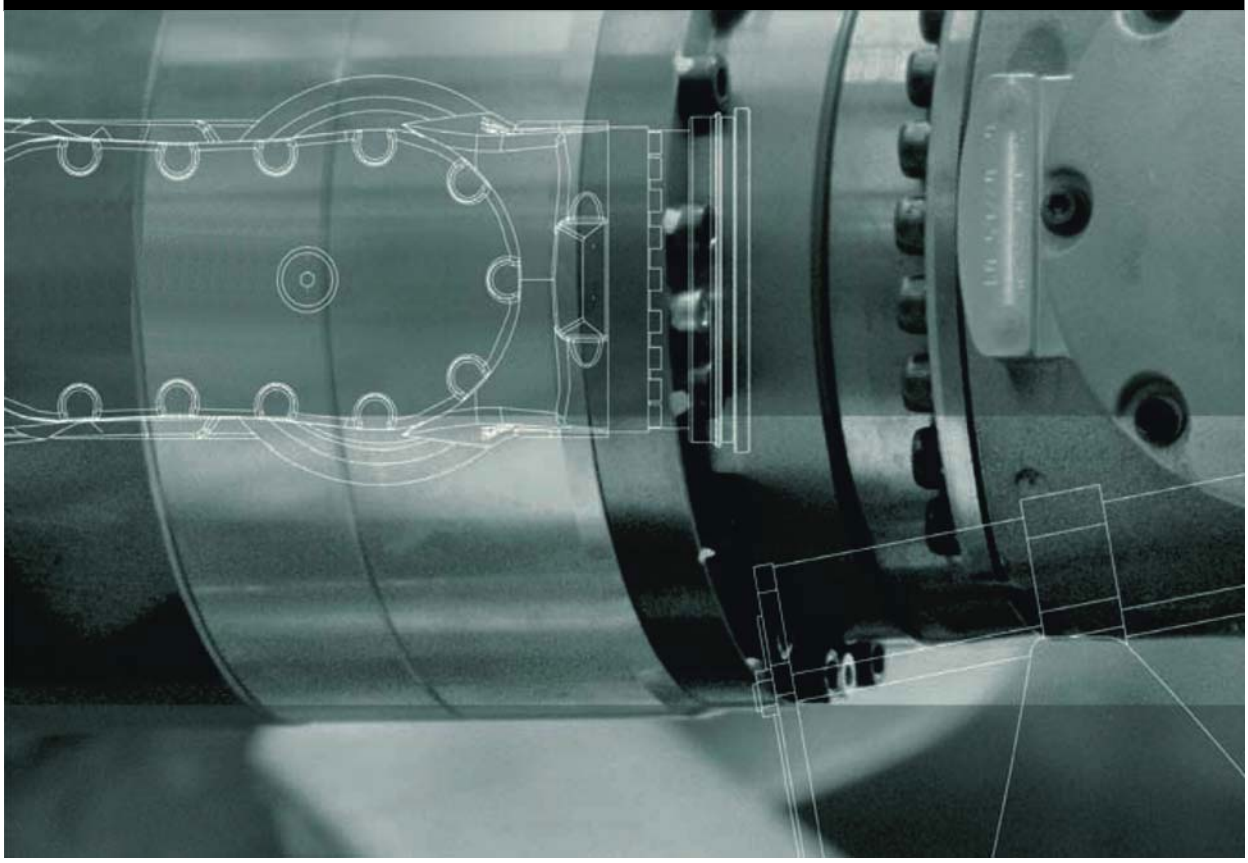
System Software

KUKA Roboter GmbH

KUKA Sunrise.OS 1.7
KUKA Sunrise.Workbench 1.7

Operating and Programming Instructions



Issued: 24.04.2015

Version: KUKA Sunrise.OS 1.7 V1

© Copyright 2015

KUKA Roboter GmbH
Zugspitzstraße 140
D-86165 Augsburg
Germany

This documentation or excerpts therefrom may not be reproduced or disclosed to third parties without the express permission of KUKA Roboter GmbH.

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 KUKA Sunrise.OS 1.7 (PDF) en
Book structure:	KUKA Sunrise.OS 1.7 V1.2
Version:	KUKA Sunrise.OS 1.7 V1

Contents

1 Introduction ... 15

1.1 Target group ... 15
1.2 Industrial robot documentation ... 15
1.3 Representation of warnings and notes ... 15
1.4 Trademarks ... 16
1.5 Terms used ... 16

2 Product description ... 19

2.1 Overview of the robot system ... 19
2.2 Overview of the software components ... 20
2.3 Overview of KUKA Sunrise.OS ... 20
2.4 Overview of KUKA Sunrise.Workbench ... 21
2.5 Intended use of the system software ... 21

3 Safety ... 23

3.1 Legal framework ... 23
3.1.1 Liability ... 23
3.1.2 Intended use of the industrial robot ... 23
3.1.3 EC declaration of conformity and declaration of incorporation ... 24
3.2 Safety functions ... 24
3.2.1 Terms used ... 25
3.2.2 Personnel ... 27
3.2.3 Workspace, safety zone and danger zone ... 28
3.2.4 Safety-oriented functions ... 28
3.2.4.1 EMERGENCY STOP device ... 29
3.2.4.2 Enabling device ... 29
3.2.4.3 Operator safety ... 30
3.2.4.4 External EMERGENCY STOP device ... 30
3.2.4.5 External safety stop 1 (path-maintaining) ... 31
3.2.4.6 External enabling device ... 31
3.2.4.7 External safe operational stop ... 31
3.2.5 Triggers for safety-oriented stop reactions ... 31
3.2.6 Non-safety-oriented functions ... 33
3.2.6.1 Mode selection ... 33
3.2.6.2 Software limit switches ... 34
3.3 Additional protective equipment ... 34
3.3.1 Jog mode ... 34
3.3.2 Labeling on the industrial robot ... 35
3.3.3 External safeguards ... 35
3.4 Safety measures ... 36
3.4.1 General safety measures ... 36
3.4.2 Transportation ... 37
3.4.3 Start-up and recommissioning ... 37
3.4.4 Manual mode ... 39
3.4.5 Automatic mode ... 39
3.4.6 Maintenance and repair ... 40
3.4.7 Decommissioning, storage and disposal ... 41
3.4.8 Safety measures for “single point of control” ... 41

3.5 Applied norms and regulations ... 41

4 Installing KUKA Sunrise.Workbench ... 43

4.1 PC system requirements ... 43
4.2 Installing Sunrise.Workbench ... 43
4.3 Uninstalling Sunrise.Workbench ... 43
4.4 Installing the language package in Sunrise.Workbench ... 44

5 Operation of KUKA Sunrise.Workbench ... 45

5.1 Starting Sunrise.Workbench ... 45
5.2 Overview of the user interface of Sunrise.Workbench ... 45
5.2.1 Repositioning the views ... 47
5.2.2 Displaying different perspectives on the user interface ... 47
5.2.3 Toolbars ... 48
5.3 Creating a Sunrise project with a template ... 48
5.4 Creating a new robot application ... 51
5.4.1 Creating a new Java package ... 51
5.4.2 Creating a robot application with a package ... 52
5.4.3 Creating a robot application for an existing package ... 52
5.5 Creating a new background task ... 52
5.5.1 Creating a background task with a package ... 52
5.5.2 Creating a background task for an existing package ... 53
5.6 Workspace ... 53
5.6.1 Creating a new workspace ... 53
5.6.2 Switching to an existing workspace ... 53
5.6.3 Switching between the most recently opened workspaces ... 54
5.6.4 Archiving projects ... 54
5.6.5 Loading projects from archive to the workspace ... 54
5.6.6 Loading projects from the directory to the workspace ... 54
5.7 Sunrise projects with referenced Java projects ... 55
5.7.1 Creating a new Java project ... 55
5.7.1.1 Inserting robot-specific class libraries in a Java project ... 55
5.7.2 Referencing Java projects ... 56
5.7.3 Canceling the reference to Java projects ... 56
5.8 Renaming an element in the **Package Explorer** ... 56
5.8.1 Renaming a project or Java package ... 56
5.8.2 Renaming a Java file ... 57
5.9 Removing an element from **Package Explorer** ... 57
5.9.1 Deleting an element from a project ... 57
5.9.2 Removing a project from **Package Explorer** ... 57
5.9.3 Deleting a project from the workspace ... 57
5.10 Activating the automatic change recognition ... 58

6 Operating the KUKA smartPAD ... 59

6.1 KUKA smartPAD control panel ... 59
6.1.1 Front view ... 59
6.1.2 Rear view ... 61
6.2 Switching the robot controller on/off ... 62
6.2.1 Switching on the robot controller and starting the system software ... 62

6.2.2 Switching off the robot controller ... 62
6.3 Automatic update of the smartPAD software ... 62
6.4 KUKA smartHMI user interface ... 63
6.4.1 Navigation bar ... 64
6.4.2 Status display ... 65
6.4.3 Keypad ... 66
6.4.4 Station view ... 66
6.4.5 Robots view ... 68
6.5 Calling the main menu ... 69
6.6 Changing the operating mode ... 70
6.7 Coordinate systems ... 72
6.8 Jogging the robot ... 74
6.8.1 “**Jogging options**” window ... 74
6.8.2 Setting the jog override (HOV) ... 75
6.8.3 Axis-specific jogging with the jog keys ... 76
6.8.4 Cartesian jogging with the jog keys ... 77
6.8.4.1 Null space motion ... 77
6.9 CRR mode – controlled robot retraction ... 78
6.10 Manually guiding the robot ... 78
6.11 Resuming the safety controller ... 79
6.12 Opening the holding brakes ... 79
6.13 Teaching and manually addressing frames ... 80
6.13.1 Displaying frames ... 80
6.13.2 Teaching frames ... 81
6.13.3 “**Jogging type**” window ... 83
6.13.4 Manually addressing frames ... 84
6.14 Program execution ... 85
6.14.1 Selecting a robot application ... 85
6.14.2 Selecting the program run mode ... 87
6.14.3 Setting the manual override ... 88
6.14.4 Starting a program forwards (manually) ... 88
6.14.5 Starting a program forwards (automatically) ... 88
6.14.6 Repositioning the robot after leaving the path ... 88
6.15 Activating the user keys ... 89
6.16 Display functions ... 90
6.16.1 Displaying the end frame of the motion currently being executed ... 90
6.16.2 Displaying the axis-specific actual position ... 91
6.16.3 Displaying the Cartesian actual position ... 92
6.16.4 Displaying axis-specific torques ... 92
6.16.5 Displaying an I/O group and changing the value of an output ... 93
6.16.6 Displaying the IP address and software version ... 95
6.16.7 Displaying the robot type and serial number ... 95
6.16.8 Displaying messages of the virus scanner ... 95

7 Start-up and recommissioning ... 97

7.1 Position mastering ... 97
7.1.1 Mastering axes ... 97
7.1.2 Manually unmastering axes ... 97
7.2 Calibration ... 98

7.2.1 Tool calibration ... 98
7.2.1.1 TCP calibration: XYZ 4-point method ... 99
7.2.1.2 Defining the orientation: ABC 2-point method ... 101
7.2.1.3 Defining the orientation: ABC World method ... 102
7.2.2 Calibrating the base: 3-point method ... 103
7.3 Determining tool load data ... 105

8 Brake test ... 109

8.1 Overview of the brake test ... 109
8.2 Creating the brake test application from the template ... 111
8.2.1 Adapting the brake test application for testing against the minimum brake holding torque 114
8.2.2 Changing the motion sequence for torque value determination ... 114
8.2.3 Changing the starting position for the brake test ... 115
8.3 Programming interface for the brake test ... 115
8.3.1 Evaluating the torques generated and determining the maximum absolute value 115
8.3.2 Polling the evaluation results of the maximum absolute torques ... 117
8.3.3 Creating an object for the brake test ... 118
8.3.4 Starting the execution of the brake test ... 119
8.3.5 Evaluating the brake test ... 120
8.3.5.1 Polling the results of the brake test ... 122
8.4 Performing a brake test ... 124
8.4.1 Evaluation results of the maximum absolute torques (display) ... 125
8.4.2 Results of the brake test (display) ... 125

9 Project management ... 127

9.1 Sunrise projects – overview ... 127
9.2 Frame management ... 127
9.2.1 Creating a new frame ... 128
9.2.2 Designating a frame as a base ... 128
9.2.3 Moving frames ... 129
9.2.4 Deleting frames ... 129
9.2.5 Displaying and modifying the properties of a frame ... 130
9.2.6 Inserting a frame in a motion instruction ... 131
9.3 Object management ... 132
9.3.1 Geometric structure of tools ... 132
9.3.2 Geometric structure of workpieces ... 133
9.3.3 Creating a tool or workpiece ... 133
9.3.4 Creating a frame for a tool or workpiece ... 134
9.3.5 Defining a default motion frame ... 135
9.3.6 Load data ... 136
9.3.6.1 Entering load data ... 136
9.3.7 Safety-oriented tool ... 137
9.3.7.1 Defining a safety-oriented tool ... 138
9.3.8 Safety-oriented workpieces ... 140
9.3.8.1 Defining safety-oriented workpieces ... 142
9.4 Synchronization of projects ... 143
9.4.1 Transferring the project to the robot controller ... 143
9.4.2 Updating the project on the robot controller or in Sunrise.Workbench ... 144

9.5 Loading the project from the robot controller ... 145

10 Station configuration and installation ... 147

10.1 Opening the station configuration ... 147
10.1.1 Configuring parameters for calibration ... 147
10.2 Installing the system software ... 148
10.2.1 Converting the safety configuration to a new software version ... 149
10.3 Installing a language package ... 149
10.4 Installing or updating the virus scanner ... 150

11 Bus configuration ... 153

11.1 Configuration and I/O mapping in WorkVisual – overview ... 153
11.2 Overview of field buses ... 154
11.3 Creating a new I/O configuration ... 154
11.4 Opening an existing I/O configuration ... 154
11.5 Creating Sunrise I/Os ... 155
11.5.1 “Create I/O signals” window ... 156
11.5.2 Creating an I/O group and inputs/outputs within the group ... 157
11.5.3 Editing an I/O group ... 158
11.5.4 Deleting an I/O group ... 158
11.5.5 Changing an input/output of a group ... 158
11.5.6 Deleting an input/output of a group ... 158
11.5.7 Exporting an I/O group as a template ... 158
11.5.8 Importing an I/O group from a template ... 159
11.6 Mapping the bus I/Os ... 159
11.6.1 **I/O Mapping** window ... 159
11.6.2 Buttons in the “**I/O Mapping**” window ... 160
11.6.3 Mapping Sunrise I/Os ... 161
11.7 Exporting the I/O configuration to the Sunrise project ... 161

12 External control ... 163

12.1 Configuring external control ... 163
12.1.1 External control inputs ... 164
12.1.2 External control outputs ... 164
12.1.3 Signal diagrams ... 165
12.1.4 Configuring external control in the project properties ... 166
12.2 Selecting a robot application as the default application ... 167
12.3 Defining the signal outputs for a project that is not externally controlled ... 168

13 Safety configuration ... 169

13.1 Overview of safety configuration ... 169
13.2 Safety concept ... 169
13.3 Permanent Safety Monitoring ... 172
13.4 Event-driven Safety Monitoring ... 173
13.5 Overview of Atomic Monitoring Functions ... 174
13.5.1 Standard Atomic Monitoring Functions ... 174
13.5.2 Parameterizable Atomic Monitoring Functions ... 175
13.5.3 Extended Atomic Monitoring Functions ... 177
13.5.4 Availability of the AMFs depending on the kinematic system ... 177
13.6 Safety configuration with KUKA Sunrise.Workbench ... 178

13.6.1 Overview: changing the safety configuration and activating it on the controller ... 179
13.6.2 Opening the safety configuration ... 180
13.6.2.1 Evaluating the safety configuration ... 180
13.6.2.2 Overview of the graphical user interface for the safety configuration ... 181
13.6.3 Configuring the safety functions of the PSM mechanism ... 182
13.6.3.1 Opening the PSM table **Customer PSM** ... 182
13.6.3.2 Creating safety functions for the PSM mechanism ... 183
13.6.3.3 Deleting safety functions of the PSM mechanism ... 184
13.6.3.4 Editing existing safety functions of the PSM mechanism ... 184
13.6.4 Configuring the safe states of the ESM mechanism ... 185
13.6.4.1 Adding a new ESM state ... 185
13.6.4.2 Opening a table for an ESM state ... 185
13.6.4.3 Deleting an ESM state ... 187
13.6.4.4 Creating a safety function for the ESM state ... 187
13.6.4.5 Deleting a safety function of an ESM state ... 187
13.6.4.6 Editing an existing safety function of an ESM state ... 187
13.6.4.7 Deactivating the ESM mechanism ... 188
13.6.4.8 Switching between ESM states ... 188
13.7 Activating the safety configuration on the robot controller ... 189
13.7.1 Deactivating the safety configuration ... 189
13.7.2 Restoring the safety configuration ... 190
13.7.3 Changing the password for activating the safety configuration ... 190
13.8 Use and parameterization of the Atomic Monitoring Functions ... 190
13.8.1 Evaluating the safety equipment on the KUKA smartPAD ... 190
13.8.2 Evaluating the operating mode ... 191
13.8.3 Evaluating the motion enable ... 191
13.8.4 Monitoring safe inputs ... 192
13.8.5 Manual guidance with enabling device and velocity monitoring ... 192
13.8.5.1 Monitoring of enabling switches on hand guiding devices ... 193
13.8.5.2 Monitoring functions during manual guidance ... 194
13.8.5.3 Velocity monitoring during manual guidance ... 195
13.8.6 Evaluating the position referencing ... 195
13.8.7 Evaluating the torque referencing ... 196
13.8.8 Velocity monitoring functions ... 197
13.8.8.1 Defining axis-specific velocity monitoring ... 197
13.8.8.2 Defining Cartesian velocity monitoring ... 197
13.8.8.3 Direction-specific monitoring of Cartesian velocity ... 199
13.8.9 Monitoring spaces ... 204
13.8.9.1 Defining Cartesian workspaces ... 206
13.8.9.2 Defining Cartesian protected spaces ... 208
13.8.9.3 Defining axis-specific monitoring spaces ... 210
13.8.10 Monitoring the tool orientation ... 212
13.8.11 Standstill monitoring (safe operational stop) ... 214
13.8.12 Activation delay for safety functions ... 215
13.8.13 Monitoring of forces and torques ... 216
13.8.13.1 Axis torque monitoring ... 216
13.8.13.2 Collision detection ... 217
13.8.13.3 TCP force monitoring ... 218
13.9 Example of a safety configuration ... 219
13.9.1 Task ... 220

13.9.2 Requirement ... 220
13.9.3 Suggested solution for the task ... 220
13.10 Position and torque referencing ... 223
13.10.1 Position referencing ... 223
13.10.2 Torque referencing ... 224
13.10.3 Creating an application for position and torque referencing ... 225
13.11 Safety acceptance overview ... 226
13.11.1 Checklist for general safety functions ... 226
13.11.2 Checklists for the safety-oriented tool ... 229
13.11.2.1 Geometry data of the safety-oriented tool ... 229
13.11.2.2 Load data of the safety-oriented tool ... 230
13.11.3 Checklist for safety-oriented workpieces ... 231
13.11.4 Checklist for rows used in the PSM tables ... 232
13.11.5 Checklists for ESM states ... 232
13.11.5.1 Used ESM states ... 232
13.11.5.2 Non-used ESM states ... 234
13.11.6 Checklists for AMFs used ... 234
13.11.6.1 AMF **smartPAD Emergency Stop** ... 234
13.11.6.2 AMF **smartPAD enabling switch inactive** ... 234
13.11.6.3 AMF **smartPAD enabling switch panic active** ... 234
13.11.6.4 AMF **Hand guiding device enabling inactive** ... 234
13.11.6.5 AMF **Hand guiding device enabling active** ... 235
13.11.6.6 AMF **Test mode** ... 235
13.11.6.7 AMF **Automatic mode** ... 235
13.11.6.8 AMF **Reduced-velocity mode** ... 235
13.11.6.9 AMF **High-velocity mode** ... 235
13.11.6.10 AMF **Motion enable** ... 236
13.11.6.11 AMF **Input signal** ... 236
13.11.6.12 AMF **Standstill monitoring of all axes** ... 236
13.11.6.13 AMF **Axis torque monitoring** ... 236
13.11.6.14 AMF **Axis velocity monitoring** ... 236
13.11.6.15 AMF **Position referencing** ... 237
13.11.6.16 AMF **Torque referencing** ... 237
13.11.6.17 AMF **Axis range monitoring** ... 237
13.11.6.18 AMF **Cartesian velocity monitoring** ... 237
13.11.6.19 AMF **Cartesian workspace monitoring / Cartesian protected space monitoring** 238
13.11.6.20 AMF **Collision detection** ... 238
13.11.6.21 AMF **TCP force monitoring** ... 239
13.11.6.22 AMF **Time delay** ... 239
13.11.6.23 AMF **Tool orientation** ... 239
13.11.6.24 AMF **Tool-related velocity component** ... 241
13.11.7 Creating a safety configuration report ... 241

14 Basic principles of motion programming ... 243

14.1 Overview of motion types ... 243
14.2 PTP motion type ... 243
14.3 LIN motion type ... 244
14.4 CIRC motion type ... 244
14.5 SPL motion type ... 245
14.6 Spline motion type ... 245

14.6.1 Velocity profile for spline motions ... 246
14.6.2 Modifications to spline blocks ... 248
14.6.3 LIN-SPL-LIN transition ... 250
14.7 Manual guidance motion type ... 251
14.8 Approximate positioning ... 252
14.9 Orientation control with LIN, CIRC, SPL ... 254
14.9.1 CIRC – reference system for the orientation control ... 256
14.9.2 CIRC – combinations of reference system and type for the orientation control ... 257
14.10 Redundancy information ... 258
14.10.1 Redundancy angle ... 259
14.10.2 Status ... 259
14.10.3 Turn ... 260
14.11 Singularities ... 261
14.11.1 Kinematic singularities ... 261
14.11.2 System-dependent singularities ... 263

15 Programming ... 265

15.1 Java Editor ... 265
15.1.1 Opening a robot application in the Java Editor ... 265
15.1.2 Structure of a robot application ... 265
15.1.3 Edit functions ... 266
15.1.3.1 Renaming a variable ... 266
15.1.3.2 Auto-complete ... 266
15.1.3.3 Templates – Fast entry of Java statements ... 267
15.1.3.4 Creating user-specific templates ... 268
15.1.3.5 Extracting methods ... 268
15.1.4 Displaying Javadoc information ... 269
15.1.4.1 Configuration of the Javadoc browser ... 271
15.2 Symbols and fonts ... 274
15.3 Data types ... 274
15.4 Variables ... 275
15.5 Network communication via UDP and TCP/IP ... 275
15.6 RoboticsAPI version information ... 276
15.6.1 Displaying the RoboticsAPI version ... 276
15.6.2 Structure of the RoboticsAPI version number: ... 276
15.7 Motion programming: PTP, LIN, CIRC ... 276
15.7.1 Structure of a motion command (move/moveAsync) ... 276
15.7.2 PTP ... 277
15.7.3 LIN ... 278
15.7.4 CIRC ... 278
15.7.5 LIN REL ... 279
15.7.6 MotionBatch ... 280
15.8 Motion programming: spline ... 281
15.8.1 Programming tips for spline motions ... 281
15.8.2 Creating a CP spline block ... 282
15.8.3 Creating a JP spline block ... 283
15.8.4 Using spline in a motion instruction ... 284
15.9 Motion parameters ... 284
15.9.1 Programming axis-specific motion parameters ... 285

15.10 Programming manual guidance ... 286
15.10.1 Axis-specific limits for manual guidance ... 288
15.11 Using tools and workpieces in the program ... 290
15.11.1 Declaring tools and workpieces ... 291
15.11.2 Initializing tools and workpieces ... 291
15.11.3 Attaching tools and workpieces to the robot ... 292
15.11.3.1 Attaching a tool to the robot flange ... 292
15.11.3.2 Attaching a workpiece to other objects ... 293
15.11.3.3 Detaching objects ... 294
15.11.4 Moving tools and workpieces ... 294
15.11.5 Defining user-specific object classes ... 295
15.11.6 Commanding load changes to the safety controller ... 298
15.12 Inputs/outputs ... 300
15.12.1 Creating a data array for an I/O group ... 301
15.12.2 Initializing a data array for an I/O group ... 302
15.12.3 Reading inputs/outputs ... 302
15.12.4 Setting outputs ... 303
15.13 Polling axis torques ... 303
15.14 Reading Cartesian forces and torques ... 305
15.14.1 Polling calculated force/torque data ... 305
15.14.2 Polling individual force/torque values ... 306
15.14.3 Checking the reliability of the calculated force/torque values ... 307
15.14.4 Polling individual values of a vector ... 308
15.15 Polling the robot position ... 309
15.15.1 Polling the axis-specific actual or setpoint position ... 309
15.15.2 Polling the Cartesian actual or setpoint position ... 310
15.15.3 Polling the Cartesian setpoint/actual value difference ... 311
15.16 HOME position ... 312
15.16.1 Changing the HOME position ... 312
15.17 Polling system states ... 313
15.17.1 Polling the HOME position ... 313
15.17.2 Polling the mastering state ... 314
15.17.3 Polling “ready for motion” ... 314
15.17.3.1 Reacting to changes in the “ready for motion” signal ... 315
15.17.4 Polling the robot activity ... 315
15.17.5 Polling and evaluating safety signals ... 316
15.17.5.1 Polling the state of the safety signals ... 316
15.17.5.2 Reacting to a change in state of safety signals ... 318
15.18 Changing and polling the program run mode ... 319
15.19 Changing and polling the override ... 320
15.19.1 Reacting to an override change ... 321
15.20 Conditions ... 321
15.20.1 Conditions in the RoboticsAPI ... 321
15.20.2 Complex conditions ... 323
15.20.3 Axis torque condition ... 324
15.20.4 Force condition ... 324
15.20.4.1 Condition for Cartesian force from all directions ... 326
15.20.4.2 Condition for normal force ... 327
15.20.4.3 Condition for shear force ... 328

15.20.5 Force component condition ... 329
15.20.6 Condition for Cartesian torque ... 331
15.20.6.1 Condition for Cartesian torque from all directions ... 332
15.20.6.2 Condition for torque ... 333
15.20.6.3 Condition for tilting torque ... 334
15.20.7 Torque component condition ... 335
15.20.8 Path-related condition ... 336
15.20.9 Condition for Boolean signals ... 338
15.20.10 Condition for the range of values of a signal ... 339
15.21 Break conditions for motion commands ... 339
15.21.1 Defining break conditions ... 339
15.21.2 Evaluating the break conditions ... 340
15.21.2.1 Polling a break condition ... 341
15.21.2.2 Polling the robot position at the time of termination ... 342
15.21.2.3 Polling a terminated motion (spline block, MotionBatch) ... 342
15.22 Path-related switching actions (Trigger) ... 343
15.22.1 Programming triggers ... 343
15.22.2 Programming a path-related switching action ... 344
15.22.3 Evaluating trigger information ... 345
15.23 Monitoring processes (Monitoring) ... 347
15.23.1 Listener for monitoring conditions ... 347
15.23.2 Creating a listener object to monitor the condition ... 348
15.23.3 Registering a listener for notification of change in state ... 349
15.23.4 Activating or deactivating the notification service for listeners ... 350
15.23.5 Programming example for monitoring ... 351
15.24 Blocking wait for condition ... 351
15.25 Recording and evaluating data ... 352
15.25.1 Creating an object for data recording ... 353
15.25.2 Specifying data to be recorded ... 354
15.25.3 Starting data recording ... 356
15.25.4 Ending data recording ... 357
15.25.5 Polling states from the DataRecorder object ... 357
15.25.6 Example program for data recording ... 358
15.26 Defining user keys ... 358
15.26.1 Creating a user key bar ... 359
15.26.2 Adding user keys to the bar ... 360
15.26.3 Defining the function of a user key ... 362
15.26.4 Labeling and graphical assignment of the user key bar ... 363
15.26.4.1 Assigning a text element ... 364
15.26.4.2 Assigning an LED icon ... 366
15.26.5 Identifying safety-critical user keys ... 367
15.26.6 Publishing a user key bar ... 367
15.27 Message programming ... 368
15.27.1 Programming user messages ... 368
15.27.2 Programming user dialogs ... 369
15.28 Program execution control ... 370
15.28.1 Pausing an application ... 370
15.28.2 Pausing motion execution ... 371
15.28.3 FOR loop ... 371

15.28.4 WHILE loop ... 372
15.28.5 DO WHILE loop ... 373
15.28.6 IF ELSE branch ... 374
15.28.7 SWITCH branch ... 375
15.28.8 Examples of nested loops ... 377
15.29 Continuing a paused application in Automatic mode (recovery) ... 377
15.30 Error treatment ... 379
15.30.1 Handling of failed motion commands ... 379
15.30.2 Handling of failed synchronous motion commands ... 380
15.30.3 Handling of failed asynchronous motion commands ... 381

16 Background tasks ... 385

16.1 Using background tasks ... 385
16.2 Cyclical background task ... 386
16.3 Non-cyclic background task ... 389
16.4 Data exchange between tasks ... 389
16.4.1 Declaring task functions ... 390
16.4.2 Implementing task functions ... 392
16.4.3 Creating the providing task ... 393
16.4.4 Using task functions ... 395

17 Programming with a compliant robot ... 399

17.1 Sensors and control ... 399
17.2 Available controllers – overview ... 399
17.3 Using controllers in robot applications ... 399
17.3.1 Creating a controller object ... 400
17.3.2 Defining controller parameters ... 400
17.3.3 Transferring the controller object as a motion parameter ... 400
17.4 Position controller ... 401
17.5 Cartesian impedance controller ... 401
17.5.1 Calculation of the forces on the basis of Hooke's law ... 401
17.5.2 Parameterization of the Cartesian impedance controller ... 403
17.5.2.1 Representation of Cartesian degrees of freedom ... 404
17.5.2.2 Defining controller parameters for individual degrees of freedom ... 404
17.5.2.3 Controller parameters specific to the degrees of freedom ... 405
17.5.2.4 Controller parameters independent of the degrees of freedom ... 406
17.6 Cartesian impedance controller with overlaid force oscillation ... 409
17.6.1 Overlaying a simple force oscillation ... 409
17.6.2 Overlaying superposed force oscillations (Lissajous curves) ... 410
17.6.3 Parameterization of the impedance controller with overlaid force oscillation ... 411
17.6.3.1 Controller parameters specific to the degrees of freedom ... 412
17.6.3.2 Controller parameters independent of the degrees of freedom ... 414
17.7 Static methods for impedance controller with superposed force oscillation ... 416
17.7.1 Overlaying a constant force ... 416
17.7.2 Overlaying a simple force oscillation ... 417
17.7.3 Overlaying a Lissajous oscillation ... 418
17.7.4 Overlaying a spiral-shaped force oscillation ... 419
17.8 Axis-specific impedance controller ... 421
17.8.1 Parameterization of the axis-specific impedance controller ... 421

17.8.2 Methods of the axis-specific impedance controller 421
17.9 Holding the position under servo control 422

18 Diagnosis 425

18.1 Field bus diagnosis 425
18.1.1 Displaying general field bus errors 425
18.1.2 Displaying the error state of I/Os and I/O groups 425
18.2 Displaying the protocol 425
18.2.1 “**Protocol**” view 426
18.2.2 Filtering protocol entries 427
18.3 Display of error messages (Applications view) 428
18.4 Collecting diagnostic information for error analysis at KUKA 431
18.4.1 Creating a diagnosis package with the smartHMI 431
18.4.2 Creating a diagnosis package with the smartHMI 431
18.4.3 Creating a diagnosis package with Sunrise.Workbench 432
18.4.4 Loading existing diagnosis packages from the robot controller 432

19 KUKA Service 433

19.1 Requesting support 433
19.2 KUKA Customer Support 433

Index **441**

1 Introduction

1.1 Target group

This documentation is aimed at users with the following knowledge and skills:

- Advanced knowledge of the robot controller system
- Advanced Java programming skills

For optimal use of our products, we recommend that our customers take part in a course of training at KUKA College. Information about the training program can be found at www.kuka.com or can be obtained directly from our subsidiaries.

1.2 Industrial robot documentation

The industrial robot documentation consists of the following parts:

- Documentation for the manipulator
- Documentation for the robot controller
- Operating and programming instructions for the System Software
- Instructions for options and accessories
- Parts catalog on storage medium

Each of these sets of instructions is a separate document.

1.3 Representation of warnings and notes

Safety

These warnings are relevant to safety and **must** be observed.

DANGER These warnings mean that it is certain or highly probable that death or severe injuries **will** occur, if no precautions are taken.

WARNING These warnings mean that death or severe injuries **may** occur, if no precautions are taken.

CAUTION These warnings mean that minor injuries **may** occur, if no precautions are taken.

NOTICE These warnings mean that damage to property **may** occur, if no precautions are taken.

These warnings contain references to safety-relevant information or general safety measures.
These warnings do not refer to individual hazards or individual precautionary measures.

This warning draws attention to procedures which serve to prevent or remedy emergencies or malfunctions:

SAFETY INSTRUCTIONS Procedures marked with this warning **must** be followed exactly.

Instructions

These notices serve to make your work easier or contain references to further information.

Tip to make your work easier or reference to further information.

1.4 Trademarks

Java is a trademark of Sun Microsystems (Oracle Corporation).

Windows is a trademark of Microsoft Corporation.

EtherCAT Technology Group is a trademark of Beckhoff Automation GmbH.

1.5 Terms used

Term	Description
AMF	Atomic Monitoring Function Smallest unit of a monitoring function
API	Application Programming Interface Interface for programming applications.
CIB-SR	Cabinet Interface Board, Small Robot
ESM	Event-Driven Safety Monitoring Safety monitoring functions which are activated using defined events
Exception	Exception or exceptional situation An exception describes a procedure for forwarding information about certain program statuses, mainly error states, to other program levels for further processing.
Frame	A frame is a 3-dimensional coordinate system that is described by its position and orientation relative to a reference system. Points in space can be easily defined using frames. Frames are often arranged hierarchically in a tree structure.
FSoE	Fail Safe over EtherCAT FSoE is a protocol for transferring safety-relevant data via EtherCAT in conjunction with an FSoE master and an FSoE slave.
Javadoc	Javadoc is a documentation generated from specific Java comments.
JRE	Java Runtime Environment Runtime environment of the Java programming language
KLI	KUKA Line Interface Ethernet interface of the robot controller (not real-time-capable) for external communication.
KMP	KUKA Mobile Platform Designation for mobile platforms from KUKA

Term	Description
CRR	**C**ontrolled **R**obot **R**etraction CRR is an operating mode which is available when the industrial robot is stopped by the safety controller for one of the following reasons: ■ Industrial robot violates a safely monitored space. ■ Orientation of the safety-oriented tool is outside the safely monitored range. ■ Industrial robot violates a safely monitored force or torque limit. ■ A position sensor is not mastered during active Cartesian velocity monitoring. In CRR mode, the robot can be jogged and moved back to a position in which the monitoring function that triggered the stop is no longer violated.
KUKA RoboticsAPI	Java programming interface for KUKA robots RoboticsAPI is an object-oriented Java interface for controlling robots and peripheral devices.
KUKA smartHMI	KUKA smart human-machine interface Name of the graphical user interface of the robot controller
KUKA smartPAD	The smartPAD is the hand-held control panel for the robot cell (station). It has all the operator control and display functions required for operation of the station.
KUKA Sunrise Cabinet	Control hardware for operating the LBR iiwa industrial robot
KUKA Sunrise.OS	KUKA Sunrise.Operating System System software for industrial robots which are operated with the robot controller KUKA Sunrise Cabinet
HRC	Human-robot collaboration
PROFINET	PROFINET is an Ethernet-based field bus.
PROFIsafe	PROFIsafe is a PROFINET-based safety interface for connecting a safety PLC to the robot controller. (PLC = master, robot controller = slave)
PSM	Permanent Safety Monitoring Safety monitoring functions which are permanently active
PLC	Programmable logic controller
TCP	Tool Center Point The TCP is the working point of a tool. Multiple working points can be defined for a tool.

2 Product description

2.1 Overview of the robot system

A robot system (>>> Fig. 2-1) comprises all the assemblies of an industrial robot, including the manipulator (mechanical system and electrical installations), controller, connecting cables, end effector (tool) and other equipment.

The industrial robot consists of the following components:

- Manipulator
- KUKA Sunrise Cabinet robot controller
- KUKA smartPAD control panel
- Connecting cables
- Software
- Options, accessories

Fig. 2-1: Overview of robot system

1 Connecting cable to the smartPAD
2 KUKA smartPAD control panel
3 Manipulator
4 Connecting cable to KUKA Sunrise Cabinet robot controller
5 KUKA Sunrise Cabinet robot controller

2.2 Overview of the software components

The following software components are used:

- KUKA Sunrise.OS 1.7
- KUKA Sunrise.Workbench 1.7
- WorkVisual 3.0

2.3 Overview of KUKA Sunrise.OS

Description

With KUKA Sunrise.OS, the operator control and programming of the robot system are strictly separated.

- Robot applications are programmed with KUKA Sunrise.Workbench.
- A robot cell (station) is operated using the KUKA smartPAD control panel.
- A station consists of a robot controller, a manipulator and further devices.
- A station may carry out multiple applications (tasks).

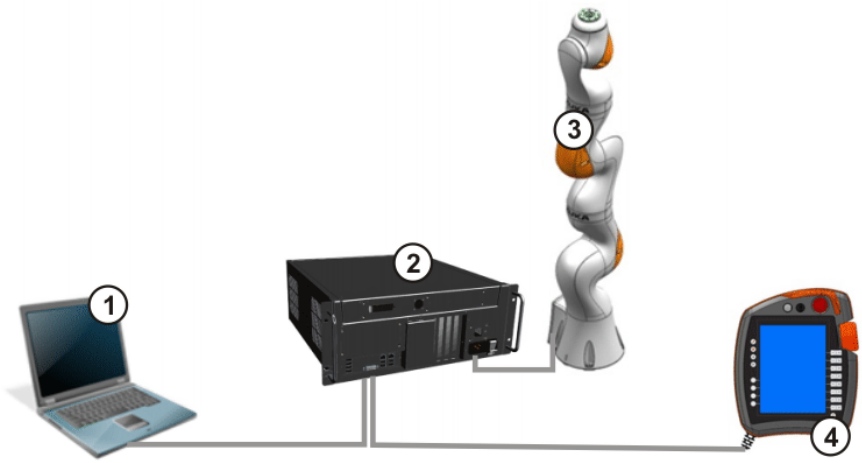


Fig. 2-2: Separation of operator control and programming

1 Development computer with KUKA Sunrise.Workbench (connection via the KLI of the robot controller)
2 KUKA Sunrise Cabinet robot controller
3 Manipulator
4 KUKA smartPAD control panel

> The development computer is not included in the scope of supply of the industrial robot.

Division of tasks

KUKA Sunrise.Workbench is the tool for the start-up of a station and the development of robot applications. WorkVisual is used for bus configuration and bus mapping.

The smartPAD is only required in the start-up phase for tasks which for practical or safety reasons cannot be carried out using KUKA Sunrise.Workbench, e.g. for mastering, calibration and teaching points.

After start-up and application development, the operator can carry out simple servicing work and operating tasks using the smartPAD. The operator cannot change the station and safety configuration or the programming.

Overview

Task	WorkVisual	Workbench	smartPAD
Station configuration		✓	
Software installation		✓	
Bus configuration/diagnosis	✓		
Bus mapping	✓		
Configuring safety settings		✓	
Activating the safety configuration			✓
Programming		✓	
Debugging		✓	
Managing/editing runtime data		✓	
Teaching frames			✓
Operating mode selection			✓
Jogging			✓
Mastering			✓
Calibration			✓
Load data determination			✓
Setting/polling outputs			✓
Polling inputs			✓
Starting/stopping robot applications			✓
Creating a diagnosis package		✓	✓

2.4 Overview of KUKA Sunrise.Workbench

KUKA Sunrise.Workbench is the development environment for the robot cell (station). It offers the following functionalities for start-up and application development:

Start-up

- Installing the system software
- Configuring the robot cell (station)
- Editing the safety configuration
- Creating the I/O configuration
- Transferring the project to the robot controller

Application development

- Programming robot applications in Java
- Managing projects and programs
- Editing and managing runtime data
- Project synchronization

2.5 Intended use of the system software

Use

The system software is intended exclusively for the operation of KUKA axes in an industrial setting in conjunction with KUKA Sunrise Cabinet. KUKA axes include, for example, industrial robots and mobile platforms.

Each version of the system software may be operated exclusively in accordance with the specified system requirements.

Misuse

Any use or application deviating from the intended use is deemed to be misuse and is not allowed. KUKA Roboter GmbH is not liable for any damage resulting from such misuse. The risk lies entirely with the user.

Examples of such misuse include:

- Operating axes that are not KUKA axes
- Operation of the system software not in accordance with the specified system requirements
- Use of any debugger other than that provided by Sunrise.Workbench
- Use for non-industrial applications for which specific product requirements/standards exist (e.g. medical applications)

3 Safety

3.1 Legal framework

3.1.1 Liability

The device described in this document is either an industrial robot or a component thereof.

Components of the industrial robot:

- Manipulator
- Robot controller
- Hand-held control panel
- Connecting cables
- Software
- Options, accessories

The industrial robot is built using state-of-the-art technology and in accordance with the recognized safety rules. Nevertheless, misuse of the industrial robot may constitute a risk to life and limb or cause damage to the industrial robot and to other material property.

The industrial robot may only be used in perfect technical condition in accordance with its designated use and only by safety-conscious persons who are fully aware of the risks involved in its operation. Use of the industrial robot is subject to compliance with this document and with the declaration of incorporation supplied together with the industrial robot. Any functional disorders affecting safety must be rectified immediately.

Safety information

Safety information cannot be held against KUKA Roboter GmbH. Even if all safety instructions are followed, this is not a guarantee that the industrial robot will not cause personal injuries or material damage.

No modifications may be carried out to the industrial robot without the authorization of KUKA Roboter GmbH. Additional components (tools, software, etc.), not supplied by KUKA Roboter GmbH, may be integrated into the industrial robot. The user is liable for any damage these components may cause to the industrial robot or to other material property.

In addition to the Safety chapter, this document contains further safety instructions. These must also be observed.

3.1.2 Intended use of the industrial robot

The industrial robot is intended exclusively for the use designated in the “Purpose” chapter of the operating instructions or assembly instructions.

Any use or application deviating from the intended use is deemed to be misuse and is not allowed. The manufacturer is not liable for any damage resulting from such misuse. The risk lies entirely with the user.

Operation of the industrial robot in accordance with its intended use also requires compliance with the operating and assembly instructions for the individual components, with particular reference to the maintenance specifications.

The user is responsible for the performance of a risk analysis. This indicates the additional safety equipment that is required, the installation of which is also the responsibility of the user.

Misuse

Any use or application deviating from the intended use is deemed to be misuse and is not allowed. This includes e.g.:

- Transportation of persons and animals
- Use as a climbing aid
- Operation outside the specified operating parameters
- Use in potentially explosive environments
- Operation without the required additional safety equipment
- Outdoor operation
- Underground operation

3.1.3 EC declaration of conformity and declaration of incorporation

The industrial robot constitutes partly completed machinery as defined by the EC Machinery Directive. The industrial robot may only be put into operation if the following preconditions are met:

- The industrial robot is integrated into a complete system.

  Or: The industrial robot, together with other machinery, constitutes a complete system.

  Or: All safety functions and safeguards required for operation in the complete machine as defined by the EC Machinery Directive have been added to the industrial robot.
- The complete system complies with the EC Machinery Directive. This has been confirmed by means of an assessment of conformity.

Declaration of conformity

The system integrator must issue a declaration of conformity for the complete system in accordance with the Machinery Directive. The declaration of conformity forms the basis for the CE mark for the system. The industrial robot must always be operated in accordance with the applicable national laws, regulations and standards.

The robot controller is CE certified under the EMC Directive and the Low Voltage Directive.

Declaration of incorporation

The industrial robot as partly completed machinery is supplied with a declaration of incorporation in accordance with Annex II B of the EC Machinery Directive 2006/42/EC. The assembly instructions and a list of essential requirements complied with in accordance with Annex I are integral parts of this declaration of incorporation.

The declaration of incorporation declares that the start-up of the partly completed machinery is not allowed until the partly completed machinery has been incorporated into machinery, or has been assembled with other parts to form machinery, and this machinery complies with the terms of the EC Machinery Directive, and the EC declaration of conformity is present in accordance with Annex II A.

3.2 Safety functions

Safety functions are distinguished according to the safety requirements that they fulfill:

- Safety-oriented functions for the protection of personnel

  The safety-oriented functions of the industrial robot meet the following safety requirements:
  - **Category 3** and **Performance Level d** in accordance with EN ISO 13849-1:2008
  - **SIL 2** according to EN 62061

  The requirements are only met on the following condition, however:

- All safety-relevant mechanical and electromechanical components of the industrial robot are tested for correct functioning during start-up and at least once every 12 months, unless otherwise determined in accordance with a workplace risk assessment. This includes:
  - EMERGENCY STOP device on the smartPAD
  - Enabling device on the smartPAD
  - Enabling device on the media flange Touch (if present)
  - Keyswitch on the smartPAD
  - Safe outputs of the discrete safety interface
- Non-safety-oriented functions for the protection of machines

  The non-safety-oriented functions of the industrial robot do not meet specific safety requirements:

DANGER In the absence of the required operational safety functions and safeguards, the industrial robot can cause personal injury or material damage. If the required safety functions or safeguards are dismantled or deactivated, the industrial robot may not be operated.

During system planning, the safety functions of the overall system must also be planned and designed. The industrial robot must be integrated into this safety system of the overall system.

3.2.1 Terms used

Term	Description
Axis range	Range of each axis, in degrees or millimeters, within which it may move. The axis range must be defined for each axis.
Stopping distance	Stopping distance = reaction distance + braking distance The stopping distance is part of the danger zone.
Workspace	The manipulator is allowed to move within its workspace. The workspace is derived from the individual axis ranges.
Automatic (AUT)	Operating mode for program execution. The manipulator moves at the programmed velocity.
Operator (User)	The user of the industrial robot can be the management, employer or delegated person responsible for use of the industrial robot.
Danger zone	The danger zone consists of the workspace and the stopping distances.
Service life	The service life of a safety-relevant component begins at the time of delivery of the component to the customer. The service life is not affected by whether the component is used in a robot controller or elsewhere or not, as safety-relevant components are also subject to aging during storage.

Term	Description
CRR	**C**ontrolled **R**obot **R**etraction CRR is an operating mode which is available when the industrial robot is stopped by the safety controller for one of the following reasons: ■ Industrial robot violates a safely monitored space. ■ Orientation of the safety-oriented tool is outside the safely monitored range. ■ Industrial robot violates a safely monitored force or torque limit. ■ A position sensor is not mastered during active Cartesian velocity monitoring. In CRR mode, the robot can be jogged and moved back to a position in which the monitoring function that triggered the stop is no longer violated.
KUKA smartPAD	The smartPAD is the hand-held control panel for the robot cell (station). The smartPAD has all the operator control and display functions required for operation.
Manipulator	The robot arm and the associated electrical installations
Safety zone	The manipulator is not allowed to move within the safety zone. The safety zone is the area outside the danger zone.
Safety stop	The safety stop is triggered by the safety controller, interrupts the work procedure and causes all robot motions to come to a standstill. The program data are retained in the case of a safety stop and the program can be resumed from the point of interruption. The safety stop can be executed as a Stop category 0, Stop category 1 or Stop category 1 (path-maintaining). **Note:** In this document, a safety stop of Stop category 0 is referred to as safety stop 0, a safety stop of Stop category 1 as safety stop 1 and a safety stop of Stop category 1 (path-maintaining) as safety stop 1 (path-maintaining).
Stop category 0	The drives are deactivated immediately and the brakes are applied.
Stop category 1	The manipulator is braked and does not stay on the programmed path. The manipulator is brought to a standstill with the drives. As soon as an axis is at a standstill, the drive is switched off and the brake is applied. The internal electronic drive system of the robot performs safety-oriented monitoring of the braking process. Stop category 0 is executed in the event of a fault.
Stop category 1 (path-maintaining)	The manipulator is braked and stays on the programmed path. At standstill, the drives are deactivated and the brakes are applied. If Stop category 1 (path-maintaining) is triggered by the safety controller, the safety controller monitors the braking process. The brakes are applied and the drives are switched off after 1 s at the latest. Stop category 0 is executed in the event of a fault.
System integrator (plant integrator)	System integrators are people who safely integrate the industrial robot into a complete system and commission it.
T1	Test mode, Manual Reduced Velocity (<= 250 mm/s) **Note:** With manual guidance in T1, the velocity is not reduced. The maximum velocity of 250 mm/s does not apply to a mobile platform.
T2	Test mode, Manual High Velocity (> 250 mm/s permissible)

3.2.2 Personnel

The following persons or groups of persons are defined for the industrial robot:

- User
- Personnel

All persons working with the industrial robot must have read and understood the industrial robot documentation, including the safety chapter.

User

The user must observe the labor laws and regulations. This includes e.g.:

- The user must comply with his monitoring obligations.
- The user must carry out instructions at defined intervals.

Personnel

Personnel must be instructed, before any work is commenced, in the type of work involved and what exactly it entails as well as any hazards which may exist. Instruction must be carried out regularly. Instruction is also required after particular incidents or technical modifications.

Personnel includes:

- System integrator
- Operators, subdivided into:
  - Start-up, maintenance and service personnel
  - Operator
  - Cleaning personnel

Installation, exchange, adjustment, operation, maintenance and repair must be performed only as specified in the operating or assembly instructions for the relevant component of the industrial robot and only by personnel specially trained for this purpose.

System integrator

The industrial robot is safely integrated into a complete system by the system integrator.

The system integrator is responsible for the following tasks:

- Installing the industrial robot
- Connecting the industrial robot
- Performing risk assessment
- Implementing the required safety functions and safeguards
- Issuing the declaration of conformity
- Attaching the CE mark
- Creating the operating instructions for the complete system

Operator

The operator must meet the following preconditions:

- The operator must be trained for the work to be carried out.
- Work on the industrial robot must only be carried out by qualified personnel. These are people who, due to their specialist training, knowledge and experience, and their familiarization with the relevant standards, are able to assess the work to be carried out and detect any potential hazards.

Work on the electrical and mechanical equipment of the manipulator may only be carried out by KUKA Roboter GmbH.

3.2.3 Workspace, safety zone and danger zone

Working zones are to be restricted to the necessary minimum size in order to prevent danger to persons or the risk of material damage. Safe axis range limitations required for personnel protection are configurable.

> Further information about configuring safe axis range limitations is contained in the “Safety configuration” chapter of the operating and programming instructions. (>>> 13 "Safety configuration" Page 169)

The danger zone consists of the workspace and the stopping distances of the manipulator. In the event of a stop, the manipulator is braked and comes to a stop within the danger zone. The safety zone is the area outside the danger zone.

The danger zone must be protected by means of physical safeguards, e.g. by light barriers, light curtains or safety fences. If there are no physical safeguards present, the requirements for collaborative operation in accordance with EN ISO 10218 must be met. There must be no shearing or crushing hazards at the loading and transfer areas.

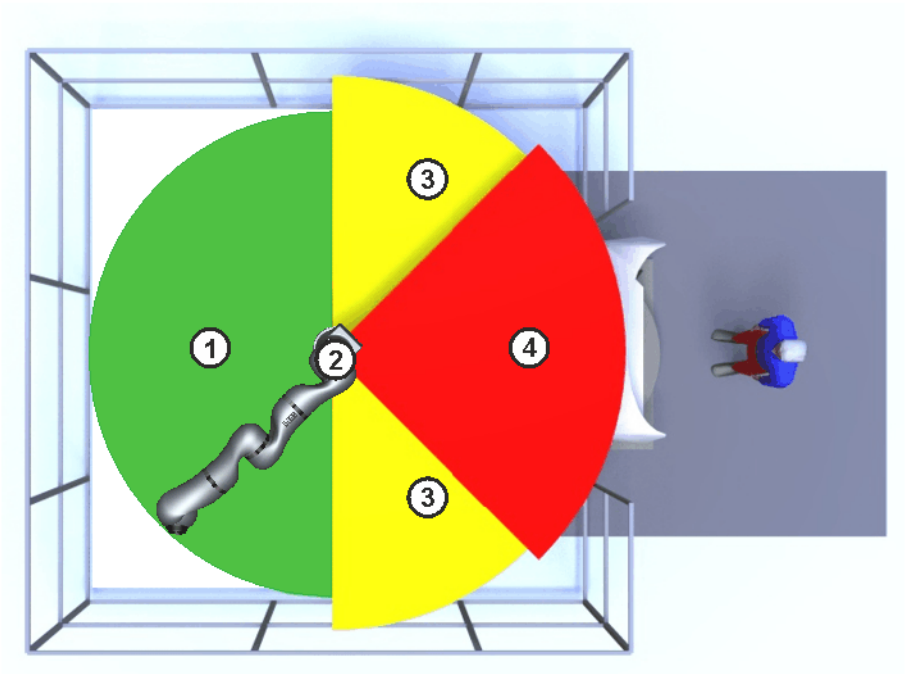


Fig. 3-1: Example: axis range A1

1	Workspace	3	Stopping distance
2	Manipulator	4	Safety zone

3.2.4 Safety-oriented functions

The following safety-oriented functions are present and permanently defined in the industrial robot:

- EMERGENCY STOP device
- Enabling device
- Locking of the operating mode (by means of a keyswitch)

The following safety-oriented functions are preconfigured and can be integrated into the system via the safety interface of the robot controller:

- Operator safety (= connection for the guard interlock)

- External EMERGENCY STOP device
- External safety stop 1 (path-maintaining)

Other safety-oriented functions that are not present by default may be configured, e.g.:

- External enabling device
- External safe operational stop
- Axis-specific workspace monitoring
- Cartesian workspace monitoring
- Cartesian protected space monitoring
- Velocity monitoring
- Standstill monitoring
- Axis torque monitoring
- Collision detection

> Further information about configuring the safety functions is contained in the “Safety configuration” chapter of the operating and programming instructions. (>>> 13 "Safety configuration" Page 169)

The default configuration of the preconfigured safety functions is described in the following sections on safety.

3.2.4.1 EMERGENCY STOP device

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

Reaction of the industrial robot if the EMERGENCY STOP device is pressed:

- The manipulator stops with a safety stop 1 (path-maintaining).

Before operation can be resumed, the EMERGENCY STOP device must be turned to release it.

> **⚠ 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.

If a holder is used for the smartPAD and conceals the EMERGENCY STOP device on the smartPAD, an external EMERGENCY STOP device must be installed that is accessible at all times.

(>>> 3.2.4.4 "External EMERGENCY STOP device" Page 30)

3.2.4.2 Enabling device

The enabling devices of the industrial robot are the enabling switches on the smartPAD.

There are 3 enabling switches installed on the smartPAD. The enabling switches have 3 positions:

- Not pressed
- Center position
- Fully pressed (panic position)

In the test modes and in CRR, the manipulator can only be moved if one of the enabling switches is held in the central position.

- Releasing the enabling switch triggers a safety stop 1 (path-maintaining).
- Fully pressing the enabling switch triggers a safety stop 1 (path-maintaining).
- It is possible to hold 2 enabling switches in the center position simultaneously for several seconds. This makes it possible to adjust grip from one enabling switch to another one. If 2 enabling switches are held simultaneously in the center position for longer than 15 seconds, this triggers a safety stop 1.

If an enabling switch malfunctions (e.g. jams in the central position), the industrial robot can be stopped using the following methods:

- Press the enabling switch down fully.
- Actuate the EMERGENCY STOP device.
- Release the Start key.

⚠ WARNING The enabling switches must not be held down by adhesive tape or other means or tampered with in any other way.
Death, injuries or damage to property may result.

3.2.4.3 Operator safety

The operator safety signal is used for interlocking physical safeguards, e.g. safety gates. In the default configuration, automatic operation is not possible without this signal. Alternatively, the requirements for collaborative operation in accordance with EN ISO 10218 must be met.

Reaction of the industrial robot in the event of a loss of signal during automatic operation, e.g. safety gate is opened (default configuration):

- The manipulator stops with a safety stop 1 (path-maintaining).

By default, operator safety is not active in the modes T1 (Manual Reduced Velocity) and CRR, i.e. the signal is not evaluated. Operator safety is active in mode T2 (Manual High Velocity).

⚠ WARNING Following a loss of signal, automatic operation must not be resumed merely by closing the safeguard; the signal for operator safety must first be set by an additional device, e.g. by an acknowledge button. It is the responsibility of the system integrator to ensure this. This 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.

- This additional device must be designed in such a way that an actual check of the danger zone can be carried out first. Devices that do not allow this (e.g. because they are automatically triggered by closure of the safeguard) are not permissible.
- Failure to observe this may result in death to persons, severe injuries or considerable damage to property.

3.2.4.4 External EMERGENCY STOP device

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.

Reaction of the industrial robot if the external EMERGENCY STOP device is pressed (default configuration):

- The manipulator stops with a safety stop 1 (path-maintaining).

Multiple external EMERGENCY STOP devices can be connected via the safety interface of the robot controller. External EMERGENCY STOP devices are not included in the scope of supply of the industrial robot.

3.2.4.5 External safety stop 1 (path-maintaining)

The external safety stop 1 (path-maintaining) can be triggered via an input on the safety interface (default configuration). 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.

3.2.4.6 External enabling device

External enabling devices are required if it is necessary for more than one person to be in the danger zone of the industrial robot.

Multiple external enabling devices can be connected via the safety interface of the robot controller. External enabling devices are not included in the scope of supply of the industrial robot.

An external enabling device can be used for manual guidance of the robot. When enabling is active, the robot may only be moved at reduced velocity.

For manual guidance, safety-oriented velocity monitoring with a maximum permissible velocity of 250 mm/s is preconfigured. The maximum permissible velocity can be adapted.

The value for the maximum permissible velocity must be determined as part of a risk assessment.

3.2.4.7 External 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.

The safe operational stop can be triggered via an input on the safety 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.

3.2.5 Triggers for safety-oriented stop reactions

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

Overview

In KUKA Sunrise a distinction is made between the following triggers:

- Permanently defined triggers

  Permanently defined triggers for stop reactions and the associated stop category are preset by the system and cannot be changed. However, it is possible for the implemented stop reaction to be stepped up in the user-specific safety configuration.
- User-specific triggers

  In addition to the permanently defined triggers, the user can also configure other triggers for stop reactions including the associated stop category.

> Further information about configuring the safety functions is contained in the “Safety configuration” chapter of the operating and programming instructions. (>>> 13 "Safety configuration" Page 169)

Permanently defined triggers

The following triggers for stop reactions are permanently defined:

Trigger	T1, T2, CRR	AUT
Operating mode changed during operation	Safety stop 1 (path-maintaining)	
Enabling switch released	Safety stop 1 (path-maintaining)	-
Enabling switch pressed fully down (panic position)	Safety stop 1 (path-maintaining)	-
Local E-STOP pressed	Safety stop 1 (path-maintaining)	
Error in safety controller	Safety stop 1	

User-specific triggers

The robot controller is shipped with a safety configuration that is active on initial start-up. This contains the following user-specific stop reaction triggers preconfigured by KUKA (in addition to the permanently defined triggers).

Trigger	T1, CRR	T2, AUT
Safety gate opened (operator safety)	-	Safety stop 1 (path-maintaining)

When creating a new Sunrise project, the system automatically generates a project-specific safety configuration. This contains the following user-specific stop reaction triggers preconfigured by KUKA (in addition to the permanently defined triggers).

When the Sunrise project is transferred to the robot controller, the factory-set safety configuration is overwritten by the project-specific safety configuration. This makes it necessary for the safety configuration to be activated.
Further information about activating the safety configuration is contained in the “Safety configuration” chapter of the operating and programming instructions. (>>> 13 "Safety configuration" Page 169)

Trigger	T1, CRR	T2, AUT
Safety gate opened (operator safety)	-	Safety stop 1 (path-maintaining)
External E-STOP pressed	Safety stop 1 (path-maintaining)	
External safety stop	Safety stop 1 (path-maintaining)	

Triggers for manual guidance

If an enabling device is configured for manual guidance, the following additional triggers for stop reactions are permanently defined:

Trigger	T1, CRR	T2, AUT
Manual guidance enabling switch released	Safety stop 1 (path-maintaining)	-
Manual guidance enabling switch pressed fully down (panic position)	Safety stop 1 (path-maintaining)	-
Maximum permissible velocity exceeded while manual guidance enabling signal is set	Safety stop 1 (path-maintaining)	

A maximum permissible velocity of 250 mm/s is preconfigured for manual guidance. The maximum permissible velocity can be adapted.

The value for the maximum permissible velocity must be determined as part of a risk assessment.

(>>> 13.8.5.3 "Velocity monitoring during manual guidance" Page 195)

3.2.6 Non-safety-oriented functions

3.2.6.1 Mode selection

The industrial robot can be operated in the following modes:

- Manual Reduced Velocity (T1)
- Manual High Velocity (T2)
- Automatic (AUT)
- Controlled robot retraction (CRR)

Operating mode	Use	Velocities
T1	Programming, teaching and testing of programs.	■ Program verification: Reduced programmed velocity, maximum 250 mm/s ■ Jog mode: Jog velocity, maximum 250 mm/s ■ Manual guidance: No limitation of the velocity, but safety-oriented velocity monitoring in accordance with the safety configuration **Note:** The maximum velocity of 250 mm/s does not apply to a mobile platform.
T2	Testing of programs Only possible with safety gate closed	■ Program verification: Programmed velocity ■ Jog mode: Not possible

Operating mode	Use	Velocities
AUT	Automatic execution of programs For industrial robots with and without higher-level controllers	■ Program mode: Programmed velocity ■ Jog mode: Not possible
CRR	■ Motion taking the industrial robot out of a violated Cartesian or axis-specific range ■ Motion taking the industrial robot out of a violated range of the tool orientation ■ Retraction of the industrial robot when it has become jammed due to a violation of force or torque limits ■ Motion of the industrial robot if mastering is lost for at least one position sensor during active Cartesian velocity monitoring CRR is an operating mode which is available when the industrial robot is stopped by the safety controller for one of the following reasons: ■ Industrial robot violates a safely monitored space. ■ Orientation of the safety-oriented tool is outside the safely monitored range. ■ Industrial robot violates a safely monitored force or torque limit. ■ A position sensor is not mastered during active Cartesian velocity monitoring.	■ Program mode: Reduced programmed velocity, maximum 250 mm/s ■ Jog mode: Jog velocity, maximum 250 mm/s

3.2.6.2 Software limit switches

The axis ranges of all manipulator axes are limited by means of non-safety-oriented software limit switches. These software limit switches only serve as machine protection and are preset in such a way that the manipulator is stopped under servo control if the axis limit is exceeded, thereby preventing damage to the mechanical equipment.

3.3 Additional protective equipment

3.3.1 Jog mode

In the operating modes T1 (Manual Reduced Velocity), T2 (Manual High Velocity) and CRR, the robot controller can only execute programs in jog mode. This means that it is necessary to hold down an enabling switch and the Start key in order to execute a program.

- Releasing the enabling switch on the smartPAD triggers a safety stop. (>>> 3.2.5 "Triggers for safety-oriented stop reactions" Page 31)
- Pressing the enabling switch on the smartPAD fully down triggers a safety stop 1.

- Releasing the Start key triggers a stop of Stop category 1 (path-maintaining).

3.3.2 Labeling on the industrial robot

All plates, labels, symbols and marks constitute safety-relevant parts of the industrial robot. They must not be modified or removed.

Labeling on the industrial robot consists of:

- Identification plates
- Warning signs
- Safety symbols
- Designation labels
- Cable markings
- Rating plates

> Further information is contained in the technical data of the operating instructions or assembly instructions of the components of the industrial robot.

3.3.3 External safeguards

The access of persons to the danger zone of the industrial robot must be prevented by means of safeguards. Alternatively, the requirements for collaborative operation in accordance with EN ISO 10218 must be met. It is the responsibility of the system integrator to ensure this.

Physical safeguards must meet the following requirements:

- They meet the requirements of EN 953.
- They prevent access of persons to the danger zone and cannot be easily circumvented.
- They are sufficiently fastened and can withstand all forces that are likely to occur in the course of operation, whether from inside or outside the enclosure.
- They do not, themselves, represent a hazard or potential hazard.
- The prescribed minimum clearance from the danger zone is maintained.

Safety gates (maintenance gates) must meet the following requirements:

- They are reduced to an absolute minimum.
- The interlocks (e.g. safety gate switches) are linked to the configured operator safety inputs of the robot controller.
- Switching devices, switches and the type of switching conform to the requirements of Performance Level d and category 3 according to EN ISO 13849-1.
- Depending on the risk situation: the safety gate is additionally safeguarded by means of a locking mechanism that only allows the gate to be opened if the manipulator is safely at a standstill.
- The device for setting the signal for operator safety, e.g. the button for acknowledging the safety gate, is located outside the space limited by the safeguards.

> Further information is contained in the corresponding standards and regulations. These also include EN 953.

Other safety equipment

Other safety equipment must be integrated into the system in accordance with the corresponding standards and regulations.

3.4 Safety measures

3.4.1 General safety measures

The industrial robot may only be used in perfect technical condition in accordance with its intended use and only by safety-conscious persons. Operator errors can result in personal injury and damage to property.

It is important to be prepared for possible movements of the industrial robot even after the robot controller has been switched off and locked out. Incorrect installation (e.g. overload) or mechanical defects (e.g. brake defect) can cause the manipulator to sag. If work is to be carried out on a switched-off industrial robot, the manipulator must first be moved into a position in which it is unable to move on its own, whether the payload is mounted or not. If this is not possible, the manipulator must be secured by appropriate means.

DANGER In the absence of operational safety functions and safeguards, the industrial robot can cause personal injury or material damage. If safety functions or safeguards are dismantled or deactivated, the industrial robot may not be operated.

WARNING Standing underneath the robot arm can cause death or serious injuries. Especially if the industrial robot is moving objects that can become detached (e.g. from a gripper). For this reason, standing underneath the robot arm is prohibited!

smartPAD

The user must ensure that the industrial robot is only operated with the smartPAD by authorized persons.

Modifications

After modifications to the industrial robot, checks must be carried out to ensure the required safety level. The valid national or regional work safety regulations must be observed for this check. The correct functioning of all safety functions must also be tested.

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

After modifications to the industrial robot, existing programs must always be tested first in Manual Reduced Velocity mode (T1). This applies to all components of the industrial robot and includes modifications to the software and configuration settings.

The robot may not be connected and disconnected when the robot controller is running.

Faults

The following tasks must be carried out in the case of faults in the industrial robot:

- Switch off the robot controller and secure it (e.g. with a padlock) to prevent unauthorized persons from switching it on again.
- Indicate the fault by means of a label with a corresponding warning (tag-out).
- Keep a record of the faults.
- Eliminate the fault and carry out a function test.

3.4.2 Transportation

Manipulator

The prescribed transport position of the manipulator must be observed. Transportation must be carried out in accordance with the operating instructions or assembly instructions of the robot.

Avoid vibrations and impacts during transportation in order to prevent damage to the manipulator.

Robot controller

The prescribed transport position of the robot controller must be observed. Transportation must be carried out in accordance with the operating instructions or assembly instructions of the robot controller.

Avoid vibrations and impacts during transportation in order to prevent damage to the robot controller.

3.4.3 Start-up and recommissioning

Before starting up systems and devices for the first time, a check must be carried out to ensure that the systems and devices are complete and operational, that they can be operated safely and that any damage is detected.

The valid national or regional work safety regulations must be observed for this check. The correct functioning of all safety functions must also be tested.

The password to activate the safety configuration must be changed before start-up. This password may only be communicated to trained safety maintenance technicians who are authorized to activate the safety configuration.
(>>> 13.7.3 "Changing the password for activating the safety configuration" Page 190)

DANGER The robot controller is preconfigured for the specific industrial robot. If cables are interchanged, the manipulator may receive incorrect data and can thus cause personal injury or material damage. If a system consists of more than one manipulator, always connect the connecting cables to the manipulators and their corresponding robot controllers.

If additional components (e.g. cables), which are not part of the scope of supply of KUKA Roboter GmbH, are integrated into the industrial robot, the user is responsible for ensuring that these components do not adversely affect or disable safety functions.

NOTICE If the internal cabinet temperature of the robot controller differs greatly from the ambient temperature, condensation can form, which may cause damage to the electrical components. Do not put the robot controller into operation until the internal temperature of the cabinet has adjusted to the ambient temperature.

Function test

The following tests must be carried out before start-up and recommissioning:

General test:

It must be ensured that:

- The industrial robot is correctly installed and fastened in accordance with the specifications in the documentation.
- There are no foreign bodies or loose parts on the industrial robot.
- All required safety equipment is correctly installed and operational.

- The power supply ratings of the industrial robot correspond to the local supply voltage and mains type.
- The ground conductor and the equipotential bonding cable are sufficiently rated and correctly connected.
- The connecting cables are correctly connected and the connectors are locked.

Test of the safety functions:

A function test must be carried out for all the safety-oriented functions to ensure that they are working correctly:

(>>> 13.11 "Safety acceptance overview" Page 226)

Test of the safety-relevant mechanical and electromechanical components:

The following tests must be performed prior to start-up and at least once every 12 months unless otherwise determined in accordance with a workplace risk assessment:

- Press the EMERGENCY STOP device on the smartPAD. A message must appear on the smartPAD indicating that the EMERGENCY STOP has been triggered, and no error message for the EMERGENCY STOP device may be displayed.
- For all 3 enabling switches on the smartPAD and for the enabling switch on the media flange Touch (if present)

  Move the robot in Test mode and release the enabling switch. Robot motion must be stopped and no error message for the enabling device may be displayed. If the state of the enabling switch is configured at an output, the test can also be performed via the output.
- For all 3 enabling switches on the smartPAD and for the enabling switch on the media flange Touch (if present)

  Move the robot in Test mode and press the enabling switch down fully. Robot motion must be stopped and no error message for the enabling device may be displayed. If the state of the enabling switch is configured at an output, the test can also be performed via the output.
- Turn the keyswitch on the smartPAD to the right and then back again. No error message may be displayed on the smartPAD.
- Test the switch-off capability of the safe inputs by switching the robot controller off and then on again. After it is switched on, no error message for a safe output may be displayed.

In the case of incomplete start-up of the system, additional substitute measures for minimizing risk must be taken and documented, e.g. installation of a safety fence, attachment of a warning sign, locking of the main switch, etc. Start-up is incomplete, for example, if not all safety functions have yet been implemented, or if a function test of the safety functions has not yet been carried out.

Test of the functional capability of the brakes:

For the KUKA LBR iiwa (all variants) a brake test is available which can be used to check whether the brakes on all axes apply sufficient braking torque.

(>>> 8 "Brake test" Page 109)

Unless otherwise determined by a risk assessment, the brake test must be performed regularly:

- The brake test must be carried out for each axis during start-up and recommissioning of the industrial robot.
- The brake test must be performed daily during operation.

The user can carry out a risk assessment to determine whether the brake test is required for the specific application and, if so, how often it is to be performed.

3.4.4 Manual mode

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

- Jog mode
- Teaching
- Program verification

The following must be taken into consideration in manual mode:

- New or modified programs must always be tested first in Manual Reduced Velocity mode (T1).
- The manipulator and its tooling must never touch or project beyond the safety fence.
- Workpieces, tooling and other objects must not become jammed as a result of the industrial robot motion, nor must they lead to short-circuits or be liable to fall off.
- All setup work must be carried out, where possible, from outside the safeguarded area.

If the setup work has to be carried out inside the safeguarded area, the following must be taken into consideration:

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

- If it can be avoided, there must be no other persons inside the safeguarded area.

  If it is necessary for there to be several persons inside the safeguarded area, the following must be observed:
  - Each person must have an enabling device.
  - All persons must have an unimpeded view of the industrial robot.
  - 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)**:

- This mode may only be used if the application requires a test at a velocity higher than Manual Reduced Velocity.
- Teaching is not permissible in this operating mode.
- Before commencing the test, the operator must ensure that the enabling devices are operational.
- There must be no-one present inside the safeguarded area. It is the responsibility of the operator to ensure this.

3.4.5 Automatic mode

Automatic mode is only permissible in compliance with the following safety measures:

- All safety equipment and safeguards are present and operational.
- There are no persons in the system. Alternatively, the requirements for collaborative operation in accordance with EN ISO 10218 have been met.
- The defined working procedures are adhered to.

If the manipulator comes to a standstill for no apparent reason, the danger zone must not be entered until an EMERGENCY STOP has been triggered.

3.4.6 Maintenance and repair

After maintenance and repair work, checks must be carried out to ensure the required safety level. The valid national or regional work safety regulations must be observed for this check. The correct functioning of all safety functions must also be tested.

The purpose of maintenance and repair work is to ensure that the system is kept operational or, in the event of a fault, to return the system to an operational state. Repair work includes troubleshooting in addition to the actual repair itself.

The following safety measures must be carried out when working on the industrial robot:

- Carry out work outside the danger zone. If work inside the danger zone is necessary, the user must define additional safety measures to ensure the safe protection of personnel.
- Switch off the industrial robot and secure it (e.g. with a padlock) to prevent it from being switched on again. If it is necessary to carry out work with the robot controller switched on, the user must define additional safety measures to ensure the safe protection of personnel.
- If it is necessary to carry out work with the robot controller switched on, this may only be done in operating mode T1.
- Label the system with a sign indicating that work is in progress. This sign must remain in place, even during temporary interruptions to the work.
- The EMERGENCY STOP systems must remain active. If safety functions or safeguards are deactivated during maintenance or repair work, they must be reactivated immediately after the work is completed.

⚠ DANGER Before work is commenced on live parts of the robot system, the main switch must be turned off and secured against being switched on again. The system must then be checked to ensure that it is deenergized.
It is not sufficient, before commencing work on live parts, to execute an EMERGENCY STOP or a safety stop, or to switch off the drives, as this does not disconnect the robot system from the mains power supply. Parts remain energized. Death or severe injuries may result.

Faulty components must be replaced using new components with the same article numbers or equivalent components approved by KUKA Roboter GmbH for this purpose.

Cleaning and preventive maintenance work is to be carried out in accordance with the operating instructions.

Robot controller

Even when the robot controller is switched off, parts connected to peripheral devices may still carry voltage. The external power sources must therefore be switched off if work is to be carried out on the robot controller.

The ESD regulations must be adhered to when working on components in the robot controller.

Voltages in excess of 60 V can be present in various components for several minutes after the robot controller has been switched off! To prevent life-threatening injuries, no work may be carried out on the industrial robot in this time.

Water and dust must be prevented from entering the robot controller.

3.4.7 Decommissioning, storage and disposal

The industrial robot must be decommissioned, stored and disposed of in accordance with the applicable national laws, regulations and standards.

3.4.8 Safety measures for “single point of control”

Overview

If certain components in the industrial robot are operated, safety measures must be taken to ensure complete implementation of the principle of “single point of control” (SPOC).

Components:

- Tools for configuration of bus systems with online functionality

> The implementation of additional safety measures may be required. This must be clarified for each specific application; this is the responsibility of the user of the system.

Since only the system integrator knows the safe states of actuators in the periphery of the robot controller, it is his task to set these actuators to a safe state.

T1, T2, CRR

In modes T1, T2 and CRR, a robot motion can only be initiated if an enabling switch on the smartPAD is held down.

Tools for configuration of bus systems

If these components have an online functionality, they can be used with write access to modify programs, outputs or other parameters of the robot controller, without this being noticed by any persons located inside the system.

- KUKA Sunrise.Workbench
- WorkVisual from KUKA
- Tools from other manufacturers

Safety measures:

- In the test modes, programs, outputs or other parameters of the robot controller must not be modified using these components.

3.5 Applied norms and regulations

Name	Definition	Edition
2006/42/EC	**Machinery Directive:** Directive 2006/42/EC of the European Parliament and of the Council of 17 May 2006 on machinery, and amending Directive 95/16/EC (recast)	2006
2004/108/EC	**EMC Directive:** Directive 2004/108/EC of the European Parliament and of the Council of 15 December 2004 on the approximation of the laws of the Member States relating to electromagnetic compatibility and repealing Directive 89/336/EEC	2004
EN ISO 13850	**Safety of machinery:** Emergency stop - Principles for design	2008

EN ISO 13849-1	**Safety of machinery:** Safety-related parts of control systems - Part 1: General principles of design	2008
EN ISO 13849-2	**Safety of machinery:** Safety-related parts of control systems - Part 2: Validation	2012
EN ISO 12100	**Safety of machinery:** General principles of design, risk assessment and risk reduction	2010
EN ISO 10218-1	**Industrial robots:** Safety **Note:** Content equivalent to **ANSI/RIA R.15.06-2012, Part 1**	2011
EN 614-1 + A1	**Safety of machinery:** Ergonomic design principles - Part 1: Terms and general principles	2009
EN 61000-6-2	**Electromagnetic compatibility (EMC):** Part 6-2: Generic standards; Immunity for industrial environments	2005
EN 61000-6-4 + A1	**Electromagnetic compatibility (EMC):** Part 6-4: Generic standards; Emission standard for industrial environments	2011
EN 60204-1 + A1	**Safety of machinery:** Electrical equipment of machines - Part 1: General requirements	2009

4 Installing KUKA Sunrise.Workbench

4.1 PC system requirements

Hardware

Minimum requirements

- PC with Pentium IV processor, min. 1500 MHz
- 1 GB RAM
- 1 GB free hard disk space
- DirectX8-compatible graphics card with a resolution of 1024x768 pixels

Recommended specifications

- PC with Pentium IV processor 2500 MHz
- 2 GB RAM
- DirectX8-compatible graphics card with a resolution of 1280x1024 pixels

Software

- Windows 7

The following software is required for bus configuration:

- WorkVisual 3.0

4.2 Installing Sunrise.Workbench

Precondition

- Local administrator rights

Procedure

1. Start installation via the file **Setup.exe**.
2. Select the language and confirm with **OK**. The installation wizard **Sunrise Workbench Setup** opens.
3. Press **Next >** to switch to the next page.
4. Accept the license agreement. Press **Next >** to switch to the next page.
5. The default folder for installation is specified in the **Folder** box.

   A different installation folder can be selected by pressing **Browse…**. Press **Next >** to switch to the next page.
6. If no desktop link for Sunrise.Workbench should be created: deactivate the **Desktop** option (uncheck the box).
7. Press **Next >** to switch to the next page and click on **Install**. Sunrise.Workbench is installed.
8. Once installation is completed, click on **Finish** to close the installation wizard.

   Sunrise.Workbench is then started directly. If this is not desired, deactivate the **Launch Sunrise Workbench** option (uncheck the box).

4.3 Uninstalling Sunrise.Workbench

> It is advisable to archive all relevant data before uninstalling a software package.

Precondition

- Local administrator rights

Procedure

1. In the Windows Start menu, select **Control Panel** (> **Programs**) > **Programs and Functions**. The list of installed programs is displayed.
2. Select the entry **Sunrise Workbench**. Click on **Uninstall** and answer the request for confirmation with **Yes**.

Alternative procedure

- In the Windows Start menu, open the installation directory specified during installation and click on **Uninstall**.

4.4 Installing the language package in Sunrise.Workbench

Description

The user interface of Sunrise.Workbench is currently available in the following languages:

German	Italian
English	Spanish
French	

Languages which are only available after software is delivered can be installed later if required.

Precondition

- Local administrator rights
- Language containing the desired language is available.

Procedure

1. Close Sunrise.Workbench.
2. Execute the file **SWB_LanguagePack-<Version>-Setup.exe**. The language package is installed.
3. Restart Sunrise.Workbench. Sunrise.Workbench starts automatically in the Windows language that is locally set.

5 Operation of KUKA Sunrise.Workbench

5.1 Starting Sunrise.Workbench

Procedure

1. Double-click on the **Sunrise.Workbench** icon on the desktop.

   Alternative:

   In the Windows Start menu, open the installation directory and double-click on **Sunrise Workbench**.

   The **Workspace Launcher** window opens.
2. In the **Workspace** box, specify the directory for the workspace in which projects are to be saved.
   - A default directory is suggested. The directory can be changed by clicking on the **Browse…** button.
   - If the workspace should not be queried the next time Sunrise.Workbench is started, activate the option **Use this as the default value[…]** (set check mark).

   Confirm the settings with **OK**.
3. A welcome screen opens the first time Sunrise.Workbench is started. There are different options here.
   - Click on **Workbench** to open the user interface of Sunrise.Workbench.

     (>>> 5.2 "Overview of the user interface of Sunrise.Workbench" Page 45)
   - Click on **New Sunrise project** to create a new Sunrise project directly. The project creation wizard opens.

     (>>> 5.3 "Creating a Sunrise project with a template" Page 48)

5.2 Overview of the user interface of Sunrise.Workbench

The user interface of KUKA Sunrise.Workbench consists of several views. The combination of several views is called a perspective. KUKA Sunrise.Workbench offers various preconfigured perspectives.

The **Programming** perspective is opened by default. Additional perspectives can be displayed.

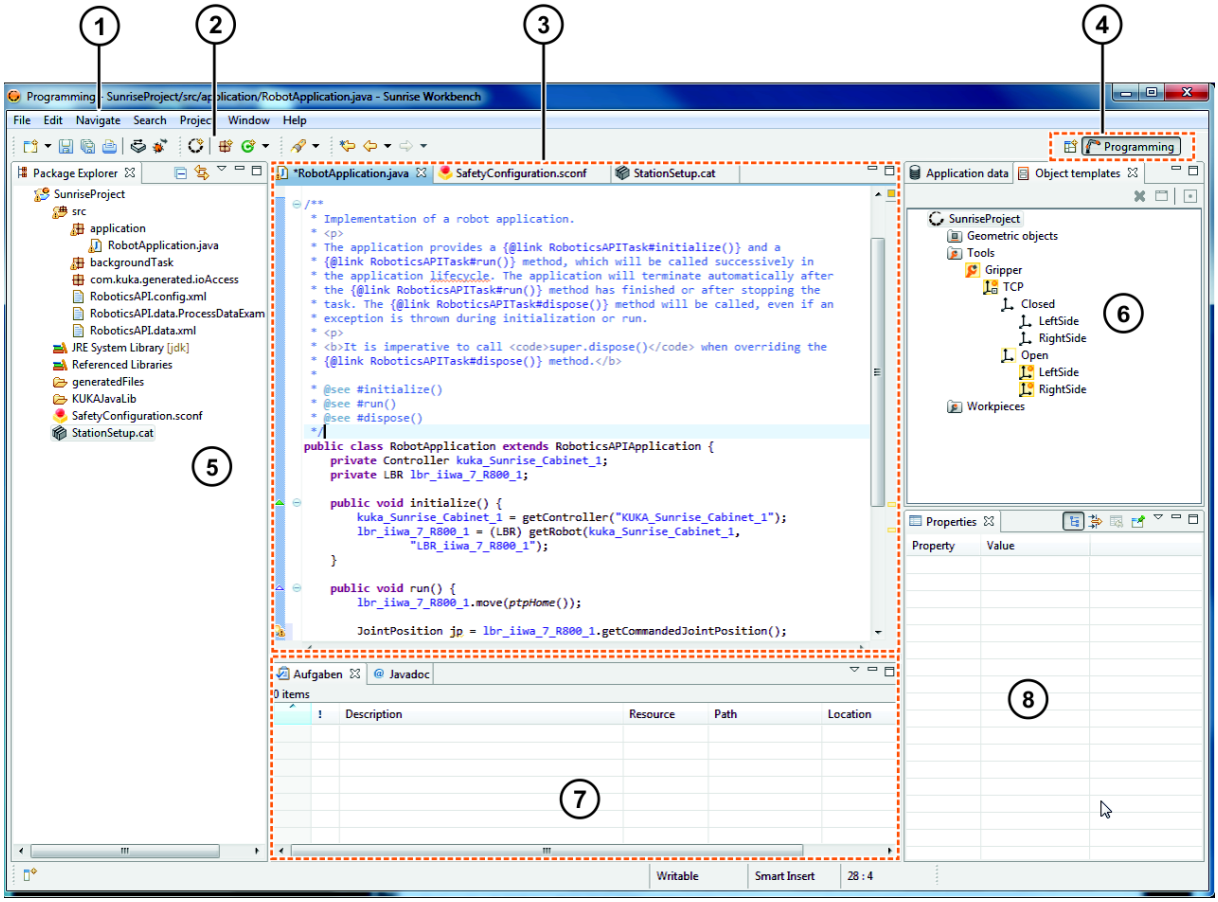


Fig. 5-1: Overview of user interface – “Programming” perspektive

Item	Description
1	Menu bar
2	Toolbars (>>> 5.2.3 "Toolbars" Page 48)
3	Editor area Opened files, e.g. robot applications, are displayed and edited here.
4	Perspective selection Here it is possible to switch between various previously-used perspectives by clicking on the name of the appropriate perspective or on the **Open Perspective** icon. (>>> 5.2.2 "Displaying different perspectives on the user interface" Page 47)
5	**Package Explorer** view This view contains the projects created and their corresponding files.
6	**Application data** and **Object templates** views ■ **Application data**: This view displays the frames created for a project in a tree structure. ■ **Object templates**: This view displays the geometrical objects, tools and workpieces created for a project in a tree structure.

Item	Description
7	**Tasks** and **Javadoc** views ■ **Tasks**: Displays tasks which a user has created ■ **Javadoc**: Displays Javadoc information about the selected elements of a Java application
8	**Properties** view If an object, e.g. a project, frame or tool, is selected in a view, its properties are displayed here.

5.2.1 Repositioning the views

Procedure

1. Grip the view by the title bar while holding down the left mouse button and move it to the desired position on the user interface.
   The possible positions for the view are indicated here by a gray frame.
2. Release the mouse button when the desired position for the view is selected.

5.2.2 Displaying different perspectives on the user interface

Description

The user interface can be displayed in different perspectives. These can be selected via the menu sequence **Window** > **Open Perspective** or by clicking on the **Open Perspective** icon.

The perspectives are tailored to different types of work:

Perspective	Type of work
Programming	This perspective has views suitable for editing Sunrise projects. For example, for station configuration, safety configuration and application development.
Debug	This perspective has views suitable for locating faults and eliminating programming faults.

Perspectives can be adapted to the needs of the user. Examples:

- Creating own perspectives
- Showing/hiding views
- Showing/hiding menus
- Showing/hiding menu items

It is possible to save the adapted perspective as a default setting for the perspective or under a separate name of its own.

Procedure

To display views in the current perspective:

- Select the menu sequence **Window** > **Show View** and the desired view.
  Further views can be selected by clicking the menu item **Other…**.

To reset the current perspective to the default setting:

- Select the menu sequence **Window** > **Reset Perspective…** and answer the request for confirmation with **Yes**.

To save user-defined perspectives:

1. Select the menu sequence **Window** > **Save Perspective As...**.
2. In the **Name** box, enter a name for the perspective and confirm it with **OK**.

If an existing perspective is selected and overwritten, the perspective will be opened with these settings in the future.

5.2.3 Toolbars

The buttons available by default on the toolbar depend on the active perspective. The buttons of the **Programming** perspective are described here.

Programming

Icon	Name / description
	New Opens the wizard for creating new documents.
	Save Saves the currently opened and selected file.
	Save All Saves all files and projects that have been edited since the last save.
	Print Opens the menu for printing a file.
	Synchronize Project Synchronizes the selected project with the robot controller.
	Debug project Establishes a remote connection to the robot controller in order to debug an application during ongoing operation.
	Sunrise Project Opens the wizard for creating a new Sunrise project.
	New Java Package Opens the wizard for creating a new Java package in the selected project.
	New Java Class Opens the wizard for creating a new Java class in the selected project.
	Search Opens the wizard to search for words or text modules.
	Last Edit Location Switches to the last edit location in the currently opened and selected file.
	Back to ... Switches back to the previous edit steps.
	Forward to ... Switches forward again to the subsequent edit steps.

5.3 Creating a Sunrise project with a template

Procedure

1. Select the menu sequence **File** > **New** > **Sunrise project**. The project creation wizard opens.
2. Enter the IP address of the robot controller to be created for the project in the **IP address of controller:** box.

   It is possible to change the address again during subsequent project configuration.

The following address ranges are used by default by the robot controller for internal purposes. IP addresses from these ranges cannot therefore be assigned by the user.

- 192.168.0.0 … 192.168.0.255
- 172.16.0.0 … 172.16.255.255
- 172.17.0.0 … 172.17.255.255

3. Retain the **Create new project (offline)** setting.

   Press **Next >** to switch to the next page.
4. Enter a name for the project in the **Project name:** box.
5. The default directory for projects is given in the **Location:** box.

   A different directory can be selected if desired: To do so, remove the check mark at **Use default location** and select **Browse…**. Then use the **Browse for Folder** dialog to select the desired file path and confirm with **OK**.

   Press **Next >** to switch to the next page.
6. Select a template from the **Topology template** list.

   The template determines which elements are subsequently preselected in the station configuration on the **Topology** tab. Irrespective of the template that is selected here, all elements are always available in **Topology** and the preselection can be modified as required.

   Press **Next >** to switch to the next page.
7. If the selected template is a robot with a media flange, select the corresponding media flange.

The weight and height of the selected media flange are automatically taken into consideration by the system software.

8. By default, the direction of installation of a floor-mounted robot is set (A=0°, B=0°, C=0°).

   In the case of a ceiling- or wall-mounted robot, enter the direction of installation relative to the floor-mounted robot:

   a. **Rotation about the Z axis in ° (A angle)**: Rotation of angle A about the Z axis of the robot base coordinate system (-180° ≤ A ≤ 180°).
   b. **Rotation about the Y axis in ° (B angle)**: Rotation of angle B about the Y axis (-90° ≤ B ≤ 90°). The rotation about the Y axis is relative to the rotated coordinate system from step a.
   c. **Rotation about the X axis in ° (C angle)**: Rotation of angle C about the X axis (-180° ≤ C ≤ 180°). The rotation about the X axis is relative to the rotated coordinate system from step b.

   (>>> 6.7 "Coordinate systems" Page 72)

The mounting orientation of the robot must be entered correctly. An incorrectly entered mounting orientation can have the following effects:

- Unexpected robot behavior under impedance control
- Changed position of previously taught frames
- Prevention of motion enable due to collision detection and TCP force monitoring
- Unexpected behavior during jogging in the world or base coordinate system

   Press **Next >** to switch to the next page.
9. A summary of information on the project is displayed.

Remove the check mark at **Create application (starts other wizard)** and click on **Finish**. The project is created and added to the **Package Explorer**.

If the check mark has been set at **Create application (starts other wizard)**, the wizard for application creation opens. The first robot application for this newly created project can then be created directly.

(>>> 5.4.2 "Creating a robot application with a package" Page 52)

Description

The figure shows the structure of a newly created Sunrise project, in which no robot applications have yet been created or other changes have been made. The robot configured for the Sunrise project has a media flange.

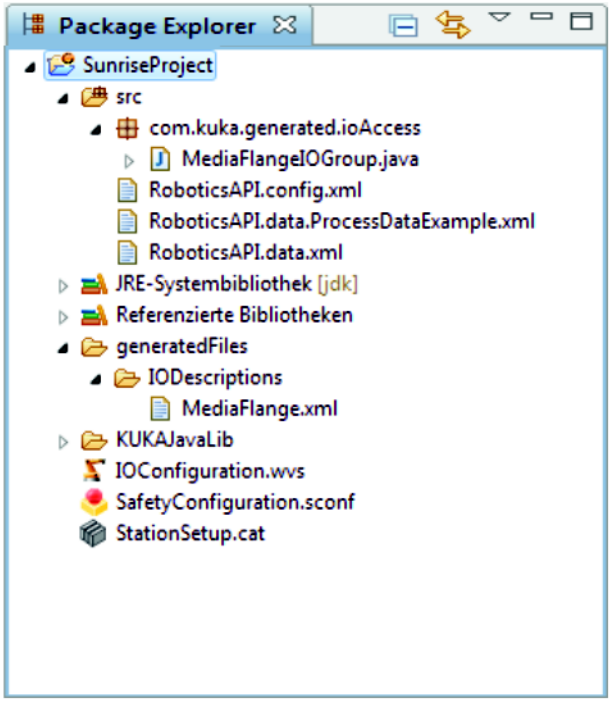


Fig. 5-2: Overview of the project structure

Element	Description
src	Source folder of the project The created robot applications and Java classes are stored in the source folder. The Java package **com.kuka.generated.ioAccess** contains the Java class **MediaFlangeIOGroup.java**. The class already contains the methods required for programming in order to access the inputs/outputs of the media flange. Further information on the use of inputs/outputs in programming can be found here: (>>> 15.12 "Inputs/outputs" Page 300) The source folder also contains various XML files in which, in addition to the configuration data, the runtime data are saved; for example, the frames and tools created by the user. The XML files can be displayed but not edited.
JRE System Library	Java Runtime Environment system library The system library contains the Java class libraries which can be used for standard Java programming.

Element	Description
Referenced libraries	Referenced libraries The referenced libraries can be used in the project. By default, the robot-specific Java class libraries are automatically added when a Sunrise project is created. The user has the option of adding further libraries.
generatedFiles	Folder with subfolder **IODescriptions** The data for the inputs/outputs configured for the media flange are saved in an XML file. The XML file can be displayed but not edited.
KUKAJavaLib	Folder with special libraries required for robot programming.
IOConfiguration.wvs	I/O configuration for the media flange The I/O configuration contains the complete bus structure of the media flange, including the I/O mapping. The I/O configuration can be opened, edited and re-exported into the Sunrise project in WorkVisual. **Note**: The I/O configuration is only carried out automatically for the existing inputs and outputs on the selected media flange. Further EtherCAT devices connected to the media flange must be configured with WorkVisual. (>>> 11 "Bus configuration" Page 153)
SafetyConfiguration.sconf	The file contains the safety functions preconfigured by KUKA. The configuration can be displayed and edited. (>>> 13 "Safety configuration" Page 169)
StationSetup.cat	The file contains the station configuration for the station (controller) selected when the project was created. The configuration can be displayed and edited. The system software can be installed on the robot controller via the station configuration. (>>> 10 "Station configuration and installation" Page 147)

The **generatedFiles** folder is used by the system and must not be used for saving files created by the user.

5.4 Creating a new robot application

Robot applications are Java programs. They define tasks that are to be executed in a station. They are transferred to the robot controller with the project and can be selected and executed using the smartPAD.

Robot applications are grouped into packages. This makes programming more transparent and makes it easier to use a package later in other projects.

5.4.1 Creating a new Java package

Procedure

1. Select the project in the **Package Explorer**.
2. Select the menu sequence **File** > **New** > **Package**. The **New Java Package** window opens.
3. Enter a name for the package in the **Name** box.
4. Click on **Finish**. The package is created and added to the “src” folder for the project.

The package does not yet contain any files. An empty package is indicated by a white package icon. As soon as a package contains files, the icon turns brown.

5.4.2 Creating a robot application with a package

Procedure

1. Select the project in the **Package Explorer**.
2. Select the menu sequence **File** > **New** > **Robot application**. The **New robot application** window opens.
3. In the **Package** box, enter the name of the package in which the application should be created.
4. Enter a name for the package in the **Name** box.
5. Click on **Finish**. The application and package are created and inserted into the project.

   The *Name*.java application is opened in the editor area.

5.4.3 Creating a robot application for an existing package

Procedure

1. Select the package in the **Package Explorer**.
2. Select the menu sequence **File** > **New** > **Robot application**. The **New robot application** window opens.
3. Enter a name for the package in the **Name** box.
4. Click on **Finish**. The application is created and inserted into the package.

   The *Name*.java application is opened in the editor area.

5.5 Creating a new background task

Background tasks are Java programs that are executed on the robot controller parallel to the robot application. For example, they can perform control tasks for peripheral devices.

The use and programming of background tasks are described here:
(>>> 16 "Background tasks" Page 385)

The following properties are defined when the task is created:

- Start type of the task
  - **Automatic**

    The task is automatically started after the robot controller has booted (default).
  - **Manual**

    The task must be started manually via the smartPAD. (This function is not yet supported.)
- Task template
  - **Cyclic background task**

    Template for tasks that are to be executed cyclically (default)
  - **Non-cyclic background task**

    Template for tasks that are to be executed once

5.5.1 Creating a background task with a package

Procedure

1. Select the project in the **Package Explorer**.
2. Select the menu sequence **File** > **New** > **Background task**. The **New background task** window is opened.

3. In the **Package** box, enter the name of the package in which the task is to be created.
4. Enter a name for the task in the **Name** box.
5. Click on **Next >** and select the start type of the task.
6. Click on **Next >** and select the task template.
7. Click on **Finish**. The task and package are created and inserted into the project.

   The *Name*.java task is opened in the editor area.

5.5.2 Creating a background task for an existing package

Procedure

1. Select the package in the **Package Explorer**.
2. Select the menu sequence **File** > **New** > **Background task**. The **New background task** window is opened.
3. Enter a name for the task in the **Name** box.
4. Click on **Next >** and select the start type of the task.
5. Click on **Next >** and select the task template.
6. Click on **Finish**. The task is created and inserted into the package.

   The *Name*.java task is opened in the editor area.

5.6 Workspace

The directory in which the created projects and user-defined settings for Sunrise.Workbench are saved is called the workspace. The directory for the workspace must be defined by the user when Sunrise.Workbench is started for the first time. It is possible to create additional workspaces in Sunrise.Workbench and to switch between them.

5.6.1 Creating a new workspace

Procedure

1. Select the menu sequence **File** > **Switch Workspace** > **Other...**. The **Workspace Launcher** window opens.
2. In the **Workspace** box, manually enter the path to the new project directory.

   Alternative:

   - Click on **Browse...** to navigate to the directory where the new workspace should be created.
   - Create the new project directory by clicking on **Create new folder**. Click on **OK** to confirm.

     The path to the new project directory is inserted in the **Workspace** box.
3. Click on **OK** to confirm the new workspace. Sunrise.Workbench restarts and the welcome screen opens.

5.6.2 Switching to an existing workspace

Precondition

- Other workspaces are available.

Procedure

1. Select the menu sequence **File** > **Switch Workspace** > **Other...**. The **Workspace Launcher** window opens.
2. Navigate to the desired workspace using **Browse…** and select it.
3. Confirm with **OK**. The path to the new project directory is applied in the **Workspace Launcher** window.

4. Confirm the selected workspace with **OK**. Sunrise.Workbench restarts and opens the selected workspace.

5.6.3 Switching between the most recently opened workspaces

Precondition

- Other workspaces are available.

Procedure

1. Select the menu sequence **File** > **Switch Workspace**. The most recently used workspaces are displayed in a list (max. 4).
2. Select the desired workspace from the list. Sunrise.Workbench restarts and opens the selected workspace.

5.6.4 Archiving projects

Procedure

1. Select the menu sequence **File** > **Export...**. The file export wizard opens.
2. In the **General** folder, select the **Archive File** option and click on **Next >**.
3. All the projects in the workspace are displayed in a list in the top left-hand area of the screen. Select the projects to be archived (set check mark).
4. Click on **Browse…** to navigate to the desired file location, enter the file name for the archive and click on **Save**.
5. Click on **Finish**. The archive file is created.

5.6.5 Loading projects from archive to the workspace

Precondition

- An archive file (e.g. a ZIP file) with the projects to be loaded is available.
- The workspace does not contain any project with the name of the project to be loaded.

Procedure

1. Select the menu sequence **File** > **Import…**. The file import wizard opens.
2. In the **General** folder, select the **Existing Projects into Workspace** option and click on **Next >**.
3. Activate the **Select archive file** radio button, click on **Browse…** to navigate to the desired archive file and select it.
4. Click on **Open**. All the projects in the archive are displayed in a list under **Projects**.
5. Select projects to be loaded to the workspace (check mark must be set).
6. Click on **Finish**. The selected projects are loaded.

5.6.6 Loading projects from the directory to the workspace

Precondition

- One or more projects are available in any directory.
- The workspace does not contain any project with the name of the project to be loaded.

Procedure

1. Select the menu sequence **File** > **Import…**. The file import wizard opens.
2. In the **General** folder, select the **Existing Projects into Workspace** option and click on **Next >**.
3. Activate the **Select root directory** radio button, click on **Browse…** to navigate to the desired directory and select it.
4. Click on **OK**. All the projects in the selected directory are displayed in a list under **Projects**.
5. Select projects to be loaded to the workspace (check mark must be set).
6. Click on **Finish**. The selected projects are loaded.

5.7 Sunrise projects with referenced Java projects

One or more Java projects can be referenced within a Sunrise project. The referencing of Java projects allows them to be used in any number of Sunrise projects and thus on different robot controllers.

The referenced Java projects can in turn reference further Java projects. Only one Sunrise project may exist among all the cross-referenced projects.

> When Sunrise projects are synchronized, referenced Java projects are also transferred onto the robot controller. If a further Sunrise project is referenced within a Sunrise project, synchronization is aborted with an error message.

5.7.1 Creating a new Java project

Procedure

1. Select the menu sequence **File** > **New** > **Project...**. The project creation wizard opens.
2. In the **Java** folder, select the **Java Project** option and click on **Next >**.
3. Enter the name of the Java project in the **Project name** box.
4. In the **JRE** area, select the JRE version that corresponds to the JRE version of the Sunrise project. This is generally JavaSE-1.6.
5. Click on **Next >** and then on **Finish**.
6. The first time a Java project is created in the workspace – or if the user's preference has not yet been specified in previous Java projects – a query is displayed asking whether the **Java** perspective should be opened.
   - Select **Yes** or **No** as appropriate.
   - If the query should not be displayed when the next Java project is created in the workspace, activate the **Remember my decision** option (set check mark).

> In the Java projects, all classes which should be referenced externally must be stored in a defined Java package. If referenced classes are created in the standard package, they cannot be found in the Sunrise project.

5.7.1.1 Inserting robot-specific class libraries in a Java project

Description

If a Java project is used for robot programming, the specific KUKA libraries required for this purpose must be inserted into the project. By default, these libraries are not contained in a Java project.

The KUKA libraries must be copied from a compatible Sunrise project. Ideally, this should be a Sunrise project in which the Java project is referenced or will be referenced. The precondition for compatibility of referenced projects is that the RoboticsAPI versions match.

Precondition

- At least one compatible Sunrise project is available in the workspace.

Procedure

1. Copy the **KUKAJavaLib** folder of a compatible Sunrise project: Right-click on the folder in the **Package Explorer** and select **Copy** from the context menu.
2. Insert the **KUKAJavaLib** folder into the Java project: Right-click on the desired Java project in the **Package Explorer** and select **Insert** from the context menu.
3. Right-click again on the Java project and select **Build Path** > **Configure Build Path…** from the context menu. The **Properties for Project** window opens.

4. Select the **Libraries** tab in the **Java Build Path** and click on the **Add JARs…** button. The **JAR Selection** window opens.
5. All the projects in the workspace are displayed in a list. Expand the Java project where the referenced libraries are to be inserted.
6. Expand the **KUKAJavaLib** folder and select the existing JAR files.
7. Confirm your selection with **OK**. The JAR files are inserted on the **Libraries** tab of the build path.
8. Close the window by clicking on **OK**. The referenced libraries are inserted into the Java project.

5.7.2 Referencing Java projects

Precondition

- The referenced classes are saved in a defined Java package (not in the standard package).
- For Java projects which use referenced KUKA libraries: In the referenced projects, the RoboticsAPI versions must match.

Procedure

1. In the **Package Explorer**, right-click on the project which is to be referenced for the Java project.
2. Select **Build Path** > **Configure Build Path…** from the context menu. The **Properties for Project** window opens.
3. Select the **Projects** tab in the **Java Build Path** and click on the **Add …** button. The **Required Project Selection** window opens.
4. All the projects in the workspace are displayed in a list. Select the Java projects to be referenced (set check mark).
5. Confirm your selection with **OK**. The selected projects are inserted on the **Projects** tab of the build path.
6. Close the window by clicking on **OK**.

5.7.3 Canceling the reference to Java projects

Description

References to inadvertently added projects or projects that are not required (any longer) can be removed.

Procedure

1. In the **Package Explorer**, right-click on the project from which referenced projects should be removed.
2. Select **Properties** from the context menu. The **Properties for Project** window opens.
3. Select the **Projects** tab in the **Java Build Path**.
4. Select the projects that are not required and click on **Remove**.
5. Close the window by clicking on **OK**.

5.8 Renaming an element in the Package Explorer

In the **Package Explorer** view, the names of inserted elements can be changed, e.g. the names of projects, Java packages and Java files.

5.8.1 Renaming a project or Java package

Procedure

1. Right-click on the desired project or Java package. Select **Refactoring** > **Rename** in the context menu. The **Rename Java Project** or **Rename Java Package** window opens.
2. In the **New name** box, enter the desired name. Confirm with **OK**.

5.8.2 Renaming a Java file

Procedure

1. Right-click on the desired Java file. Select **Refactoring** > **Rename** in the context menu. The **Rename Compilation Unit** window opens.
2. In the **New name** box, enter the desired name. Click on **Finish**.
3. Possible conflicts are indicated before the renaming is completed. After acknowledging and checking these, click on **Finish** once more.

5.9 Removing an element from Package Explorer

In the **Package Explorer** view, inserted elements can be removed again, e.g. entire projects or individual Java packages and Java files of a project.

5.9.1 Deleting an element from a project

Description

Elements created for a project can be deleted again. The elements are permanently deleted from the workspace and cannot be restored.

It is also possible to remove some – but not all – of the default elements of a project.

Procedure

1. Right-click on the element. Select **Delete** in the context menu.
2. Answer the request for confirmation with **OK**. The element is deleted.

5.9.2 Removing a project from Package Explorer

Description

With this procedure, a project is only removed from the **Package Explorer** and is retained in the directory for the workspace on the data storage medium.

If required, the project can be reloaded from the directory into the workspace. The project is then available again in the **Package Explorer**.

(>>> 5.6.6 "Loading projects from the directory to the workspace" Page 54)

Procedure

1. Right-click on the desired project. Select **Delete** in the context menu. A request for confirmation is displayed, asking if the project is really to be deleted.
2. The check box next to **Delete project content on disk (cannot be undone)** is activated by default. Leave it like this.
3. Confirm the request for confirmation with **OK**.

5.9.3 Deleting a project from the workspace

Description

With this procedure, a project is removed from the **Package Explorer** and permanently deleted from the directory for the workspace on the data storage medium. The project cannot be restored.

Procedure

1. Right-click on the desired project. Select **Delete** in the context menu. A request for confirmation is displayed, asking if the project is really to be deleted.
2. Activate the check box next to **Delete project content on disk (cannot be undone)**.
3. Confirm the request for confirmation with **OK**.

5.10 Activating the automatic change recognition

Description

The automatic change recognition is activated by default in Sunrise.Workbench. If it has been deactivated, this could mean that the Java classes and files required for the use of the signals may not be created in the Sunrise project when exporting an I/O configuration from WorkVisual.

Procedure

1. Select the menu sequence **Window** > **User definitions**. The **User definitions** window is opened.
2. Select **General** > **Workspace** in the directory structure in the left area of the window.
3. Activate the **Update via native hooks or polling** to activate the automatic change recognition.

6 Operating the KUKA smartPAD

6.1 KUKA smartPAD control panel

6.1.1 Front view

Function

The smartPAD is the hand-held control panel for the industrial robot. The smartPAD has all the operator control and display functions required for operation.

The smartPAD has a touch screen: the smartHMI can be operated with a finger or stylus. An external mouse or external keyboard is not necessary.

Overview

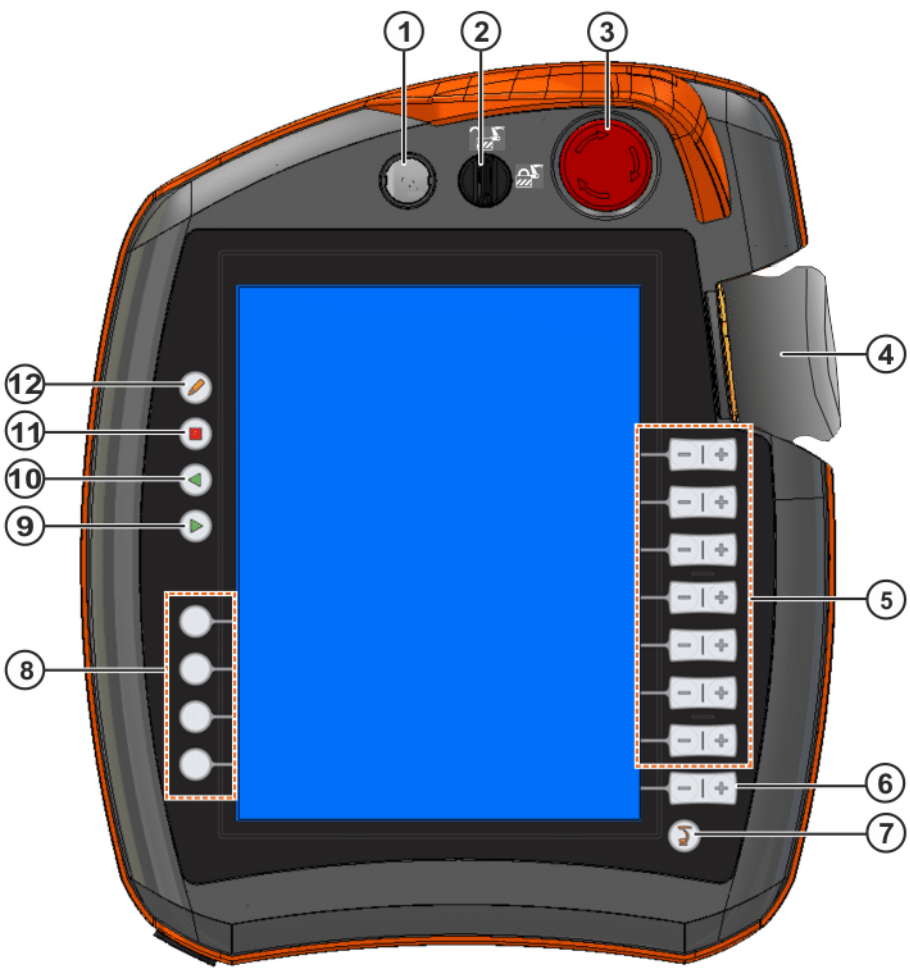


Fig. 6-1: KUKA smartPAD, front view

Item	Description
1	Button for disconnecting the smartPAD Currently without function
2	Keyswitch The connection manager is called by means of the keyswitch. The switch can only be turned if the key is inserted. The connection manager is used to change the operating mode. (>>> 6.6 "Changing the operating mode" Page 70)

Item	Description
3	EMERGENCY STOP device The robot can be stopped in hazardous situations using the EMERGENCY STOP device. The EMERGENCY STOP device locks itself in place when it is pressed.
4	Space Mouse Currently without function
5	Jog keys The jog keys are used to move the robot manually. (>>> 6.8 "Jogging the robot" Page 74)
6	Key for setting the jog override
7	Main menu key The main menu key shows and hides the main menu on the smartHMI.
8	User keys The function of the user keys is freely programmable. Uses of the user keys include controlling peripheral devices or triggering application-specific actions. (>>> 15.26 "Defining user keys" Page 358)
9	Start key The Start key is used to start a program. The Start key is also used to manually address frames and to move the robot back onto the path. (>>> 6.14 "Program execution" Page 85) (>>> 6.13.4 "Manually addressing frames" Page 84) (>>> 6.14.6 "Repositioning the robot after leaving the path" Page 88)
10	Start backwards key Currently without function
11	STOP key The STOP key is used to stop a running application.
12	Keyboard key Currently without function

The following applies to the jog keys, the user keys and the Start, Start backwards and STOP keys:

- The current function is displayed next to the key on the smartHMI.
- If there is no display, the key is currently without function.

6.1.2 Rear view

Overview

Fig. 6-2: KUKA smartPAD, rear view

1	Enabling switch	4	USB connection
2	Start key (green)	5	Enabling switch
3	Enabling switch	6	Identification plate

Description

Element	Description
Identification plate	Identification plate
Start key	The Start key is used to start a program. The Start key is also used to manually address frames and to move the robot back onto the path.

Element	Description
Enabling switch	The enabling switch has 3 positions: ■ Not pressed ■ Center position ■ Fully pressed (panic position) The enabling switch must be held in the center position in operating modes T1, T2 and CRR in order to be able to jog the manipulator. By default, the enabling switch has no function in Automatic mode.
USB connection	The USB connection is used e.g. for archiving data. Only for FAT32-formatted USB sticks.

6.2 Switching the robot controller on/off

6.2.1 Switching on the robot controller and starting the system software

Procedure

- Turn the main switch on the robot controller to the “I” position.
  The system software starts automatically.

The robot controller is ready for operation when the status display for the boot state of the robot controller lights up green (Station view / **KUKA_Sunrise_Cabinet** tile).

6.2.2 Switching off the robot controller

Procedure

- Turn the main switch on the robot controller to the “0” position.

NOTICE If an application is still running when the robot controller is switched off, active motions are stopped. This can result in the robot being damaged. For this reason, the robot controller must only be switched off when no more applications are running and the robot is stationary.

6.3 Automatic update of the smartPAD software

When the robot controller is rebooted or the smartPAD is plugged into a running robot controller, the version of the smartPAD software is automatically checked. If there are conflicts between the smartPAD software and the system software on the robot controller, the smartPAD software must be updated.

Characteristics of the smartPAD software update:

- The update is carried out automatically in T1, T2 and CRR modes.
- In Automatic mode, the smartPAD software cannot be updated automatically.
  If the smartPAD is connected in Automatic mode and a version conflict is recognized, no user input may be entered on the smartPAD. In this case, it is necessary to switch to T1 or T2 mode in order for the automatic update to be carried out.
- No user input may be entered during the smartPAD update.

NOTICE The update must not be interrupted as the smartPAD may otherwise be damaged. Care must be taken to ensure that the smartPAD is not disconnected from the robot controller and that the robot controller is not disconnected from the power supply during the update.

- Following the update, the smartPAD is automatically rebooted. The upper area of the smartHMI is overlaid with a bar.

Following the automatic update, the robot controller must be rebooted in order to fully display the smartHMI and use the system.

6.4 KUKA smartHMI user interface

Fig. 6-3: KUKA smartHMI user interface

Item	Description
1	Navigation bar: Main menu and status display (>>> 6.4.1 "Navigation bar" Page 64)
2	Display area Display of the level selected in the navigation bar, here the Station view

Item	Description
3	**Jogging options** button Displays the current coordinate system for jogging with the jog keys. Touching the button opens the **Jogging options** window, in which the reference coordinate system and further parameters for jogging can be set. (>>> 6.8.1 "“Jogging options” window" Page 74)
4	Jog keys display If axis-specific jogging is selected, the axis numbers are displayed here (A1, A2, etc.). If Cartesian jogging is selected, all the directions of the coordinate system (X, Y, Z, A, B, C) as well as the elbow angle (R) for executing a null space motion are displayed here. (>>> 6.8 "Jogging the robot" Page 74)
5	**HOV** button Indicates the current jog override. Touching the button opens the **Jog velocity** window, in which the jog override can be set. (>>> 6.8.2 "Setting the jog override (HOV)" Page 75)
6	Life sign display A steadily flashing life sign indicates that the smartHMI is active.
7	**Language selection** button Touching the button opens the **Language selection** menu, in which the display language of the smartHMI can be changed.
8	**User key selection** button Touching the button opens the **User key selection** window, in which the currently available user key bars can be selected. (>>> 6.15 "Activating the user keys" Page 89)
9	Clock button The clock displays the system time. Touching the button displays the system time in digital format, together with the current date.
10	**Jogging type** button Displays the currently set mode of the Start key. Touching the button opens the **Jogging type** window, in which the mode can be changed. (>>> 6.13.3 "“Jogging type” window" Page 83)
11	Back button Return to the previous view by touching this button.

6.4.1 Navigation bar

The navigation bar is the main menu of the user interface and is divided into 4 levels. It is used for navigating between the different levels.

Some of the levels are divided into two parts:

- Lower selection list: Opens a list for selecting an application, a robot or an I/O group, depending on the level.
- Upper button: If a selection has been made in the list, this button shows the corresponding Applications view, Robots view or I/O group.

Alternatively, the main menu can be called using the main menu key on the smartPAD. The main menu contains further menus which cannot be accessed from the navigation bar.

(>>> 6.5 "Calling the main menu" Page 69)

Overview

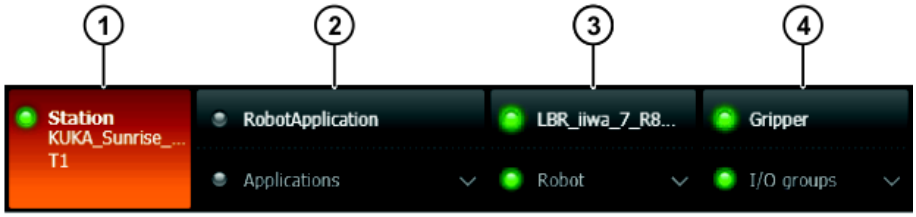


Fig. 6-4: KUKA smartHMI navigation bar

Item	Description
1	**Station** level Displays the controller name and the selected operating mode (>>> 6.4.4 "Station view" Page 66)
2	**Applications** level Displays the selected applications (>>> 6.14.1 "Selecting a robot application" Page 85)
3	**Robot** level Displays the selected robot (>>> 6.4.5 "Robots view" Page 68)
4	**I/O groups** level Displays the selected I/O group. (>>> 6.16.5 "Displaying an I/O group and changing the value of an output" Page 93)

6.4.2 Status display

The status of the system components is indicated by colored circles on the smartHMI.

The “collective status” is displayed in the lower part of the navigation bar (>>> Fig. 6-4). The status of each of the selected components is displayed in the upper part. For example, it is possible for one application to be executed while another application is in the error state.

Status	Description
	Serious error The system component cannot be used. The reason for this may be an operator error or an error in the system component.
	Warning There is a warning for the system component. The operability of the component may be restricted. It is therefore advisable to remedy the problem. For applications, the yellow status indicator means that the application is paused.

Status	Description
	Status OK There are no warnings or faults for the system component.
	Status unknown The status of the system component cannot be determined.

6.4.3 Keypad

There is a keypad on the smartHMI for entering letters and numbers. The smartHMI detects when the entry of letters or numbers is required and automatically displays the appropriate keypad.

Fig. 6-5: Example of keypad

> **SYM** must be pressed to activate the secondary characters assigned to the keys, e.g. the “=” character on the “S” key. The key remains activated for one keystroke. In other words, it does not need to be held down.

6.4.4 Station view

The station view gives access to information and functionalities which affect the entire station.

Fig. 6-6: Station view

Item	Description
1	**Process data** tile Information regarding the process data is displayed here. The configuration of process data is currently not possible.
2	**Safety** tile Opens the **Safety** view and displays the safety status of the station. (>>> 13.7 "Activating the safety configuration on the robot controller" Page 189)
3	**Frames** tile Opens the Frames view. The view contains the frames from the Sunrise project created for the station. (>>> 6.13 "Teaching and manually addressing frames" Page 80)
4	**KUKA_Sunrise_Cabinet** tile Displays the state of the robot controller and opens a view containing the **Boot state** and **Field buses** tiles. The **Boot state** tile displays the boot state of the robot controller. The **Field buses** tile displays the state of the field buses. It is only displayed if I/O groups have been created for the Sunrise project and corresponding signals have been mapped with WorkVisual.
5	**HMI state** tile Displays the connection status between the smartHMI and the robot controller.

Item	Description
6	**Information** tile Displays system information, e.g. the IP address of the robot controller.
7	**Protocol** tile Opens the **Protocol** view and displays the logged events and changes in state of the system. The display can be filtered based on various criteria. (>>> 18.2 "Displaying the protocol" Page 425)

6.4.5 Robots view

The Robots view gives access to information and functionalities which affect the selected robot.

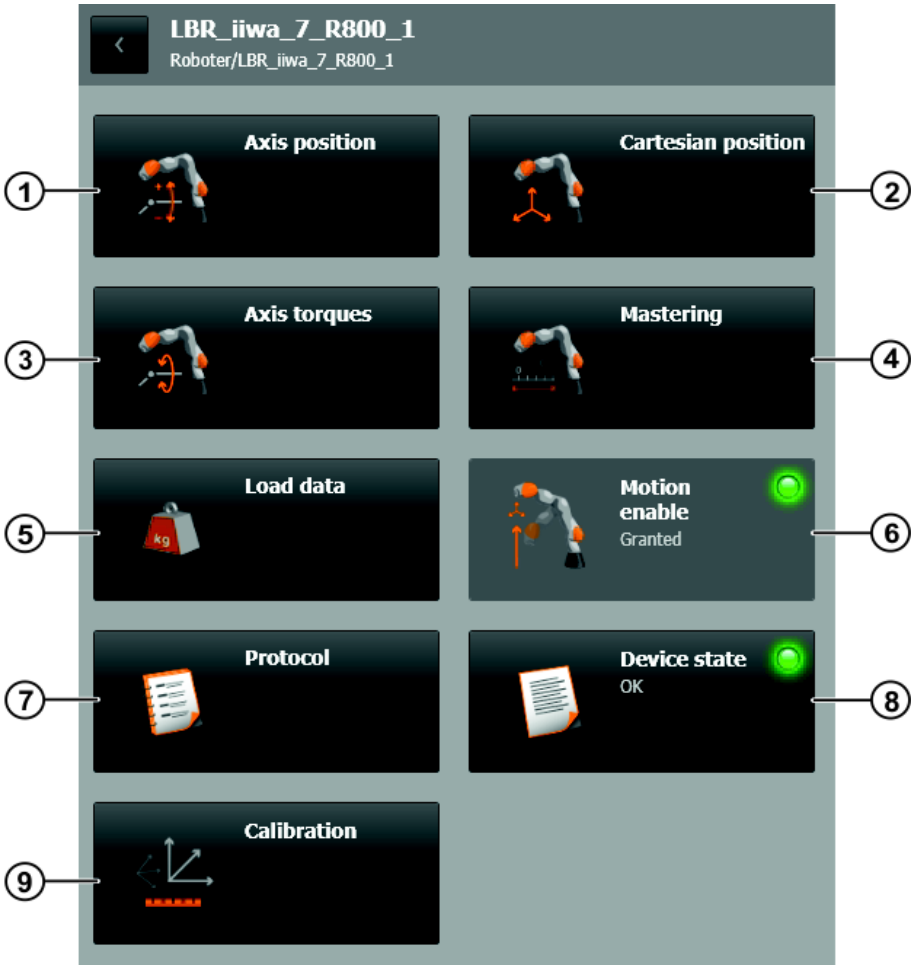


Fig. 6-7: Robots view

Item	Description
1	**Axis position** tile Opens the **Axis position** view. The axis-specific actual position of the robot is displayed. (>>> 6.16.2 "Displaying the axis-specific actual position" Page 91)
2	**Cartesian position** tile Opens the **Cartesian position** view. The Cartesian actual position of the robot is displayed. (>>> 6.16.3 "Displaying the Cartesian actual position" Page 92)
3	**Axis torques** tile Opens the **Axis torques** view. The axis torques of the robot are displayed. (>>> 6.16.4 "Displaying axis-specific torques" Page 92)
4	**Mastering** tile Opens the **Mastering** view. The mastering status of the robot axes is displayed. The axes can be mastered or unmastered individually. (>>> 7.1 "Position mastering" Page 97)
5	**Load data** tile Opens the **Load data** view for automatic load data determination. (>>> 7.3 "Determining tool load data" Page 105)
6	**Motion enable** tile Displays whether the robot has received the motion enable.
7	**Protocol** tile Opens the **Protocol** view and displays the logged events and changes in state of the system. The display can be filtered based on various criteria. By default, the **Source(s)** filter is already set on the robot in question. (>>> 18.2 "Displaying the protocol" Page 425)
8	**Device state** tile The status of the robot drive system is displayed.
9	**Calibration** tile Opens the **Calibration** view which contains the **Base calibration** and **Tool calibration** tiles. (>>> 7.2 "Calibration" Page 98)

6.5 Calling the main menu

Procedure

- Press the main menu key on the smartPAD. The **Main menu** view opens.

Description

Properties of the **Main menu** view:

- The main menu is displayed in the left-hand column.
  The first 4 buttons are identical to the levels in the navigation bar.
- Touching a button that contains an arrow opens the relevant areas for the level, e.g. **Station**.
  Further navigation options are described in the following table.

Fig. 6-8: Example view of the main menu

Item	Description
1	Back button Touch this button to return to the view which was visible before the main menu was opened.
2	Home button Closes all opened areas.
3	Button for closing the level Closes the lowest opened level.
4	The views most recently opened from the main menu are displayed here (maximum 3). By touching the view in question, it is possible to switch to these views again without having to navigate the main menu.

6.6 Changing the operating mode

Description

The operating mode can be set with the smartPAD using the connection manager.

It is possible to change the operating mode while an application is running on the robot controller. The industrial robot then stops with a safety stop 1 and the application is paused. Once the new operating mode has been set, the application can resume.

Precondition

- The key is in the switch for calling the connection manager

Procedure

1. On the smartPAD, turn the switch for the connection manager to the right. The connection manager is displayed.
2. Select the operating mode.
3. Turn the switch for the connection manager to the left.

   The selected operating mode is now active and is displayed in the navigation bar of the smartHMI.

Operating mode	Use	Velocities
T1	Programming, teaching and testing of programs.	■ Program verification: Reduced programmed velocity, maximum 250 mm/s ■ Jog mode: Jog velocity, maximum 250 mm/s ■ Manual guidance: No limitation of the velocity, but safety-oriented velocity monitoring in accordance with the safety configuration **Note:** The maximum velocity of 250 mm/s does not apply to a mobile platform.
T2	Testing of programs Only possible with safety gate closed	■ Program verification: Programmed velocity ■ Jog mode: Not possible

Operating mode	Use	Velocities
AUT	Automatic execution of programs For industrial robots with and without higher-level controllers	■ Program mode: Programmed velocity ■ Jog mode: Not possible
CRR	■ Motion taking the industrial robot out of a violated Cartesian or axis-specific range ■ Motion taking the industrial robot out of a violated range of the tool orientation ■ Retraction of the industrial robot when it has become jammed due to a violation of force or torque limits ■ Motion of the industrial robot if mastering is lost for at least one position sensor during active Cartesian velocity monitoring CRR is an operating mode which is available when the industrial robot is stopped by the safety controller for one of the following reasons: ■ Industrial robot violates a safely monitored space. ■ Orientation of the safety-oriented tool is outside the safely monitored range. ■ Industrial robot violates a safely monitored force or torque limit. ■ A position sensor is not mastered during active Cartesian velocity monitoring.	■ Program mode: Reduced programmed velocity, maximum 250 mm/s ■ Jog mode: Jog velocity, maximum 250 mm/s

6.7 Coordinate systems

Coordinate systems or frames determine the position and orientation of an object in space.

Overview

The following coordinate systems are relevant for the robot controller:

- World
- Robot base
- Base
- Flange
- Tool

Description

World coordinate system

The world coordinate system is a permanently defined Cartesian coordinate system. It is the original coordinate system for all other coordinate systems, in particular for base coordinate systems and the robot base coordinate system.

By default, the world coordinate system is located at the robot base.

Robot base coordinate system

The robot base coordinate system is a Cartesian coordinate system, which is always located at the robot base. It defines the position of the robot relative to the world coordinate system.

By default, the robot base coordinate system is identical to the world coordinate system. It is possible to define an offset of the robot relative to the world coordinate system by changing the mounting orientation when creating the Sunrise project. By default, the direction of installation of the floor-mounted robot is set (A=0°, B=0°, C=0°).

(>>> 5.3 "Creating a Sunrise project with a template" Page 48)

Base coordinate system

In order to define motions in Cartesian space, a reference coordinate system (base) must be specified.

By default, the world coordinate system is used as the base coordinate system for a motion. Additional base coordinate systems can be defined relative to the world coordinate system.

(>>> 7.2.2 "Calibrating the base: 3-point method " Page 103)

Flange coordinate system

The flange coordinate system describes the current position and orientation of the robot flange center point. It does not have a fixed location and is moved with the robot.

The flange coordinate system is used as an origin for coordinate systems which describe tools mounted on the flange.

Tool coordinate system

The tool coordinate system is a Cartesian coordinate system which is located at the working point of the mounted tool. This is called the TCP (Tool Center Point).

Any number of frames can be defined for a tool and can be selected as the TCP. The origin of the tool coordinate system is generally identical to the flange coordinate system.

(>>> 9.3.1 "Geometric structure of tools" Page 132)

The tool coordinate system is offset to the tool center point by the user.

(>>> 7.2.1 "Tool calibration" Page 98)

Position and orientation

In order to determine the position and orientation of an object, translation and rotation relative to a reference coordinate system are specified. 6 coordinates are used for this purpose.

Translation

Coordinate	Description
Distance X	Translation along the X axis of the reference system
Distance Y	Translation along the Y axis of the reference system
Distance Z	Translation along the Z axis of the reference system

Rotation

Coordinate	Description
Angle A	Rotation about the Z axis of the reference system
Angle B	Rotation about the Y axis of the reference system
Angle C	Rotation about the X axis of the reference system

6.8 Jogging the robot

Overview

There are 2 ways of jogging the robot:

- Cartesian jogging
  The set TCP is jogged in the positive or negative direction along the axes of a coordinate system or rotated about these axes.
- Axis-specific jogging
  Each axis can be moved individually in the positive or negative direction.

6.8.1 “Jogging options” window

Procedure

Open the **Jogging options** window:

- Touch the **Jogging options** button.

Close the **Jogging options** window.

- Touch the **Jogging options** button or an area outside the window.

Description

All parameters for jogging the robot can be set in the **Jogging Options** window.

Fig. 6-9: “Jogging options” window

Item	Description
1	**Jogging options** button The icon displayed depends on the programmed jogging type.
2	Select the jogging type. Axis-specific jogging or Cartesian jogging in different coordinate systems is possible. The selected jogging type is indicated in green and displayed on the **Jogging options** button. ■ **Axes**: The robot is moved by axis-specific jogging. ■ **World**: The selected TCP is moved in the world coordinate system by means of Cartesian jogging. ■ **Base**: The selected TCP is moved in the base coordinate system by means of Cartesian jogging. ■ **Tool**: The selected TCP is moved in its own tool coordinate system by means of Cartesian jogging.

Item	Description
3	Select the robot flange or mounted tool. Not possible while an application is being executed. The frames of the selected tool can be selected as the TCP for Cartesian jogging. The set load data of the tool are taken into consideration. If a robot application is paused, the tool currently being used in the application is available under the name **Application tool**. (>>> "Application tool" Page 75)
4	Select the TCP. All the frames of the selected tool are available as the TCP. The TCP which was set manually here is retained. This is also the case if an application which has used a different TCP is paused. **Exceptions:** ■ The application tool is selected in the jogging options. In this case, if the tool used is changed in the application and a motion command is then executed, the selected application tool is also adapted in the jogging options. The TCP set in the jogging options is then automatically the frame of the application tool used to execute the last commanded motion. ■ The active program TCP of the application tool is selected in the jogging options. In this case, the TCP for jogging changes depending on the TCP currently being used in the application.
5	Base selection. Only possible when the jogging type **Base** is selected. All frames which were designated in Sunrise.Workbench as a base are available as a base.

Application tool

The application tool consists of all the frames located below the robot flange during the runtime. These can be frames of a tool or workpiece attached to the robot flange with the attachTo command as well as frames created in the application and linked directly or indirectly to the flange.

The application tool is then only available in the jogging options when a robot application is paused, and if a motion command was sent to the robot controller prior to pausing.

If the application tool is selected, the frame with which the current motion command is executed is automatically set as the TCP in the jogging options. All other frames located hierarchically under the flange coordinate system during the runtime can also be selected as the TCP for jogging.

The robot flange frame is available under the name **ApplicationTool(Root)** for selection as the TCP for jogging.

6.8.2 Setting the jog override (HOV)

Description

The jog override determines the velocity of the robot during jogging. The velocity actually achieved by the robot with a jog override setting of 100% depends on various factors, including the robot type. However, the velocity of the set working point cannot exceed 250 mm/s.

Procedure

1. Touch the **HOV** button. The **Jog velocity** window opens.
2. Set the desired jog override. It can be set using either the plus/minus keys or by means of the slider.

- Plus/minus keys: The override can be set in steps to the following values: 100%, 75%, 50%, 30%, 10%, 5%, 3%, 1%, 0%.
- Slider: The override can be adjusted in 1% steps.

3. Touch the **HOV** button or an area outside the window to close the window.

Alternative procedure

Alternatively, the override can be set using the plus/minus key on the right of the smartPAD.

The value can be set in the following steps: 100%, 75%, 50%, 30%, 10%, 5%, 3%, 1%.

6.8.3 Axis-specific jogging with the jog keys

Precondition

- Operating mode T1

Procedure

1. Select the jogging type **Axes** from the jogging options.
   Axes A1 to A7 are displayed next to the jog keys.
2. Set the jog override.
3. Hold down the enabling switch.
   When motion is enabled, the display elements next to the jog keys are highlighted in white.
4. Press the plus or minus jog key to move an axis in the positive or negative direction.

Description

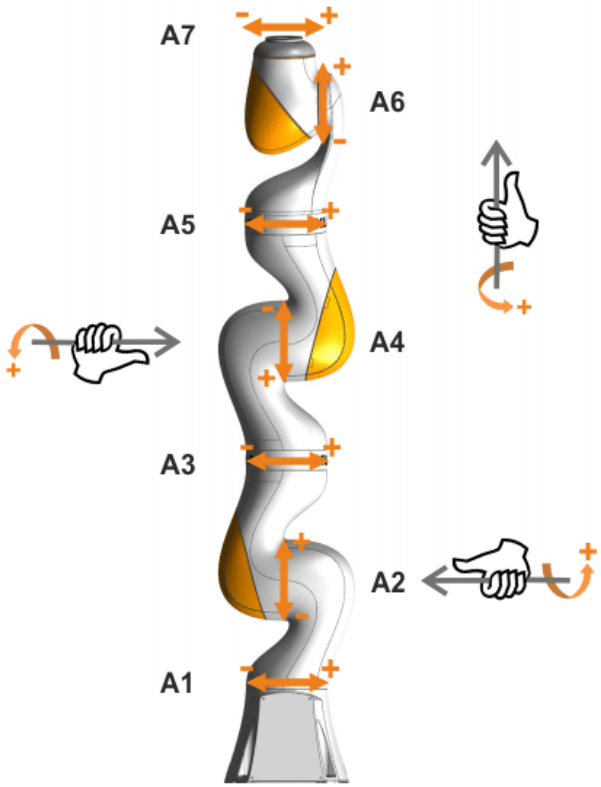


Fig. 6-10: Axis-specific jogging

The positive direction of rotation of the robot axes can be determined using the right-hand rule. Imagine the cable bundle which runs inside the robot from the base to the flange. Mentally close the fingers of your right hand around the cable bundle at the axis in question. Keep your thumb exended while doing so. Your thumb is now positioned on the cable bundle so that it points in the same direction as the cable bundle runs inside the axis on its way to the flange. The other fingers of your right hand point in the positive direction of rotation of the robot axis.

6.8.4 Cartesian jogging with the jog keys

Precondition

- Operating mode T1

Procedure

1. Select the coordinate system for Cartesian jogging. **World**, **Base** and **Tool** are available.

   The following designations are displayed next to the jog keys:

   - **X**, **Y**, **Z**: for the linear motions along the axes of the selected coordinate system
   - **A**, **B**, **C**: for the rotational motions about the axes of the selected coordinate system
   - **R**: for the null space motion
2. Select the desired tool and TCP.
3. If the **Base** coordinate system for Cartesian jogging is selected, select the desired base frame.

   > The frames which are to be available as a base for jogging must first be designated as a base in Sunrise.Workbench.
   > (>>> 9.2.2 "Designating a frame as a base" Page 128)

4. Set the jog override.
5. Press and hold down the enabling switch.

   When motion is enabled, the display elements next to the jog keys are highlighted in white.
6. Press the plus or minus jog key to move the robot in the positive or negative direction.

6.8.4.1 Null space motion

Description

The lightweight robot has 7 axes, making it kinematically redundant. This means that theoretically, it can move to every point in the work envelope with an infinite number of axis configurations.

Due to the kinematic redundancy, a so-called null space motion can be carried out during Cartesian jogging. In the null space motion, the axes are rotated in such a way that the position and orientation of the set TCP are retained during the motion.

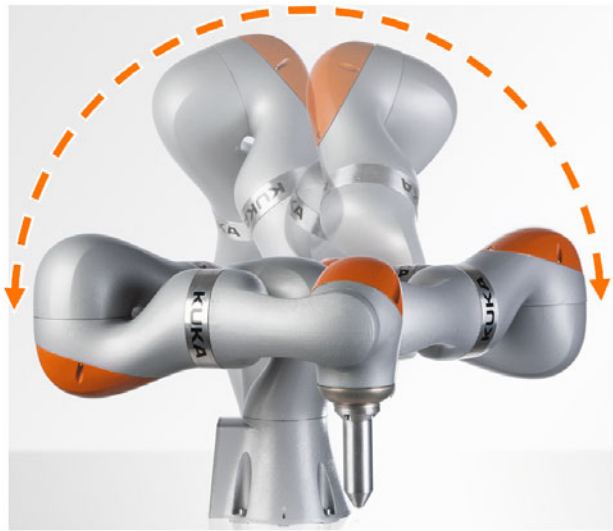


Fig. 6-11: Null space motion

Characteristics

- The null space motion is carried out via the “elbow” of the robot arm.

- The position of the elbow is defined by the elbow angle (R).
- The position of the elbow angle (R) can be modified using the jog keys during Cartesian jogging.

Areas of application

- The optimal axis configuration can be set for a given position and orientation of the TCP. This is especially useful in a limited working space.
- When a software limit switch is reached, you can attempt to move the robot out of the range of the limit switches by changing the elbow angle.

6.9 CRR mode – controlled robot retraction

Description

CRR is an operating mode which is available when the robot was stopped by the safety controller and the triggering line in the safety configuration contains one of the following monitoring functions:

- Axis range monitoring
- Cartesian velocity monitoring
- Cartesian workspace monitoring
- Cartesian protected space monitoring
- Tool orientation
- Axis torque monitoring
- Collision detection
- TCP force monitoring

The robot can be moved in CRR mode if it is stopped by the safety controller for one of the following reasons:

- Robot violates a safely monitored space.
- Orientation of the safety-oriented tool is outside the safely monitored range.
- Robot violates a safely monitored force or torque limit.
- A position sensor is not mastered during active Cartesian velocity monitoring.

The motion velocity of the set working point in CRR mode corresponds to the jog velocity in T1 mode, 250 mm/s maximum.

In CRR mode, the robot can be moved, irrespective of what monitoring functions are active. No stop is triggered if it passes through other monitoring limits. The velocity monitoring functions remain active in CRR mode.

Procedure

1. Switch to CRR mode.
2. Jog the robot and move to a position in which the monitoring function that triggered the stop is no longer violated.

   If the monitored parameters are again outside the violated range and remain in a permissible range after 4 seconds, the operating mode will automatically change to T1.

6.10 Manually guiding the robot

⚠ CAUTION The operator must ensure that the smartPAD cannot be operated by any other person unnoticed during manual guidance. Failure to observe this precaution may result in injuries.

6.11 Resuming the safety controller

Description

If there are connection or periphery errors, the safety controller is paused (after one or more occurrences depending on the error). Pausing the safety controller causes the robot to stop and all safe outputs to be switched off. The application can resume once the error has been eliminated.

Procedure

1. Select **Safety** > **State** in the Station view. The **State** view opens.

   The cause of the error is displayed in the view. The **Resume safety controller** button is not active.
2. Eliminate the error. The **Resume safety controller** button is now activated.
3. Press **Resume safety controller**. The safety controller is resumed.

6.12 Opening the holding brakes

Description

If the robot becomes jammed because the maximum torque of the joint torque sensor has been exceeded on at least one axis, the safety controller is paused. The robot can no longer be moved.

In this case, it is possible to reduce the torque on the affected axes by briefly opening the holding brakes.

> **⚠ CAUTION** Once the maximum torque has been exceeded, the referencing of the position and torque sensors of the affected axes is discarded. To ensure the safety integrity of the monitoring functions dependent on position and axis torque, the position and torque sensors of the affected axes must be referenced again once the jamming has been rectified.

Maximum torque

The maximum torque lies between the rated and limit torques of the sensor. The sensor may be loaded up to the limit torque without this causing lasting damage. In order to allow for increased measurement errors beyond the rated torque, the safety controller is paused when the maximum torque is reached.

As long as the value determined by the sensor does not exceed the maximum torque, it is ensured that the limit torque is not exceeded.

Precondition

- The maximum torque of the joint torque sensor is exceeded on at least one axis.

Procedure

1. Select **Safety** > **State** in the Station view. The **State** view opens.

   An error message is displayed in the view indicating that the maximum torque has been exceeded.
2. Press and hold down the enabling switch. The holding brakes of the affected axes are opened briefly.

   > The robot moves slightly when the brakes are opened.

3. Release the enabling switch once the brakes have closed again.
4. If the maximum torque of the joint torque sensor is still exceeded on at least one axis: Repeat steps 1 and 2 until the torque on all axes lies below the maximum torque.
5. Press **Resume safety controller**. The safety controller is resumed.
6. If necessary, jog the robot to an uncritical position.
7. If safety functions are configured, use the axis torque-based or position-based AMFs: Perform torque and/or position referencing.

(>>> 13.10.2 "Torque referencing" Page 224)
(>>> 13.10.1 "Position referencing" Page 223)

6.13 Teaching and manually addressing frames

6.13.1 Displaying frames

Procedure

- Select **Frames** in the Station view. The Frames view opens.

Description

The view contains the frames created for the Sunrise project. The frames are taught here. The position and orientation of a frame in space and the associated redundancy information are recorded during teaching.

- Taught frames can be addressed manually.
- Taught frames can be used as end points of motions. If an application is run and the end frame of a motion is addressed, this is selected in the Frames view.

  (>>> 6.16.1 "Displaying the end frame of the motion currently being executed" Page 90)

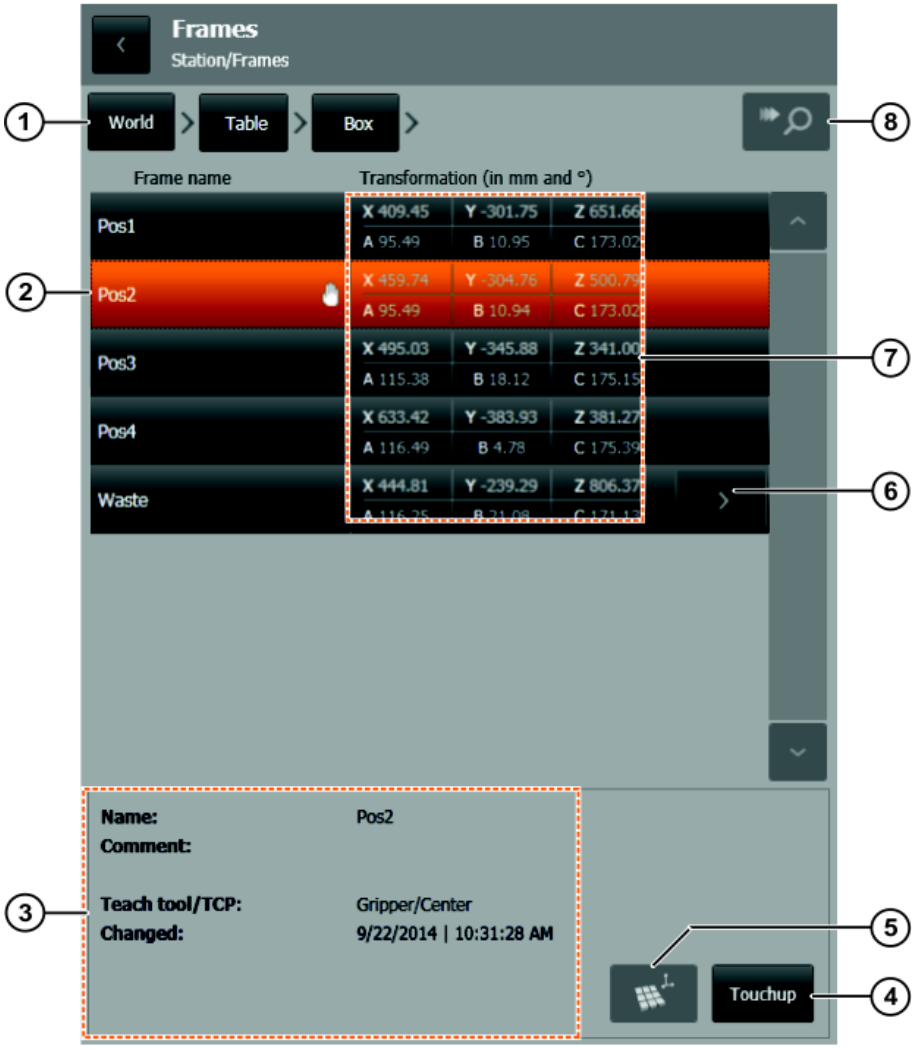


Fig. 6-12: Frames view

Item	Description
1	Frame path Path to the frames of the currently displayed hierarchy level. Goes from **World** to the direct parent frame (**Box** here).
2	Frames of the current hierarchy level A frame can be selected by touching it. The selected frame is marked here with a hand icon. The hand icon means that this frame can be used as the base for jogging and can be calibrated.
3	Properties of the selected frame ■ Name of the frame ■ Comment ■ Tool used while teaching the frame ■ Date and time of the last modification
4	**Touchup** button A selected frame can be taught. If no frame is selected, the button is disabled.
5	Button for setting the base for jogging The button sets the selected frame as the base for jogging in the jogging options. (>>> 6.8.1 "“Jogging options” window" Page 74) The button is only active if the **Base** jogging type is selected from the jogging options and the selected frame is marked as the base in Sunrise.Workbench.
6	Button for displaying child frames The button is only available if a frame has child elements. The button displays the direct child elements of a frame.
7	Frame coordinates with reference to the parent frame
8	Search button The search button is only active if an application is running and the end frame of a motion is being addressed. Use the button to switch to this end frame if it is not yet displayed.

6.13.2 Teaching frames

Description

The coordinates of a frame can be modified on the smartHMI. This is done by moving to the new position of the frame with the desired TCP and teaching the frame. In the process, the new position and orientation are applied.

> It is advisable to synchronize the project immediately after teaching the frames so that the new frame data will also be updated in the corresponding project in Sunrise.Workbench.

Precondition

- The tool with the desired TCP is set in the jogging options.
  (>>> 6.8.1 "“Jogging options” window" Page 74)

> When teaching frames, it is not advisable to use the application tool because the application tool is only available in the jogging options during the pauses in the application. Instead, it is possible to use the tool which was created in the object templates of the project and which corresponds to the current application tool.

- Operating mode T1

Procedure

1. Move the TCP to the desired position of the frame.
2. In the Frames view, select the frame whose position is to be taught.
3. Press **Touchup** to apply the current TCP coordinates to the selected frame.
4. The coordinates and redundancy information of the taught point are displayed in the **Apply touchup data** dialog. Press **Apply** to save the new values.

⚠ WARNING If a frame is changed, the change affects all applications in which the frame is used. Modified programs must always be tested first in Manual Reduced Velocity mode (T1).

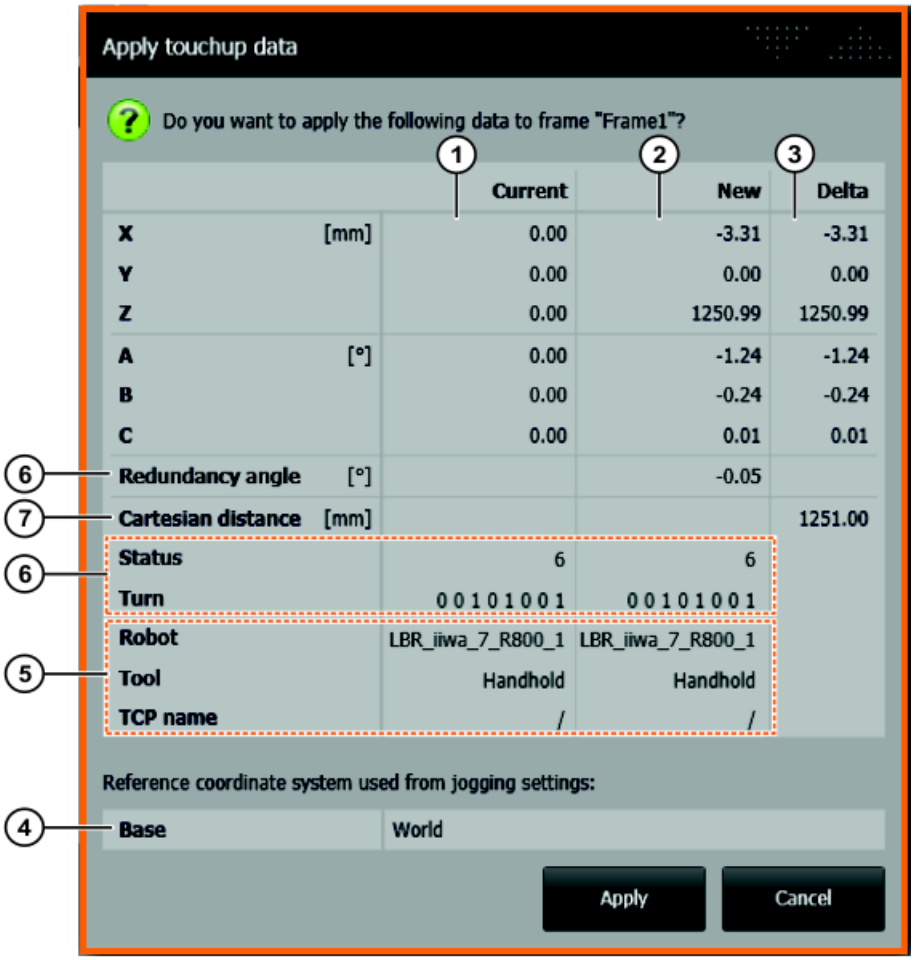


Fig. 6-13: Apply touchup data

Item	Description
1	Values saved up to now
2	New values
3	Changes between the values saved until now and new values
4	Base for jogging All coordinate values of the frame which are displayed in the dialog refer to the jogging base set in the jogging options. These values generally differ from the coordinate values of the frame with respect to its parent frame: (>>> 6.8.1 "“Jogging options” window" Page 74)

Item	Description
5	Information on the robot and tool used during teaching (>>> 9.2.5 "Displaying and modifying the properties of a frame" Page 130)
6	Redundancy informationon on the taught point (>>> 9.2.5 "Displaying and modifying the properties of a frame" Page 130)
7	Cartesian distance between the current and new position of the frame

6.13.3 “Jogging type” window

Procedure

Open the **Jogging type** window:

- Touch the **Jogging type** button next to the Start key.

Close the **Jogging type** window.

- Touch the **Jogging type** button or an area outside the window.

Description

The functionality of the Start key can be configured in the **Jogging type** window.

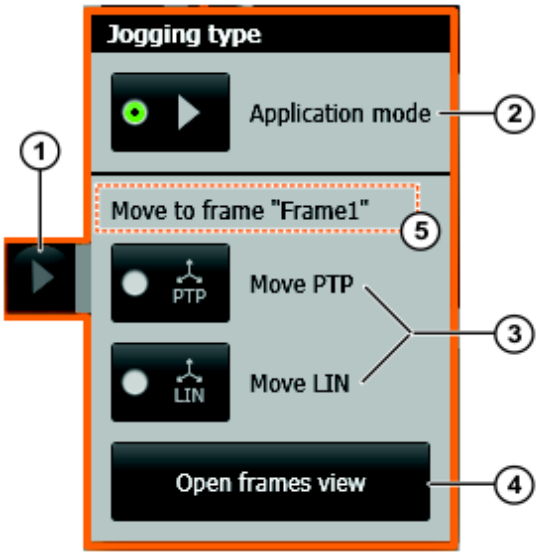


Fig. 6-14: “Jogging type” window

Item	Description
1	**Jogging type** button The icon displayed depends on the programmed jogging type.
2	**Application mode** jogging type In this jogging type, the Start key is used to start applications. **Note**: When switching to T2 or Automatic mode, **Application mode** is set automatically.

Item	Description
3	**Move PTP** / **Move LIN** jogging types In these jogging types, the Start key is used to manually address frames with a PTP or LIN motion. These can only be selected if a frame is selected in the frames view and if T1 mode is set. **Note**: In the **Move PTP** jogging type, the Status of the end frame is taken into consideration. This can cause the axes to move, even if the end point has already been reached in Cartesian form. **Note**: In the **Move LIN** jogging type, the Status of the end frame is not taken into consideration.
4	**Open frames view** button. Press the button to switch to the frames view.
5	Frame display The name of the frame currently selected in the frames view is displayed here.

Icons

The following icons are displayed on the **Jogging type** button depending on the jogging type set:

Icon	Description
	Application mode jogging type
	Move PTP jogging type
	Move LIN jogging type

6.13.4 Manually addressing frames

Description

Taught frames can be manually addressed with a PTP or LIN motion. In a PTP motion, the frame is approached by the quickest route, whereas in a LIN motion it is approached on a predictable path.

When a frame is being addressed, a warning message is displayed in the following cases:

- The selected tool does not correspond to the tool with which the frame was taught.
- The selected TCP does not correspond to the TCP with which the frame was taught.
- The transformation of the TCP frame has been modified.

If the frame can still be reached, it is possible to move to it.

Precondition

- The frame can be addressed with the selected TCP.
- T1 mode

Procedure

1. Select the desired frame in the frames view.
2. Select the desired jogging type in the **Jogging type** window.
3. Press and hold down the enabling switch.
4. Press the Start key and hold it down until the frame is reached.

If the selected working point is already at the end position or if the frame cannot be reached with the current settings, the robot will not execute any motion.

6.14 Program execution

6.14.1 Selecting a robot application

Procedure

- In the navigation bar, select the desired application under **Applications**. The Applications view opens and the application goes into the **Selected** state.

Description

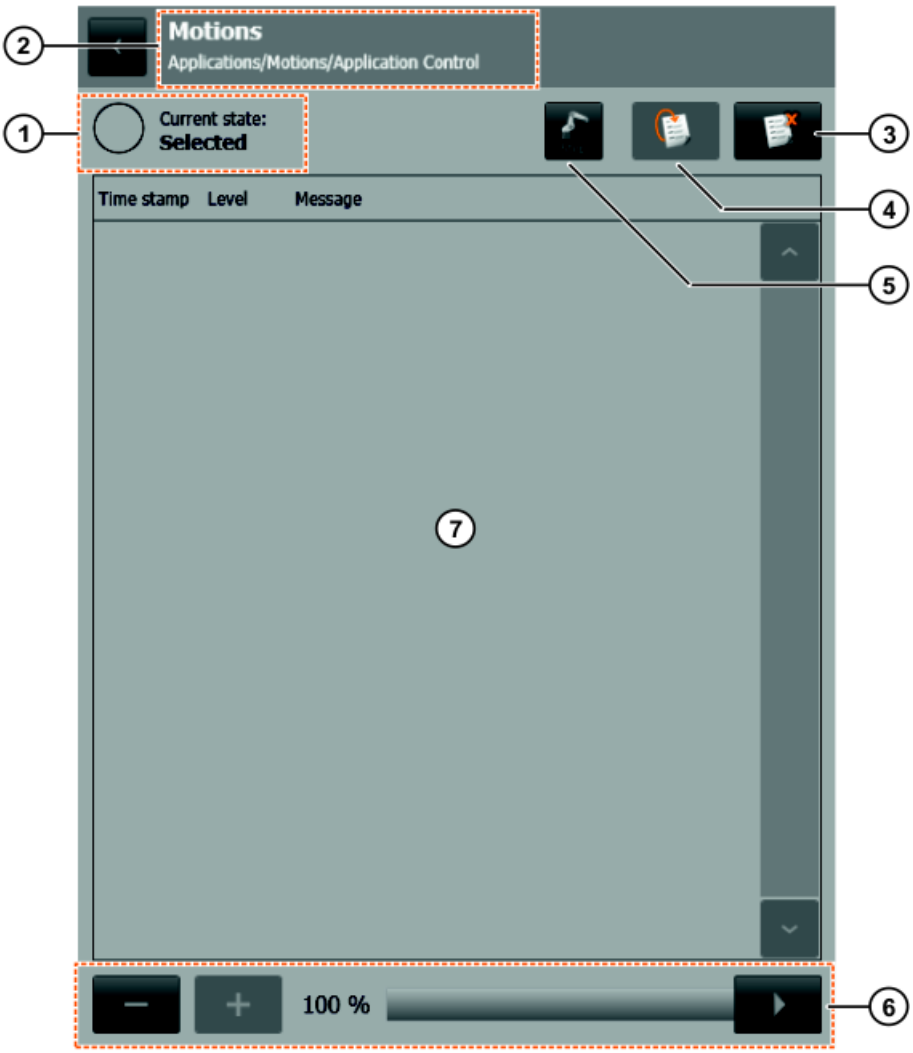


Fig. 6-15: Applications view – application selected

Item	Description
1	Current status of the application The status is displayed as text and as an icon. (>>> "Application status display" Page 86)
2	Application display The name of the selected application is displayed, here **Motions**.

Item	Description
3	**Deselect** button Deselects the selected application and closes the Applications view. A paused application is reset before it is deselected. The button is only active if the application is in the **Selected**, **Motion paused** or **Error** state.
4	**Reset** button Resets a paused application. “Reset” means that the application is reset to the start of the program and goes into the **Selected** state. The button is only active when the application is paused.
5	**Step** button Pressing the button switches between Step mode and standard mode. (>>> 6.14.2 "Selecting the program run mode" Page 87)
6	Manual override The manual override can be set using either the plus/minus keys or by means of the slider.
7	Message window Error messages and user messages programmed in the application are displayed.

Application status display

Icon	State	Description
	Selected	The application is selected.
	Start	The application is initialized.
	Running	The application is executed.
	Motion paused	The application is paused. If the application is paused using the smartPAD, for example by pressing the STOP key, only motion execution is stopped. Other commands, e.g. switching of outputs, are executed in the **Motion paused** state until a synchronous motion command is reached.
	Error	An error occurred while the application was running.
	Repositioning	The robot is repositioned. The application is paused because the robot has left the path.
	Stopping	The application is reset to the start of the program and goes into the **Selected** state.

Start key

The following functions are available in application mode using the Start key:

Icon	Description
	Start application. A selected application can be started or a paused application can be continued.
	Reposition robot. If the robot has left the path, it must be repositioned in order to continue the application.

If an application is paused, the robot can be jogged. The tool used in a paused application and the current TCP are not automatically set as the tool and TCP for Cartesian jogging.
(>>> 6.8.1 "“Jogging options” window" Page 74)

STOP key

The following function is available using the STOP key:

Icon	Description
	Pause application. A running application can be paused in Automatic mode.

6.14.2 Selecting the program run mode

Precondition

- The application is selected.
- T1 or T2 mode

Procedure

- Select the program run mode using the **Step** button.

Description

Program run mode	Description
Continuous	Standard mode The program is executed through to the end without stopping.
Step	Step mode The program is executed with a stop after each motion command. The Start key must be pressed again for each motion command. ■ The end point of an approximated motion is not approximated but rather addressed with exact positioning. Exception: Approximated motions which were sent to the robot controller asynchronously before Step mode was activated and which are waiting there to be executed will stop at the approximate positioning point. For these motions, the approximate positioning arc will be executed when the program is resumed. ■ In a spline motion, the entire spline block is executed as one motion and then stopped. ■ In a MotionBatch, the entire batch is not executed but rather exact positioning is carried out after each individual motion of the batch.

> The program run mode can also be set and polled in the source code of the application. (>>> 15.18 "Changing and polling the program run mode" Page 319)

6.14.3 Setting the manual override

Description

The manual override determines the velocity of the robot during program execution.

The manual override is not necessarily identical to the effective override with which the robot actually moves. If an application override is programmed which is set by the application, the effective override results as follows:

Effective override = manual override application override

(>>> 15.19 "Changing and polling the override" Page 320)

The override is specified as a percentage of the programmed velocity. In T1 mode, the maximum velocity is 250 mm/s, irrespective of the override that is set.

Precondition

- The application is open.

Procedure

- Set the desired manual override. It can be set using either the plus/minus keys or by means of the slider.
  - Plus/minus keys: The override can be set in steps to the following values: 100%, 75%, 50%, 30%, 10%, 5%, 3%, 1%, 0%.
  - Slider: The override can be adjusted in 1% steps.

6.14.4 Starting a program forwards (manually)

Precondition

- The application is selected.
- T1 or T2 mode

Procedure

1. Select the program run mode.
2. Press and hold down the enabling switch.
3. Press Start key and hold it down. The application is executed.

To pause a program that has been started manually, release the Start key. If the application is paused, it can be reset by pressing the **Reset** button.

6.14.5 Starting a program forwards (automatically)

Precondition

- The application is selected.
- Automatic mode
- The project is not controlled externally.

Procedure

- Press the Start key. The application is executed.

To pause a program that has been started in Automatic mode, press the Start key again. If the application is paused, it can be reset by pressing the **Reset** button.

6.14.6 Repositioning the robot after leaving the path

Description

The following events can cause the robot to leave its planned path:

- Triggering of a non-path-maintaining stop
- Jogging during a paused applicaton

The robot can be repositioned using the Start key. Repositioning means that the robot is returned to the Cartesian position at which it left the path. The application can then be resumed from there.

Characteristics of the motion which is used to return to the path:

- A PTP motion is executed.

  The path used to return to the path is different than that taken when leaving the path.
- The robot is moved at 20% of the maximum possible axis velocity and the effective override (Effective override = manual override application override).

> The currently set jog override is irrelevant for repositioning.

- It is moved with the load data which were set when the application was interrupted.
- It is moved with the controller mode which was set when the application was interrupted.

  Additional forces or force modulations overlaid by an impedance controller are withdrawn during repositioning.

> **NOTICE** Repositioning a robot under impedance control may result in unexpected robot motions. The robot is always repositioned to the command position; this means that, in the case of a robot under impedance control, the actual position after repositioning does not necessarily corresond to the actual position at which it left the path. This can lead to unexpectedly high forces in contact situations.
> Prior to repositioning, manually move a robot under impedance control to a position as close as possible to the one at which it left the path. Failure to observe this precaution may result in damage to property.

> **NOTICE** Repositioning may only be carried out if there is no risk of a collision while it is returning to the path. If this is not assured, first move the robot into a suitable position from which it can be safely repositioned.

Procedure

1. In T1 or T2 mode: press and hold down the enabling switch.
2. Press Start key and hold it down. The robot returns to the path.

6.15 Activating the user keys

Description

The user keys on the smartPAD can be assigned functions. All the user key functions defined in a running robot application or a background task are available to the operator. In order to be able to use the desired functions, the operator must activate the corresponding user key bar.

Procedure

1. Touch the **User key selection** button.

   The **User key selection** window opens. The user key bars currently available are displayed.
2. Select the desired user key bar by pressing the corresponding name.

   The text or image on the smartHMI next to the user keys changes according to the bar selected. The user keys now have the corresponding functions.
3. Touch the **User key selection** button or an area outside the window.

   The **User key selection** window closes.

Example

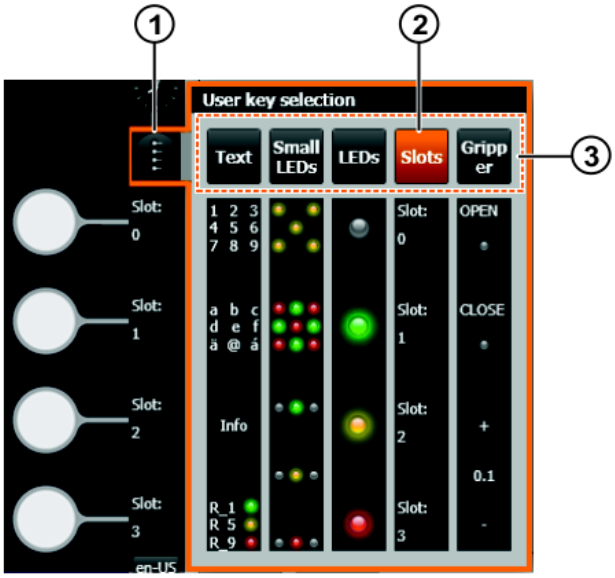


Fig. 6-16: “User key selection” window

1 **User key selection** button
2 Currently active user key bar
3 Names of the available user key bars

6.16 Display functions

6.16.1 Displaying the end frame of the motion currently being executed

Description

If a frame from the frame tree is addressed in an application, this is indicated in the frames view. If the end frame of the motion currently being executed is located at the displayed hierarchy level, the frame name is marked with an arrow icon (3 arrowheads):

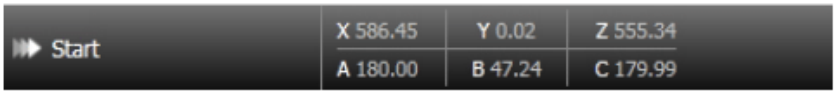


Fig. 6-17: The arrow icon marks the current end frame

If the end frame is located hierarchically below a displayed frame, the button for displaying child frames is marked with an additional arrow icon (3 arrowheads):

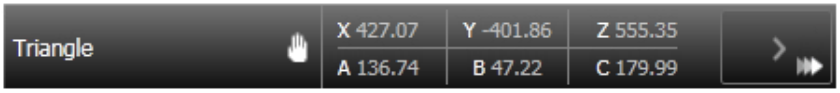


Fig. 6-18: The button switches to the current end frame

You can switch directly to the current end frame using the search button in the upper right-hand area of the frames view:

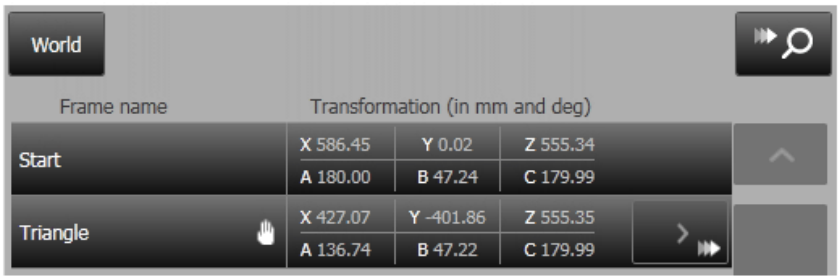


Fig. 6-19: The search button switches directly to the current end frame

The search button is inactive if no frame is being addressed.

Precondition

- The application is selected.
- Application status **Running** or **Motion paused**.
- The motion uses an end frame created in the application data.

Procedure

1. Select **Frames** in the station view. The Frames view opens.
2. Switch to the end frame using the button for displaying child frames or using the search button.

6.16.2 Displaying the axis-specific actual position

Procedure

- Select **Axis position** in the Robots view.

Description

The current position of axes A1 to A7 is displayed. In addition, the range within which each axis can be moved (limitation by end stops) is indicated by a white bar.

The actual position can also be displayed while the robot is moving.

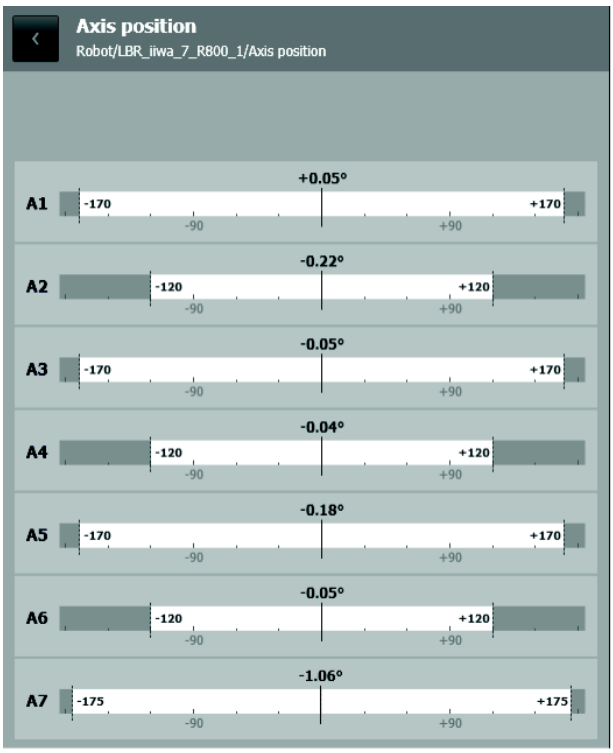


Fig. 6-20: Axis-specific actual position

6.16.3 Displaying the Cartesian actual position

Procedure

1. Select **Cartesian position** in the Robots view.
2. Set the TCP and base in the **Jogging options** window.

Description

The Cartesian actual position of the selected TCP is displayed. The values refer to the base set in the jogging options.

The display contains the following data:

- Current position (X, Y, Z)
- Current orientation (A, B, C)
- Current redundancy information: Status, Turn, redundancy angle (E1)
- Current tool, TCP and base

The actual position can also be displayed while the robot is moving.

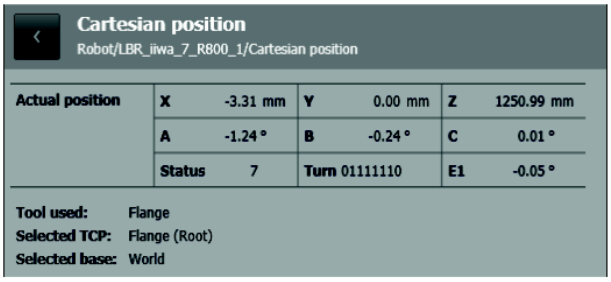


Fig. 6-21: Cartesian actual position

6.16.4 Displaying axis-specific torques

Procedure

- Select **Axis torques** in the Robots view.

Description

The current torque values for axes A1 to A7 are displayed. In addition, the sensor measuring range for each axis is displayed (white bar).

If the maximum permissible torque on a joint is exceeded, the dark gray area of the bar for the axis in question turns orange. Only the violated area is indicated in color, i.e. either the negative or positive part.

> The refresh rate of the displayed values is limited. Briefly occurring peak values are therefore not displayed under certain circumstances.

The display contains the following data:

- Current absolute torques
- Current external torques

> The external torques are only displayed correctly if the correct tool has been specified.

- Current tool

The axis-specific torques can also be displayed while the robot is moving.

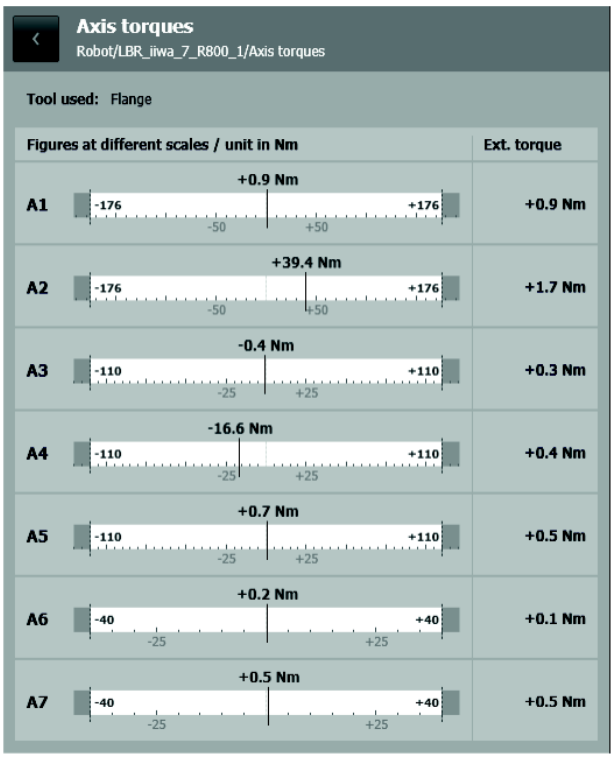


Fig. 6-22: Axis-specific torques

6.16.5 Displaying an I/O group and changing the value of an output

Precondition

- The I/O group has been created for the Sunrise project and corresponding signals have been mapped with WorkVisual.
- To change an output: Operating mode T1, T2 or CRR

> The outputs can be changed irrespective of the safety controller status, for example even if an EMERGENCY STOP is pressed.

Procedure

1. In the navigation bar, select the desired I/O group from **I/O groups**. The inputs/outputs of the selected group are displayed.
2. Select the output to be changed.
3. An input box is displayed for numeric outputs. Enter the desired value.
4. Press and hold down the enabling switch. Change the value of the input with the appropriate button.

Description

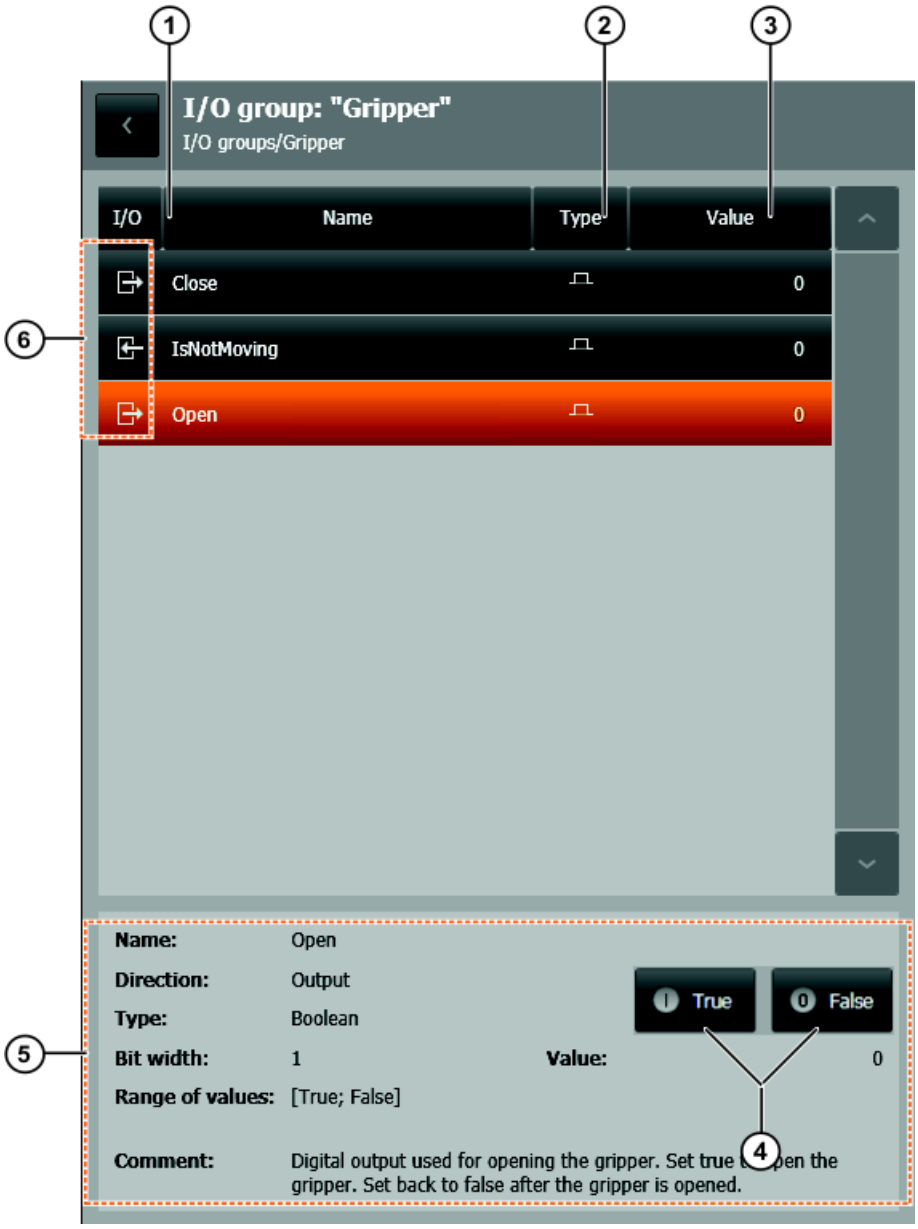


Fig. 6-23: Inputs/outputs of an I/O group

Item	Description
1	Name of the input/output
2	Type of input/output
3	Value of the input/output The value is displayed as a decimal number.
4	Buttons for changing outputs If an output is selected, its value can be changed. Precondition: The enabling switch is pressed. The buttons available depend on the output type.
5	Signal properties The properties and the current value of the selected input or output are displayed.
6	Signal direction The icons indicate whether the signal is an input or an output.

The following buttons are available depending on the type of the selected output:

Button	Description
True	Buttons for changing Boolean outputs
False	Sets the selected Boolean outputs to the value True (1) or False (0).
Set	Button for changing numeric outputs Sets the selected numeric output to the entered value.

Signal direction

The following icons indicate the direction of a signal:

Icon	Description
	Icon for an output
	Icon for an input

I/O types

The following icons indicate the type of input/output:

Icon	Description
	Icon for an analog signal
	Icon for a binary signal
	Icon for a signed digital signal
	Icon for an unsigned digital signal

6.16.6 Displaying the IP address and software version

Procedure

- Select the **Information** tile in the Station view.

The current installed version of the system software and the IP address of the robot controller are displayed under the **Station** node.

6.16.7 Displaying the robot type and serial number

Procedure

- Select the **Information** tile in the Station view.

Information about the robot currently being used is displayed under the **Robot name/type plate** node.

- **Serial number**: Serial number of the connected robot
- **Connected robot**: Robot type of the connected robot
- **Installed robot**: Robot type specified in the the station configuration of the Sunrise project

6.16.8 Displaying messages of the virus scanner

Precondition

- Virus scanner is installed.

(>>> 10.4 "Installing or updating the virus scanner" Page 150)

Procedure

- Select **KUKA_Sunrise_Cabinet** > **Virus scanner** in the Station view. The **Virus scanner** view opens.

> Messages from the virus scanner can also be displayed using the **Protocol** tile.

Description

The **Virus scanner** view contains the following data:

- **Virus scanner state:** active / inactive
- **Version of virus definition file:** version of the virus scanner
- Messages about viruses that have been found: if a virus is found, the message contains the name of the virus, the name of the file in which the virus is located, including the path, and the date and time of the discovery.

  If viruses are found, the status display of the **Virus scanner** tile switches to “Warning”. The status of the files affected by the viruses is automatically set to “Quarantine”.

> If the robot can no longer be moved due to a virus infection, the following options are available to the user:
>
> - Reinstall the system software on the robot controller.
> - If the robot can still not be moved, create the diagnosis package **KRCDiag** and contact KUKA Service.

7 Start-up and recommissioning

7.1 Position mastering

During position mastering, a defined mechanical robot axis position is assigned to a motor angle. Only with a mastered robot is it possible for taught positions to be addressed with high repeatability. An unmastered robot can only be moved manually (axis-specific jogging in T1 or CRR mode).

7.1.1 Mastering axes

Description

The LBR iiwa has a Hall effect mastering sensor in every axis. The mastering position of the axis (zero position) is located in the center of a defined series of magnets. It is automatically detected by the mastering sensor when it passes over the series of magnets during a rotation of the axis.

Before the actual mastering takes place, an automatic search run is performed in order to find a defined premastering position.

If the search run is successful, the axis is moved into the premastering position. The axis is then moved in such a way that the mastering sensor passes over the series of magnets. The motor position at the moment when the mastering position of the axis is detected is saved as the zero position of the motor.

> The repeatability and reproducibility of mastering are only guaranteed if the procedure is always identical. The following rules must be observed during mastering:
> - When one axis is being mastered, all axes should be in the vertical stretch position. If this is not possible, mastering must always be carried out in the same axis position.
> - Always master the individual axes in the same order.
> - Always carry out mastering without a load. Mastering with a load is not currently supported.

Precondition

- Operating mode T1 or CRR

Procedure

1. Select **Mastering** in the Robots view. The **Mastering** view opens.
2. Press and hold down the enabling switch.
3. Press the **Master** button for the unmastered axis.

   First of all, the premastering position is located by means of a search run. The mastering run is then performed. Once mastering has been carried out successfully, the axis moves to the calculated mastering position (zero position).

> If the search run or the mastering fails, the process is aborted and the robot stops.

7.1.2 Manually unmastering axes

Description

The saved mastering position of an axis can be deleted. This unmasters the axis. No motion is executed during unmastering.

Precondition

- Operating mode T1

Procedure

1. Select **Mastering** in the Robots view. The **Mastering** view opens.
2. Press the **Unmaster** button for the mastered axis. The axis is unmastered.

7.2 Calibration

7.2.1 Tool calibration

Description

During tool calibration, the user assigns a Cartesian coordinate system (tool coordinate system) to a tool mounted on the mounting flange.

The tool coordinate system has its origin at a user-defined point. This is called the TCP (Tool Center Point). The TCP is generally situated at the working point of the tool. A tool can have several TCPs.

Advantages of tool calibration:

- The tool can be moved in a straight line in the tool direction.
- The tool can be rotated about the TCP without changing the position of the TCP.
- In program mode: The programmed velocity is maintained at the TCP along the path.

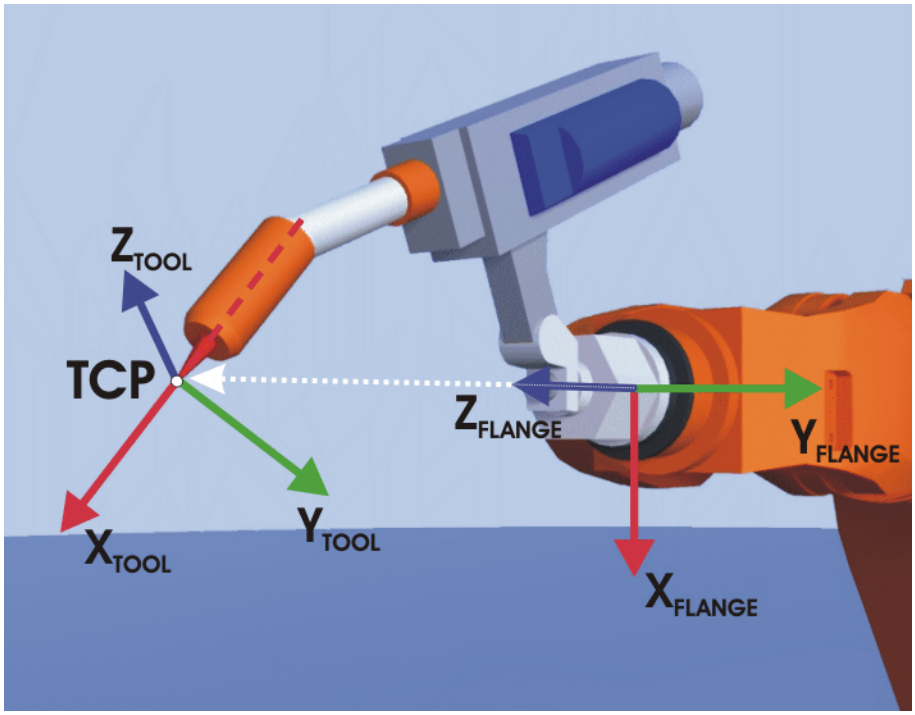


Fig. 7-1: TCP calibration principle

Overview

Tool calibration consists of 2 steps:

Step	Description
1	**Define the origin of the tool coordinate system** The following methods are available: ■ XYZ 4-point (>>> 7.2.1.1 "TCP calibration: XYZ 4-point method" Page 99)
2	**Definition of the orientation of the tool coordinate system** The following methods are available: ■ ABC -2-point (>>> 7.2.1.2 "Defining the orientation: ABC 2-point method" Page 101) ■ ABC World (>>> 7.2.1.3 "Defining the orientation: ABC World method" Page 102)

7.2.1.1 TCP calibration: XYZ 4-point method

Description

The TCP of the tool to be calibrated is 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 with which the reference point is addressed must maintain a certain minimum distance between one another. If the points are too close to one another, the position data cannot be saved. A corresponding error message is generated.

The quality of the calibration can be assessed by means of the translational calculation error which is determined during calibration. If this error exceeds a defined limit value, it is advisable to calibrate the TCP once more.

The minimum distance and maximum calculation error can be changed in Sunrise.Workbench. (>>> 10.1.1 "Configuring parameters for calibration" Page 147)

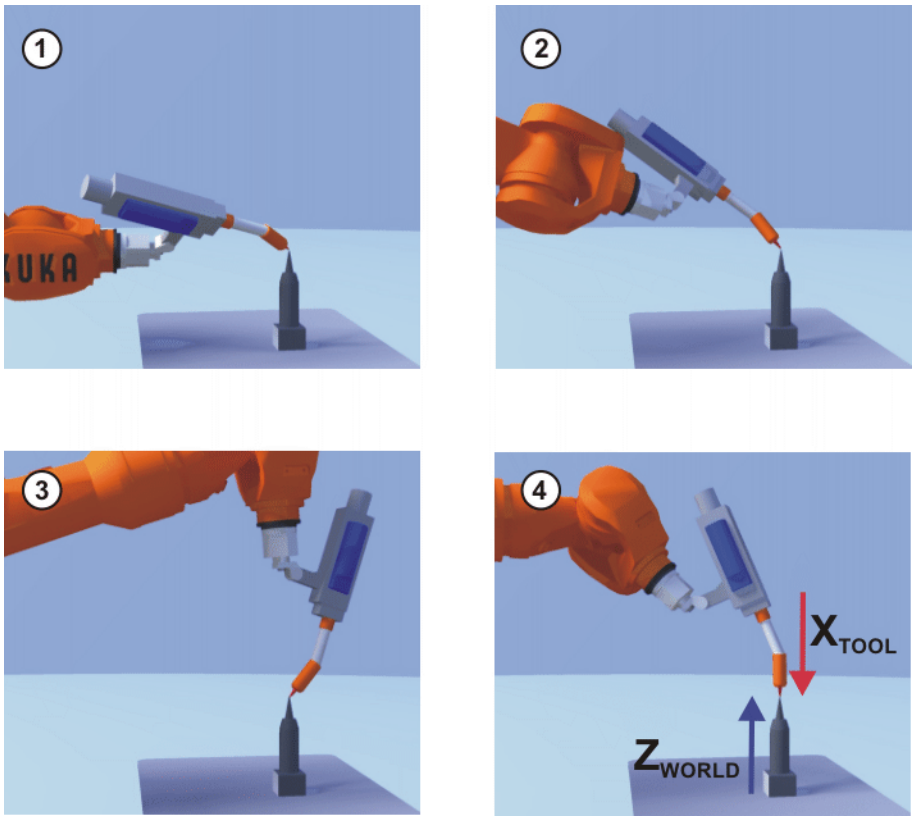


Fig. 7-2: XYZ 4-Point method

Precondition

- The tool to be calibrated is mounted on the mounting flange.
- The tool to be calibrated and the frame used as the TCP have been created in the object templates of the project and transferred to the robot controller by means of synchronization.
- T1 mode

Procedure

1. Select **Calibration** > **Tool calibration** in the Robots view. The **Tool calibration** view opens.
2. Select the tool to be calibrated and the corresponding TCP.
3. Select the **TCP calibration(XYZ 4-point)** method. The measuring points of the method are displayed as buttons:
   - **Measurement point 1 ... Measurement point 4**

   In order to be able to record a measuring point, it must be selected (button is orange).
4. Move the TCP to any reference point. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
5. Move the TCP to the reference point from a different direction. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
6. Repeat step 5 two more times.
7. Press **Determine tool data**. The calibration data and the calculation error are displayed in the **Apply tool data** dialog.
8. If the calculation error exceeds the maximum permissible value, a warning is displayed. Press **Cancel** and recalibrate the TCP.
9. If the calculation error is below the configured limit, press **Apply** to save the calibration data.
10. Either close the Calibration view or define the orientation of the tool coordinate system with the ABC 2-point or ABC World method.

(>>> 7.2.1.2 "Defining the orientation: ABC 2-point method" Page 101)
(>>> 7.2.1.3 "Defining the orientation: ABC World method" Page 102)

11. Synchronize the project in order to save the calibration data including the calculation error in Sunrise.Workbench.

7.2.1.2 Defining the orientation: ABC 2-point method

Description

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.

The points must maintain a defined minimum distance from one another. If the points are too close to one another, the position data cannot be saved. A corresponding error message is generated.

The minimum distance can be changed in Sunrise.Workbench.
(>>> 10.1.1 "Configuring parameters for calibration" Page 147)

This method is used for tools with edges and corners which the user can employ for orientation. Furthermore, it is used if it is necessary to define the axis directions with particular precision.

This method is not available for safety-oriented tools.

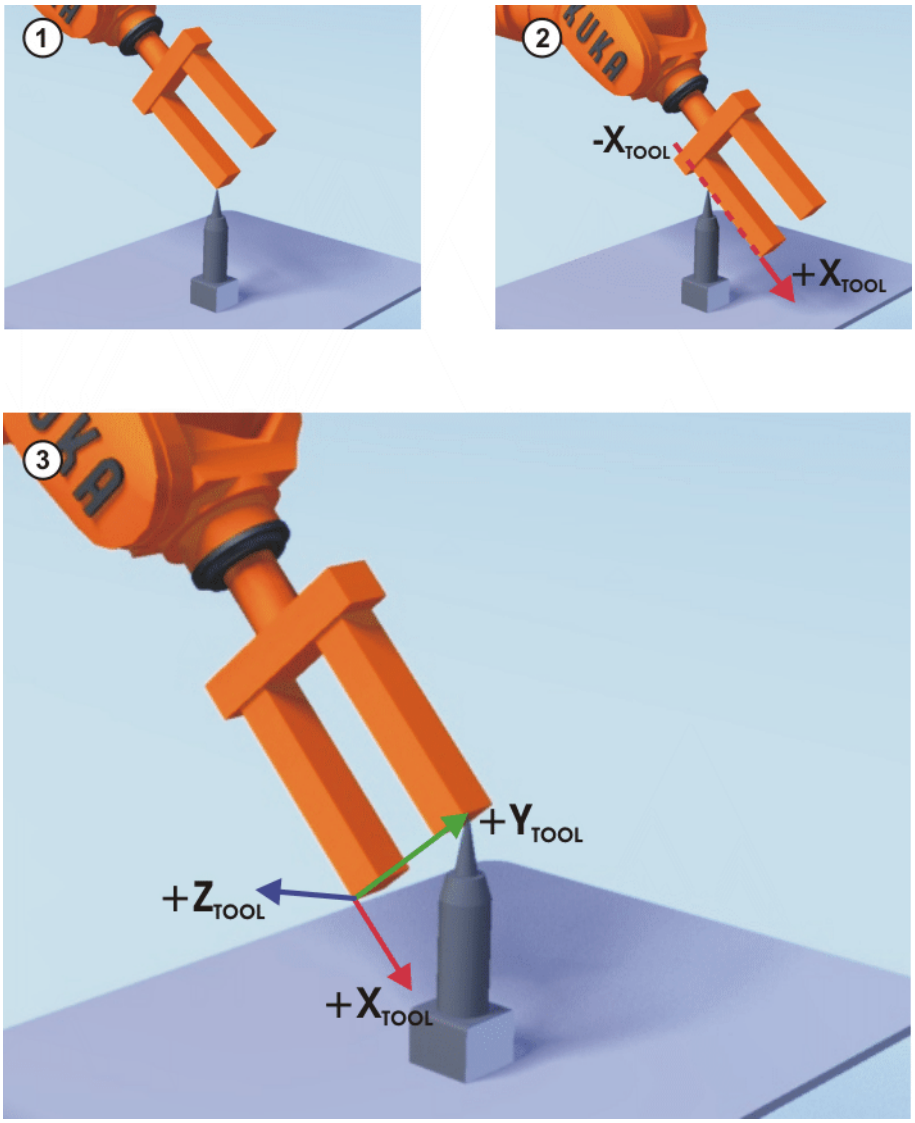


Fig. 7-3: ABC 2-Point method

Precondition

- The tool to be calibrated is mounted on the mounting flange.

- The TCP of the tool has already been measured.
- Operating mode T1

Procedure

1. Only if the Calibration view was closed following TCP calibration:

   Select **Calibration** > **Tool calibration** in the Robots view. The **Tool calibration** view opens.
2. Only if the Calibration view was closed following TCP calibration:

   Select the mounted tool and the corresponding TCP of the tool.
3. Select the **Defining the orientation(ABC 2-point)** method. The measuring points of the method are displayed as buttons:
   - **TCP**
   - **Negative X axis**
   - **Positive Y value on XY plane**

   In order to be able to record a measuring point, it must be selected (button is orange).
4. Move the TCP to any reference point. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
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 **Record calibration point**. The position data are applied and displayed for the selected measuring point.
6. Move the tool so that the reference point in the XY plane has a positive Y value. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
7. Press **Determine tool data**. The calibration data are displayed in the **Apply tool data** dialog.
8. Press **Apply** to save the calibration data.
9. Synchronize the project in order to save the calibration data in Sunrise.Workbench.

7.2.1.3 Defining the orientation: ABC World method

Description

The user aligns the axes of the tool coordinate system 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**: The user communicates the tool direction 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 influenced by the user.

  The system always defines the orientation of the other axes in the same way. If the tool subsequently has to be calibrated again, e.g. after a crash, it is therefore sufficient to define the tool direction again. Rotation about the tool direction need not be taken into consideration.
- **6D**: The user communicates the direction of all 3 axes to the robot controller.

This method is used for tools that do not have corners which the user can employ for orientation, e.g rounded tools such as adhesive or welding nozzles.

Precondition

- The tool to be calibrated is mounted on the mounting flange.
- The TCP of the tool has already been measured.
- Operating mode T1

Procedure

1. Only if the Calibration view was closed following TCP calibration:

Select **Calibration** > **Tool calibration** in the Robots view. The **Tool calibration** view opens.

2. Only if the Calibration view was closed following TCP calibration:
   Select the mounted tool and the corresponding TCP of the tool.
3. Select the **Defining the orientation(ABC world)** method.
4. Select the **ABC World 5D** or **ABC world 6D** option.
5. If **ABC World 5D** is selected:
   Align $+X_{TOOL}$ parallel to $-Z_{WORLD}$. ($+X_{TOOL}$ = tool direction)
   If **ABC world 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 $+X_{WORLD}$
6. Press **Determine tool data**. The calibration data are displayed in the **Apply tool data** dialog.
7. Press **Apply** to save the calibration data.
8. Synchronize the project in order to save the calibration data in Sunrise.Workbench.

7.2.2 Calibrating the base: 3-point method

Description

During base calibration, the user assigns a Cartesian coordinate system (base coordinate system) to a frame selected as the base. The base coordinate system has its origin at a user-defined point.

Advantages of base calibration:

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

The origin and 2 further points of a base are addressed with the 3-point method. These 3 points define the base.

The points must maintain a defined minimum distance from the origin and minimum angles between the straight lines (origin – X axis and origin – XY plane). If the points are too close to one another or if the angle between the straight lines is too small, the position data cannot be saved. A corresponding error message is generated.

The minimum distance and angles can be changed in Sunrise.Workbench. (>>> 10.1.1 "Configuring parameters for calibration" Page 147)

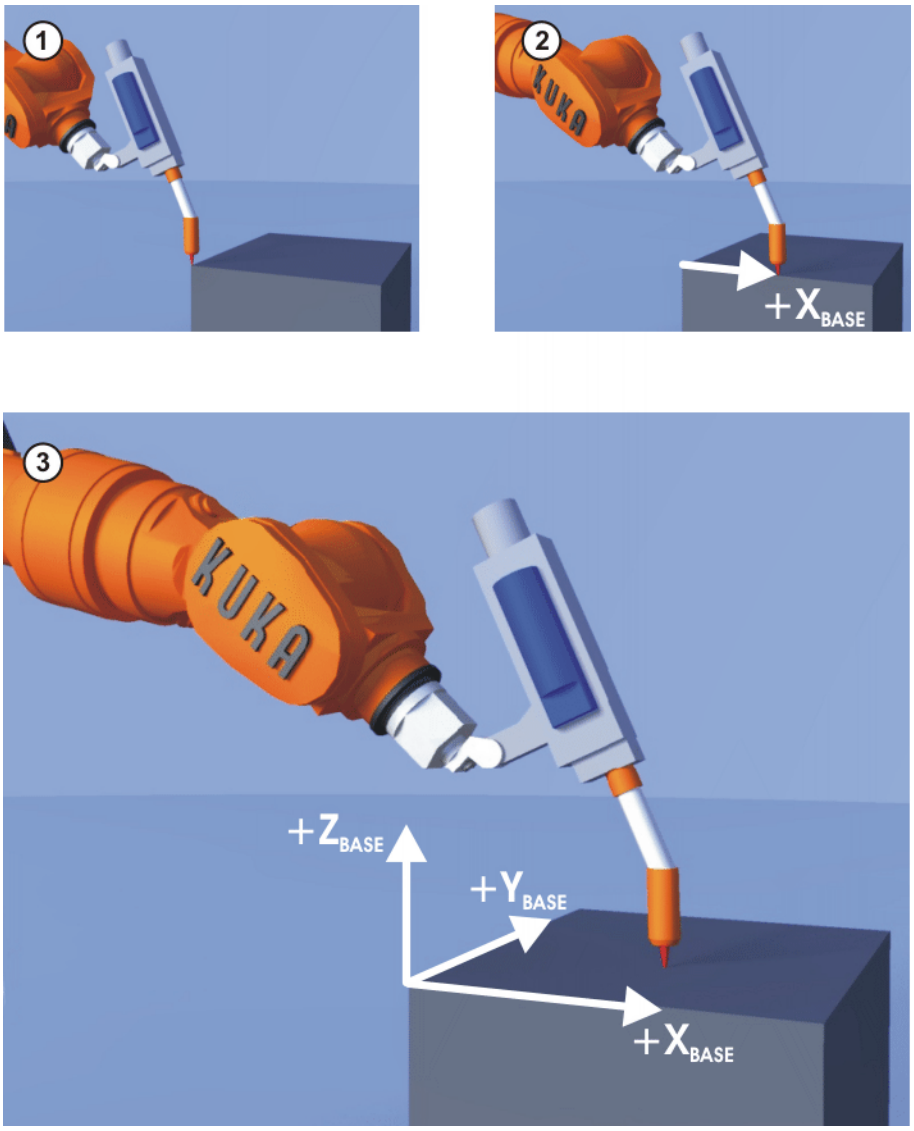


Fig. 7-4: 3-point method

Precondition

- A previously calibrated tool is mounted on the mounting flange.
- The frame to be calibrated has been selected as the base in the application data of the project and transferred to the robot controller by means of synchronization.
- T1 mode

Procedure

1. Select **Calibration** > **Base calibration** in the Robots view. The **Base calibration** view opens.
2. Select the base to be calibrated.
3. Select the mounted tool and the TCP of the tool with which the measuring points of the base are addressed.

   The measuring points of the 3-point method are displayed as buttons:

   - **Origin**
   - **Positive X axis**
   - **Positive Y value on XY plane**

   In order to be able to record a measuring point, it must be selected (button is orange).
4. Move the TCP to the origin of the base. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.

5. Move the TCP to a point on the positive X axis of the base. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
6. Move the TCP to a point in the XY plane with a positive Y value. Press **Record calibration point**. The position data are applied and displayed for the selected measuring point.
7. Press **Determine base data**. The calibration data are displayed in the **Apply base data** dialog.
8. Press **Apply** to save the calibration data.
9. Synchronize the project in order to save the calibration data in Sunrise.Workbench.

7.3 Determining tool load data

Description

During load data determination, the robot performs multiple measurement runs with different orientations of wrist axes A5, A6 and A7. The load data are calculated from the data recorded during the measurement runs.

The mass and the position of the center of mass of the tool mounted on the robot flange can currently be determined. It is also possible to specify the mass and to determine the position of the center of mass on the basis of the mass that is already known.

At the start of load data determination, axis A7 is moved to the zero position and axis A5 is positioned in such a way that axis A6 is aligned perpendicular to the weight. During the measurement runs, axis A6 has to be able to move between -95° and +95°, while axis A7 has to be able to move from 0° to -90°.

The remaining robot axes (A1 to A4) are not moved during load data determination. They remain in the starting position during measurement.

The quality of the load data determination may be influenced by the following constraints:

- Mass of the tool

  Load data determination becomes more reliable as the mass of the tool increases. This is because measurement uncertainties have a greater influence on a smaller mass.

  > Load data determination cannot yet be used reliably for masses of less than one kilogram.

- Supplementary loads

  Load data determination is only suitable for loads which are fixed to the robot flange. Supplementary loads, i.e. loads which are attached to the robot structure, e.g. dress packages, are not supported by load data determination.

- Start position from which load data determination is started

  A suitable start position should be determined first and meet the following criteria:

  - Axes A1 to A5 are as far away as possible from singularity positions.

    The criterion is relevant if the mass is to be determined during load data determination. If load data determination is only possible in poses for which axes A1 to A5 are close to singularity positions, the mass can be specified. If only the center of mass is to be determined on the basis of the specified mass, the criterion of axis position is irrelevant.
  - The suitability of the start position for load data determination in the case of a robot for which automatic load data determination is to be carried out must be checked before the load that is to be determined is mounted on the robot.

A robot pose is suitable as a start position for load data determination if there are only slight external torques acting on the load-free robot in this position. This can be checked via the display of the external axis torques.

(>>> 6.16.4 "Displaying axis-specific torques" Page 92)

If the mass is to be determined during load data determination, all external axis torques are relevant and should be checked where possible in advance for the load-free robot. If only the center of mass is to be determined on the basis of a specified mass, only the external axis torque of axis A6 is relevant.

- Interference torques during the measurement runs
  - During the measurement runs, no interference torques may be applied, e.g. by pulling or pushing the robot.
  - Moving parts, e.g. dress packages, generate interference torques that shift the center of mass during the measurement run.

> The application of interference torques during load data determination results in falsified load data. (>>> 9.3.6 "Load data" Page 136)

For the load data determination for safety-oriented tools, it must be ensured that the modified load data are not automatically transferred to the safety configuration located on the robot controller. (>>> 9.3.7 "Safety-oriented tool" Page 137)

The load data determined for a safety-oriented tool must first be updated in Sunrise.Workbench by means of project synchronization. This changes the safety configuration of the project in Sunrise.Workbench; the project must then be re-transferred to the robot controller by synchronizing the project again.

> If a changed safety configuration is activated on the robot controller, the safety maintenance technician must carry out safety acceptance. (>>> 13.11 "Safety acceptance overview" Page 226)
> To avoid spending unnecessary time performing verifications, it is advisable to mark a tool as safety-oriented only when the load data have been correctly entered or determined and have been transferred to Sunrise.Workbench.

Preparation

- Determine the start position from which load data determination is to be started.

Precondition

- The tool is mounted on the mounting flange.

  The tool has been created in the object templates of the project and transferred to the robot controller by means of synchronization.
- The robot is in the desired start position.
- There is a sufficiently large workspace available in the wrist axis range.
- T1, T2 or AUT mode

> **NOTICE** If parts of the mounted tool project behind the flange plane (in the negative Z direction relative to the flange coordinate system), there is a risk of the tool colliding with the manipulator during the measurement runs.
> If the motion of the axes during load data determination is unknown to the operator, or if a collision between the tool and manipulator cannot be ruled out (e.g. for the initial load data determination for a tool), it is advisable to determine the load data in T1 mode. This does not affect the quality of the measurement results.

Procedure

1. Select **Load data** in the Robots view. The **Load data** view opens.
2. Select the mounted tool from the selection list.

3. If T1 or T2 mode is set, press and hold down the enabling switch until load data determination has been completed.
4. Press **Determining the load data**.
5. If the tool already has a mass, the operator will be asked if the mass is to be redetermined.
   - Select **Use existing mass** if the currently saved mass is to be retained.
   - Select **Redetermine mass** if the mass is to be determined again.

> If the currently saved mass is to be retained in the load determination, it must be ensured that the specified mass value is correct. Otherwise, the center of mass cannot be determined accurately.

6. The robot starts the measurement runs and the load data are determined. A progress bar is displayed.

> If the motion enable signal is withdrawn during load data determination, e.g. by an EMERGENCY STOP or by releasing the enabling switch (T1, T2), the load data determination is aborted and must be restarted.

   Once load data determination has been completed, the determined load data are displayed in the **Apply load data** dialog.

   Press **Apply** to save the determined load data.
7. Synchronize the project so that the load data are saved in Sunrise.Workbench.

   When the load data for a safety-oriented tool have been determined, the safety configuration changes as a result of the project synchronization.
8. If necessary, synchronize the project in order to transfer the changed safety configuration to the robot controller.

Overview

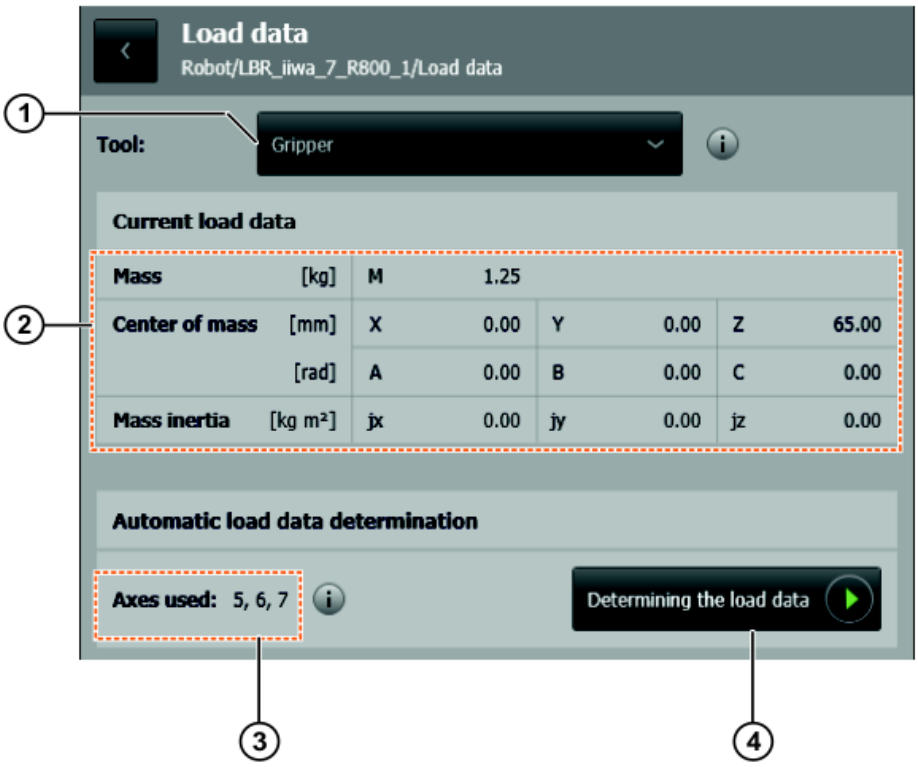


Fig. 7-5: Determining the load data

Item	Description
1	Tool selection list The tools created in the object templates are available for selection here.
2	Load data display Displays the current load data of the selected tool.
3	Display of axes used Displays the axes that are moved for load data determination.
4	**Determining the load data** button Starts load data determination. The button is only active if a tool has been selected and the motion enable signal has been issued.

8 Brake test

8.1 Overview of the brake test

Description

Each robot axis has a holding brake integrated into the drive train. The brakes have 2 functions:

- Stopping the robot when the servo control is deactivated or the robot is de-energized.
- Switching the robot to the safe state “Standstill” in the event of a fault.

The brake test checks whether the brake holding torque applied by each brake is high enough, i.e. whether it exceeds a specific reference torque. This reference torque can be specified by the programmer or read from the motor data.

> Unless otherwise determined by a risk assessment, the brake test must be performed regularly:
> - The brake test must be carried out for each axis during start-up and recommissioning of the industrial robot.
> - The brake test must be performed daily during operation.
>
> The user can carry out a risk assessment to determine whether the brake test is required for the specific application and, if so, how often it is to be performed.

Execution

A precondition for execution of the brake test is that the robot is at operating temperature.

The brake test is manually executed by means of an application. A prepared brake test application for the LBR iiwa is available from Sunrise.Workbench.

If the prepared brake test application is used, the robot is moved prior to the actual brake test and the resulting maximum absolute torque is determined for each axis. In the brake test application, the torque determined is communicated to the brake test as the reference holding torque.

The determination of the maximum absolute torques is referred to in the following as torque value determination.

> It is advisable to remove the torque value determination from the application and to test the brakes against the minimum brake holding torque. If a torque specified by the programmer is tested, a risk analysis must first be carried out to determine whether there is any danger if the brakes show a lower torque than the minimum brake holding torque.

Procedure

When a brake is tested, the following steps are carried out by default:

1. The axis moves at constant velocity over a small axis angle of max. 5° (on the output side). The gravitation and friction are determined during this motion.
2. When the axis has returned to its starting position and the axis drive is stationary, the brake is closed.
3. One of the following values is used as the holding torque to be tested: the reference holding torque determined, the minimum brake holding torque or the motor holding torque.

   The holding torque to be tested is defined internally by the system according to the following rules:

   a. If the reference holding torque is greater than the lowest value of the minimum brake and motor holding torques, then the lowest value of the minimum brake and motor holding torques is used as the holding torque to be tested.

b. If the reference holding torque is lower than 20% of the lowest value of the minimum brake and motor holding torques, then 20% of the lowest value of the minimum brake and motor holding torques is used as the holding torque to be tested.

c. In all other cases, the reference holding torque is used.

At the start of the brake test, with the brake closed, the setpoint torque of the drive is set to 80% of the holding torque to be tested.

> The minimum and maximum brake holding torques are saved in the motor data. The motor holding torque is derived from the motor data.

4. The drive torque is gradually increased until a change in position is detected or the maximum brake holding torque (derived from the motor data) is reached. The brake test ends when the maximum brake holding torque is reached.
5. The torque applied against the brake when a change in position is detected is measured. This is the measured holding torque.
6. The measured holding torque is evaluated relative to the holding torque to be tested.

   The brake test is successful if the measured holding torque lies within the following range:

   - ≥ 105% of the holding torque to be tested … ≤ maximum brake holding torque

   If the measured holding torque lies below the holding torque to be tested, the brake test has failed, i.e. the brake is identified as being defective.

   The test result is displayed on the smartHMI.

   (>>> 8.4.2 "Results of the brake test (display) " Page 125)
7. When the brake test has ended and the robot is stationary, the brake is briefly opened and closed again. This releases any remaining tension in the brake and prevents undesired robot motions.

> If the brake test for an axis fails, i.e. a brake has been identified as being defective, the brake test can be repeated for confirmation.

> If the application is paused during the brake test or if a safety stop is triggered, e.g. by an EMERGENCY STOP, the brake test is aborted. The brake test for the axis is repeated when the application is resumed.

> The brake test does not depend on the loads mounted on the robot, as gravitation and friction are taken into consideration when the test is carried out.

Overview

The following describes the steps for executing the brake test with the template available in Sunrise.Workbench.

The brake test application can be adapted and expanded. The comments contained in the template must be observed.

> **NOTICE** If a brake is defective, the corresponding axis may slip during the brake test and the robot may sag. The brake test must be executed in a position in which no damage could result from potential sagging. The starting position for the brake test must be selected accordingly.

NOTICE If the brake test fails for an axis, i.e. if a brake has been identified as being defective, the application must ensure that the robot is automatically moved to a safe position. A safe position is a position in which the robot is supported in such a way that it either cannot sag or cannot cause damage in the event of sagging.

Step	Description
1	Create the brake test application from the template. (>>> 8.2 "Creating the brake test application from the template" Page 111)
2	In the brake test application, remove or adapt the application-specific maximum absolute torques determined. At the start of the brake test application, 2 predefined axis positions are addressed by default. The maximum absolute torque for each axis is thus determined and communicated to the brake test as the reference holding torque. It is advisable to test the brakes against the minimum brake holding torque, which is stored in the motor data. To do so, the prepared brake test application must be adapted. (>>> 8.2.1 "Adapting the brake test application for testing against the minimum brake holding torque" Page 114) If the brake test requires the maximum absolute torques which occur when a user-specific robot application is executed, the user-specific robot application can be added to the brake test application. Since the brakes are not tested against the minimum brake holding torque in this case, a risk analysis must first be carried out. (>>> 8.2.2 "Changing the motion sequence for torque value determination" Page 114)
3	Change the starting position for the brake test. The starting position is the vertical stretch position by default. If required, a different starting position can be selected. (>>> 8.2.3 "Changing the starting position for the brake test" Page 115)
4	If necessary, make further user-specific adaptations in the brake test application. Examples: ■ Setting the output for a failed brake test. ■ Saving the test results in a file. (>>> 8.3 "Programming interface for the brake test" Page 115)
5	Synchronize the project in order to transfer the brake test application to the robot controller.
6	Execute the brake test application. (>>> 8.4 "Performing a brake test" Page 124)

8.2 Creating the brake test application from the template

Procedure

1. Select the Sunrise project in the **Package Explorer**.
2. Select the menu sequence **File** > **New** > **Other...**.

3. In the **Sunrise** folder, select the **Application for the brake test of LBR iiwa** option and click on **Finish**.

   The **BrakeTestApplication.java** application is created in the source folder of the project and opened in the editor area of Sunrise.Workbench.

Description

In the run() method of the **BrakeTestApplication.java** application (limited here to the relevant command lines), the execution of the brake test is implemented for all axes of the LBR iiwa.

An optional data evaluation preceding the actual brake test is also implemented. 2 predefined axis positions are addressed in order to determine the maximum absolute torque for each axis.

```
1  public void run() {
2     ...
3     lbr_iiwa.move(ptpHome());
4     ...
5     TorqueEvaluator evaluator = new TorqueEvaluator(lbr_iiwa);
6     ...
7     evaluator.setTorqueMeasured(false);
8
9     evaluator.startEvaluation();
10    ...
11    lbr_iiwa.move(new PTP(new JointPosition(
12       0.5,  0.8,  0.2,  1.0, -0.5, -0.5, -1.5)).
13       setJointVelocityRel(relVelocity));
14    lbr_iiwa.move(new PTP(new JointPosition(
15       -0.5, -0.8, -0.2, -1.0,  0.5,  0.5,  1.5)).
16       setJointVelocityRel(relVelocity));
17    ...
18    TorqueStatistic maxTorqueData = evaluator.stopEvaluation();
19
20    boolean allAxesOk = true;
21
22    for (int axis : axes) {
23       try {
24          BrakeTest brakeTest = new BrakeTest(axis,
25             maxTorqueData.getMaxAbsTorqueValues()[axis]);
26          IMotionContainer motionContainer = lbr_iiwa.move(brakeTes
   t);
27          BrakeTestResult brakeTestResult =
28             BrakeTest.evaluateResult(motionContainer);
29          switch(brakeTestResult.getState().getLogLevel())
30          {
31             case Info:
32                getLogger().info(brakeTestResult.toString());
33                break;
34             case Warning:
35                getLogger().warn(brakeTestResult.toString());
36                break;
37             case Error:
38                getLogger().error(brakeTestResult.toString());
39                allAxesOk = false;
40                break;
41             default:
42                break;
43          }
44       catch (CommandInvalidException ex) {
45          ...
46          allAxesOk = false;
47       }
48    }
```

```
49
50     if (allAxesOk){
51        getLogger().info("Brake test was successful for all axes.");
52     }
53     else{
54        getLogger().error("Brake test failed for at least one
   axis.");
55     }
56 }
```

Line	Description
3	Address the starting position from which the robot is moved to determine the maximum absolute torque for each axis. The starting position is the vertical stretch position by default.
5	Prepare the data evaluation. In order to perform an axis-specific evaluation of the torques determined during a motion sequence, an instance of the TorqueEvaluator class must be created.
7	Select the torques to be used for the data evaluation. The measured torques are not used, but instead the torques that are calculated using the robot model during the motion sequence. Measurements are susceptible to malfunctions. The calculation of the torque values ensures that no interference torques resulting from dynamic effects (e.g. robot acceleration) are incorporated into the data evaluation.
9	Start the data evaluation. The data evaluation is started with the startEvaluation() command of the TorqueEvaluator class.
11 … 16	Carry out the motion sequence to determine the maximum absolute torques 2 predefined axis positions are each addressed with a PTP motion.
18	End the data evaluation and poll the data. The stopEvaluation() command of the TorqueEvaluator class ends the data evaluation and returns the result as a value of type TorqueStatistic. The result is saved in the variable maxTorqueData.
20	Variable for the evaluation of the brake test The result of the brake test is saved for later evaluation via the variable allAxesOk. It is set to “false” if the brake test of an axis fails or is aborted due to an error. Otherwise it retains the value “true”.
22 … 54	Execute the brake test The brakes are tested one after the other, starting with the brake of axis A1. 1. Lines 24, 25: An object of type BrakeTest is created. In the process, the corresponding axis and the previously determined maximum absolute torque are transferred as the reference holding torque. 2. Line 26: The brake test is executed as a motion command. 3. Lines 27 ... 54: The result of the brake test is evaluated and displayed on the smartHMI.

8.2.1 Adapting the brake test application for testing against the minimum brake holding torque

Description

The brake test checks whether the brakes apply the minimum brake holding torque. It is therefore advisable to adapt the prepared brake test application in accordance with the following description.

If the brake test is to be executed without reference holding torques being determined and made available to the brake test, all the command lines relevant for torque value determination must be removed from the brake test application. As a consequence, the brake test application then starts with the motion to the starting position for the brake test.

In addition, when creating the BrakeTest instance, the parameter with which the reference torque is transferred must be removed.

```
boolean allAxesOk = true;
for (int axis : axes) {
    try {
        BrakeTest brakeTest = new BrakeTest(axis, maxTorqueData.getMaxAbsTorqueValues()[axis]);

        IMotionContainer motionContainer = lbr_iiwa.move(brakeTest);
        BrakeTestResult brakeTestResult = BrakeTest.evaluateResult(motionContainer);
```

1

Fig. 8-1: Transferring the reference torque for the brake test

1 Constructor of the class BrakeTest with transfer of reference torque

Procedure

1. Open the brake test application in Sunrise.Workbench.
2. Make the following changes to the run() method of the application:
   - Delete all command lines which are relevant for torque value determination.
   - When calling the constructor of the BrakeTest class, delete the following parameters:

     `maxTorqueData.getMaxAbsTorqueValues()[axis]`

     The following code remains in the line:

```
BrakeTest brakeTest = new brakeTest(axis);
```

3. Save changes.

8.2.2 Changing the motion sequence for torque value determination

Description

The brake test application created from the template contains a prepared motion sequence for determining the maximum absolute torques generated in each axis.

The robot is moved from the vertical stretch position by default. A different starting position can be selected.

2 predefined axis positions are each addressed from the starting position with a PTP motion. In order to determine the maximum absolute torques in a specific robot application and to use these as reference holding torques for the brake test, the robot application must be added in the brake test application.

```
public void run() {
    // initial position
    getLogger().info("Moving into initial position.");
    lbr_iiwa.move(ptpHome());

    // start monitoring for maximum torques
    getLogger().info("Start evaluation of torque statistic.");
    TorqueEvaluator evaluator = new TorqueEvaluator(lbr_iiwa);

    // select static model torque values
    evaluator.setTorqueMeasured(false);

    evaluator.startEvaluation();

    getLogger().info("Perform desired robot motion.");
    lbr_iiwa.move(new PTP(new JointPosition( 0.5,  0.8,  0.2,  1.0, -0.5, -0.5, -1.5)).setJointVelocityRel(relVelocity));
    lbr_iiwa.move(new PTP(new JointPosition(-0.5, -0.8, -0.2, -1.0,  0.5,  0.5,  1.5)).setJointVelocityRel(relVelocity));

    // stop monitoring for maximum torques and store results
    getLogger().info("Stop evaluation of torque statistic.");
    TorqueStatistic maxTorqueData = evaluator.stopEvaluation();
    getLogger().info("The result of evaluation:\n" + maxTorqueData.toString());
```

Fig. 8-2: Motion sequence for torque value determination

1 ptpHome() motion to starting position

2 Predefined motion sequence for torque value determination

Procedure

1. Open the brake test application in Sunrise.Workbench.
2. If necessary, make the following changes to the run() method of the application:
   - Replace the ptpHome() motion that brings the robot to the starting position with a motion to the desired starting position.
   - Replace the predefined motion sequence with the appropriate application code.
3. Save changes.

8.2.3 Changing the starting position for the brake test

Description

By default, the brake test application created from the template executes the brake test to the end position of the motion sequence in order to determine the maximum absolute torque. If this position is not suitable for the brake test, a motion to the desired starting position must be programmed before the brake test is executed.

> To determine the gravitation and friction, the axes of an LBR iiwa are moved towards the mechanical zero position. The maximum travel is 5° on the output side.

8.3 Programming interface for the brake test

With the BrakeTest class, the RoboticsAPI offers a programming interface for the execution of the brake test. The brake test is executed as a motion command.

In addition, using the TorqueEvaluator class, the torques measured during a motion sequence can be evaluated and the maximum absolute torque for each axis can be determined. This torque can be used as the reference holding torque for the brake test.

8.3.1 Evaluating the torques generated and determining the maximum absolute value

Description

In order to perform an axis-specific evaluation of the torques determined during a motion sequence, an object of the TorqueEvaluator class must first be created. The LBR instance for whose axes the maximum absolute torque values are to be determined is transferred to the constructor of the TorqueEvaluator class.

The evaluation can be started and then ended with the following methods of the TorqueEvaluator class:

- startEvaluation(): Starts the evaluation.

  Once the method has been called, the motion sequence to be evaluated must be commanded.
- stopEvaluation(): Ends the evaluation.

  The method returns an object of type TorqueStatistic. The results of the evaluation can be polled via this object.

The torques generated during the motion sequence can be determined in different ways:

- Measured torques: The torques measured by the joint torque sensors are used.
- Static torques (model-based): The torques calculated using the static robot model are used.

The setTorqueMeasured(…) method of the TorqueEvaluator class can be used to define whether the measured or static (model-based) torques are to be used for the evaluation.

Syntax

```
TorqueEvaluator evaluator = new TorqueEvaluator(lbr_iiwa);
evaluator.setTorqueMeasured(isTorqueMeasured);
evaluator.startEvaluation();
//Motion sequence
TorqueStatistic maxTorqueData = evaluator.stopEvaluation();
```

Explanation of the syntax

Element	Description
evaluator	Type: TorqueEvaluator Variable to which the created TorqueEvaluator instance is assigned. The evaluation of the torques during a motion sequence is started and ended via the variable.
isTorque Measured	Type: Boolean Input parameter of the setTorqueMeasured(…) method: Defines whether the measured torque values or the values calculated using the static robot model are to be used for the evaluation. ■ **true**: measured torques are used ■ **false**: static torques (model-based) are used **Note**: When using the static (model-based) torques, dynamic effects, which can for example be generated by robot acceleration, have no influence on the determined values.
lbr_iiwa	Type: LBR LBR instance of the application. Represents the robot for which the maximum absolute torque values are to be determined.
maxTorque Data	Type: TorqueStatistic Variable for the return value of stopEvaluation(). The return value contains the determined maximum absolute torque values and further information for the evaluation.

8.3.2 Polling the evaluation results of the maximum absolute torques

When the evaluation of the maximum absolute torque values has ended, the results of the evaluation can be polled.

Overview

The following methods of the TorqueStatistic class are available:

Method	Description
getMaxAbs TorqueValues()	Return value type: double[]; unit: Nm Returns a double array containing the determined maximum absolute torque values (output side) for all axes.
getSingleMaxAbs TorqueValue(...)	Return value type: double; unit: Nm Returns the maximum absolute torque value (output side) for the axis which is transferred as the parameter (type: JointEnum).
areDataValid()	Return value type: Boolean The system polls whether the determined data are valid (= true). The data are valid if no errors occur during command processing.
getStartTimestamp()	Return value type: java.util.Date Returns the time at which the evaluation was started.
getStopTimestamp()	Return value type: java.util.Date Returns the time at which the evaluation was ended.
isTorqueMeasured()	Return value type: Boolean Polls whether the measured torques or the torques calculated using the static robot model were used for evaluating the maximum absolute torque. ■ **true**: measured torques are used ■ **false**: statisc torques (model-based) are used

Example

The maximum torques which occur during a joining task are to be used as reference torques in a brake test. For this purpose, the torques which are measured during the execution of the joining task are evaluated, and the maximum absolute torque for each axis is determined.

Once the evaluation has been started, the motion commands of the joining process are executed. When the joining process is completed, the evaluation is ended and the results of the evaluation for axes A2 and A4 are saved in the process data. If the determined data are invalid, an output is set.

testEvaluator.setTorqueMeasured(true);

```
private LBR testLBR;
private BrakeTestIOGroup brakeTestIOs;
private Tool testGripper;
private Workpiece testWorkpiece;
...
public void run() {

testGripper.attachTo(testLBR.getFlange());
testWorkpiece.attachTo(testGripper.getFrame("/GripPoint"));

// create TorqueEvaluator
TorqueEvaluator testEvaluator = new TorqueEvaluator(testLBR);

// select measured torque values
testEvaluator.setTorqueMeasured(true);
```

```
// start evaluation
testEvaluator.startEvaluation();

// performs assembly task
testAssemblyTask();

// finish evaluation and store result in variable testMaxTrqData
TorqueStatistic testMaxTrqData = testEvaluator.stopEvaluation();

// get maximum absolute measured torque value for joint 2
double maxTrqA2 = testMaxTrqData
   .getSingleMaxAbsTorqueValue(JointEnum.J2);

// save result
getApplicationData().getProcessData("maxTrqA2").setValue(maxTrqA2);

// get maximum absolute measured torque value for joint 4
double maxTrqA4 = testMaxTrqData
   .getSingleMaxAbsTorqueValue(JointEnum.J4);

// save result
getApplicationData().getProcessData("maxTrqA4").setValue(maxTrqA4);

// check if evaluated data is valid
boolean areDataValid = testMaxTrqData.areDataValid();
if(areDataValid == false){
   // if data is not valid, set output signal
   brakeTestIOs.setEvaluatedTorqueInvalid(true);
}
...
}

public void exampleAssemblyTask(){

testLBR.move(ptp(getFrame("/StartAssembly")));

ForceCondition testForceCondition =
   ForceCondition.createNormalForceCondition
   (testWorkpiece.getDefaultMotionFrame(), CoordinateAxis.Z, 15.0);
testWorkpiece.move(linRel(0.0, 0.0, 100.0)
   .breakWhen(testForceCondition));

CartesianSineImpedanceControlMode testAssemblyMode =
   CartesianSineImpedanceControlMode.createLissajousPattern(
   CartPlane.XY, 5.0, 10.0, 500.0);

testWorkpiece.move(positionHold(
   testAssemblyMode, 3.0, TimeUnit.SECONDS));

openGripper();
testWorkpiece.detach();

testGripper.move(linRel(0.0, 0.0, -100.0));
}
```

8.3.3 Creating an object for the brake test

Description

In order to be able to execute the brake test, an object of the BrakeTest class must first be created. The index of the axis for which the brake test is to be executed is transferred to the constructor of the BrakeTest class.

Optionally, the *torque* parameter can be used to transfer a reference holding torque, e.g. the maximum absolute axis torque which occurs in a specific application.

As a general rule, the brake test must check whether the brakes apply the minimum brake holding torque. It is therefore advisable not to specify the *torque* parameter.

Syntax

`BrakeTest` *brakeTest* `= new BrakeTest(`*axis*`, <`*torque*`>);`

Explanation of the syntax

Element	Description
brakeTest	Type: BrakeTest Variable to which the created BrakeTest instance is assigned. The execution of the brake test is commanded via the variable as a motion command.
axis	Type: int Index of the axis whose brake is to be tested. ■ **0 … 6**: Axes A1 … A7
torque	Type: double; unit: Nm Reference holding torque (output side) specified by the user, e.g. the maximum absolute torque that has been determined beforehand for an axis-specific motion sequence. If no reference holidng torque is specified, the brake test uses the lowest of the following values as the holding torque: minimum brake holding torque or motor holding torque. If a reference holding torque is specified, one of the following values is used as the holding torque to be tested: the specified reference holding torque (*torque*), the minimum brake holding torque or the motor torque. The holding torque to be tested is defined internally by the system according to the following rules: 1. If the reference holding torque is greater than the lowest value of the minimum brake and motor holding torques, then the lowest value of the minimum brake and motor holding torques is used as the holding torque to be tested. 2. If the reference holding torque is lower than 20% of the lowest value of the minimum brake and motor holding torques, then 20% of the lowest value of the minimum brake and motor holding torques is used as the holding torque to be tested. 3. In all other cases, the reference holding torque is used. **Note**: The minimum and maximum brake holding torques are saved in the motor data. The motor holding torque is derived from the motor data.

8.3.4 Starting the execution of the brake test

Description

The brake test is executed by a motion command which is made available via the BrakeTest class. In order to execute the brake test, the move(…) or moveAsync(…) method is called with the robot instance used in the application, and the object created for the brake test is transferred.

In order to evaluate the result of the brake test, the return value of the motion command must be saved in a variable of type Typ IMotionContainer.

If an error is detected while the brake test is being executed, the brake test is aborted. In order to be able to react to errors in the program, it is advisable to command the execution and evaluation of the brake test within a try block and to deal with the CommandInvalidException arising from the error.

Syntax

```
try{
BrakeTest brakeTest = ...;
IMotionContainer brakeTestMotionContainer =
robot.move|moveAsync(brakeTest);
...
} catch(CommandInvalidException ex{
...
}
```

Explanation of the syntax

Element	Description
brakeTest	Type: BrakeTest Variable to which the created BrakeTest instance is assigned. The instance defines the axis for which the brake test is to be executed and can optionally define a reference holding torque specified by the programmer.
brakeTest Motion Container	Type: IMotionContainer Variable for the return value of the move(…) or moveAsync(…) motion command used to carry out the brake test. When the brake test has ended, the result can be evaluated using the variable.
robot	Type: Robot Instance of the robot used in the application. The brake test is to be performed on the axes of this robot.
ex	Type: CommandInvalidException Exception which occurs when the brake test is aborted due to an error. It is advisable to treat the exception within the catch block in such a way that an aborted brake test for a single brake does not cancel the entire brake test application.

8.3.5 Evaluating the brake test

Description

When the brake test has ended, the result can be evaluated. For this purpose, the return value of the motion command used to carry out the brake test must be assigned to a variable of type IMotionContainer.

In order to evaluate the brake test, the IMotionContainer instance of the corresponding motion command is transferred to the static method evaluateResult(…). The method belongs to the BrakeTest class and returns an object of type BrakeTestResult. Various information concerning the executed brake test can be polled from this object.

Syntax

```
IMotionContainer brakeTestMotionContainer =
robot.move|moveAsync(brakeTest);
BrakeTestResult result =
BrakeTest.evaluateResult(brakeTestMotionContainer);
```

Explanation of the syntax

Element	Description
brakeTest Motion Container	Type: IMotionContainer Variable for the return value of the move(…) or moveAsync(…) motion command used to carry out the brake test.
result	Type: BrakeTestResult Variable for the return value of evaluateResult(…). The return value contains the results of the brake test and further information concerning the brake test which can be polled via the variable.

Overview

The following methods of the BrakeTestResult class are available for evaluating the brake test:

Method	Description
getAxis()	Return value type: int Returns the index of the axis whose brake has been tested. The index starts with 0 (= axis A1).
getBrakeIndex()	Return value type: int Returns the index of the tested brake of the motor (starting with 0). In a brake test for the LBR iiwa, the value 0 is always returned.
getFriction()	Return value type: double; unit: Nm Returns the frictional torque (output side) determined during the test motion.
getGravity()	Return value type: double; unit: Nm Returns the gravitational torque (output side) determined during the test motion.
getMaxBrake HoldingTorque()	Return value type: double; unit: Nm Returns the torque (output side) determined from the motor data which the brake must not exceed. (= maximum brake holding torque)
getMeasuredBrake HoldingTorque()	Return value type: double; unit: Nm Returns the holding torque (output side) measured during the brake test. This value is compared with the holding torque to be tested.
getMinBrake HoldingTorque()	Return value type: double; unit: Nm Returns the minimum brake torque (output side) that can be reached, as determined from the motor data. (= minimum brake holding torque)
getMotor HoldingTorque()	Return value type: double; unit: Nm Returns the motor holding torque (output side) determined from the motor data.
getMotorIndex()	Return value type: int Returns the index of the tested motor of the drive (starting with 0). In a brake test for the LBR iiwa, the value 0 is always returned.
getMotor MaximalTorque()	Return value type: double; unit: Nm Returns the maximum motor torque (output side) determined from the motor data.
getState()	Return value type: Enum of type BrakeState Returns the results of the brake test. (>>> 8.3.5.1 "Polling the results of the brake test" Page 122)

Method	Description
getTestedTorque()	Return value type: double; unit: Nm Returns the test holding torque with which the holding torque (output side) applied and measured during the brake test is compared.
getTimestamp()	Return value type: java.util.Date Returns the time at which the brake test was started.

8.3.5.1 Polling the results of the brake test

Description

The test results are polled via the BrakeTestResult method getState(). An enum of type BrakeState is returned; its values describe the possible test results.

The possible test results are assigned to specific log levels. The log level corresponding to the test result can be polled with getLogLevel().

Syntax

```
BrakeTestResult result = ...;
BrakeState state = result.getState();
LogLevel logLevel = state.getLogLevel();
```

Explanation of the syntax

Element	Description
result	Type: BrakeTestResult Variable for the return value of the static method evaluateResult(...) which provides the BrakeTest class for evaluation of the brake test. The return value contains the results of the brake test and further information concerning the brake test which can be polled via the variable.
state	Type: Enum of type BrakeState Variable for the return value of getState(). The return value contains the test results. (>>> "BrakeState" Page 122)
logLevel	Type: Enum of type LogLevel Variable for the return value of getLogLevel(). The return value contains the log level of the test results. ■ **LogLevel.Error**: The brake test could not be executed or has failed. ■ **LogLevel.Info**: The brake test has been executed successfully. ■ **LogLevel.Warning**: The holding torque to be tested has been reached, but problems occurred while the brake test was being carried out.

BrakeState

The enum of type BrakeState has the following values (with specification of the corresponding log level):

Value	Description
BrakeUntested	The brake test could not be executed or was aborted during execution due to faults. Log level: LogLevel.Error
BrakeUnknown	The brake test could not be executed because not enough torque could be generated (e.g. due to excessive friction). Log level: LogLevel.Error
BrakeError	The brake test has failed. The measured holding torque falls below the holding torque to be tested. The brake is defective. Log level: LogLevel.Error
BrakeWarning	The measured holding torque is less than 5% above the holding torque to be tested. The brake has reached the wear limit and will soon be identified as defective. Log level: LogLevel.Warning
BrakeMax Unknown	The holding torque to be tested has been reached, but the brake could not be tested against the maximum brake holding torque. Log level: LogLevel.Warning
BrakeExcessive	The measured holding torque is greater than the maximum brake holding torque. Stopping using the brake can cause damage to the machine. Log level: LogLevel.Warning
BrakeReady	The measured holding torque exceeds the holding torque to be tested by more than 5 %. The brake is fully operational. Log level: LogLevel.Info

Example

A brake test is executed for axis A2. If the brake test is aborted, this is indicated by a corresponding output signal. If the brake test is fully executed, a message containing the measured holding torque is generated and the test results are polled. Depending on whether the measured holding torque is too low, within the tolerance range or in the ideal range, a corresponding output is also set in each case.

```
private LBR exampleLBR_iiwa;
private BrakeTestIOGroup brakeTestIOs;
...
 public void run() {
 ...

   try {
      int indexA2 = 1;
      BrakeTest exampleBrakeTest = new BrakeTest(indexA2);

      IMotionContainer exampleBrakeTestMotionContainer =
         exampleLBR_iiwa.move(exampleBrakeTest);

      BrakeTestResult resultA2 = BrakeTest.evaluateResult(
         exampleBrakeTestMotionContainer);

      double measuredTorque =
         resultA2.getMeasuredBrakeHoldingTorque();
```

```
        getLogger().info("Measured torque for A2: " + measuredTorque);

        BrakeState state = resultA2.getState();
        if(state == BrakeState.BrakeError)
           brakeTestIOs.setA2_BrakeError(true);
        else if(state == BrakeState.BrakeWarning)
           brakeTestIOs.setA2_BrakeWarning(true);
        else if(state == BrakeState.BrakeReady)
           brakeTestIOs.setA2_BrakeOK(true);

     } catch (CommandInvalidException ex) {
        brakeTestIOs.setBrakeTest_Aborted(true);
        ex.printStackTrace();
     }
  }
```

8.4 Performing a brake test

NOTICE If a brake is identified as being defective and the drives are deactivated, the robot may sag.

Description

If the template for the brake test application is adopted unchanged, the torques measured during a motion sequence are first evaluated for each axis, and the maximum absolute torque for each axis is determined. The result of the evaluation is displayed on the smartHMI. The brakes are then tested one after the other, starting with axis A1. The brake test results are displayed individually for each axis on the smartHMI.

Precondition

- No persons or objects are present within the motion range of the robot.
- Program run mode **Continuous** (standard mode)
- The robot is at operating temperature.

Procedure

- Select and start the brake test application.

If the brake test application is paused while a brake is being tested (e.g. by pressing the Start button on the smartPAD or due to a stop request), the brake test is aborted.
If the brake test application is resumed, the aborted brake test will be repeated for the axis in question. If the axis is no longer in the position in which the aborted brake test was started, it must be repositioned before the application can be resumed. This is done by pressing the Start key.

8.4.1 Evaluation results of the maximum absolute torques (display)

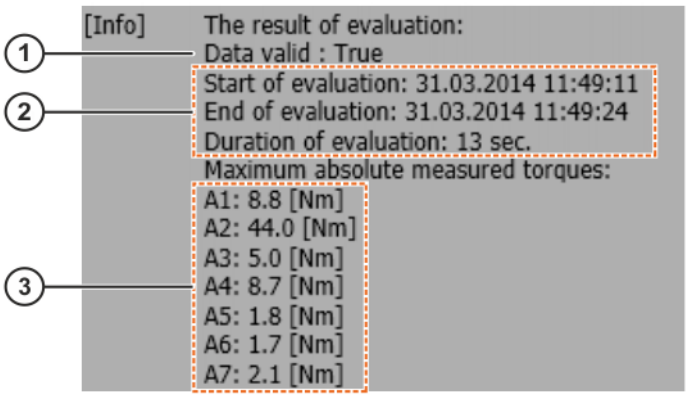


Fig. 8-3: Results of an evaluation of the maximum absolute torques

Item	Description
1	Validity Indicates whether the determined data are valid. The data are valid if no errors occur during command processing.
2	Time indications Start time, end time and overall duration of the evaluation.
3	Determined data The maximum absolute torque determined from the evaluation is displayed for each axis.

8.4.2 Results of the brake test (display)

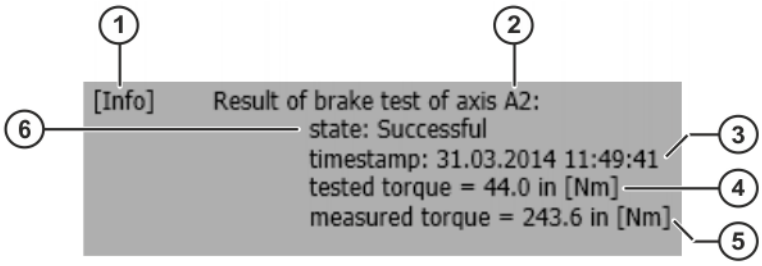


Fig. 8-4: Results of a brake test for axis A2

Item	Description
1	Log level Depending on the results of the brake test, the message is generated with a specific log level. ■ **Info**: The brake test has been executed successfully. ■ **Warning**: The holding torque to be tested has been reached, but problems occurred while the brake test was being carried out (see item 6 for descriptions of the possible test results). ■ **Error**: The brake test could not be executed or has failed.
2	Tested axis
3	Time stamp Time stamp at which the brake test was started for the axis.
4	Holding torque to be tested

Item	Description
5	Measured holding torque
6	Result of the brake test ■ **Untested**: The brake test could not be executed or was aborted during execution due to faults. ■ **Unknown**: The brake test could not be executed because not enough torque could be generated (e.g. due to excessive friction). ■ **Failed**: The brake test has failed. The measured holding torque falls below the holding torque to be tested. The brake is defective. ■ **Warning**: The measured holding torque is less than 5% above the holding torque to be tested. The brake has reached the wear limit and will soon be identified as defective. ■ **Maximum unknown**: The holding torque to be tested has been reached, but the brake could not be tested against the maximum brake holding torque. ■ **Excessive**: The measured holding torque is greater than the maximum brake holding torque. Stopping using the brake can cause damage to the machine. ■ **Successful**: The measured holding torque exceeds the holding torque to be tested by more than 5 %. The brake is fully operational.

⚠ WARNING If the functional capability of a brake is not guaranteed, the robot can sag. If the brake test shows that the brake on at least one axis of the LBR iiwa cannot apply the desired holding torque, the robot must be taken out of operation immediately.

9 Project management

9.1 Sunrise projects – overview

A Sunrise project contains all the data which are required for the operation of a station. A Sunrise project comprises:

- Station configuration

  The station configuration describes the static properties of the station. Examples include hardware and software components.
- Applications

  Applications contain the source code for executing a task for the station. They are programmed in Java with KUKA Sunrise.Workbench and are executed on the robot controller. A Sunrise project can have any number of applications.
- Runtime data

  Runtime data are all the data which are used by the applications during the runtime. These include, for example, end points for motions, tool data and process parameters.
- Safety configuration

  The safety configuration contains the configured safety functions.
- I/O configuration (optional)

  The I/O configuration contains the inputs/outputs of the used field buses mapped in WorkVisual. The inputs/outputs can be used in the application.

Sunrise projects are created and managed with KUKA Sunrise.Workbench.

(>>> 5.3 "Creating a Sunrise project with a template" Page 48)

There may only be 1 Sunrise project on the robot controller. This is transferred from Sunrise.Workbench to the robot controller by means of project synchronization.

(>>> 9.4 "Synchronization of projects " Page 143)

9.2 Frame management

Overview

Frames are coordinate transformations which describe the position of points in space or objects in a station. The coordinate transformations are arranged hierarchically in a tree structure. In this hierarchy, each frame has a higher-level parent frame with which it is linked through the transformation.

The root element or origin of the transformation is the world coordinate system which by default is located at the robot base. This means that all frames are directly or indirectly related to the world coordinate system.

A transformation describes the relative position of 2 coordinate systems to each other, i.e. how a frame is offset and oriented with respect to its parent frame.

The position of a frame with respect to its parent frame is defined by the following transformation data:

- X, Y, Z: offset of the origin along the axes of the parent frame
- A, B, C: Rotational offset of the axis angles of the parent frame

  Rotational angle of the frames:

  - Angle A: rotation about the Z axis
  - Angle B: rotation about the Y axis
  - Angle C: rotation about the X axis

9.2.1 Creating a new frame

Description

In Sunrise.Workbench, created frames are project-specific and can be used in every robot application of the project.

Once the project has been synchronized, the frames are available on the smartHMI. On the smartHMI, the frames can be taught in order to determine the position of the frames in space. Taught frames can be addressed manually.

(>>> 6.13 "Teaching and manually addressing frames" Page 80)

Procedure

1. Select the project in the **Package Explorer**.
2. Right-click on the desired parent frame in the **Application data** view and select **Insert new frame** from the context menu. The new frame is created and inserted in the frame tree as a child element of the parent frame.
3. The system automatically generates a frame name. It is advisable to change the name in the **Properties** view.

   A descriptive frame name makes both programming and orientation within the program easier. The frame names must be unique within the hierarchy level and may not be assigned more than once.

Example

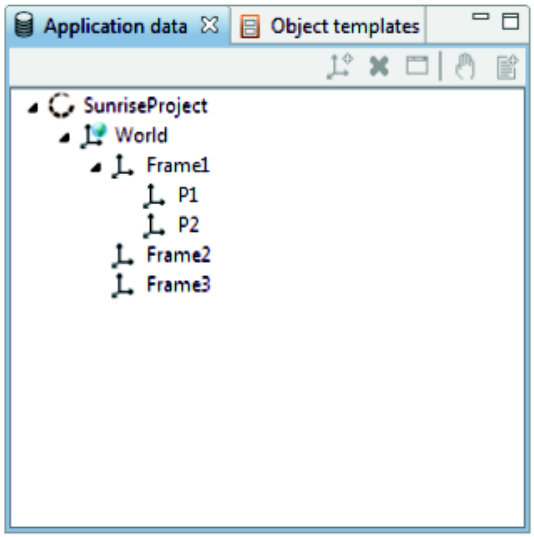


Fig. 9-1: Application data – frames

Frame1, 2 and 3 are child elements of World and are located on the same hierarchical level. P1 and P2 are child elements of Frame1 and are located one level below it.

9.2.2 Designating a frame as a base

Description

Frames can be marked as a base in the **Application data** view.

Only frames marked in this way can be selected and calibrated on the smartHMI as a base for jogging after synchronization of the project.

(>>> 6.8.1 "“Jogging options” window" Page 74)

(>>> 7.2.2 "Calibrating the base: 3-point method " Page 103)

> It is advisable to assign clear names to these frames to make it easier for the operator to select the jogging base on the smartPAD.

Procedure

- Right-click on the desired frame and select **Base** from the context menu.
  Alternative:
  Select the frame and click on the **Base** hand icon.
  The frame is marked with a hand icon.

Example

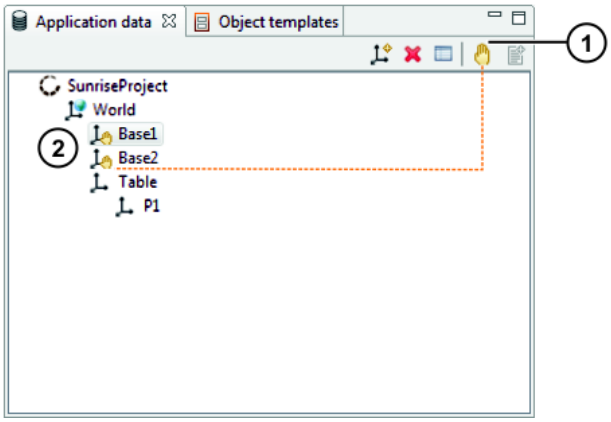


Fig. 9-2: Designating a frame as a base

1 **Base** hand icon
2 Base1 and Base2 are designated as the base

9.2.3 Moving frames

Description

A frame can be moved in the **Application data** view and assigned to a new parent frame. The following points must be taken into consideration:

- The subordinate frames are automatically moved at the same time.
- The absolute position of the moved frames in space is retained. The relative transformation of the frames to the new parent frame is adapted.
- Frames cannot be inserted under one of their child elements.
- The names of the direct child elements of a frame must be unique.

> If a frame is moved, its path changes. Since frames are used via this path in the source code of applications, the path specification must be corrected accordingly in the applications.

Procedure

1. Click on the desired frame and hold down the left mouse button.
2. Drag the frame to the new parent frame with the mouse.
3. When the desired new parent frame is selected, release the mouse button.

9.2.4 Deleting frames

Description

Frames can be removed from the frame tree in the **Application data** view. If a frame has child elements, the following options are available:

- **Move children to parent**: Only the selected frame is deleted. The subordinate frames are retained, are moved up a level and assigned to a new parent frame.
  The absolute position of the moved frames in space is retained. The relative transformation of the frames to the new parent frame is adapted.

> If a frame is moved, its path changes. Since frames are used via this path in the source code of applications, the path specification must be corrected accordingly in the applications.

- **Delete parent and child frames**: Deletes the selected frame and all subordinate frames.

Procedure

1. Right-click on the frame to be deleted and select **Delete** from the context menu. A frame without child elements is deleted immediately.
2. If the frame has child elements, the system asks whether these should also be deleted. Select the desired option.
3. Only with the **Move children to parent** option: if a name conflict occurs when moving the child elements, a notification message appears and the delete operation is canceled.

   Remedy: Rename one of the frames in question and repeat the delete operation.

9.2.5 Displaying and modifying the properties of a frame

The position and orientation of a frame is generally defined during teaching with the robot. However, it is also possible to enter the position values of a frame manually or to change them at a later stage. The following points must be taken into consideration:

- Modifying the transformation data not only moves the current frame but also all of its subordinate child elements, and it applies to all applications in which these frames are used.
- The taught values of Status, Turn and redundancy angle are retained. Under certain circumstances, it may no longer be possible to address the frame or its child elements.
- After a modification to the transformation data, all programs in which the frame is used must be tested in Manual Reduced Velocity mode (T1).

Description

Frames which are selected as the base can be calibrated with the robot.

(>>> 7.2.2 "Calibrating the base: 3-point method " Page 103)

If the data of a calibrated base are saved in Sunrise.Workbench by means of synchronization, the transformation data of the frame change in accordance with the calibration. The transformation data of the child elements of the frame are not changed by the calibration, i.e. only the position of the frame relative to the world coordinate system changes. The redundancy information also remains unchanged.

Procedure

1. Select the frame in the **Application data** view. The properties of the frame are displayed in the **Properties** view.
2. If required, display the properties by category using the (**Display categories**) button.
3. Enter the new value in the **Value** box of the property and confirm with the Enter key.

For physical variables, the value can be entered with the unit. If this is compatible with the preset unit, the value is converted accordingly (e.g. cm into mm or ° into rad). If no unit is entered, the preset unit is used.

Overview

The properties, arranged by categories, contain the following information:

Category	Description
General	General information ■ **Name**: Name of the frame ■ **Comment**: Optional ■ **Project**: Corresponding project ■ **Last modification**: Date and time of the last modification
Transformation	Transformation data ■ **X, Y, Z**: Translational offset of the frame relative to its parent frame ■ **A, B, C**: Rotational offset of the frame relative to its parent frame
Redundancy	Redundancy information ■ **E1**: Value of the redundancy angle ■ **Status** ■ **Turn**
Teach information	Information about the taught frame ■ **Device**: Robot used ■ **Tool**: Tool used ■ **TCP**: Frame path to TCP used ■ **X, Y, Z**: Translational offset of the TCP relative to the origin frame of the tool ■ **A, B, C**: Rotational offset of the TCP relative to the origin frame of the tool
Measurement	Information about the base calibration ■ **Measurement method**: Method used ■ **Last modification**: Date and time of the last modification **Note**: If the transformation data of a calibrated base are edited or if the frame is retaught, the calibration information are deleted.

9.2.6 Inserting a frame in a motion instruction

Description

A frame created in the application data can be inserted as the end point in a motion instruction.

Procedure

1. Program the motion instruction, e.g. robot.move(ptp()… .
2. In the **Application data** view, click on the frame which is to be used as the end point and hold down the left mouse button.
3. Drag the frame to the editor area with the mouse and position it so that the mouse pointer is between the brackets of the motion.
4. Release the mouse button. The frame is inserted as the end point of the motion.

Example

```
robot.move(ptp(getApplicationData().getFrame("/P2/Target")));
```

The **getApplicationData().getFrame()** method indicates that a frame created in the application data has been inserted. The end point of the motion is the **Target** frame.

As the transfer parameter, the method receives the path of the frame in the frame tree. The **Target** frame is a child element of **P2**.

9.3 Object management

Tools and workpieces are created and managed in Sunrise.Workbench. They belong to the runtime data of a project.

Tools

Properties:

- Tools are mounted on the robot flange.
- Tools can be used as movable objects in the robot application.
- The tool load data affect the robot motions.
- Tools can have any number of working points (TCPs) which are defined as frames.

Workpieces

Properties:

- Workpieces can be a wide range of objects which are used, processed or moved in the course of a robot application.
- Workpieces can be coupled to tools or other workpieces.
- Workpieces can be used as movable objects in the robot application.
- The workpiece load data affect the robot motions, e.g. when a gripper grips the workpiece.
- Workpieces can have any number of frames which mark relevant points, e.g. points on which a gripper grips a workpiece.

9.3.1 Geometric structure of tools

Every tool has an origin frame (root). By default, the origin of the tool is defined to match the flange center point in position and orientation when the tool is mounted on the robot flange. The origin frame is always present and does not have to be created separately.

A tool can have any number of working points (TCPs), which are defined relative to the origin frame of the workpiece (root) or to one of its child elements.

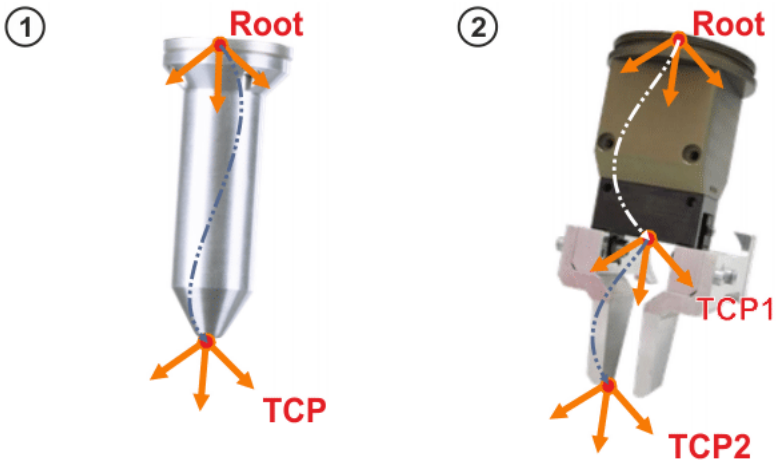


Fig. 9-3: Examples of TCPs of tools

1 Guiding tool with 1 TCP
2 Gripper with 2 TCPs

> The transformation of the frames is static. For active tools, e.g. grippers, this means that the TCP does not adapt to the current position of jaws or fingers.

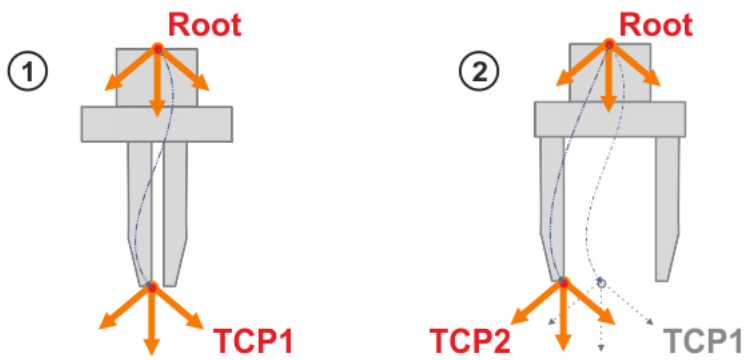


Fig. 9-4: Static TCP on a gripper

1 Gripper closed 2 Gripper open

9.3.2 Geometric structure of workpieces

Every workpiece has an origin frame (root). The origin frame is always present and does not have to be created separately.

A workpiece can have any number of frames, which are defined relative to the origin frame of the workpiece (root) or to one of its child elements.

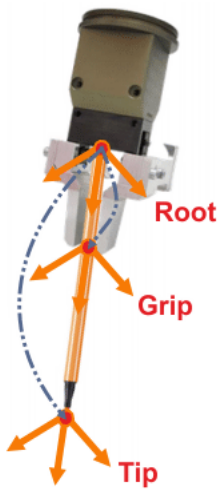


Fig. 9-5: Examples of frames of workpieces

9.3.3 Creating a tool or workpiece

Description

In Sunrise.Workbench, created tools and workpieces are project-specific and can be used in every robot application of the project.

The created tools can be selected in the jogging options on the smartHMI after the project is synchronized.

(>>> 6.8.1 "“Jogging options” window" Page 74)

Procedure

1. Select the project in the **Package Explorer**.
2. In the **Object templates** view, open the list of object templates.
3. To create a tool, right-click on the object type **Template Group for Tools** and select **Insert new tool** from the context menu. The object template for the tool is created.

4. To create a workpiece, right-click on the object type **Template Group for Workpieces** and select **Insert new workpiece** from the context menu. The object template for the workpiece is created.
5. The system automatically generates an object name. It is advisable to change the name in the **Properties** view.

   The object names must be unique. A descriptive name makes both programming and orientation within the program easier.
6. Enter the load data in the **Properties** view.

   (>>> 9.3.6 "Load data" Page 136)

9.3.4 Creating a frame for a tool or workpiece

Description

Each frame created for a tool or workpiece can be programmed in the robot application as the reference point for motions.

After the project is synchronized, the frames of a tool can be selected as the TCP for Cartesian jogging on the smartHMI.

(>>> 6.8.1 "“Jogging options” window" Page 74)

The frames of a tool (TCPs) can be calibrated with robot relative to the flange coordinate system.

(>>> 7.2.1 "Tool calibration" Page 98)

If the data of a calibrated tool are saved in Sunrise.Workbench by means of synchronization, the transformation data of the frame change in accordance with the calibration.

> The tool data of the TCP used to execute a Cartesian motion influence the robot velocity. Incorrectly entered tool data can cause unexpectedly high Cartesian velocities at the installed tool. The velocity of 250 mm/s may be exceeded in T1 mode.

Procedure

1. Select the project in the **Package Explorer**.
2. In the **Object templates** view, open the list of object templates.
3. Right-click on the object template and select **Insert new frame** from the context menu. The frame is created.

   In the upper hierarchy level, the parent frame of the created frame is the origin frame of the object.
4. To insert a new frame under an existing frame of the object, right-click on this parent frame and select **Insert new frame** from the context menu. The frame is created.
5. The system automatically generates a frame name. It is advisable to change the name in the **Properties** view.

   A descriptive frame name makes both programming and orientation within the program easier. The frame names must be unique within the hierarchy level and may not be assigned more than once.
6. In the **Properties** view, enter the transformation data of the frame with respect to its parent frame:
   - Boxes **X**, **Y**, **Z**: offset of the frame along the axes of the parent frame
   - Boxes **A**, **B**, **C**: orientation of the frame with reference to the parent frame

Properties

The properties, arranged by categories, contain the following information:

Category	Description

General	General information
	■ **Name**: Name of the frame
	■ **Comment**: Optional
	■ **Safety-oriented**: Only relevant for safety-oriented tool
	  ■ **Yes**: Frame is safety-oriented frame
	  ■ **No**: Frame is not a safety-oriented frame
	(>>> 9.3.7 "Safety-oriented tool" Page 137)

Geometry	Radius of the sphere on the safety-oriented frame ■ **Radius**
Transformation	Transformation data ■ **X, Y, Z**: Translational offset of the frame relative to its parent frame ■ **A, B, C**: Rotational offset of the frame relative to its parent frame
Measurement	Information on tool calibration ■ **Measurement method**: Method used ■ **Calculation error**: Translational calculation error which specifies the quality of the calibration (units: mm) ■ **Last modification**: Date and time of the last modification **Note**: If the transformation data of a calibrated tool are edited, the calibration information are deleted.

9.3.5 Defining a default motion frame

Description

If a tool or workpiece has a frame with which a large part of the motions must be executed, this frame can be defined as the default frame for motions.

Defining an appropriate default frame for a tool or workpiece simplifies the motion programming.

(>>> 15.11.4 "Moving tools and workpieces" Page 294)

If no default frame is defined, the origin frame of the tool or workpiece is automatically used as the default frame for motions.

Procedure

1. Select the project in the **Package Explorer**.
2. In the **Object templates** view, select the object type **Tools** or **Workpieces**.
3. Select the desired tool or workpiece.
4. Right-click on the desired frame and select **Default frame for motions** from the context menu.

   Alternative:

   Select the frame and click on the **Default frame for motions** icon.

   The frame is marked as the default motion frame.

Example

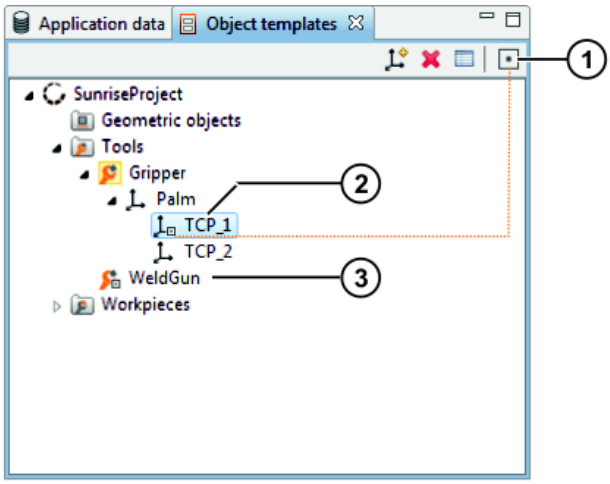


Fig. 9-6: Default frame for motions

1 **Default frame for motions** icon
2 Default frame of the **Gripper** tool: TCP_1
3 Default frame of the **WeldGun** tool: origin frame

9.3.6 Load data

Load data are all loads mounted on or connected to the robot flange. They form an additional mass mounted on the robot which must also be moved together with the robot.

The load data of tools and workpieces must be specified when the corresponding object templates are created. If several tools and workpieces are connected to the robot, the resulting total load is automatically calculated from the individual load data.

The load data are integrated into the calculation of the paths and accelerations. Correct load data are an important preconditon for the optimal functioning of the servo control and help to optimize the cycle times.

⚠ WARNING The robot must not be operated with incorrect load data or unsuitable loads. Failure to observe this precaution may result in severe injuries or considerable damage to property, e.g. because braking the robot takes too long due to incorrect load data.

Sources

Load data can be obtained from the following sources:

- Manufacturer information
- Manual calculation
- CAD programs
- The load data of tools can be determined automatically.
  (>>> 7.3 "Determining tool load data" Page 105)

9.3.6.1 Entering load data

Procedure

1. Select the Sunrise project in the **Package Explorer**.
2. In the **Object templates** view, open the list of object templates.
3. Select the desired tool or workpiece.
4. In the **Properties** view, enter the load data:
   - **Mass**: Mass of the object (tool or workpiece)

- Boxes **MS X**, **MS Y**, **MS Z**: Position of the center of mass relative to the origin frame of the object
- Boxes **MS A**, **MS B**, **MS C**: Orientation of the principal inertia axes relative to the origin frame of the object. The principal inertia axes run through the center of mass.
- Boxes **jX**, **jY**, **jZ**: Principal moments of inertia

Example

Principal moment of inertia **jX**:

jX is the inertia about the X axis of the principal inertia axes. This is rotated through **MS A**, **MS B** and **MS C** relative to the origin frame of the object and shifted in the center of mass.

jY and **jZ** are the analogous principal moments of inertia about the Y and Z axes.

9.3.7 Safety-oriented tool

Description

A maximum of 1 safety-oriented tool may be defined in a Sunrise project and modeled with up to 6 configured spheres. A safety-oriented tool is coupled, by default, to all kinematic systems available in the project.

The properties of the safety-oriented tool are relevant for the following configurable safety functions:

- Monitoring of Cartesian spaces

  The spheres can be monitored against the limits of activated Cartesian monitoring spaces.

  (>>> 13.8.9 "Monitoring spaces" Page 204)
- Monitoring of the translational Cartesian velocity

  The velocity of the sphere center points is monitored.

  (>>> 13.8.8 "Velocity monitoring functions" Page 197)
- Collision detection and TCP force monitoring

  Only correctly specified load data ensure the accuracy of these monitoring functions. The load data of the safety-oriented tool, in particular the mass and center of mass of the tool, must be configured. In the case of tools with comparatively high moments of inertia (> 0.1 kg m^2), these data must also be specified in order to ensure the accuracy of these monitoring functions.

The safety-oriented tool is transferred to the robot controller by means of synchronization and activated once the robot controller has been rebooted, i.e. it is permanently active for the safety controller, irrespective of which tool is used in the application or set in the jogging options.

Safety-oriented frames

Just like any other tool, a safety-oriented tool can have any number of frames. In order to configure the monitoring spheres, suitable frames must be defined as safety-oriented frames. The center of the sphere is situated, by definition, at the origin of the safety-oriented frame. The radius of the sphere is defined in the frame properties.

If workpieces are used that are to be taken into consideration for safety-oriented Cartesian space or velocity monitoring, e.g. due to the dimensions of the workpieces, the spheres of the safety-oriented tool must be configured accordingly.

Safety-oriented frames are also relevant for the following configurable, tool-specific safety monitoring functions:

- Monitoring of the tool orientation (only available for robots)

  One of the safety-oriented frames can be defined as the tool orientation frame. Safety-oriented monitoring of the orientation of this frame can be carried out with the AMF *Tool orientation*.

  (>>> 13.8.10 "Monitoring the tool orientation" Page 212)
- Direction-specific monitoring of the Cartesian velocity (available for robots and mobile platforms)

  One of the safety-oriented frames can be defined as the monitoring point for the tool-specific velocity monitoring. A second frame can additionally be defined as the orientation for the monitoring. This orientation frame defines the orientation of the coordinate system in which the velocity of the monitoring point is described. In tool-specific velocity monitoring, a component of this velocity can be monitored.

  (>>> 13.8.8.3 "Direction-specific monitoring of Cartesian velocity" Page 199)

Example

For a safety-oriented gripper, 3 monitoring spheres are configured.

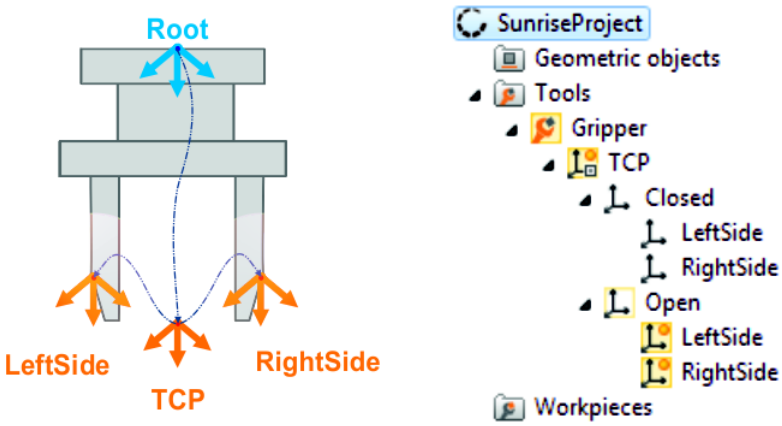


Fig. 9-7: Safety-oriented gripper

9.3.7.1 Defining a safety-oriented tool

Precondition

- Tool and corresponding frames have been created.
- When using the AMFs *Collision detection* and *TCP force monitoring*: The correct load data of the tool, in particular the mass and center of mass of the tool, are known.

Procedure

1. Select the project in the **Package Explorer**.
2. In the **Object templates** view, select the tool that is to be safety-oriented.
3. Define the properties of the safety-oriented tool in the **Properties** view.
   - If they have not yet been defined, enter the load data of the tool.

     Tools with load data outside the specified range of values cannot be used as safety-oriented tools.

     (>>> "Load data" Page 139)
   - Set the property **Safety-oriented** to **Yes**.

   The tool icon in the **Object templates** view is highlighted in yellow.

   The properties of the tool are expanded to include the safety parameters required for the configuration of tool-specific monitoring functions.

   (>>> "Safety parameters" Page 140)
4. In the **Object templates** view, select the frame that is to be safety-oriented.
5. Define the properties of the safety-oriented frame in the **Properties** view.

- Enter the radius of the monitoring sphere.
- If they have not yet been defined, enter the transformation data of the frame with respect to its parent frame.

  Frames with transformation data outside the specified range of values cannot be used as safety-oriented frames.

  (>>> "Frames" Page 139)
- Set the property **Safety-oriented** to **Yes**.

The frame icon in the **Object templates** view is highlighted in yellow and marked with a sphere symbol.

6. Repeat steps 4 to 5 to define further safety-oriented frames.
7. If required, set the safety parameters that are necessary for tool-specific safety monitoring functions:
   a. Select the safety-oriented tool in the **Object templates** view.
   b. In the **Properties** view, assign a safety-oriented frame to the desired safety parameters.

      (>>> "Safety parameters" Page 140)

   The frame icon in the **Object templates** view is expanded by an arrow.

> Alternative procedure for marking a tool or frame as safety-oriented:
> - Right-click on the tool or frame in the **Object templates** view and select **Safety-oriented** from the context menu.

> Alternative procedure for defining a safety-oriented frame as a safety parameter for tool-specific safety monitoring functions:
> - Right-click on the desired safety-oriented frame in the **Object templates** view and select the desired safety parameter from the context menu.
> - Select the safety-oriented frame in the **Object templates** view and set the desired safety parameter to **Yes** in the **Properties** view.

Load data

Load data of the safety-oriented tool:

Parameter	Description
Mass	Mass of the safety-oriented tool ■ **≤2,000 kg**
MS X, *MS Y*, *MS Z*	Position of the center of mass relative to the origin frame of the safety-oriented tool ■ **-10,000 mm … +10,000 mm**
MS A, *MS B*, *MS C*	Orientation of the principal inertia axes relative to the origin frame of the safety-oriented tool ■ Any
jX, *jY*, *jZ*	Mass moments of inertia of the safety-oriented tool ■ **0 kg·m^2… 1,000 kg·m^2**

> To avoid spending unnecessary time performing verifications, mark a tool as safety-oriented only when the load data have been correctly entered or determined and have been transferred to Sunrise.Workbench.

Further information on the load data can be found here: (>>> 9.3.6 "Load data" Page 136)

Frames

Properties of safety-oriented frames:

Parameter	Description
Radius	Radius of the sphere on the safety-oriented frame ■ **25 mm … 10,000 mm**
X, **Y**, **Z**	Offset of the safety-oriented frame along the axes of the parent frame ■ **-10,000 mm … +10,000 mm**
A, **B**, **C**	Orientation of the safety-oriented frame relative to the parent frame ■ Any

Safety parameters

Safety parameters of the safety-oriented tool:

Parameter	Description
Tool orientation frame	Safety-oriented frame, the orientation of which can be monitored using the AMF *Tool orientation*. If no tool orientation frame is set, the origin frame of the tool (= flange coordinate system of the robot) is used as the tool orientation frame.
Point for tool-related velocity	Safety-oriented frame defining a point on the tool at which the Cartesian velocity in a specific direction can be monitored using the AMF *Tool-related velocity component*. If no point is defined for the tool-specific velocity, the origin frame of the tool (= flange coordinate system of the robot) is used, i.e. the velocity at the origin of the flange coordinate system is monitored. **Note:** In the case of a mobile platform, the origin frame of the tool corresponds to the coordinate system at the center point of the platform.
Orientation for tool-related velocity	Safety-oriented frame, the orientation of which determines the directions in which the Cartesian velocity can be monitored using the AMF *Tool-related velocity component*. If no orientation is defined for the tool-specific velocity, the origin frame of the tool (= flange coordinate system of the robot) is used, i.e. the orientation of the flange coordinate system determines the monitoring direction. **Note:** In the case of a mobile platform, the origin frame of the tool corresponds to the coordinate system at the center point of the platform.

9.3.8 Safety-oriented workpieces

Description

Loads picked up by the robot, e.g. a gripped workpiece, exert an additional force on the robot and influence the torques measured by the joint torque sensors. For this reason, the load data of workpieces which are picked up must be taken into consideration in the safe monitoring of collisions and forces. For this, these workpieces must be configured as safety-oriented workpieces.

A maximum of 8 safety-oriented workpieces can be configured for a Sunrise project.

The properties of safety-oriented workpieces are relevant for the following configurable safety functions:

- Collision detection

  The load data of the active safety-oriented workpiece are taken into consideration when the external torque is calculated.

  (>>> 13.8.13.2 "Collision detection" Page 217)

- TCP force monitoring

  The load data of the heaviest safety-oriented workpiece are taken into consideration when the Cartesian force exerted beneath the flange is determined.

  (>>> 13.8.13.3 "TCP force monitoring" Page 218)

During a process, picking up and setting down different workpieces can result in load changes. During TCP force monitoring, the safety controller automatically uses the load data of the heaviest safety-oriented workpiece. During collision detection, the user must explicitly inform the safety controller which safety-oriented workpiece is currently activated.

(>>> 15.11.6 "Commanding load changes to the safety controller" Page 298)

If a safety-oriented workpiece is activated, the safety controller permanently takes its load data into consideration. If this workpiece is to be deactivated or if a different safety-oriented workpiece is to be activated, an explicit command is required. No safety-oriented workpiece is activated after the robot controller is rebooted.

Safety-oriented workpieces are not activated in the source code of robot applications and background tasks in a safety-oriented way. This is why, in the event of an error, collision detection and TCP force monitoring may use load data which deviate from the actual workpiece load. These deviations are misinterpreted as an external torque or external TCP force. At low velocities and accelerations, the maximum deviation corresponds to the weight of the heaviest workpiece which can be picked up in the application.

When using the AMFs *Collision detection* and *TCP force monitoring*, it is advisable to configure as safety-oriented workpieces all those workpieces picked up by the robot while one of the monitoring functions is active.

When using the AMF *TCP force monitoring*, the heaviest workpiece picked up by the robot while the monitoring function is active must be configured as a safety-oriented workpiece. Incorrect configuration of the heaviest workpiece can cause the safety integrity of the TCP force monitoring function to be lost.

The way in which the load data of a workpiece influences collision monitoring depends on how the workpiece is picked up. For a safety-oriented workpiece, the safety controller requires that the origin frame of the safety-oriented workpiece is identical to the standard frame for motions of the safety-oriented workpiece.

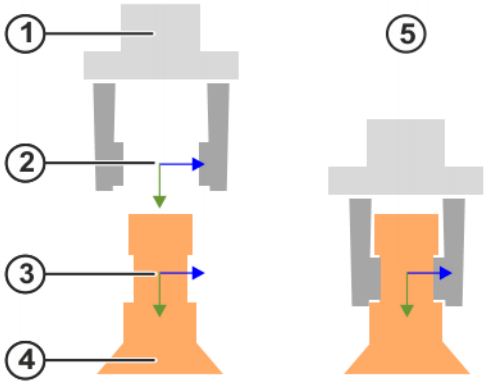


Fig. 9-8: Configuring a safety-oriented workpiece

Item	Description
1	Safety-oriented tool
2	Standard frame for motions of the safety-oriented tool: Frame of the safety-oriented tool on which the safety-oriented workpiece must be picked up. It is not necessary for this frame to be a safety-oriented frame.
3	Origin frame of the safety-oriented workpiece Frame of the safety-oriented workpiece on which the safety-oriented tool must pick up the workpiece.
4	Safety-oriented workpiece
5	State after activation of the safety-oriented workpiece The origin frame of the safety-oriented workpiece is identical to the standard frame for motions of the safety-oriented tool.

9.3.8.1 Defining safety-oriented workpieces

Precondition

- Workpiece has been created.
- When using the AMFs *Collision detection* and *TCP force monitoring*: The correct load data of the workpiece, in particular the mass and center of mass of the workpiece, are known.

Procedure

1. Select the project in the **Package Explorer**.
2. In the **Object templates** view, select the workpiece that is to be safety-oriented.
3. Define the properties of the safety-oriented workpiece in the **Properties** view.
   - If they have not yet been defined, enter the load data of the workpiece.
     Workpieces with load data outside the specified range of values cannot be used as safety-oriented workpieces.
     (>>> "Load data" Page 143)
   - Set the property **Safety-oriented** to **Yes**.

   The tool icon in the **Object templates** view is highlighted in yellow.

Once the project is synchronized, the safety-oriented workpiece can be activated in the application.

> Alternative procedure for marking a workpiece as safety-oriented:
> - Right-click on the workpiece in the **Object templates** view and select **Safety-oriented** from the context menu.

Load data

Load data of the safety-oriented workpiece:

Parameter	Description
Mass	Mass of the safety-oriented workpiece ■ **0.001 kg + 2,000 kg**
MS X, *MS Y*, *MS Z*	Position of the center of mass relative to the origin frame of the safety-oriented workpiece ■ **-10,000 mm … +10,000 mm**
MS A, *MS B*, *MS C*	Orientation of the principal inertia axes relative to the origin frame of the safety-oriented workpiece (not relevant for safety-oriented monitoring) ■ Any
jX, *jY*, *jZ*	Mass moments of inertia of the safety-oriented workpiece (not relevant for safety-oriented monitoring) ■ Any

Further information on the load data can be found here: (>>> 9.3.6 "Load data" Page 136)

9.4 Synchronization of projects

Overview

In the synchronization of projects, the project data are transferred between Sunrise.Workbench and the robot controller. In the process, the projects are compared with one another. If there are different projects or version conflicts, the user can choose the direction in which to transfer the project data. The following cases are distinguished:

- The project only exists in Sunrise.Workbench.
- The project on the robot controller is replaced by another project.
- There are different versions of the project:
  - When the project data is modified in Sunrise.Workbench only
  - When the project data is modified on the robot controller only
  - When the project data is modified on both sides

9.4.1 Transferring the project to the robot controller

Description

The procedure described here applies if no project is on the robot controller yet or if there is a different project from the one to be transferred.

Precondition

- Network connection to the robot controller
- The project to be transferred has at least one application.
- The system software is now installed.
  (>>> 10.2 "Installing the system software" Page 148)
- The installed system software is compatible with the station configuration of the project to be transferred.

Procedure

The procedure described here applies if no project is on the robot controller yet or if there is a different project from the one to be transferred.

1. Select the project to be transferred in the **Package Explorer**.
2. Click on the **Synchronize project** button in the toolbar. The system scans the controller for existing project data. If the scan fails, the cause of the error is displayed in a message.
3. If the scan is successful, the **Project synchronization** window opens.

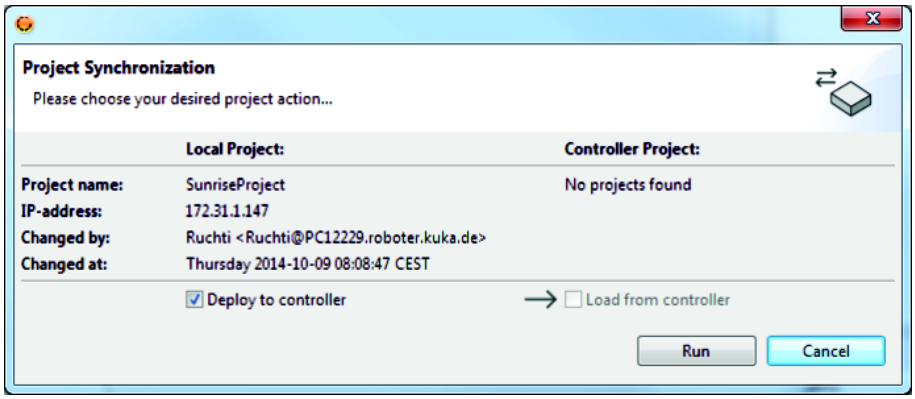


Fig. 9-9: Transferring the project to the controller

4. Click on **Execute**. The project is transferred to the robot controller.

   The progress is displayed in a dialog both in Sunrise.Workbench and on the smartPAD. When the transfer is completed, the dialog is automatically closed.
5. If the transfer fails, a corresponding dialog is displayed both in Sunrise.Workbench and on the smartPAD. In addition, the cause of the error is displayed in Sunrise.Workbench.

   Confirm the dialog in Sunrise.Workbench and on the smartPAD with **OK**.
6. If the transfer is successful, the robot controller must be rebooted when the safety configuration or I/O configuration is modified.

   Confirm the reboot prompt with **OK**. The robot controller is rebooted.
7. If the safety configuration is modified, activate this on the robot controller.

   (>>> 13.7 "Activating the safety configuration on the robot controller" Page 189)

The transferred project is now active on the controller. All created project data are available.

9.4.2 Updating the project on the robot controller or in Sunrise.Workbench

Description

The procedure described here applies if the same project exists on both sides but in different versions.

Precondition

- Network connection to the robot controller
- Only if a project is transferred to the robot controller: No application is running on the robot controller.

Procedure

1. Select the project to be transferred in the **Package Explorer**.
2. Click on the **Synchronize project** button in the toolbar. The system scans the controller for existing project data. If the scan fails, the cause of the error is displayed in a message.
3. If the project is identical to the project in Sunrise.Workbench, no synchronization is necessary. The process is then automatically aborted.

   If the scan is successful, the **Project synchronization** window opens.

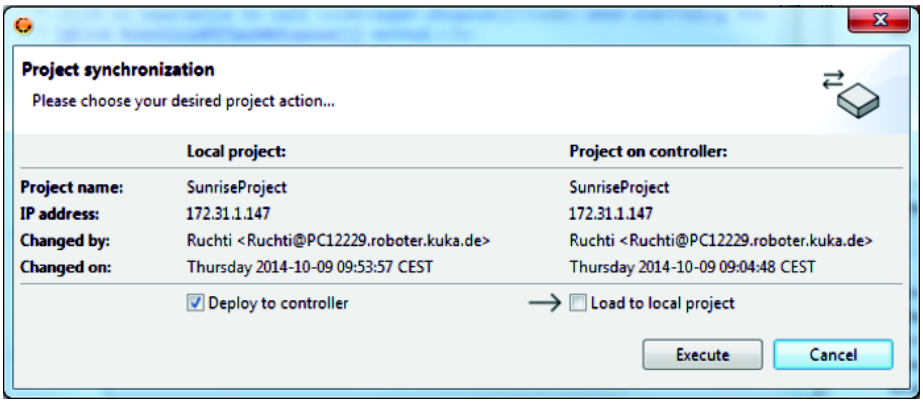


Fig. 9-10: Updating a project

Information regarding both projects is displayed here. The direction of synchronization is set by default to transfer the more current project version.

If modifications have been made to the project data on both sides, the system recognizes this as a conflict and displays a warning. The direction of synchronization can be set:

- Check mark set at **Deploy to controller**: The project is transferred from Sunrise.Workbench to the robot controller.
- Check mark set at **Load to local project**: The project is transferred from the robot controller to Sunrise.Workbench.

4. If required, change the direction of synchronization and click on **Execute**. The project is transferred. If the older version is to be transferred, a warning is displayed.

   The progress is displayed in a dialog both in Sunrise.Workbench and on the smartPAD. When the transfer is completed, the dialog is automatically closed.
5. If the transfer fails, a corresponding dialog is displayed both in Sunrise.Workbench and on the smartPAD. In addition, the cause of the error is displayed in Sunrise.Workbench.

   Confirm the dialog in Sunrise.Workbench and on the smartPAD with **OK**.
6. Only if transfer to the robot controller is successful:
   - If the safety configuration or I/O configuration is modified, the robot controller must be rebooted.

     Confirm the reboot prompt with **OK**. The robot controller is rebooted.
   - If the safety configuration is modified, activate this on the robot controller.

     (>>> 13.7 "Activating the safety configuration on the robot controller" Page 189)

9.5 Loading the project from the robot controller

Description

It is also possible to load a project from the robot controller if the project is not located in the workspace of Sunrise.Workbench.

Precondition

- Network connection to the robot controller
- The workspace does not contain any project with the name of the project to be loaded.

Procedure

1. Select the menu sequence **File** > **New** > **Sunrise project**. The project creation wizard opens.
2. Enter the IP address of the robot controller from which the project is to be loaded in the **IP address of controller** box.

> The IP address of the robot controller can be displayed on the smartHMI. (>>> 6.16.6 "Displaying the IP address and software version" Page 95)

3. Activate the **Load project from controller** radio button.
4. Click on **Next >**. The system checks whether there is a project on the controller.
5. If there is a project is on the controller and there is no project with the same name in the workspace, a summary of information on the project is displayed.

   Click on **Finish**. The project is created in the workspace and then added to the **Package Explorer**.

10 Station configuration and installation

10.1 Opening the station configuration

Procedure

- Double-click on the **StationSetup.cat** file in the **Package Explorer**.

The file contains the station configuration of the project. This can be edited and installed using the following tabs:

Topology

The **Topology** tab displays the hardware components of the station. The topology can be restructured or modified.

Software

The **Software** tab displays the software components of the station. The components to be installed as well as their version can be selected. The components available for selection depend on the specific topology.

Configuration

The **Configuration** tab displays the configuration of the robot controller. The configuration can be changed.

Installation

The system software is installed on the robot controller via the **Installation** tab.

10.1.1 Configuring parameters for calibration

Procedure

1. Open the station configuration and select the **Configuration** tab.
2. Configure the parameters as desired in the catalog element **SmartHMI** under **Measurement**.
3. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.

Overview

Parameter	Description
Minimum calibration point distance (tool) in mm	Minimum distance which must be maintained between 2 measuring points (XYZ 4-point and ABC 2-point methods) during tool calibration ■ **0 … 200** Default: 8
Maximum calculation error in mm	Maximum translational calculation error during tool calibration up to which the quality of the calibration is considered sufficient ■ **0 … 200** Default: 50
Minimum calibration point distance (base) in mm	Minimum distance which must be maintained between 2 measuring points during base calibration ■ **0 … 200** Default: 50
Minimum angle in °	Minimum angle to be maintained between the straight lines which are defined by the 3 measuring points during base calibration (3-point method) ■ **0 … 360** Default: 2.5

10.2 Installing the system software

Precondition

- The station configuration is completed.
- Network connection to the robot controller

Procedure

1. Select the **Installation** tab.
2. By default, the **Installation events** window displays the warnings and errors which occur during installation: The check mark is set at **Show only warnings and errors.**.

   To display all events which occur during installation, remove the check mark from **Show only warnings and errors.**.
3. Click on **Install**. The installation is prepared and the **Installation** window opens. The **Configured IP** box is marked in color:
   - Marked in green: The network connection to the robot controller has been established; installation is possible.
   - Marked in red: The network connection to the robot controller could not be established. Possible causes include: the network cable is not connected correctly or the configured IP address does not match the actual address.

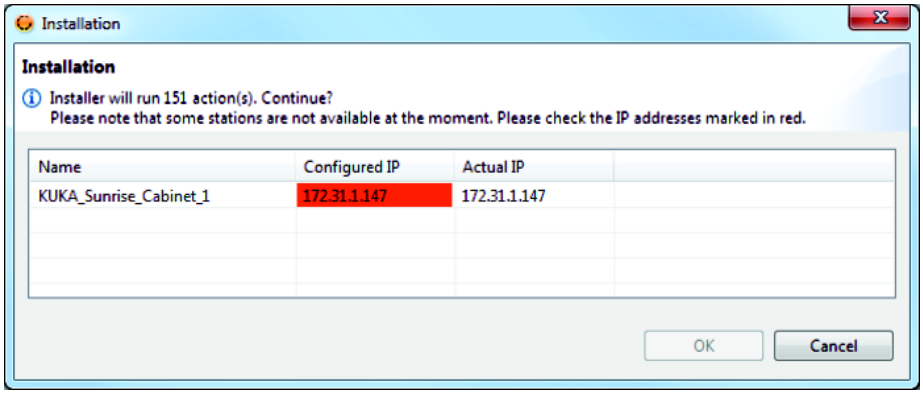


Fig. 10-1: The robot controller cannot be reached

4. Only if the **Configured IP** box is marked in red:
   - If the configured IP address of the robot controller matches the actual address, there is no network connection to the robot controller. Establish a network connection.
   - If the IP address of the robot controller is different from the configured address, enter the current IP address of the robot controller in the **Actual IP** box. To do this, double-click in the box.
5. To continue the installation, click on **OK**.
6. Only relevant if the IP address in the **Actual IP** box has been changed: If the red marking under **Configured IP** persists or the installation fails, there is no network connection to the robot controller with this IP address.

   Establish a network connection and restart the installation process (return to step 3).
7. Confirm the reboot prompt with **OK**. The robot controller is rebooted and the installation is completed.

> During installation, the safety configuration is transferred to the robot controller but is not yet activated. In order to be able to move the robot, the safety configuration must be activated via the smartHMI.
> (>>> 13.7 "Activating the safety configuration on the robot controller" Page 189)

> The robot type selected in the station configuration of the Sunrise project and the set media flange are saved in the configuration data on the robot controller during installation. If this information does not match the corresponding data on the electronic identification plate of the robot connected to the robot controller, the robot cannot be moved.

Installation

Reinstallation of the system software is required in the following cases:

- Change to the station configuration on the **Topology** tab
- Change to the station configuration on the **Software** tab

  Examples:

  - Installation of additional software packages
  - Software update
  - Incompatible version changes of existing software packages

    Incompatible version changes can occur if a project is not opened with the same version of Sunrise.Workbench with which it was installed.

    (>>> 15.6 "RoboticsAPI version information" Page 276)
- Change to the station configuration on the **Configuration** tab

10.2.1 Converting the safety configuration to a new software version

Description

If a new software version of Sunrise.Workbench is installed, a Sunrise project which was created with an earlier software version can be loaded to the workspace and continue to be used.

The station configuration changes when the Sunrise project is loaded. Saving the station configuration will transfer the corresponding safety configuration to the new version.

Precondition

- The Sunrise project is archived or saved in any directory.
- New version of Sunrise.Workbench is installed.

Procedure

1. Load the Sunrise project into the workspace.
2. Open the station configuration of the project and click on **Save**.
3. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.
4. The safety configuration is updated and its parameters are converted. When the operation is completed, this is indicated by a message. Confirm with **OK**.
5. Further steps are required in order to be able to use the updated project on the robot controller:
   a. Install the system software.
   b. Synchronize the project.
   c. Reactivate the safety configuration.
   d. Carry out safety acceptance.

10.3 Installing a language package

Description

The user interface on the smartHMI is currently available in the following languages:

German	Italian
English	Spanish
French	

Languages which are only available after software is delivered can be installed later if required.

Precondition

- The language package is available in Sunrise.Workbench.
  (>>> 4.4 "Installing the language package in Sunrise.Workbench" Page 44)

Procedure

1. Open the station configuration and select the **Software** tab.
2. Select the **SmartHMI LanguagePack** software package for installation:
   - Set the check mark in the **Install** column.
3. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.
4. Reinstall the system software on the robot controller. Once the robot controller has been rebooted, the newly installed languages are available on the smartHMI.

10.4 Installing or updating the virus scanner

Description

Installation is carried out in 2 parts:

- Installation of the virus scanner in KUKA Sunrise.Workbench
  The virus scanner is an option that must be installed subsequently in Sunrise.Workbench. Only after installation is the virus scanner available in the software catalog of Sunrise.Workbench (entry **Ikarus AntiVirus**).
- Installation of the system software on the robot controller
  Once the system software has been installed, the virus scanner is active on the robot controller. A tile is then available on the smartHMI that can be used to open the **Virus scanner** view.
  (>>> 6.16.8 "Displaying messages of the virus scanner" Page 95)

> It is advisable to check what version is currently installed on the robot controller before updating the virus scanner. Do not perform a downgrade.

Precondition

- Data storage medium with the software to be installed:
  - File **com.kuka.ikarus.repository-***[Version]***.ZIP**
- In the case of an update on the robot controller: Network connection without Internet access or with an active firewall
  During the update, the virus scanner is briefly inactive.

Procedure

1. Select the menu sequence **Help** > **Install new software...**. The **Install** window is opened.
2. To the right of the **Work with** box, click on **Add**. The **Add repository** window is opened.
3. Click on **Archive…** and navigate to the installation file:
   File **com.kuka.ikarus.repository-***[Version]***.ZIP**
4. Select the file and confirm with **Open**. The **Position** box now displays the path to the file.
5. Click on **OK**.
   - In the **Install** window, the **Work with** box now displays the path to the file.
   - The window now also displays the check box **KUKA.Ikarus T3**.
6. Activate the **KUKA.Ikarus T3** check box.
7. Leave the other settings in the **Install** window as they are and click on **Next >**.

8. The overview **Installation details** is displayed. Leave the settings as they are and click on **Next >**.
9. A license agreement is displayed. In order to be able to install the virus scanner, the agreement must be accepted. Then click on **Finish**.

   The installation operation begins.
10. A safety warning concerning unsigned contents is displayed. Confirm with **OK**.
11. Answer the request for confirmation with **OK**.
12. A message indicates that Sunrise.Workbench must be restarted in order to apply the changes. Click on **Restart now**.
13. Sunrise.Workbench restarts. This completes installation in Sunrise.Workbench.
14. Reinstall the system software on the robot controller. Once the robot controller has been rebooted, the virus scanner is active on the robot controller.

11 Bus configuration

11.1 Configuration and I/O mapping in WorkVisual – overview

Step	Description
1	Install the **Sunrise** option package in WorkVisual. The option package is available as a KOP file and is supplied together with Sunrise.Workbench. The file **Sunrise.kop** is located on the data storage medium with the software to be installed (delivered in the directory **WorkVisual AddOn**). **Note**: The option package supplied with Sunrise.Workbench must always be used. If an old version of Sunrise.Workbench is uninstalled and a new version installed, the option package must also be exchanged in WorkVisual.
2	Only necessary if a media flange “Touch” or FSoE slaves are used: 1. Copy the directories **Config** and **SafeDevDescr** (located on the data storage medium with the software to be installed; delivered in the directory **WorkVisual AddOn/ FSoE_Patch**). 2. Copy the directories **Config** and **SafeDevDescr** to the WorkVisual installation directory (the existing directories are overwritten).
3	Terminate WorkVisual and create a new I/O configuration in Sunrise.Workbench or open an existing I/O configuration. WorkVisual is started automatically and the WorkVisual project corresponding to the I/O configuration is opened. (>>> 11.3 "Creating a new I/O configuration" Page 154) (>>> 11.4 "Opening an existing I/O configuration" Page 154)
4	Only necessary if devices are used for which no device description files have yet been imported: 1. Close the WorkVisual project. 2. Import the required device description files. 3. Reopen the WorkVisual project.
5	Configure the field bus. (>>> 11.2 "Overview of field buses" Page 154)
6	Create the Sunrise I/Os and map them. (>>> 11.5 "Creating Sunrise I/Os" Page 155) (>>> 11.6.3 "Mapping Sunrise I/Os" Page 161)
7	Export the I/O configuration to the Sunrise project. (>>> 11.7 "Exporting the I/O configuration to the Sunrise project" Page 161)
8	Transfer the I/O configuration to the robot controller (Synchronize Project) and reboot the robot controller. (>>> 9.4 "Synchronization of projects " Page 143)

Information about installing and managing option packages can be found in the **WorkVisual** documentation.

Information about importing device description files and general information about configuring field buses can be found in the **WorkVisual** documentation.

11.2 Overview of field buses

The following field buses are supported by Sunrise and can be configured with WorkVisual:

Field bus	Description
PROFINET	An Ethernet-based field bus. Data exchange is carried out on a client-server basis. PROFINET is installed on the robot controller.
PROFIBUS	Universal field bus which enables communication between devices from different manufacturers without special interface adaptations. Data exchange is carried out on a master-slave basis.
EtherCAT	An Ethernet-based field bus suitable for real-time requirements.

For configuration of a field bus, the documentation of the field bus is required.

If the robot controller is used as a PROFINET master or device, hardware problems can result in an inability to access bus devices. In this case, use of a diagnostic tool, such as WorkVisual, Step 7 or WireShark, is recommended.

When connecting additional EtherCAT devices to a media flange with an EtherCAT output, e.g. media flange IO pneumatic, it must be ensured that the number of available signals on the bus is limited.
If there are too many connected devices, this can result in overloading of the bus and loss of communication. Under certain circumstances, the robot can then no longer be moved.

11.3 Creating a new I/O configuration

Precondition

- Sunrise project without I/O configuration

Procedure

1. Select the project in the **Package Explorer**.
2. Select the menu sequence **File** > **New** > **I/O configuration**.

   WorkVisual is started and the WorkVisual project corresponding to the I/O configuration is opened. The file **IOConfiguration.wvs** is inserted into the Sunrise project; this can be used to call the corresponding WorkVisual project.

11.4 Opening an existing I/O configuration

Precondition

- Sunrise project with I/O configuration

Procedure

1. Double-click on the file **IOConfiguration.wvs**. WorkVisual is started and the WorkVisual project corresponding to the I/O configuration is opened.
2. Right-click on the inactive robot controller on the **Hardware** tab in the **Project structure** window.

3. Select **Set as active controller** from the context menu. The **I/O Mapping** window opens. The Sunrise I/Os can be edited.

11.5 Creating Sunrise I/Os

Precondition

- Field bus configuration has been completed.
- The robot controller has been set as the active controller.

Procedure

1. Select an input or output module of the configured bus on the **Field buses** tab in the top right-hand corner of the **I/O Mapping** window.

   (>>> 11.6.1 "I/O Mapping window" Page 159)
2. Select the **Sunrise I/Os** tab in the top left-hand corner of the **I/O Mapping** window.
3. In the bottom left-hand corner of the **I/O Mapping** window, click on the **Creates signals at the provider** button. The **Create I/O signals** window is opened.

   (>>> 11.5.1 "“Create I/O signals” window" Page 156)
4. Create an I/O group and inputs/outputs within the group.

   (>>> 11.5.2 "Creating an I/O group and inputs/outputs within the group" Page 157)
5. Click on **OK**. The Sunrise I/Os are saved. The **Create I/O signals** window is closed.

The created I/O group is displayed on the **Sunrise I/Os** tab of the **I/O Mapping** window. The signals can now be mapped.

(>>> 11.6.3 "Mapping Sunrise I/Os" Page 161)

11.5.1 “Create I/O signals” window

Overview

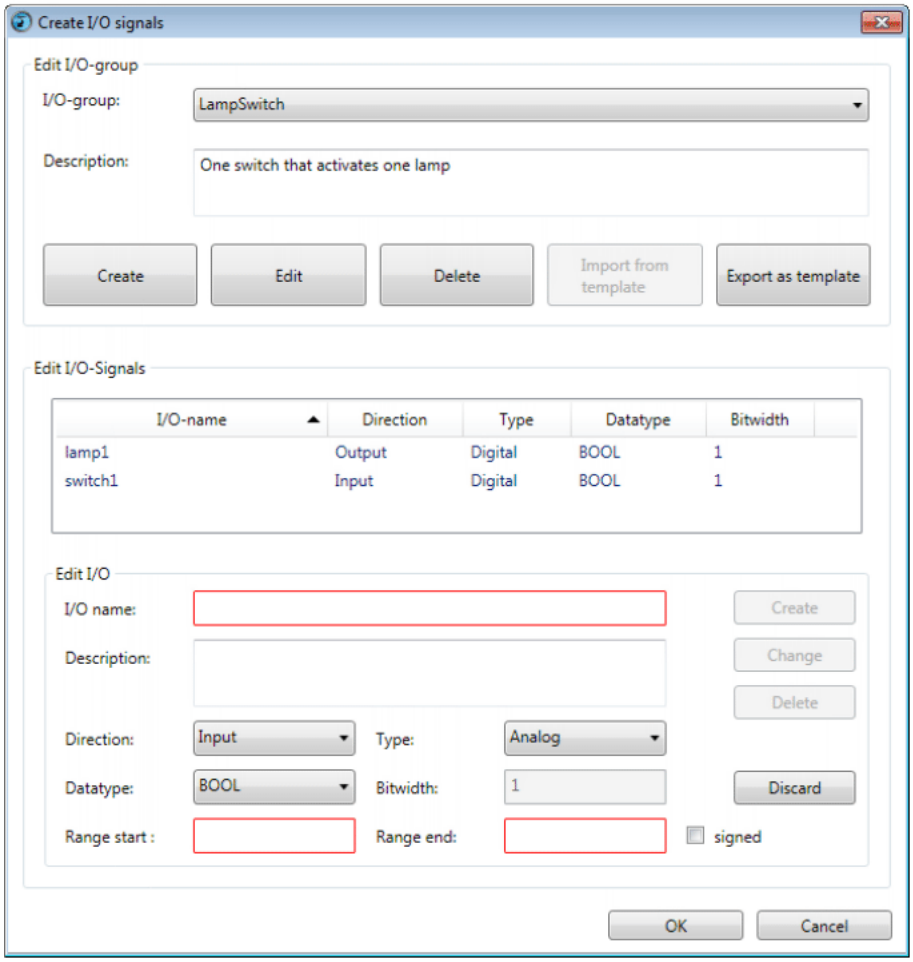


Fig. 11-1: “Create I/O signals” window

The window for creating and editing the Sunrise I/Os and Sunrise I/O groups consists of the following areas:

Area	Description
Edit I/O group	In this area, I/O groups are created and edited. It is also possible to save I/O groups as a template or to import previously created templates.
Edit I/O Signals	In this area, the input/output signals of an I/O group are displayed.
Edit I/O	In this area, the inputs/outputs of an I/O group are created and edited.

> Input boxes are displayed with a red frame if values must be entered or if incorrect values have been entered. A help text is displayed when the mouse pointer is moved over the box.

Signal properties

In the **Edit I/O** area, new signals can be created and their properties defined:

Property	Description
I/O name	Enter the name of the input or output.
Description	Enter a description for the input or output (optional).

Property	Description
Direction	Specify whether the signal is an input or output. ■ **Input**, **Output**
Type	Specify whether the signal is an analog or digital signal. ■ **Analog**, **Digital**
Data type	Select the data type of the signal. In WorkVisual, a total of 15 different data types are available for selection. For use with Java, these data types are mapped to the following data types: ■ integer, long, double, boolean
Bit width	Enter the number of bits that make up the signal. With the data type BOOL, the bit width is always 1. **Note**: The value must be a positive integer which does not exceed the maximum permissible length of the selected data type.
Range start	Only relevant for analog inputs/outputs! Enter the start and end of the range for the analog value and if applicable set the check mark at **Signed**. **Note**: These values can generally be found in the data sheet of the field bus module. The range start must be lower than the range end. It is also possible to enter decimal values.
Range end	
Signed	

11.5.2 Creating an I/O group and inputs/outputs within the group

Precondition

■ The **Create I/O signals** window is open.

Procedure

1. In the **Edit I/O group** area, click on **Create**.
   The **Create I/O group** window is opened.

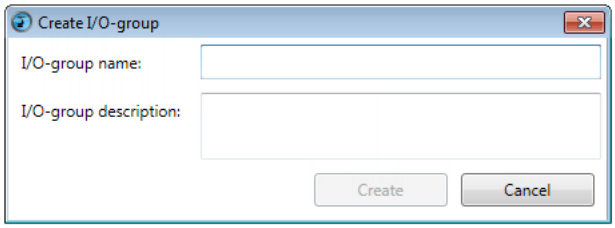


Fig. 11-2: Create I/O group

2. Enter a name for the I/O group.
3. Enter a description for the I/O group (optional).

> It is advisable to enter a description in all cases. This description is displayed later as a help text in the robot application and on the smartHMI.

4. Click on **Create**. The I/O group is created and displayed in the selection menu **I/O group**.
5. In the **Edit I/O** area, enter a name for the input or output of the group and define the signal properties.
   (>>> "Signal properties" Page 156)
6. In the **Edit I/O group** area, click on **Create**. The input or output signal is created and displayed in the **Edit I/O Signals** area.
7. Repeat steps 5 and 6 to define further inputs/outputs in the group.

11.5.3 Editing an I/O group

Precondition

- The **Create I/O signals** window is open.
- The inputs/outputs of the group are not mapped.

Procedure

1. Select the desired I/O group from the **I/O group** selection menu.
2. Click on **Edit**. The **Rename I/O group** window is opened.
3. Change the name of the I/O group and/or the corresponding description (optional). Confirm with **Apply**.

11.5.4 Deleting an I/O group

Precondition

- The **Create I/O signals** window is open.
- The inputs/outputs of the group are not mapped.

Procedure

1. Select the desired I/O group from the **I/O group** selection menu.
2. Click on **Delete**. If signals have already been created for the I/O group, a request for confirmation is displayed.
3. Reply to the request for confirmation with **Yes**. The I/O group is deleted.

11.5.5 Changing an input/output of a group

Precondition

- The **Create I/O signals** window is open.
- The signals that are to be changed are not mapped.

Procedure

1. Select the I/O group of the signal from the **I/O group** selection menu.
2. In the **Edit I/O Signals** area, click on the desired input or output.
3. In the **Edit I/O** area, edit the signal properties as required.
   (>>> "Signal properties" Page 156)

   > All the changes can be discarded by clicking on the **Discard** button.

4. Click on **Change**. The changes are saved.

11.5.6 Deleting an input/output of a group

Precondition

- The **Create I/O signals** window is open.
- The signals that are to be deleted are not mapped.

Procedure

1. Select the I/O group of the signal from the **I/O group** selection menu.
2. In the **Edit I/O Signals** area, click on the desired input or output.
3. Click on **Delete**.

11.5.7 Exporting an I/O group as a template

Description

I/O groups can be saved as a template. The template contains all the inputs/outputs belonging to the saved I/O group. This enables I/O groups, once created, to be reused. The mapping of the inputs and outputs is not saved, however.

After exporting the template, the templates created in WorkVisual are available in Sunrise.Workbench in the **IOTemplates** folder of the project.

Precondition

- The **Create I/O signals** window is open.

Procedure

1. In the **Edit I/O group** area, select the I/O group that is to be exported as a template.
2. Click on **Export as template**. The **Save I/O group as template** window is opened.
3. Enter a name for template.

> If a template with the same name already exists in the Sunrise project, it will be overwritten during the export operation.

4. Enter a description for the template (optional).
5. Click on **Export**. The template is saved.

11.5.8 Importing an I/O group from a template

Precondition

- The **Create I/O signals** window is open.
- There is at least one I/O group available in Sunrise.Workbench as a template in the Sunrise project.

Procedure

1. In the **Edit I/O group** area, click on **Import from template**. The **Import I/O group from template** window is opened.
2. In the selection list **Used template**, select the template to be imported.
3. Enter a name in the **I/O-group name** box for the I/O group to be created.
4. Click on **Import**. An I/O group configured in accordance with the template is imported and can be edited.

11.6 Mapping the bus I/Os

11.6.1 I/O Mapping window

Overview

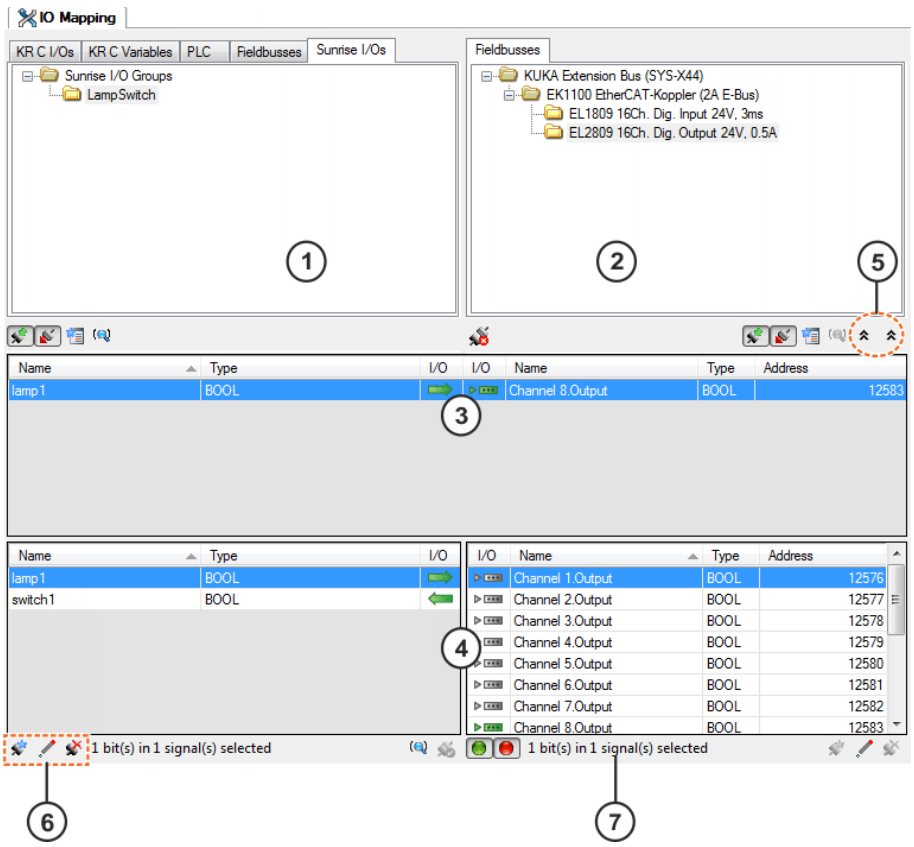


Fig. 11-3: “I/O Mapping” window

Item	Description
1	Displays the Sunrise I/O groups The signals in the I/O group selected here are displayed in the overviews lower down.
2	Displays the inputs/outputs of the bus modules The signals in the module selected here are displayed in the overviews lower down.
3	Connection overview Displays the mapped signals. These are the signals of the I/O group selected under **Sunrise I/Os**, which are mapped to the bus module selected under **Field buses**.
4	Signal overview Here the signals can be mapped. (>>> 11.6.3 "Mapping Sunrise I/Os" Page 161)
5	The arrow buttons allow the connection and signal overviews to be collapsed and expanded independently of one another. ■ **Collapse connection view** (left-hand arrow symbol pointing up) ■ **Expand connection view** (left-hand arrow symbol pointing down) ■ **Collapse signal view** (right-hand arrow symbol pointing up) ■ **Expand signal view** (right-hand arrow symbol pointing down)
6	Buttons for creating and editing the Sunrise I/Os
7	Displays how many bits the selected signals contain.

For the I/O mapping in Sunrise, only the **Sunrise I/Os** and **Field buses** tabs are relevant.

11.6.2 Buttons in the “I/O Mapping” window

Some of these buttons are displayed in several places. In such cases, they refer to the side of the **I/O Mapping** window on which they are located.

Editing

Button	Name / description
	Creates signals at the provider Opens the **Create I/O signals** window. (>>> 11.5.1 "“Create I/O signals” window" Page 156) The button is only active if an input or output module is selected on the **Field buses** tab and a signal of the I/O group is selected in the signal overview.

Button	Name / description
	Edit signals at the provider Opens the **Edit I/O signals** window. The button is only active if an I/O group is selected on the **Sunrise I/Os** tab and a signal of the I/O group is selected in the signal overview.
	Deletes signals at the provider Deletes all the selected inputs/outputs. If all the inputs/outputs of a group are selected, the I/O group is also deleted. The button is only active if an I/O group is selected on the **Sunrise I/Os** tab and a signal of the I/O group is selected in the signal overview.

Mapping

Button	Name / description
	Disconnect Disconnects the selected mapped signals. It is possible to select and disconnect a number of connections simultaneously.
	Connect Connects signals which are selected in the signal overview.

11.6.3 Mapping Sunrise I/Os

Description

This procedure is used to map the Sunrise I/Os to the inputs/outputs of the field bus module. It is only possible to map inputs to inputs and outputs to outputs if they are of the same data type. For example, it is possible to map BOOL to BOOL or INT to INT, but not BOOL to INT or BYTE.

Precondition

- The robot controller has been set as the active controller.

Procedure

1. On the **Sunrise I/Os** tab in the left-hand half of the window, select the I/O group for which the I/Os are to be mapped.
   The signals of the group are displayed in the bottom area of the **I/O Mapping** window.
2. On the **Field buses** tab in the right-hand half of the window, select the desired input or output module.
   The signals of the selected field bus module are displayed in the bottom area of the **I/O Mapping** window.
3. Drag the signal of the group onto the input or output of the module. (Or alternatively, drag the input or output of the device onto the signal of the group.)
   The signals are now mapped. Mapped signals are indicated by green arrows.

Alternative procedure for mapping:

- Select the signals to be mapped and click on the **Connect** button.

11.7 Exporting the I/O configuration to the Sunrise project

Description

When exporting an I/O configuration from WorkVisual, a separate Java class is created for each I/O group in the corresponding Sunrise project. Each of these Java classes contains the methods required for programming, in order

to be able to read the inputs/outputs of an I/O group and write to the outputs of an I/O group.

The classes and methods are saved in the Java package **com.kuka.generated.ioAccess** in the source folder of the Sunrise project.

> The source code of the Java classes of the package **com.kuka.generated.ioAccess** must not be changed manually. To expand the functionality of an I/O group, it is possible to derive further classes from the classes created or to continue to use objects from these classes, e.g. as arrays of their own classes (aggregating).

The structure of the Sunrise project after exporting an I/O configuration is described here:

(>>> 15.12 "Inputs/outputs" Page 300)

Precondition

- The robot controller has been set as the active controller.
- The automatic change recognition is activated in Sunrise.Workbench.
  (>>> 5.10 "Activating the automatic change recognition" Page 58)

Procedure

1. Select the menu sequence **File** > **Import / Export**. The import/export wizard for files opens.
2. Select **Export the I/O configuration to the Sunrise Workbench project.**.
3. Click on **Next >** and then on **Finish**. The configuration is exported to the Sunrise project.
4. A message is displayed as to whether the export was successfully completed. If Sunrise I/Os have not been mapped, this is also indicated.

   > It is not essential to map all the Sunrise I/Os that have been created.

   Click **Close** to terminate the wizard.
5. Close WorkVisual by selecting **File** > **Exit**.

12 External control

12.1 Configuring external control

Description

If the processes of the station are to be controlled by a higher-level controller in Automatic mode, the Sunrise project on the robot controller must be configured for external control.

Every Sunrise project that is configured for external control uses a default application that is automatically selected when switching to Automatic mode. The default application of the externally controlled project cannot be deselected again in Automatic mode.

The default application cannot be started using the Start key on the smartPAD, but rather only via an external input.

The robot controller and the higher-level controller communicate via predefined input/output signals:

- The higher-level controller can start, pause and resume the default application via the input signals.
- The output signals can be used to provide information about the status of the default application and the station to the higher-level controller.

Overview

The following steps are required in order to be able to use external control:

Step	Description
1	Configure and map inputs/outputs for communication with the higher-level controller in WorkVisual. (>>> 12.1.1 "External control inputs" Page 164) (>>> 12.1.2 "External control outputs" Page 164)
2	Export I/O configuration from WorkVisual to Sunrise.Workbench.
3	Create the default application for external control of the Sunrise project in Sunrise.Workbench.
4	Configure external control of the Sunrise project in Sunrise.Workbench. (>>> 12.1.4 "Configuring external control in the project properties" Page 166)
5	Transfer the Sunrise project to the robot controller by means of synchronization.

The physical inputs/outputs used for communication with the higher-level controller must not be multiply mapped.

Precondition

In order to be able to start an application, the following preconditions must be met:

- The robot is mastered (all axes).
- A Sunrise project has been configured for external control.
- AUT mode
- The input App_Start has been configured.
- If configured: the input App_Enable has the level HIGH (TRUE).
- The motion enable signal is present.

12.1.1 External control inputs

App_Start

The input App_Start is absolutely vital for an externally controlled project.

The default application is started and resumed in Automatic mode by the higher-level controller by means of a rising edge of the input signal (change from FALSE to TRUE).

App_Enable

The input App_Enable can optionally be configured.

The default application can be paused by the higher-level controller in Automatic mode by means of a LOW level at this input.

System response

The input App_Enable has a higher priority than the input App_Start. If the input App_Enable is configured, the default application can only be started if there is a high-level signal at App_Enable.

App_Start	App_Enable	Application status	Reaction
FALSE --> TRUE	FALSE	**Selected**	None
FALSE --> TRUE	FALSE	**Motion paused**	None
FALSE --> TRUE	TRUE	**Selected**	Application is started.
FALSE --> TRUE	TRUE	**Motion paused**	Application is resumed.
			If the path is left: the robot is repositioned. Application is then paused.
Any	TRUE --> FALSE	**Running**	Application is paused.
Any	TRUE --> FALSE	**Repositioning**	Application is paused.

12.1.2 External control outputs

The configuration of these outputs is optional.

AutExt_Active

A high level at this output signals to the higher-level controller that Automatic mode is active and the project on the robot controller can be controlled externally.

AutExt_AppReadyToStart

A high level at this output signals to the higher-level controller that the default application is ready to start (status **Selected** or **Motion paused**).

DefaultApp_Error

A high level at this output signals to the higher-level controller that an error occurred when the default application was run (status **Error**).

Station_Error

A high level at this output signals to the higher-level controller that the station is in an error state. The error state is active if one of the following conditions is met:

- Motion enable signal not present.
- Drive error or bus error active.
- At least one robot axis is not mastered and the operating mode is not set to T1.

NOTICE It is not permissible to set in a robot application outputs that signal system states to the higher-level controller. Failure to observe this precaution may result in malfunctioning of the higher-level controller and damage to property.

12.1.3 Signal diagrams

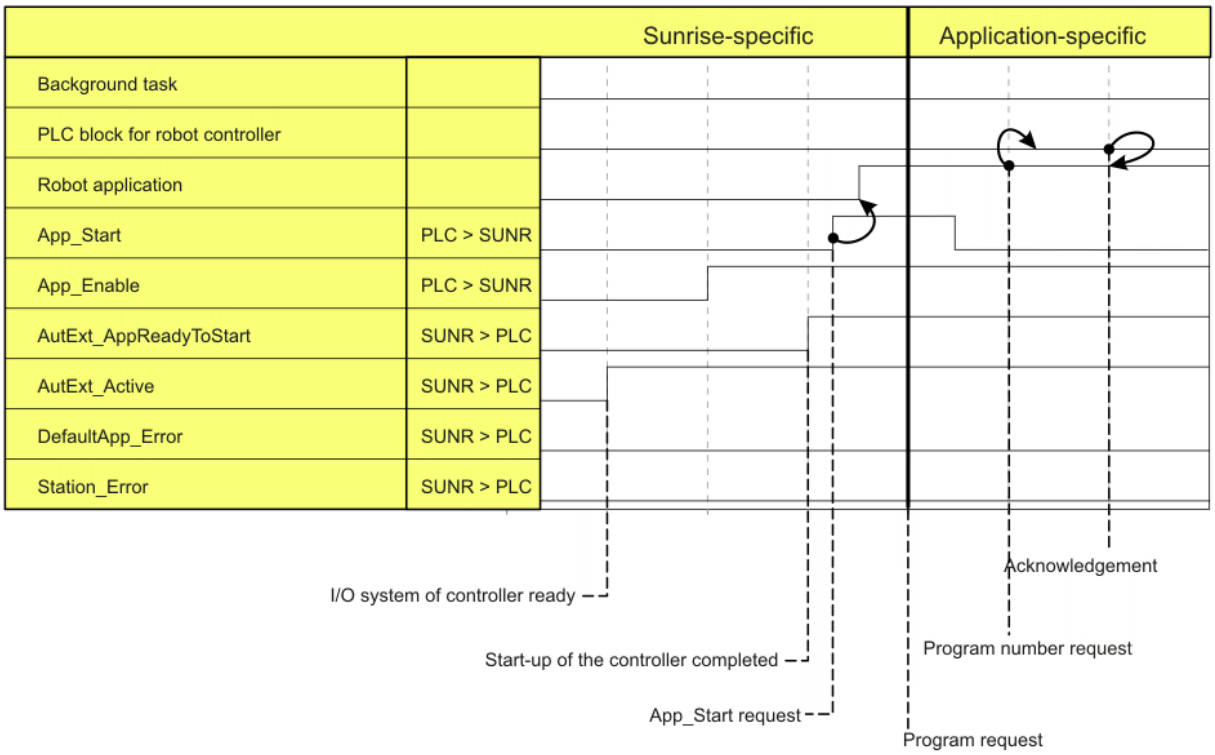


Fig. 12-1: Automatic system start and normal operation

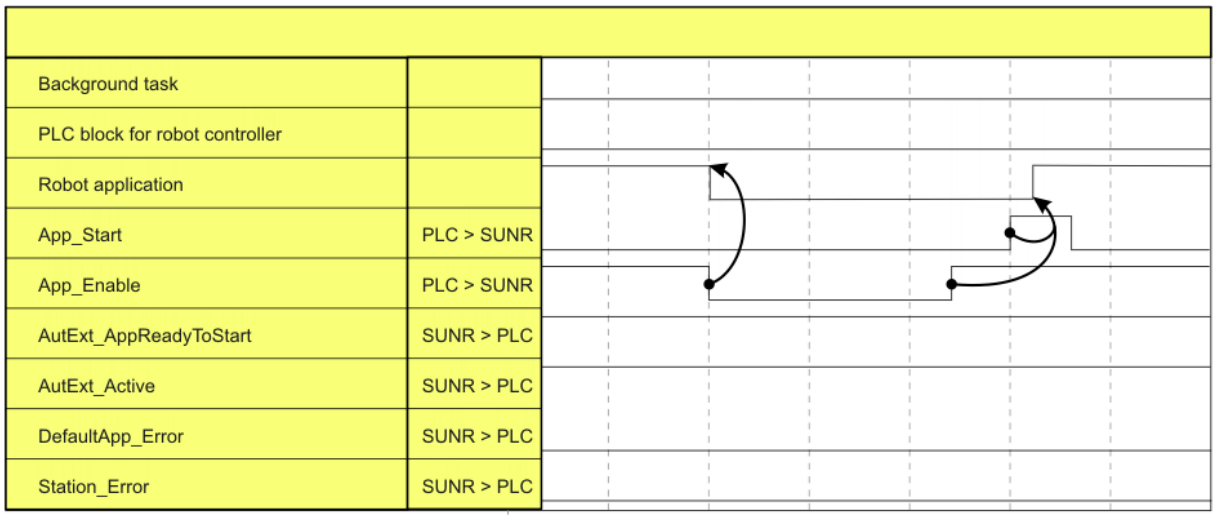


Fig. 12-2: Restart after user stop

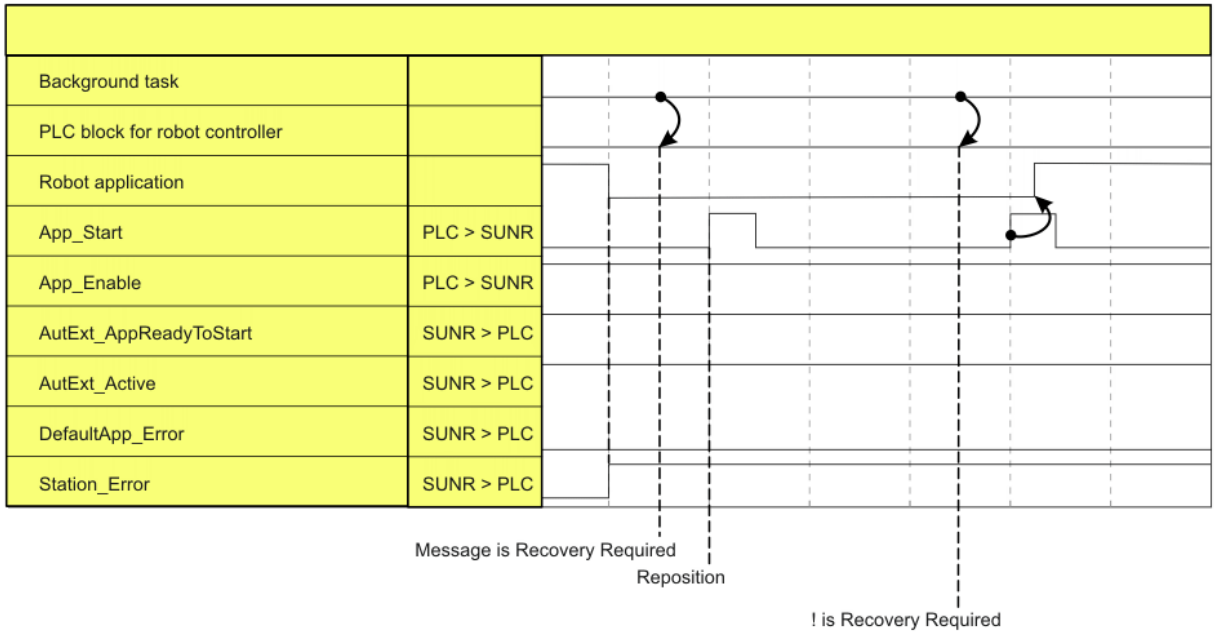


Fig. 12-3: Restart after external EMERGENCY STOP

12.1.4 Configuring external control in the project properties

Procedure

1. Right-click on the desired Sunrise project in the **Package Explorer** and select **Sunrise** > **Change project settings** from the context menu.

   The **Properties for Project** window opens.
2. Select **Sunrise** > **External Control** in the directory structure in the left area of the window.
3. Make the settings for external control of the project in the right-hand area of the window.
   - Set the check mark at **Project is controlled externally**.
   - In the **Default Application** area, select the desired default application.
   - In the **Input configuration** area, configure the inputs for the external communication.
     - **IO group** column: select the I/O group containing the desired input.
     - **Boolean input** column: select input from the I/O group.
   - Optional: In the **Output configuration** area, configure the signal outputs for the project.
     - **IO group** column: select the I/O group containing the desired output.
     - **Boolean output** column: select output from the I/O group.
4. Click on **Apply** to apply the settings and close the window with **OK**.

> The default application is indicated in the **Package Explorer** by the following icon:
> If the default application is renamed, the icon is no longer displayed and the application must be selected as the default application once again. (>>> 12.2 "Selecting a robot application as the default application" Page 167)

Description

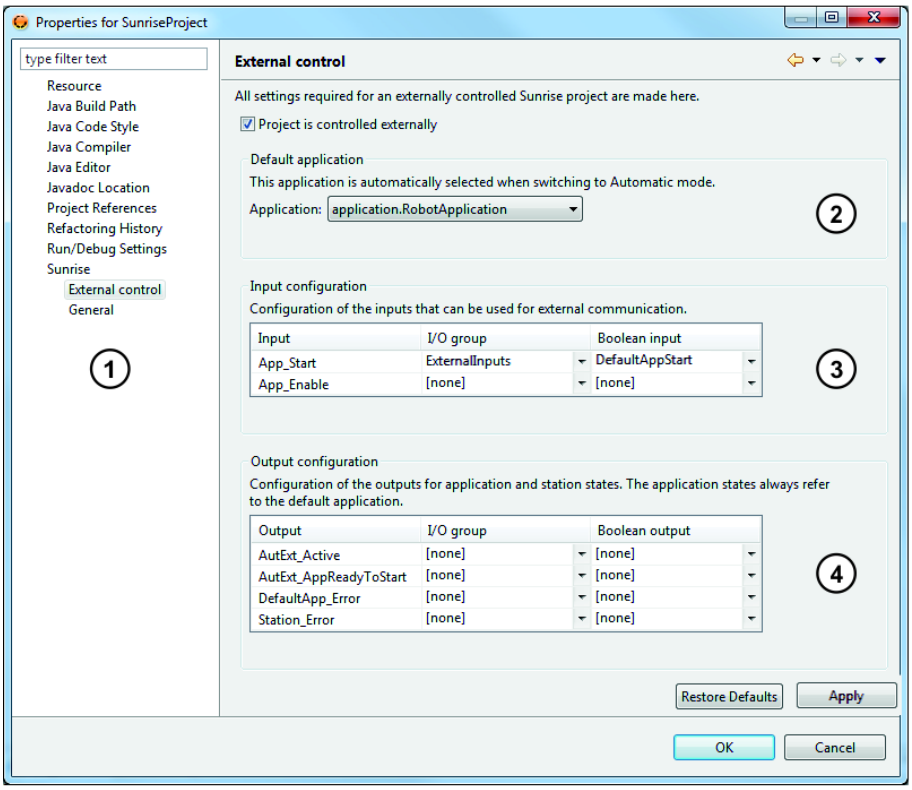


Fig. 12-4: Minimal configuration: external control of a project

Item	Description
1	Project properties directory structure
2	**Default application** area All robot applications of the project are available for selection.
3	**Input configuration** area All I/O groups of the I/O configuration of the project and the corresponding inputs are available for selection. The input App_Start is absolutely vital for external control of a project.
4	**Output configuration** area All I/O groups of the I/O configuration of the project and the corresponding outputs are available for selection.

12.2 Selecting a robot application as the default application

Description

A default application can be defined for every Sunrise project; it is automatically selected after a reboot of the robot controller or synchronization of the project.

In the case of an externally controlled project, it is essential to define a default application. This is automatically selected when the operating mode is switched to Automatic.

Procedure

- Right-click on the desired robot application in the **Package Explorer** and select **Sunrise** > **Set as default application** from the context menu.

  The robot application is indicated as the default application in the **Package Explorer** and automatically set as the default application in the project properties.

Example

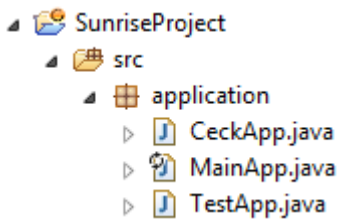


Fig. 12-5: Default application MainApp.java

12.3 Defining the signal outputs for a project that is not externally controlled

Description

The predefined output signals for communication with the higher-level controller can also be used to signal application and station statuses in projects that are not externally controlled.

The application statuses always refer to the default application of the project.

Precondition

- The I/O configuration of the project contains the outputs configured and mapped in WorkVisual.
  (>>> 12.1.2 "External control outputs" Page 164)

Procedure

1. Right-click on the desired Sunrise project in the **Package Explorer** and select **Sunrise** > **Change project settings** from the context menu.
   The **Properties for Project** window opens.
2. Select **Sunrise** > **General** in the directory structure in the left area of the window.
3. Make the general settings for the project in the right-hand area of the window.
   - If application statuses are to be signaled, select the desired default application in the **Default Application** area.
   - In the **Output configuration** area, configure the signal outputs for the project.
     - **IO group** column: select the I/O group containing the desired output.
     - **Boolean output** column: select output from the I/O group.
4. Click on **Apply** to apply the settings and close the window with **OK**.

13 Safety configuration

13.1 Overview of safety configuration

The safety configuration defines the safety-oriented functions in order to integrate the industrial robot safely into the system. Safety-oriented functions serve to protect human operators when they work with the robot.

The safety configuration is an integral feature of a Sunrise project and is managed in tabular form. The individual safety functions are grouped in KUKA Sunrise.Workbench on an application-specific basis. The safety configuration is then transferred with the project to the controller and activated there.

⚠ WARNING Serious damage and injury or death can result from incorrect safety configuration. If a new or changed safety configuration is activated, the safety maintenance technician must conduct tests to ensure that the configured safety parameters have been correctly applied and that the safety functions of the configuration are fully functional (safety acceptance).

⚠ Configuration of the safety functions, activation and deactivation of the safety functions and safety acceptance may only be carried out by a trained safety maintenance technician. The safety maintenance technician is responsible for ensuring that the safety configuration is only activated on those robots for which it is intended.
The safety configuration is not currently checked for plausibility by KUKA Sunrise.Workbench.

⚠ In the case of incomplete start-up of the system, additional substitute measures for minimizing risk must be taken and documented, e.g. installation of a safety fence, attachment of a warning sign, locking of the main switch, etc. Start-up is incomplete, for example, if not all safety functions have yet been implemented, or if a function test of the safety functions has not yet been carried out.

⚠ The system integrator must verify that the safety configuration sufficiently reduces risks during collaborative operation (HRC).
It is advisable to perform this verification in accordance with the information and instructions for operating collaborative robots in ISO/TS 15066.

⚠ States with various safety settings are defined in the safety configuration as part of the ESM mechanism (Event-Driven Safety Monitoring). It is possible to switch between these in the application. Since switching between these states is carried out by means of non-safety-oriented signals, all configured states must be consistent. This means that each state must ensure a sufficient degree of safety, regardless of the time or place of activation (i.e. regardless of the current process step)

13.2 Safety concept

Overview

The safety configuration must implement all safety functions which are required to operate the industrial robot. A safety function monitors the entire system on the basis of specific criteria. These are described by individual monitoring functions, so-called AMFs (Atomic Monitoring Functions). To configure a safety function, several AMFs can be linked to form complex safety monitoring functions. In addition, the safety function defines a suitable reaction which is triggered in case of error.

Example: In a specific area of the robot's workspace, the velocity at the TCP must not exceed 500 mm/s (“Workspace monitoring” and “Velocity monitoring” monitoring functions). Otherwise, the robot must stop immediately (reaction in case of error).

PSM and ESM

The Sunrise safety concept provides 2 different monitoring mechanisms:

- Permanent safety-oriented monitoring

  The safety functions of the PSM mechanism (Permanent Safety Monitoring) are always active. It is only possible to deactivate individual safety functions by changing the safety configuration.

  The PSM mechanism is used to constantly monitor the system. It implements basic safety settings which are independent of the process step being carried out. These include, for example, EMERGENCY STOP functions, the enabling switch on the smartPAD, the definition of a cell area or safety functions that depend on the operating mode.
- Event-dependent safety-oriented monitoring

  The ESM mechanism (Event-driven Safety Monitoring) defines safe states. It is possible to switch between these in the application. A safe ESM state contains the safety functions required in the corresponding process step.

  Since switching is carried out by means of non-safety-oriented signals, the defined state must ensure a sufficient degree of safety, regardless of the time or place of activation.

  The ESM mechanism allows specific safety functions to be adapted for specific processes. This is of particular importance for human-robot cooperation applications, as these often require various safety settings depending on the situation. The required parameters, such as permissible velocity, collision values or spatial limits, can be individually defined for each process step using an ESM state.

AMF

The smallest unit of a safety monitoring function is called an Atomic Monitoring Function (AMF).

Each AMF supplies an elementary, safety-relevant piece of information, for example if a safe input is set or if the Automatic operating mode is selected.

Atomic Monitoring Functions can have 2 different states and are LOW-active. This means that if a monitoring function is violated, the state switches from “1” to “0”.

- State “0”: The AMF is violated.
- State “1”: The AMF is not violated.

For example, the AMF **smartPAD Emergency Stop** is violated if the EMERGENCY STOP device on the operator panel is pressed.

Safety function

A safe ESM state is defined with up to 20 safety functions. The safety functions of the ESM mechanism use exactly one AMF. If this AMF is violated, the safety function and thus the entire ESM state is considered to be violated.

For safety functions of the PSM mechanism, up to 3 AMFs are logically linked to one another. This allows complex safety monitoring functions to be implemented. If all AMFs of a safety function of the PSM mechanism are violated, the entire safety function is considered to be violated.

Safety interfaces

Various safety interfaces are available for exchanging safety-oriented signals between a higher-level controller and a robot controller. The safe inputs of these interfaces can be used to connect safety devices, for example external EMERGENCY STOP devices or safety gates, and to evaluate the corresponding input signals. The safe outputs of these interfaces can be used to signal the violation of safety functions.

- Ethernet safety interfaces (only slave function available)
  - PROFINET / PROFIsafe
  - EtherCAT/FSoE
- Discrete safety interfaces
  - CIB_SR/X11

> The PROFINET bus can be configured in WorkVisual. Further information about the concrete configuration of the field bus is contained in the corresponding field bus documentation.

> Further information on interface X11 can be found in the operating instructions for **KUKA Sunrise Cabinet**.

Reactions

A suitable reaction is defined for each safety function. This reaction must take place in the case of an error and put the system into a safe state.

The following reactions can be configured:

- Safety stop 0 is triggered.

> It is advisable to only configure a safety stop 0 if it is necessary to immediately switch off the drives and apply the brakes as a reaction.

- Safety stop 1 is triggered.
- Safety stop 1 (path-maintaining) is triggered.

  This is the recommended stop reaction. It has the lowest impact on the process, as an application can be resumed without the need to reposition the robot.

> **⚠ CAUTION** In crushing situations, safety stop 1 and safety stop 1 (path-maintaining) can result in higher crushing forces due to the controlled stop on a planned braking path. It is therefore advisable to use safety stop 0 for safety monitoring functions which recognize crushing situations (e.g. the AMF *Collision detection*, *TCP force monitoring*).

- Safe output is set to “0” (LOW level).

> Setting a safe output can only be configured as a reaction for safety functions of the PSM mechanism. It cannot be configured for safety functions of the ESM mechanism.

The reactions can be used for any number of safety functions. A reaction is triggered once one of the safety functions using this reaction is violated. This makes it possible, for example, to inform a higher-level controller via a safe output when specific errors occur.

With the PSM mechanism, it is possible to trigger several different reactions when a specific combination of AMFs is violated. For example, a safety stop can be triggered as well as a safe output. To do so, 2 safety functions must be configured with identical AMF combinations.

If different stop reactions are configured, a violation triggers the stronger stop reaction. In other words, it triggers the stop reaction which causes an earlier safety-oriented disconnection of the drives. If several safety functions use the same output signal as a reaction, this signal is set to “0” once one of the safety functions is violated.

Time behavior

All the safety-oriented outputs use LOW as a safe state.

If a safety function which uses a safety output as a reaction is violated, this output is immediately set to LOW.

If the violation state is cancelled, the output is only set to HIGH again when the following conditions have been met:

- The safety function is not violated for at least 24 ms. The reaction to cancellation of the violation state is always delayed.
- If an Ethernet safety interface is used:

  The output has the LOW level for at least 500 ms beforehand. If the LOW level has not yet been present for this time, the level change to HIGH waits until the 500 ms has elapsed.
- If the discrete safety interface is used:

  The output has the LOW level for at least 200 ms beforehand. If the LOW level has not yet been present for this time, the level change to HIGH waits until the 200 ms has elapsed.

When using safety functions with a safe output as a reaction, it must be noted that connection errors (i.e. communication errors) at safe inputs or outputs are automatically acknowledged by the safety controller when the connection is restored. Accordingly, the level of the safe output can switch from LOW to HIGH once the connection is restored.
For this reason, the safety maintenance technician must ensure that peripheral devices do not automatically restart.

13.3 Permanent Safety Monitoring

The safety functions of the PSM mechanism (Permanent Safety Monitoring) are permanently active and use the criteria defined by these functions to ensure that the overall system is constantly monitored.

For a safety function of the PSM mechanism, up to 3 AMFs (Atomic Monitoring Functions) can be linked to one another. The entire safety function is only considered violated if all of these AMFs are violated. The safety function also defines a reaction. This is triggered if the entire safety function is violated.

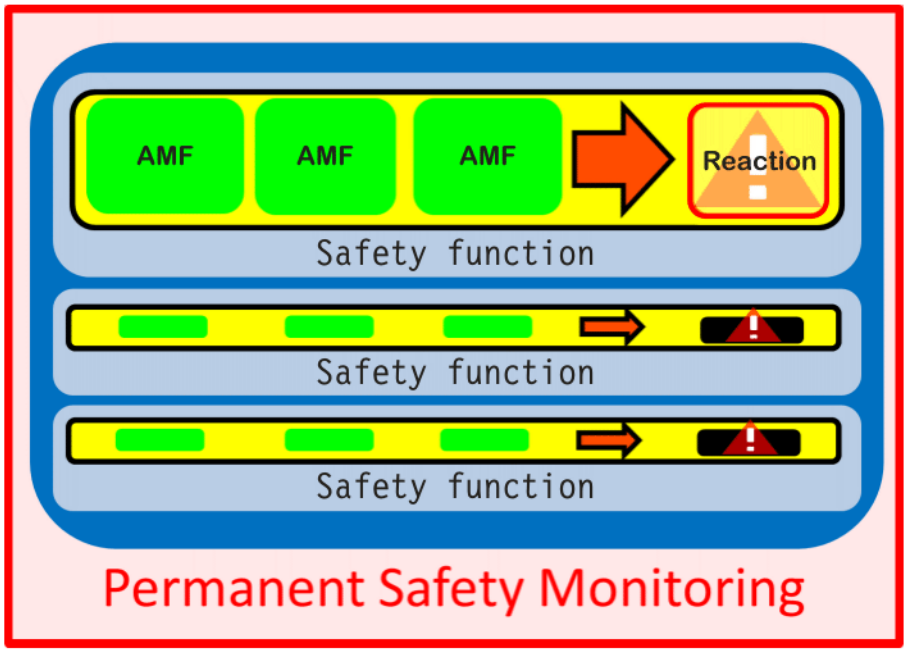


Fig. 13-1: Safety functions of the PSM mechanism

Categories

For diagnosis in case of error, a category is assigned to each safety function of the PSM mechanism. Depending on the category, errors are displayed on

the smartPAD and saved in the LOG file. For this reason, it is advisable to select these carefully.

The following categories are available:

Category	Recommended use
None	For safety functions which cannot be assigned a category
Output	For safety functions which use setting an output as a reaction In this category, no diagnostic information is provided in case of violation.
Enabling device	For safety functions which evaluate an enabling switch In this category, no diagnostic information is provided in case of violation because enabling is a normal operating state and not an error state.
Local EMERGENCY STOP	For safety functions which evaluate an EMERGENCY STOP triggered by the EMERGENCY STOP device on the smartPAD
External EMERGENCY STOP	For safety functions which evaluate an EMERGENCY STOP triggered by an external EMERGENCY STOP device
Operator safety	For safety functions which evaluate the signal for operator safety
Safe operational stop	For safety functions which monitor robot standstill
Collision detection	For safety functions which are used for collision detection or force monitoring
Safety stop	For safety functions which use a safety stop as a reaction and cannot be assigned to another category. Example: external safety stop
Velocity monitoring	For safety functions which are used for monitoring an axis-specific or Cartesian velocity
Workspace monitoring	For safety functions which are used for monitoring an axis-specific or Cartesian space

13.4 Event-driven Safety Monitoring

The ESM mechanism (Event-driven Safety Monitoring) makes it possible to switch between different safe ESM states depending on the situation.

Up to 10 safe states can be defined. Switching between states can be carried out in the robot application or in a background task.

(>>> 13.6.4.8 "Switching between ESM states" Page 188)

A safe ESM state is defined with up to 20 safety functions which must ensure a sufficient degree of safety in every situation. Switching to an ESM state in the program makes this state active. As long as the ESM state is active, all corresponding safety functions are monitored in addition to the permanently active safety functions.

Use of the ESM mechanism is optional. The ESM mechanism is deactivated if no ESM state is defined in the safety configuration.

When using the ESM mechanism, exactly one safe state is always active. It is not possible to switch it off in the application.

The safety functions of an ESM state each contain a single AMF which is assigned to a suitable stop reaction.

Once a safety function of the active ESM state is violated, a stop is triggered. The type of stop reaction will be the strongest of all the violated safety functions in all ESM states (both active and inactive). In other words, it triggers the

stop reaction which causes the earliest safety-oriented disconnection of the drives.

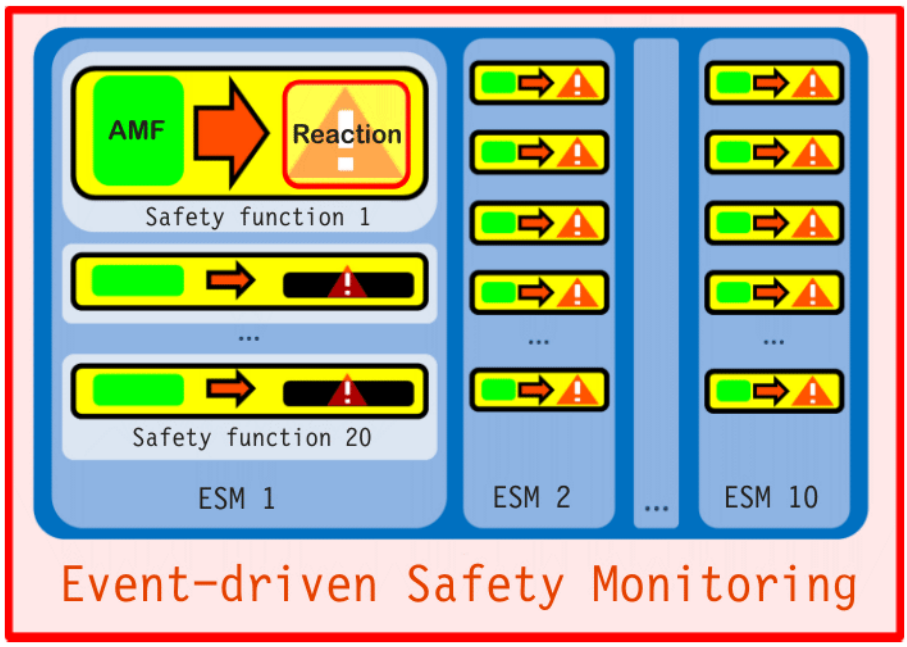


Fig. 13-2: Safety functions of the ESM mechanism

13.5 Overview of Atomic Monitoring Functions

The smallest unit of a safety function is designated as the Atomic Monitoring Function (AMF). This can be, for example, evaluating the enabling switch on the smartPAD or monitoring the velocity of an axis.

Atomic Monitoring Functions are divided into 3 categories:

- Standard AMFs
- Parameterizable AMFs
- Extended AMFs

13.5.1 Standard Atomic Monitoring Functions

Description

Standard Atomic Monitoring Functions provide information about system components or system states, for example the safety equipment on the smartPAD or the active operating mode. Standard AMFs can be used in any number of safety functions.

Overview

AMFs for evaluating the safety equipment on the smartPAD:

AMF	Task
smartPAD Emergency Stop	Monitors the EMERGENCY STOP device on the smartPAD
smartPAD enabling switch inactive	Checks whether the enabling signal has not been issued on the smartPAD.
smartPAD enabling switch panic active	Checks whether an enabling switch on the smartPAD has been pressed down fully (panic position).

(>>> 13.8.1 "Evaluating the safety equipment on the KUKA smartPAD" Page 190)

AMFs for evaluating the operating mode:

AMF	Task
Test mode	Checks whether a test operating mode is active (T1, T2 , CRR)
Automatic mode	Checks whether the Automatic operating mode is active (AUT)
Reduced-velocity mode	Checks whether an operating mode with reduced velocity is active (T1, CRR) **Note**: In the case of a mobile platform, the velocity is not reduced in T1 and CRR mode.
High-velocity mode	Checks whether an operating mode with programmed velocity is active (T2 , AUT)

(>>> 13.8.2 "Evaluating the operating mode" Page 191)

AMFs for evaluating the motion enable:

AMF	Task
Motion enable	Monitors the motion enable, i.e. checks whether a safety stop is active.

(>>> 13.8.3 "Evaluating the motion enable" Page 191)

13.5.2 Parameterizable Atomic Monitoring Functions

Description

In contrast to standard AMFs, parameterizable Atomic Monitoring Functions additionally have one or more parameters. These can be configured depending on the values at which the AMF is to be considered violated. Example: monitoring limits.

For parameterizable AMFs, a fixed number of instances is available. The number indicates how many differently parameterizable versions of the AMF can be configured and used. Each instance can have different parameter values. The instance of an AMF may be used multiple times in the table in which the safety functions are configured.

Overview

AMF for evaluating safe inputs:

AMF	Task
Input signal	Monitors a safe input (>>> 13.8.4 "Monitoring safe inputs" Page 192)

AMFs for evaluating the enabling signal on the hand guiding device:

AMF	Task
Hand guiding device enabling inactive	Checks whether the enabling signal has not been issued on the hand guiding device.
Hand guiding device enabling active	Checks whether the enabling signal has been issued on the hand guiding device. The AMF is used to activate further monitoring functions during manual guidance with an enabling device.

(>>> 13.8.5 "Manual guidance with enabling device and velocity monitoring" Page 192)

AMFs for evaluating the referencing status:

AMF	Task
Position referencing	Monitors the referencing status of the position values for the axes of a kinematic system (>>> 13.8.6 "Evaluating the position referencing" Page 195)
Torque referencing	Monitors the referencing status of the joint torque sensors for the axes of a kinematic system (>>> 13.8.7 "Evaluating the torque referencing" Page 196)

AMFs for velocity monitoring:

AMF	Task
Axis velocity monitoring	Monitors the velocity of one of the axes of a kinematic system (>>> 13.8.8.1 "Defining axis-specific velocity monitoring" Page 197)
Cartesian velocity monitoring	Monitors the Cartesian translational velocity at defined points of a kinematic system (>>> 13.8.8.2 "Defining Cartesian velocity monitoring" Page 197)
Tool-related velocity component	Checks whether the Cartesian translational velocity in a specific direction is below the configured limit. (>>> 13.8.8.3 "Direction-specific monitoring of Cartesian velocity" Page 199)

AMFs for space monitoring:

AMF	Task
Cartesian workspace monitoring	Checks whether a part of the structure of a kinematic system being monitored is located outside of its permissible workspace (>>> 13.8.9.1 "Defining Cartesian workspaces" Page 206)
Cartesian protected space monitoring	Checks whether a part of the structure of a kinematic system being monitored is located within a non-permissible protected space (>>> 13.8.9.2 "Defining Cartesian protected spaces" Page 208)
Axis range monitoring	Monitors the position of one of the axes of a kinematic system (>>> 13.8.9.3 "Defining axis-specific monitoring spaces" Page 210)

AMF for monitoring the tool orientation:

AMF	Task
Tool orientation	Checks whether the orientation of the tool of a kinematic system is outside a permissible range (>>> 13.8.10 "Monitoring the tool orientation" Page 212)

AMFs for the safe monitoring of forces and torques (HRC):

AMF	Task
Axis torque monitoring	Monitors the torque of one of the axes of a kinematic system (>>> 13.8.13.1 "Axis torque monitoring" Page 216)

AMF	Task
Collision detection	Monitors the external torque of the axes of a kinematic system (>>> 13.8.13.2 "Collision detection" Page 217)
TCP force monitoring	Monitors the external force acting on the TCP of the tool or on the robot flange of a kinematic system (>>> 13.8.13.3 "TCP force monitoring" Page 218)

13.5.3 Extended Atomic Monitoring Functions

Description

Extended Atomic Monitoring Functions differ from the standard AMFs and parameterizable AMFs in that monitoring parameters are only defined during operation. The parameters are set at the time of activation. For the AMF *Standstill monitoring of all axes*, for example, the axis angles are set as reference angles for monitoring at the time of activation. An Extended AMF is activated if all other AMFs used by the safety function are violated. As long as at least one of the other AMFs is not violated, the Extended AMF is not active and not evaluated.

> Extended AMFs are only evaluated one cycle after they are activated. This can result in an extension of the reaction time by up to 12 ms.

Multiple instances are available for Extended AMFs. It is advisable to use each instance only once in the safety configuration, even with Extended AMFs which cannot be parameterized.

> Extended AMFs are not available for the safety functions of the ESM mechanism.

Overview

AMF for standstill monitoring:

AMF	Task
Standstill monitoring of all axes	Monitors the standstill of all axes of a kinematic system. (>>> 13.8.11 "Standstill monitoring (safe operational stop)" Page 214)

AMF for switching a delay:

AMF	Task
Time delay	Delays the triggering of the reaction of a safety function for a defined time. (>>> 13.8.12 "Activation delay for safety functions" Page 215)

13.5.4 Availability of the AMFs depending on the kinematic system

Description

Some safety monitoring functions (AMFs) provided by the system software are kinematic-specific, i.e. the kinematic system to be monitored must be selected during configuration of these AMFs. (Parameter *Monitored kinematic system* with the values **First kinematic system** … **Fourth kinematic system**)

If kinematic-specific AMFs are used in the safety configuration, the kinematic system that is to be monitored must be specified as follows:

- **First kinematic system**: if the LBR iiwa is to be monitored
- **Second kinematic system**: if the mobile platform is to be monitored

The third and fourth kinematic systems are not currently assigned to any kinematic system.

Overview

Not all kinematic system-specific AMFs can be used for the mobile platform, as the required safety-oriented sensor information is not available. If an AMF cannot be used for the monitored kinematic system, it is always evaluated as violated.

AMF	LBR iiwa	KMP
Position referencing	✔	✘
Torque referencing	✔	✘
Axis velocity monitoring	✔	✔
Cartesian velocity monitoring	✔	✔
Tool-related velocity component	✔	✔
Cartesian workspace monitoring	✔	✘
Cartesian protected space monitoring	✔	✘
Axis range monitoring	✔	✘
Tool orientation	✔	✘
Axis torque monitoring	✔	✘
Collision detection	✔	✘
TCP force monitoring	✔	✘
Standstill monitoring of all axes	✔	✘

13.6 Safety configuration with KUKA Sunrise.Workbench

The safety configuration is an integral feature of a Sunrise project. It is managed in tabular form.

The safety configuration defines the safety functions of the PSM mechanism and the safe states of the ESM mechanism.

When creating a new Sunrise project, the system automatically generates a standard safety configuration.

(>>> 5.3 "Creating a Sunrise project with a template" Page 48)

The standard safety configuration contains permanently active safety functions predefined by KUKA. The ESM mechanism has no preconfigured states and is therefore deactivated.

> Further information on the standard safety configuration can be found in the “Safety” chapter.

In KUKA Sunrise.Workbench, the safety configuration is displayed, edited and transferred to the controller when the system software is installed or the project is synchronized. The safety configuration can be activated and deactivated via the smartHMI.

13.6.1 Overview: changing the safety configuration and activating it on the controller

Step	Description
1	Open the safety configuration. (>>> 13.6.2 "Opening the safety configuration" Page 180)
2	Edit the safety functions in the *Customer PSM* table or create new safety functions. (>>> 13.6.3 "Configuring the safety functions of the PSM mechanism" Page 182)
3	Configure event-dependent monitoring functions if required. To do so, create safe ESM states and corresponding safety functions. Existing ESM states can be changed by adapting safety functions which are already configured or by adding new ones. (>>> 13.6.4 "Configuring the safe states of the ESM mechanism" Page 185)
4	Save the safety configuration.
5	When using the ESM mechanism Program the necessary switch between the safe states in robot applications and background tasks. (>>> 13.6.4.8 "Switching between ESM states" Page 188)
6	When using position-based AMFs (>>> "Position-based AMFs" Page 229) If necessary, create the application prepared by KUKA for the position and torque referencing of the LBR iiwa or create a separate reference run application. (>>> 13.10.1 "Position referencing" Page 223)
7	When using axis torque-based AMFs (>>> "Axis torque-based AMFs" Page 229) Create the application prepared by KUKA for the position and torque referencing of the LBR iiwa and integrate the safety-oriented tool into the application. Further adaptations in the application may be necessary. (>>> 13.10.2 "Torque referencing" Page 224)
8	Transfer the project with the safety configuration to the robot controller. ■ When the system software is installed or the project is synchronized
9	Reboot the robot controller to apply the changed safety configuration.
10	Activate the safety configuration on the robot controller (>>> 13.7 "Activating the safety configuration on the robot controller" Page 189)
11	When using position-based AMFs (>>> "Position-based AMFs" Page 229) Carry out position referencing.

Step	Description
12	When using axis torque-based AMFs (>>> "Axis torque-based AMFs" Page 229) Carry out torque referencing.
13	Test the safety functions of the activated safety configuration. (>>> 13.11 "Safety acceptance overview" Page 226)

13.6.2 Opening the safety configuration

Procedure

- In the Sunrise project, double-click on the file **SafetyConfiguration.sconf**.

Description

The safety configuration contains several tables.

- **KUKA PSM**

  The table contains the safety functions prescribed by KUKA. These cannot be modified or deactivated.

  The table documents the system behavior and, in conjunction with the **Customer PSM** table, provides a full description of the permanently active safety functions.
- **Customer PSM**

  The user-specific safety functions are configured in this table. It contains the safety functions preconfigured by KUKA. These can be deactivated, changed or deleted.
- **ESM**

  A table is created for each ESM state. It contains the safety functions of the state. The standard configuration does not contain any preconfigured ESM states.

13.6.2.1 Evaluating the safety configuration

When the safety configuration is evaluated, the **Customer PSM** and **KUKA PSM** tables are always checked simultaneously. It is possible for the two tables to contain identical safety functions with different reactions. If different stop reactions are configured, a violation triggers the stronger stop reaction. In other words, it triggers the stop reaction which causes an earlier safety-oriented disconnection of the drives.

If the ESM mechanism is used, all safety functions of the currently active ESM state are additionally monitored.

13.6.2.2 Overview of the graphical user interface for the safety configuration

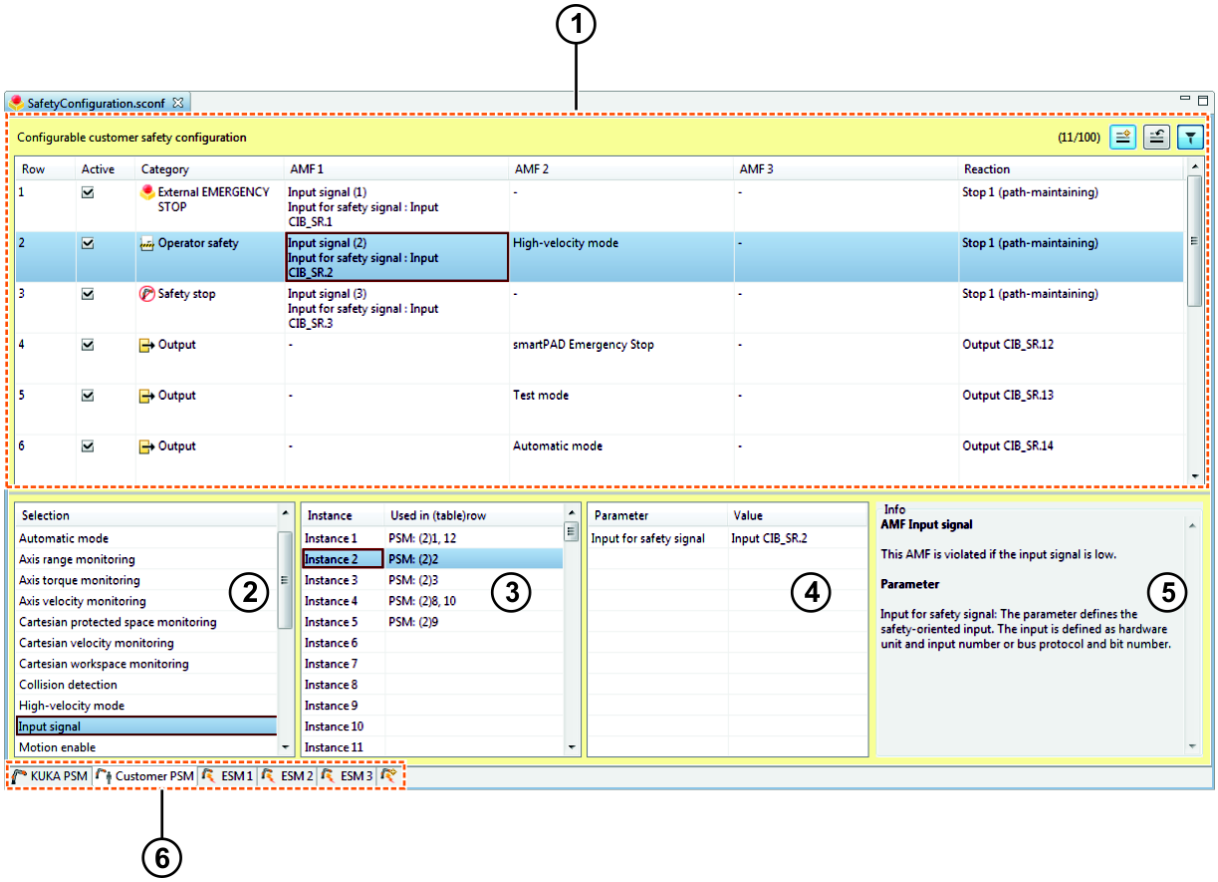


Fig. 13-3: Graphical user interface for safety configuration

Item	Description
1	Table selected Contains the configured safety functions of the selected PSM table or of the selected ESM state.
2	Main selection table With respect to the cell selected in the highlighted table row, the category, AMF or reaction of a safety function can be selected here.
3	Instance table This area displays the instances of the AMF marked in the Main selection table as well as the table rows in which they are used.
4	Parameter table The parameter values of the AMF instance selected in the Instance table are displayed here. The values can be changed.
5	Information display Displays the description of the selected category, AMF or reaction
6	List of tables In this area, the desired tables can be selected and new ESM states can be added.

The list of tables in the lower area of the Editor is used to select the table to be displayed and edited.

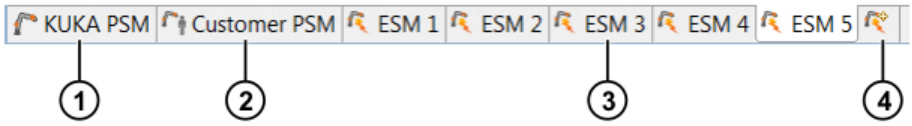


Fig. 13-4: List of tables for safety configuration

Item	Description
1	"*KUKA PSM*" tab Opens the *KUKA PSM* PSM table. It is not possible to edit the table.
2	"*Customer PSM*" tab Opens the *Customer PSM* PSM table. The table can be edited.
3	Tab for an ESM state Opens the ESM state. The ESM state can be edited.
4	**Add new ESM state** button Adds a new ESM state. The new state is automatically opened and can be edited.

13.6.3 Configuring the safety functions of the PSM mechanism

The PSM mechanism defines safety monitoring functions which are permanently active.

The safety functions are displayed in tabular form. Each row in the table contains a safety function.

In the PSM table **Customer PSM**, new safety functions are added and existing settings are adapted. This means that the category, the Atomic Monitoring Functions (AMFs) used, the parameterization of the AMF instances and the reaction can be changed. Individual safety functions can be activated or deactivated.

13.6.3.1 Opening the PSM table *Customer PSM*

Procedure

1. Open the safety configuration.
2. Select the *Customer PSM* tab from the list of tables. The PSM table *Customer PSM* is displayed and can be edited.

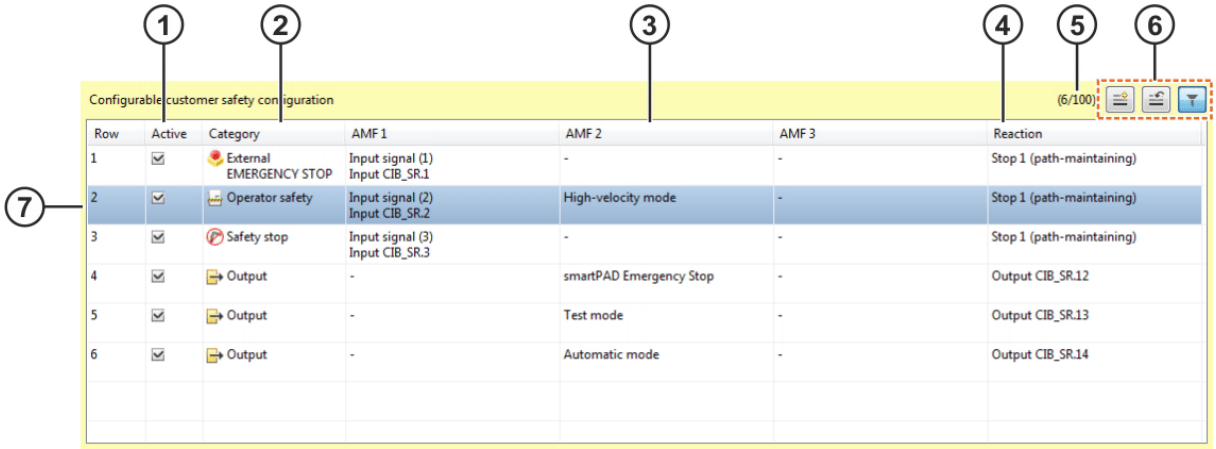


Fig. 13-5: PSM table *Customer PSM*

Item	Description
1	*Active* column Defines whether the safety function is active. Deactivated safety functions are not monitored. ■ Check mark set: safety function is active. ■ Check mark not set: safety function is deactivated.
2	*Category* column Defines the category of the safety function. In the event of an error, the category is shown on the smartHMI as the cause of error.
3	Columns *AMF 1*, *AMF 2*, *AMF 3* Define the individual AMFs of the safety function. Up to 3 AMFs can be used. The safety function is violated if all of the AMFs used are violated.
4	*Reaction* column Defines the reaction of the safety function. It is triggered if the safety function is violated.
5	Number of safety functions currently configured A total of 100 rows are available for configuring the user-specific safety monitoring functions.
6	Buttons for editing the table
7	Selected row The row containing the currently selected safety function is highlighted in gray.

The following buttons are available:

Button	Description
	Add row Adds a new row to the table (only possible when the default rows are hidden). The new row has the standard configuration and is activated automatically.
	Reset row Resets the configuration of the selected row to the standard configuration. The safety function is deactivated.
	Show empty rows / **Hide empty rows** All empty rows which are not configured are deactivated and preset with the standard configuration. ■ *Category*: None ■ *AMF 1*, *AMF 2*, *AMF 3*: None ■ *Reaction*: Stop 1 The empty rows can be shown or hidden. The empty rows are hidden by default.

13.6.3.2 Creating safety functions for the PSM mechanism

Precondition

■ The PSM table *Customer PSM* is open.

Procedure

Non-configured empty rows are displayed:

1. Select an empty row in the table.
2. Set the category, the AMFs used and the reaction of the safety function in the corresponding columns.
3. Set the check mark in the *Active* column if the row is to be activated.

Non-configured empty rows are not displayed:

1. Click on **Add row**. A preconfigured row is added to the table. The row is automatically activated (check mark in the *Active* column).
2. Set the category, the AMFs used and the reaction of the safety function in the corresponding columns.

13.6.3.3 Deleting safety functions of the PSM mechanism

Precondition

- The PSM table *Customer PSM* is open.

Procedure

1. In the table, select the row with the safety function to be deleted.
2. Click on **Reset row**. The safety function is deactivated and is given the standard configuration (None, AMF, Reaction: Stop 1).

13.6.3.4 Editing existing safety functions of the PSM mechanism

Precondition

- The PSM table *Customer PSM* is open.

Procedure

Changing the category:

1. Select the *Category* column in the desired row. The available categories are displayed in the Main Selection table.
2. Select the desired category from the Main selection table. The category is applied to the safety function.

Changing the AMF used:

1. Select the *AMF 1*, *AMF 2* or *AMF 3* column in the desired row. The available AMFs are displayed in the Main selection table.
2. Select the desired AMF from the Main selection table. The AMF is applied to the safety function.
3. For multiply instanced AMFs: select the desired instance from the Main selection table. The instance is applied to the safety function.
4. For parameterizable AMFs: in the parameter table, set the parameter of the AMF in the **Value** column and insert the settings with the Enter key.

Changing a reaction:

1. Select the *Reaction* column in the desired row. The available reactions are displayed in the Main selection table.
2. Select the desired reaction from the Main selection table. The reaction is applied to the safety function.
3. If the **Output** reaction has been selected: in the Parameter table, select the output bit whose signal is to be set to LOW if a safety function is violated. Accept the setting with the Enter key.

Activating/deactivating a safety function:

- Click on the *Active* column in the desired row. The check mark is set / removed.

> Once the safety configurations are transferred to the robot controller and activated, only the activated safety functions are available.

13.6.4 Configuring the safe states of the ESM mechanism

Using the ESM mechanism, various safety settings are defined by configurable safe states. Up to 10 safe states can be created. The states are numbered sequentially from 1 to 10 and can therefore be identified unambiguously.

A safe state is defined in a table with up to 20 safety functions. These safety functions define the safety settings which must be valid for the state.

A safe state is represented in a table. Each row in the table contains a safety function.

Use of the ESM mechanism is optional. The ESM mechanism is activated if at least one ESM state is configured. If no ESM states are configured, the mechanism is deactivated.

If the ESM mechanism is active, exactly one safe state is valid. The safety functions of this state are monitored in addition to the permanently active safety functions. Depending on the situation, it is possible to switch between the configured safe states. Switching can be carried out in the robot application or in a background task.

(>>> 13.6.4.8 "Switching between ESM states" Page 188)

An ESM state is active until it is commanded to switch to another ESM state.

The configured ESM state with the lowest number is automatically active when the controller is booted.

13.6.4.1 Adding a new ESM state

Up to 10 safe states can be created for the ESM mechanism. If this number is reached, the tab for adding new states is hidden.

Procedure

1. Open the safety configuration.
2. Select the **Add new ESM state** tab from the list of tables. A new ESM state is created.

   The new ESM state is given the lowest state number which has not yet been assigned. It has an active safety function with a standard configuration. A new tab for the state is added to the list of tables. The table for the state is automatically opened and can be edited.

13.6.4.2 Opening a table for an ESM state

Precondition

- The ESM mechanism is activated.

Procedure

1. Open the safety configuration.
2. Select the tab for the desired ESM state from the list of tables. The table for the ESM state is automatically displayed and can be edited.

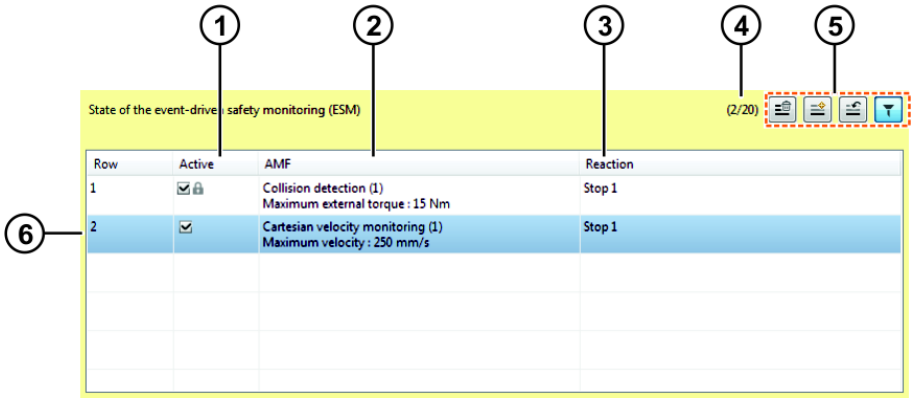


Fig. 13-6: Table for an ESM state

Item	Description
1	*Active* column Defines whether the safety function is active. Deactivated safety functions are not monitored. ■ Check mark set: safety function is active. ■ Check mark not set: safety function is deactivated. The safety function in the first row of the table is always active. It cannot be deactivated (indicated by the lock icon).
2	*AMF* column Defines the AMF of the safety function. Only one AMF is used for safety functions of ESM states. If this AMF is violated, the safety function and thus the entire state is violated.
3	*Reaction* column Defines the reaction of the safety function. It is triggered if the safety function is violated.
4	Number of safety functions currently configured A total of 20 rows are available for configuring the safety monitoring functions of an ESM state.
5	Buttons for editing the table
6	Selected row The row containing the currently selected safety function is highlighted in gray.

The following buttons are available:

Button	Description
	Delete state Deletes the entire state The delete operation must be confirmed via a dialog.
	Add row Adds a new row to the table (only possible when the default rows are hidden). The new row has the standard configuration and is activated automatically.

Button	Description
	Reset row Resets the configuration of the selected row to the standard configuration. The safety function is deactivated (exception: the first row of the table is always active).
	Show empty rows / **Hide empty rows** All empty rows which are not configured are deactivated and preset with the standard configuration. ■ *AMF*: None ■ *Reaction*: Stop 1 The empty rows can be shown or hidden. The empty rows are hidden by default.

13.6.4.3 Deleting an ESM state

Procedure

1. Open the safety configuration.
2. Select the tab for the ESM state to be deleted from the list of tables.
3. Click on **Delete state**.
4. Reply to the request for confirmation with **Yes**. The state is deleted.

Once the safety configuration is saved and closed, an ESM state is automatically removed if it has the following settings:

- All rows have the standard configuration (AMF: None, Reaction: Stop 1)
- The first row is activated and all other rows are deactivated.

13.6.4.4 Creating a safety function for the ESM state

Precondition

- The table for the desired ESM state is open.

Procedure

Non-configured empty rows are displayed:

1. Select an empty row in the table.
2. Set the AMF used and the reaction of the safety function in the corresponding columns.
3. Set the check mark in the *Active* column if the row is to be activated.

Non-configured empty rows are not displayed:

1. Click on **Add row**. A preconfigured row is added to the table. The row is automatically activated (check mark in the *Active* column).
2. Set the AMF used and the reaction of the safety function in the corresponding columns.

13.6.4.5 Deleting a safety function of an ESM state

Precondition

- The table for the desired ESM state is open.

Procedure

1. In the table, select the row with the safety function to be deleted.
2. Click on **Reset row**. The safety function is deactivated and is given the standard configuration (None, AMF, Reaction: Stop 1).

13.6.4.6 Editing an existing safety function of an ESM state

Precondition

- The table for the desired ESM state is open.

Procedure

Changing the AMF used:

1. Select the **AMF** column in the desired row. The available AMFs are displayed in the Main selection table.
2. Select the desired AMF from the Main selection table. The AMF is applied to the safety function.
3. For multiply instanced AMFs: select the desired instance from the Main selection table. The instance is applied to the safety function.
4. For parameterizable AMFs: in the parameter table, set the parameter of the AMF in the **Value** column and insert the settings with the Enter key.

Changing a reaction:

1. Select the **Reaction** column in the desired row. The available reactions are displayed in the Main selection table.
2. Select the desired reaction from the Main selection table. The reaction is applied to the safety function.

Activating/deactivating a safety function:

- Click on the **Active** column in the desired row. The check mark is set / removed.

13.6.4.7 Deactivating the ESM mechanism

Use of the ESM mechanism is optional. It can be deactivated.

Procedure

- Delete all ESM states.

13.6.4.8 Switching between ESM states

The setESMState(…) method can be used to activate an ESM state and switch between the different ESM states. The method belongs to the LBR class and can be used in robot applications or background tasks.

Syntax

lbr`.setESMState(`*state*`);`

Explanation of the syntax

Element	Description
lbr	Type: LBR Name of the robot for which the ESM state is activated
state	Type: String Number of the ESM state which is activated ■ **1 … 10** If a non-configured ESM state is specified, the robot stops with a safety stop 1.

Example

In an application, the LBR iiwa is to be guided by hand. For this purpose, a suitable start position is addressed. In order to address the start position, ESM state 3 must be activated. ESM state 3 ensures sensitive collision detection and monitors the Cartesian velocity.

Manual guidance is to begin once the start point has been reached. ESM state 8 must be activated for manual guidance. ESM state 8 requires enabling on the hand guiding device but permits a higher Cartesian velocity than ESM state 3.

```
private LBR lbr;
...
```

```
public void run() {
...
lbr.setESMState("3");
lbr.move(lin(getFrame("Start")).setCartVelocity(300));

   lbr.setESMState("8");
   lbr.move(handGuiding());
...
}
```

13.7 Activating the safety configuration on the robot controller

Description

A safety configuration is effective only after it has been transferred to the robot controller and activated on the smartHMI. If no safety configuration is active, the robot cannot be moved.

When it is activated, the safety configuration is assigned a unique ID (= checksum of the safety configuration). This is displayed in the **Safety** level in the **Activation** area. With this ID, the safety maintenance technician can clearly identify the safety configuration activated on the robot controller.

When installing the system software, the robot controller must be rebooted. If a changed safety configuration is transferred along with the installation, it must be activated after the reboot.

If changes to an active safety configuration are transferred to the robot controller by means of project synchronization, the system response depends on whether the synchronization was completed by rebooting the robot controller:

- For synchronization with reboot: the old safety configuration is no longer active and the new configuration is not yet active. The robot can no longer be moved.
- For synchronization without reboot: the old safety configuration remains active until the robot controller has been rebooted. The robot can be moved until then.

> In case the newly transferred safety configuration is not activated, the old safety configuration can be restored.
> (>>> 13.7.2 "Restoring the safety configuration" Page 190)

A password is required for activation. The default password is “argus”.

> The password to activate the safety configuration must be changed before start-up. This password may only be communicated to trained safety maintenance technicians who are authorized to activate the safety configuration.
> (>>> 13.7.3 "Changing the password for activating the safety configuration" Page 190)

Procedure

1. In the Station view, select **Safety** > **Activation**.
2. Enter the password and press **Activate**.
3. Only required if the robot controller was not rebooted after installation of the system software or after project synchronization: reboot the robot controller.

13.7.1 Deactivating the safety configuration

Description

An activated safety configuration can be deactivated again.

It is the responsibility of the safety maintenance technician to check that the deactivation has been successful. If the safety configuration is deactivated,

the robot can no longer be moved. This is displayed in a message in the Station view (**Safety** tile).

Procedure

1. In the Station view, select **Safety** > **Activation**.
2. Enter the password and press **Deactivate**.

13.7.2 Restoring the safety configuration

Description

This function is only available if a new safety configuration has been transferred to the robot controller but not activated. This procedure can be used to restore the most recently activated safety configuration.

Procedure

1. In the Station view, select **Safety** > **Activation**.
2. Enter the password and press **Reset**.
3. Only required if the robot controller was not rebooted after project synchronization: reboot the robot controller.

13.7.3 Changing the password for activating the safety configuration

Procedure

1. In the Station view, select **Safety** > **Change password**.
2. Enter the old password in the **Password** box.
3. In the boxes below enter the new password twice.

   For security reasons, the entries are displayed encrypted. Upper and lower case are distinguished.
4. Press **Change**. The new password is valid immediately.

> If the password has been forgotten, the memory card in the robot controller must be exchanged by KUKA Service. Using the default password, the password for activating the safety configuration must then be changed again.

13.8 Use and parameterization of the Atomic Monitoring Functions

The following describes the use, functional principle and parameterization of the individual AMFs. Configuration examples are added to the description.

13.8.1 Evaluating the safety equipment on the KUKA smartPAD

The KUKA smartPAD has an EMERGENCY STOP device and an enabling device. The corresponding safety-oriented functions are preconfigured in the KUKA table and cannot be changed.

Further safety functions evaluated by the safety equipment on the smartPAD can be configured. The following AMFs are available for this purpose:

AMF	Description
smartPAD Emergency Stop	The AMF is violated if the EMERGENCY STOP device on the smartPAD is pressed. Standard AMF

AMF	Description
smartPAD enabling switch inactive	The AMF is violated if no enabling signal is issued on the smartPAD (no enabling switch is pressed on the smartPAD or an enabling switch is fully pressed). Standard AMF
smartPAD enabling switch panic active	The AMF is violated if an enabling switch on the smartPAD is fully pressed (panic position). Standard AMF

13.8.2 Evaluating the operating mode

The set operating mode has a powerful effect on the behavior of the industrial robot and determines which safety precautions are required.

The following AMFs are available for configuring a safety function to evaluate the set operating mode:

AMF	Description
Test mode	This AMF is violated if a test operating mode is active (T1, T2 , CRR). Standard AMF
Automatic mode	This AMF is violated if an Automatic operating mode is active (AUT). Standard AMF
Reduced-velocity mode	This AMF is violated if an operating mode is active whose velocity is reduced to a maximum of 250mm/s (T1, CRR). Standard AMF
High-velocity mode	This AMF is violated if an operating mode is active in which the robot is moved with a programmed velocity (T2, AUT). Standard AMF

13.8.3 Evaluating the motion enable

The robot cannot be moved without the motion enable. The motion enable can be cancelled for various reasons, for example if enabling is not issued in Test mode or if the EMERGENCY STOP is pressed on the smartPAD.

The AMF for motion enable functions like a group signal for all configured stop conditions. In particular, it can be be used for switching off peripheral devices. For safety functions which receive the evaluation of the motion enable, a safe output should therefore be configured as the reaction. If a safety stop is set as the reaction, the robot cannot be moved.

AMF	Description
Motion enable	This AMF is violated if the motion enable is not issued due to a stop request. **Note**: This AMF is only suitable for use with an output as a reaction. Standard AMF

Example Switching off a tool (category: **output**)

A tool, e.g. a laser, which is connected to an output, is to be switched off when the motion enable is canceled. It is only to be switched off if the operator safety is violated.

AMF1	AMF2	AMF3	Reaction
Input signal (operator safety)	-	Motion enable	Output (tool)

13.8.4 Monitoring safe inputs

Description

The inputs of the discrete safety interface or the safe inputs of the Ethernet safety interface can be used as safe inputs as long as they are configured in WorkVisual.

(>>> "Safety interfaces" Page 170)

Safety equipment can be connected to the safe inputs, for example external EMERGENCY STOP devices or safety gates. The AMF *Input signal* is used to evaluate the associated input signal.

> If a robot with a media flange Touch is used, the safe inputs at which enabling and panic switches for the media flange are connected can be used in the AMF.

AMF	Description
Input signal	The AMF is violated if the safe input used is low (state “0”). Parameterizable AMF with 100 available instances

Parameter	Description
Input for safety signal	Safe input to be monitored

Example 1

Operator safety (category: *Operator safety*)

A safety gate is connected to a safe input. If the safety gate is opened in Automatic or T2 mode, a safety stop 1 (path-maintaining) is to be triggered.

AMF1	AMF2	AMF3	Reaction
Input signal	*High-velocity mode*	-	*Stop 1 (path-maintaining)*

Example 2

External E-STOP (category: *External EMERGENCY STOP*)

An external EMERGENCY STOP device is connected to a safe input. If the external EMERGENCY STOP device is pressed, a safety stop 1 (path-maintaining) is triggered.

AMF1	AMF2	AMF3	Reaction
Input signal	-	-	*Stop 1 (path-maintaining)*

13.8.5 Manual guidance with enabling device and velocity monitoring

An application of human-robot collaboration involves manually guiding the robot, e.g. to teach points on a path. This requires a hand guiding device with an enabling switch.

In the case of manual guidance in T1 mode, the velocity is not limited to 250 mm/s. For this reason, the velocity must be safely monitored during man-

ual guidance. The maximum permissible velocity during manual guidance must be defined in a risk assessment.

(>>> 13.8.5.3 "Velocity monitoring during manual guidance" Page 195)

> If the robot is manually guided, an EMERGENCY STOP device must be installed. It must always be within reach of the operator.

13.8.5.1 Monitoring of enabling switches on hand guiding devices

Description

The AMF *Hand guiding device enabling inactive* serves to evaluate 3-step enabling devices. Up to 3 enabling switches and 3 panic switches can be configured. 3-step enabling devices with only one output which process the panic signal internally can also be evaluated.

The AMF fulfils the following normative requirements and measures against predictable misuse:

- If the enabling switch has been fully pressed down, the signal will not be issued if the switch is released to the center position.
- The signal is cancelled in case of a stop request. To issue the signal again, the enabling switch must be released and pressed again.
- The signal is only issued 100 ms after the enabling switch has been pressed.

The following applies if several enabling switches are used:

- If all 3 enabling switches of an enabling device are held simultaneously in the center position, a safety stop 1 is triggered.
- It is possible to hold 2 enabling switches of an enabling device in the center position simultaneously for up to 15 seconds. This makes it possible to adjust grip from one enabling switch to another one. If 2 enabling switches are held simultaneously in the center position for longer than 15 seconds, this triggers a safety stop 1.
- If the enabling switches of different enabling devices are pressed simultaneously, e.g. an enabling switch on the smartPAD and an enabling switch on the hand guiding device, a safety stop 1 (path-maintaining) is triggered.

AMF	Description
Hand guiding device enabling inactive	Parameterizable AMF with one available instance The AMF is violated if all safe inputs connected to an enabling switch are LOW (state “0”) or at least one of the safe inputs connected to a panic switch is LOW (state “0”).

> If a robot with a media flange Touch is used, the safe inputs at which enabling and panic switches for the media flange are connected can be used in the AMF.

Parameter	Description
Enabling switch 1 used	Indicates whether the enabling switch is connected to a safe input ■ **true**: An input is connected. ■ **false**: No input is connected. Default: **False**
Enabling switch 2 used	
Enabling switch 3 used	

Parameter	Description
Enabling switch 1 input signal	Safe input to which the enabling switch is connected The inputs of the discrete safety interface or the safe inputs of the Ethernet safety interface can be used as safe inputs as long as they are configured in WorkVisual. (>>> "Safety interfaces" Page 170)
Enabling switch 2 input signal	
Enabling switch 3 input signal	
Panic switch 1 used	Indicates whether the panic switch is connected to a safe input ■ **true**: An input is connected. ■ **false**: No input is connected. Default: **False**
Panic switch 2 used	
Panic switch 3 used	
Panic switch 1 input signal	Safe input to which the panic switch is connected The inputs of the discrete safety interface or the safe inputs of the Ethernet safety interface can be used as safe inputs as long as they are configured in WorkVisual. (>>> "Safety interfaces" Page 170)
Panic switch 2 input signal	
Panic switch 3 input signal	

Example

Manual guidance with signal (category: *Enabling device*)

A robot equipped with a hand guiding device is to be manually guided in a defined area in order to teach the points on a path. The area for manual guidance is defined by a Cartesian protected space. In this protected space, a Cartesian velocity of 250 mm/s must not be exceeded unless the signal is issued via the hand guiding device. Otherwise, a safety stop 1 (path-maintaining) is triggered.

AMF1	AMF2	AMF3	Reaction
Hand guiding device enabling inactive	*Cartesian protected space monitoring*	*Cartesian velocity monitoring*	*Stop 1 (path-maintaining)*

13.8.5.2 Monitoring functions during manual guidance

Description

The AMF *Hand guiding device enabling active* makes it possible to implement safety functions that activate other monitoring functions during manual guidance with the enabling device, e.g. Cartesian velocity monitoring.

AMF	Description
Hand guiding device enabling active	Standard AMF This AMF is violated if the enabling signal for manual guidance is issued. The AMF *Hand guiding device enabling active* represents the inverse state of the AMF *Hand guiding device enabling inactive*: ■ The AMF *Hand guiding device enabling active* is violated if the AMF *Hand guiding device enabling inactive* is not violated. ■ The AMF *Hand guiding device enabling active* is not violated as long as the AMF *Hand guiding device enabling inactive* is violated. The AMF *Hand guiding device enabling active* takes into account the enabling device configured for the AMF *Hand guiding device enabling inactive*.

Example

Space and velocity monitoring during manual guidance with enabling device (category: *Workspace monitoring*, *Velocity monitoring*)

During manual guidance of a robot with an enabling device, the robot must not leave a defined workspace. Furthermore, the robot is to move with a maximum velocity during manual guidance of 600 mm/s. If the workspace is left while enabling is active, or if the velocity limit is exceeded, a safety stop 1 (path-maintaining) is to be executed.

AMF1	AMF2	AMF3	Reaction
Hand guiding device enabling active	*Cartesian workspace monitoring*	-	*Stop 1 (path-maintaining)*
Hand guiding device enabling active	*Cartesian velocity monitoring*	-	*Stop 1 (path-maintaining)*

13.8.5.3 Velocity monitoring during manual guidance

For manual guidance of the robot, a maximum permissible velocity must be defined that may not be exceeded during manual guidance. The value for this velocity must be defined in a risk assessment.

A safety function for safe velocity monitoring during manual guidance is configured in line 3 of the PSM table *KUKA PSM*. The safety function takes into account the enabling device configured for the AMF *Hand guiding device enabling inactive*. Once the enabling signal has been issued, a safety stop 1 (path-maintaining) is carried out if the velocity limit is exceeded.

The instance of the AMF *Cartesian velocity monitoring* that is used has the following preset parameter values:

- *Monitored kinematic system*: First kinematic system
- *Maximum velocity [mm/s]*: 250

The parameter values can be modified. (>>> 13.8.8.2 "Defining Cartesian velocity monitoring" Page 197)

13.8.6 Evaluating the position referencing

Description

Position referencing checks whether the saved zero position of the motor of an axis (= saved mastering position) corresponds to the actual mechanical zero position.

The safety integrity of the safety functions based upon this is limited until the position referencing test has been performed. This includes safely monitored Cartesian and axis-specific robot positions, Cartesian velocities, forces and collisions.

The AMF *Position referencing* can be used to check whether the position values of all axes are referenced.

AMF	Description
Position referencing	Parameterizable AMF with 4 available instances The AMF is violated if the position of at least one of the axes of the monitored kinematic system is not referenced, or if the position referencing test on at least one axis has failed.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Currently without function ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function

Example

Monitoring the position referencing status (category: *Safety stop*)

To prevent danger in the case of failure or non-execution of a position referencing test, a robot whose axes are not referenced may only be moved at a reduced maximum velocity of 250 mm/s.

In order to guarantee this, the referencing status of all axes in the operating modes with high velocity (T2 and AUT) is monitored and a safety stop 1 (path-maintaining) is triggered as soon as the position of one axis has not been successfully referenced.

AMF1	AMF2	AMF3	Reaction
High-velocity mode	-	*Position referencing*	*Stop 1 (path-maintaining)*

13.8.7 Evaluating the torque referencing

Description

The referencing test of the joint torque sensors checks whether the expected external torque, which can be calculated for an axis based on the robot model and the given load data, corresponds to the value determined on the basis of the measured value of the joint torque sensor. If the difference between these values exceeds a certain tolerance value, the referencing of the torque sensors has failed.

The safety integrity of the safety functions based upon this is limited until the torque referencing test has been performed successfully. This includes axis torque and TCP force monitoring as well as collision detection.

The AMF *Torque referencing* can be used to check whether the joint torque sensors of all axes are referenced.

AMF	Description
Torque referencing	Parameterizable AMF with 4 available instances This AMF is violated if the joint torque sensor of at least one axis of the monitored kinematic system is not referenced, or if the referencing test of at least one joint torque sensor has failed.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Currently without function ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function

Example

Monitoring the referencing status (category: *Safety stop*)

A safe collision detection is configured for a station. If a torque of more than 20 Nm is detected in at least one axis of the robot, a safety stop 0 is triggered. Since the safety integrity of this function is only ensured for successfully refer-

enced joint torque sensors, the referencing status of the sensors must be monitored simultaneously. As soon as at least one joint torque sensor has not been referenced or referencing has failed, a safety stop 1 (path-maintaining) is to be triggered in high-velocity operating modes (T2 and AUT).

AMF1	AMF2	AMF3	Reaction
Collision detection	-	-	*Stop 0*
High-velocity mode	-	*Torque referencing*	*Stop 1 (path-maintaining)*

13.8.8 Velocity monitoring functions

A moving kinematic system always presents a danger to persons in its vicinity. In order to protect persons, it may be necessary to impose a defined maximum velocity, for example to give persons time to move out of the way of the robot. This means that the velocity must be monitored continuously.

The axis-specific velocities and the Cartesian velocity of a kinematic system can be monitored.

13.8.8.1 Defining axis-specific velocity monitoring

The AMF *Axis velocity monitoring* is used to define an axis-specific velocity monitoring function.

AMF	Description
Axis velocity monitoring	Parameterizable AMF with 16 available instances The AMF is violated if the absolute velocity of the monitored axis of the monitored kinematic system exceeds the configured limit.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Mobile platform ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function
Monitored axis	Axis of the kinematic system to be monitored ■ **Axis1 … Axis16** **Axis1 … Axis7** are used for an LBR iiwa. In the case of a mobile platform, the axes are assigned as follows: ■ **Axis 1**: front left drive ■ **Axis 2**: front right drive ■ **Axis 3**: rear left drive ■ **Axis 4**: rear right drive
Maximum velocity [°/s]	Maximum permissible velocity at which the monitored axis may move in the positive and negative direction of rotation ■ **1 … 500°/s**

13.8.8.2 Defining Cartesian velocity monitoring

Description The AMF *Cartesian velocity monitoring* is used to define a Cartesian velocity monitoring function.

- In the case of a robot, the translational Cartesian velocity is monitored at all axis center points as well as at the robot flange.

  If a safety-oriented tool is active on the robot controller, the system also monitors the velocity at the center points of the spheres which are used to configure the tool.

  (>>> 9.3.7 "Safety-oriented tool" Page 137)

> The system does not monitor the entire structure of the robot and tool against the violation of a velocity limit, but rather only the monitoring spheres. In particular with protruding tools and workpieces, the monitoring spheres of the safety-oriented tool must be planned and configured in such a way as to assure the safety integrity of the velocity monitoring.

- In the case of a mobile platform, the translational Cartesian velocity is monitored at 4 defined points near the corners of the platform. Only velocity components within the plane of the platform are taken into consideration.

> If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be taken into consideration that the Cartesian velocity of the monitoring spheres relative to the carrier kinematic system is monitored and not the absolute velocity of the monitoring spheres in space.

AMF	Description
Cartesian velocity monitoring	Parameterizable AMF with 30 available instances The AMF is violated if the Cartesian translational velocity at at least one point of the monitored kinematic system exceeds the defined limit. The AMF is additionally violated in the following cases: ■ An axis is not mastered. ■ The referencing of a mastered axis has failed. **Note**: If an AMF is violated due to loss of mastering, the robot can only be moved and mastered again by switching to CRR mode.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Mobile platform ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function
Maximum velocity [mm/s]	Maximum permissible Cartesian velocity which must not be exceeded at any monitored point ■ **1 … 10,000 mm/s**

Example

Category: *Velocity monitoring*

If a Cartesian workspace is violated, the Cartesian velocity of the robot must not exceed 300 mm/s. If this velocity is exceeded, a safety stop 1 is triggered.

AMF1	AMF2	AMF3	Reaction
Cartesian workspace monitoring	-	*Cartesian velocity monitoring*	*Stop 1*

13.8.8.3 Direction-specific monitoring of Cartesian velocity

Description

The AMF *Tool-related velocity component* is used to check whether the Cartesian translational velocity in a specific direction is below the configurable limit value.

The AMF can be used, for example, to ensure that the velocity in the working direction of a sharp-pointed tool is not too high. The AMF can also be used to monitor the motion direction.

The monitored kinematic system must be specified in the configuration of the AMF. The AMF can be applied to robots (**First kinematic system**) and mobile platforms (**Second kinematic system**).

The AMF monitors the velocity at a reference frame on the safety-oriented tool. The position and orientation of the reference frame are defined in the properties of the tool by means of safety-oriented frames.

The following safety parameters are available for this in the properties of the safety-oriented tool:

- **Point for tool-related velocity**: The safety-oriented frame set here determines the position of the reference frame.
- **Orientation for tool-related velocity**: The safety-oriented frame set here determines the orientation of the reference frame.

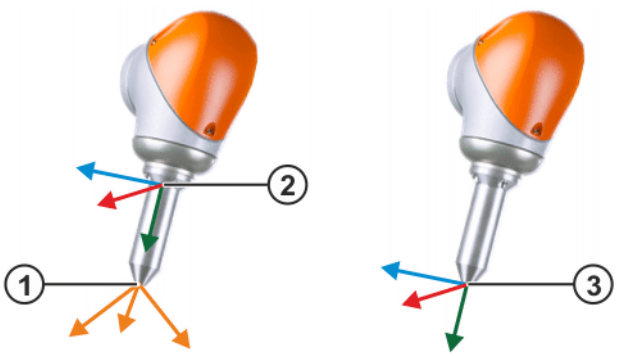


Fig. 13-7: Reference frame for tool-specific velocity

1 Point for the tool-specific velocity
2 Orientation for the tool-specific velocity
3 Position and orientation of the reference frame for the tool-specific velocity (combination of 1 and 2)

(>>> 9.3.7.1 "Defining a safety-oriented tool" Page 138)

If the monitored kinematic system is a mobile platform, it is assumed, when defining the reference frame, that the safety-oriented tool is fastened at the center point of the platform.

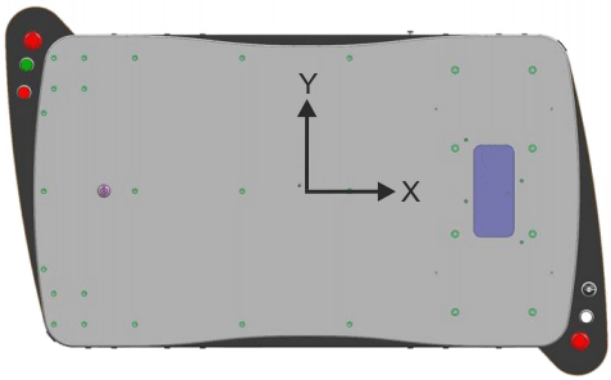


Fig. 13-8: Reference frame at the center point of the mobile platform

If no safety-oriented tool is active, the following reference frame is used depending on the monitored kinematic system:

- Robot (**First kinematic system**): frame at the center point of the robot flange
- Mobile platform (**Second kinematic system**): frame at the center point of the platform

The component of the velocity vector in a specific direction of the reference frame is monitored. During configuration of the AMF, this direction is specified as a component of the velocity vector in the coordinate system of the reference frame. One of the total of 6 components of the coordinate system (X, Y and Z components, each in the positive and negative direction) can be selected.

Furthermore, the maximum velocity that the selected component of the velocity vector must not exceed is also defined.

If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be taken into consideration that the velocity of the monitored point relative to the carrier kinematic system is monitored and not the absolute velocity in space.

AMF	Description
Tool-related velocity component	Parameterizable AMF with 30 available instances The AMF is violated if the configured component of the velocity vector in the coordinate system of the reference frame of the monitored kinematic system exceeds the maximum defined value. In the case of an LBR iiwa, the AMF is additionally violated in the following cases: - An axis is not mastered. - The referencing of a mastered axis has failed.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Mobile platform ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function
Maximum velocity [mm/s]	Maximum Cartesian velocity for the monitored component of the velocity vector ■ **1 … 10000 mm/s** **Note:** When selecting the maximum velocity, it must be noted that, particularly in the case of highly dynamic motions, low velocities against the commanded direction of motion may occur due to overshoot. For this reason, the maximum velocity should not be set too low.
Component of the velocity vector	Monitored component of the velocity vector (direction of monitoring) ■ **X positive** or **X negative** ■ **Y positive** or **Y negative** ■ **Z positive** or **Z negative**

Example 1

A sharp-pointed tool may be moved in its working direction at a maximum of 25 mm/s. For this, the tool is marked as safety-oriented and the tool tip is created as a safety-oriented frame. This frame is used to define the position and orientation of the reference frame in the properties of the safety-oriented tool.

An instance of the AMF *Tool-related velocity component* is configured in such a way that the positive Z component of the velocity vector in the coordinate system of the tool tip may not exceed a value of 25 mm/s. For this, the following parameters are set for the instance used:

- *Monitored kinematic system*: First kinematic system
- *Maximum velocity [mm/s]*: 25
- *Component of the velocity vector*: Z positive

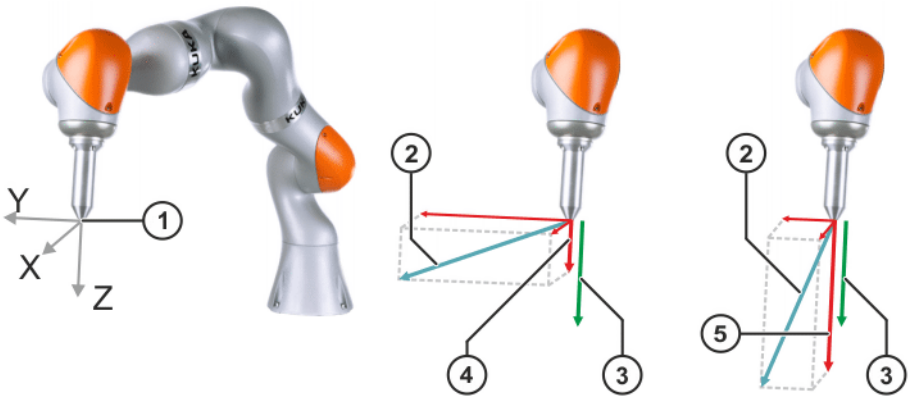


Fig. 13-9: Velocity monitoring in the tool direction

Item	Description
1	Reference frame for the tool-specific velocity component
2	Velocity vector of the translational Cartesian velocity
3	Maximum permissible velocity for the positive Z component of the velocity vector

Item	Description
4	Positive Z component of the velocity vector The velocity is below the maximum permissible velocity; the AMF is not violated.
5	Positive Z component of the velocity vector The velocity is above the maximum permissible velocity; the AMF is violated.

The safety function configured with the AMF monitors the positive Z component of the velocity vector. If the maximum velocity of 25 mm/s is exceeded by the monitored component in Automatic mode, a safety stop 1 (path-maintaining) is to be executed.

Category: *Velocity monitoring*

AMF1	AMF2	AMF3	Reaction
Automatic mode	*Tool-related velocity component (1)* *First kinematic system*	-	*Stop 1 (path-maintaining)*

Example 2

In order to keep the dimensions of the protected space of a mobile platform to the rear and to both sides as small as possible, the direction of motion of the platform must be monitored in such a way that only forward motions can be carried out at high velocity.

Configuration:

- 3 instances of the AMF *Tool-related velocity component* are required. Reference frame is the center point of the platform in all cases.
- The motion to the left and right is to be carried out with a maximum velocity of 50 mm/s. For this, the positive and negative Y components of the velocity vector are limited to 50 mm/s.
- The backward motion is to be carried out with a maximum velocity of 20 mm/s. For this, the negative X component of the velocity vector is limited to 20 mm/s.

Parameterization of the configured instances:

- Instance 1:
  - *Monitored kinematic system*: Second kinematic system
  - *Maximum velocity [mm/s]*: 50
  - *Component of the velocity vector*: Y positive
- Instance 2:
  - *Monitored kinematic system*: Second kinematic system
  - *Maximum velocity [mm/s]*: 50
  - *Component of the velocity vector*: Y negative
- Instance 3:
  - *Monitored kinematic system*: Second kinematic system
  - *Maximum velocity [mm/s]*: 20
  - *Component of the velocity vector*: X negative

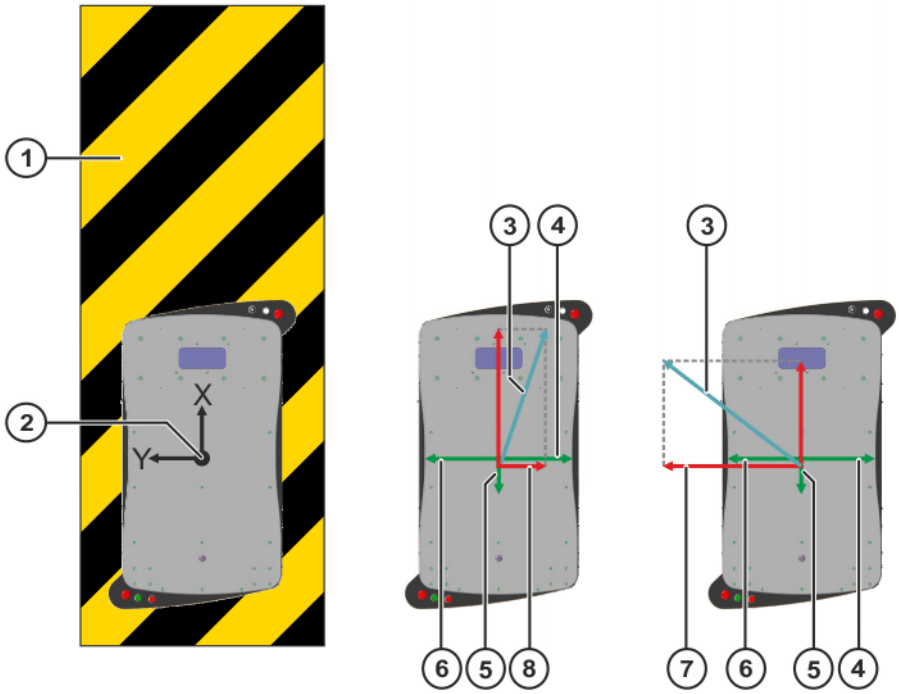


Fig. 13-10: Protected space limitation by means of velocity monitoring in the direction of motion

Item	Description
1	Approximate dimensions of the desired protected space
2	Reference frame for the tool-specific velocity component
3	Velocity vector of the translational Cartesian velocity
4	Maximum permissible velocity for the negative Y component of the velocity vector
5	Maximum permissible velocity for the negative X component of the velocity vector
6	Maximum permissible velocity for the positive Y component of the velocity vector
7	Positive Y component of the velocity vector The velocity is above the maximum permissible velocity of instance 1; the AMF is violated.
8	Negative Y component of the velocity vector The velocity is below the maximum permissible velocity of instance 3; none of the 3 instances of the AMF is violated.

3 safety functions are configured, each of which uses one of the 3 instances. If the configured maximum velocity of is exceeded in at least one of the 3 monitored components in Automatic mode, a safety stop 1 (path-maintaining) is to be executed.

Category: *Velocity monitoring*

AMF1	AMF2	AMF3	Reaction
Automatic mode	*Tool-related velocity component (1)* *Second kinematic system*	-	*Stop 1 (path-maintaining)*
Automatic mode	*Tool-related velocity component (2)* *Second kinematic system*	-	*Stop 1 (path-maintaining)*
Automatic mode	*Tool-related velocity component (3)* *Second kinematic system*	-	*Stop 1 (path-maintaining)*

13.8.9 Monitoring spaces

Description

The robot environment can be divided into areas in which it must remain for execution of the application, and areas which it must not enter or may only enter under certain conditions. The system must then continuously monitor whether parts of the robot structure are within or outside of such a monitoring space.

A monitoring space can be defined as a Cartesian cuboid or by means of individual axis ranges.

A Cartesian monitoring space can be configured as a workspace in which the robot must remain, or as a protected space which it must not enter.

Via the link to other Atomic Monitoring Functions, it is possible to define further conditions which must be met when a monitoring space is violated. For example, a monitoring space can be activated by a safe input or applicable in Automatic mode only.

> If the robot has violated a safely monitored space and been stopped by the safety controller, it can only be moved out of the violated area in CRR mode.
> (>>> 6.9 "CRR mode – controlled robot retraction" Page 78)

Spheres on the robot

Spheres are modeled around selected points on the robot, enclosing and moving with the robot. These spheres are predefined and are monitored against the limits of activated Cartesian monitoring spaces by default.

The centers and radii of the monitored spheres are defined in the machine data of the robot. A sphere is defined for each robot axis, for the robot base and for the robot flange. The sphere center lies on the center point of each axis, of the robot base and of the robot flange.

The dimensions of the monitored spheres vary according to robot type and the media flange used:

- r = sphere radius
- z, y = sphere center point relative to the robot base coordinate system

Variant 1: LBR iiwa 7 R800 with media flange Touch

	Base	A1	A2	A3	A4	A5	A6	A7	Flange
r [mm]	135	90	125	90	125	90	80	85	65
z [mm]	50	90	340	538	740	935	1140	1130	1240
y [mm]							-30		

Variant 2: LBR iiwa 7 R800 with media flange (all variants except media flange Touch)

	Base	A1	A2	A3	A4	A5	A6	A7	Flange
r [mm]	135	90	125	90	125	90	80	85	65
z [mm]	50	90	340	538	740	935	1140	1130	1220
y [mm]							-30		

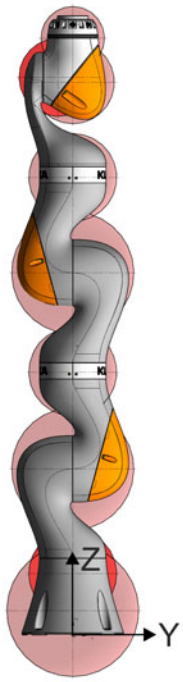


Fig. 13-11: Spheres on the LBR iiwa 7 R800 (variant 2)

Variant 3: LBR iiwa 14 R820 with media flange Touch

	Base	A1	A2	A3	A4	A5	A6	A7	Flange
r [mm]	150	100	140	90	131	90	80	85	65
z [mm]	50	160	360	580	780	980	1180	1170	1280
y [mm]							-30		

Variant 4: LBR iiwa 14 R820 with media flange (all variants except media flange Touch)

	Base	A1	A2	A3	A4	A5	A6	A7	Flange
r [mm]	150	100	140	90	131	90	80	85	65
z [mm]	50	160	360	580	780	980	1180	1170	1260
y [mm]							-30		

Spheres on tool

If a safety-oriented tool is active on the robot controller, the system not only monitors the spheres on the robot by default but also the spheres used to configure the safety-oriented tool.

(>>> 9.3.7 "Safety-oriented tool" Page 137)

> The system does not monitor the entire structure of the robot and tool against the violation of a space, but rather only the monitoring spheres. In particular with protruding tools and workpieces, the monitoring spheres of the safety-oriented tool must be planned and configured in such a way as to assure the safety integrity of workspaces and protected spaces.

Selecting monitoring spheres

It is not necessary or appropriate to include all robot and tool spheres in the Cartesian workspace monitoring of every application.

Example: If the entry of a tool into a protected space is programmed to activate further monitoring functions, only the tool spheres must be monitored.

The structure to be monitored can be selected when configuring Cartesian monitoring spaces:

- Robot and tool (default)
- Only tool
- Only robot

Stopping distance

If the robot is stopped by a monitoring function, it requires a certain stopping distance before coming to a standstill.

The stopping distance depends primarily on the following factors:

- Robot type
- Velocity of the robot
- Position of the robot axes
- Payload

> **WARNING** The stopping distance of the robot depends primarily on the dynamic characteristics of the robot type. The robot's acceleration within the reaction time of the monitoring functions varies depending on the robot type in the event of an error. This affects the actual stopping distance.
> This aspect must be taken into account by the system integrator during parameterization of the monitoring functions as part of the safety assessment.

13.8.9.1 Defining Cartesian workspaces

Description

A Cartesian workspace is defined as a cuboid whose position and orientation in space are defined relative to the world coordinate system.

These monitoring spheres are monitored against the limits of activated Cartesian workspaces and must move within these workspaces.

The AMF *Cartesian workspace monitoring* is used to define a Cartesian workspace. The AMF is violated as soon as one of the monitored spheres is no longer completely within the defined workspace.

The AMF is additionally violated in the following cases:

- An axis is not mastered.
- The referencing of a mastered axis has failed.

8 instances are available for this parameterizable AMF, e.g. a maximum of 8 Cartesian workspaces can be configured.

If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be taken into consideration that the position and orientation of the monitoring space are relative to the world coordinate system and are thus defined relative to the alignment of the base of the monitored kinematic system. For this reason, the monitoring space is also moved in the event of a change in position or inclination of the carrier kinematic system.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Currently without function ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function
Monitored structure	Structure to be monitored ■ *Robot and tool*: The spheres on the robot and the spheres used to configure the safety-oriented tool are monitored. (Default) ■ *Robot*: The spheres on the robot are monitored. ■ *Tool*: The spheres used to configure the safety-oriented tool are monitored. **Note**: If no safety-oriented tool is configured and the tool is selected as the structure to be monitored, the sphere on the robot flange is monitored. (>>> "Spheres on the robot" Page 204)

One corner of the cuboid is defined relative to the world coordinate system. This is the origin of the workspace and is defined by the following parameters:

Parameter	Description
X, **Y**, **Z [mm]**	Offset of the origin of the workspace along the X, Y and Z axes of the world coordinate system ■ **-100,000 mm … +100,000 mm**
A, **B**, **C [°]**	Orientation of the origin of the workspace about the axes of the world coordinate system, specified by the rotational angles A, B, C ■ **0° … 359°**

Based on this defined origin, the size of the workspace is determined along the coordinate axes:

Parameter	Description
Length [mm]	Length of the workspace (= distance along the positive X axis of the origin) ■ **0 mm … 100,000 mm**
Width [mm]	Width of the workspace (= distance along the positive Y axis of the origin) ■ **0 mm … 100,000 mm**
Height [mm]	Height of the workspace (= distance along the positive Z axis of the origin) ■ **0 mm … 100,000 mm**

The violation of a Cartesian workspace is only rectified when all monitored spheres have returned to within the workspace limits with a minimum distance of 10 mm to these limits.

Example

The diagram shows an example of a Cartesian workspace. Its origin is offset in the negative X and Y directions with reference to the world coordinate system.

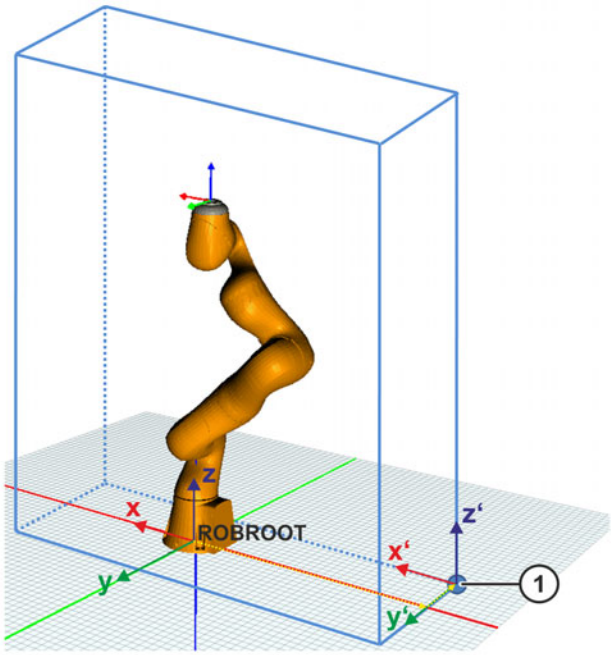


Fig. 13-12: Example of a Cartesian workspace

1 Origin of the workspace

13.8.9.2 Defining Cartesian protected spaces

Description

A Cartesian protected space is defined as a cuboid whose position and orientation in space are defined relative to the world coordinate system.

These monitoring spheres are monitored against the limits of activated protected spaces and must move outside of these protected spaces.

The AMF *Cartesian protected space monitoring* is used to define a Cartesian protected space. The AMF is violated as soon as one of the monitored spheres is no longer completely outside of the defined protected space.

The AMF is additionally violated in the following cases:

- An axis is not mastered.
- The referencing of a mastered axis has failed.

8 instances are available for this parameterizable AMF, e.g. a maximum of 8 Cartesian protected spaces can be configured.

If a very narrow protected space is configured, the robot may be able to move into and out of the protected space without the space violation being detected. Possible cause: Due to a very high tool velocity, the protected space is only violated during a very short interval.

Assuming that the following minimum values are configured:

- Radius of tool sphere: 25 mm
- Thickness of protected space: 0 mm

In this case, tool velocities of over 4 m/s would be required for the robot to pass through the protected space without detection.

The following measures are recommended in order to prevent robots from passing through protected spaces undetected:

- Configure Cartesian velocity monitoring of 4 m/s.
- OR: When configuring the protected space, select sufficient values for the length, width and height of the protected space.
- OR: When configuring the tool spheres, select sufficient values for the radius.

If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be taken into consideration that the position and orientation of the monitoring space are relative to the world coordinate system and are thus defined relative to the alignment of the base of the monitored kinematic system. For this reason, the monitoring space is also moved in the event of a change in position or inclination of the carrier kinematic system.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Currently without function ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function
Monitored structure	Structure to be monitored ■ *Robot and tool*: The spheres on the robot and the spheres used to configure the safety-oriented tool are monitored. (Default) ■ *Robot*: The spheres on the robot are monitored. ■ *Tool*: The spheres used to configure the safety-oriented tool are monitored. **Note**: If no safety-oriented tool is configured and the tool is selected as the structure to be monitored, the sphere on the robot flange is monitored. (>>> "Spheres on the robot" Page 204)

One corner of the cuboid is defined relative to the world coordinate system. This is the origin of the protected space and is defined by the following parameters:

Parameter	Description
X, **Y**, **Z [mm]**	Offset of the origin of the protected space along the X, Y and Z axes of the world coordinate system. ■ **-100,000 mm … +100,000 mm**
A, **B**, **C [°]**	Orientation of the origin of the protected space about the axes of the world coordinate system, specified by the rotational angles A, B, C ■ **0° … 359°**

Based on this defined origin, the size of the protected space is determined along the coordinate axes:

Parameter	Description
Length [mm]	Length of the protected space (= distance along the positive X axis of the origin) ■ **0 mm … 100,000 mm**
Width [mm]	Width of the protected space (= distance along the positive Y axis of the origin) ■ **0 mm … 100,000 mm**
Height [mm]	Height of the protected space (= distance along the positive Z axis of the origin) ■ **0 mm … 100,000 mm**

The violation of a Cartesian protected space is only rectified when all monitored spheres have returned to outside the protected space limits with a minimum distance of 10 mm to these limits.

Example

The diagram shows an example of a Cartesian protected space. Its origin is offset in the negative X and positive Y directions with reference to the world coordinate system.

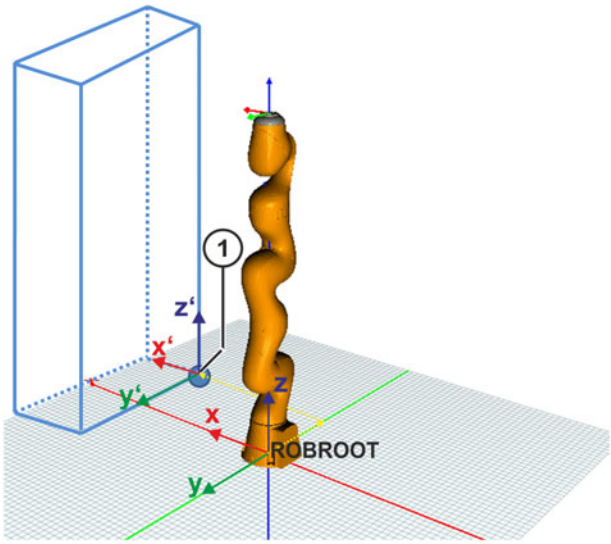


Fig. 13-13: Example of a Cartesian protected space

1 Origin of the protected space

13.8.9.3 Defining axis-specific monitoring spaces

Description

The axis limits can be defined individually and safely monitored for each axis. The axis angle must lie within the defined axis range.

The AMF *Axis range monitoring* is used to define an axis-specific monitoring space. The AMF is violated if an axis is not inside the defined axis range.

The AMF is additionally violated in the following cases:

- An axis is not mastered.
- The referencing of a mastered axis has failed.

16 instances are available for this parameterizable AMF, e.g. a maximum of 16 Cartesian workspaces can be configured.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Currently without function ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function
Monitored axis	Axis to be monitored ■ **Axis1 … Axis16** **Note**: **Axis1 … Axis7** are used for an LBR iiwa.
Lower limit [°]	Lower limit of the allowed axis range in which the monitored axis may move ■ **-180° … +180°**
Upper limit [°]	Upper limit of the allowed axis range in which the monitored axis may move ■ **-180° … +180°**

The permissible axis range runs in the positive direction of rotation of the axis from the upper to the lower limit.
If the axis position at ±180° lies within the permissible angle range, the lower limit must be greater than the upper limit.

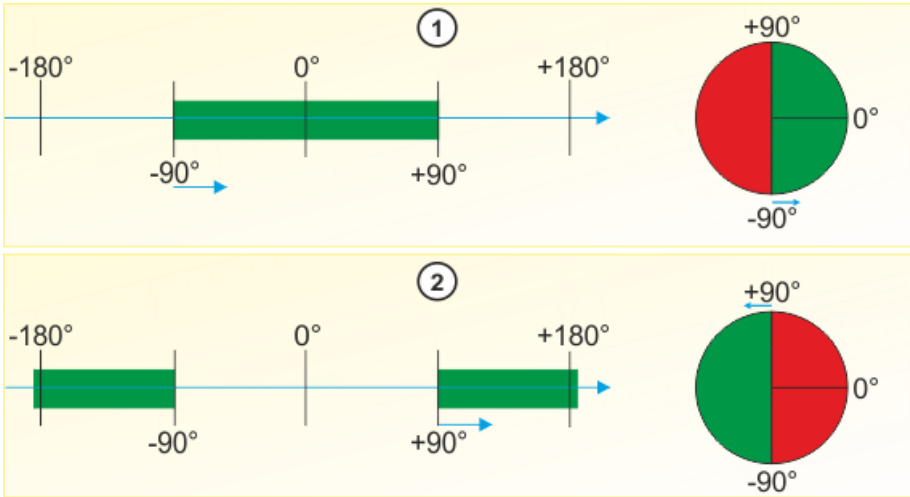


Fig. 13-14: Examples of axis-specific workspaces

1 Lower limit: -90°; Upper limit: +90°
2 Lower limit: +90°; Upper limit: -90°

For personnel protection, only the position of the axis is relevant. For this reason, the positions are converted to the axis range **-180° … +180°**, even for axes which can rotate more than 360°.

Example

Axes A1, A2 and A4 are to be monitored so that the robot may only be moved in a limited space. The monitoring is activated by a safe input. The permitted range of each axis is defined by an upper and lower limit, and is shown in green in the corresponding chart in the PSM table.

As soon as one of the monitored axis ranges is violated, a safety stop 1 (path-maintaining) is triggered. For this purpose, an individual table row must be used for each axis.

Configurable customer safety configuration (11/100)

Row	Active	Category	AMF 1	AMF 2	AMF 3	Reaction
8	☑	Workspace monitoring	Input signal (4) Input for safety signal : Input CIB_SR.4	Axis range monitoring (1) Monitored axis : Axis 1	-	Stop 1 (path-maintaining)
9	☑	Workspace monitoring	Input signal (4) Input for safety signal : Input CIB_SR.4	Axis range monitoring (2) Monitored axis : Axis 2	-	Stop 1 (path-maintaining)
10	☑	Workspace monitoring	Input signal (4) Input for safety signal : Input CIB_SR.4	Axis range monitoring (3) Monitored axis : Axis 4	-	Stop 1 (path-maintaining)

Fig. 13-15: PSM table – simultaneous monitoring of 3 axes

13.8.10 Monitoring the tool orientation

The AMF *Tool orientation* can be used to monitor the orientation of the safety-oriented tool. It checks whether a specific axis of the tool orientation frame is within a permissible direction range.

This function can for example be used to prevent dangerous parts of the mounted tool, e.g. sharp edges, from pointing towards humans in HRC applications.

It monitors the orientation of the Z axis of the tool orientation frame of the safety-oriented tool. If no safety-oriented tool is configured, the Z axis of the flange coordinate system is monitored.

(>>> 9.3.7.1 "Defining a safety-oriented tool" Page 138)

The permissible range for the orientation angle is defined by a reference vector with a fixed orientation relative to the world coordinate system and a permissible deviation angle of this reference vector.

The reference vector is defined by the rotation of the unit vector of the Z axis of the world coordinate system about the 3 Euler angles A, B and C relative to the world coordinate system. A monitoring cone is extended around the reference vector. The opening of the cone is defined by a configurable deviation angle. The deviation angle defines the permissible angle between the tool orientation and reference vector. The values of the angle of the reference vector and the deviation angle are defined in the parameterization of the AMF.

The monitoring sphere defines the permissible range for the tool orientation.

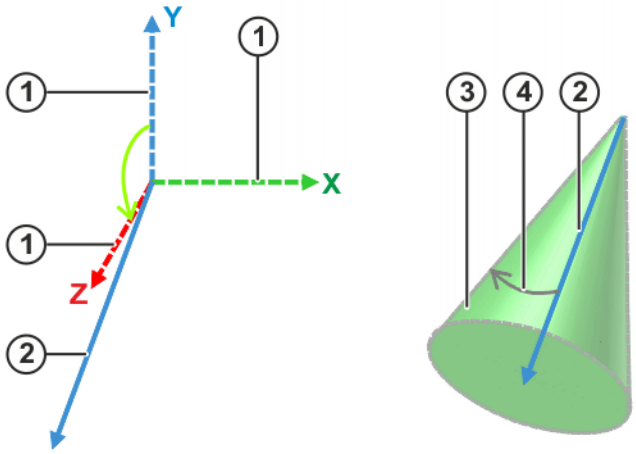


Fig. 13-16: Monitoring cone for tool orientation

Item	Description
1	Axes of the world coordinate system
2	Reference vector The reference vector defines a fixed orientation relative to the world coordinate system.

Item	Description
3	Monitoring cone Defines the permissible range for the tool orientation.
4	Deviation angle The deviation angle determines the opening of the monitoring cone.

If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform), it must be taken into consideration that the orientation of the reference vector is relative to the world coordinate system and is thus defined relative to the alignment of the base of the monitored kinematic system. For this reason, the reference orientation is also moved in the event of a change in inclination of the carrier kinematic system (e.g. due to driving up a ramp).

AMF	Description
Tool orientation	Parameterizable AMF with 8 available instances The AMF is violated if the angle between the reference vector and Z axis of the tool orientation frame is greater than the configured deviation angle. The AMF is additionally violated in the following cases: ■ An axis is not mastered. ■ The position referencing of a mastered axis has failed.

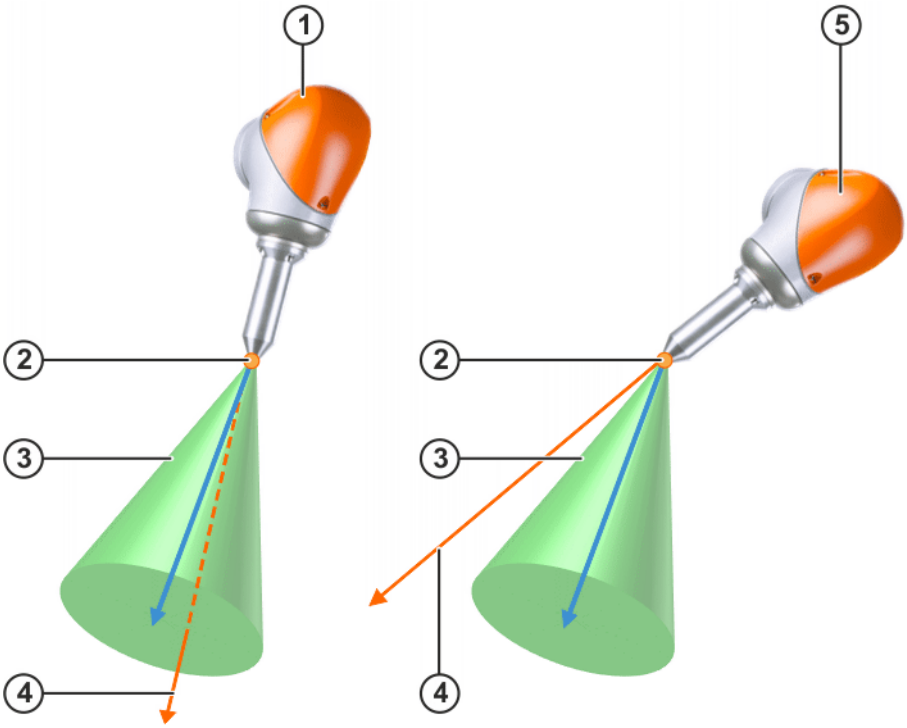


Fig. 13-17: Tool orientation (not violated and violated)

Item	Description
1	Robot is not violating the AMF *Tool orientation*. The Z axis of the tool orientation frame is within the range defined by the monitoring cone.
2	Origin of the tool orientation frame

Item	Description
3	Monitoring cone
4	Z axis of the tool orientation frame
5	Robot is violating the AMF *Tool orientation*. The Z axis of the tool orientation frame is outside of the range defined by the monitoring cone.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Currently without function ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function
A [°]	Rotation of the reference vector about the Z axis of the world coordinate system ■ **0° … 359°**
B [°]	Rotation of the reference vector about the Y axis of the world coordinate system ■ **0° … 359°**
C [°]	Rotation of the reference vector about the X axis of the world coordinate system ■ **0° … 359°**
Operating angle [°]	Workspace of the tool orientation Defines the maximum permissible deviation angle between the reference vector and the Z axis of the tool orientation frame. ■ **1° … 179°**

13.8.11 Standstill monitoring (safe operational stop)

Description

If, under certain conditions, the robot must not move but must remain under servo-control, the standstill of all axes must be safely monitored. The AMF *Standstill monitoring of all axes* is used for this purpose.

This AMF is an Extended AMF, meaning that the monitoring only begins when all other AMFs of the safety function are violated.

> Extended AMFs are not available for the safety functions of the ESM mechanism.

Standstill is defined as retaining the axis positions. At the start of standstill monitoring, the axis positions are saved and compared to the current joint values for as long as the monitoring is active.

Since standstill monitoring is monitored in a narrow tolerance range, monitoring can also be violated if the motion of the robot is caused by an outside force, for example if the robot is jolted.

AMF	Description
Standstill monitoring of all axes	Parameterizable extended AMF with 8 available instances. The AMF is violated as soon as the joint value of an axis is outside of a tolerance range of +/- 0.1° of the value saved when standstill monitoring was activated, or if one of the axes moves at an absolute value of more than 1 °/s.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Currently without function ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function

Example

Category: *Safe operational stop*

If the robot is situated outside of its workspace, it must be assured that the robot is no longer moving as soon as persons are in its vicinity. The workspace is configured by means of a Cartesian workspace. There is a sensor connected to a safe input which detects persons at risk. If both the workspace and the input signal are violated, the standstill monitoring is activated.

AMF1	AMF2	AMF3	Reaction
Input signal	*Cartesian workspace monitoring*	*Standstill monitoring of all axes*	*Stop 1*

13.8.12 Activation delay for safety functions

Description

The AMF *Time delay* can be used to delay the triggering of the reaction of a safety function for a defined time.

This AMF is an extended AMF, meaning that the delay time only starts running when all other AMFs of the safety function are violated.

> Extended AMFs are not available for the safety functions of the ESM mechanism.

AMF	Description
Time delay	Parameterizable extended AMF with 16 available instances. This AMF is violated if the set time has expired.

> If the same instance of the AMF is used for several safety functions, the delay time begins running from the first activation.

Parameter	Description
Delay time	Amount of time by which the triggering of the reaction of a safety function is delayed. ■ **12 ms … 24 h** The time can be entered in milliseconds (ms), seconds (s), minutes (min) and hours (h). Each delay is a multiple of 12 ms, meaning that it is rounded up to the next multiple of 12.

Example

Category: *Safety stop*

A robot whose axes are not referenced is to be allowed to be moved in Automatic mode for a limited time. Once this time has elapsed, for example after 2 hours, a safety stop 1 (path-maintaining) is triggered.

AMF1	AMF2	AMF3	Reaction
Automatic mode	*Position referencing*	*Time delay*	*Stop 1 (path-maintaining)*

13.8.13 Monitoring of forces and torques

The LBR iiwa is fitted with position and joint torque sensors in all axes. These make it possible to measure and react to external forces and torques.

When using the AMFs *Collision detection*, *TCP force monitoring* and *Axis torque monitoring* it must be taken into account that, due to the stopping distances of the robot, the interaction forces may continue to increase if the AMF is violated and a safety stop is triggered. For this reason it is advisable to carry out collaborative operation (HRC) at reduced velocities only, i.e. to combine these AMFs with the AMF *Cartesian velocity monitoring*, *Axis velocity monitoring* or *Tool-related velocity component*.

When using the AMFs *Collision detection* and *TCP force monitoring*, it is important to note that external forces on the robot structure with short distances to the robot axes only cause slight external torques in the robot axes under certain circumstances. This can pose a safety risk particularly in potential crushing situations during collaborative operation. Incidents of crushing may occur due to jamming within the robot structure or between the robot and the environment.
It is therefore advisable to avoid potentially critical incidents of crushing by using suitable equipment for the robot cell and/or by additionally using one of the following AMFs: *Cartesian workspace monitoring*, *Cartesian protected space monitoring*, *Axis range monitoring* or *Tool orientation*.

When using the AMFs *Collision detection* and *TCP force monitoring*, the operator must ensure that workpieces that are picked up are not inadvertently permitted to become loose and fall off while one of the monitoring functions is active.

13.8.13.1 Axis torque monitoring

Axis torque monitoring can limit and monitor the torques of individual axes.

If the permissible axis torque is exceeded continuously due to jamming, it is possible to move the robot free by changing to CRR mode.

AMF	Description
Axis torque monitoring	Parameterizable AMF with 16 available instances The AMF is violated if the torque of the monitored axis exceeds or falls below the configured torque limit.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Currently without function ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function
Monitored axis	Axis to be monitored ■ **Axis1 … Axis16** **Note**: **Axis1 … Axis7** are used for an LBR iiwa.
Minimum torque [Nm]	Minimum permissible torque for the given axis ■ **-500 … 500 Nm**
Maximum torque [Nm]	Maximum permissible torque for the given axis ■ **-500 … 500 Nm**

13.8.13.2 Collision detection

Collision detection monitors the external axis torques against a definable limit value.

The external axis torque is defined as that part of the torque of an axis which is generated from the forces and torques occurring as the robot interacts with its environment. It is not measured directly but is rather calculated using the dynamic robot model. The accuracy of the calculated values depends on the dynamics of the robot motion and of the interaction forces of the robot with its environment.

The following points must be observed when using collision detection:

- Successful position and torque referencing are preconditions.
- The load data of the safety-oriented tool are taken into consideration (if configured).
- If the safety-oriented tool is configured, it must also be mounted on the robot flange.
- The load data of safety-oriented workpieces (if configured) are only taken into consideration if the currently active safety-oriented workpiece is communicated to the safety controller.

  (>>> 15.11.6 "Commanding load changes to the safety controller" Page 298)

In der AMF *Collision detection*, possible errors when activating the safety-oriented workpiece are not automatically taken into consideration.
When configuring the collision detection, it is therefore necessary to set the lowest possible values for the maximum permissible external torque so as to ensure that deviations in the load data will be interpreted as collisions and will cause the AMF to be violated.

If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be ensured that the carrier kinematic system does not move while the AMF is being used.
As long as the robot base of the monitored kinematic system is being accelerated, the safety integrity of the AMF is not assured.

If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be ensured, during use of the AMF, that the mounting direction of the monitored kinematic system does not differ from the configured mounting direction (e.g. due to tilting of the mobile platform). Otherwise, the safety integrity of the AMF is not assured.

AMF	Description
Collision detection	Parameterizable AMF with 8 available instances This AMF is violated if the external torque of at least one axis exceeds the configured limit value.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Currently without function ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function
Maximum external torque [Nm]	Maximum permissible external torque ■ **0 … 30 Nm**

13.8.13.3 TCP force monitoring

Description

In TCP force monitoring, the external force acting on the TCP of the safety-oriented tool is monitored against a definable limit value. If no tool is used, the external force acting on the center point of the robot flange is monitored.

The external force on the TCP is not measured directly but is rather calculated using the dynamic robot model. The accuracy of the calculated external force depends on the dynamics of the robot motion and of the actual force, among other things.

The following points must be observed when using TCP force monitoring:

- Successful position and torque referencing are preconditions.
- The load data of the safety-oriented tool are taken into consideration (if configured).
- If the safety-oriented tool is configured, it must also be mounted on the robot flange.
- The load data of the heaviest safety-oriented workpiece are taken into consideration (if configured).

In the AMF *TCP force monitoring*, possible errors when activating a safety-oriented workpiece are not automatically taken into consideration.
For this reason, when configuring the TCP force monitoring, it is necessary to set a value for the maximum permissible external force at the TCP which is greater than the weight of the heaviest workpiece to be picked up.

If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be ensured that the carrier kinematic system does not move while the AMF is being used.
As long as the robot base of the monitored kinematic system is being accelerated, the safety integrity of the AMF is not assured.

If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), it must be ensured, during use of the AMF, that the mounting direction of the monitored kinematic system does not differ from the configured mounting direction (e.g. due to tilting of the mobile platform). Otherwise, the safety integrity of the AMF is not assured.

AMF	Description
TCP force monitoring	Parameterizable AMF with 8 available instances This AMF is violated if the external force acting on the TCP of the safety-oriented tool (or center point of the robot flange) exceeds the configured limit value.

Parameter	Description
Monitored kinematic system	Kinematic system to be monitored ■ **First kinematic system**: Robot ■ **Second kinematic system**: Currently without function ■ **Third kinematic system**: Currently without function ■ **Fourth kinematic system**: Currently without function
Maximum TCP force [N]	Maximum permissible external force on the TCP ■ **50 … 1,000 N**

Accuracy of force detection

The accuracy of TCP force detection is also dependent on the robot pose. The safety controller recognizes non-permissible poses and sets the AMF *TCP force monitoring* to “violated” with a corresponding diagnostic message.

Non-permissible poses are those in which it is possible for TCP forces to have a short distance to all robot axes. This applies to singularity poses and poses near singularities. Axis angles close to 0° in A2, A4 and A6 must be avoided in particular:

- Axis A2: -35° to 35°
- Axis A4: -55° to 55°
- Axis A6: -20° to 20°

- External forces on the robot structure reduce the accuracy of TCP force detection. In many cases, the safety controller can automatically detect the external forces acting on the robot structure. The AMF *TCP force monitoring* is violated in this case.

It is not possible to guarantee that the safety controller will always automatically detect external forces acting on the robot structure. The user must ensure that the external forces act exclusively on the TCP in order to assure the safety integrity of the AMF *TCP force monitoring*.

13.9 Example of a safety configuration

The sole purpose of this example is to illustrate the safety configuration with KUKA Sunrise.Workbench.

The basis for a system’s safety configuration is always a risk analysis carried out by the user. The safety configuration displayed below serves only as an example and does not claim to be comprehensive.

13.9.1 Task

The LBR iiwa is used in an application in which it cooperates with a human. The tool installed on the robot is set as safety-oriented.

The operator places a workpiece in a workpiece pick-up position at regular intervals. One task of the robot is to check that the workpiece is present. It proceeds as follows: from a start position, it moves through the area accessible to humans (collaboration space) with a transfer motion. The purpose of this transfer motion is to achieve a pre-position of 20 cm above the waiting workpiece. It then moves toward the workpiece.

This lowering motion is parametrized with a force break condition (process threshold value: 20 N). After reaching the process threshold value, the robot uses its current position to determine whether the workpiece is present or not. The workpiece is not present if the robot can move all the way to the workpiece pick-up position. After it has finished checking, the robot moves back to the pre-position and out of the collaboration space.

13.9.2 Requirement

The following safety functions are required as part of the risk assessment for the above-described process:

1. It must be possible to stop the robot by pressing an external EMERGENCY STOP switch within reach of the operator.
2. The robot must not leave a defined workspace. The collaboration space is part of the workspace.
3. A transfer motion between the start position and pre-position can cause unintentional collisions with the operator. However, the space is designed in such a way that the human cannot be crushed. For this reason, the maximum permissible robot velocity for this space has been defined as 500 mm/s.
4. Collisions must be safely recognized during the transfer motion and cause the robot to come to a standstill if a torque of 15 Nm is exceeded on at least one axis.
5. Motions between the pre-position and the workpiece pick-up position can cause the hand and arm of the operator to be crushed. In order to ensure that the operator can respond appropriately to a robot motion and that the braking distances are sufficiently short, the robot velocity must not exceed 100 mm/s.
6. Furthermore, the robot must be brought to a standstill if crushing forces of more than 50 N arise during motions between the pre-position and the workpiece pick-up position. Force values of 20 N or more cause the lowering motion in the process to be aborted and are thus sufficiently below the latter limit.

13.9.3 Suggested solution for the task

In order for the requirements to be implemented, permanent and switchable safety monitoring functions must be configured:

- Permanent monitoring of the external EMERGENCY STOP device and workspace
- ESM state for the transfer motion between the start position and the pre-position
- ESM state for the motion between the pre-position and the workpiece pick-up position

To ensure a stable and smooth process sequence, the application must be designed in such a way that the defined limit values for the safety functions (velocity and workspace) are maintained.

The robot application implemented in the process described is not mentioned here.

Permanent safety monitoring

The EMERGENCY STOP function must be active throughout operation and the robot must not leave the workspace. Corresponding safety functions are configured in the Customer PSM table.

Configurable customer safety configuration (6/100)

Row	Active	Category	AMF 1	AMF 2	AMF 3	Reaction
1	☑	External EMERGENCYSTOP	Input signal (1) Input for safety signal : InputCIB_SR.1	-	-	Stop 1 (path-maintaining)
2	☑	Workspace monitoring	Cartesian workspace monitoring(1)	-	-	Stop 1 (path-maintaining)

Fig. 13-18: Permanent safety monitoring

Row	Description
1	External EMERGENCY STOP Implements requirement 1 An external EMERGENCY STOP is connected to a safe input. If the operator actuates the EMERGENCY STOP, a safety stop 1 (path-maintaining) is executed.
2	Cartesian workspace monitoring 2 Implements requirement 2 The workspace is represented by a safely monitored Cartesian workspace. If the robot leaves the configured space, a safety stop 1 (path-maintaining) is executed.

ESM state for transfer motion

An ESM state is defined for the transfer motion through the collaboration space between the start and pre-position. This is activated in the application before the transfer motion begins.

Velocity monitoring and collision detection must be active during the transfer motion in order to sufficiently reduce the danger of a collision between human and robot.

In order to avoid crushing at all times, an additional protected space is defined. This brings the robot to a standstill as soon as the distance between the robot or tool and the workpiece pick-up position becomes less than 15 cm.

State of the event-driven safety monitoring (ESM) (3/20)

Row	Active	AMF	Reaction
1	☑ 🔒	Cartesian velocity monitoring (1) Maximum velocity : 500 mm/s	Stop 1 (path-maintaining)
2	☑	Collision detection (1) Maximum external torque : 15 Nm	Stop 1 (path-maintaining)
3	☑	Cartesian protected space monitoring (1)	Stop 1 (path-maintaining)

Fig. 13-19: ESM state for transfer motion

Row	Description
1	Cartesian velocity monitoring Implements requirement 3 If a Cartesian velocity exceeds 500 mm/s, a safety stop 1 (path-maintaining) is executed.
2	Collision detection Implements requirement 4 If a collision causes an external torque of more than 15 Nm in at least one robot axis, a safety stop 1 (path-maintaining) is executed.
3	Protected space monitoring Implements the safety of the state regardless of the time and place of activation The safely monitored protected space encompasses the space above the workpiece pick-up position. As soon as the robot or the safely monitored tool enters this space, a safety stop 1 (path-maintaining) is executed.

ESM state for workpiece pick-up position

A specific ESM state is defined for the motions between the pre-position and the workpiece pick-up position. This is activated in the application before the lowering motion begins.

Velocity monitoring and force monitoring must be active during the motion in order to sufficiently reduce the danger of crushing the operator's hand or lower arm.

The state must ensure a sufficient degree of safety, regardless of the time or place of activation. The low permissible velocity and the active force monitoring mean that no further measures are necessary.

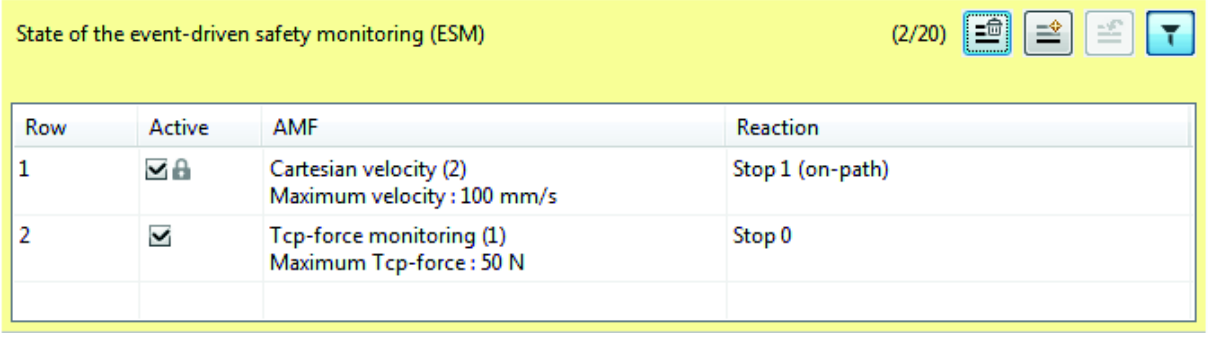


Fig. 13-20: ESM state for workpiece pick-up position

Row	Description
1	Cartesian velocity monitoring Implements requirement 5 If a Cartesian velocity exceeds 100 mm/s, a safety stop 1 (path-maintaining) is executed.
2	Force monitoring Implements requirement 6 If a contact situation causes a force of more than 50 N to be exerted at the TCP, a stop 0 is executed.

13.10 Position and torque referencing

13.10.1 Position referencing

Description

Position referencing checks whether the saved zero position of the motor of an axis (= saved mastering position) corresponds to the actual mechanical zero position.

Referencing is carried out continuously by the system when an axis moves at less than 30 °/s. Referencing is successful when the mastering sensor detects the mechanical zero position of the axis in a narrow range around the saved zero position of the motor. Referencing fails if the mastering sensor does not detect the mechanical zero position of the axis within the range of the saved zero position of the motor, or if it is detected at an unexpected position.

The safety integrity of the safety functions based upon this is limited until the position referencing test has been performed. This includes safely monitored Cartesian and axis-specific robot positions, Cartesian velocities, forces and collisions.

If position referencing fails on at least one axis, all AMFs based on safe axis positions are violated. (>>> "Position-based AMFs" Page 229)

Requirement

The position of an axis is not referenced after the following events:

- Robot controller is rebooted.
- The axis is remastered.
- Torque referencing of the axis fails.
- The maximum torque of the joint torque sensor of the axis has been exceeded.

> Following these events, the safety functions based on safe positions are not violated. The robot can be moved, but the safety integrity of the safety functions is not given.
> The position-based safety functions are only violated after these events if position referencing of one axis fails. Referencing must be successfully carried out before safety-critical applications can be executed.
> The position referencing status can be used as an AMF in the safety configuration. (>>> 13.8.6 "Evaluating the position referencing" Page 195)

Precondition

The position of an axis is referenced when it is moved over the saved zero position of the motor and when the zero position of the axis is then detected in a range of 0° +/- 0.5° by the mastering sensor. Preconditions for this:

- The velocity at which the axis is moved over the zero position must be < 30 °/s.
- An axis-specific range before and after the zero position must be passed through as a minimum. The motion direction is not relevant.

  The axis-specific range of motion is robot-specific:

Robot variant	A1	A2	A3	A4	A5	A6	A7
LBR iiwa 7 R800	±10.5°	±10.5°	±10.5°	±10.5°	±10.5°	±14°	±14°
LBR iiwa 14 R820	±9.5°	±9.5°	±10.5°	±10.5°	±10.5°	±14°	±14°

Performance

Position referencing of all axes is continuously performed by the system when the above conditions are met. Position referencing can be carried out in a targeted manner the following ways:

- Automatically while the program is running, when an axis moves over the zero position at less than 30 °/s.
- Jogging each axis individually over the zero position.

- Executing the application prepared by KUKA for referencing. The axes are moved over the zero position from the vertical stretch position.

  A prepared application for position and torque referencing of the LBR iiwa is available from Sunrise.Workbench. Position and torque referencing can be carried out simultaneously with this application.

  (>>> 13.10.3 "Creating an application for position and torque referencing" Page 225)

> If it is not possible to reference from the vertical stretch position, a customized application for position referencing must be created and executed.

13.10.2 Torque referencing

Description

The LBR iiwa has a joint torque sensor in each axis which reliably determines the torque currently acting on the axis. These data are used for calculating and monitoring externally acting torques or Cartesian forces, for example.

The referencing test of the joint torque sensors checks whether the expected external torque, which can be calculated for an axis based on the robot model and the given load data, corresponds to the value determined on the basis of the measured value of the joint torque sensor. If the difference between these values exceeds a certain tolerance value, the referencing of the torque sensors has failed.

The safety integrity of the safety functions based upon this is limited until the torque referencing test has been performed successfully. This includes axis torque and TCP force monitoring as well as collision detection.

If torque referencing fails on at least one axis, all AMFs based on safe torque values are violated. (>>> "Axis torque-based AMFs" Page 229)

Requirement

The joint torque sensor of an axis is not referenced after the following events:

- Robot controller is rebooted.
- Position referencing of the axis fails.
- The maximum torque of the joint torque sensor of the axis has been exceeded.

> Following these events, the safety functions based on safe torques are not violated. The robot can be moved, but the safety integrity of the safety functions is not given.
> The torque-based safety functions are only violated after these events if torque referencing of one axis fails. Referencing must be successfully carried out before safety-critical applications can be executed.
> The torque referencing status can be used as an AMF in the safety configuration. (>>> 13.8.7 "Evaluating the torque referencing" Page 196)

Execution

A prepared application for position and torque referencing of the LBR iiwa is available from Sunrise.Workbench. Position and torque referencing can be carried out simultaneously with this application.

(>>> 13.10.3 "Creating an application for position and torque referencing" Page 225)

A total of 10 measured joint torque values must be given for each axis. For this purpose, 5 measurement poses are defined in the application, each of which can be addressed with positive and negative directions of axis rotation. If the poses cannot be addressed, they must be adapted in the application.

During torque referencing, each of the measurement poses must be addressed in sequence with positive and negative directions of axis rotation before the next measurement pose is addressed. The safety integrity of the referencing of the joint torque sensors is otherwise not given.

Before torque referencing is executed, the user must ensure that the load data of the tool mounted on the robot correspond to the load data specified for the safety-oriented tool. Additionally, the user must ensure that the load data of the workpiece picked up (if present) correspond to the load data of the activated safety-oriented workpiece. The safety integrity of the referencing of the joint torque sensors is otherwise not given.
In particular, no supplementary loads may be mounted on the robot, e.g. loads attached to the robot structure or workpieces which are picked up but not taken into consideration by the safety controller.

During torque referencing, no external forces may act on the robot, the tool mounted on the flange and the workpiece being picked up (if present). The user must ensure this when referencing is executed. The safety integrity of the referencing of the joint torque sensors is otherwise not given.

The safety controller evaluates the external torque for all 10 measured values and determines the mean value of the external torque for each axis. Referencing is successful if this mean value is below a defined tolerance. Otherwise, referencing has failed.

If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit), the user must ensure that the carrier kinematic system is not moved during torque referencing and that the mounting direction of the kinematic system to be referenced does not differ from the configured mounting direction (e.g. due to tilting of the mobile platform). The safety integrity of the referencing of the joint torque sensors is otherwise not given.

13.10.3 Creating an application for position and torque referencing

Description

The following points must be observed if the application for torque referencing needs to be edited due to measurement poses which cannot be addressed:

- The joint torque values must be measured while the robot is stationary.

  A wait time of at least 2.5 seconds in which the robot does not move is required between the moment the measurement pose is reached and the measurement itself. Wait times which are too short can reduce the referencing accuracy due to oscillations of the robot structure.
- The measurement is started with the method sendSafetyCommand().
- There may be a maximum of 15 s between 2 consecutive measurements.

Procedure

1. Select the Sunrise project in the **Package Explorer**.
2. Select the menu sequence **File** > **New** > **Other...**.
3. In the **Sunrise** folder, select the **Application for position and GMS referencing of the LBR iiwa** option and click on **Finish**.

   The **PositionAndGMSReferencing.java** application is created in the source folder of the project and opened in the editor area of Sunrise.Workbench.
4. If measurement poses cannot be addressed due to the system configuration, adapt them in the application.

5. Synchronize the project in order to transfer the application to the robot controller.

13.11 Safety acceptance overview

The system must not be put into operation until the safety acceptance procedure has been completed successfully. For successful safety acceptance, the points in the checklists must be completed fully and confirmed in writing by the safety maintenance technician.

> The completed checklists, confirmed in writing, must be kept as documentary evidence.

Safety acceptance must be carried out in the following cases:

- Following initial start-up or recommissioning of the industrial robot
- After a change to the industrial robot
- After a change to the safety configuration
- After a software update, e.g. of the system software

  Safety acceptance after a software update is only necessary if the ID of the safety configuration (= checksum) has changed as a result of the update.

The system integrator determines the required safety functions using the risk analysis as a basis. Once the safety configuration is activated on the robot controller, the safety functions must be tested for correct functioning.

> If a test requires persons to be present in the danger zone, the test must be conducted in T1 mode.

The following checklists must be used to verify whether the configured safety parameters have been correctly transferred.

The checklists must be processed in the following order:

1. Checklist for basic test of the safety configuration

   (>>> 13.11.1 "Checklist for general safety functions" Page 226)
2. Checklists for checking the safety-oriented tool

   (>>> 13.11.2 "Checklists for the safety-oriented tool" Page 229)
3. Checklists for checking the safety-oriented workpieces

   (>>> 13.11.3 "Checklist for safety-oriented workpieces" Page 231)
4. Checklist for checking the rows used in the PSM table *KUKA PSM* and in the PSM table *Customer PSM*

   (>>> 13.11.4 "Checklist for rows used in the PSM tables" Page 232)
5. Checklists for checking the ESM states which have been used and not used

   (>>> 13.11.5 "Checklists for ESM states" Page 232)
6. Checklists for checking the AMFs used

   (>>> 13.11.6 "Checklists for AMFs used" Page 234)

It is possible to create a report of the current safety configuration.

(>>> 13.11.7 "Creating a safety configuration report" Page 241)

13.11.1 Checklist for general safety functions

Checklist

- Serial number of the robot: _____
- ID of the safety configuration: _____

- Name of safety maintenance technician: _____

No.	Activity	Yes	Not relevant
1	Operator safety: is all operator safety equipment configured, properly connected and tested for correct function?		
2	Operator safety: a stop is triggered if AUT or T2 mode is active with the operator safety open.		
3	Operator safety: a manual reset function is present and activated.		
4	Hand guiding device enabling state: is the enabling device of the hand guiding device configured, properly connected and tested for correct function?		
5	Local EMERGENCY STOP: are all local EMERGENCY STOP devices configured, properly connected and tested for correct function?		
6	External EMERGENCY STOP: are all external EMERGENCY STOP devices configured, properly connected and tested for correct function?		
7	Local and external EMERGENCY STOP: are the local and external EMERGENCY STOPs each configured as an individual AMF in a row of the PSM table?		
8	Safety stop: is all safety stop equipment configured, properly connected and tested for correct function?		
9	Safe operational stop: is all equipment for the safe operational stop configured, properly connected and tested for correct function?		
10	When using position-based AMFs: is the limited safety integrity of the position-based AMFs taken into consideration in the absence of position referencing? (>>> "Position-based AMFs" Page 229) **Note**: Initiation of the safe state in the absence of position referencing can be configured by using the AMF *Position referencing*.		
11	When using position-based AMFs: has position referencing been carried out successfully?		
12	Velocity monitoring: have all necessary velocity monitoring tests been configured and tested?		
13	Manual guidance: has it been configured in such a way that appropriate velocity monitoring is active in every operating mode for manual guidance?		
14	Workspace monitoring: have all necessary workspace monitoring tests been configured and tested?		
15	Cartesian workspace monitoring functions: has it been taken into consideration that the system does not monitor the entire structure of the robot, tool and workpiece against the space violation, but only the monitoring spheres on the robot and tool?		
16	Collision detection: have all necessary HRC functionalities been configured?		
17	Collision detection: has it been configured in such a way that velocity monitoring is also active when collision detection is active?		

No.	Activity	Yes	Not relevant
18	Collision detection: has it been configured in such a way that velocity monitoring is also active when TCP force monitoring is active?		
19	Collision detection: is a safety stop 0 configured for all safety monitoring functions in order to detect crushing situations?		
20	When using axis torque-based AMFs: is the limited safety integrity of the axis torque-based AMFs taken into consideration in the absence of position referencing and/or torque referencing? (>>> "Axis torque-based AMFs" Page 229) **Note**: Initiation of the safe state in the absence of position and/or torque referencing can be configured by using the AMF *Position referencing* and the AMF *Torque referencing*.		
21	In the configuration of all rows in the PSM table and ESM states, has it been taken into account that the safe state of the AMFs is the “violated” state (state “0”)? **Note**: In the event of an error, an AMF goes into the safe state.		
22	PSM configuration: in the configuration of output signals, has it been taken into account for the safety reaction that an output is LOW (state “0”) in the safe state?		
23	ESM configuration: are all ESM states consistent, i.e. does each individual ESM state sufficiently reduce all dangers?		
24	Have torque and position referencing been carried out successfully?		
25	If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit): Has the fact been taken into consideration that, with the AMF *Cartesian workspace monitoring* / *Cartesian protected space monitoring*, the monitoring space is defined relative to the base of the monitored kinematic system, i.e. it moves with the carrier kinematic system?		
26	If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit): Has the fact been taken into consideration that, with the AMF *Cartesian velocity monitoring*, it is not the absolute velocity, but the velocity of the monitored kinematic system relative to the carrier kinematic system that is monitored?		
27	If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit): Has the fact been taken into consideration that, with the AMF *Tool-related velocity component*, it is not the absolute velocity, but the velocity of the monitored kinematic system relative to the carrier kinematic system that is monitored?		

No.	Activity	Yes	Not relevant
28	If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform): Has the fact been taken into consideration that, with the AMF *Tool orientation*, the reference orientation is defined relative to the carrier kinematic system, i.e. it moves with the carrier kinematic system?		
29	If the monitored kinematic system is fastened to a carrier kinematic system (e.g. mobile platform, linear unit): Has the fact been taken into consideration that the safety integrity of the AMFs *Collision detection* and *TCP force monitoring* is only assured as long as the carrier kinematic system is at a standstill?		

Place, date	
Signature	

By signing, the signatory confirms the correct and complete performance of the safety acceptance test.

Position-based AMFs

The safety integrity of position-based AMFs is only given without limitations when position referencing has been carried out successfully. (Referencing is not required for a mobile platform.)

AMF	Position referencing	Torque referencing
Standstill monitoring of all axes	✔	✘
Axis range monitoring	✔	✘
Cartesian velocity monitoring	✔	✘
Tool-related velocity component	✔	✘
Cartesian workspace monitoring	✔	✘
Cartesian protected space monitoring	✔	✘
Tool orientation	✔	✘

Axis torque-based AMFs

The safety integrity of axis torque-based AMFs is only given without limitations when position and/or torque referencing has been carried out successfully. (Referencing is not required for a mobile platform.)

AMF	Position referencing	Torque referencing
Axis torque monitoring	✘	✔
Collision detection	✔	✔
TCP force monitoring	✔	✔

13.11.2 Checklists for the safety-oriented tool

13.11.2.1 Geometry data of the safety-oriented tool

Description

If one of the following AMFs is used in the safety configuration, it is necessary to check that the geometric tool data have been entered correctly:

- *Cartesian velocity monitoring*
- *Cartesian workspace monitoring*

  Only for spaces in which the monitoring spheres on the tool are configured as a structure to be monitored.
- *Cartesian protected space monitoring*

  Only for spaces in which the monitoring spheres on the tool are configured as a structure to be monitored.

The geometric tool data can be tested by intentionally violating one of the configured monitoring spaces with each tool sphere and checking the reaction.

If no space monitoring functions are used, only the position of the sphere center points is relevant. The configured Cartesian velocity limit can be tested by intentionally exceeding this velocity for each tool sphere and checking the reaction.

Precondition

- Position referencing has been carried out successfully.

Checklist

No.	Activity	Yes	Not relevant
1	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		
2	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		
3	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		
4	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		
5	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		
6	Tool sphere (frame name) _____ Have the radius and position of the tool sphere been correctly entered and checked?		

13.11.2.2 Load data of the safety-oriented tool

Description

If one of the following AMFs is used in the safety configuration, it is necessary to check that the load data of the safety-oriented tool have been entered correctly.

- *Collision detection*
- *TCP force monitoring*

It is advisable to check the load data by performing torque referencing in several suitable poses. Suitable poses include those with similar axis angles in the horizontal extended position which have the following characteristics:

- Axes A2, A4 and A6 are loaded.
- The poses differ in their axis value of A7 by 90°.

If the load data are correct, torque referencing must be successful.

Precondition

- Position and torque referencing have been carried out successfully.
- If a safety-oriented workpiece is picked up by the tool to check the load data: The correct safety-oriented workpiece is active.

Checklist

No.	Activity	Yes	Not relevant
1	Have the load data of the tool been correctly entered and checked?		

13.11.3 Checklist for safety-oriented workpieces

Description

If one of the following AMFs is used in the safety configuration, it is necessary to check that the load data of the safety-oriented workpieces have been entered correctly.

- *Collision detection*
- *TCP force monitoring*

It is advisable to check the load data by performing torque referencing in several suitable poses. Suitable poses include those with similar axis angles in the horizontal extended position which have the following characteristics:

- Axes A2, A4 and A6 are loaded.
- The poses differ in their axis value of A7 by 90°.

If the load data are correct, torque referencing must be successful.

Precondition

- Position and torque referencing have been carried out successfully.
- The correct safety-oriented workpiece is active.

Checklist

No.	Activity	Yes	Not relevant
1	Name of workpiece: _____ Have the load data of the workpiece been correctly entered and checked?		
2	Name of workpiece: _____ Have the load data of the workpiece been correctly entered and checked?		
3	Name of workpiece: _____ Have the load data of the workpiece been correctly entered and checked?		
4	Name of workpiece: _____ Have the load data of the workpiece been correctly entered and checked?		
5	Name of workpiece: _____ Have the load data of the workpiece been correctly entered and checked?		
6	Name of workpiece: _____ Have the load data of the workpiece been correctly entered and checked?		

No.	Activity	Yes	Not relevant
7	Name of workpiece: _____ Have the load data of the workpiece been correctly entered and checked?		
8	Name of workpiece: _____ Have the load data of the workpiece been correctly entered and checked?		

13.11.4 Checklist for rows used in the PSM tables

Description

Each row in the PSM table *KUKA PSM* and in the PSM table *Customer PSM* must be tested to verify that the expected reaction is triggered. If the reaction is to switch off an output, the test must also ensure that the output is correctly connected.

A row in the PSM table can be tested by violating 2 of its AMFs at a time. It is then possible to test the remaining AMF separately in a targeted manner. If fewer than 3 AMFs are used in a row, the unassigned columns are regarded as violated AMFs.

(>>> 13.11.6 "Checklists for AMFs used" Page 234)

> For each row in the PSM table, the points in the checklist must be executed and separately documented.

Checklist

- Row no.: _____

No.	Activity	Yes	Not relevant
1	AMF 1 was tested successfully. Precondition: AMF 2 and AMF 3 are violated. AMF 1: _____		
2	AMF 2 was tested successfully. Precondition: AMF 1 and AMF 3 are violated. AMF 2: _____		
3	AMF 3 was tested successfully. Precondition: AMF 1 and AMF 2 are violated. AMF 3: _____		

13.11.5 Checklists for ESM states

13.11.5.1 Used ESM states

Description

Each row in the ESM state must be tested to verify that the expected reaction is triggered when the configured AMF is violated.

(>>> 13.11.6 "Checklists for AMFs used" Page 234)

> For each ESM state, the points in the checklist must be executed and separately documented.

Checklist

- ESM state: _____

No.	Activity	Yes	Not relevant
1	AMF row 1 was tested successfully. AMF row 1: _____		
2	AMF row 2 was tested successfully. AMF row 2: _____		
3	AMF row 3 was tested successfully. AMF row 3: _____		
4	AMF row 4 was tested successfully. AMF row 4: _____		
5	AMF row 5 was tested successfully. AMF row 5: _____		
6	AMF row 6 was tested successfully. AMF row 6: _____		
7	AMF row 7 was tested successfully. AMF row 7: _____		
8	AMF row 8 was tested successfully. AMF row 8: _____		
9	AMF row 9 was tested successfully. AMF row 9: _____		
10	AMF row 10 was tested successfully. AMF row 10: _____		
11	AMF row 11 was tested successfully. AMF row 11: _____		
12	AMF row 12 was tested successfully. AMF row 12: _____		
13	AMF row 13 was tested successfully. AMF row 13: _____		
14	AMF row 14 was tested successfully. AMF row 14: _____		
15	AMF row 15 was tested successfully. AMF row 15: _____		
16	AMF row 16 was tested successfully. AMF row 16: _____		
17	AMF row 17 was tested successfully. AMF row 17: _____		
18	AMF row 18 was tested successfully. AMF row 18: _____		
19	AMF row 19 was tested successfully. AMF row 19: _____		
20	AMF row 20 was tested successfully. AMF row 20: _____		

13.11.5.2 Non-used ESM states

Description All ESM states which are not used must be tested as to whether a safety stop is triggered when the ESM state is selected.

Checklist

No.	Activity	Yes	Not relevant
1	Selection of non-used ESM state 1 was tested successfully.		
2	Selection of non-used ESM state 2 was tested successfully.		
3	Selection of non-used ESM state 3 was tested successfully.		
4	Selection of non-used ESM state 4 was tested successfully.		
5	Selection of non-used ESM state 5 was tested successfully.		
6	Selection of non-used ESM state 6 was tested successfully.		
7	Selection of non-used ESM state 7 was tested successfully.		
8	Selection of non-used ESM state 8 was tested successfully.		
9	Selection of non-used ESM state 9 was tested successfully.		
10	Selection of non-used ESM state 10 was tested successfully.		

13.11.6 Checklists for AMFs used

An AMF which is used in more than one row in the PSM table must be separately tested in each row.

13.11.6.1 AMF *smartPAD Emergency Stop*

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered by pressing the E-STOP on the smartPAD.		

13.11.6.2 AMF *smartPAD enabling switch inactive*

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered by releasing an enabling switch on the smartPAD.		

13.11.6.3 AMF *smartPAD enabling switch panic active*

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered by pressing an enabling switch down fully on the smartPAD.		

13.11.6.4 AMF *Hand guiding device enabling inactive*

Checklist

- Used input enabling switch 1: _____
- Used input enabling switch 2: _____

- Used input enabling switch 3: _____
- Used input panic switch 1: _____
- Used input panic switch 2: _____
- Used input panic switch 2: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered by releasing an enabling switch on the hand guiding device.		
2	The configured reaction is triggered by pressing fully down on an enabling switch on the hand guiding device.		

13.11.6.5 AMF *Hand guiding device enabling active*

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered by pressing an enabling switch on the hand guiding device.		

13.11.6.6 AMF *Test mode*

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered in T1.		
2	The configured reaction is triggered in T2.		
3	The configured reaction is triggered in CRR.		

13.11.6.7 AMF *Automatic mode*

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered in AUT.		

13.11.6.8 AMF *Reduced-velocity mode*

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered in T1.		
2	The configured reaction is triggered in CRR.		

13.11.6.9 AMF *High-velocity mode*

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered in T2.		
2	The configured reaction is triggered in AUT.		

13.11.6.10AMF *Motion enable*

Checklist

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the E-STOP is pressed on the smartPAD.		

13.11.6.11AMF *Input signal*

Checklist

- Input used: _____
- Instance of the input used: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the input is LOW (state “0”).		

13.11.6.12AMF *Standstill monitoring of all axes*

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if one axis of the monitored kinematic system is moved.		

13.11.6.13AMF *Axis torque monitoring*

Description

The AMF can be tested by displaying the current measured axis torques on the smartPAD and then subjecting the monitored axis to gravitational force or manual loading.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Monitored axis: _____
- Maximum permissible axis torque: _____
- Minimum permissible axis torque: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the axis torque exceeds the maximum permissible value.		
2	The configured reaction is triggered if the axis torque falls below the minimum permissible value.		

13.11.6.14AMF *Axis velocity monitoring*

Description

The AMF can be tested by moving the monitored axis at a velocity of approx. 10% over the configured velocity limit.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Monitored axis: _____

- Maximum permissible axis velocity: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the maximum permissible axis velocity is exceeded.		

13.11.6.15 AMF *Position referencing*

> This AMF is violated after the robot controller is rebooted.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if one or more axes of the monitored kinematic system is not referenced.		

13.11.6.16 AMF *Torque referencing*

> This AMF is violated after the robot controller is rebooted.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if one or more axes of the monitored kinematic system is not referenced.		

13.11.6.17 AMF *Axis range monitoring*

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Monitored axis: _____
- Lower limit of the permissible axis range: _____
- Upper limit of the permissible axis range: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the lower limit of the permissible axis range is exceeded.		
2	The configured reaction is triggered if the upper limit of the permissible axis range is exceeded.		

13.11.6.18 AMF *Cartesian velocity monitoring*

Description

The AMF can be tested by moving a monitored point of the monitored kinematic system at a Cartesian velocity of approx. 10% over the configured velocity limit.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Maximum permissible Cartesian velocity: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the maximum permissible Cartesian velocity of a monitored point is exceeded.		

13.11.6.19AMF *Cartesian workspace monitoring / Cartesian protected space monitoring*

Description

The first step is to test whether the orientation of the monitoring space is correctly configured. This involves violating 2 adjoining space surfaces at a minimum of 3 different points in each case.

The second step is to test whether the size of the monitoring space is correctly configured. This involves violating the other space surfaces at a minimum of 1 point in each case. In total, at least 10 points must be addressed.

The third step is to test whether the structure to be monitored is correctly configured. This involves violating the space monitoring, both with the monitoring spheres on the robot and on the tool (if both structures are to be monitored), or just with the monitoring spheres on the robot or on the tool.

Checklist

- Type of monitoring space: _____
- Instance of the monitoring space: _____
- Monitored kinematic system: _____
- Monitored structure: _____
- Offset of the origin of the monitoring space:
  - X: _____ mm
  - Y: _____ mm
  - Z: _____ mm
- Orientation of the origin of the monitoring space:
  - A: _____ °
  - B: _____ °
  - C: _____ °
- Length of the monitoring space: _____ mm
- Width of the monitoring space: _____ mm

No.	Activity	Yes	Not relevant
1	The correct configuration of the monitoring space has been tested as above and the configured reaction is triggered if the monitoring space is violated.		
2	The configured reaction is triggered if the space monitoring is violated on the monitoring spheres on the robot.		
3	The configured reaction is triggered if the space monitoring is violated on the monitoring spheres on the tool.		

13.11.6.20AMF *Collision detection*

Description

The AMF can be tested by displaying the current measured external axis torques on the smartPAD and then loading the individual axes.

Precondition

- Torque referencing has been carried out successfully.

Checklist

- Monitoring instance: _____
- Monitored kinematic system:
- Maximum permissible external axis torque: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the external torque of one or more axes of the monitored kinematic system exceeds the maximum permissible external torque.		

13.11.6.21 AMF *TCP force monitoring*

Description

In order to test the AMF, suitable measuring equipment is required, e.g. a spring balance.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Maximum permissible external TCP force: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the external force acting on the TCP exceeds the maximum permissible force.		

13.11.6.22 AMF *Time delay*

Checklist

- Instance of delay: _____
- Delay time: _____

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered after the configured time.		

13.11.6.23 AMF *Tool orientation*

Description

In order to test the AMF, the permissible orientation cone must be violated at 3 straight lines offset by approx. 120° to one another. This ensures that the permissible orientation angle, the orientation of the reference vector and the tool orientation are correctly configured.

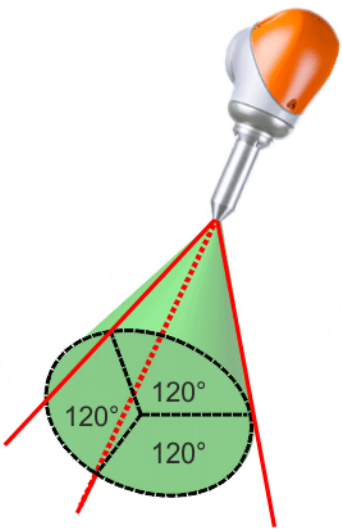


Fig. 13-21: Position of the straight lines on the monitoring cone

The orientation angles of the Z axis of the tool orientation frame are defined using 3 straight lines situated on the edge of the monitoring cone and offset at 120° to one another. These angles must be set in order to test the AMF *Tool orientation*. The AMF must be violated when all 3 tool orientation angles are exceeded.

Procedure

The procedure describes an example of how the correct configuration of the monitoring cone can be tested.

1. Orient the Z axis of the tool orientation frame according to the reference vector relative to the world coordinate system.
2. Exceed the permissible deviation angle by tilting the tool orientation frame in B or C.

   The configured reaction must be triggered.
3. Orient the Z axis of the tool orientation frame according to the reference vector relative to the world coordinate system.

   If a stop reaction has been configured, the robot must be switched to CRR mode in order for it to be moved.
4. Rotate the tool orientation frame by 120° in A.
5. Exceed the permissible deviation angle by tilting the tool orientation frame in B or C.

   The configured reaction must be triggered.
6. Orient the Z axis of the tool orientation frame according to the reference vector relative to the world coordinate system.

   If a stop reaction has been configured, the robot must be switched to CRR mode in order for it to be moved.
7. Rotate the tool orientation frame by 120° in A.
8. Exceed the permissible deviation angle by tilting the tool orientation frame in B or C.

   The configured reaction must be triggered.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Orientation of the reference vector relative to the world coordinate system:
  - A: _____ °
  - B: _____ °
  - C: _____ °
- Permissible workspace (deviation angle): _____ °

No.	Activity	Yes	Not relevant
1	The correct configuration of the monitoring cone has been checked and the configured reaction is triggered when the permissible angle for all 3 straight lines has been exceeded.		

13.11.6.24AMF *Tool-related velocity component*

Description

For the test, a motion with the configured point for the tool-specific velocity component must be programmed. It must be ensured that the test motions include a reorientation of the tool in order to check the correct configuration of the monitored point.

The test must be performed twice:

- Once at a velocity slightly above the maximum permissible velocity.
- Once at a velocity slightly below the maximum permissible velocity.

This is to ensure that the velocity limit is only violated by the configured monitored point.

Checklist

- Monitoring instance: _____
- Monitored kinematic system: _____
- Monitored component of the velocity vector: _____
- Maximum permissible Cartesian velocity of the monitored component: _____
- Orientation of the monitored component of the velocity vector relative to the flange coordinate system:
  - A: _____ rad
  - B: _____ rad
  - C: _____ rad
- Monitored point relative to the flange coordinate system:
  - X: _____ mm
  - Y: _____ mm
  - Z: _____ mm

No.	Activity	Yes	Not relevant
1	The configured reaction is triggered if the motion is executed with a velocity that exceeds the maximum permissible velocity.		
2	The configured reaction is not triggered if the motion is executed with a velocity that is below the maximum permissible velocity.		

13.11.7 Creating a safety configuration report

Description

A report of the current safety configuration can be created and displayed in the Editor. The report can be edited and printed for documentation purposes.

The safety configuration report provides the following checklists matching the safety configuration:

- Checklist for checking the rows used in the Customer PSM table
- Checklists for checking the ESM states which have been used and not used
- Checklists for checking the AMFs used

The checklists provided by the safety configuration report are not sufficient for a complete safety acceptance procedure. The following additional checklists must be used for complete safety acceptance:

- Checklist for basic test of the safety configuration
- Checklists for checking the safety-oriented tool
- Checklist for checking the safety-oriented workpieces

The safety configuration report contains the following information for the unambiguous assignment of the safety configuration:

- Name of the Sunrise project to which the safety configuration belongs
- Safety version used
- Safety ID (checksum of the safety configuration)

  The safety ID must match the ID of the safety configuration which is activated on the robot controller and is to be tested.
- Date and time of the last modification to the safety configuration

Procedure

- Right-click on the desired project in the **Package Explorer** and select **Sunrise** > **Create safety configuration report** from the context menu.

  The report of the current safety configuration is created and opened in the editor area.

14 Basic principles of motion programming

This chapter describes the theoretical principles of motion programming.

The programming of motions in KUKA Sunrise.Workbench is described in the following chapter: (>>> 15 "Programming" Page 265)

14.1 Overview of motion types

> The start point of a motion is always the end point of the previous motion.

The following motion types can be programmed as an individual motion:

- Point-to-point motions (PTP)
  (>>> 14.2 "PTP motion type" Page 243)
- Linear motions (LIN)
  (>>> 14.3 "LIN motion type " Page 244)
- Circular motions (CIRC)
  (>>> 14.4 "CIRC motion type" Page 244)
- Manual guidance motion with hand guiding device
  (>>> 14.7 "Manual guidance motion type" Page 251)

The following types of motion can be programmed as segments of a CP spline block:

- Linear motions (LIN)
- Circular motions (CIRC)
- Polynomial motions (SPL)

The following types of motion can be programmed as segments of a JP spline block:

- Point-to-point motions (PTP)

(>>> 14.6 "Spline motion type" Page 245)

The following motions are known as CP (“Continuous Path”) motions:

- LIN, CIRC, SPL, CP spline blocks

The following motions are known as JP (“Joint Path”) motions:

- PTP, JP spline blocks

14.2 PTP motion type

The robot guides the TCP along the fastest path to the end point. The fastest path is generally not the shortest path in space and is thus not a straight line. As the motions of the robot axes are simultaneous and rotational, curved paths can be executed faster than straight paths.

PTP is a fast positioning motion. The exact path of the motion is not predictable, but is always the same, as long as the general conditons are not changed.

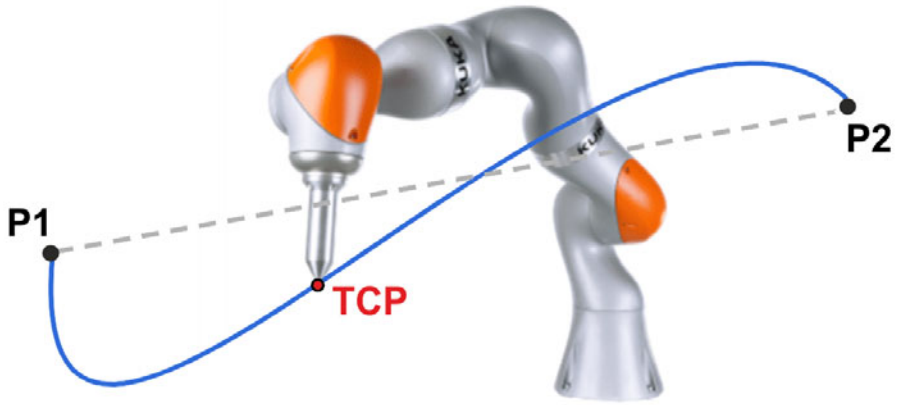


Fig. 14-1: PTP motion

14.3 LIN motion type

The robot guides the TCP at the defined velocity along a straight path in space to the end point.

In a LIN motion, the robot configuration of the end pose is not taken into account.

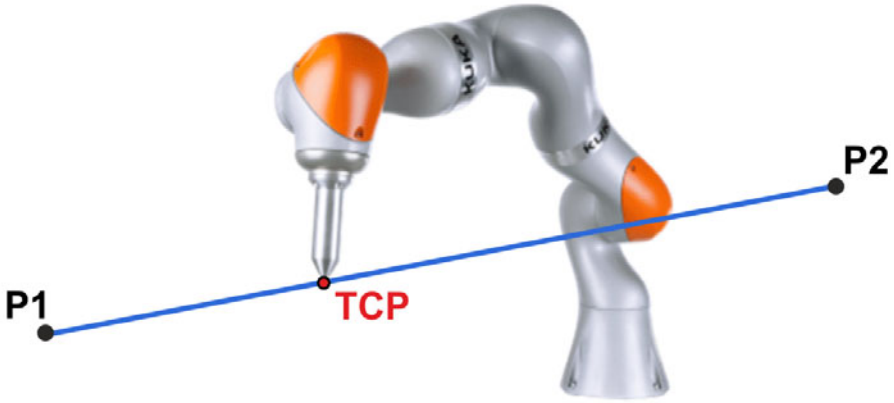


Fig. 14-2: LIN motion

14.4 CIRC motion type

The robot guides the TCP at the defined velocity along a circular path to the end point. The circular path is defined by a start point, auxiliary point and end point.

In a CIRC motion, the robot configuration of the end pose is not taken into account.

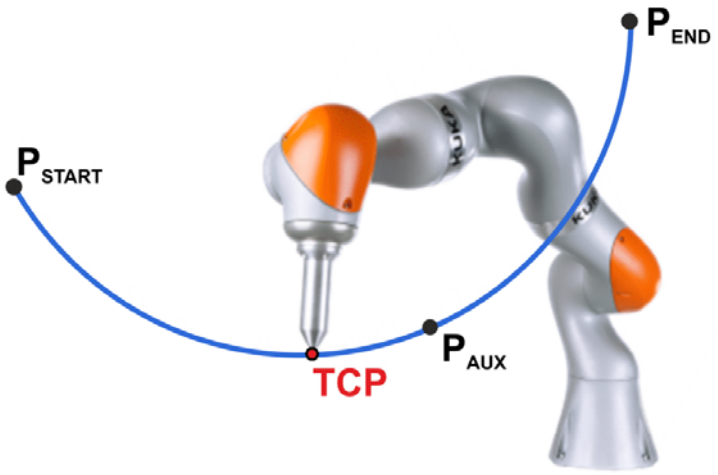


Fig. 14-3: CIRC motion

14.5 SPL motion type

The motion type SPL enables the generation of curved paths. SPL motions are always grouped together in spline blocks. The resulting paths run smoothly through the end points of the SPL motion.

In an SPL motion, the robot configuration of the end pose is not taken into account.

> Curved lines are achieved by grouping together 2 or more SPL segments. If a single SPL segment is executed, the result is the same as for a LIN command.

14.6 Spline motion type

Spline is a motion type that is particularly suitable for complex, curved paths. With a spline motion, the robot can execute these complex paths in a continuous motion. Such paths can also be generated using approximated LIN and CIRC motions, but splines have advantages, however.

Splines are programmed in spline blocks. A spline block is used to group together several individual motions as an overall motion. The spline block is planned and executed by the robot controller as a single motion block.

The motions contained in a spline block are called spline segments.

- A CP spline block can contain SPL, LIN and CIRC segments.
- A JP spline block can contain PTP segments.

In a Cartesian spline motion, the robot configuration of the end pose is not taken into account.

The configuration of the end pose of a spline segment depends on the robot configuration at the start of the spline segment.

Path of a spline block

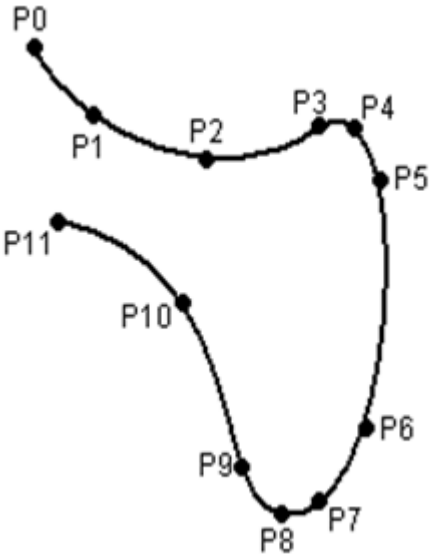


Fig. 14-4: Curved path with spline block

- The path is defined by means of points that are located on the path. These points are the end points of the individual spline segments.
  - All points are passed through without exact positioning.

    Exception: The velocity is reduced to 0.

    (>>> 14.6.1 "Velocity profile for spline motions" Page 246)
  - If all points are situated in a plane, then the path is also situated in this plane.
  - If all points are situated on a straight line, then the path is also a straight line.
- There are a few cases in which the velocity is reduced.

  (>>> 14.6.1 "Velocity profile for spline motions" Page 246)
- The path always remains the same, irrespective of the override setting, velocity or acceleration.
- Circles and tight radii are executed with great precision.

14.6.1 Velocity profile for spline motions

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.

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

Only dynamic effects, such as those caused by high tool loads or the installation angle of the robot, may result in slight path deviations.

Reduction of the velocity

With spline motions, the velocity falls below the programmed velocity in the following cases:

- Tight corners, e.g. due to abrupt change in direction
- Major reorientation
- Motion in the vicinity of singularities

Reduction of the velocity due to major reorientation can be avoided with spline segments by programming the orientation control `SplineOrientationType.Ignore`.

(>>> 14.9 "Orientation control with LIN, CIRC, SPL" Page 254)

Reduction of the velocity to 0

With spline motions, exact positioning is carried out in the following cases:

- Successive spline segments with the same end points
- Successive LIN and/or CIRC segments. Cause: inconstant velocity direction.

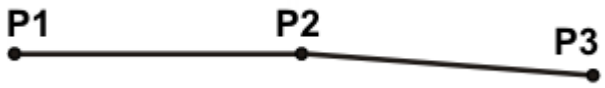


Fig. 14-5: Exact positioning at P2

In the case of LIN-CIRC transitions, the velocity also drops to 0 if the straight line is a tangent of the circle. This is caused by the fact that at the transition point between the straight line (curvature equals 0) and the circle (curvature is not equal to 0) the curvature characteristic is not constant.

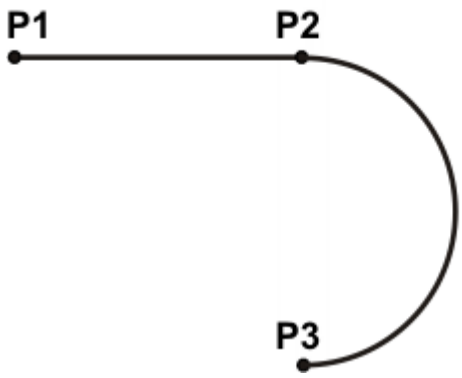


Fig. 14-6: Exact positioning at P2

Exceptions:

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

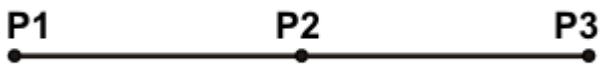


Fig. 14-7: P2 is executed without exact positioning.

- In the case of a CIRC-CIRC transition, the velocity is not reduced if both circles have the same center point and the same radius and if the orientations change uniformly. Since the required accuracy is difficult to achieve by teaching the end point and auxiliary point, calculation of the points on the circle is recommended.

14.6.2 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 LIN segment or vice versa, the path changes in the previous segment and the next segment.

Example 1

Original path:

```
Spline mySpline = new Spline(
   spl(getApplicationData().getFrame("/P1")),
   spl(getApplicationData().getFrame("/P2")),
   spl(getApplicationData().getFrame("/P3")),
   spl(getApplicationData().getFrame("/P4")),
   circ(getApplicationData().getFrame("/P5"),
        getApplicationData().getFrame("/P6")),
   spl(getApplicationData().getFrame("/P7")),
   lin(getApplicationData().getFrame("/P8"))
   );
...
robot.move(ptp(getApplicationData().getFrame("/P0")));
robot.move(mySpline);
```

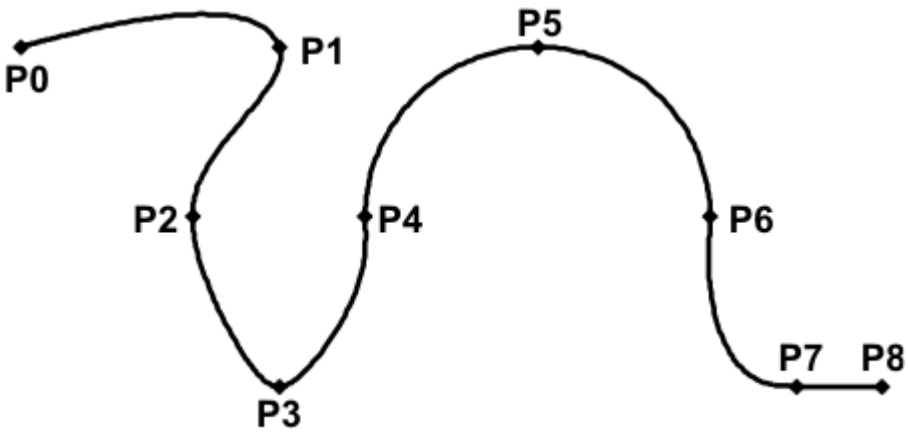


Fig. 14-8: Original path

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 a CIRC segment and a circular path is thus defined.

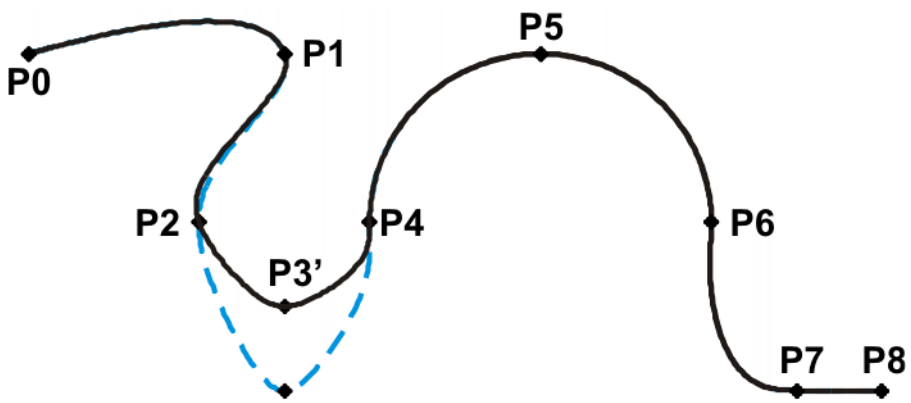


Fig. 14-9: Point has been offset

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 LIN. The path changes in segments P1 - P2, P2 - P3 and P3 - P4.

```
Spline mySpline = new Spline(
   spl(getApplicationData().getFrame("/P1")),
   spl(getApplicationData().getFrame("/P2")),
   lin(getApplicationData().getFrame("/P3")),
   spl(getApplicationData().getFrame("/P4")),
   circ(getApplicationData().getFrame("/P5"),
        getApplicationData().getFrame("/P6")),
   spl(getApplicationData().getFrame("/P7")),
   lin(getApplicationData().getFrame("/P8"))
   );
...
robot.move(ptp(getApplicationData().getFrame("/P0")));
robot.move(mySpline);
```

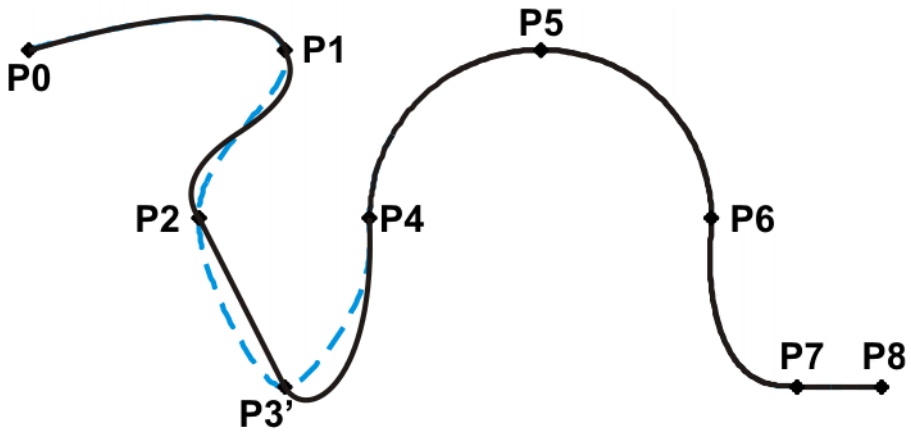


Fig. 14-10: Segment type has been changed

Example 2

Original path:

```
Spline mySpline = new Spline(
   spl(getApplicationData().getFrame("/P2")),
   spl(getApplicationData().getFrame("/P3")),
   spl(getApplicationData().getFrame("/P4")),
   spl(getApplicationData().getFrame("/P5")),
```

```
    );
...
robot.move(mySpline);
```

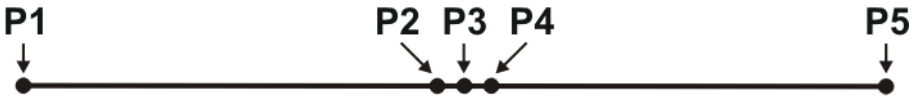


Fig. 14-11: Original path

The following frame coordinates were taught:

Frame	X	Y	Z
P2	100.0	0.0	0.0
P3	102.0	0.0	0.0
P4	104.0	0.0	0.0
P5	204.0	0.0	0.0

A point is offset relative to the original path:

P3 is moved slightly in the Y direction. This causes the path to change in all the segments illustrated.

Frame	X	Y	Z
P3	102.0	**1.0**	0.0

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.

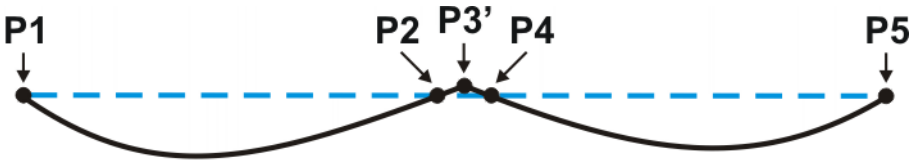


Fig. 14-12: Point has been offset

Remedy:

- Distribute the points more evenly.
- Program straight lines (except very short ones) as LIN segments

14.6.3 LIN-SPL-LIN transition

In the case of a LIN-SPL-LIN 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.

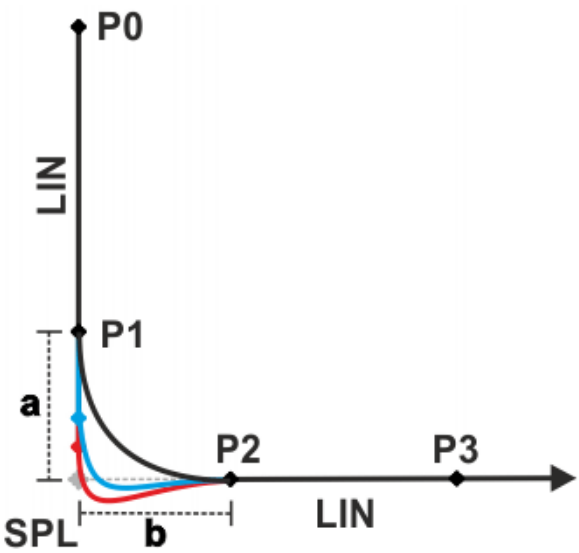


Fig. 14-13: LIN-SPL-LIN

The path remains inside the smaller angle if the following conditions are met:

- The extensions of the two LIN segments intersect.
- 2/3 ≤ a/b ≤ 3/2

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

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

14.7 Manual guidance motion type

Description

The robot can be guided using a hand guiding device. The hand guiding device is a device equipped with an enabling device and which is required for the manual guidance of the robot.

Manual guidance mode can be switched on in the application using the motion command handGuiding(). Manual guidance begins at the actual position which was reached before the mode was switched on.

(>>> 15.10 "Programming manual guidance" Page 286)

In Manual guidance mode, the robot reacts compliantly to outside forces and can be manually guided to any point in the Cartesian space. The impedance parameters are automatically set when the robot is switched to Manual guidance mode. It is not possible to change the impedance parameters for manual guidance.

Precondition

- Hand guiding device with enabling device
- ESM state for manual guidance motion has been configured.

  The ESM state contains the AMF *Hand guiding device enabling inactive*, which checks whether the enabling signal has not been issued on the hand guiding device.

  (>>> 13.8.5 "Manual guidance with enabling device and velocity monitoring" Page 192)

  The robot can be guided manually when the enabling switch on the hand guiding device is pressed and held in the center position. If the enabling switch is pressed down fully or released, the signal for manual guidance is cancelled and the robot remains in its current position.

- ESM state for all motions except manual guidance motion has been configured.

  The ESM state does not contain the AMF *Hand guiding device enabling inactive*. The signal at the hand guiding device is not evaluated.
- Automatic mode

  The safety equipment must be HRC-compliant.

In Manual guidance mode, incorrectly selected parameters (e.g. incorrect load data, incorrect tool), incorrect information (e.g. from defective torque sensors) or additional overlaid forces can be interpreted as external forces, resulting in unpredictable robot motions.

If the signal for manual guidance is issued before Manual guidance mode is switched on in the application, Manual guidance mode will be active as soon as it is switched on. This means that motion execution is not paused when the mode is switched on, making for a smooth transition between Application mode and Manual guidance mode.
Precondition for this response: the application velocity is less than the maximum permissible velocity configured for manual guidance.
(>>> 13.8.5.3 "Velocity monitoring during manual guidance" Page 195)
If the application is executed at a higher velocity, the application is paused before switching to manual guidance mode. (Then release the enabling switch, press the Start key and wait until the application is paused again.)

14.8 Approximate positioning

Approximate positioning means that the motion does not stop exactly at the end point of the programmed motion, allowing continuous robot motion. During motion programming, different parameters can influence the approximate positioning.

The point at which the original path is left and the approximate positioning arc begins is referred to as the approximate positioning point.

To approximate motions without exact positioning, they must be executed asynchronously or grouped in a MotionBatch.
(>>> 15.7.1 "Structure of a motion command (move/moveAsync)" Page 276)
(>>> 15.7.6 "MotionBatch" Page 280)

In the case of approximate positioning of motions executed synchronously, an exact positioning point is executed at the start of the approximate positioning arc. This also applies, in the case of synchronous execution, to the last motion within a MotionBatch.

PTP motion

The TCP leaves the path that would lead directly to the end point and moves, instead, along a path that allows it to pass the end point without exact positioning. The path thus goes past the point and no longer passes through it.

During programming, the relative maximum distance from the end point at which the TCP may deviate from its original path in axis space is defined. A relative distance of 100% corresponds to the entire path from the start point to the end point of the motion.

The approximation contour executed by the TCP is not necessarily the shorter path in Cartesian space. The approximated point can thus also be located within the approximate positioning arc.

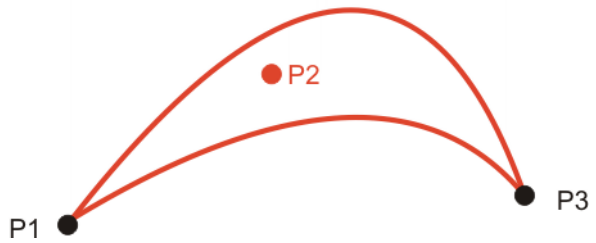


Fig. 14-14: PTP motion, P2 is approximated

LIN motion

The TCP leaves the path that would lead directly to the end point and moves along a shorter path. During programming of the motion, the maximum distance from the end point at which the TCP may deviate from its original path is defined.

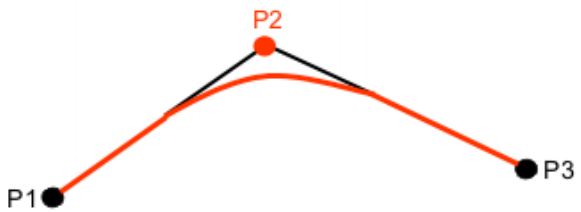


Fig. 14-15: LIN motion, P2 is approximated

CIRC motion

The TCP leaves the path that would lead directly to the end point and moves along a shorter path. During programming of the motion, the maximum distance from the end point at which the TCP may deviate from its original path is defined.

The auxiliary point may fall within the approximate positioning range and not be passed through exactly. This is dependent on the position of the auxiliary point and the programmed approximation parameters.

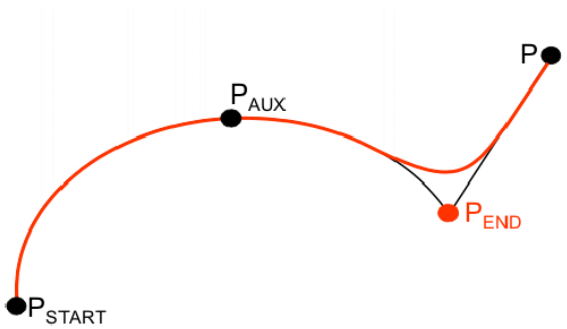


Fig. 14-16: CIRC motion, P_{END} is approximated

All spline blocks and all individual motions can be approximated with one another. It makes no difference whether they are CP or JP 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 PTP-LIN approximation, for example, the approximate positioning arc is of type CP.

If a spline block is approximated, the entire last segment is approximated. If the spline block only consists of one segment, a maximum of half the segment is approximated (this also applies for PTP, LIN and CIRC).

Approximate positioning not possible due to time:

If approximation is not possible due to delayed motion commanding, the robot waits at the start of the approximate positioning arc. The robot moves again as soon as it has been possible to plan the next block. The robot then executes the approximate positioning arc. Approximate positioning is thus technically possible; it is merely delayed.

No approximate positioning in Step mode:

In Step mode, 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 in relation to the path in standard mode.

In all other segments of both spline blocks, the path is identical in both program run modes.

Approximated motions which were sent to the robot controller asynchronously before Step mode was activated and which are waiting there to be executed will stop at the approximate positioning point. For these motions, the approximate positioning arc will be executed when the program is resumed.

14.9 Orientation control with LIN, CIRC, SPL

Description

The orientation of the TCP can be different at the start point and end point of a motion. During motion programming, it is possible to define how to deal with the different orientations.

Orientation control is set as a motion parameter by the setOrientationType(…) method. Orientation control is a value of type Enum SplineOrientationType.

Orientation control	Description
Constant	The orientation of the TCP remains constant during the motion. The orientation of the start point is retained. The orientation of the end point is not taken into consideration.
Ignore	The orientation of the TCP changes during the motion. This option is only available for individual spline segments, not for the entire spline block or individual motions. The controller calculates the orientation control on the basis of the orientation of the surrounding control points, unless their orientation is also ignored. Ignore is used if no specific orientation is required for a spline segment. (>>> "Ignore" Page 255) **Note**: In the case of Ignore, the orientation of the end point is not taken into consideration. If it is important for the taught orientation to be maintained at the end point, e.g. to avoid collisions, Ignore must not be used.

Orientation control	Description
OriJoint	The orientation of the TCP changes continuously during the motion. This is done by linear transformation (axis-specific motion) of the wrist axis angles. **Note**: Use OriJoint if, with VariableOrientation, the robot passes through a wrist axis singularity. The orientation of the TCP changes continuously during the motion, but not uniformly. OriJoint is thus not suitable if a specific orientation must be maintained exactly, e.g. in the case of laser welding.
VariableOrientation	During the motion, a continuous transition of the orientation of the TCP occurs from the orientation of the start point to the orientation of the end point. If the orientation control is not set, this orientation control applies as the default.

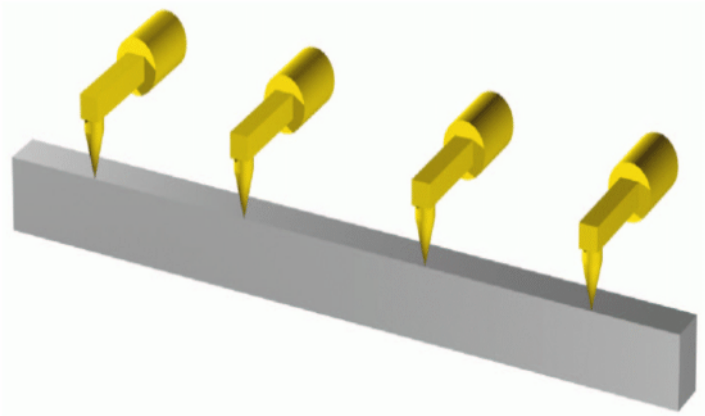


Fig. 14-17: Constant orientation (constant)

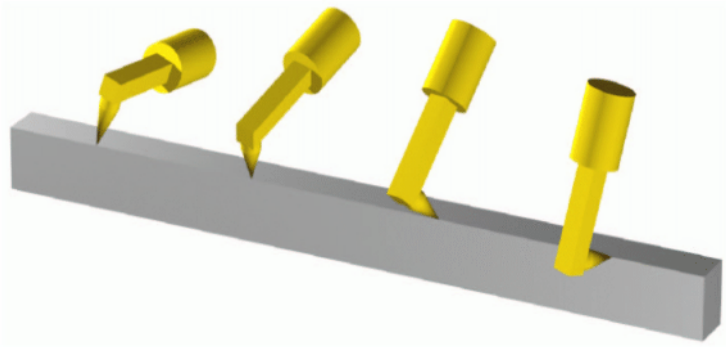


Fig. 14-18: Variable orientation (VariableOrientation or OriJoint)

CIRC motion

It is possible to define for CIRC motions whether the orientation control is to be space-related or path-related.

(>>> 14.9.1 "CIRC – reference system for the orientation control" Page 256)

During CIRC motions, the robot controller only takes the orientation of the end point into consideration. It is possible to define whether, and to what extent, the orientation of the auxiliary point is to be taken into consideration. The orientation behavior at the end point can also be defined.

Ignore

The orientation type SplineOrientationType.Ignore is used if no specific orientation is required at a point. The robot controller ignores the taught or programmed orientation of the point. Instead, it calculates the optimal orientation

for this point on the basis of the orientations of the surrounding points. This reduces the cycle time.

Example:

```
robot.move(P0);
Spline path6 = new Spline(
   spl(P1),
   spl(P2),
   spl(P3).setOrientationType(SplineOrientationType.Ignore),
   spl(P4).setOrientationType(SplineOrientationType.Ignore),
   spl(P5),
   spl(P6)
   ),
...
robot.move(path6);
```

The taught or programmed orientation of P3 and P4 is ignored.

SplineOrientationType.Ignore is not allowed for the following spline segments:

- The first and last segment in a spline block
- CIRC segments with OrientationReferenceSystem.Path
- Segments followed by a CIRC segment with OrientationReferenceSystem.Path
- Segments followed by a segment with SplineOrientationType.Constant
- Successive segments in a spline block with the same end point

14.9.1 CIRC – reference system for the orientation control

Description

It is possible to define for CIRC motions whether the orientation control is to be space-related or path-related.

The reference system of the orientation control is set as a motion parameter by the setOrientationReferenceSystem(…) method. Orientation control is a value of type Enum OrientationReferenceSystem.

The reference system of the orientation control can only be specified before the orientation type.

Reference system	Description
Base	Base-related orientation control during the circular motion
Path	Path-related orientation control during the circular motion

Example

Path-related circular motion with constant orientation:

```
robot.move(circ(P6, P7)
.setOrientationReferenceSystem(OrientationReferenceSystem.Path)
.setOrientationType(SplineOrientationType.Constant));
```

Restriction

OrientationReferenceSystem.Path is not allowed for the following spline segments:

- CIRC segments with SplineOrientationType.Ignore
- CIRC segments preceded by a segment with SplineOrientationType.Ignore

14.9.2 CIRC – combinations of reference system and type for the orientation control

> If the reference system of the orientation control is combined with SplineOrientationType.OriJoint, the reference system has no influence on the orientation control.

Path-related circular motion with constant orientation:

- OrientationReferenceSystem.Path
- SplineOrientationType.Constant

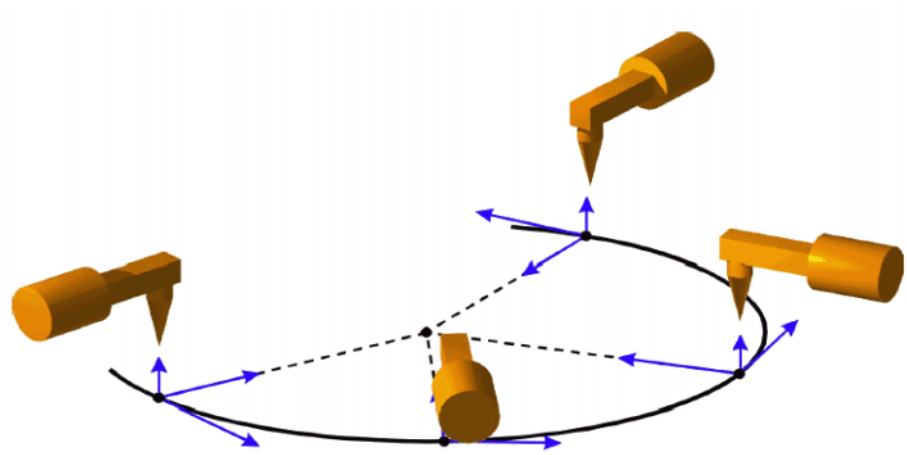


Fig. 14-19: Constant orientation, path-related

Path-related circular motion with variable orientation:

- OrientationReferenceSystem.Path
- SplineOrientationType.VariableOrientation

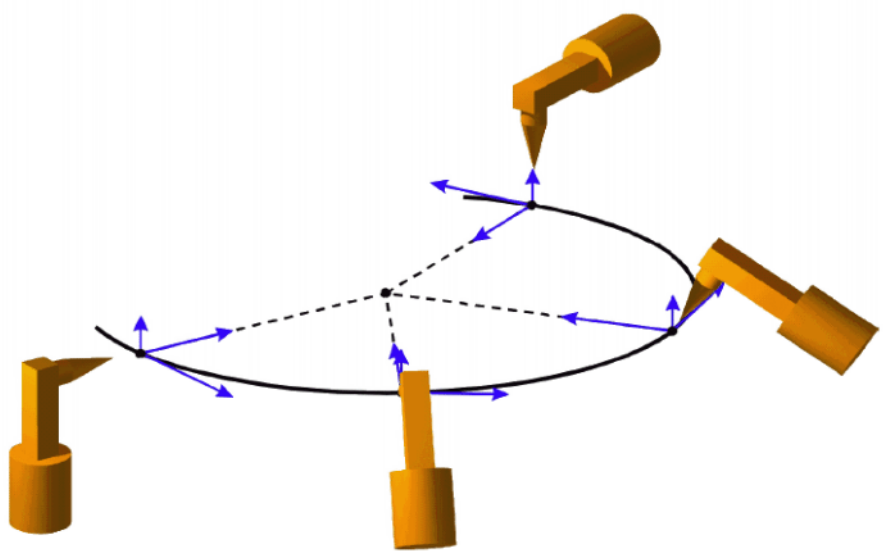


Fig. 14-20: Variable orientation, path-related

Base-related circular motion with constant orientation:

- OrientationReferenceSystem.Base
- SplineOrientationType.Constant

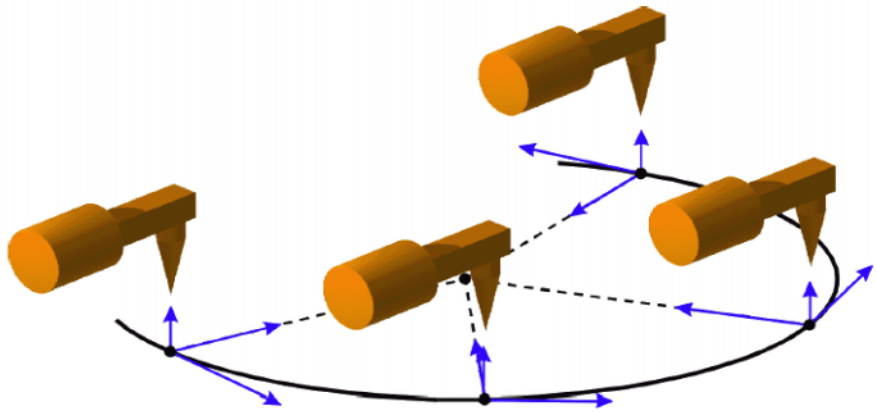


Fig. 14-21: Constant orientation, base-related

Base-related circular motion with variable orientation:

- OrientationReferenceSystem.Base
- SplineOrientationType.VariableOrientation

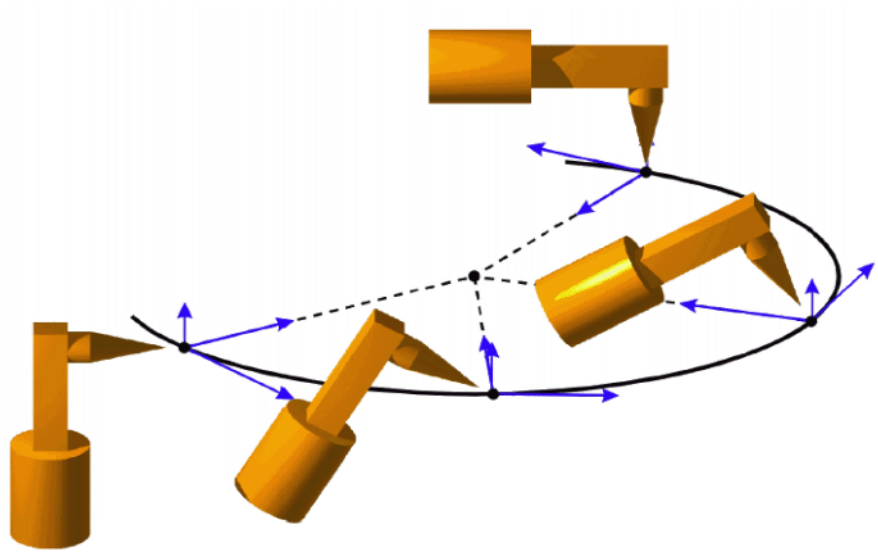


Fig. 14-22: Variable orientation, base-related

14.10 Redundancy information

For a given axis position of a robot, the resulting point in Cartesian space at which the TCP is located is unambiguously defined. Conversely, however, the axis position of the robot cannot be unambiguously determined from the Cartesian position X, Y, Z and orientation A, B, C of the TCP. A Cartesian point can be reached with multiple axis configurations. In order to determine an unambiguous configuration, the Status parameter must be specified.

Robots with 6 axes already have ambiguous axis positions for a given Cartesian point. With its additional 7th axis, the KUKA LBR iiwa can theoretically reach a given position and orientation with any number of axis poses. To unambiguously determine the axis pose for an LBR iiwa, the redundancy angle, in addition to the Status, must be specified.

The Turn parameter is required for axes which can exceed the angle ±180°. In PTP motions, this helps to unambiguously define the direction of rotation of the axes. Turn has no influence on CP motions.

Status, Turn und the redundancy angle are saved during the teaching of a frame. They are managed as arrays of the data type AbstractFrame.

Programming

The Status of a frame is only taken into account in PTP motions to this frame. With CP motions, the Status given by the axis configuration at the start of the motion is used.

In order to avoid an unpredictable motion at the start of an application and to define an unambiguous axis configuration, it is advisable to program the first motion in an application with one of the following instructions: The axis configuration should not be in the vicinity of a singular axis position.

- PTP motion to a specified axis configuration with specification of all axis values:

  `ptp(double a1, double a2, double a3, double a4, double a5, double a6, double a7)`
- PTP motion to a specified axis configuration:

  `ptp(JointPosition joints)`
- PTP motion to a taught frame (AbstractFrame type):

  `ptp(getApplicationData().getFrame(String frameName));`

14.10.1 Redundancy angle

With its 7th axis, the KUKA LBR iiwa is able to reach a point in space with a theoretically unlimited number of different axis configurations. An unambiguous pose is defined via the redundancy angle.

In an LBR iiwa, the redundancy angle has the value of the 3rd axis.

The following applies for all motions:

- The redundancy angle of the end frame is taken into account when the robot that was used when teaching the frame also executes the motion command. In particular, the robot name defined in the station configuration must match the device specified in the frame properties.
- If the robots do not match or if calculated frames are used, the redundancy angle given at the start of motion by the axis configuration is retained.

14.10.2 Status

The Status specification prevents ambiguous axis positions. The Status is described by a binary number with 3 bits.

Bit 0

Bit 0 specifies the position of the wrist root point (intersection of axes A5, A6, A7) with reference to the X-axis of the coordinate system of axis A1. The alignment of the A1 coordinate system is identical to the robot base coordinate system if axis A1 is at 0°. It moves with axis A1.

Position	Value
Overhead area The robot is in the overhead area if the x-value of the position of the wrist root point, relative to the A1 coordinate system, is negative.	Bit 0 = 1
Basic area The robot is in the basic area if the x-value of the position of the wrist root point, relative to the A1 coordinate system, is positive.	Bit 0 = 0

Bit 1

In an LBR iiwa, bit 1 specifies the position of axis A4.

Position	Value
A4 < 0°	Bit 1 = 1
A4 ≥ 0°	Bit 1 = 0

Bit 2

In an LBR iiwa, bit 2 specifies the position of axis A6.

Position	Value
A6 ≤ 0°	Bit 2 = 1
A6 > 0°	Bit 2 = 0

The following applies for PTP motions:

- The Status of the end frame is taken into account when the robot which was used when teaching the frame also executes the motion command. In particular, the robot name defined in the station configuration must match the device specified in the frame properties.
- If the robots do not match or if calculated frames are used, the Status given at the start of motion by the axis configuration is retained.

The following applies for CP motions:

- The Status of the end frame is not taken into account. The Status given by the axis configuration at the start of the motion is retained.
- **Exception**: A change of Status is possible if the end frame is addressed with the `SplineOrientationType.OriJoint` orientation control. The status of the end frame is not taken into consideration in this case either. The Status at the end of the motion is determined by the path planning, which selects the shortest route to the end frame.

14.10.3 Turn

The Turn specification makes it possible to move axes through angles greater than +180° or less than -180° without the need for special motion strategies (e.g. auxiliary points). The Turn is specified by a binary number with 7 bits.

With rotational axes, the individual bits determine the sign before the axis value in the following way:

Bit = 0: Angle ≥ 0°

Bit = 1: Angle < 0°

Value	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	A7 ≥ 0°	A6 ≥ 0°	A5 ≥ 0°	A4 ≥ 0°	A3 ≥ 0°	A2 ≥ 0°	A1 ≥ 0°
1	A7 < 0°	A6 < 0°	A5 < 0°	A4 < 0°	A3 < 0°	A2 < 0°	A1 < 0°

The Turn is not taken into account in an LBR iiwa because none of its axes can rotate over ±180°.

14.11 Singularities

Due to the axis position, Cartesian motions of the robot may be limited. Due to the combination of axis positions of the entire robot, no motions can be transferred from the drives to the flange (or to an object on the flange, e.g. a tool) in at least one Cartesian direction. In this case, or if very slight Cartesian changes require very large changes to the axis angles, one speaks of singularity positions.

> It is advisable to move the robot as slowly as possible near singularities.

14.11.1 Kinematic singularities

The flexibility due to the redundancy of a 7-axis robot, in contrast to the 6-axis robot, requires 2 or more kinematic conditions (e.g. extended position, 2 rotational axes coincide) to be active at the same time in order reach a singularity position. There are 4 different robot positions in which flange motion in one Cartesian direction is no longer possible. Here only the position of 1 or 2 axes is important in each case. The other axes can take any position.

A4 singularity

This kinematic singularity is given when A4 = 0°. It is called the extended position.

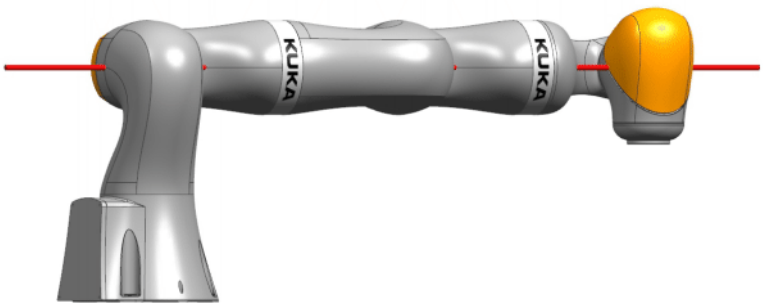


Fig. 14-23: Extended position A4 = 0°

Motion is blocked in the direction of the robot base or parallel to axis A3 or A5. An additional kinematic condition for this singularity is reaching the workspace limit. It is automatically met through A4 = 0°.

An extended robot arm causes a degree of freedom for the motion of the wrist root point to be lost (it can no longer be moved along the axis of the robot arm). The position of axes A3 and A5 can no longer be resolved.

A4/A6 singularity

This kinematic singularity is given when A4 = 90° and A6 = 0°.

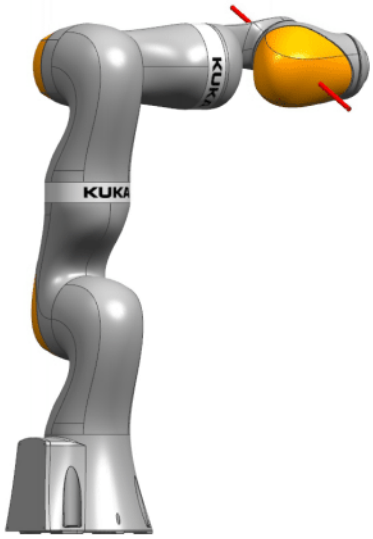


Fig. 14-24: A4 = 90° and A6 = 0°

Motion parallel to axis A6 or A2 is blocked.

A2/A3 singularity

This kinematic singularity is given when A2 = 0° and A3 = ±90° (π/2).

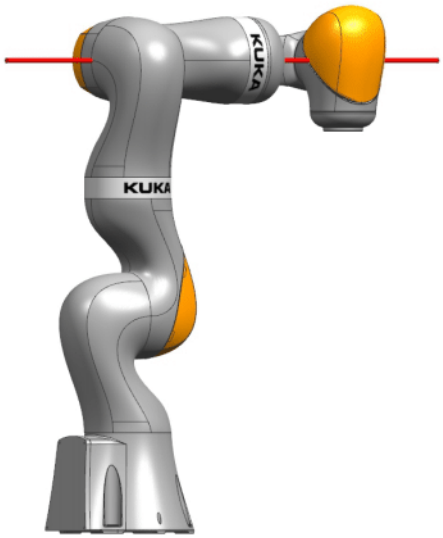


Fig. 14-25: A2 = 0° and A3 = ±90° (π/2).

Motion is blocked in the direction of the robot or parallel to axis A2 or A5.

A5/A6 singularity

This kinematic singularity is given when A5 = ±90° (π/2) and A6 = 0°.

Fig. 14-26: A5 = ±90° (π/2) and A6 = 0°

Motion parallel to axis A6 is blocked.

14.11.2 System-dependent singularities

The redundant configuration of the LBR with its 7th axis allows the robot arm to move without the flange moving. In this so-called “null space motion” all axes move except A4, the “elbow axis”. In addition to the normal redundancy, it is possible, under certain circumstances, that only subchains of the robot can move and not all axes.

For all of the robot settings in this category, slight Cartesian changes result in very large changes to the axis angles. They are very similar to the singularities in 6-axis robots, since a division is also made in the position and orientation part of the wrist root point in the LBR.

Wrist axis singularity

Wrist axis singularity means the axis position A6 = 0°. Thus the position of axes A5 and A7 cannot be restored and there are an infinite number of ways to position these two axes to generate the same position on the flange.

A1 singularity

If the wrist root point is directly over A1, no reference value can be given for the redundancy circle according to the definition above, because any A1 value is permissible here for A3 = 0°.

Every axis position of A1 can be compensated for with a combination of A5, A6 and A7 so that the flange position remains unchanged.

A2 singularity

With an extended “shoulder”, the position of axes A1 and A3 can no longer be resolved according to the pattern above.

A2/A4 singularity

If A1 and A7 coincide, the position of axes A1 and A7 can no longer be resolved according to the pattern above.

> System-dependent singularities can be avoided in most cases by a suitable elbow position.

15 Programming

15.1 Java Editor

15.1.1 Opening a robot application in the Java Editor

Description

The Java Editor allows more than one file to be open simultaneously. If required, they can be displayed side by side or one above the other. This provides a convenient way of comparing contents, for example.

Precondition

- The robot application has been created.
  (>>> 5.4 "Creating a new robot application" Page 51)

Procedure

1. Double-click on a Java file in the **Package Explorer**.
   Or: Select the file and the menu sequence **Navigate** > **Open**.
   Or: Right-click on the file and select **Open** or **Open With** > **Java Editor** from the context menu.
2. To close the file: Click on the “X” at the top right of the corresponding tab.

15.1.2 Structure of a robot application

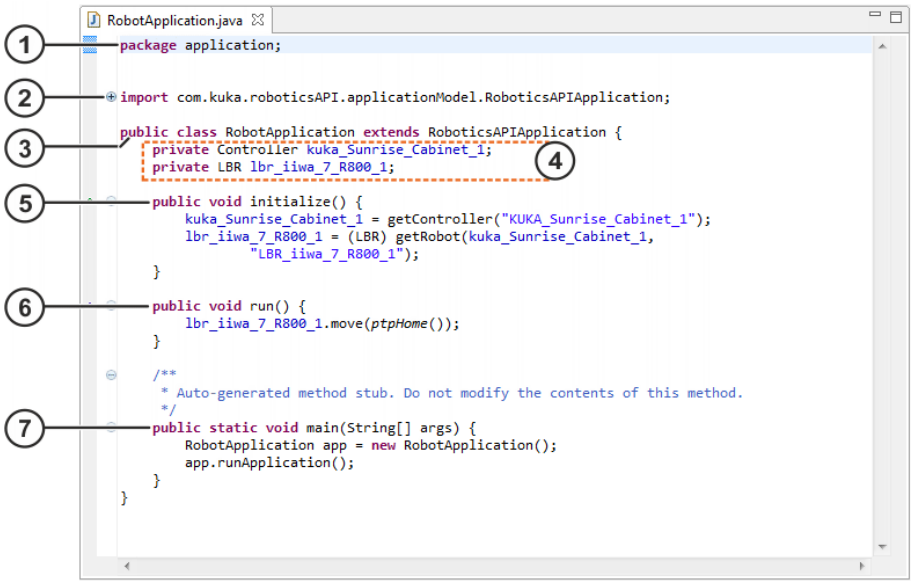


Fig. 15-1: Structure of a robot application

Item	Description
1	This line contains the name of the package in which the application is located.
2	The **import** section contains the imported classes which are required for programming the application. **Note**: Clicking on the “+” icon opens the section, displaying the imported classes.
3	Header of the application (contains the class names of the application) (>>> "Header" Page 266)

Item	Description
4	Declaration section The data arrays of the classes required in the application are declared here. When the application is created, one instance is automatically created for each of the following classes. ■ **Controller**: robot controller used ■ **LBR**: robot used
5	**initialize()** method In this method, the data arrays created in the declaration section are assigned initial values.
6	**run()** method The robot is programmed in this method. When the application is created, a motion instruction which moves the robot to the HOME position is automatically inserted. (>>> 15.16 "HOME position" Page 312)
7	**main()** method This method is necessary for the application to be executed. It must not be changed.

Header

In a robot application, this is the special form of Java class:

```
public class RobotApplication extends RoboticsAPIApplication
```

Element	Description
public	The keyword **public** designates a class which is publically visible. Public classes can be used across all packages.
class	The keyword **class** designates a Java class. The name of the class is derived from the name of the application.
extends	The application is subordinate to the **RoboticsAPIApplication** class.

15.1.3 Edit functions

15.1.3.1 Renaming a variable

Description

A variable name can be changed in a single action at all points where it occurs.

Procedure

1. Select the desired variable at any point.
2. Right-click and select **Refactor** > **Rename…** from the context menu.
3. The variable is framed in blue and can now be edited. Change the name and confirm with the Enter key.

15.1.3.2 Auto-complete

Description

An auto-complete function is available in the Java Editor.

When entering code, it is possible to display an “Auto-complete” list containing entries which are compatible with characters which have already been entered. These entries are prioritized according to their frequency of use, i.e. the selection is dynamically adapted to the user’s actions.

An entry from the “Auto-complete” list can be inserted into the program code as needed. This makes it unnecessary to retype the complex syntax of meth-

ods, for example. All that is then required is to enter the variable elements in the syntax manually.

Procedure

1. Begin typing the code.

> When entering a dot operator for a data array or enum, the “Auto-complete” list is automatically displayed. The list contains the following entries:
> - Available methods of the corresponding class (only for data arrays)
> - Available constants of the corresponding class

2. Press CTRL + space bar. The “Auto-complete” list containing the available entries is displayed.

> If the list contains only one appropriate entry, this can automatically be inserted into the program code by pressing CTRL + space bar.

3. Select the appropriate entry from the list and press the Enter key. The entry is inserted in the program code.

   If an entry is selected, the Javadoc information on this entry is displayed automatically.

   (>>> 15.1.4 "Displaying Javadoc information" Page 269)
4. Complete the syntax if necessary.

Navigating and filtering

There are various ways to navigate to the “Auto-complete” list and to filter the available entries:

- Use the arrow keys on the keyboard to move from one entry to the next (up or down)
- Scroll
- Complete the entered code with additional characters. The list is filtered and only the entries which correspond to the characters are displayed.
- Press CTRL + space bar. Only the available template suggestions are displayed.

15.1.3.3 Templates – Fast entry of Java statements

Description

Templates for fast entry are available in the Java Editor for common Java statements, e.g. a FOR loop.

Procedure

1. Begin typing the code.
2. Press CTRL + space bar. A list of the template suggestions that are compatible with the characters already entered is displayed.
3. Accept the instruction with the Enter key. Or double-click on a different instruction.
4. Complete the syntax.

Alternative procedure

Selecting templates in the **Templates** view:

1. Select the menu sequence **Window** > **Show View** > **Other...**. The **Show View** window opens.
2. In the **General** folder, select **Templates**. Click on **OK** to confirm. The Templates view opens.
3. Position the cursor in the line in which the code template is to be inserted.
4. In the Templates view, double-click on the desired template under **Java statements**. The code is inserted in the editor.
5. Complete the syntax.

15.1.3.4 Creating user-specific templates

Description

Users can create their own templates, e.g. templates for motion blocks with specific motion parameters which are used frequently during programming.

Procedure

1. In the Templates view, select the context in which the template is to be inserted.
2. Right-click on the context and select **New...** from the context menu.

   Or: Click on the **Create a New Template** icon.

   The **New Template** window opens.
3. Enter a name for the template in the **Name** box.
4. Enter a description in the **Description** box (optional).
5. In the **Pattern** box, enter the desired code.
6. Confirm the template properties with **OK**. The template is created and inserted into the Templates view.

15.1.3.5 Extracting methods

Description

Parts of the program code can be extracted from the robot application and made available as a separate method. This makes particular sense for frequently recurring tasks, as it increases clarity within the robot application.

Procedure

1. Select the desired program code.
2. Right-click in the editor area.
3. Select **Refactor** > **Extract Method…** from the context menu. The **Extract Method** window opens.
4. In **Method name**, enter a unique method name and select the desired **Access modifier**. Click on **OK** to confirm.

   The selected program code is removed and the new method is created. The new method is called in the position in which the code was just removed.

Access modifier

This option defines which classes can call the extracted method.

Option	Description
private	This method can only be called by the corresponding class itself.
default	The following classes can call the method: ■ The corresponding class ■ The inner classes of the corresponding class ■ All classes of the package in which the corresponding class is located
protected	The following classes can call the method: ■ The corresponding class ■ The subclasses of the corresponding class (inheritance) ■ The inner classes of the corresponding class ■ All classes of the package in which the corresponding class is located
public	All classes can call the method, regardless of the relationship to the corresponding class and of the package assignment.

15.1.4 Displaying Javadoc information

Description

Javadoc is a documentation generated from specific Java comments. The functionalities and use of classes, methods and libraries are described in Javadoc.

The Javadoc information can be displayed during programming. The information is only available in English.

The various display functions are described using the example of the LBR class.

Procedure

Displaying Javadoc information in auto-complete:

1. In auto-complete (CTRL + space bar), select an entry in the “Auto-complete” list. The associated Javadoc information is displayed in a separate window in the editor area.

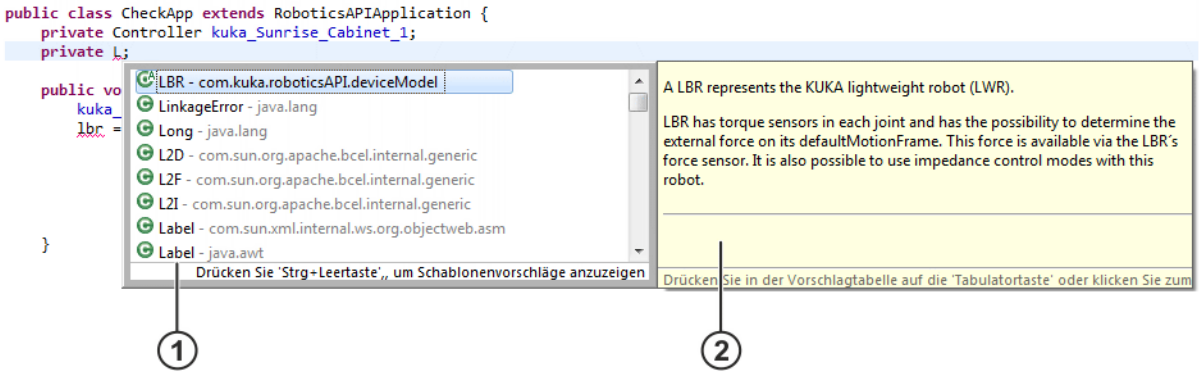


Fig. 15-2: Displaying Javadoc information in auto-complete

1	“Auto-complete” list	2	Javadoc information

2. In order to pin the window in the editor area, press the tab key or click inside the window.

   Pinning the window makes it possible to navigate to the Javadoc description, e.g. by scrolling.

Displaying Javadoc information using the mouse pointer:

- Move the mouse pointer to the desired element name in the program code. The associated Javadoc information is automatically displayed in a window in the editor area.

  The following elements react to the mouse pointer:

  - Methods
  - Classes (data types, not user-defined data arrays)
  - Interfaces
  - Enums

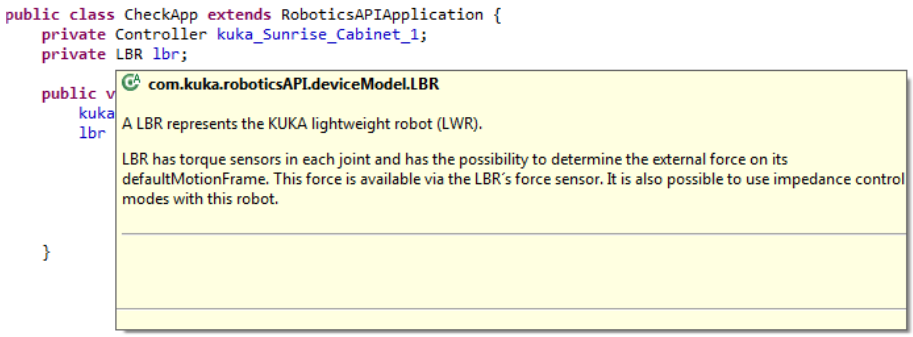


Fig. 15-3: Displaying Javadoc information using the mouse pointer

- Further display options are available from here:
  - In order to be able to navigate to the Javadoc description, e.g. by scrolling, move the mouse pointer in the window.

    The window is not pinned. If the mouse pointer is moved out of the window, the window closes.
  - In order to pin the window in the editor area, press the F2 key or click inside the window.

    It is also possible to navigate to the Javadoc description in the pinned window.
  - To additionally display the Javadoc information in the **Javadoc** view, left-click on the selected element.

    If the window is not pinned in the editor area, it is closed.

Navigation

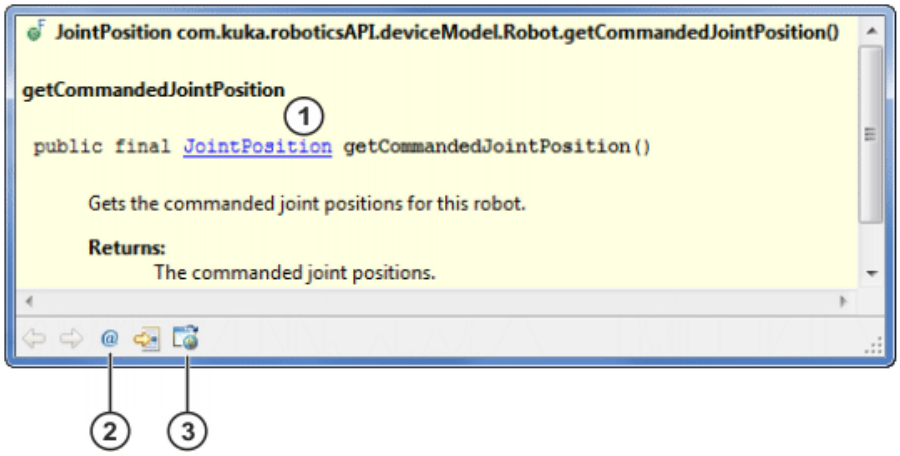


Fig. 15-4: Navigating to the Javadoc description

Item	Description
1	Linked class Left-clicking on the linked class displays the complete Javadoc information relating to this class in the Javadoc browser. **Note**: If the corresponding link in the **Javadoc** view is selected, the complete Javadoc information is displayed in the view itself.
2	**Show in Javadoc View** button The window in the editor section closes and the Javadoc information is displayed in the **Javadoc** view.
3	**Open Attached Javadoc Browser** button The window in the editor section closes and the complete Javadoc information relating to the corresponding class is displayed in the Javadoc browser.

There is a further option for displaying the complete Javadoc information on a specific element in the Javadoc browser: Select the desired element in the program code and press SHIFT + F2.

15.1.4.1 Configuration of the Javadoc browser

The configuration of the Javadoc browser is described briefly using the example of the LBR class.

1

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

com.kuka.roboticsAPI.deviceModel

Class LBR

2

```
java.lang.Object
  └ com.kuka.common.TaggableObject
      └ com.kuka.roboticsAPI.geometricModel.SceneGraphObject
          └ com.kuka.roboticsAPI.geometricModel.SpatialObject
              └ com.kuka.roboticsAPI.geometricModel.PhysicalObject
                  └ com.kuka.roboticsAPI.deviceModel.Device
                      └ com.kuka.roboticsAPI.deviceModel.Robot
                          └ com.kuka.roboticsAPI.deviceModel.LBR
```

All Implemented Interfaces:

com.kuka.common.ITaggable, IOperationModeProvider

Direct Known Subclasses:

SunriseLBR

```
public abstract class LBR
extends Robot
```

3

A LBR represents the KUKA lightweight robot (LWR).

LBR has torque sensors in each joint and has the possibility to determine the external force on its defaultMotionFrame. This force is available via the LBR's force sensor. It is also possible to use impedance control modes with this robot.

Field Summary

protected double[]	gmsSensorLimits torque sensor limits.

Fields inherited from class com.kuka.roboticsAPI.deviceModel.Device

hardwareVersion

4

Constructor Summary

LBR(Controller controller, java.lang.String name) Creates a new LBR instance with the given controller.

Method Summary

boolean	checkTorqueSensor(JointEnum joint) Checks if the torque sensor of the given axis works properly.

Field Detail

_gmsSensorLimits

```
protected double[] _gmsSensorLimits
```

torque sensor limits.

Constructor Detail

LBR

5

```
public LBR(Controller controller,
           java.lang.String name)
```

Creates a new LBR instance with the given controller.

Parameters:

controller - The controller to which this device belongs.

name - The name of the device.

Method Detail

initializeJointCount

```
protected int initializeJointCount()
```

Fig. 15-5: Configuration of the Javadoc browser

Item	Description
1	Navigation
2	Class hierarchy (>>> Fig. 15-6) The inheritance relationships of the class are displayed here.
3	Description of the class The task of the class and its functionality is described here. Special aspects of using the class are normally indicated in this area. It may also contain short examples for using the class. The earliest library version in which the class is available is normally specified at the end of the description. The description may additionally contain a list of references to further classes or methods which may be of interest.
4	Overviews ■ **Field Summary** Overview of the data fields which belong to the class The data fields inherited from a parent class are listed here. ■ **Constructor Summary** Overview of the constructors which belong to the class ■ **Method Summary** Overview of the methods which belong to the class The methods inherited from a parent class are listed here. The overviews contain short descriptions of the data fields, constructors and methods of the class, provided that these were specified during the creation of Javadoc. Inherited data fields and methods are only listed. Detailed descriptions on the data fields, constructors and methods can be found in the Details area. Click on the respective name to directly access the detailed description.
5	Details ■ **Field Detail** Detailed description of the data fields which belong to the class ■ **Constructor Detail** Detailed description of the constructors which belong to the class ■ **Method Detail** Detailed description of the methods which belong to the class The detailed description may, for example, contain a list and description of the transferred parameters and return value. Provided there are any, the exceptions which may occur when executing a method or constructor are also named here.

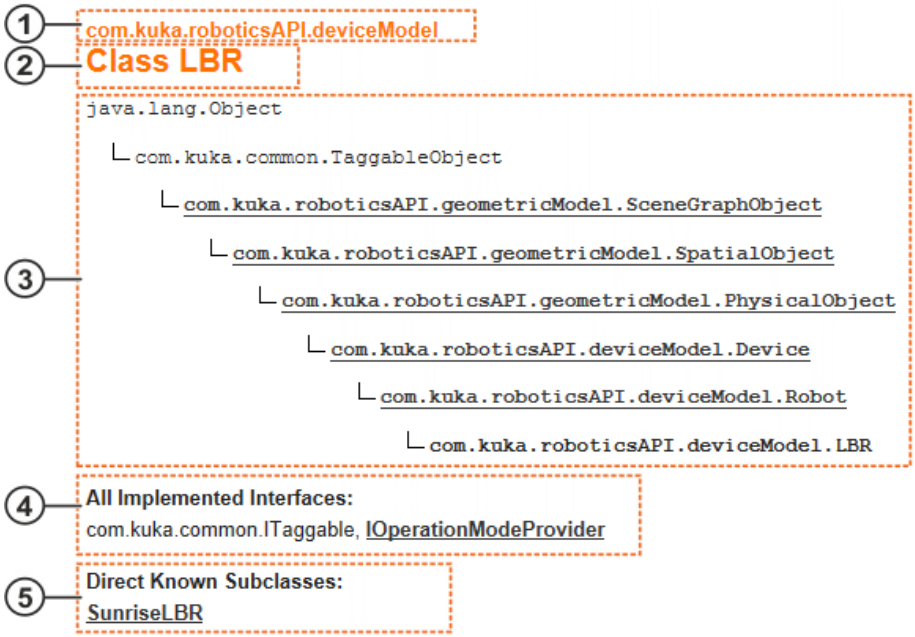


Fig. 15-6: Class hierarchy

Item	Description
1	Name of the package to which the class belongs
2	Name of the class
3	Class hierarchy (parentage of the class)
4	List of interfaces implemented by the class
5	List of subclasses derived from the class

15.2 Symbols and fonts

The following symbols and fonts are used in the syntax descriptions:

Syntax element	Appearance
Java code	■ Courier font ■ Upper/lower-case letters Examples: `private`; `new`; `linRel`; `Tool`
Elements that must be replaced by program-specific entries	■ Italics ■ Upper/lower-case letters Examples: *endpoint*; *name*; *mode*
Optional elements	■ In angle brackets Example: <`.setVelocity(`*value*`)`>
Elements that are mutually exclusive	■ Separated by the "|" symbol Example: `++` |`--`

15.3 Data types

Overview

There are 2 kinds of data type in Java:

- Primitive data types
- Complex data types
  Complex data types are defined in Java by classes.

Overview of important data types:

Data type	Description
int	Integer ■ **$-2^{31}-1$ … $+2^{31}-1$** Examples: -1; 32; 8000
double	Double-precision floating-point number ■ **-1.7E+308 … +1.7E+308** Examples: 1.25; -98.76; 123.456
Boolean	Logic state ■ **true** ■ **false**
char	Character (1 character) ■ **ASCII character** Examples: ‘A’; ‘1’; ‘q’
String	Character string ■ **ASCII characters** Examples: “KUKA”; “tool”

The names of the primitive data types are displayed in violet in the Java Editor.

15.4 Variables

Description

Before a variable can be used in the program, it must be declared, i.e. the data type and identifier must be defined. A variable can be declared in the run() method of an application, for example.

Syntax

Data type Name;

Explanation of the syntax

Element	Description
Data type	Data type of the variable
Name	Name of the variable

Examples

```
int counter;
double value;
boolean isObjectPlaced;
```

15.5 Network communication via UDP and TCP/IP

Certain ports are enabled on the robot controller for communication with external devices via UDP or TCP/IP.

The following port numbers (client or server socket) can be used in a robot application:

- 30,000 to 30,010

15.6 RoboticsAPI version information

RoboticsAPI is the programming interface for all robot-specific commands. It has a 4-figure version number. When comparing several versions, this provides information about changes to the interface.

15.6.1 Displaying the RoboticsAPI version

The RoboticsAPI version number of a project can be found in the file StationSetup.cat (station configuration).

Procedure

1. Open the project in the **Package Explorer**.
2. Open the station configuration
3. Select the **Software** tab.
4. Set the check mark at **Show libraries**.

Description

The RoboticsAPI version number is displayed in the **BasicRoboticsJavaLib** row.

- **Currently installed version** column: version last installed on the controller
- **Selected version** column: version currently being used in the Sunrise project

15.6.2 Structure of the RoboticsAPI version number:

The RoboticsAPI version number has the format XX.YY.ZZ.Build number.

- XX: Release version
- YY: Incompatible changes between versions

  In applications created while using the older API version, errors may occur when changing over to a more current version.
- ZZ: Compatible changes between versions

  Applications created while using the older API version can be also be used with the current version.
- Build number: Assigned automatically.

15.7 Motion programming: PTP, LIN, CIRC

15.7.1 Structure of a motion command (move/moveAsync)

Description

In Sunrise, motion commands can be used for all movable objects of a station. A movable object can be a robot, for example, but also a tool which is attached to the robot flange or a workpiece held by a tool (e.g. a gripper).

Motion commands can be executed synchronously and asynchronously. The methods move(…) and moveAsync(…) are available to this end:

- move(…) for synchronous execution

  Synchronous means that the motion commands are sent in steps to the real-time controller and executed. The further execution of the program is interrupted until the motion has been executed. Only then is the next command sent.
- moveAsync(…) for asynchronous execution

  Asynchronous means that the next program line is executed directly after the motion command is sent. The asynchronous execution of motions is required for approximating motions, for example.

The way in which the different motion types are programmed is shown by way of example for the object “robot”.

Motion programming for tools and workpieces is described here:
(>>> 15.11.4 "Moving tools and workpieces" Page 294)

> During programming, it is possible to specify values with a higher accuracy than the robot can achieve. For example, it is possible to specify position data in the nanometer range, but it is not possible to achieve this accuracy.

Syntax

Executing a motion synchronously:

Object`.move(`*Motion*`);`

Executing a motion asynchronously:

Object`.moveAsync(`*Motion*`);`

Explanation of the syntax

Element	Description
Object	Object of the station which is being moved This specifies the object variable name which was declared and initialized in the application.
Movement	Motion which is being executed The motion to be executed is defined by the following elements: ■ Motion type or block: **ptp**, **lin**, **circ**, **spl** or **spline**, **splineJP**, **batch** ■ End position ■ Further optional motion parameters

15.7.2 PTP

Description

Executes a point-to-point motion to the end point. The coordinates of the end point are absolute.

The end point can be programmed in the following ways:

- Insert a frame from the application data in a motion instruction.
- Create a frame in the program and use it in the motion instruction.

> The redundancy information for the end point – Status, Turn and redundancy angle – must be correctly specified. Otherwise, the end point cannot be correctly addressed.

- Specify the angles of axes A1 … A7. All axis values must always be specified.

Syntax

PTP motion with a specified frame:

`ptp(getApplicationData().getFrame("`*End point*`")<.`*Motion parameter*`>)`

PTP motion with specified axis angles:

`ptp(`*A1, A2, … A7*`)<.`*Motion parameter*`>)`

Explanation of the syntax

Element	Description
End point	Path of the frame in the frame tree or variable name of the frame (if created in the program)
A1 … A7	Axis angles of axes A1 … A7 (type: double; unit: rad)
Motion parameter	Further motion parameters, e.g. velocity and acceleration

Examples

PTP motion to the “StartPos” frame:

```
robot.move(ptp(getApplicationData().getFrame("/StartPos")));
```

PTP motion into the vertical stretch position:

```
robot.move(ptp(0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0));
```

PTP motion to the “StartPos” frame with a specified relative velocity:

```
robot.move(ptp(getApplicationData().getFrame("/StartPos"))
.setJointVelocityRel(0.25));
```

15.7.3 LIN

Description

Executes a linear motion to the end point. The coordinates of the end point are Cartesian and absolute.

The end point can be programmed in the following ways:

- Insert a frame from the application data in a motion instruction.
- Create a frame in the program and use it in the motion instruction.

Syntax

`lin(getApplicationData().getFrame("`*End point*`")<.`*Motion parameter*`>)`

Explanation of the syntax

Element	Description
End point	Path of the frame in the frame tree or variable name of the frame (if created in the program) The redundancy information for the end point – Status and Turn – are ignored in the case of LIN (and CIRC) motions. Only the redundancy angle is taken into account.
Motion parameter	Further motion parameters, e.g. velocity and acceleration

Examples

LIN motion to the “/Table/P1” frame:

```
robot.move(lin(getApplicationData().getFrame("/Table/P1")));
```

LIN motion with the Cartesian velocity specified:

```
robot.move(lin(getApplicationData().getFrame("/Table/P1"))
.setCartVelocity(150.0));
```

15.7.4 CIRC

Description

Executes a circular motion. An auxiliary point and an end point must be specified in order for the controller to be able to calculate the circular motion. The coordinates of the auxiliary point and end point are Cartesian and absolute.

The auxiliary point and end point can be programmed in the following ways:

- Insert a frame from the application data in a motion instruction.
- Create a frame in the program and use it in the motion instruction.

Syntax

```
circ(getApplicationData().getFrame("Auxiliary point"),
getApplicationData().getFrame("End point")
<.Motion parameter>)
```

Explanation of the syntax

Element	Description
Auxiliary point	Path of the frame in the frame tree or variable name of the frame (if created in the program) The redundancy information for the end point – Status, Turn and redundancy angle – are ignored.
End point	Path of the frame in the frame tree or variable name of the frame (if created in the program) The redundancy information for the end point – Status and Turn – are ignored in the case of CIRC (and LIN) motions. Only the redundancy angle is taken into account.
Motion parameter	Further motion parameters, e.g. velocity and acceleration

Examples

CIRC motion to the end frame “/Table/P4” via the auxiliary frame “/Table/P3”:

```
robot.move(circ(getApplicationData().getFrame("/Table/P3"),
getApplicationData().getFrame("/Table/P4")));
```

CIRC motion with the absolute acceleration specified:

```
robot.move(circ(getApplicationData().getFrame("/Table/P3"),
getApplicationData().getFrame("/Table/P4")).setCartAcceleration(25));
```

15.7.5 LIN REL

Description

Executes a linear motion to the end point. The coordinates of the end point are relative to the end position of the previous motion, unless this previous motion is terminated by a break condition. In this case, the coordinates of the end point are relative to the position at which the motion was interrupted.

In a relative motion, the end point is by default offset in the coordinate system of the moved frame. Another reference coordinate system in which to execute the relative motion can optionally be specified. The coordinates of the end point then refer to this reference coordinate system. This can for example be a frame created in the application data or a calibrated base.

The end point can be programmed in the following ways:

- Enter the Cartesian offset values individually.
- Use a frame transformation of type Transformation. The frame transformation has the advantage that the rotation can also be specified in degrees.

Syntax

LinRel motion with offset values:

```
linRel(x, y, z<, a, b, c>
<, Reference system>)
```

LinRel motion with frame transformation:

```
linRel(Transformation.ofDeg|ofRad(x, y, z, a, b, c)
<, Reference system>)
```

Explanation of the syntax

Element	Description
x, *y*, *z*	Offset in the X, Y and Z directions (type: double, unit: mm)
a, *b*, *c*	Rotation about the Z, Y or X axis (type: double) The unit depends on the method used: ■ Offset values and Transformation.ofRad: rad ■ Transformation.ofDeg: degrees
Reference system	Type: AbstractFrame Reference coordinate system in which the motion is executed

Examples

The moving frame is the TCP of a gripper. This TCP moves 100 mm in the X direction and 200 mm in the negative Z direction from the current position in the tool coordinate system. The orientation of the TCP does not change.

```
gripper.getFrame("/TCP2").move(linRel(100, 0, -200));
```

The robot moves 10 mm from the current position in the coordinate system of the P1 frame. The robot additionally rotates 30° about the Z and Y axes of the coordinate system of the P1 frame.

```
robot.move(linRel(Transformation.ofDeg(10, 10, 10, 30, 30, 0),
getApplicationData().getFrame("/P1")));
```

15.7.6 MotionBatch

Description

Several individual motions can be grouped in a MotionBatch and thus transmitted to the robot controller at the same time. As a result, motions can be approximated within the MotionBatch.

The motion parameters, e.g. velocity, acceleration, orientation control, etc. can be programmed for the entire batch or per motion.

> Only axis-specific motion parameters – setJoint…Rel(…) – can be specified for the entire batch. Cartesian motion parameters – setCart…(…) – must be specified in the individual block.

Both variants can appear together, e.g. to assign another parameter value to an individual motion than to the batch.

> The individual block parameter overwrites the batch parameter. This also applies if a lower parameter value is specified for the batch than for the individual block.

Syntax

Object`.move(batch(`

Motion`,`

Motion`,`

…

Motion`,`

Motion

`)<.`*Motion parameter*`>);`

Explanation of the syntax

Element	Description
Object	Object of the station which is being moved
Motion	Motion with or without motion parameters ■ **ptp, lin, circ or spline**
Motion parameter	Motion parameters which are programmed at the end of the batch apply to the entire batch. Only axis-specific motion parameters can be programmed!

15.8 Motion programming: spline

15.8.1 Programming tips for spline motions

- The number of spline segments in a spline block is only limited by the available memory.
- The planning of a spline motion with many small segments and small distances between points can take a very long time. To avoid excessively long planning times:
  - Program a maximum of 500 segments per spline block.
  - If possible, program distances between points > 5 mm.
- Spline motions (with many segments) can be programmed using an array of spline segments.
- A spline block should cover only 1 process (e.g. 1 adhesive seam). More than one process in a spline block leads to a loss of structural clarity within the program and makes changes more difficult.
- Use LIN and CIRC segments in cases where the workpiece necessitates straight lines and arcs. (Exception: use SPL segments for very short straight lines.) Otherwise, use SPL segments, particularly if the points are close together.
- Procedure for defining the path:
  a. First teach or calculate a few characteristic points. Example: points at which the curve changes direction.
  b. Test the path. At points where the accuracy is still insufficient, add more SPL points.
- Avoid successive LIN and/or CIRC segments, as this often reduces the velocity to 0. To avoid this:
  - Program SPL segments between LIN and CIRC segments. The length of the SPL segments must be at least > 0.5 mm. Depending on the actual path, significantly larger SPL segments may be required.
  - Replace a LIN segment with several SPL segments in a straight line. In this way, the path becomes a straight line.
- Avoid successive points with identical Cartesian coordinates, as this reduces the velocity to 0.
- If the robot executes points which lie on a work surface, a collision with the work surface is possible when approaching the first point.

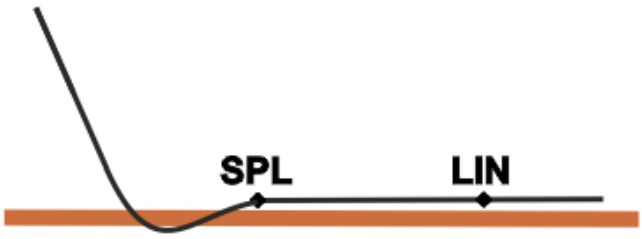


Fig. 15-7: Collision with work surface

A collision can be avoided by inserting a LIN segment before the work surface. Observe the recommendations for the LIN-SPL-LIN transition.

(>>> 14.6.3 "LIN-SPL-LIN transition" Page 250)

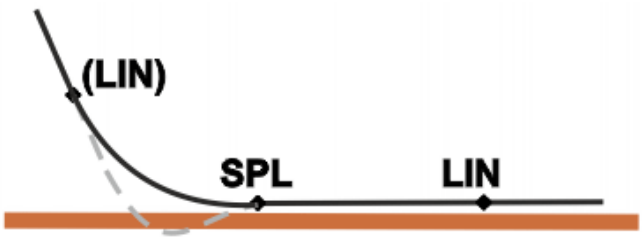


Fig. 15-8: Avoiding a collision with the work surface

- Avoid using SPL segments if the robot moves near the workspace limit. It is possible to exceed the workspace limit with SPL, even though the robot can reach the end frame in another motion type or by means of jogging.

15.8.2 Creating a CP spline block

Description

A CP spline block can be used to group together several SPL, LIN and/or CIRC segments to an overall motion.

A spline block must not include any other instructions, e.g. variable assignments or logic statements.

The motion parameters, e.g. velocity, acceleration, orientation control, etc. can be programmed for the entire spline block or per segment. Both variants can appear together, e.g. to assign a different parameter value to an individual segment than to the block.

> The individual block parameter overwrites the block parameter. This also applies if a lower parameter value is specified for the block than for the individual block.

Syntax

```
Spline Name = new Spline(
Segment,
Segment,
…
Segment,
Segment
)<.Motion parameter>;
```

Explanation of the syntax

Element	Description
Name	Name of the spline block
Segment	Motion with or without motion parameters ■ **spl**, **lin** or **circ**
Motion parameters	Motion parameters which are programmed at the end of the spline block apply to the entire spline block.

Example

```
Spline mySpline = new Spline(
   spl(getApplicationData().getFrame("/P1")),
   circ(getApplicationData().getFrame("/P2"),
        getApplicationData().getFrame("/P3")),
   spl(getApplicationData().getFrame("/P4")).setCartVelocity(150),
   lin(getApplicationData().getFrame("/P5"))
   ).setCartVelocity(250);
```

15.8.3 Creating a JP spline block

Description

A JP spline block can be used to group together several PTP segments as an overall motion.

A spline block must not include any other instructions, e.g. variable assignments or logic statements.

The motion parameters, e.g. velocity, acceleration, etc. can be programmed for the entire spline block or per segment. Both variants can appear together, e.g. to assign a different parameter value to an individual segment than to the block.

> The individual block parameter overwrites the block parameter. This also applies if a lower parameter value is specified for the block than for the individual block.

Syntax

`SplineJP` *Name* `= new SplineJP(`

Segment`,`

Segment`,`

…

Segment`,`

Segment

`)<.`*Motion parameter*`>;`

Explanation of the syntax

Element	Description
Name	Name of the spline block
Segment	PTP motion with or without motion parameters
Motion parameters	Motion parameters which are programmed at the end of the spline block apply to the entire spline block.

Example

```
SplineJP mySpline = new SplineJP(
   ptp(getApplicationData().getFrame("/P1")),
   ptp(getApplicationData().getFrame("/P2"))
   ).setJointVelocityRel(0.75);
```

15.8.4 Using spline in a motion instruction

Description

The spline motion programmed in a spline block is used as the motion type in the motion instruction.

Syntax

Object`.move(`*Spline name*`);`

Explanation of the syntax

Element	Description
Object	Object of the station which is being moved
Spline name	Name of the spline block

Example

```
robot.move(mySpline);
```

15.9 Motion parameters

The required motion parameters can be added in any order to the motion instruction. Dot separators and “set” methods are used for this purpose.

Overview

Method	Description
setCartVelocity(…)	Absolute Cartesian velocity (type: double, unit: mm/s) ■ **> 0.0** This value specifies the maximum Cartesian velocity at which the robot may move during the motion. Due to limitations in path planning, the maximum velocity may not be reached and the actual velocity may be lower. If no velocity is specified, the motion is executed with the fastest possible velocity. **Note**: This parameter cannot be set for PTP motions.
setJointVelocity-Rel(…)	Axis-specific relative velocity (type: double, unit: %) ■ **0.0 … 1.0** Refers to the maximum value of the axis velocity in the machine data. (>>> 15.9.1 "Programming axis-specific motion parameters" Page 285)
setCartAccelera-tion(…)	Absolute Cartesian velocity (type: double, unit: mm/s^2) ■ **> 0.0** If no acceleration is specified, the motion is executed with the fastest possible acceleration. **Note**: This parameter cannot be set for PTP motions.
setJointAcceleration-Rel(…)	Axis-specific relative acceleration (type: double, unit: %) ■ **0.0 … 1.0** Refers to the maximum value of the axis acceleration in the machine data. (>>> 15.9.1 "Programming axis-specific motion parameters" Page 285)
setCartJerk(…)	Absolute Cartesian jerk (type: double, unit: mm/s^3) ■ **> 0.0** If no jerk is specified, the motion is executed with the fastest possible change in acceleration. **Note**: This parameter cannot be set for PTP motions.

Method	Description
setJointJerkRel(…)	Axis-specific relative jerk (type: double, unit: %) ■ **0.0 … 1.0** Refers to the maximum value of the axis-specific change in acceleration in the machine data. (>>> 15.9.1 "Programming axis-specific motion parameters" Page 285)
setBlendingRel(…)	Relative approximation distance (type: double) ■ **0.0 … 1.0** The relative approximation distance is the furthest distance before the end point at which approximate positioning can begin. If “0.0” is set, the approximation parameter does not have any effect. The maximum distance (= 1.0) is always the length of the individual motion or the length of the last segment in the case of splines. For motions which are not commanded within a spline, only the range between 0% and 50% is available for approximate positioning. In this case, if a value greater than 50% is parameterized, approximate positioning nevertheless begins at 50% of the block length.
setBlendingCart(…)	Absolute approximation distance (type: double, unit: mm) ■ **≥ 0.0** The absolute approximation distance is the furthest distance before the end point at which approximate positioning can begin. If “0.0” is set, the approximation parameter does not have any effect.
setBlendingOri(…)	Orientation parameter for approximate positioning (type: double, unit: rad) ■ **≥ 0.0** Approximation starts, at the earliest, when the absolute difference of the dominant orientation angle for the end orientation falls below the value set here. If “0.0” is set, the approximation parameter does not have any effect.
setOrientation-Type(…)	Orientation control (type: Enum) ■ **Constant** ■ **Ignore** ■ **OriJoint** ■ **VariableOrientation** (default) (>>> 14.9 "Orientation control with LIN, CIRC, SPL" Page 254)
setOrientationRefer-enceSystem(…)	Only relevant for CIRC motions: Reference system of orientation control (type: Enum) ■ **Base** ■ **Path** (>>> 14.9.1 "CIRC – reference system for the orientation control" Page 256)

15.9.1 Programming axis-specific motion parameters

Description

The following axis-specific motion parameters can be programmed:

- Relative velocity setJointVelocityRel(…)
- Relative acceleration setJointAccelerationRel(…)
- Relative jerk setJointJerkRel(…)

There are various ways of specifying these axis-specific relative values. A valid value for all axes, different values for each individual axis or a value for an individual axis.

By way of example, these possibilities are described using the relative velocity:

- `setJointVelocityRel(Value)`

  If a value of type double is transferred, the relative velocity applies to all axes.
- `setJointVelocityRel(Array_variable)`

  In order to assign each axis its own relative velocity, a double array is transferred with the corresponding axis values. In an array, the axis values of up to 12 axes can be defined, beginning with axis A1.
- `setJointVelocityRel(Axis, Value)`

  To specify the relative velocity of an individual axis, this axis is transferred as an Enum of type JointEnum. This Enum has 12 axes (JointEnum.J1 … JointEnum.J12).

Examples

All axes move at 50% of maximum velocity:

```
robot.move(ptp(getApplicationData().getFrame("/P1"))
.setJointVelocityRel(0.5));
```

Axis A5 moves at 50%, all other axes move at 20% of maximum velocity:

```
double[] velRelJoints = {0.2, 0.2, 0.2, 0.2, 0.5, 0.2, 0.2};
robot.move(ptp(getApplicationData().getFrame("/P1"))
.setJointVelocityRel(velRelJoints));
```

Axis A4 moves at 50% of maximum velocity, all other axes move at maximum velocity:

```
robot.move(ptp(getApplicationData().getFrame("/P1"))
.setJointVelocityRel(JointEnum.J4, 0.5));
```

15.10 Programming manual guidance

Description

The robot can be guided using a hand guiding device. Manual guidance mode can be switched on in the application using the motion command handGuiding(). Manual guidance begins at the actual position which was reached before the mode was switched on.

If Manual guidance mode is used in the application, at least 2 ESM states must be configured:

- ESM state for manual guidance motion

  The ESM state contains the AMF *Hand guiding device enabling inactive*, which checks whether the enabling signal has not been issued on the hand guiding device.

  (>>> 13.8.5 "Manual guidance with enabling device and velocity monitoring" Page 192)

  It is advisable to configure a safety stop 1 (path-maintaining) as the stop reaction for the AMF *Hand guiding device enabling inactive*. Following a path-maintaining stop, the application can be resumed directly by pressing the Start key.

  If a non-path-maintaining stop reaction is configured for the AMF *Hand guiding device enabling inactive*, the robot must first be repositioned following manual guidance before the application can be resumed.

> ⚠ A risk assessment must determine whether it is permissible to configure a path-maintaining stop reaction for the EMS state which monitors the enabling switch on the hand guiding device.

- ESM state for all motions except manual guidance motion

  The ESM state does not contain the AMF *Hand guiding device enabling inactive*. The signal at the hand guiding device is not evaluated.

In the application, motions before and after manual guidance are generally required. It is advisable to monitor each of these motions using an ESM state which does not evaluate the signal on the hand guiding device, and to only switch to the ESM state for the manual guidance motion directly before switching to Manual guidance mode. If this is carried out in the application in this manner, the response is as follows:

- If the signal for manual guidance is issued before Manual guidance mode is switched on in the application, Manual guidance mode will be active as soon as it is switched on. This means that the application is not paused when the mode is switched on, making for a smooth transition between Application mode and Manual guidance mode.

  Precondition for this response: the application velocity is less than the maximum permissible velocity configured for manual guidance.

  (>>> 13.8.5.3 "Velocity monitoring during manual guidance" Page 195)

  If the application is executed at a higher velocity, the application is paused before switching to manual guidance mode. (Then release the enabling switch, press the Start key and wait until the application is paused again.)
- If the signal for manual guidance is first issued when Manual guidance mode is already switched on in the application, the Start key must be pressed in order to manually guide the robot. The pause in the application allows the operator to move his hand to the hand guiding device.
- Manual guidance mode has ended when the signal for manual guidance has been cancelled, e.g. by releasing the enabling switch. The application is paused and can only be resumed by pressing the Start key. The pause in the application allows the operator to remove his hand from the hand guiding device.

> **⚠ CAUTION** If, when switching to Manual guidance mode, the application is in an ESM state which does not contain a *Hand guiding device enabling inactive* AMF, the robot can nevertheless be manually guided in one situation: the enabling switch on the hand guiding device is pressed and a *Hand guiding device enabling inactive* AMF is configured in any other ESM state or in the PSM table.
> This combination must be avoided under all circumstances: in a situation like this, the application is not paused when manual guidance is terminated and the enabling switch on the hand guiding device is released. Instead, the application is resumed without any further operator actions. If further motions follow manual guidance, these are executed directly while the operator's hand is still on the hand guiding device and thus within the robot's motion range.

> ⚠ Switching between ESM states is effected via non-safety-oriented signals. For this reason, it must be ensured that the defined ESM state has a sufficient degree of safety, regardless of the time or place of activation. (>>> 13.2 "Safety concept" Page 169)

Preparation

The handGuiding() motion command belongs to the MMCMotions class. The class must be manually inserted into the import section of the robot application. The following line must be programmed:

```
import static com.kuka.roboticsAPI.motionModel.MMCMotions.*;
```

Syntax

Object`.move(handGuiding());`

Explanation of the syntax

Element	Description
Object	Object of the station which is being moved

Example

```
1 robot.setESMState("1");
2 robot.move(ptp(getApplicationData().getFrame("/P1")));
3 robot.setESMState("2");
4 robot.move(handGuiding());
5 robot.setESMState("1");
6 robot.move(ptp(getApplicationData().getFrame("/P2")));
```

Line	Description
1	ESM state 1 is activated for the robot. In this example, ESM state 1 monitors the operator safety.
2	Frame "/P1" is addressed with a PTP motion.
3	ESM state 2 is activated for the robot. ESM state 2 monitors the enabling switch on the hand guiding device. If a signal has not yet been issued via the switch, the configured stop reaction is triggered and the application is paused.
4	Manual guidance mode is activated. The robot can be guided manually as soon as the enabling switch on the hand guiding device is pressed and held in the center position. When the signal for manual guidance has been cancelled, e.g. by releasing the enabling switch, Manual guidance mode has ended. The stop reaction configured for ESM state 2 is triggered and motion execution is paused.
5	ESM state 1 is activated for the robot. In this example, ESM state 1 monitors the operator safety. Motion execution remains paused. The Start key must be pressed in order to resume the application.
6	Frame "/P2" is addressed with a PTP motion.

15.10.1 Axis-specific limits for manual guidance

Description

The axis range of each axis is limited by means of software limit switches. For manual guidance, user-specific axis limits can additionally be defined in the robot application.

Defining a lower and an upper axis limit results in a permissible axis range, in which manual guidance is freely possible, and 2 non-permissible axis ranges between the upper/lower axis limit and the respective software limit switch.

If a user-defined axis limit is reached during manual guidance, a virtual spring damper system is tensioned. This generates a resistance against any further motion towards the limit switch, with the resistance becoming greater the nearer an axis comes to the limit switch.

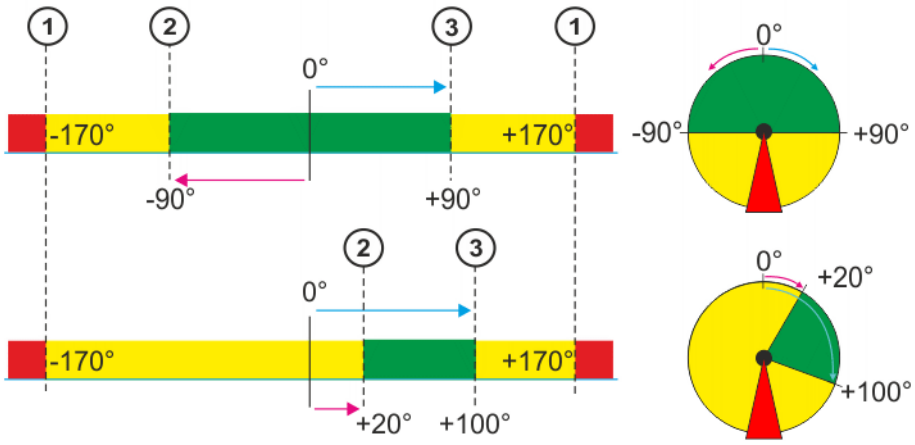


Fig. 15-9: Axis-specific limits for manual guidance (examples)

1 Position of software limit switch
2 Lower limit of the axis range
3 Upper limit of the axis range

The limits for manual guidance must be set and activated individually for each axis.

The following applies by default:

- If an axis limit is reached, all axes involved in the motion work against a further motion towards the limit switch.

  It is possible to define that only the axis that has reached the limit works against a further motion towards the limit switch.
- If an axis limit is already exceeded at the start of manual guidance, the affected axis must be moved manually out of the non-permissible range.

  It is possible to define that the axis is to move automatically out of the non-permissible range.

Overview

The required motion parameters can be added in any order to the motion command handGuiding(). Dot operators and “set” methods are used for this purpose.

Method	Description
setAxisLimits Enabled(…)	Activation of the user-specific axis limits for manual guidance (type: boolean[]) ■ **true**: Axis limit active ■ **false**: Axis limit not active **Note:** This method refers to the limits that the user can set using the methods setAxisLimitsMax(…) and setAxisLimitsMin(…). The outermost axis limits of the robot (software limit switches) are always monitored.
setAxisLimitsMax(…)	Upper axis limits (type: double[]; unit: rad)
setAxisLimitsMin(…)	Lower axis limits (type: double[]; unit: rad) **Note:** The lower axis limit must always be lower than the corresponding upper axis limit.

Method	Description
setAxisLimitViolation FreezesAll(…)	Response if an axis limit is reached (type: boolean) ■ **true**: If an axis limit is reached, all axes involved in the motion work against a further motion towards the limit switch. ■ **false**: If an axis limit is reached, only the affected axis works against a further motion towards the limit switch. Default: **true** If this value is not set, the default value is automatically applied.
setPermanentPullOn ViolationAtStart(…)	Response if an axis limit is already exceeded at the start of manual guidance (type: boolean) ■ **true**: When the enabling signal for manual guidance is issued, the axis is moved automatically out of the non-permissible range. When the permissible range is reached, the motion is stopped automatically. ■ **false**: When the enabling signal for manual guidance is issued, the axis does not move. It must be moved out of the non-permissible range manually. Default: **False** If this value is not set, the default value is automatically applied.

Example

```
private LBR robot;
private HandGuidingMotion motion;
...
motion = handGuiding()
   .setAxisLimitsMax(+1.407, +0.872, +0.087, -0.785, +0.087,
   +1.571, +0.087)
   .setAxisLimitsMin(-1.407, +0.175, -0.087, -1.571, -0.087,
   -1.571, -0.087)
   .setAxisLimitsEnabled(false, true, false, true, false,
   true, false)
   .setAxisLimitViolationFreezesAll(true)
   .setPermanentPullOnViolationAtStart(true);

robot.move(motion);
```

15.11 Using tools and workpieces in the program

An application consitutes a programmed model of a real station and must therefore contain all movable objects and fixed geometric objects in the station. Examples of movable objects for a station are robots, tools and workpieces. Examples of fixed objects are support tables or conveyors.

The robot controller and robot are automatically declared and initialized when the application is created. Tools and workpieces used in the application must be declared and initialized by the user.

Tools and workpieces with load data and geometric data are created and managed in the **Object templates** view.

(>>> 9.3 "Object management" Page 132)

Data types

The data types for the objects in a station are predefined in the RoboticsAPI:

Data type	Object
Controller	Robot controller
LBR	Lightweight robot

Data type	Object
Tool	Tool
Workpiece	Workpiece
GeometricObject	Fixed geometric objects

15.11.1 Declaring tools and workpieces

Syntax

`private` *Data type Object name*`;`

Explanation of the syntax

Element	Description
`private`	The keyword designates locally valid variables. Locally valid means that the data array can only be used by the corresponding class.
Data type	Object type
Object name	Name of the object variable

Example

In the application, 2 tools (gripper, guiding tool) and 1 workpiece (pen) are used.

```
private Tool gripper;
private Tool guidingTool;
private Workpiece pen;
```

15.11.2 Initializing tools and workpieces

Description

This section describes how tools and workpieces which were created in the object templates of the corresponding project are initialized.

Syntax

Object name `=`
`getApplicationData().createFromTemplate("`*Object template*`");`

Explanation of the syntax

Element	Description
Object name	Name of the object variable
Object template	Name of the object template as specified in the **Object templates** view

Example

The following tools and workpieces were created in the object templates:

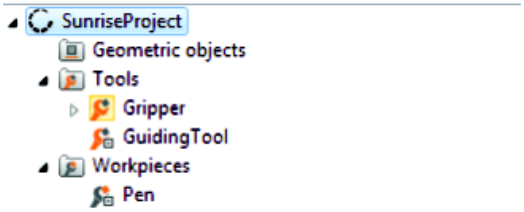


Fig. 15-10: Object templates

```
private Tool gripper;
private Tool guidingTool;
private Workpiece pen;

public void initialize() {
...
gripper = getApplicationData().createFromTemplate("Gripper");
guidingTool = getApplicationData().createFromTemplate("GuidingTool");
```

```
pen = getApplicationData().createFromTemplate("Pen");
...
}
```

15.11.3 Attaching tools and workpieces to the robot

In order to be able to use tools and workpieces as movable objects in motion instructions, they must be attached to the robot in the application via the method attachTo(…).

- Tools are directly or indirectly attached to the robot flange.
- Workpieces are indirectly attached to the robot via a tool or another workpiece.

As soon as a tool or workpiece is attached to the robot via the method attachTo(…), the load data from the robot controller are taken into account. In addition, all frames of the attached object can be used for the motion programming.

(>>> 9.3.6 "Load data" Page 136)

15.11.3.1 Attaching a tool to the robot flange

Description

Via the method attachTo(…), the origin frame of a tool is attached to the flange of a robot used in the application. The robot flange is accessed via the method getFlange().

Syntax

Tool`.attachTo(`*Robot*`.getFlange());`

Explanation of the syntax

Element	Description
Tool	Name of the tool variable
Robot	Name of the robot

Example

A guiding tool is attached to the robot flange.

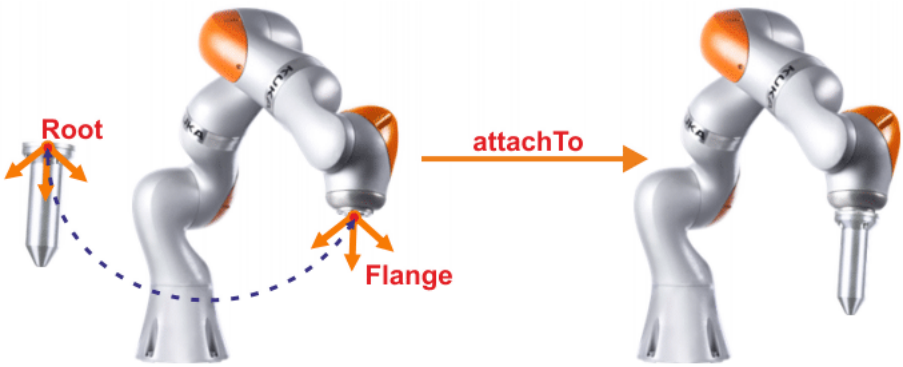


Fig. 15-11: Attaching the guiding tool to the flange.

```
private LBR robot;
private Tool guidingTool;
...
public void run() {
  ...
  guidingTool.attachTo(robot.getFlange());
  ...
}
```

15.11.3.2 Attaching a workpiece to other objects

Description

By default the origin frame of the workpiece is used to attach it to the frame of another object.

However, every other frame created for a workpiece can also be used as a reference point for attaching to another object.

Frames for tools and workpieces are created in the **Object templates** view. In order to use a frame in the program, the tool or workpiece object is polled with the method getFrame(…). As an input parameter, this contains the path of the frame as a string.

(>>> 9.3.4 "Creating a frame for a tool or workpiece" Page 134)

Syntax

To use the origin frame for the attachment:

Workpiece`.attachTo(`*Object*`.getFrame("`*End frame*`"));`

To use another reference frame for the attachment:

Workpiece`.getFrame("`*Reference frame*`").attachTo(`*Object*`.getFrame("`*End frame*`"));`

Explanation of the syntax

Element	Description
Workpiece	Name of the workpiece variable
Reference frame	Reference frame of the workpiece which is used for the attachment to the other object
End frame	Frame of the object to which the reference frame of the workpiece is attached.

After the attach, the reference frame of the workpiece and the end frame of the object connected to it match.

Example 1

A pen is attached to the gripper frame via its origin frame.

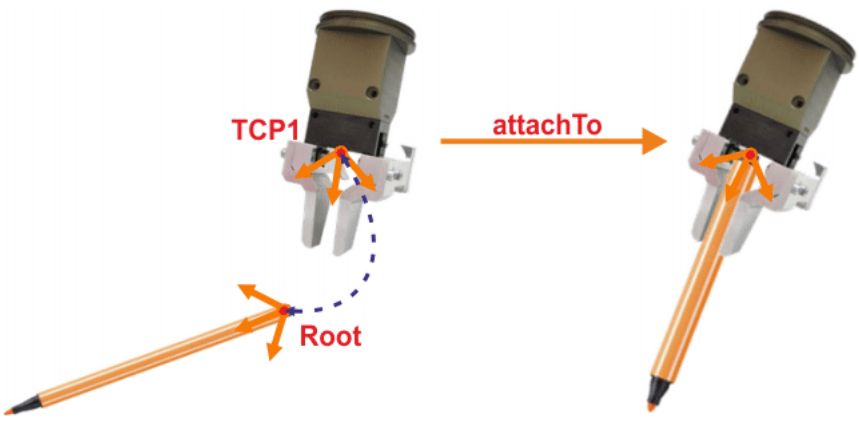


Fig. 15-12: Pen in gripper (attachment via origin frame)

```
private LBR robot;
private Tool gripper;
private Workpiece pen;
...
public void run() {
  ...
  pen.attachTo(gripper.getFrame("/TCP1"));
  ...
}
```

Example 2

A 2nd frame is defined at the tip of the gripper. If this is to be used to grip the pen, a connection via the origin frame of the pen is not possible. For this purpose, a grip point was created on the pen. This is used as the reference frame for the attachment to the gripper.

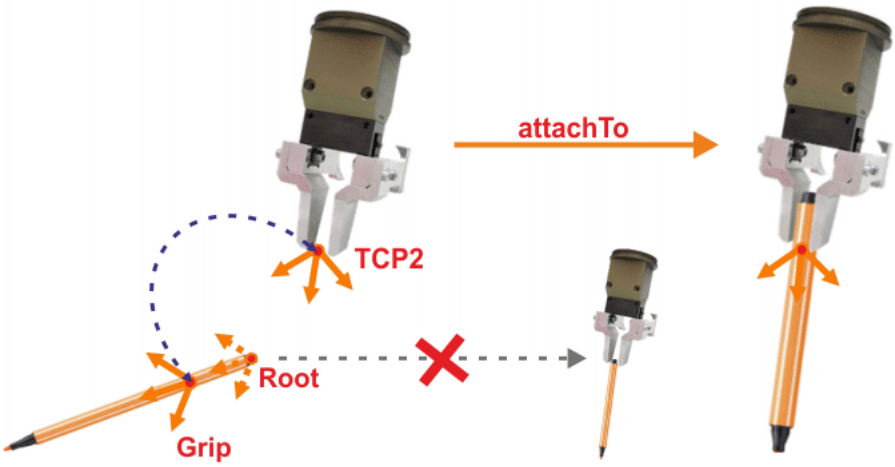


Fig. 15-13: Pen in gripper (connection via grip frame)

```
private LBR robot;
private Tool gripper;
private Workpiece pen;
...
public void run() {
  ...
  pen.getFrame("/Grip").attachTo(gripper.getFrame("/TCP2"));
  ...
}
```

15.11.3.3 Detaching objects

Description

If a tool is removed or a workpiece is set down, the object must also be detached in the application. The method detach() is used for this purpose.

Syntax

Object`.detach();`

Explanation of the syntax

Element	Description
Object	Name of the object variable

Example

The guidance tool is detached.

```
guidingTool.detach();
```

15.11.4 Moving tools and workpieces

Description

Every movable object in a station can be moved with move(…) and moveAsync(…). The reference point of the motion is dependent on the object type:

- If a robot is moved, the reference point is always the robot flange center point.

- If a tool or workpiece is moved, the reference point is by default the default motion frame which was defined for this object in the **Object templates** view.

  (>>> 9.3.5 "Defining a default motion frame" Page 135)

  In this case, the tool or workpiece is linked directly to the motion command via the variable name declared in the application.
- However, any other frame created for a tool or workpiece can also be programmed as a reference point of the motion.

  In this case, using the method getFrame(…), the path to the frame of the object used for the motion must be specified (on the basis of the origin frame of the object).

Syntax

To use the default frame of the object for the motion:

Object`.move(`*Motion*`);`

To use a different frame of the object for the motion:

Object`.getFrame("`*Moved frame*`").move(`*Motion*`));`

Explanation of the syntax

Element	Description
Object	Object of the station which is being moved This specifies the object variable name which was declared and initialized in the application.
Moved frame	Path to the frame of the object which is used for the motion
Motion	Motion which is being executed

Examples

The PTP motion to point P1 is executed with the default frame of the gripper.

```
gripper.attachTo(robot.getFlange());
gripper.move(ptp(getApplicationData().getFrame("/P1")));
```

The PTP motion to point P1 is executed with a different frame than the default frame of the gripper, here TCP1:

```
gripper.attachTo(robot.getFlange());
gripper.getFrame("/TCP1").move(ptp(getApplicationData().getFrame("/
P1")));
```

A pen is gripped. The next motion is a PTP motion to point P20. This point is executed with the default frame of the workpiece “pen”.

```
gripper.attachTo(robot.getFlange());
...
pen.attachTo(gripper.getFrame("/TCP1"));
pen.move(ptp(getApplicationData().getFrame("/P20"));
```

15.11.5 Defining user-specific object classes

Description

Tools, workpieces and fixed geometric objects created in the object templates are based on the classes Tool, Workpiece and GeometricalObject. Specific properties or functions that tools, workpieces and fixed geometric objects generally have are not considered by these basic classes. For a gripper, examples might include functions for opening and closing.

Such specific object properties and functions can be defined in separate object classes. The following steps are required in order to be able to use the user-defined object classes in the same way as the basic classes in applications:

Step	Description
1	Derive a new object class from a suitable basic class: ■ Basic class for tools: com.kuka.roboticsAPI.geometricModel.Tool ■ Basic class for workpieces: com.kuka.roboticsAPI.geometricModel.Workpiece ■ Basic class for fixed geometric objects: com.kuka.roboticsAPI.geometricModel.GeometricObject **Note:** The constructor of the created object class must have the following properties: ■ Visibility **public** ■ Transfer parameter of type String (name of the object template used in the application is transferred)
2	Define object properties and functions in the new object class.
3	In the object templates, assign the new object class to the desired objects. For this, enter the full name of the object class under **Template class** in the **Properties** view. **Note:** Object templates that use an object class derived from a basic class are initialized in an application such as this by means of the command getApplicationData().createFromTemplate(…).

Procedure

Derive a new object class from a basic class:

1. Select the desired Sunrise project in the **Package Explorer**.
2. Select the menu sequence **File** > **New** > **Class**. The **New Java class** window opens.
3. In the **Package** box, enter a name for the Java package in which the new class is to be created, e.g. **tools**.
4. Enter a name for the new class in the **Name** box.
5. To the right of the **Superclass:** box, click on **Browse...**. The **Superclass selection** window is opened.
6. Enter the name of the basic class, e.g. **Tool**, in the **Select type** box and confirm the selection with **OK**. The name of the basic class is now displayed in the **Superclass:** box.
7. Activate the **Constructors from superclass** check box.
8. Click on **Finish**. The Java package with the newly created class is inserted into the source folder of the Sunrise project and opened in the editor area.
9. The required arrays and methods can now be defined.

Example

Step 1:

For a gripper, the object class Gripper is created using the procedure described above. The class Gripper is derived from the basic class Tool and expands the basic class to include the functions for opening and closing the gripper.

```
1 package tools;
2 import com.kuka.roboticsAPI.geometricModel.Tool;
3 public class Gripper extends Tool {
4    public Gripper(String name) {
5       super(name);
6       // TODO Automatisch generierter Konstruktorstub
7    }
8    /**
```

```
 9     * Opens the gripper
10     */
11     public void openGripper(){
12        ...
13     }
14     /**
15     * Closes the gripper
16     */
17     public void closeGripper(){
18        ...
19     }
20
21     public Gripper(String name, LoadData load) {
22        super(name, load);
23        // TODO Automatisch generierter Konstruktorstub
24     }
25 }
```

Line	Description
1	Name of the Java package that contains the class Gripper
4 … 19	Standard constructor of the class Gripper for definition of the gripper functions
11 … 13	Method openGripper() for opening the gripper
17 … 19	Method closeGripper() for closing the gripper
21 … 24	Expanded constructor of the class Gripper (not required)

Step 2:

An object template with the name “ExampleGripper” is created for the gripper. The object class Gripper is assigned to the object template:

- Entry under **Template class** in the **Properties** view: tools.Gripper

  The name of the Java package that contains the class Gripper must be specified.

Step 3:

The object class Gripper and the corresponding functions can be used in the application.

```
 1 public class ExampleApplication extends RoboticsAPIApplication {
 2    private Controller kuka_Sunrise_Cabinet;
 3    private LBR lbr_iiwa;
 4    private Gripper gripper;
 5
 6    public void initialize() {
 7       ...
 8       gripper = createFromTemplate("ExampleGripper");
 9       gripper.attachTo(lbr_iiwa.getFlange());
10
11    }
12
13    public void run() {
14       ...
15       gripper.openGripper();
16       gripper.move(
17          lin(getApplicationData().getFrame("GripPos")));
18       gripper.closeGripper();
19       ...
20    }
```

Line	Description
4	A tool of type Gripper is declared. The tool has the gripper functions defined with the object class Gripper.
8, 9	The tool is initialized with the object template with the name “ExampleGripper” and connected to the robot flange.
15 … 18	The gripper functions defined for the class Gripper are used to program a gripping process (open gripper, move to grip position, close gripper).

15.11.6 Commanding load changes to the safety controller

Description

The safety controller requires the load data of a workpiece for calculation of the external torques. The safety controller can only process the load data of safety-oriented workpieces.

(>>> 9.3.8 "Safety-oriented workpieces" Page 140)

During a process, picking up and setting down different workpieces can result in load changes. During collision detection, the user must explicitly inform the safety controller via the method setSafetyWorkpiece(…) which safety-oriented workpiece is currently activated. For this purpose, this workpiece is transferred as an input parameter.

> Only a safety-oriented workpiece may be transferred to setSafetyWorkpiece(…) . If a non-safety-oriented workpiece is transferred, an exceptional error occurs.

The method setSafetyWorkpiece(…) belongs to the LBR class and can be used in robot applications and background tasks. A precondition for the transfer of a safety-oriented workpiece to the method is that an instance of the workpiece has been created from the object templates.

(>>> 15.11.2 "Initializing tools and workpieces" Page 291)

To deactivate an active safety-oriented workpiece and communicate to the safety controller that a safety-oriented workpiece is no longer gripped, the value `null` is communicated to the method setSafetyWorkpiece(…).

The load change is commanded for the safety controller with setSafetyWorkpiece(…). If the workpiece load data are not to be taken into consideration in the safety-oriented part of the robot controller, the load change must also be programmed with the corresponding commands.

(>>> 15.11.3.2 "Attaching a workpiece to other objects" Page 293)

Syntax

lbr`.setSafetyWorkpiece(`*Workpiece*`);`

Explanation of the syntax

Element	Description
lbr	Type: LBR Name of the robot for which the load change is programmed
Workpiece	Type: Workpiece Safety-oriented workpiece whose load data are to be transferred to the safety controller If no safety-oriented workpiece is to be taken into consideration any longer, `null` must be transferred.

Example

A safety-oriented tool and 2 safety-oriented tools are created in the object templates.

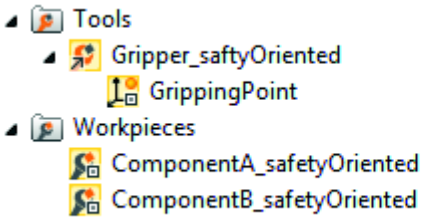


Fig. 15-14: Workpieces and tool (object templates)

The tool contains the frame "GrippingPoint", which serves as a gripping point for workpieces and which is selected as the standard frame for motions.

In the application, the workpiece "ComponentA_safetyOriented" is picked up and set down. The workpiece "ComponentB_safetyOriented" is then picked up. All 3 load changes are to be taken into consideration in both the safety-oriented and non-safety-oriented part of the robot controller.

```
public class ChangeOfLoadExample extends RoboticsAPIApplication {
...
// safety-oriented tool and workpieces
private Tool gripper;
private Workpiece componentA, componentB;

public void initialize() {
   ...
   // initialize safety-oriented components
   gripper = getApplicationData().
   createFromTemplate("Gripper_safetyOriented");
   componentA = getApplicationData().
   createFromTemplate("ComponentA_safetyOriented");
    componentB = getApplicationData().
    createFromTemplate("ComponentB_safetyOriented");

   // attach gripper to robot flange
   gripper.attachTo(lbr_iiwa.getFlange());
   }

public void run() {
   ...
   // after pick-up, attach workpiece to set load data for
   // motion control
   componentA.attachTo(gripper.getDefaultMotionFrame());
   // set load data for safety controller
   lbr_iiwa.setSafetyWorkpiece(componentA);
   ...
   // after putting it down, detach workpiece to no longer
   // consider its load for motion control
   gripper.detach();
   // workpiece is no longer considered for safety
   // controller
   lbr_iiwa.setSafetyWorkpiece(null);
   ...
   // pick-up of second workpiece
   componentB.attachTo(gripper.getDefaultMotionFrame());
   lbr_iiwa.setSafetyWorkpiece(componentB);
   ...
```

```
    }
...
}
```

15.12 Inputs/outputs

When exporting an I/O configuration from WorkVisual, a separate Java class is created for each I/O group in the corresponding Sunrise project. Each of these Java classes contains the methods required for programming, in order to be able to read the inputs/outputs of an I/O group and write to the outputs of an I/O group.

> The source code of the Java classes of the package **com.kuka.generated.ioAccess** must not be changed manually. To expand the functionality of an I/O group, it is possible to derive further classes from the classes created or to continue to use objects from these classes, e.g. as arrays of their own classes (aggregating).

To use the inputs/outputs of an I/O group in the application, the user must create and install a data array of the relevant type for the I/O group.

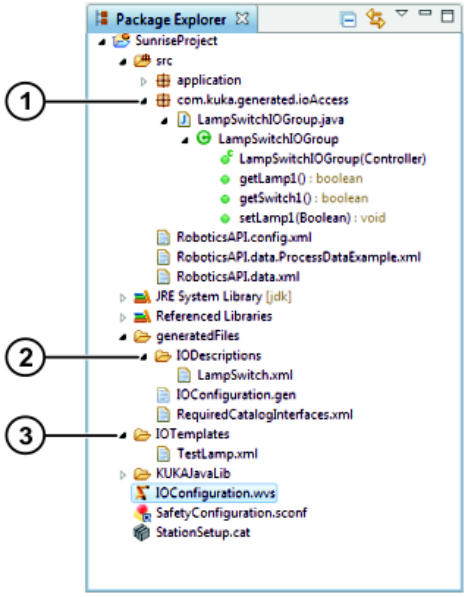


Fig. 15-15: Project structure after exporting the I/O configuration

Item	Description
1	**com.kuka.generated.ioAccess** Java package The class created for an I/O group and the corresponding methods are saved in the package. The Java class *Name*`IOGroup.java` (here: LampSwitchIOGroup.java) contains the following elements: ■ Class name of the I/O group: *Name*`IOGroup` ■ Constructor for assigning the robot controller to the I/O group: *Name*`IOGroup(`*Controller*`)` ■ "Get" and "set" methods for every configured output: `get`*Output*`()`, `set`*Output*`(`*Value*`)` ■ "Get" method for every configured input: `get`*Input*`()`
2	**generatedFiles** > **IODescriptions** folder The data in an I/O group are saved in an XML file. The XML file can be displayed but not edited.
3	**IOTemplates** folder The data of an I/O group saved as a template are saved in an XML file. The XML file can be displayed but not edited. A template can be copied into another Sunrise project in order to be used there. The template can then be imported into WorkVisual, edited there and re-exported. (>>> 11.5.8 "Importing an I/O group from a template" Page 159) (>>> 11.5.7 "Exporting an I/O group as a template" Page 158)

The **generatedFiles** folder is used by the system and must not be used for saving files created by the user.

15.12.1 Creating a data array for an I/O group

Description

Declaring the data array of the type of the I/O group automatically imports the Java package com.kuka.generated.ioAccess with the classes and methods of the I/O group.

Syntax

`private` *Class name Array name*`;`

Explanation of the syntax

Element	Description
`private`	The keyword `private` designates data arrays which can only be used by the surrounding class.
Class name	Type of the data array Class name of the I/O group: ■ *Name*`IOGroup` *Name* = Name of the I/O group, as defined in WorkVisual
Array name	Name of the data array used in programming

Example

For the I/O group "SwitchLamp", the data array "switchLamp" is created.

```
public class RobotApplication extends RoboticsAPIApplication {
  ...
  private Controller controller;
  private SwitchLampIOGroup switchLamp;
```

```
  ...
}
```

15.12.2 Initializing a data array for an I/O group

Description

When the data array is initialized, the robot controller to which the inputs/outputs of the group are connected via the field bus, is specified via the constructor of the class.

Syntax

Array name `=` `new` *Class name* `(`*Controller*`);`

Explanation of the syntax

Element	Description
Array name	Name of the data array
`new`	Operator with which a new instance of the class *Class name* is created
Class name	Class name of the I/O group: ■ *Name*`IOGroup` *Name* = Name of the I/O group, as defined in WorkVisual
Controller	Name of the object assigned to the robot controller The assignment generally takes place in the initialize() method of a robot application. (>>> 15.1.2 "Structure of a robot application" Page 265)

Example

The data array “switchLamp” is initialized.

```
public void initialize() {
  ...
  switchLamp = new SwitchLampIOGroup(controller);
  ...
}
```

15.12.3 Reading inputs/outputs

Description

The “get” method of an input/output is used to poll the state of the input/output.

Syntax

Array name`.get`*Input/output*`();`

Explanation of the syntax

Element	Description
Array name	Name of the data array
Input/output	Name of the input/output (as defined in WorkVisual)

Example

The state of the switch at input “Switch1” and of the lamp at output “Lamp1” is polled.

```
public void run() {
  ...
  switchLamp.getLamp1();
  switchLamp.getSwitch1();
  ...
}
```

15.12.4 Setting outputs

WARNING Outputs are switched in certain situations although a safety-oriented stop request is present (e.g. in the case of a pressed EMERGENCY STOP or violated space monitoring). This can cause unexpected motions of the connected periphery (e.g opening of a gripper).

The following are examples of cases which may occur:

- Background task switches output.
- Function called via user key switches output.
- Robot applications continue running to the next synchronous motion command after a stop request. The code executed up to that point switches the output.

The behavior described can also be desirable; however, there must never be any danger to human and machine. This must be ensured by the system integrator, e.g. by means of de-energizing outputs with hazard potential.

NOTICE It is not permissible to set in a robot application outputs that signal system states to the higher-level controller. Failure to observe this precaution may result in malfunctioning of the higher-level controller and damage to property.

Description

The “set” method of an output is used to change the value of the output.

No “set” methods are available for inputs. They can only be read.

Syntax

Array name`.setOutput(`*Value*`);`

Explanation of the syntax

Element	Description
Array name	Name of the data array
Output	Name of the output (as defined in WorkVisual)
Value	Value of the output. The data type of the value to be transferred depends on the output type.

Example

The lamp at output “Lamp1” is switched on and then switched off after 2000 ms.

```
public void run() {
  ...
  switchLamp.setLamp1(true);
  ThreadUtil.milliSleep(2000);
  switchLamp.setLamp1(false);
  ...
}
```

15.13 Polling axis torques

Description

A joint torque sensor which measures the torque acting on the axis is located in each axis of the KUKA LBR iiwa. The measured torque values can be polled and evaluated in the application via the method getMeasuredTorque() of the LBR class.

Frequently, it is not the pure measured values which are of interest but rather only the externally acting torques, without the component resulting from the weight of the robot structure and mass inertias during motion. These values

are referred to as external torques. These external torques be accessed via the LBR method getExternalTorque().

Both commands return an object of the type TorqueSensorData, which contains the torque sensor data for all axes. From this object, it is then possible to poll either all values as an array with getTorqueValues(…) or a single axis value with getSingleTorqueValue(…).

> When polling the torque sensor data with Java, no real-time behavior is available. This means that the data supplied by the system in the program were already created several milliseconds earlier.

Syntax

To poll the measured sensor data:

`TorqueSensorData` *measuredData* `=` *lbr*`.getMeasuredTorque();`

To poll externally acting torque data:

`TorqueSensorData` *externalData* `=` *lbr*`.getExternalTorque();`

To poll torque values of all axes from the sensor data:

`double[]` *allValues* `=` *measuredData*|*externalData*`.getTorqueValues();`

To poll torque values of a specific axis from the sensor data:

`double` *singleValue* `=` *measuredData*|*externalData*`.getSingleTorqueValues(`*joint*`);`

Explanation of the syntax

Element	Description
measured Data	Type: TorqueSensorData Variable for the return value of getMeasuredTorque(). The return value contains the measured sensor data.
externalData	Type: TorqueSensorData Variable for the return value of getExternalTorque(). The return value contains the externally acting torques.
lbr	Type: LBR Name of the robot from which the sensor data are polled
allValues	Type: double[]; unit: Nm Array with all torque values which are polled from the sensor data
singleValue	Type: double; unit: Nm Torque value of the axis which is polled from the sensor data
joint	Type: Enum of type JointEnum Axis whose torque value is to be polled ■ **JointEnum.J1 … JointEnum.J12**: Axes A1 … A12

Example

For a specific process step, the measured and externally acting torques are polled in all axes and saved in an array to be evaluated later. The measured torque in axis A2 is read and displayed on the smartHMI.

```
TorqueSensorData measuredData = lbr.getMeasuredTorque();
TorqueSensorData externalData = lbr.getExternalTorque();

double[] measuredTorques = measuredData.getTorqueValues();
double[] externalTorques = externalData.getTorqueValues();
```

```
double torqueA2 = measuredData.getSingleTorqueValue(JointEnum.J2);
getLogger().info("Currently measured torque for joint 2 [Nm]:" +
torqueA2);
```

15.14 Reading Cartesian forces and torques

It is possible to read the external Cartesian forces and torques currently acting on the robot flange, the TCP of a tool or any point of a gripped workpiece.

The following points must be taken into consideration:

- The torques of the axes are measured by the torque sensors.
- The Cartesian forces and torques are calculated from the measured torques.
- The reliability of the calculated values can decrease considerably in extreme poses, e.g. extended positions or singularities.
- There are methods available in the RoboticsAPI for checking the quality and validity of the calculated values.

When changing the load data, e.g. with the attachTo command, the poll can only be executed after the motion command has been sent to the controller. For this purpose, a null space motion or the motion command positionHold(…) is sufficient.

The Cartesian forces and torques are estimated based on the measured values of the joint torque sensors. A force application point must be specified for the calculation. The forces and torques determined for the specified point of application are only meaningful in terms of the physics involved if there are no external forces acting on any other points on the robot structure.

15.14.1 Polling calculated force/torque data

Description

The method getExternalForceTorque(…) of the LBR class is used to poll the external Cartesian forces and torques currently acting on a specific point.

The method receives a frame as the transfer parameter. The transferred frame is the reference frame for calculating the forces and torques, e.g. the tip of a probe. The method calculates the externally applied forces and torques for the position described by the frame.

For a meaningful calculation in terms of the physics involved, the transferred frame must describe a point which is mechanically fixed to the flange. The given frame must also be statically connected to the robot flange frame in the frame structure.

Optionally, a second frame can be transferred to the method as a parameter. This frame specifies the orientation of a coordinate system in which the forces and torques are represented.

Syntax

```
ForceSensorData data = lbr.getExternalForceTorque(
measureFrame<, orientationFrame>);
```

Explanation of the syntax

Element	Description
data	Type: ForceSensorData Variable for the return value of getExternalForce-Torque(…). The return value contains the polled force/torque data.
lbr	Type: LBR Name of the robot
measure Frame	Type: AbstractFrame Reference frame for which the currently acting forces and torques are calculated
orientation Frame	Type: AbstractFrame Optional: Orientation of the frame in which the forces and torques are represented.

Examples

Poll of the force/torque data on the robot flange:

```
ForceSensorData data =
robot.getExternalForceTorque(robot.getFlange());
```

Poll of the force/torque data on the robot flange, with reference to the orientation of the world coordinate system:

```
ForceSensorData data =
robot.getExternalForceTorque(robot.getFlange(),
World.Current.getRootFrame());
```

15.14.2 Polling individual force/torque values

Description

The force/torque data read with getExternalForceTorque() can be polled separately from each other via the methods getForce() and getTorque(…) of the ForceSensorData class.

The result of these pollings is a vector in each case. The values for each degree of freedom can be polled individually with the “get” methods of the Vector class.

(>>> 15.14.4 "Polling individual values of a vector" Page 308)

Syntax

To poll a force vector:

`Vector` *force* `=` *data*`.getForce();`

To poll a torque vector:

`Vector` *torque* `=` *data*`.getTorque();`

Explanation of the syntax

Element	Description
force	Type: vector (com.kuka.roboticsAPI.geometricModel.math) Vector with the Cartesian forces which act in the X, Y and Z directions (unit: N)
torque	Type: vector (com.kuka.roboticsAPI.geometricModel.math) Vector with the Cartesian torques which act about the X, Y and Z axes (unit: Nm)
data	Type: ForceSensorData Variable for the return value of getExternalForce-Torque(…). The return value contains the polled force/torque data.

Example

Poll of the Cartesian force which is currently acting on the robot flange in the X direction:

```
ForceSensorData data =
robot.getExternalForceTorque(robot.getFlange());

Vector force = data.getForce();
double forceInX = force.getX();
```

15.14.3 Checking the reliability of the calculated force/torque values

Description

In unfavorable positions, the calculated force/torque values can deviate from the actual forces and torques applied. In particular near singularities, several of the calculated values are highly unreliable and can be invalid. Depending on the axis position, this only applies to some of the calculated values.

The quality and validity of the calculated values can be evaluated and polled in the program. The following methods of the ForceSensorData class are available:

- getForceInaccuracy(), getTorqueInaccuracy()

  The inaccuracies of the calculated force values and the calculated torque values can be polled. The result of these pollings is a vector in each case. The values for each degree of freedom can be polled individually with the “get” methods of the Vector class.

  Depending on the axis position, the quality of the calculated values for the individual degrees of freedom may be different. By polling the individual values, it is possible to determine the degrees of freedom for which the calculation of forces and torques in the current pose supplies valid values.

  (>>> 15.14.4 "Polling individual values of a vector" Page 308)
- isForceValid(…), isTorqueValid(…)

  It is possible to poll whether the calculated force/torque values are valid. Each method is transferred a limit value for the maximum permissible inaccuracy, up to which point the calculated values are still valid as parameters.

Syntax

Polling the inaccuracy of the calculated force/torque values:

`Vector` *force* `=` *data*`.getForceInaccuracy();`

`Vector` *torque* `=` *data*`.getTorqueInaccuracy();`

Polling the validity of the force/torque values:

`boolean` *valid* `=` *data*`.isForceValid(`*tolerance*`);`

`boolean` *valid* `=`*data*`.isTorqueValid(`*tolerance*`);`

Explanation of the syntax

Element	Description
force	Type: vector (com.kuka.roboticsAPI.geometricModel.math) Vector with the values for the inaccuracy with which the Cartesian forces acting in the X, Y and Z directions are calculated (unit: N)
torque	Type: vector (com.kuka.roboticsAPI.geometricModel.math) Vector with the values for the inaccuracy with which the Cartesian torques acting about the X, Y and Z axes are calculated (unit: Nm)

Element	Description
data	Type: ForceSensorData Variable for the return value of the method getExternalForceTorque(…). The return value contains the polled force/torque data.
tolerance	Type: double; unit: N or Nm Limit value for the maximum permissible inaccuracy up to which the calculated force/torque values are still valid
valid	Type: Boolean Variable for the return value of isForceValid(…) or isTorqueValid(…) ■ **true**: The inaccuracy value in all Cartesian directions is less than or equal to the limit value defined with *tolerance*. ■ **false**: The inaccuracy value in one or more Cartesian directions exceeds the *tolerance* value

Example

A certain statement block should only be executed if the external Cartesian forces currently acting along the axes of the flange coordinate system have been calculated with an accuracy of 20 N or better.

```
ForceSensoData data =
robot.getExternalForceTorque(robot.getFlange());

if (data.isForceValid(20)){
   //do something
}
```

15.14.4 Polling individual values of a vector

Methods which poll data from a frame generally return an object of the Vector class (package: com.kuka.roboticsAPI.geometricModel.math). The components of the vector can be polled individually.

Overview

The following methods of the Vector class are available:

Method	Description
getX()	Return value type: double Polls for the X component of the vector
getY()	Return value type: double Polls for the Y component of the vector
getZ()	Return value type: double Polls for the Z component of the vector
get(*index*)	Return value type: double Polls for the components determined by the *index* parameter Values of *index* (type: int): ■ **0**: X component of the vector ■ **1**: Y component of the vector ■ **2**: Z component of the vector

15.15 Polling the robot position

The axis-specific and Cartesian robot position can be polled in the application. It is possible to poll the actual and the setpoint position for each.

Overview

The following methods of the Robot class are available:

Method	Description
getCommandedCartesianPosition(…)	Return value type: Frame Polls for the Cartesian setpoint position
getCommandedJointPosition()	Return value type: JointPosition Polls for the axis-specific setpoint position
getCurrentCartesianPosition(…)	Return value type: Frame Polls for the Cartesian actual position
getCurrentJointPosition()	Return value type: JointPosition Polls for the axis-specific actual position
getPositionInformation(…)	Return value type: PositionInformation Polls for the Cartesian position information The return value contains the following information: ■ Axis-specific actual position ■ Axis-specific setpoint position ■ Cartesian actual position ■ Cartesian setpoint position ■ Cartesian setpoint/actual value difference (rotational) ■ Cartesian setpoint/actual value difference (translational)

15.15.1 Polling the axis-specific actual or setpoint position

Description

For polling the axis-specific actual or setpoint position of the robot, the position of the robot axes is first saved in a variable of type JointPosition.

From this variable, the axis values can then be polled individually. The desired axis is specified either via its index or the corresponding JointEnum value.

Syntax

To poll the axis-specific actual position:

```
JointPosition position = robot.getCurrentJointPosition();
```

To poll the axis-specific setpoint position:

```
JointPosition position = robot.getCommandedJointPosition();
```

To poll the axis values individually:

```
double value = position.get(axis);
```

Explanation of the syntax

Element	Description
position	Type: JointPosition Variable for the return value. The return value contains the polled axis positions.
robot	Type: Robot Name of the robot from which the axis positions are polled
value	Type: double; unit: rad Axis angle of the polled axis

Element	Description
axis	Type: int 1st option: specify the index of the axis whose axis value is polled ■ **0 … 11**: axes A1 … A12
	Type: Enum of type JointEnum 2nd option: specify the JointEnum value of the axis whose axis value is polled ■ **JointEnum.J1 … JointEnum.J12**: Axes A1 … A12

Example

First the axis-specific actual position of the robot and then the axis value of axis A3 are polled via the index. The angle for axis A4 is displayed in degrees on the smartHMI.

```
JointPosition actPos = lbr.getCurrentJointPosition();
double a3 = actPos.get(2);
getLogger().info(Math.toDegrees(a3));
```

15.15.2 Polling the Cartesian actual or setpoint position

Description

It is possible to poll the Cartesian actual or setpoint position of the robot flange as well as every other frame below it. This means every frame of an object which is attached to the robot flange via the attachTo command, e.g. the TCP of a tool or the frame of a gripped workpiece.

The result of the polling, i.e. the Cartesian position, refers by default to the world coordinate system. Optionally, it is possible to specify another reference coordinate system relative to which the Cartesian position is polled. This can for example be a frame created in the application data or a calibrated base.

The result of the polling is saved in a variable of type Frame and contains all the necessary redundancy information (redundancy angle, Status and Turn). From this variable, the position (X, Y, Z) and orientation (A, B, C) of the frame can be polled via the type-specific get methods.

Syntax

To poll the Cartesian actual position:

`Frame` *position* `=` *robot*`.getCurrentCartesianPosition(` *frameOnFlange*`<,` *referenceFrame*`>);`

To poll the Cartesian setpoint position:

`Frame` *position* `=` *robot*`.getCommandedCartesianPosition(` *frameOnFlange*`<,` *referenceFrame*`>);`

Explanation of the syntax

Element	Description
position	Type: Frame Variable for the return value. The return value contains the polled Cartesian position.
robot	Type: Robot Name of the robot from which the Cartesian position is polled

Element	Description
frameOn Flange	Type: ObjectFrame Robot flange or a frame subordinated to the flange whose Cartesian position is polled
reference Frame	Type: AbstractFrame Reference coordinate system relative to which the Cartesian position is polled. If no reference coordinate system is specified, the Cartesian position refers to the world coordinate system.

Examples

Cartesian actual position of the robot flange with reference to the world coordinate system:

```
Frame cmdPos = lbr.getCurrentCartesianPosition(lbr.getFlange());
```

Cartesian actual position of the TCP of a tool with reference to a base:

```
tool.attachTo(lbr.getFlange());
...
Frame cmdPos = lbr.getCurrentCartesianPosition(tool.getFrame("/TCP"),
getApplicationData().getFrame("/Base"));
```

15.15.3 Polling the Cartesian setpoint/actual value difference

Description

The Cartesian setpoint/actual value difference (= difference between the programmed and measured position) can be polled with the getPositionInformation(…) method.

The result of the polling is saved in a variable of type PositionInformation. From this variable, the translational and rotational setpoint/actual value differences can be polled separately from each other.

Syntax

To poll position information:

`PositionInformation` *info* `=` *robot*`.getPositionInformation(` *frameOnFlange*`<,` *referenceFrame*`>);`

To poll the translational setpoint/actual value difference:

`Vector` *translatoryDiff* `=` *info*`.getTranslationOffset();`

To poll the rotational setpoint/actual value difference:

`Rotation` *rotatoryDiff* `=` *info*`.getRotationOffset();`

> The Cartesian actual/setpoint value position saved in the PositionInfomation object can be read with the methods getCurrentCartesianPosition(…) and getCommandedCartesianPosition(…) that have already been described.

Explanation of the syntax

Element	Description
info	Type: PositionInformation Variable for the return value. The return value contains the polled position information.
robot	Type: Robot Name of the robot from which the position information is polled

Element	Description
frameOn Flange	Type: ObjectFrame Robot flange or a frame subordinated to the flange whose position information is being polled
reference Frame	Type: AbstractFrame Reference coordinate system relative to which the position information is polled. If no reference coordinate system is specified, the position information refers to the world coordinate system.
translatoryDiff	Type: vector (com.kuka.roboticsAPI.geometricModel.math) Translational setpoint/actual value difference in the X, Y, Z directions (type: double, unit: mm) The offset values for each degree of freedom can be polled individually with the “get” methods of the Vector class. (>>> 15.14.4 "Polling individual values of a vector" Page 308)
rotatoryDiff	Type: rotation (com.kuka.roboticsAPI.geometric-Model.math) Setpoint/actual value difference of the axis angles A, B, C (type: double, unit: rad) The offset values for each degree of freedom can be polled individually with the “get” methods of the Rotation class - getAlphaRad(), getBetaRad, getGammaRad().

Example

Reading of the translational setpoint/actual value difference in the X direction and the setpoint/actual value difference of the axis angle C.

```
tool.attachTo(lbr.getFlange());
...
PositionInformation posInf =
lbr.getPositionInformation(tool.getFrame("/TCP"),
getApplicationData().getFrame("/Base"));

Vector transDiff = posInf.getTranslationOffset();
Rotation rotDiff = posInf.getRotationOffset();

double transOffsetInX = transDiff.getX();
double rotOffsetofC = rotDiff.getGammaRad();
```

15.16 HOME position

The HOME position is an application-specific position of the robot. It can be reset for an application during initialization.

The HOME position has the following values by default:

Axis	A1	A2	A3	A4	A5	A6	A7
Pos.	0°	0°	0°	0°	0°	0°	0°

15.16.1 Changing the HOME position

Description

The HOME position in an application can be changed with setHomePosition(…). The method belongs to the Robot class.

A HOME position must meet the following conditions:

- Good starting position for program execution
- Good standstill position. For example, the stationary robot must not be an obstacle.

The new HOME position can be transferred as an axis-specific or Cartesian position (frame). It is only applicable in the application in which it was changed. Other applications continue to use the HOME position with the default values.

Syntax

robot`.setHomePosition(`*home*`);`

Explanation of the syntax

Element	Description
robot	Type: Robot Name of the robot to which the new HOME position refers
home	Type: JointPosition; unit: rad 1st option: transfer the axis position of the robot in the new HOME position.
	Type: AbstractFrame 2nd option: transfer a frame as the new HOME position. **Note:** the frame must contain all redundancy information so that the axis positions of the robot in the HOME position are unambiguous. This is for example the case with a taught frame.

Examples

To transfer an axis-specific position as the HOME position:

```
private LBR lbr;
...
JointPosition newHome = new JointPosition(0.0, 0.0, 0.0,
Math.toRadians(90), 0.0, 0.0, 0.0);
lbr.setHomePosition(newHome);
```

To transfer the taught frame as the HOME position and move to it with ptpHome():

```
private LBR lbr;
...
ObjectFrame newHome = getApplicationData().getFrame("/Homepos");
lbr.setHomePosition(newHome);
lbr.moveAsync(ptpHome());
```

15.17 Polling system states

Different system states can be polled from the robot and processed in the application. The polling of system states is primarily required when using a higher-level controller so that the controller can react to changes in state.

15.17.1 Polling the HOME position

Description

The following methods of the Robot class are available for polling the HOME position:

- getHomePosition()
  Polls for the HOME position currently defined for the robot
- isInHome()
  Polls whether the robot is currently in the HOME position

Syntax

To poll the HOME position:

```
JointPosition homePos = robot.getHomePosition();
```

To check whether the robot is currently in the HOME position:

```
boolean result = robot.isInHome();
```

Explanation of the syntax

Element	Description
homePos	Type: JointPosition Variable for the return value of getHomePosition(). The return value contains axis angles of the polled HOME position.
robot	Type: Robot Name of the robot from which the HOME position is polled
result	Type: Boolean Variable for the return value of isInHome(). The return value is **true** when the robot is in the HOME position.

Example

As long as the robot is not yet in the HOME position, a certain statement block is to be executed.

```
private LBR lbr;
...
while(! lbr.isInHome()){
   //do something
}
```

15.17.2 Polling the mastering state

Description

The method isMastered() is available for polling the mastering state. The method belongs to the Robot class.

Syntax

```
boolean result = robot.isMastered();
```

Explanation of the syntax

Element	Description
robot	Type: Robot Name of the robot whose mastering state is polled
result	Type: Boolean Variable for the return value ■ **true**: All axes are mastered. ■ **false**: One or more axes are unmastered.

15.17.3 Polling “ready for motion”

Description

The method isReadyToMove() is available for polling whether the robot is ready for motion. The method belongs to the Robot class.

Syntax

```
boolean result = robot.isReadyToMove();
```

Explanation of the syntax

Element	Description
robot	Type: Robot Name of the robot which is polled whether it is ready for motion
result	Type: Boolean Variable for the return value ■ **true**: Robot is ready for motion. ■ **false**: A safety stop is activated or the robot drives are in the error state.

> If the return value is **true**, this does not necessarily mean that the brakes are open and that the robot is under servo control.

15.17.3.1 Reacting to changes in the “ready for motion” signal

Description

There is a notification service of the Controller class in RoboticsAPI which reports changes in the “ready for motion” signal. To register for the service, transfer an IControllerStateListener object to the Controller attribute in the robot application. The method addControllerListener(…) is used for this purpose.

The method onIsReadyToMoveChanged(…) is called every time the “ready to move” signal changes. The reaction to the change can be programmed in the body of the method onIsReadyToMoveChanged(…).

Syntax

```
kuka_Sunrise_Cabinet.addControllerListener(new
IControllerStateListener() {
...
@Override
public void onIsReadyToMoveChanged(Device device,
boolean isReadyToMove) {
// Reaction to change
}
...
});
```

Explanation of the syntax

Element	Description
kuka_Sunrise_Cabinet	Type: Controller Controller attribute of the robot application (= name of the robot controller in the application)

15.17.4 Polling the robot activity

Description

A robot is active if a motion command is active. This affects both motion commands from the application and jogging commands.

The method hasActiveMotionCommand() is available for polling whether the robot is active. The method belongs to the Robot class.

Syntax

```
boolean result = robot.hasActiveMotionCommand();
```

Explanation of the syntax

Element	Description
robot	Type: Robot Name of the robot whose activity is polled
result	Type: Boolean Variable for the return value ■ **true**: A motion command is active. ■ **false**: No motion command is active.

Polling does not provide any information as to whether the robot is currently in motion.
If the return value is **false**, this does not necessarily mean that the robot is stationary.
For example, robot activity may be polled directly after a synchronous motion command with a break condition. If the break condition occurs, the poll supplies the value **false** when the robot is braked and moved.
If the return value is **true**, this does not necessarily mean that the robot is in motion. For example, the poll supplies the value **true** if a position-controlled robot executes the motion command positionHold(…) and is stationary.

15.17.5 Polling and evaluating safety signals

Overview

In a robot application, the state of the following safety signals can be polled and evaluated.

- Operating mode
- Local E-STOP
- External E-STOP
- Operator safety
- Stop request (safety stop)

Precondition

To evaluate an E-STOP or operator safety:

- The appropriate category is selected in the corresponding row of PSM table in the safety configuration.
  - **Local E-STOP** category for a local E-STOP
  - **External E-STOP** category for an external E-STOP
  - **Operator safety** category for operator safety
- The configured reaction is a safety stop (no output).

15.17.5.1 Polling the state of the safety signals

Description

The current state of the different safety signals is first polled via the method getSafetyState() and grouped in an object of type ISafetyState.

From this object, the state of individual safety signals can then be polled via specific methods. The possible safety states are Enums of the respective return value type.

Syntax

```
ISafetyState currentState = robot.getSafetyState();
```

Explanation of the syntax

Element	Description
currentState	Type: ISafetyState Variable for the return value. The return value contains the state of the safety signals at the time of polling.
robot	Type: LBR Name of the robot from which the state of the safety signals is polled

Overview

The following methods are available for polling the safety signals of ISafetyState individually:

Method	Description
getOperationMode()	Return value type: OperationMode (com.kuka.roboticsAPI.deviceModel package) Poll of the currently set operating mode Enum values of the return value type: ■ **T1**, **T2**, **AUT** ■ **CRR**: CRR mode
getEmergencyStopInt()	Return value type: EmergencyStop Poll of whether a local E-STOP is activated Enum values of the return value type: ■ **ACTIVE**: Local E-STOP is activated. ■ **INACTIVE**: Local E-STOP is not activated. ■ **NOT_CONFIGURED**: Not relevant, as a local E-STOP is always configured.
getEmergencyStopEx()	Return value type: EmergencyStop Poll of whether an external E-STOP is activated Enum values of the return value type: ■ **ACTIVE**: External E-STOP is activated. ■ **INACTIVE**: External E-STOP is not activated. ■ **NOT_CONFIGURED**: No external EMERGENCY STOP is configured.
getOperatorSafetyState()	Return value type: OperatorSafety Poll of the operator safety signal Enum values of the return value type: ■ **OPERATOR_SAFETY_OPEN**: Operator safety is violated (e.g. safety gate is open). ■ **OPERATOR_SAFETY_CLOSED**: Operator safety is not violated. ■ **NOT_CONFIGURED**: No operator safety is configured.
getSafetyStopSignal()	Return value type: SafetyStopType Poll of whether a safety stop is activated Enum values of the return value type: ■ **NOSTOP**: No safety stop is activated. ■ **STOP0**: A safety stop 0 or a safety stop 1 is activated. ■ **STOP1**: A safety stop 1 (path-maintaining) is activated. ■ **STOP2**: This value is currently not returned.

Example

The system polls whether a safety stop is activated. If this is the case, the operator safety is then checked. If this is violated, a message is displayed on the smartHMI.

```
ISafetyState safetyState = robot.getSafetyState();

SafetyStopType safetyStop = safetyState.getSafetyStopSignal();

if(safetyStop != SafetyStopType.NOSTOP){
   OperatorSafety operatorSafety =
   safetyState.getOperatorSafetyState();
   if (operatorSafety == OperatorSafety.OPERATOR_SAFETY_OPEN)
   getLogger().warn("The safety gate is open!");
}
```

15.17.5.2 Reacting to a change in state of safety signals

Description

There is a notification service of the Controller class in the RoboticsAPI which reports changes in the state of safety signals. This service allows the user to directly react to the change in a signal state.

To register for the service, transfer an ISunriseControllerStateListener object to the Controller attribute in the robot application. The method addControllerListener(…) is used for this purpose.

The method onSafetyStateChanged(…) is called every time the state of a safety signal changes. The reaction to the change can be programmed in the body of the method onSafetyStateChanged(…).

Syntax

kuka_Sunrise_Cabinet`.addControllerListener(new ISunriseControllerStateListener() {`

...

`@Override`

`public void onSafetyStateChanged(Device device, SunriseSafetyState safetyState) {`

// Reaction to change in state

`}`

`});`

Explanation of the syntax

Element	Description
kuka_Sunrise_Cabinet	Type: Controller Controller attribute of the robot application (= name of the robot controller in the application)

Example

If the state of a safety signal changes, the operator safety is checked via the method onSafetyStateChanged()(…). If this is violated, a message is displayed on the smartHMI.

```
kuka_Sunrise_Cabinet.addControllerListener(new
ISunriseControllerStateListener() {
   ...
   @Override
   public void onSafetyStateChanged(Device device,
      SunriseSafetyState safetyState) {
   OperatorSafety operatorSafety =
   safetyState.getOperatorSafetyState();
      if (operatorSafety == OperatorSafety.OPERATOR_SAFETY_OPEN)
         getLogger().warn("The saftey gate is open!");
```

```
    }
});
```

15.18 Changing and polling the program run mode

Description

The program run mode can be changed and polled via the methods setExecutionMode(…) and getExecutionMode() of the SunriseExecutionService. The SunriseExecutionService itself is polled by the Controller.

Preparation

1. Variable of type SunriseExecutionService.
2. Poll the SunriseExecutionService via the method getExecutionService() and save in the variable.

Syntax

To change the program run mode:

service`.setExecutionMode(ExecutionMode.`*newMode*`);`

To poll the current program run mode:

currentMode `=` *service*`.getExecutionMode();`

Explanation of the syntax

Element	Description
service:	Type: SunriseExecutionService Variable for the return value (contains the SunriseExecutionService polled by the Controller)
newMode	Type: Enum of type ExecutionMode New program run mode ■ **ExecutionMode.Step**: Step mode (program sequence with a stop after each motion command) ■ **ExecutionMode.Continuous**: Standard mode (continuous program sequence without stops)
currentMode	Type: ExecutionMode Variable for the return value (contains the program run mode polled by the SunriseExecutionService)

Example

The SunriseExecutionService is polled by the Controller and saved in the variable “serv”.

```
private SunriseExecutionService serv;
...
public void initialize() {
  controller = getController("Controller_LBR5");
  robot = (LBR) getRobot(controller, "LBR5_7kg");
  serv = (SunriseExecutionService)controller.getExecutionService();
  ...
}
```

The system first switches to Step mode and then back to standard mode.

```
public void run() {
  ...
  serv.setExecutionMode(ExecutionMode.Step);
  ...
  serv.setExecutionMode(ExecutionMode.Continuous);
  ...
}
```

The current program run mode is polled.

```
public void run() {
  ...
  ExecutionMode currentMode;
  currentMode = serv.getExecutionMode();
  ...
}
```

15.19 Changing and polling the override

The interface IApplicationOverrideControl provides methods with which the current override can be polled or changed in the application. For this, the interface IApplicationControl must be accessed in the first step using the method getApplicationControl().

The following override types are distinguished:

- Manual override: Override which can be adjusted manually by the user via the smartPAD
  (>>> 6.14.3 "Setting the manual override" Page 88)
- Application override: Programmed override set by the application
- Effective override: Product of the manual and application override
  Effective override = manual override application override

Overview

Methods used for polling the current override:

Method	Description
getApplicationOverride()	Return value type: double Polls the application override
getManualOverride()	Return value type: double Polls the manual override
getEffectiveOverride()	Return value type: double Polls the effective override

Methods used for changing the override:

Method	Description
setApplicationOverride(…)	Sets the application override to the specified value (type: double) ■ **0 … 1**
clipApplicationOverride(…)	Reduces the application override to the specified value (type: double) ■ **0 … 1** If a value is specified that is higher than the value currently programmed for the application override, the statement clipApplicationOverride(…) is ignored.
clipManualOverride(…)	Reduces the manual override to the specified value (type: double) ■ **0 … 1** If a value is specified that is higher than the currently programmed manual override, the statement clipManualOverride(…) is ignored.

Example

```
getApplicationControl().setApplicationOverride(0.5);
```

```
...
double actualOverride =
getApplicationControl().getEffectiveOverride();
```

15.19.1 Reacting to an override change

Description

It is possible for an application to inform itself when an override changes. A listener of type IApplicationOverrideListener must be defined and registered for this purpose.

When changing an override, the method overrideChanged(…) is called. The reaction to the change can be programmed in the body of the method overrideChanged(…).

Syntax

Defining a listener:

```
IApplicationOverrideListener overrideListener =
new IApplicationOverrideListener(){
@Override
public void overrideChanged(double effectiveOverride,
double manualOverride, double applicationOverride) {
// Reaction to override change
};
};
```

Registering a listener:

```
getApplicationControl().
addOverrideListener(overrideListener);
```

Removing a listener:

```
getApplicationControl().
removeOverrideListener(overrideListener);
```

Explanation of the syntax

Element	Description
override Listener	Type: IApplicationOverrideListener Name of the listener

15.20 Conditions

Often, values are to be monitored in applications and if definable limits are exceeded or not reached, specific reactions are to be triggered. Possible sources for these values are, for example, the sensors of the robot or configured inputs. The progress of a motion can also be monitored. Possible reactions are the termination of a motion being executed or the execution of a handling routine.

15.20.1 Conditions in the RoboticsAPI

Description

A condition can have 2 states: It is met (state = TRUE) or or not met (state = FALSE). To define a condition, an expression is formulated. In this expression, data, such as measurements provided by the system, are compared with a permissible limit value. The result of the evaluation of the expression defines the state of the condition.

Since different system data can be used for formulating conditions, there are different kinds of conditions. Each condition type is made available as its own

class in the RoboticsAPI. They belong to the com.kuka.roboticsAPI.condition-Model package and implement the ICondition interface.

Overview

The following condition types are available:

Data type	Description
JointTorqueCondition	The axis torque condition is met if the torque measured in an axis lies outside of a defined range of values. (>>> 15.20.3 "Axis torque condition" Page 324)
ForceCondition	The force condition is met if the Cartesian force exerted on a frame below the robot flange (e.g. at the TCP) exceeds a defined magnitude. (>>> 15.20.4 "Force condition" Page 324)
ForceComponentCondition	The force component condition is met if the Cartesian force exerted along an axis of a frame below the robot flange (e.g. along an axis of the TCP) exceeds a defined range. (>>> 15.20.5 "Force component condition" Page 329)
CartesianTorqueCondition	The conditions for the Cartesian torque is met if the Cartesian torque acting about the axis of a frame below the robot flange (e.g. about the axis of the TCP) exceeds a defined value. (>>> 15.20.6 "Condition for Cartesian torque" Page 331)
TorqueComponentCondition	The torque component condition is met if the Cartesian torque exerted about an axis of a frame below the robot flange (e.g. about an axis of the TCP) is outside a defined range. (>>> 15.20.7 "Torque component condition" Page 335)
MotionPathCondition	The path-related condition is met if a defined distance on the planned path, from the start or end point of the motion, is reached. In addition, it is possible to define a time delay which must be met. (>>> 15.20.8 "Path-related condition" Page 336)
BooleanIOCondition	The condition for Boolean signals is met if a Boolean digital input has a specific state. (>>> 15.20.9 "Condition for Boolean signals" Page 338)
IORangeCondition	The condition for the value range of a signal is met if the value of an analog or digital input lies within a defined range. (>>> 15.20.10 "Condition for the range of values of a signal" Page 339)

Areas of application

- Abortion of motions

  A motion is terminated as soon as a specific event occurs. The event occurs if the condition already has the state TRUE before the start of the motion or if it switches to the state TRUE during the motion.

  (>>> 15.21 "Break conditions for motion commands" Page 339)
- Path-related switching actions (Trigger)

  An action is triggered as soon as a specific event occurs. The event occurs if the condition already has the state TRUE before the start of the motion or if it switches to the state TRUE during the motion.

  (>>> 15.22 "Path-related switching actions (Trigger)" Page 343)
- Monitoring of processes (Monitoring)

  The state of a condition is checked cyclically using a listener. If the state of the condition changes, it is possible to react.

  (>>> 15.23 "Monitoring processes (Monitoring)" Page 347)

- Blocking wait for condition

  An application is stopped until a certain condition is met or a certain wait time has expired.

  (>>> 15.24 "Blocking wait for condition" Page 351)

15.20.2 Complex conditions

Conditions can be logically linked to one another so that it is possible to define complex conditions. The logic operators required for this are available as ICondition methods. The calling ICondition object is linked to one or more conditions, which are transferred as parameters.

The operators can be called several times in a row and in this way, parentheses and nesting of operations can be realized. The evaluation is thus dependent on the order of calling.

Operators

Operator	Description/syntax
NOT	Inversion of the calling ICondition object `ICondition invert();`
XOR	EITHER/OR operation linking the calling ICondition object with a further condition `ICondition xor(ICondition` *other*`);` *other*: further condition
AND	AND operation linking the calling ICondition object with one or more additional conditions `ICondition and(ICondition` *other1*`, ICondition` *other2*`, …);` *other1*, *other2*, …: further conditions
OR	OR operation linking the calling ICondition object with one or more additional conditions `ICondition or(ICondition` *other1*`, ICondition` *other2*`, …);` *other1*, *other2*, …: further conditions

Example

```
JointTorqueCondition condA = …;
JointTorqueCondition condB = …;
JointTorqueCondition condC = …;
JointTorqueCondition condD = …;

ICondition combi1, combi2, combi3, combi4;

// NOT A
combi1 = condA.invert();

// A AND B AND C
combi2 = condA.and(condB, condC);

// (A OR B) AND C
combi3 = condA.or(condB).and(condC);

// (A OR B) AND (C OR D)
combi4 = condA.or(condB).and(condC.or(condD));
```

15.20.3 Axis torque condition

Description

The axis torque condition is used to check whether the external torque determined in an axis lies outside of a defined range of values.

(>>> 15.13 "Polling axis torques" Page 303)

> The load data must be specified correctly during programming. Only then can the axis torque condition be used meaningfully.

Constructor syntax

`JointTorqueCondition(JointEnum` *joint*`, double` *minTorque*`, double` *maxTorque*`)`

Element	Description
joint	Axis whose torque value is to be checked ■ **JointEnum.J1 … JointEnum.J12**: Axes A1 … A12
minTorque	Lower limit value for the axis torque (unit: Nm) The condition is met if the torque is less than or equal to *minTorque*.
maxTorque	Upper limit value for the axis torque (unit: Nm) The condition is met if the torque is greater than or equal to *minTorque*.

The following must apply when determining the upper and lower limit values for the torque: *minTorque* ≤ *maxTorque*.

Example

```
JointTorqueCondition torqueCondJ3 =
new JointTorqueCondition(JointEnum.J3, -2.5, 4.0);
```

The condition is met if a torque value of ≤ -2.5 Nm or ≥ 4.0 Nm is measured in axis A3.

15.20.4 Force condition

Description

The force condition is used to check whether a Cartesian force exerted on a frame below the robot flange exceeds a defined limit value.

For example, it is possible to react to the force generated when the robot presses on a surface using a tool mounted on the flange. For the force condition, the projections of the force vector exerted on a frame below the flange are considered. The position of this frame is defined by the point of application of the force (here the tool tip). The orientation of the frame should correspond to the orientation of the surface.

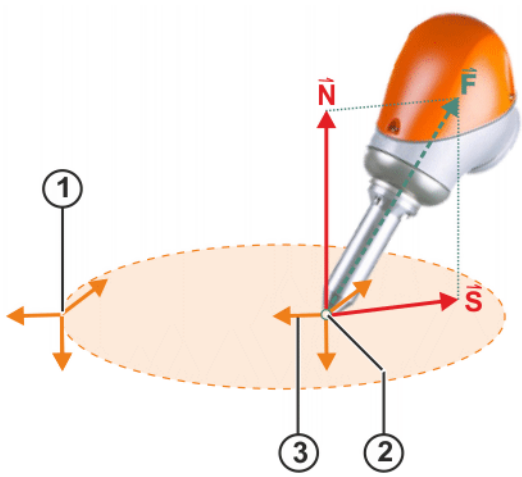


Fig. 15-16: Force vectors

1 Frame which specifies the orientation of the reference frame (here: orientation of the surface)
2 Point of application of the force, here the tip of the tool
3 Reference frame below the flange onto which the force vector is projected. The position of the frame corresponds to the point of application of the force. The orientation corresponds to the orientation of the surface.

The following force vectors are relevant:

- Normal force N:

  Projection of the force exerted on the surface normal (= vector which is perpendicular to the surface). This results in the part of the force exerted vertically on the surface. For example, pressure is exerted via the normal force in order to fit a component.
- Shear force S:

  Projection of the force exerted on the surface. This results in the part of the force exerted parallel to the surface. The shear force is generated by friction.

> The load data must be specified correctly during programming. Only then can the force condition be used meaningfully.

> The force estimation cannot return meaningful values near singularity positions. It is advisable not to use the force condition for this type of axis configuration. Alternatively, the axis torque condition can be used or the axis position can be adapted using the redundancy so as to ensure that no singular poses occur.

Methods

Force conditions are of the data type ForceCondition. ForceCondition contains the following static methods for programming conditions:

- createSpatialForceCondition(…): Condition for Cartesian force from all directions
- createNormalForceCondition(…): Condition for normal force
- createShearForceCondition(…): Condition for shear force

To formulate the condition, a frame below the flange coordinate system (e.g. the tip of a tool) is defined as a reference system. The forces which are exerted relative to this frame are determined. The orientation of the reference system can be optionally defined via an orientation frame. This can be used, for example, to define the position of the surface on which the force is exerted.

A limit value is defined to determine the minimum force magnitude which meets the condition.

The Cartesian force is calculated from the values of the joint torque sensors. The reliability of the calculated force values varies depending on the axis configuration. If the quality of the force calculation is also to be taken into account, it is possible to specify a value for the maximum permissible inaccuracy. If the system calculates an inaccuracy exceeding this value, the force condition is also met.

15.20.4.1 Condition for Cartesian force from all directions

Description

The static method createSpatialForceCondition(…) is used to define a condition which is valid regardless of the direction from which the Cartesian force is exerted on a frame below the flange.

Syntax

`ForceCondition.createSpatialForceCondition(`
`AbstractFrame` *measureFrame*`<,` `AbstractFrame` *orientationFrame*`>,`
`double` *threshold*`<,` `double` *tolerance*`>)`

Element	Description
measure Frame	Frame below the robot flange relative to which the exerted force is determined. The position of the point of application of the force is defined using this parameter.
orientation Frame	Optional. The orientation of the reference system is defined using this parameter. If the *orientationFrame* parameter is not specified, *measureFrame* defines the orientation of the reference system.
threshold	Maximum magnitude of force which may act on the reference system (unit: N). ■ **≥ 0.0** The condition is met if the magnitude of force exerted on the reference system from any direction exceeds the value specified here.
tolerance	Optional. Maximum permissible inaccuracy of the calculated values (unit: N). ■ **> 0.0** Default: 10.0 The condition is met if the inaccuracy of the force calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

Example

The condition is met as soon as the magnitude of the force acting from any direction on the TCP of a tool exceeds 30 N.

```
Tool gripper = ...;
gripper.attachTo(lbr.getFlange());

ForceCondition spatialForce_tcp =
ForceCondition.createSpatialForceCondition(
    gripper.getFrame("/TCP"),
    30.0

);
```

15.20.4.2 Condition for normal force

Description

A condition for the normal force can be defined via the static method createNormalForceCondition(…). The component of the force exerted along a definable axis of a frame below the flange (e.g. along an axis of the TCP) is considered here. This axis is generally defined so that it is perpendicular to the surface on which the force is exerted (surface normal).

Syntax

`ForceCondition.createNormalForceCondition(AbstractFrame` *measureFrame*`<, AbstractFrame` *orientationFrame*`>, CoordinateAxis` *direction*`, double` *threshold*`<, double` *tolerance*`>)`

Element	Description
measure Frame	Frame below the robot flange relative to which the exerted force is determined. The position of the point of application of the force is defined using this parameter.
orientation Frame	Optional. The orientation of the reference system is defined using this parameter. If the *orientationFrame* parameter is not specified, *measureFrame* defines the orientation of the reference system.
direction	Coordinate axis of the reference system. The force component acting on the axis specified here is checked with the condition. ■ **CoordinateAxis.X** ■ **CoordinateAxis.Y** ■ **CoordinateAxis.Z**
threshold	Maximum magnitude of force which may act along the axis of the reference system (unit: N). ■ **≥ 0.0** The condition is met if the magnitude of force exceeds the value specified here.
tolerance	Optional. Maximum permissible inaccuracy of the calculated values (unit: N). ■ **> 0.0** Default: 10.0 The condition is met if the inaccuracy of the force calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

Example

A gripper mounted on the flange presses on a table plate. The robot is to react to that part of the force exerted at the TCP of the gripper which acts vertically on the table plate. The reference system is therefore defined such that its Z axis runs along the surface normal of the table plate.

The condition is met as soon as the normal force exceeds a magnitude of 45 N. The condition is also to be considered met if the inaccuracy value of the calculated data exceeds 8.

```
Tool gripper = ...;
gripper.attachTo(lbr.getFlange());

SpatialObject table = ...;
```

```
ForceCondition normalForce_z =
ForceCondition.createNormalForceCondition(
   gripper.getFrame("/TCP"),
   table.getFrame("/Tabletop"),
   CoordinateAxis.Z,
   45.0,
   8.0

);
```

15.20.4.3 Condition for shear force

Description

A condition for the shear force can be defined via the static method createShearForceCondition(…). The component of the force acting parallel to a plane is considered here. The position of the plane is determined by specifying the axis which is vertical to the plane.

Syntax

`ForceCondition.createShearForceCondition(AbstractFrame` *measureFrame*`<, AbstractFrame` *orientationFrame*`>, CoordinateAxis` *normalDirection*`, double` *threshold*`<, double` *tolerance*`>)`

Element	Description
measure Frame	Frame below the robot flange relative to which the exerted force is determined. The position of the point of application of the force is defined using this parameter.
orientation Frame	Optional. The orientation of the reference system is defined using this parameter. If the *orientationFrame* parameter is not specified, *measureFrame* defines the orientation of the reference system.
normal Direction	Coordinate axis of the reference system. The axis specified here defines the surface normal of a plane. The force component acting parallel to this plane is checked. ■ **CoordinateAxis.X** ■ **CoordinateAxis.Y** ■ **CoordinateAxis.Z**
threshold	Maximum magnitude of force which may be exerted parallel to the reference system plane defined by its surface normal (unit: N). ■ **≥ 0.0** The condition is met if the magnitude of force exceeds the value specified here.
tolerance	Optional. Maximum permissible inaccuracy of the calculated values (unit: N). ■ **> 0.0** Default: 10.0 The condition is met if the inaccuracy of the force calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

Example

A gripper mounted on the flange presses on a table plate. The force at the TCP of the gripper is to be determined using the orientation of the table plate. This

process considers the shear force which acts parallel to the XY plane of the measurement point, defined by the TCP and the position of the table.

To define the XY plane, the axis perpendicular to this plane must be specified as a parameter. This is the Z axis.

The condition is met as soon as the shear force exceeds a magnitude of 25 N. The condition is also to be considered met if the inaccuracy value of the calculated data exceeds 5.

```
Tool gripper = ...;
gripper.attachTo(lbr.getFlange());

SpatialObject table = ...;

ForceCondition shearForce_xyPlane =
ForceCondition.createShearForceCondition(
   gripper.getFrame("/TCP"),
   table.getFrame("/Tabletop"),
   CoordinateAxis.Z,
   25.0,
   5.0

);
```

15.20.5 Force component condition

Description

The force component condition can be used to check whether the Cartesian force exerted on a frame below the robot flange (e.g. at the TCP) in the X, Y or Z direction exceeds a defined range.

> The load data must be specified correctly during programming. Only then can the force component condition be used meaningfully.

> The force estimation cannot return meaningful values near singularity positions. It is advisable not to use the force component condition for this type of axis configuration. Alternatively, the axis torque condition can be used or the axis position can be adapted using the redundancy so as to ensure that there is no singularity.

The force component condition belongs to the ForceComponentCondition class. For the force component condition, a frame below the flange coordinate system is defined as a reference system. The force is determined at this frame, e.g. at the tip of a tool. The orientation of the reference system can be optionally defined via an orientation frame.

The direction from which the force is checked is defined with one of the coordinate axes of the reference system. The force component condition is met if the Cartesian force along the defined coordinate axis of the reference system lies outside of a definable range of values.

The Cartesian force is calculated from the values of the joint torque sensors. The reliability of the calculated force values varies depending on the axis configuration. If the quality of the force calculation is also to be taken into account, it is possible to specify a value for the maximum permissible inaccuracy. If the system calculates an inaccuracy exceeding this value, the force component condition is also met.

Constructor syntax

The ForceComponentCondition class has several constructors which differ in their number of input parameters:

`ForceComponentCondition(AbstractFrame` *measureFrame* `<, AbstractFrame` *orientationFrame*`>, CoordinateAxis` *coordinateAxis*`, double` *min*`, double` *max*`<, double` *tolerance*`>)`

Element	Description
measure Frame	Frame below the robot flange relative to which the exerted force is determined. The position of the point of application of the force is defined using this parameter.
orientation Frame	Optional. The orientation of the reference system is defined using this parameter. If the *orientationFrame* parameter is not specified, *measureFrame* defines the orientation of the reference system.
coordinate Axis	Coordinate axis of the frame relative to which the exerted force is determined. Defines the direction from which the acting force is checked. ■ **CoordinateAxis.X** ■ **CoordinateAxis.Y** ■ **CoordinateAxis.Z**
min	Lower limit of the range of values for the force exerted along the coordinate axis of the reference system (unit: N). The force component condition is met if the force falls below the value specified here.
max	Upper limit of the range of values for the force exerted along the coordinate axis of the reference system (unit: N). The force component condition is met if the force exceeds the value specified here. **Note**: The upper limit value must be greater than the lower limit value: *max* > *min*.
tolerance	Optional. Maximum permissible inaccuracy of the calculated values. ■ **> 0.0** Default: 10.0 The force component condition is met if the inaccuracy of the force calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

Example

A joining process is ideally executed with a force of between 20 N and 25 N. A force component condition is to be defined, and is met if the force acting in the Z direction at the free end of a gripped workpiece is between 20 N and 25 N.

To this end, a force component condition is first defined which has the status FALSE in this range of values. The desired result is then realized by inversion.

```
Workpiece peg = ...;
peg.attachTo(...);

ForceComponentCondition assemblyForce_inverted = new
ForceComponentCondition(
   peg.getFrame("/Assembly"),
   CoordinateAxis.Z,
   20.0,
```

```
    25.0
);

ForceComponentCondition assemblyForce = (ForceComponentCondition)
assemblyForce_inverted.invert();
```

15.20.6 Condition for Cartesian torque

Description

The condition is used to check whether a Cartesian torque exerted on a frame below the robot flange exceeds a defined limit value. The point of application of the torque is specified by means of a frame below the robot flange coordinate system.

One application for this condition is the monitoring of torques that occur in a screw fastening process.

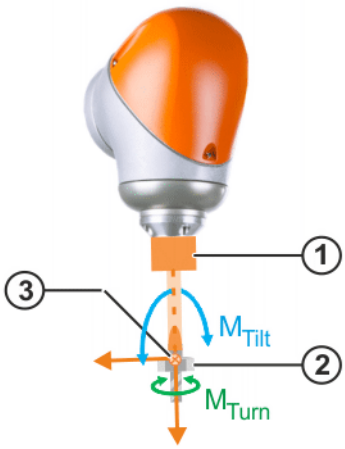


Fig. 15-17: Torque vectors in the screw fastening process

1 Power wrench
2 Screw
3 Reference frame, here the tip of the power wrench

The condition for the Cartesian torque can be used to check different projections of the torque vector acting on the axes of the reference frame:

- Torque M_{Turn}

  The torque exerted about an axis arises from the projection of the torque vector on this axis.
- Tilting torque M_{Tilt}

  The tilting torque arises from the projection of the torque vector on a plane.

The torque is applied about the longitudinal axis of the power wrench during a screw fastening process in order to screw in the screw. If the condition for the torque is used, it is possible to ensure that the maximum permissible values are not exceeded when fastening screws.

The tilting torque arises during a screw fastening process as a result of undesired tilting of the power wrench about the longitudinal axis, forwards or to the side. If the condition for the tilting torque is configured, it is possible to check whether the tilting torque is within an acceptable range of values.

> The load data must be specified correctly during programming. Only then can the condition for the Cartesian torque be used meaningfully.

> The force estimation cannot return meaningful values near singularity positions. It is advisable not to use the Cartesian torque condition for this type of axis configuration. Alternatively, the axis torque condition can be used or the axis position can be adapted using the redundancy so as to ensure that there is no singularity.

Methods

Conditions for the Cartesian torque are of data type CartesianTorqueCondition. CartesianTorqueCondition contains the following static methods for programming conditions:

- createSpatialTorqueCondition(…): Condition for Cartesian torque from all directions
- createTurningTorqueCondition(…): Condition for torque
- createTiltingTorqueCondition(…): Condition for tilting torque

To formulate the condition, a frame is defined as a reference system below the flange coordinate system. The torque is determined at this frame, e.g. at the tip of a power wrench. The orientation of the reference system can be optionally defined via an orientation frame. In this way, the desired orientation of the screw can be specified, for example.

A limit value is defined to determine the minimum Cartesian torque magnitude which meets the condition.

The Cartesian torque is calculated from the values of the joint torque sensors. The reliability of the calculated Cartesian torques varies depending on the axis configuration. If the quality of the calculation is also to be taken into account, it is possible to specify a value for the maximum permissible inaccuracy. If the system calculates an inaccuracy exceeding this value, the condition for the Cartesian torque is also met.

15.20.6.1 Condition for Cartesian torque from all directions

Description

The static method createSpatialTorqueCondition(…) is used to define a condition which is valid regardless of the direction from which the Cartesian torque is exerted on a frame below the flange.

Syntax

`CartesianTorqueCondition.createSpatialTorqueCondition(AbstractFrame` *measureFrame*`<, AbstractFrame` *orientationFrame*`>, double` *threshold*`<, double` *tolerance*`>)`

Element	Description
measure Frame	Frame below the robot flange at which the exerted torque is determined. The position of the point of application of the torque is defined using this parameter.
orientation Frame	Optional. The orientation of the reference system is defined using this parameter. If the *orientationFrame* parameter is not specified, *measureFrame* defines the orientation of the reference system.

Element	Description
threshold	Maximum magnitude of the torque which may act on the reference system (unit: Nm). ■ **≥ 0.0** The condition is met if the magnitude of the torque exerted on the reference system from any direction exceeds the value specified here.
tolerance	Optional. Maximum permissible inaccuracy of the calculated values (unit: Nm). ■ **> 0.0** Default: 10.0 The condition is met if the inaccuracy of the torque calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

15.20.6.2 Condition for torque

Description

A condition for the torque can be defined via the static method createTurningTorqueCondition(…). The component of the overall torque applied about a definable axis of a frame below the flange (e.g. about an axis of the TCP) is considered here.

Syntax

`CartesianTorqueCondition.createTurningTorqueCondition(AbstractFrame` *measureFrame*`<, AbstractFrame` *orientationFrame*`>, CoordinateAxis` *direction*`, double` *threshold*`<, double` *tolerance*`>)`

Element	Description
measure Frame	Frame below the robot flange at which the exerted torque is determined. The position of the point of application of the torque is defined using this parameter.
orientation Frame	Optional. This parameter defines the orientation of the frame relative to which the torque is determined. If the *orientationFrame* parameter is not specified, *measureFrame* defines the orientation of the reference system.
direction	Coordinate axis of the reference system. The component of the overall torque acting on the axis specified here of the reference system is checked using this condition. ■ **CoordinateAxis.X** ■ **CoordinateAxis.Y** ■ **CoordinateAxis.Z**

Element	Description
threshold	Maximum magnitude of the torque that may be applied to the axis of the reference system (unit: Nm). ■ **≥ 0.0** The condition is met if the magnitude of the torque exceeds the value specified here.
tolerance	Optional. Maximum permissible inaccuracy of the calculated values (unit: Nm). ■ **> 0.0** Default: 10.0 The condition is met if the inaccuracy of the torque calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

15.20.6.3 Condition for tilting torque

Description

A condition for the tilting torque can be defined via the static method createTiltingTorqueCondition(…). The component of the overall torque applied to a plane of the reference system is considered here. The position of the plane is determined by specifying the axis which is vertical to the plane (surface normal).

Syntax

`CartesianTorqueCondition.createTiltingTorqueCondition(AbstractFrame` *measureFrame*`<, AbstractFrame` *orientationFrame*`>, CoordinateAxis` *normalDirection*`, double` *threshold*`<, double` *tolerance*`>)`

Element	Description
measure Frame	Frame below the robot flange at which the exerted torque is determined. The position of the point of application of the torque is defined using this parameter.
orientation Frame	Optional. This parameter defines the orientation of the frame relative to which the torque is determined. If the *orientationFrame* parameter is not specified, *measureFrame* defines the orientation of the reference system.
normal Direction	Coordinate axis of the reference system. The axis specified here defines the surface normal of a plane. The component of the overall torque applied to the plane is checked. ■ **CoordinateAxis.X** ■ **CoordinateAxis.Y** ■ **CoordinateAxis.Z**

Element	Description
threshold	Maximum magnitude of the tilting torque that may be applied to the plane of the reference system defined by its surface normal (unit: Nm). ■ **≥ 0.0** The condition is met if the magnitude of the torque exceeds the value specified here.
tolerance	Optional. Maximum permissible inaccuracy of the calculated values (unit: Nm). ■ **> 0.0** Default: 10.0 The condition is met if the inaccuracy of the torque calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

15.20.7 Torque component condition

Description

The torque component condition can be used to check whether the Cartesian torque exerted about the X, Y or Z axis of a frame below the robot flange (e.g. about an axis of the TCP) is outside a defined range. It is used for monitoring the Cartesian torque in a specific direction, e.g. for monitoring screw fastening processes.

> The load data must be specified correctly during programming. Only then can the torque component condition be used meaningfully.

> The force estimation cannot return meaningful values near singularity positions. It is advisable not to use the torque component condition for this type of axis configuration. Alternatively, the axis torque condition can be used or the axis position can be adapted using the redundancy so as to ensure that there is no singularity.

The torque component condition is represented by the class TorqueComponentCondition. For the torque component condition, a frame below the flange coordinate system is defined as a reference system. The torque is determined at this frame, e.g. at the tip of a power wrench. The orientation of the reference system can be optionally defined via an orientation frame.

The direction in which the torque is checked is defined with one of the coordinate axes of the reference system. The torque component condition is met if the Cartesian torque about the defined coordinate axis of the reference system lies outside a definable range of values.

The Cartesian torque is calculated from the values of the joint torque sensors. The reliability of the calculated Cartesian torques varies depending on the axis configuration. If the quality of the calculation is also to be taken into account, it is possible to specify a value for the maximum permissible inaccuracy. If the system calculates an inaccuracy exceeding this value, the torque component condition is also met.

Constructor syntax

The TorqueComponentCondition class has several constructors which differ in their number of input parameters:

`TorqueComponentCondition(AbstractFrame` *measureFrame* `<, AbstractFrame` *orientationFrame*`>, CoordinateAxis` *component*`, double` *min*`, double` *max*`<, double` *tolerance*`>)`

Element	Description
measure Frame	Frame below the robot flange relative to which the exerted torque is determined. The position of the point of application of the torque is defined using this parameter.
orientation Frame	Optional. This parameter defines the orientation of the frame relative to which the torque is determined. If the *orientationFrame* parameter is not specified, *measureFrame* defines the orientation of the reference system.
coordinateAxis	Coordinate axis of the frame relative to which the exerted torque is determined. Defines the direction in which the acting torque is checked. ■ **CoordinateAxis.X** ■ **CoordinateAxis.Y** ■ **CoordinateAxis.Z**
min	Lower limit of the range of values for the torque exerted about the coordinate axis of the reference system (unit: Nm). The torque component condition is met if the torque falls below the value specified here.
max	Upper limit of the range of values for the torque exerted about the coordinate axis of the reference system (unit: Nm). The torque component condition is met if the torque exceeds the value specified here. **Note**: The upper limit value must be greater than the lower limit value: *max* > *min*.
tolerance	Optional. Maximum permissible inaccuracy of the calculated values (unit: Nm). ■ **> 0.0** Default: 10.0 The condition is met if the inaccuracy of the torque calculation is greater than or equal to the value specified here. If the parameter is not specified, the default value is automatically used.

15.20.8 Path-related condition

Description

Path-related conditions are always used in conjunction with a motion command. They serve as break conditions or triggers for path-related switching actions.

The condition defines a point on the planned path (switching point) on which a motion is to be terminated or a desired action is to be triggered. If the switching point is reached, the condition is met.

> The braking process or the defined action is only triggered when the switching point is reached. When using a path-related condition as a break condition, this results in the robot coming to a standstill after the switching point rather than directly at it.

The switching point can be defined by a shift in space and/or time. The shift can optionally refer to the start or end point of a motion.

> If a time offset is defined, a change to the override influences the switching point. The action linked to a path-related condition is therefore only triggered with an effective override of 100% and at the defined switching point in T2 or Automatic modes.

Path-related conditions are of data type MotionPathCondition.

Constructor syntax

The MotionPathCondition class has the following constructor:

`MotionPathCondition(ReferenceType` *reference*`, double` *distance*`, long` *delay*`)`

Static methods

A MotionPathCondition object can also be created via one of the following static methods:

`MotionPathCondition.createFromDelay(ReferenceType` *reference*`, long` *delay*`)`

`MotionPathCondition.createFromDistance(ReferenceType` *reference*`, double` *distance*`)`

Element	Description
reference	Data type: com.kuka.roboticsAPI.conditionModel.ReferenceType Reference point of the condition ■ **ReferenceType.START**: Start point ■ **ReferenceType.DEST**: End point
distance	Offset in space relative to the reference point of the condition. For CP motions, *distance* specifies the Cartesian distance between the switching point and reference point (= distance along the path which connects the switching point and reference point) and not the shortest distance between these points. (unit: mm) For PTP motions, *distance* does not specify a Cartesian distance but rather a path parameter without a unit. ■ Negative value: Offset contrary to the direction of motion ■ Positive value: Offset in the direction of motion (>>> "Maximum offset" Page 337)
delay	Offset in time relative to the path point defined by *distance*. Or if *distance* is not defined, to the reference point of the condition. (unit: ms) ■ Negative value: Offset contrary to the direction of motion ■ Positive value: Offset in the direction of motion Phases in which the application is paused are not included in the time measurement. (>>> "Maximum offset" Page 337)

Maximum offset

The switching point can only be offset within certain limits. The limits apply to the entire offset, comprising the shift in space and time.

- Negative offset, at most to the start point of the motion
- Positive offset, at most to the end point of the motion

The following parameterizations may not be used, as they will inevitably lead to an offset beyond the permissible limits and thus to a runtime error:

Value combination	Effect
reference = ReferenceType.START distance < 0	The switching point is before the start of the motion.
reference = ReferenceType.START distance = 0 delay < 0	
reference = ReferenceType.DEST distance > 0	The switching point is after the end of the motion.
reference = ReferenceType.DEST distance = 0 delay > 0	

Even if a valid value combination has been used, the switching point can nevertheless be offset beyond the permissible limits. In these cases, the response is as follows:

- A condition which is met before the start of the motion triggers the motion at the start point.
- A condition which is met after the end of the motion is never a trigger.

Example

A path-related condition is to be formulated for an adhesive bonding application. The adhesive bead is to end 5 cm before the end point of the motion. In order for the flow of adhesive to end in time, the condition must be met 700 ms before this distance to the end is reached.

```
MotionPathCondition glueStop = new
MotionPathCondition(ReferenceType.DEST, -50.0, -700);
```

15.20.9 Condition for Boolean signals

Description

The Boolean signal condition can be used to check digital 1-bit inputs. The condition is met if a Boolean input has a specific state.

Boolean signal conditions are of data type BooleanIOCondition.

Constructor syntax

`BooleanIOCondition(AbstractIO` *booleanSignal*`, boolean` *booleanIOValue*`)`

Element	Description
boolean Signal	Boolean input signal that is checked
boolean IOValue	State of the input signal with which the condition is met ■ **true**, **false**

Example

A Boolean digital input signal is returned via a switch. In order to react to the signal in an application, a Boolean signal condition is to be formulated. The condition must be fulfilled as soon a high level (state TRUE) is present when the switch is activated.

```
SwitchesIOGroup switches = new SwitchesIOGroup(...);
AbstractIO switch_1 = switches.getInput("Switch1");

BooleanIOCondition switch1_active = new BooleanIOCondition(switch_1,
true);
```

15.20.10 Condition for the range of values of a signal

Description

The value of a digital or analog input can be checked with the condition for the range of values of a signal. The condition is met if the value of the signal lies within a defined range.

Conditions for ranges of values are of data type ForceComponentCondition.

Constructor syntax

`IORangeCondition(AbstractIO` *signal*`, Number` *minValue*`, Number` *maxValue*`)`

Element	Description
signal	Analog or digital signal that is checked
minValue	Lower limit of the range of values in which the condition is met The value returned by the signal must be greater than or equal to *minValue*.
maxValue	Upper limit of the range of values in which the condition is met The value returned by the signal must be less than or equal to *maxValue*.

Example

A temperature sensor returns an analog input signal whose value can lie in the range between 0 °C and 2000 °C. As soon as a threshold of 35 °C is exceeded, a condition for monitoring the sensor signal should be met.

```
SensorIOGroup sensors = new SensorIOGroup(...);
AbstractIO temperatureSensor =
sensors.getInput("TemperatureSensor2");

IORangeCondition tempHigher35 = new
IORangeCondition(temperatureSensor, 35.0, 2000.0);
```

15.21 Break conditions for motion commands

For certain processes a planned motion must not be fully executed but rather terminated when definable events occur. For example, in joining processes, the robot must stop if a force threshold is reached.

15.21.1 Defining break conditions

Description

Break conditions are conditions which cause a motion to be terminated. A break condition is met if it already has the state TRUE before the start of the motion or if it switches to the state TRUE during the motion.

Conditions are defined as objects of type ICondition. The available condition types belong to the package com.kuka.roboticsAPI.conditionModel.

An overview of the available condition types can be found here:
(>>> 15.20.1 "Conditions in the RoboticsAPI" Page 321)

To define a break condition for a motion, an object of the desired condition type is transferred to the motion command via the motion method breakWhen(…).

breakWhen(…) can be called several times when programming a motion command to define different break conditions for a motion. The individual break conditions are then linked by a logic OR operation.

The following points must be taken into consideration when programming break conditions:

- For a spline block, break conditions can only be programmed for the entire spline block. Break conditions for individual splines segments are not permissible.
- If a break condition defined for a motion within a MotionBatch is triggered, this is terminated, and then the next motion command in the batch is executed. If a break condition defined for the entire MotionBatch occurs, the entire MotionBatch is terminated.
- A break condition causes the motion currently being executed to be terminated. If no appropriate reaction strategy is programmed in the application, subsequent motions are carried out immediately after the terminated motion.
- In the case of approximated motions, the approximate positioning arc is part of the path of the subsequent motion. For this reason, only the break conditions for the subsequent motion affect the approximate positioning arc.
- If the break condition in an approximated motion occurs just before the approximate positioning point is reached, and if this will not cause the robot to come to a standstill until it is on the approximate positioning arc, the robot is accelerated again when the approximate positioning arc is reached in order to execute the subsequent motion.

Syntax

motion`.breakWhen(` *condition_1*`<,` *condition_2*`, … >);`

Explanation of the syntax

Element	Description
motion	Type: Motion Motion for which a break condition is to be defined Example: ■ ptp(getApplicationData().getFrame("/P1"))
condition	Type: ICondition Parameterized ICondition object which describes a break condition

Example

A LIN motion is terminated if the torque in axis A3 is less than or equal to -12 Nm or greater than or equal to 0 Nm.

```
JointTorqueCondition cond_1 = new JointTorqueCondition(JointEnum.J3,
-12.0, 0.0);
robot.move(lin(getApplicationData().getFrame("/P10"))
           .breakWhen(cond_1));
```

15.21.2 Evaluating the break conditions

Description

If break conditions have been defined for a motion command, it is possible to view various information on the termination of a motion: For this purpose, the motion command is temporarily stored in an IMotionContainer variable. Via the method getFiredBreakConditionInfo(), this variable can be polled for an object of type IFiredConditionInfo, which contains the information about termination of the motion. If no break condition occurs during the motion, getFiredBreakConditionInfo() returns `zero`.

Syntax

`IMotionContainer` *motionCmd* = *motion*`.breakWhen(...);`

`IFiredConditionInfo` *firedCondInfo* = *motionCmd*`.getFiredBreakConditionInfo();`

Explanation of the syntax

Element	Description
motion	Motion instruction Example: ■ lbr.move(ptp(getApplicationData().getFrame("/P1"))
motionCmd	Type: IMotionContainer Temporary memory for the motion command
firedCondInfo	Type: IFiredConditionInfo Information about termination of the motion

Overview

The following methods are available in the IFiredConditionInfo interface:

Method	Description
getFiredCondition()	Return value type: ICondition Polls for the condition which caused a motion to be terminated
getPositionInfo()	Return value type: PositionInformation Polls for robot position valid at the time when the break condition was triggered.
getStoppedMotion()	Return value type: IMotion Polls for the segment of a spline block or the motion of a MotionBatch which was terminated

15.21.2.1 Polling a break condition

Description

The condition which caused the termination of a motion can be polled via the method getFiredCondition(). The return value is of type ICondition and can be compared to the transferred break conditions via the equals(…) method.

The poll is particularly useful if several break conditions for a motion have been defined by repeatedly calling the breakWhen(…) method.

Syntax

`ICondition` *firedCondition* `=` *firedCondInfo*`.getFiredCondition();`

Explanation of the syntax

Element	Description
firedCondition	Type: ICondition Variable for the return value. The variable contains the condition which caused the motion to be terminated.
firedCondInfo	Type: IFiredConditionInfo Information about termination of the motion

Example

The break conditions “cond1” and “cond2” are generated.

```
ICondition cond1;
ICondition cond2;
cond1 = new ...;
cond2 = new ...;
```

The break conditions “cond1” and “cond2” are transferred to a LIN motion with breakWhen(…). The “motionCmd” variable of type IMotionContainer can be used to evaluate the motion command.

```
IMotionContainer motionCmd =
lbr.move(lin(getApplicationData().getFrame("P10")).breakWhen(cond1).b
reakWhen(cond2));
```

The information about the termination of the motion are polled by "motionCmd". If the polled information is not equal to `null`, the motion has been terminated. The system only polls for the triggered break condition in this case.

```
IFiredConditionInfo firedInfo = motionCmd.getFiredConditionInfo();
if(firedInfo != null){

   ICondition firedCond = firedInfo.getFiredCondition();
   if(firedCond.equals(cond1)){
   ...
   }
...
}
```

15.21.2.2 Polling the robot position at the time of termination

Description

The robot position at the time when the break condition was triggered can be polled via the method getPositionInfo().

The following position information can be accessed via the return value of type PositionInformation.

- Axis-specific actual position
- Cartesian actual position
- Axis-specific setpoint position
- Cartesian setpoint position
- Setpoint/actual value difference (translational)
- Setpoint/actual value difference (rotational)

Syntax

`PositionInformation` *firedPosInfo* `=` *firedCondInfo*`.getPositionInfo();`

Explanation of the syntax

Element	Description
firedPosInfo	Type: PositionInformation Variable for the return value. The return value contains the position information at the time when the break condition was triggered.
firedCondInfo	Type: IFiredConditionInfo Information about termination of the motion

Example

The Cartesian actual position of the robot at the time when the break condition was triggered is polled via the method getCurrentCartesianPosition().

```
PositionInformation firedPosInfo = firedInfo.getPositionInfo();
Frame firedCurrPos = firedPosInfo.getCurrentCartesianPosition();
```

15.21.2.3 Polling a terminated motion (spline block, MotionBatch)

Description

Break conditions can be defined for an entire spline block or MotionBatch. If a break condition occurs, the entire spline block or MotionBatch is terminated.

The method getStoppedMotion() can be used to poll which spline segment or which motion of a MotionBatch has been terminated. The return value is of type IMotion.

Syntax

`IMotion` *stoppedMotion* `=` *firedCondInfo*`.getStoppedMotion();`

Explanation of the syntax

Element	Description
stoppedMotion	Type: IMotion Variable for the return value. The variable contains the terminated motion.
firedCondInfo	Type: IFiredConditionInfo Information about termination of the motion

Example

Poll using the example of a spline block:

```
ICondition stopCondition = new ...;
...
Spline splineMotion = new Spline(
   spl(getApplicationData().getFrame("/P1")),
   circ(getApplicationData().getFrame("/P2"),
        getApplicationData().getFrame("/P3")),
   spl(getApplicationData().getFrame("/P4")).setCartVelocity(150),
   lin(getApplicationData().getFrame("/P5"))
   ).setCartVelocity(250).breakWhen(stopCondition);

IMotionContainer splineCont = robot.move(splineMotion);

IFiredConditionInfo firedInfoSpline =
splineCont.getFiredConditionInfo();
   if(firedInfoSpline != null){
   IMotion stoppedMotion = firedInfoSpline.getStoppedMotion();
...
}
```

15.22 Path-related switching actions (Trigger)

A trigger is an event which is used to activate user-defined, path-related actions. If a specific event occurs while a motion is being executed, the action is triggered. The action is performed in parallel with the robot motion. For example, during a positioning motion, the gripper must be opened at the right time in order to be open when a setdown position for the workpiece it is transporting is free.

15.22.1 Programming triggers

Description

Events which activate path-related switching actions are called triggers. Events are defined using conditions. An event occurs if the defined condition already has the state TRUE before the start of the motion or if it switches to the state TRUE during the motion.

Conditions are defined as objects of type ICondition. The available condition types belong to the package com.kuka.roboticsAPI.conditionModel.

An overview of the available condition types can be found here:
(>>> 15.20.1 "Conditions in the RoboticsAPI" Page 321)

To program a trigger, an object of the desired condition type and an ITriggerAction object which describes the action to be executed are transferred to the motion command via the motion method triggerWhen(…).

triggerWhen(…) can be called several times when programming a motion command to define different triggers for a motion. The execution of the corresponding switching actions is only dependent on whether the triggering event occurs, and is not influenced by the order of calling via triggerWhen(…).

> The triggering event cannot re-trigger an action while it is being executed. The trigger is not effective again until the method triggerWhen(…) has ended. It is possible to poll for the number of events missed while the method was being executed.
> (>>> 15.22.3 "Evaluating trigger information" Page 345)

Syntax

motion`.triggerWhen(`*condition*`,` *action*`);`

Explanation of the syntax

Element	Description
motion	Type: Motion Motion for which a trigger must be defined Example: ■ ptp(getApplicationData().getFrame("/P1"))
condition	Type: ICondition Parameterized ICondition object which describes the condition for the trigger
action	Type: ITriggerAction ITriggerAction object which describes the action to be executed (>>> 15.22.2 "Programming a path-related switching action" Page 344)

15.22.2 Programming a path-related switching action

Description

The path-related action to be executed when an event occurs is defined via an ITriggerAction object. ITriggerAction is an interface from the package com.kuka.roboticsAPI.conditionModel. This interface currently does not offer any methods.

The ICallbackAction interface, which is derived from ITriggerAction, can be used for programming actions. The interface has the method onTriggerFired(…). The action to be carried out when the trigger is activated can be programmed in the body of the method onTriggerFired(…).

An ICallbackAction object can be used in any number of triggers.

> The onTriggerFired(…) method is not called in real time: It is therefore not possible to guarantee specific time behavior. This can lead to delayed execution of the action.

Syntax

```
ICallbackAction action = new ICallbackAction() {
@Override
public void onTriggerFired(IFiredTriggerInfo
triggerInformation) {
//Action to be executed
}
};
```

Explanation of the syntax

Element	Description
action	Type: ICallbackAction ICallbackAction object which describes the action transferred with triggerWhen(…)
`onTriggerFired(…)`	Method whose execution is fired by the trigger
`triggerInformation`	Type: IFiredTriggerInfo Contains information about the firing trigger (>>> 15.22.3 "Evaluating trigger information" Page 345)

Example

During motion to point “P1”, output “DO1” is always switched at the moment when input “DI1” is TRUE.

```
//set trigger action
ICallbackAction toggleOut_1 = new ICallbackAction() {

   @Override
   public void onTriggerFired(IFiredTriggerInfo triggerInformation)
   {
      //toggle output state when trigger fired
      if(IOs.getDO1())
      {
         IOs.setDO1(false);
      }
      else
      {
         IOs.setDO1(true);
      }

   }
};

//set trigger condition
BooleanIOCondition buttonPressed = new
BooleanIOCondition(IOs.getInput("DI1"), true);

//motion with trigger
robot.move(ptp(P1)).triggerWhen(buttonPressed, toggleOut_1));
robot.move(ptp(P2));
```

15.22.3 Evaluating trigger information

The method onTriggerFired(…) is called when a trigger is activated. The object `triggerInformation` of type IFiredTriggerInfo, which contains various information about the activating trigger, is transferred to the method onTriggerFired(…). This trigger information can be polled.

Overview

The following methods of the IFiredTriggerInfo class are available:

Method	Description
getFiredCondition()	Return value type: ICondition Polls for the condition which fired the trigger
getMissedEvents()	Return value type: int Polls for how many times the event which fired the trigger still occurred while the triggered action was being executed **Note**: The triggering event cannot re-trigger an action while it is being executed.
getMotionContainer()	Return value type: IMotionContainer Polls for the motion command, during the execution of which the trigger was fired
getPositionInformation()	Return value type: PositionInformation Polls for position information valid at the time when the trigger was fired. The return value contains the following position information: ■ Axis-specific actual position ■ Cartesian actual position ■ Axis-specific setpoint position ■ Cartesian setpoint position ■ Setpoint/actual value difference (translational) ■ Setpoint/actual value difference (rotational)
getTriggerTime()	Return value type: java.util.Date Polls for the time at which the trigger was fired

To poll for the position information obtained with getPositionInformation(), the following methods of the PositionInformation class are available:

Method	Description
getCommandedCartesianPosition()	Return value type: Frame Polls for the Cartesian setpoint position at triggering time
getCommandedJointPosition()	Return value type: JointPosition Polls for the axis-specific setpoint position at triggering time
getCurrentCartesianPosition()	Return value type: Frame Polls for the Cartesian actual position at triggering time
getCurrentJointPosition()	Return value type: JointPosition Polls for the axis-specific actual position at triggering time

Example 1

When the trigger is fired, the triggering time and condition are displayed on the smartHMI.

```
BooleanIOCondition in1 = new BooleanIOCondition(_input_1, true);

ICallbackAction ica = new ICallbackAction() {

   @Override
   public void onTriggerFired(IFiredTriggerInfo triggerInformation)
   {
      getLogger().info("TriggerTime: "+
triggerInformation.getTriggerTime().toString());
      getLogger().info("TriggerCondition: "+
triggerInformation.getFiredCondition().toString());
   }
```

```
};

robot.move(P1).triggerWhen(in1, ica));
```

Example 2 The axis-specific and Cartesian robot position at triggering time are polled.

```
BooleanIOCondition in1 = new BooleanIOCondition(_input_1, true);

ICallbackAction ica = new ICallbackAction() {

   @Override
   public void onTriggerFired(IFiredTriggerInfo triggerInformation)
   {
      PositionInformation posInfo =
triggerInformation.getPositionInformation();
      posInfo.getCommandedCartesianPosition();
      posInfo.getCommandedJointPosition();
      posInfo.getCurrentCartesianPosition();
      posInfo.getCommandedJointPosition();}
   }
};

robot.move(P1).triggerWhen(in1, ica));
```

15.23 Monitoring processes (Monitoring)

Monitoring means keeping a process under surveillance using a listener so that it is possible to react to certain events while an application is running.

These events are changes in state of defined conditions. The listener monitors the state of the condition. If the state of the condition changes, the listener is notified and the predetermined handling routine is triggered as a reaction.

During execution of a handling routine, the listener is not informed if further events occur. Once the handling routine has been completed, these events are only transferred to the listener and handled if the appropriate notification type has been defined.

(>>> 15.23.3 "Registering a listener for notification of change in state" Page 349)

15.23.1 Listener for monitoring conditions

Various listener interfaces are available from the package com.kuka.roboticsAPI.conditionModel for monitoring a condition. The listeners differ in type in that they are each notified of a certain change in state of the monitored condition.

Each listener type declares a method which is executed when the listener is notified. The desired handling routine is programmed in the body of this method.

Data type	Description
IRisingEdgeListener	Notification when the monitored condition is met (rising edge, change in state FALSE > TRUE). Method for the handling routine: ■ onRisingEdge(…)
IFallingEdgeListener	Notification when the monitored condition is no longer met (falling edge, change in state TRUE > FALSE). Method for the handling routine: ■ onFallingEdge(…)
IAnyEdgeListener	Notification for every change in state of the condition (rising or falling edge, change in state FALSE > TRUE or TRUE > FALSE). Method for the handling routine: ■ onAnyEdge(…)

Overview

The following programming steps are required in order to be able to react to the change in state of a condition:

Step	Description
1	Create a listener object to monitor the condition. (>>> 15.23.2 "Creating a listener object to monitor the condition" Page 348)
2	Program the the desired handling routine in the listener method.
3	Register the listener for notification in case of a change in state of the condition. (>>> 15.23.3 "Registering a listener for notification of change in state" Page 349)
4	If this has not already been done by the method selected for registration, activate the notification service for the listener. (>>> 15.23.4 "Activating or deactivating the notification service for listeners" Page 350)

15.23.2 Creating a listener object to monitor the condition

Description

The syntax of a listener object is described here using the listener IAnyEdgeListener as an example. The listener method onAnyEdge(…), which is automatically declared when the object is created, has input parameters. These input parameters contain information about the event triggered by the execution of the method, and can be polled and evaluated.

The listener objects of the other listener types are created in the same way and are structured analogously.

Syntax

```
IAnyEdgeListener condListener = new IAnyEdgeListener() {
@Override
public void onAnyEdge(ConditionObserver conditionObser-
ver, Date time, int missedEvents, boolean conditionValue)
{
// Reaction to change in state
}
};
```

Explanation of the syntax

Element	Description
condListener	Type: IAnyEdgeListener Name of the listener object
Input parameters of the listener method:	
`condition Observer`	Type: ConditionObserver Object notified by the listener
`time`	Type: Date Date and time the listener was notified
`missed Events`	Type: int Number of changes in state which have occurred but not been handled. Possible causes of non-handled events: ■ The notification service was deactivated when the triggering event occurred. ■ The handling routine was being executed when the triggering event occurred again. These events can be handled using the notification type **NotificationType.MissedEvents**. (>>> "NotificationType" Page 350)
`condition Value`	Type: Boolean Only present with the listener method onAnyEdge(…). Specifies the edge via which the method was called. ■ **true**: rising edge (change in state FALSE > TRUE) ■ **false**: falling edge (change in state TRUE > FALSE)

15.23.3 Registering a listener for notification of change in state

Description

An object of type ConditionObserver is required to register a listener for notification in case of a change in state.

To create an object of type ConditionObserver, the ObserverManager of the application must first be polled via the method getObserverManager(). The ObserverManager class provides various methods for creating the required object.

- createAndEnableConditionObserver(…)
  The notification service for the listener is active immediately.
- createConditionObserver(…)
  The notification service for the listener is not active immediately, but rather must be explicitly activated.
  (>>> 15.23.4 "Activating or deactivating the notification service for listeners" Page 350)

The transferred parameters in each case are identical for both methods.

Syntax

```
ConditionObserver myObserver =
getObserverManager().createAndEnableConditionObserver
(condition, notificationType, listener)
```

Explanation of the syntax

Element	Description
myObserver	Type: ConditionObserver Object which monitors the defined condition
condition	Type: ICondition Condition which is monitored
notificationType	Type: Enum of type NotificationType Notification type Defines the events at which the listener is to be notified in order to execute the desired handling routine. (>>> "NotificationType" Page 350)
listener	Type: IRisingEdgeListener, IFallingEdgeListener or IAnyEdgeListener Listener object which is registered

NotificationType

The Enum of type NotificationType has the following values:

Value	Description
EdgesOnly	The listener is only notified in the event of an edge change (according to the listener type used).
OnEnable	The listener is notified in the event of an edge change (according to the listener type used). In addition, the state of the monitored condition is checked upon activation of the listener. Depending on the listener type, the listener is notified when the following events occur: ■ IRisingEdgeListener: only if the condition is met upon activation ■ IFallingEdgeListener: only if the condition is not met upon activation ■ IAnyEdgeListener: if the condition is met or not met upon activation
MissedEvents	The listener is notified in the event of an edge change (according to the listener type used). In addition, following the execution of the handling routine, the listener is notified if triggering events were missed. This means that if the triggering edge change again occurs during execution of the handling routine, the listener is also notified again, and the handling routine is executed a second time.
All	Combination of OnEnable and MissedEvents The listener is notified in the case of all events described under OnEnable and MissedEvents.

15.23.4 Activating or deactivating the notification service for listeners

Description

The methods for activating or deactivating the notification service belong to the ConditionObserver class.

The notification service must only be activated if the method createConditionObserver(…) was used to register the listener.

Syntax

To activate the notification service:

myObserver`.enable()`

To deactivate the notification service:

myObserver`.disable()`

Explanation of the syntax

Element	Description
myObserver	Type: ConditionObserver Object which monitors the defined condition

15.23.5 Programming example for monitoring

A listener of type IRisingEdgeListener is defined for monitoring a force condition. As soon as a force of 35 N on the TCP is exceeded, this is considered a collision. The listener is notified and a warning lamp lights up.

NotificationType.MissedEvents is defined as the notification type. If the permissible force on the TCP is exceeded multiple times while the warning lamp is switched on, the listener will be informed promptly.

```
ForceCondition collision = ForceCondition
.createSpatialForceCondition(tool.getDefaultMotionFrame(), 35);

IRisingEdgeListener collisionListener = new IRisingEdgeListener() {

   @Override
   public void onRisingEdge(ConditionObserver conditionObserver,
   Date time, int missedEvents) {

   signals.setWarningLED(true);

   }
});

ConditionObserver collisionObserver = getObserverManager()
.createConditionObserver(collision, NotificationType.MissedEvents,
collisionListener);
collisionObserver.enable();
```

15.24 Blocking wait for condition

Description

With waitFor(…), an application is stopped until a certain condition is met or a certain wait time has expired. The application is then resumed.

> Latency times may occur while the command is being processed. It is not possible to guarantee that the programmed wait time will be maintained exactly.

waitFor(…) must access the ObserverManager of the application. This is called with getObserverManager().

All condition types are supported with the exception of MotionPathCondition.

An overview of the available condition types can be found here:
(>>> 15.20.1 "Conditions in the RoboticsAPI" Page 321)

Syntax

Blocking wait with no time limit:

`getObserverManager().waitFor(`*condition*`)`

Blocking wait with a time limit:

`boolean` *result* `= getObserverManager().waitFor(`*condition*`,` *timeout*`,` *timeUnit*`)`

Explanation of the syntax

Element	Description
condition	Type: ICondition Condition which is waited for. If the condition is already met when waitFor(…) is called, the application is immediately resumed.
timeout	Type: long Maximum wait time. If the condition of the defined wait time does not occur, the application is also resumed without the occurrence of the condition.
timeUnit	Type: Enum of type TimeUnit Unit of the given wait time. The Enum TimeUnit is an integral part of the standard Java library.
result	Type: Boolean Variable for the return value of waitFor(…). The return value is true if the condition occurs within the specified wait time. **Note**: If no wait time is defined, waitFor(…) does not supply a return value.

Example

A wait for a Boolean input signal is required in the application. The application is to be blocked for a maximum of 30 seconds. If the input signal is not supplied within this time, a defined handling routine is then to be executed.

```
SwitchIOGroup inputs = new SwitchIOGroup(kuka_Sunrise_Cabinet);
Input input = inputs.getInput ("Input");

BooleanIOCondition inputCondition = new BooleanIOCondition(input,
true);

boolean result = getObserverManager().waitFor(inputCondition, 30,
TimeUnit.SECONDS);

if(!result){
   //do something
}
else{
   //continue program
}
```

15.25 Recording and evaluating data

While an application is being executed, specific data, for example external forces and torques, can be recorded for later evaluation. The DataRecorder class (package: com.kuka.roboticsAPI.sensorModel) is available for programming the data recording.

The recorded data are saved in a file and stored on the robot controller in the directory C:\KRC\Roboter\Log\DataRecorder.

The file name is defined with the DataRecorder object to be created. If an error has occurred during recording, the file name begins with “FaultyDataRecorder…”.

The file can be opened with a text editor or read into an Excel table.

15.25.1 Creating an object for data recording

Description

For data recording, an object of type DataRecorder must first be created and parameterized. The following default parameters are set if the standard constructor is used for this purpose:

- The file name under which the recorded data are saved is created automatically. The name also contains an ID which is internally assigned by the system: `DataRecorderID.log`
- No recording duration is defined. Data are recorded until the buffer (currently 16 MB) is full or the maximum number of data sets (currently 30,000) is reached.
- The recording rate, i.e. the minimum time between 2 recordings, is 1 ms.

Constructor syntax

The DataRecorder class has the following constructors:

`DataRecorder()` (standard constructor)

`DataRecorder(String` *fileName*`, long` *timeout*`, TimeUnit` *timeUnit*`, int` *sampleRate*`)`

Explanation of the syntax

Element	Description
fileName	File name (with extension) under which the recorded data are saved Example: “Recording_1.log”
timeout	Recording duration - **-1**: No recording duration is defined. - **≥ 1** Default: -1 The time unit is defined with *timeUnit*.
timeUnit	Time unit for the recording duration Example: TimeUnit.SECONDS The Enum TimeUnit is an integral part of the standard Java library.
sampleRate	Recording rate (unit: ms) - **≥ 1** Default: 1

The DataRecorder class offers “set” methods which can be used to adapt the parameter values, in particular when using the standard constructor.

- setFileName(…), setSampleRate(…), setTimeout(…, …)

In setTimeout(…, …), the first parameter defines the recording duration and the second parameter defines the corresponding time unit.

Example 1

Data are to be recorded every 100 ms for a duration of 5 s and written to the file Recording_1.log.

```
DataRecorder rec_1 = new DataRecorder("Recording_1.log", 5,
TimeUnit.SECONDS, 100);
```

Example 2

The DataRecorder object is generated using the standard constructor. This only specifies that data are recorded every 1 ms for an indefinite duration. The

recorded data are to be written to the file Recording_2.log. The file name is defined with the corresponding “set” method.

```
DataRecorder rec_2 = new DataRecorder();
rec_2.setFileName("Recording_2.log");
```

15.25.2 Specifying data to be recorded

Using dot operators and the corresponding “add” method, the data to be recorded are added to the DataRecorder object created for this purpose. The simultaneous recording of various data is possible.

Overview The following “add” methods of the DataRecorder class are available:

Method	Description
addInternalJointTorque(…)	Return value type: DataRecorder Recording of the measured axis torques of the robot which is transferred as a parameter (type: robot)
addExternalJointTorque(…)	Return value type: DataRecorder Recording of the external axis torques (adjusted to the model) of the robot which is transferred as a parameter (type: robot)
addCartesianForce(…)	Return value type: DataRecorder Recording of the Cartesian forces along the X, Y and Z axes of the frame which is transferred as a parameter (unit: N). The variance of the Cartesian forces is also recorded. A second frame can be transferred as a parameter in order to define the orientation for the force measurement. If no separate frame is specified for the orientation, `null` must be transferred.
addCartesianTorque(…)	Return value type: DataRecorder Recording of the Cartesian torques along the X, Y and Z axes of the frame transferred as a parameter (unit: Nm). The variance of the Cartesian forces is also recorded. A second frame can be transferred as a parameter in order to define the orientation for the torque measurement. If no separate frame is specified for the orientation, `null` must be transferred.
	Parameters: ■ `AbstractFrame` *measureFrame* Frame connected to the robot flange, e.g. the TCP of a tool. Defines the position of the measurement point. ■ `AbstractFrame` *orientationFrame* Defines the orientation of the measurement point. **Note**: Both parameters must always be transferred together. The orientation may be `null`.
addCommandedJointPosition(…)	Return value type: DataRecorder Recording of the axis-specific setpoint position of the robot which is transferred as a parameter (type: robot). As a second parameter, the unit in which the axis angles are recorded must be transferred (Enum of type: AngleUnit).

Method	Description
addCurrentJointPosition(…)	Return value type: DataRecorder Recording of the axis-specific actual position of the robot which is transferred as a parameter (type: robot) As a second parameter, the unit in which the axis angles are recorded must be transferred (Enum of type: AngleUnit).

	Parameters:
	■ `Robot` *robot*
	■ `AngleUnit` *angleUnit*
	  ■ **AngleUnit.Degree**: Axis angle in degrees
	  ■ **AngleUnit.Radian**: Axis angle in rad

addCommandedCartesianPositionXYZ(…)	Return value type: DataRecorder Recording of the Cartesian setpoint position (translational section) The measurement point and reference coordinate system relative to which the position is recorded are transferred as parameters.
addCurrentCartesianPositionXYZ(…)	Return value type: DataRecorder Recording of the Cartesian actual position (translational section) The measurement point and reference coordinate system relative to which the position is recorded are transferred as parameters.

	Parameters:
	■ `AbstractFrame` *measureFrame*
	  Frame connected to the robot flange, e.g. the TCP of a tool. Defines the position of the measurement point.
	■ `AbstractFrame` *referenceFrame*
	  Defines the reference coordinate system.
	Note: Both parameters must always be transferred together. None of the parameters may be `null`.

Example

For an LBR iiwa, the following data are to be recorded using a DataRecorder object:

- Axis torques which are measured on the robot
- Force on the TCP of a gripper mounted on the robot with the orientation of a base frame

```
private LBR lbr_iiwa;
private Tool gripper;
...
gripper.attachTo(lbr_iiwa.getFlange());

DataRecorder rec = new DataRecorder();
rec.addInternalJointTorque(lbr_iiwa);
rec.addCartesianForce(gripper.getFrame("TCP"),
getApplicationData().getFrame("/Base"));
```

15.25.3 Starting data recording

Data recording can be started independent of robot motion (possible at any point in the application), or synchronous with robot motion by means of a trigger.

Independent of robot motion

Before motion-independent recording is started, the DataRecorder object must be activated via the enable() method. Recording is started via the startRecording() method.

When recording has ended, the DataRecorder object is automatically deactivated. If data are to be recorded again with the same DataRecorder object, the DataRecorder must be re-activated.

> It is not possible for more than one DataRecorder object to be activated at any one time.

Synchronous via a trigger

A condition of type ICondition and an action must be formulated for a trigger. When this condition is met, the trigger is fired, causing the action to be carried out.

(>>> 15.22.1 "Programming triggers" Page 343)

This action starts the data recording. An object of type StartRecordingAction must be transferred for this purpose. When the object is created, the DataRecorder object to be used for data recording must be specified.

Constructor syntax:

`StartRecordingAction(DataRecorder` *recorder*`)`

The ICondition object and the StartRecordingAction object are subsequently linked to a motion command with triggerWhen(…).

Example 1

Data recording is to start when the robot has carried out the approach motion to a pre-position. The DataRecorder object is activated before the pre-position is addressed so as to reduce the delay when starting the recording.

```
private LBR lbr_iiwa;
...
DataRecorder rec = new DataRecorder();
...
rec.enable();
...
lbr_iiwa.move(lin(getApplicationData().getFrame("/Pre-position")));
rec.startRecording();
```

Example 2

Data recording is to begin 2 s after the start of a motion. A MotionPathCondition object is parameterized for this.

```
private LBR lbr_iiwa;
...
DataRecorder rec = new DataRecorder();
...
StartRecordingAction startAct = new StartRecordingAction(rec);
MotionPathCondition startCond = new
MotionPathCondition(ReferenceType.START, 0.0, 2000);
lbr_iiwa.move(lin(getApplicationData().getFrame("/
Destination")).triggerWhen(startCond, startAct));
```

15.25.4 Ending data recording

Data recording can be ended independent of robot motion (possible at any point in the application), or synchronous with robot motion by means of a trigger.

In addition, recording is automatically ended when the application ends or when the recording duration specified in the DataRecorder object used has been reached.

Independent of robot motion

Recording can be stopped at any time via the stopRecording() method.

Synchronous via a trigger

A condition of type ICondition and an action must be formulated for a trigger. When this condition is met, the trigger is fired, causing the action to be carried out.

(>>> 15.22.1 "Programming triggers" Page 343)

This action ends the data recording. An object of type StopRecordingAction must be transferred for this purpose. When the object is created, the DataRecorder object to be used for data recording must be specified.

Constructor syntax:

`StopRecordingAction(DataRecorder` *recorder*`)`

The ICondition object and the StopRecordingAction object are linked to a motion command with triggerWhen(…).

15.25.5 Polling states from the DataRecorder object

Overview

The following methods of the DataRecorder class are available:

Method	Description
isEnabled()	Return value type: Boolean The system polls whether the DataRecorder object is activated (= true).
isRecording()	Return value type: Boolean The system polls whether data recording is running (= true).
isFileAvailable()	Return value type: Boolean The system polls whether the file with the recorded data is already saved on the robot controller and whether it is available for evaluation (= true).
awaitFileAvailable(…)	Return value type: Boolean Blocks the calling application or background task until the defined blocking duration has expired or until the file with the recorded data is saved on the robot controller and is available for evaluation (= true). The blocking statement returns the value “false” if the file is not available within the maximum blocking duration. Syntax: ■ `awaitFileAvailable(long` *timeout*`, java.util.concurrent.TimeUnit` *timeUnit*`)` Parameters: ■ *timeout*: maximum blocking duration ■ *timeUnit*: time unit for the maximum blocking time

15.25.6 Example program for data recording

The following are to be recorded during an assembly process: the torques acting externally on the axes of an LBR iiwa and the Cartesian forces acting on the TCP of a gripper on the robot flange. The data are to be recorded every 10 ms.

Recording is to begin synchronously with robot motion when the force acting from any direction on the TCP of the gripper exceeds 20 N. When the assembly process ends, recording is to end as well.

The file is then to be evaluated if it is available after a maximum of 5 s.

```
private LBR lbr_iiwa;
private Tool gripper;
...
gripper.attachTo(lbr_iiwa.getFlange());
...
DataRecorder rec = new DataRecorder();
rec.setFileName("Recording.log");
rec.setSampleRate(10);

rec.addExternalJointTorque(lbr_iiwa);
rec.addCartesianForce(gripper.getFrame("/TCP"), null);

StartRecordingAction startAction = new StartRecordingAction(rec);
ForceCondition startCondition =
ForceCondition.createSpatialForceCondition(gripper.getFrame("/TCP"),
20.0);

lbr_iiwa.move(ptp(getApplicationData().getFrame("/StartPosition")));
lbr_iiwa.move(lin(getApplicationData().getFrame("/
MountingPosition")).triggerWhen(startCondition, startAction));
lbr_iiwa.move(lin(getApplicationData().getFrame("/DonePosition")));

rec.stopRecording();

if (rec.awaitFileEnable(5, TimeUnit.SECONDS)){
// Evaluation of the file if available
}
```

15.26 Defining user keys

Description

Functions can be freely assigned to the 4 user keys on the smartPAD. For this purpose, various user key bars can be defined in the source code of the robot application or background tasks.

The user keys are assigned functions using the user key bar. One user key on the bar must be assigned a function, but it is not necessary for all of the keys to be assigned. In addition, graphical or text elements illustrating the function of each user key are located on the side panel of the smartHMI screen next to the user keys.

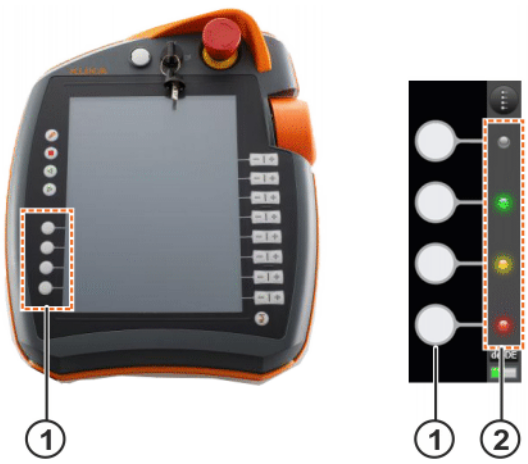


Fig. 15-18: User keys on the smartPAD (example)

1 User keys
2 Bar with LED icons

All the user key bars defined in the running robot application or the background task are available to the operator. For example, one user key bar can be used for controlling a gripper, and in another bar the same keys can be used to select different program sections.

User key bars are available until the robot application or background task which created them has ended.

Overview

The following steps are required in order to program a user key bar:

Step	Description
1	Create a user key bar. (>>> 15.26.1 "Creating a user key bar" Page 359)
2	Add user keys to the bar (at least one). (>>> 15.26.2 "Adding user keys to the bar" Page 360)
3	Define the function which is to be executed if the user key is actuated. (>>> 15.26.3 "Defining the function of a user key" Page 362)
4	Assign at least one graphical or text element to the area along the left side panel of the smartHMI next to the user key. (>>> 15.26.4 "Labeling and graphical assignment of the user key bar" Page 363)
5	For user keys which trigger functions associated with a risk: Define the warning message to be displayed when the user key is actuated. The message appears before the function can be triggered. (>>> 15.26.5 "Identifying safety-critical user keys" Page 367)
6	Publish the user key bar. (>>> 15.26.6 "Publishing a user key bar" Page 367)

15.26.1 Creating a user key bar

Description

The following methods are required in order to create a user key bar:

- getApplicationUI()

  This method is used to access the interface to the smartHMI graphical user interface from a robot application or a background task. Return value type: ITaskUI
- createUserKeyBar(…)

  This method is used to create the user key bar. It is part of the ITaskUI interface.

Syntax

```
IUserKeyBar keybar =
getApplicationUI().createUserKeyBar("name");
```

Explanation of the syntax

Element	Description
keybar	Type: IUserKeyBar Name of the user key bar created with createUserKeyBar(…)
name	Type: String Name under which the user key bar is displayed on the smartHMI (>>> Fig. 6-16) The number of characters which can be displayed is limited. ■ A maximum of 12 to 15 characters is recommended.

Example

A user key bar for controlling a gripper is created.

```
IUserKeyBar gripperBar =
getApplicationUI().createUserKeyBar("Gripper");
```

15.26.2 Adding user keys to the bar

Description

A newly created user key bar does not have any user keys to start with. The user keys to be used must be added to the bar.

The IUserKeyBar interface provides the following methods for this purpose:

- addUserKey(…)

  Adds a single user key to the bar.
- addDoubleUserKey(…)

  Combines 2 neighboring user keys to a double key and adds this to the bar. The corresponding areas on the side panel of the smartHMI screen are also combined into a larger area.

When adding a user key to a bar, the user defines the function to be executed when the user key is actuated (e.g. opening a gripper, changing a parameter, etc.). Depending on the programming, both pressing and releasing the user key can be interpreted as actuation and linked to a function.

A user key bar must have at least one user key. Each user key is assigned a unique number. This number is transferred when a user key is added.

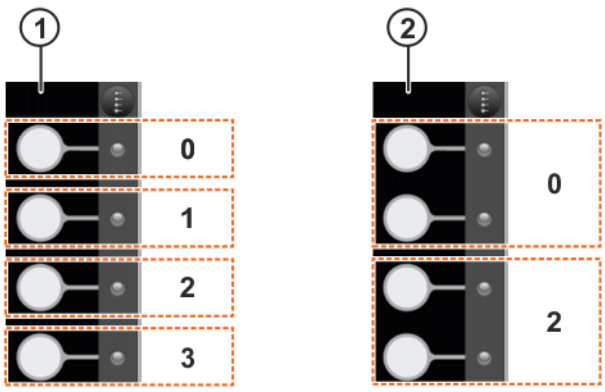


Fig. 15-19: Numbering of the user keys

1 Single keys 2 Double keys

Syntax

Adding a single key:

`IUserKey` *key* `=` *keybar*`.addUserKey(`*slot*`,` *listener*`,` *ignoreEvents*`);`

Adding a double key:

`IUserKey` *doubleKey* `=` *keybar*`.addDoubleUserKey(`*slot*`,` *listener*`,` *ignoreEvents*`);`

Explanation of the syntax

Element	Description
keybar	Type: IUserKeyBar Name of the user key bar to which a user key is added
key	Type: IUserKey Name of the single key added to the bar
doubleKey	Type: IUserKey Name of the double key added to the bar
slot	Type: int Number of the user key which is added. Single keys: ■ **0 … 3** Double keys: ■ **0, 2**
listener	Type: IUserKeyListener Name of the listener object used to define the function to be executed when the user key is actuated (>>> 15.26.3 "Defining the function of a user key" Page 362)
ignoreEvents	Type: Boolean Defines whether there is a reaction if the user key is re-actuated while the key function is being executed ■ **true**: If the key is actuated while the function is being executed, it has no effect. ■ **false**: It is counted how many times the key is actuated while the function is being executed. The function is repeated this many times.

Example

The user keys are assigned the following functions for controlling a gripper:

- The top user key is to be used to open the gripper, and the key below it is to close the gripper.
- The two lower user keys are combined in a double key. This is to be used to increase and decrease the velocity of the gripper.
- The functions for opening and closing the gripper are not to be called again until the respective function has ended.

```
IUserKeyBar gripperBar =
getApplicationUI().createUserKeyBar("Gripper");

IUserKeyListener openGripperListener = ...;
IUserKeyListener closeGripperListener = ...;
IUserKeyListener gripperVelocityListener = ...;

IUserKey openKey = gripperBar.addUserKey(0,
   openGripperListener, true);
IUserKey closeKey = gripperBar.addUserKey(1,
   closeGripperListener, true);
IUserKey velocityKey = gripperBar.addDoubleUserKey(2,
   gripperVelocityListener, false);
```

15.26.3 Defining the function of a user key

Description

In order to define which function is to be executed when a user key is actuated, a listener object of type IUserKeyListener must be created. The onKeyEvent(…) method is automatically declared when the object is created.

The listener method onKeyEvent(…) is called when the following events occur:

- The user key is pressed.
- The user key is released.

> Only one OnKeyEvent(…) can be carried out even if different listeners are used. For example, if the user triggers the OnKeyEvent(…) of user key 2 while the OnKeyEvent(…) of user key 1 is being executed, the second OnKeyEvent(…) will not start until the first has been completed.

Syntax

```
IUserKeyListener listener = new IUserKeyListener() {
@Override
public void onKeyEvent(IUserKey key, UserKeyEvent event) {
// Reaction to event
}
};
```

Explanation of the syntax

Element	Description
listener	Type: IUserKeyListener Name of the listener object
Input parameters of the listener method onKeyEvent(…):	

Element	Description
`key`	Type: IUserKey User key which has been actuated The parameter can be used to directly access the user key, for example to change the corresponding labelling or graphical assignment. In addition, it is possible to determine which user key has been actuated, especially when the same reaction is used for different user keys.
`event`	Type: Enum of type UserKeyEvent Event called by the listener method onKeyEvent(…) Enum values for single keys: ■ **UserKeyEvent.KeyDown**: Key has been pressed. ■ **UserKeyEvent.KeyUp**: Key has been released. Enum values for double keys: ■ **UserKeyEvent.FirstKeyDown**: Of the two keys, the upper one has been pressed. ■ **UserKeyEvent.SecondKeyDown**: Of the two keys, the lower one has been pressed. ■ **UserKeyEvent.FirstKeyUp**: Of the two keys, the upper one has been released. ■ **UserKeyEvent.SecondKeyUp**: Of the two keys, the lower one has been released.

Example

The user key bar for controlling a gripper is expanded by a method which can be used to adapt the velocity of the gripper. The two lower user keys combined in a double key are used for this purpose.

The attribute `velocity` is declared for setting the velocity. The attribute specifies the current velocity as a proportion of the maximum velocity (range of values: 0.1 … 1.0). Pressing the upper user key increases the value by 0.1 and pressing the lower user key decreases it by 0.1.

```
final double velocity = 0.1;
...
IUserKeyBar gripperBar = ...;
...
IUsertKeyListener gripperVelocityListener = new IUserKeyListener(){
   @Override
   public void onKeyEvent(IUserKey key, IUserKeyEvent event){
     if(event == UserKeyEvent.FirstKeyDown && velocity <= 0.9){
       velocity = velocity + 0.1;
     }
     else if(event == UserKeyEvent.SecondKeyDown && velocity >= 0.2){
       velocity = velocity - 0.1;
     }
   }
};
...
IUserKey velocityKey = gripperBar.addDoubleUserKey(2,
   gripperVelocityListener, false);
```

15.26.4 Labeling and graphical assignment of the user key bar

Description

At least one graphical or text element must be assigned to the area along the left side panel of the smartHMI next to the user key. LED icons of various col-

ors and sizes are available as graphical elements. These elements can be adapted during the runtime of the robot application or the background task.

In order to clearly position the individual elements, the area next to the user key is divided into a grid with 3x3 spaces. The same also applies to user keys which have been combined into a double key. In this case, the grid extends across both spaces.

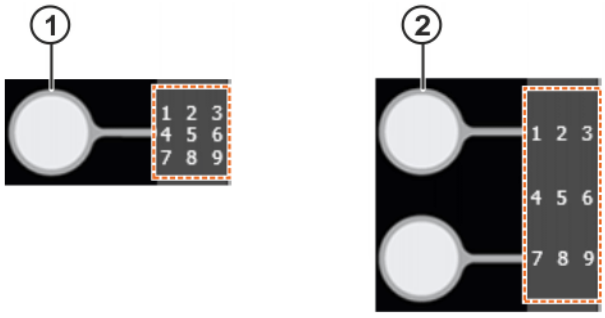


Fig. 15-20: Division of the grid

1 Single keys 2 Double keys

One element can be set in each grid space. This grid space is defined by the value of the enum UserKeyAlignment. If a new element is allocated to a grid space which has already been assigned, the existing element is deleted.

UserKey alignment

Grid space no.	Value
1	UserKeyAlignment.TopLeft
2	UserKeyAlignment.TopMiddle
3	UserKeyAlignment.TopRight
4	UserKeyAlignment.MiddleLeft
5	UserKeyAlignment.Middle
6	UserKeyAlignment.MiddleRight
7	UserKeyAlignment.BottomLeft
8	UserKeyAlignment.BottomMiddle
9	UserKeyAlignment.BottomRight

15.26.4.1 Assigning a text element

Description

Each grid space can be assigned a text element. The setText(…) method is used for this purpose. The method belongs to the IUserKey interface.

Syntax

key`.setText(`*position*`, "`*text*`");`

Explanation of the syntax

Element	Description
key	Type: IUserKey User key to which a text element is assigned

Element	Description
position	Type: Enum of type UserKeyAlignment Position of the element (grid space) (>>> "UserKey alignment" Page 364)
text	Type: String Text to be displayed Often, a text length of 2 or more characters will exceed the size of the grid space. The text display area is then expanded; however, it is only practical to use a limited number of characters. This depends on the text elements of the neighboring grid spaces and the characters used.

Example

The user key bar for controlling a gripper is to be expanded. A suitable label should be displayed continuously next to each of the user keys.

- Label for the user keys for opening and closing the gripper: OPEN and CLOSE
- Label for the user keys for increasing and decreasing the gripper velocity: Plus sign and minus sign

  In addition, the current velocity is to be displayed and automatically updated each time a change is made.

```
final double velocity = 0.1;
...
IUserKeyBar gripperBar = ...;
...
IUserKeyListener gripperVelocityListener = new IUserKeyListener(){
   @Override
   public void onKeyEvent(IUserKey key, IUserKeyEvent event){
     if(event == UserKeyEvent.FirstKeyDown && velocity <= 0.9){
       velocity = velocity + 0.1;
     }
     else if(event == UserKeyEvent.SecondKeyDown && velocity >= 0.2){
       velocity = velocity - 0.1;
     }
     // The following line formats the velocity display
     // Display of the first 3 characters
     String value = String.valueOf(velocity).substring(0, 3);
     key.setText(UserKeyAlignment.Middle, value);
     }
   }
};
IUserKey openKey = ...;
openKey.setText(UserKeyAlignment.TopLeft, "OPEN");

IUserKey closeKey = ...;
closeKey.setText(UserKeyAlignment.TopLeft, "CLOSE");

IUserKey velocityKey = ...;
velocityKey.setText(UserKeyAlignment.TopMiddle, "+");
velocitykey.setText(UserKeyAlignment.Middle,
   Double.toString(velocity));
velocityKey.setText(UserKeyAlignment.BottomMiddle, "-");
```

15.26.4.2 Assigning an LED icon

Description

Each grid space can be assigned an LED icon. The setLED(…) method is used for this purpose. The method belongs to the IUserKey interface.

Syntax

key`.setLED(`*position*`,` *led*`,` *size*`);`

Explanation of the syntax

Element	Description
key	Type: IUserKey User key to which a graphical element is assigned
position	Type: Enum of type UserKeyAlignment Position of the element (grid space) (>>> "UserKey alignment" Page 364)
led	Type: Enum of type UserKeyLED Color of the LED icon ■ **UserKeyLED.Grey**: Gray ■ **UserKeyLED.Green**: Green ■ **UserKeyLED.Yellow**: Yellow ■ **UserKeyLED.Red**: Red
size	Type: Enum of type UserKeyLEDSize Size of the LED icon ■ **UserKeyLEDSize.Small**: Small ■ **UserKeyLEDSize.Normal**: Large

Example

The user key bar for controlling a gripper is to be expanded. The user keys for opening and closing the gripper should each be assigned a small LED icon.

As long as the gripper is opening or closing, the LED icon should be displayed in green. If the gripper is stationary, the LED icon should be displayed in gray.

```
IUserKeyBar gripperBar =
getApplicationUI().createUserKeyBar("Gripper");

IUserKeyListener openGripperListener = new IUserKeyListener(){
@Override
public void onKeyEvent(IUserKey key, UserKeyEvent event) {
   key.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Green,
      UserKeyLEDSize.Small);
   openGripper(); // Method for opening the gripper
   key.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Grey,
      UserKeyLEDSize.Small);
}
};

IUserKeyListener openGripperListener = new IUserKeyListener(){
@Override
public void onKeyEvent(IUserKey key, UserKeyEvent event) {
   key.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Green,
      UserKeyLEDSize.Small);
   closeGripper(); // Method for closing the gripper
   key.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Grey,
      UserKeyLEDSize.Small);
}
};

IUserKeyListener gripperVelocityListener = ...;
```

```
...
IUserKey openKey = ...;
openKey.setText...;
openKey.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Grey,
   UserKeyLEDSize.Small);

IUserKey closeKey = ...;
closeKey.setText...;
closeKey.setLED(UserKeyAlignment.BottomMiddle, UserKeyLED.Grey,
   UserKeyLEDSize.Small);

IUserKey velocityKey = ...;
```

15.26.5 Identifying safety-critical user keys

Description

User keys can trigger functions that are associated with a risk. In order to prevent damage caused by the unintentional actuation of such user keys, a warning message can be added identifying them as safety-critical. The setCriticalText(…) method is used for this purpose. The method belongs to the IUserKey interface.

If the operator actuates a user key designated as safety-critical, the message defined with setCriticalText(…) is displayed on the smartHMI in a window with the name **Critical operation**. The user key is then deactivated for approx. 5 s. Once this time has elapsed, the operator can trigger the desired function by actuating the user key again within 5 s.

If the user key is not actuated within this time or if an area outside of the **Critical operation** window is touched, the window is closed and the user key is reset to its previous state.

Syntax

key`.setCriticalText("`*text*`");`

Explanation of the syntax

Element	Description
key	Type: IUserKey User key which is provided with a warning message
text	Type: String Message text displayed when the user key is actuated

Example

The user key bar for controlling a gripper is to be expanded. If the user key for opening the gripper is actuated, a warning message should appear. The user is requested to ensure that no damage can result from workpieces falling out when the gripper is opened.

```
IUserKeyBar gripperBar =
getApplicationUI().createUserKeyBar("Gripper");
...
IUserKey openKey = ...;
openKey.setText...;
openKey.setLED...;
openKey.setCriticalText("Gripper opens when key is actuated again.
Ensure that no damage can result from workpieces falling out!");
```

15.26.6 Publishing a user key bar

Description

Once a user key bar has been equipped with all the necessary user keys and functionalities, it must be published with the publish() method. Only then can the operator access it on the smartPAD.

Once a user key bar has been published, further user keys may not be added later in the program sequence. In other words, it is not possible to add an unassigned user key and assign a function to it at a later time. It is, however, possible to change the labeling or graphical element displayed next to the user key on the smartHMI at a later time.

Syntax

keybar`.publish();`

Explanation of the syntax

Element	Description
keybar	Type: IUserKeyBar Name of the user key bar created with createUserKeyBar(…).

Example

The user key bar for controlling a gripper is published.

```
IUserKeyBar gripperBar =
getApplicationUI().createUserKeyBar("Gripper");
...
gripperBar.publish();
```

15.27 Message programming

15.27.1 Programming user messages

Description

It is possible to program notification, warning and error messages which are displayed on the smartHMI and written to the log file of the application while the application is running. In addition, it is possible to program messages which are not displayed on the smartHMI but are only written to the log file.

It is advisable to only display messages on the smartHMI which are absolutely essential. A too intensive use of the message display can have a negative effect on the runtime of the application and the operation of the smartHMI.

For message output, it is advisable to use only the commands described here and not other logging functionalities, e.g. the Java commands System.out.println(…) or System.err.println(…). If these commands are used, it is not possible to guarantee that the message will be displayed on the smartHMI.

Syntax

Notification message:

`getLogger().info("`*Message text*`");`

Warning message:

`getLogger().warn("`*Message text*`");`

Error message:

`getLogger().error("`*Message text*`");`

Message that is only written to the log file:

`getLogger().fine("`*Message text*`");`

Explanation of the syntax

Element	Description
Message text	Text which is to be displayed on the smartHMI and/or written to the log file

Example

Once the robot has reached an end point, a notification message is to be displayed. If the motion ended with a collision, a warning notification is displayed instead.

```
IMotionContainer motion = lbr.move(lin(getFrame("/P20"))
.breakWhen(collision));

if(motion.getFiredBreakConditionInfo() == null){
   getLogger().info("End point reached.");
   }
else{
   getLogger().warn("Motion canceled after collision!");
}
```

15.27.2 Programming user dialogs

Description

User dialogs can be programmed in an application. These user dialogs are displayed in a dialog window on the smartHMI while the application is being run and require user action.

Different dialog types can be programmed via the method displayModalDialog(…). The following icons are displayed on the smartHMI according to type:

Icon	Type
	INFORMATION Dialog with information of which the user must take note
	QUESTION Dialog with a question which the user must answer
	WARNING Dialog with a warning of which the user must take note
	ERROR Dialog with an error message of which the user must take note

The user answers by selecting a button that can be labeled by the programmer. Up to 12 buttons can be defined.

The application or the background task from which the dialog was called is stopped until the user reacts. How program execution continues can be made dependent on which button the user selects. The method displayModalDialog(…) returns the index of the button which the user selects on the smartHMI. The index begins at “0” (= index of the first button).

Syntax

`getApplicationUI().displayModalDialog(`*Dialog_type*`,` *“Dialog_text”*`,` *“Button_1”*`<,` … *“Button_12”*`>)`

Explanation of the syntax

Element	Description
Dialog_type	Type: Enum of type ApplicationDialogType ■ INFORMATION: The dialog with the information icon is displayed. ■ QUESTION: The dialog with the question icon is displayed. ■ WARNING: The dialog with the warning icon is displayed. ■ ERROR: The dialog with the error icon is displayed.
Dialog_text	Type: String Text which is displayed in the dialog window on the smartHMI
Button_1 … Button_12	Type: String Labeling of buttons 1 … 12 (proceeding from left to right on the smartHMI)

Example

The following user dialog of type QUESTION is to be displayed on the smartHMI:

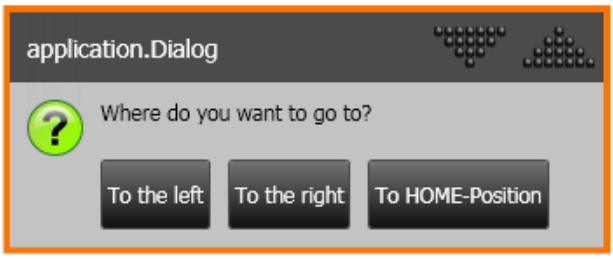


Fig. 15-21: Example of a user dialog

```
int direction = getApplicationUI().displayModalDialog(
ApplicationDialogType.QUESTION, "Where do you want to go to?", "To
the left", "To the right", "To HOME-Position");

switch (direction) {
  case 0:
     lbr.move(ptp(getApplicationData().getFrame("/Left")));
     break;
  case 1:
     lbr.move(ptp(getApplicationData().getFrame("/Right")));
     break;
  case 2:
     lbr.move(ptpHome());
     break;
}
```

15.28 Program execution control

15.28.1 Pausing an application

Description

An application can be paused with the halt() method.

The halt() method pauses the motion currently being executed, and the application state on the smartHMI switches to **Motion paused**.

halt() causes a blocking stop of the calling thread. If further threads are running at the same time, these will continue to be executed. The application execution

is only stopped if halt() is called in the application thread. It is therefore advisable not to call halt() in handling routines for path-related switching actions or in handling routines for monitoring processes. Instead, it is advisable to use the pause() method in these handling routines.

(>>> 15.28.2 "Pausing motion execution" Page 371)

The motion and paused thread may only be resumed via the Start key on the smartPAD. Pressing the Start key causes the paused motion to resume. The paused thread is resumed with the instruction following halt() in the source code.

Syntax

```
getApplicationControl().halt();
```

15.28.2 Pausing motion execution

Description

Motion execution can be paused with the pause() method.

The behavior corresponds to pausing the application via the smartPAD. The pause() method pauses the motion currently being executed, and the application state on the smartHMI switches to **Motion paused**.

pause() does not cause a blocking wait. The application continues to be executed until a synchronous motion command is reached.

Motion execution may only be resumed via the Start key on the smartPAD.

Syntax

```
getApplicationControl().pause();
```

15.28.3 FOR loop

Description

The FOR loop, also called counting loop, repeats a statement block as long as a defined condition is met.

A counter is defined, which is increased or decreased by a constant value with each execution of the loop. At the beginning of a loop execution, the system checks if a defined condition is met. This condition is generally formulated by comparing the counter with a limit value. If the condition is no longer met, the loop is no longer executed and the program is continued after the loop.

The FOR loop is generally used if it is known how often a loop must be executed.

FOR loops can be nested. (>>> 15.28.8 "Examples of nested loops" Page 377)

Syntax

```
for (int Counter = Start value; Condition; Counting statement) {
Statement_1;
<...
Statement_n;
}
```

Explanation of the syntax

Element	Description
Counter	Counter for the number of loops executed. The counter is assigned a start value. With each execution of the loop, the counter is increased or decreased by a constant value.
Start value	Start value of the counter

Element	Description
Condition	Condition for the loop execution The counter is generally compared with a limit value. The result of the comparison is always of type Boolean. The loop is ended as soon as the comparison returns FALSE, meaning that the condition is no longer met.
Counting statement	The counting statement determines the amount by which the counter is changed with each execution of the loop. The increment and counting direction can be specified in different ways. Examples: ■ *Start value* ++|−−: With each execution of the loop, the start value is increased or decreased by a value of 1. ■ *Start value* +|− *Increment*: With each execution of the loop, the start value is increased or decreased by the specified increment.

Example

```
for (int i = 0; i < 10; i++){
   getLogger().info(i);
}
```

The value of the variable `i` is increased by 1 with every cycle. The current value of `i` is displayed on the smartHMI with every cycle. The loop is executed a total of 10 times. The values of 0 to 9 are displayed in the process.

15.28.4 WHILE loop

Description

The WHILE loop repeats a statement block for as long as a certain condition is fulfilled. It is also called a rejecting loop because the condition is checked before every loop execution.

If the condition is no longer met, the statement block of the loop is no longer executed and the program is resumed after the loop. If the condition is not already fulfilled before the first execution, the statement block is not executed at all.

The WHILE loop is generally used if it is unknown how often a loop must be executed, e.g. because the repetition condition is calculated or is a specific signal.

WHILE loops can be nested. (>>> 15.28.8 "Examples of nested loops" Page 377)

Syntax

`while` (*Repetition condition*) {

Statement_1;

<...

Statement_n;

}

Explanation of the syntax

Element	Description
Repetition condition	Type: Boolean Possible: ■ Variable of type Boolean ■ Logic operation, e.g. a comparison, with a result of type Boolean

Example 1

```
while(input1 == true){
   getLogger().info("Input 1 is TRUE.");
}
getLogger().info("Input 1 is FALSE.");
```

Before the loop is executed the system checks whether an input signal is set. As long as this is the case, the loop will be executed again and again and the smartHMI will display the input as TRUE. If the input signal has been reset, the loop will not be executed (any longer) and the input will be displayed as FALSE.

Example 2

```
int w = 0;
Random num = new Random();

while (w <= 21) {
   w = w + (num.nextInt(6) + 1);
}
```

With every loop execution, the value of the variable `w` is increased by a random number between 1 and 6. As long as the sum of all random numbers is less than 21, the loop will be executed. It is not possible to predict the exact number of cycles. It is possible that the loop is ended after 4 cycles (3 x 6 and 1 x 3) or only after 21 cycles (21 x1).

15.28.5 DO WHILE loop

Description

The DO WHILE loop repeats a statement block until a certain condition is fulfilled. It is also called a post-test loop because the condition is only checked after every loop execution.

The statement block is executed at least once. When the condition is met, the loop is terminated and the program is resumed.

The DO WHILE loop is generally used if a loop must be executed at least once, but it is unknown how often e.g. because the break condition is being calculated or is a specific signal.

DO WHILE loops can be nested. (>>> 15.28.8 "Examples of nested loops" Page 377)

Syntax

`do {`

Statement_1;

<...

Statement_n;

`} while (`*Break condition*`);`

Explanation of the syntax

Element	Description
Break condition	Type: Boolean Possible: ■ Variable of type Boolean ■ Logic operation, e.g. a comparison, with a result of type Boolean

Example

```
int num;

do {
    num = (int) (Math.random()*6+1);
} while (num!=6);
```

Random numbers between 1 and 6 are generated until the “dice” shows a 6. The dice must be thrown at least once.

15.28.6 IF ELSE branch

Description

The IF ELSE branch is also called a conditional branch. Depending on a condition, either the first statement block (IF block) or the second statement block (ELSE block) is executed.

The ELSE block is executed if the IF condition is not met. The ELSE block may be omitted. If the IF condition is not met, then no further statements are executed.

It is possible to check further conditions and to link them to statements after the IF block using `else if`. As soon as one of these conditions is met and the corresponding statements are executed, the subsequent branches are no longer checked.

Several IF statements can be nested in each other.

Syntax

```
if (Condition_1){
Statement_1;
<...
Statement_n;
}
<else if (Condition_2){
Statement_1;
<...
Statement_n;
}>
<else {
Statement_1;
<...
Statement_n;
}>
```

Explanation of the syntax

Element	Description
Condition	Type: Boolean Possible: ■ Variable of type Boolean ■ Logic operation, e.g. a comparison, with a result of type Boolean

Example 1

IF branch without `else`

```
int a;
int b;

if (a == 17){
   b = 1;
}
```

If variable `a` has the value 17, variable `b` is assigned the value 1.

Example 2 IF branch within a FOR loop without `else`

```
for(int a = 1; a <= 10; a++){
   if(a == 3){
      a = a + 5;
   }
   getLogger().info(a);
}
```

The loop is executed 5 times. If variable `a` has the value 3, the value of `a` is increased by 5 once only.

The values 1, 2, 8, 9 and 10 are displayed on the smartHMI.

Example 3 IF ELSE branch with `else if`

```
double velAct = 0.0;
double velDesired = 130.0;

...

if (velAct < velDesired) {
   accelerating();
}
else if (velAct > velDesired) {
   braking();
}
else {
   testrun();
}
```

In a program, a test run for a vehicle is to be carried out. This test run is only meaningful at a specific command velocity.

The IF statement checks whether the actual velocity `velAct` is lower than the command velocity `velDesired`. If this is the case, the vehicle accelerates. If this is not the case, it continues with `else if`.

The IF ELSE statement checks whether the actual velocity `velAct` is higher than the command velocity `velDesired`. If this is the case, the vehicle is braked. If this is not the case, the ELSE block is excecuted with the test run.

15.28.7 SWITCH branch

Description The SWITCH branch is also called a multiple branch. Generally, a SWITCH branch corresponds to a multiply nested IF branch.

In a SWITCH block, different CASE blocks can be executed which are designated by CASE labels (jump labels). Depending on the result of an expression, the corresponding CASE block is selected and executed. The program jumps to the CASE label and is resumed at this point.

The keyword `break` at the end of a CASE block means that the SWITCH block is left. If no `break` follows at the end of an instruction block, all subsequent instructions (not only instructions with CASE labels) are executed until either a BREAK label is reached or all instructions have been executed.

A DEFAULT block can optionally be programmed. If no condition is met for jumping to a CASE label, the DEFAULT block is executed.

Syntax

`switch (`*expression*`){`

`case` *Constant_1*`:`

Statement_1`;`

```
<...
Statement_n;>
< break;>
<...
case Constant_n:
Statement_1;>
<...
Statement_n;>
< break;>
< default:
Statement_1;>
<...
Statement_n;>
< break;>
}
```

Explanation of the syntax

Element	Description
Expression	Type: int, byte, short, char, enum
Constant	Type: int, byte, short, char, enum The data type of the constant must match the data type of the expression. **Note**: Constants of type char must be specified with ' , e.g. case 'a'

Example

SWITCH branch with BREAK and DEFAULT instruction:

```
int a, b;
switch (a){
   case 1:
      b = 10;
   case 2:
   case 3:
      b = 20;
      break;
   case 4:
      b = 30
      break;
   default:
      b = 40;
}
```

If variable `a` has the value 1, the program jumps to the label `case 1`. The variable `b` is assigned the value 10.

The program is resumed. No command is executed for the label `case 2`. The program changes to the label `case 3`. The variable `b` is assigned the value 20. If variable `a` has the value 2 or 3, variable `b` is also assigned the value 20. The BREAK instruction causes the SWITCH block to be left.

If variable `a` has the value 4 at the start, variable `b` is assigned the value 30.

The BREAK instruction causes the SWITCH block to be left.

If variable `a` has a different value (e.g. 5), variable `b` is assigned the value 40. The execution of the SWITCH block is ended.

15.28.8 Examples of nested loops

The outer loop is first executed until the inner loop is reached. The inner loop is then executed completely. The outer loop is then executed until the end, and the system checks whether the outer loop must be executed again. If this is the case, the inner loop must also be executed again.

There is no limit on the nesting depth of loops. The inner loops are always executed as often as the outer loop.

FOR in FOR loop

```
for (int i = 1; i < 4; i++) {
   getLogger().info(i + ".Cycle begins");

   for (int k = 10; k > 0; k--) {
      getLogger().info("..." + k);
   }
}
```

The outer loop determines that the inner loop is executed 3 times. The counter of the outer loop starts with the value `i = 1`.

Once the smartHMI has displayed the start of the 1st cycle, the counter of the inner loop starts with the value `k = 10`. The value of variable `k` is decreased by 1 with every cycle. The current value of `k` is displayed on the smartHMI with every cycle. If variable `k` has the value 1, the inner loop will be executed for the last time.

Then the outer loop is ended and the value of variable `i` is increased by 1. The 2nd cycle begins.

FOR in WHILE loop

```
int sum = 0;
int round = 1;
int diceRoll = 0;
Random num = new Random();

while (sum < 21) {
      round ++;

      for (int i = 1; i <= 3; i++){
         diceRoll = (num.nextInt(6) + 1);
         if (diceRoll % 2 == 0)
            sum += diceRoll;
   }
}
```

The following rules apply in a dice game:

- The total sum of all rolls must be at least 21 (poll with WHILE loop).
- The dice are rolled 3 times in each round (FOR loop).
- Only even numbers (2, 4 and 6) are counted (IF poll with modulo).

15.29 Continuing a paused application in Automatic mode (recovery)

Description

If a paused robot application is to be continued in Automatic mode, the higher-level controller must be able to determine whether the robot is still situated on its programmed path. If the robot is no longer situated on the path, e.g. following a non-path-maintaining stop or because it was jogged while the program

was paused, there must be a suitable strategy for automatically repositioning the robot.

This return strategy may only be applied if it can be ensured that there is no risk of a collision while the robot is returning to the path. If this is not ensured, the robot must be manually repositioned by the user.

RoboticsAPI provides the interface IRecovery for automatic repositioning. It is possible to access the interface from robot applications and background tasks:

- IRecovery getRecovery()

Overview

The interface IRecovery provides methods for polling whether robots must be repositioned in order to resume a paused application and which return strategy is applied.

Method	Description
isRecoveryRequired()	Return value type: Boolean Checks whether one or more robots used in the application must be repositioned in a paused application. **true**: At least one robot must be repositioned for the application to be resumed. **false**: The application can be resumed immediately.
isRecoveryRequired(…)	Return value type: Boolean Checks whether a specific robot must be repositioned in a paused application. The robot is transferred as a parameter (type: robot). **true**: The robot must be repositioned for the application to be resumed. **false**: The application can be resumed immediately.
getRecoveryStrategy(…)	Return value type: RecoveryStrategy Polls the strategy being applied in order to return a specific robot to the path. The robot is transferred as a parameter (type: robot). - **PTPRecoveryStrategy**: The robot is repositioned with a PTP motion. The robot is moved at 20% of the maximum possible axis velocity and the effective override. No further strategies are available at this time. The method returns `null` in the following cases: - No return strategy is required or available. - The application is not paused.

PTPRecovery Strategy

The class PTPRecoveryStrategy provides “get” methods which are used to poll the characteristics of the PTP motion. With these methods, it is possible to evaluate whether the return strategy may be carried out in Automatic mode.

Method	Description
getStartPosition()	Return value type: JointPosition Polls for the start position of the PTP motion (= axis position from which the robot can be repositioned) The start position is the currently commanded setpoint position of the robot and not the currently measured actual position.
getMotion()	Return value type: PTP Polls for the PTP motion carried out on execution of the strategy Further information can be polled from the returned motion object: ■ getDestination(): Target position of the PTP motion (= axis position at which the robot left the path) ■ getMode(): Controller mode of the motion which was interrupted

External controller

The robot controller must inform the higher-level controller whether the robot must be repositioned. The higher-level controller may only allow the return strategy to be carried out if this can be done without risk. Otherwise, the robot may only be manually repositioned.

The following system signals are available:

- Output AutExt_AppReadyToStart

  With this output, the robot controller communicates to the higher-level controller whether or not the application may be resumed.

  a. If isRecoveryRequired(…) supplies the value **false** (= no repositioning required), the output can be set to TRUE.
  b. If getRecoveryStrategy(…) supplies `null` (= no return strategy available), the output must be set to FALSE.
  c. If the evaluation of the return strategy shows that it can be executed in Automatic mode, the output can be set to TRUE.

     If this is not the case, the output must be set to FALSE.
- Input App_Start

  The higher-level controller informs the robot controller via a rising edge that the application should resume. (Precondition: AutExt_AppReadyToStart is TRUE)

The higher-level controller must send the start signal App_Start twice:

1. Start signal for repositioning
2. Start signal for resuming the application

15.30 Error treatment

15.30.1 Handling of failed motion commands

Motion commands that are communicated to the robot controller can fail for various reasons, e.g.:

- End point lies outside of a workspace
- End point cannot be reached with the given axis configuration
- The frame used is not present in the application data

A failed motion command results by default in a termination of the application. Handling routines can be defined in order to prevent the application from terminating in case of error.

The following handling options are available depending on the error:

- Failed synchronous motion commands are handled using a try-catch block
- Failed asynchronous motion commands are handled using an event handler

15.30.2 Handling of failed synchronous motion commands

Description

Synchronously executed motion commands (`.move(…);`) are sent in steps to the real-time controller and executed. The further execution of the program is interrupted until the motion has been executed. Only then is the next command sent.

Using a try-catch block, predictable runtime errors or exceptions can be executed in the program sequence without the application being aborted.

A defined method for error treatment is triggered within a try-catch block. When the keyword `try` is called, an attempt is made to execute the listed command. If an error occurs during execution, the corresponding handling routine is started in the catch block.

Syntax

```
try{
// Code in which a runtime error could occur when executed
}
catch(Exception e){
// Code for treating the runtime error
}
< finally{
// Final treatment (optional)
}>
```

Explanation of the syntax

Element	Description
`try{…}`	The try block contains a code which could result in a runtime error. If an error occurs, the execution of the try block is terminated and the catch block is executed.
`catch(…){…}`	The catch block contains the code for treating the runtime error. The catch block will only be executed if an error occurs in the try block.

Element	Description
`Exception e`	The error data type (here: Exception) can be used to define the error type to be handled in the catch block. The error type `Exception` is the superclass of most error data types. However, it is also possible to focus on more specific errors. Information about errors which have occurred can be polled using the parameter `e`. In particular, the error data type CommandInvalidException (package: com.kuka.roboticsAPI.executionModel) is important. It occurs, for example, when the end point of the motion cannot be reached.
`finally {…}`	The finally block is optional. Here it is possible to specify a final treatment to be executed in all cases, whether or not an error occurs in the try block.

Example

A robot executes a motion under impedance control with very low stiffness. For this reason, it is not guaranteed to reach the end position. It is then to move relatively by 50 cm in the positive Z direction of the flange coordinate system. If the robot is in an unfavorable position following the motion under impedance control, the linear motion cannot be executed and a runtime error will occur. In order to prevent the application from aborting in this case, the critical linear motion is programmed in a try-catch block. If the motion planning fails, the robot should be moved to an auxiliary point before the application is resumed.

```
CartesianImpedanceControlMode softMode = new
CartesianImpedanceControlMode();

softMode.parametrize(CartDOF.ALL).setStiffness(10.0);
exampleRobot.move(ptp(getFrame("/Start"))
   .setMode(softMode).setJointVelocityRel(0.3));

try{
   getLogger().info("1: Try to execute linear motion");
   exampleRobot.move(linRel(0.0, 0.0, 500.0)
      .setJointVelocityRel(0.5));
}

catch(CommandInvalidException e){
   getLogger().info("2: Bewegung nicht möglich");
   exampleRobot.move(ptp(getFrame("/Auxiliary point"))
      .setJointVelocityRel(0.5));
}

finally{
   getLogger().info(
      "3: Final treatment in finally block is executed");
}

getLogger().info("4: Further in the program");
```

15.30.3 Handling of failed asynchronous motion commands

Description

In the case of asynchronously executed motion commands (`.moveAsync(…);`), the next program line is executed directly after the motion command is sent.

An event handler is used in order to react to a failed asynchronous motion command.

This event handler is an object of type IErrorHandler and defines the method handleError(…). The transfer of further motion commands to the real-time controller is blocked during execution of the method handleError(…). The application remains at a standstill.

The handling routine is defined with handleError(…). Information on the failed motion command can be accessed via the input parameters of the method. The method returns a parameter of type ErrorHandlingAction. The final reaction to the error is selected via this parameter.

The following reactions are available:

- The application is terminated with an error.
- The motion execution is paused and can only be resumed by the user pressing the Start key on the smartPAD.
- The error is ignored and the application is resumed.

The defined event handler must be registered before it can be used in the application. The method getApplicationControl().registerMoveAsyncErrorHandler(…) is used for this purpose. The method belongs to the IApplicationControl interface.

Syntax

Defining the event handler:

```
IErrorHandler errorHandler = new IErrorHandler(){
@Override
public ErrorHandlingAction handleError
(Device device, IMotionContainer failedContainer,
List<IMotionContainer> canceledContainers){
// Code which is executed in case of error
return ErrorHandlingAction.reaction;
}
};
```

Registering the event handler:

```
getApplicationControl().registerMoveAsyncErrorHandler(errorHandler);
```

Explanation of the syntax

Element	Description
errorHandler	Type: IErrorHandler Name of the event handler responsible for handling failed asynchronous motion commands.
Input parameters of the handleError(…) method:	
`device`	Type: Device The parameter can be used to access the robot for which the failed motion command is commanded.
`failed Container`	Type: IMotionContainer The parameter can be used to access the failed motion command.

Element	Description
canceledContainers	Type: List<IMotionContainer> The parameter can be used to access a list of all deleted motion commands. It contains all motion commands which have already been sent to the real-time controller when the method handleError(…) is called.
reaction	Type: Enum of type ErrorHandlingAction Return value of the method handleError(…) by means of which the final reaction to the error is defined. ■ **ErrorHandlingAction.EndApplication:** The application is terminated with an error. ■ **ErrorHandlingAction.PauseMotion:** The motion execution is paused until the user resumes the application via the smartPAD. ■ **ErrorHandlingAction.Ignore:** The error is ignored and the application is resumed.

Example

Several asynchronous motion commands are to be executed in an application. By registering an event handler of type IErrorHandler, a handling routine is defined using the method handleError(…) for the event that one of the asynchronous motion commands fails:

- The smartHMI displays which motion command has failed.
- The smartHMI displays which motion commands are no longer executed.

The method handleError(…) is ended with the return of the value ErrorHandlingAction.Ignore.

```
public void initialize(){
kuka_Sunrise_Cabinet_1 = getController("kuka_Sunrise_Cabinet_1");
robot = (LBR)getRobot(kuka_Sunrise_Cabinet_1, "LBR_iiwa_14_R820_1");

IErrorHandler errorHandler = new IErrorHandler()
      @Override
      public ErrorHandlingAction handleError(Device device,
            IMotionContainer failedContainer,
            List<IMotionContainer> canceledContainers) {
         getLogger().warn("The following motion command has failed: "
+ failedContainer.toString());
         getLogger().info("The following motion commands are not
executed: ");
         for(int i = 0; i < canceledContainers.size(); i++){
            getLogger().info(canceledContainers.get(i).toString());
         }
         return ErrorHandlingAction.Ignore
      }
   };
   getApplicationControl().
      registerMoveAsyncErrorHandler(errorHandler);
}

public void run(){
...
   robot.moveAsync(ptp(getFrame("/P1")));
   robot.moveAsync(ptp(getFrame("/P2")));

   robot.moveAsync(lin(getFrame("/P3")));
```

```
    robot.moveAsync(ptp(getFrame("/P4")));
    robot.moveAsync(ptp(getFrame("/P5")));
    robot.moveAsync(ptp(getFrame("/P6")));

    robot.moveAsync(ptp(getFrame("/P7")));
    robot.moveAsync(ptp(getFrame("/P8")));
    robot.moveAsync(ptp(getFrame("/P9")));
...
}
```

To explain the system behavior, it is assumed that the linear motion to P3 cannot be planned. This means that the method handleError(…) is called. In our example, the robot is situated at end point P2 at this time.

If, for example the motion commands to P4, P5, P6 are already in the real-time controller at the same time, these motion commands will be deleted and no longer executed.

Calling the method handleError(…) will block further motion commands from being sent to the real-time controller. In this case, the application will be stopped before the motion command to P7. If the method handleError(…) is ended with the return of the value `ErrorHandlingAction.Ignore`, the application is resumed. The robot then moves directly from its current position P2 to P7.

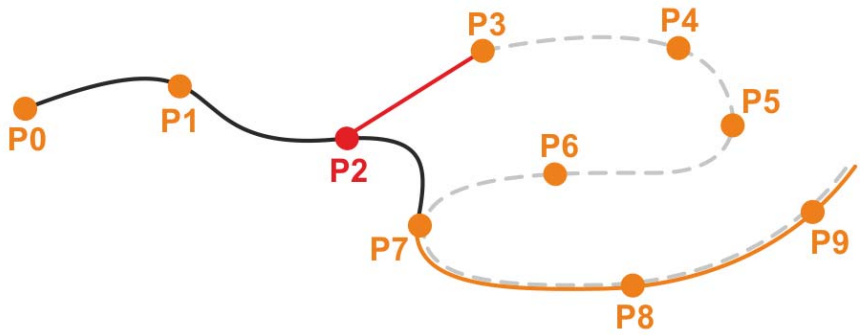


Fig. 15-22: Failed motion to P3 (example of path)

16 Background tasks

16.1 Using background tasks

Activities

Background tasks are used in order to be able to perform tasks in the background, parallel to a running robot application, or to implement cyclical processes that are to be run continuously in the background. Multiple background tasks can run simultaneously and independently of the running robot application.

Background tasks are used, in particular, to control and monitor peripheral devices and to implement the corresponding higher-level logic. Examples:

- Switching signal lamps
- Monitoring and evaluating sensor information

This means that no higher-level controller, e.g. a PLC, is required for smaller applications, as the robot controller can perform such tasks by itself.

> ⚠ In the case of outputs that are switched by a background task, the following points must be observed:
> - The outputs are switched, irrespective of whether a robot application is currently being executed.
> - The outputs are also switched if the robot application is paused due to an EMERGENCY STOP or missing enabling signal.
> - The outputs are also switched if a stop request from the safety controller is active (this also applies if outputs are switched by a robot application).

> **⚠ WARNING** Background tasks must not be used for moving the robot or influencing parameters that might affect motions. This is the task of the robot application. Calling motion commands or modifying motion-specific parameters in a background task can result in unspecified behavior of the robot and thus cause personal injury and damage to property.

Properties

Background tasks, like robot applications, are implemented as Java classes. They are similar in structure to robot applications: they have a run() method that contains the commands to be executed.

Background tasks are an integral feature of the Sunrise project. They are created in Sunrise.Workbench and transferred to the robot controller when the project is synchronized.

(>>> 5.5 "Creating a new background task" Page 52)

There are 2 types of background task that differ in terms of their duration:

- Cyclic background tasks

  Executed cyclically. The cyclical behavior can be adapted by the programmer depending on the task to be performed.
- Non-cyclic background tasks

  Executed once.

Background tasks also differ in terms of their start type:

- **Manual**

  The task must be started manually via the smartPAD. (This function is not yet supported.)
- **Automatic**

  The task is automatically started in the following cases:
  - After the robot controller has booted

- After project synchronization of Sunrise.Workbench on the robot controller

Background tasks that are running are automatically stopped in the following cases:

- During shutdown of the robot controller
- During project synchronization of Sunrise.Workbench on the robot controller
- If an unhandled error occurs during execution of the task

> If a background task has been terminated because of an unhandled error, the task can only be restarted by rebooting the robot controller or carrying out project synchronization of Sunrise.Workbench on the robot controller again.

Every background task has its own instance of type “Controller”. This enables, for example, access to configured I/O groups for controlling peripheral devices or polling robot data.

If a background task requires access to information from the running robot application or from other background tasks, a separate interface is available for data exchange between tasks.

(>>> 16.4 "Data exchange between tasks" Page 389)

16.2 Cyclical background task

Structure

```
(1) package backgroundTask;

(2) import java.util.concurrent.TimeUnit;

(3) public class BackgroundTaskCyclic extends RoboticsAPICyclicBackgroundTask {
        private Controller kuka_Sunrise_Cabinet_1;                    (4)

(5)     public void initialize() {
            kuka_Sunrise_Cabinet_1 = getController("KUKA_Sunrise_Cabinet_1");
            initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
                    CycleBehavior.BestEffort);
        }

(6)     public void runCyclic() {
        }
    }
```

Fig. 16-1: Structure of a cyclic background task

Item	Description
1	This line contains the name of the package in which the task is located.
2	Import section The section contains the imported classes which are required for programming the task
3	Header of the task The cyclic background task is a subclass of RoboticsAPICyclicBackgroundTask.
4	Declaration section When the task is created, one instance of type Controller is automatically created. The controller instance can be used, for example, to generate instances of I/O groups which, in turn, can be used to activate the connected inputs/outputs.

Item	Description
5	initialize() method Here, the data arrays created in the declaration section are assigned initial values. The method initializeCyclic(…) is available by default. initializeCyclic(…) is used to define the cyclical behavior of the task. (>>> "Initialization" Page 387) **Note**: The method must not be deleted or renamed.
6	runCyclic() method The code that is to be executed cyclically is programmed here. **Note**: The method must not be deleted or renamed.

Initialization

initializeCyclic(…) is used to define the cyclical behavior of the background task.

When a cyclical background task is created, the call for initializeCyclic(…) is automatically inserted. The input parameters of the method are assigned initial values, which can lead to the following cyclical behavior:

- Delay: 0 ms
- Period: 500 ms
- Behavior if the defined period is exceeded: execution of runCyclic() continues.

The initial values can be changed by the programmer.

`initializeCyclic(long` *initialDelay*`, long` *period*`, TimeUnit` *timeUnit*`, CycleBehavior` *behavior*`);`

Element	Description
initialDelay	Delay after which the cyclical background task is executed for the first time after the start. All further cycles are executed without a delay. The time unit is defined with *timeUnit*.
period	Period (= time between 2 calls of runCyclic()) The period is maintained even if the execution time of runCyclic() is less than the defined period. The behavior in the event of runCyclic() exceeding the period is defined by *behavior*. The time unit is defined with *timeUnit*.
timeUnit	Time unit of *initialDelay* and *period* The Enum TimeUnit is an integral part of the standard Java library.
behavior	Timeout behavior The behavior of the background task if the period defined with *period* is exceeded by the runtime of runCyclic() is defined here. - **CycleBehavior.BestEffort** runCyclic() is executed completely and then called again. - **CycleBehavior.Strict** Execution of the background task is canceled with an error of type CycleExceededException.

Example

A robot is to assemble workpieces that it takes from a magazine. The magazine can contain a maximum of 100 workpieces and is loaded manually. If the remaining number of workpieces in the magazine falls below 20, this is signaled to the robot controller via a digital input. An LED is then to flash every 500 ms to signal to the operator that the magazine needs filling. Another LED is to flash if the force determined at the robot flange exceeds a limit of 150 N.

A cyclical background task is used for data evaluation and activation of the LEDs. The background task is executed every 500 ms.

```
public class LEDTask extends RoboticsAPICyclicBackgroundTask {
   private Controller _sunrise_Cabinet;
   private LBR _lbr_iiwa;
   private ProcessParametersIOGroup _processParaIOs;
   private ProcessParametersLEDsIOGroup _LED_IOs;

   public void initialize() {
      _sunrise_Cabinet = getController("KUKA_Sunrise_Cabinet_1");
      initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
        CycleBehavior.BestEffort);

      // initialize LBR instance
      getRobot(_sunrise_Cabinet, "LBR_iiwa_7_R800_1");

      // IOs for process parameters
      _processParaIOs = new
      ProcessParametersIOGroup(_sunrise_Cabinet);
      // IOs for controlling LED-signals
      _LED_IOs = new
      ProcessParametersLEDsIOGroup(_sunrise_Cabinet);
   }

   public void runCyclic() {
      // Check if refill is required (value true)
      boolean refillRequired =
      _processParaIOs.getSensor_RefillRequired();

      if(refillRequired){
        /*
         * If refill is required, the appropriate LED changes
         * its state with every execution of runCyclic()
         */
        boolean currentState = _LED_IOs.getLED_RefillRequired();
        _LED_IOs.setLED_RefillRequired(!currentState);
      }
      else{
        /*
         * If refill is not required, the LED remains off
         */
        _LED_IOs.setLED_RefillRequired(false);
      }

      // Query the applied force
      Vector forceVector = _lbr_iiwa.getExternalForceTorque(
      _lbr_iiwa.getFlange()).getForce();
      // Calculate the absolute force value
      double forceValue = forceVector.length();
      // Check if force exceeds 150N
      if(forceValue > 150.0){
        /*
         * If the force limit is exceeded, the appropriate LED
```

```
         * changes its state with every execution of runCyclic()
         */
        boolean currentStateForceLED =
        _LED_IOs.getLED_ForceExceeded();
        _LED_IOs.setLED_ForceExceeded(!currentStateForceLED);
      } else{
          _LED_IOs.setLED_ForceExceeded(false);
        }
    }
}
```

16.3 Non-cyclic background task

Structure

```
(1) package backgroundTask;

(2) import com.kuka.roboticsAPI.applicationModel.tasks.RoboticsAPIBackgroundTask;

(3) public class BackgroundTask extends RoboticsAPIBackgroundTask {
      private Controller kuka_Sunrise_Cabinet_1;                    (4)

(5)   public void initialize() {
          kuka_Sunrise_Cabinet_1 = getController("KUKA_Sunrise_Cabinet_1");
      }

(6)   public void run() {
      }
    }
```

Fig. 16-2: Structure of a non-cyclic background task

Item	Description
1	This line contains the name of the package in which the task is located.
2	Import section The section contains the imported classes which are required for programming the task
3	Header of the task The non-cyclic background task is a subclass of RoboticsAPI-BackgroundTask.
4	Declaration section When the task is created, one instance of type Controller is automatically created. The controller instance can be used, for example, to generate instances of I/O groups which, in turn, can be used to activate the connected inputs/outputs.
5	initialize() method Here, the data arrays created in the declaration section are assigned initial values. **Note**: The method must not be deleted or renamed.
6	run() method The code that is to be executed once is programmed here. The runtime is not limited. **Note**: The method must not be deleted or renamed.

16.4 Data exchange between tasks

Description

The mechanism described here can be used to exchange data between running tasks. One task can provide task functions (providing task) that can be

accessed by other tasks (requesting tasks). It is thus possible, for example, in a background task to access information from the running robot application.

It is not relevant for programming whether data are exchanged between a background task and a robot application or between 2 background tasks. It is also irrelevant whether the robot application or the background task is the providing task. For this reason, background tasks and robot applications are grouped together as tasks.

Overview

The following steps are required in order for the providing task and the requesting task to be able to communicate with one another:

Step	Description
1	Create an interface derived from the type ITaskFunction (com.kuka.roboticsAPI.applicationModel.tasks) and declare the desired task functions. (>>> 16.4.1 "Declaring task functions" Page 390)
2	Create a class that implements the interface derived from ITaskFunction and program the declared task functions. (>>> 16.4.2 "Implementing task functions" Page 392) **Note**: The interface can be implemented directly by the providing task or alternatively by a purpose-created class.
3	Create the providing task. The providing task must implement the interface ITaskFunctionProvider and announce that it is providing the task functions defined in the interface derived from ITaskFunction. (>>> 16.4.3 "Creating the providing task" Page 393)
4	In the requesting task, fetch an object of type ITaskFunctionAccessor using a static method. This object is used to access the task functions. For the ITaskFunctionAccessor instance, the interface derived from ITaskFunction in which the required task functions are declared must be specified. (>>> 16.4.4 "Using task functions" Page 395)

Example

The data exchange between tasks is described step by step in the sections below, using the following example:

An assembly process is to be implemented using the robot application “AssemblyApplication”. An LED is to flash during the assembly process. If the robot leaves the path during the application and has to be repositioned, a further LED is to flash.

The LEDs are activated by the background task “LEDTask”. In this example, the background task is the requesting task.

The robot application is the providing task. It must enable access to your Recovery interface, which is used to check whether repositioning of the robot is required. Furthermore, it must also signal the start and end of the assembly process.

16.4.1 Declaring task functions

Description

The task functions are declared via an interface derived from the type ITaskFunction.

> The interface may only declare those methods that are to be made available to the requesting task. It is thus advisable not to use set methods for setting fields in the first interface. Instead, such methods can be offered by the implementing class.

Example

Declaration of the task functions via the interface IApplicationInformationFunction

The method for setting the Recovery interface is not to be available for the requesting task and is thus not declared by the interface.

```
1  public interface IApplicationInformationFunction extends
2      ITaskFunction {
3
4     /**
5      * Signifies whether assembly is currently executed
6      * @return true, if assembly is executed
7      */
8     public boolean isAssemblyRunning();
9     /**
10     * Called from application when assembly is
11     * started and finished
12     * @param assembly Set to true when assembly is started.
13     * Reset when assembly is stopped.
14     */
15    public void setAssemblyRunning(boolean assembly);
16
17    /**
18     * Returns whether the application requires
19     * repositioning of the robot
20     * @return true if repositioning is required
21     */
22    public boolean isApplicationRecoveryRequired();
23    /**
24     * Called from application to give access to its
25     * recovery interface
26     * @param applicationRecoveryInterface Recovery
27     * interface of the application
28     */
29 }
```

Line	Description
1 … 29	Interface IApplicationInformationFunction
8	Method isAssemblyRunning() Called by the requesting task to poll whether the assembly process is currently running.
15	Method setAssemblyRunning(…) Called by the robot application when the assembly process is started or ended.
22	Method isApplicationRecoveryRequired() Called by the requesting task to poll whether repositioning of the robot is required.

16.4.2 Implementing task functions

Description

A class must be made available that implements the interface derived from ITaskFunction and in which the declared task functions are programmed. The providing task or a purpose-created class can be used for this.

Example

Implementation of the interface IApplicationInformationFunction using the class ApplicationInformation

The robot application requires a method for making its Recovery interface available. This method is declared and implemented not by the interface IApplicationInformationFunction, but by the class ApplicationInformation.

```
1  public class ApplicationInformation implements
2     IApplicationInformationFunction {
3     private boolean _assembly;
4     private IRecovery _applicationRecoveryInterface;
5
6     @Override
7     public boolean isAssemblyRunning() {
8        return _assembly;
9     }
10
11    @Override
12    public void setAssemblyRunning(boolean assembly) {
13       _assembly = assembly;
14    }
15
16    @Override
17    public boolean isApplicationRecoveryRequired() {
18       return _applicationRecoveryInterface.isRecoveryRequired();
19    }
20
21    @Override
22    public void setApplicationRecoveryInterface(
23       IRecovery applicationRecoveryInterface) {
24       _applicationRecoveryInterface =
25       applicationRecoveryInterface;
26    }
27 }
```

Line	Description
1 … 27	Class ApplicationInformation The task functions are programmed in the class.
3, 4	Declaration of the data fields ■ _assembly: saves the current status of the assembly process ■ _applicationRecoveryInterface: refers to the Recovery interface of the robot application
6 … 9	Method isAssemblyRunning() Called by the requesting task to poll whether the assembly process is currently running.
11 … 14	Method setAssemblyRunning(…) Called by the robot application if the assembly process is started or ended.

Line	Description
16 … 19	Method isApplicationRecoveryRequired() Called by the requesting task to poll whether repositioning of the robot is required.
21 … 26	Method setApplicationRecoveryInterface(…) Called by the robot application for making its Recovery interface available.

16.4.3 Creating the providing task

A task can provide task functions of multiple different interfaces derived from ITaskFunction.

In order for a task to be able to provide task functions, a number of adaptations must be made to the task class.

Annotation

The providing task must announce that it is providing the task functions defined in the interface derived from ITaskFunction.

For this, the following annotation is inserted above the class header of the robot application:

`@ProvidedFunctions(`*ITaskFunctionType*`.class)`

- *ITaskFunctionType*: Name of the interface derived from ITaskFunction. The task provides the task functions defined by this interface.

If a task provides task functions from multiple interfaces derived from ITaskFunction, the data types must be specified as an array:

`@ProvidedFunctions({`*ITaskFunctionType_1*`.class,`
ITaskFunctionType_2`.class, ...})`

> Each interface derived from ITaskFunction may only be provided once. This means that there must not be 2 tasks that specify the same interface in their @ProvidedFunctions annotation.

Example:

The robot application “AssemblyApplication” provides the task functions declared by the interface IApplicationInformationFunction. This is announced by the annotation above the class header of the robot application:

```
@ProvidedFunctions(IApplicationInformationFunction.class)

public class AssemblyApplication ...
```

Interface

The providing task must implement the interface ITaskFunctionProvider and provide the method createTaskFunctions():

```
public Map<Class<? extends ITaskFunction>, ITaskFunction>
createTaskFunctions(){
...
}
```

> If the providing task does not, itself, implement the interface derived from ITaskFunction, it requires an instance of the implementing class. It is advisable to create this instance as an array.

The method createTaskFunctions() returns an object of type “map” that contains all ITaskFunction instances of the providing task. The map instance is generated by the following command and can be adopted in the source code without the need for further adaptation:

```
Map<Class<? extends ITaskFunction>, ITaskFunction> map
= new HashMap<Class<? extends ITaskFunction>, ITaskFunc-
tion>();
```

- *map*: Designation of the map instance

The map command put(…) is used to save the ITaskFunction instances in the map object. The type of the interface derived from ITaskFunction and the corresponding ITaskFunction instance are transferred to the command:

`map.put(`*iTaskFunctionType*`.class, `*value*`);`

- *iTaskFunctionType*: Designation of the interface derived from ITaskFunction and containing the declared task functions.
- *value*: ITaskFunction instance of the providing task. The data type of *value* is defined by the parameter *iTaskFunctionType*`.class`.

> If the providing task implements the interface itself, the instance of the task (reference `this`) is transferred for the parameter *value*:
> `map.put(`*iTaskFunctionType*`.class, this)`

Overall example

The robot application contains a data array of type ApplicationInformation. Its method setApplicationRecoveryInterface(…) provides the Recovery interface of the robot application. Calling the method setAssembly(…) announces that the assembly process is being carried out.

```
// Annotation, required for all tasks which provide task functions
@ProvidedFunctions(IApplicationInformationFunction.class)

public class AssemblyApplication extends RoboticsAPIApplication
implements ITaskFunctionProvider {
   private Controller _sunrise_Cabinet;
   private LBR _lbr_iiwa;

   /**
    * ITaskFunction instance with which
    * the task's functions are provided
    */
   private ApplicationInformation _function;

   public void initialize() {
      _sunrise_Cabinet = getController("KUKA_sunrise_Cabinet_1");
      _lbr_iiwa = (LBR) getRobot(_sunrise_Cabinet,
      "LBR_iiwa_7_R800_1");

      _function = new ApplicationInformation();
      // Gives access to recovery interface
      _function.setApplicationRecoveryInterface(getRecovery());
   }

   public void run() {
      // Moves robot to initial pose
      _lbr_iiwa.move(ptp(getFrame("/StartPos")));
      // Announces that assembly is running
      _function.setAssemblyRunning(true);
      assembly();
      // Announces that assembly is finished
      _function.setAssemblyRunning(false);

      // Moves robot to initial pose
      _lbr_iiwa.move(ptp(getFrame("/StartPos")));
   }
```

```
    /**
     * Implements the assembly process
     */
    private void assembly() {
        ...
    }

    public static void main(String[] args) {
        AssemblyApplication app = new AssemblyApplication();
        app.runApplication();

    }
    /**
     * ITaskFunctionProvider method that has to be
     * implemented by the task
     */
    @Override
    public Map<Class<? extends ITaskFunction>, ITaskFunction>
    createTaskFunctions() {
        // Creation of the map containing all ITaskFunction instances
        Map<Class<? extends ITaskFunction>, ITaskFunction> map = new
        HashMap<Class<? extends ITaskFunction>, ITaskFunction>();
        // Fill the map with functions and corresponding interface
        map.put(IApplicationInformationFunction.class, _function);
        // Return map which references the ITaskFunction instances
        return map;
    }
}
```

16.4.4 Using task functions

Task functions that provide a task can be used by other tasks.

Enabling access

Access to the task functions is carried out in the requesting task by an object of type ITaskFunctionAccessor. When the ITaskFunctionAccessor instance is generated, the interface derived from ITaskFunction declaring the required task functions must be specified. The following command is used for this:

`ITaskFunctionAccessor<` *iTaskFunctionType*`>` *accessor* `= getTaskFunction(`*iTaskFunctionType*`.class);`

- *iTaskFunctionType*: Designation of the interface derived from ITaskFunction and containing the declared task functions.
- *accessor*: Variable of type `ITaskFunctionAccessor<`*iTaskFunctionType*`.class>` that refers to the generated instance.

Example:

In the requesting background task “LEDTask”, access to the functions defined by IApplicationInformationFunction is to be enabled. The ITaskFunctionAccessor<IApplicationInformationFunction> instance required for this is created as a data array and generated in the initialize() method of the task:

```
public class LEDTask extends RoboticsAPICyclicBackgroundTask {
    ...
    private ITaskFunctionAccessor<IApplicationInformationFunction>
    _accessor;

    public void initialize() {
        ...
        _accessor =
```

```
        getTaskFunction(IApplicationInformationFunction.class);
        ...
```

Using functions

An ITaskFunctionAccessor instance enables use of the methods of the associated interface derived from ITaskFunctions. The ITaskFunctionAccessor method get() provides access to the ITaskFunction instance of the providing task. The dot operator then makes the individual methods of the interface available.

iTaskFunctionType taskFunction = *accessor*.`get();`

- *iTaskFunctionType*: Designation of the interface derived from ITaskFunction and containing the declared task functions.
- *taskFunction*: Variable of the type of the interface
- *accessor*: Variable of type `ITaskFunctionAccessor<`*iTaskFunctionType*`>` that can be used to access the task functions defined by the interface.

Example:

In the method runCyclic() of the background task “LEDTask”, polling is to be carried out to ascertain whether the assembly process is currently being executed. For this, the interface IApplicationInformationFunction offers the method isAssemblyRunning().

The methods of the interface can be called directly after the call of get():

```
if(_accessor.get().isAssemblyRunning()){
    ...
}
```

Checking availability

The task functions of the providing task are only available if an instance of this task exists. For example, the task functions provided by a robot application are only available if the robot application is executed or paused.

> If an attempt is made to access unavailable task functions using the ITaskFunctionAccessor instance, this results in a runtime error in the requesting task. If this error is not handled, execution of the task is aborted.

The data ITaskFunctionAccessor offers the following methods for checking whether the providing task is available:

Method	Description
isAvailable()	Return value type: Boolean Specifies whether the task functions of the providing task are available (**true** = available).
await(*time*, *unit*)	Return value type: Boolean If the task functions are not available when the providing task is called, the system waits a defined time for them to become available (**true** = task functions available within the defined time). Parameters: ■ *time* (typ: long): duration of maximum wait time. The unit is defined by the parameter *unit*. ■ *unit* (type: TimeUnit): unit of *time*

Example:

The requesting background task “LEDTask” can only poll whether the assembly process is being executed if the robot application is running or paused. For

this reason, the availability of the function must be checked before isAssemblyRunning() is called:

```
if(_accessor.isAvailable()){
   if(_accessor.get().isAssemblyRunning()){
   ...
   }
}
```

Overall example

The requesting task “LED Task” is executed cyclically every 500 ms. It first checks whether the required task functions of the robot application are available. If they are available, polling is carried out to ascertain whether the assembly process is running and the corresponding LED is activated. The system then polls whether repositioning of the robot is required. If this is the case, a further LED is activated.

```
public class LEDTask extends RoboticsAPICyclicBackgroundTask {
   private Controller _sunrise_Cabinet;
   private ProcessParametersLEDsIOGroup _LED_IOs;

   /**
    * ITaskFunctionAccessor for functions defined by
    * IApplicationInformationFunction
    */
   private ITaskFunctionAccessor<IApplicationInformationFunction>
   _accessor;

   public void initialize() {
      _sunrise_Cabinet = getController("KUKA_Sunrise_Cabinet_1");
      initializeCyclic(0, 500, TimeUnit.MILLISECONDS,
        CycleBehavior.BestEffort);

      // Create ITaskFunctionAccessor for IApplicationInformation
      _accessor =
      getTaskFunction(IApplicationInformationFunction.class);

      // IOs for controlling LED-signals
      _LED_IOs = new ProcessParametersLEDsIOGroup(_sunrise_Cabinet);
   }

   public void runCyclic() {

      // Check if task functions are available
      if(_accessor.isAvailable()){
        /*
         * Use task function to check if assembly is
         * currently executed
         */
         if(_accessor.get().isAssemblyRunning()){
           /*
            * If assembly is running, the appropriate LED changes
            * its state with every execution of runCyclic()
            */
           boolean currentStateAssemblyLED =
           _LED_IOs.getLED_Assembly();
           _LED_IOs.setLED_Assembly(!currentStateAssemblyLED);

         } else{
           _LED_IOs.setLED_Assembly(false);
         }
```

```
        /*
         * Use task function to check whether the application
         * requires repositioning
         */
        boolean recoveryRequired =
        _accessor.get().isApplicationRecoveryRequired();

        if(recoveryRequired){
          /*
           *If recovery is required, the appropriate LED changes
           * its state with every execution of runCyclic()
           */
          boolean currentStateRecoveryLED =
          _LED_IOs.getLED_RecoveryRequired();
          _LED_IOs.setLED_ForceExceeded(!currentStateRecoveryLED);
        } else{
          _LED_IOs.setLED_RecoveryRequired(false);


      } else{
        // If application is not running, LEDs remain off
        _LED_IOs.setLED_Assembly(false);
        _LED_IOs.setLED_RecoveryRequired(false);
      }
    }
}
```

17 Programming with a compliant robot

17.1 Sensors and control

Without additional equipment, a standard industrial robot can only be operated under position control. The aim of position control is to keep the difference between the specified and actual robot position as small as possible at all times.

Apart from position sensors for determining the current joint position, the KUKA LBR iiwa also has joint torque sensors in every axis, which allow the current joint torques to be measured. These data enable the use of an impedance controller in addition to position control, thus making it possible to implement compliant behavior of the robot. The underlying model is a virtual spring damper system with configurable values for stiffness and damping. Furthermore, additional forces and force oscillations can be overlaid.

The special sensor technology and the available controller mechanisms make the KUKA LBR iiwa highly sensitive and compliant. This enables it to react very quickly to process forces and makes it particularly suitable for a wide range of joining tasks and for interaction with humans.

17.2 Available controllers – overview

The KUKA LBR iiwa can be operated with a number of different controllers. For each control type, a separate class is provided by the RoboticsAPI in the package com.kuka.roboticsAPI.motionModel.controlModeModel. The shared superclass is AbstractMotionControlMode.

Controller	Description
Position controller	Data type: PositionControlMode The aim of position control is to execute the specified path with the maximum possible positional accuracy and without path deviation. By default, external influences such as obstacles or process forces are not taken into account.
Cartesian impedance controller	Data type: CartesianImpedanceControlMode The Cartesian impedance controller is modeled on a virtual spring damper system with configurable values for stiffness and damping. This spring is extended between the setpoint and actual positions of the TCP. This allows the robot to react in a compliant manner to external influences.
Cartesian impedance controller with overlaid force oscillation	Data type: CartesianSineImpedanceControlMode Special form of the Cartesian impedance controller. In addition to the compliant behavior, constant force setpoints and sinusoidal force oscillations can be overlaid. This controller can be used to implement force-dependent search runs and vibration motions for joining processes, for example.
Axis-specific impedance controller	Data type: JointImpedanceControlMode The axis-specific impedance controller is modeled on a virtual spring damper system with configurable values for stiffness and damping for each axis.

17.3 Using controllers in robot applications

Description

In robot applications, the controller to be used is set separately for every motion command. The following steps are required by default for this:

Procedure

1. Create the controller object of the desired controller data type.
2. Parameterize the controller object to define the control response.
3. Set the controller as the motion parameter for a motion command.

17.3.1 Creating a controller object

Description

To be able to use a controller, a variable of the desired controller data type must first be created and initialized. By default, the controller object is generated using the standard constructor.

Syntax

Controller mode controlMode;

controlMode = `new` *Controller mode*();

Explanation of the syntax

Element	Description
Controller mode	Data type of the controller. Subclass of AbstractMotionControlMode.
controlMode	Name of the controller object

Example

Creating a Cartesian impedance controller:

```
CartesianImpedanceControlMode cartImpCtrlMode;
cartImpCtrlMode = new CartesianImpedanceControlMode();
```

17.3.2 Defining controller parameters

The parameters that can be set depend on the type of the controller used. The individual controller classes in the RoboticsAPI provide specific “set” and “get” methods for each parameter.

(>>> 17.5.2 "Parameterization of the Cartesian impedance controller" Page 403)

(>>> 17.6.3 "Parameterization of the impedance controller with overlaid force oscillation" Page 411)

(>>> 17.8 "Axis-specific impedance controller" Page 421)

17.3.3 Transferring the controller object as a motion parameter

Description

The controller object is transferred to a motion as a parameter using the command setMode(…). If no controller object is transferred as a parameter to a motion, the motion is automatically executed with position control.

> Motions which use the Cartesian impedance controller must not contain any poses in the proximity of singularity positions.

Syntax

movableObject.move(*motion*.setMode(*controlMode*));

Explanation of the syntax

Element	Description
motion	Type: Motion Motion to be executed
controlMode	Type: Subclass of AbstractMotionControlMode Name of the controller object

17.4 Position controller

With position control, the motors are controlled in such a way that the current position of the robot always matches the setpoint position specified by the controller with just a minimal difference. The position controller is particularly suitable in cases where precise positioning is required.

The position controller is represented by the class PositionControlMode. The data type has no configurable parameters for adapting the robot.

If the controller mode of a motion is not explicitly specified, then the position controller is used.

17.5 Cartesian impedance controller

The Cartesian impedance controller is represented by the class CartesianImpedanceControlMode.

The impedance controller refers by default to the coordinate system with which the motion command is executed.

Examples:

- robot.move(…);

  The impedance controller refers to the flange coordinate system of the robot.
- gripper.move(…);

  The impedance controller refers to the tool coordinate system currently used for the gripper or to the standard frame defined for gripper motions.
- gripper.getFrame("/TipCenter").move(...);

  The impedance controller refers to the tool coordinate system that extends from the “TipCenter” frame on the gripper.

Behavior of the robot

Under impedance control, the robot’s behavior is compliant. It is sensitive and can react to external influences such as obstacles or process forces. The application of external forces can cause the robot to leave the planned path.

The underlying model is based on virtual springs and dampers, which are stretched out due to the difference between the currently measured and the specified position of the TCP. The characteristics of the springs are described by stiffness values, and those of the dampers are described by damping values. These parameters can be set individually for every translational and rotational dimension.

> If the robot is moved under impedance control, the programmed robot configuration, e.g. the status value, cannot be guaranteed.

17.5.1 Calculation of the forces on the basis of Hooke’s law

If the measured and specified robot positions correspond, the virtual springs are slack. As the robot’s behavior is compliant, an external force or a motion command results in a deviation between the setpoint and actual positions of the robot. This results in a deflection of the virtual springs, leading to a force in accordance with Hooke’s law.

The resultant force F can be calculated on the basis of Hooke’s law using the set spring stiffness C and the deflection Δx:

$F = C \cdot \Delta x$

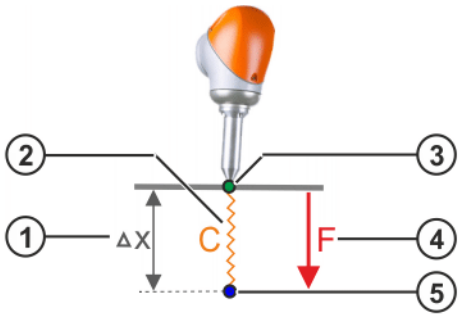


Fig. 17-1: Virtual spring with spring stiffness C

1	Deflection Δx	4	Resulting force F
2	Virtual spring	5	Command position
3	Actual position		

If the robot is at a resistance, it exerts the calculated force. If it is positioned in free space, it moves toward the setpoint position; due to internal friction forces in the joints, path deviations occur here too, whose magnitude depends on the set spring stiffness. Higher stiffness values lead to smaller deviations.

If the robot is already at the setpoint position and an external force is applied to the system, the robot yields to this force until the forces resulting from compliance control cancel out the external forces.

Examples

The force exerted at the contact point depends on the difference between the setpoint position and the actual position and the set stiffness.

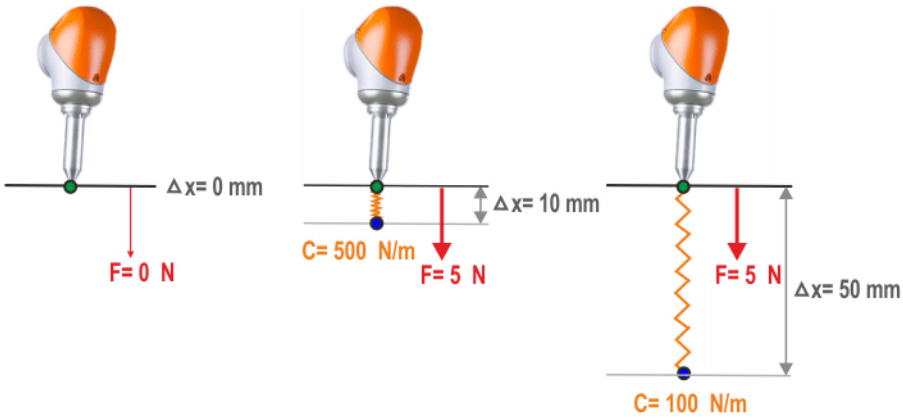


Fig. 17-2: Force exerted on contact

As shown in the figure (>>> Fig. 17-2), a large position difference and low stiffness can result in the same force as a smaller position difference and greater stiffness. If the force is increased by a motion in a contact situation, the time required to reach this force differs if the Cartesian velocity is identical.

If higher stiffness values are used, a desired force can be reached earlier, as only a small position difference is required. Since the setpoint position is reached quickly, a jerk can be produced in this way.

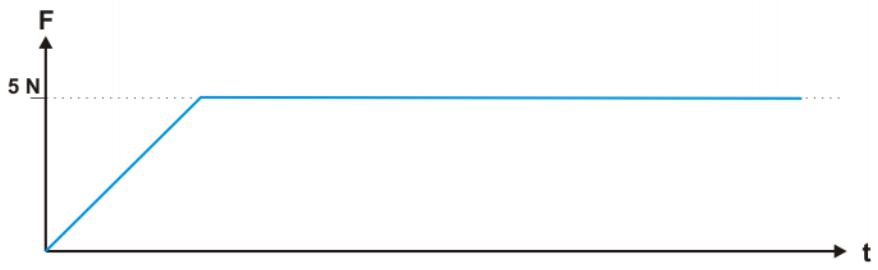


Fig. 17-3: Force over time (high stiffness, small position difference)

In the case of a large position difference and low stiffness, the force is built up more slowly. This can be used, for example, if the robot moves to the contact point and the impact loads are to be reduced.

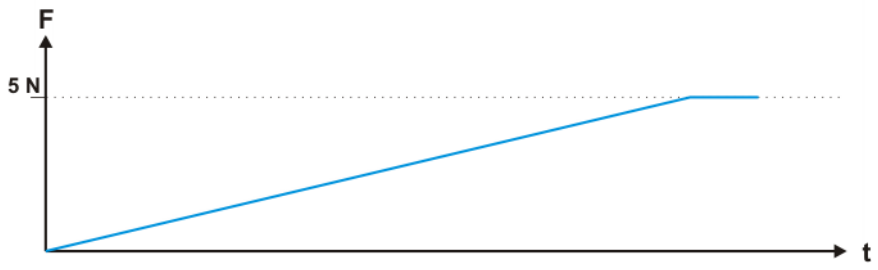


Fig. 17-4: Force over time (low stiffness, large position difference)

Setpoint/actual deviations in more than one direction lead to deflection of all the affected virtual springs. The magnitude and direction of the overall force results from vector addition of the individual forces for each direction.

The deflection in the x direction by Δx and in the y direction by Δy result in force F_x in the x direction and F_y in the y direction. The vector addition results in the overall force F_{res}.

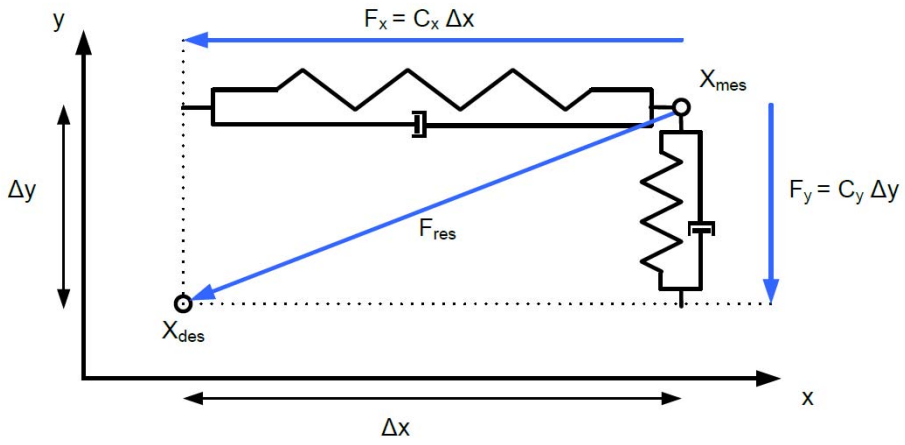


Fig. 17-5: Overall force in the case of deflection in 2 directions

17.5.2 Parameterization of the Cartesian impedance controller

Under impedance control, the robot behaves like a spring. The characteristics of this spring are defined by different parameters. This results in the behavior of the robot.

With a Cartesian impedance controller, forces can be overlaid for all Cartesian degrees of freedom. Forces acting about an axis generate a torque. For this reason, the overlaid torque and not the overlaid force is specified for the rota-

tional degrees of freedom. For the sake of simplification, the terms “force” and “force oscillation” are taken to include the terms “torque” and “torque oscillation” for the rotational degrees of freedom in the following text.

⚠ CAUTION In impedance control, inaccurate sensor information or incorrectly selected parameters (e.g. incorrect load data, incorrect tool) can be interpreted as external forces, resulting in unpredictable motions of the robot.

The following controller properties can be defined individually for each Cartesian degree of freedom:

- Stiffness
- Damping
- Force to be applied in addition to the spring

The following controller properties can be defined irrespective of the degree of freedom:

- Stiffness of the redundancy degree of freedom
- Damping of the redundancy degree of freedom
- Limitation of the maximum force on the TCP
- Maximum Cartesian velocity
- Maximum Cartesian path deviation

17.5.2.1 Representation of Cartesian degrees of freedom

In the RoboticsAPI, the degrees of freedom of the Cartesian impedance controller are represented by the Enum CartDOF (package: com.kuka.roboticsAPI.geometricModel). The values of this Enum can be used to describe either each degree of freedom individually or the combination of a number of degrees of freedom.

Enum value	Description
CartDOF.X	Translational degree of freedom in the X direction
CartDOF.Y	Translational degree of freedom in the Y direction
CartDOF.Z	Translational degree of freedom in the Z direction
CartDOF. TRANSL	Combination of the translational degrees of freedom in the X, Y and Z directions
CartDOF.A	Rotational degree of freedom about the Z axis
CartDOF.B	Rotational degree of freedom about the Y axis
CartDOF.C	Rotational degree of freedom about the X axis
CartDOF.ROT	Combination of rotational degrees of freedom about the X, Y and Z axes
CartDOF.ALL	Combination of all Cartesian degrees of freedom

17.5.2.2 Defining controller parameters for individual degrees of freedom

Description

Some parameters of the Cartesian impedance controller can be defined individually for each Cartesian degree of freedom.

During programming, the Cartesian degree of freedom for which the controller parameter is to apply is specified first. The parametrize(…) method of the controller data types is used for this purpose. To define the degree of freedom, one or more parameters of the type CartDOF are transferred to this method.

After this, the “set” method of the desired controller parameter is called via the point operator. This controller parameter is set to the value specified as the in-

put parameter of the set method for all degrees of freedom specified in parametrize(…).

Syntax

controlMode`.parametrize(CartDOF.`*degreeOfFreedom_1*
`<, CartDOF.`*degreeOfFreedom_2*`,…>).set`*Parameter*`(`*value*`);`

Explanation of the syntax

Element	Description
controlMode	Type: CartesianImpedanceControlMode Name of the controller object
degreeOfFreedom_1, *degreeOfFreedom_2*, …	Type: CartDOF List of degrees of freedom to be described
`set`*Parameter*`(`*value*`)`	Method for setting a controller parameter A separate method is available for each settable *parameter* (*value* = value of the parameter).

Example

A LIN motion is to be executed to a defined point under impedance control. The Cartesian impedance controller is configured in such a way that the currently used TCP – here the robot flange frame – is compliant in the Z direction.

```
CartesianImpedanceControlMode cartImpCtrlMode = new
CartesianImpedanceControlMode();

cartImpCtrlMode.parametrize(CartDOF.X,
CartDOF.Y).setStiffness(3000.0);
cartImpCtrlMode.parametrize(CartDOF.Z).setStiffness(1.0);
cartImpCtrlMode.parametrize(CartDOF.ROT).setStiffness(300.0);
cartImpCtrlMode.parametrize(CartDOF.ALL).setDamping(0.7);

lbr.move(lin(getApplicationData().getFrame("/
P1")).setCartVelocity(800).setMode(cartImpCtrlMode));
```

17.5.2.3 Controller parameters specific to the degrees of freedom

Overview

The following methods are available for the parameters of the Cartesian impedance controller that are specific to the degrees of freedom:

Method	Description
setStiffness(…)	Spring stiffness (type: double) The spring stiffness determines the extent to which the robot yields to an external force and deviates from its planned path. Translational degrees of freedom (unit: N/m): ■ **0.0 … 5000.0** Default: 2000.0 Rotational degrees of freedom (unit: Nm/rad): ■ **0.0 … 300.0** Default: 200.0 **Note**: If no spring stiffness is specified for a degree of freedom, the default value is used for this degree of freedom.
setDamping(…)	Spring damping (type: double) The spring damping determines the extent to which the virtual springs oscillate after deflection. For all degrees of freedom (without unit: Lehr's damping ratio): ■ **0.1 … 1.0** Default: 0.7 **Note**: If no spring damping is specified for a degree of freedom, the default value is used for this degree of freedom.
setAdditionalControl-Force(…)	Force applied in addition to the spring (type: double) The additional force results in a Cartesian force at the TCP. This force acts in addition to the forces resulting from the spring stiffness. Translational degrees of freedom (unit: N): ■ Negative and positive values possible. Default: 0.0 Rotational degrees of freedom (unit: Nm): ■ Negative and positive values possible. Default: 0.0 **Note**: If no additional force is specified for a degree of freedom, the default value is used for this degree of freedom. **Note**: The force is overlaid without a delay. If the force to be overlaid is too great, this can result in overloading of the robot and cancelation of the program. The class CartesianSineImpedanceControlMode has the option of overlaying forces after a delay.

17.5.2.4 Controller parameters independent of the degrees of freedom

Some settings apply irrespective of the Cartesian degrees of freedom. The set methods used to define these controller parameters belong to the class CartesianImpedanceControlMode and are called directly on the controller object.

Overview

The following methods are available for the parameters of the Cartesian impedance controller that are independent of the degrees of freedom:

Method	Description
setNullSpaceStiffness(…)	Spring stiffness of the redundancy degree of freedom (type: double, unit: Nm/rad) The spring stiffness determines the extent to which the robot yields to an external force and deviates from its planned path. ■ **≥ 0.0** **Note**: If no spring stiffness is specified for the redundancy degree of freedom, a default value is used for this degree of freedom.
setNullSpaceDamping(…)	Spring damping of the redundancy degree of freedom (type: double) The spring damping determines the extent to which the virtual springs oscillate after deflection. ■ **0.3 … 1.0** **Note**: If no spring damping is specified for the redundancy degree of freedom, a default value is used for this degree of freedom.

setMaxControlForce(…)	Limitation of the maximum force on the TCP
	
	The maximum force applied to the TCP by the virtual springs is limited. The maximum force required to deflect the virtual spring is thus also defined. Whether or not the motion is to be aborted if the maximum force at the TCP is exceeded is also defined.
	
	Syntax:
	
	■ `setMaxControlForce(`*maxForceX*`,` *maxForceY*`,` *maxForceZ*`,` *maxTorqueA*`,` *maxTorqueB*`,` *maxTorqueC*`,` *addStopCondition*`)`
	
	Explanation of the syntax:
	
	■ *maxForceX*|*Y*|*Z*: Maximum force at the TCP in the corresponding Cartesian direction (type: double, unit: N)
	  ■ **≥ 0.0**
	■ *maxTorqueA*|*B*|*C*: Maximum torque at the TCP in the corresponding rotational direction (type: double, unit: Nm)
	  ■ **≥ 0.0**
	■ *addStopCondition*: Cancelation of the motion if the maximum force at the TCP is exceeded (type: boolean)
	  ■ **true**: Motion is aborted.
	  ■ **false**: Motion is not aborted.

Method	Description
setMaxCartesianVelocity(…)	Maximum Cartesian velocity The motion is aborted if the defined velocity limit is exceeded. **Syntax**: ■ `setMaxCartesianVelocity(`*maxVelocityX*`,` *maxVelocityY*`,` *maxVelocityZ*`,` *maxVelocityA*`,` *maxVelocityB*`,` *maxVelocityC*`)` **Explanation of the syntax**: ■ *maxVelocityX*|*Y*|*Z*: Maximum permissible translational velocity at the TCP in the corresponding Cartesian direction (type: double, unit: mm/s) ▪ **≥ 0.0** ■ *maxVelocityA*|*B*|*C*: Maximum permissible rotational velocity at the TCP in the corresponding rotational direction (type: double, unit: rad/s) ▪ **≥ 0.0**
setMaxPathDeviation(…)	Maximum Cartesian path deviation Defines the maximum permissible Cartesian path deviation from the currently planned setpoint position for a compliant motion. The motion is aborted if the defined maximum path deviation is exceeded. **Syntax**: ■ `setMaxPathDeviation(`*maxDeviationX*`,` *maxDeviationY*`,` *maxDeviationZ*`,` *maxDeviationA*`,` *maxDeviationB*`,` *maxDeviationC*`)` **Explanation of the syntax**: ■ *maxDeviationX*|*Y*|*Z*: Maximum permissible path deviation at the TCP in the corresponding Cartesian direction (type: double, unit: mm) ▪ **≥ 0.0** ■ *maxDeviationA*|*B*|*C*: Maximum permissible rotational deviation at the TCP in the corresponding rotational direction (type: double, unit: rad/s) ▪ **≥ 0.0**

Example 1

A robot under impedance control is to be compliant in its redundant degree of freedom in order to be able to respond to obstacles during the motion. For this, stiffness and damping of the redundant degree of freedom are parameterized for the impedance controller.

```
CartesianImpedanceControlMode mode = new
CartesianImpedanceControlMode();

mode.setNullSpaceStiffness(10.0);
mode.setNullSpaceDamping(0.7);
```

Example 2

A robot is to move along a table plate in compliant mode. A Cartesian impedance controller is parameterized for this. A high stiffness value is set for the Z direction of the tool coordinate system in the TCP. An additional force of 20 N is also to be applied. The motion is aborted if a force limit of 50 N in the Z direction is exceeded. A low stiffness value is set in the XY plane. The Cartesian deviation in the X and Y directions must not exceed 1 cm, however. Suitable higher values are specified for all other parameters.

```
CartesianImpedanceControlMode mode = new
CartesianImpedanceControlMode();

mode.parametrize(CartDOF.Z).setStiffness(3000.0);
mode.parametrize(CartDOF.Z).setAdditionalControlForce(20.0);
```

```
mode.setMaxControlForce(100.0, 100.0, 50.0, 20.0, 20.0, 20.0, true);

mode.parametrize(CartDOF.X, CartDOF.Y).setStiffness(10.0);
mode.setMaxPathDeviation(10.0, 10.0, 50.0, 2.0, 2.0, 2.0);
```

17.6 Cartesian impedance controller with overlaid force oscillation

The Cartesian impedance controller with overlaid force oscillation is a special form of the Cartesian impedance controller. The force can be overlaid separately for each Cartesian degree of freedom.

Force oscillations about an axis generate torque oscillations. Overlaying torque oscillations can result in the generation of rotational oscillations.

Overlaying constant or sinusoidal forces causes the robot to move. Suitable combinations of oscillations in the individual degrees of freedom can be used to generate different motion patterns.

Using overlaid oscillations, it is possible to implement compliant pendulum motions for search runs and vibrations in the tool for joining processes.

The Cartesian impedance controller with overlaid force oscillation is represented by the class CartesianSineImpedanceControlMode.

Behavior of the robot

In this form of impedance control, the overlaid force causes the robot to leave the planned path in a targeted way. The new path is thus determined by a wide range of different parameters.

In addition to stiffness and damping, further parameters can be defined, e.g. frequency and amplitude. The programmed velocity of the robot also plays a significant role for the actual path.

> ⚠ Overlaying additional forces has a strong influence on the robot motion and the forces exerted by the robot. For example, low stiffness and high overlaid forces can cause the robot to accelerate suddenly. Parameterization must therefore be carried out with caution if working with force activations. For example, begin by overlaying low forces and approach the appropriate force values step by step. In addition, the motion resulting from the overlaid force must always be tested in T1 mode first.

17.6.1 Overlaying a simple force oscillation

By overlaying a simple force oscillation, the working point is diverted from the planned path (= path without overlaid oscillations) and is instead moved from the start point to the end point of the motion in a sinusoidal path.

Example

The robot executes a relative motion in the Y direction of the tool coordinate system in the TCP. A sinusoidal force oscillation in the X direction is overlaid. The result is a wave-like path in the XY plane of the coordinate system.

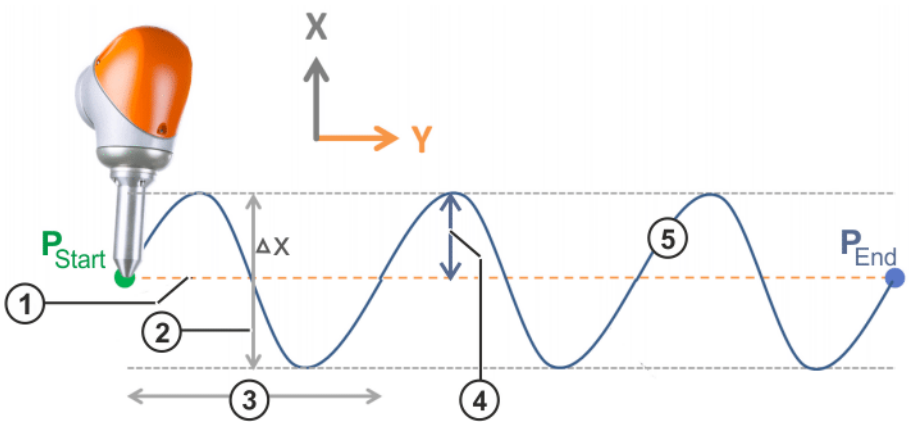


Fig. 17-6: Overlaying a simple force oscillation

1	Original path	4	Amplitude
2	Deflection Δx	5	New path
3	Wavelength		

The maximum deflection Δx is the deviation from the original path in the positive and negative X directions. The maximum deflection is determined by the stiffness and amplitude which are defined for the impedance controller in the Cartesian X direction, e.g.:

- Cartesian stiffness: C = 500 N/m
- Amplitude: F = 5 N

The maximum deflection results from Hooke's law:

Δx = F / C = 5 N / (500 N/m) = 1 / (100 1/m) = 1 cm

The wavelength can be used to determine how many oscillations the robot is to execute between the start point and end point of the motion. The wavelength is determined by the frequency which is defined for the impedance controller with overlaid force oscillation, as well as by the programmed robot velocity.

Wavelength λ is calculated as follows:

λ = c / f = robot velocity / frequency

17.6.2 Overlaying superposed force oscillations (Lissajous curves)

Lissajous curves result when a sinusoidal force oscillation is overlaid in 2 different Cartesian directions. The superposition of the two oscillations makes it possible to create very different forms for the path. The exact path depends on a number of parameters.

Application

Two sinusoidal force oscillations of different frequencies can be superposed to generate vibrations at the TCP. For example, such vibrations can remove tension and jamming which occur during an assembly process.

Example

A sinusoidal force oscillation is overlaid in both the X and Y directions of the tool coordinate system in the TCP. The maximum deflections Δx and Δy are determined by the stiffness and amplitude, which are defined for the impedance controller in the Cartesian X and Y directions.

In addition to the known parameters of the impedance controller, the phase offset between the two oscillations plays a significant role in the path.

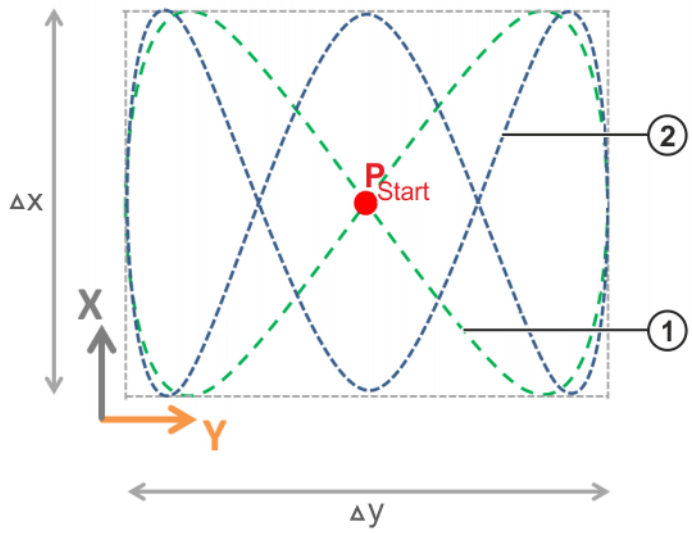


Fig. 17-7: Path of a Lissajous curve

1 Path without phase offset (frequency ratio X:Y = 2:1)
2 Path with phase offset (frequency ratio X:Y = 3:1)

The form of the path is mainly determined by the ratio of the two frequencies and the phase offset between the two oscillations. The resulting curve is always axisymmetric and point-symmetric. The set power amplitude and stiffness for an oscillation direction results in its position amplitude. The ratio between the two position amplitudes determines the ratio between the width to the height of the curve.

17.6.3 Parameterization of the impedance controller with overlaid force oscillation

The Cartesian impedance controller with overlaid force oscillation is a special form of the standard impedance controller.

With a Cartesian impedance controller with overlaid force oscillation, forces can be overlaid for all Cartesian degrees of freedom. Forces acting about an axis generate a torque. For this reason, the overlaid torque and not the overlaid force is specified for the rotational degrees of freedom. For the sake of simplification, the terms “force” and “force oscillation” are taken to include the terms “torque” and “torque oscillation” for the rotational degrees of freedom in the following text.

⚠ CAUTION In impedance control, inaccurate sensor information or incorrectly selected parameters (e.g. incorrect load data, incorrect tool) can be interpreted as external forces, resulting in unpredictable motions of the robot.

The Cartesian impedance controller with overlaid force oscillation is parameterized in the same way as the standard impedance controller. The controller parameters specific to the degrees of freedom and the controller parameters independent of the degrees of freedom as described for the standard impedance controller can be used in the same way for the impedance controller with overlaid force oscillation.

(>>> 17.5.2 "Parameterization of the Cartesian impedance controller" Page 403)

> Exception: The setAdditionalControlForce(…) method of the class CartesianImpedanceControlMode for overlaying a force to be applied in addition to the spring is available for the class CartesianSineImpedanceControlMode, but should not be used.
> The setBias(…) method is available for overlaying constant forces in the class CartesianSineImpedanceControlMode.

The following additional controller properties can be defined individually for each Cartesian degree of freedom:

- Amplitude of the force oscillation
- Frequency of the force oscillation
- Phase offset of the force oscillation
- Superposed constant force
- Force limitation of the force oscillation
- Limitation of the deflection due to the force oscillation

The following additional controller properties can be defined irrespective of the degree of freedom:

- Rise time of the force oscillation
- Hold time of the force oscillation
- Fall time of the force oscillation
- Overall duration of the force oscillation

17.6.3.1 Controller parameters specific to the degrees of freedom

Overview

The following methods are available for the parameters of the Cartesian impedance controller with overlaid force oscillation that are specific to the degrees of freedom:

Method	Description
setAmplitude(…)	Amplitude of the force oscillation (type: double) Amplitude and stiffness determine the position amplitude. Translational degrees of freedom (unit: N): ■ **≥ 0.0** Default: 0.0 Rotational degrees of freedom (unit: Nm): ■ **≥ 0.0** Default: 0.0 **Note**: If no amplitude is specified for a degree of freedom, the default value is used for this degree of freedom.
setFrequency(…)	Frequency of the force oscillation (type: double; unit: Hz) Frequency and Cartesian velocity determine the wavelength of the force oscillation. ■ **0.0 … 15.0** Default: 0.0 **Note**: If no frequency is specified for a degree of freedom, the default value is used for this degree of freedom.

Method	Description
setPhaseDeg(…)	Phase offset of the force oscillation at the start of the force overlay (type: double; unit: °) ■ **≥ 0.0** Default: 0.0 **Note**: If no phase offset is specified for a degree of freedom, the default value is used for this degree of freedom.
setBias(…)	Constant force overlaid (type: double) Using setBias(…), a constant force can be overlaid in addition to the overlaid force oscillation. This force adds to the force resulting from the spring stiffness and defined force oscillation. If a constant force is overlaid without an additional force oscillation, this results in a force characteristic which rises as a function of the rise time defined with setRiseTime(…) and then remains constant. setRise-Time(…) belongs to the controller parameters independent of the degrees of freedom (>>> 17.6.3.1 "Controller parameters specific to the degrees of freedom" Page 412). If a constant force is overlaid in addition to a force oscillation, the force oscillation is offset in the defined direction. Translational degrees of freedom (unit: N): ■ Negative and positive values possible. Default: 0.0 Rotational degrees of freedom (unit: Nm): ■ Negative and positive values possible. Default: 0.0 **Note**: If no additional constant force is overlaid for a degree of freedom, the default value is used for this degree of freedom.

Method	Description
setForceLimit(…)	Force limitation of the force oscillation (type: double) Defines the limit value that the overall force, i.e. the sum of the amplitude of the force oscillation and additionally overlaid constant force, must not exceed. If the overall force exceeds the limit value, the overlaid force is reduced to the limit value. Translational degrees of freedom (unit: N): ■ **≥ 0.0** Default: Not limited. Rotational degrees of freedom (unit: Nm): ■ **≥ 0.0** Default: Not limited. **Note**: If no force limit is specified for a degree of freedom, the default value is used for this degree of freedom.
setPositionLimit(…)	Maximum deflection due to the force oscillation (type: double) If the maximum permissible deflection is exceeded, the force is deactivated. The force is reactivated as soon as the robot is back in the permissible range. Translational degrees of freedom (unit: mm): ■ **≥ 0.0** Default: Not limited. Rotational degrees of freedom (unit: rad): ■ **≥ 0.0** Default: Not limited. **Note**: If no maximum deflection is specified for a degree of freedom, the default value is used for this degree of freedom.

Example

During a joining process, an oscillation about the Z axis of the tool coordinate system in the TCP is to be generated. The Cartesian impedance controller with overlaid force oscillation is used for this. With a stiffness of 10 Nm/rad and an amplitude of 15 Nm, the position amplitude is approx. 1.5 rad. The frequency is set to 5 Hz. In order to exert an additional pressing force in the direction of motion, a constant force of 5 N is generated in the Z direction and superposed on the overlaid force oscillation about the Z axis.

```
CartesianSineImpedanceControlMode sineMode = new
CartesianSineImpedanceControlMode();

sineMode.parametrize(CartDOF.Z).setStiffness(4000.0);
sineMode.parametrize(CartDOF.Z).setBias(5.0);

sineMode.parametrize(CartDOF.A).setStiffness(10.0);
sineMode.parametrize(CartDOF.A).setAmplitude(15.0);
sineMode.parametrize(CartDOF.A).setFrequency(5.0);

tool.getFrame("/TCP").move(linRel(0.0, 0.0,
10.0).setCartVelocity(10.0).sineMode(sineMode));
```

17.6.3.2 Controller parameters independent of the degrees of freedom

Some settings apply irrespective of the Cartesian degrees of freedom. The set methods used to define these controller parameters belong to the class Car-

tesianSineImpedanceControlMode and are called directly on the controller object.

Overview

The following methods are available for the parameters of the Cartesian impedance controller with overlaid force oscillation that are independent of the degrees of freedom:

Method	Description
setTotalTime(…)	Overall duration of the force oscillation (type: double; unit: s) (>>> "Overall duration of the force oscillation" Page 415) ■ **≥ 0.0** Default: Unlimited
setRiseTime(…)	Rise time of the force oscillation (type: double; unit: s) ■ **≥ 0.0** Default: 0.0 **Note**: If no rise time is specified for a degree of freedom, the default value is used. This means that the amplitude rises abruptly to the defined value without a transition. If the force to be overlaid is too great, this can result in overloading of the robot and cancelation of the program.
setHoldTime(…)	Hold time of the force oscillation (type: double; unit: s) ■ **≥ 0.0** Default: Unlimited **Note**: If no hold time is specified for a degree of freedom, the default value is used. This means that the overlaid force oscillation ends with the corresponding motion.
setFallTime(…)	Fall time of the force oscillation (type: double; unit: s) ■ **≥ 0.0** Default: 0.0 **Note**: If no fall time is specified for a degree of freedom, the default value is used. This means that the amplitude falls abruptly to zero without a transition. If the drop in force is too great, this can result in overloading of the robot and cancelation of the program.
setStayActiveUntil-PatternFinished(…)	Response if the motion duration is exceeded (type: boolean) If the force oscillation lasts longer than the motion, it is possible to define whether the oscillation is terminated or continued after the end of the motion. ■ **true**: Oscillation is continued after the end of the motion. ■ **false**: Oscillation is terminated at the end of the motion. Default: false **Note**: If the response when the motion duration is exceeded is not specified, the default value is used.

Overall duration of the force oscillation

The overall duration is the sum of the rise time, hold time and fall time of the force oscillation:

- Rise time
  Time in which the amplitude of the force oscillation is built up.
- Hold time
  Time in which the force oscillation is executed with the defined amplitude.

- Fall time

  Time in which the amplitude of the force oscillation is reduced back to zero.

Rise time, hold time and fall time of the force oscillation can be defined individually, or indirectly by defining the overall duration of the force oscillation.

If the overall duration is defined using setTotalTime(…), the rise time and fall time are defined automatically.

Calculation:

- Rise time = fall time = (1/frequency) 0.5
- Of the frequencies defined for the force oscillation (relative to all degrees of freedom), the frequency that results in the largest possible rise and fall times is used for the calculation.
- If exclusively constant forces are overlaid, the frequency of all degrees of freedom is 0.0 Hz. Rise and fall time are set to 0.0 s.
- If the calculated sum of rise time and fall time exceeds the defined overall duration, the rise time and fall time are each set to 25% of the overall duration and the hold time to 50%.

If the overall duration of the force oscillation is shorter than the duration of the corresponding motion, the force oscillation ends before the end of the motion. The response if the motion duration is exceeded is defined using setStayActiveUntilPatternFinished(…).

17.7 Static methods for impedance controller with superposed force oscillation

Overview

The Cartesian impedance controller with overlaid force oscillation can also be configured via static methods of the class CartesianSineImpedanceControlMode. This simplifies the programming, in particular of Lissajous curves, as the user only has to specify a few parameters. The remaining parameters which are important for the implementation are calculated and set automatically. Default values are used for all other parameters. Additional settings are made as described using the parametrize(…) function and the set methods of CartesianSineImpedanceControlMode.

- createDesiredForce(…): Static method for constant force
- createSinePattern(…): Static method for simple force oscillations
- createLissajousPattern(…): Static method for Lissajous curves
- createSpiralPattern(…): Static method for spirals

Specification of Cartesian planes

In contrast to simple oscillations, no individual degree of freedom is transferred to Lissajous curves and spirals, but rather the plane in which the path is to run. The plane is specified via the Enum CartPlane (the package com.kuka.roboticsAPI.geometricModel).

Enum value	Description
CartPlane.XY	Path in the XY plane
CartPlane.XZ	Path in the XZ plane
CartPlane.YZ	Path in the YZ plane

17.7.1 Overlaying a constant force

Description

The createDesiredForce(…) method overlays a constant force, that does not change over time, in one Cartesian direction.

Syntax

controlMode `= CartesianSineImpedanceControlMode.createDesiredForce(CartDOF.`*degreeOfFreedom*`,` *force*`,` *stiffness*`);`

Explanation of the syntax

Element	Description
controlMode	Type: CartesianSineImpedanceControlMode Name of the controller object
degreeOfF-reedom	Type: CartDOF Degree of freedom for which the constant force is to be overlaid.
force	Type: double Value of the overlaid constant force. Corrsponds to the call of setBias(…) for the specified degree of freedom. Translational degrees of freedom (unit: N): ■ **≥ 0.0** Rotational degrees of freedom (unit: Nm): ■ **≥ 0.0**
stiffness	Type: double Stiffness value for the specified degree of freedom Translational degrees of freedom (unit: N/m): ■ **0.0 … 5000.0** Rotational degrees of freedom (unit: Nm/rad): ■ **0.0 … 300.0**

17.7.2 Overlaying a simple force oscillation

Description

The createSinePattern(…) method overlays a simple force oscillation in one Cartesian direction.

Syntax

controlMode `= CartesianSineImpedanceControlMode.create-SinePattern(CartDOF.`*degreeOfFreedom*`,` *frequency*`,` *amplitude*`,` *stiffness*`);`

Explanation of the syntax

Element	Description
controlMode	Type: CartesianSineImpedanceControlMode Name of the controller object
degreeOfF-reedom	Type: CartDOF Degree of freedom for which the force oscillation is to be overlaid.
frequency	Type: double Frequency of the oscillation (unit: [Hz]) ■ **0.0 … 15.0**

Element	Description
amplitude	Type: double Amplitude of the oscillation which is overlaid in the direction of the specified degree of freedom Translational degrees of freedom (unit: N): ■ **≥ 0.0** Rotational degrees of freedom (unit: Nm): ■ **≥ 0.0**
stiffness	Type: double Stiffness value for the specified degree of freedom Translational degrees of freedom (unit: N/m): ■ **0.0 … 5000.0** Rotational degrees of freedom (unit: Nm/rad): ■ **0.0 … 300.0**

Example

From the current position, a relative motion of 15 cm is to be executed in the Y direction. The motion is to run in a wave path with a deflection of approx. 10 cm (derived from the amplitude and stiffness) and a frequency of 2 Hz in the X direction.

```
CartesianSineImpedanceControlMode sineMode;

sineMode =
CartesianSineImpedanceControlMode.createSinePattern(CartDOF.X, 2.0,
50.0, 500.0);

lbr.move(linRel(0.0, 150.0,
0.0).setCartVelocity(100).setMode(sineMode));
```

17.7.3 Overlaying a Lissajous oscillation

Description

The createLissajousPattern(…) method is used to generate a 2-dimensional oscillation in one plane. The plane is transferred as a value of type CartPlane. The other transferred parameters refer to the first degree of freedom of the specified plane (example: for CartPlane.XY, the specified values are relative to CartDOF.X).

The parameters of the second degree of freedom of the plane are calculated to produce a Lissajous curve with the following characteristics:

- Amplitude ratio, 1st degree of freedom : 2nd degree of freedom: 1 : 1
- Frequency ratio, 1st degree of freedom : 2nd degree of freedom: 1 : 0.4
- Phase offset between 1st and 2nd degree of freedom: ½ · pi

Syntax

controlMode `= CartesianSineImpedanceControlMode.createLissajousPattern(CartPlane.`*plane*`,` *frequency*`,` *amplitude*`,` *stiffness*`);`

Explanation of the syntax

Element	Description
controlMode	Type: CartesianSineImpedanceControlMode Name of the controller object
plane	Type: Enum of type CartPlane Plane in which the Lissajous oscillation is to be overlaid

Element	Description
frequency	Type: double Frequency of the oscillation for the first degree of freedom of the specified plane (unit: Hz) ■ **0.0 … 15.0** The frequency for the second degree of freedom is calculated as follows: ■ *frequency* · 0.4
amplitude	Type: double Amplitude of the oscillation for both degrees of freedom of the specified plane (unit: N) ■ **≥ 0.0**
stiffness	Type: double Stiffness values for both degrees of freedom of the specified plane (unit: N/m) ■ **0.0 … 5000.0**

Example

An oscillation in the form of a Lissajous curve with a frequency ratio X : Y of 1 : 0.4 and a phase offset in Y of pi/2 is to be generated on the robot flange. Path without phase offset (= blue line (>>> Fig. 17-7)).

```
CartesianSineImpedanceControlMode lissajousMode;

lissajousMode =
CartesianSineImpedanceControlMode.createLissajousPattern(CartPlane.XY
, 10.0, 50.0, 500.0);

lbr.move(linRel(0.0, 150.0,
0.0).setCartVelocity(100).setMode(lissajousMode));
```

17.7.4 Overlaying a spiral-shaped force oscillation

Description

The createSpiralPattern(…) method is used to generate a spiral-shaped force oscillation in one plane.

The force characteristic is created by overlaying 2 sinusoidal force oscillations. The oscillations are shifted in phase by π/2 (90°). The amplitudes of the oscillations rise constantly up to the defined value and then return to zero. This results in a spiral pattern which extends up to the defined amplitude value and then contracts again.

In the resulting robot motion, the TCP moves along this spiral. The Cartesian extent of the spiral depends on the values defined for stiffness and amplitude as well as any obstacles present.

The plane in which the spiral-shaped oscillation is to be overlaid is transferred as a value of type CartPlane. The values defined for the parameters stiffness, frequency and amplitude are identical for both degrees of freedom of the plane.

In addition, a value is transferred for the total time of the force oscillation. The time is divided evenly between the upward and downward motion of the oscillation:

Rise time = Total time / 2

Hold time = 0

Fall time = Total time / 2

Syntax

controlMode = `CartesianSineImpedanceControlMode.createSpiralPattern(CartPlane.`*plane*`,` *frequency*`,` *amplitude*`,` *stiffness*`,` *totalTime*`);`

Explanation of the syntax

Element	Description
controlMode	Type: CartesianSineImpedanceControlMode Name of the controller object
plane	Type: Enum of type CartPlane Plane in which the spiral-shaped oscillation is to be overlaid
frequency	Type: double Frequency of the oscillation for both degrees of freedom of the specified plane (unit: N) ■ **0.0 … 15.0**
amplitude	Type: double Amplitude of the oscillation for both degrees of freedom on the specified plane (unit: N) ■ **≥ 0.0**
stiffness	Type: double Stiffness values for both degrees of freedom of the specified plane (unit: N/m) ■ **0.0 … 5000.0**
totalTime	Type: double Total time for the spiral-shaped oscillation. The time is divided evenly between the upward and downward motion of the oscillation (unit: s). ■ **≥ 0.0**

Example

At the current position of the robot flange, a spiral-shaped force oscillation is to be overlaid in the XY plane of the flange coordinate system. The force is to rise helically up to a maximum value of 100 N. Once per second, the force characteristic is to turn around the start point of the spiral (frequency of the force oscillation: 1.0 Hz). The force spiral must rise and fall within 10 seconds.

```
CartesianSineImpedanceControlMode spiralMode;
spiralMode =
CartesianSineImpedanceControlMode.createSpiralPattern(CartPlane.XY,
1.0, 100, 500, 10);
lbr.move(positionHold(spiralMode, 10, TimeUnit.SECONDS));
```

The number of turns is a function of the total time for a turn (t_{period}). The time for a turn corresponds to the duration of an oscillation period, e.g.:

- Frequency of the force oscillation: f = 1.0 Hz
- Total time: t = 10 s

The number of turns is calculated as follows:

$Number_{Turns}$ = Total time / t_{Period} = 10 s / 1 s = 10

t_{Period} = 1 / f = 1 / 1.0 Hz = 1 s

The maximum deflection results from Hooke's law:

Δx = F / C = 100 N / (500 N/m) = 0.2 m = 20 cm

17.8 Axis-specific impedance controller

The axis-specific impedance controller is represented by the class JointImpedanceControlMode. In this control mode, the robot's behavior is compliant.

The underlying model uses virtual springs and dampers. Unlike with the Cartesian impedance controller, however, these springs and dampers are stretched out due to the difference between the currently measured and the specified axis positions. For this reason, singularity positions of the robot have no influence on the impedance behavior.

17.8.1 Parameterization of the axis-specific impedance controller

⚠ CAUTION In impedance control, inaccurate sensor information or incorrectly selected parameters (e.g. incorrect load data, incorrect tool) can be interpreted as external forces, resulting in unpredictable motions of the robot.

If the application is paused with the spring tensioned under impedance control, the motion command is interrupted. When the application is resumed, the spring is tensioned again. This can result in jerky motion of the robot.

The following controller properties can be defined individually for each axis:

- Stiffness
- Damping

17.8.2 Methods of the axis-specific impedance controller

Overview

Method	Description
setStiffness(…)	Spring stiffness (type: double[]; unit: Nm/rad) The axis-specific spring stiffness determines the degree of compliance of an axis when force is applied. ■ **≥ 0.0** **Note**: The spring stiffness must be specified for every axis.
setDamping(…)	Spring damping (type: double[]; without unit: Lehr's damping ratio) The axis-specific spring damping determines the extent to which the virtual springs oscillate after deflection. ■ **0.0 … 1.0** Default: 0.7 **Note**: The spring damping must be specified for every axis.
setStiffness ForAllJoints(…)	Spring stiffness (type: double; unit: Nm/rad) A value determines the degree of compliance of all axes when force is applied. ■ **≥ 0.0**
setDamping ForAllJoints(…)	Spring damping (type: double; without unit: Lehr's damping ratio) A value determines the extent to which the virtual springs in all axes oscillate after deflection. ■ **0.0 … 1.0**

Constructor syntax

`JointImpedanceControlMode` *jointImp* `= new JointImpedanceControlMode(`*A1, A2, ... A7*`);`

Explanation of the syntax

Element	Description
jointImp	Type: JointImpedanceControlMode Name of the controller object
A1 ... A7	Type: double; unit: Nm/rad Axis-specific spring stiffnesses The number of values is dependent on the axis selection (here: 7 axes).

Example 1

7 axes are to be controlled using the axis-specific impedance controller. Initial values for the axis-specific spring stiffnesses are defined in the constructor of the controller. The stiffness for axis A4 is to be modified subsequently. The spring damping is to be identical for all axes.

```
JointImpedanceControlMode jointImp
 = new JointImpedanceControlMode(2000.0, 2000.0, 2000.0, 2000.0,
100.0, 100.0, 100.0);
...
jointImp.setStiffness(2000.0, 2000.0, 2000.0, 1500.0, 100.0, 100.0,
100.0);
jointImp.setDampingForAllJoints(0.5);
```

Example 2

7 axes are to be controlled using the axis-specific impedance controller. Initial values for the axis-specific spring stiffnesses are defined in the constructor of the controller. The spring stiffness and spring damping are subsequently to be identical for all axes.

```
JointImpedanceControlMode jointImp
 = new JointImpedanceControlMode(2000.0, 2000.0, 2000.0, 2000.0,
100.0, 100.0, 100.0);
...
jointImp.setStiffnessForAllJoints(100);
jointImp.setDampingForAllJoints(0.5);
```

17.9 Holding the position under servo control

Description

Using the motion command positionHold(…), the robot can hold its Cartesian setpoint position over a set period of time and remain under servo control.

If the robot is operated in compliance control, it can remove itself from its setpoint position. Whether, how far and in which direction the robot moves from the current Cartesian setpoint position (= position at the start of the command positionHold(…)) depends on the set controller parameters and the resulting forces. In addition, the compliant robot under servo control can be forced off its setpoint position by external forces.

Syntax

object`.move(positionHold(`*controlMode, time, unit*`));`

Explanation of the syntax

Element	Description
controlMode	Type: Subclass of AbstractMotionControlMode Name of the controller object

Element	Description
time	Type: long Indicates how long the specified *controlMode* is to be held. The value must be >= 0. A value of < 0 indicates infinite.
unit	Type: Enum of type TimeUnit Unit of the specified time. The Enum TimeUnit is an integral part of the standard Java library.

Example

The robot is to be held in its current position for 10 seconds. During this time, the robot is switched to “soft” mode in the Cartesian X direction.

```
CartesianImpedanceControlMode controlMode = new
CartesianImpedanceControlMode();

controlMode.parametrize(CartDOF.X).setStiffness(1000.0);
controlMode.parametrize(CartDOF.ALL).setDamping(0.7);

lbr.move(positionHold(controlMode, 10, TimeUnit.SECONDS));
```

18 Diagnosis

18.1 Field bus diagnosis

> WorkVisual can be used for precise error analysis. More information on field bus diagnosis with WorkVisual is contained in the **WorkVisual** documentation.

> If the robot controller is used as a PROFINET master or device, hardware problems can result in an inability to access bus devices. In this case, use of a diagnostic tool, such as WorkVisual, Step 7 or WireShark, is recommended.

18.1.1 Displaying general field bus errors

Description

The general error state of the connected field buses can be displayed on the smartHMI.

Procedure

1. Open Station view.
2. Select the **KUKA_Sunrise_Cabinet** tile.

   The status indicator of the **Field buses** tile indicates the collective state of all field buses connected to the controller.
3. Select the **Field buses** tile.

   The detail view opens with error information about the currently connected field buses.

18.1.2 Displaying the error state of I/Os and I/O groups

Description

The status indicator in the **I/O groups** area of the navigation bar of the smartHMI displays the state of the configured I/O groups.

- The lower indicator shows the collective state of all configured I/O groups.
- The upper indicator shows the state of the selected I/O group.

Procedure

- In the navigation bar, select the desired I/O group from **I/O groups**.

  The detail view of the I/O group opens. Any faulty inputs/outputs are indicated.

> If PROFINET is used and errors occur at individual terminals, not only the affected inputs/outputs, but all configured PROFINET I/Os are marked as faulty.

18.2 Displaying the protocol

A protocol of the events and changes in state of the system can be displayed on the smartHMI.

Procedure

1. Open the Station view or the Robots view.
2. Select the **Protocol** tile. The **Protocol** view opens.

   If the view is opened via the Robots view, only those protocol entries are displayed by default which affect the robot selected in the navigation bar.

18.2.1 “Protocol” view

Overview

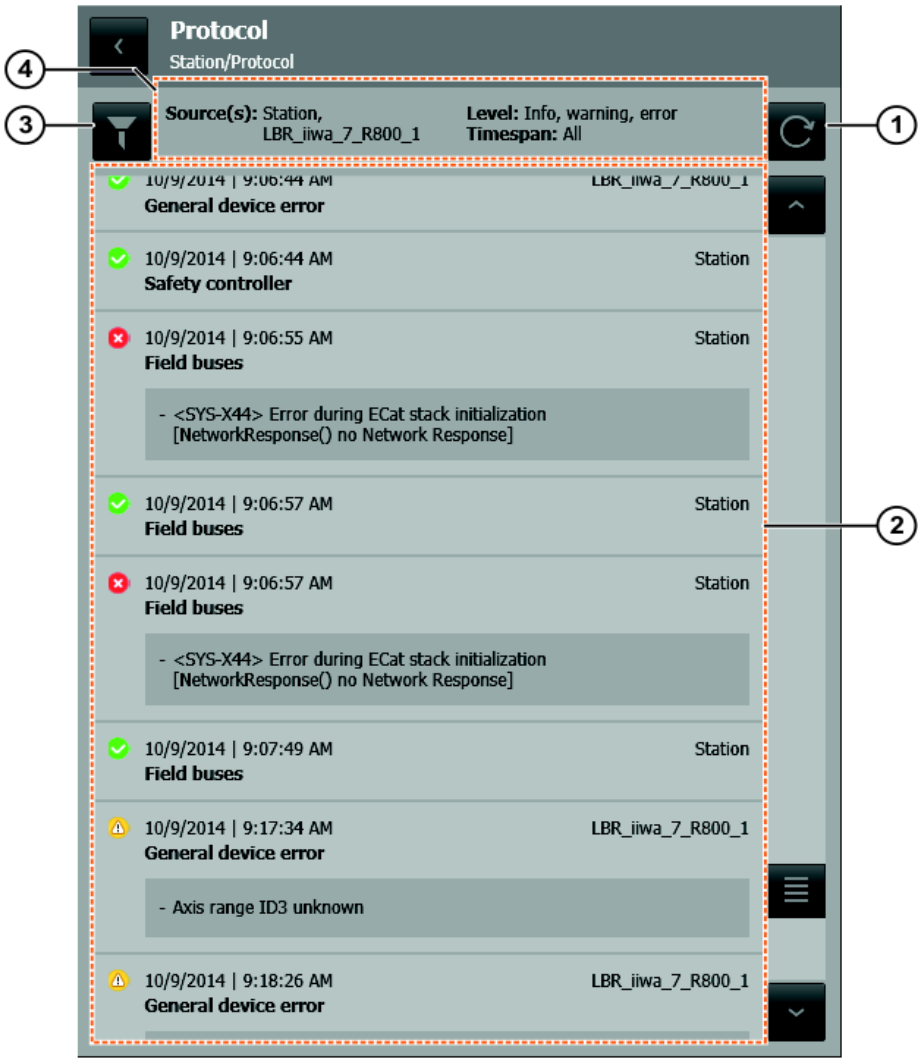


Fig. 18-1: “Protocol” view

Item	Description
1	**Refresh** button Refreshes the displayed protocol entries. After refreshing, the most recent entry is shown by default at the top of the list. If a time filter is active, the oldest entry is shown at the top of the list.
2	List of protocol entries (>>> "Log event" Page 426)
3	**Filter settings** button Opens the **Filter settings** window in which the protocol entries can be filtered according to various criteria.
4	Filter settings display The currently active filters are displayed here.

Log event

The protocol entries contain various information pertaining to each log event.

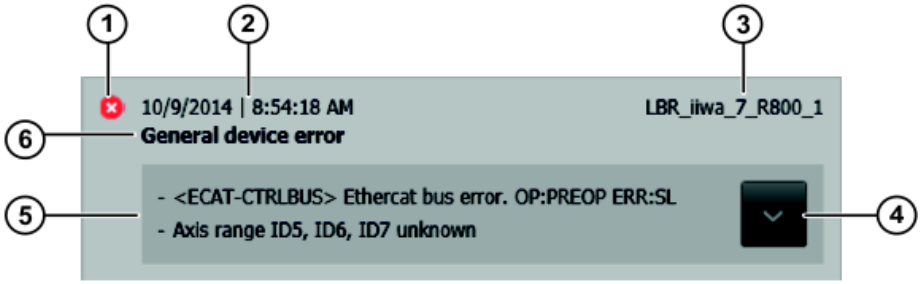


Fig. 18-2: Information about the log event

Item	Description
1	Log level of the event (>>> "Log level" Page 427)
2	Date and time of the log event (system time of the robot controller)
3	Source of the log event (robot or station)
4	Button to maximize/minimize the detail view The button is only available if more than 2 symptoms are present.
5	Symptoms of the log event (detail view) By default, up to 2 symptoms are displayed per event.
6	Category or brief description of the log event

Log level

The following icons display the log level of an event:

Icon	Description
	Error Critical event which results in a system error state
	Warning Critical event which can result in an error
	Information Non-critical event or information pertaining to the change in state

18.2.2 Filtering protocol entries

Precondition

- The **Protocol** view is open.

Procedure

1. Touch the **Filter settings** button. The **Filter settings** window opens.
2. Select the desired filters with the appropriate buttons.
3. Touch the **Filter settings** button or an area outside the window.
   The **Filter settings** window is closed and the selected filters are activated.

> The filters are reset when the **Protocol** view is closed. When the view is re-opened, the default settings are reactivated.

Description

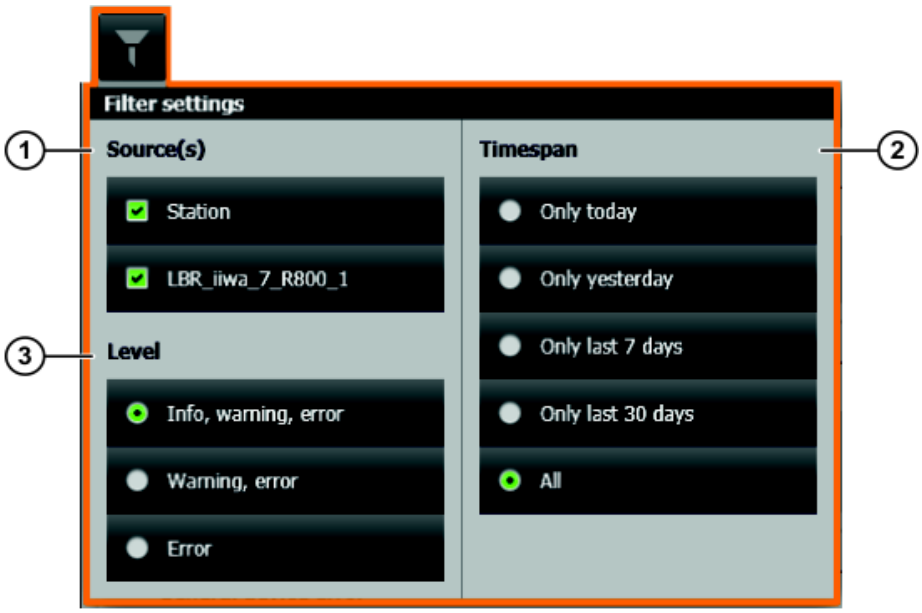


Fig. 18-3: “Filter settings” window

Item	Description
1	Filter **Source(s)** The protocol entries can be filtered according to the sources that caused the log event. ■ **Station**: All protocol entries are displayed which affect the station and the inputs/outputs of field buses. ■ **Robot**: Only those protocol entries are displayed which affect the robot selected in the navigation bar, here an LBR iiwa 7 R800. Default for Protocol in Station view: Both sources are selected. Default for Protocol in Robots view: The source is the robot selected in the navigation bar.
2	Filter **Timespan** A time filter can be activated to display only the protocol entries of a specific timespan. Default: **All** (no time filter active)
3	Filter **Level** The protocol entries can be filtered according to their log level. Default: **Info, warning, error** (no filter active for log level)

18.3 Display of error messages (Applications view)

If errors occur while an application is being executed, the corresponding error messages are displayed on the smartHMI.

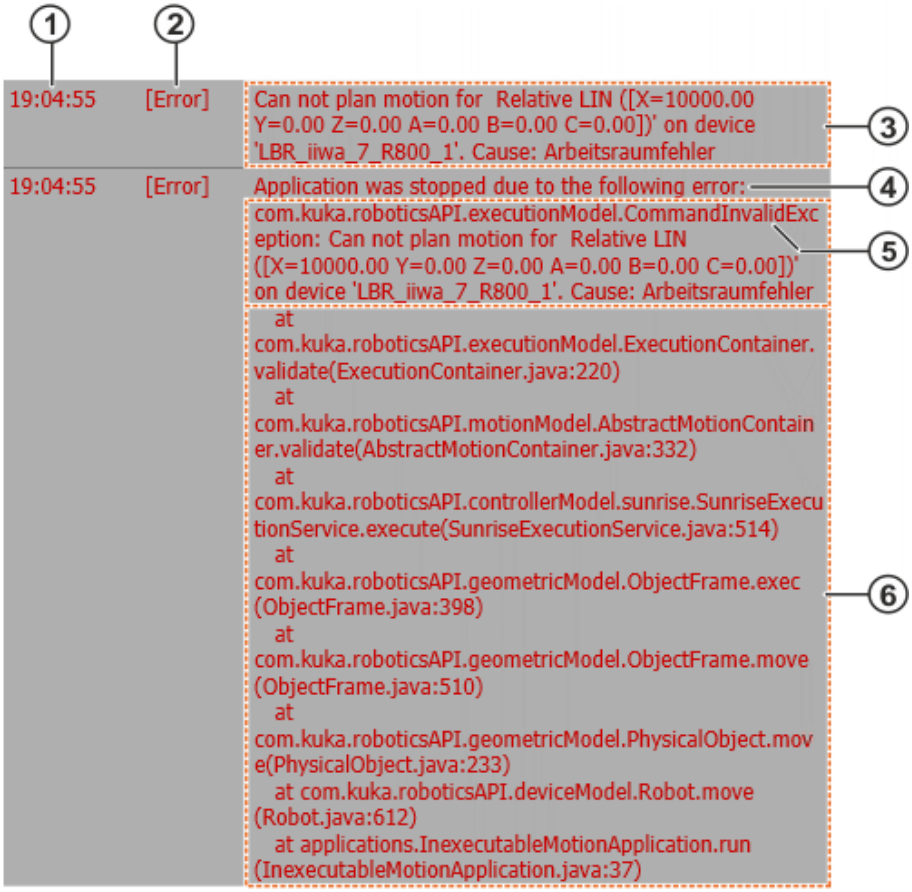


Fig. 18-4: Configuration of error message (example)

Item	Description
1	**Time stamp** Time at which the error occurred
2	**Level** Log level of the message. Errors have the log level **Error**.
3	Error message
4	Information when application is terminated, e.g. following a real-time error

Item	Description
5	Error type Errors are defined as Java classes. The name of the class and the corresponding package are displayed. The error message follows (see item 3).
6	Stack trace The method calls which led to the error are displayed in ascending order. The methods are specified with their full identifiers. In addition, the number of the program line in which the error occurred is displayed. The stack trace can be used to determine the program position at which the method which ultimately caused the error was called. Example, read from the bottom to the top: ■ Origin of the error: Method run() of the application InexecutableMotion.java, line 37 ■ In line 37 of the application, the method move(…) of the robot class was called. In the source code of the class robot.java, the error occurred in line 612 when the method move(…) of the class PhysicalObject was called. ■ ... ■ The actual error occurred in line 220 in the source code of the class ExecutionContainer.java when the method validate(…) was called.

Often, an error is the result of a chain of preceding errors. In this case, the entire error chain is displayed in descending order.

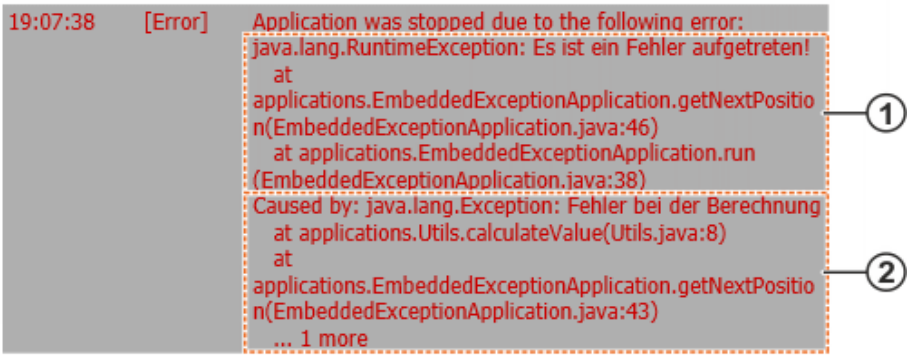


Fig. 18-5: Display of error chain (example)

Item	Description
1	Consequential error The last element in the error chain is displayed here. In the example, this is an error of type RuntimeException which occurred during execution of the method run() in line 38 of the application EmbeddedExceptionApplication.java.
2	Causative error The display of the causative error is always initiated as follows: ■ Caused by: *Error type* In the example, the causative error is of type Exception and occurred when the method calculateValue(…) of the class Utils was called. The entire error chain is thus displayed up to the actual cause of error.

18.4 Collecting diagnostic information for error analysis at KUKA

For error analysis, KUKA Customer Support requires diagnostic data from the robot controller.

For this purpose, a ZIP file called **KRCDiag** is created, which can be archived on the robot controller under D:\DiagnosisPackages or on a USB stick connected to the robot controller. The diagnosis package **KRCDiag** contains the data which KUKA Customer Support requires to analyze an error. These include information about the system resources, machine data and much more.

Sunrise.Workbench can also be used to access the diagnostic information. For this purpose, either an existing diagnosis package is loaded from the robot controller or a new package is created.

> Projects and applications are not included in the diagnosis package. It is advisible to transfer these data separately, as they can contain important information for troubleshooting.

> Recommendation: If possible, only collect diagnostic information when the robot is stationary.

> If the collection of diagnostic information fails while an application is running, stop and cancel the application and restart the diagnostic process.

18.4.1 Creating a diagnosis package with the smartHMI

Description

With this procedure, the diagnois package **KRCDiag** can be created and archived on the robot controller under D:\DiagnosisPackages or on a USB stick.

Procedure

1. For archiving to a USB stick: Plug the USB stick into the robot controller and wait until the LED on the USB stick remains permanently lit.
2. In the main menu, select **Diagnosis** > **Create diagnosis package** and select the desired file location.
   - **Hard disk**
   - **USB stick**

   The diagnostic information is compiled. Progress is displayed in a window. Once the operation has been completed, this is also indicated in the window. The window is then automatically hidden again.

18.4.2 Creating a diagnosis package with the smartHMI

Description

This procedure uses keys on the smartPAD instead of menu items. It can thus also be used if the smartHMI is not available.

The **KRCDiag** diagnosis package is created and archived on the robot controller under D:\DiagnosisPackages.

> The key sequence described in the procedure must be executed within 2 seconds.

Procedure

1. Press the “Main menu” key and hold it down.
2. Press the keypad key twice.
3. Release the “Main menu” key.

The diagnostic information is grouped. Progress is displayed in a window. Once the operation has been completed, this is also indicated in the window. The window is then automatically hidden again.

18.4.3 Creating a diagnosis package with Sunrise.Workbench

Precondition

- Network connection to the robot controller

Procedure

1. Right-click on the project in the **Package Explorer** and select **Sunrise** > **Create diagnosis package** from the context menu. The wizard for creating the diagnosis package opens.
2. Select **Browse...** and navigate to the directory in which the diagnosis package **KRCDiag** is to be created. If necessary, create a folder for the diagnosis package by clicking on **Create new folder**. Click on **OK** to confirm.
3. Click on **Next >**. The diagnosis package is created in the specified folder.
4. To navigate to the folder in which the diagnosis package was created, e.g. to send it directly by e-mail, click on **Open target folder in Windows Explorer**.
5. Click on **Finish**. The wizard is closed.

> Projects and applications are not included in the diagnosis package. It is advisible to transfer these data separately, as they can contain important information for troubleshooting.

18.4.4 Loading existing diagnosis packages from the robot controller

Precondition

- Network connection to the robot controller

Procedure

1. Right-click on the project in the **Package Explorer** and select **Sunrise** > **Create diagnosis package** from the context menu. The wizard for creating the diagnosis package opens.
2. Select **Browse...** and navigate to the directory in which the diagnosis package **KRCDiag** is to be copied. If necessary, create a folder for the diagnosis package by clicking on **Create new folder**. Click on **OK** to confirm.
3. Activate the radio button **Load existing diagnosis packages from controller** and select the desired diagnosis packages.
4. Click on **Next >**. The diagnosis package is copied into the specified folder.

   If the folder already contains a diagnosis package of the same name, a user dialog is displayed. The copying operation can be canceled.
5. To navigate to the folder into which the diagnosis package was copied, e.g. to send it directly by e-mail, click on **Open target folder in Windows Explorer**.
6. Click on **Finish**. The wizard is closed.

19 KUKA Service

19.1 Requesting support

Introduction

This documentation provides information on operation and operator control, and provides assistance with troubleshooting. For further assistance, please contact your local KUKA subsidiary.

Information

The following information is required for processing a support request:

- Description of the problem, including information about the duration and frequency of the fault
- As comprehensive information as possible about the hardware and software components of the overall system

  The following list gives an indication of the information which is relevant in many cases:
  - Model and serial number of the kinematic system, e.g. the manipulator
  - Model and serial number of the controller
  - Model and serial number of the energy supply system
  - Designation and version of the system software
  - Designations and versions of other software components or modifications
  - Diagnostic package **KrcDiag**:

    Additionally for KUKA Sunrise: Existing projects including applications

    For versions of KUKA System Software older than V8: Archive of the software (**KrcDiag** is not yet available here.)
  - Application used
  - External axes used

19.2 KUKA Customer Support

Availability

KUKA Customer Support is available in many countries. Please do not hesitate to contact us if you have any questions.

Argentina

Ruben Costantini S.A. (Agency)
Luis Angel Huergo 13 20
Parque Industrial
2400 San Francisco (CBA)
Argentina
Tel. +54 3564 421033
Fax +54 3564 428877
ventas@costantini-sa.com

Australia

KUKA Robotics Australia Pty Ltd
45 Fennell Street
Port Melbourne VIC 3207
Australia
Tel. +61 3 9939 9656
info@kuka-robotics.com.au
www.kuka-robotics.com.au

Belgium

KUKA Automatisering + Robots N.V.
Centrum Zuid 1031
3530 Houthalen
Belgium
Tel. +32 11 516160
Fax +32 11 526794
info@kuka.be
www.kuka.be

Brazil

KUKA Roboter do Brasil Ltda.
Travessa Claudio Armando, nº 171
Bloco 5 - Galpões 51/52
Bairro Assunção
CEP 09861-7630 São Bernardo do Campo - SP
Brazil
Tel. +55 11 4942-8299
Fax +55 11 2201-7883
info@kuka-roboter.com.br
www.kuka-roboter.com.br

Chile

Robotec S.A. (Agency)
Santiago de Chile
Chile
Tel. +56 2 331-5951
Fax +56 2 331-5952
robotec@robotec.cl
www.robotec.cl

China

KUKA Robotics China Co., Ltd.
No. 889 Kungang Road
Xiaokunshan Town
Songjiang District
201614 Shanghai
P. R. China
Tel. +86 21 5707 2688
Fax +86 21 5707 2603
info@kuka-robotics.cn
www.kuka-robotics.com

Germany

KUKA Roboter GmbH
Zugspitzstr. 140
86165 Augsburg
Germany
Tel. +49 821 797-4000
Fax +49 821 797-1616
info@kuka-roboter.de
www.kuka-roboter.de

France

KUKA Automatisme + Robotique SAS
Techvallée
6, Avenue du Parc
91140 Villebon S/Yvette
France
Tel. +33 1 6931660-0
Fax +33 1 6931660-1
commercial@kuka.fr
www.kuka.fr

India

KUKA Robotics India Pvt. Ltd.
Office Number-7, German Centre,
Level 12, Building No. - 9B
DLF Cyber City Phase III
122 002 Gurgaon
Haryana
India
Tel. +91 124 4635774
Fax +91 124 4635773
info@kuka.in
www.kuka.in

Italy

KUKA Roboter Italia S.p.A.
Via Pavia 9/a - int.6
10098 Rivoli (TO)
Italy
Tel. +39 011 959-5013
Fax +39 011 959-5141
kuka@kuka.it
www.kuka.it

Japan

KUKA Robotics Japan K.K.
YBP Technical Center
134 Godo-cho, Hodogaya-ku
Yokohama, Kanagawa
240 0005
Japan
Tel. +81 45 744 7691
Fax +81 45 744 7696
info@kuka.co.jp

Canada

KUKA Robotics Canada Ltd.
6710 Maritz Drive - Unit 4
Mississauga
L5W 0A1
Ontario
Canada
Tel. +1 905 670-8600
Fax +1 905 670-8604
info@kukarobotics.com
www.kuka-robotics.com/canada

Korea

KUKA Robotics Korea Co. Ltd.
RIT Center 306, Gyeonggi Technopark
1271-11 Sa 3-dong, Sangnok-gu
Ansan City, Gyeonggi Do
426-901
Korea
Tel. +82 31 501-1451
Fax +82 31 501-1461
info@kukakorea.com

Malaysia

KUKA Robot Automation (M) Sdn Bhd
South East Asia Regional Office
No. 7, Jalan TPP 6/6
Taman Perindustrian Puchong
47100 Puchong
Selangor
Malaysia
Tel. +60 (03) 8063-1792
Fax +60 (03) 8060-7386
info@kuka.com.my

Mexico

KUKA de México S. de R.L. de C.V.
Progreso #8
Col. Centro Industrial Puente de Vigas
Tlalnepantla de Baz
54020 Estado de México
Mexico
Tel. +52 55 5203-8407
Fax +52 55 5203-8148
info@kuka.com.mx
www.kuka-robotics.com/mexico

Norway

KUKA Sveiseanlegg + Roboter
Sentrumsvegen 5
2867 Hov
Norway
Tel. +47 61 18 91 30
Fax +47 61 18 62 00
info@kuka.no

Austria

KUKA Roboter CEE GmbH
Gruberstraße 2-4
4020 Linz
Austria
Tel. +43 7 32 78 47 52
Fax +43 7 32 79 38 80
office@kuka-roboter.at
www.kuka.at

Poland

KUKA Roboter Austria GmbH
Spółka z ograniczoną odpowiedzialnością
Oddział w Polsce
Ul. Porcelanowa 10
40-246 Katowice
Poland
Tel. +48 327 30 32 13 or -14
Fax +48 327 30 32 26
ServicePL@kuka-roboter.de

Portugal

KUKA Sistemas de Automatización S.A.
Rua do Alto da Guerra n° 50
Armazém 04
2910 011 Setúbal
Portugal
Tel. +351 265 729780
Fax +351 265 729782
kuka@mail.telepac.pt

Russia

KUKA Robotics RUS
Werbnaja ul. 8A
107143 Moskau
Russia
Tel. +7 495 781-31-20
Fax +7 495 781-31-19
info@kuka-robotics.ru
www.kuka-robotics.ru

Sweden

KUKA Svetsanläggningar + Robotar AB
A. Odhners gata 15
421 30 Västra Frölunda
Sweden
Tel. +46 31 7266-200
Fax +46 31 7266-201
info@kuka.se

Switzerland

KUKA Roboter Schweiz AG
Industriestr. 9
5432 Neuenhof
Switzerland
Tel. +41 44 74490-90
Fax +41 44 74490-91
info@kuka-roboter.ch
www.kuka-roboter.ch

Spain

KUKA Robots IBÉRICA, S.A.
Pol. Industrial
Torrent de la Pastera
Carrer del Bages s/n
08800 Vilanova i la Geltrú (Barcelona)
Spain
Tel. +34 93 8142-353
Fax +34 93 8142-950
Comercial@kuka-e.com
www.kuka-e.com

South Africa

Jendamark Automation LTD (Agency)
76a York Road
North End
6000 Port Elizabeth
South Africa
Tel. +27 41 391 4700
Fax +27 41 373 3869
www.jendamark.co.za

Taiwan

KUKA Robot Automation Taiwan Co., Ltd.
No. 249 Pujong Road
Jungli City, Taoyuan County 320
Taiwan, R. O. C.
Tel. +886 3 4331988
Fax +886 3 4331948
info@kuka.com.tw
www.kuka.com.tw

Thailand

KUKA Robot Automation (M)SdnBhd
Thailand Office
c/o Maccall System Co. Ltd.
49/9-10 Soi Kingkaew 30 Kingkaew Road
Tt. Rachatheva, A. Bangpli
Samutprakarn
10540 Thailand
Tel. +66 2 7502737
Fax +66 2 6612355
atika@ji-net.com
www.kuka-roboter.de

Czech Republic

KUKA Roboter Austria GmbH
Organisation Tschechien und Slowakei
Sezemická 2757/2
193 00 Praha
Horní Počernice
Czech Republic
Tel. +420 22 62 12 27 2
Fax +420 22 62 12 27 0
support@kuka.cz

Hungary

KUKA Robotics Hungaria Kft.
Fö út 140
2335 Taksony
Hungary
Tel. +36 24 501609
Fax +36 24 477031
info@kuka-robotics.hu

USA

KUKA Robotics Corporation
51870 Shelby Parkway
Shelby Township
48315-1787
Michigan
USA
Tel. +1 866 873-5852
Fax +1 866 329-5852
info@kukarobotics.com
www.kukarobotics.com

UK

KUKA Robotics UK Ltd
Great Western Street
Wednesbury West Midlands
WS10 7LL
UK
Tel. +44 121 505 9970
Fax +44 121 505 6589
service@kuka-robotics.co.uk
www.kuka-robotics.co.uk

Index

Symbols
“Ready for motion”, polling 314

Numbers
2004/108/EC 41
2006/42/EC 41
3-point method 103
89/336/EEC 41
95/16/EC 41

A
ABC 2-point method 101
ABC World method 102
Accessories 19, 23
Activation delay, for safety functions 215
Actual position, axis-specific 91
Actual position, Cartesian 92
addCartesianForce(…) 354
addCartesianTorque(…) 354
addCommandedCartesianPositionXYZ(…) 355
addCommandedJointPosition(…) 354
addControllerListener(…) 315, 318
addCurrentCartesianPositionXYZ(…) 355
addCurrentJointPosition(…) 355
addDoubleUserKey(…) 360
addExternalJointTorque(…) 354
addInternalJointTorque(…) 354
addUserKey(…) 360
AMF 16
ANSI/RIA R.15.06-2012 42
API 16
App_Enable 164
App_Start 164, 379
Application data (view) 46
Application mode 83
Application override 88, 320
Application tool 75
Application, pausing 88, 370
Applied norms and regulations 41
Approximate positioning 252
Approximate positioning point 252
areDataValid() 117
attachTo(…) 292
AUT 25
AutExt_Active 164
AutExt_AppReadyToStart 164, 379
Auto-complete 266
Automatic 25
Automatic mode 39
Auxiliary point 244, 278
awaitFileAvailable(…) 357
Axis limit 210
Axis range 25, 210
Axis torque condition 324
Axis torque monitoring 216
Axis torques, polling 303
Axis-specific impedance controller 399, 421
Axis-specific monitoring spaces, defining 210
Axis-specific position, polling 309

B
Background task, new 52
Background tasks 385
Base coordinate system 73, 103
Base for jogging 128
Base, calibration 103
Blocking wait 351
BooleanIOCondition 322
Brake defect 36
Brake test 109
Brake test application, template 111
Brake test, evaluation 120
Brake test, performing 124
Brake test, polling results 122
Brake test, programming interface 115
Brake test, results (display) 125
Brake test, start of execution 119
Brake test, starting position 115
Brake, defective 110, 111, 124
BrakeState (enum) 122
BrakeTest (class) 115, 118
BrakeTestResult (class) 121
Braking distance 25
Break conditions for motions 339
Break conditions, evaluating 340
breakWhen(…) 339, 341
Bus I/Os, mapping 159

C
Calibration 98
Calibration, base 103
Calibration, tool 98
Cartesian impedance controller 399, 401, 409
Cartesian position, polling 310
Cartesian protected spaces, defining 208
Cartesian setpoint/actual value difference, polling 311
Cartesian workspaces 206
CartesianTorqueCondition 322
CE mark 24
Checksum, safety configuration 189
CIB-SR 16
CIRC 278
CIRC, motion type 244
Circular motion 278
Cleaning work 40
clipApplicationOverride(…) 320
clipManualOverride(…) 320
Collision detection 217
Complex conditions 323
Condition for Boolean signals 338
Condition for the range of values of a signal 339
Condition, Cartesian torque 331
Conditional branch 374
Conditions 321
Connecting cables 19, 23

Constant force, overlaying 416
Continuous Path 243
Controller object, creating 400
Controller parameters, defining 400
Controllers, overview 399
Coordinate system, for jog keys 64
Coordinate systems 72
Counting loop 371
CP motions 243
CP spline block 243
CP Spline block, creating 282
createAndEnableConditionObserver(…) 349
createConditionObserver(…) 349
createDesiredForce(…) 416
createLissajousPattern(…) 416
createNormalForceCondition(…) 325, 327
createShearForceCondition(…) 325, 328
createSinePattern(…) 416
createSpatialForceCondition(…) 325, 326
createSpatialTorqueCondition(…) 332
createSpiralPattern(…) 416
createTiltingTorqueCondition(…) 332, 334
createTurningTorqueCondition(…) 332, 333
createUserKeyBar(…) 360
CRR 17, 26, 78
Customer PSM, PSM table 180
Cyclical background task 386

D
Danger zone 25
Data types 274
Data, recording and evaluating 352
DataRecorder 352, 353
Debug (perspective) 47
Declaration of conformity 24
Declaration of incorporation 23, 24
Decommissioning 41
Default frame for motions 135
DefaultApp_Error 164
Deselect (button) 86
detach() 294
Diagnosis 425
Diagnosis package, creating 431, 432
Diagnosis package, loading from the robot controller 432
Diagnostic information, collecting 431
displayModalDialog(…) 369
Disposal 41
DO WHILE loop 373
Documentation, industrial robot 15

E
EC declaration of conformity 24
Effective override 88, 89, 320
Electromagnetic compatibility (EMC) 42
EMC Directive 24, 41
EMERGENCY STOP 60
EMERGENCY STOP device 28, 29, 30
EMERGENCY STOP, external 29, 30
EN 60204-1 + A1 42
EN 61000-6-2 42
EN 61000-6-4 + A1 42
EN 614-1 + A1 42
EN ISO 10218-1 42
EN ISO 12100 42
EN ISO 13849-1 42
EN ISO 13849-2 42
EN ISO 13850 41
enable(), DataRecorder 356
Enabling device 28, 29
Enabling device, external 29, 31
Enabling switch 61, 62
Enabling switches 29
equals(…) 341
Error treatment 379
ESM 16
ESM mechanism 185
ESM state, deleting 187
ESM state, new 185
ESM table 180
Event-driven Safety Monitoring 173
Exception 16
External control 163
External E-STOP 192

F
Fast entry, Java 267
Faults 36
Field bus diagnosis 425
Field buses, overview 154
Filter settings 426
Flange coordinate system 73
Fonts 274
FOR loop 371
Force component condition 329
Force condition 324
ForceComponentCondition 322
ForceCondition 322
Frame 16
Frame management 127
Frame, designation as base 128
Frames, addressing 84
Frames, creating 128
Frames, deleting 129
Frames, displaying 80
Frames, moving 129
Frames, teaching 81
FSoE 16
Function test 37

G
General safety measures 36
getAlphaRad() 312
getApplicationData().createFromTemplate() 291
getApplicationData().getFrame() 131
getApplicationOverride() 320
getApplicationUI() 360
getAxis() 121
getBetaRad() 312
getBrakeIndex() 121
getCommandedCartesianPosition(…) 309
getCommandedCartesianPosition() 346

getCommandedJointPosition() 309, 346
getCurrentCartesianPosition() 309, 346
getCurrentJointPosition() 309, 346
getEffectiveOverride() 320
getEmergencyStopEx() 317
getEmergencyStopInt() 317
getExecutionMode() 319
getExternalForceTorque(…) 305, 306
getExternalTorque() 304
getFiredBreakConditionInfo() 340
getFiredCondition() 341, 342, 346
getFlange() 292
getForce() 306
getForceInaccuracy() 307
getFrame(…) 293, 295
getFriction() 121
getGammaRad() 312
getGravity() 121
getHomePosition() 313
getLogLevel() 122
getManualOverride() 320
getMaxAbsTorqueValues() 117
getMaxBrakeHoldingTorque() 121
getMeasuredBrakeHoldingTorque() 121
getMeasuredTorque() 303
getMinBrakeHoldingTorque() 121
getMissedEvents() 346
getMotion() 379
getMotionContainer() 346
getMotorHoldingTorque() 121
getMotorIndex() 121
getMotorMaximalTorque() 121
getObserverManager() 349, 351
getOperationMode() 317
getOperatorSafetyState() 317
getPositionInfo() 341
getPositionInformation(…) 309
getPositionInformation() 311, 346
getRecovery() 378
getRecoveryStrategy(…) 378
getRotationOffset() 311
getSafetyState() 316
getSafetyStopSignal() 317
getSingleMaxAbsTorqueValue(...) 117
getSingleTorqueValue(…) 304
getStartPosition() 379
getStartTimestamp() 117
getState() 121, 122
getStoppedMotion() 341, 342
getStopTimestamp() 117
getTestedTorque() 122
getTimestamp() 122
getTorque() 306
getTorqueInaccuracy() 307
getTorqueValues(…) 304
getTranslationOffset() 311
getTriggerTime() 346
Graphics card 43

H

halt() 370
Hand-held control panel 19, 23
handGuiding() 251, 286
Handling of failed motion commands 379
Hard disk space 43
Hardware 43
hasActiveMotionCommand() 315
Holding brakes, opening 79
HOME position 312
HOME position, changing 312
HOME position, polling 313
Hooke's law 401
HOV 64, 75
HRC 17

I

I/O configuration, exporting 161
I/O configuration, new 154
I/O configuration, opening 154
I/O group, creating 157
I/O group, deleting 158
I/O group, editing 158
I/O group, exporting as a template 158
I/O group, importing from a template 159
I/O Mapping (window) 159, 160
IAnyEdgeListener 348
IApplicationOverrideControl (interface) 320
ICallbackAction, interface 344
ICondition, inferface 322
IControllerStateListener 315
Identification plate 61
IF ELSE branch 374
IFallingEdgeListener 348
Industrial robot 23
initialize() 266, 387, 389
initializeCyclic(…) 387
Inputs/outputs, display 93
Installation 147
Installation directory 43
Installation, KUKA Sunrise.Workbench 43
Intended use 21, 23
Interlock, physical safeguards 30
Introduction 15
IORangeCondition 322
IP address, displaying 95
IP addresses 49
IRecovery, interface 378
IRecovery, Schnittstelle 378
IRisingEdgeListener 348
isEnabled() 357
isFileAvailable() 357
isForceValid(…) 307
isInHome() 313
isMastered() 314
isReadyToMove() 314
isRecording() 357
isRecoveryRequired(…) 378
isRecoveryRequired() 378
isTorqueMeasured() 117
isTorqueValid(…) 307
ISunriseControllerStateListener 318
ITriggerAction, interface 344

IUserKeyBar, interface 360

J
Java Editor 265
Java Editor, opening 265
Java file, renaming 57
Java package, new 51
Java project, new 55
Java projects, referencing 56
Javadoc 16
Javadoc (view) 47
Javadoc browser, configuration 271
Javadoc information, displaying 269
Jog keys 60, 76, 77
Jog mode 34
Jog override 64, 75
Jogging type 64
Jogging, axis-specific 74, 76
Jogging, Cartesian 74, 77
Jogging, robot 74
Joint Path 243
JointTorqueCondition 322
JP motions 243
JP spline block 243
JP Spline block, creating 283
JRE 16

K
Keyboard key 60
Keypad 66
KLI 16
KMP 16
Knowledge, required 15
KRCDiag 431
KUKA Customer Support 433
KUKA RoboticsAPI 17
KUKA smartHMI 17, 63
KUKA smartPAD 17, 26, 36, 59
KUKA Sunrise Cabinet 19
KUKA PSM, PSM table 180
KUKA Sunrise Cabinet 17
KUKA Sunrise.OS 17

L
Labeling 35
Language package, installation 44
Language package, installing 149
Liability 23
LIN 278
LIN REL 279
LIN, motion type 244
Linear motion 278, 279
Lissajous oscillation, overlaying 418
Load data 136
Load data, entering 136
Loops, nesting 377
Low Voltage Directive 24

M
Machinery Directive 24, 41
Main menu key 60
Main menu, calling 69
main() 266
Maintenance 40
Manipulator 19, 23, 26, 28
Manual guidance mode 251, 286
Manual guidance, motion type 251
Manual guidance, programming 286
Manual mode 39
Manual override 86, 88, 320
Mapping, inputs/outputs 161
Mastering 97
Mastering state, polling 314
Mastering, deleting 97
Media flange Touch 192, 193
Menu bar 46
Message programming 368
Message window 86
Methods, extracting 268
Mode selection 33
Monitoring 322, 347
Monitoring of processes 322
Monitoring processes 347
Monitoring spaces 204
Monitoring, tool orientation 212
Motion execution, pausing 371
Motion parameters 284
Motion programming, basic principles 243
Motion types 243
MotionBatch 280
MotionPathCondition 322
Mounting orientation 49, 73
move(…) 276, 294, 380
moveAsync(…) 276, 294, 381
Multiple branches 375

N
Navigation bar 64
Non-cyclic background task 389
Non-safety-oriented functions 33
Normal force 325
NotificationType, Enum 350
Null space motion 77

O
Object management 132
Object templates (view) 46
ObserverManager 349, 351
onIsReadyToMoveChanged(…) 315
onKeyEvent(…), IUserKeyListener 362
onSafetyStateChanged(…) 318
onTriggerFired(…) 344
Operating mode, changing 70
Operation, KUKA smartPAD 59
Operation, KUKA Sunrise.Workbench 45
Operator 25, 27
Operator safety 28, 30
Operators 323
Options 19, 23
Orientation control 285
Orientation control, LIN, CIRC, SPL 254
Output, change 93

Overload 36
Override 64, 75, 86, 88, 320
Override, changing and polling 320
Overview of the robot system 19
Overview, safety acceptance 226

P
Package Explorer (view) 46
Panic position 29
Passwort, changing 190
Path-related condition 336
Path-related switching actions 322, 343
Pausing, application 88
Pausing, program 88
Performance Level 24
Permanent Safety Monitoring 172
Personnel 27
Perspectives, display 47
Perspektive, selection 46
Plant integrator 26
PLC 17
Point-to-point 243
Point-to-point motion 277
Pollling, robot position 309
Position and torque referencing 223
Position controller 399, 401
Position referencing 223
positionHold(…) 422
Post-test loop 373
Preventive maintenance work 40
Processor 43
Product description 19
PROFINET 17
PROFIsafe 17
Program execution 85
Program execution control 370
Program run mode, changing and polling 319
Program run mode, selecting 87
Program, pausing 88
Program, starting 88
Program, starting automatically 88
Program, starting manually 88
Programming 265
Programming (perspective) 47
Project management 127
Project, loading from the robot controller 145
Project, synchronizing 143
Project, transferring to the robot controller 143
Projects, archiving 54
Projects, loading to workspace 54
Properties (view) 47
Protected space 204, 208
Protective equipment 34
Protocol entries, filtering 427
Protocol, display 425
Protocol, view 426
PSM 17
PSM mechanism 182
PSM table, Customer PSM 180
PSM table, KUKA PSM 180
PTP 277
PTP, motion type 243
PTPRecoveryStrategy (class) 378

R
RAM 43
Reaction distance 25
Ready for motion signal, reacting to change 315
Recommissioning 37, 97
Redundancy angle 259
Redundancy information 131, 258
Reference, canceling 56
Rejecting loop 372
Renaming, variable 266
Repair 40
Reset (button) 86
Retraction, robot 78
Robot activity, polling 315
Robot application, new 51
Robot application, selecting 85
Robot base coordinate system 72
Robot controller 23
Robot controller, switching on/off 62
Robot position, polling 309
Robot type, display 95
Robot, repositioning 88
RoboticsAPI 17
RoboticsAPI, displaying the version 276
Robots view 68
run() 266, 389
runCyclic() 387

S
Safe operational stop 214
Safe operational stop, external 29, 31
Safeguards, external 35
Safety 23
Safety acceptance, overview 226
Safety concept 169
Safety configuration 169
Safety configuration, activating 189
Safety configuration, conversion 149
Safety configuration, deactivating 189
Safety configuration, opening 180
Safety configuration, restoring 190
Safety controller, resuming 79
Safety function, new for ESM 187
Safety function, new for PSM 183
Safety functions 24
Safety functions, configuration 182, 185
Safety instructions 15
Safety of machinery 41, 42
Safety signals, evaluating 316
Safety signals, polling 316
Safety stop 26
Safety stop 0 26
Safety stop 1 26
Safety stop 1 (path-maintaining) 26
Safety stop, external 29, 31
Safety zone 26, 28
Safety-oriented functions 28
Safety-oriented stop reactions 31

Safety-oriented tool 137
Safety-oriented tool, defining 138
Safety-oriented workpieces 140
Safety-oriented workpieces, defining 142
Safety, legal framework 23
SafetyConfiguration.sconf (file) 51, 180
Serial number, display 95
Service life 25
Service, KUKA Roboter 433
Set methods 284
setAdditionalControlForce(…) 406
setAmplitude(…) 412
setApplicationOverride(…) 320
setAxisLimitsEnabled(…) 289
setAxisLimitsMax(…) 289
setAxisLimitsMin(…) 289
setAxisLimitViolationFreezesAll(…) 290
setBias(…) 413
setBlendingCart(…) 285
setBlendingOri(…) 285
setBlendingRel(…) 285
setCartAcceleration(…) 284
setCartJerk(…) 284
setCartVelocity(…) 284
setCriticalText(…), IUserKey 367
setDamping(…) 406, 421
setDampingForAllJoints(…) 421
setExecutionMode(…) 319
setFallTime(…) 415
setForceLimit(…) 414
setFrequency(…) 412
setHoldTime(…) 415
setHomePosition(…) 312
setJointAccelerationRel(…) 284, 285
setJointJerkRel(…) 285
setJointVelocityRel(…) 284, 285
setLED(…), IUserKey 366
setMaxCartesianVelocity(…) 408
setMaxControlForce(…) 407
setMaxPathDeviation(…) 408
setNullSpaceDamping(…) 407
setNullSpaceStiffness(…) 407
setOrientationReferenceSystem(…) 256, 285
setOrientationType(…) 254, 285
setPermanentPullOnViolationAtStart(…) 290
setPhaseDeg(…) 413
setPositionLimit(…) 414
setRiseTime(…) 415
setSafetyWorkpiece(…) 298
setStayActiveUntilPatternFinished(…) 415
setStiffness(…) 406, 421
setStiffnessForAllJoints(…) 421
setText(…), IUserKey 364
setTotalTime(…) 415
Shear force 325
Signal state, polling 316
Signal state, reacting to change 318
Simple force oscillation, overlaying 417
Single point of control 41
Singularities 261
Singularity 255
smartHMI 17, 63
smartPAD 17, 26, 36, 59
Software 19, 23, 43
Software components 20
Software limit switches 34
Software version, displaying 95
Space Mouse 60
Spiral-shaped force oscillation, overlaying 419
SPL, motion type 245
Spline segment 245
Spline, motion type 245
SPOC 41
Standstill monitoring 214
Start backwards key 60
Start key 60, 61
Start-up 37, 97
startEvaluation() 116
Starting, program 88
Starting, system software 62
startRecording() 356
StartRecordingAction 356
Station configuration 147
Station configuration, opening 147
Station view 66
Station_Error 164
StationSetup.cat (file) 51, 147
Status 259
Status display 65
Step (button) 86
Stop category 0 26
Stop category 1 26
Stop category 1 (path-maintaining) 26
STOP key 60
Stop reactions, safety-oriented 31
stopEvaluation() 116
Stopping distance 25, 28, 206
stopRecording() 357
StopRecordingAction 357
Storage 41
Structure of a motion command 276
Structure, robot application 265
Sunrise I/Os, changing 158
Sunrise I/Os, creating 155
Sunrise I/Os, deleting 158
Sunrise project, new 48
Sunrise.Workbench, installation 43
Sunrise.Workbench, starting 45
Sunrise.Workbench, uninstallation 43
Sunrise.Workbench, user interface 45
SunriseExecutionService 319
Support request 433
Surface normal 325
SWITCH branch 375
Switching off, robot controller 62
Switching on, robot controller 62
Symbols 274
Synchronization, project 143
System integrator 24, 26, 27
System requirements, PC 43
System software, installing 148
System states, polling 313

T
T1 26
T2 26
Target group 15
Tasks (view) 47
TCP 17, 98, 134
TCP force monitoring 218
Template, for Sunrise project 48
Templates 267
Templates, user-specific 268
Terms used 16
Terms used, safety 25
Terms, used 16
Tilting torque 331
Tool calibration 98
Tool Center Point 98
Tool coordinate system 73, 98
Tool frame, creating 134
Tool load data, determining 105
Tool orientation 212
Tool, creating 133
Tool, switching off 191
Toolbars 46, 48
Tools, declaring 291
Tools, initializing 291
Torque 331
Torque component condition 335
Torque referencing 224
Torque value determination 114
TorqueComponentCondition 322
TorqueEvaluator (class) 113, 115, 116
Torques, axis-specific 92
TorqueStatistic (class) 113, 116, 117
Touch screen 59
Trademarks 16
Training 15
Transportation 37
Trigger 322, 343
Trigger information, evaluating 345
Triggers, programming 343
triggerWhen(…) 343
Turn 260

U
Unmastering 97
USB connection 61
Use, contrary to intended use 23
Use, improper 23
User 25, 27
User dialogs, programming 369
User interface, KUKA smartHMI 63
User interface, Sunrise.Workbench 45
User key bar, creation 359
User keys 60
User keys, activation 89
User keys, defining 358
User messages, programming 368
UserKeyAlignment (enum) 364

V
Variable, renaming 266
Variables 275
Velocity 75
Velocity monitoring functions 197
Version information, RoboticsAPI 276
View, protocol 426
Views, repositioning 47
Virus scanner, displaying messages 95
Virus scanner, installing 150

W
waitFor(…) 351
Warnings 15
WHILE loop 372
Workpiece frame, creating 134
Workpiece, creating 133
Workpieces, declaring 291
Workpieces, initializing 291
Workspace 25, 28, 204, 206, 210
Workspace, new 53
Workspace, Sunrise.Workbench 53
Workspace, switching 54
Workspaces, switching 53
World coordinate system 72

X
XYZ 4-point method 99