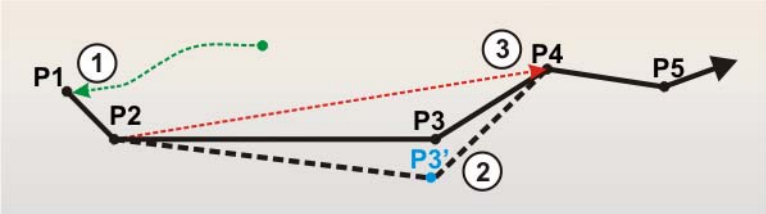


Fig. 4-1: Examples of reasons for a BCO run

1 BCO run to the home position following program selection or reset
2 BCO run following modification of a motion command: point deleted, taught, etc.
3 BCO run following block selection

Reasons for a BCO run

A BCO run is necessary to ensure that the current robot position matches the coordinates of the current point in the robot program.

Path planning can only be carried out if the current robot position is the same as a programmed position. The TCP must therefore always be moved onto the path.

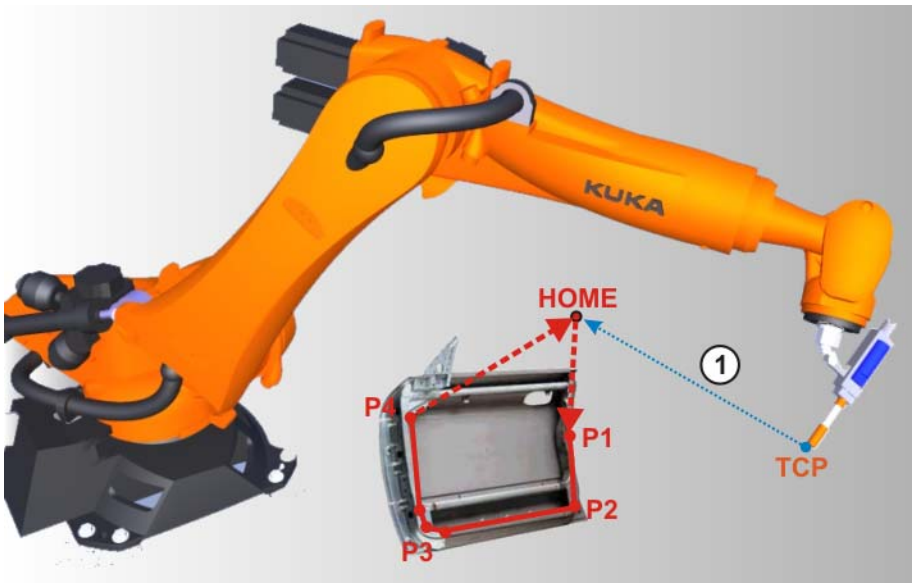


Fig. 4-2: Example of a BCO run on program selection

1 BCO run to the home position following program selection or reset

4.3 Selecting and starting robot programs

Selecting and starting robot programs

If a robot program is to be executed, it must be selected. The robot programs are available in the Navigator in the user interface. Motion programs are generally created in folders. The Cell program (management program for controlling the robot from a PLC) is always located in the folder “R1”.

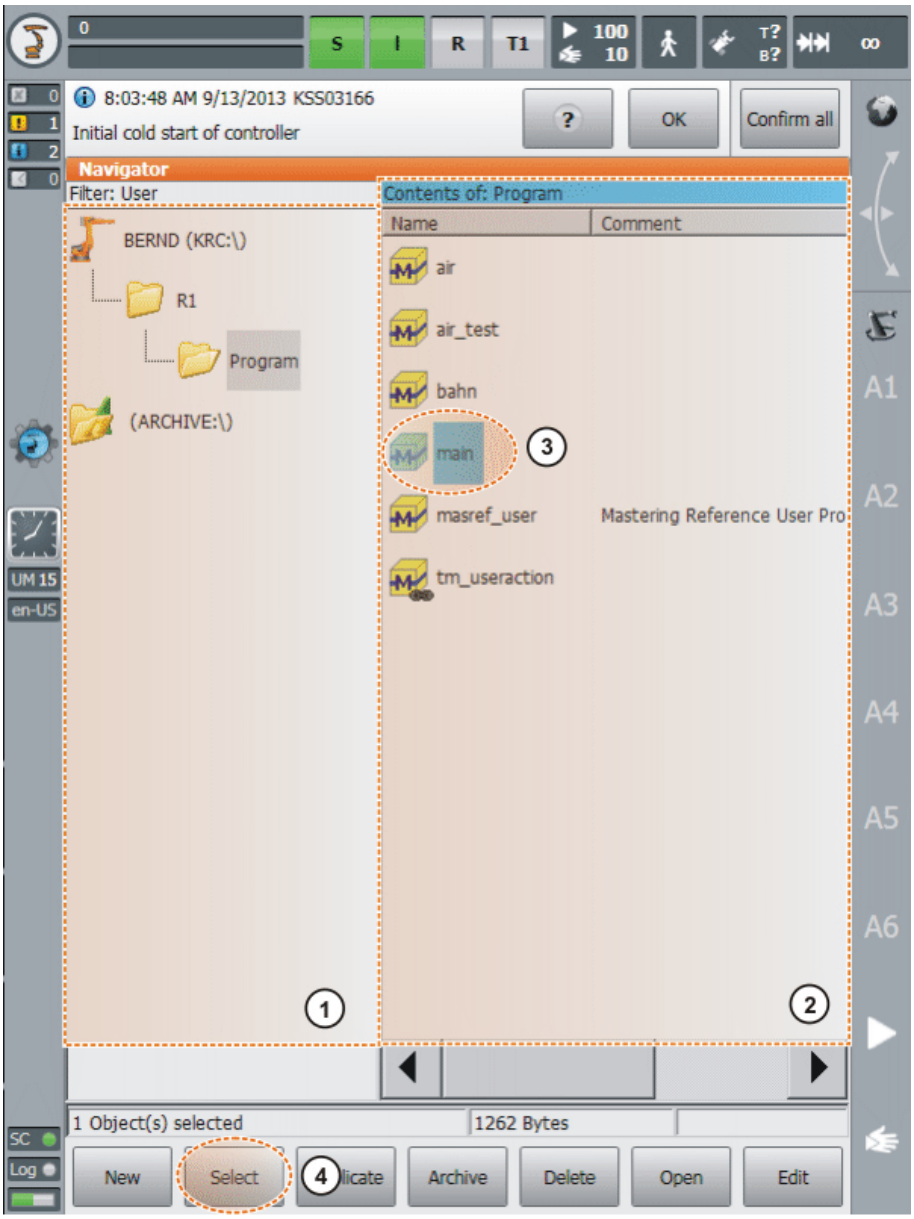


Fig. 4-3: Navigator

1 Navigator: directory/drive structure
2 Navigator: directory/data list
3 Selected program
4 Button for selecting a program

The Start forwards key and Start backwards key are available for starting a program.

It is only possible to execute a program backwards with the key if the INI line of a program has been executed and the robot has only just executed in the forward direction the path that is now to be executed backwards. If, however, the robot is moved manually, e.g. after a program stop, backward motion is no longer possible.

Fig. 4-4: Program execution directions: forwards/backwards

Item	Description
1	Start key **forwards**
2	Start key **backwards**

When a program is executed, there are various **program run modes** available for program-controlled robot motion:

	**GO** ■ Program runs continuously to the end of the program. ■ In test mode, the Start key must be held down.
	**Motion Step** ■ In the program run mode Motion Step, each motion command is executed separately. ■ At the end of each motion, Start must be pressed again.
	**Single Step** | Only available in the user group “Expert”! ■ In Incremental Step mode, the program is executed line by line (irrespective of the contents of the individual lines). ■ The Start key must be pressed again after every line.

What does a robot program look like?

```
 1 DEF KUKA_Prog()
 2
 3 INI
 4 SPTP HOME VEL= 100 % DEFAULT
 5 SPTP P1 VEL= 100 % PDAT1 Tool[5] Base[10]
 6 SPTP P2 VEL= 100 % PDAT1 Tool[5] Base[10]
 7 SLIN P3 VEL= 1 m/s CPDAT1 Tool[5] Base[10]
 8 SLIN P4 VEL= 1 m/s CPDAT2 Tool[5] Base[10]
 9 SPTP P5 VEL= 100% PDAT1 Tool[5] Base[10]
10 SPTP HOME VEL= 100 % DEFAULT
11
12 END
```

Line	Description
1 and 12	**Only visible in the user group “Expert”:** ■ “DEF *Program name()*” always stands at the start of a program. ■ “END” defines the end of a program.
3	■ The “INI” line contains calls of standard parameters that are required for correct execution of the program. ■ The “INI” line must always be executed first!
4 to 10	■ Actual program text with motion commands, wait commands, logic commands, etc. ■ The motion command “PTP Home” is often used at the start and end of a program, as this is a known and clearly defined position.

Program state

Icon	Color	Description
	Gray	No program is selected.
	Yellow	The block pointer is situated on the first line of the selected program.
	Green	The program is selected and is being executed.
	Red	The selected and started program has been stopped.
	Black	The block pointer is situated at the end of the selected program.

Starting a program

Procedure for starting robot programs:

1. Select program

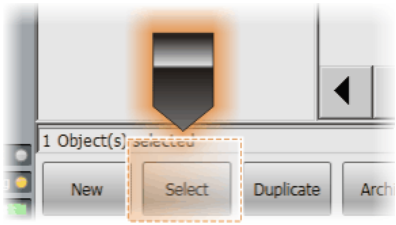


Fig. 4-5: Program selection

2. Set program velocity (program override, POV).

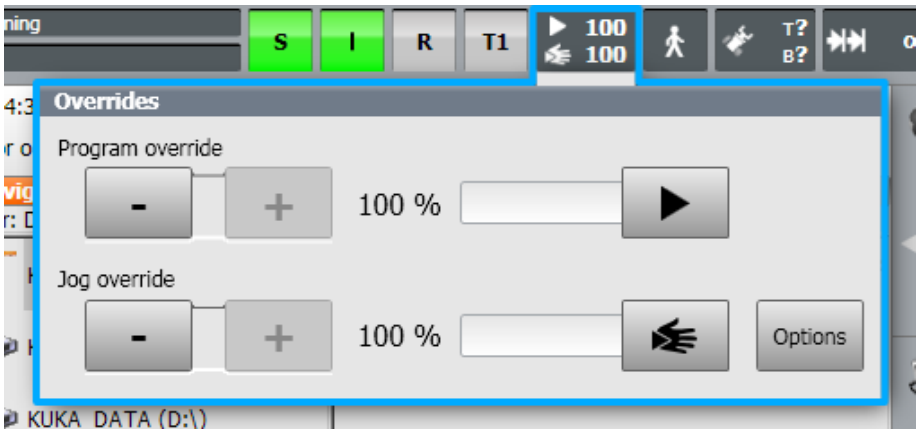


Fig. 4-6: POV setting

3. Press enabling switch.

Fig. 4-7: Enabling switches

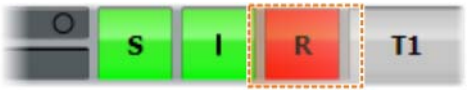
4. Press and hold down the Start (+) key.
   - The “INI” line is executed.
   - The robot performs the BCO run.

Fig. 4-8: Program execution directions: forwards/backwards

Item	Description
1	Start key **forwards**
2	Start key **backwards**

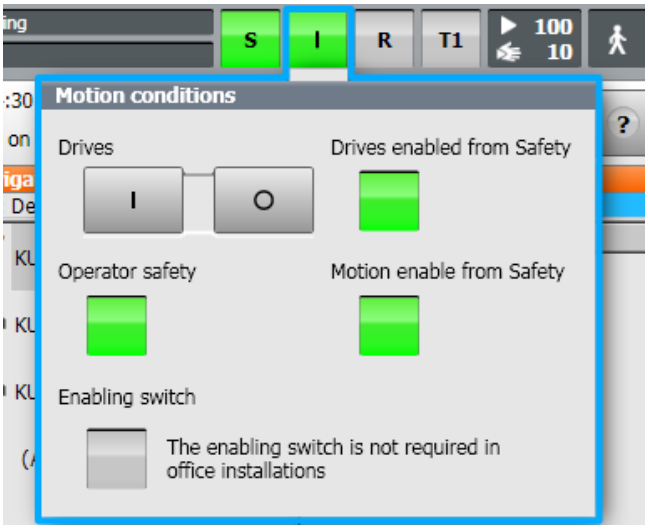
⚠ WARNING A BCO run is executed as a PTP motion from the actual position to the target position if the selected motion block contains the motion command PTP. If the selected motion block contains LIN or CIRC, the BCO run is executed as a LIN motion. The motion must be observed in order to avoid collisions. The velocity is automatically reduced during the BCO run.

5. Once the end position has been reached, the motion is stopped.



The notification message "*Programmed path reached (BCO)*" is displayed.

6. Continued sequence (depending on what operating mode is set):
   - **T1 and T2**: Continue the program by pressing the Start key.
   - **AUT**: Activate drives.



Then start the program by pressing *Start*.

- In the Cell program, switch the operating mode to **EXT** and transfer the motion command from the PLC.

4.4 Exercise: Executing robot programs

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Select and deselect programs
- Run, stop and reset programs in the required operating modes (test program execution)
- Perform and understand block selection
- Carry out a BCO run.

Preconditions

The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of how to use the Navigator
- Knowledge of selecting and canceling programs

Task description

1. Select the module “Air”.

> **⚠ WARNING** **Danger!**
> The safety regulations contained in the safety instruction must be observed!

2. Test the program in the different operating modes as follows:
   - T1 with 100%
   - T2 with 10%, 30%, 50%, 75%, 100%
   - Automatic with 100%
3. Test the program in the program run modes Go and Motion.

5 Working with program files

5.1 Overview

The following contents are explained in this training module:

- Creating and editing program modules
- Archiving and restoring robot programs
- Working with the logbook

5.2 Creating program modules

Program modules in the Navigator

Program modules should always be stored in the folder “Program”. It is also possible to create new folders and save program modules there. Modules are identified by the icon with the letter “M”. A module can be provided with a comment. A comment may contain a brief description of the functions of the program, for example.

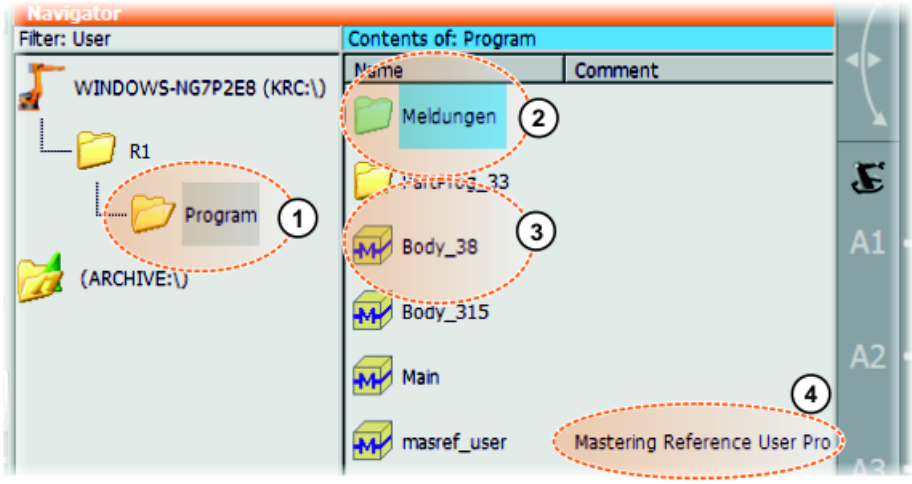


Fig. 5-1: Modules in Navigator

1 Main folder for programs: “Program”
2 Subfolder for additional programs
3 Program module/module
4 Comment of a program module

Properties of program modules

A module always consists of two parts:

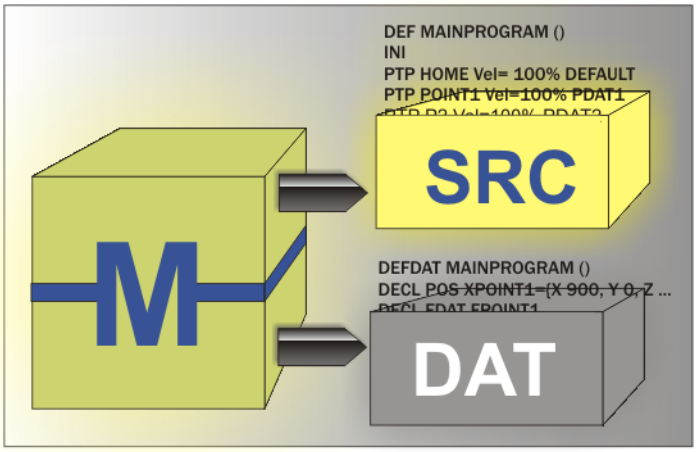


Fig. 5-2: Module structure

- **Source code:** The SRC file contains the program code.

```
DEF MAINPROGRAM()
INI
SPTP HOME Vel= 100% DEFAULT
SPTP P1 Vel=100% PDAT1 TOOL[1] BASE[2]
SPTP P2 Vel=100% PDAT2 TOOL[1] BASE[2]
…
END
```

- **Data list:** The DAT file contains permanent data and point coordinates.

```
DEFDAT MAINPROGRAM()
DECL E6POS XP1={X 900, Y 0, Z 800, A 0, B 0, C 0, S 6, T 27, E1 0, E2
0, E3 0, E4 0, E5 0, E6 0}
DECL FDAT FPOINT1 …
…
ENDDAT
```

Procedure for creating program modules

1. In the directory structure, select the folder in which the program is to be created, e.g. the folder **Program**.
2. Press the **New** softkey.
3. Enter a name for the program, and a comment if desired, and confirm it with **OK**.

5.3 Editing program modules

Editing options

Just like in other commonly-used file systems, program modules can also be edited in the Navigator of the KUKA smartPad.

Editing tasks include:

- Duplicate/Copy
- Delete
- Rename

Procedure for duplicating a program

1. In the directory structure, select the folder in which the file is located.
2. Select the file in the file list.
3. Press the **Duplicate** softkey.
4. Give the new module a new file name and confirm it with **OK**.

In the user group “Expert” with the filter setting “Detail”, two files are displayed in the Navigator for each module (SRC and DAT file). If this is the case, both files must be duplicated!

Procedure for deleting a program

1. In the directory structure, select the folder in which the file is located.
2. Select the file in the file list.
3. Press the **Delete** softkey.
4. Confirm the request for confirmation with **Yes**. The module is deleted.

In the user group “Expert” with the filter setting “Detail”, two files are displayed in the Navigator for each module (SRC and DAT file). If this is the case, both files must be deleted! Deleted files cannot be restored!

Procedure for renaming a program

1. In the directory structure, select the folder in which the file is located.
2. Select the file in the file list.
3. Select the softkey **Edit** > **Rename**.
4. Overwrite the file name with the new name and confirm with **OK**.

In the user group “Expert” with the filter setting “Detail”, two files are displayed in the Navigator for each module (SRC and DAT file). If this is the case, both files must be renamed!

5.4 Archiving and restoring robot programs

Archiving options

Every archiving operation generates a ZIP file on the corresponding target medium with the same name as the robot. The name of the individual file can be modified under **Robot data**.

File paths: There are three different file paths available:

- **USB (KCP)** | USB stick on KCP (smartPAD)
- **USB (cabinet)** | USB stick on robot control cabinet
- **Network** | Archiving to a network path

  The desired network path must be configured under **Robot data**.

Parallel to the ZIP file generated on the selected storage medium during every archiving operation, an additional archive file (INTERN.ZIP) is stored on drive D:\.

Data: The following selections can be made for archiving data:

- **All:**

  The data that are required to restore an existing system are archived.
- **Applications:**

  All user-defined KRL modules (programs) and their corresponding system files are archived.
- **System data:**

  The machine data are archived.
- **Log data:**

  The log files are archived.
- **KrcDiag:**

  Archiving of data for fault analysis by KUKA Roboter GmbH. A folder is generated here (name **KRCDiag**) in which up to ten ZIP files can be written. Parallel to this, archiving is carried out on the controller under C:\KUKA\KRCDiag.

Restoring data

⚠ WARNING Generally, only archives with the right software version may be loaded. If other archives are loaded, the following may occur:

- Error messages
- Robot controller is not operable.
- Personal injury and damage to property.

The following menu items are available for restoring data:

- **All**
- **Applications**
- **System data**

The system generates an error message in the following cases:

- If the archived data have a different version from those in the system.
- If the version of the technology packages does not match the installed version.

Procedure for archiving

NOTICE Only the KUKA.USB data stick may be used. Data may be lost or modified if any other USB stick is used.

1. Select the menu sequence **File** > **Archive** > **USB (KCP)** or **USB (cabinet)** and the desired menu item.
2. Confirm the request for confirmation with **Yes**.
   The message window indicates completion of the archiving process.
3. The stick can be removed when the LED on the stick is no longer lit.

Procedure for restoration

1. Select the menu sequence **File** > **Restore** and then the desired subitems.
2. Confirm the request for confirmation with **Yes**. Archived files are restored to the robot controller. A message indicates completion of the restoration process.
3. If data have been restored from a USB stick: remove the USB storage medium.

NOTICE In the case of restoring data from a USB medium: the medium must not be removed until the LED on the USB medium is no longer lit. Otherwise, the medium could be damaged.

4. Reboot the robot controller. A cold start is required for this.
   - User profile “Expert”
     Menu path: *KUKA key > Configuration > User group > Expert*
   - Initiate cold start via:
     Menu path: *KUKA key > Shutdown > Cold start*

5.5 Tracking program modifications and changes of state by means of the logbook

Logging options

The operator actions on the smartPAD are automatically logged. The command **Logbook** displays the logbook.

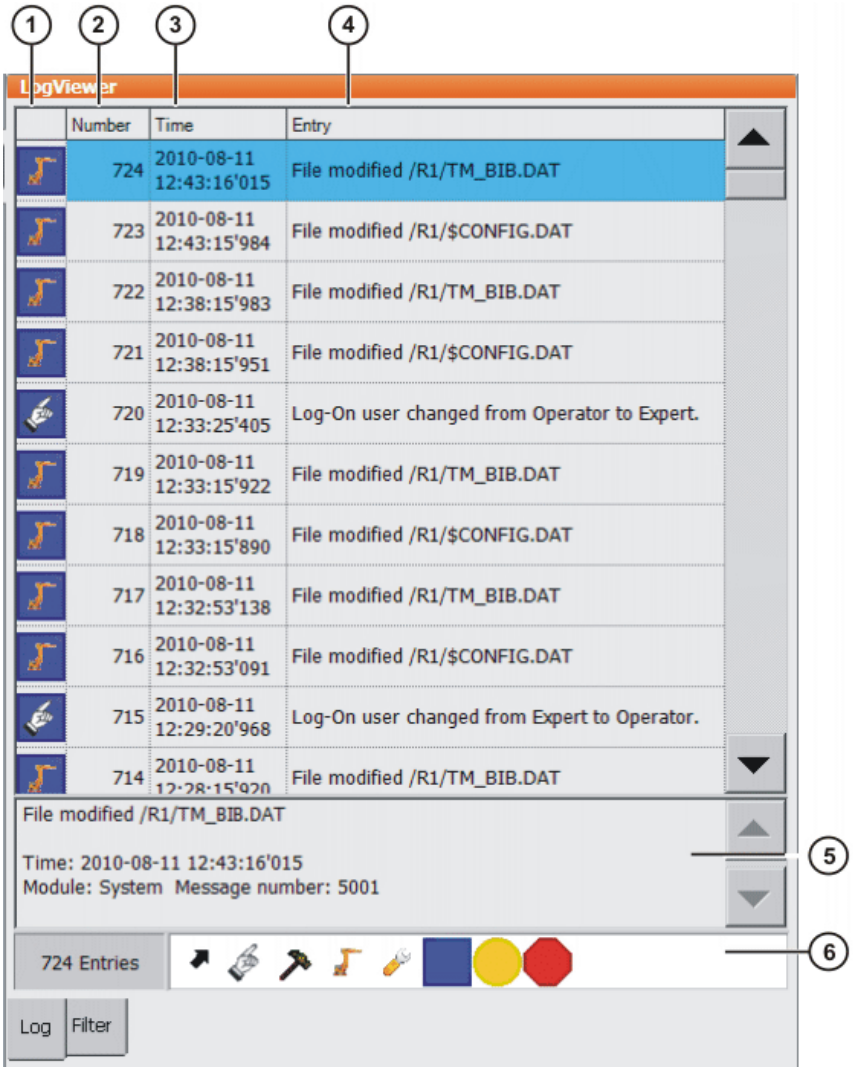


Fig. 5-3: Logbook, Log tab

Item	Description
1	Type of log event The individual filter types and filter classes are listed on the **Filter** tab.
2	Log event number
3	Date and time of the log event
4	Brief description of the log event
5	Detailed description of the selected log event
6	Indication of the active filter

Filtering log events

Fig. 5-4: Logbook, Filter tab

Using the logbook function

Viewing and configuration can be carried out in any user group.

Displaying the logbook:

- In the main menu, select **Diagnosis** > **Logbook** > **Display**.

Configuring the logbook:

1. In the main menu, select **Diagnosis** > **Logbook** > **Configuration**.
2. Press **OK** to save the configuration and close the window.

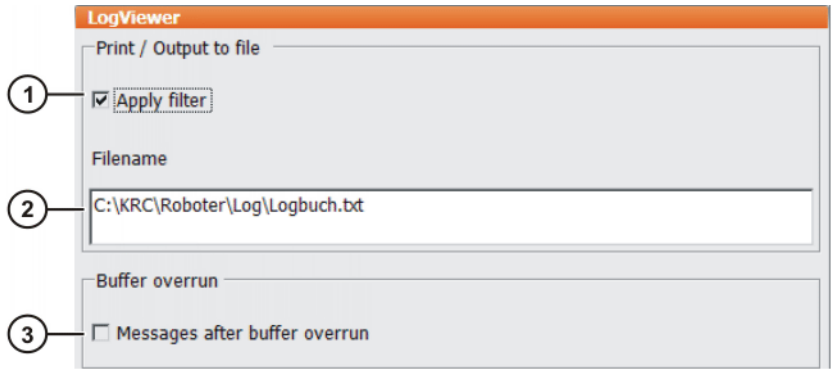


Fig. 5-5: Logbook configuration window

1	Apply filter settings for the display. If the check box is not activated, the display is unfiltered.

2	Path for the text file.
3	Log data deleted because of a buffer overflow are indicated in gray in the text file.

6 Creating and modifying programmed motions

6.1 Overview

The following contents are explained in this training module:

- Creating new motion commands
- Creating cycle-time optimized motions
- Creating CP motions
- Approximation of motions
- Modifying motion commands

6.2 Correction of existing motion points

Programming robot motions

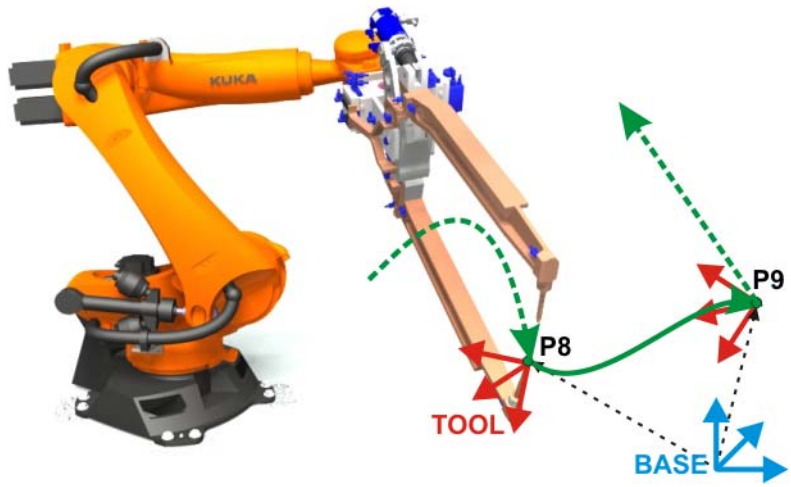


Fig. 6-1: Robot motion

When robot motions have to be programmed, many questions are raised:

Question	Solution	Keyword
How does the robot remember its positions?	The positions of the tool in space are saved (robot position in accordance with the tool and base that are set).	POS E6POS
How does the robot know how to move?	From the specification of the motion type: point-to-point, linear or circular.	SPTP SLIN SCIRC
How fast does the robot move?	The velocity between two points and the acceleration are specified during programming.	Vel. Acc.
Does the robot have to stop at every point?	To save cycle time, points can also be approximated; no exact positioning is carried out in this case.	CONT

Question	Solution	Keyword
What orientation does the tool adopt when a point is reached?	The orientation control can be set individually for each motion. This setting is valid for CP motions only. (>>> "Motion types" Page 116)	ORI_TYPE
Does the robot recognize obstacles?	No, the robot “stubbornly” follows its programmed path. The programmer is responsible for ensuring that there is no risk of collisions. There is also a collision monitoring function, however, for protecting the machine.	Collision monitoring

This information must be transferred when programming robot motions using the teaching method. Inline forms, into which the information can easily be entered, are used for this.

Fig. 6-2: Inline form for motion programming

Motion types

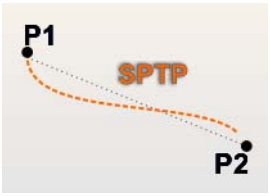
Various motion types are available for programming motion commands. Motions can be programmed in accordance with the specific requirements of the robot’s work process.

- Axis-specific motions (SPTP: point-to-point)
- CP motions: SLIN (linear) and SCIRC (circular)

PTP, LIN and CIRC motions are not covered by this training documentation. More detailed information can be found in the *Operating and Programming Instructions for KUKA System Software 8.x*.

6.3 Creating cycle-time optimized motion (axis motion)

Motion type SPTP

Motion type	Meaning	Application example
	*Point-to-point*: ■ Axis-specific motion: The robot guides the TCP along the **fastest** path to the end point. The fastest path is generally not the shortest path and is thus **not** a straight line. As the motions of the robot axes are rotational, curved paths can be executed faster than straight paths. ■ The exact path of the motion cannot be predicted. ■ The leading axis is the axis that takes longest to reach the end point. ■ SYNCHRO PTP: All axes start together and also stop in a synchronized manner. ■ The first motion in the program must be a PTP motion, as Status and Turn are evaluated only here.	Point applications, e.g.: ■ Spot welding ■ Transfer ■ Measuring, inspection Auxiliary positions: ■ Intermediate points ■ Free points in space

Synchro PTP

The leading axis is the axis that takes longest to reach the end point. The velocity specified in the inline form is taken into consideration.

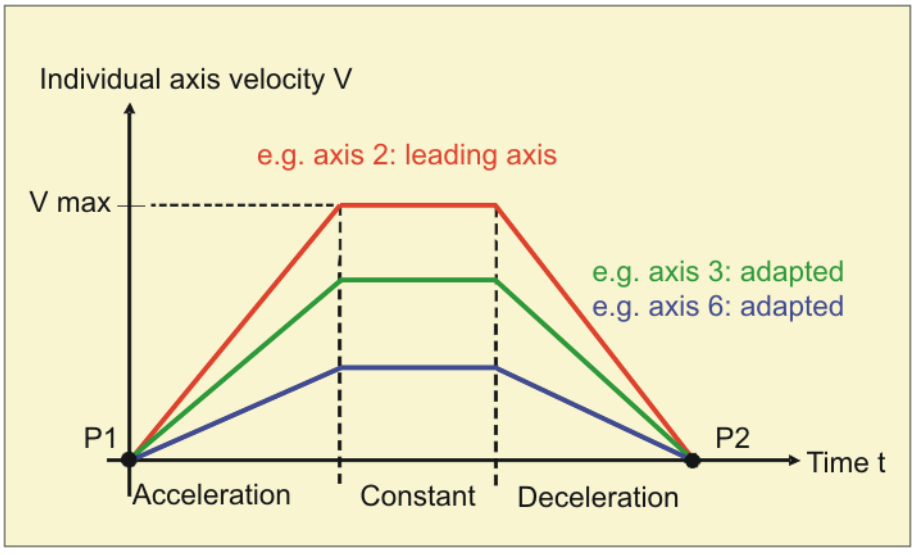


Fig. 6-3: Synchro PTP

Status & Turn

Status and Turn serve to define an unambiguous axis position in such cases where the same TCP position can be achieved with different axis positions.

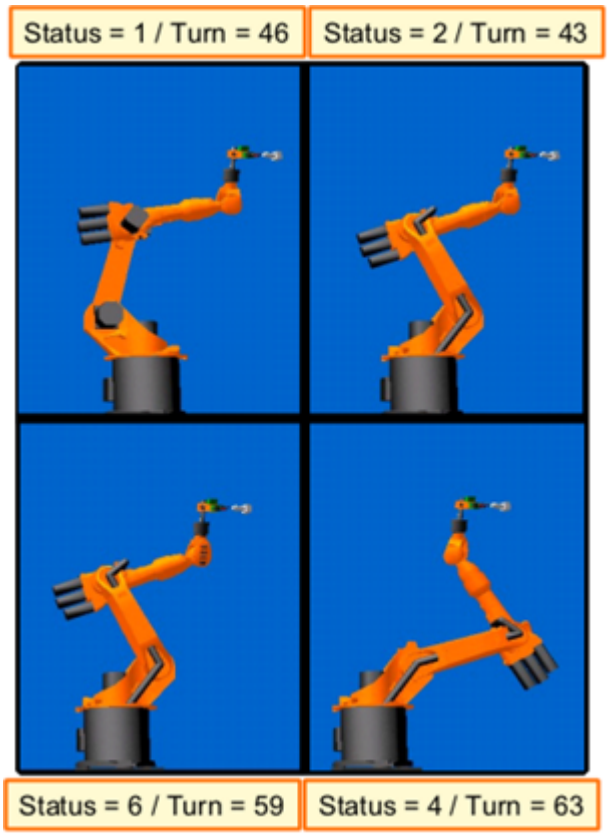


Fig. 6-4: Different axis positions due to different Status and Turn values

The robot controller only takes the programmed Status and Turn values into consideration for SPTP motions. They are ignored for path (*CP*) motions. The first motion instruction in a KRL program must therefore be a complete SPTP instruction of type POS or E6POS in order to define an unambiguous starting position (or a complete SPTP instruction of type AXIS or E6AXIS).

```
DEFDAT MAINPROGRAM ()
DECL POS XPOINT1={X 900, Y 0, Z 800, A 0, B 0, C 0, S 6, T 27}
DECL FDAT FPOINT1 …
…
ENDDAT
```

Approximate positioning

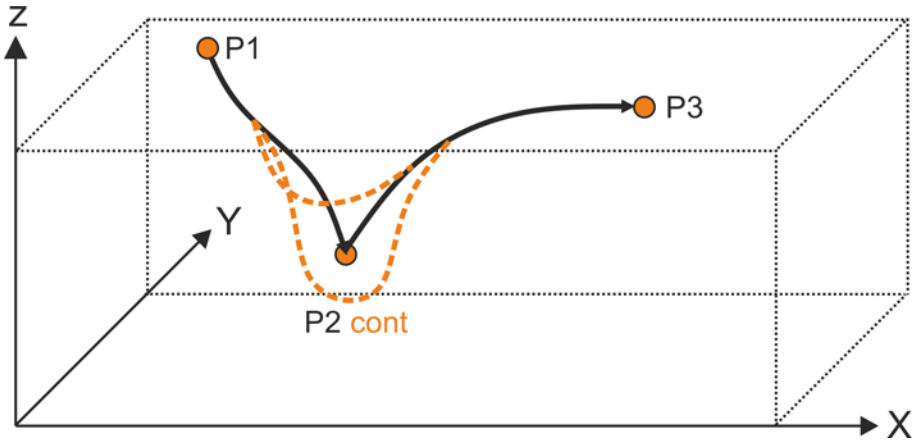


Fig. 6-5: Approximation of an SPTP point

In order to accelerate the motion sequence, the controller is able to approximate motion commands labeled with CONT. Approximate positioning means that the point coordinates are not addressed exactly. The robot leaves the path of the exact positioning contour before it reaches them. The TCP is guided

along an approximate positioning contour that leads into the exact positioning contour of the next motion command.

Advantages of approximate positioning:

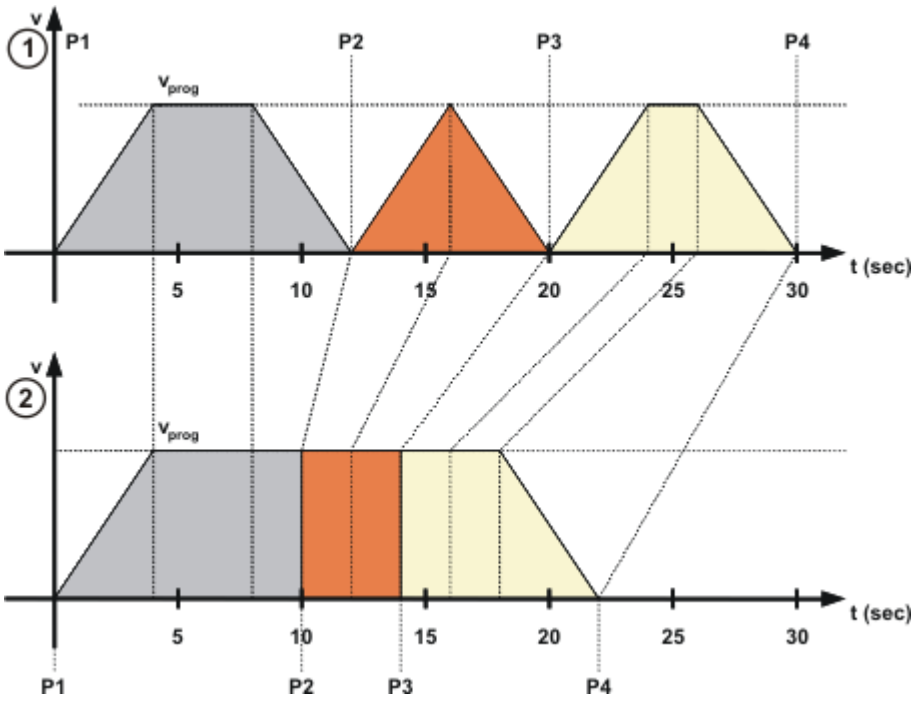
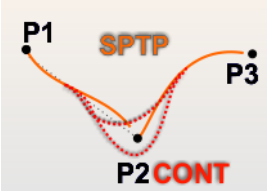


Fig. 6-6: Comparison of exact positioning and approximate positioning

- There is less wear to the kinematic system, as it is no longer necessary to brake and accelerate between the points (see point 1).
- The cycle time is optimized and the program is executed more quickly (see point 2).

In order to be able to perform an approximate positioning motion, the controller must be able to load the following motion commands. This is carried out by the computer advance run.

Approximate positioning in the SPTP motion type

Motion type	Feature	Approximation distance
	■ The approximate positioning contour cannot be predict-ed!	Specified in % or mm

Procedure for creating SPTP motions

Preconditions

- T1 mode is set.
- A robot program is selected.

1. Move the TCP to the position that is to be taught as the end point.

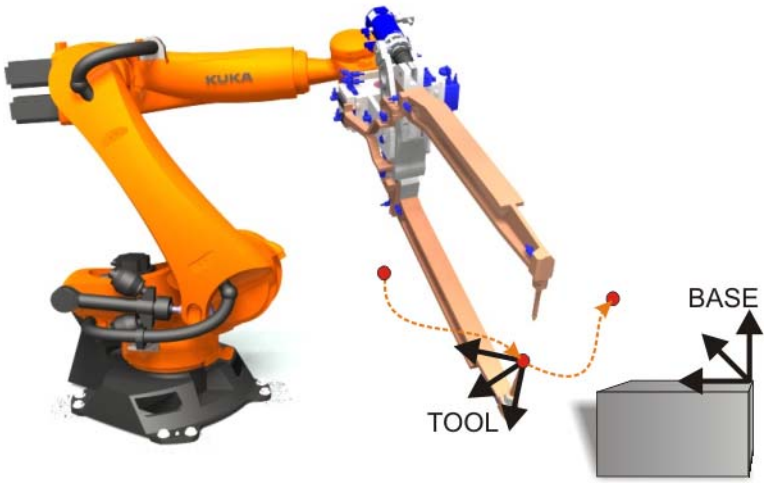


Fig. 6-7: Motion command

2. Position the cursor in the line after which the motion instruction is to be inserted.
3. Menu sequence **Commands** > **Motion** > **SPTP**.
   Alternatively, the softkey **Motion** can be pressed in the corresponding line.

   An inline form appears:

   - **SPTP inline form**

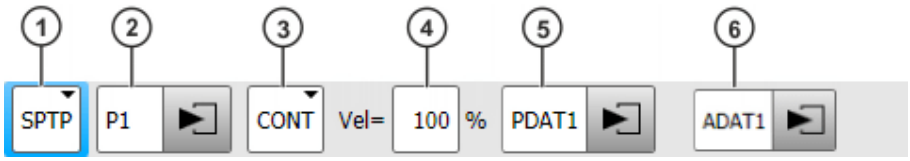


Fig. 6-8: Inline form “SPTP” (individual motion)

4. Enter parameters in the inline form.

Item	Description
1	Motion type **SPTP**
2	Point name for end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened.
3	▪ **CONT**: end point is approximated. ▪ **[Empty box]**: the motion stops exactly at the end point.
4	Velocity ▪ **1 … 100% for SPTP** ▪ **0.001 … 2 m/s for SLIN**
5	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened.
6	This box can be displayed or hidden by means of **Switch parameter**. Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened.

5. Enter the correct data for the tool and base coordinate system in the option window “Frames”, together with details of the interpolation mode (external TCP: on/off) and the collision monitoring function.

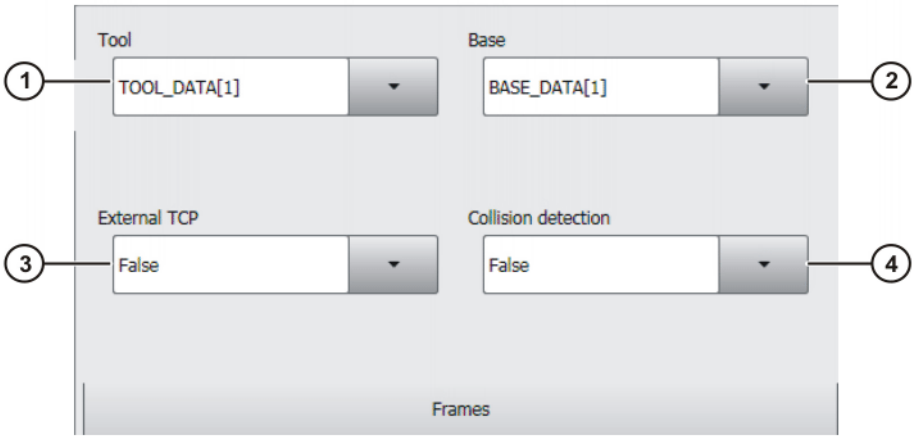


Fig. 6-9: Option window: Frames

Item	Description
1	Tool selection. If **True** in the box **External TCP**: workpiece selection. Range of values: [1] … [16]
2	Base selection. If **True** in the box **External TCP**: fixed tool selection. Range of values: [1] … [32]
3	Interpolation mode External TCP: ■ **False**: The tool is mounted on the mounting flange. ■ **True**: The tool is a fixed tool.
4	Collision detection ■ **True**: For this motion, the robot controller calculates the axis torques. These are required for collision detection. ■ **False**: For this motion, the robot controller does not calculate the axis torques. Collision detection is thus not possible for this motion.

6. The acceleration can be reduced from the maximum value in the option window “Motion parameters”. If approximate positioning has been activated, the approximation distance can also be modified. Depending on the configuration, the distance is set in **mm** or **%**.

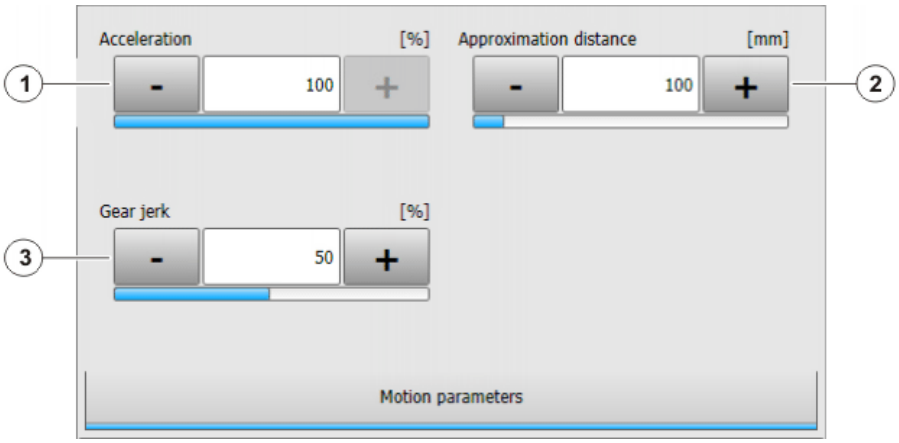


Fig. 6-10: Option window “Motion parameters” (SPTP)

Item	Description
1	Axis acceleration. The value refers to the maximum value specified in the machine data. ▪ **1 … 100 %**
2	This box is not available for SPTP segments. In the case of individual SPTP motions, this box is only displayed if **CONT** was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum permissible value is half the distance between the start point and the end point. If a higher value is entered, this is ignored and the maximum value is used.
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. ▪ **1 … 100 %**

7. Save instruction with **Cmd OK**. The current position of the TCP is taught as the end point.

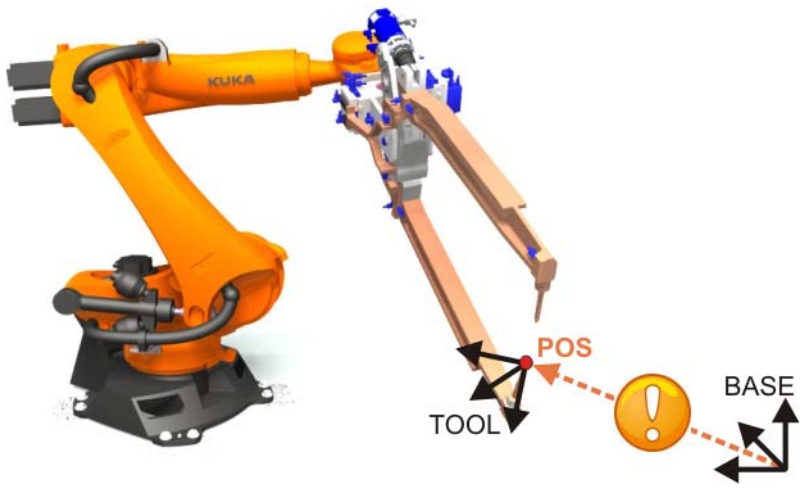


Fig. 6-11: Saving the point coordinates with “Cmd OK” and “Touchup”

6.4 Exercise: Dummy program – program handling and SPTP motions

Aim of the exercise

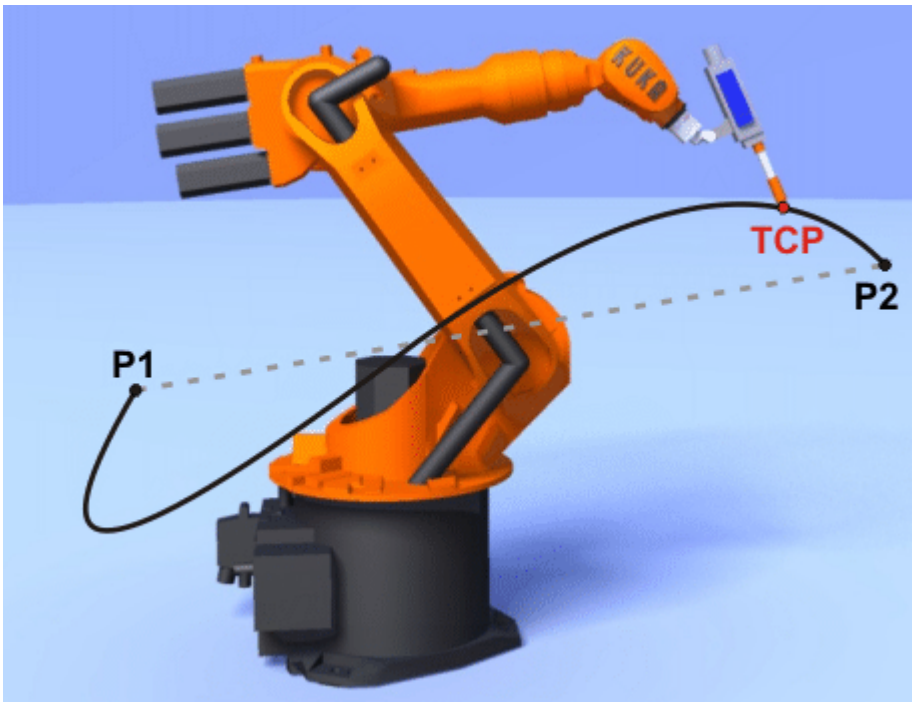
On successful completion of this exercise, you will be able to carry out the following activities:

- Select and deselect programs
- Run, stop and reset programs in the required operating modes (test program execution)
- Correct existing program points
- Delete motion blocks and insert new SPTP motions
- Change the program run mode and carry out step-by-step movement to programmed points
- Perform and understand block selection
- Carry out a BCO run.

Preconditions

The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of how to use the Navigator
- Theoretical knowledge of the SPTP motion type



Task, part A

Carry out the following tasks: Create and test programs.

1. Create a new module and give it a meaningful name.

> **⚠ WARNING** **Danger!**
> The safety regulations contained in the safety instruction must be observed!

2. Create a motion sequence of approx. five SPTP blocks.
3. If the motion is not collision-free, delete the relevant point(s) and create a new one in each case.
4. Test the program in T1 mode at different program velocities (POV).
5. Test the program in T2 mode at different program velocities (POV).
6. Test the program in Automatic mode.

Task, part B

Carry out the following tasks: program correction

1. Set various velocities for your space points

2. Call the same point several times in the program
3. Delete the motion blocks and insert new ones at a different point in the program
4. Carry out a block selection
5. Stop your program during testing and use the function **Program start backwards**.
6. Test your program in the modes T1, T2 and Automatic.

What you should know after the exercise:

1. What is the difference between selecting a program and opening it?

. . . . . .

. . . . . .

. . . . . .

2. What program run modes are there and what are they used for?

. . . . . .

. . . . . .

. . . . . .

3. What is a BCO run?

. . . . . .

. . . . . .

. . . . . .

4. How can you influence the program velocity?

. . . . . .

. . . . . .

. . . . . .

5. What are the characteristics of an SPTP motion?

. . . . . .

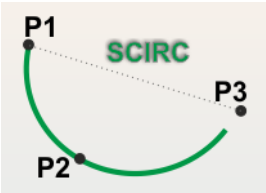
. . . . . .

. . . . . .

6.5 Creating CP motions

Motion types

SLIN and SCIRC

Motion type	Meaning	Application example
	*Linear*: ■ Motion in a straight line: ■ The TCP of the tool is guided from the start point to the end point with constant velocity and a defined orientation. ■ The velocity and orientation refer to the TCP.	CP applications, e.g.: ■ Arc welding ■ Adhesive bonding ■ Laser welding / cutting
	*Circular*: ■ Circular path motion is defined by a start point, auxiliary point and end point. ■ The TCP of the tool is guided from the start point to the end point with constant velocity and a defined orientation. ■ The velocity and orientation refer to the TCP of the tool (tool coordinate system).	Path applications, similar to SLIN: ■ Circles, radii, curves

Singularity positions

KUKA robots with 6 degrees of freedom have 3 different singularity positions.

A singularity position is characterized by the fact that unambiguous reverse transformation (conversion of Cartesian coordinates to axis-specific values) is not possible, even though Status and Turn are specified. In this case, or if very slight Cartesian changes cause very large changes to the axis angles, one speaks of singularity positions. This is a mathematical property, not a mechanical one, and thus only exists for CP motions and not axis motions.

Overhead singularity α1

In the overhead singularity, the wrist root point (= center point of axis A5) is located vertically above axis A1 of the robot.

The position of axis A1 cannot be determined unambiguously by means of reverse transformation and can thus take any value.

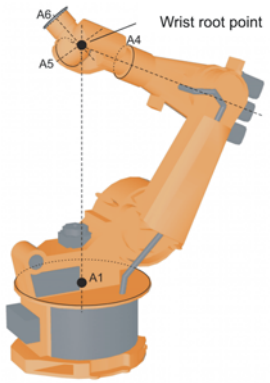


Fig. 6-12: Overhead singularity (α1 position)

Extended position singularity α2

In the extended position, the wrist root point (= center point of axis A5) is located in the extension of axes A2 and A3 of the robot.

The robot is at the limit of its work envelope.

Although reverse transformation does provide unambiguous axis angles, low Cartesian velocities result in high axis velocities for axes A2 and A3.

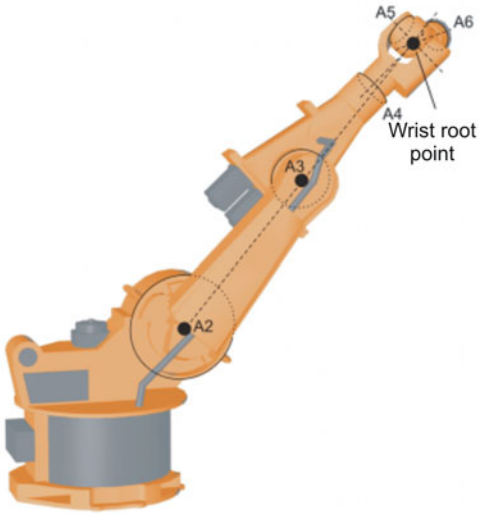


Fig. 6-13: Extended position (α2 position)

Wrist axis singularity α5

In the wrist axis singularity position, the axes A4 and A6 are parallel to one another and axis A5 is within the range ±0.01812°.

The position of the two axes cannot be determined unambiguously by reverse transformation. There is an infinite number of possible axis positions for axes A4 and A6 with identical axis angle sums.

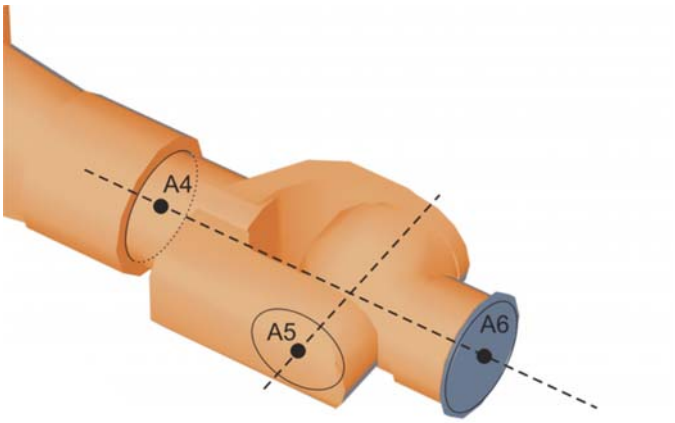


Fig. 6-14: Wrist axis singularity (α5 position)

Orientation control with CP motions

In the case of CP motions, it is possible to define the orientation control precisely. The orientation of a tool can be different at the start point and end point of a motion.

Orientation control with the motion type **SLIN**

- **Standard** or **Wrist PTP**

  The orientation of the tool changes continuously during the motion.

  Use Wrist PTP if, with Standard, the robot passes through a wrist axis singularity, as the orientation is carried out by means of linear transformation (axis-specific jogging) of the wrist axis angles.

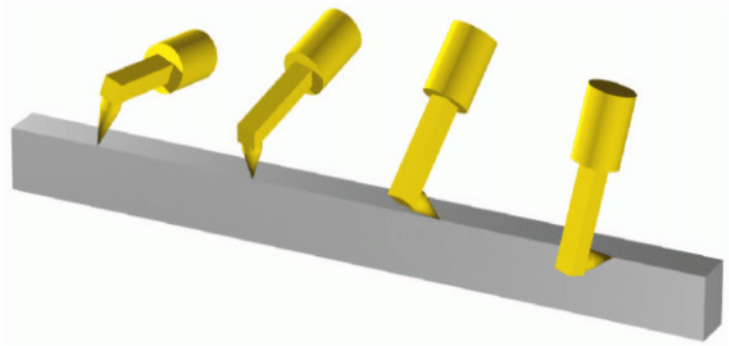


Fig. 6-15: Standard

- **Constant**

  The orientation of the tool remains constant during the motion, i.e. as taught at the start point. The orientation taught at the end point is disregarded.

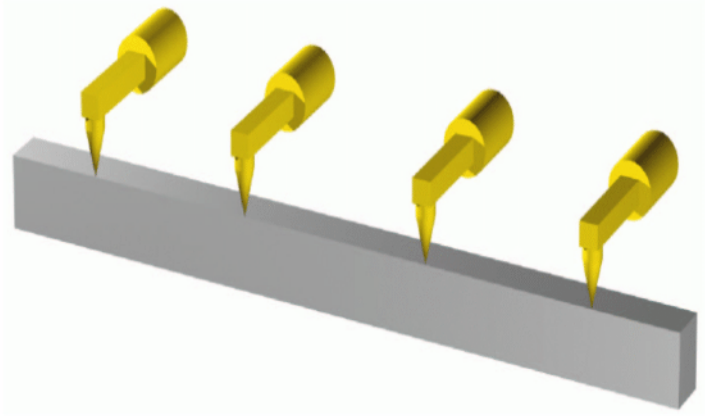


Fig. 6-16: Constant orientation

Orientation control with the motion type **SCIRC**

- **Standard** or **Wrist PTP**

  The orientation of the tool changes continuously during the motion.

  Use Wrist PTP if, with Standard, the robot passes through a wrist axis singularity, as the orientation is carried out by means of linear transformation (axis-specific jogging) of the wrist axis angles.

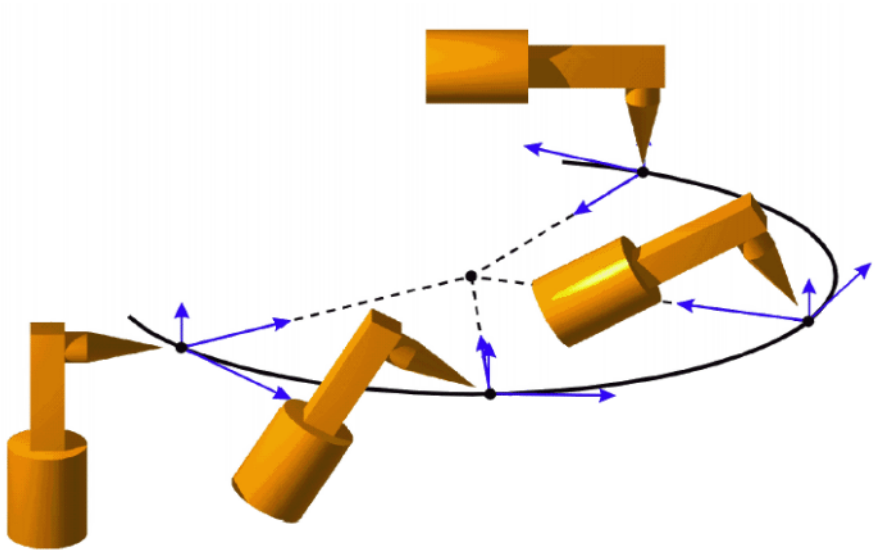


Fig. 6-17: Standard + base-related

- **Constant**

  The orientation of the tool remains constant during the motion, i.e. as taught at the start point. The orientation taught at the end point is disregarded.

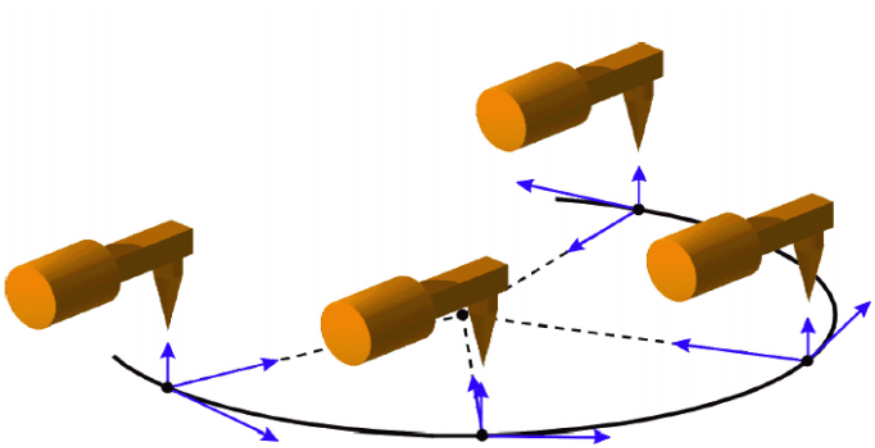


Fig. 6-18: Constant orientation control + base-related

- **Constant, path-related**

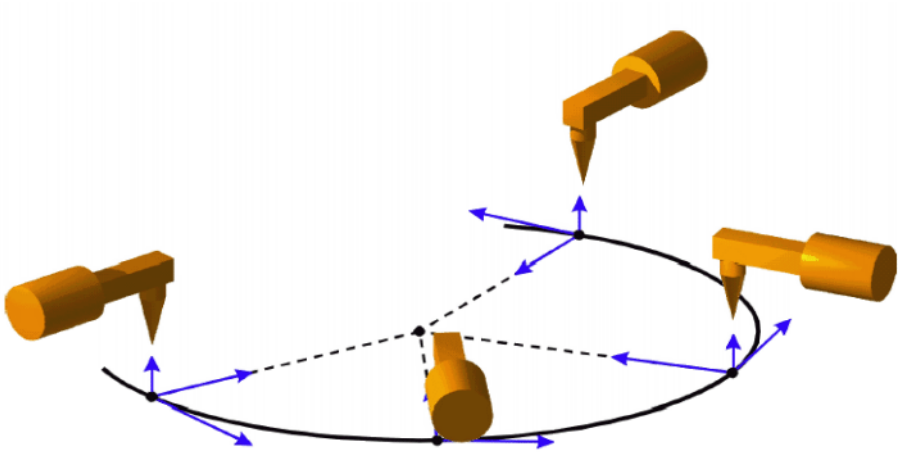


Fig. 6-19: Constant orientation, path-related

Motion sequence for SCIRC

Here, the TCP or workpiece reference point moves to the end point along an arc. The path is defined using start, auxiliary and end points. The end point of a motion instruction serves as the start point for the subsequent motion.

The orientation of the tool at the auxiliary point can be ignored or applied.

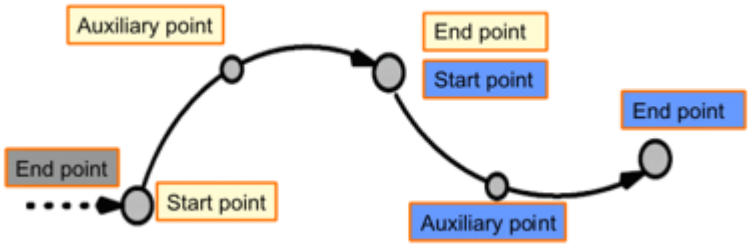
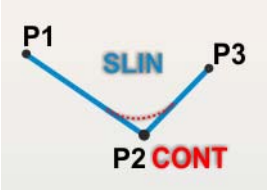
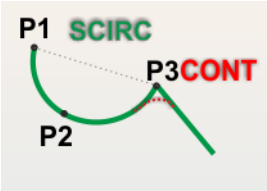


Fig. 6-20: Two circle segments with SCIRC

Approximation of CP motions

The approximate positioning function is not suitable for generating circular motions. It is purely for preventing an exact stop at the point.

Approximate positioning in the motion types PTP, LIN and CIRC

Motion type	Feature	Approximation distance
	■ Path corresponds to two parabolic branches	Specified in mm
	■ Path corresponds to two parabolic branches ■ The end point is approximated.	Specified in mm

Procedure for creating SLIN motions

Preconditions

- T1 mode is set.
- A robot program is selected.

1. Move the TCP to the position that is to be taught as the end point.

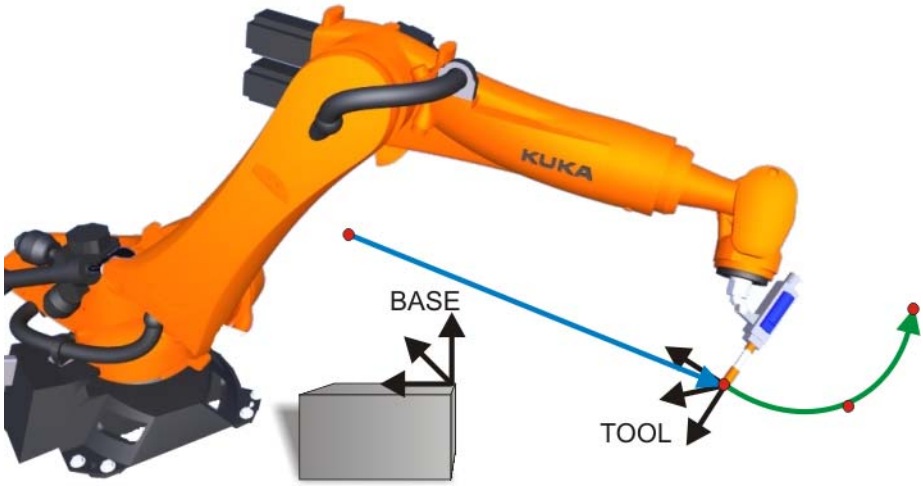


Fig. 6-21: Motion command with SLIN and SCIRC

2. Position the cursor in the line after which the motion instruction is to be inserted.
3. Select the menu sequence **Commands** > **Motion** > **SLIN**.
   Alternatively, the softkey **Motion** can be pressed in the corresponding line.
   An inline form appears:
   - **SLIN inline form**

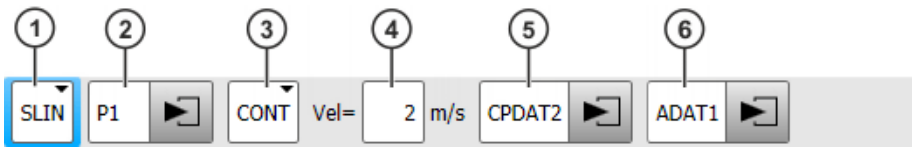


Fig. 6-22: Inline form “SLIN” (individual motion)

Item	Description
1	Motion type **SLIN**
2	Point name for end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened.
3	▪ **CONT**: end point is approximated. ▪ **[Empty box]**: the motion stops exactly at the end point.
4	Velocity ▪ **0.001 … 2 m/s for SLIN**
5	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened.
6	This box can be displayed or hidden by means of **Switch parameter**. Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened.

4. Enter parameters in the inline form.
5. Enter the correct data for the tool and base coordinate system in the option window “Frames”, together with details of the interpolation mode (external TCP: on/off) and the collision monitoring function.

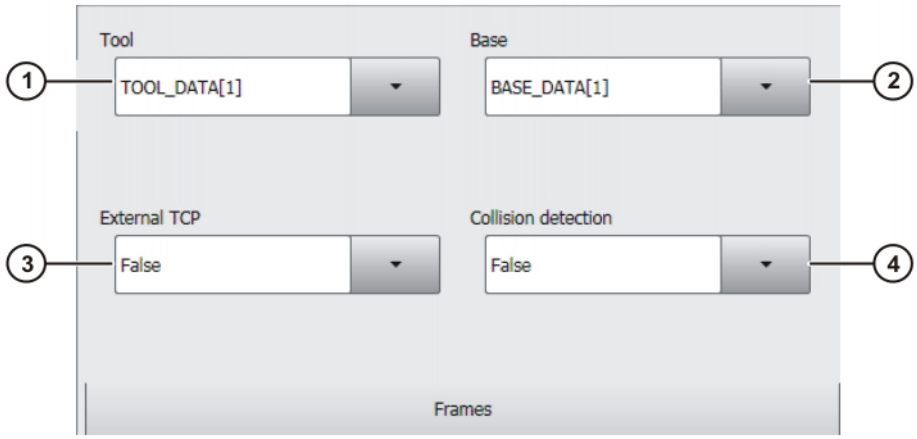


Fig. 6-23: Option window: Frames

Item	Description
1	Tool selection. If **True** in the box **External TCP**: workpiece selection. Range of values: [1] … [16]
2	Base selection. If **True** in the box **External TCP**: fixed tool selection. Range of values: [1] … [32]

Item	Description
3	Interpolation mode ■ **False**: The tool is mounted on the mounting flange. ■ **True**: The tool is a fixed tool.
4	Collision detection ■ **True**: For this motion, the robot controller calculates the axis torques. These are required for collision detection. ■ **False**: For this motion, the robot controller does not calculate the axis torques. Collision detection is thus not possible for this motion.

6. The acceleration and gear jerk can be reduced from the maximum value in the option window “Motion parameters”. If approximate positioning has been activated, the approximation distance can also be modified. Furthermore, the orientation control can also be modified.

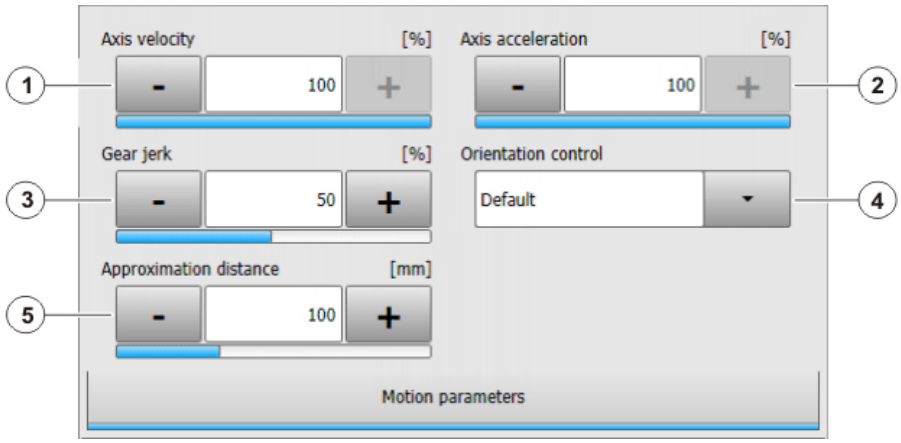


Fig. 6-24: Option window “Motion parameters” (SLIN)

Item	Description
1	Axis velocity. The value refers to the maximum value specified in the machine data. ■ **1 … 100 %**
2	Axis acceleration. The value refers to the maximum value specified in the machine data. ■ **1 … 100 %**
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. ■ **1 … 100 %**
4	Orientation control selection. ■ **Standard** ■ **Wrist PTP** ■ **Constant orientation control**
5	This box is only displayed if **CONT** was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum permissible value is half the distance between the start point and the end point. If a higher value is entered, this is ignored and the maximum value is used.

7. Save instruction with **Cmd OK**. The current position of the TCP is taught as the end point.

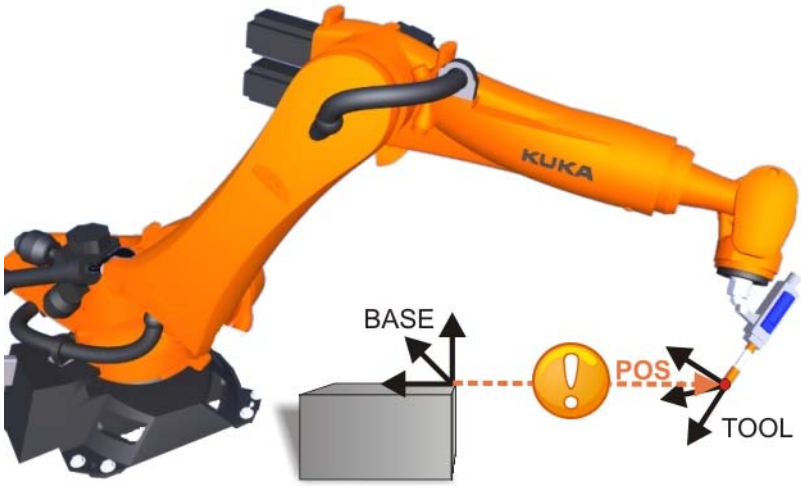


Fig. 6-25: Saving the point coordinates with “Cmd OK” and “Touchup”

Procedure for creating SCIRC motions

Preconditions

- T1 mode is set.
- A robot program is selected.

1. Position the cursor in the line after which the motion instruction is to be inserted.
2. Select the menu sequence **Commands** > **Motion** > **SCIRC**.

   Alternatively, the softkey **Motion** can be pressed in the corresponding line.

   An inline form appears:

   - **SCIRC inline form**

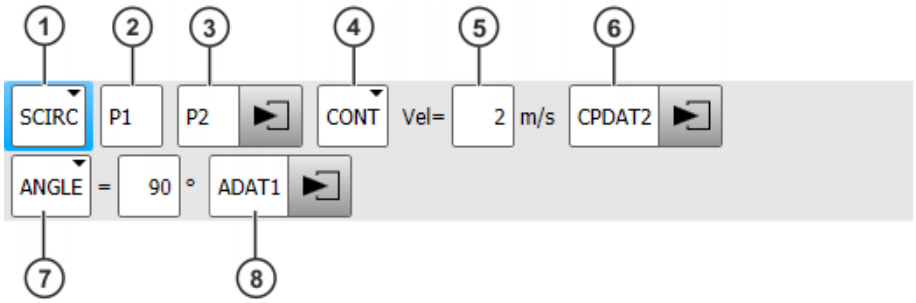


Fig. 6-26: Inline form “SCIRC” (individual motion)

Item	Description
1	Motion type **SCIRC**
2	Point name for the auxiliary point. The system automatically generates a name. The name can be overwritten.
3	Point name for the end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened.
4	▪ **CONT**: end point is approximated. ▪ **[Empty box]**: the motion stops exactly at the end point.

Item	Description
5	Velocity ▪ **0.001 … 2 m/s**
6	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened.
7	Circular angle ▪ **- 9 999° … + 9 999°** If a circular angle less than -400° or greater than +400° is entered, a request for confirmation is generated when the inline form is saved asking whether entry is to be confirmed or rejected.
8	This box can be displayed or hidden by means of **Switch parameter**. Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened.

3. Enter parameters in the inline form.
4. Enter the correct data for the tool and base coordinate system in the option window “Frames”, together with details of the interpolation mode (external TCP: on/off) and the collision monitoring function.

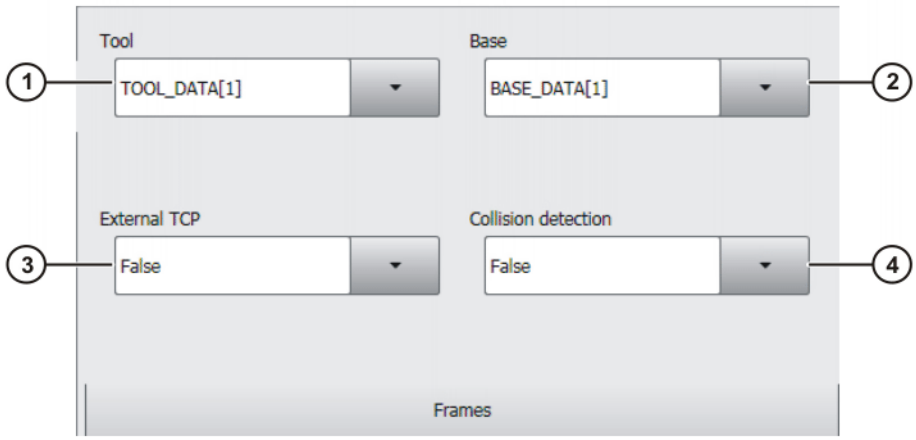


Fig. 6-27: Option window: Frames

Item	Description
1	Tool selection. If **True** in the box **External TCP**: workpiece selection. Range of values: [1] … [16]
2	Base selection. If **True** in the box **External TCP**: fixed tool selection. Range of values: [1] … [32]

Item	Description
3	Interpolation mode ▪ **False**: The tool is mounted on the mounting flange. ▪ **True**: The tool is a fixed tool.
4	Collision detection ▪ **True**: For this motion, the robot controller calculates the axis torques. These are required for collision detection. ▪ **False**: For this motion, the robot controller does not calculate the axis torques. Collision detection is thus not possible for this motion.

5. The acceleration and gear jerk can be reduced from the maximum value in the option window “Motion parameters”. If approximate positioning has been activated, the approximation distance can also be modified. Furthermore, the orientation control can also be modified.

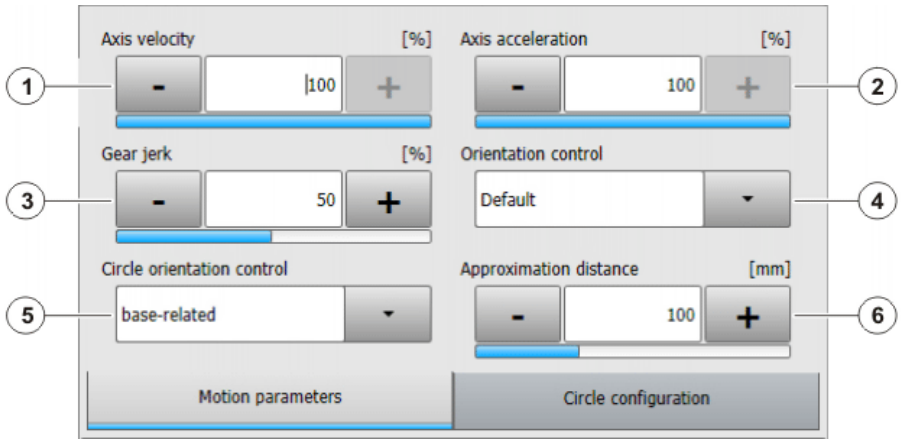


Fig. 6-28: Motion parameters (SCIRC)

Item	Description
1	Axis velocity. The value refers to the maximum value specified in the machine data. ▪ **1 … 100 %**
2	Axis acceleration. The value refers to the maximum value specified in the machine data. ▪ **1 … 100 %**
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. ▪ **1 … 100 %**
4	Orientation control selection ▪ **Standard** ▪ **Wrist PTP** ▪ **Constant orientation control**

Item	Description
5	Orientation control reference system selection. ■ **base-related** ■ **path-oriented**
6	This box is only displayed if **CONT** was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum permissible value is half the distance between the start point and the end point. If a higher value is entered, this is ignored and the maximum value is used.

6. Set the response at the auxiliary point (this is only possible in the user group “Expert” or higher).

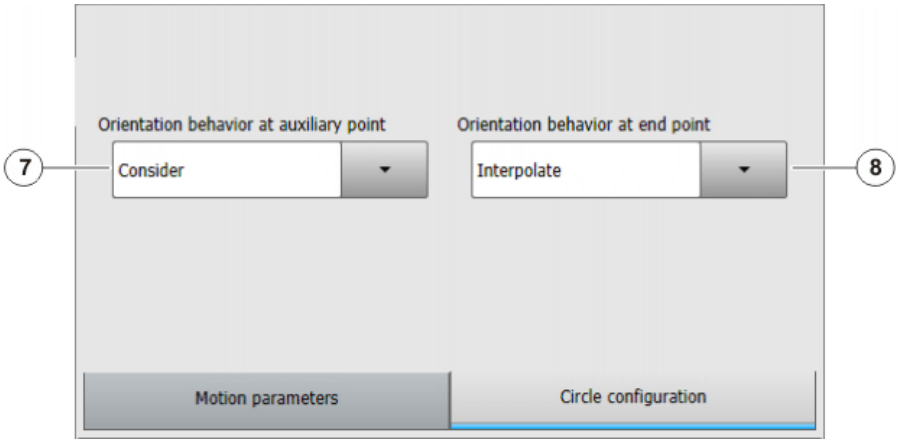


Fig. 6-29: Circle configuration (SCIRC)

Item	Description
7	Selection of orientation behavior at auxiliary point ■ **Consider** ■ **Interpolate** ■ **Ignore**
8	This box is only displayed if **ANGLE** was selected in the inline form. Selection of orientation behavior at end point ■ **Extrapolate** ■ **Interpolate**

7. Move the TCP to the position that is to be taught as the auxiliary point. Save the point data by means of *Teach Aux*.
8. Move the TCP to the position that is to be taught as the end point. Save the point data by means of *Teach End*.
9. Save instruction with **Cmd OK**.

SCIRC circle configuration

During SCIRC motions, the robot controller can take the programmed orientation of the auxiliary point into consideration. The operator can define whether and to what extent it is actually taken into consideration via the **Circle configuration** tab in the **Motion parameters** option window.

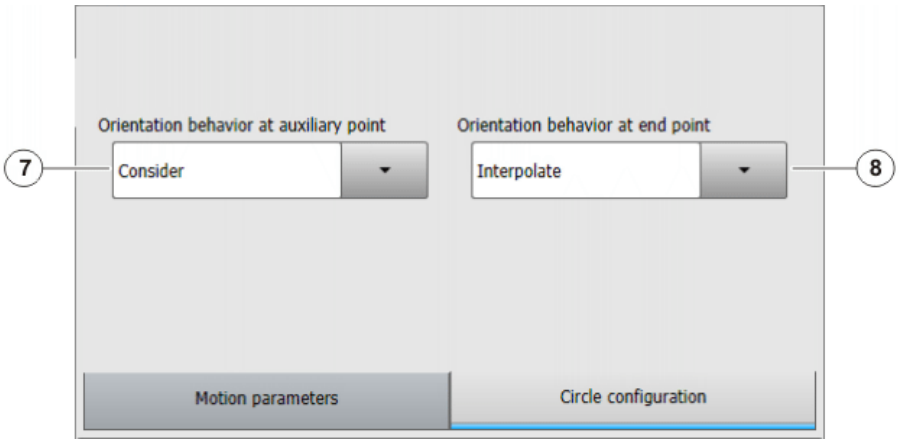


Fig. 6-30: Circle configuration (SCIRC)

In the case of SCIRC statements with circular angles, the same procedure can also be used to define whether the end point is to have the programmed orientation or whether the orientation is to be scaled according to the circular angle.

Element	Description
Orientation behavior at auxiliary point	Pull-down menu ■ **INTERPOLATE**: At the auxiliary point, the TCP adopts the programmed orientation. ■ **IGNORE**: The robot controller ignores the programmed orientation of the auxiliary point. The transition from the start orientation of the TCP to the end orientation is carried out over the shortest possible distance. ■ **CONSIDER** (default): The robot controller selects the path that comes closest to the programmed orientation of the auxiliary point.
Orientation behavior at end point	Pull-down menu ■ **INTERPOLATE**: The programmed orientation of the end point is accepted at the actual end point. (Only possibility for SCIRC without specification of circular angle. If EXTRAPOLATE is set, INTERPOLATE is nonetheless executed.) ■ **EXTRAPOLATE**(default for SCIRC with specification of circular angle): The orientation is adapted to the circular angle: If the circular angle makes the motion longer, the programmed orientation is accepted at the programmed end point. The orientation is continued accordingly to the actual end point. If the circular angle makes the motion shorter, the programmed orientation is not reached.

6.5.1 SCIRC: Orientation behavior – example: auxiliary point

Description

The following orientations have been programmed for the TCP:

- Start point: 0°
- Auxiliary point: 98°
- End point: 197°

The re-orientation is thus 197°. If the auxiliary point is ignored, the end orientation can also be achieved by means of the shorter re-orientation of 360° - 197° = 163°.

- The dotted orange arrows show the programmed orientation.
- The gray arrows indicate schematically what the actual orientation would be where this differs from the programmed orientation.

#INTERPOLATE

At the auxiliary point, the TCP adopts the programmed orientation of 98°. The re-orientation is 197°.

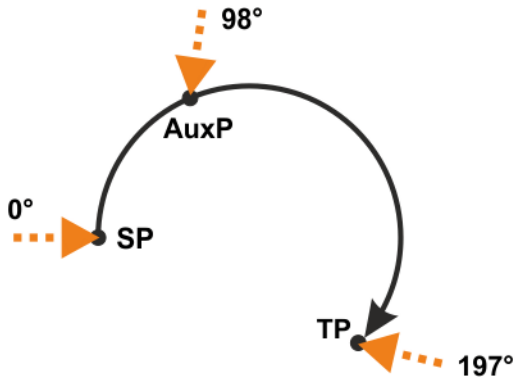


Fig. 6-31: #INTERPOLATE

SP	Start point
AuxP	Auxiliary point
TP	End point

#IGNORE

The short re-orientation through 163° is used. The programmed orientation of the auxiliary point is disregarded.

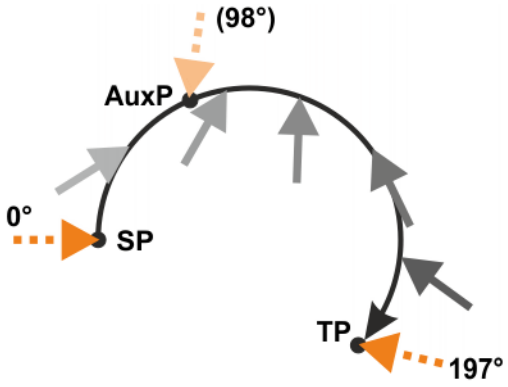


Fig. 6-32: #IGNORE

#CONSIDER

#CONSIDER is suitable if the user wants to specify the re-orientation direction of the TCP without the need for a specific orientation at the auxiliary point. The user can specify the direction using the auxiliary point.

The programmed orientation of the auxiliary point is 98° and is thus on the longer path. The robot controller thus selects the longer path for the re-orientation.

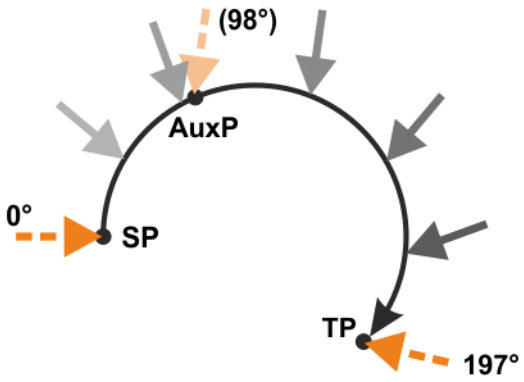


Fig. 6-33: #CONSIDER

Additional example for #CONSIDER:

If the auxiliary point were to be programmed with 262°, it would be on the shorter path. The robot controller would therefore select the shorter path for the re-orientation. The gray arrows indicate that it does not necessarily adopt the programmed orientation of the auxiliary point.

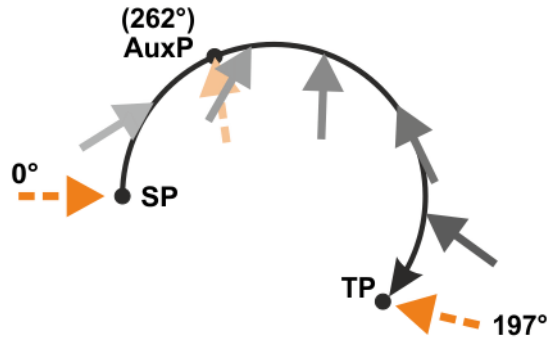


Fig. 6-34: #CONSIDER, additional example

6.5.2 SCIRC: Orientation behavior – example: end point

Description

- The dotted orange arrows show the programmed orientation.
- The gray arrows show the actual orientation where this differs from the programmed orientation.

#INTERPOLATE

At **TP**, which is situated before **TP_CA**, the programmed orientation has not yet been reached. The programmed orientation is accepted at **TP_CA**.

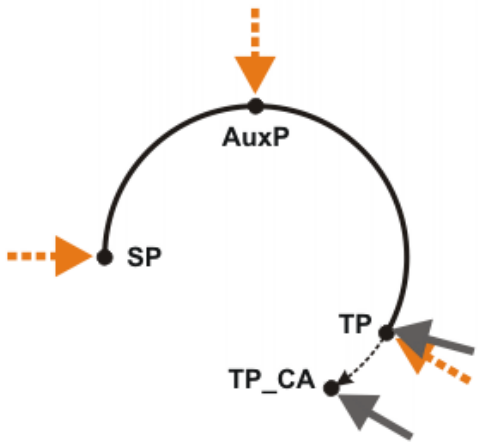


Fig. 6-35: #INTERPOLATE

SP	Start point
AuxP	Auxiliary point
TP	Programmed end point
TP_CA	Actual end point. Determined by the circular angle.

#EXTRAPOLATE The programmed orientation is accepted at **TP**. For **TP_CA**, this orientation is continued in accordance with the circular angle.

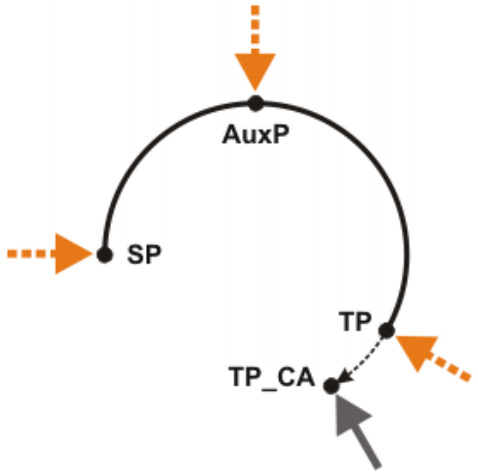


Fig. 6-36: #EXTRAPOLATE

6.5.3 Restrictions for $CIRC_MODE

Restrictions Description from description of the system variables

- If $ORI_TYPE = #IGNORE for a SCIRC segment, $CIRC_MODE is not evaluated.
- If a SCIRC segment is preceded by a SCIRC or SLIN segment with $ORI_TYPE = #IGNORE, #CONSIDER cannot be used in this SCIRC segment.

For SCIRC with circular angle:

- #INTERPOLATE must not be set for the auxiliary point.
- If $ORI_TYPE = #IGNORE, #EXTRAPOLATE must not be set for the end point.

- If it is preceded by a spline segment with $ORI_TYPE = #IGNORE, #EXTRAPOLATE must not be set for the end point.

Explanation of error messages

Some error messages contain the text “Error due to rule x”.

When programming $CIRC_MODE, the following must be observed for both the orientation and the external axes:

1. Rule 1: #CONSIDER is allowed if the start and end point are not ignored.
2. Rule 2: $CIRC_TYPE=#PATH is allowed if the start and end point are not ignored.
3. Rule 3: If $ORI_TYPE=#IGNORE or $EX_AX_IGNORE is set, $CIRC_MODE is no longer evaluated.
4. Rule 4: If a circular angle is programmed, interpolation in the auxiliary point is prohibited.
5. Rule 5: If a circular angle is programmed, the end point may be determined by means of extrapolation if the start and end point are not ignored.
6. Rule 6: If a circular angle is programmed, the end point may be accepted (interpolated) if it is not ignored.
7. Rule 7: #CONSIDER distance only has any effect with infinitely rotating external axes. With other axes, the short distance is always selected; this corresponds to #IGNORE.
8. Rule 8: The component TARGET_PT is only taken into consideration if a circular angle is programmed.
9. Rule 9: Reading of $CIRC_MODE is not allowed anywhere, writing only in the WITH token of a SCIRC.

6.6 Modifying motion commands

Modifying motion commands

There are different reasons for modifying existing motion commands:

Examples of reasons	Modification to be made
Position of the part to be gripped changes. The position of one of five boreholes changes during processing. A weld seam needs to be shortened.	Modification of the position data
Position of the pallet changes.	Modification of the frame data: base
A position has been inadvertently taught with the wrong base or the wrong TOOL.	Modification of the frame data: base and/or tool with update of position
Processing too slow: the cycle time must be improved.	Modification of the motion data: velocity, acceleration Modification of the motion type

Effects of modifying motion commands

Modifying position data

- Only the data set of the point is modified: the point receives new coordinates, as the values are updated with “Touchup”.

  The old point coordinates are overwritten and are subsequently no longer available!

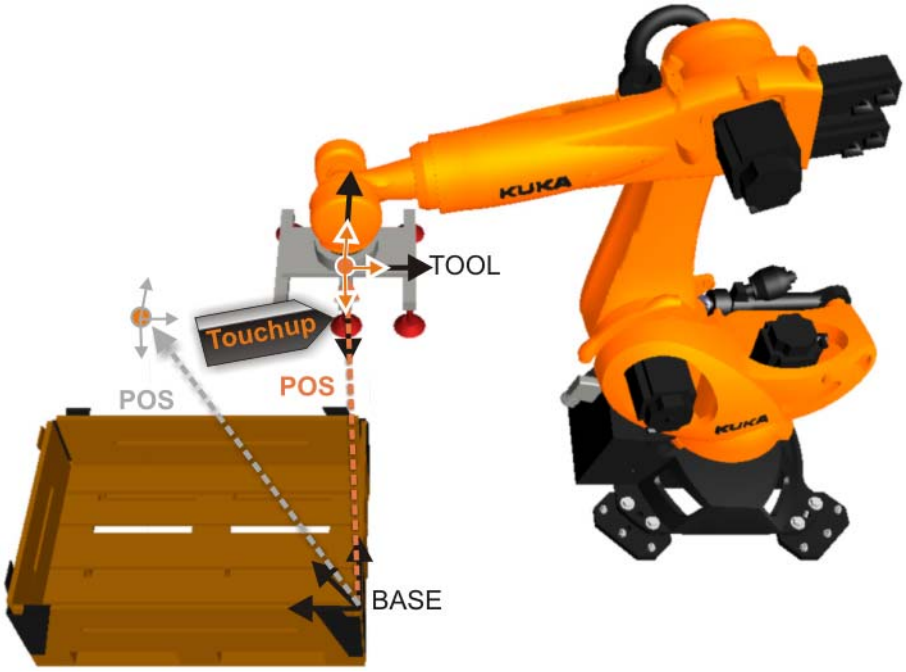


Fig. 6-37: Modification of the robot position with “Touchup”

Modifying frame data

- When modifying frame data (e.g. tool, base), the position is offset (cf. “vector translation”).
- The robot position changes (corresponds to the axis position of the robot)!
  The old coordinates of the point remain saved and valid. Only the reference is changed (e.g. the base).
- The workspace may be exceeded! Certain robot positions are thus not accessible.
- If the robot position is to remain the same despite modification of the frame parameters, the coordinates must be updated by means of “Touchup” in the desired position following modification of the parameters (e.g. Base)!

A user dialog also warns: “Caution: risk of collision when changing point-related frame parameters!”

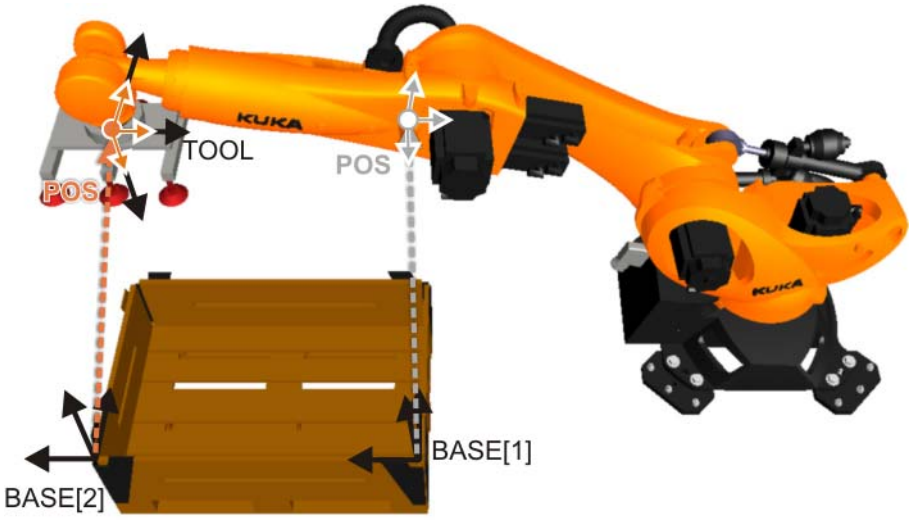


Fig. 6-38: Modifying frame data (example: base)

Modifying motion data

- Modifying the velocity or acceleration changes the motion profile. This can have an effect on the process, particularly in the case of CP applications:
  - Thickness of an adhesive bead
  - Quality of a weld seam

Modifying the motion type

- Modification of the motion type always results in a modification of the path planning! In unfavorable circumstances, this can result in collisions, as the path may change unpredictably.

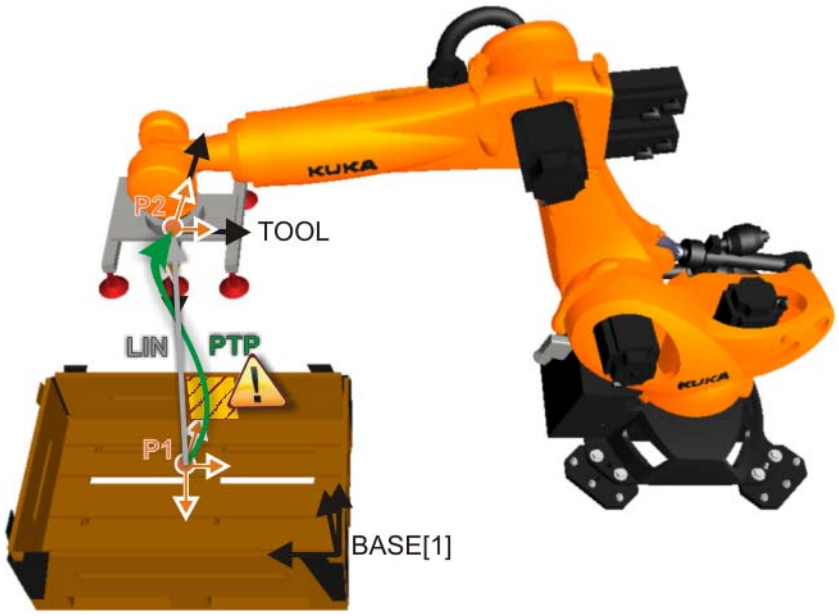


Fig. 6-39: Modifying the motion type

Safety instructions relating to the modification of motion commands

> **NOTICE** Every time motion commands are modified, the robot program must be tested at reduced velocity (T1 mode). Starting the robot program immediately at high velocity can result in damage to the robot system and the overall system, as unforeseeable motions may occur. There is a danger of life-threatening injuries to any person in the danger zone.

Modifying motion parameters – Frames

1. Position the cursor in the line containing the instruction that is to be changed.
2. Press **Change**. The inline form for this instruction is opened.
3. Open the option window “Frames”.
4. Set new “Tool” or “Base” or “External TCP”.
5. Confirm the user dialog “Caution: risk of collision when changing point-related frame parameters!” with **OK**.
6. If you wish to **retain** the **current robot position** with modified tool and/or base settings, it is essential to press the **Touch Up** key to recalculate and save the current position. The desired position must previously have been addressed.
7. Save changes by pressing **Cmd Ok**.

> If frame parameters are modified, the programs must be tested again to ensure there is no risk of a collision.

Modifying the position

Procedure for modifying the robot position:

1. Set T1 mode and position the cursor in the line containing the instruction that is to be changed.
2. Move the robot into the desired position.
3. Press **Change**. The inline form for this instruction is opened.
4. For SPTP and SLIN motions:
   - Press **Touchup** to accept the current position of the TCP as the new end point.

   For SCIRC motions:
   - Press **Teach Aux** to accept the current position of the TCP as the new auxiliary point.
   - Or press **Teach End** to accept the current position of the TCP as the new end point.
5. Confirm the request for confirmation with **Yes**.
6. Save change by pressing **Cmd Ok**.

Modifying motion parameters

This procedure can be used for the following modifications:

- Motion type
- Velocity
- Acceleration
- Approximation
- Approximation distance

1. Position the cursor in the line containing the instruction that is to be changed.
2. Press **Change**. The inline form for this instruction is opened.
3. Modify parameters.
4. Save changes by pressing **Cmd Ok**.

If motion parameters are modified, the programs must be checked again for process reliability and to ensure there is no risk of a collision.

Special features of programming with inline forms

- **Renaming points**
  a. Use the cursor to mark the point that is to be renamed.
  b. Click on the “Change” button to open the inline form.
  c. Change the point name in the inline form (“Point name” box).
  d. Apply the changes with the “Cmd OK” button.
  e. The following dialog is displayed: *Keep previous coordinates for point “Point name”? (TOOL_DATA[n], BASE_DATA[n], #BASE)*
  f. The previous coordinates of the original point are applied with the “Yes” button.
- **Multiple use of points**
  a. Insert a new inline form at the desired point in the program.
  b. Enter in the inline form the point name of the point to be re-used (“Point name” box).

     Attention must be paid to the exact spelling of the original point name.
  c. Make further settings and apply with the “Cmd OK” button.
  d. The following dialog is displayed: *Point “Point name” already exists – overwrite? (TOOL_DATA[n], BASE_DATA[n], #BASE)*
  e. Clicking on the “No” button causes the existing point parameters of the original point not to be overwritten.
- **Inserting program lines**

a. Use the cursor to mark an existing program line.
b. Click on the “Motion” button to open a new inline form.

The new inline form is always inserted after the marked program line. Subsequent program lines are moved down by one line.

- **Deleting program lines**
  a. Use the cursor to mark the program line that is to be deleted.
  b. Menu path: **Edit > Delete**

The point-specific values are retained in the corresponding .dat file. See also “Multiple use of points”.

6.7 Exercise: CP motion and approximate positioning

Aim of the exercise

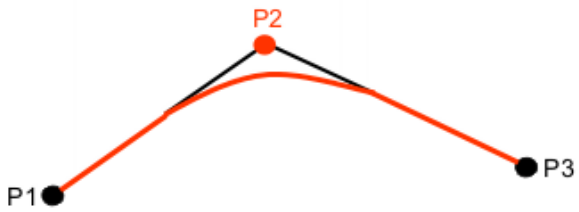
On successful completion of this exercise, you will be able to carry out the following activities:

- Create simple motion programs with the motion types SPTP, SLIN and SCIRC
- Create motion programs with exact positioning points and approximate positioning motions
- Handle programs in the Navigator (copy, duplicate, rename, delete)

Preconditions

The following are preconditions for successful completion of this exercise:

- Basic principles of motion programming with the motion types SPTP, SLIN and SCIRC
- Theoretical knowledge of approximation of motions



- Theoretical knowledge of the HOME position

Task, part A

Carry out the following tasks: program creation, component contour 1

1. Create a new program and give it a meaningful name.
2. Teach the component contour 1 marked on the worktable, using the blue base and pen1 as the tool.
   - The jog velocity on the worktable is 0.3 m/s.
   - Make sure that the longitudinal axis of the tool is always perpendicular to the path contour (orientation control: *Standard*).
3. Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

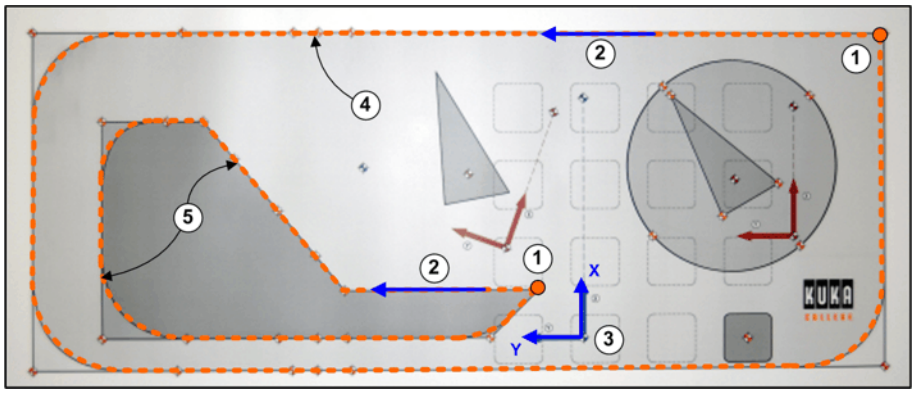


Fig. 6-40: CP motion and approximate positioning: component contour 1 and 2

1	Start points	2	Direction of motion
3	Reference base	4	Component contour 1
5	Component contour 2		

Task, part B

Carry out the following tasks: copying the program and approximate positioning

1. Create a duplicate of your program and give it a meaningful name.
2. Insert the approximate positioning instruction into the motion commands of the new program so that the robot follows the contour continuously.
3. The corners of the contour are to be approximated using different approximation parameters.
4. Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

Supplementary task

Carry out the following tasks: program creation, component contour 2

1. Create a second program and give it a meaningful name. Use the same base and the same tool.
   - The jog velocity on the worktable is 0.3 m/s.
   - Make sure that the longitudinal axis of the tool is always perpendicular to the path contour (orientation control).
2. Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.
3. Create a duplicate of the program Component_contour2 with the name **Component2_CONT**.
4. Insert the approximate positioning instruction into the motion commands of the new program so that the robot follows the contour continuously.
5. Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. What are the characteristics of SLIN and SCIRC motions?

. . . . . .

. . . . . .

. . . . . .

2. How is the velocity specified for SPTP, SLIN and SCIRC motions and what does this velocity refer to?

. . . . . .

. . . . . .

. . . . . .

3. How is the approximation distance specified for SPTP, SLIN and SCIRC motions and what does this distance refer to?

. . . . . .

. . . . . .

4. What must be taken into consideration when `CONT` statements are pasted into existing motion commands?

. . . . . .

. . . . . .

5. What must be taken into consideration when changing the HOME position?

. . . . . .

. . . . . .

6. What must be taken into consideration when correcting, modifying or deleting programmed points?

. . . . . .

7 Programming spline motions

7.1 Overview

The following contents are explained in this training module:

- Creation of spline blocks with inline forms
- Velocity profiles with spline blocks
- Modifications to spline blocks
- Approximation of spline motions

7.2 Programming spline blocks with inline forms

General

- In addition to the individual blocks with SPTP, SLIN, SCIRC and the relative motions, a “SPLINE block” is available.
- The SPLINE block is regarded and planned as an individual motion with a “complex path”.
- There are two kinds of SPLINE block:
  - CP SPLINE block: Spline with CP motions (SPL, SLIN, SCIRC)
  - PTP SPLINE block: Spline with motions solely in the axis space (SPTP only)
- A SPLINE block is a motion block with a TOOL, a BASE and an IPO_MODE, but with different velocities and accelerations in the individual segments.
- The path is planned through all points and thus passes through all points.
- The path is completely calculated in advance. In this way, the entire path is known and the planning can place the path optimally in the workspace of the axes.
- Paths with very tight contours will always result in reduced velocity, as the robot axes are the limiting factor.
- No approximate positioning is required within a spline block, as a continuous path is routed through all points.
- Additional functions can be configured, such as “constant velocity” or “predefined time”.
- The number of segments in the block is only limited by the memory capacity.
- Apart from the motion segments, a spline block may also contain the following elements:
  - Inline commands from technology packages that support the spline functionality
  - Comments and blank lines
- A spline block must not include any other instructions, e.g. variable assignments or logic statements.

The start point of a spline block is the last point before the spline block.
The end point of a spline block is the last point in the spline block.
A spline block does not trigger an advance run stop.

7.3 Velocity profile for spline motions

SPLINE velocity

The path always remains the same, irrespective of the override setting, velocity or acceleration.

The robot controller already takes the physical limits of the robot into consideration during planning. The robot moves as fast as possible within the constraints of the programmed velocity, i.e. as fast as its physical limits will allow. This is an advantage over conventional LIN and CIRC motions for which the physical limits are not taken into consideration during planning. It is only during motion execution that these limits have any effect and can cause stops to be triggered.

Reduction of the velocity

Prime examples of cases in which the velocity has to fall below the programmed velocity include:

- Tight corners
- Major reorientation
- Large motions of the external axes
- Motion in the vicinity of singularities

Reduction of the velocity due to major reorientation can be avoided with spline segments by selecting the orientation control option **Ignore orientation**.

Reduction of the velocity to 0

This is the case for:

- Successive points with the same coordinates.
- Successive SLIN and/or SCIRC segments. Cause: inconstant velocity direction.

  In the case of SLIN-SCIRC transitions, the velocity is also reduced to 0 if the straight line is a tangent of the circle, as the circle, unlike the straight line, is curved.

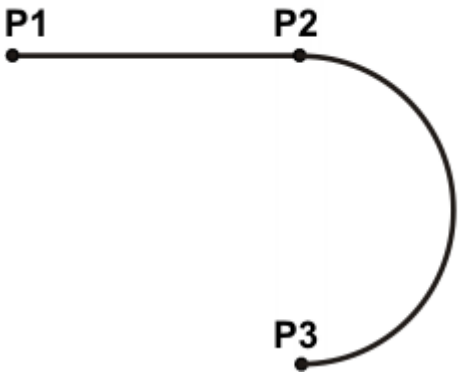


Fig. 7-1: Exact positioning at P2

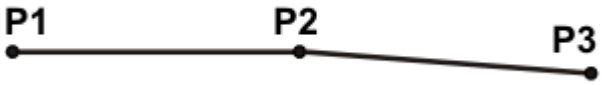


Fig. 7-2: Exact positioning at P2

Exceptions:

- In the case of successive SLIN segments that result in a straight line and in which the orientations change uniformly, the velocity is not reduced.

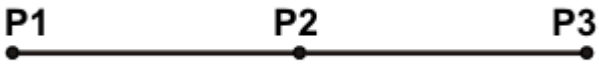


Fig. 7-3: P2 is executed without exact positioning.

- In the case of a SCIRC-SCIRC transition, the velocity is not reduced if both circles have the same center point and the same radius and if the orientations change uniformly. (This is difficult to teach, so calculate and program points.)

> Circles with the same center point and the same radius are sometimes programmed to obtain circles ≥ 360°. A simpler method is to program a circular angle.

Reductions in velocity due to uneven teaching

An uneven distribution of changes in orientation/changes of the external axes relative to the Cartesian arc length often results in undesired drops in velocity.

Example of an uneven distribution:

```
PTP {x 0, y 0, z 0, A 0, B 0, C 0} ; Start point of the spline
SPLINE
SPL {x 0, y 100, z 0, A 10, B 0, C 0} ; 0,1° Reorientation per mm
Cartesian distance
SPL {x 0, y 110, z 0, A 20, B 0, C 0} ; 1° Reorientation per mm
Cartesian distance
SPL {x 0, y 310, z 0, A 31, B 0, C 0} ; 0,055° Reorientation per mm
Cartesian distance
 ENDSPLINE
```

> Cartesian position, orientation and external axes are firmly coupled together by the geometry planning. Pulling the TCP along the Cartesian path tangent of a spline causes the orientation and the external axes to rotate automatically (due to the coupling) and vice versa (in particular, all segment limits are normally completely interpolated). Many undesired drops in the Cartesian velocity are necessary (due to the coupling) in order to adhere to the programmed values
> - orientation jerks ($JERK.ORI),
> - orientation accelerations ($ACC.ORI1) and
> - orientation velocities ($VEL.ORI1)

In the case of an uneven distribution of orientation distances on the Cartesian distance (arc length), the orientation has to accelerate and decelerate very frequently, which tends to go hand in hand with large orientation jerks. An uneven distribution of distances leads to drops in velocity far more frequently than an even (proportional) distribution of the orientation distances. Additionally, in the case of a large jerk, the robot and the robot wrist may start to oscillate.

Remedy:

Distribute orientation and external axes as evenly as possible.

Example of an even distribution:

```
PTP {x 0, y 0, z 0, A 0, B 0, C 0} ; Start point of the spline
SPLINE
SPL {x 0, y 100, z 0, A 10, B 0, C 0} ; 0,1° Reorientation per mm
Cartesian distance
SPL {x 0, y 110, z 0, A 11, B 0, C 0} ; 0,1° Reorientation per mm
Cartesian distance
SPL {x 0, y 310, z 0, A 31, B 0, C 0} ; 0,1° Reorientation per mm
Cartesian distance
 ENDSPLINE
```

Deactivate orientation control via inline form or KRL.

```
$ORI_TYPE = #IGNORE
```

Comparison with and without programmed orientation control.

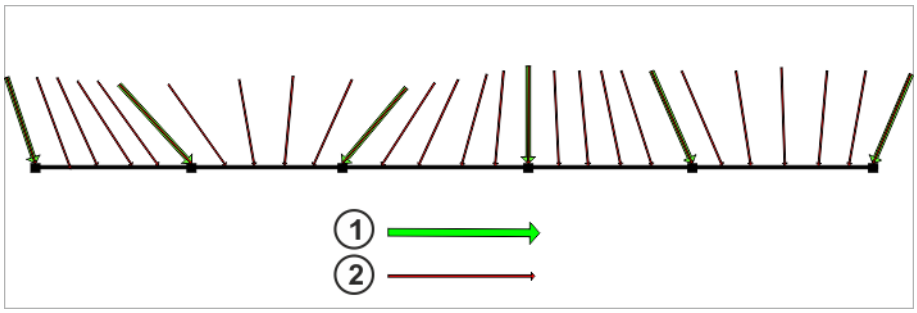


Fig. 7-4: With programmed orientation

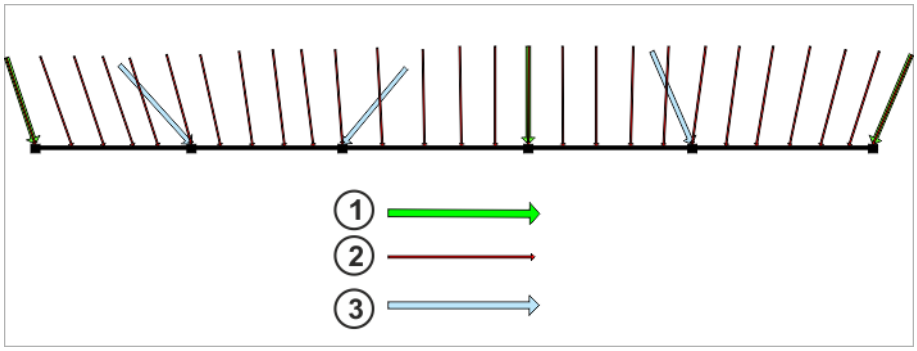


Fig. 7-5: Ignore orientation

Item	Comments
1	Taught position with corresponding orientation
2	Interpolated position
3	Taught position with corresponding orientation, but orientation not applied

NOTICE It is often the case that many points are programmed relatively close together. However, it is primarily the Cartesian path (x, y, z) that is of interest. The spline also interpolates the programmed orientation, however, which can result in reductions in velocity. For this reason, the inline form **IGNORE** should be selected for preference in this case.

7.4 Modifications to spline blocks

Description

- Modification of the position of the point:
  If a point within a spline block is offset, the path is modified, at most, in the 2 segments before this point and the 2 segments after it.
  Small point offsets generally result in small modifications to the path. If, however, very long segments are followed by very short segments or vice versa, small modifications can have a very great effect.
- Modification of the segment type:
  If an SPL segment is changed into an SLIN segment or vice versa, the path changes in the previous segment and the next segment.

Example 1

Original path:

```
PTP P0
SPLINE
 SPL P1
```

```
 SPL P2
 SPL P3
 SPL P4
 SCIRC P5, P6
 SPL P7
 SLIN P8
ENDSPLINE
```

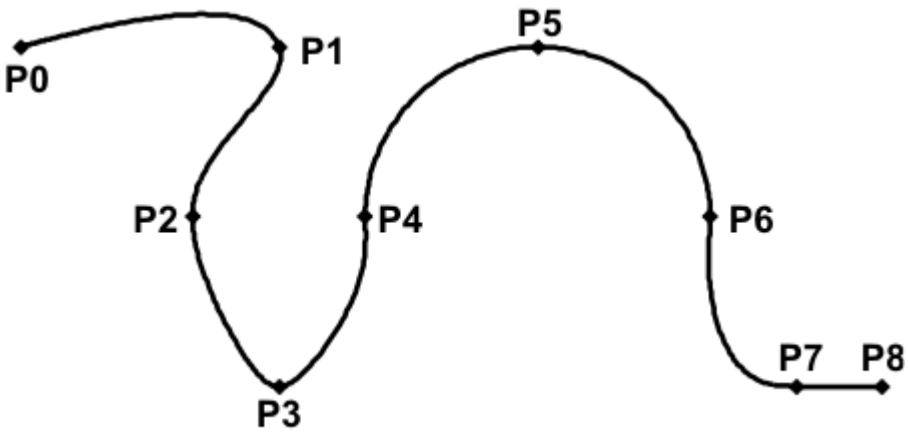


Fig. 7-6: Original path

A point is offset relative to the original path:

P3 is offset. This causes the path to change in segments P1 - P2, P2 - P3 and P3 - P4. Segment P4 - P5 is not changed in this case, as it belongs to an SCIRC and a circular path is thus defined.

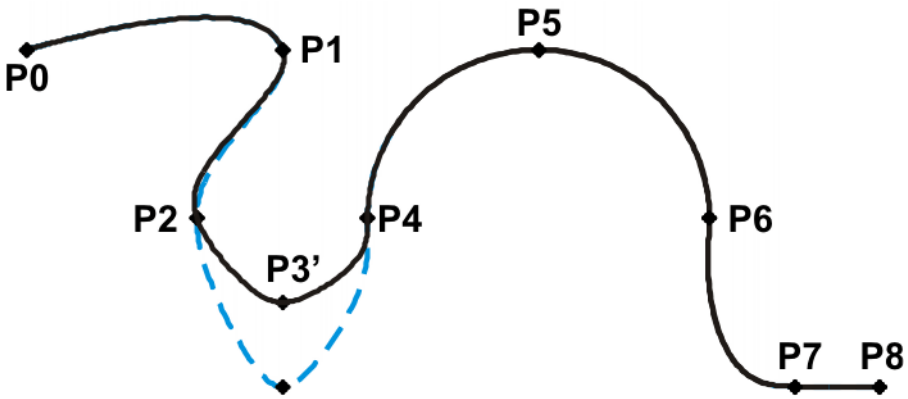


Fig. 7-7: Point has been offset

The type of a segment is changed relative to the original path:

In the original path, the segment type of P2 - P3 is changed from SPL to SLIN. The path changes in segments P1 - P2, P2 - P3 and P3 - P4.

```
PTP P0
SPLINE
 SPL P1
 SPL P2
 SLIN P3
 SPL P4
 SCIRC P5, P6
 SPL P7
```

```
 SLIN P8
ENDSPLINE
```

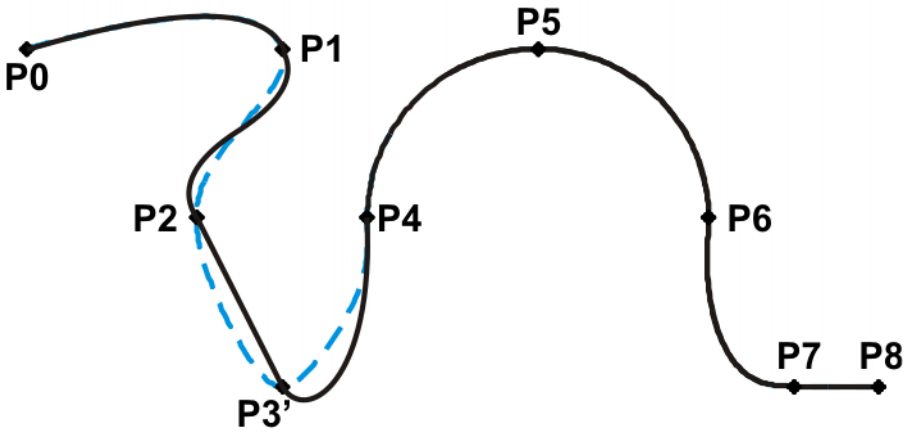


Fig. 7-8: Segment type has been changed

Example 2

Original path:

```
...
SPLINE
 SPL {X 100, Y 0, ...}
 SPL {X 102, Y 0}
 SPL {X 104, Y 0}
 SPL {X 204, Y 0}
ENDSPLINE
```

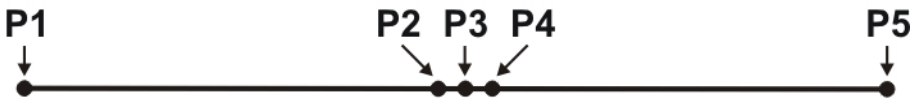


Fig. 7-9: Original path

A point is offset relative to the original path:

P3 is offset. This causes the path to change in all the segments illustrated. Since P2 - P3 and P3 - P4 are very short segments and P1 - P2 and P4 - P5 are long segments, the slight offset causes the path to change greatly.

```
...
SPLINE
 SPL {X 100, Y 0, ...}
 SPL {X 102, Y 1}
 SPL {X 104, Y 0}
 SPL {X 204, Y 0}
ENDSPLINE
```

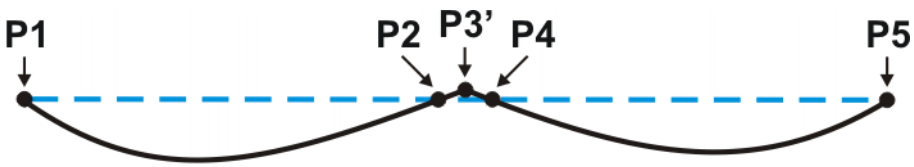


Fig. 7-10: Point has been offset

Remedy:

- Distribute the points more evenly

- Program straight lines (except very short ones) as SLIN segments

7.5 Block selection with spline motions

Spline block

Block selection can be made to the segments of a spline block.

- CP spline block:
  The BCO run is executed as a conventional LIN motion. This is indicated by means of a message that must be acknowledged.
- PTP spline block:
  The BCO run is executed as a conventional PTP motion. This is not indicated by a message.

Following a block selection, the path is generally the same as if the spline were to be executed during normal program execution.

Exceptions are possible if the spline has never yet been executed prior to the block selection and the block selection is made here to the start of the spline block:

The start point of the spline motion is the last point before the spline block, i.e. the start point is outside the block. The robot controller saves the start point during normal execution of a spline. In this way, it is already known in the case of a block selection being carried out subsequently. If the spline block has never yet been executed, however, the start point is unknown.

If the Start key is pressed after the BCO run, the modified path is indicated by means of a message that must be acknowledged.

Example: modified path in the case of block selection to P1

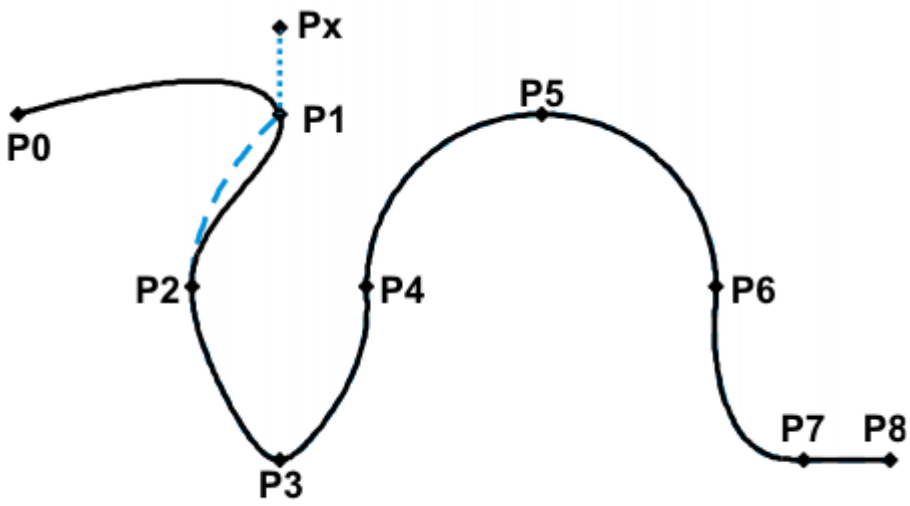


Fig. 7-11: Example: modified path in the case of block selection to P1

```
1  PTP P0
2  SPLINE
3   SPL P1
4   SPL P2
5   SPL P3
6   SPL P4
7   SCIRC P5, P6
8   SPL P7
9   SLIN P8
10 ENDSPLINE
```

Line	Description
2	Header/start of the CP spline block
3 … 9	Spline segments
10	End of the CP spline block

SCIRC

In the case of block selection to a SCIRC segment for which a circular angle has been programmed, the motion is executed to the end point, taking into consideration the circular angle, provided that the robot controller knows the start point.

If the robot controller does not know the start point, the motion is executed to the programmed end point. In this case, a message is generated, indicating that the circular angle is not being taken into consideration.

In the case of a block selection to an individual SCIRC motion, the circular angle is never taken into consideration.

7.6 Approximation of spline motions

All spline blocks and all individual spline motions can be approximated with one another. It makes no difference whether they are CP or PTP spline blocks, nor is the motion type of the individual motion relevant.

The motion type of the approximate positioning arc always corresponds to the second motion. In the case of SPTP-SLIN approximation, for example, the approximate positioning arc is of type CP.

Spline motions cannot be approximated with conventional motions (LIN, CIRC, PTP).

Approximation not possible due to time or advance run stops:

If approximation is not possible for reasons of time or due to an advance run stop, the robot waits at the start of the approximate positioning arc.

- In the case of time reasons: the robot moves again as soon as it has been possible to plan the next block.
- In the case of an advance run stop: the end of the current block is reached at the start of the approximate positioning arc. This means that the advance run stop is canceled and the robot controller can plan the next block. Robot motion is resumed.

In both cases, the robot now moves along the approximate positioning arc. Approximate positioning is thus technically possible; it is merely delayed.

This response differs from that for LIN, CIRC or PTP motions. If approximate positioning is not possible for the reasons specified, the motion is executed to the end point with exact positioning.

No approximate positioning in MSTEP and ISTEP:

In the program run modes MSTEP and ISTEP, the robot stops exactly at the end point, even in the case of approximated motions.

In the case of approximate positioning from one spline block to another spline block, the result of this exact positioning is that the path is different in the last segment of the first block and in the first segment of the second block from the path in program run mode GO.

In all other segments of both spline blocks, the path is identical in MSTEP, ISTEP and GO.

7.7 Replacing an approximated CP motion with a spline block

Description

In order to replace approximated conventional CP motions with spline blocks, the program must be modified as follows:

- Replace LIN - LIN with SLIN - SPL - SLIN.
- Replace LIN - CIRC with SLIN - SPL - SCIRC.

  Recommendation: Allow the SPL to project a certain way into the original circle. The SCIRC thus starts later than the original CIRC.

In approximated motions, the corner point is programmed. In the spline block, the points at the start and end of the approximation are programmed instead.

The following approximated motion is to be reproduced:

```
LIN P1 C_DIS
LIN P2
```

Spline motion:

```
SPLINE
 SLIN P1A
 SPL P1B
 SLIN P2
ENDSPLINE
```

P1A = start of approximation, P1B = end of approximation

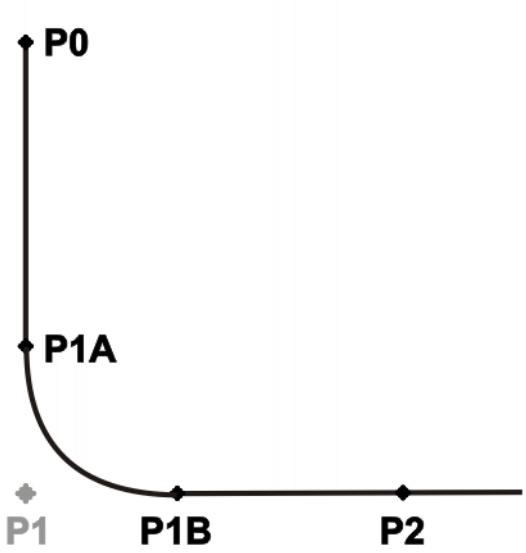


Fig. 7-12: Approximated motion - spline motion

Ways of determining P1A and P1B:

- Execute the approximated path and save the positions at the desired point by means of Trigger.
- Calculate the points in the program with KRL.
- The start of the approximation can be determined from the approximate positioning criterion. Example: If C_DIS is specified as the approximate positioning criterion, the distance from the start of the approximation to the corner point corresponds to the value of $APO.CDIS.

  The end of the approximation is dependent on the programmed velocity.

The SPL path does not correspond exactly to the approximate positioning arc, even if P1A and P1B are exactly at the start/end of the approximation. In order to recreate the exact approximate positioning arc, additional points must be inserted into the spline. Generally, one point is sufficient.

Example

The following approximated motion is to be reproduced:

```
$APO.CDIS=20
$VEL.CP=0.5
LIN {Z 10} C_DIS
LIN {Y 60}
```

Spline motion:

```
SPLINE WITH $VEL.CP=0.5
 SLIN {Z 30}
 SPL {Y 30, Z 10}
 SLIN {Y 60}
ENDSPLINE
```

The start of the approximate positioning arc has been calculated from the approximate positioning criterion.

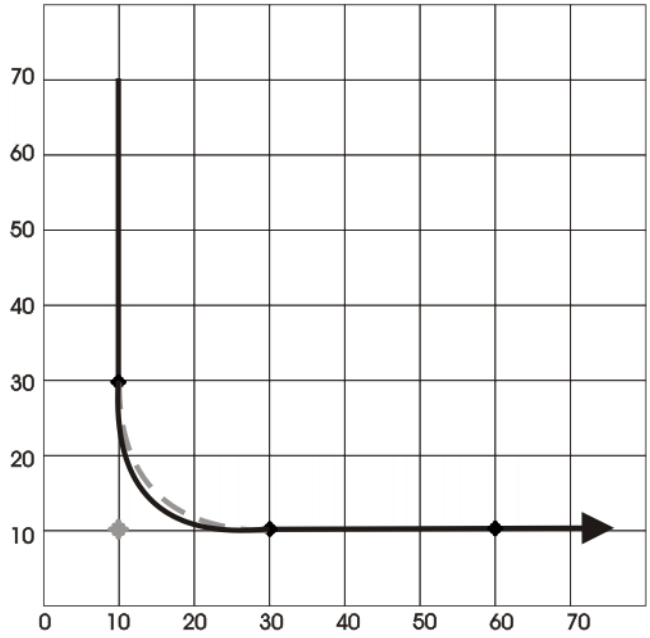


Fig. 7-13: Example: Approximated motion - spline motion 1

The SPL path does not yet correspond exactly to the approximate positioning arc. For this reason, an additional SPL segment is inserted into the spline.

```
SPLINE WITH $VEL.CP=0.5
 SLIN {Z 30}
 SPL {Y 15, Z 15}
 SPL {Y 30, Z 10}
 SLIN {Y 60}
ENDSPLINE
```

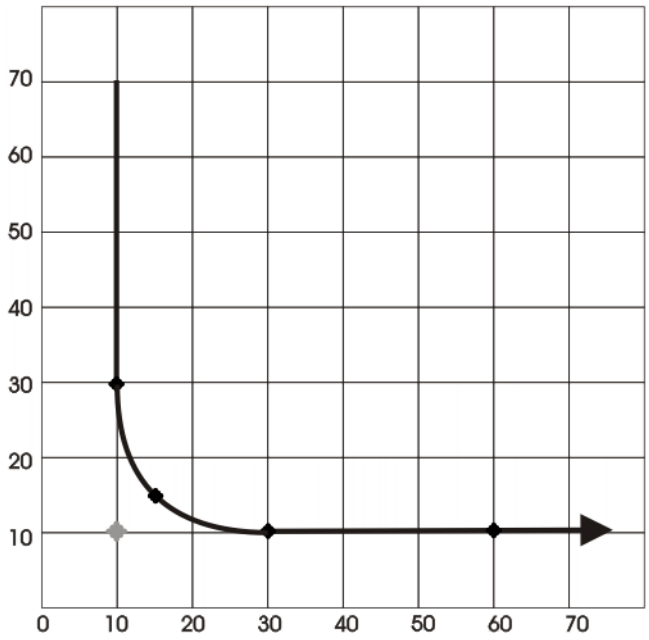


Fig. 7-14: Example: Approximated motion - spline motion 2

With the additional point, the path now corresponds to the approximate positioning arc.

7.7.1 SLIN-SPL-SLIN transition

In the case of a SLIN-SPL-SLIN segment sequence, it is usually desirable for the SPL segment to be located within the smaller angle between the two straight lines. Depending on the start and end point of the SPL segment, the path may also move outside this area.

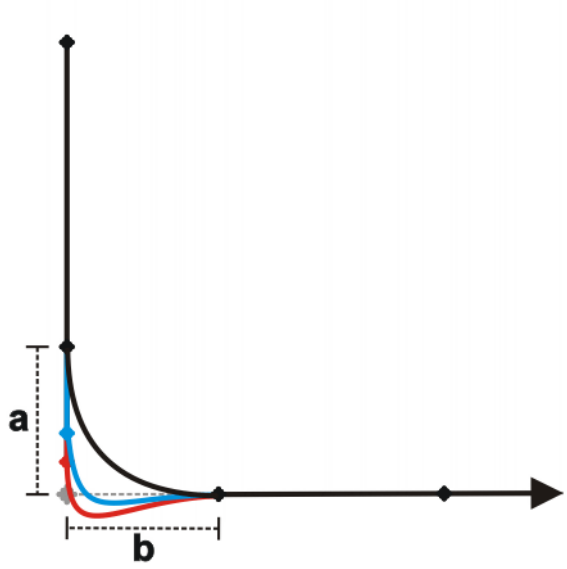


Fig. 7-15: SLIN-SPL-SLIN

The path remains inside if the following conditions are met:

- The extensions of the two SLIN segments intersect.

- 2/3 ≤ a/b ≤ 3/2

  a = distance from start point of the SPL segment to intersection of the SLIN segments

  b = distance from intersection of the SLIN segments to end point of the SPL segment

7.8 Programming CP SPLINE blocks via inline forms

CP SPLINE block

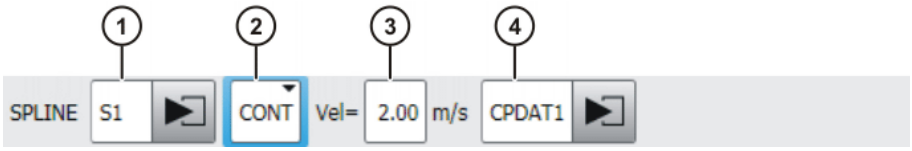


Fig. 7-16: Inline form for CP spline block

Item	Description
1	Name of the spline block. The system automatically generates a name. The name can be overwritten. Position the cursor in this box to edit the motion data. The corresponding option window is opened.
2	■ **CONT**: end point is approximated. ■ **[Empty box]**: the motion stops exactly at the end point.
3	Cartesian velocity ■ **0.001 … 2 m/s**
4	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Position the cursor in this box to edit the motion data. The corresponding option window is opened.

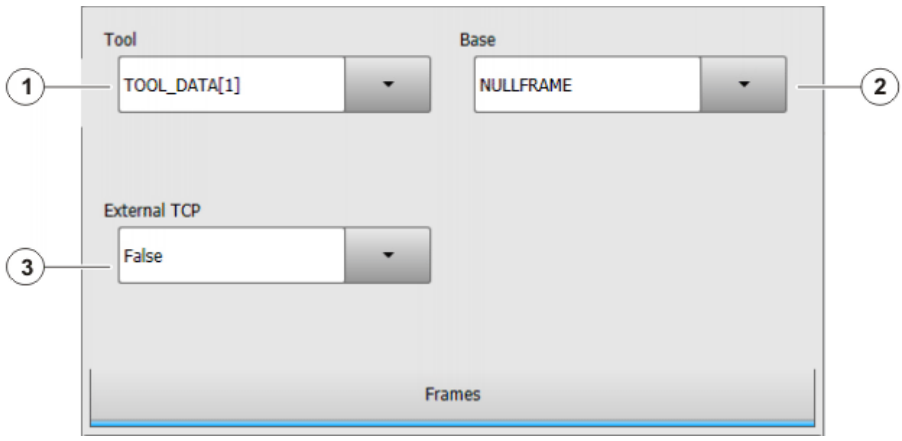


Fig. 7-17: Option window “Frames” (CP and PTP spline block)

Item	Description
1	Tool selection. Or: If **True** in the box **External TCP**: workpiece selection. ■ **[1]** … **[16]**
2	Base selection. Or: If **True** in the box **External TCP**: fixed tool selection. ■ **[1]** … **[32]**
3	Interpolation mode ■ **False**: The tool is mounted on the mounting flange. ■ **True**: The tool is a fixed tool.

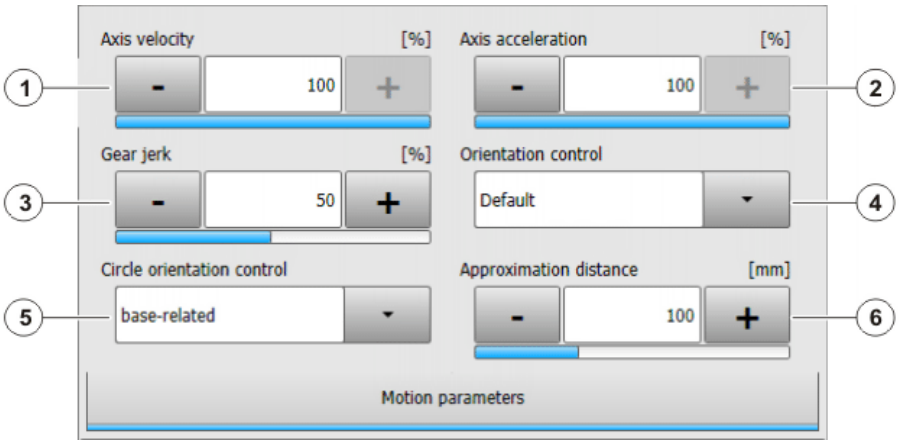


Fig. 7-18: Option window “Motion parameters” (CP spline block)

Item	Description
1	Axis velocity. The value refers to the maximum value specified in the machine data. ■ **1 … 100%**
2	Axis acceleration. The value refers to the maximum value specified in the machine data. ■ **1 … 100%**
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. ■ **1 … 100%**
4	Orientation control selection.
5	Orientation control reference system selection. This parameter only affects SCIRC segments (if present).
6	This box is only displayed if **CONT** was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum distance is that of the last segment in the spline. If there is only one segment present, the maximum distance is half the segment length. If a higher value is entered, this is ignored and the maximum value is used.

Programming in the CP SPLINE block

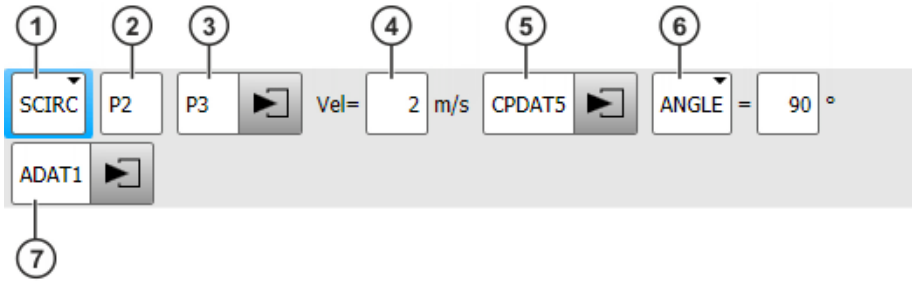


Fig. 7-19: Inline form for CP spline segment

By default, not all boxes of the inline form are displayed. The boxes can be displayed or hidden using the **Switch parameter** button.

Item	Description
1	Motion type ■ **SPL**, **SLIN** or **SCIRC**
2	Only for **SCIRC**: Point name for the auxiliary point. The system automatically generates a name. The name can be overwritten.
3	Point name for the end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened.
4	Cartesian velocity By default, the value that is valid for the spline block is also valid for the segment. A separate value can be assigned here for the segment if required. The value applies only for this segment. ■ **0.001 … 2 m/s**
5	Name for the motion data set. The system automatically generates a name. The name can be overwritten. By default, the values that are valid for the spline block are also valid for the segment. Separate values can be assigned here for the segment if required. The values apply only for this segment. Touch the arrow to edit the data. The corresponding option window is opened.
6	Circular angle Only available if the motion type **SCIRC** has been selected. ■ **- 9 999° … + 9 999°** If a value less than -400° or greater than +400° is entered, a request for confirmation is generated when the inline form is saved asking whether entry is to be confirmed or rejected.
7	Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened.

7.9 Programming PTP SPLINE blocks via inline form

PTP SPLINE block

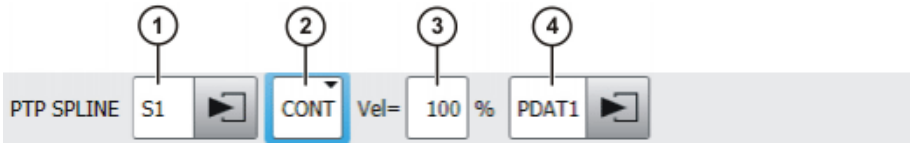


Fig. 7-20: Inline form “PTP SPLINE block”

Item	Description
1	Name of the spline block. The system automatically generates a name. The name can be overwritten. Position the cursor in this box to edit the motion data. The corresponding option window is opened.
2	■ **CONT**: end point is approximated. ■ **[Empty box]**: the motion stops exactly at the end point.
3	Axis velocity ■ **1 … 100%**
4	Name for the motion data set. The system automatically generates a name. The name can be overwritten. Position the cursor in this box to edit the motion data. The corresponding option window is opened.

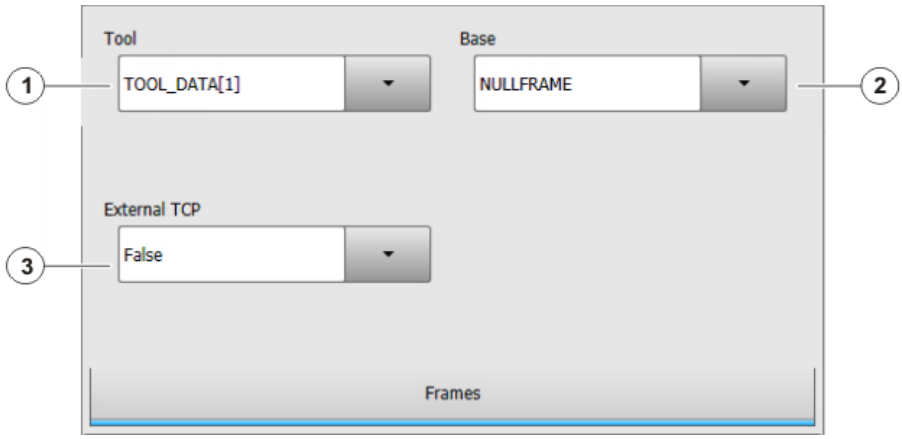


Fig. 7-21: Option window “Frames” (CP and PTP spline block)

Item	Description
1	Tool selection. Or: If **True** in the box **External TCP**: workpiece selection. ■ **[1]** … **[16]**
2	Base selection. Or: If **True** in the box **External TCP**: fixed tool selection. ■ **[1]** … **[32]**
3	Interpolation mode ■ **False**: The tool is mounted on the mounting flange. ■ **True**: The tool is a fixed tool.

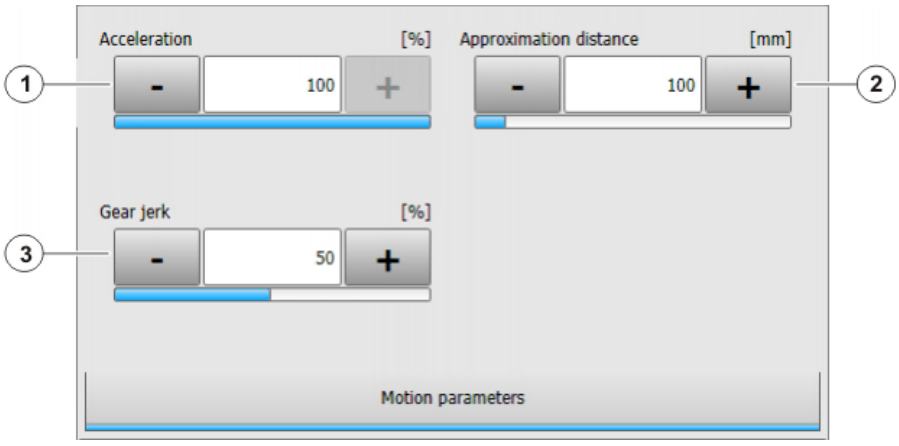


Fig. 7-22: Option window “Motion parameters” (PTP spline block)

Item	Description
1	Axis acceleration. The value refers to the maximum value specified in the machine data. ■ **1 … 100%**
2	This box is only displayed if **CONT** was selected in the inline form. Furthest distance before the end point at which approximate positioning can begin. The maximum distance is that of the last segment in the spline. If there is only one segment present, the maximum distance is half the segment length. If a higher value is entered, this is ignored and the maximum value is used.
3	Gear jerk. The jerk is the change in acceleration. The value refers to the maximum value specified in the machine data. ■ **1 … 100%**

Programming in the PTP SPLINE block

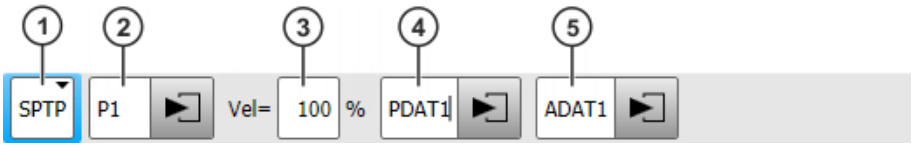


Fig. 7-23: Inline form for SPTP segment

By default, not all boxes of the inline form are displayed. The boxes can be displayed or hidden using the **Switch parameter** button.

Item	Description
1	Motion type **SPTP**
2	Point name for end point. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the point data. The corresponding option window is opened.

Item	Description
3	Axis velocity By default, the value that is valid for the spline block is also valid for the segment. A separate value can be assigned here for the segment if required. The value applies only for this segment. ■ **1 … 100%**
4	Name for the motion data set. The system automatically generates a name. The name can be overwritten. By default, the values that are valid for the spline block are also valid for the segment. Separate values can be assigned here for the segment if required. The values apply only for this segment. Touch the arrow to edit the point data. The corresponding option window is opened.
5	Name of the data set containing logic parameters. The system automatically generates a name. The name can be overwritten. Touch the arrow to edit the data. The corresponding option window is opened.

7.10 Exercise: Path contour with spline block

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Create complex motion programs with the motion types SPTP, SLIN and SCIRC
- Create motion programs with spline blocks

Preconditions

The following are preconditions for successful completion of this exercise:

- Basic principles of motion programming with the motion types SPTP, SLIN, SCIRC, SPL
- Knowledge of the use of spline blocks

Task description

Carry out the following tasks: Program creation with spline contour

1. Create a new program and give it a meaningful name.
2. Use sensible tool and base data.
3. The velocity on the path is to be 0.25 m/s.
4. Teach the spline contour on the plastic sheet. For this, use a spline block, SPL, SLIN and SCIRC.

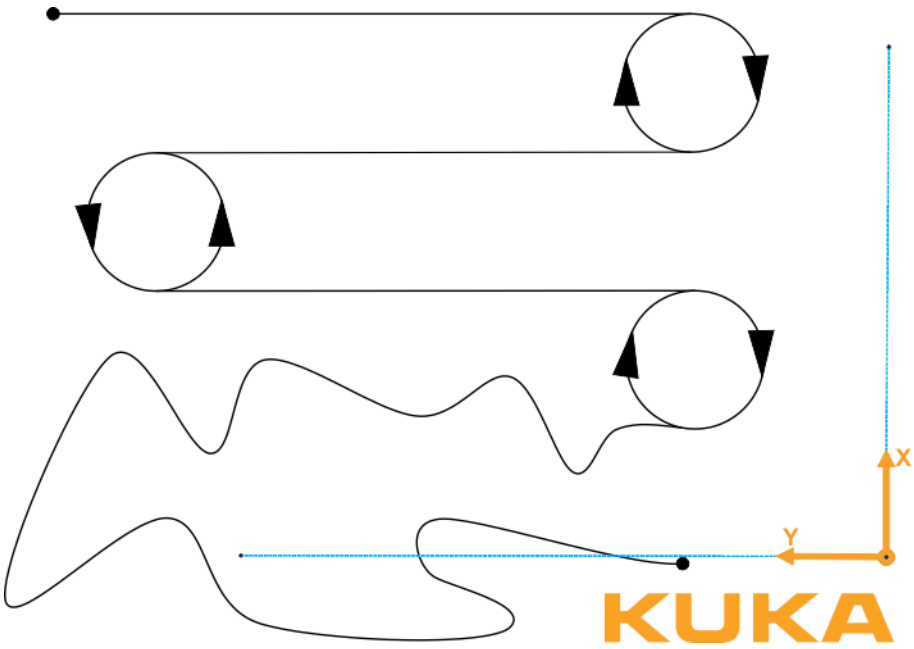


Fig. 7-24: Path contour from spline block

5. Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. Which motion commands can be used in a PTP spline block?

. . . . . .

. . . . . .

. . . . . .

2. Where is the start point for a spline block?

. . . . . .

. . . . . .

. . . . . .

3. Under what conditions does the velocity during spline motions fall below the programmed velocity?

. . . . . .

. . . . . .

4. What must be observed when modifying spline motions?

. . . . . .

. . . . . .

8 Using logic functions in the robot program

8.1 Overview

The following contents are explained in this training module:

- Programming wait functions
- Programming switching functions
- Programming time-distance functions

8.2 Introduction to logic programming

Use of inputs and outputs in logic programming

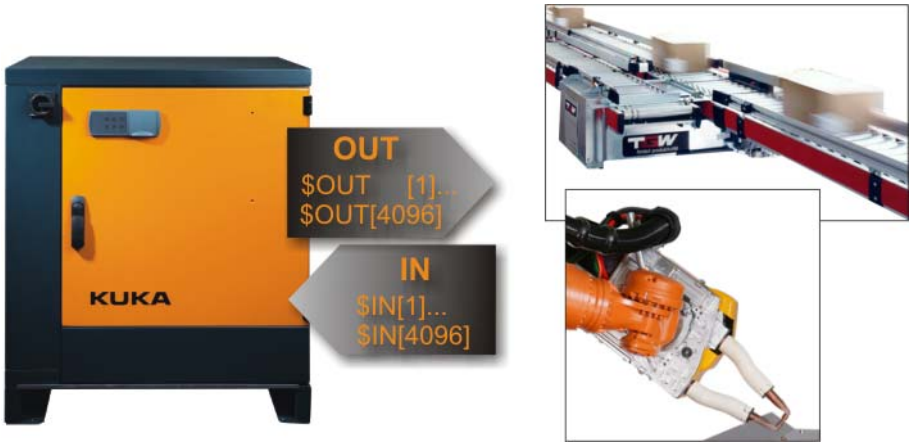


Fig. 8-1: Digital inputs and outputs

In order to implement **communication** *with the* **periphery** *of the robot controller,* **digital** *and* **analog inputs/outputs** *can be used.*

Explanation of terms

Term	Explanation	Example
Communication	Signal exchange via a serial interface	Polling a state (gripper open/closed)
Periphery	“Surroundings”	Tool (e.g. gripper, weld gun, etc.), sensors, material conveyor systems, etc.
Digital	Digital technology: value- and time-discrete signals	Sensor signal: part present: value 1 (TRUE), part not present: value 0 (FALSE)
Analog	Mapping of a physical variable	Temperature measurement
Inputs	The signals *arriving* in the controller via the field bus interface	Sensor signal: gripper is open / gripper is closed
Outputs	The signals *sent* by the controller to the periphery via the field bus interface	Command for switching a valve to close a finger gripper.

Input and output signals are used for logic statements in the programming of KUKA robots:

- **OUT** | Switches an output at a specific point in the program

- **WAIT FOR** | Signal-dependent wait function: the controller waits for a signal here:
  - Input **IN**
  - Output **OUT**
  - Time signal **TIMER**
  - Internal memory address in the controller (cyclical flag/1-bit memory) **FLAG** or **CYCFLAG** (if continuously and cyclically evaluated)
- **WAIT** | Time-dependent wait function: the controller waits a specified time at this point in the program.

8.3 Programming wait functions

Computer advance run

The computer advance run loads the motion blocks in the advance run (not visible for the operator) to allow the controller to carry out path planning in the case of approximate positioning commands. It is not only motion data that are processed in the advance run, however, but also arithmetical data and commands for controlling the periphery.

```
 1 DEF Depal_Box1()
 2
 3 INI
 4 SPTP HOME VEL= 100 % DEFAULT
 5 SPTP P1 VEL= 100 % PDAT1 Tool[5] Base[10]
 6 SPTP P2 VEL= 100 % PDAT1 Tool[5] Base[10]
 7 SLIN P3 VEL= 1 m/s CPDAT1 Tool[5] Base[10]
 8 SLIN P4 VEL= 1 m/s CPDAT2 Tool[5] Base[10]
 9 SPTP P5 VEL= 100% PDAT1 Tool[5] Base[10]
10 OUT 26'' State=TRUE
11 SPTP HOME VEL= 100 % DEFAULT
12
13 END
```

Line	
6	Position of the main run pointer
9	Possible position of the advance run pointer (not visible)
10	Command set that triggers an advance run stop

Certain statements trigger an advance run stop. These include statements that influence the periphery, e.g. OUT statements (close gripper, open weld gun). If the advance run pointer is stopped, approximate positioning cannot be carried out.

Wait functions

Wait functions in a motion program can be programmed very simply using inline forms. A distinction is made between time-dependent wait functions and signal-dependent wait functions.

With **WAIT**, the robot motion is stopped for a programmed time. WAIT always triggers an advance run stop.

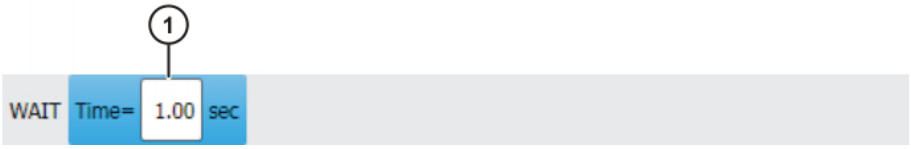


Fig. 8-2: Inline form “WAIT”

Item	Description
1	Wait time ■ **≥ 0** s

Example program:

```
SPTP P1 Vel=100% PDAT1 Tool[1] Base[1]
SPTP P2 Vel=100% PDAT2 Tool[1] Base[1]
WAIT Time=2 sec
SPTP P3 Vel=100% PDAT3 Tool[1] Base[1]
```

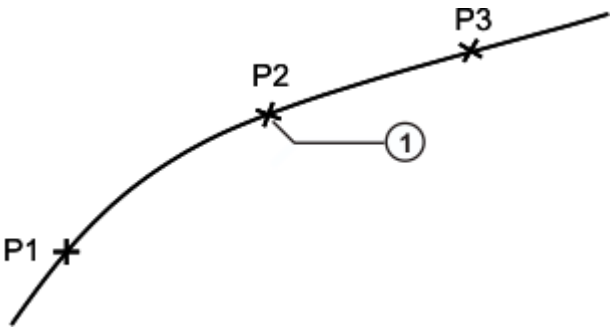


Fig. 8-3: Example motion for logic

Item	Comments
1	Motion is interrupted for 2 seconds at point P2.

WAIT FOR sets a signal-dependent wait function.

If required, several signals (maximum 12) can be linked. If a logic operation is added, boxes are displayed in the inline form for the additional signals and links.

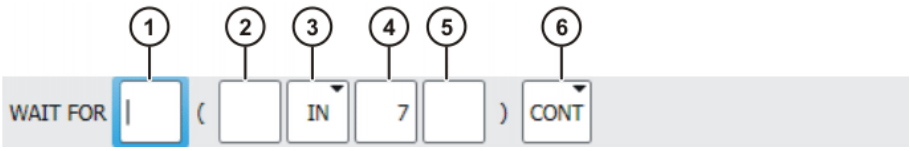


Fig. 8-4: Inline form “WAITFOR”

Item	Description
1	Add external logic operation. The operator is situated between the bracketed expressions. ■ **AND** ■ **OR** ■ **EXOR** Add NOT. ■ **NOT** ■ **[Empty box]** Enter the desired operator by means of the corresponding button.
2	Add internal logic operation. The operator is situated inside a bracketed expression. ■ **AND** ■ **OR** ■ **EXOR** Add NOT. ■ **NOT** ■ **[Empty box]** Enter the desired operator by means of the corresponding button.
3	Signal for which the system is waiting ■ **IN** ■ **OUT** ■ **CYCFLAG** ■ **TIMER** ■ **FLAG**
4	Number of the signal ■ **1 … 4096**
5	If a name exists for the signal, this name is displayed. Only for the user group “Expert”: A name can be entered by pressing **Long text**. The name is freely selectable.
6	■ **CONT**: Execution in the advance run ■ **[Empty box]**: Execution with advance run stop

If the entry CONT is used, it must be taken into consideration that the signal is polled in the advance run. A signal change after the advance run will not be detected.

Logic operations

If signal-dependent wait functions are used, so too are logic operations. Logic operations can be used for combined polling of different signals or states: dependencies can be created, for example, or specific states can be excluded.

The result of a function with a logic operator always provides a truth value, i.e. “True” (value 1) or “False” (value 0).

Fig. 8-5: Example and principle of a logic operation

Operators for logic operations are:

- **NOT** | This operand is used for negation, i.e. the value is inverted (“True” becomes “False”).
- **AND** | The result of the expression is true if both linked expressions are true.
- **OR** | The result of the expression is true if at least one of the linked expressions is true.
- **EXOR** | The result of the expression is true if both of the expressions linked by the operator have different truth values.

Processing with and without advance run (CONT)

Signal-dependent wait functions can be programmed with or without processing in the advance run. **Without advance run** means that the motion will always stop at the point and that the signal will be checked there: (1) (>>> Fig. 8-6). In other words, the point cannot be approximated.

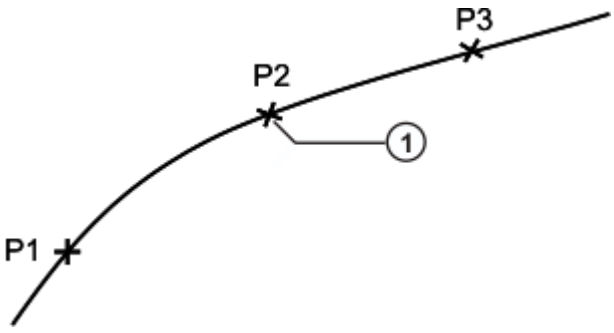


Fig. 8-6: Example motion for logic

```
SPTP P1 Vel=100% PDAT1 Tool[1] Base[1]
SPTP P2 CONT Vel=100% PDAT2 Tool[1] Base[1]
WAIT FOR IN 10 'door_signal'
SPTP P3 Vel=100% PDAT3 Tool[1] Base[1]
```

If a WAIT FOR line without CONT is processed, a notification message is generated: “Approximate positioning not possible”.

Signal-dependent wait functions programmed **with advance run** allow approximate positioning to be carried out for the point before the command line. However, the current position of the advance run pointer is not unambiguous (default value: three motion blocks), so the precise moment at which the signal will be checked is not specified (1) (>>> Fig. 8-7). Furthermore, signal changes after the signal has been checked are not detected!

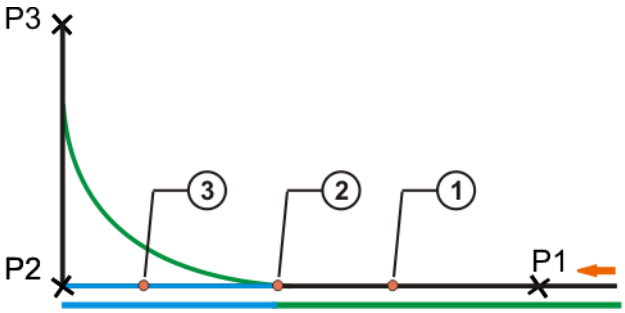


Fig. 8-7: Example motion for logic with advance run

	Position	Switching range
1	Robot position in which the condition is met so that the robot approximates the motion	Green switching range for *activation* of the approximate positioning contour. This is only set and can no longer be deactivated.
2	Start of the approximate positioning motion	*Activation* monitoring ■ TRUE: Approximation ■ FALSE: Motion to the end point
3	Robot position in which the condition is met so that the robot does not approximate the motion	Blue switching range for motion to and stop at point P2

```
SPTP P1 Vel=100% PDAT1 Tool[1] Base[1]
SPTP P2 CONT Vel=100% PDAT2 Tool[1] Base[1]
WAIT FOR IN 10 'door_signal' CONT
SPTP P3 Vel=100% PDAT3 Tool[1] Base[1]
```

Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **WAIT FOR** or **WAIT**.
3. Set the parameters in the inline form.
4. Save instruction with **Cmd Ok**.

8.4 Programming simple switching functions

Simple switching function

A switching function can be used to send a digital signal to the periphery. For this, an output number defined beforehand and corresponding to the interface is used.

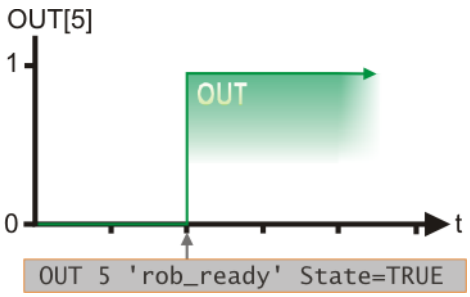


Fig. 8-8: Static switching function

The signal is set statically, i.e. it remains active until a different value is assigned to the output. The switching function is implemented in the program by means of an inline form:

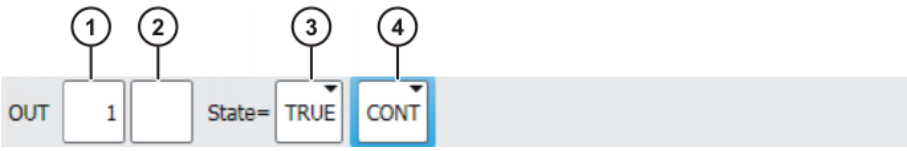


Fig. 8-9: Inline form “OUT”

Item	Description
1	Output number ■ **1 … 4096**
2	If a name exists for the output, this name is displayed. Only for the user group “Expert”: A name can be entered by pressing **Long text**. The name is freely selectable.
3	State to which the output is switched ■ **TRUE** ■ **FALSE**
4	■ **CONT**: Execution in the advance run ■ **[Empty box]**: Execution with advance run stop

If the entry CONT is used, it must be taken into consideration that the signal is set in the advance run.

Pulsed switching functions

As in the case of simple switching functions, the value of an output is changed here, as well. With pulsing, however, the signal is withdrawn again after a defined time.

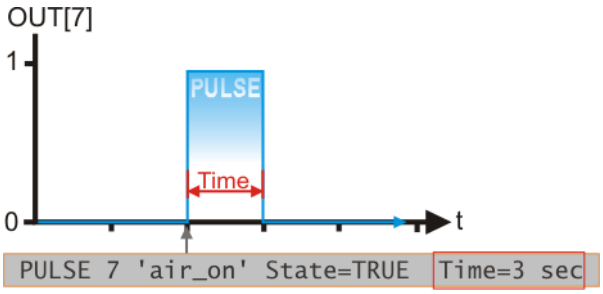


Fig. 8-10: Pulsed signal level

Once again, programming is carried out by means of an inline form in which a pulse of a defined length is set.

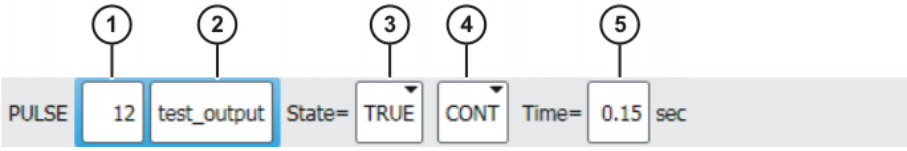


Fig. 8-11: Inline form “PULSE”

Item	Description
1	Output number ■ **1 … 4096**
2	If a name exists for the output, this name is displayed. Only for the user group “Expert”: A name can be entered by pressing **Long text**. The name is freely selectable.
3	State to which the output is switched ■ **TRUE**: “High” level ■ **FALSE**: “Low” level
4	■ **CONT**: Execution in the advance run ■ **[Empty box]**: Execution with advance run stop
5	Length of the pulse ■ **0.10 … 3.00 s**

Effects of CONT on switching functions

without CONT:

If the entry CONT is left out in the inline form OUT, an **advance run stop** is forced during the switching operation and exact positioning is carried out at the point before the switching command. Once the output has been set, the motion is resumed.

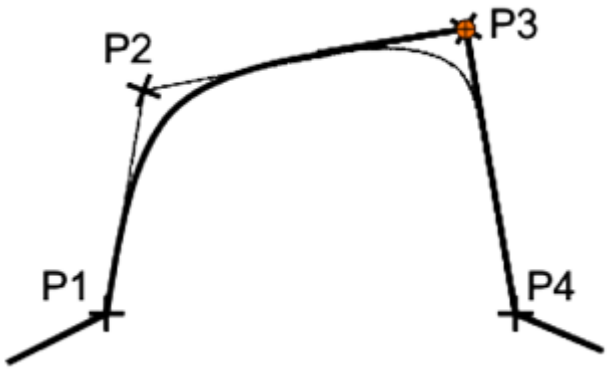


Fig. 8-12: Example motion with switching with advance run stop

```
SLIN P1 Vel=0.2 m/s CPDAT1 Tool[1] Base[1]
SLIN P2 CONT Vel=0.2 m/s CPDAT2 Tool[1] Base[1]
SLIN P3 CONT Vel=0.2 m/s CPDAT3 Tool[1] Base[1]
OUT 5 'rob_ready' State=TRUE
SLIN P4 Vel=0.2 m/s CPDAT4 Tool[1] Base[1]
```

with CONT:

Setting the entry CONT has the effect that the advance run pointer is not stopped (no advance run stop is triggered). This means that a motion before the switching command can be approximated. The signal is set in the **advance run**.

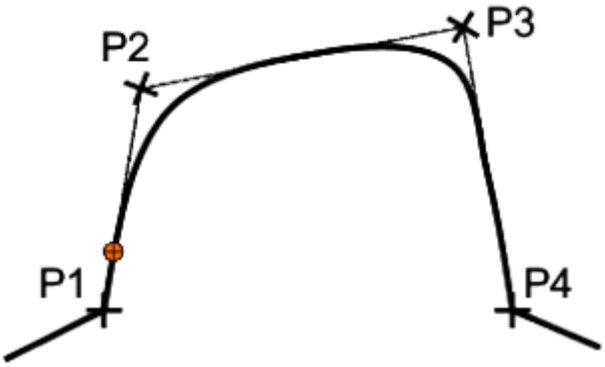


Fig. 8-13: Example motion with switching in the advance run

```
SLIN P1 Vel=0.2 m/s CPDAT1 Tool[1] Base[1]
SLIN P2 CONT Vel=0.2 m/s CPDAT2 Tool[1] Base[1]
SLIN P3 CONT Vel=0.2 m/s CPDAT3 Tool[1] Base[1]
OUT 5 'rob_ready' State=TRUE CONT
SLIN P4 Vel=0.2 m/s CPDAT4 Tool[1] Base[1]
```

The default value for the advance run pointer is three motion blocks. The advance run may vary, however; i.e. it has to be taken into consideration that the switching point may not always occur at the same time!

Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **OUT** > **OUT** or **PULSE**.
3. Set the parameters in the inline form.
4. Save instruction with **Cmd Ok**.

8.5 Logic programming with SPLINE

When programming individual SPLINE blocks, or within the SPLINE blocks, logic can additionally be used in the inline form for the new motions. This logic is also available as a separate inline form. This logic can naturally also be programmed using KRL.

- The following are always available:
  - Trigger (switching command on the path)
  - Conditional stop
- Only in a CP SPLINE block
  - Constant velocity range
- Only in a CP SPLINE block and only as a KRL command
  - Time block

Programming in the inline form of the motion

- Programming with individual blocks:

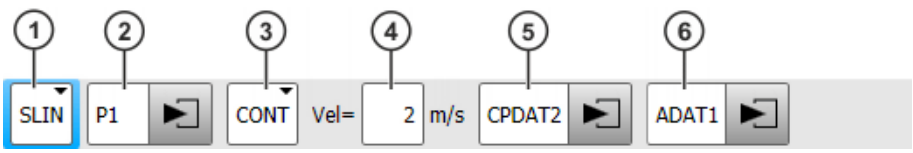


Fig. 8-14: Inline form “SLIN” (individual motion)

The data set ADATx is available.

- Programming in the SPLINE block:

Fig. 8-15: Inline form for SPL segment

By default, only the motion type and point name are displayed. The **Switch parameter** button can be used to add and modify **Velocity**(3), **Motion parameter**(4) and/or **Logic**(5). Here, once again, the logic parameters are modified and saved under ADATx.

- Option window “Logic parameters”, complete

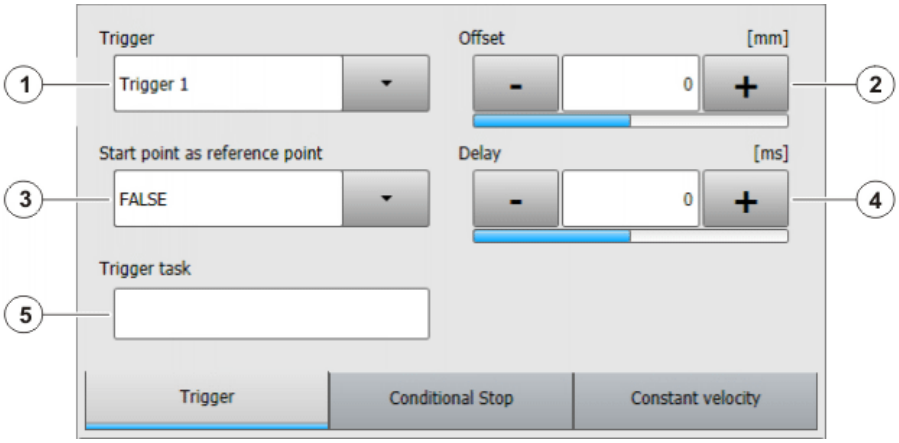


Fig. 8-16: Trigger

Outside the CP SPLINE block, the constant velocity range is not displayed.

Logic inline form for SPLINE motions

Select the menu sequence **Commands** > **Logic** > **Spline trigger** or **Spline Stop Condition**.

- Spline trigger (Trigger)

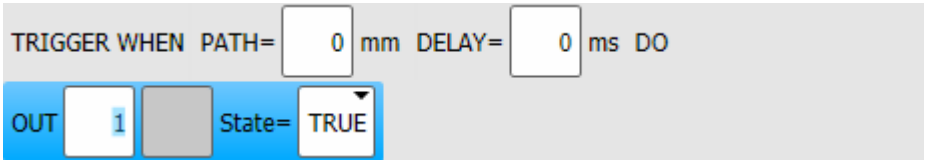


Fig. 8-17: Inline form “Spline Trigger”

- Spline Stop Condition (conditional stop)

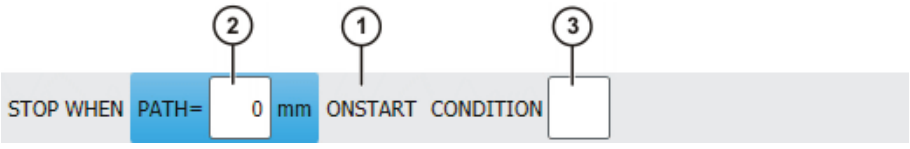


Fig. 8-18: Inline form “Spline Stop Condition”

8.5.1 Programming the spline trigger

Description

The Trigger triggers a user-defined statement. The robot controller executes the statement parallel to the robot motion.

The trigger can optionally refer to the start or end point of the motion. The statement can either be triggered directly at the reference point, or it can be shifted in space and/or time.

■The trigger cannot be used for PTP motions.
■If the trigger is used in a spline block, it must not be between the last segment and ENDSPLINE.

Programming options

- Inline form for option window “Logic Trigger”
- Inline form for spline trigger
- Programming using KRL command `TRIGGER WHEN PATH`

Option window “Logic Trigger”

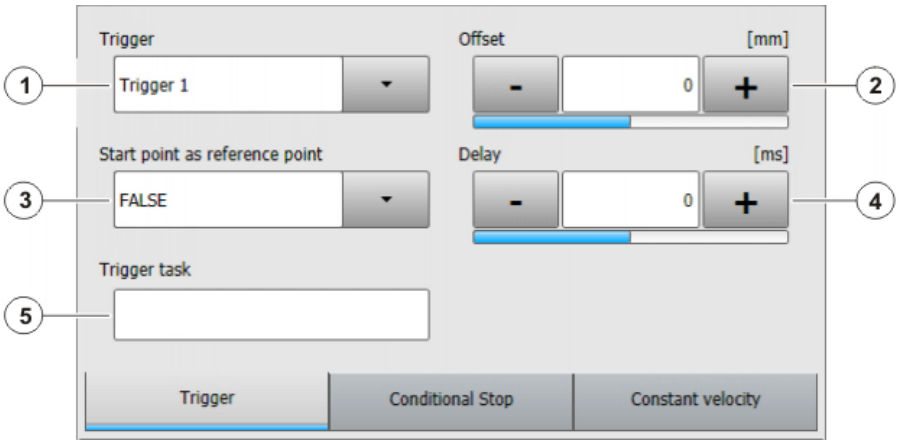


Fig. 8-19: Trigger

- A (further) trigger can be assigned to the motion by means of the button **Select action** > **Add trigger**. If it is the first trigger for this motion, this command also causes the **Trigger** box to be displayed.
- (A trigger can be removed again by means of **Select action** > **Remove trigger**.)
- Teach the spatial offset of the path via the button **Select action** > **Teach trigger path**.

Item	Description
1	A (further) trigger can be assigned to the motion by means of the button **Select action** > **Add trigger**. If it is the first trigger for this motion, this command also causes the **Trigger** box to be displayed. A maximum of 8 triggers per motion are possible. (A trigger can be removed again by means of **Select action** > **Remove trigger**.)
2	Reference point of the trigger ■ **TRUE**: Start point ■ **FALSE**: End point
3	Spatial offset relative to the end or start point ■ Negative value: Offset towards the start of the motion ■ Positive value: Offset towards the end of the motion The shift in space can also be taught. In this case, the box **Start point is reference point** is automatically set to **FALSE**.

Item	Description
4	Shift in time relative to **Offset** ■ Negative value: Shift towards the start of the motion. ■ Positive value: Trigger is switched after *Time* has elapsed.

5	Statement that is to be initiated by the trigger. The following are possible:
	■ **Assignment of a value to a variable**
	Note: There must be no runtime variable on the left-hand side of the assignment. In other words, the variable must be declared in a *.dat file.
	`value = 12`
	■ OUT statement; PULSE statement; CYCFLAG statement
	`$OUT[2]=TRUE ; Output is activated`
	`$OUT[99]=False ; Output is deactivated`
	■ Subprogram call. In this case, the priority must be specified.
	`UP()PRIO=-1 ; Caution: There must be no spaces in the entry.`
	Priorities 1, 2, 4 to 39 and 81 to 128 are available. Priorities 40 to 80 are reserved for cases in which the priority is automatically assigned by the system. If the priority is to be assigned automatically by the system, the following is programmed: `PRIO = -1`.
	If several triggers call subprograms at the same time, the trigger with the highest priority is processed first, then the triggers of lower priority. 1 = highest priority.

Inline form for spline trigger

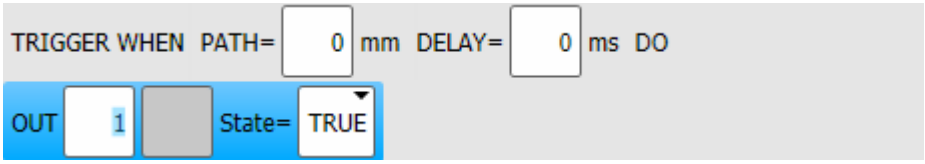


Fig. 8-20: Inline form “Spline Trigger”

Programming and function like the inline form SYN OUT PATH

Max. offset

The switching point can only be offset within certain limits. If larger, and thus invalid, offsets are programmed, the robot controller switches the trigger at the permissible limit at the latest. In T1/T2, it generates a corresponding message.

Maximum offset for *Distance* + negative *Time* value:

The limits apply to the entire offset, comprising shift in space and negative shift in time.

Maximum negative offset …	Maximum positive offset …
Up to start point (provided that this is not approximated)	Up to end point (provided that this is not approximated)
If the start point is an approximate positioning point: ■ If the start point is an approximated PTP point: Up to the end of the approximate positioning arc ■ If the start point is a different approximated point: Up to the start of the approximate positioning arc	If the end point is an approximate positioning point: ■ In the case of homogenous approximate positioning: up to the next exact positioning point after the TRIGGER statement ■ In the case of mixed approximate positioning (spline): up to the switching point an ON-START trigger with PATH = 0 would have if it were in the motion to which approximate positioning is being carried out. (>>> 8.5.1.3 "Reference point for mixed approximate positioning (spline)" Page 184) ■ In the case of mixed approximate positioning (LIN/CIRC/PTP): up to the start of the approximate positioning arc

Maximum offset for positive *Time* value:

The maximum positive shift in time is 1,000 ms. Any shift in time between 0 and 1,000 ms will be switched, even if the program has already been deselected in the meantime!

Example of trigger with KRL

```
1  SPTP P0
2  SPLINE
3   SPL P1
4   SPL P2
5   SPL P3
6   SPL P4
7   TRIGGER WHEN PATH=0 ONSTART DELAY=10 DO $OUT[5]=TRUE
8   SCIRC P5, P6
9   SPL P7
10  TRIGGER WHEN PATH=-20.0 DELAY=0 DO SUBPR_2() PRIO=-1
11  SLIN P8
12 ENDSPLINE
```

The Trigger in line 10 would have the same result if it was positioned directly before the spline block (i.e. between line 1 and line 2). In both cases, it refers to the last point of the spline motion: P8.

It is advisable, however, to position the trigger as shown in the example, and not directly before the spline block.

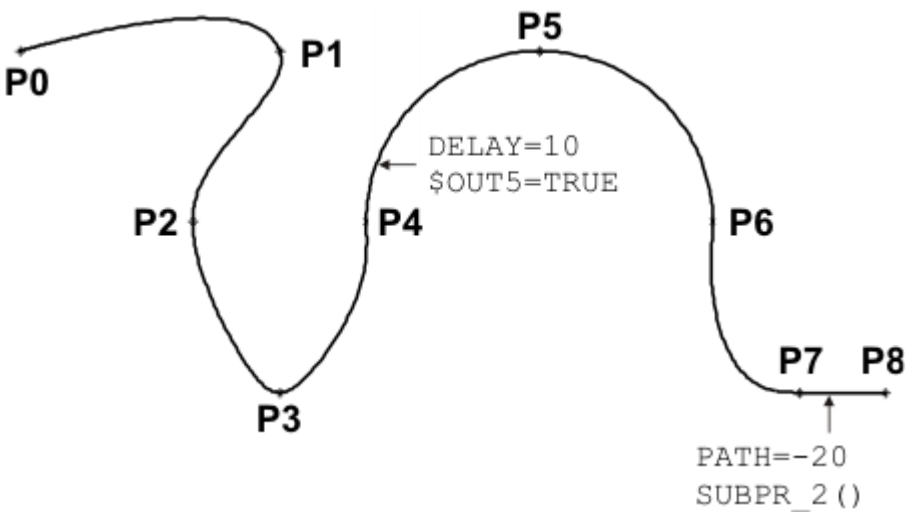


Fig. 8-21: Example of TRIGGER WHEN PATH (for spline)

8.5.1.1 Reference point for approximate positioning – overview

Where is the reference point of a PATH trigger if the start or end point is approximated?

This depends primarily on whether homogenous or mixed approximate positioning is being carried out.

Homogenous

Homogenous approximate positioning

- From a CP spline motion to a CP spline motion
- From a PTP spline motion to a PTP spline motion
- From a LIN or CIRC spline motion to a LIN or CIRC spline motion

Every spline motion can be a spline block or an individual instruction.

(>>> 8.5.1.2 "Reference point for homogenous approximate positioning" Page 182)

Mixed

Mixed approximate positioning

In this case, the position of the reference point also depends on whether the motions are spline motions or conventional motions.

- From a CP spline motion to a PTP spline motion or vice versa

  Every spline motion can be a spline block or an individual instruction.

  (>>> 8.5.1.3 "Reference point for mixed approximate positioning (spline)" Page 184)
- From a PTP spline motion to a LIN or CIRC spline motion or vice versa

  (>>> 8.5.1.4 "Reference point for mixed approximate positioning (LIN/CIRC/PTP)" Page 185)

8.5.1.2 Reference point for homogenous approximate positioning

The principle is explained here using an example with CP spline blocks. It also applies to other types of homogenous approximate positioning.

Example

```
SPLINE
  ...
  SLIN P2
  TRIGGER WHEN PATH=0 DELAY=0 DO ...  ;Trigger 1
  SLIN P3
```

```
ENDSPLINE C_SPL
SPLINE
  TRIGGER WHEN PATH=0 ONSTART DELAY=0 DO ...  ;Trigger 2
  SLIN P4
  ...
ENDSPLINE
```

Triggers 1 and 2 both refer to P3. P3 is approximated. The robot controller transfers the point onto the approximate positioning arc by a distance corresponding to the approximation distance (= P3').

End point approximated

Reference point of Trigger 1:

The robot controller calculates how far the distance would be from the start of the approximate positioning arc to the end point with exact positioning. This distance is then applied to the approximate positioning arc.

The distance $P_{StartApprox} \rightarrow P3$ is the same as $P_{StartApprox} \rightarrow P3'_{Trigger\ 1}$.

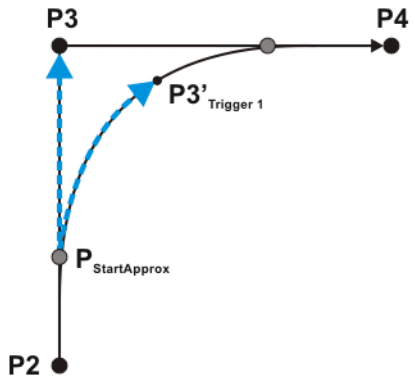


Fig. 8-22: Trigger 1: Reference point for homogenous approximate positioning

$P_{StartApprox}$	Start of the approximate positioning arc
P3	Reference point for exact positioning
$P3'_{Trigger\ 1}$	Reference point for approximate positioning

Start point approximated

Reference point of Trigger 2:

The robot controller calculates how far the distance would be from the end of the approximate positioning arc back to the start point with exact positioning. This distance is then applied to the approximate positioning arc.

The distance $P_{EndApprox} \rightarrow P3$ is the same as $P_{EndApprox} \rightarrow P3'_{Trigger\ 2}$.

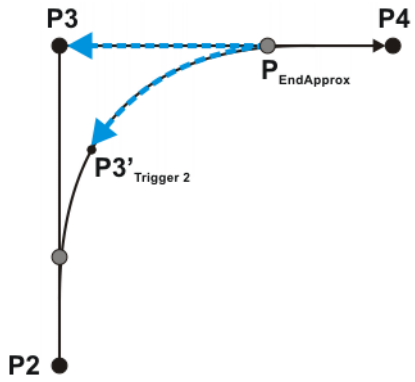


Fig. 8-23: Trigger 2: Reference point for homogenous approximate positioning

$P_{EndApprox}$	End of the approximate positioning arc
P3	Reference point for exact positioning
$P3'_{Trigger\ 2}$	Reference point for approximate positioning

8.5.1.3 Reference point for mixed approximate positioning (spline)

Example

```
PTP_SPLINE
  ...
  SPTP P2
  TRIGGER WHEN PATH=0 DELAY=0 DO ...  ;Trigger 1
  SPTP P3
ENDSPLINE C_SPL

SPLINE
  TRIGGER WHEN PATH=0 ONSTART DELAY=0 DO ...  ;Trigger 2
  SLIN P4
  ...
ENDSPLINE
```

Triggers 1 and 2 both refer to P3. P3 is approximated.

Start point approximated

Reference point of Trigger 2:

> This reference point must be considered first, as the reference point of Trigger 1 refers to it!

The reference point is determined in the same way as for homogenous approximate positioning.

(>>> "Start point approximated" Page 183)

End point approximated

Reference point of Trigger 1:

The reference point for Trigger 1 is in the same position as that for Trigger 2.

The distance $P_{StartApprox}$ → $P3'_{Trigger\ 1}$ is generally shorter than $P_{StartApprox}$ → P3.

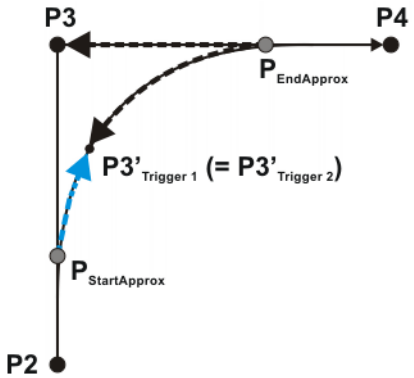


Fig. 8-24: Trigger 1: Reference point for mixed approximate positioning

$P_{StartApprox}$	Start of the approximate positioning arc
$P_{EndApprox}$	End of the approximate positioning arc
P3	Reference point for exact positioning
P3'	Reference point for approximate positioning

If Trigger 1 were to be shifted to between $P_{StartApprox}$ and P3', the exact position would be determined as follows:

The robot controller calculates the percentage of the distance **$P_{StartApprox}$ → P3** at which the switching point would be located if the end point were an exact positioning point. This proportion is then applied to the approximate positioning arc. The switching point is thus at x% of the distance **$P_{StartApprox}$ → $P3'_{Trigger\ 1}$**

8.5.1.4 Reference point for mixed approximate positioning (LIN/CIRC/PTP)

Start point approximated

PTP-CP approximate positioning:

The reference point is at the end of the approximate positioning arc.

End point approximated

CP-PTP approximate positioning:

The reference point is at the start of the approximate positioning arc.

8.5.1.5 Constraints for functions in the trigger

The values for `DELAY` and `PATH` can be assigned using functions. The following constraints apply to these functions:

- The KRL program containing the function must have the attribute **Hidden**.
- The function must be globally valid.
- The functions may only contain the following statements or elements:
  - Value assignments
  - IF statements
  - Comments
  - Blank lines
  - RETURN
  - Read system variable
  - Call predefined KRL function

8.5.2 Programming a conditional stop

Description

The “conditional stop” enables the user to define a point on the path at which the robot stops if a certain condition is met. This point is called the “stop point”. As soon as the condition is no longer met, the robot resumes its motion.

During the runtime, the robot controller calculates the latest point at which the robot must brake in order to be able to stop at the stop point. From this point (braking point) onwards, it monitors whether or not the condition is met.

- If the condition is met at the braking point, the robot brakes in order to stop at the stop point.

  If, however, the condition then switches back to “not met” before the stop point is reached, the robot accelerates again and does not stop.
- If the condition is not met at the braking point, the robot motion is continued without braking.

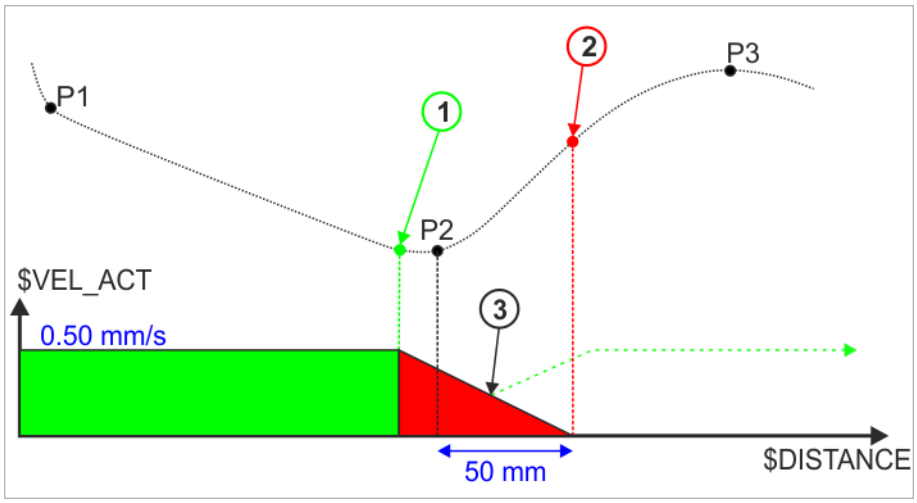


Fig. 8-25: Spline with conditional stop

Item	Explanation
1	Calculated braking point dependent on distance and velocity
2	Specified stop point (defined in inline form or KRL)
3	Possible withdrawal of Boolean expression -> acceleration to programmed velocity

Essentially, any number of conditional stops can be programmed. A maximum of 10 “braking point → stop point” paths may overlap, however.

While the robot is braking, the robot controller displays the following message in T1/T2 mode: *Conditional stop active (line {Line number}).*

Programming options

- In the spline block (CP and PTP) or in the individual spline block: Inline form for option window “Logic Conditional Stop”
- Before a spline block (CP and PTP): Inline form “Spline Conditional Stop”
- Programming using KRL command `STOP WHEN PATH`

Option window “Logic Conditional Stop”

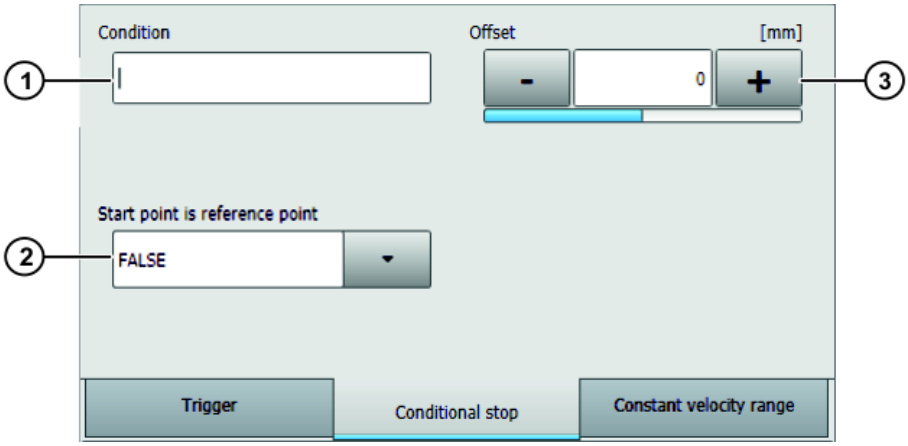


Fig. 8-26: Conditional stop

Item	Description
1	Stop condition. The following are permitted: ■ a global Boolean variable ■ a signal name ■ a comparison ■ a simple logic operation: NOT, OR, AND or EXOR
2	The conditional stop can refer to either the start point or the end point of the motion. ■ **TRUE**: Start point ■ **FALSE**: End point If the reference point is approximated, the same rules apply as for the PATH trigger.
3	The stop point can be shifted in space. For this, the desired distance from the start or end point must be specified. If no shift in space is desired, enter “0”. ■ Positive value: Offset towards the end of the motion ■ Negative value: Offset towards the start of the motion There are limits to the distance the stop point can be offset. The same limits apply as for the PATH trigger. The shift in space can also be taught. In this case, the box **Start point is reference point** is automatically set to **FALSE**.

- Teach the spatial offset of the stop point via the button **Select action** > **Teach conditional stop path**.

Inline form “Spline Stop Condition”

This inline form may only be used before a spline block. There may be other statements between the inline form and the spline block, but no motion instructions.

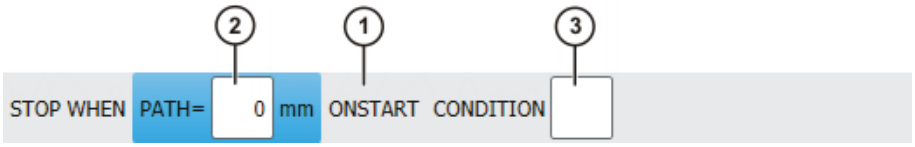


Fig. 8-27: Inline form “Spline Stop Condition”

Item	Description
1	Point to which the conditional stop refers ■ With ONSTART: last point before the spline block ■ Without ONSTART: last point in the spline block If the spline is approximated, the same rules apply as for the PATH trigger. **Note:** Information about approximate positioning with the PATH trigger is contained in the Operating and Programming Instructions for System Integrators. ONSTART can be set or removed using the **Toggle OnStart** button.
2	The stop point can be shifted in space. For this, the desired distance from the reference point must be specified. If no shift in space is desired, enter “0”. ■ Positive value: Offset towards the end of the motion ■ Negative value: Offset towards the start of the motion There are limits to the distance the stop point can be offset. The same limits apply as for the PATH trigger. The shift in space can also be taught.
3	Stop condition. The following are permitted: ■ a global Boolean variable ■ a signal name ■ a comparison ■ a simple logic operation: NOT, OR, AND or EXOR

Teach path

Button	Description
Teach path	If an offset is desired, it is not necessary to enter the value into the inline form numerically; the offset can also be taught. This is done via **Teach path**. If an offset is taught, ONSTART is automatically removed, if set in the inline form, as the taught distance always refers to the end point of the motion. The teaching sequence is the same as that for the option window **Logic parameters**.

8.5.3 Programming the constant velocity range

Description

In a CP spline block, a range can be defined in which the robot maintains the programmed velocity constant where possible. This range is called the “constant velocity range”.

- 1 constant velocity range can be defined per CP spline block.
- A constant velocity range is defined by a start statement and an end statement.
- The range cannot extend beyond the spline block.
- There is no lower limit to the size of the range.

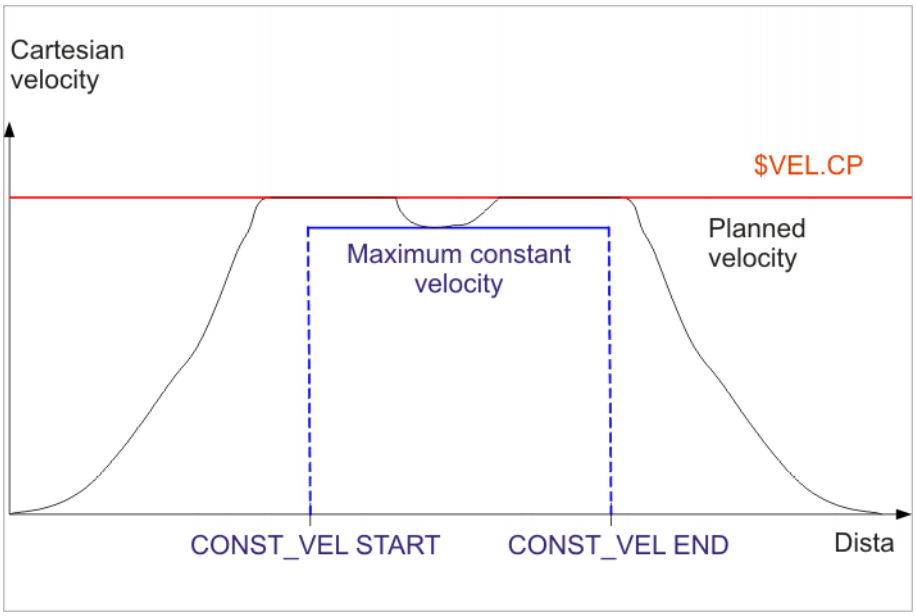


Fig. 8-28: Spline constant velocity range

If it is not possible to maintain the programmed velocity constant, the robot controller indicates this by means of a message during program execution.

Constant velocity range over several segments:

A constant velocity range can extend over several segments with different programmed velocities. In this case, the lowest of the velocities is valid for the whole range.

Even in the segments with a higher programmed velocity, the motion is executed with the lowest velocity in this case. No message is generated indicating that the velocity has not been maintained. This only occurs if the lowest velocity cannot be maintained.

Programming options

- Inline form for option window “Logic Constant Velocity Range”: the start or end of the range is stored in the corresponding CP segment.
- Programming via KRL command `CONST_VEL START` and `CONST_VEL END`

Option window “Logic Constant Velocity Range”

Constant velocity range is only available for CP spline segments.

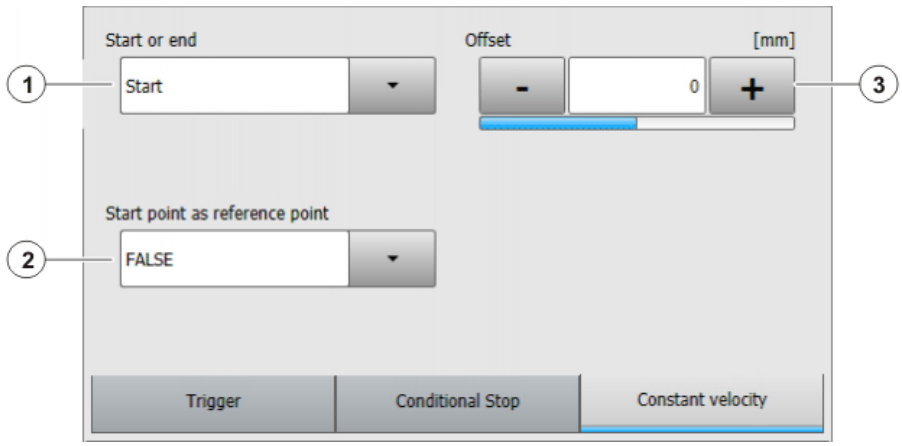


Fig. 8-29: Constant velocity range

Item	Description
1	■ **Start**: Defines the start of the constant velocity range. ■ **End**: Defines the end of the constant velocity range.
2	**Start** and **End** can refer to either the start point or the end point of the motion. ■ **TRUE**: **Start** or **End** refers to the start point. If the start point is approximated, the reference point is generated in the same way as for homogenous approximate positioning with the PATH trigger. ■ **FALSE**: **Start** or **End** refers to the end point. If the end point is approximated, **Start** or **End** refers to the start of the approximate positioning arc.
3	The start or end of the constant velocity range can be shifted in space. For this, the desired distance must be specified. If no shift in space is desired, enter “0”. ■ Positive value: Offset towards the end of the motion ■ Negative value: Offset towards the start of the motion The shift in space can also be taught. In this case, the box **Start point is reference point** is automatically set to **FALSE**.

- Teach the spatial offset of the constant range via the button **Select action** > **Teach constant velocity range path**.

8.5.3.1 Block selection to the constant velocity range

Description

If a block selection to a constant velocity range is carried out, the robot controller ignores it and generates a corresponding message. The motions are executed as if no constant velocity range were programmed.

A block selection to the path section defined by the offset values is considered as a block selection to the constant velocity range. The motion blocks in which the start and end of the range are programmed, however, are irrelevant.

Example

```
SPLINE S1 Vel=2 m/s CPDAT1 Tool[1] Base[0]

SLIN P1 ADAT1

   CONST_VEL START = 50

   SLIN XP1

SLIN P2

SLIN P3

SLIN P4 ADAT2

   CONST_VEL END = -50 ONSTART

   SLIN XP4

SLIN P5

ENDSPLINE
```

Fig. 8-30: Constant velocity range example (inline programming)

The folds in the program are expanded. The indentations are not present by default and have been inserted here for greater clarity.

The start of the constant velocity range is at P1 in the program. The end is at P4. What counts, however, when deciding what constitutes a block selection to the constant velocity range is where the range is located on the path:

Fig. 8-31: Constant velocity range example (path)

What is considered as a block selection to the constant velocity range?

Block selection to point ...	P1	P2	P3	P4
= in constant velocity range?	No	No	Yes	No

8.5.3.2 Maximum limits

If the start or end point of the spline block is an exact positioning point:

- The constant velocity range starts at the start point at the earliest.
- The constant velocity range ends at the end point at the latest.

If the offset value is such that these limits would be exceeded, the robot controller automatically reduces the offset and generates the following message: *CONST_VEL {Start/End} = {Offset} cannot be implemented; {New offset} will be used.*

The robot controller reduces the offset far enough to create a range in which the constant programmed velocity can be maintained. In other words, it does

not necessarily shift the limit exactly to the start or end point of the spline block, but possibly further inwards.

The same message is generated if the range is already in the spline block beforehand, but the defined velocity cannot be maintained due to the offset. In this case, once again, the robot controller reduces the offset.

If the start or end point of the spline block is approximated:

- The constant velocity range starts at the start of the approximate positioning arc of the start point at the earliest.
- The constant velocity range ends at the start of the approximate positioning arc of the end point at the latest.

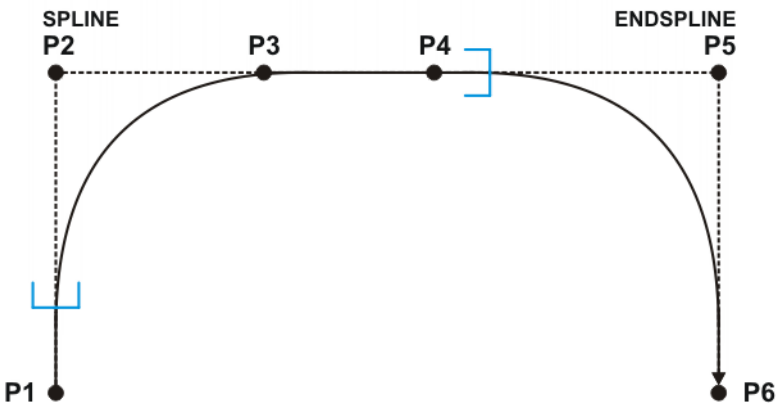


Fig. 8-32: Maximum limits for approximated SPLINE/ENDSPLINE

If the offset is such that these limits would be exceeded, the robot controller automatically sets the limit to the start of the corresponding approximate positioning arc. It does not generate a message.

8.6 Exercise: Motion programming with spline

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Use logic commands in the spline inline form

Preconditions

The following are preconditions for successful completion of this exercise:

- Knowledge of motion programming with splines.
- Use of time-distance functions in the spline inline form

Task description

Carry out the following tasks:

1. Duplicate your program with the spline contour.
2. Activate output 15 ($OUT[15]) in the red area.
3. Activate output 16 ($OUT[16]) in the green area.

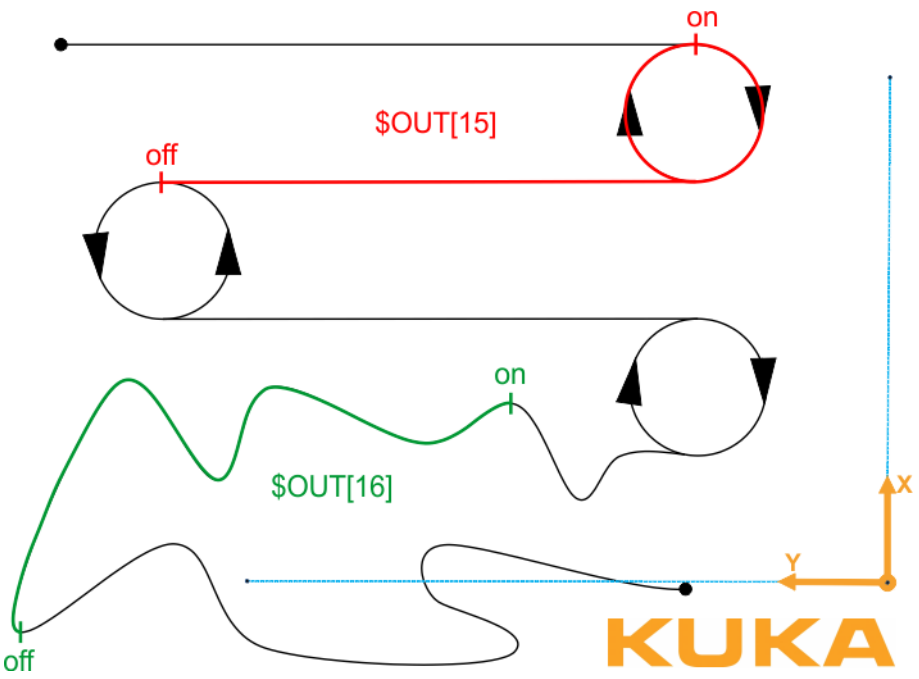


Fig. 8-33: Logic with spline

4. Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. What must be taken into consideration when using the WAIT function?

. . . . . .

. . . . . .

. . . . . .

2. What is achieved by selecting CONT in the WAITFOR inline form?

. . . . . .

. . . . . .

. . . . . .

3. What is the difference between an OUT and a PULSE?

. . . . . .

. . . . . .

4. What is meant by a trigger?

. . . . . .

. . . . . .

9 Using technology packages

9.1 Overview

The following contents are explained in this training module:

- KUKA.GripperTech operation
- KUKA.GripperTech programming
- KUKA.GripperTech configuration

9.2 Gripper operation with KUKA.GripperTech

Technology package KUKA.GripperTech

KUKA.Gripper&SpotTech is an add-on technology package. It simplifies the use of a gripper in terms of:

- **Operator control**
  Switch between different gripper states manually by means of status keys.
- **Programming**
  Simple programming of gripper commands using predefined inline forms.
- **Configuration**
  Gripper setup using five predefined gripper types or a user-defined gripper type.

The following status keys are required for operating the gripper:

Status key	Description
	Select gripper. The number of the gripper is displayed. ■ Pressing the upper key counts upwards. ■ Pressing the lower key counts downwards.
	Toggle between the gripper states (e.g. open or close). The current state is not displayed. The possible states depend on the configured gripper type. In the case of weld guns, the possible states depend on the configuration of the manual gun control.

Procedure for gripper operation

NOTICE Before a gripper can be operated using the status keys, the status keys must first be activated!
In the main menu, select **Configuration** > **Status keys** > **GripperTech**.

⚠ WARNING **Warning!**
When using the gripper system there is a risk of crushing and cutting. Personnel using the gripper must ensure that no part of the body can be crushed by the gripper.

1. Select the gripper using the status key.
   

2. Activate operating mode T1 or T2.

3. Press enabling switch.
4. Control the gripper using the status key.

9.3 Gripper programming with KUKA.GripperTech

Programming gripper commands

The technology package KUKA.GripperTech allows the programming of gripper commands directly in the selected program using ready-made inline forms. Two commands are available for this:

- **SET Gripper** | Command for opening/closing the gripper in the program
- **CHECK Gripper** | Command for checking the gripper state

Gripper programming functions

Gripper command during motion

- It is generally possible to program the gripper command so that it is executed relative to the start or end point.
- To do so, simply activate the entry **CONT** in the inline form and specify the duration of the delay in ms (**Delay**).
-

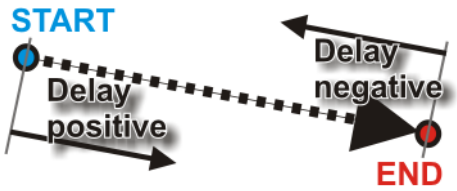


Fig. 9-1: Schematic diagram of delay

⚠ WARNING Gripper commands to be processed during the motion must be selected carefully, as careless use can result in injuries and material damage due to flying parts or collisions!

Gripper settings for exact positioning

-

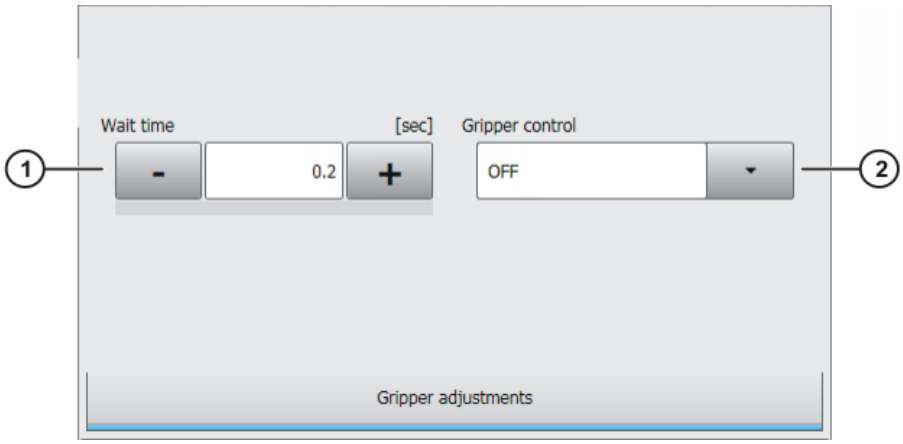


Fig. 9-2: Gripper adjustments

- Use gripper monitoring:
  - If gripper monitoring is activated with **ON**, the parameterized sensor systems are polled.
  - If there is no feedback from the sensors, a timeout error occurs and the sensor can be simulated in test mode.

- If gripper monitoring is deactivated (OFF), the system waits for the parameterized wait time before the program is resumed.

Procedure for gripper programming

Procedure

1. Select the menu sequence **Commands** > **GripperTech** > **Gripper**.
2. Set the parameters in the inline form.
3. Save with **Cmd Ok**.

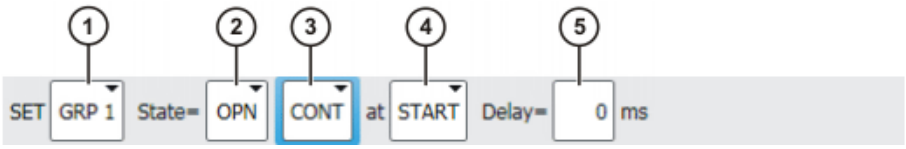


Fig. 9-3: Inline form for gripper with approximate positioning

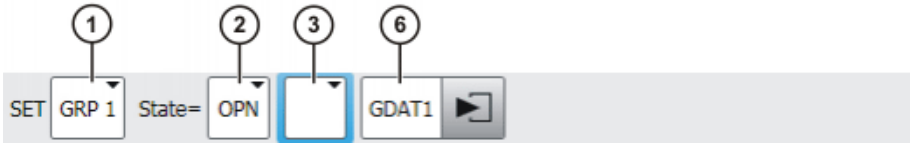


Fig. 9-4: Inline form for gripper without approximate positioning

Item	Description
1	Select gripper. ■ All configured grippers are available for selection.
2	Select the switching state of the gripper. ■ The number depends on the gripper type. ■ The designation depends on the configuration.
3	Execution in the advance run. ■ **CONT**: Execution during motion ■ [*blank*]: Execution with advance run stop.
4	Box only available if **CONT** selected. ■ **START**: The gripper action is executed at the start point of the motion. ■ **END**: The gripper action is executed at the end point of the motion.
5	Box only available if **CONT** selected. Define a wait time, relative to the start or end point of the motion, for execution of the gripper action. ■ -200 ... 200 ms
6	Data set with gripper parameters

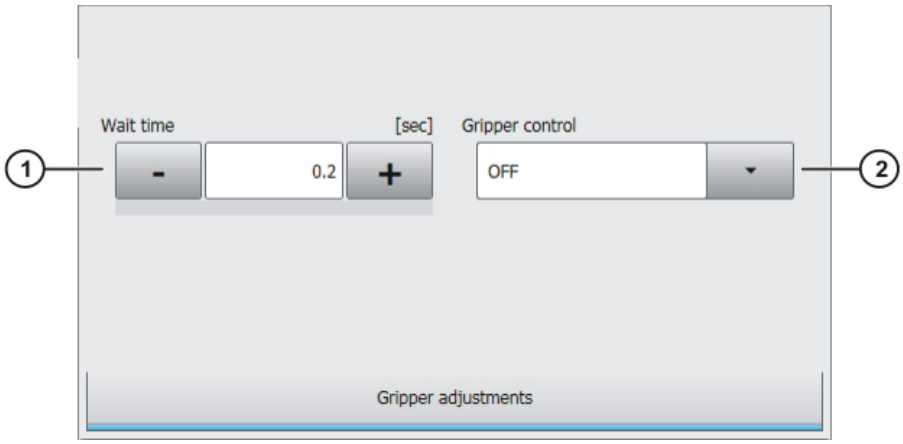


Fig. 9-5: Gripper adjustments

Item	Description
1	Wait time after which the programmed motion is resumed. ■ 0 … 10 s
2	Gripper control ■ **OFF** (default): Wait for the wait time set above ■ **ON**: Wait for the sensors

Procedure for programming gripper monitoring

Procedure

1. Select the menu sequence **Commands** > **GripperTech** > **Check Gripper**.
2. Set the parameters in the inline form.
3. Save with **Cmd Ok**.

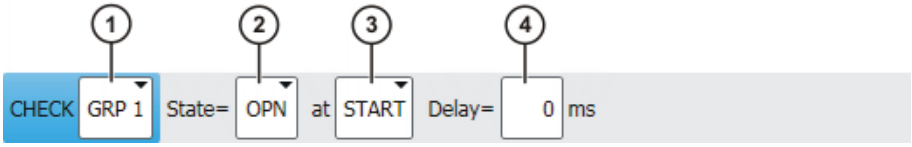


Fig. 9-6: Inline form “Check Gripper”

Item	Description
1	Select gripper. ■ All configured grippers are available for selection.
2	Select the switching state of the gripper. ■ The number depends on the gripper type. ■ The designation depends on the configuration.
3	Select the time at which the sensor interrogation is executed. ■ **START**: The sensor interrogation is executed at the start point of the motion. ■ **END**: The sensor interrogation is executed at the end point of the motion.
4	Define a wait time, relative to the start or end point of the motion, for execution of the sensor interrogation.

9.4 KUKA.GripperTech configuration

Configuration options and gripper types

KUKA.GripperTech enables simple gripper configuration. For this, five predefined gripper types are available for selection. Additionally, user-defined grippers can also be configured.

NOTICE Up to 16 different grippers can be configured on the controller.

Gripper types

Type	OUT	IN	States	Example
Type 1	2	4	2	Simple gripper with the functions OPEN and CLOSE
Type 2	2	2	3	Slide with center position
Type 3	2	2	3	Vacuum gripper with the functions SUCTION, RELEASE and OFF
Type 4	3	2	3	The same as type 3 but with three control outputs
Type 5	2	4	2	The same as type 1 but with a pulse signal instead of a continuous signal
free	Freely configurable			

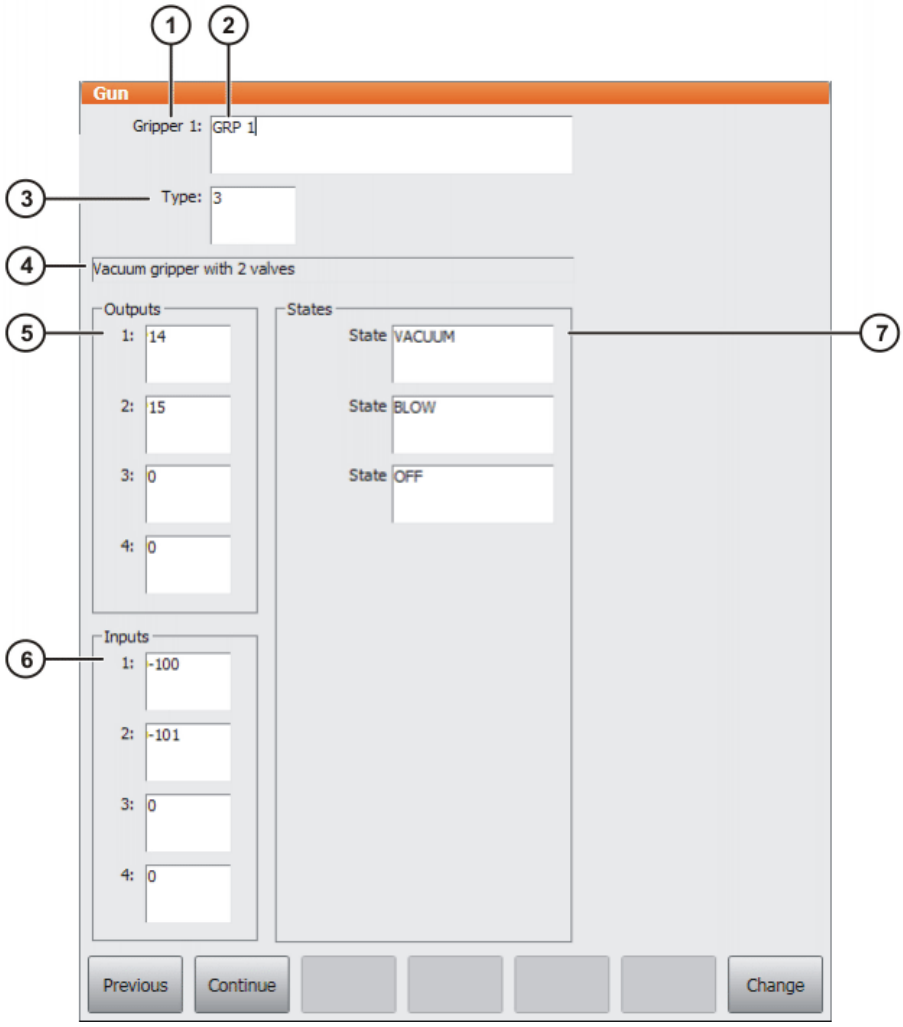


Fig. 9-7: Example: predefined gripper

Item	Description
1	Number of the gripper ■ 1 … 16
2	Name of the gripper The name is displayed in the inline form. The default name can be changed. ■ 1 … 24 characters
3	Type ■ For predefined grippers: 1 … 5 (See “Gripper types” table)
4	Designation of the gripper type (not updated until saved) The designation cannot be changed.
5	Assignment of the output numbers Outputs that are not required can be assigned the value “0”. In this way, they are immediately identifiable as unused. If they are nonetheless assigned a number, this has no effect.

Item	Description
6	Assignment of the input numbers Inputs that are not required can be assigned the value “0”. In this way, they are immediately identifiable as unused. If they are nonetheless assigned a number, this has no effect.
7	Switching states The default names can be changed. The names are displayed in the inline forms if the corresponding gripper is selected there.

Free gripper types

A freely programmable gripper type has been integrated in order to cover all user requirements. Any number of completely freely definable grippers can be configured by means of entries in the files $CONFIG.DAT, USERGRP.DAT and USER_GRP.SRC.

NOTICE More detailed information about the configuration of grippers can be found in the operating instructions for KUKA System Technology KUKA.Gripper&SpotTech.

Procedure for gripper configuration

Configuration with predefined gripper type

1. In the main menu, select **Configure** > **I/O** > **Gripper**. A window opens.
2. Select the desired gripper number with **Next** or **Previous**.
3. If desired, change the default name of the gripper.
4. Assign one of the types 1 to 5 to the gripper.
5. Assign the inputs and outputs.
6. If desired, change the default names of the states.
7. Save the configuration with **Change**.

9.5 Exercise: Gripper programming – plastic panel

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Program statements for controlling and monitoring grippers and weld guns (KUKA.Gripper & SpotTech)
- Activate and work with the technology-specific status keys

Preconditions

The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of the KUKA.Gripper & SpotTech technology package

Task description

Carry out the following tasks: fetch and set down plastic panel

1. Create a new program with the name **Fetch_panel**, using the gripper tool and the blue base.
2. Teach the *Fetch_panel* process with the setdown and pick-up positions illustrated in the figure (>>> Fig. 9-8). To do so, reduce the velocity to 0.3 m/s for removing the panel and for setting it back down.
3. Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.
4. Create a second program with the name **Set_down_panel**. Use a suitable base and the corresponding tool.
5. Teach the procedure “Set_down_panel”.
6. Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.
7. Archive your programs.

Fig. 9-8: Panel with setdown position

1 Panel 2 Setdown position

What you should now know:

1. Which inline forms are available with the **KUKA.GripperTech** technology package? Describe their functions.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

2. What is the effect of the wait time in the *gripper settings*?

. . . . . .

. . . . . .

. . . . . .

10 Configuration and programming of external tools

10.1 Overview

The following contents are explained in this training module:

- Jogging with a fixed tool
- Starting up the robot
  - Calibration of a fixed tool
  - Calibration of a robot-guided workpiece
- Motion programming with an external TCP

10.2 Moving the robot

10.2.1 Jogging with a fixed tool

Advantages and areas of application

Some production and machining processes require the robot to handle the **workpiece** and not the **tool**. The advantage is that it is not necessary to set the workpiece down first before it can be machined – thus saving on clamping fixtures. This is the case, for example, for:

- Adhesive bonding applications
- Welding applications
- etc.

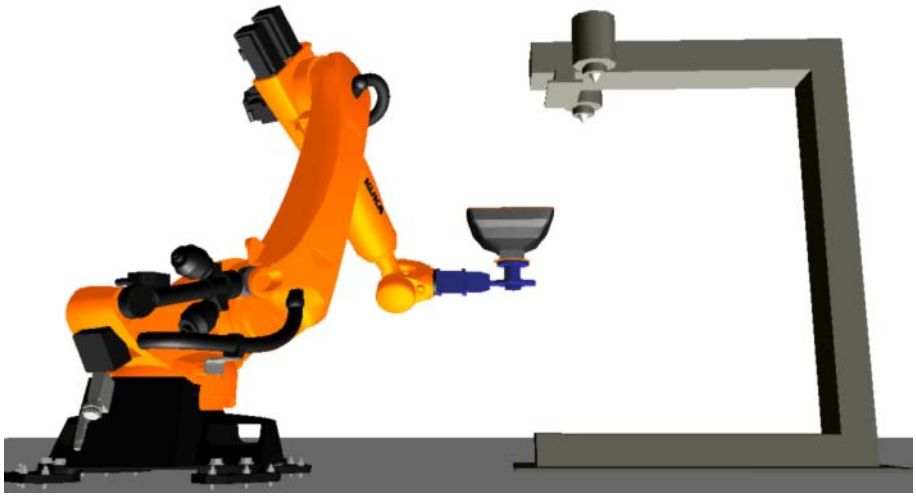


Fig. 10-1: Example of a fixed tool

NOTICE In order to program such an application successfully, both the external TCP of the fixed tool and the workpiece must be calibrated.

Modified motion sequence with fixed tool

Although the tool is a fixed (non-mobile) object, it nonetheless has a tool reference point with an associated coordinate system. In this case, the reference point is called the **external TCP**. Since it is a non-mobile coordinate system, the data are managed in the same way as a base coordinate system and correspondingly saved as **Base**!

The (mobile) **workpiece**, on the other hand, is saved as **Tool**. This means that motion along the workpiece edges relative to the TCP is possible!

NOTICE It must be taken into consideration that the motions during jogging with a fixed tool are relative to the external TCP!

Procedure for jogging with a fixed tool

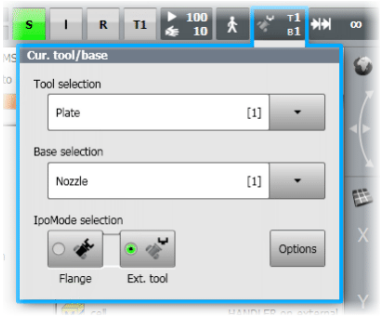


Fig. 10-2: Selecting the external TCP in the Options menu

1. Select the robot-guided workpiece in the tool selection window.
2. Select the fixed tool in the base selection window.
3. Set IpoMode selection to **External tool**.
4. Set Tool as the option for the jog keys/Space Mouse:
   - Set tool in order to be able to jog in the coordinate system of the workpiece.
   - Set base in order to be able to jog in the coordinate system of the external tool.
5. Set jog override.
6. Press the enabling switch into the center position and hold it down.
7. Move in the desired direction using the jog keys/Space Mouse.

Selecting **Ext. tool** in the option window **Jog options** switches the controller: all motions are now carried out relative to the external TCP and not to a robot-guided tool.

10.2.2 Exercise: Jogging with a fixed tool

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Jog a robot guiding a workpiece relative to a fixed tool

Preconditions

The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of the general operator control of a KUKA industrial robot system
- Theoretical knowledge of jogging with an external tool

Task description

1. Set the tool coordinate system *D_Panel*.
2. Set the base coordinate system *D_Glue Nozzle*.
3. In the option window “Jog options”, set *Ext. tool*. (>>> Fig. 10-2)
4. Move the *panel* to the *external pen*.
5. Move and orient the *panel* at the *external pen*. Test the differences between jogging in the tool and base coordinate systems.
6. In the option window “Jog options”, set *Flange*.
7. Move and orient the *panel* at the *external pen*. (>>> Fig. 10-2)

10.3 Starting up the robot

10.3.1 Calibration of a fixed tool

Overview

Calibration of the fixed tool consists of two steps:

1. Determination of the position of the external TCP relative to the origin of the world coordinate system.
2. Orientation of the coordinate system at the external TCP.

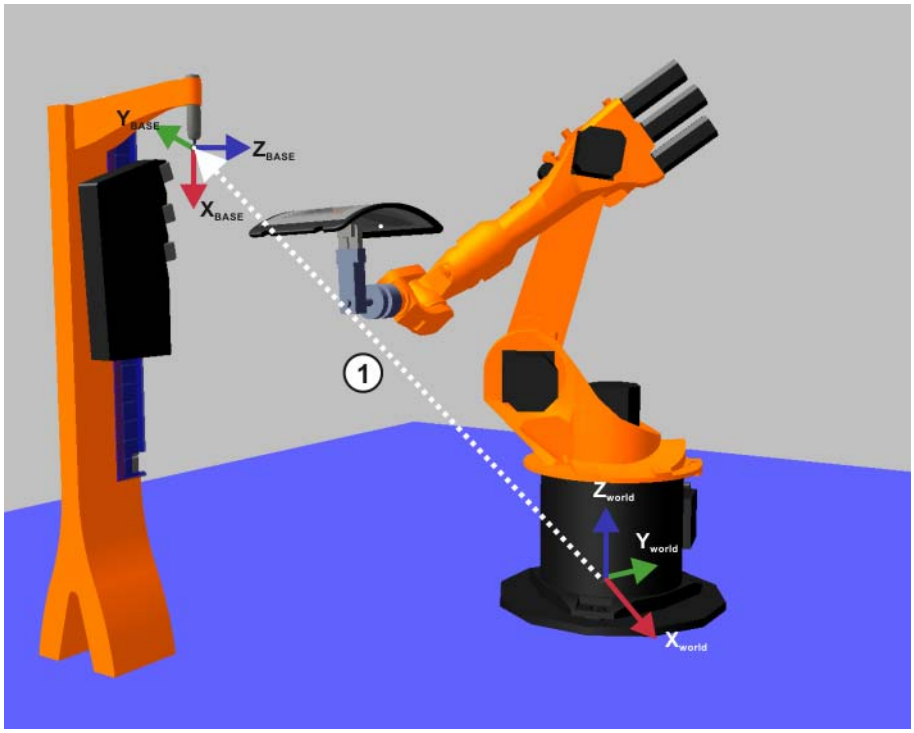


Fig. 10-3: Calibration of the fixed tool

As illustrated in (1) (>>> Fig. 10-3), the external TCP is managed relative to $WORLD (or $ROBROOT), i.e. in the same way as a base coordinate system.

Description of calibration

1. **Determination of the external TCP**
   A calibrated, robot-guided tool is required for calibration of the external TCP. Its tool tip is moved to the external TCP.

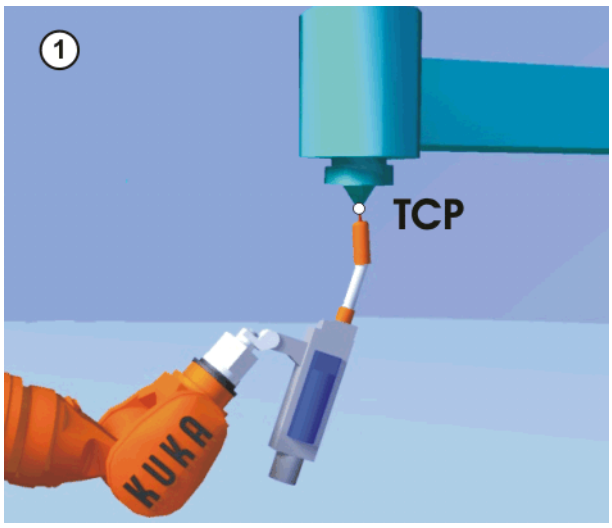


Fig. 10-4: Moving to the external TCP

2. **Determination of the orientation**

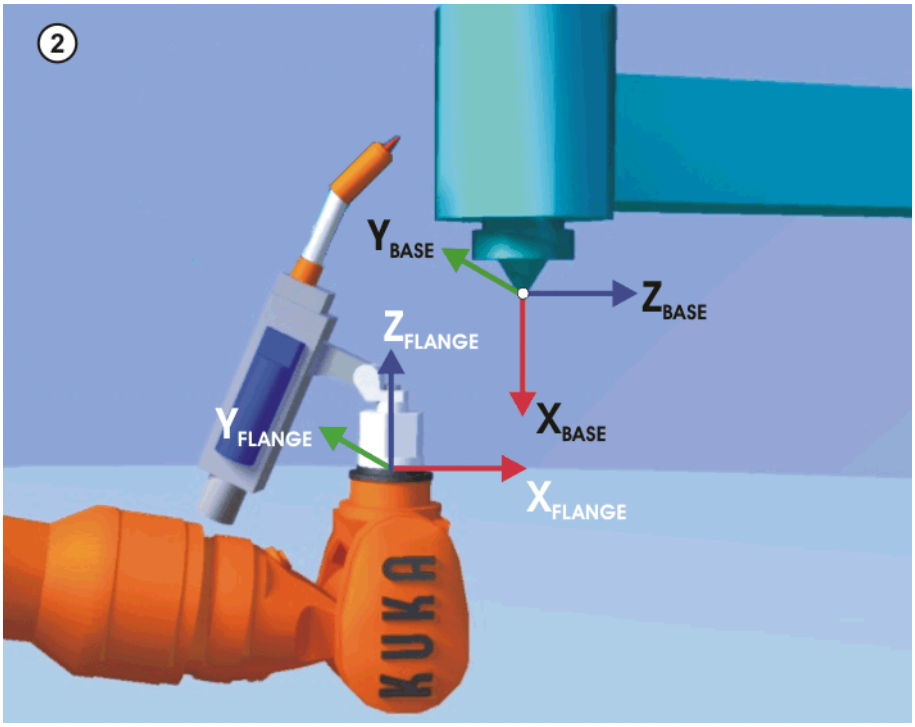


Fig. 10-5: Aligning the coordinate systems parallel to one another

To determine the orientation, the flange coordinate system is aligned parallel to the new coordinate system.

There are 2 variants:

- **5D**: Only the tool direction of the fixed tool is communicated to the robot controller. By default, the tool direction is the X axis. The orientation of the other axes is defined by the system and cannot be detected easily by the user.
- **6D**: The orientation of all 3 axes is communicated to the robot controller.

Procedure

1. In the main menu, select **Start-up** > **Calibrate** > **Fixed tool** > **Tool**.
2. Assign a number and a name for the fixed tool. Confirm with **Next**.
3. Enter the number of the reference tool used. Confirm with **Next**.
4. Select a variant in the box **5D/6D**. Confirm with **Next**.
5. Move the TCP of the calibrated tool to the TCP of the fixed tool. Press **Calibrate**. Confirm position with **Yes**.
6. If **5D** is selected:

   Align $+X_{BASE}$ parallel to $-Z_{FLANGE}$.

   (i.e. align the mounting flange perpendicular to the tool direction of the fixed tool.)

   If **6D** is selected:

   Align the mounting flange so that its axes are parallel to the axes of the fixed tool:

   - $+X_{BASE}$ parallel to $-Z_{FLANGE}$

     (i.e. align the mounting flange perpendicular to the tool direction.)
   - $+Y_{BASE}$ parallel to $+Y_{FLANGE}$
   - $+Z_{BASE}$ parallel to $+X_{FLANGE}$
7. Press **Calibrate**. Confirm position with **Yes**.
8. Press **Save**.

10.3.2 Calibration of a robot-guided workpiece

Overview: Direct calibration

NOTICE Only the direct calibration method is explained here. Indirect calibration is extremely rare and is described in greater detail in the documentation *KUKA System Software 8.2 – Operating and Programming Instructions*.

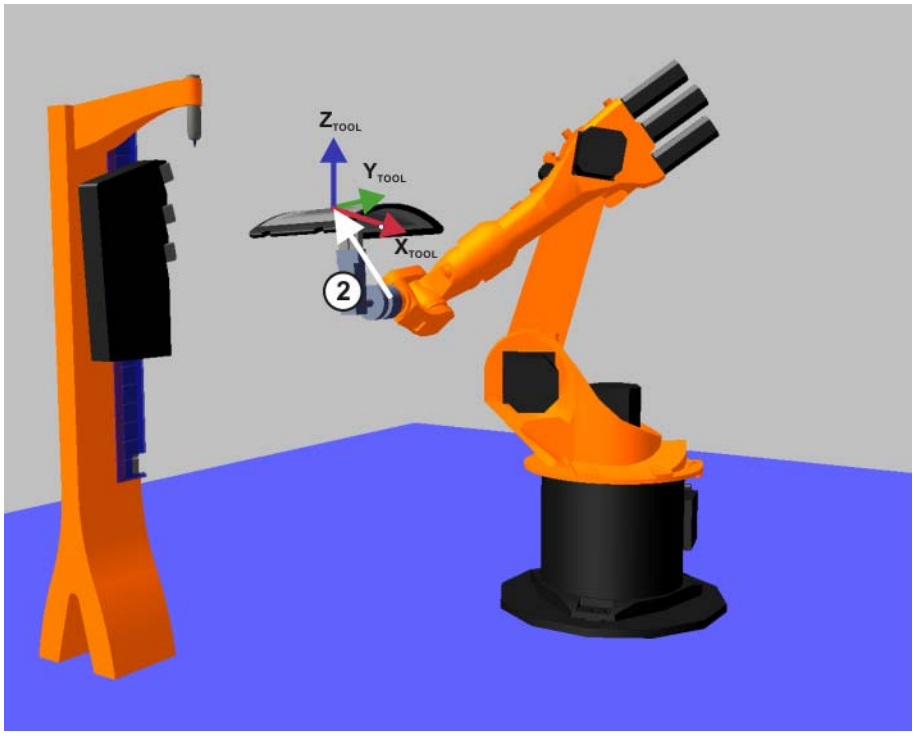


Fig. 10-6: Workpiece calibration by means of direct calibration

Part	Calibration
2	Calibration of the workpiece

Description

The origin and 2 further points of the workpiece are communicated to the robot controller. These 3 points uniquely define the workpiece.

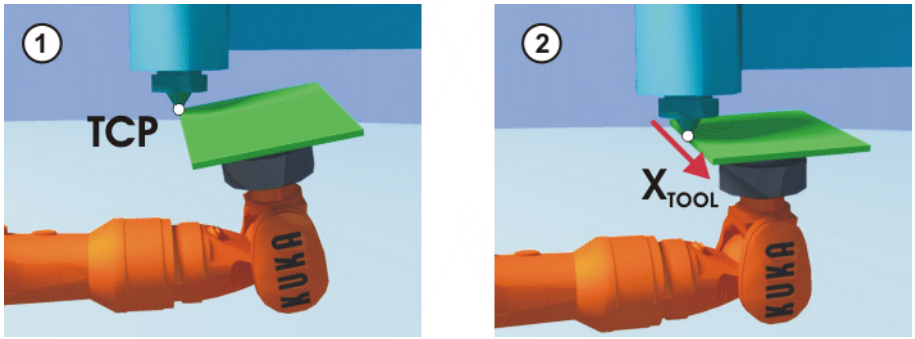


Fig. 10-7

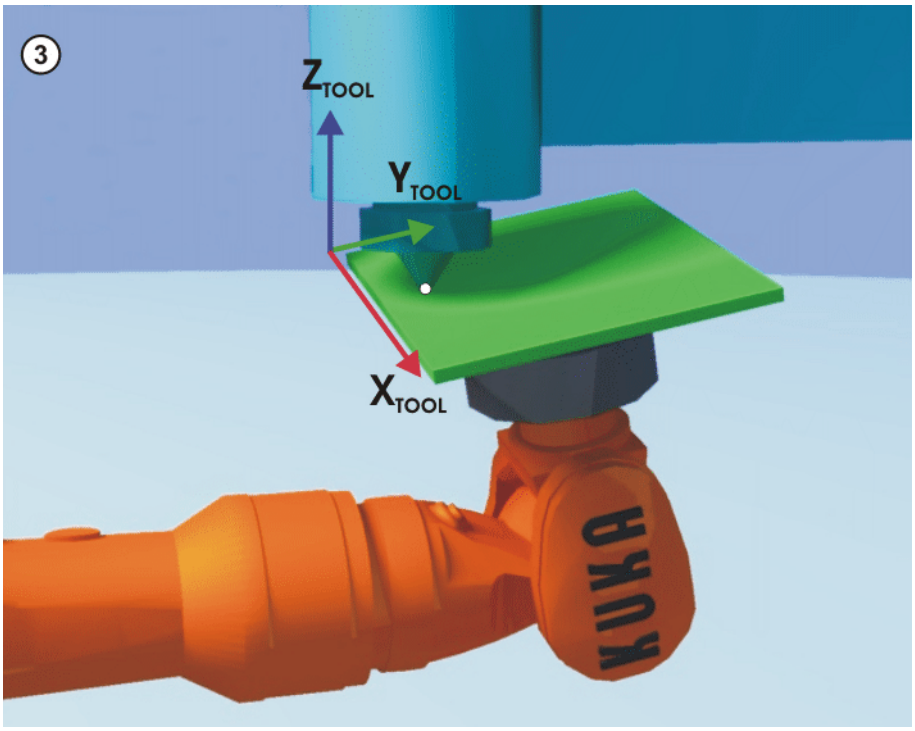


Fig. 10-8: Workpiece calibration: direct method

Procedure

1. Select the menu **Start-up** > **Calibrate** > **Fixed tool** > **Workpiece** > **Direct calibration**.
2. Assign a number and a name for the workpiece. Confirm with **Next**.
3. Enter the number of the fixed tool. Confirm with **Next**.
4. Move the origin of the workpiece coordinate system to the TCP of the fixed tool.

   Press **Calibrate** and confirm the position with **Yes**.
5. Move a point on the positive X axis of the workpiece coordinate system to the TCP of the fixed tool.

   Press **Calibrate** and confirm the position with **Yes**.
6. Move a point with a positive Y value in the XY plane of the workpiece coordinate system to the TCP of the fixed tool.

   Press **Calibrate** and confirm the position with **Yes**.
7. Enter the load data of the workpiece and confirm with **Next**.
8. Press **Save**.

10.3.3 Exercise: Calibrating an external tool and robot-guided workpiece

Aim of the exercise

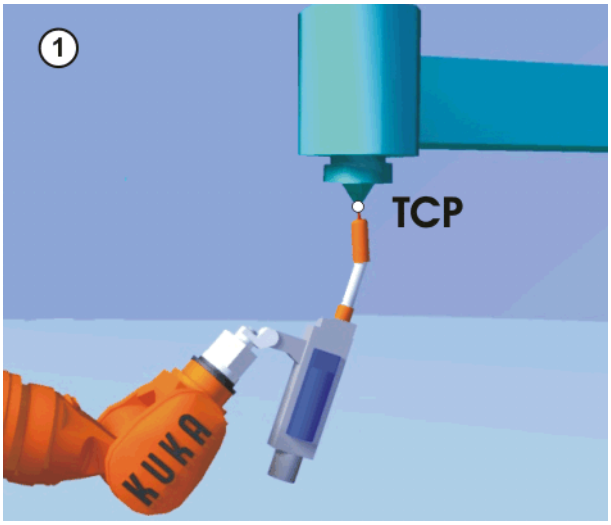
On successful completion of this exercise, you will be able to carry out the following activities:

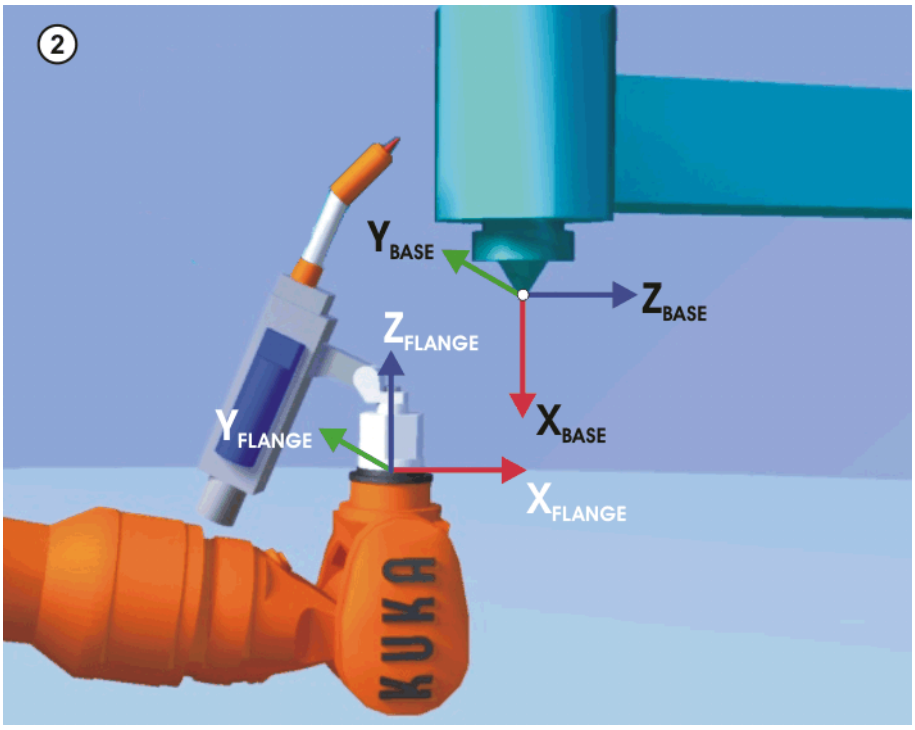
- Calibrate fixed tools
- Calibrate movable, robot-guided workpieces
- Carry out jogging with an external tool

Preconditions

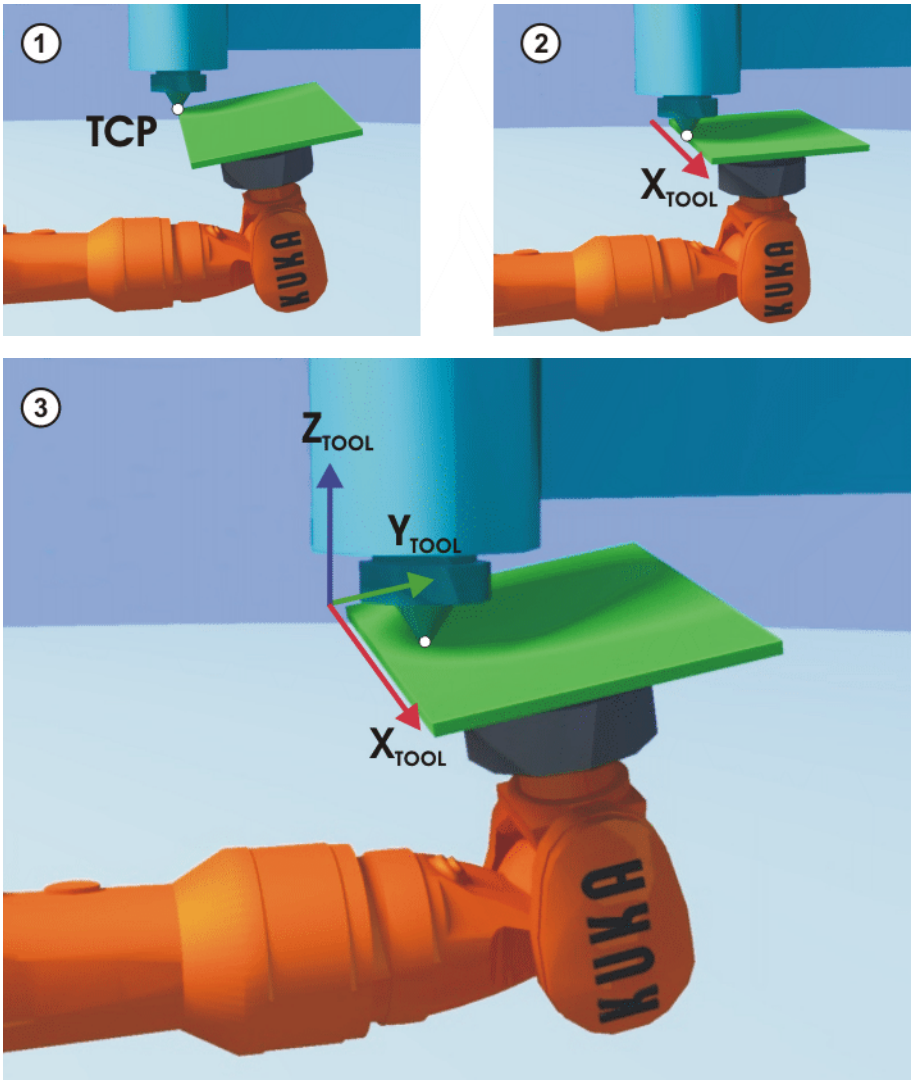
The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of the methods for calibrating fixed tools





- Theoretical knowledge of workpiece calibration with fixed tools, especially the direct method



Task description

Carry out the following tasks: calibrate nozzle and panel

1. For the calibration of the fixed tool, pen1 that is already calibrated (tool number 2) is to be used as a reference tool. Assign the **tool number 10** and the name **Nozzle** for the fixed tool.
   - Be sure to save your data each time you carry out calibration!
2. Calibrate the workpiece guided by the robot. Assign the **workpiece number 12** and the name **Panel**.

   Enter the load data.

   Use the correct gripper with panel for this (see below).
3. Once calibration is complete, activate the external tool for jogging. Make appropriate use of the Base and Tool coordinate systems to move the robot.
4. Move the TCP to the BASE coordinate origin of the tool being measured, causing the actual position to be displayed in “Cartesian” mode.

   Actual position:

X	Y	Z	A	B	C
.....	.....	.....	.....	.....	.....

Tool load data for gripper with panel

- Training gripper for KR 16

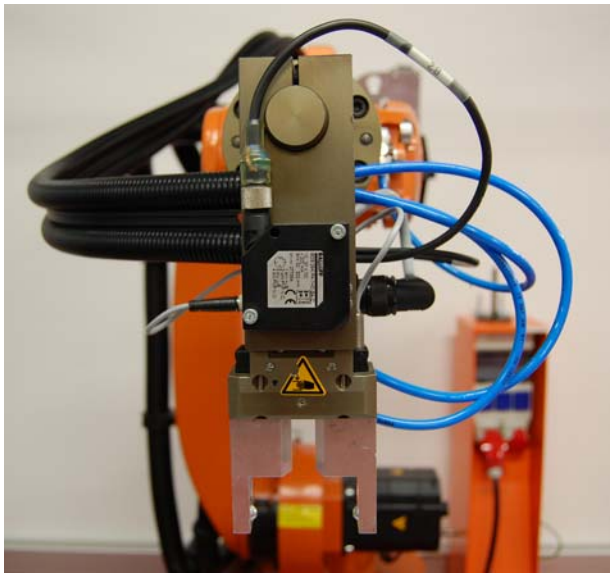


Fig. 10-9: Training gripper for KR 16

Mass:		
M = 6 kg		
Center of mass:		
X = 69 mm	Y = 67 mm	Z = 84 mm
Orientation:		
A = 0°	B = 0°	C = 0°
Moments of inertia:		
J_X = 0.02 kgm^2	J_Y = 0.06 kgm^2	J_Z = 0.18 kgm^2

- Training gripper for modular cell

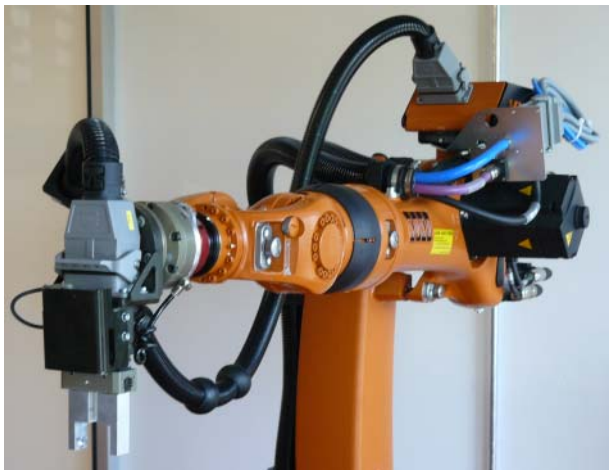


Fig. 10-10: Training gripper for modular cell

Mass:		
M = 6.5 kg		
Center of mass:		
X = 42 mm	Y = -54 mm	Z = 93 mm
Orientation:		
A = 0°	B = 0°	C = 0°

Moments of inertia:		
J_X = 0.02 kgm^2	J_Y = 0.11 kgm^2	J_Z = 0.18 kgm^2

- Training gripper for mobile cell

Fig. 10-11: Training gripper for mobile cell

Mass:		
M = 2.18 kg		
Center of mass:		
X = 34 mm	Y = 0 mm	Z = 68 mm
Orientation:		
A = 0°	B = 0°	C = 0°
Moments of inertia:		
J_X = 0.002 kgm^2	J_Y = 0.005 kgm^2	J_Z = 0.004 kgm^2

What you should now know:

1. How is a coordinate system calibrated for a workpiece mounted on the robot flange?

. . . . . .

. . . . . .

. . . . . .

2. How is the TCP of an external tool determined?

. . . . . .

. . . . . .

. . . . . .

3. What are the advantages of using an external TCP?

. . . . . .

. . . . . .

. . . . . .

4. What settings are required to move in the tool direction with an external TCP?

. . . . . .

10.4 Creating and modifying a programmed motion

10.4.1 Motion programming with external TCP

Motion programming with external TCP

In the case of motion programming with a fixed tool, the motion sequence differs from that of a standard motion in the following ways:

- Labeling in inline form: the entry **External TCP** in the option window **Frames** must be set to TRUE.

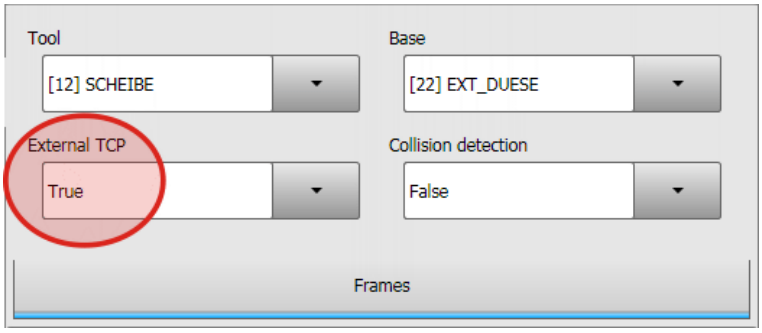


Fig. 10-12: Option window “Frames”: External TCP

- The **motion velocity** then refers to the external TCP.
- The **orientation** along the path then also refers to the external tool.
- Both the correct base coordinate system (fixed tool/external TCP) and the correct tool coordinate system (moving workpiece) must be specified.

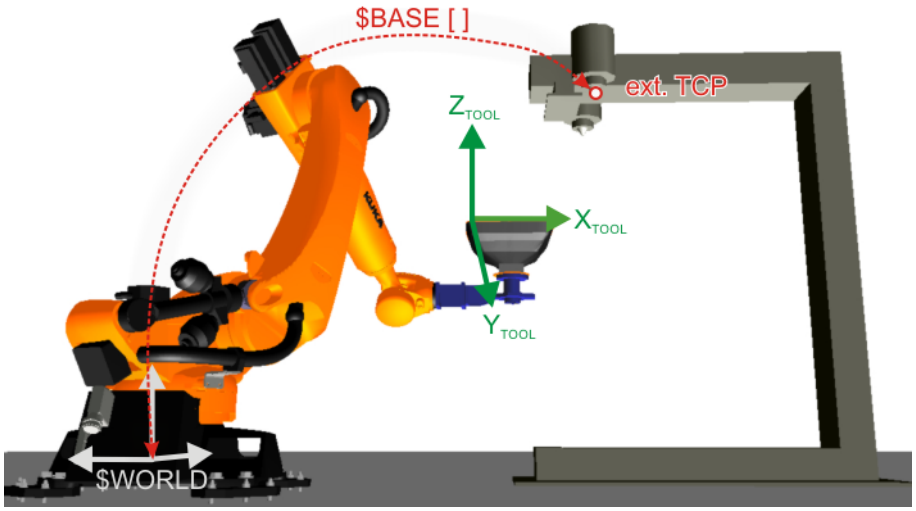


Fig. 10-13: Coordinate systems for fixed tool

10.4.2 Exercise: Motion programming with external TCP

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Program motions with a robot guiding a workpiece relative to a fixed tool

Preconditions

The following are preconditions for successful completion of this exercise:

- Knowledge of how to activate an external tool when programming motions

Task description

Carry out the following tasks: Program the contour for adhesive application:

1. Manually clamp the panel in the gripper.
2. Teach the contour on the plastic panel using the program name **Glue_panel**.
   - Do this using your calibrated external tool **Nozzle** and workpiece **Panel**.
   - Make sure that the longitudinal axis of the fixed tool is always perpendicular to the adhesive application contour
   - The jog velocity on the plastic panel is 0.2 m/s.
   - For this, use the individual blocks SPTP, SLIN and SCIRC.
3. Test your program in accordance with the instructions.
4. Archive your program.

What you should now know:

1. What does the adhesive velocity you have programmed refer to?

. . . . . .

. . . . . .

2. How do you activate the external tool in your program?

. . . . . .

. . . . . .

11 Introduction to Expert level

11.1 Overview

The following contents are explained in this training module:

- Using Expert level
- Structuring robot programs
- Linking robot programs

11.2 Using Expert level

Description

The robot controller offers various **user groups** with different functions. The following **user groups** can be selected:

- **Operator**
  User group for the operator. This is the default user group.
- **User**
  User group for the operator. (By default, the user groups “Operator” and “User” are defined for the same target group.)
- **Expert**
  User group for the programmer. This user group is protected by means of a password.
- **Administrator**
  The range of functions is the same as that for the user group “Expert”. It is additionally possible, in this user group, to integrate plug-ins into the robot controller. This user group is protected by means of a password. The factory password *kuka* should be changed.
- **Safety Recovery**
  This user can activate an existing safety configuration of the robot using an activation code. If no safe option, e.g. KUKA.SafeOperation or KUKA.SafeRangeMonitoring, is being used, the safety recovery technician has more extensive rights. In this case he is authorized, for example, to configure the standard safety functions. The user group is protected by means of a password. The factory password *kuka* should be changed.
- **Safety Maintenance**
  This user group is only relevant if KUKA.SafeOperation or KUKA.SafeRangeMonitoring is used. The user group is protected by means of a password. The factory password *kuka* should be changed.

Advanced functions of the “Expert” user group:

- Password-protected (default: *kuka*)
- Programming in Editor possible using KRL
- The “Expert” user group is left again automatically:
  - if the operating mode is switched to AUT or AUT EXT.
  - if no action is carried out on the user interface within a specific time (300 s).

All functions in the **Edit** menu are available at “Expert” level.

Functions

Creating programs with templates

- **Cell**: Existing Cell program can only be replaced or, if Cell is deleted, recreated.

- **Expert**: Module consisting of SRC and DAT file with just the program header and the end of the program.
- **Expert Submit**: Additional Submit file (SUB) consisting of the program header and the end of the program.
- **Function**: Creation of SRC function consisting of just the function header with a BOOL variable. The end of the function is present, but the return still has to be programmed.
- **Module**: Module consisting of SRC and DAT file with the program header, the end of the program and the basic framework (INI and 2x PTP HOME).
- **Submit**: Additional Submit file (SUB) consisting of the program header, the end of the program and the basic framework (DECLARATION, INI, LOOP/ENDLOOP).

The **filter** defines how programs are displayed in the file list. The following **filters** are available:

- Deselect or close the program.
- Select **Edit > Filter** and then **Detail** or **Module**.
- **Detail**

  Programs are displayed as SRC and DAT files. (Default setting)
- **Module**

  Programs are displayed as modules.

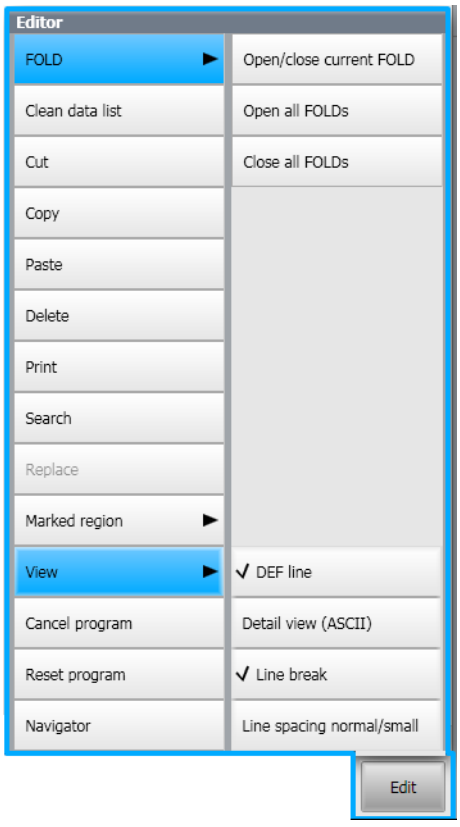


Fig. 11-1: Edit menu

Open/close **FOLD**

- The **FOLDs** are always closed for the user and can be opened by the Expert.
- The Expert can also program his own **FOLDs**.

- The **syntax** for a FOLD is:

  ;FOLD *Name*

  Instructions

  ;ENDFOLD *<Name>*

  The ENDFOLD lines can be assigned more easily if the name of the Fold is entered here as well. Folds can be nested.

Hide/show **DEF line**

- By default, the **DEF line** is hidden. Declarations can only be made in a program if the **DEF line** is visible.
- The **DEF line** is displayed and hidden separately for opened and selected programs. If detail view (ASCII mode) is activated, the **DEF line** is visible and does not need to be activated separately.

Procedure for activating Expert level and eliminating errors

Activating Expert level

1. Select **Configuration** > **User group** in the main menu.
2. Log on as **Expert**: Press **Login**. Select the user group **Expert** and confirm with **Login**.
3. Enter the password (default: kuka) when prompted and confirm with **Login**.

Eliminating errors in the program

1. Select faulty module in the Navigator.

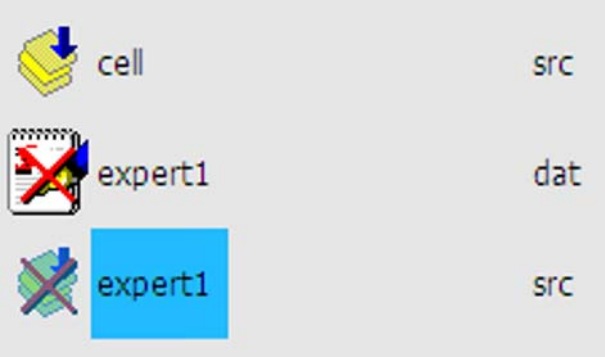


Fig. 11-2: Program containing errors

2. Select the **Error list** menu.
3. The error display (*program_name*.ERR) is opened.
4. Select error; a detailed description is displayed at the bottom of the error display.
5. In the error display window, press the **Display** key and jump to the faulty program.
6. Eliminate the error.
7. Exit editor and save changes.

11.3 Structuring robot programs

Robot program structuring options

The structure of a robot program is an important factor for its usability. The more structured a program is, the more comprehensible, effective, legible and economical it will be. The following techniques can be used for structuring a program:

- **Commenting** | Comment and stamp
- **Indentation** | Spaces
- **Hiding** | Folds

- **Module technique** | Subprograms

Comments and stamps

The addition of a comment makes it possible to store a text in the robot program, destined solely for the reader of the program. The text is thus not loaded by the robot interpreter. The text is only there to make the program more legible.

Comments can be used in many different ways in robot programs:

- **Information** about the program text: author, version, creation date

```
Editor
1  DEF welding1( )
2  ; Programmed by JACK SPARROW
3  ; Version 1.5 (10/10/2010)
4  INI
5
6  PTP HOME  Vel= 100 % DEFAULT
```

Fig. 11-3: Example of a comment: Information

- **Structuring** the program text: particularly using graphic means (special characters: #, *, ~,)

```
7  ;**************Initialisation************
8  INIT
9  BASISTECH INI
10 CHECK HOME
```

Fig. 11-4: Example of a comment: Structure

- **Commenting out** (Expert level): starting a program line with a semicolon turns it into a comment, i.e. the text is recognized as a comment and is ignored during program execution.

```
10 CHECK HOME
11 PTP HOME  Vel= 100 % DEFAULT
12 ;$OUT[33]=TRUE
13 AUTOEXT INI
14   LOOP
```

Fig. 11-5: Example of a comment: Commenting out

Inline forms **cannot** be commented out using a **;**.

- **Explanations** relating to individual lines
  and **notes about work to be done**: labeling of inadequate program sections

```
17
18      CASE 1
19        P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 ) ; Reset
  ↳ Progr.No.-Request
20        Main_Prog ( ) ; Call User-Program
21
```

Fig. 11-6: Example of a comment: Explanations

NOTICE Comments are only of any use if they are kept up to date. It is vital to ensure that the comments are updated following subsequent modification of the corresponding instructions!

Comments can be inserted in three different ways:

- By inserting a semicolon (Expert level): if a semicolon (“ ; ”) is inserted, the following part of the line is turned into a comment.
- Insertion of the inline form “Comment”

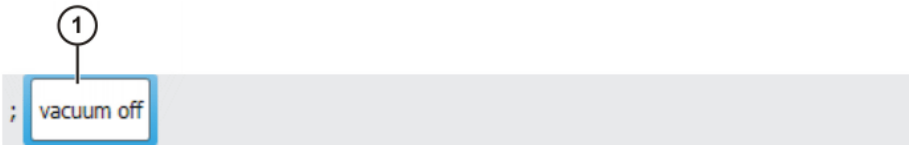


Fig. 11-7: Inline form “Comment”

Item	Description
1	Any text

- Insertion of the inline form “Stamp”: an additional time stamp is inserted. In addition, the name of the editor can also be inserted.

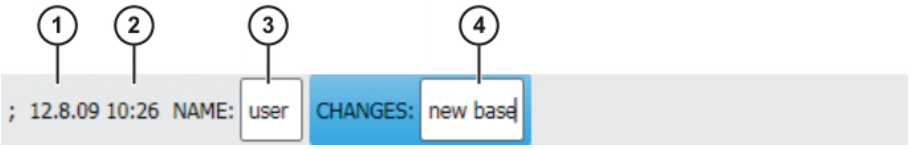


Fig. 11-8: Inline form “Stamp”

Item	Description
1	System date (cannot be edited)
2	System time (cannot be edited)
3	Name or ID of the user
4	Any text

Procedure for inserting comments and stamps

1. Select the line after which the comment or stamp is to be inserted.
2. Select the menu sequence **Commands** > **Comment** > **Normal** or **Stamp**.
3. Enter the desired data. If a comment or stamp has already been entered previously, the inline form still contains the same entries.
   - In the case of a comment, the box can be cleared using **New text** ready for entry of a new text.
   - In the case of a stamp, the system time can also be updated using **New time** and the **NAME** box can be cleared using **New name**.
4. Save with **Cmd Ok**.

Indenting program lines

Indenting program lines is an effective way of increasing the legibility of a robot program. It highlights program modules that belong together.

```
13  AUTOEXT INI
14  LOOP
15      P00 (#EXT_PGNO,#PGNO_GET,DMY[],0 )
16      SWITCH  PGNO ; Select with Programnumber
17         CASE 1
18           P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
19           Main_Prog ( ) ; Call User-Program
20         CASE 2
21           P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
22           Sub_Prog1 ( ) ; Call User-Program
23         DEFAULT
24           P00 (#EXT_PGNO,#PGNO_FAULT,DMY[],0 )
25      ENDSWITCH
26  ENDLOOP
```

Fig. 11-9: Indenting program lines

NOTICE The effect of indentation is purely optical. Indented program sections are processed during program execution in exactly the same way as program sections that are not indented.

Hiding program lines by means of folds

- The KUKA Robot Language offers the possibility of grouping program lines together and hiding them in *folds*.
- This makes program lines invisible for the user, making the program easier to read.
- Folds can be opened and edited in the user group Expert.

```
13
14
15  CHECK HOME
16
```

Fig. 11-10: Closed fold

```
14
15  CHECK HOME
16    $H_POS=XHOME
17    IF CHECK_HOME==TRUE THEN
18      P00 (#CHK_HOME,#PGNO_GET,DMY[],0 ) ;Test HPos
19    ENDIF
20
```

Fig. 11-11: Opened fold

Color coding of folds:

Color	Description
Dark red	Closed fold
Light red	Opened fold
Dark blue	Closed sub-fold
Light blue	Opened sub-fold
Green	Contents of the fold

11.4 Linking robot programs

Subprograms

The use of subprograms makes it possible to structure robot programs in a modular manner, thus giving them a more efficient layout. The aim is not to

write all commands into a program, but to outsource certain sequences, calculations or processes to separate programs.

There are numerous advantages to using subprograms:

- The main program receives a clear structure and is easier to read, as the program length is reduced.
- Subprograms can be developed independently of one another: the programming effort can be shared and error sources are minimized.
- Subprograms can be used repeatedly.

There are basically two different types of subprogram:

- Global subprograms

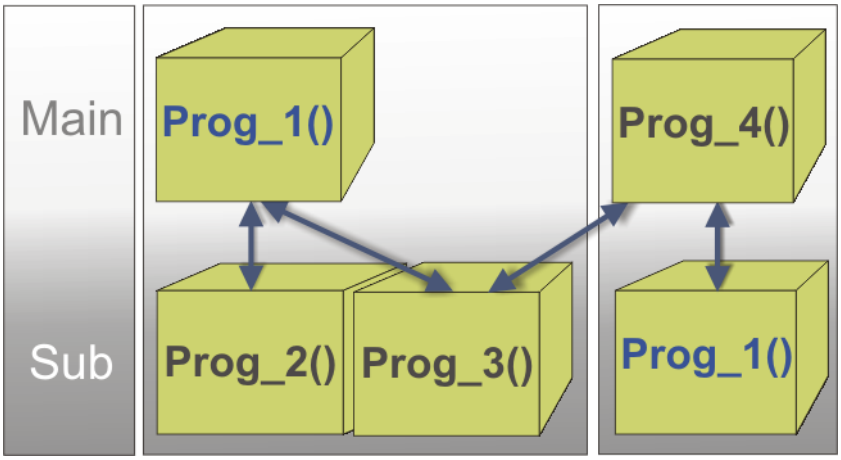


Fig. 11-12: Example diagram for global subprograms

A global subprogram is an independent robot program that is called from a different robot program. The branching of the programs can be requirement-specific, i.e. the program can function as a main program on one occasion and as a subprogram on a different occasion.

- Local subprograms

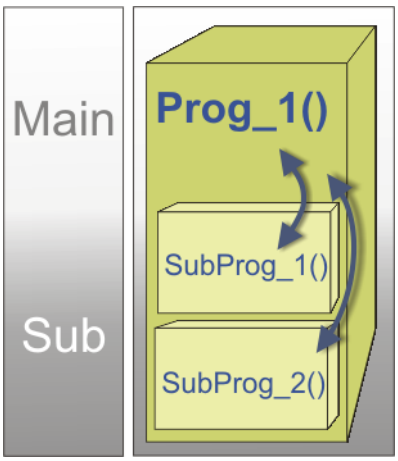


Fig. 11-13: Schematic diagram: local subprograms

Local subprograms are programs that are integrated into a main program, i.e. the commands are contained in the same SRC file. Point coordinates of the subprogram are thus saved in the same DAT file.

Subprogram call sequence

Every program begins with a DEF line and ends with an END line. If a subprogram is called in a main program, the subprogram is executed by default from DEF to END. Once the END line is reached, the program pointer jumps back to the program (main program) from which the subprogram was called.

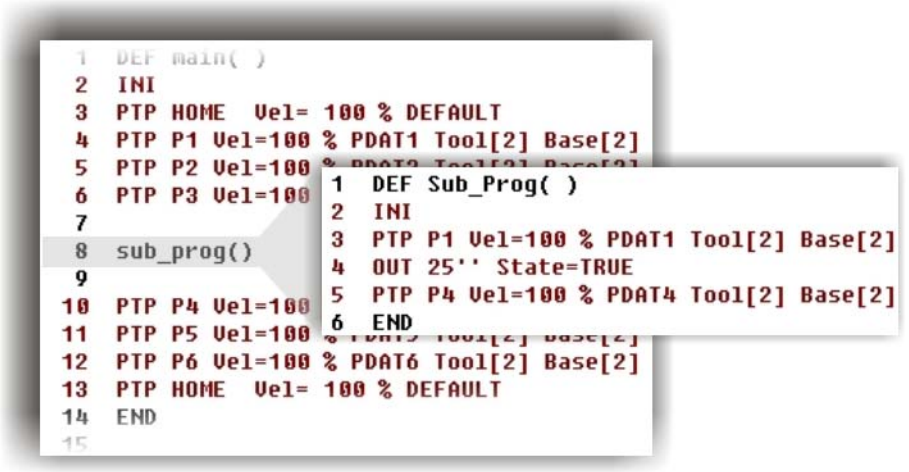


Fig. 11-14: Subprogram call sequence

NOTICE In order to exit a subprogram prematurely (i.e. before the END line), the command RETURN can be programmed in the subprogram. When this program line is read, the subprogram is terminated prematurely.

Procedure for calling a subprogram

In order to be able to program a subprogram call, the user group Expert must be selected. The **syntax** for a subprogram call is:

```
Name( )
```

1. Select **Configuration** > **User group** in the main menu. The current user group is displayed.
2. To switch to a different user group: Press **Login...**. Select the user group **Expert**.
3. Enter the password **kuka** and confirm with **Login**.
4. Load the desired main program into the editor with **Open**.

```
INI
PTP HOME Vel= 100% DEFAULT

PTP HOME Vel= 100% DEFAULT
```

5. Position the cursor in the desired line.
6. Enter the subprogram name with both brackets, e.g. **myprog( )**.

```
INI
PTP HOME Vel= 100% DEFAULT
myprog( )
PTP HOME Vel= 100% DEFAULT
```

7. Close the editor by means of the Close icon and save the changes.

11.5 Exercise: Programming a subprogram call

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Program subprogram calls

Preconditions

The following are preconditions for successful completion of this exercise:

- Knowledge of using the Navigator at Expert level
- Basic knowledge of programming at Expert level (KRL)

Task description

Carry out the following tasks:

1. Create a new module at Expert level with the name **Procedure**.
2. All other programs will be called as subprograms from this central program.
   The exact program sequence is indicated in the program flowchart.
3. Once the robot has executed all global subprograms in sequence, it should jump back to the program **Procedure** and call the subprograms again. The endless loop *LOOP - ENDLOOP*, which is described in detail in a later chapter, is required for this. (>>> 13.2.1 "Programming an endless loop" Page 243) Expand your program as follows for the exercise:

```
....
LOOP
  subprogram_1()
  subprogram_2()
  subprogram_3()
  subprogram_4()
  subprogram_5()
  subprogram_6()
ENDLOOP
...
```

4. Test your new program **Procedure** in T1, T2 and Automatic modes. Observe the relevant safety instructions.

What you should now know:

1. What do the KUKA filename extensions `SRC` and `DAT` mean?

. . . . . .

. . . . . .

2. How is a subprogram called and what is important here?

. . . . . .

. . . . . .

3. What is the difference between local and global subprograms?

. . . . . .

. . . . . .

12 Variables and declarations

12.1 Overview

The following contents are explained in this training module:

- Data management in KRL
- Working with simple data types
- Variable display

12.2 Data management in KRL

General information about variables

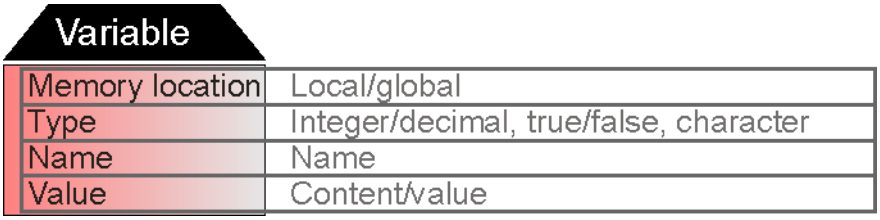


Fig. 12-1: Labeling of variables

- In the context of robot programming with KRL, a variable, in the broadest sense, is simply a container for values that arise in the course of a robot process.
- A variable has a specific address assigned to it in the memory of the computer.
- A variable is identified by a name which is not a KUKA keyword.
- Every variable is linked to a specific data type.
- The data type must be declared before use.
- A distinction is made in KRL between local and global variables.

Naming convention

The following rules must be observed when selecting the names of variables:

- Names in KRL can have a maximum length of 24 characters.
- Names in KRL can consist of letters (A-Z), numbers (09) and the signs “_” and “$”.
- Names in KRL must not begin with a number.
- Names in KRL must not be keywords.
- No distinction is made between uppercase and lowercase letters.

Tips

- Use meaningful, self-explanatory variable names.
- Do not use cryptic names or abbreviations.
- Use sensible name lengths, i.e. do not always use 24 characters.

Double declaration of variables

- A double declaration always occurs if the same variable name (character string) is used.
- It is **not** a double declaration if the same name is used in different **.SRC* or **.DAT* files.
- Double declarations in the same **.SRC* and **.DAT* file are not permissible and generate an error message.
- Double declarations in the **.SRC* or **.DAT* file and $CONFIG.DAT are permissible.
  - During execution of the program routine in which the variable was declared, only the local value is modified, not the value in $CONFIG.DAT.

- During execution of an “external” program routine, only the value from $CONFIG.DAT is accessed and modified.

Data types in KRL

- **Predefined standard data types**
  - **BOOL:** classic “YES”/“NO” results.
  - **REAL:** floating-point number, results of calculations to avoid rounding errors.
  - **INT:** classic counting variable for counting loops or part counters.
  - **CHAR:** one character only
    A string or text can only be implemented as a CHAR array.

Simple data types	Integer	Floating-point number	Logic values	Individual character
Keyword	INT	REAL	BOOL	CHAR
Range of values	-2^{31} ... $(2^{31}-1)$	$\pm 1.1\ 10^{-38}$... $\pm 3.4\ 10^{+38}$	TRUE / FALSE	ASCII character set
Examples	-199 or 56	-0.0000123 or 3.1415	TRUE or FALSE	"A" or "q" or "7"

- **Array**

```
Voltage[10] = 12.75
Voltage[11] = 15.59
```

  - Save multiple variables of the same data type by means of an index.
  - The index must be specified when initializing the variable or changing the values.
  - The maximum array size depends on the amount of memory required by the data type.
- **Enumeration data type**

```
color = #red
```

  - All values of the enumeration type are defined with a name (in plain text) when created.
  - The system also defines a sequence.
  - The maximum number of elements depends on the available memory.
- **Composite data type / structure**

```
Date = {day 14, month 12, year 1996}
```

  - Composite data type consisting of components of various data types
  - The components can consist of simple data types or of structures.
  - It is possible to access individual components.

Creating variables

Declaration of variables

- The variables must always be declared before use.
- Every variable must be assigned a data type.
- The naming convention must be observed when assigning names.
- The keyword for the declaration is **DECL**.
- The keyword DECL can be omitted in the case of the four simple data types.
- Value assignments are carried out in the advance run.
- Variables can be declared at different places. This affects the life and validity of the variable concerned.

Variable life and validity of variables

Variable life in KRL

- The variable life is the time in which a memory location has been reserved for the variable.
- When the program or function is exited, runtime variables free up their memory location again.
- Variables in a data list retain their current (last) value in their memory location permanently.

Validity of variables in KRL

- Variables declared as local are only available and visible in the program in which they were declared.
- Global variables are created in a central (global) data list.
- Global variables can also be created in a local data list and are assigned the keyword **global** during declaration.

Memory location-specific variable declaration

Variable in the *.SRC file

- A variable created in the **.SRC file* is called a runtime variable.
- It cannot always be displayed.
- It is only available in the program routine in which it was declared. The variable is thus available during program execution (main program **or** local subprogram).
- It frees up its memory location again on reaching the last line of the program (END line).

Variable in the local *.DAT file

- The variable can always be displayed during program execution of the corresponding **.SRC* file.
- The value of the variable is retained after the program is terminated.
- The variable is available in the entire **.SRC file*, i.e. also in local subprograms.
- The variable can also be created as a global variable.
- The variable is assigned the current value in the **.DAT file* and starts with the saved value when called again.
- If the variable is declared as global, it is also globally available. Read/write access is possible in all program routines if the DAT file is assigned the keyword `PUBLIC` and additionally the keyword `GLOBAL` is used in the declaration.

Variable in the system file $CONFIG.DAT

- The variable is available in all programs (global).
- The variable can always be displayed, even if no program is active.
- The variable is globally available, i.e. read/write access is possible in all program routines.
- The variable saves the current value in *$CONFIG.DAT*.

KUKA system data

- KUKA system data come in all data types, for example:
  - Enumeration data type, e.g. operating mode
  - Structure, e.g. date/time
- System information is obtained from KUKA system variables. These
  - read the current system information.
  - modify current system configurations.
  - are predefined and begin with the *"$"* sign.
    - *$DATE* (current date and time)
    - *$POS_ACT* (current robot position)

- *$MODE_OP* (current operating mode)
- ...

12.3 Working with simple data types

The creation, initialization and modification of variables is explained below. Only simple data types are used here.

Simple data types with KRL

- Integers (INT)
- Floating-point numbers (REAL)
- Logic values (BOOL)
- Individual character (CHAR)

12.3.1 Declaration of variables

Principle of variable declaration

Program structure in the SRC file (example)

- Variables must be declared in the declaration section.
- The initialization section begins with the first value assignment, which is usually the “INI” line, however.
- Values are assigned or modified in the instruction section.

```
DEF main( )
;Declaration section
...
;Initialization section
INI
...
;Instruction section
PTP HOME Vel=100% DEFAULT
...
END
```

Changing the standard view

- It is only possible to display the DEF line in Expert mode.
- This operation is necessary for accessing the declaration section before the “INI” line in modules.
- In order to be able to see the DEF and END line, but also important for variable transfer in subprograms.

Planning variable declaration

- Defining the variable life
- Defining validity/availability
- Defining the data type
- Name assignment and declaration

Procedure for the declaration of variables with a simple data type

Creating a variable in the SRC file

1. “Expert” user group
2. Display the DEF line.
3. Open the SRC file in the editor.
4. Carry out declaration of the variable.

```
DEF MY_PROG ( )
DECL INT counter
DECL REAL price
DECL BOOL error
```

```
DECL CHAR symbol
INI
...
END
```

5. Close and save the program.

Creating a variable in the DAT file

1. “Expert” user group
2. Open the DAT file in the editor.
3. Carry out declaration of the variable.

```
DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL INT counter
DECL REAL price
DECL BOOL error
DECL CHAR symbol
...
ENDDAT
```

4. Close and save the data list.

Creating a variable in $CONFIG.DAT

1. “Expert” user group
2. In the folder SYSTEM, open $CONFIG.DAT in the editor.

```
DEFDAT $CONFIG
BASISTECH GLOBALS
AUTOEXT GLOBALS
USER GLOBALS
ENDDAT
```

3. Select the fold “USER GLOBALS” and open it with the softkey “Fold open/cls”.
4. Carry out declaration of the variable.

```
DEFDAT $CONFIG
...
;================================
; Userdefined Types
;================================
;================================
; Userdefined Externals
;================================
;================================
; Userdefined Variables
;================================
DECL INT counter
DECL REAL price
DECL BOOL error
DECL CHAR symbol
...
ENDDAT
```

5. Close and save the data list.

Creating a global variable in the DAT file

1. “Expert” user group
2. Open the DAT file in the editor.
3. Expand the program header in the data list to include the keyword PUBLIC.

```
DEFDAT MY_PROG PUBLIC
```

4. Carry out declaration of the variable.

```
DEFDAT MY_PROG PUBLIC
EXTERNAL DECLARATIONS
DECL GLOBAL INT counter
DECL GLOBAL REAL price
DECL GLOBAL BOOL error
DECL GLOBAL CHAR symbol
...
ENDDAT
```

5. Close and save the data list.

12.3.2 Initialization of variables with simple data types

Description of initialization with KRL

- Following declaration, a variable only has a memory location reserved; its value is always an invalid value.
- In the SRC file, the declaration and initialization are always carried out in two separate lines.
- In the DAT file, the declaration and initialization are always carried out in one line.

  A constant can only be declared in a data list and must be initialized there immediately.
- The initialization section begins with the first value assignment.

Initialization principle

Initialization of integers

- Initialization as a decimal number

```
value = 58
```

- Initialization as a binary number

```
value = 'B111010'
```

Calculation: 1*32+1*16+1*8+0*4+1*2+0*1 = 58

Binary	2^5	2^4	2^3	2^2	2^1	2^0
Dec	32	16	8	4	2	1

- Initialization as a hexadecimal number

```
value = 'H3A'
```

Calculation: 3*16 +10 = 58

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Procedure for initialization with KRL

Declaration and initialization in the SRC file

1. Open the SRC file in the editor.
2. Declaration has been carried out.
3. Carry out initialization.

```
DEF MY_PROG ( )
DECL INT counter
DECL REAL price
DECL BOOL error
DECL CHAR symbol
INI
counter = 10
```

```
price = 0.0
error = FALSE
symbol = "X"
...
END
```

4. Close and save the program.

Declaration and initialization in the DAT file

1. Open the DAT file in the editor.
2. Declaration has been carried out.
3. Carry out initialization.

```
DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL INT counter = 10
DECL REAL price = 0.0
DECL BOOL error = FALSE
DECL CHAR symbol = "X"
...
ENDDAT
```

4. Close and save the data list.

Declaration in the DAT file and initialization in the SRC file

1. Open the DAT file in the editor.
2. Carry out the declaration.

```
DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL INT counter
DECL REAL price
DECL BOOL error
DECL CHAR symbol
...
ENDDAT
```

3. Close and save the data list.
4. Open the SRC file in the editor.
5. Carry out initialization.

```
DEF MY_PROG ( )
...
INI
counter = 10
price = 0.0
error = FALSE
symbol = "X"
...
END
```

6. Close and save the program.

Declaration and initialization of a constant

Description

- Constants are created using the keyword **CONST**.
- Constants may only be created in data lists.

Creation of constants

1. Close and save the DAT file in the editor.
2. Carry out declaration and initialization.

```
DEFDAT MY_PROG
EXTERNAL DECLARATIONS
DECL CONST INT max_size = 99
DECL CONST REAL PI = 3.1415
...
ENDDAT
```

3. Close and save the data list.

12.3.3 Manipulation of variable values of simple data types with KRL

List of options for modifying variable values with KRL

Modification of the variable values in the program routines (SRC file) varies according to the specific task. The most commonly used methods are described below. Manipulation by means of bit operations and standard functions is also possible, but is not dealt with in detail here.

Data manipulation by means of:

- **Basic arithmetic operations**
  - **(+)** Addition
  - **(-)** Subtraction
  - **(*)** Multiplication
  - **(/)** Division
- Comparison operations
  - **(==)** identical / equal to
  - **(<>)** not equal to
  - **(>)** greater than
  - **(<)** less than
  - **(>=)** greater than or equal to
  - **(<=)** less than or equal to
- Logic operations
  - **(NOT)** Inversion
  - **(AND)** Logic AND
  - **(OR)** Logic OR
  - **(EXOR)** Exclusive OR
- Bit operations
  - **(B_NOT)** Bit-by-bit inversion
  - **(B_AND)** Bit-by-bit ANDing
  - **(B_OR)** Bit-by-bit ORing
  - **(B_EXOR)** Bit-by-bit exclusive ORing

Standard functions

- Absolute function
- Root function
- Sine and cosine function
- Tangent function
- Arc cosine function
- Arc tangent function
- Multiple functions for string manipulation

Data manipulation relationships

Value modification using the data types REAL and INT

- Rounding up/down

```
; Declaration
DECL INT A,B,C
```

```
DECL REAL R,S,T
;initialization
A = 3       ; A=3
B = 5.5     ; B=6 (> x.5 is rounded up)
C = 2.25    ; C=2 (rounded down)
R = 4       ; R=4.0
S = 6.5     ; S=6.5
T = C       ; T=2.0 (the rounded-down value is taken)
```

- Results of arithmetic operations **(+;-;*)**

Operands	INT	REAL
INT	INT	REAL
REAL	REAL	REAL

```
; Declaration
DECL INT D,E
DECL REAL U,V
;initialization
D = 2
E = 5
U = 0.5
V = 10.6
; Instruction section (data manipulation)
D = D*E ; D = 2 * 5 = 10
E = E+V ; E= 5 + 10.6 = 15.6 -> rounded up E=16
U = U*V ; U= 0.5 * 10.6 = 5.3
V = E+V ; V= 16 + 10.6 = 26.6
```

- Results of arithmetic operations **(/)**

  The following must be noted for arithmetic operations with integer values:

  - In the case of interim results for operations with integers only, all decimal places are simply cut off.
  - In the case of value assignments to an integer variable, the result is rounded up or down in the normal manner.

```
; Declaration
DECL INT F
DECL REAL W
;initialization
F = 10
W = 10.0
; Instruction section (data manipulation)
; INT / INT -> INT
F = F/2 ; F=5
F = 10/4 ; F=2 (10/4 = 2.5 -> decimal place eliminated)
; REAL / INT -> REAL
F = W/4 ; F=3 (10.0/4=2.5 -> rounded up)
W = W/4 ; W=2.5
```

Comparison operations

Using relational operators, it is possible to form logical expressions. The result of a comparison is always of data type BOOL.

Operator/ KRL	Description	Permissible data types
==	identical / equal to	INT, REAL, CHAR, BOOL
<>	not equal to	INT, REAL, CHAR, BOOL
>	greater than	INT, REAL, CHAR
<	less than	INT, REAL, CHAR
>=	greater than or equal to	INT, REAL, CHAR
<=	less than or equal to	INT, REAL, CHAR

```
; Declaration
DECL BOOL G,H
; Initialization / Instruction section
G = 10>10.1 ; G=FALSE
H = 10/3 == 3 ; H=TRUE
G = G<>H ; G=TRUE
```

Logic operations

Logic expressions can be formed using logic operations. The result of such an operation is always of data type BOOL.

Operations		NOT A	A AND B	A OR B	A EXOR B
A=FALSE	B=FALSE	TRUE	FALSE	FALSE	FALSE
A=FALSE	B=TRUE	TRUE	FALSE	TRUE	TRUE
A=TRUE	B=FALSE	FALSE	FALSE	TRUE	TRUE
A=TRUE	B=TRUE	FALSE	TRUE	TRUE	FALSE

```
; Declaration
DECL BOOL K,L,M
; Initialization / Instruction section
K = TRUE
L = NOT K ; L=FALSE
M = (K AND L) OR (K EXOR L) ; M=TRUE
L = NOT (NOT K) ; L=TRUE
```

Operators are executed in order of priority.

Priority	Operator
1	NOT (B_NOT)
2	Multiplication (*); division (/)
3	Addition (+), subtraction (-)
4	AND (B_AND)
5	EXOR (B_EXOR)
6	OR (B_OR)
7	Any comparison (==; <>; ...)

```
; Declaration
DECL BOOL X, Y
DECL INT Z
; Initialization / Instruction section
X = TRUE
Z = 4
Y = (4*Z+16 <> 32) AND X ; Y=FALSE
```

Procedure for data manipulation

1. Define the data type for the variable(s).
2. Determine the validity and variable life of the variable.
3. Carry out variable declaration.
4. Initialize the variable.
5. Manipulate the variable in the program routines, i.e. always in the **.SRC* file.
6. Close and save the **.SRC* file.

HALT command

`HALT` is used primarily for testing purposes during the programming phase, e.g. to display the contents of a runtime variable.

- The `HALT` command stops the program. The last motion instruction to be executed will, however, be completed.
- Execution of the program can only be resumed using the Start key. The next instruction after HALT is then executed.

In an interrupt program, program execution is only stopped after the advance run has been completely executed.

Example:

```
DEF program()
DECL BOOL a,b
INI
...
SPTP XP1
a=$IN[1]
b=$IN[2]
HALT
IF ((a == TRUE) AND (b == FALSE)) THEN
..
ENDIF
...
```

12.4 Displaying variables

Displaying and modifying the value of a variable

1. In the main menu, select **Display** > **Variable** > **Single**.
   The **Variable display - Single** window is opened.
2. Enter the name of the variable in the **Name** box.
3. If a program has been selected, it is automatically entered in the **Module** box.
   If a variable from a different program is to be displayed, enter the program as follows:
   /R1/*Program name*
   Do not specify a folder between /R1/ and the program name. Do not add a file extension to the file name.
4. Press Enter.
   The current value of the variable is displayed in the **Current value** box. If nothing is displayed, no value has yet been assigned to the variable.

5. Enter the desired value in the **New value** box.
6. Press Enter.
   The new value is displayed in the **Current value** box.

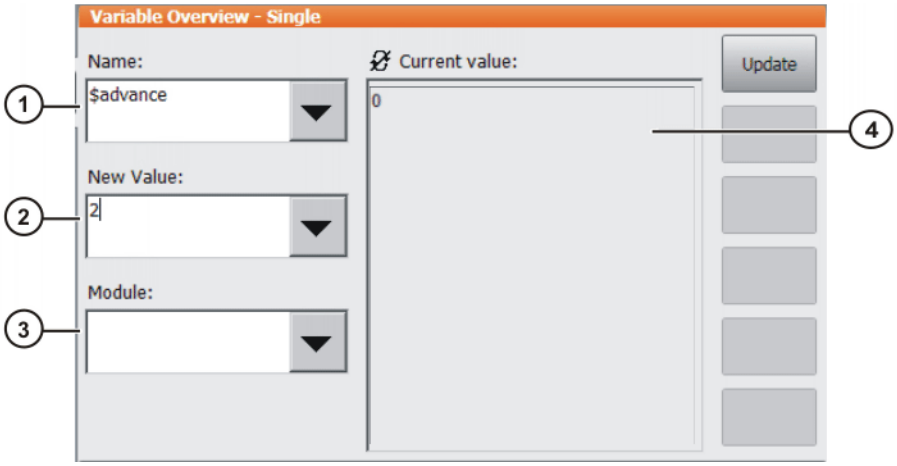


Fig. 12-2: Variable Overview - Single window

Item	Description
1	Name of the variable to be modified.
2	New value to be assigned to the variable.
3	Program in which the search for the variable is to be carried out. In the case of system variables, the **Module** box is irrelevant.
4	This box has two states: ■ : The displayed value is not refreshed automatically. ■ : The displayed value is refreshed automatically. Switching between the states: ■ Press **Update**. ■ Alternatively: **Shift + Enter**

System information display

Procedure for displaying flags, counters and timers:

- Select **Display** > **Variable** in the main menu.
  The various system variables are available for selection:
  - **Cyclical flags**
  - **Flags**
  - **Counters**
  - **Timers**

Procedure for displaying inputs and outputs:

- Select **Display** > **I/O** > **Digital Outputs** or **Digital Inputs** in the main menu.

12.5 Exercise: Simple data types

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Use simple data types
- Declaration, initialization and use of variables
- Correct use of variables with regard to their lifetime

Preconditions

The following are preconditions for successful completion of the exercise:

- Theoretical knowledge of simple data types and how they are handled

Task description

Create a new program with the name Procedure2:

- Copy the program Procedure and delete the program sections with the pen and the workpiece contour so that only the panel remains to be fetched, bonded and set down.
- A variable counts how often the program *Glue_panel* has been executed since the last program selection.
- A variable counts how often the program *Glue_panel* has been executed altogether.
- A variable is to add up the total length of the applied adhesive (in m). The length of a path for the program *Glue_panel* is 0.91 m. The length of the path is to be declared as a constant.
- A variable that is set to *TRUE* while the panel is being fetched and is otherwise set to *FALSE*.
- A variable that contains the letter “O” when the gripper is open and the letter “C” when the gripper is closed. During initialization, the variable is assigned the value “X”.

Use appropriate variable names and data types. It is also important to declare the variable appropriately in the correct place.

Variable name	Data type	Declaration point

- Define where your variables are declared.
- Expand your existing program flowchart to include these new variables.
- Observe the different initializations of the variables.
- Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

What you should now know:

1. What is the maximum length of a variable name?

. . . . . .

. . . . . .

2. What simple data types are there?

. . . . . .

. . . . . .

3. Where are variables declared in the SRC file?

. . . . . .

. . . . . .

4. What is the lifetime of a variable declared in $CONFIG.DAT?

. . . . . .

. . . . . .

5. Declare a floating-point number with the name “Value” in the DAT file with the value 138.74.

. . . . . .

. . . . . .

13 Using program execution control functions

13.1 Overview

The following contents are explained in this training module:

- Loops
- Branches
- Switch statements
- Jump command
- Wait functions in KRL

13.2 Programming loops

In addition to dedicated motion commands and communication commands (switching functions and wait functions), robot programs also include a large number of routines that can be used for program execution control.

General information regarding loops

- Loops are control structures.
- They go on repeating program instructions until a break condition is fulfilled.
- It is not permissible to jump into a loop from outside.
- Loops can be nested.
- There are various different types of loop:
  - Endless loop
  - Counting loop
  - Conditional loops
    - Rejecting loop
    - Non-rejecting loop

13.2.1 Programming an endless loop

Description of an endless loop

- The endless loop is a loop that is executed again every time execution has been completed.
- Execution can be canceled by external influences.
- Syntax

```
LOOP
;   Statement
...
;   Statement
ENDLOOP
```

Principle of an endless loop

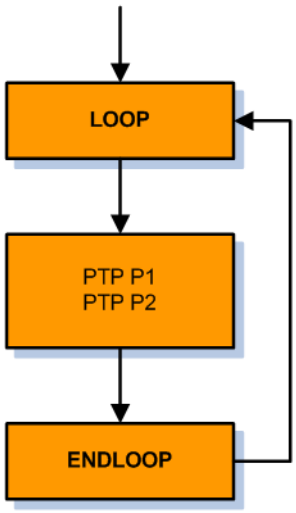


Fig. 13-1: Program flowchart: endless loop

- The endless loop can be exited with `EXIT`.
- When exiting an endless loop with `EXIT`, it must be ensured that there is no risk of a collision.
- If two endless loops are nested in one another, two `EXIT` commands are required in order to exit both loops.

Examples for programming of an endless loop

Endless loop without break

```
DEF MY_PROG( )
INI
PTP HOME Vel=100% DEFAULT

LOOP
   SPTP XP1
   SPTP XP2
   SPTP XP3
   SPTP XP4
ENDLOOP

SPTP P5 Vel=30% PDAT5 Tool[1] Base[1]
SPTP HOME Vel=100% DEFAULT
END
```

Point P5 is never addressed in the program.

Endless loop with break

```
DEF MY_PROG( )
INI
SPTP HOME Vel=100% DEFAULT

LOOP
   SPTP XP1
   SPTP XP2
   IF $IN[3]==TRUE THEN ; Condition for break
      EXIT
   ENDIF
   SPTP XP3
   SPTP XP4
```

```
ENDLOOP

SPTP P5 Vel=30% PDAT5 Tool[1] Base[1]
SPTP HOME Vel=100% DEFAULT
END
```

Point P5 is addressed as soon as input 3 is active.
Important: The motion between P2 and P5 must be checked to ensure there is no risk of a collision.

13.2.2 Programming a counting loop

Description of a counting loop

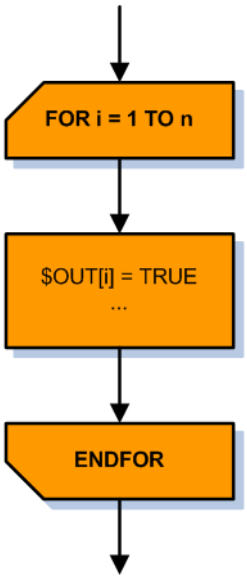


Fig. 13-2: Program flowchart: counting loop

- The FOR loop is a control structure which can be used to execute one or more statements with a defined number of repetitions.
- For a counting loop, a previously declared loop counter of data type “Integer” is required.
- The counting loop starts with the value start and ends, at the latest, with the value last.

  Syntax with step size +1

```
FOR counter = start TO last
;Statement
ENDFOR
```

- The step size (increment) can also be specified as an integer using the keyword STEP.

```
FOR counter = start TO last STEP increment
;Statement
ENDFOR
```

- The counting loop can be exited immediately with EXIT.

Counting forwards with a counting loop

```
DECL INT counter

FOR counter = 1 TO 3 Step 1
;Statement
ENDFOR
```

1. Loop counter is initialized with the start value: `counter = 1`
2. At `ENDFOR`, the loop counter is incremented by the value of `STEP`.
3. Loop begins again at the `FOR` line.
4. The entry condition is checked: loop counter must be less than the specified end value, otherwise the loop is terminated.
5. Depending on the result of the check, either the loop counter is incremented again, or the loop is terminated and the program is resumed after the `ENDFOR` line.

Examples:

- Simple counting loop without specification of step size

```
DECL INT counter

FOR counter = 1 TO 50
$OUT[counter] = FALSE
ENDFOR
```

If no step size is specified with `STEP`, the step size +1 is used by default.

- Simple counting loop with specification of step size

```
DECL INT counter

FOR counter = 1 TO 4 STEP 2
$OUT[counter] = TRUE
ENDFOR
```

This loop is only executed twice: once with the start value `counter=1` and the second time with `counter=3`. Once the counter value reaches 5, the loop is terminated immediately.

Counting backwards with a counting loop

```
DECL INT counter

FOR counter = 15 TO 1 Step -1
;Statement
ENDFOR
```

The initial value or start value of the loop must be greater than the end value in order to allow the loop to be executed several times.

Example:

- Counting loop with specification of a negative step size

```
DECL INT counter

FOR counter = 10 TO 1 STEP -1
    ;Statement
ENDFOR
```

- Nested counting loop with specification of step size

```
DECL INT counter1, counter2

FOR counter1 = 1 TO 21 STEP 2
    FOR counter2 = 20 TO 2 STEP -2
        ...
    ENDFOR
ENDFOR
```

The inner loop is always executed first (here `counter2`), and then the outer one (`counter1`).

13.2.3 Programming a rejecting loop

Description of a rejecting loop

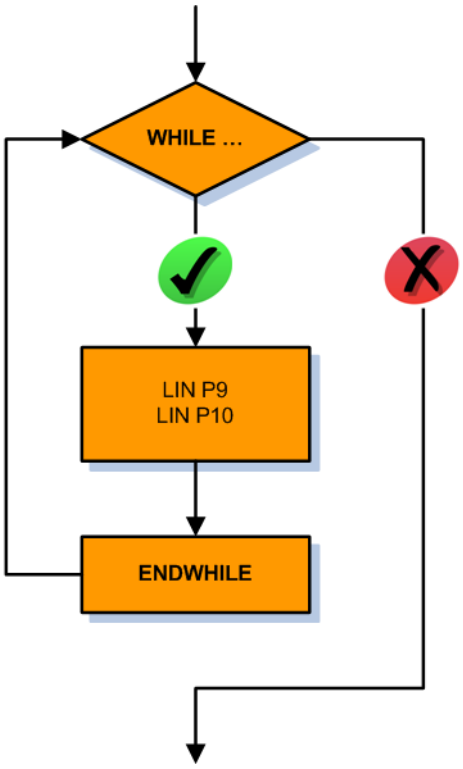


Fig. 13-3: Program flowchart: Rejecting loop

- A WHILE loop is also referred to as a *pre-test loop*.
- A WHILE loop is a *rejecting* or *pre-test* loop that checks the break condition before the instruction section of the loop is executed.
- This type of loop repeats operations as long as a specified `execution condition` is fulfilled.
- If the execution condition is not met, the loop is terminated immediately and the statements after `ENDWHILE` are executed.
- Syntax

```
WHILE condition
  ; Statement
ENDWHILE
```

- The rejecting loop can be exited immediately with `EXIT`.

Programming with a rejecting loop

- **Rejecting loop with simple execution condition**

```
...
WHILE IN $41]==TRUE ; Part is ready in magazine
   PICK_PART( )
ENDWILE
...
```

The expression `WHILE $IN[41]==TRUE` can also be reduced to `WHILE $IN[41]`. Omission always signifies comparison with `TRUE`.

- **Rejecting loop with simple negated execution condition**

```
...
WHILE  NOT $IN[42]==TRUE ; Input 42: magazine is empty
   PICK_PART( )
ENDWILE...
```

or

```
...
WHILE  $IN[42]==FALSE ; Input 42: magazine is empty
   PICK_PART( )
ENDWILE...
```

- **Rejecting loop with complex execution condition**

```
...
WHILE  (($IN[40]==TRUE) AND ($IN[41]==FALSE) OR (counter>20))
   PALLET( )
ENDWILE
...
```

13.2.4 Programming a non-rejecting loop

Description of a non-rejecting loop

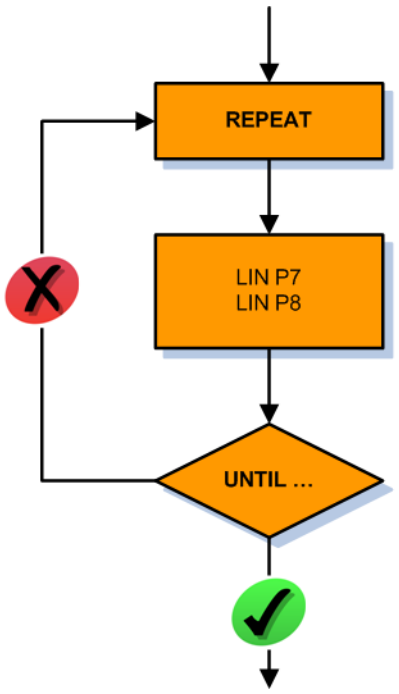


Fig. 13-4: Program flowchart: Non-rejecting loop

- A non-rejecting loop is also referred to as a *post-test loop*.
- A REPEAT loop is a *non-rejecting* or *post-test* loop that does not check the break condition until after the instruction section of the loop has been executed for the first time.
- Once the statement section has been executed, the system checks whether a `condition` has been met in order to be able to exit the loop.
  - If the result of the condition is positive, the loop is exited and the statements after `UNTIL` are executed.
  - If the result of the condition is negative, the loop is started again from `REPEAT`.
- The non-rejecting loop can be exited immediately with `EXIT`.
- Syntax

```
REPEAT
  ; Statement
UNTIL condition
```

Programming a non-rejecting loop

- Non-rejecting loop with simple execution condition

```
...
REPEAT
PICK_PART( )
UNTIL $IN[42]==TRUE ; Input 42: magazine is empty
...
```

The expression `UNTIL $IN[42]==TRUE` can also be reduced to `UNTIL $IN[42]`. Omission always signifies comparison with `TRUE`.

- Non-rejecting loop with complex execution condition

```
...
REPEAT
PALLET( )
UNTIL (($IN[40]==TRUE) AND ($IN[41]==FALSE) OR (counter>20))
...
```

If the result is positive, the loop is terminated!

13.3 Programming conditional statements or branches

Description of branches

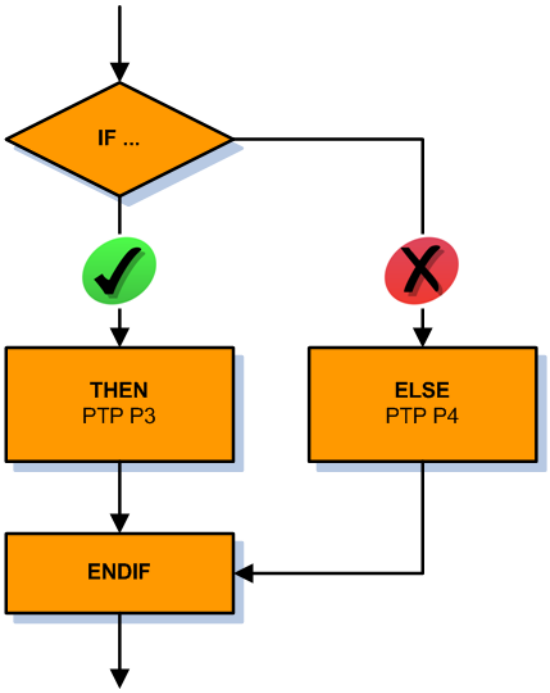


Fig. 13-5: Program flowchart: IF branch

- Branches can be used if program sections are only to be executed under certain conditions.
- A branch is used to divide a program into several paths.
- A *conditional branch* (IF statement) consists of a condition and two instruction sections.
- The `IF` statement checks whether this condition is TRUE or FALSE.
  - If the condition is fulfilled, the first instruction can be executed.
  - If the condition is not met, the second, alternative instruction is executed.
- Variants of the IF statement:
  - The second instruction section can be omitted: IF statement without ELSE. This means that if the condition is not met, the program is resumed directly after the branch.
  - Several IF statements can be nested in each other (*multiple branch*): the statements are executed in sequence and checked to see whether a condition is met.

Programming a branch

Syntax

- **with alternative branch**

```
IF condition THEN
Statement
ELSE
Statement
ENDIF
```

- **without alternative branch (conditional statement)**

```
IF condition THEN
Statement
ENDIF
```

Examples of branches

Branch with alternative branch

```
DEF MY_PROG( )
DECL INT error_nr
...
INI
error_nr = 4
...
; P21 is only addressed with error_nr = 5, otherwise P22
IF error_nr == 5 THEN
   SPTP XP21
ELSE
   SPTP XP22
ENDIF
...
END
```

Branch without alternative branch

```
DEF MY_PROG( )
DECL INT error_nr
...
INI
error_nr = 4
...
; P21 is only addressed with error_nr = 5
IF error_nr == 5 THEN
   SPTP XP21
ENDIF
...
END
```

Branch with complex execution conditions

```
DEF MY_PROG( )
DECL INT error_nr
...
INI
error_nr = 4
...
; P21 is only addressed with error_nr = 1 or 10 or greater than 99
IF ((error_nr == 1) OR (error_nr == 10) OR (error_nr > 99)) THEN
   SPTP XP21
ENDIF
...
END
```

Branch with Boolean expressions

```
DEF MY_PROG( )
DECL BOOL no_error
...
INI
no_error = TRUE
...
; P21 is only addressed if there is no error (no_error)
IF no_error == TRUE THEN
   SPTP XP21
ENDIF
...
END
```

The expression `IF no_error==TRUE THEN` can also be reduced to `IF no_error THEN`. Omission always signifies comparison with `TRUE`.

13.4 Programming a switch statement (SWITCH – CASE)

Description of the switch statement (SWITCH – CASE)

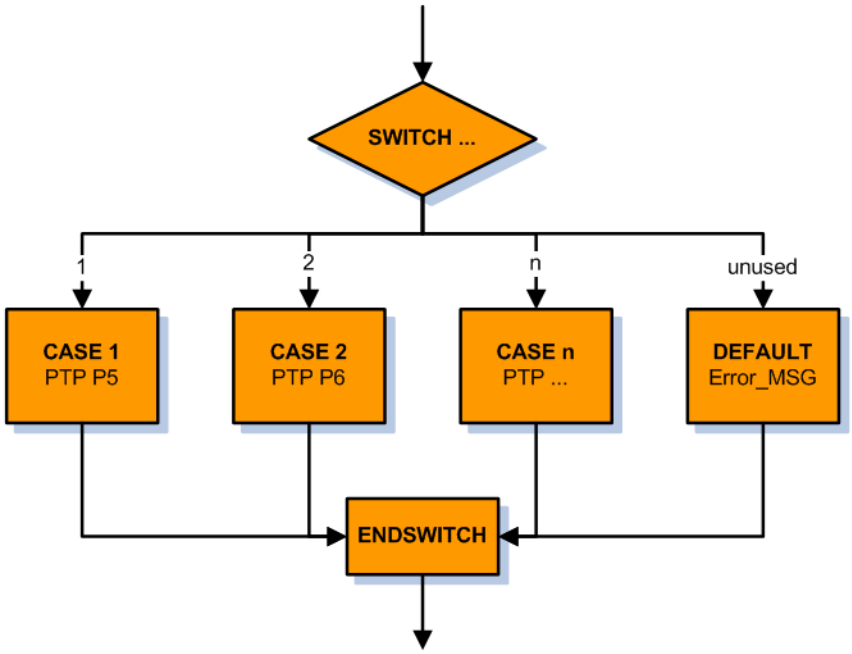


Fig. 13-6: Program flowchart: SWITCH – CASE statements

- A `SWITCH – CASE` statement can be used to differentiate between numerous different cases and execute different actions for each case.
- A `SWITCH – CASE` branch is a *switch statement* or *multiple branch* and is used for case distinction.
- A transferred variable in the `SWITCH` statement is used as the switch and jumps to the predefined `CASE` statements in the statement block.
- If the `SWITCH` statement finds no predefined `CASE`, the `DEFAULT` section is executed, provided that it has been defined beforehand.

Syntax

```
SWITCH Selection criterion
CASE Value
Statement
CASE Value
Statement
CASE Value
Statement
...
DEFAULT
Statement
ENDSWITCH
```

Value transfer via simple data types

- **SWITCH – CASE** can be used in combination with the following data types:
  - **INT (integer)**

    Example:

```
DEF MY_PROG( )
DECL INT my_number
...
INI
my_number = 2
...
SWITCH my_number
   CASE 1
      SPTP XP21
   CASE 2
      SPTP XP22
   CASE 3
      SPTP XP23
ENDSWITCH
...
```

- **CHAR (character)**
  Example:

```
DEF MY_PROG( )
DECL CHAR my_sign
...
INI
my_sign = "a"
...
SWITCH my_sign
   CASE "a"
      SPTP XP21
   CASE "b"
      SPTP XP22
   CASE "c"
      SPTP XP23
ENDSWITCH
...
```

SWITCH – CASE variants

A SWITCH – CASE statement can be programmed:

- **Only with defined switch statements and no alternative path**

```
DEF MY_PROG( )
DECL INT error_nr
...
INI
error_nr = 4
...
; Motion is possible in a defined case only
SWITCH error_nr
   CASE 1
      SPTP XP21
   CASE 2
      SPTP XP22
   CASE 3
      SPTP XP23
ENDSWITCH
...
```

If *error_nr* is not equal to 1 or 2 or 3, the program jumps directly to ENDSWITCH without a statement being executed.

- **Only with defined switch statements and an alternative case**
  Example:

```
DEF MY_PROG( )
DECL INT error_nr
...
INI
error_nr = 99
...
; In a non-defined case, robot moves to HOME
SWITCH error_nr
   CASE 1
      SPTP XP21
   CASE 2
      SPTP XP22
   CASE 3
      SPTP XP23
   DEFAULT
      SPTP XHOME
ENDSWITCH
...
```

If *error_nr* is not equal to 1 or 2 or 3, the program jumps to the DEFAULT case in order to execute the statement(s) contained therein.

- **With several solutions in a switch statement and an alternative path**

```
SWITCH number
CASE 1,2
...
CASE 3,4,5
...
CASE 6
...
DEFAULT
...
ENDSWITCH
```

If *number* is 3, 4 or 5, the program jumps to the second CASE in order to execute the statement(s) contained therein.

13.5 Programming a jump command

Description

- Unconditional jump to a specified position in the program. Program execution is resumed at this position.
- The destination must be in the same program section or function as the GOTO statement.
- The following jumps are **not** possible:
  - Into an IF statement from outside.
  - Into a loop from outside.
  - From one CASE statement to another CASE statement.

In the case of an unconditional jump, the program does not check whether a specific condition is met or not.
The jump is always executed. If undesired jumps occur, these must be remedied in the programming. See also the second program example.
Programming with GOTO can lead to a loss of structural clarity in programs. It is better to work with IF, SWITCH or a loop instead.

Syntax

```
...
GOTO Marke
...
Marke:
...
```

Element	Description
Label (Marke)	Position to which a jump is made. At the destination position, *Label* must be followed by a colon.

Examples

- Unconditional jump to the program position GLUESTOP.

```
GOTO GLUE_STOP
...
GLUE_STOP:
```

- Conversion of an unconditional jump to a conditional jump by means of expansion with an IF statement. The jump is to the program position *GLUE_END*.

```
IF X>100 THEN
   GOTO GLUE_END
ELSE
   X=X+1
ENDIF
A=A*X
...
GLUE_END:
END
```

13.6 Programming wait functions in KRL

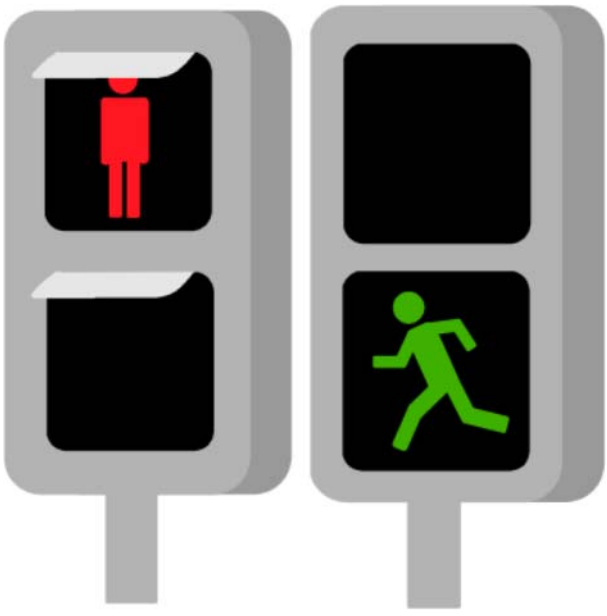


Fig. 13-7

KRL programming of:

- Time-dependent wait function
- Signal-dependent wait function

13.6.1 Time-dependent wait function

Description of a time-dependent wait function with KRL

- The time-dependent wait function waits for the specified `time` before the process can be resumed.
- Syntax

```
WAIT SEC time
```

Principle of the time-dependent wait function

- The time base for a time-dependent wait function is seconds (s).
- The maximum time is 2147484 seconds, i.e. more than 24 days.

> The inline form for the time-dependent wait function can wait a maximum of 30 seconds.

- The time value can also be transferred with a suitable variable.
- The smallest meaningful unit of time is 0.012 seconds (interpolation cycle).
- If the specified time is negative, the program does not wait.
- A time-dependent wait function triggers an advance run stop; approximate positioning is thus not possible.
- In order to generate just one advance run stop, the command `WAIT SEC 0` is used.

Programming a time-dependent wait function

- Time-dependent wait function with a fixed time

```
SPTP P1 Vel=100% PDAT1
SPTP P2 Vel=100% PDAT2
WAIT SEC 5.25
SPTP P3 Vel=100% PDAT3
```

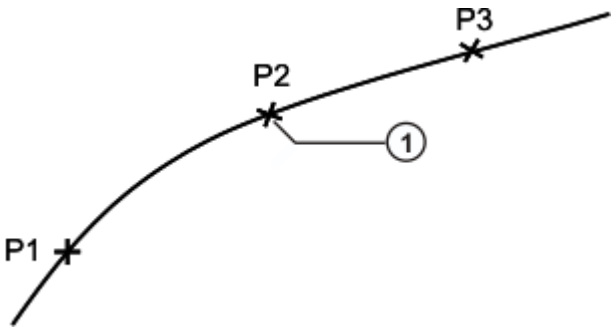


Fig. 13-8: Example motion for logic

Item	Comments
1	Motion is interrupted for 5.25 seconds at point P2.

- Time-dependent wait function with a calculated time

```
WAIT SEC 3*0.25
```

- Time-dependent wait function with a variable

```
DECL REAL time
time = 12.75
WAIT SEC time
```

13.6.2 Signal-dependent wait function

Description of a signal-dependent wait function

- The signal-dependent wait function switches when the `condition` is met and the process is resumed.
- Syntax

```
WAIT FOR condition
```

Principle of the signal-dependent wait function

- The signal-dependent wait function triggers an advance run stop; approximate positioning is thus not possible.
- Even if the condition has already been met, an advance run stop is still generated.
- If the command CONTINUE is programmed in the line immediately before the wait command, an advance run stop can be prevented if the condition is met in time.

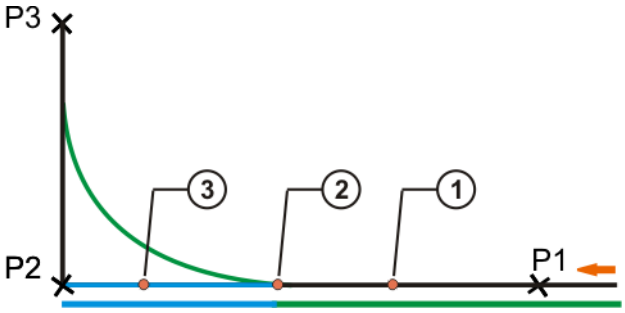


Fig. 13-9: Example motion for logic with advance run

	Position	Switching range
1	The robot carries out approximate positioning if the wait condition is already met before the start of the approximate positioning contour.	Green switching range for *activation* of the approximate positioning contour. This is only set and can no longer be deactivated.
2	Start of the approximate positioning motion (start of the approximate positioning contour)	*Activation* monitoring ■ TRUE: Approximate positioning ■ FALSE: Motion to the end point
3	The robot does not carry out approximate positioning if the wait condition is not met until after the start of the approximate positioning contour.	Blue switching range for motion to and stop at point P2

Programming a signal-dependent wait function

- WAIT FOR with advance run stop

```
SPTP P1 Vel=100% PDAT1
SPTP P2 CONT Vel=100% PDAT2
WAIT FOR $IN[20]==TRUE
SPTP P3 Vel=100% PDAT3
```

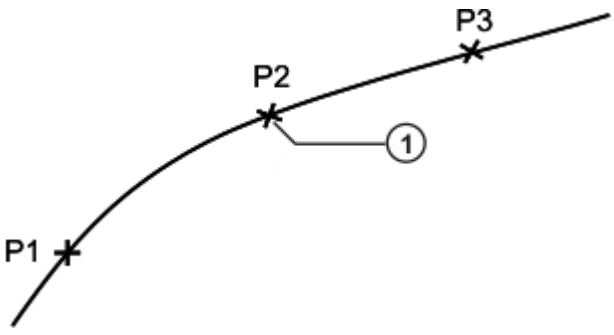


Fig. 13-10: Example motion for logic

	Comments
1	Motion is interrupted at point P2. Input 20 is checked after the exact positioning. If the input has the expected state, the motion can be continued directly; otherwise, the system waits for the required state.

- WAIT FOR with processing in the advance run (use of CONTINUE)

```
SPTP P1 Vel=100% PDAT1
SPTP P2 CONT Vel=100% PDAT2
CONTINUE
WAIT FOR ($IN[10] OR $IN[20])
SPTP P3 Vel=100% PDAT3
```

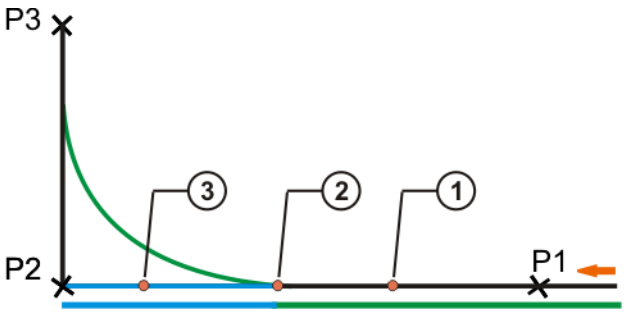


Fig. 13-11: Example motion for logic with advance run

	Action
1	Input 10 or input 20 is or was already set to TRUE in the advance run. Approximate positioning is thus carried out.
2	Condition met shortly beforehand; the motion is thus approximated.
3	Condition met too late; the motion can thus not be approximated and the robot must move to point P2. At point P2, however, motion can be resumed immediately if the condition has been met by then. If the condition has not been met, however, the robot waits at point P2 until the condition has been met. The corresponding display in the message window is “Wait for (input 10 or input 20)”. The button “Simulate” is available in the test modes (T1 and T2).

14 Working with a higher-level controller

14.1 Overview

The following contents are explained in this training module:

- Preparing program start from a PLC
- Adapting the PLC interface

14.2 Preparation for program start from PLC

Robot in system group

If robot processes are to be controlled centrally (by a host computer or PLC), this is carried out using the *Automatic External* interface.

Fig. 14-1: PLC connection

System structure principle

The *Automatic External* interface allows robot processes to be controlled by a higher-level controller (e.g. a PLC).

The following are required for communication between the PLC and the robot:

- There must be a configured field bus physically present between the robot and the PLC, e.g. PROFINET.
- Signals for the robot processes must be transmitted via the field bus. This is achieved by means of configurable digital inputs and outputs in the *Automatic External* interface.
  - **Control signals to the robot (inputs):**
    The higher-level controller transmits the signals for the robot processes (e.g. motion enable, fault acknowledgement, program start, etc.) to the robot controller via the Automatic External interface.
  - **Robot state (outputs):**
    The robot controller transmits information about operating states and fault states to the higher-level controller.
- An adapted **CELL.SRC**
  Organization program for selecting robot programs from outside.
- Selection of **Automatic External mode**
  Operating mode in which a host computer or PLC assumes control of the robot system.

Safety instructions – external program start

Once the CELL program has been selected, a BCO run must be carried out in T1 or T2 mode.

⚠ WARNING A BCO run is executed as a PTP motion from the actual position to the target position if the selected motion block contains the motion command PTP. If the selected motion block contains LIN or CIRC, the BCO run is executed as a LIN motion. Observe the motion to avoid collisions. The velocity is automatically reduced during the BCO run.

If the BCO run is successful, no further BCO run is performed in the case of the external start.

⚠ WARNING There is no BCO run in Automatic External mode. This means that the robot moves to the first programmed position after the start at the programmed (not reduced) velocity and does not stop there.

Procedure – external program start

Preconditions

- T1 or T2 mode
- Inputs/outputs for Automatic External and the program CELL.SRC are configured.

1. Select the program CELL.SRC in the Navigator. The CELL program is always located in the directory KRC:\R1.
2. Set program override to 100%. (This is the recommended setting. A different value can be set if required.)

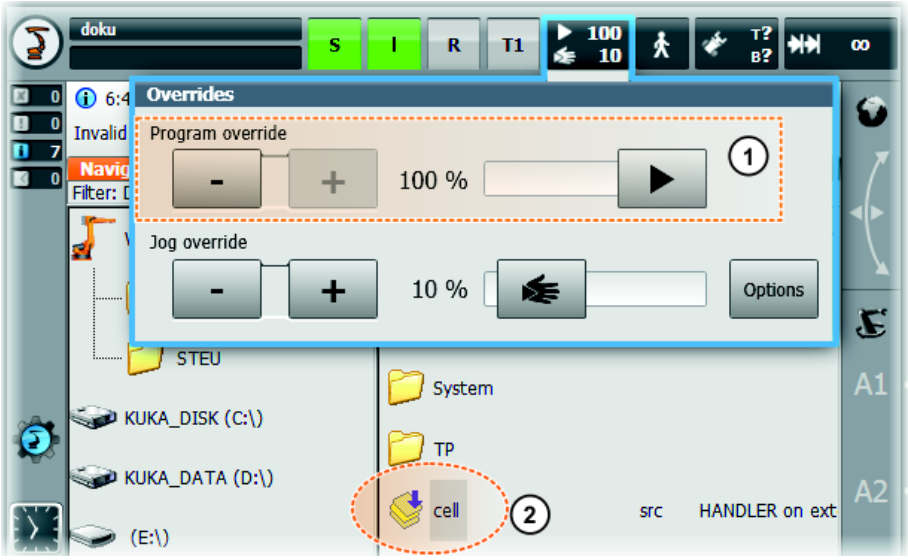


Fig. 14-2: Cell selection and program override setting

1 POV setting
2 Cell.src selection

3. Carry out a BCO run:
   Press and hold down the enabling switch. Then press the Start key and hold it down until the message “Programmed path reached (BCO)” is displayed in the message window.
4. Select “Automatic External” mode.
5. Start the program from a higher-level controller (PLC).

14.3 Adapting the PLC interface (Cell.src)

Cell.src organization program

The organization program Cell.src is used to manage the program numbers transferred by the PLC. It is located in the folder “R1”. Like any other program, the Cell program can be customized, but the basic structure of the program must be retained.

Structure and functionality of the Cell program

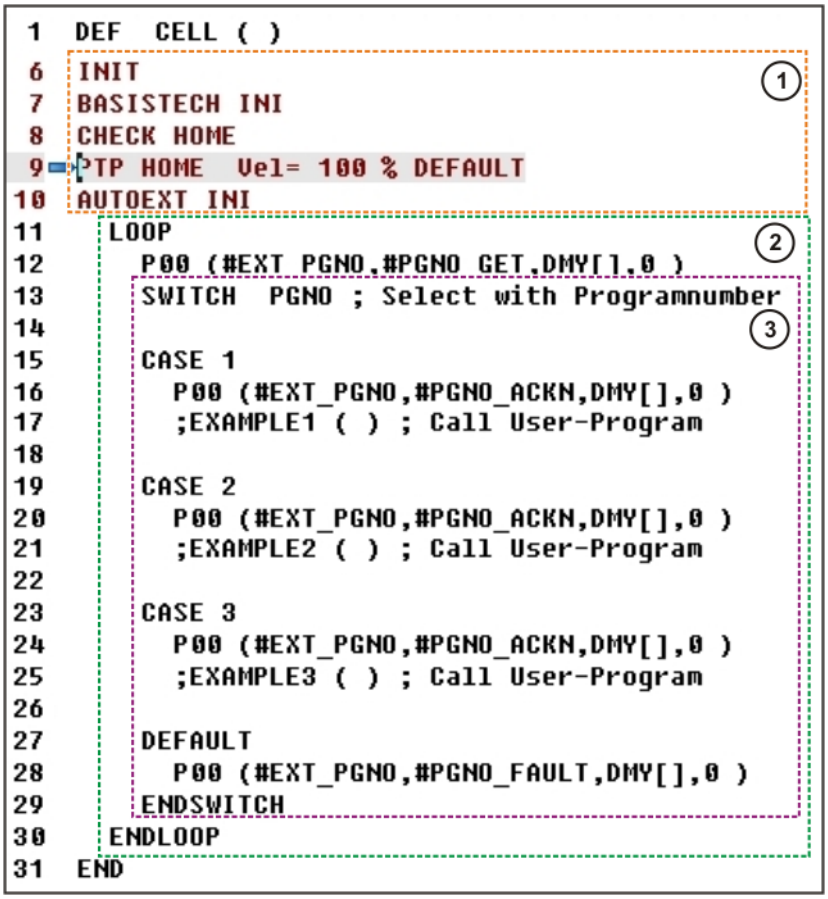


Fig. 14-3: Cell program

1	Initialization and home position ■ Initialization of the basic parameters ■ Check of the robot position after the home position ■ Initialization of the Automatic External interface
2	Endless loop: ■ Polling of the program number by the "P00" module ■ Entry into the selection loop with the program number determined.
3	Program number selection loop ■ Depending on the program number (stored in the variable “PG-NO”), the program jumps to the corresponding branch (“CASE”). ■ The robot program entered in the branch is then executed. ■ Invalid program numbers cause the program to jump to the default branch. ■ Once executed, the loop is repeated.

Procedure

1. Switch to the user group “Expert”.
2. Open CELL.SRC.

3. In the “CASE” sections, replace the name “EXAMPLE” with the name of the program that is to be called via the respective program number. Delete the semicolon in front of the name.

```
LOOP
  P00 (#EXT_PGNO,#PGNO_GET,DMY[],0 )
  SWITCH  PGNO ; Select with Programnumber

  CASE 1
    P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
    main()

  CASE 2
    P00 (#EXT_PGNO,#PGNO_ACKN,DMY[],0 )
    body_38()
    body_515()

  DEFAULT
    P00 (#EXT_PGNO,#PGNO_FAULT,DMY[],0 )
  ENDSWITCH
ENDLOOP
```

Fig. 14-4: Example of an adapted Cell program

4. Close the program and save the changes.

14.4 Questions: Working on a higher-level controller

What you should now know:

1. What is the prerequisite for communicating with a PLC?

. . . . . .

. . . . . .

2. What global subprogram is used to transfer the program number from the PLC?

. . . . . .

. . . . . .

3. What can be programmed in the “CASEs” of the SWITCH CASE loop of the CELL.SRC program? What cannot/must not be programmed?

. . . . . .

. . . . . .

15 Appendix

15.1 Programming motions with collision detection

Description

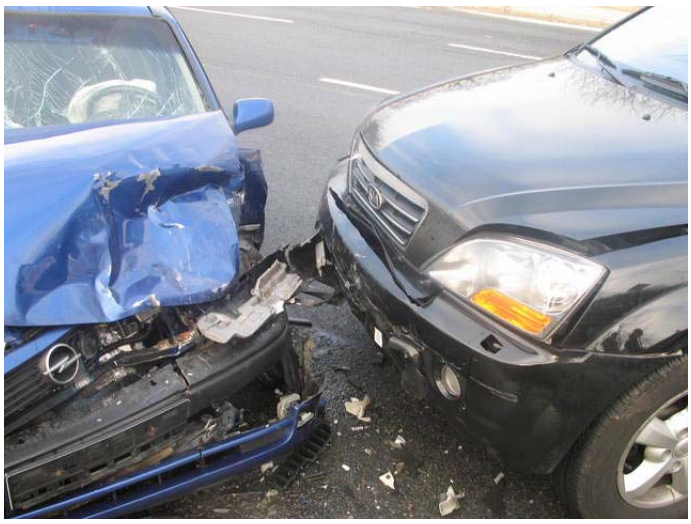


Fig. 15-1: Collision

Monitoring of axis torques is implemented in robotics in order to detect whether the robot has collided with an object. This collision is undesirable in most cases and can result in destruction of the robot, tool or components.

Collision detection

- If the robot collides with an object, the robot controller increases the axis torques in order to overcome the resistance. This can result in damage to the robot, tool or other objects.
- Collision detection reduces the risk and severity of such damage. It monitors the axis torques.
- The user can define the procedure to be executed after a collision once the algorithm has detected a collision and the robot has stopped:
  - The robot stops with a STOP 1.
  - The robot controller calls the program tm_useraction. This is located in the Program folder and contains the HALT statement. Alternatively, the user can program other reactions in the program tm_useraction.
- The robot controller automatically calculates the tolerance range.
- A program must generally be executed 2 or 3 times before the robot controller has calculated a practicable tolerance range.
- The user can define an offset via the user interface for the tolerance range calculated by the robot controller.
- If the robot is not operated for a longer period (e.g. over the weekend), the motors, gear units, etc., cool down. Different axis torques are required in the first few runs after such a break than in the case of a robot that is already at operating temperature. The robot controller automatically adapts the collision detection to the changed temperature.

Limitations

- Collision detection is **not** possible in **T1 mode**.
- Collision detection is not possible for HOME positions and other global positions.
- Collision detection is not possible for external axes.
- Collision detection is not possible during backward motion.

- High axis torques arise when the stationary robot starts to move. For this reason, the axis torques are not monitored in the starting phase (approx. 700 ms).
- The collision detection function reacts much less sensitively for the first 2 or 3 program executions after the program override value has been modified. Thereafter, the robot controller has adapted the tolerance range to the new program override.

Principle of collision detection

Teaching a program with collision detection

- Acceleration adaptation must be activated with the system variable `$ADAP_ACC`.
  - The system variable is located in the file C:\KRC\Roboter\KRC\R1\MaDa\$ROBCOR.DAT
  - `$ADAP_ACC = #NONE` Acceleration adaptation not activated
  - `$ADAP_ACC = #STEP1` Dynamic model without kinetic energy
  - `$ADAP_ACC = #STEP2` Dynamic model with kinetic energy
- To activate collision detection for a motion, the parameter **Collision detection** must be set to TRUE during programming. This can be seen from the addition `CD` in the program code:

```
PTP P2  Vel= 100 % PDAT1 Tool[1] Base[1] CD
```

> The parameter **Collision detection** is only available if the motion is programmed via an inline form.

- The tolerance range is only calculated for motion blocks that have been executed completely.

Setting the offset values

- An offset for the torque and for the impact can be defined for the tolerance range.
- **Torque:** The torque is effective if the robot meets a continuous resistance. Examples:
  - The robot collides with a wall and pushes against the wall.
  - The robot collides with a container. The robot pushes against the container and moves it.
- **Impact:** The impact is effective if the robot meets a brief resistance. Example:
  - The robot collides with a panel which is sent flying by the impact.
- The lower the offset, the more sensitive the reaction of the collision detection.
- The higher the offset, the less sensitive the reaction of the collision detection.

> If the collision detection reacts too sensitively, do not immediately increase the offset. Instead, recalculate the tolerance range first and test whether the collision detection now reacts as desired.

- Option window “Collision detection”

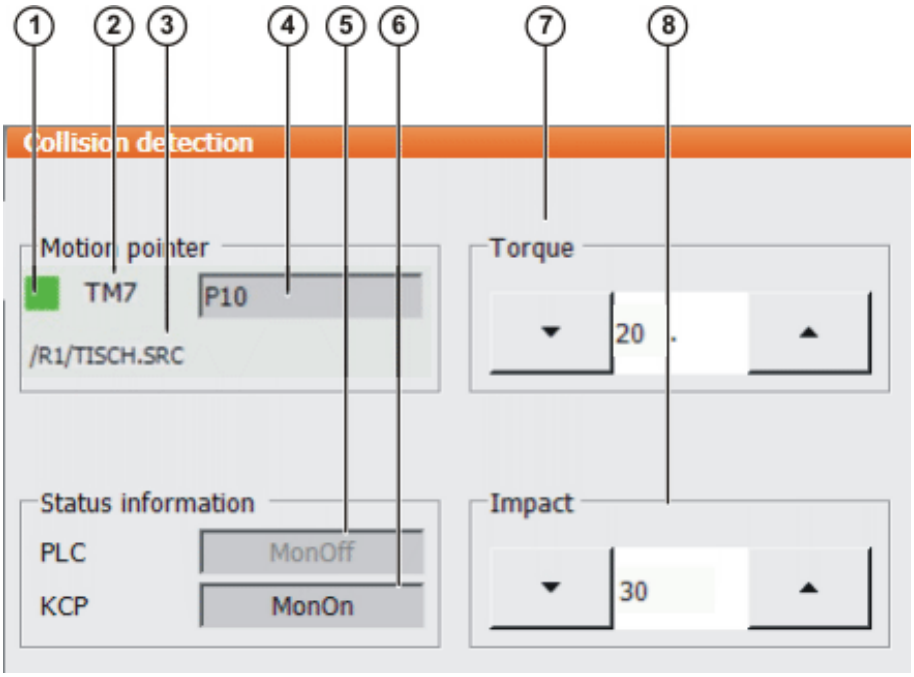


Fig. 15-2: Option window “Collision detection”

The values in the option window **Collision detection** do not always refer to the current motion. Deviations are particularly possible in the case of points which are close together and approximated motions.

Item	Description
1	The button indicates the status of a motion. ▪ Red: the current motion is not monitored. ▪ Green: the current motion is monitored. ▪ Orange: a button for setting the numeric value of the torque or impact on the left or right has been pressed. The window remains focused on the motion and the offset can be modified. The change can be applied by pressing **Save**. ▪ Pixelated: a program must generally be executed 2 or 3 times before the robot controller has calculated a practicable tolerance range. While the robot controller is in the learning phase, the button remains pixelated.
2	Number of the TMx variable The robot controller creates a TMx variable for each motion block in which the parameter **Collision detection** is set to TRUE. TMx contains all the values for the tolerance range of this motion block. If 2 motion blocks refer to the same point Px, the robot controller creates 2 TMx variables.
3	Path and name of the selected program
4	Point name

Item	Description
5	This box is only active in “Automatic External” mode. It appears gray in all other modes. **MonOn**: collision detection has been activated by the PLC. If collision detection is activated by the PLC, the PLC sends the input signal **sTQM_SPSACTIVE** to the robot controller. The robot controller responds with the output signal **sTQM_SPSSTATUS**. The signals are defined in the file $config.dat. **Note:** Collision detection is only active in Automatic External mode if both the **PLC** box and the **KCP** box show the entry **MonOn**.
6	**MonOn**: collision detection has been activated from the KCP. **Note:** Collision detection is only active in Automatic External mode if both the **PLC** box and the **KCP** box show the entry **MonOn**.
7	Offset for the torque. The lower the offset, the more sensitive the reaction of the collision detection. Default value: 20. The window remains focused on the motion and the offset can be modified. See also: (>>> "Adapting the offset for motions" Page 269). The change can be applied by pressing **Save**. **N.A.**: the option **Collision detection** in the inline form is set to FALSE for this motion.
8	Offset for the impact. The lower the offset, the more sensitive the reaction of the collision detection. Default value: 30. The window remains focused on the motion and the offset can be modified. See also (>>> "Adapting the offset for motions" Page 269). The change can be applied by pressing **Save**. **N.A.**: the option **Collision detection** in the inline form is set to FALSE for this motion.

Button	Description
Activate	Activates collision detection. This button is not displayed if the torque or impact has been changed, but the changes have not yet been saved.
Deactivate	Deactivates collision detection. This button is not displayed if the torque or impact has been changed, but the changes have not yet been saved.
Save	Saves changes to the torque and/or impact.
Cancel	Rejects changes to the torque and/or impact.

Activating collision detection in the inline form

1. Create a motion using an inline form.
2. Open the option window “Frames” and activate collision detection.

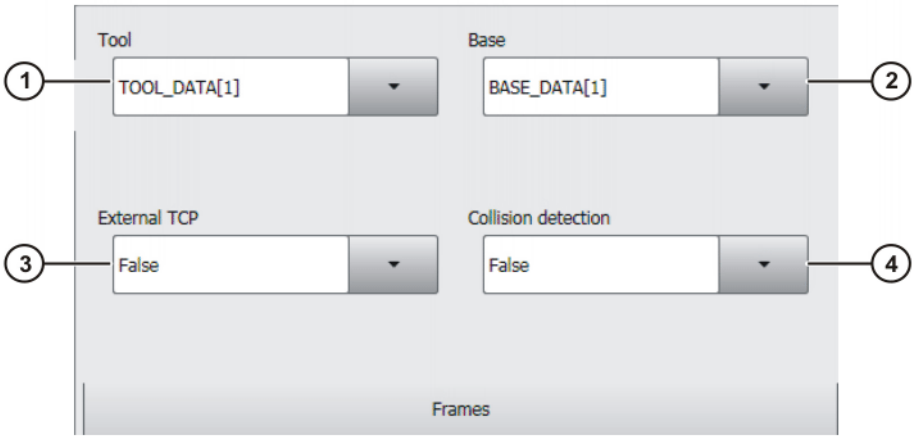


Fig. 15-3: Option window: Frames

Item	Description
1	Tool selection. If **True** in the box **External TCP**: workpiece selection. Range of values: [1] … [16]
2	Base selection. If **True** in the box **External TCP**: fixed tool selection. Range of values: [1] … [32]
3	Interpolation mode ▪ **False**: The tool is mounted on the flange. ▪ **True**: The tool is a fixed tool.
4	Collision detection ▪ **True**: For this motion, the robot controller calculates the axis torques. These are required for collision detection. ▪ **False**: For this motion, the robot controller does not calculate the axis torques. Collision detection is thus not possible for this motion.

3. Complete the motion.

Calculating the tolerance range and activating collision detection

1. In the main menu, select **Configuration** > **Miscellaneous** > **Collision detection**.
   (>>> Fig. 15-2)
2. The box **KCP** must contain the entry **MonOff**. If this is not the case, press **Deactivate**.
3. Start the program and execute it several times. After 2 or 3 program executions, the robot controller has calculated a practicable tolerance range.
4. Press **Activate**. The box **KCP** in the **Collision detection** window now contains the entry **MonOn**.
   Save the configuration by pressing **Close**.

Adapting the offset for motions

1. Select program.
2. In the main menu, select **Configuration** > **Miscellaneous** > **Collision detection**.
3. The offset for a motion can be modified while a program is running: If the desired motion is displayed in the **Collision detection** window, press the arrow keys in the **Torque or Impact** box. The window remains focused on this motion. Change the offset using these buttons.

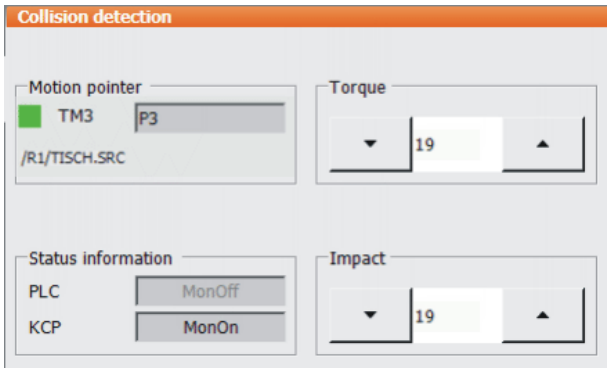


Fig. 15-6: Modified collision detection values

Alternatively, a block selection to the desired motion can be carried out.

4. Save the change by pressing **Save**.
5. Save the configuration by pressing **Close**.
6. Set the original operating mode and program run mode.

15.2 Programming time-distance functions

General

A time-distance function can be used to set an output at a specific point on the path without interrupting the robot motion. A distinction is made between “static” (SYN OUT) and “dynamic” (SYN PULSE) switching. In the case of SYN OUT 5 switching, the same signal is switched as with SYN PULSE 5. It is only the switching method that differs. The selection options PATH, START and END are explained below.

In practice, it is mainly the option PATH that is used; the training thus focuses on this option.

The options START and END are contained in the documentation for the sake of completeness.

Option PATH

With the option PATH, a switching action can be triggered relative to the end point of a motion block. The switching action can be shifted in space and/or delayed or brought forward. The reference motion block can be a LIN or CIRC motion. It must **not** be a PTP motion.

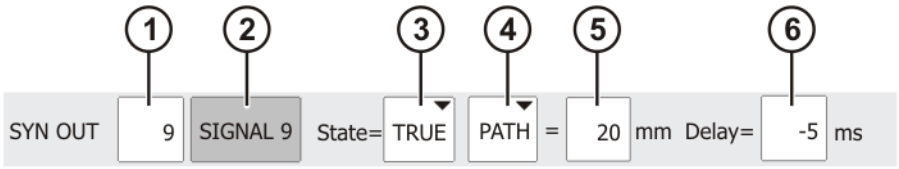


Fig. 15-7: Inline form “SYN OUT”, option “PATH”

Item	Description	Range of values
1	Output number	1 … 4096
2	If a name exists for the output, this name is displayed. Only for the user group “Expert”: A name can be entered by pressing the **Longtext** softkey.	Freely selectable
3	State to which the output is switched	TRUE, FALSE

Item	Description	Range of values
4	Point at which switching is carried out ■ **PATH**: Switching is carried out relative to the end point of the motion block.	Option PATH START, END
5	Switching action offset **Note:** The specification of the location is relative to the end point of the motion block. The position of the switching point does not vary, despite the changed velocity of the robot. This compensates for the response time of the controlled device (e.g. adhesive gun).	-2000 … +2000 mm
6	Switching action delay **Note:** The delay is relative to the offset. The time specification is absolute, i.e. the switching point varies according to the velocity of the robot.	-1000 … +1000 ms

SYN PULSE can be used to trigger **a pulse** at the start or end point of the motion. The pulse can be shifted in time and/or space, i.e. it does not have to be triggered exactly at the point, but can also be triggered before or after it.

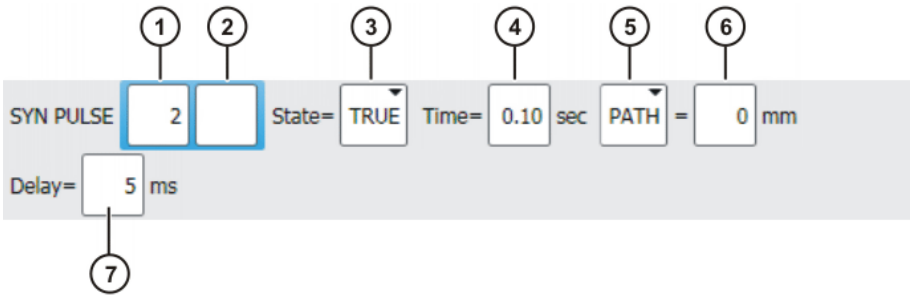


Fig. 15-8: Inline form “SYN PULSE”

Item	Description	Range of values
1	Output number	1 … 4096
2	If a name exists for the output, this name is displayed. Only for the user group “Expert”: A name can be entered by pressing **Long text**. The name is freely selectable.	Freely selectable
3	State to which the output is switched	TRUE, FALSE
4	Duration of the pulse	0.1 ... 3 s
5	Point to which SYN PULSE refers: ■ **PATH**: SYN PULSE refers to the end point. An offset in space is also possible.	Option PATH START, END

Item	Description	Range of values
6	Switching action offset **Note:** The specification of the location is relative to the end point of the motion block. The position of the switching point does not vary, despite the changed velocity of the robot.	-2000 … +2000 mm
7	Switching action delay **Note:** The delay is relative to the offset. The time specification is absolute. The switching point varies according to the velocity of the robot.	-1000 … +1000 ms

Effect of the switching option PATH

Example program:

A milling tool has to be switched on the path. Machining of the workpiece is to begin 20 mm after P2. For the milling tool to have reached its full speed 20 mm (Path = 20) after P2, it must already be switched on 5 ms beforehand (Delay = -5 ms).

The switching option **Path** always refers to the **end point**.

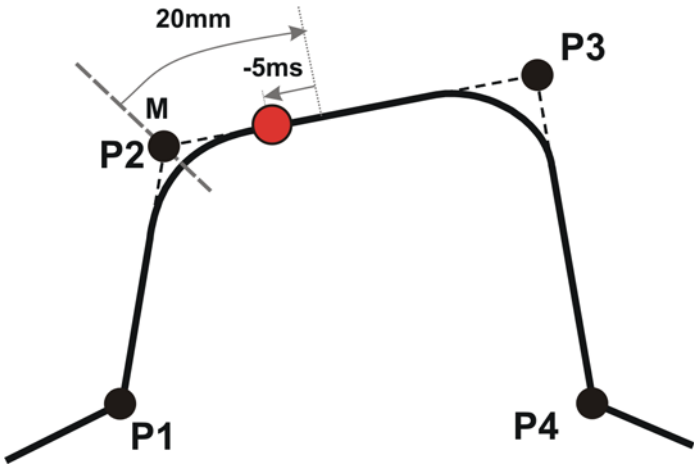


Fig. 15-9: SYN OUT PATH switching point

```
LIN P1 VEL=0.3m/s CPDAT1 TOOL[1] BASE[1]
;Switching function relative to P2
SYN OUT 9 'SIGNAL 9' Status= True Path=20 Delay=-5ms
LIN P2 CONT VEL=0.3m/s CPDAT2 TOOL[1] BASE[1]
LIN P3 CONT VEL=0.3m/s CPDAT3 TOOL[1] BASE[1]
LIN P4 VEL=0.3m/s CPDAT4 TOOL[1] BASE[1]
```

Switching limits

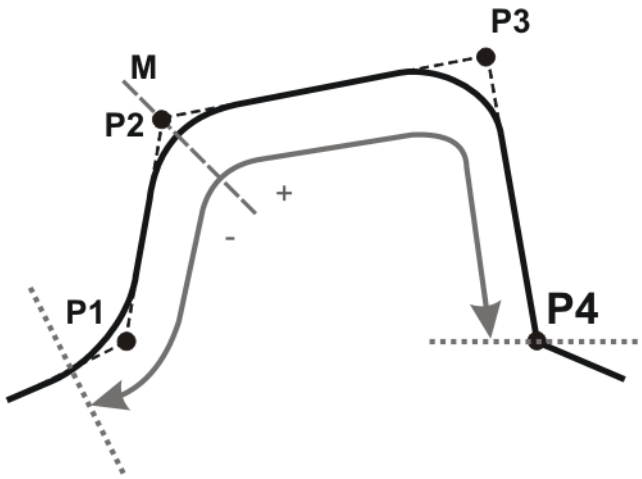


Fig. 15-10: SYN OUT PATH switching limits

Procedure

1. Position the cursor in the line after which the logic instruction is to be inserted.
2. Select the menu sequence **Commands** > **Logic** > **OUT** > **SYN OUT** or **SYN PULSE**.
3. Set the parameters in the inline form.
4. Save instruction with **Cmd Ok**.

Option Start/End

A switching action can be triggered relative to the start or end point of a motion block. The switching action can be delayed or brought forward **in time**. The reference motion block can be a LIN, CIRC or PTP motion.

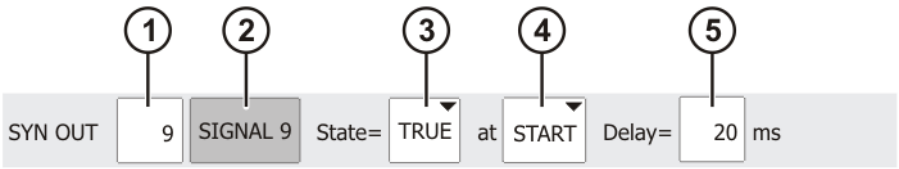


Fig. 15-11: Inline form “SYN OUT”, option “START”

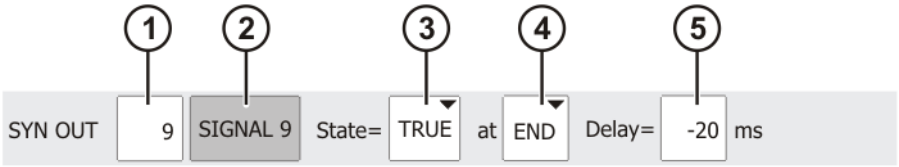


Fig. 15-12: Inline form “SYN OUT”, option “END”

Item	Description	Range of values
1	Output number	1 … 4096
2	If a name exists for the output, this name is displayed. Only for the user group “Expert”: A name can be entered by pressing the **Longtext** softkey.	Freely selectable
3	State to which the output is switched	TRUE, FALSE

Item	Description	Range of values
4	Point at which switching is carried out ■ **START**: Switching is carried out relative to the start point of the motion block. ■ **END**: Switching is carried out relative to the end point of the motion block.	START, END PATH (>>> Fig. 15-7)
5	Switching action delay **Note:** The time specification is absolute. The position of the switching point thus varies according to the velocity of the robot.	-1000 … +1000 ms

Effect of the switching options Start/End

- **Example program 1:**
  Option Start

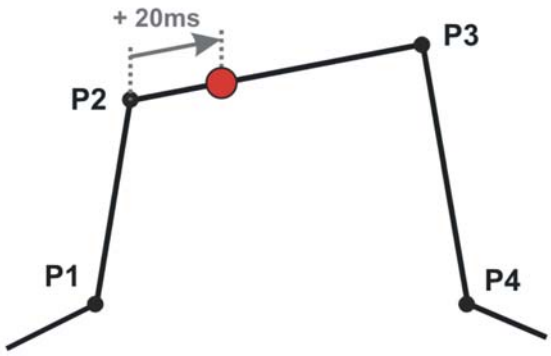


Fig. 15-13: SYN OUT Start with positive delay

```
LIN P1 VEL=0.3m/s CPDAT1 TOOL[1] BASE[1]
LIN P2 VEL=0.3m/s CPDAT2 TOOL[1] BASE[1]
;Switching function relative to P2
SYN OUT 8 'SIGNAL 8' State= TRUE at Start Delay=20ms
LIN P3 VEL=0.3m/s CPDAT3 TOOL[1] BASE[1]
LIN P4 VEL=0.3m/s CPDAT4 TOOL[1] BASE[1]
```

- **Example program 2:**
  Option Start with CONT and positive delay

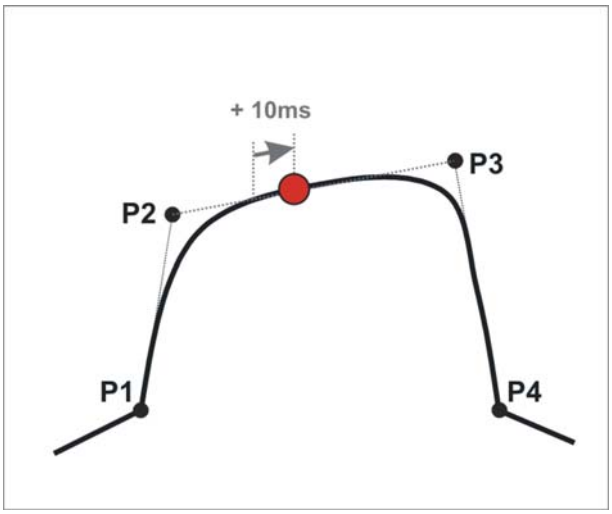


Fig. 15-14: SYN OUT Start with CONT and positive delay

```
LIN P1 VEL=0.3m/s CPDAT1 TOOL[1] BASE[1]
LIN P2 CONT VEL=0.3m/s CPDAT2 TOOL[1] BASE[1]
;Switching function relative to P2
SYN OUT 8 'SIGNAL 8' State= TRUE at Start Delay=10ms
LIN P3 CONT VEL=0.3m/s CPDAT3 TOOL[1] BASE[1]
LIN P4 VEL=0.3m/s CPDAT4 TOOL[1] BASE[1]
```

- **Example program 3:**
  Option End with negative delay

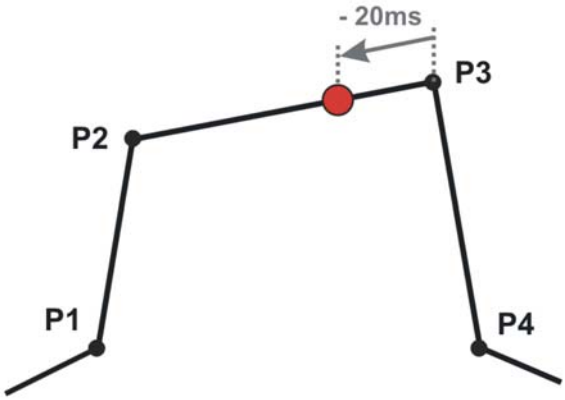


Fig. 15-15: SYN OUT END with negative delay

```
LIN P1 VEL=0.3m/s CPDAT1 TOOL[1] BASE[1]
LIN P2 VEL=0.3m/s CPDAT2 TOOL[1] BASE[1]
;Switching function relative to P3
SYN OUT 9 'SIGNAL 9' Status= TRUE at End Delay=-20ms
LIN P3 VEL=0.3m/s CPDAT3 TOOL[1] BASE[1]
LIN P4 VEL=0.3m/s CPDAT4 TOOL[1] BASE[1]
```

- **Example program 4:**
  Option End with CONT and negative delay

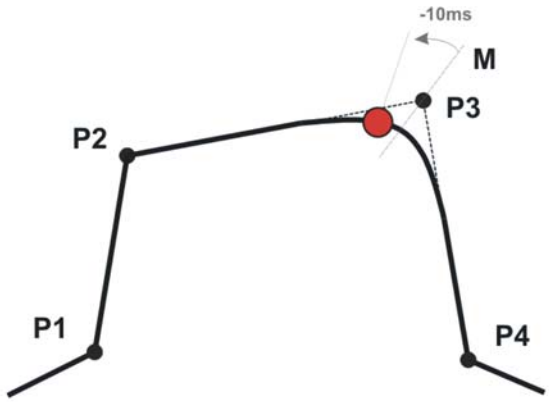


Fig. 15-16: SYN OUT with option END with negative delay

```
LIN P1 VEL=0.3m/s CPDAT1 TOOL[1] BASE[1]
LIN P2 VEL=0.3m/s CPDAT2 TOOL[1] BASE[1]
;Switching function relative to P3
SYN OUT 9 'SIGNAL 9' Status= TRUE at End Delay=-10ms
LIN P3 CONT VEL=0.3m/s CPDAT3 TOOL[1] BASE[1]
LIN P4 VEL=0.3m/s CPDAT4 TOOL[1] BASE[1]
```

- **Example program 5:**
  Option End with CONT and positive delay

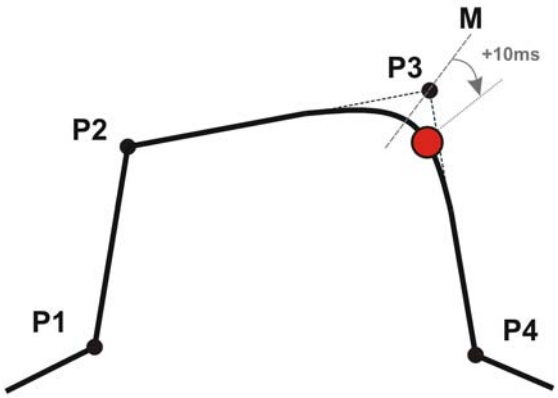


Fig. 15-17: SYN OUT with option END and positive delay

```
LIN P1 VEL=0.3m/s CPDAT1 TOOL[1] BASE[1]
LIN P2 VEL=0.3m/s CPDAT2 TOOL[1] BASE[1]
;Switching function relative to P3
SYN OUT 9 'SIGNAL 9' Status= TRUE at End Delay=10ms
LIN P3 CONT VEL=0.3m/s CPDAT3 TOOL[1] BASE[1]
LIN P4 VEL=0.3m/s CPDAT4 TOOL[1] BASE[1]
```

Switching limits with time-distance functions

- **Switching limits**
  without CONT

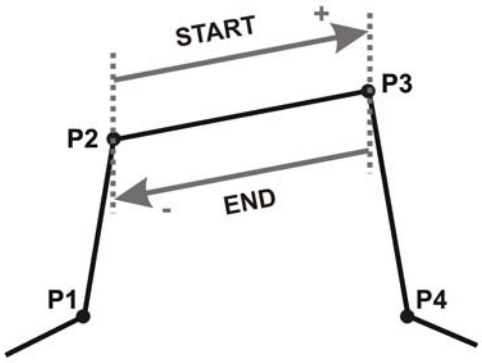


Fig. 15-18: Switching limits, option Start/End without CONT

- **Switching limits with CONT:**
  with CONT

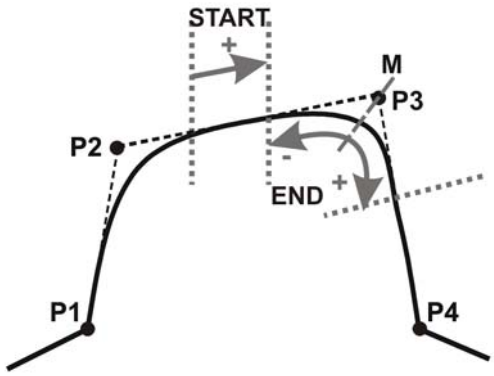


Fig. 15-19: Switching limits, option Start/End with CONT

15.3 Configuring and implementing Automatic External

Working with a higher-level controller

- See: (>>> 14.2 "Preparation for program start from PLC" Page 259)

Using the inputs/ outputs of the Automatic External interface

Overview of the principal signals of the interface

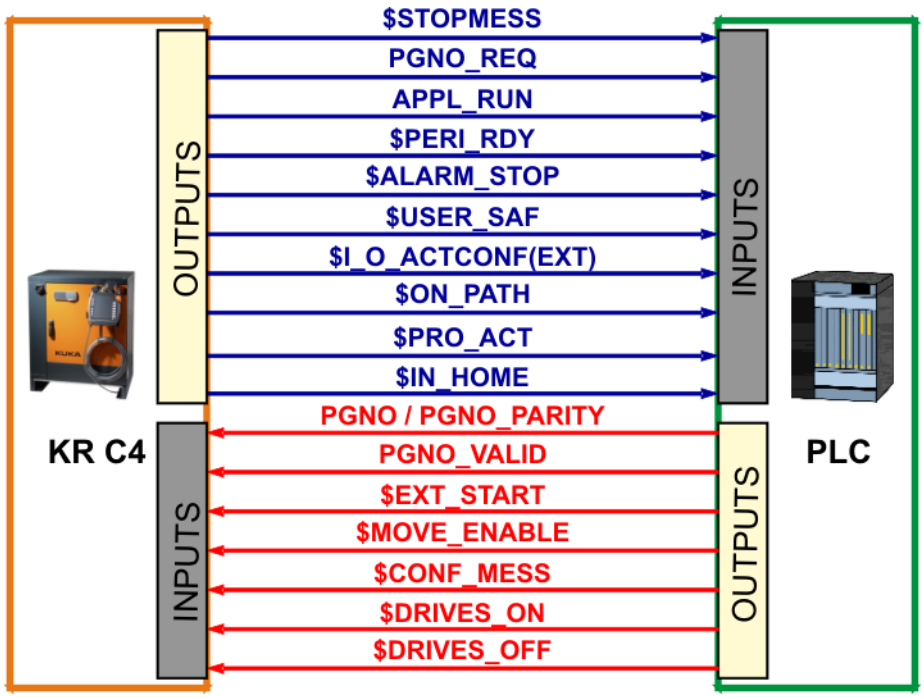


Fig. 15-20: Overview of the principal Automatic External signals

Inputs (from robot controller's point of view)

- **PGNO_TYPE** – Program number type

  This variable defines the format in which the program number sent by the higher-level controller is read.

Value	Description	Example
1	Read as binary number. The program number is transmitted by the higher-level controller as a binary coded integer.	0 0 1 0 0 1 1 1 => PGNO = 39
2	Read as BCD value. The program number is transmitted by the higher-level controller as a binary coded decimal.	0 0 1 0 0 1 1 1 => PGNO = 27
3	Read as "1 of n"*. The program number is transmitted by the higher-level controller or the periphery as a "1 of n" coded value.	0 0 0 0 0 0 0 1 => PGNO = 1 0 0 0 0 1 0 0 0 => PGNO = 4

  * When using this transmission format, the values of PGNO_REQ, PGNO_PARITY and PGNO_VALID are not evaluated and are thus of no significance.

- **PGNO_LENGTH** – Program number length

  This variable determines the number of bits in the program number sent by the higher-level controller. Range of values: 1 … 16.

  If PGNO_TYPE has the value 2, only 4, 8, 12 and 16 are permissible values for the number of bits.

- **PGNO_PARITY** – Program number parity bit

  Input to which the parity bit is transferred from the higher-level controller.

Input	Function
Negative value	Odd parity
0	No evaluation
Positive value	Even parity

If PGNO_TYPE has the value 3, PGNO_PARITY is not evaluated.

- **PGNO_VALID** – Program number valid

  Input to which the command to read the program number is transferred from the higher-level controller.

Input	Function
Negative value	Number is transferred at the falling edge of the signal.
0	Number is transferred at the rising edge of the signal on the EXT_START line.
Positive value	Number is transferred at the rising edge of the signal.

- **$EXT_START** – External start

  If the I/O interface is active, this input can be set to start or continue a program (normally CELL.SRC).

Only the rising edge of the signal is evaluated.

⚠ WARNING There is no BCO run in Automatic External mode. This means that the robot moves to the first programmed position after the start at the programmed (not reduced) velocity and does not stop there.

- **$MOVE_ENABLE** – Motion enable

  This input is used by the higher-level controller to check the robot drives.

Signal	Function
TRUE	Jogging is possible in modes T1 and T2. Program execution is possible.
FALSE	All drives are stopped and all active commands inhibited.

If the drives have been switched off by the higher-level controller, the message “GENERAL MOTION ENABLE” is displayed. It is only possible to move the robot again once this message has been reset and another external start signal has been given.

During commissioning, the variable $MOVE_ENABLE is often configured with the value $IN[1025]. If a different input is not subsequently configured, no external start is possible.

- **$CONF_MESS** – Message acknowledgement

  Setting this input enables the higher-level controller to acknowledge error messages automatically as soon as the cause of the error has been eliminated.

Only the rising edge of the signal is evaluated.

- **$DRIVES_ON** – Drives on

  If there is a high-level pulse of at least 20 ms duration at this input, the higher-level controller switches on the robot drives.

- **$DRIVES_OFF** – Drives off

  If there is a low-level pulse of at least 20 ms duration at this input, the higher-level controller switches off the robot drives.

Outputs (from robot controller's point of view)

- **$ALARM_STOP** – Emergency Stop

  This output is reset in the following EMERGENCY STOP situations:

  - The EMERGENCY STOP button on the KCP is pressed. (Int. E-Stop)
  - External E-STOP

In the case of an EMERGENCY STOP, the nature of the EMERGENCY STOP can be recognized from the states of the outputs **$ALARM_STOP** and **Int. E-Stop**:

- Both outputs are FALSE: the EMERGENCY STOP was triggered on the KCP.
- **$ALARM_STOP** is FALSE, **Int. E-Stop** is TRUE: external EMERGENCY STOP.

- **$USER_SAF** – Operator safety / safety gate

  This output is reset if the safety fence monitoring switch is opened (AUT mode) or an enabling switch is released (T1 or T2 mode).

- **$PERI_RDY** – Drives ready

  By setting this output, the robot controller communicates to the higher-level controller the fact that the robot drives are switched on.

- **$STOPMESS** – Stop messages

  This output is set by the robot controller in order to communicate to the higher-level controller any message occurring which requires the robot to be stopped. (Examples: EMERGENCY STOP, Motion enable or Operator safety)

- **$I_O_ACTCONF** – Automatic External active

  This output is TRUE if *Automatic External* mode is selected and the input $I_O_ACT is TRUE (normally always at $IN[1025]).

- **$PRO_ACT** – Program is active/running

  This output is set whenever a process is active at robot level. The process is therefore active as long as a program or an interrupt is being processed. Program processing is set to the inactive state at the end of the program only after all pulse outputs and all triggers have been processed.

- **PGNO_REQ** – Program number request

  A change of signal at this output requests the higher-level controller to send a program number.

  If PGNO_TYPE has the value 3, PGNO_REQ is not evaluated.

- **APPL_RUN** – Application program running

  By setting this output, the robot controller communicates to the higher-level controller the fact that a program is currently being executed.

- **$IN_HOME** – Robot in HOME position

  This output communicates to the higher-level controller whether or not the robot is in its HOME position.

- **$ON_PATH** – Robot is on the path

  This output remains set as long as the robot stays on its programmed path. The output ON_PATH is set after the BCO run. This output remains set until the robot leaves the path, the program is reset or block selection is carried out. The ON_PATH signal has no tolerance window, however; as soon as the robot leaves the path the signal is reset.

Principle of Automatic External communication

Overview of the complete procedure

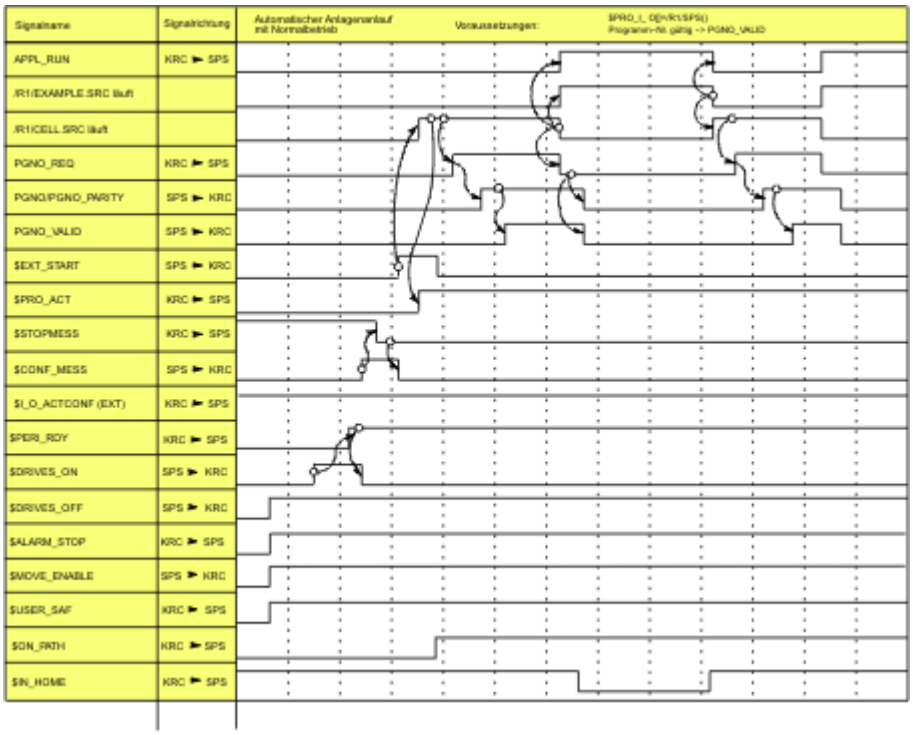


Fig. 15-21: Automatic system start and normal operation with program number acknowledgement by means of PGNO_VALID

Subdivision into subareas

1. Switch on drives
2. Acknowledge messages
3. Start Cell program
4. Transfer program number and execute application

Each of these areas is implemented as a handshake. Conditions must be met, the PLC sends signals and the robot controller communicates the corresponding robot states to the PLC.

Fig. 15-22: Handshake

It is advisable to use these predefined handsakes.

Switch on drives

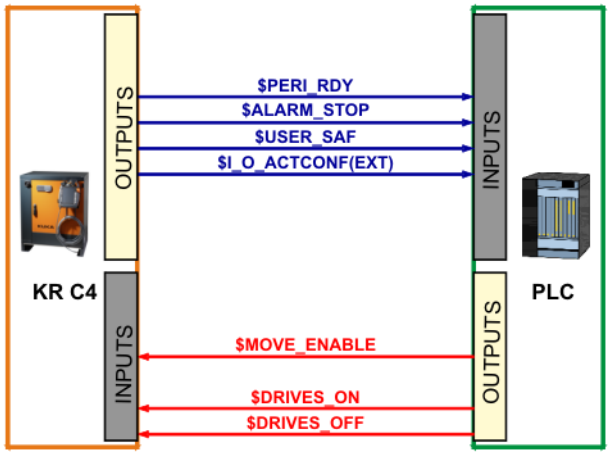


Fig. 15-25

- Preconditions
  - $USER_SAF – Safety gate closed
  - $ALARM_STOP – No Emergency Stop active
  - $I_O_ACTCONF – Automatic External active
  - $MOVE_ENABLE – Motion enable present
  - $DRIVES_OFF – Drives off not activated
- Switch on drives

  $DRIVES_ON – Signal to switch drives on for at least 20 ms
- Drives ready

  $PERI_RDY – The $DRIVES_ON signal is reset as soon as the check-back signal for the drives is received.

Acknowledge messages

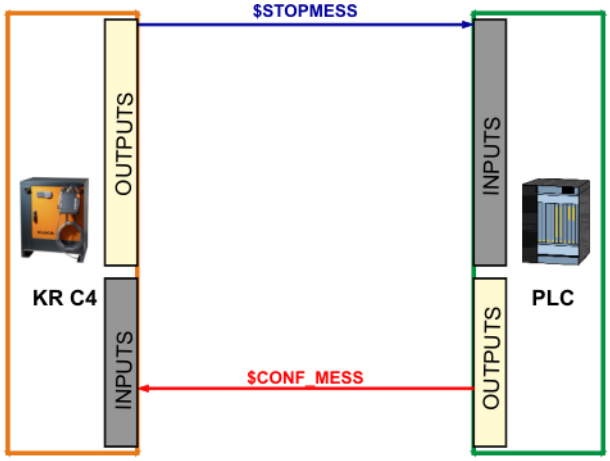


Fig. 15-29

- Preconditions

  $STOPMESS – Stop message is active
- Acknowledge message

  $CONF_MESS – Acknowledge message
- Acknowledgeable messages are cleared

  $STOPMESS – Stop message is no longer active; $CONF_MESS can now be reset.

Start program (CELL.SRC) externally

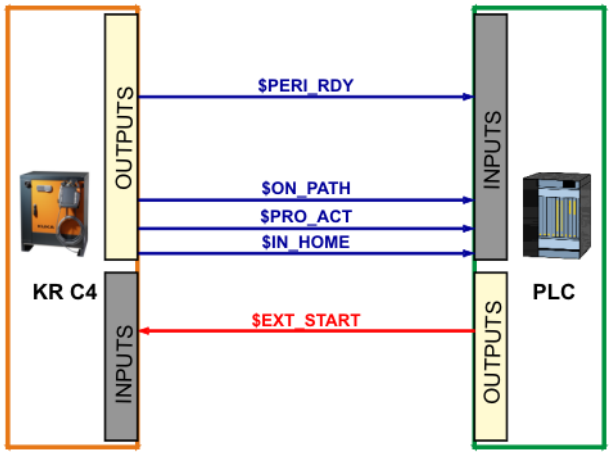


Fig. 15-34

- Preconditions
  - $PERI_RDY – Drives are ready
  - $IN_HOME – Robot is in HOME position
  - No $STOPMESS – No stop message is active
- External start

  $EXT_START – Activate external start (positive edge)
- CELL program running
  - $PRO_ACT – Signals that CELL program is running
  - $ON_PATH – The $EXT_START signal is reset as soon as the robot is on its programmed path.

Execute program number transfer and application program

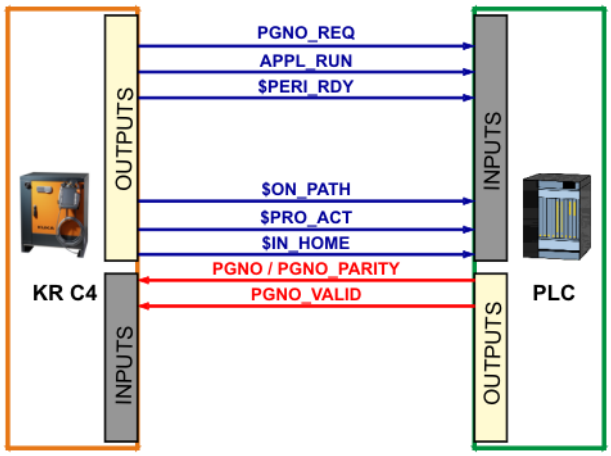


Fig. 15-41

- Preconditions
  - $PERI_RDY – Drives are ready
  - $PRO_ACT – CELL program is running
  - $ON_PATH – Robot is on the path
  - $IN_HOME – Robot is in HOME position, not required for restart
  - PGNO_REQ – Program number request is active
- Program number transfer and confirmation
  - Program number transfer

    (Correct data type (PGNO_TYPE), program number length (PGNO_LENGTH) and first bit for program nummer (PGNO_FBIT) are set)

- PGNO_VALID – Set program number valid (confirmation) (positive edge)

- Application program is running
  - APPL_RUN – Signals that application program is running
  - The robot leaves the HOME position. On termination of the application program, the robot returns to the HOME position.

Procedure

1. In the main menu, select **Configuration** > **Inputs/outputs** > **Automatic External**.
2. In the **Value** column, select the cell to be edited and press **Edit**.
3. Enter the desired value and save it by pressing **OK**.
4. Repeat steps 2 and 3 for all values to be edited.
5. Close the window. The changes are saved.

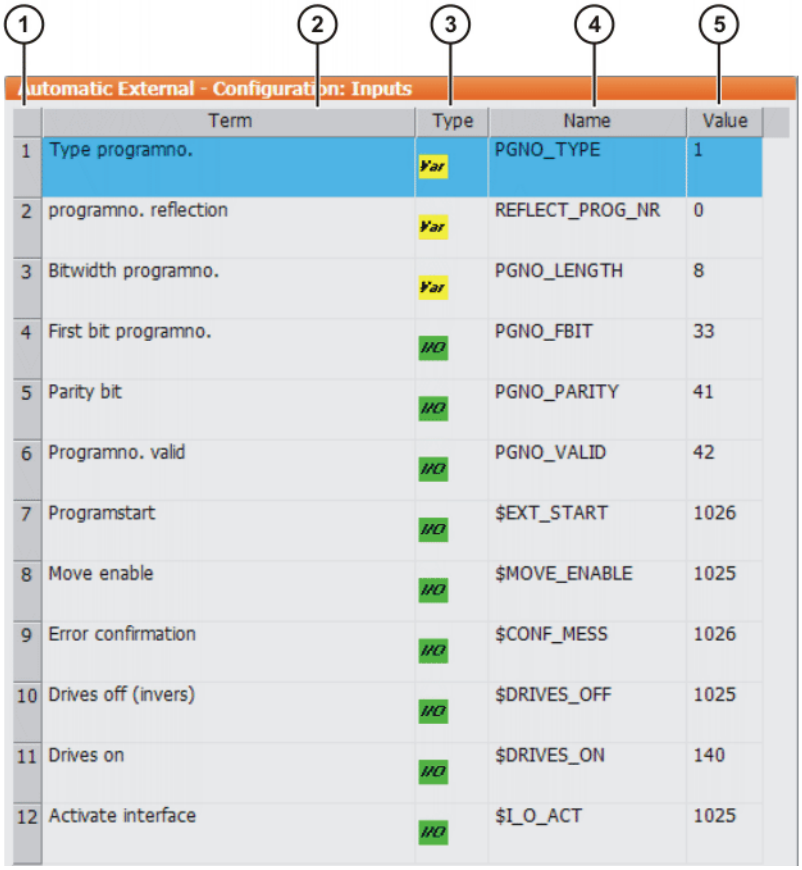


Fig. 15-44: Configuring Automatic External inputs

Item	Description
1	Number
2	Long text name of the input/output
3	Type ■ **Green**: Input/output ■ **Yellow**: Variable or system variable ($...)
4	Name of the signal or variable
5	Input/output number or channel number
6	The outputs are thematically assigned to tabs.

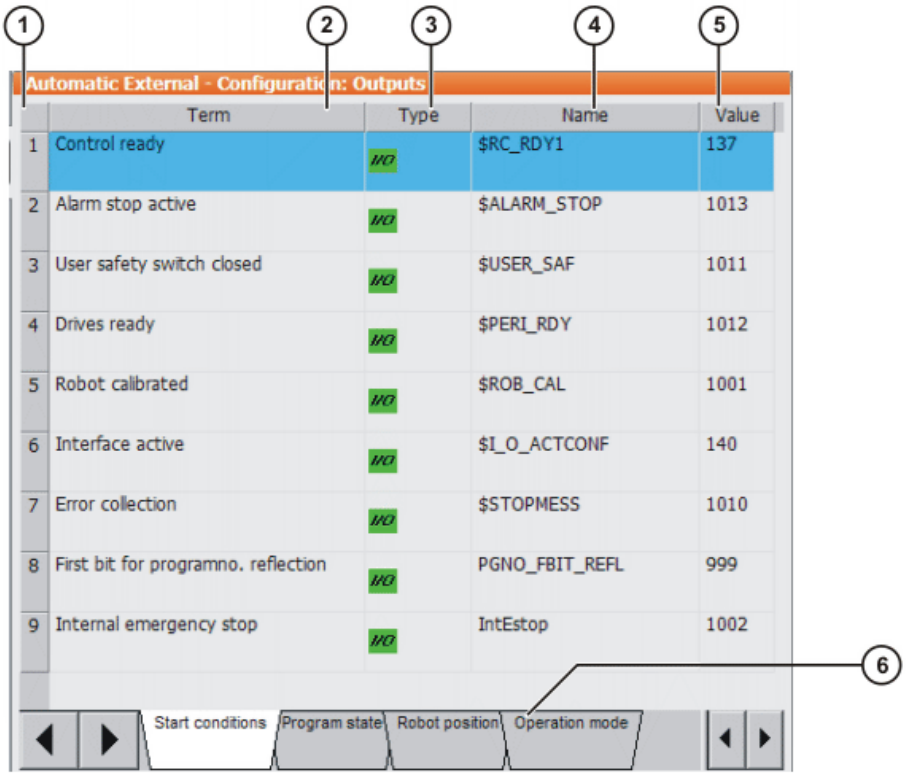


Fig. 15-45: Configuring Automatic External outputs

15.4 Abbreviations

The following list of the most important abbreviations and their meaning is intended to help you in your work with the KR C4 controller.

Term	Description
CCU	**C**abinet **C**ontrol **U**nit
CCUsr	**C**abinet **C**ontrol **U**nit **s**mall **r**obot
CIB	**C**abinet **I**nterface **B**oard
CIBsr	**C**abinet **I**nterface **B**oard **s**mall **r**obot
CK	**C**ustomer-built **K**inematics
CSP	**C**ontroller **S**ystem **P**anel
Dual NIC card	Dual network card
EDS	**E**lectronic **D**ata **S**torage (memory card)
EMD	**E**lectronic **M**astering **D**evice (formerly EMT) for mastering the robot
EMC	**E**lectro**M**agnetic **C**ompatibility
GBE	**G**iga**b**it **E**therNet
Catalog	Can contain different elements, e.g. templates, components, kinematic systems.
KCB	**K**UKA **C**ontroller **B**us
KEB	**K**UKA **E**xtension **B**us
KCP	**K**UKA **C**ontrol **P**anel (teach pendant), new term: smartPAD
KLI	**K**UKA **L**ine **I**nterface

Term	Description
KOI	**K**UKA **O**perator Panel **I**nterface
KONI	**K**UKA **C**ustomer **N**etwork **I** nterface
KPC	**K**UKA control **PC**
KPP	**K**UKA **P**ower **P**ack
KPPsr	**K**UKA **P**ower **P**ack **s**mall **r**obot
KRL	**K**UKA **R**obot **L**anguage (KUKA robot programming language)
KSB	**K**UKA **S**ystem **B**us
KSP	**K**UKA **S**ervo **P**ack
KSPsr	**K**UKA **S**ervo **P**ack **s**mall **r**obot
KSI	**K**UKA **S**ervice **I**nterface
KUKA.HMI	**H**uman **M**achine **I**nterface KUKA.HMI is the KUKA user interface.
FOC	**F**iber **O**ptic **C**able
MEMD	**M**icro **E**lectronic **M**astering **D**evice
MCFB	**M**otion **C**ontrol **F**unction **B**lock Program module for programming motion tasks. These modules are PLC-compliant and KUKA-specific.
MGU	**M**otor **G**ear **U**nit KUKA motor/gear combination for kinematic systems
OPI	**O**perator **P**anel **I**nterface (smartPAD connection)
PMB	**P**ower **M**anagement **B**oard
RCD	**R**esidual **C**urrent **D**evice
RDC	**R**esolver **D**igital **C**onverter
SATA	**S**erial **A**dvanced **T**echnology **A**ttachment (data bus between processor and hard drive)
SEMD	**S**tandard **E**lectronic **M**astering **D**evice
SIB	**S**afety **I**nterface **B**oard
SBC	**S**ingle **B**rake **C**ontrol
STO	**S**afe **T**orque **O**ff
SION	**S**afety **I**nput **O**utput **N**ode
USB	**U**niversal **S**erial **B**us (bus system for connecting additional devices to a computer)
UPS	**U**ninterruptible **P**ower **S**upply
WorkVisual Catalog Editor	Software for creating catalog elements for WorkVisual.

15.5 Additional exercises

15.5.1 Exercise: Gripper programming – pen

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Program statements for controlling and monitoring grippers and weld guns (KUKA.Gripper & SpotTech)
- Activate and work with the technology-specific status keys

Preconditions

The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of the KUKA.Gripper & SpotTech technology package

Task description

Carry out the following tasks: fetch and set down pen 1

1. Create two new programs with the names **Fetch_pen1** and **Set_down_pen1** (duplicate).
2. During programming, make use of the advantages of jogging in the tool direction.
3. Make sure that the jog velocity when fetching pens from the pen magazine and setting them down in the pen magazine does not exceed 0.3 m/s.
4. Before fetching a pen, carry out a safety query with regard to the gripper position.

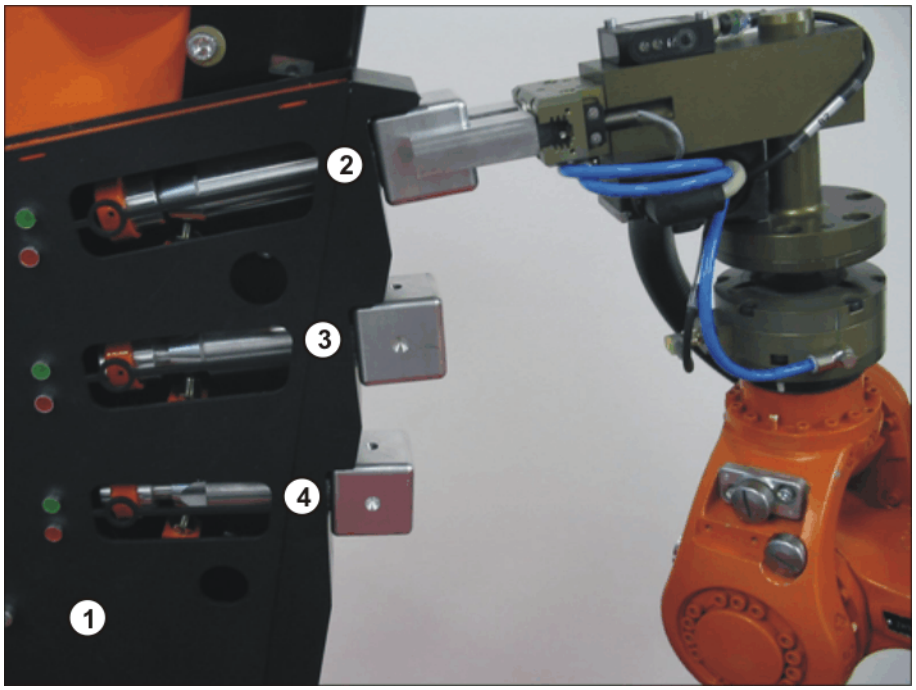


Fig. 15-46: Pen magazine

1	Pen magazine	2	Pen 1
3	Pen 2	4	Pen 3

What you should now know:

1. What is the difference between a wait time and “Gripper monitoring ON/OFF”?

. . . . . .

. . . . . .

2. You receive the notification message *Approximation not possible*. What are the possible causes of this?

. . . . . .

. . . . . .

. . . . . .

3. How many standard KUKA gripper types are there?

. . . . . .

. . . . . .

15.5.2 Exercise: Use of loops

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Programming of loops in KRL

Preconditions

The following are preconditions for successful completion of this exercise:

- Theoretical knowledge of the functionality of the different loop techniques in programming

Task description

1. Duplicate your program **Procedure** with the new name **Procedure_loop**.
2. Replace the *LOOP - ENDLOOP* loop with a *FOR - WHILE* loop.
3. For this, declare an integer variable as a loop counter.
4. Repeat the loop 5 times.
5. It must also be possible to exit the program execution early. Use $IN[11] as the break condition for this.

What you should now know:

1. In connection with SWITCH/CASE, there is also a “DEFAULT” statement. What is the purpose of the “DEFAULT” statement?

. . . . . .

. . . . . .

2. What command can you use with the FOR loop to set the increment?

. . . . . .

. . . . . .

3. What loops can be left using the “EXIT” command?

. . . . . .

. . . . . .

4. Which component can be omitted when using a branch? a. IF b. THEN c. ELSE d. ENDIF

. . . . . .

. . . . . .

5. What is wrong in the following example program?

. . . . . .

. . . . . .

```
IF $IN[14]==FALSE THEN
   $OUT[12]=TRUE
   GOTO MARKE1
ELSE
   $OUT[12]=FALSE
   GOTO MARKE2
ENDIF
WHILE $IN[17]==TRUE
   SPTP P1
   MARKE1: ...
ENDWHILE
MARKE2: …
SPTP HOME
```

15.5.3 Exercise: Constant velocity range and conditional stop

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Use logic commands in the spline inline form

Preconditions

The following are preconditions for successful completion of this exercise:

- Knowledge of motion programming with splines.
- Use of “Constant velocity range” and “Conditional stop” in the spline inline form

Task description

Carry out the following tasks:

1. Duplicate your program with the spline contour.
2. Define the green area as the constant velocity range.
3. Execute a conditional stop at the point on the path labeled in red.

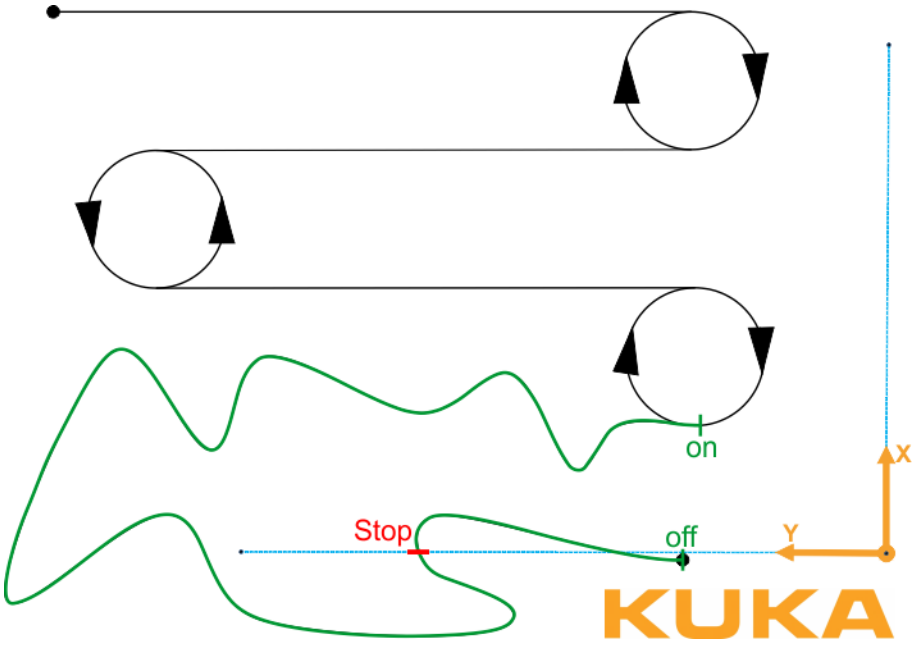


Fig. 15-47: Logic expanded with spline

4. Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.

15.5.4 Exercise: Automatic External

Aim of the exercise

On successful completion of this exercise, you will be able to carry out the following activities:

- Targeted integration of a robot program into Automatic External operation
- Adaptation of the “Cell” program
- Configuration of the Automatic External interface
- Recognition of sequence in Automatic External mode

Preconditions

The following are preconditions for successful completion of this exercise:

- Knowledge of how to edit the “Cell” program
- Knowledge of how to configure the Automatic External interface
- Theoretical knowledge of the signal sequence in Automatic External

Task description

1. Configure the Automatic External interface to the requirements of your control panel.
2. Expand your Cell program by any 3 modules, the functions of which you have verified beforehand.
3. Test your program in the modes T1, T2 and Automatic. Observe the relevant safety instructions.
4. Use buttons to simulate the functionality of the PLC.

Index

Symbols
$LDC_LOADED 58

A
Abbreviations 285
Acknowledgement message 19
Adapting the offset for motions 269
Administrator 219
Advance run 230
Appendix 265
Approximate positioning SCIRC 129
Approximate positioning SLIN 129
Approximate positioning SPTP 118
Approximate positioning, homogenous 182
Approximate positioning, mixed 182
Archiving 109
Automatic External 259, 277
Axis-specific jogging 28

B
Base calibration 87
Base coordinate system 35
Basic arithmetic operations 236
BCO 97
Bit operations 236
Block selection 155
Branches 250

C
Calibration of a fixed tool 208
Calibration of a robot-guided workpiece 210
Catalog 285
CCU 285
CCUsr 285
CIB 285
CIBsr 285
CIRC motion 125
CK 285
Collision detection 132, 135, 265, 269
Collision detection (menu item) 269
Collision detection, activating in the inline form 268
Collision detection, Automatic External 268
Collision detection, variable 267
Comment 222
Comparison operations 236
Conditional statements 250
Conditional stop 185
Configuring Automatic External, exercise 291
Connection manager 14
Constant velocity range 188
Coordinate system 35
Counting loop 245
CSP 285

D
DECL 230, 232
Declaration 230, 232
Declarations 229
Delete program 109
Dialog message 20
Displaying a variable, single 239
Dual NIC card 285
Duplicate program 108

E
EDS 285
EMC 285
EMD 285
EMERGENCY STOP 14
EMERGENCY STOP button 17
Endless loop 243
Exercise, approximate positioning 146
Exercise, base calibration – table 92
Exercise, calibrating an external tool 212
Exercise, constant velocity range and conditional stop 290
Exercise, CP motion 146
Exercise, dummy program 123
Exercise, executing robot programs 105
Exercise, gripper programming – plastic panel 202
Exercise, gripper programming, pen 287
Exercise, jogging with a fixed tool 207
Exercise, jogging, tool 65
Exercise, load mastering with offset 55
Exercise, motion programming with external TCP 218
Exercise, operator control and axis-specific jogging 34
Exercise, operator control and jogging in the world coordinate system 41
Exercise, robot mastering 55
Exercise, spline 193
Exercise, spline block 166
Exercise, tool calibration – gripper 78
Exercise, tool calibration – numeric 78
Exercise, tool calibration – pen 75
Expert level 219

F
Fixed tool, jogging 205
Flange coordinate system 35
FOC 286
Fold 224

G
GBE 285
Global 231
GOTO 254
Gripper configuration 199
Gripper operation 195
Gripper programming 196

H
Homogenous approximate positioning 182

I
Impact 266
Increment 31
Indenting 223
Initialization 97, 234
Inline form 116
Interpolation mode 121, 132, 135, 161, 163, 269

J
Jerk 122, 132, 135, 161, 164
Jog keys 14
Jogging, base 82
Jogging, fixed tool 205
Jogging, tool 61
Jogging, world 36
Jump 254
Jump command 254

K
KCB 285
KCP 285
KEB 285
Keyboard 15
Keyboard key 15
Keyword 230
KLI 285
KOI 286
KONI 286
KPC 286
KPP 286
KRL 286
KSB 286
KSI 286
KSP 286
KUKA.GripperTech 195
KUKA.HMI 286

L
LIN motion 125
Line mark for mastering 54
Loads on the robot 57
Local 231
Logbook 110
Logic operations 236
Logic, general 169
Loops 243

M
Manipulation 236
MCFB 286
MEMD 286
Messages 19
MGU 286
Mixed approximate positioning 182
Modifying a variable 239
Modifying motion commands 141
Motion programming 115
Moving individual axes 28

N
Non-rejecting loop 248
Notification message 20

O
Operating mode 21
Operator 219
Operator safety 18
OPI 286
Orientation control 127

P
Panic position 29
Payload data (menu item) 58
PMB 286
Priority 238
Program execution control 243
Program selection 98
Program start 98
Program start from PLC 259
Programmer 219
Programming a subprogram call 227
Programming, external TCP 217
PTP motion 117

R
RCD 286
RDC 286
Rejecting loop 247
Release device 31
Rename program 109
Restoring 109
Robot safety 17
Robroot 35

S
Safe operational stop 23
Safety gate 18
Safety STOP 0 23
Safety STOP 1 23
Safety STOP 2 23
Safety STOP 0 23
Safety STOP 1 23
Safety STOP 2 23
Safety stop, external 18
SATA 286
SBC 286
SEMD 286
SIB 286
Single (menu item) 239
Singularity 125
SION 286
Space Mouse 14
Spline block 149
Spline motion 149
Stamp 222
Standard functions 236
Start backwards key 15
Start key 15
Start-up mode 43
Status keys 15
Status message 19
STO 286

STOP 0 23
STOP 1 23
STOP 2 23
Stop category 0 23
Stop category 1 23
Stop category 2 23
STOP key 15
Subprogram global 225
Subprogram local 225
Subprograms 224
Supplementary load data (menu item) 61
Switch statement (SWITCH – CASE) 252
Switching functions, simple 174
Switching functions, time-distance 270

T

tm_useraction 265
TMx 267
Tool calibration 66
Tool coordinate system 35
Tool load data 57
Torque 266

U

UPS 286
USB 286
User group, default 219

V

Variable life 231
Variables 229, 232

W

WAIT 170
WAIT FOR 171
Wait function 170
Wait function, signal-dependent 256
Wait function, time-dependent 256
Wait message 20
WHILE 247
WorkVisual Catalog Editor 286
World coordinate system 35
Wrist root point 125, 126