

Rexroth IndraMotion MLC 12VRS RCL Programming Instruction

R911333852
Edition 02

Application Manual



Title	Rexroth IndraMotion MLC 12VRS RCL Programming Instruction
Type of Documentation	Application Manual
Document Typecode	DOK-MLC***-RCL*PRO*V12-AP02-EN-P
Internal File Reference	RS-e3eb11e15b29718d0a6846a00109078e-2-en-US-2
Purpose of Documentation	This documentation provides information on the RobotControl. The programming language RCL (RobotControl Language) is focused. The program structure, variables, functions, motion statements and the required system parameters are described.

Record of Revision

Edition	Release Date	Notes
Edition 01	05.2011	First edition for MLC 12VRS
Edition 02	02.2012	Conversion from mm to mm/inch, K-0-0054

Copyright © Bosch Rexroth AG 2012

This document, as well as the data, specifications and other information set forth in it, are the exclusive property of Bosch Rexroth AG. It may not be reproduced or given to third parties without its consent.

Liability The specified data is intended for product description purposes only and shall not be deemed to be a guaranteed characteristic unless expressly stipulated in the contract. All rights are reserved with respect to the content of this documentation and the availability of the product.

Published by Bosch Rexroth AG

Bgm.-Dr.-Nebel-Str. 2 • 97816 Lohr a. Main, Germany
Phone +49 (0)93 52 / 18-0 • Fax +49 (0)93 52 / 18 49 41
<http://www.boschrexroth.com/>

System IndraLogic & Basis Motion Logic GBU (vha/MePe)

Note This document has been printed on chlorine-free bleached paper.

Table of Contents

	Page
1 About this Documentation.....	9
1.1 Validity of the Documentation.....	9
1.2 Required and Supplementing Documentations.....	9
1.3 Information Representation.....	13
1.3.1 Safety Instructions.....	13
1.3.2 Symbols Used.....	14
1.3.3 Names and Abbreviations.....	14
2 Coordinates and Coordinate Systems.....	15
2.1 General.....	15
2.2 Joint Coordinate System (JCS).....	16
2.3 Basic Coordinate System (BCS).....	16
2.4 World Coordinate System (WCS).....	18
2.5 Workpiece Coordinate System (WpCS).....	19
3 Using Coordinate Systems.....	21
3.1 Defining Coordinate Systems.....	21
3.2 Defining the Basic Coordinate System Relative to the World Coordinate System.....	21
3.3 Defining the Workpiece Coordinate System Relative to the World Coordinate System.....	23
3.4 Workpiece Coordinate File.....	24
3.5 Workpiece Coordinate Editor.....	25
3.6 Library Functions.....	26
4 Travel Range Limits in the RobotControl.....	33
5 RCL Program Structure.....	35
5.1 General Information.....	35
5.2 Mode of the Compiler Operation.....	35
5.2.1 Use.....	35
5.2.2 Source Files.....	35
5.2.3 Output Files.....	35
5.3 Compiler Statements.....	36
5.3.1 General Information.....	36
5.3.2 Kinematic Definition.....	36
5.3.3 World Coordinate Names (WC Name Definition).....	36
5.3.4 Axis Names (JC Name Definition).....	37
5.3.5 Including Files.....	37
5.3.6 Selectable Extension in the Include Statement.....	38
5.3.7 Debug Information.....	38
5.4 Main Program Structure.....	38
5.4.1 General.....	38
5.4.2 Declaration Part.....	39
5.4.3 Statement Part.....	40

Table of Contents

	Page
5.5	Program Declaration..... 40
5.6	Main Program Call in Main Program..... 41
5.7	Subroutine Declaration..... 44
5.8	Subroutine Sequence..... 46
6	RCL Constants..... 51
6.1	RCL Constant Declaration 51
6.2	Predefined RCL Constants..... 51
7	RCL Variables..... 53
7.1	General..... 53
7.2	RCL Data Types..... 53
7.2.1	Overview..... 53
7.2.2	Simple RCL Data Types..... 54
7.2.3	Composed RCL Data Types..... 55
7.2.4	User-Defined RCL Types..... 55
7.3	RCL Variable Declaration..... 57
7.4	Point Variables..... 58
7.4.1	Declaration of Point Variables..... 58
7.4.2	Points and Point File "pkt"..... 59
7.4.3	Complete Value Assignment..... 59
7.4.4	Assignment of Variables for Individual Components..... 60
7.5	Working with Text Variables..... 61
7.6	Working with Array Variables (Arrays)..... 62
7.7	PLC Communication Channels..... 64
7.7.1	Communication Channels to PLC, General..... 64
7.7.2	Communication Channels to PLC, Declaration and Programming..... 66
8	RCL Program Control..... 69
8.1	WAIT Statement..... 69
8.2	PAUSE Statement..... 73
8.3	HALT Statement..... 73
8.4	REPEAT Statement..... 74
8.5	JUMP Statement..... 75
8.6	IF-THEN-(ELSE) Statement..... 77
8.7	CASE Statement..... 80
9	Value Assignments and Operations..... 81
9.1	Value Assignments..... 81
9.2	Operations..... 81
9.2.1	Arithmetic Expressions..... 81
9.2.2	Comparative Operations..... 83
9.2.3	Logic Expressions..... 83
9.2.4	Sequence of Operations..... 84

Table of Contents

	Page
10 RCL Functions.....	87
10.1 SIN Function.....	87
10.2 COS Function.....	87
10.3 ATAN Function.....	87
10.4 SQRT Root Function.....	88
10.5 ABS Function.....	89
10.6 TRUNC Function.....	89
10.7 ORD Function.....	89
10.8 CHR Function.....	89
10.9 ROUND Function.....	89
10.10 EOF Function.....	90
10.11 ASSIGN Function.....	90
10.12 INT_ASC Function.....	90
10.13 ASC_INT Function.....	91
10.14 SIZEOF Function.....	92
10.15 LIMIT_MIN, LIMIT_MAX, LIMIT_OFF Functions.....	93
10.16 JC Standard Function.....	93
10.17 WC Standard Function.....	93
11 RCL Special Functions.....	95
11.1 RCL Special Functions, General.....	95
11.2 RCL Special Functions, Listed as Table.....	96
11.3 Position-accurate Switching.....	97
11.3.1 Position-accurate Switching, General.....	97
11.3.2 SPC_FCT: 1, Position-accurate Switching of Digital Outputs on the Path.....	98
11.3.3 SPC_FCT: 2, Position-accurate Switching of Process Data on the Path.....	99
11.4 SPC_FCT: 21, Belt Type.....	100
11.5 SPC_FCT: 23, DateTime.....	101
11.6 SPC_FCT: 24, SysClk.....	101
11.7 SPC_FCT: 28, Set Belt Counter.....	102
11.8 SPC_FCT: 53, Belt Synchronous Operating Range of Belt Type 4.....	102
11.9 RCL Programming "Blending Mode".....	105
11.9.1 SPC_FCT: 60, Set Blending mode.....	105
11.9.2 SPC_FCT: 59, Query blending mode.....	106
12 RCL Motion Statements.....	109
12.1 General Information.....	109
12.2 Direct Motion Statements.....	109
12.2.1 Overview.....	109
12.2.2 Motion Commands.....	110
12.2.3 Interpolation Method	114
12.2.4 Mode of Operation of Motion-Influencing RCL Statements and Parameters.....	118
12.2.5 Additional Information.....	119
Additional information, General.....	119
Additional Information: Velocity.....	119

Table of Contents

	Page
Additional Information: Velocity Factor.....	121
Additional Information: Acceleration.....	122
Additional Information: Acceleration Factor.....	124
Additional Information: Deceleration Factor.....	124
Additional Information on the Jerk.....	125
12.3 Motion-influencing Statements	127
12.3.1 General Information.....	127
12.3.2 Belt Synchronization.....	127
12.3.3 Block Transitions (BLOCK_SLOPE, PROGR_SLOPE, CONT_SLOPE)	129
12.3.4 Look-Ahead Across Several Blocks at PROGR_SLOPE and CONT_SLOPE.....	133
12.3.5 Late Commanding.....	133
12.3.6 Spatial Blending.....	134
Overview.....	134
Spatial Blending for PROGR_SLOPE and CONT_SLOPE.....	135
Spatial Blending for BLOCK_SLOPE.....	140
Late Blending.....	142
Blending Mode.....	144
13 Setup Mode.....	147
13.1 Setup Mode, General Information.....	147
13.2 Jogging in Different Coordinate Systems.....	148
13.2.1 Parameters to be Set to Jog in Axis Coordinates.....	148
13.2.2 Parameter to Be Set to Jog in World Coordinates.....	149
13.2.3 Jogging in World Coordinates.....	150
14 RCL Parameters.....	151
14.1 Axis-specific Parameters Used by the Kinematics of an MLC.....	151
14.1.1 General Information.....	151
14.1.2 A-0-0028, Travel range limit switch.....	151
14.1.3 A-0-0029, Position polarities.....	151
14.1.4 A-0-0030, Position limit value, positive.....	152
14.1.5 A-0-0031, Position limit value, negative.....	152
14.1.6 A-0-0032, Velocity limit value, positive.....	153
14.1.7 A-0-0033, Velocity limit value, negative.....	153
14.1.8 A-0-0034, Acceleration limit value, bipolar.....	153
14.1.9 A-0-0036, Jerk limit value, bipolar.....	153
14.1.10 A-0-0038, Torque/force limit value, bipolar.....	154
14.1.11 A-0-0048, Velocity polarities.....	154
14.1.12 A-0-0049, Torque-/force polarities.....	155
14.1.13 A-0-0080, Jogging, short distance.....	155
14.1.14 A-0-0081, Jogging, long distance.....	156
14.1.15 A-0-0082, Jogging, low velocity.....	157
14.1.16 A-0-0083, Jogging, high velocity.....	158
14.1.17 A-0-0084, Jogging, acceleration.....	159
14.1.18 A-0-0085, Jogging, deceleration.....	159

Table of Contents

	Page
14.1.19 A-0-0086, Jogging, acceleration jerk.....	159
14.1.20 A-0-0087, Jogging, deceleration jerk.....	160
14.1.21 A-0-0224, Rapid stop deceleration.....	160
14.1.22 A-0-0225, Rapid stop jerk.....	160
14.1.23 A-0-0230, Blending distance PTP.....	161
14.1.24 A-0-0240, Feedrate override.....	161
14.1.25 A-0-0241, Feedrate override control word.....	161
14.1.26 A-0-0242, Position control, synchronization acceleration.....	161
14.1.27 A-0-0243, Position control, synchronization velocity.....	162
14.2 Common C-Parameters of All Kinematics of an MLC.....	163
14.2.1 General Information.....	163
14.2.2 C-0-0123, Maximum number of kinematics.....	163
14.2.3 C-0-2480, Configured kinematic number.....	163
14.2.4 C-0-2482, List of non-linked axes.....	164
14.2.5 C-0-2483, List of configured kinematics.....	164
14.2.6 C-0-2484, Kinematic configuration list.....	164
14.3 Parameter of a Kinematics in the MLC.....	165
14.3.1 General Information.....	165
14.3.2 K-0-0001, Logic kinematic number.....	165
14.3.3 K-0-0002, Kinematic name.....	165
14.3.4 K-0-0003, Maximum path speed.....	165
14.3.5 K-0-0004, reserved.....	166
14.3.6 K-0-0005, Software limit switches, positive, WC.....	166
14.3.7 K-0-0006, Software limit switches, negative, WC.....	166
14.3.8 K-0-0007, Axis number of kinematics.....	166
14.3.9 K-0-0008, Axis configuration list.....	166
14.3.10 K-0-0009, Osci: list of all K-parameters that can be oscilloscoped.....	168
14.3.11 K-0-0010, List of all K-parameters.....	168
14.3.12 K-0-0011, List of all invalid K-parameters.....	169
14.3.13 K-0-0012, Checksum of all K-parameters.....	169
14.3.14 K-0-0013, IDN list of checksum K-parameters.....	169
14.3.15 K-0-0014, Path acceleration, change time.....	169
14.3.16 K-0-0015, Path deceleration, change time.....	172
14.3.17 K-0-0016, Project identification number.....	174
14.3.18 K-0-0020, Kinematic diagnostic message.....	175
14.3.19 K-0-0021, Kinematic status.....	175
14.3.20 K-0-0022, Extended kinematic status.....	177
14.3.21 K-0-0023, Kinematics diagnostic number.....	179
14.3.22 K-0-0031, Transformation scheme.....	179
14.3.23 K-0-0032, Referencing axes, command configuration.....	180
14.3.24 K-0-0033, IPOS, update when coordinate change.....	180
14.3.25 K-0-0036, WC axis monitoring.....	180
14.3.26 K-0-0040, Transformation, lengths of the mechanics.....	181
14.3.27 K-0-0041, Transformation, factors.....	181
14.3.28 K-0-0042, WC system offset.....	181
14.3.29 K-0-0043, Assignment list of workpiece coordinates.....	182

Table of Contents

	Page
14.3.30 K-0-0050, Distance units WC.....	183
14.3.31 K-0-0051, Velocity units WC.....	184
14.3.32 K-0-0052, Acceleration units WC.....	184
14.3.33 K-0-0053, Jerk units WC.....	185
14.3.34 K-0-0054, Unit of measurement for kinematic data.....	185
14.3.35 K-0-0070, Belt number of kinematics.....	186
14.3.36 K-0-0071, Coupling factors, belt 1.....	186
14.3.37 K-0-0072, Coupling factors, belt 2.....	187
14.3.38 K-0-0073, Coupling factors, belt 3.....	187
14.3.39 K-0-0074, Coupling factors, belt 4.....	188
14.3.40 K-0-0080, Jogging, distance short in WC system.....	188
14.3.41 K-0-0081, Jogging, distance long in WC system.....	189
14.3.42 K-0-0082, Jogging, velocity low in WC system.....	190
14.3.43 K-0-0083, Jogging, velocity high in WC system.....	191
14.3.44 K-0-0084, Jogging, acceleration in WC system.....	192
14.3.45 K-0-0085, Jogging, deceleration in WC system.....	192
14.3.46 K-0-0086, Jogging, acceleration jerk in WC system.....	193
14.3.47 K-0-0087, Jogging, deceleration jerk in WC system.....	193
14.3.48 K-0-0100, Current position, Cartesian original coordinates.....	194
14.3.49 K-0-0102, Current path velocity.....	194
14.3.50 K-0-0103, Current path acceleration.....	194
14.3.51 K-0-0104, Current path jerk.....	194
14.3.52 K-0-0105, Actual position, Cartesian workpiece coordinates.....	194
14.3.53 K-0-0106, Coordinate values of tool.....	195
14.3.54 K-0-0107, Coordinate values of workpiece.....	195
14.3.55 K-0-0121 to K-0-0136, Coordinate Names.....	195
14.3.56 K-0-0121, Coordinate name C1.....	195
14.3.57 K-0-0140, Tool name.....	195
14.3.58 K-0-0141, Workpiece name.....	195
14.3.59 K-0-0150, Interpolated command position.....	195
14.3.60 K-0-0152, Interpolated path velocity.....	195
14.3.61 K-0-0153, Interpolated path acceleration.....	195
14.3.62 K-0-0154, Interpolated path jerk.....	195
14.3.63 K-0-0155, Safe zones on/off.....	195
14.3.64 K-0-0156, Safe zone 1 min. point.....	196
14.3.65 K-0-0157, Safe zone 1 max. point.....	196
14.3.66 K-0-0156/K-0-0157 Safezone1 min./max. Point till K-0-0218/K-0-0219 Safezone32 min./max. Point	197
14.3.67 K-0-0220, File transfer name.....	198
14.3.68 K-0-0221, File name to be compiled.....	199
14.3.69 K-0-0240, Feedrate override.....	199
14.3.70 K-0-0241, Feedrate override control word.....	199
14.3.71 K-0-0242, Acceleration override.....	199
14.3.72 K-0-0243, Deceleration override.....	200
14.3.73 K-0-1060, Command: Load file from CF.....	200
14.3.74 K-0-1061, Command: Save file to CF.....	200

Table of Contents

	Page
14.3.75 K-0-1062, Command: Compile file to MLC.....	200
14.3.76 K-0-2200, Programmed position 1 (active).....	201
14.3.77 K-0-2201, Programmed position 2 (active).....	201
14.3.78 K-0-2202, Type of actively programmed position.....	201
14.3.79 K-0-2203, Interpolation type.....	201
14.3.80 K-0-2204, Currently active QLL file name.....	201
14.3.81 K-0-2205, Currently active line number.....	201
14.3.82 K-0-2300, PLC-Bit input (RCL).....	201
14.3.83 K-0-2301, PLC-Integer input (RCL).....	201
14.3.84 K-0-2302, PLC-Real input (RCL).....	201
15 Kinematic User Interface.....	203
15.1 General Information.....	203
15.2 RobotControl Menu Structure.....	203
15.2.1 Menu Item: Robot.....	203
15.2.2 Menu Item: Kinematics.....	204
15.2.3 Menu Item: Programs.....	204
15.2.4 Menu Item: Prog.QLL.....	204
15.3 RCL Editor.....	205
15.3.1 General Information.....	205
15.3.2 Characteristics.....	205
15.3.3 Options.....	206
15.3.4 Creating a New RCL Program.....	206
15.3.5 Working with the Editor.....	206
15.4 Compiler.....	210
15.5 Debugger.....	211
15.5.1 General Information.....	211
15.5.2 Setting and Deleting Breakpoints.....	211
15.5.3 Debugging a Program.....	212
15.5.4 Monitoring Variables.....	213
16 Using the RoCo Multi-Kinematic Technology.....	215
16.1 General Information.....	215
16.2 Creating Kinematics.....	215
16.2.1 General Information.....	215
16.2.2 Defining the Kinematic Type (Robot Type, K-0-0031).....	216
16.2.3 Defining the Axis Assignment (K-0-0008).....	217
16.2.4 Deleting Kinematics.....	218
16.3 Coordinated Motion of the Kinematics.....	219
16.4 RCL Programs of the Kinematics.....	220
16.5 Point Tables of the Kinematics.....	221
16.5.1 General Information.....	221
16.5.2 Point Tables, Assignment and Response.....	222
16.6 PLC Shared Memory Interfaces (KinData) of the Kinematics.....	223

Table of Contents

	Page
17 RobotControl Functionality Programmed in the PLC.....	225
18 Transformation Rules.....	227
18.1 Transformation Rules, Overview.....	227
18.2 Robot Type 00, Cartesian Robot.....	227
18.3 Robot Type 07, 2-Axis Delta with 1 Optional Manual Axis.....	231
18.4 Robot Type 09, 3-Axis Scara with 1 Optional Manual Axis.....	234
18.5 Robot Type 10, 4-Axis Scara with 1 Optional Manual Axis.....	236
18.6 Robot Type 12, 2-Axis Parallelogram.....	239
18.7 Robot Type 13, 2-Axis Parallelogram.....	241
18.8 Robot Type 18, Contact Kinematics.....	243
18.9 Robot Type 19, 2-Axis Parallelogram V1.....	244
18.10 Robot type 20, 3-Axis Delta with 1 Optional Manual Axis,.....	246
18.11 Robot Type 21, 2-Axis Parallelogram V2.....	251
18.12 Robot Type 26, Robot with Offset Wrist with Six (Seven) Axes.....	254
18.13 Robot Type 27, Robot with Offset Wrist with Six Axes + Three Linear Axes.....	260
18.14 Robot Type 29, 2-Axis Delta with Rotary and Translatory Axis.....	267
18.15 Robot Type 30, Adabot.....	270
18.16 Robot Type 32, 2-Axis Delta with Asymmetric TCP.....	273
18.17 Robot Type 100, PLC.....	274
19 RCL Keywords.....	277
20 Service and Support.....	287
Index.....	289

1 About this Documentation

1.1 Validity of the Documentation

Target group This documentation provides an overview on the RobotControl of the IndraMotion MLC.

The programming language RCL (RobotControl Language) is focused.

Describes are:

- Program structure
- Variables, functions, motion statements
- Required parameters of the system
- Programming references for further information
- Transformation rules
- Features at PLC programming

This documentation supports the user during the phases:

- Programming the RCL program
- Debugging and commissioning

1.2 Required and Supplementing Documentations

Documentation titles with type codes and parts numbers

IndraWorks		MLC	XLC
/36/	Rexroth IndraWorks 12VRS Software Installation DOK-IWORKS-SOFTINS*V12-COxx-EN-P, R911334396 This documentation describes the IndraWorks installation.	X	X
/5/	Rexroth IndraWorks 12VRS Engineering DOK-IWORKS-ENGINEE*V12-APxx-EN-P, R911334388 This documentation describes the application of IndraWorks in which the Rexroth Engineering tools are integrated. It includes instructions on how to work with IndraWorks and how to operate the oscilloscope function.	X	X
/20/	Rexroth IndraMotion MLC 12VRS Functional Description DOK-MLC***-FUNC****V12-APxx-EN-P, R911333848 This documentation describes wizards, context menus, dialogs, control commissioning, device configuration and functionalities of the IndraMotion MLC.	X	
/20/	Rexroth IndraLogic XLC 12VRS Functional Description DOK-XLC***-FUNC****V12-APxx-EN-P, R911333878 This documentation describes wizards, context menus, dialogs, control commissioning, device configuration and functionalities of the IndraLogic XLC.		X
/7/	Rexroth IndraWorks 12VRS CamBuilder DOK-IWORKS-CAMBUIL*V12-APxx-EN-P, R911333842 This documentation describes the basic principles and operation of the CamBuilder, the cam editing tool.	X	X

About this Documentation

/37/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Automation Interface DOK-XLCMLC-AUT*INT*V12-APxx-EN-P, R911334178 This documentation describes the script-based access to IndraWorks project data via the interface of the Automation Interface.	X	X
/38/	Rexroth IndraWorks 12VRS FDT Container DOK-IWORKS-FDT*CON*V12-APxx-EN-P, R911334398 This documentation describes the IndraWorks FDT Container functionality. It includes the activation of the functionality in the project and working with DTMs.	X	X
/29/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Project Conversion DOK-XLCMLC-PROCONV*V12-APxx-EN-P, R911334187 This documentation describes the project conversion of IndraLogic 04VRS and IndraMotion MLC04VRS on IndraWorks version 12 with IndraLogic 2G. Changes with regard to Motion and PLC are described in detail.	X	X
/28/	Rexroth IndraMotion MLC 12VRS Commissioning DOK-MLC***-STARTUP*V12-COxx-EN-P, R911333858 This documentation describes the steps required to commission and perform service on the IndraMotion MLC system. It includes checklists for tasks to be frequently performed and a detailed description of the steps.	X	

Motion		MLC	XLC
/23/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS PLCopen Libraries DOK-XLCMLC-FUNLIB**V12-LIxx-EN-P, R911334182 This documentation describes the function blocks, functions and data types of the RIL_Common-Types, ML_Base and ML_PLCOpen libraries for the IndraLogic XLC/IndraMotion MLC. It also includes the error reactions of function blocks.	X	X
/27/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Generic Application Template DOK-XLCMLC-TF*GAT**V12-APxx-EN-P, R911334191 This documentation provides a structured template to the IndraLogic PLC programmer. This template can be used to add and edit the PLC programming code. It includes the template, the template wizard and example applications.	X	X
/31/	Rexroth IndraMotion MLC 12VRS RCL Programming Instruction DOK-MLC***-RCL*PRO*V12-APxx-EN-P, R911333852 This documentation provides information on the RobotControl. The programming language RCL (RobotControl Language) is focused. The program structure, variables, functions, motion statements and the required system parameters are described.	X	
/21/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Parameters DOK-XLCMLC-PARAM***V12-RExx-EN-P, R911334176 This documentation describes the parameters of the XLC/MLC systems as well as the interaction between parameterization and programming. It includes the axis parameters, control parameters, kinematic parameters, touch probe parameters and programmable limit switch parameters.	X	X
/10/	Rexroth IndraDrive Firmware for Drive Controllers MPH-, MPB-, MPD-, MPC-07 DOK-INDRV*-MP*-07VRS**-FKxx-EN-P, R911328670		
/11/	Rexroth IndraDrive MPx-16 Functions DOK-INDRV*-MP*-16VRS**-APxx-EN-P, R911326767		

About this Documentation

Field buses		MLC	XLC
/39/	Rexroth IndraMotion MLC 11VRS PLCopen Field Bus DOK-IM*ML*-PLCFBUS*V11-APxx-EN-P, R911333896 This documentation describes the creation of field bus drives in an IndraWorks project, function blocks, functions and data types of the RIL_CommonTypes.library (excerpt for field bus drives), the RMB_PLCOpenFieldBus.library and the RIL_Utility.library (excerpt for field bus drives). It also includes the error reactions of function blocks.	X	X
/4/	Rexroth IndraWorks 12VRS Field Buses DOK-IWORKS-FB*****V12-APxx-EN-P, R911334394 This documentation describes the supported field buses and their diagnostic function blocks.	X	X
HMI		MLC	XLC
/8/	Rexroth IndraWorks 12VRS HMI DOK-IWORKS-HMI*****V12-APxx-EN-P, R911334392 This documentation describes the functions, configuration and operation of the user interfaces IndraWorks HMI Engineering and IndraWorks HMI Operation.	X	X
/6/	Rexroth IndraWorks 12VRS WinStudio DOK-IWORKS-WINSTUD*V12-APxx-EN-P, R911333844 This documentation describes the installation of the software, working with WinStudio and the creation and operation of applications.	X	X
/50/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS HMI Connection DOK-XLCMLC-HMI*****V12-APxx-EN-P, R911334184 This documentation describes the visualization systems supported by the IndraLogic XLC and IndraMotion MLC and their connection.	X	X
PLC		MLC	XLC
/3/	Rexroth IndraWorks 12VRS IndraLogic 2G Programming Instruction DOK-IWORKS-IL2GPRO*V12-APxx-EN-P, R911334390 This documentation describes the PLC programming tool IndraLogic 2G and its use. It includes the basic use, first steps, visualization, menu items and editors.	X	X
/33/	Rexroth IndraWorks 12VRS, Basic Libraries, IndraLogic 2G DOK-IL*2G*-BASLIB**V12-LIxx-EN-P, R911333835 This documentation describes the system-comprehensive PLC libraries.	X	X
Technology			
/30/	Rexroth IndraMotion MLC 12VRS Technology Libraries DOK-MLC***-TF*LIB**V12-LIxx-EN-P, R911333868 This documentation describes the function blocks, functions and data types of the "ML_TechInterface.library", "ML_TechMotion.library", "RMB_TechCam.library" and "ML_TechBase.library". It also includes libraries for the winder functionality, register controller functionality and CrossCutter functionality.	X	
/60/	Rexroth IndraMotion MLC 12VRS RegisterControl (Library) DOK-MLC***-REGI*CO*V12-LIxx-EN-P, R911333856 This documentation describes the inputs and outputs of the individual function blocks and provides notes on their use.	X	

About this Documentation

/62/	Rexroth IndraMotion MLC 12VRS RegisterControl (Application Manual) DOK-MLC***-REGI*CO*V12-APxx-EN-P, R911333854 This documentation describes the application of the integrated register control for a rotogravure printing machine. The components of the mark stream sensor, the HMI application and the error recovery options are described. This instruction provides information on how to operate the register control, react to errors and query diagnostics. This documentation is written for machine setters and machine operators.	X	
/49/	Rexroth IndraMotion MLC 12VRS Winder Function Application DOK-MLC***-TF*WIND*V12-APxx-EN-P, R911333870 This application-related system documentation describes the application of the winder technology functions.	X	

Hardware		MLC	XLC
/1/	Rexroth IndraControl L45/L65 DOK-CONTRL-IC*L45*L65*-PRxx-EN-P, R911324661	X	X
/2/	Rexroth IndraControl L25 DOK-CONTRL-IC*L25*****-PRxx-EN-P, R911328474	X	X
/24/	Rexroth IndraControl Lxx 12VRS Function Modules DOK-CONTRL-FM*LXX**V12-APxx-EN-P, R911333830 This documentation describes all function modules of the Lxx controls including engineering and diagnostics.	X	X
/12/	Rexroth IndraDrive Drive Controllers MPx-02 to MPx-07 DOK-INDRV*-GEN-**VRS**-PAxx-EN-P, R911297317		
/13/	Rexroth IndraDrive MPx-02 to MPx-07 and HMV DOK-INDRV*-GEN-**VRS**-WAxx-EN-P, R911297319		
/35/	Rexroth IndraDrive Drive Controller Control Sections CSB01, CSH01, CDB01 DOK-INDRV*-CSH*****-PR08-EN-P, R911295012		

Diagnostics and service		MLC	XLC
/26/	Rexroth IndraMotion MLC/XLC 11VRS Service Tool DOK-IM*ML*-IMST***V11-RExx-EN-P, R911331940	X	X
/22/	Rexroth IndraLogic XLC IndraMotion MLC 12VRS Diagnostics DOK-XLCMLC-DIAG***V12-RExx-EN-P, R911334180 This documentation includes all control parameters implemented in the control systems IndraLogic XLC and IndraMotion MLC.	X	X

System Overview		MLC	XLC
/48/	Rexroth IndraMotion for Printing 12VRS System Overview DOK-IM*PR*-SYSTEM**V11-PRxx-EN-P, R911333840 This documentation describes the product IndraMotion for Packaging. It introduces the control systems, drive systems and I/O systems as well as the commissioning and programming.	X	
/48/	Rexroth IndraMotion for Packaging 12VRS System Overview DOK-IM*PA*-SYSTEM**V12-PRxx-EN-P, R911333838 This documentation describes the product IndraMotion for Packaging. It introduces the control systems, drive systems and I/O systems as well as the commissioning and programming.	X	

About this Documentation

/9/	Rexroth IndraMotion MLC 12VRS System Overview DOK-MLC***-SYSTEM**V12-PRxx-EN-P, R911333860 This documentation provides an overview on the hardware/software components of the automation system IndraMotion MLC in the mentioned version. It helps assembling a system.	X	
/9/	Rexroth IndraLogic XLC 12VRS System Overview DOK-XLC***-SYSTEM**V12-PRxx-EN-P, R911333880 This documentation provides an overview on hardware/software components of the automation system IndraLogic XLC in the respective version. It helps assembling a system.		X

First steps		MLC	XLC
/25/	Rexroth IndraMotion MLC 12VRS First Steps DOK-MLC***-F*STEP**V12-COxx-EN-P, R911333846 This documentation describes the first steps of the IndraMotion MLC and the RobotControl. It includes the hardware and software prerequisites as well as the creation of a project.	X	
/25/	Rexroth IndraLogic XLC 12VRS First Steps DOK-XLC***-F*STEP**V12-COxx-EN-P, R911333876 This documentation describes the first steps of the IndraLogic XLC. It includes the hardware and software prerequisites as well as the creation of a project.		X

xx Corresponding edition
Fig. 1-1: XCL/MLC documentation overview

1.3 Information Representation

1.3.1 Safety Instructions

If there are safety instructions, they contain certain signal words (Danger, Warning, Caution, Notice) and if applicable, signal alert symbols (acc. to ANSI Z535.6-2006).

The signal word draws the attention to the safety instruction and indicates the risk potential.

The signal alert symbol (warning triangle with exclamation mark) positioned in front of the signal words Danger, Warning and Caution indicates hazards for individuals.

DANGER

In case of non-compliance with this safety instruction, death or serious injury **will** occur.

WARNING

In case of non-compliance with this safety instruction, death or serious injury **can** occur.

CAUTION

In case of non-compliance with this safety instruction, minor or moderate injury can occur.

About this Documentation

NOTICE

In case of non-compliance with this safety instruction, material or property damage can occur.

1.3.2 Symbols Used

Note Notes are represented as follows:



This is a note for the user.

Tip Tips are represented as follows:



This is a tip for the user.

1.3.3 Names and Abbreviations

Term	Explanation
JCS	Joint Coordinate System
BCS	Basic Coordinate System
RCL	Programming language: Robot Control Language
TCP	Tool Center Point
WCS	World Coordinate System
WpCS	Workpiece coordinate system

Fig. 1-2: Terms and abbreviations used

2 Coordinates and Coordinate Systems

2.1 General

In robotics, terms such as robot axes, coordinate axes and coordinate systems are central terms.

Therefore, the name specification of robot axes, the reference of coordinate systems and the usage of coordinate transformations have significant importance.

In the following, the coordinate systems supported in the RobotControl are introduced.

In general, coordinate axes are Cartesian coordinates allocated in a right angle to each other. The positive directions of the coordinate axes are determined by a right-handed, rectangular coordinate system.

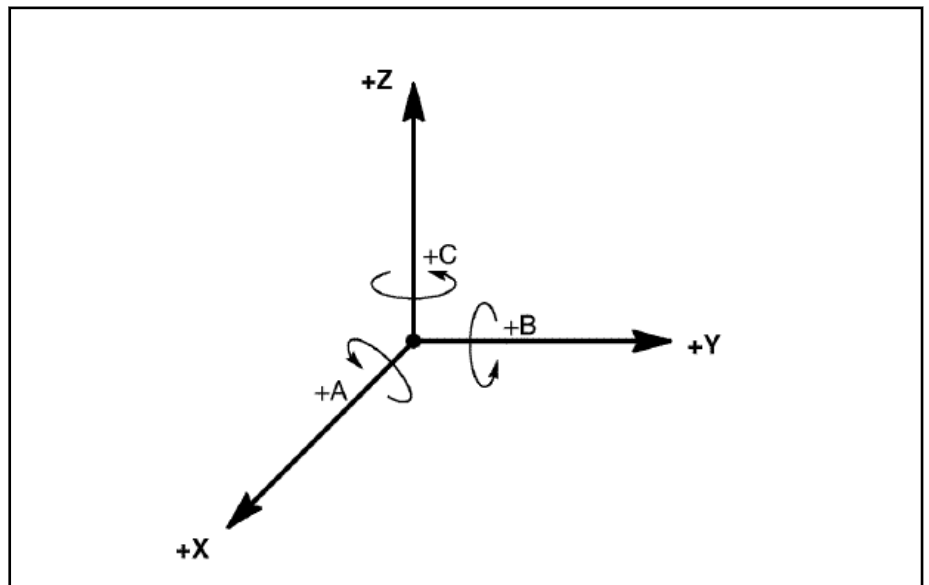


Fig.2-1: Cartesian coordinate system, axis name

To freely position and orient an object in a Cartesian coordinate system, all six degrees of freedom are generally required. The position consists of three degrees of freedom of the positioning and three degrees of freedom of the orientation.



The allocation of the individual robot axes towards each other cannot be rectangular even in actual usage. Thus, the result is a non-orthogonal joint coordinate system.

The following coordinate systems are currently supported in the RobotControl:

Overview on coordinate systems

- [Joint Coordinate System \(JCS\), page 16,](#)
- [Basic Coordinate System \(BCS\), page 16,](#)
- [World Coordinate System \(WCS\), page 18,](#)
- [Workpiece Coordinate System \(WpCS\), page 19.](#)

2.2 Joint Coordinate System (JCS)

The joint coordinate system (JCS) of a robot is defined by the entity of all positioning and orientation axes assigned to one kinematics.

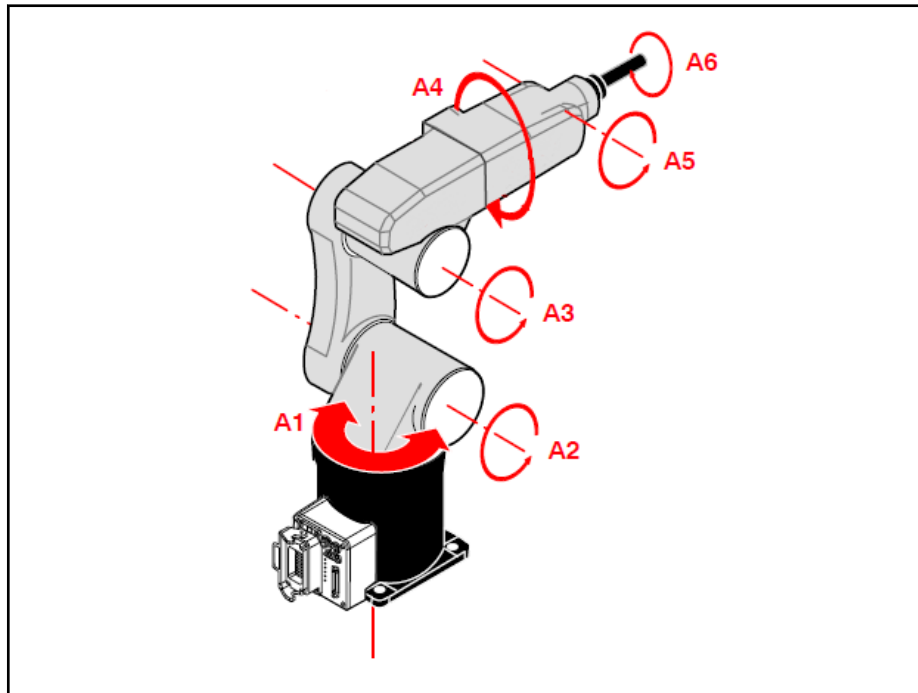


Fig. 2-2: Joint coordinates (A1,A2,A3,A4,A5,A6)

To setup the joint coordinate system, the robot axes are brought into zero positions. These can be found in the transformation guideline of the respective kinematics. Subsequently, the position measuring systems have to be referenced.

The basic requirements to perform the coordinate transformation correctly are:

- Exact settings of the axis zero positions
- Exact measurement of the arm lengths or center offsets
- exact determination of the coupling factors (if present)
- Correct directions of rotation of the axes



If one of these basic requirements is not fulfilled completely, a precise robot motion cannot be guaranteed. In worst case, this can cause incorrect and unexpected robot motions.

2.3 Basic Coordinate System (BCS)

The basic coordinate system (BCS) is a Cartesian coordinate system fixedly connected to the robot.

Coordinates and Coordinate Systems

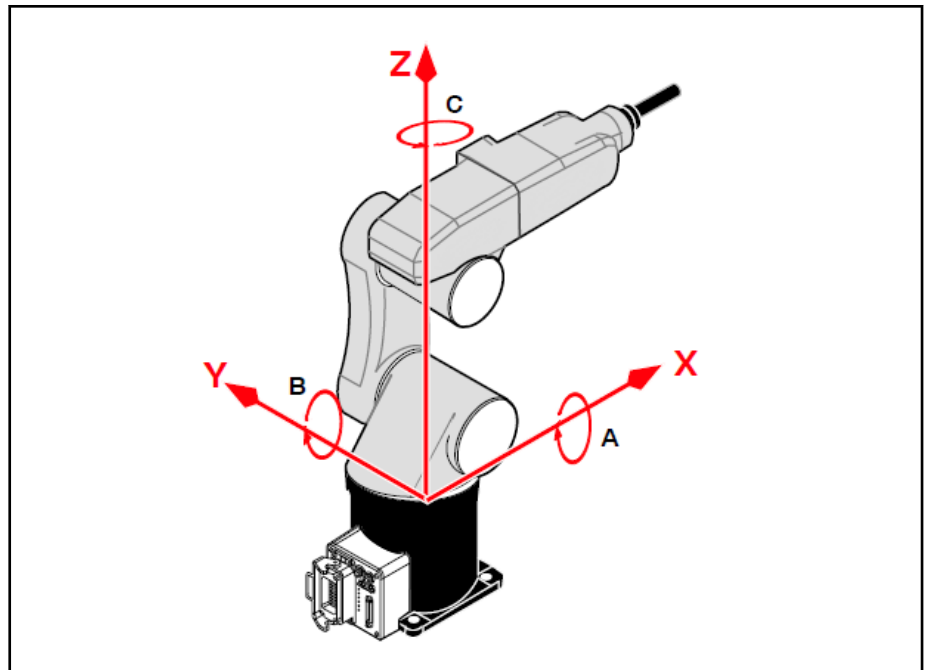


Fig. 2-3: Basic coordinates ($X_0, Y_0, Z_0, A_0, B_0, C_0$)

If the kinematic structure of the robot is not Cartesian, the relation between the joint and the Cartesian basic coordinates is established via a transformation.

Schematic representation of a kinematic transformation

Forward position kinematics/direct kinematic problem

If the current position of the tool working point has to be determined in space, the easiest way is to determine the joint coordinates and use these to calculate the position and orientation of the tool in the basic coordinate system.

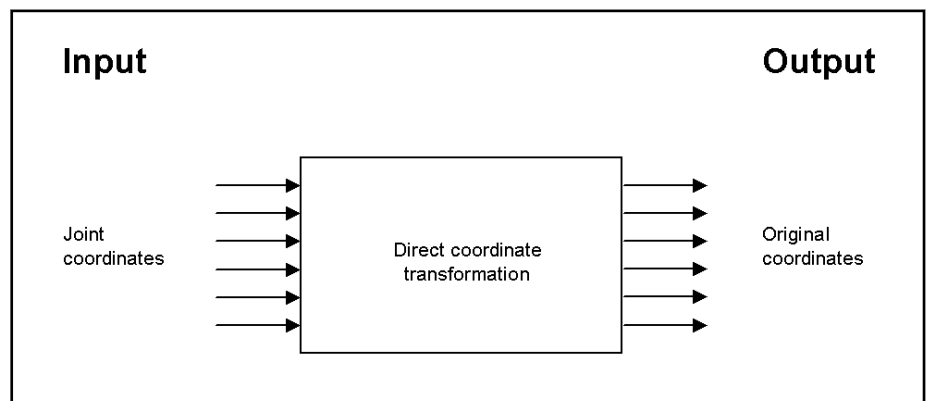


Fig. 2-4: Forward position kinematics

Inverse position kinematics

If the position of the tool working point is given by the position and the orientation in the basic coordinate system, this information is used to calculate the respective joint coordinates.

Coordinates and Coordinate Systems

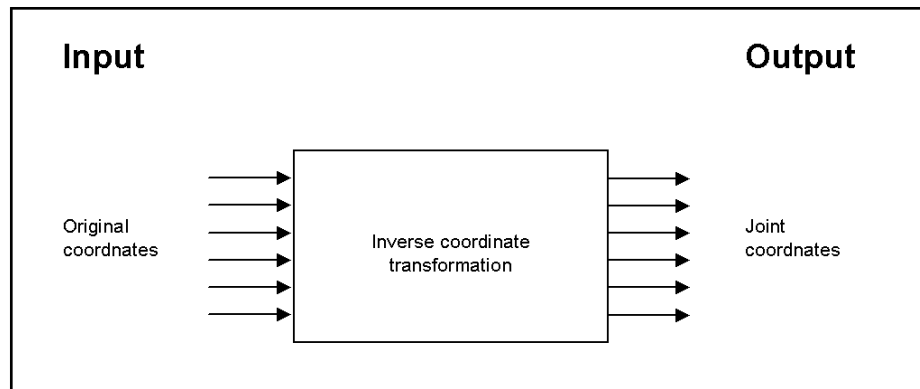
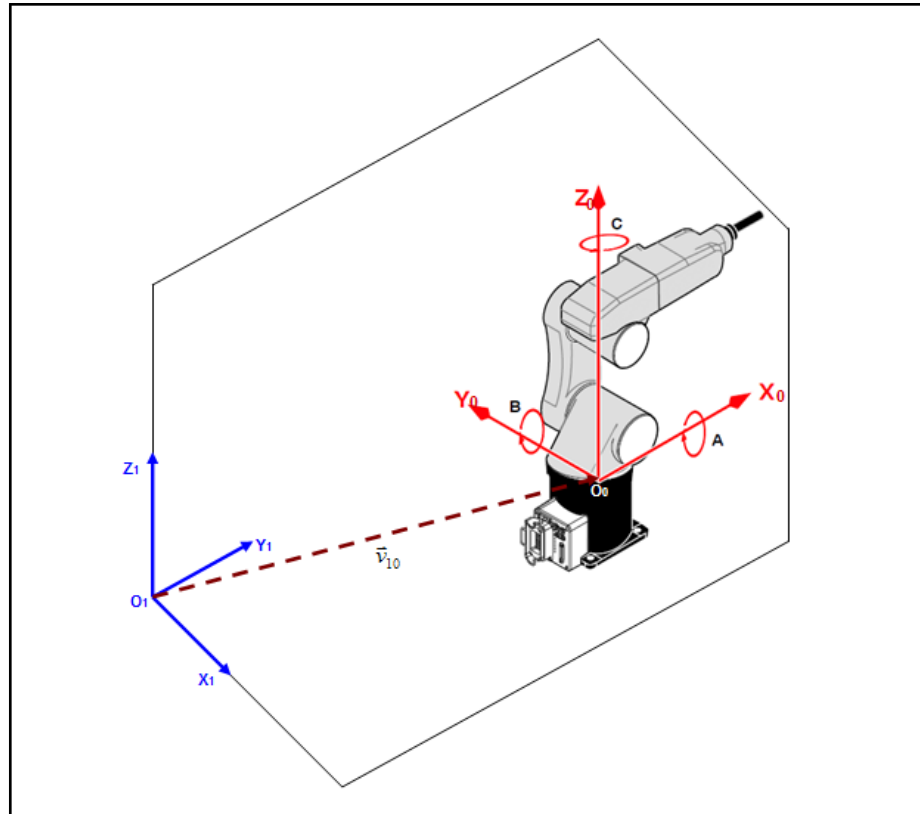


Fig.2-5: Inverse position kinematics

2.4 World Coordinate System (WCS)

The world coordinate system (WCS) is a coordinate system fix in location. The world coordinate system mostly orients at a distinctive point of the work cell of the robot.

Fig.2-6: World coordinates ($X_1, Y_1, Z_1, A_1, B_1, C_1$)

In practice, several robots are mounted within one work cell. If the workspaces of the respective robots overlap, the motion of the respective kinematics has to be controlled with regard to each other.

Therefore, the respective motions should be shown in the same coordinate system.

2.5 Workpiece Coordinate System (WpCS)

The workpiece coordinate system (WpCS = **W**orkpiece**C**oordinate **S**ystem) has a fixed position and orientation with regard to the world coordinate system.

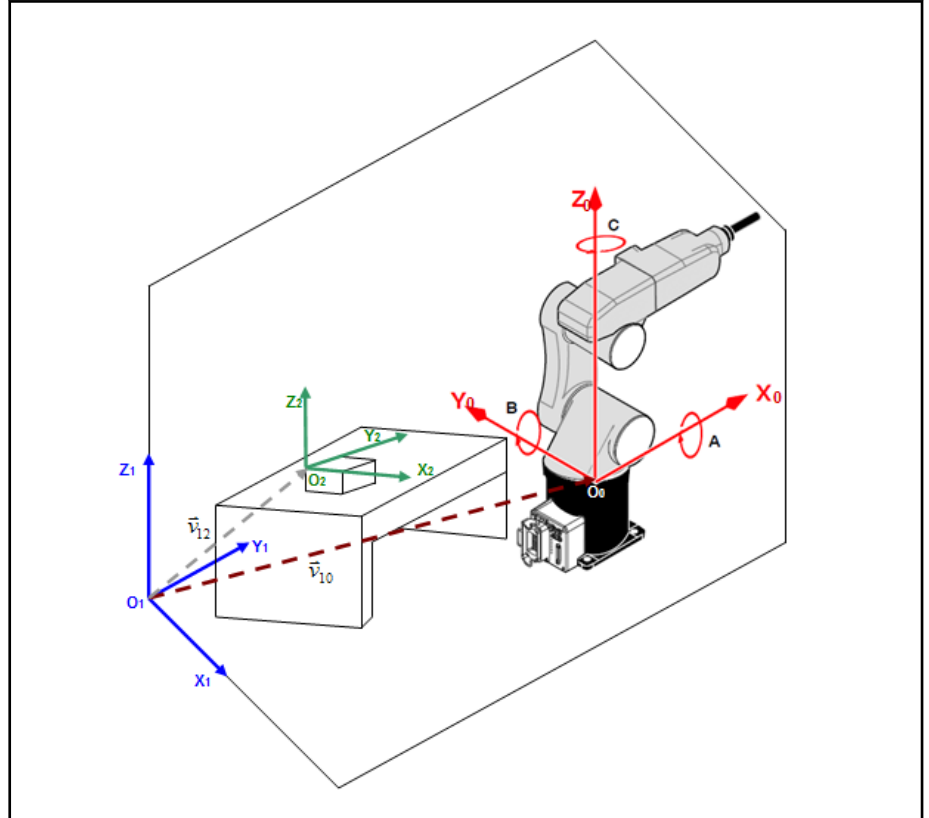


Fig.2-7: Workpiece coordinate system ($X_2, Y_2, Z_2, A_2, B_2, C_2$)

The workpiece coordinate system has special significance when programming machining tasks.

The workpiece coordinates can be easily used by the software engineer to exactly approach the different machining positions at the workpiece.



All programmed motions in the RCL program refer to the workpiece coordinate system.

The programmed coordinates are approached with the TCP (TCP = Tool Center Point).

The TCP is in the tool working point.

The workpiece coordinate system corresponds to the world coordinate system by default.

3 Using Coordinate Systems

3.1 Defining Coordinate Systems

The transition between coordinate systems, that is between world coordinates and basic coordinates or world coordinates and workpiece coordinates is carried out via transformations in the control.

The user is provided with the following options to define the coordinate systems.

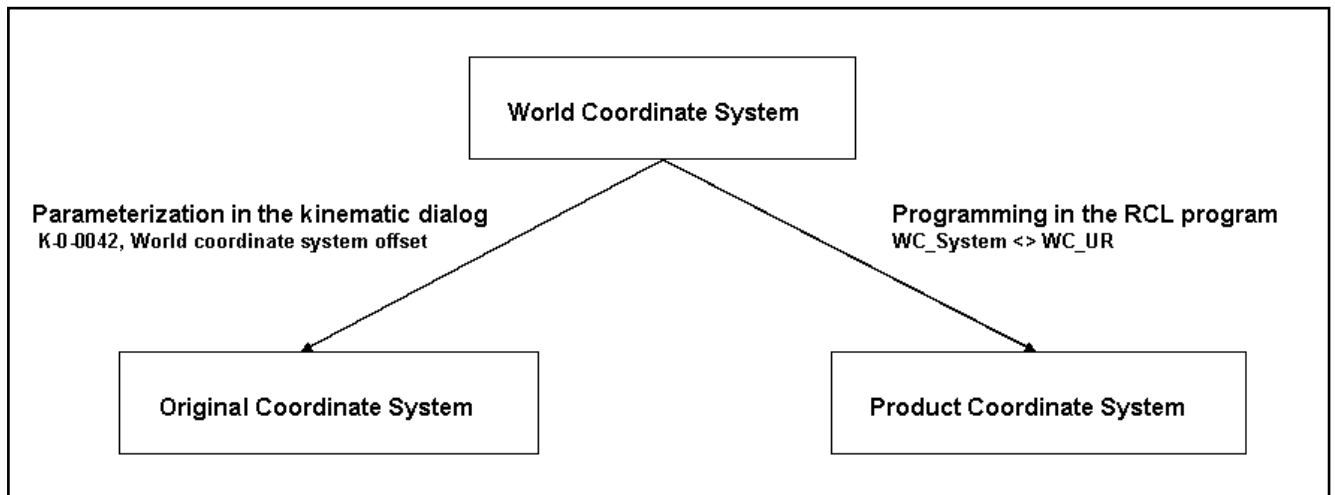


Fig.3-1: Overview on coordinate systems

3.2 Defining the Basic Coordinate System Relative to the World Coordinate System

Describing the function

The relation between the world coordinate system and the basic coordinate system is described via the kinematic parameter K-0-0042, WC system offset.

The parameter K-0-0042 consists of six elements. Three elements are used for positioning and three elements for orienting the basic coordinate system relative to the world coordinate system.



The workpiece coordinate system can only be generally used for 6-axis kinematics. If the kinematics is provided with less than six degrees of freedom, the axes to be considered have to be specified via the parameter K-0-0043. In this case, the workpiece coordinate system can be shifted arbitrarily in all three directions, but only rotated around by one axis max.

Which element of the parameter K-0-0042 acts on which offset/rotation in space can be set in the parameter K-0-0043.

Consider that the number of degrees of freedom depends on the kinematics used.

The **offset vector** starts at the origin of the world coordinate system and ends at the origin of the basic coordinate system.

The rotation of the coordinate system is only possible for particular transformation schemes (see K-0-0031). This is documented in the respective de-

Using Coordinate Systems

scriptions. (No note on the rotation of the coordinate system means that this option is not supported with this transformation schema).

Relevant parameters

K-0-0042	WC system offset
K-0-0043	Assignment list of workpiece coordinates
K-0-0031	Transformation scheme

The parameters are described in detail in the chapter [RCL Parameters](#), page 151,.

Activate

The position of the basic coordinate system is determined with regard to the world coordinate system.

Therefore, the offset vector has to be entered into the first three elements.

The orientation is determined by writing on the last three elements.

The offsets are specified in mm and the rotations in degrees.

To write on the parameter K-0-0042, the kinematic dialog "Coordinate offset" or the parameter editor can be used.



The parameters are applied when the phase is switched from P2 to BB.

According to the definition of the basic coordinate system relative to the world coordinate system, the coordinates refer in the KinData interface

`KinData[x].arPositionOriginalCoord_i`

refer to the world coordinate system.

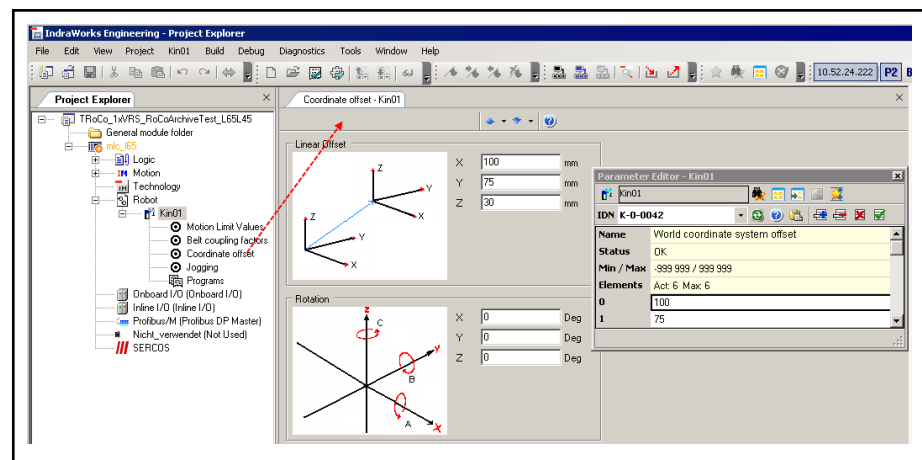


Fig.3-2: Options to activate the basic coordinate system

Deactivate

0 has to be assigned to all elements of the parameter K-0-0042, WC system offset.

After the phase switching from P2 to BB, the position of the basic coordinate system is identical to the position of the world coordinate system. Previously applied offsets and rotations are canceled.

After switching off the translation and rotation of the basic coordinate system, the coordinates in the KinData interface

`KinData[x].arPositionOriginalCoord_i`

remain related to the world coordinate system that is now identical to the basic coordinate system.

Using Coordinate Systems

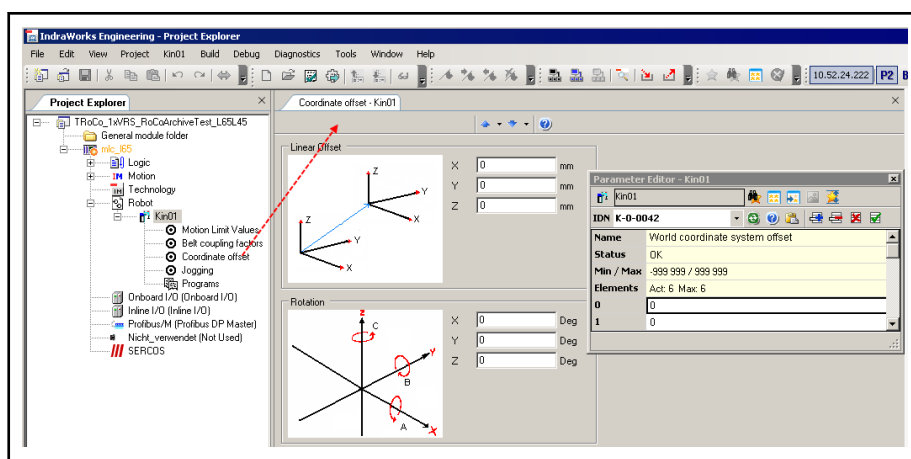


Fig.3-3: Options to deactivate the basic coordinate system

Restrictions

It has to be considered that the values of the parameter K-0-0042 become only active when switching the phase from P2 (P2 = Parameterization mode) to BB (BB = Ready for operation).

3.3 Defining the Workpiece Coordinate System Relative to the World Coordinate System

Describing the function

The relation between the workpiece coordinate system and the world coordinate system is described via the RCL command **WC_SYSTEM**. This command consists of six elements. Three elements are used to position the workpiece coordinate system and three elements to orientate the workpiece coordinate system. The option to apply rotations depends on the transformation scheme. Consider that the number of degrees of freedom depends on the kinematics used.

The translation vector starts at the origin of the world coordinate system and ends at the origin of the workpiece coordinate system.



The offsets are specified in mm and the rotations in degrees.

Relevant RCL commands

WC_SYSTEM is a kinematic-related default variable of the type **WC_FRAME**. The valid workpiece coordinate system is communicated and activated via this standard variable.

WC_FRAME is a kinematic-related default type consisting of six components of the type **REAL**. The six components describe the offset and the rotation of the workpiece coordinate system compared to the world coordinate system.

The sequence of the components is as follows:

1. Component	deltaX	x-direction
2. Component	deltaY	y-direction
3. Component	deltaZ	z-direction
4. Component	deltaO1	First orientation axis
5. Component	deltaO2	Second orientation axis
6. Component	deltaO3	Third orientation axis

The special definition and sequence of the three orientations depend on the type of robot and eventually the orientation definition selected there (cf. documentation on the respective transformation rules).

Using Coordinate Systems

WC_UR is a predefined constant of the type **WC_FRAME**. All its six components have a value of 0. If this constant is assigned to **WC_SYSTEM**, all offsets and rotations with regard to the world coordinate system become zero.

Activate *Variables, declaration, assignment*

```
<KinematicsName>.WC_FRAME : <WC-VarName>
```

<WC-VarName> is the freely-selectable name of the workpiece coordinate system. Values can now be assigned to the declared variable <WC-VarName> of type **WC_FRAME** in the RCL program. After valid values have been set, the variable <WC-VarName> has to be assigned to the command **WC_SYSTEM**. By processing the command via block processing, the workpiece coordinate system is defined.

Transformation world coordinate system → Workpiece coordinate system

```

;-----
; Transformation world coordinate system -> Workpiece coordinate system
;-----
; Declaration of different workpiece coordinate systems
WC_FRAME      : pallet1, pallet2
REAL          : Comp1, Comp2, Comp3, Comp4, Comp5, Comp6
BEGIN
; Definition of the different workpiece coordinate systems
Pallet1 = WC_FRAME(100,100,0,0,0,0)
  Comp1=50
  Comp2=50
  Comp3=75
Pallet2 = WC_FRAME(Comp1,Comp2,Comp3,Comp4,Comp5,Comp6)
; Activating the workpiece coordinate system of Pallet1
WC_SYSTEM = Pallet1

```

The coordinates in the KinData interface

`KinData[x].arPositionWorkpieceCoords_i`

refer to the workpiece coordinate system defined relatively to the world coordinate system with pallet 1.

Deactivate An active workpiece coordinate system is deactivated as follows.

WC_SYSTEM = WC_UR

With this assignment, all six components of the **WC_SYSTEM** command are assigned with a value of 0. After deactivation, the workpiece coordinate system is identical to the world coordinate system.

Deactivating the workpiece coordinate system

```
WC_SYSTEM = WC_UR
```

The coordinates in the KinData interface

`KinData[x].arPositionWorkpieceCoords_i`

are equal to the coordinates

`KinData[x].arPositionOriginalCoords_i`.

Restrictions A rotation of the workpiece coordinate system around a non-existent kinematic axis is not supported.

3.4 Workpiece Coordinate File

Describing the function Alternatively to direct programming, there is the ASCII system file **WCSYST.DAT** in which all local workpiece coordinates are defined by name

Create

Using Coordinate Systems

This file - if not available - can be created via the context menu of the "Global Data" folder. The "Global Data" folder is located below the robot node in the Project Explorer.

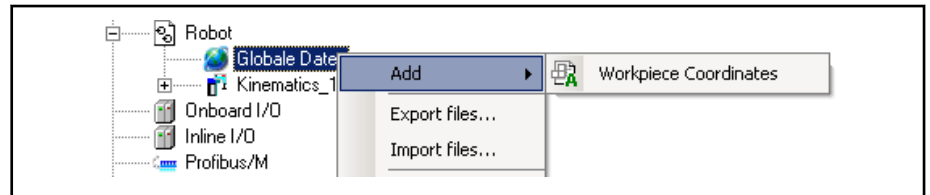


Fig.3-4: Creating workpiece coordinate file

The "Global Data" folder can be deleted at any time and created again via the context menu of the "Robot" node.



When deleting the "Global Data" folder, all files stored in this folder are also deleted!

Edit

The editor to enter the input or to edit the workpiece coordinates can be opened via the context menu of the "WCSYST.DAT" file

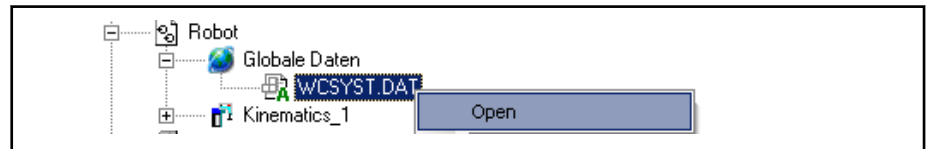


Fig.3-5:

3.5 Workpiece Coordinate Editor

General Workpiece coordinates can be entered and edited with the editor.

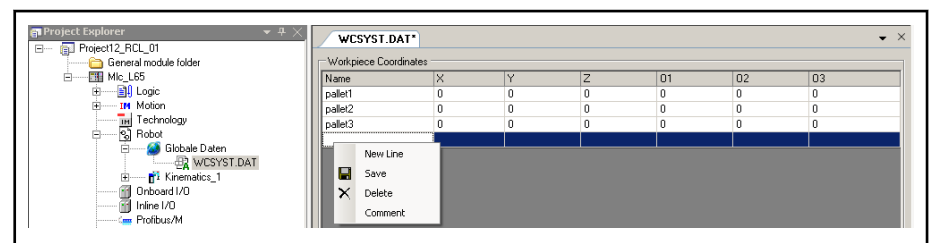


Fig.3-6: Editor

Any name can be given to the individual world coordinates. The respective coordinate values are stored under this name in the "WCSYST.DAT" file.



The name may not exceed twelve characters

For coordinate values, only the entries "0" ... "9", ".", "+", and "-" are allowed

The editor provides the following functions:

- Adding new line
- Saving workpiece coordinates to file
- Deleting line
- Entering line comment

Using Coordinate Systems

3.6 Library Functions

The class 2000 of the RCL library functions includes library functions facilitating the general usage of world coordinate systems. For a detailed description of the parameter lists, refer to the "rk_En.inc" header file.

rKxGWcSystem Provides name and coordinates of the active WC system. A structure of type "TKxGWcSystem" has to be transferred to the function. This structure looks as follows:

Transfer parameter "TKxGWcSystem" of the "rKxGWcSystem()" function

Name	Type	Data type	Meaning
ChannelId	Input	INTEGER	0 has to be set as channel ID.
KinNr	Input	INTEGER	Kinematic number Integers from 1 to n are allowed. The maximum kinematic number allowed depends on the setting of the parameter C-0-2484
WcSysCoord	Output	FARRAY_WC	Array contains six coordinate values of the workpiece coordinate system
WcSysName	Output	NAMSTRING	Name of the active workpiece coordinate system
Reserved	Reserved	ARRAY[1..32] IN-INTEGER	Reserved

Fig. 3-7: Transfer parameter of the "rKxGWcSystem()" function

Possible return values of the function

Value	Meaning
RK_NO_ERROR	No error. The function was executed correctly
RK_KIN_NR	Invalid kinematic number

Fig. 3-8: Possible return values of the function

rKSWcSystem Activates the WC system selected by name. Name and respective coordinates values have to be defined in the system file "RKSYS.DAT".

⚠ CAUTION

When using this function, the kinematic properties are changed (Tool Center Point).

The user is responsible for taking the relevant safety measures and to comply with the applicable accident prevention regulations in order to protect individuals and equipment from any kind of hazard. The user is solely responsible for the appropriate usage.

A structure of type "TKSWcSystem" has to be transferred to the function. This structure looks as follows:

Transfer parameter "TKSWcSystem" of the "rKSWcSystem()" function

Name	Type	Data type	Meaning
ChannelId	Input	INTEGER	0 has to be set as channel ID.

Using Coordinate Systems

KinNr	Input	INTEGER	Kinematic number Integers from 1 to n are allowed. The maximum kinematic number allowed depends on the setting of the parameter C-0-2484
WcSysName	Input	NAMSTRING	Name of the workpiece coordinate system to be activated The function searches for the name in the WCSYST.DAT file
Reserved	Reserved	ARRAY[1..32] INTEGER	Reserved

Fig. 3-9: Transfer parameter of the "rKSWcSystem()" function

Possible return values of the function

Value	Meaning
RK_NO_ERROR	No error. The function was executed correctly
RK_KIN_NR	Invalid kinematic number
RK_HEAPOVERF	Heap overflow
RK_TOOL_FILE	Error in WCSYST.DAT: Name of the workpiece coordinate system missing in "WCSYST.DAT" or "WCSYST.DAT"
RK_WRONGK43	Invalid machine parameter K-0-0043

Fig. 3-10: Possible return values of the function

rKCWpFrame

Calculated from the three positions P0, P1, P2, accepted in coordinates of the original coordinate system WC_{or}, the respective zero point of the local WC system WC_{loc} including rotation to the original coordinate system.



The function can only be used for kinematics with all six degrees of freedom. That means kinematics with at least six axes. Otherwise, the problem is mathematically underdetermined and output as return code (RK_AX_NR = -2).

The following prerequisites have to be met for the calculation:

- The three points P0, P1 and P2 are located on one pallet level. That means that they have the same z_{loc} coordinate.
- The vector P0P1 is parallel to the x-axis.
- The coordinates (x, y, z) of point P0 in the local WC system WC_{loc} have to be known.

Using Coordinate Systems

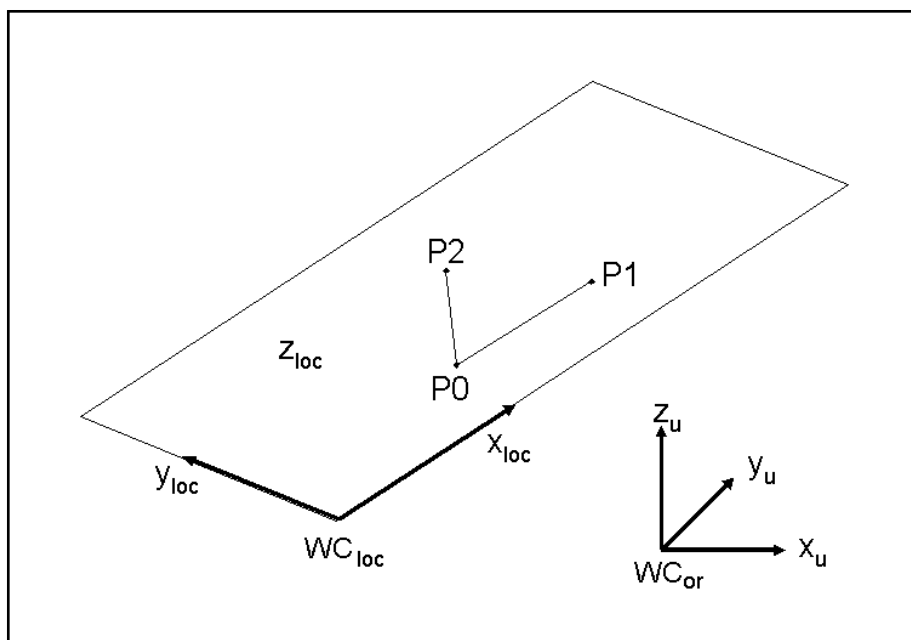


Fig.3-11: Parameters of the "rKCWpFrame" function shown schematically

Transfer parameter "TKCWpFrame" of the "rKCWpFrame()" function

Name	Type	Data type	Meaning
ChannelId	Input	INTEGER	0 has to be set as channel ID.
KinNr	Input	INTEGER	Kinematic number Integers from 1 to n are allowed. The maximum kinematic number allowed depends on the setting of the parameter C-0-2484
P0loc	Input	FARRAY_AXIS	Position of point P0; local workpiece coordinates
P0base	Input	FARRAY_AXIS	Position of point P0, robot original coordinates
P1base	Input	FARRAY_AXIS	Position of point P1, robot original coordinates
P2base	Input	FARRAY_AXIS	Position of point P2; robot original coordinates

Using Coordinate Systems

WpFrame	Output	FARRAY_WC	Coordinates of the local workpiece coordinate system. The array contains six values writing the position (three values) and the orientation (three values) of the local workpiece coordinate system in units of the robot original coordinate system. The basic level of the workpiece coordinate system is spanned by the three points P0, P1 and P2.
Reserved	Reserved	ARRAY[1..32] IN-TEGER	Reserved

Fig. 3-12: Transfer parameter of the "rKCWpFrame()" function

Possible return values of the function

Value	Meaning
RK_NO_ERROR	No error. The function was executed correctly
RK_KIN_NR	Invalid kinematic number
RK_AX_NR	Kinematics is not provided with all six degrees of freedom
RK_ANGLENOTC	Angles cannot be calculated. The points P0, P1, P2 are not suitable

Fig. 3-13: Possible return values of the function

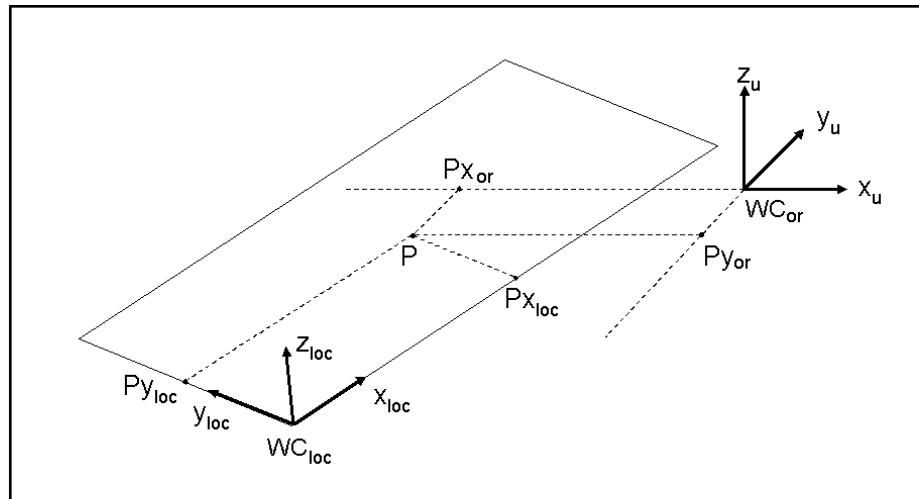
rKTrfLocOri Calculates the respective original coordinates from the local coordinates of a point.



The library function can only be used for 6-axis kinematics. The function is underdetermined at missing degrees of freedom.

The function works name-oriented with regard to the local workpiece coordinate system. The name with the respective coordinates is expected in the system file WCSYST.DAT. P_{loc} corresponds to the point in the coordinates of the local workpiece coordinate system. P_{or} is the same point in the coordinates of the original coordinate system WC_{or} .

Using Coordinate Systems

Fig.3-14: Point P in the coordinate system WC_{loc} and WC_{or} shown schematically

Transfer parameter "TKTrfLocOri" of the "rKTrfLocOri()" function

Name	Type	Data type	Meaning
ChannelId	Input	INTEGER	0 has to be set as channel ID.
KinNr	Input	INTEGER	Kinematic number Integers from 1 to n are allowed. The maximum kinematic number allowed depends on the setting of the parameter C-0-2484
PntLoc	Input	FARRAY_AXIS	Point in the coordinates of the local workpiece coordinate system "WcSysName"
WcSysName	Input	NAMSTRING	Name of the local workpiece coordinate system The function searches for the name in the WCSYST.DAT system file
PntOri	Output	FARRAY_AXIS	Respective original coordinates of the point "PntLoc". That means transformation of the point "PntLoc" to the original system of the robot.
Reserved	Reserved	ARRAY[1..32] INTEGER	Reserved

Fig.3-15: Transfer parameter of the "rKTrfLocOri()" function

Possible return values of the function

Value	Meaning
RK_NO_ERROR	No error. The function was executed correctly

Using Coordinate Systems

RK_KIN_NR	Invalid kinematic number
RK_TOOL_FILE	Error in WCSYST.DAT: Name of the workpiece coordinate system missing in "WCSYST.DAT" or "WCSYST.DAT"

Fig. 3-16: Possible return values of the function

rKTrfOriLoc

Calculates the respective local coordinates from the original coordinates of a point.



The library function can only be used for 6-axis kinematics. The function is underdetermined at missing degrees of freedom.

The function works name-oriented with regard to the local workpiece coordinate system. The name with the respective coordinates is expected in the system file WCSYST.DAT. P_{loc} corresponds to the point in the coordinates of the local workpiece coordinate system. P_{or} is the same point in the coordinates of the original coordinate system WC_{or} .

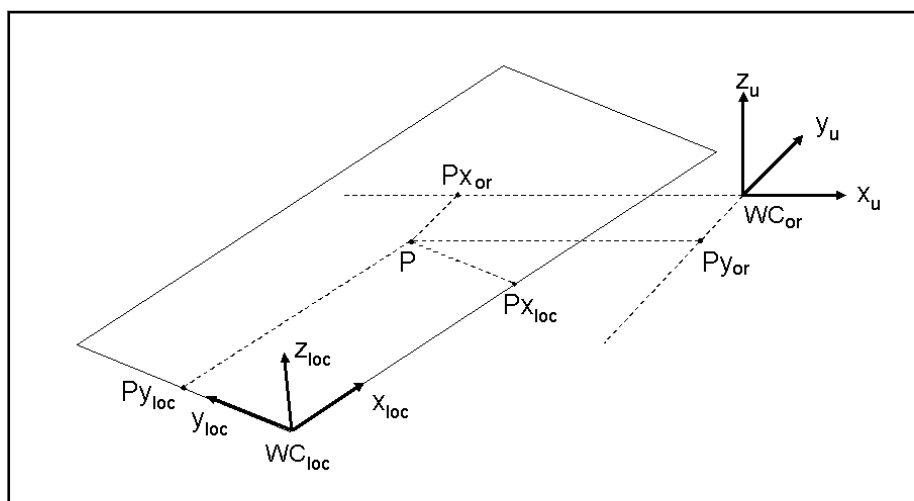


Fig. 3-17: Point P in the coordinate system WC_{loc} and WC_{or} shown schematically

Transfer parameter "TKTrfOriLoc" of the "rKTrfOriLoc()" function

Name	Type	Data type	Meaning
ChannelId	Input	INTEGER	0 has to be set as channel ID.
KinNr	Input	INTEGER	Kinematic number Integers from 1 to n are allowed. The maximum kinematic number allowed depends on the setting of the parameter C-0-2484
PntOri	Input	FARRAY_AXIS	Point in original coordinates of the robot
WcSysName	Input	NAMSTRING	Name of the local workpiece coordinate system The function searches for the name in the WCSYST.DAT system file

Using Coordinate Systems

PntLoc	Output	FARRAY_AXIS	Respective coordinates of the point "PntOri" in values of the workpiece coordinate system. That means transformation of the point "PntOri" to the local workpiece coordinate systems.
Reserved	Reserved	ARRAY[1..32] INTEGER	Reserved

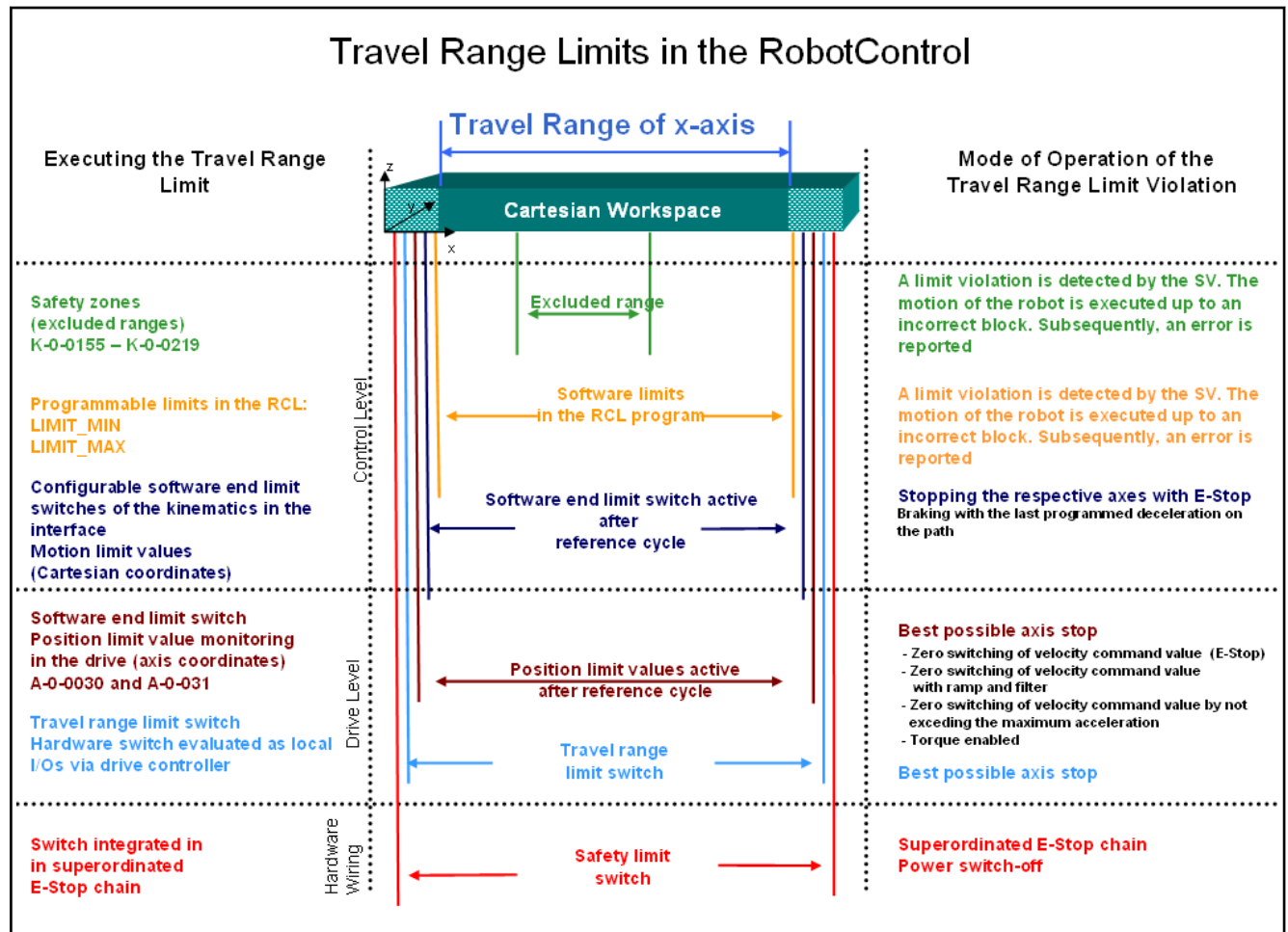
Fig. 3-18: Transfer parameter of the "rkTrfOriLoc()" function

Possible return values of the function

Value	Meaning
RK_NO_ERROR	No error. The function was executed correctly
RK_KIN_NR	Invalid kinematic number
RK_TOOL_FILE	Error in WCSYST.DAT: Name of the workpiece coordinate system missing in "WCSYST.DAT" or "WCSYST.DAT"

Fig. 3-19: Possible return values of the function

4 Travel Range Limits in the RobotControl



SV
Fig.4-1:

Block preparation
Travel range limits

5 RCL Program Structure

5.1 General Information

RCL stands for "Robot Control Language". RCL is a task-oriented program language used for programming the MLC control family.

As a programming language for robot and handling systems, RCL is an extensive, but easy to learn language. Tasks can be quickly and easily completed.

The language commands are available in English.

In this document, the general syntax of every statement is given before each detailed statement.

The following symbols are used for description purposes:

CAPITAL LETTERS	means part of the language element, i.e. an RCL keyword. It has to be written exactly like this
{ }	Can optionally be specified several times
[]	Can optionally only be specified once



Symbolic identifiers such as variable names may not exceed 12 characters.

The first character has to be a letter. Sequences of letters and digits are allowed. Umlauts are not allowed.

5.2 Mode of the Compiler Operation

5.2.1 Use

The RCL Compiler can be used in two different ways:

- An editor is provided in IndraWorks to create and compile an RCL program fast and easily.
- If a source file already exists (e.g. created with a text editor), the compiler can be started via the parameters K-0-0221 and K-0-1062.

5.2.2 Source Files

qll file This text file contains the RCL program code.
File name and program name have to be identical.

5.2.3 Output Files

The output files are created during the compilation. The name of the output files consists of the names of the qll file with the respective extension (e.g. "Prog1.qll" comes to "Prog1.ird").

ird file This file contains the program code to be processed by the control.
Additionally, the storage area required for the variables used in the program is determined. Storage location is also reserved in this file for point variables which are not declared with DEF and to which a value is assigned in the program.

pkt file This file is only generated if the program has been compiled without errors.
In this file, the storage area is reserved for point variables.
Storage is reserved for point variables

RCL Program Structure

- declared in the program with DEF.
 - not declared.
 - to which no value has been assigned in the program.
- sym file** This file contains information on the variable names used in the program.
This file is required for debugging.
- err file** This text file contains an overview on the errors that occurred during the compilation.

5.3 Compiler Statements

5.3.1 General Information

The qll file contains compiler statements. They control the compiler and reduce the time and effort required for writing.

The compiler statement always starts with two semicolons.

Compiler Statements:

```
;;COMPILER = RCL3 ; RCL Compiler Version
;;INCLUDE name ; optional
;;[kin_name.]INT = CIRCULAR | PTP | LINEAR
;;KINEMATICS: ({ integer_constant = kin_variable | , })
;;KINEMATICS = kin_variable
;;(kin_variable.JC_NAMES | WC_NAMES) = { name || , }
;;DEBUGINFO [+/-] ; optional
```

The compiler statement for the axis names and the target control has to be placed in front of the first source symbol, i.e. in front of the program declaration.

5.3.2 Kinematic Definition

In the MLC 04V12 or higher, up to 16 kinematics can be supported for each control instead of only one (see also [Using the RoCo Multi-Kinematic Technology, page 215](#)).

Each of these kinematics is controlled by its own program.

Only one kinematics should be assigned to each positioning and orientation axis.

Belt axes can be assigned to several kinematics.

Depending on the technology used, axes and belts can be transferred from one kinematics to another and they can be controlled.

5.3.3 World Coordinate Names (WC Name Definition)

World coordinate points contain the components for the position, the orientation and, if required, the belt coordinate(s). The component names can be defined.

Syntax

```
;;WC_NAMES=x_k,y_k,z_k,a_k ;WC_NAMES=wc_name,...
```

Example with kinematic names

```
;;Kin_1.WC_NAMES=x_k,y_k,z_k,a_k ;kinematics name.WC_NAMES=WC name,...
```



The coordinate declaration has to be in one line and may not be interrupted by a word wrap.

When creating a kinematics in IndraWorks, the names for the WC axes are entered. When a new qll file is generated, WC axes are automatically created.

They can be modified later on.

5.3.4 Axis Names (JC Name Definition)

The machine coordinate points contain the components for the individual axes and, if required, for the belt coordinate(s).

The axis names can be defined.

Syntax

```
;;JC_NAMES=a_1,a_2,a_3,a_4 ;JC_NAMES=JC name,...
```

Example with kinematic names

```
;;Kin_1.JC_NAMES=a_1,a_2,a_3,a_4
```



The axis name declaration has to be in one line and may not be interrupted by a word wrap.

When a new qll file is generated, the axis names are automatically accepted by IndraWorks.

They can be modified later on.

5.3.5 Including Files

The compiler statement `;;INCLUDE 'filename'` includes sections of source programs in the current program.

Example

```
;;INCLUDE rcl ;rcl.qll, File for automatic inclusion
;of the declaration during compilation
INPUT REAL: 1=grip_contact,4=measheight ;Declaration of inputs
INPUT: 11=gateswitch_1,15=li_barrier
OUTPUT REAL: 1=print,2=met_unit ;Declaration of outputs
OUTPUT: 7=alarm
```

The "rcl.qll" file contains the declarations of your inputs and outputs for example. These are defined with regard to their

- Data type
- Channel number
- Variable name of the signal

For details, refer to [PLC Communication Channels, page 64](#).



The rcl.qll file has to be in the same directory as the other files to be compiled.

RCL Program Structure

5.3.6 Selectable Extension in the Include Statement

The extension of include files can be freely selected. If only the file name is specified, the extension ".qll" is added due to compatibility purposes.

To include a file without extension, the file name has to end with a dot, refer to the "include statement" example below.

Syntax

```
;;INCLUDE filename      ;Compiler statement 'INCLUDE'
```

The following applies to the syntax above:

filename: File name to be included during compilation.

Example

```
;;INCLUDE head.inc
PROGRAM demo
;;INCLUDE constant      ;constant.qll file is included
;;INCLUDE types.INC     ;File types.INC are included
;;INCLUDE variable.     ;File variable. (without extension) is included
BEGIN

PROGRAM_END
```



The following files have to be stored in a directory: head.inc, demo.qll, constant.qll, types.inc and variable.



Include files cannot contain any INCLUDE commands (nesting is not possible).

5.3.7 Debug Information

For a more comfortable testing, the compiler statement ";;DEBUGINFO-" can be used to switch off the generation of debug information on the Irdata code.

In general, this is only practical for thoroughly tested application programs.

The additional debug information is missing in the Irdata code. The Irdata code is shorter and can be processed more quickly. However, debugging with the debugging system is no longer possible.

;;DEBUGINFO+ is preset.

Syntax

```
;;DEBUGINFO-      ;Test info generation is switched off
```

Further restrictions for "DEBUGINFO-":

Possible program errors are also displayed without line numbers.

Use ";;DEBUGINFO+" to switch on the generation of debug information again.

5.4 Main Program Structure

5.4.1 General

Each main program consists of

- declarations
- statements (instructions) and optional
- subroutine declaration(s) (subroutines)

RCL Program Structure

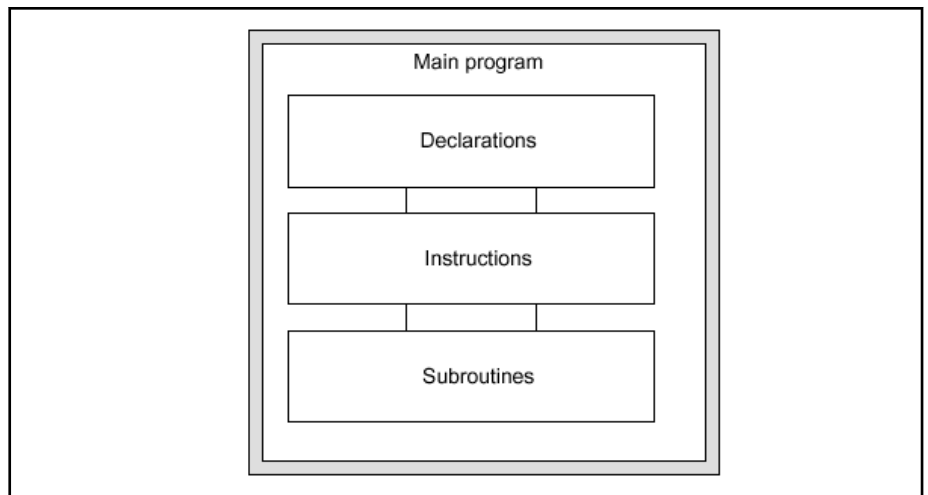


Fig. 5-1: Main program structure

5.4.2 Declaration Part

The declaration is located at the beginning of the main program. In the declaration, the names which occur in the main program are declared.

This applies to the:

- program head
name of the main program
- external declaration
names of the external main programs called in the main program
- channel declarations
names of the input and output channels used in the program
- variable and constant declaration
names of variables and constants in the program

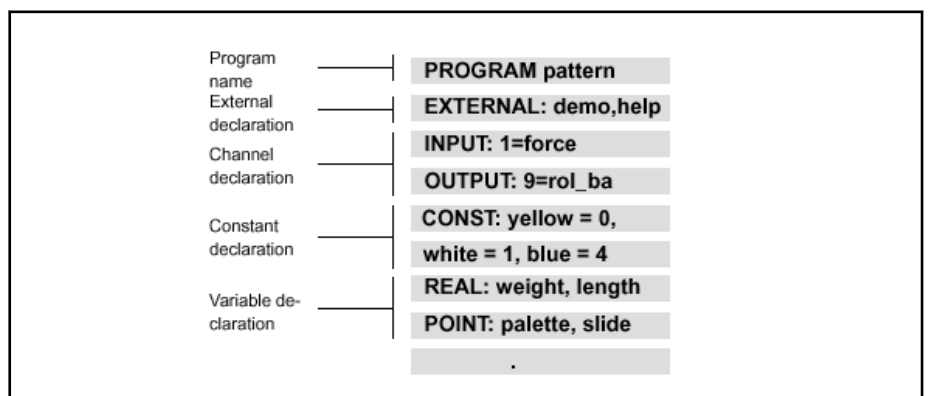


Fig. 5-2: Declaration part of the main program

Optionally, the keyword "VAR" can introduce the declaration part of variables.

RCL Program Structure

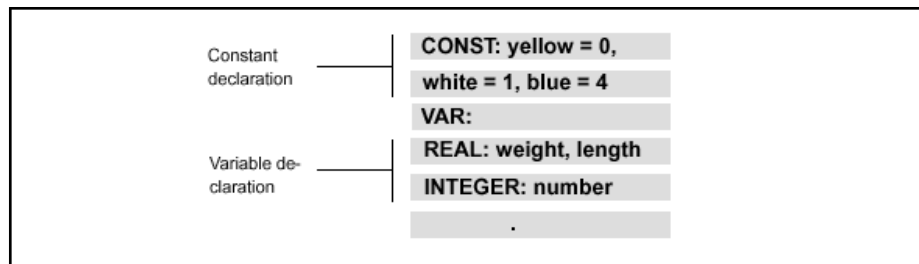


Fig.5-3: Declaration of variables

The keyword "BEGIN" has to be used to separate the statement from the declaration.



Undeclared variables are considered to be "POINT" or "JC_POINT" variables.

5.4.3 Statement Part

In the statement part, the statements to be executed are programmed.

For example:

- motion commands
- decelerations and stop
- main program calls
- subroutine calls
- program part repetitions
- program jumps
- arithmetical operations and
- function calls

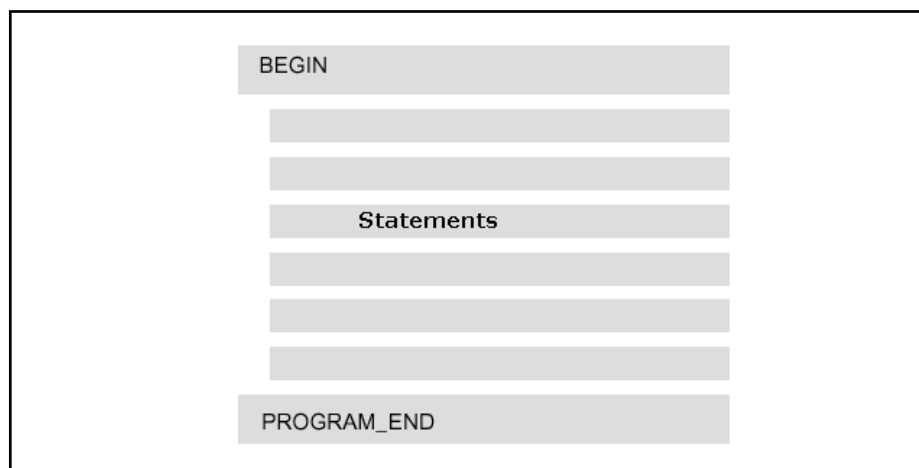


Fig.5-4: Statement part

The statement is located between the keywords "BEGIN" and "PROGRAM_END".

5.5 Program Declaration

A main program is identified at the beginning by the RCL word "PROGRAM" and its program name.

The program name consists of a maximum of eight characters. Letters, digits and underscores are permitted.

RCL Program Structure



The first character has to be a letter. Upper case and lower case letters are of equal value

The program end is identified by the RCL word "PROGRAM_END" or "SUB_END" if subroutines are listed.



The program name and the file name have to be identical.

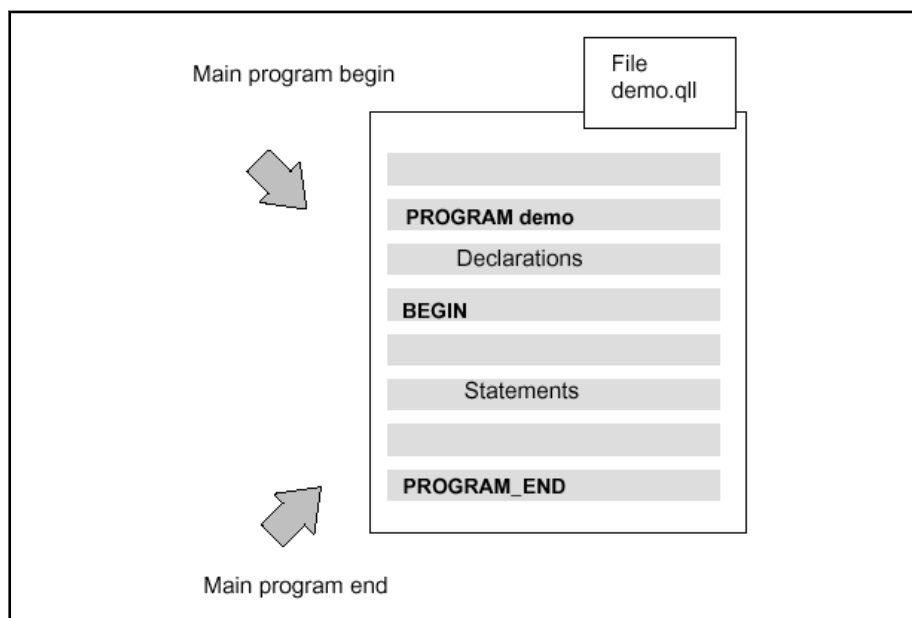


Fig.5-5: Main program structure

5.6 Main Program Call in Main Program

External declaration:

A main program can consist of several external individual main programs. Each main program is a separate QLL file.

The external main programs has to be declared with "EXTERNAL" at the beginning of the declaration part and has to be available in the control as ird file.

Transfer parameters for external main programs are optional. The number, order and data types has to match the declaration at parameter transfer.

A parameter can be transferred as value or reference. When a parameter is transferred as value, the keyword "VALUE" has to be used. This type of parameter transfer only transfers the value. When a parameter is transferred as reference and not as value, a reference on the value is transferred. Thus, it is possible to change transferred parameters in an external program and to process this change in the calling program. The following data types are allowed as parameters.

Types for parameters of external programs

RCL definition	Transfer type
VALUE CHAR	Character as value
VALUE REAL	Floating point number as value
VALUE BINARY	Binary number as value

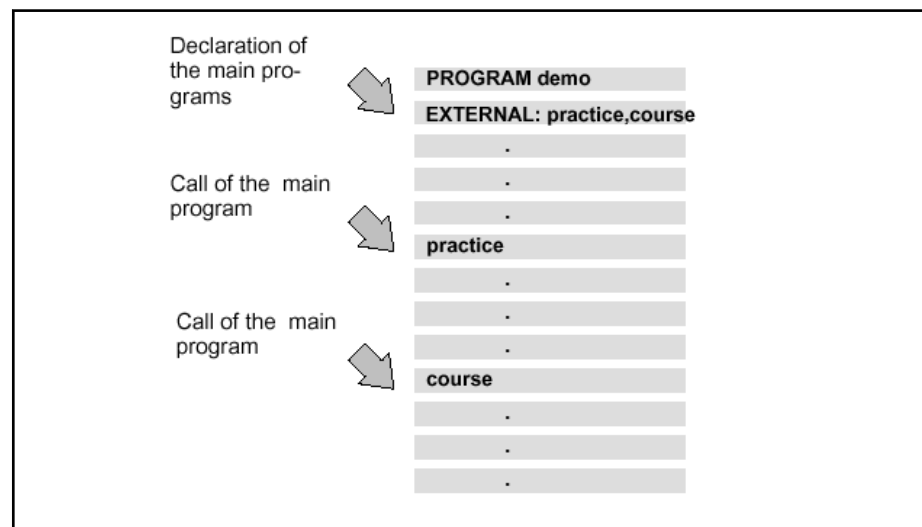
RCL Program Structure

RCL definition	Transfer type
VALUE INTEGER	Integer as value
VALUE TEXT	Character string as value
VALUE POINT	World coordinates as value
VALUE JC_POINT	Joint coordinates as value
CHAR	Character as reference
REAL	Floating point number as reference
BINARY	Binary number as reference
INTEGER	Integer as reference
POINT	World coordinates as reference
JC_POINT	Joint coordinates as reference

Fig.5-6: *Types for parameters of external programs*

External main programs with transfer parameters cannot be started as independent programs. Instead, they have to be started by a program call of a superordinate main program.

Only the specification of the declared program name is required to call external programs in the active main program.

Fig.5-7: *Working with several main programs*

The main program and the external program to be called are compiled independently of each other. During compilation, the transferred parameters cannot be checked with respect to their matching with the declaration in the external main program. It is performed, when the program is running. The number, types, order and nature (VALUE or reference) of the transfer parameters have to correspond to the declaration of the called external main program.

Programming

The program names of the external main programs are declared with the statement "EXTERNAL:"

Example

```
EXTERNAL: practice,course
```

RCL Program Structure

External declaration with parameter transfer

EXTERNAL: withpar(VALUE INTEGER: number REAL: r2)

It is defined that a main program exists. The first parameter transfers an integer as reference and the second parameter transfers a floating point number as value. The main program has to be declared accordingly.

Main Program

PROGRAM withpar(VALUE INTEGER: number REAL: r2)

Main program call in the main program

The control executes the active main program up to the external program call.

This is followed by a jump to the start of the program "practice".

The program "practice" is executed up to the "HALT" statement (or up to "PROGRAM_END").

"HALT" (or "PROGRAM_END") triggers a return to the main program "demo".

The control continues the program run with the statement following the call.

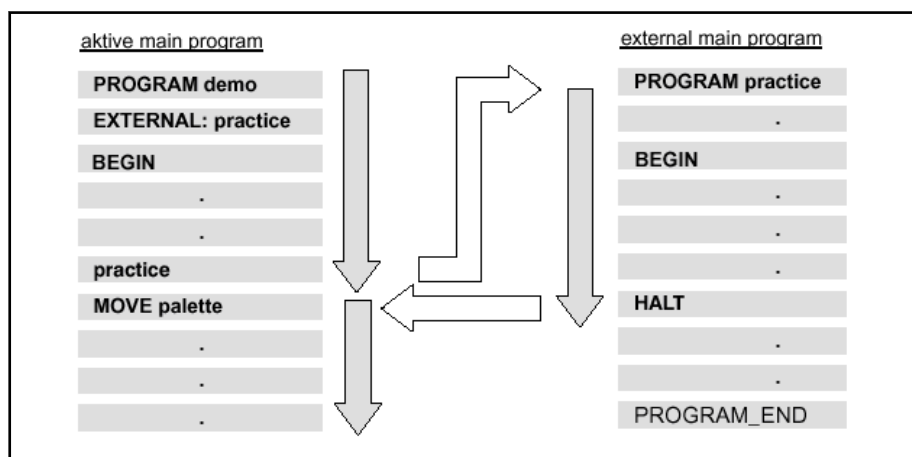


Fig. 5-8: Sequence of a main program call in the main program

Effects of global specifications

If a main program is called externally, the current global specifications such as V, A, J or the slope type are applied. Changes of these values in the calling program are not transferred to the calling program at the return from the program. For example:

RCL Program Structure

PROGRAM demo	PROGRAM movetest
EXTERNAL: movetest	.
BEGIN	BEGIN
PROGR_SLOPE	.
A = 10000	MOVE LINEAR TO P2
V = 1000	.
.	BLOCK_SLOPE
MOVE LINEAR TO P1	A = 999
.	V = 99
movetest	.
.	MOVE LINEAR TO P3
MOVE LINEAR TO P4	.
.	PROGRAM_END
PROGRAM_END	

Fig.5-9: Example of an external program call with changed global specifications

The following table shows the global specifications used to approach the individual points.

Global specifications for the motion

Point	A	V	Slope type
P1	10000	1000	PROGR_SLOPE
P2	10000	1000	PROGR_SLOPE
P3	999	999	BLOCK_SLOPE
P4	10000	1000	PROGR_SLOPE

Fig.5-10: Global specifications for the motion

5.7 Subroutine Declaration

If the same work procedures are to be performed at different points in the program, these procedures can be combined into subroutines. With these subroutines, storage area is saved while increasing the user-friendliness of the program.

Variables defined in the main program, i.e. global variables, can also be processed in the subroutine.

Variables defined in the subroutine, i.e. local variables, can only be processed in the subroutine. There is no transfer to the main program.

The subroutine declarations are located after the main program after the PROGRAM_END statement.

Identification A subroutine is identified at the start by the word "SUBROUTINE" and the subroutine name.

The subroutine name contains a maximum of twelve characters. Letters and digits are permitted.

RCL Program Structure



The first character has to be a letter. Upper case and lower case letters are of equal value.

The subroutine is ended by the RCL keyword "SUB_END"

Programming

A subroutine contains statements for the gripper and should get the name "gripper".

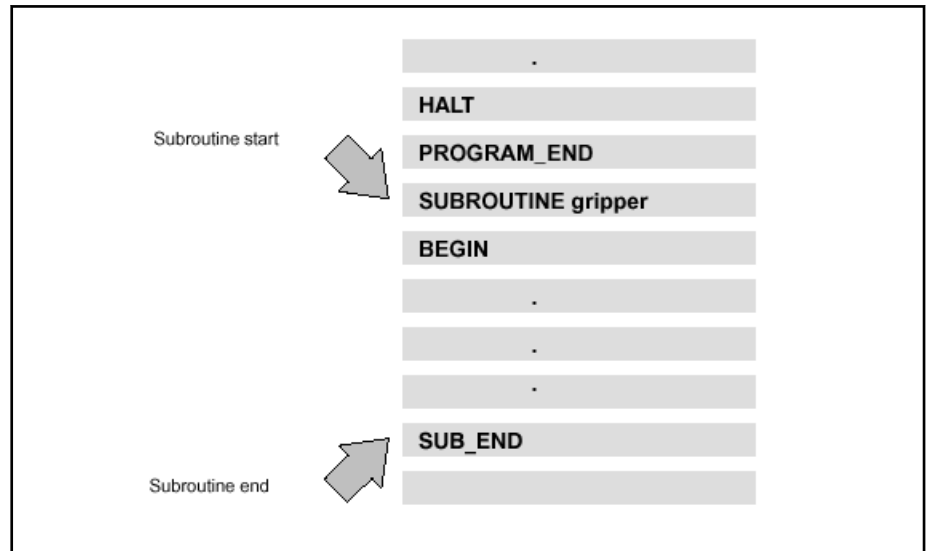


Fig.5-11: Position of the subroutine after the main program



"RETURN" can be used several times in a subroutine, e.g. with program jumps and conditional statements.

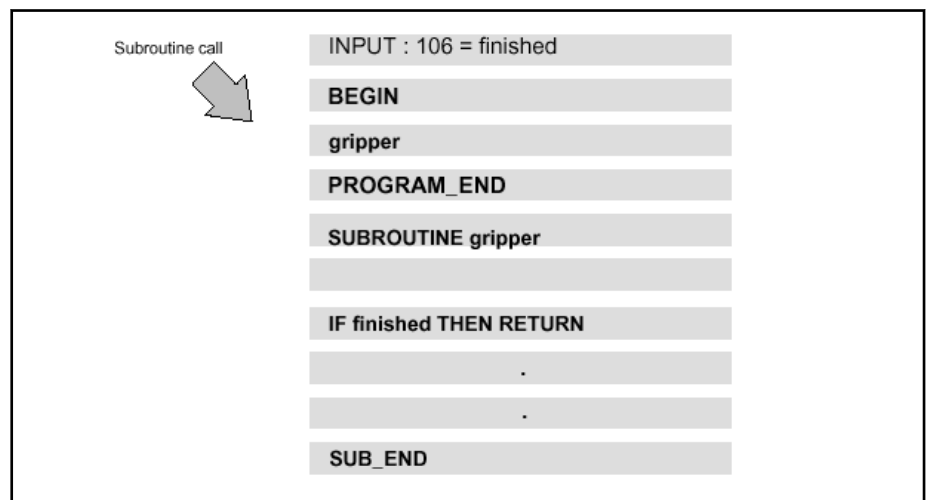


Fig.5-12: Subroutine call and conditional return

If the subroutine return can be identified from the program structure, e.g. at the end of the subroutine, the compiler automatically generates the command "RETURN".

Subroutine call

Only the declared subroutine name has to be specified, "gripper" in the example, for the subroutine call.

RCL Program Structure



The program and the subroutine to be called are compiled independently of each other. During compilation, the transferred parameters are not checked for matching. It is performed, when the program is running. The number, types, order and nature ("VALUE" or addressing) of the transfer parameters have to correspond to the declaration of the called external main program.

5.8 Subroutine Sequence

The control processes the main program up to a subroutine call.

This is followed by a jump to the start of the called subroutine. The subroutine is processed up to RETURN (or SUB_END). The command "RETURN" triggers a return to the main program.

The control continues the program run with the statement following the subroutine call.

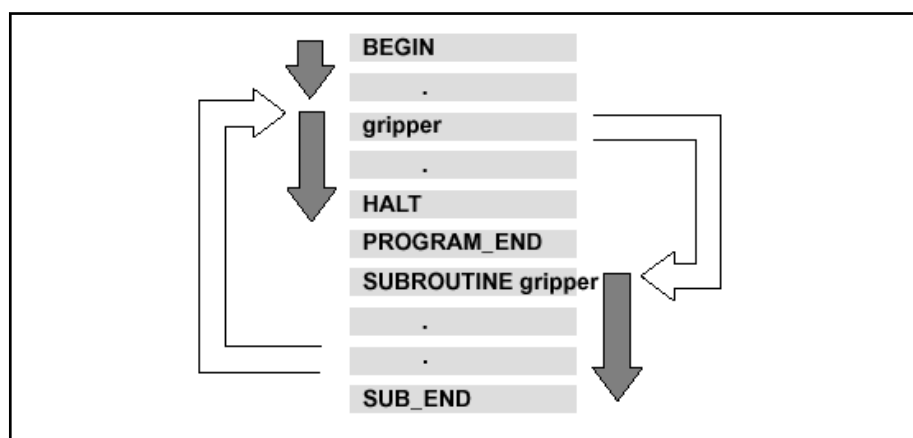


Fig.5-13: Program execution, main program/subroutine

Transfer variables or values with a subroutine call. For this purpose, the variables have to be declared accordingly in the subroutine declaration.

Example

```
gripper (1.5,6.0)           ;Value assignment
SUBROUTINE grip(VALUE REAL: force1,force2) ;when calling the subroutine,
                                ;Variable force1 has the value 1.5 and
                                ;Variable force2 has the value 6.0.
```

The value is transferred if the declaration includes the preceding RCL command "VALUE". Otherwise, the address of the variable is transferred, i.e. the variable has to be declared in the main program.

If the declaration is preceded by "VALUE", the calling program transfers information to the called subroutine. However, the called subroutine does not return information this way.

If the address is transferred, the assignments in the subroutine also apply to this variable after returning to the calling program.

Example

```
;;INCLUDE head           ;Include the compiler statements such as control type, TESTINFO+ etc.
PROGRAM demo             ;from the "head" file into the program
REAL: x                  ;Test program for subroutine with parameter transfer
BEGIN                   ;Variable declaration
  x=5
  anix(X)                ;Calling of the subroutine anix
                          ; x has the value 10
  awas(X)                ;Calling of the subroutine awas
                          ; x still has the value 10
```

RCL Program Structure

```

HALT
PROGRAM_END
SUBROUTINE awas(REAL: aw) ;Subroutine declaration awas. Transferring the value and the
                           ;variable. The subroutine calculates the variable and
                           ;returns it to the main routine.

BEGIN
  aw=aw*2                  ; aw and x have the value 10 now
  RETURN
SUB_END

SUBROUTINE anix(VALUE REAL: aw) ;Subroutine declaration anix. Transferring the value and the
                                ;Variable x remains unchanged in the main routine.
                                ;The subroutine processes the aw variable (individual storage location).
                                ;After exiting the subroutine, the variable remains unchanged
                                ;in the main routine.

BEGIN
  aw=aw*2                  ;aw has the value 20 and x has the value 10
  RETURN
SUB_END

```

Nesting Additional main program calls or subroutine calls can be programmed in called main programs or subroutines. This is called nesting.

Program examples of subroutine nesting

- A call of the subroutine "stacker" is programmed in the main program.
- In the subroutine "stacker", a further subroutine call is programmed, the subroutine "gripper".
- The control executes the main program up to the "stacker" call.
- This is followed by a jump to the subroutine "stacker".
- The control executes the subroutine "stacker" up to the call "gripper".
- This is followed by a jump to the next subroutine "gripper".
- The subroutine "gripper" is executed completely in the example shown.

With the "RETURN" command, the control jumps back to the subroutine "stacker", continues the program run up to "RETURN" and finally jumps back to the main program.



In case of nesting programs and subroutines, ensure that no end-less loops are created.

Any nesting depth is possible. The nesting depth is only limited by the stack size (4 KB).

RCL Program Structure

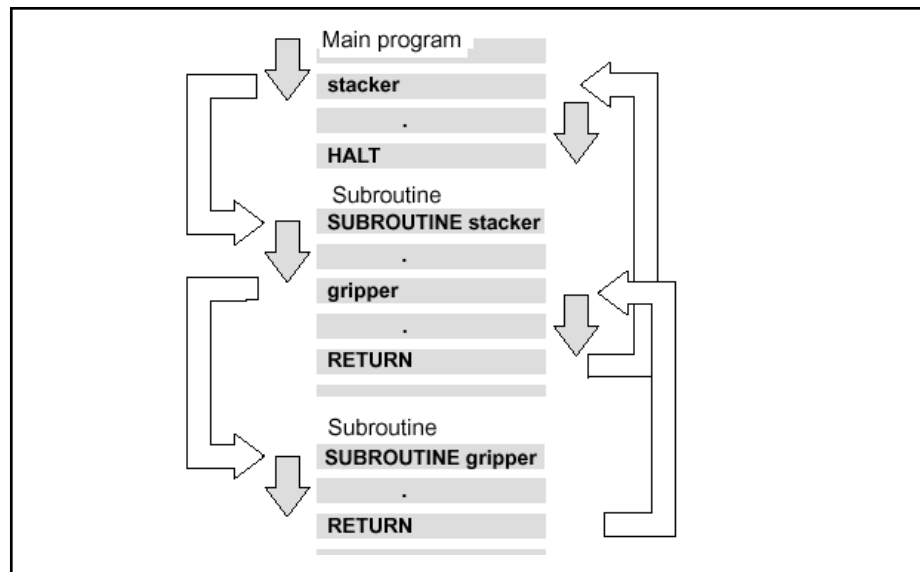


Fig.5-14: Example 1

Program part repetitions can also be nested. A second repetition is programmed in a program part repetition.

The program is executed as follows:

- The control executes the program part once up to the start of the second program part.
- The second program part is repeated three times.
- Subsequently, the control continues the program run up to the end of the first program part. The first program part has been executed once.
- For the second run, the control returns to the start of the first program part.
- The entire sequence is repeated a second time.

Any nesting depth is possible. The nesting depth is only limited by the stack size (4 KB).

RCL Program Structure

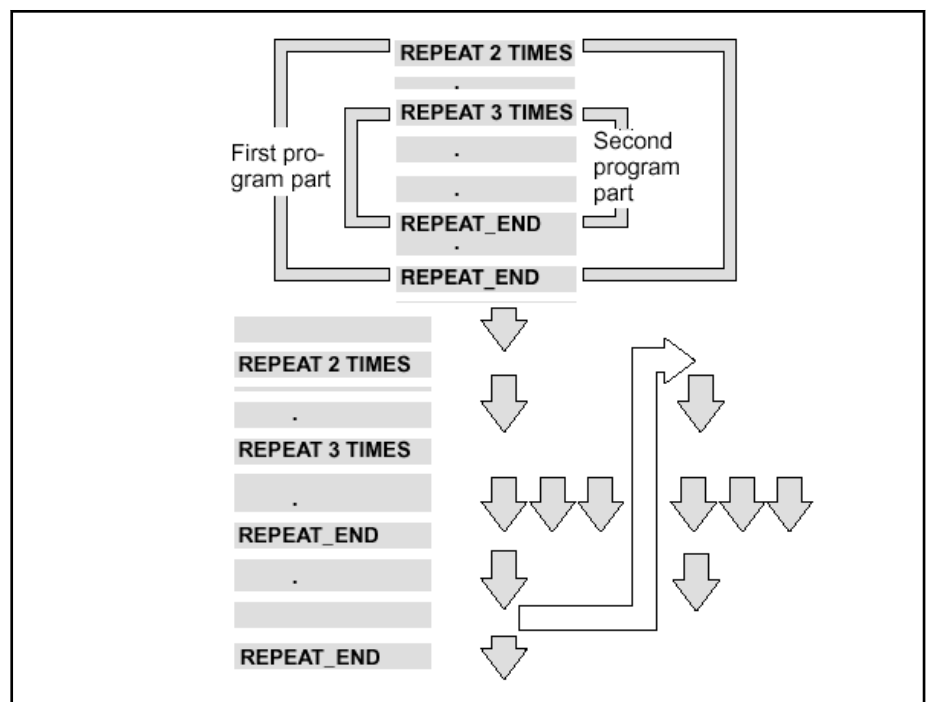


Fig. 5-15: Example 2

6 RCL Constants

6.1 RCL Constant Declaration

Within one program, numeric values, e.g. the number of repetitions, character sequences or characters can be specified by constants.

The value of a constant can also be specified by a computing rule.

The constants have to be defined in the constant declaration before the variables of a program are defined.

Example

```
CONST: red=1,yellow=2*red,green=2*yellow,lines=4,      ;Constants
      columns=20,limit_value=2.5*7.25

INTEGER: color, l_index, c_index                        ;Variables
REAL: meas_value
ARRAY[lines .. columns] REAL: distance

BEGIN....
IF meas_value > limit_value THEN ....                  ;Implementation
```

6.2 Predefined RCL Constants

Default constants are available in the RCL. These constants are part of the language scope and do thus not have to be declared.

WC_UR "WC_UR" is a constant of type WC_FRAME predefined in RCL. All its six components have the value 0.

If this constant is assigned to WC_SYSTEM, all displacements and rotations to the original coordinate system become zero, e.g. by WC_SYSTEM = WC_UR.

The workpiece coordinate system is then again equal to the original world coordinate system (Also refer to [Defining the Workpiece Coordinate System Relative to the World Coordinate System](#), page 23).

7 RCL Variables

7.1 General

Numerical values, e.g. coordinate values for positions or the number of repetitions can be replaced in a program by variables.

A variable reserves storage area under its name.

A numerical value can be assigned any number of times to this storage area. The control always stores the latest numerical value assigned.

If a variable name appears during the program run, the control replaces the variable with the value stored under its name at that point in time.

Each variable has a name. Different variables have to be provided with different names. If possible, the names should be chosen in a way so that the meaning of the variable can be identified, e.g. for position designations "bore_1" is better than "b1" or "counter" for arithmetic variables should be chosen over "i".



The name consists of 12 characters max.

Letters, digits and underscores "_" are permitted. There is no case sensitivity.

The first character has to be a letter.

Space characters are not permitted.

Point variables of type "JC_POINT" are an exception.

These variables have to start with "@".

7.2 RCL Data Types

7.2.1 Overview

The data type of a variable determines its value range and the permissible assignment and arithmetic operations. The permissible operations are listed in the section [Complete Value Assignment, page 59](#).

It has to be distinguished between simple data types and composed, structured data types.

Simple data types

- INTEGER
- REAL
- BINARY
- CHAR

Structured data types

- POINT
- JC_POINT
- BELT
- TEXT
- ARRAY
- WC_FRAME
- FILE.

A composed data type consists of two or more simple data types.

RCL Variables

7.2.2 Simple RCL Data Types

Variables are declared before using them in the declaration (before "BEGIN"). The keyword "VAR" separates this section from a type definition.

INTEGER Only positive or negative integers, i.e. numbers with a decimal point, can be assigned to "INTEGER" type variables.

Value range -2^{31} to $+(2^{31}-1)^{1)}$



The RCL data type INTEGER corresponds to the PLC data type DINT!

REAL Only positive or negative real numbers, i.e. numbers with a decimal point, can be assigned to "REAL" type variables.

Value range: 4 byte floating point number with simple accuracy²⁾.



The values for "REAL" type variables has to be entered with decimal points, not commas.

BINARY Only 0 or 1 may be assigned to variables of the type "BINARY".



Integral arithmetic operations cannot be performed with "BINARY" type variables.

The digits 0 and 1 do not represent any conventional values.

They describe two defined states, e.g.

Variable designation	0 (FALSE)	1 (TRUE)
Switch	Off	On
Question	No	Yes
Bowl	Empty	Full
Filled	False	True

Fig. 7-1: Interpretation of binary values

Variables of type "BINARY" are for example signals of binary input and output channels.

CHAR Variables of type "CHAR" may only be assigned to ASCII characters³⁾.

¹⁾ INTEGER type variables (long integers) are saved as 32 bit numbers (4 bytes) with signs in the range from -2,147,483,648 to 2,147,483,647.

²⁾ REAL data type variables (long integers) are saved as 32 bit numbers (4 bytes) -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.

³⁾ The ASCII (American Standard Code for Information Interchange) character set (7 bits) is used to represent letters and symbols used on a standard US keyboard. The ASCII character set corresponds to the first 128 characters (0-127) of the ANSI character set.

7.2.3 Composed RCL Data Types

POINT Variables of type "POINT" may only be assigned to positions in world coordinates. The individual coordinate values of a position of "POINT" type has to be of type "REAL".

JC_POINT Variables of type "JC_POINT" may only be assigned to positions in joint coordinates.
The individual coordinate values of a position of type "JC_POINT" has to be of type "REAL".



Variable names of type "JC_POINT" have to start with "@".

Variables of the types "POINT" and "JC_POINT" are also called point variables.

The number of components depends on the number of axes of the specified kinematics.

TEXT Variables of type "TEXT" may only be assigned to texts that consist of 80 ASCII characters max. The individual characters can be addressed directly like array elements using the index.

Text variables, declaration, assignment

```
TEXT: char_string           ;Declaration of text variables
char_string[1]='A'         ;Assigning individual components
char_string[2]='B'
char_string='This is ASCII text' ;Assigning a character string
```

FILE *Files, declaration*

FILE: cad_dat

A file name is defined with the "FILE" data type. It is used as a parameter when accessing with "WRITE"/"READ" or "OPEN"/"CLOSE".

7.2.4 User-Defined RCL Types

In the RCL, users can declare their own types by extending the declaration in an RCL program to include the type definition "TYPE".

ALIAS name New names can be assigned to any data types.

This applies to both standard types

- INTEGER
- REAL
- BINARY
- CHAR

as well as for the structured types based on the standard types.

- TEXT
- ARRAY
- RECORD

RCL Variables

Examples

```

PROGRAM types
TYPE:
  tmatrix=  ARRAY[0..7] ARRAY[0..7] INTEGER
  myinteger= INTEGER
;Type definition
;2 dim. ARRAY "tmatrix"
;Direct derivation from "INTEGER"

VAR:
  ARRAY [0..7] ARRAY[0..7] INTEGER: matrixone
  tmatrix: matrixtwo
  myinteger: integervar
;Variable declaration

BEGIN
  initold(matrixone)
  initnew(matrixtwo)
PROGRAM_END
SUBROUTINE initold(ARRAY[0..7] ARRAY[0..7] INTEGER: matrix) ;Long spelling
  BEGIN...
SUB_END
SUBROUTINE initnew(tmatrix: matrix) ;Short spelling
  BEGIN...
SUB_END

```

Record Types, RECORDs

Record types can be used to further improve the usability and structure of programs.

A record has a name and consists of 1..n variables (components).

Each component has its own name and data type.

A component itself can also be a record.

In PASCAL, these record types are known as "RECORD". In C, they are called "STRUCT".

Barcode identification example

Barcode information (r_barcode) consists of

- Name (name, TEXT)
- Weight (REAL)
- Volume (r_volume), itself a record of
 - Length (REAL)
 - Width (REAL)
 - Height (REAL)

Records are useful to organize complex data, particularly in large programs. They allow access to the complete data set and to each component.

The name appearing in the type declaration on the left side of the assignment character in front of the reserved word "RECORD" stands for the entire record and can be used as abbreviation for the detailed declaration.

Variables mentioned in a record are called components.

A component can have the same name as an ordinary variable without creating a conflict. They can always be distinguished by context.

A component can only be accessed using the record name. A dot separates the component name from the record name.

If the component represents a record, its components are also separated by a dot.

Example

```
PROGRAM records
TYPE:                                ;Type definition part

r_volume=RECORD
    REAL: lenght
    REAL: width
    REAL: height
RECORD_END

r_barcode=RECORD
    TEXT: name
    REAL: weight
    t_volume: volume
RECORD_END

r_barcode: code                      ;Variable declaration for the "code" of a packing
BEGIN
;
; ..
code.name='cuboid'                   ;Accessing a component
code.volume.lenght= 12.5             ;Accessing a subcomponent
code.weight= 155.5
; ..
PROGRAM_END
```

When using record variables in WRITE or READ statements, the following has to be complied with:

- Record variables can only be read or written to "FILE" type files (text files) component by component.
- Record variables can be read or written to "BNR_FILE" type files (binary files) either completely or component by component.

ARRAY Variables of the same type can also be combined in one array.

These variables have the same name; only their index is different. Thus, these variables are also called indexed variables.

Syntax

```
ARRAY[( [±] integer_constant ) ... ( [±] integer_constant )] <Type> : variablename
```

Example

```
ARRAY[-10..10] INTEGER: variablename
```

7.3 RCL Variable Declaration

Before executing a statement with variables, the control has to be informed which values or characters can be assigned to the variable.

Thus, every variable used in the program has to be declared first, i.e. the data type of the variable has to be defined.

Syntax

```
[DEF] Type: [channel =]name{ , [channel =]name }
```

- "DEF" can stand in front of the point variables "POINT" and "JC_POINT"
- "Type" has to be replaced by the actual type, e.g. "INTEGER".
- "channel" is only provided for the types "INPUT", "OUTPUT" or "BELT".

RCL Variables



All points declared with "DEF" and all points to which no value is assigned anywhere else in the program, are stored in a point file with the identification "pkt".

Points transferred to subroutines as parameters (spc.fct. and rho.fct) do not appear in the point file.

Programming The declaration consists of the data type and the variable names assigned. The data type is separated from the variable name by a colon. Several variable names of the same type are separated by commas.

Declarations can be made between:

- "PROGRAM" and "BEGIN" and
- "SUBROUTINE" and "BEGIN".

Example

```
PROGRAM Prog2
POINT: BasePnt
OUTPUT: 1=valve_1,2=valve_2
BINARY: ready
INTEGER: counter
REAL: divisor,xvalue,yvalue
TEXT: message_1,message_2
BEGIN
```



Variables of the type "POINT" and "JC_POINT" do not have to be declared, i.e. the compiler interprets undeclared variables as point variables.



"JC_POINT" variables start with "@".



RCL Keywords (see page 277), may not be used as names for user variables.

7.4 Point Variables

7.4.1 Declaration of Point Variables

Point variables are composed, structured data types that consist of components.

The components are the coordinates or the axes of the point variables.

In addition to the complete value assignment, the point variables can also be assigned component by component, e.g. corner.z_k = height.

The component designation is indicated with the following compiler statements:

- `;;JC_NAMES = axis name, ...`
as well as
- `;;WC_NAMES = coordinate name, ...`

Example

```
;;KINEMATICS: ( 1 = Kin_1)
;;KINEMATICS= Kin_1      ; Default Kinematics here
;;WC_NAMES= kx, ky, kz
;;JC_NAMES= a1, a2, a3

PROGRAM Prog2
  POINT: corner
  JC_POINT: @wpos
BEGIN
  corner.kx = 100.0
  corner.ky = 200.0
  corner.kz = 300.0
  @wpos.a1 = 45.3
  @wpos.a2 = 0.0
  @wpos.a3 = 10.2

  MOVE TO @wpos
  MOVE TO corner
PROGRAM_END
```



Point variable names of type "POINT" start with a letter.

Point variable names of type "JC_POINT" start with a special character "@".

Values are assigned for point variables only after "BEGIN".

7.4.2 Points and Point File "pkt"

While all other variables have to be declared, point variables do not need to be declared explicitly.



The RCL compiler interprets all undeclared variables as POINT or JC_POINT.

All points declared with "DEF" and all points to which no value is assigned anywhere else in the program, are stored in a point file with the identification "pkt". For details regarding multi-kinematic operation, see [Point Tables of the Kinematics, page 221](#).

Points transferred to subroutines as parameters (spc.fct.) do not appear in the point file.

This also applies to the transfer of individual point components (e.g. "Pos.X_K").



When the program is running, values can only be assigned to points in the pkt file if they were declared with "DEF".



Points that are not declared but to which values are assigned in the program, are not stored in the point file, but in the ird file.

7.4.3 Complete Value Assignment

In case of complete value assignments, values are assigned to all coordinates of the point variable. In a particular assignment, all point variables must be of one data type.

Exception:

A particular assignment can contain mixed operations with the standard functions "WC" or "JC".

RCL Variables

Assignment of numerical values

```
position=(50,0,100,0,15,10)
@edge=@(45,5.8,70,10,5.8,0)
```

7.4.4 Assignment of Variables for Individual Components

Assignment for individual components

```
hole=(xvalue,yvalue,zvalue,0,0,0)
@seat=@(50,95.8,height,40,38,0)
hole.z_c=height
```

The variables "xvalue", "yvalue", "zvalue" and "height" are of type "REAL".

Assignment by addition and subtraction

The total of the point variable "position" and the constant has been assigned to the point variable "shelf".

Constants consist of the enumeration of their coordinates placed in parentheses. In contrast to variables, constants cannot be modified while the program is running.

For addition and subtraction, the individual components are either added or subtracted.

Assignment Basis ± Offset

```
shelf= position + (10,-30,100,5,0,0,0)
```

Assignment by multiplication and division

Modify any of the coordinate values of point variables or constants by multiplying or dividing with numerical values or "REAL" type variables.

Assignment via expression (1)

```
pos_1=(10,10,10,0,0,0)
pos_2=pos_1*2
```

Each individual component of pos_1 is doubled by multiplying with 2 and is assigned to the new point variable pos_2, which then has the coordinates: pos_2 = (20, 20, 20, 0, 0, 0)

Assignment via expression (2)

```
div=4.0
pos_3=pos_2/div
```

Subsequently, pos_3 has the coordinates: pos_3 = (5, 5, 5, 0, 0, 0).

Mixed operation with point variables

Using the standard functions "WC" and "JC", mixed operations with "POINT" and "JC_POINT" type variables can be executed.

If the result of the arithmetic operations is to be assigned to a point variable of the type "POINT", the operation has to be executed using world coordinates.

If the result of the arithmetic operations is to be assigned to a "JC_POINT" type variable, the operation must be carried out using joint coordinates.

Mixed operations

```
PROGRAM Prog2
  DEF POINT: p2
  DEF JC_POINT: @p1, @p3
  BEGIN
    @p3=@(0,100,0)
```

RCL Variables

```
@p1= JC(p2)+@p3
PROGRAM_END
```

The control converts the world coordinates of the point "p2" into joint coordinates and adds them to the coordinates of @p3.

The result is assigned to the point variable "@p1".

Reading the actual position POS, @POS

The standard point variables POS (world coordinates) and @POS (joint coordinates) can be used to assign the current position of the robot to a point variable while the program is running.

Reading the actual position

```
act_pos= POS
@joint_pos= @POS
```

The standard point variables are located on the right side of the assignment. A component by component assignment is also possible.

Reading the actual position, component-by-component

```
cpos.c_3= POS.c_3
xvalue= POS.c_1
```



However, a query can only be made when the robot is in standstill.

If the query is made following a "MOVE..TO" command, a velocity-related position error is generally reported during the entire loop.

Assignment by component

For point variables, assign new values to individual coordinates.

Conversely, it is also possible to assign the coordinate values of point values to "REAL" type variables.

The coordinate names are defined in the machine parameters or by a compiler statement. In the following examples, the coordinates of a position in world coordinates are designated by c1, c2, c3, etc. (c = coordinates) and the coordinates of a position in joint coordinates are designated with a1, a2, a3, etc. (a = axis).

The coordinate name is added as a suffix to the name of the point variable using a dot.

Example

```
position.k3= 100      ;the value 100 is assigned to the coordinate k3
zvalue= anchor.k3    ;the third coordinate value of the point variable anchor
                    ;is assigned to the variable zvalue
@pal_pos.a4= radius  ;the coordinate a4 of the position described in joint coordinates
                    ;is assigned the value of the variable radius of type REAL
```

7.5 Working with Text Variables

Texts can be assigned to text variables in a program.

Declare 'message' and 'note' as TEXT variable

```
TEXT: message, note
```

RCL Variables

Text assignment The text to be assigned has to be placed between apostrophes and should not exceed 80 characters. The text must be in one line.

Character string is assigned to TEXT variable

```
message= 'gripper is defective'
note= 'change pallet'
```

Use of variables In the program text variables can be output to an output channel, e.g. as a file with the RCL command WRITE and input with the command READ.

READ/WRITE on input/output channel

```
WRITE File1, note
READ File1, entry
```



The variable itself cannot be placed between apostrophes.
For **WRITE 'note'** the control displays the word "note".

7.6 Working with Array Variables (Arrays)

Variables of the same type can be combined to arrays.

Arrays consist of a freely selectable number of array positions designated by ascending numbers. A variable can be assigned to each array position by specifying a number (index). Array variables all have the same name; only the index is different. The index is identical to the number of the assigned array position.

All types of variables can be stored in arrays.

Array declaration The array declaration consists of:

- the declaration command "ARRAY"
- array limits (position numbers)
- the declaration of array variables.

The array limits are indicated by square brackets and are generated from the first position number (first index = lower limit) and the last position number (final index = upper limit).

Declaration of array instances

```
ARRAY[1..9] POINT: placing_pos ;1-dim. ARRAY
ARRAY[0..10] TEXT: message
ARRAY[-10..10] INTEGER: number_pos
ARRAY[1..9]ARRAY[5..10]POINT: picking_pos ;multidim. ARRAY
```

The lower limit must be separated from the upper limit by two dots, e.g. [3..8].

The upper limit of the array cannot be less than the upper limit. The index is of type "INTEGER".

Example

```
ARRAY[1..5] POINT:hole;declaration of an array with five positions for the
;point variable 'hole', the first index should be 1
```

Value assignment Each array variable is addressed using its name and the index placed in square brackets. For the index, a variable or an expression of type "INTEGER" can also be inserted. A value assignment is possible using the "REPEAT" statement or other program loops.

Value assignment for array elements

```

PROGRAM Prog2
ARRAY[-5..0]  INTEGER: number
INTEGER: i,k           ;index i and value k
BEGIN
;example of direct assignment
number[-5]=1           ;the variable number on the array position -5 has the value 1,
number[-4]=2           ;the variable number on the array position -4 has the value 2,
number[-3]=4           ;etc.
number[-2]=8
number[-1]=16
number[0]=32

;example with REPEAT statement
i=-5                   ;first index
k=1                    ;first value
REPEAT 6 TIMES
number[i]=k            ;value assignment
i=i+1                  ;increase index
k=k*2                  ;value change
                        ;user-defined assignments
REPEAT_END

;example program loop
i=-5                   ;first index
k=1                    ;first value
mark:                  ;jump target
number[i]=k            ;value assignment
i=i+1                  ;index increase
k=k*2                  ;value change
                        ;user-defined assignments
IF i<=0 THEN JUMP mark;conditioned assignm. f. program loop
PROGRAM_END

```

Specification of pallet positions

```

;;INCLUDE head          ;inserting compiler statements
PROGRAM palpos          ;determining pallet positions
INTEGER: line,column,c,l,k ;declarations
ARRAY[1..12] POINT: palpos;the pallet contains 4 lines and 3 columns,
                        ;as well as 12 positions
BEGIN
dx=(30,0,0,0,0)        ;program start
dy=(0,20,0,0,0)        ;distance of positions
line=4                  ;specified in increments under dx and dy
column=3
c=0
l=0
k=0
mark_1:
k=k+1                   ;statements, point assignments
palpos[k]=pos+c*dx+l*dy ;the position pos is a teach point, i.e.
                        ;it is defined by approaching and saving,
                        ;it is the first position palpos[1].
                        ;thus, s=0 and z=0 is entered in the initialization part.
                        ;increased number of columns
c=c+1
IF c<column THEN JUMP mark_1;the next position palpos[2] is in
ELSE c=0                 ;the adjacent column.
                        ;The number of columns has to be increased.
                        ;However, the number of columns c cannot
                        ;exceed the entire number of columns.
                        ;if c is smaller than column, the
                        ;control jumps to the mark mark_1,
                        ;increases the index k by 1
                        ;and assigns the value pos+1*dx+0*dy to the
                        ;position palpos[2]. If c becomes greater than
                        ;column, the control assigns the value zero to the control
                        ;and increases the number of lines.
l=l+1
IF l<line THEN JUMP mark_1;the increase of the line number runs analogously
                        ;to the increase of the number of columns. In the
                        ;IF-THEN statement without ELSE statement
                        ;the control continues with the travel statement
                        ;in case of non-compliance with condition l<line

;;INT= LINEAR
V=1000
AFACTOR=9.999
k=0

```

RCL Variables

```

REPEAT 12 TIMES
k=k+1
MOVE TO palpops[k]           ;travel statement
MOVE_REL with V=36 EXACT (0,0,-20,0,0)
WAIT 2
MOVE_REL with V=59 EXACT (0,0,20,0,0)
REPEAT_END
MOVE_REL CIRCULAR((-50,-50,100,0,0),(-100,-100,0,0,0))
HALT
PROGRAM_END

```

7.7 PLC Communication Channels

7.7.1 Communication Channels to PLC, General

With the RCL, it is possible to write to the inputs of the PLC and read from outputs of the PLC:

- **BINARY** The interface is provided with 32 binary inputs and outputs,
The extended interface is provided with $32 + 160 = 193$ binary inputs to and outputs from the PLC.
- **INTEGER** 32 bit The interface is provided with four inputs and outputs of type "INTEGER",
The extended interface is provided with $4 + 46 = 50$ binary inputs to and outputs from the PLC of type "INTEGER".
- **REAL** The interface is provided with four inputs and outputs of type "REAL",
The extended interface is provided with $4 + 46 = 50$ binary inputs to and outputs from PLC of type "REAL".
- **POINT** Points can be transferred from the PLC to the RCL program, but not from the RCL program to the PLC.
Example
INPUT POINT: 701 = point1
OUTPUT POINT... is not available!
The interface is provided with four inputs of type "POINT",
The extended interface is provided with $4 + 46 = 50$ inputs of type POINT.
- **BELT** Belt channels allow for synchronization with any axis, i.e. a real axis, virtual axis, compound axis or encoder axis (standard position measuring system).
Belt channels are inputs of type "INTEGER". No extended interface.



Data exchange with the PLC

1. Inputs on the RCL side are outputs on the PLC side and vice versa.
 2. Interface data can be accessed on the PLC side via the Kin-Data interface and the function.
Data of the extended interface can only be accessed on the PLC side only via function.
 3. The function access on user data does not depend on the block preparation of the RCL program, but is directly executed.
-

RCL Variables

ML_KINEMATICDATA_SM	FUNCTION	INDEX
dwUserBits_i	ML_KinGetUserBits	1..32
	ML_KinGetUserBits	33..192
dwUserBits_i	ML_KinGetUserBit_DWord	1
	ML_KinGetUserBit_DWord	2..6
adiUserDInt_i[1..4]	ML_KinGetUserDInt	1..4
	ML_KinGetUserDInt	5..50
arUserReals_i[1..4]	ML_KinGetUserReal	1..4
	ML_KinGetUserReal	5..50
dwUserBits_q	ML_KinSetUserBits	1..32
	ML_KinSetUserBits	33..192
adiUserDInt_q[1..4]	ML_KinSetUserDInt	1..4
	ML_KinSetUserDInt	5..50
arUserReals_q[1..4]	ML_KinSetUserReal	1..4
	ML_KinSetUserReal	5..50
arUserPoints_q[1..4]	ML_KinSetUserPoint	1..4
	ML_KinSetUserPoint	5..50

Fig. 7-2: Erweiterung des Shared Memory-Bereiches durch Funktionszugriff aus Sicht der SPS

The respective input or output is addressed by specifying a channel number in the declaration of input or output variables.

Synchronous and asynchronous processing

User data can be accessed in the RCL program via INPUT / OUTPUT variables.

Synchronous as well as asynchronous access (channel number + 1000) is possible; synchronous ⇔ effective in block processing, asynchronous ⇔ effective in block preparation.

The user data can be accessed as follows:

- INPUT / OUTPUT BINARY,
- INPUT / OUTPUT REAL,
- INPUT / OUTPUT INTEGER ,
- INPUT POINT.

RCL access examples:

Access on 108. user bit, output, asynchronous:

OUTPUT BINARY: 1108 = bool1

bool1 = TRUE

Access on 41. user point, input, synchronous:

INPUT POINT: 741 = point1

varPoint = point1

The following channel numbers (for asynchronous access + 1000) are provided for each kinematics in MLC 10VRS:

RCL Variables

Channel number ⁴⁾	Type and meaning
1 to 32, to 192	BOOL, binary inputs and outputs
201 to 204, to 250	REAL, inputs and outputs
401 to 404, to 450	INTEGER, inputs and outputs (on the PLC side DINT!)
501 to 504	Belt inputs (no outputs)
701 to 704, to 750	Point inputs (no outputs)

Fig. 7-3: Channel declarations

Belt inputs RCL programmers can access belt data using the definition
BELT: 501...504.

From the **parameter** perspective, the "BELT: 501" data can be found in parameter K-0-0008, Axes configuration list, first belt axis, etc.

The belt coordinates are added to the coordinates of the Tool Center Point (TCP) in this way.

The **PLC** addresses the same shared memory via an instance of the data type "ML_KINEMATICS_DATA_SM" (for details see Rexroth IndraMotion MLC10VRS, ML_Robot).

The current belt position can be taken from "KinData[n].arPositionOriginalCoord_i" (first TCP coordinates then belt coordinates).

7.7.2 Communication Channels to PLC, Declaration and Programming

In channel declaration, the data type "BINARY", "INTEGER" or "REAL" and the variable name of the signal to be transferred are assigned to a channel number. It is necessary to define whether input or output signals are involved.

Individual channel assignments must be separated by a comma. Do not place a comma after the last assignment.



If the data type is not specified as "INPUT" or "OUTPUT", the control automatically assumes "BINARY".

Communication channels, declaration and usage

```
PROGRAM Prog1
INPUT:
  1=gate_switch1,
  2=met_unit,
  5=li_barrier
OUTPUT:
  7=alarm
BEGIN
  IF NOT gate_switch1 AND li_barrier THEN alarm=1
PROGRAM_END
```

When querying binary signals for 1 it is not necessary to specify "=1". The control then automatically queries for "1".

Example: Test on "1" (TRUE)

```
IF gate_switch1 THEN ..
IF NOT gate_switch1 THEN ..
```

⁴⁾ Third digit: extended interface.

Example: Signals of type "INTEGER"

```

PROGRAM Prog2
INPUT:
  1=gate_switch1
INPUT INTEGER:
  401=grip_force,
  403=meas_height
OUTPUT INTEGER:
  401=pressure,
  402=print
BEGIN
  WAIT UNTIL gate_switch= 1
  IF grip_force>=26 THEN...

WAIT UNTIL meas_height >= 212
MOVE LINEAR UNTIL meas_height >= 200 TO pos ; ok
PROGRAM_END

```

Example: Signals of type "REAL"

```

PROGRAM Prog3
INPUT REAL:
  201=torque,
  206=force
OUTPUT REAL:
  203=speed
BEGIN
;...
PROGRAM_END

```

Example: Point inputs

```

PROGRAM Prog4
;handshake for the points pick and place by the PLC
INPUT BINARY: 1= ppavail
INPUT POINT: 701= pick, 702= place
;handshake to PLC, points processed pick and place
OUTPUT BINARY: 1= ppready

BEGIN
Loop:
;Wait till PLC sets pick and place
WAIT UNTIL ppavail= 1
MOVE TO pick
MOVE TO place
;handshake to PLC, points processed pick and place
ppready= 1
JUMP Loop
PROGRAM_END

```

Querying channels/signals

The query and evaluation of channels or their assigned names occurs in conditions.

Example: Querying channels/ signals in the program

```

PROGRAM Prog5
INPUT:          2=met_unit
INPUT INTEGER:  401=grip_force
BEGIN
  IF met_unit THEN ...
  IF grip_force>=26 THEN...
PROGRAM_END

```



- Output signals cannot be queried.
 - Only input signals can be queried.
-

RCL Variables

Example: incorrect query

```
MOVE UNTIL grip_force=100 AND meas_height=200 TO pallet
```

The following is incorrect:

- MOVE UNTIL grip_force=100 AND meas_height=200 TO pallet

The compiler would interpret this statement as follows:

- MOVE UNTIL grip_force= ((100 AND meas_height) =200) TO pallet



The query in combination with a "MOVE UNTIL" statement is only permitted for 1 input signal.

Setting signals

Signals are set in value assignments; see [Arithmetic Expressions, page 81](#).

Example: Setting signals

```
PROGRAM Prog6
OUTPUT INTEGER: 402=print
BEGIN
  print= 75
PROGRAM_END
```



- Input signals cannot be set.
- Only output signals can be set.

Binary signals can be combined with "AND", "OR" or "NOT".

Exceptions include the conditions "WAIT UNTIL" and "MOVE UNTIL". No combinations are permitted here.

8 RCL Program Control

8.1 WAIT Statement

Syntax:

WAIT expression

With the "WAIT" statement, you can program delays and interruptions into the program execution.

Dwell time If the robot is supposed to remain in one position for a certain amount of time, this time can be directly specified.

Dwell time for "WAIT"

- Measuring unit: Second
- Input range: 0.0 to MAX_REAL s

Programming:

the time specification is a numerical value following the "WAIT" statement, e.g. WAIT 8.5.

Example:

the robot transports a vessel to the metering unit to have it filled.

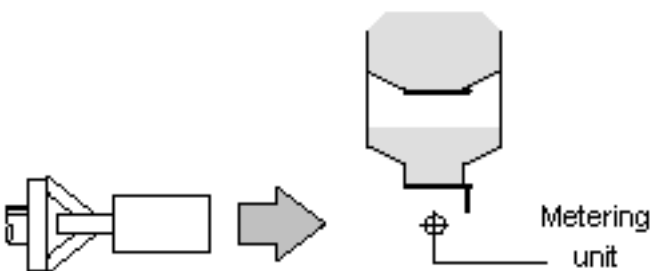
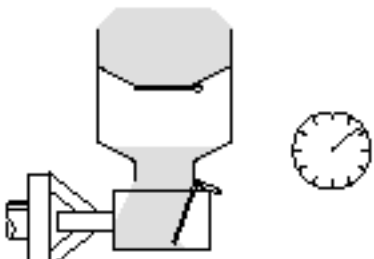
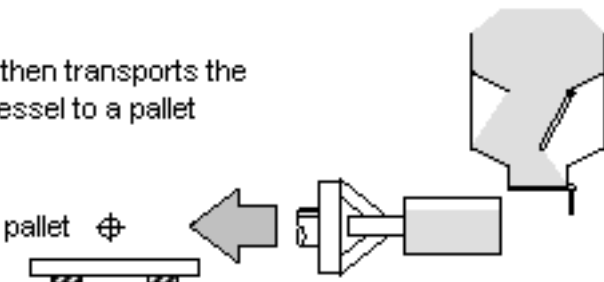
Syntax	Description
MOVE TO met_unit	
WAIT 8.5	The filling time is approx. 8.5 sec. 
MOVE TO pallet	It then transports the vessel to a pallet 

Fig.8-1: Program example 1 for WAIT statement

RCL Program Control

Waiting until condition occurs

Syntax:

```
WAIT UNTIL variable rel_operator expression [max_time=expression [ERROR statement]]
```

rel_operator: =, <>, <=, >=, <, >

If the robot is supposed to wait in one position for a condition to occur, you can specify the condition together with the "WAIT" statement.

Program execution is interrupted until the condition is satisfied.



Conditions can only be set using the input channel variables, see also [Communication Channels to PLC, Declaration and Programming](#), page 66.

Programming:

use the word "UNTIL" to append the condition to the "WAIT" statement, e.g.

Program:

```
WAIT UNTIL pal_empty=1
```

If the condition consists of several input channels, the "WAIT" statement must be divided into several steps.

Examples:

```
WAIT UNTIL signal=1  
WAIT UNTIL light=1  
MOVE TO pallet
```

This would be wrong:

- WAIT UNTIL signal=1 AND light=1

The compiler would interpret this statement as follows:

- WAIT UNTIL signal= ((1 AND light) =1)

Example:

The robot transports a workpiece to a pallet changer.

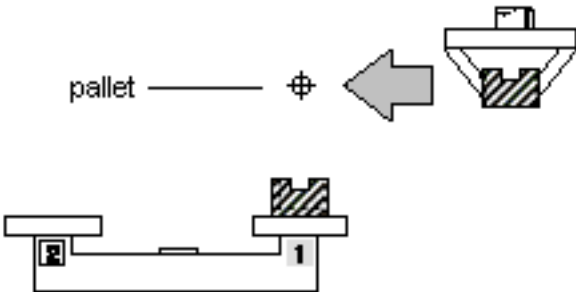
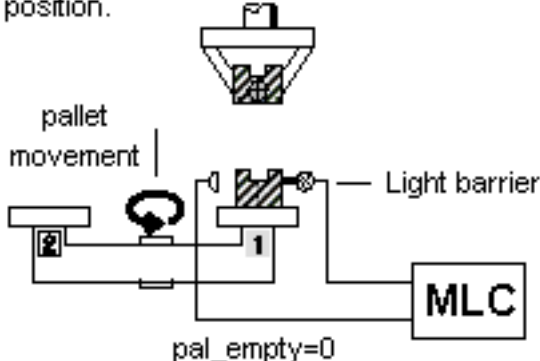
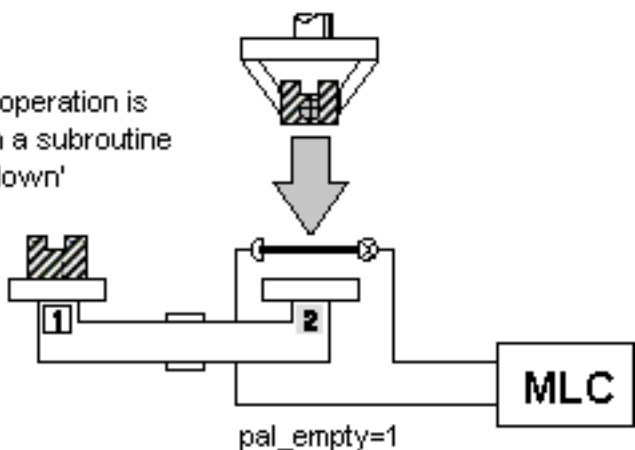
Syntax	Description
MOVE TO pallet	
WAIT UNTIL pal_empty=1	It waits until the pallet is empty. The pallet changer then puts the signal pal_empty=1, when an empty pallet has arrived at the change position. 
put_down	The deposit operation is programmed in a subroutine 'put_down'. 

Fig. 8-2: Program example 2 for WAIT statement

Maximum waiting time *MAX_TIME*

- Measuring unit: Second
- Input range: 0.5 to 32000 s

You can specify a maximum waiting time in conjunction with a WAIT condition.

RCL Program Control

Program execution will be interrupted until the condition is met or the maximum time is exceeded.

You can program an error statement using the maximum waiting time. If the maximum waiting time is exceeded, the control executes the error statement.

Programming:

specify the maximum waiting time with the RCL command MAX_TIME.

This is programmed after the WAIT condition.

Example:

```
WAIT UNTIL sig=1 MAX_TIME=60
```

The error statement is programmed after "MAX_TIME".

Example (error):

```
WAIT UNTIL sig=1 MAX_TIME=60 ERROR PAUSE
```

Sequence example A container is filled with liquid.

A sensor measures the weight of the container.

If the container is full, the sensor outputs the signal "weight= 1". The filling system closes the valve and the robot removes the container. The average filling time is approx. 25 seconds.

The container may only remain under the filling device for a maximum of 45 seconds without putting the production sequence at risk.

Syntax	Description
WAIT UNTIL weight=1 MAX_TIME=45	
ERROR JUMP f_end	
...	
f_end:	
	Error handling

Fig.8-3: Processing errors in the sequence example

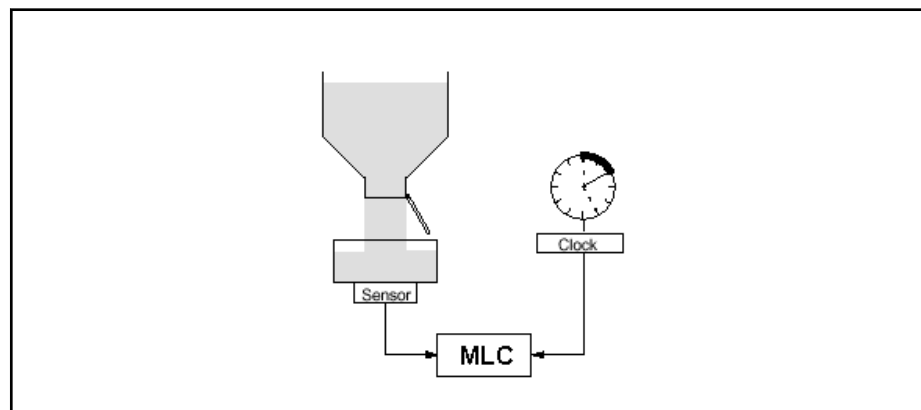


Fig.8-4: Filling procedure has begun

Case 1

- The container is completely filled.
The sensor issues the MLC "weight = 1".
- The robot removes the container.
- "MAX_TIME" was not reached.

RCL Program Control

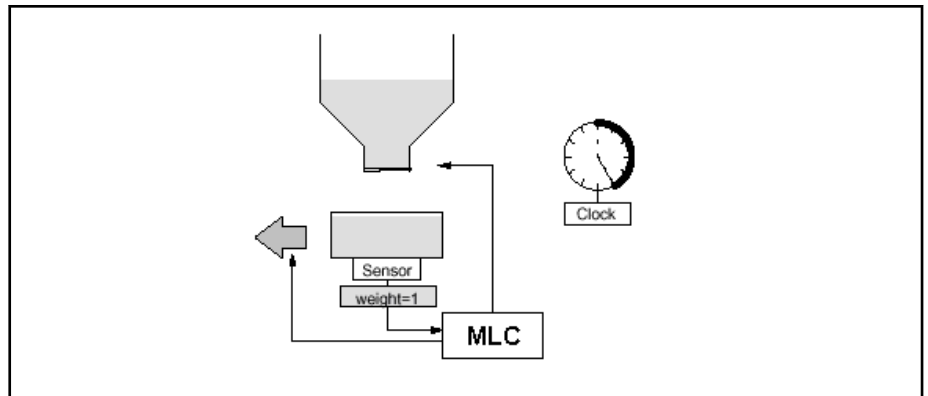


Fig. 8-5: Case 1, sufficient supply

Case 2

- The filling system is empty and the container is not completely filled.
- "MAX_TIME" is reached.
The program continues at "f_end".
- The "f_end" program instructs the robot to isolate the partially filled container.

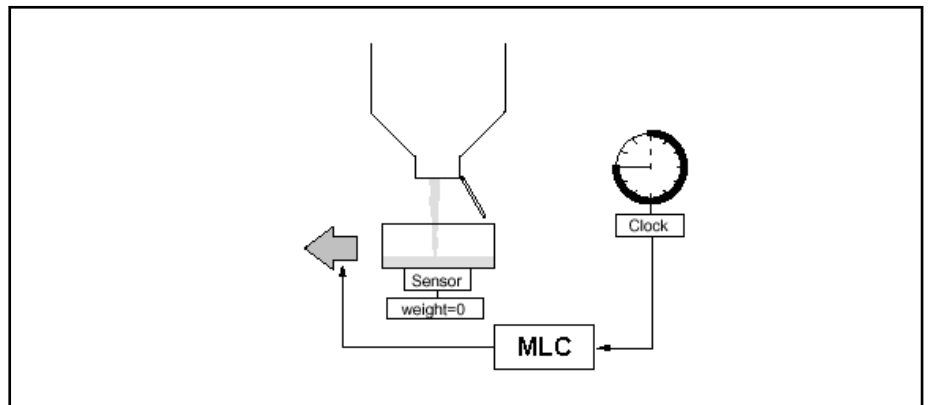


Fig. 8-6: Case 2, empty container

8.2 PAUSE Statement

The "PAUSE" statement can be used to stop the program execution.

To continue the program, the PLC program has to call the MB_IRDProgStart function block again "MB_IRDProgStart".

In this case, bit 26, IRD_PROGRAM_PAUSE is set in K-0-0021, Kinematic status.

KinData[n].dwKinStatus_i.26 is also influenced.

Programming The "PAUSE" statement consists of the RCL command "PAUSE".



Setting an output before the "PAUSE" command is recommended in order to inform the operator on the program flow.

8.3 HALT Statement

The "HALT" statement ends the execution of a statement string in the main program.

RCL Program Control

Based on the "HALT" command, the control recognizes that the program has been terminated during the program run.



In case of called external subroutines, "HALT" results in a return to the calling, active main program.

Programming Enter the "HALT" command before the subroutine declaration or program end. The HALT command can be used several times within a program.



If the program halt can be recognized from the program structure, e.g. from the RCL statement "PROGRAM_END", the control automatically generates the "HALT" command during compilation.

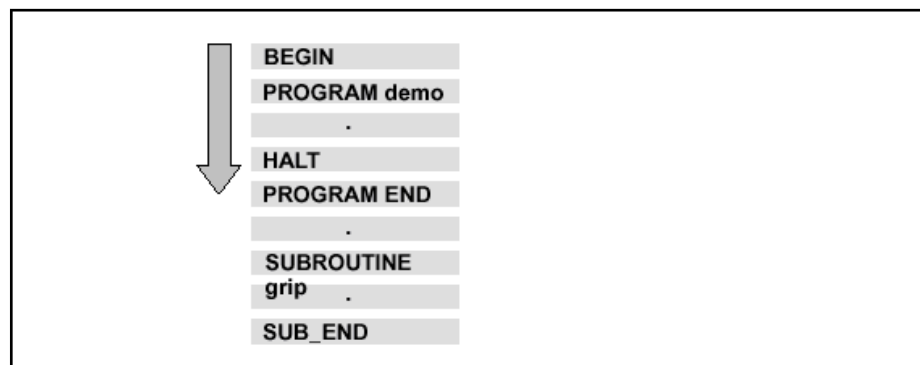


Fig.8-7: HALT statement in the program run

8.4 REPEAT Statement

Syntax:

```

REPEAT [expression] TIMES
statement(s)
REPEAT_END
  
```

With a "REPEAT" statement, a program section can be executed several times.

This is called program section repetition.

The program section can be identified by the repeat statement "REPEAT n TIMES".

To indicate the amount of repetitions, numbers, variables or "INTEGER" type expressions (see [Overview, page 53](#)) can be used.

If number= 0 or negative, the loop is not executed.

The program section is ended with the RCL command "REPEAT_END".

Programming A certain program section is to be executed eight times, i.e. the RCL statement is "REPEAT 8 TIMES".

The program section concludes with "REPEAT_END".

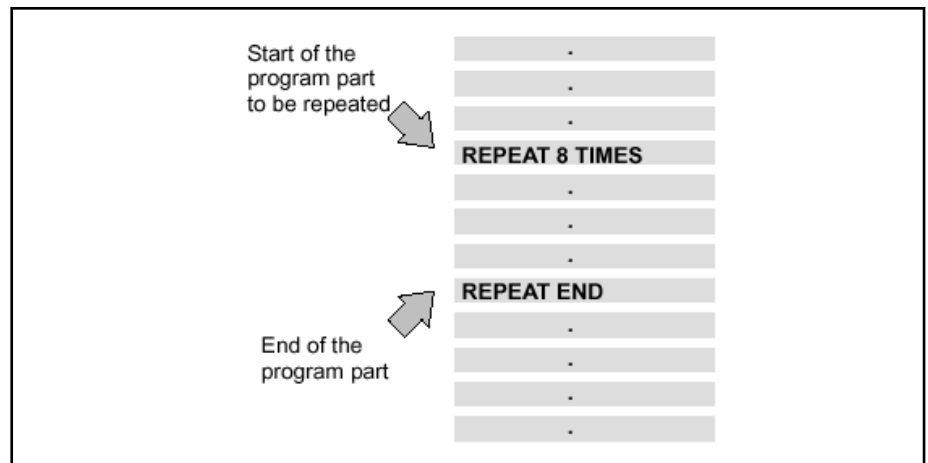


Fig. 8-8: REPEAT statement in the program run

8.5 JUMP Statement

Labels can be set in main programs and subroutines that allow jumps to and from this position using a jump statement. Forward and backward jumps are possible.

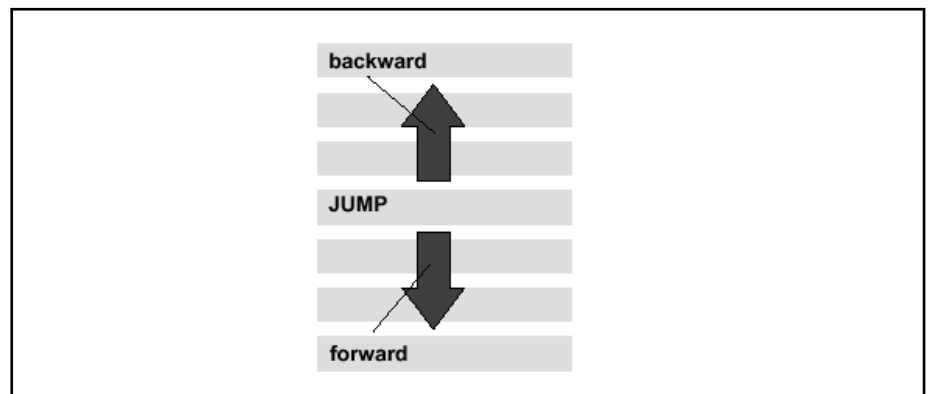


Fig. 8-9: Forward and backward jumps

The jump statement consists of the RCL command "JUMP" and the name of the respective jump label that has been set.

The labels (jump targets) are identified with names. The name contains of 12 characters max.



The first character has to be a letter. The underscore "_" is allowed as a special character. There is no case sensitivity.

To distinguish it from subroutine calls, the jump label is identified with a colon ":" to distinguish a jump from a subroutine call.



A specific jump label may only be used once! There can be any number of jump labels with different names.



Jumps from the main program to a subroutine and vice versa are not permitted!

RCL Program Control

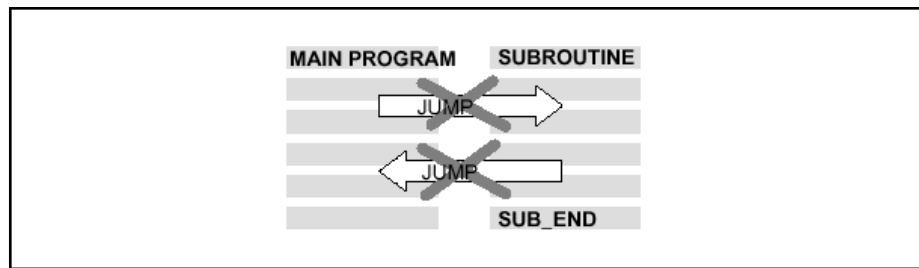


Fig.8-10: Jumps between the main program and a subroutine that are not permitted



Jumps into program section repetitions are also not permitted.
However, jumps out of program section repetitions are permitted.

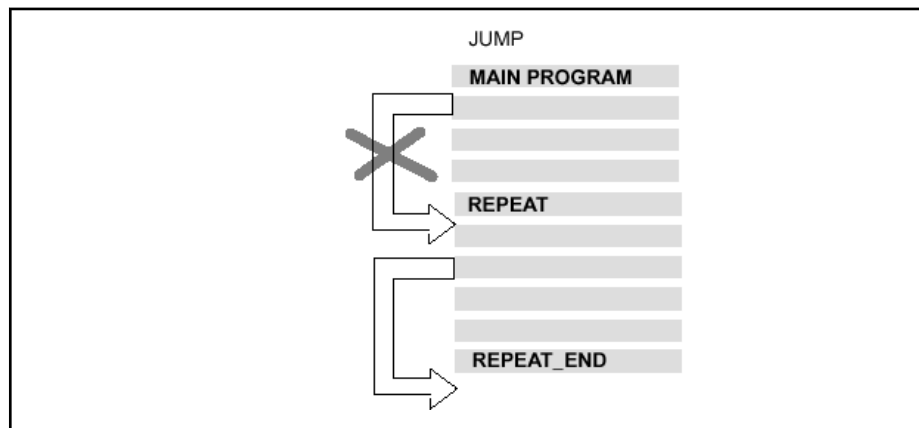


Fig.8-11: A jump into a loop is not allowed. A jump out of a loop is allowed

Programming Within a program, it is to be jumped forwards to the jump label "table".

Syntax:

```
JUMP table ;Jump statement to jump label 'table'
```

```
table: ;Set jump label called table
```

The program is executed up to the jump statement "JUMP table".
This is followed by a jump to the jump label "table".
Subsequently, the control continues the program from this label.

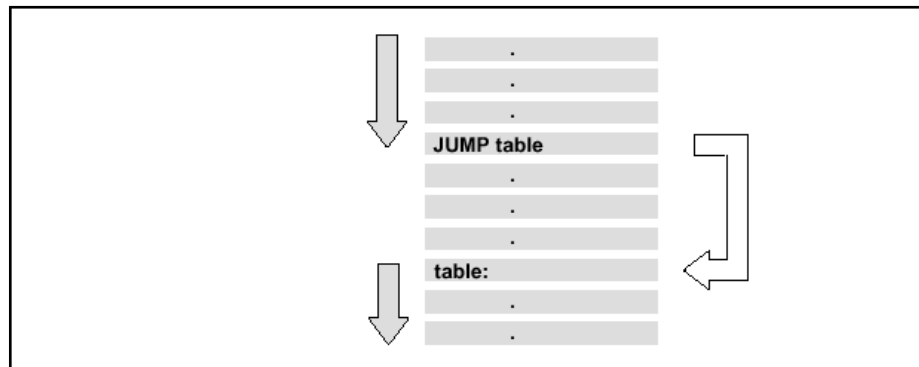


Fig.8-12: JUMP statement in the program run

8.6 IF-THEN-(ELSE) Statement

Syntax:

```
IF condition THEN statement
[ELSE statement]
```

```
IF condition THEN
  BEGIN
    statement(s)
  END
[ELSE statement]
```

```
IF condition THEN
  BEGIN
    statement(s)
  END
ELSE
  BEGIN
    statement(s)
  END
```

At freely selectable locations within a main program or subroutine, the remaining execution of the program can be made conditional (based on conditional statements).

A condition is a "BINARY" type expression.

The condition is considered to be satisfied if the statement is true, i.e. the variable actually has the specified value or a value within the specified range.

The condition is not met if the statement is false.

channel	=	1
l	=	15
sig1	AND	sig2
word	=	j
force	>	115.0
torque	>=	thresholdval



Conditions with AND, OR or NOT do not have to be placed in parentheses, see [Logic Expressions, page 83](#) in order to achieve the desired priorities for condition operation.

Programming

The conditional statement has the following form:

```
IF condition THEN statement 1
  ELSE statement 2
```

"IF" the condition is met, "THEN" the control executes statement 1; or "ELSE", if the condition is not satisfied, the control executes statement 2.

RCL Program Control

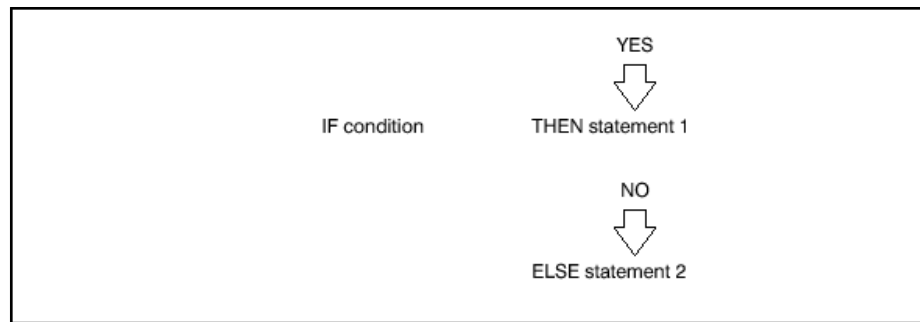


Fig.8-13: IF..THEN..ELSE

For example, possible statements include:

- Program calls (main programs or subroutines)
- Jump statements (JUMP)
- Pause statements (PAUSE), Halt statements (HALT), delays (WAIT)
- Repetitions (REPEAT)
- Motion instructions (MOVE, MOVE UNTIL)
- Conditional statements (IF...THEN...ELSE), etc.

If no jump is programmed in the "THEN" or "ELSE" statements, the control continues the program run with the program steps following the conditional statement.

The "ELSE" statement may be omitted. In this case, the control also continues the program run with the program steps following the conditional statement.

Example

The robot is to search on a shelf for a pallet loaded with a workpiece, approx. weight 200 kg. It has a sensor in its lifting device for this purpose, which informs the control about the weight of the pallet.

When it has found the workpiece, it is supposed to bring it to the machine.

Starting position: the robot is positioned in front of the top pallet.

Program:

```

next:
  in                ;Calling subroutine 'in': Moving in and lifting the pallet
  IF weight >= 180.0
    THEN MOVE TO machine
    ELSE JUMP search
  sig= 1            ;Signal to PLC, pallet delivered
  PAUSE
  search
  down              ;SUBROUTINE put down, move back and down
  JUMP next
  
```

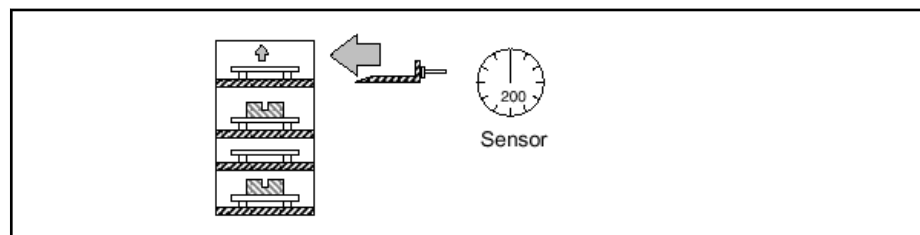


Fig.8-14: Traveling to the pallet rack

RCL Program Control

Program:

```
IF weight >= 180.0      ;Evaluating signal < 180.0 kg, empty pallet
  THEN MOVE TO machine
  ELSE JUMP search      ;Control decides for ELSE statement
                        ;Program jump on jump label 'search:'
```

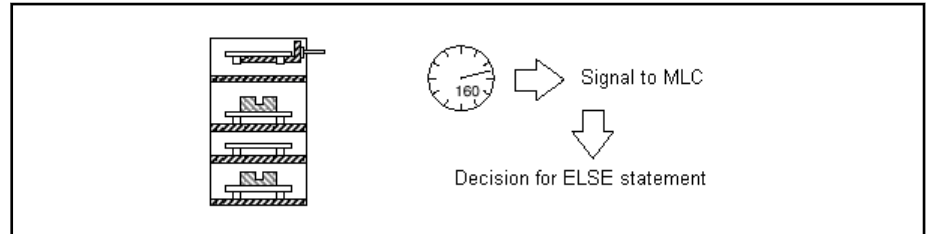


Fig. 8-15: Lifting an empty pallet

Program:

```
search
down      ;Calling subroutine 'down': Put down the pallet,
          ;Move out and move lower.
```

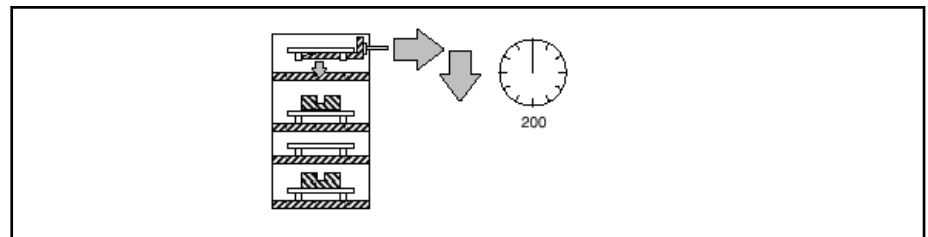


Fig. 8-16: Returning the empty pallet

Program:

```
JUMP next      ;Program jump on jump label 'next:'
next:
in             ;Calling subroutine 'in': Moving in and lifting the pallet
```

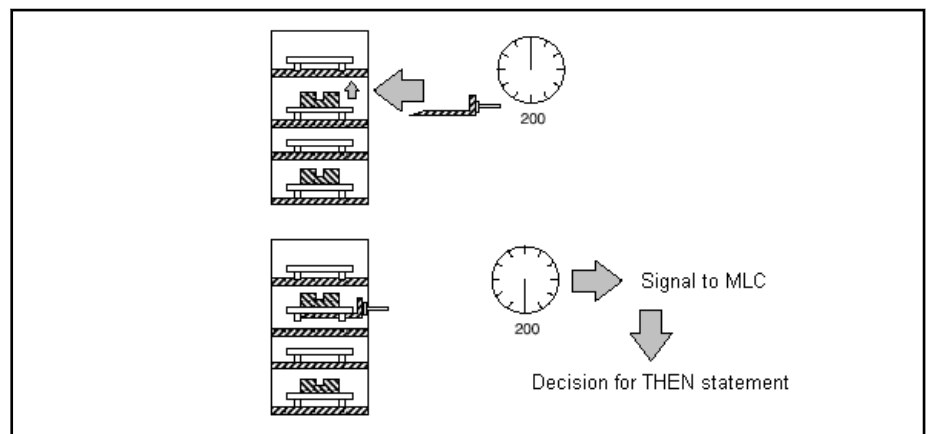


Fig. 8-17: Repetition, full pallet...

Program:

```
IF weight >= 180.0      ;Evaluating signal 200 kg ==> THEN
  THEN MOVE TO machine  ;Traveling towards 'machine'
  ELSE JUMP search
```

Program:

```
sig= 1          ;Signal to PLC, pallet delivered
PAUSE          ;Pause program and wait for PLC
```

RCL Program Control

8.7 CASE Statement

The branching statement "CASE" can be used to generate a selection from several alternatives. In this way, you can avoid nested "IF-THEN" queries and shorten the program's execution time.

Programming

The conditional statement has the following form:

Syntax:

```
CASE selection_expression
  EQUAL: statement
  EQUAL: statement
  DEFAULT: statement
CASE_END
```

Types allowed for **selection_expression** are "BINARY", "INTEGER", "CHAR" and "TEXT".

The constants in the constant list must be compatible with the selection expression. A constant may be used only once within the branching statement.

Specifying the "DEFAULT" section is optional.



If the "DEFAULT" section is not in the branching statement, no runtime error will be output during program execution.

This also applies if none of the statements are valid.

Program:

```
PROGRAM case_ex
INPUT INTEGER: amount      ;Value received from the PLC
INTEGER: offset            ;Declarations
BEGIN                     ;Program start
  offset=0
  CASE amount
    EQUAL 9,10: offset=0
    EQUAL 11,12: offset=1
    EQUAL 13: offset=2
    DEFAULT offset=5
  CASE_END
PROGRAM_END
```

9 Value Assignments and Operations

9.1 Value Assignments

Value assignments are used to assign values to variables for velocity and acceleration while a program is running. This also applies to default variables.

The value assigned has to be of the same data type as the variable.

The following section describes value assignments for general variables. Value assignment for positions can be found in [Point Variables, page 58](#).

Programming Syntax:

variable=expression

The "=" sign is used for the assignment.

The name of the variable to which a value is assigned has to be located on the left side of the assignment. An expression is located on the right side.

It is possible to enter for the expression:

- Numerical values (constants)
- Variable
- Arithmetic expressions
- Standard functions

The "+" sign can be omitted.

The "-" sign is located directly in front of the variables or constants. If two operators directly follow each other, the negative expression has to be placed in parentheses.

If all components of a variable of type "POINT" or "JC_POINT" are to be negated, all components have to be multiplied by -1. -1.0 also has to be placed in parentheses in this case.

Syntax:

counter=1	;Variable and numerical value
xvalue=yvalue+2.5	;Variable and arithmetic expression
distance=SIN(alpha)	;Variable and default function
value=-value	;Variable of type REAL and negation
d=d*(-2.0)	;Variable and negative expression
corner=(-1.0)*corner	;Variable of type POINT and negation

9.2 Operations

9.2.1 Arithmetic Expressions

Arithmetic expressions are operations with:

- Numerical values and/or
- Variables and/or
- Standard functions and/or
- Other arithmetic expressions

The operator defines the operation type.

RCL is familiar with five arithmetic operators:

- | | |
|---|------------------------------------|
| + | Addition, e.g. k = 1 + 5 |
| - | Subtraction, e.g. value = scaled-1 |

Value Assignments and Operations

*	Multiplication, e.g. length = width * 2
/	Division, e.g. height_new = height/2.0
MOD	Modulo calculation, only for INTEGER type, e.g. remaining = number MOD divisor



- The characters + and - can also be used as signs for variables and numerical values.
- Variables and numerical values with signs have to be placed in parentheses so that two operators do not directly follow each other.

No sign should appear on the left side of an assignment.

The addition and subtraction operations can be performed with variables and numeric values (constants) of the types "INTEGER", "REAL", ""POINT"" and "JC_POINT".

The multiplication and division operations can be performed with variables of the types "INTEGER" and "REAL".

"REAL" type variables and numerical values can also be used for multiplication and division of point variables, see [Point Variables, page 58](#).

The operators "*", "/" and "MOD" are executed before + and -.

Calculation takes place from left to right in these classes.

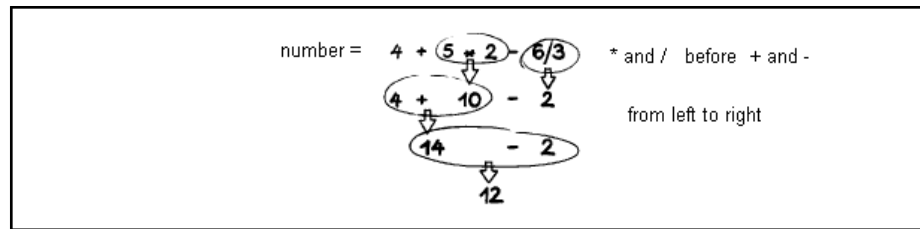


Fig.9-1: Calculation example 1

In addition, expressions that belong together can be placed in parentheses.

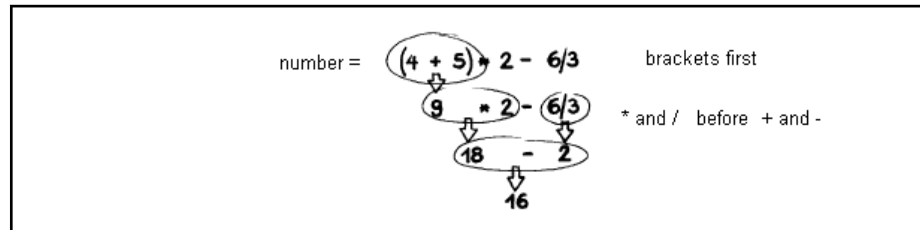


Fig.9-2: Calculation example 2

Modulo function value1 MOD value2

The modulo function, "INTEGER" data type, calculates the integral remainder calculated from the division of "value1" by "value2".

"value1", "value2" and the result are of type "INTEGER".

Determining the column of a pallet after specifying position number and column number.

Value Assignments and Operations

	Col. 0	Col. 1	Col. 2	Col. 3	Col. 4
Line 0	0	1	2	3	4
Line 1	5	6	7	8	9
Line 2	10	11	12	13	14
Line 3	15	16	17	18	19
Line 4	20	21	22	23	24

Fig.9-3: Calculation example 3

The pallet has 5 columns (0, 1, 2, 3, 4). In which column and which row can position 13 be found?

The position number has to be entered in "value1". The number of columns has to be entered in "value2".

Program:

```
column=13 MOD 5 ;Calculation: 13/5 = 2 rest 3. The rest indicates the column. Therefore,
                 ;column 3 is the wanted
line=13 / 5      ;Calculation: 13/5 = 2 rest 3. The quotient indicates the line. Therefore,
                 ;line 2 is the wanted
```

When "column" is called, the Modulo function determines where (in which column) the respective position is located based on the value of the remainder (=3 in this example).

When the line is called, the division function determines where (in which line) the respective position is located based on the value of the result (= 2 in this example).

9.2.2 Comparative Operations

The control queries values and states in conditions, e.g. UNTIL or IF conditions. These queries are made via comparisons.

Programming

The following characters can be used:

=	equal to, e.g. p = 1
<>	not equal to, e.g. p <> 1
>	greater than, e.g. p > 1
>=	greater than or equal to, e.g. p >= 1
<	less than, e.g. p < 1
<=	less than or equal to, p <= 1



"POINT", "JC_POINT" and "RECORDS" type variables (record variables) can only be queried with respect to "=" (equal to) or "<>" (not equal to).

9.2.3 Logic Expressions

The control checks conditions for validity, see also [Comparative Operations, page 83](#), and [IF-THEN-\(ELSE\) Statement, page 77](#). Conditions can only assume two values.

Value Assignments and Operations

"Value 1" for TRUE and "value 0" for FALSE.

This also applies to "BINARY" type variables.

Only one of the two values, 0 or 1, can be applied to such a variable.

Operation of conditions

The remaining program sequence often depends on several conditions at the same time.



"BINARY" type variables and expressions (conditions) can be combined with the logic operations AND, OR and NOT.

cond_1	cond_2	cond_1 AND cond_2
1	1	1
1	0	0
0	1	0
0	0	0

Fig. 9-4: Operation of two conditions cond_1 and cond_2 with AND truth table

cond_1	cond_2	cond_1 OR cond_2
1	1	1
1	0	1
0	1	1
0	0	0

Fig. 9-5: Operation of two conditions cond_1 and cond_2 with OR truth table

Negation of conditions

The validity of type "BINARY" conditions and variables can be negated with the word "NOT".

Example:

Condition cond_1 is true, so (cond_1) = 1

If the word "NOT" is placed before the condition cond_1, the following applies to the validity of "NOT" cond_1:

(NOT cond_1) = 0

or

Example:

The condition cond_2 is false, (cond_2) = 0 applies

Applies to the inversion with "NOT":

(NOT cond_2) = 1

9.2.4 Sequence of Operations

Programming operations with several conditions

Results of comparative operations are always of the "BINARY" data type. When combining several conditions, observe the following operator sequence.

1. NOT
2. *, /, MOD, AND

Value Assignments and Operations

3. +, -, OR
4. =, <>, >, >=, <, <=.

Example:

Query according to numerical values of the variables i and j of type "INTEGER"

IF i = 10 AND j = 50 THEN...

In this example, the control first processes the expression 10 AND j.

However, 10 AND j represents a type conflict for the control, since the constant 10 is of the "INTEGER" type and not of the "BINARY" data type.

Parentheses are used to define the sequence for processing expressions.

IF (i = 10) AND (j = 50) THEN...

10 RCL Functions

10.1 SIN Function

The sine function of data type "REAL" establishes the mathematical relationship between an angle and the lengths of the sides of a right triangle.

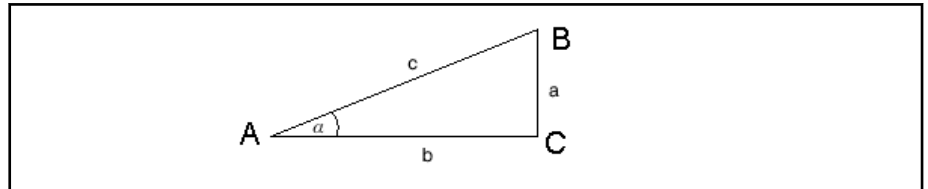


Fig. 10-1: Right-angle triangle

$$\sin \alpha = \frac{\text{opposite leg}}{\text{hypotenuse}} = \frac{a}{c}$$

Programming The angle α must be specified as radian (rad). The number is of type "REAL".

$$\text{rad} = \alpha \frac{\pi}{180^\circ} \Leftrightarrow \alpha = \text{rad} * \frac{180^\circ}{\pi}; \pi = 3,14$$

Syntax:

avalue= c*SIN(rad)



The designation "avalue" was chosen instead of "a" as "A" has already been reserved for acceleration.

10.2 COS Function

The Cosine function of data type "REAL" establishes the mathematical relationship between an angle and the lengths of the sides of a right triangle.

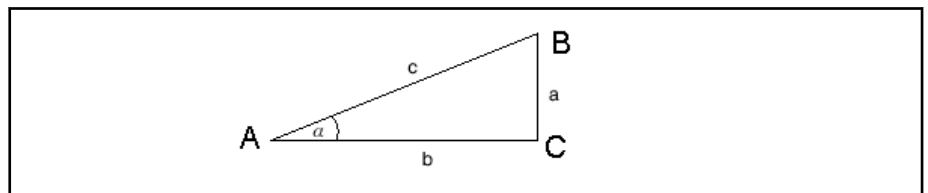


Fig. 10-2: Right-angle triangle

$$\cos \alpha = \frac{\text{adjacent leg}}{\text{hypotenuse}} = \frac{b}{c}$$

Programming The angle α must be specified as radian (rad). The number is of type "REAL".

Syntax:

b= c*COS(rad)

10.3 ATAN Function

The Arcus tangent function, of data type "REAL", determines the angle of a right triangle based on the lengths of the sides.

RCL Functions

The arc tangent function is the inverse of the tangent function, which can be described as follows:

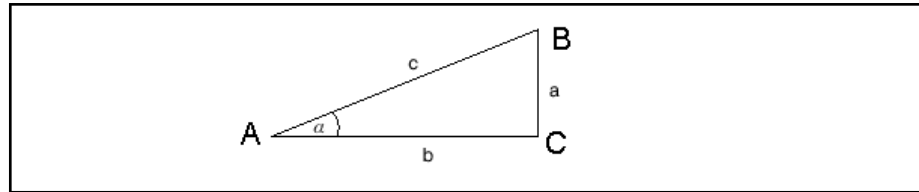


Fig.10-3: Right-angle triangle

$$\tan \alpha = \frac{\text{opposite leg}}{\text{adjacent leg}} = \frac{a}{b}$$

The inverse of the tangent function is

$$\alpha = \text{atan} \frac{a}{b}$$

The angle is established as a radian value "rad" and can be converted.

$$\text{rad} = \alpha \frac{\pi}{180^\circ} \Leftrightarrow \alpha = \text{rad} * \frac{180^\circ}{\pi}; \pi = 3,14$$

Syntax:

alpha= c***ATAN**(avalue/b)



The designation "avalue" was chosen instead of "a" as "A" has already been reserved for acceleration.

10.4 SQRT Root Function

The root function, "REAL" data type, determines the value of the square root of an expression.

Calculation of length c in a right triangle.

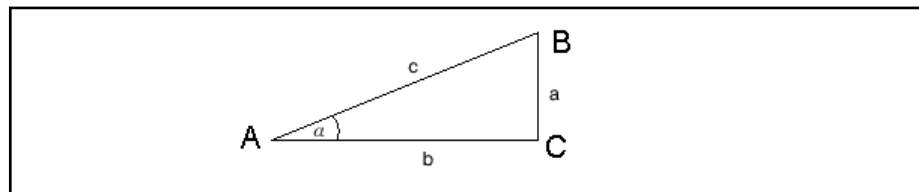


Fig.10-4: Right-angle triangle

Pythagorean theorem:

$$a^2 + b^2 = c^2 \Rightarrow c = \sqrt{a^2 + b^2}$$

Syntax:

c=**SQRT**(avalue*avalue+b*b)



The designation "avalue" was chosen instead of "a" as "A" has already been reserved for acceleration.

10.5 ABS Function

Syntax:

ABS(argument)

The result of the function is the absolute value of the transferred argument. The argument may be of type "INTEGER" or "REAL". The result is of the same type as the argument.

Example:

deviation=**ABS**(delta)

10.6 TRUNC Function

Syntax:

TRUNC(argument)

By cutting the argument, the function transforms the argument of type "REAL" into an "INTEGER" type value.

In the case of a positive argument, the result obtained is the largest integral number less than or equal to the argument, e.g. 3.7 becomes 3.

In the case of a negative argument, the result obtained is the smallest integral number greater than or equal to the argument, e.g. -3.7 becomes -3.

10.7 ORD Function

Syntax:

ORD(char_variable)

This function provides the INTEGER value of type "CHAR".

Example:

char_num=**ORD**(asc_zeich)

char_num=**ORD**('i')

10.8 CHR Function

Syntax:

CHR(integer)

This function provides a "CHAR" type value based on the "integer MOD 256" calculation.

Example:

asc_char=**CHR**(34)

10.9 ROUND Function

Syntax:

ROUND(argument)

By cutting the argument, the function transforms the argument of type "REAL" into an "INTEGER" type value.

It is rounded to the integral number closest to the respective argument.

RCL Functions

Arguments that lie exactly between two neighboring integral numbers (e.g. 0.5, 1.5) are always rounded to the even integral number, i.e. 1.5 is rounded to 2 and 6.5 is rounded to 6.

Example:

```
value_3=ROUND(7.81) ;value_3 is assigned to 8
```

```
value_4=ROUND(-5.43) ;value_4 is assigned to -5
```

10.10 EOF Function

Syntax:

```
END_OF_FILE(argument)
```

This function queries whether the end of the file has been reached when reading a dat file.

The argument is a "FILE" type variable. The function supplies the value 1 (true) if the end of the file has been reached and the value 0 (false) if the end of the file has not been reached.

Example 1:

```
IF END_OF_FILE(dat_values) ;Using the function in a conditional statement
  THEN statements
  ELSE statements
```

Example 2:

```
EOF=END_OF_FILE(dat_values) ;Using the function in an assignment
```

10.11 ASSIGN Function

With the "ASSIGN" statement, the user program can be used to change file names while the program is running.



The ASSIGN statement can only be applied to closed files and standard channels.

Example:

```
FILE datvar,wrvar ;Files datvar, wrvar
ASSIGN wrvar,wrname ;File variable wrvar <-- text variable wrname
ASSIGN datvar,datname ;File variable datvar <-- text variable datname
READ_BEGIN datvar ;Read pointer to the start of the datvar file
READ datvar, oneline ;Reading a line
WRITE_END wrvar ;Write pointer to the end of the wrvar file
WRITE wrvar, oneline ;Writing a line
CLOSE datvar ;Closing the file
CLOSE wrvar ;Closing the file
PROGRAM_END
```

10.12 INT_ASC Function

The standard procedure "INT_ASC" converts an integral value into an array of characters.



The format is right-justified, possibly with leading space characters. If an error is detected during the conversion, the character array remains unchanged and the corresponding error code is returned.

Syntax:

```

OUTPUT INTEGER: 401= toPLC
INTEGER: int_number      ;Specification
INTEGER: index_na        ;int_number = Number to be converted.
INTEGER: length_char     ;Only of type INTEGER, e.g.
INTEGER: error_ret       ;INT number <= 2147483647
ARRAY[1..10] CHAR: ascii_array ;Event feedback ascii_feld = Event array of type ARRAY[ ] CHAR
BEGIN
  int_number=-1234      ;Specification
  index_zf=1            ;Starting index in the character array
  length_char=5         ;Maximum number of characters
                        ;reserved for the number to be converted.
  INT_ASC(int_number,ascii_array,index_na,length_char,error_ret)
                        ;Previous initialization using blanks before conversion is carried out.
  IF error_ret<>0       ;Event feedback

    THEN toPLC= error_ret ;error_ret= Error number output
    ELSE toPLC= int_number

```

Description of the error numbers	0	No errors
	-1	Start index exceeds array limits
	-2	End index (start index + length) exceeds array limits
	-3	Reserved length is too small
	-4	Range exceeded (value too high)
	-5	Array length < 0
	-6	Array length = 0

10.13 ASC_INT Function

The standard procedure "ASC_INT" converts an array of characters into an integral value.

Starting with the start position, the procedure reads characters until a character is detected that is not a number or until the maximum number of characters has been read or the end of the character array has been reached.

Syntax:

```

OUTPUT INTEGER: 401= toPLC
INTEGER: int_number      ;Result: converted number only as INTEGER
INTEGER: index_na        ;Array index from where the number is to be read
INTEGER: length_char     ;Maximum number of characters to be read
INTEGER: error_ret
ARRAY [1..10] CHAR: ascii_array ;Array to be converted; of type CHAR
BEGIN
  ascii_array='AaBc'
  index_na=1
  length_char=5

  ASC_INT(ascii_array,int_number,index_na,length_char,error_ret)

  IF error_ret<>0       ;Event feedback error_ret= Error number output
    THEN toPLC= error_ret ;error_ret= Error number output
    ELSE toPLC= int_number

```

Description of error number	0	No errors
	-1	Start index exceeds array limits
	-2	End index (start index + length) exceeds array limits
	-3	Unused error codes
	-4	Range exceeded (value too high)

RCL Functions

- 5 Array length < 0
- 6 Array length = 0
- 7 Character string does not start with a number or sign

10.14 SIZEOF Function

The standard function "SIZEOF" serves to determine memory space required for any RCL variables or types. Knowing the memory requirement for a given variable (data buffer) is important, especially when dimensioning transfer buffers, e.g. for communication with TCP/IP connections.

The function returns the memory requirement for the specified type or variable in bytes. The standard function "SIZEOF" can be used within the constant definition section.

Syntax of the implicit declaration:

```
sizeof=INTEGER: SIZEOF(type_or_variable) ;Standard function
```

The following applies to the syntax above:

- type_or_variable:
Any type or variable name.

The following table shows the storage area for standard types.

Data type	Size [BYTES]	Corresponding data type (C/C++)	Value range in RCL
BINARY	4	long	[0, 1]
INTEGER	4	long	[-2147483648 ... 2147483647]
REAL	4, simple precision	float	3.4E +/-38 (7 digits)
CHAR	1	unsigned char	[CHR(0)...CHR(255)]
POINT	axis number * 4 (REAL)	typedef float POINT [axisnumber]	-
JC_POINT	axis number * 4 (REAL)	typedef float JC_POINT [axisnumber]	-
WC_FRAME	6 * 4 (REAL)	typedef float WC_FRAME[6]	-
TEXT	80 * 1 (CHAR)	typedef char TEXT[80]	-
INPUT <type>	4 + SIZEOF <type>	typedef struct IN<type> {<type> value; long address} IN<type>	see <type>
OUTPUT <type>	4 + SIZEOF <type>	typedef struct OUT<type> {<type> value; long address} OUT<type>	see <type>
FILE	28	-	-
BNR_FILE	28	-	-

Fig. 10-5: Memory requirement of RCL data types

Examples of the SIZEOF function:

```
;; JC_NAMES=a1,a2,a3,a4 ;Axis names
;; WC_NAMES=c1,c2,c3,c4 ;Coordinates
OUTPUT INTEGER: 401= szofReal, 402= szofPoint ;Transferring to PLC
PROGRAM size
POINT: EndPos
```

```
BEGIN
  szofReal= SIZEOF (REAL)           ;Edition: 4, Type REAL
  szofPoint= SIZEOF (EndPos)        ;Edition: 16 (4 axes * size of REAL)
PROGRAM_END
```

10.15 LIMIT_MIN, LIMIT_MAX, LIMIT_OFF Functions

With the assignments,

Program:

```
LIMIT_MIN= min_position
LIMIT_MAX= max_position
```

the workspace limitations can be defined.

The coordinates of the points **min_position** and **max_position** represent the spatial diagonals of the spanned cuboid.

in a formal sense, MIN_POSITION and MAX_POSITION are regular world coordinate points.

As all WC points, they refer to the local workpiece coordinate system, with the result that LIMIT_MIN and LIMIT_MAX are programmed in workpiece coordinates.

LIMIT_OFF switches the monitoring off.

10.16 JC Standard Function

The standard function "JC" supplies a conversion from world into joint coordinates, e.g.

Program:

```
@start_pos= JC(start_pos)
```

The result is the arm position of the kinematics. Since the starting point is a world coordinate point, the result depends on the active workpiece coordinate system.



"JC" always supplies values from the main area if joint or world coordinates could be ambiguous.

10.17 WC Standard Function

The standard function "WC" supplies a conversion from joint to world coordinates, e.g.

Program:

```
start_pos= WC(@start_pos)
```

This results in coordinates that refer to the local workpiece coordinate system, i.e. the result depends on the active workpiece coordinate system.



"WC" always supplies values from the main area if joint or world coordinates could be ambiguous.

11 RCL Special Functions

11.1 RCL Special Functions, General

Special functions allow the RCL software engineer to access the functions in the control for which no RCL language elements have been reserved. Special functions represent an extension of the RCL language scope.



They can be called in a program if they have been declared before the call similar to variables.

Currently, the following special functions are available for the IndraMotion MLC:

SPC_FCT	Brief functional description
1	iol (position-accurate switching of digital outputs on the path)
2	ppo (position-accurate switching of process data on the path)
21	belt_kind (belt type)
23	clock_date (system date and time)
24	sys_count (system counter, real-time counter)
28	set_belt (set belt counter reset value)
53	belt_area (belt range at belt type 4)
59	Query blending mode
60	Set blending mode

Fig. 11-1: RCL Special Functions

Declaring special functions

The declaration of a special function includes function code and name as well as name and data type of the transfer parameters. The transfer parameters are used to define when, where and how the function is to take effect. The declaration has to appear in the program declaration.

The name of the special function and the names of the transfer parameters can be freely selected by the user.



The specification of the respective special function fixedly determines the data types.

Example:

SPC_FCT: x = example (VALUE INTEGER: wap)

SPC_FCT: x	Function number
Example	Special function name
VALUE INTEGER:	Data type designation for the transfer parameter, e.g. decimal value
wap	Name of the transfer parameter (placeholder)

Fig. 11-2: Example of a special function

Calling the special function

The call in the program statement is executed by entering the special function name and defining the declared transfer parameters.

RCL Special Functions

In the call, name of the special function and transfer parameter type have to be followed as specified in the program declaration.

11.2 RCL Special Functions, Listed as Table

Syntax	Description
SPC_FCT:1=IOL (VALUE INTEGER: IONo VALUE INTEGER: KinNo VALUE INTEGER: CoordNo VALUE REAL: SwitchPos VALUE REAL: ParamValue VALUE INTEGER: RateTime)	Position-accurate switching of digital outputs on the path The declaration of the special function includes code 1 and the special function designation as well as the names and type designations of the function sizes used in the special function call to define which control value should be output when the corresponding axis is at a given position.
SPC_FCT: 2=PPPO (VALUE INTEGER: PPONo VALUE INTEGER: KinNo VALUE INTEGER: CoordNo VALUE REAL: SwitchPos VALUE REAL: ParamValue VALUE INTEGER: RateTime)	Position-accurate switching of process data on the path The declaration of the special function includes the code 2 and the special function designation as well as the names and type designations of the function sizes used in the special function call to define which process value should be output when the corresponding axis is at a given position.
SPC_FCT: 21 = belt_kind (VALUE INTEGER: belt_no VALUE INTEGER: kind_belt)	Belt synchronized workspace of belt type 4 This special function can be used to select the desired type of belt synchronization. At this time, the following synchronizations are possible: kind_belt = 4: Default
SPC_FCT: 23 = clock_date (INTEGER: hours,minutes,day,month,year)	System clock Using the special function 23, the MLC system clock can be accessed from a RoCo program. Day, month, year, hours and minutes are determined.
SPC_FCT: 24= sys_count (INTEGER: c_count)	System counter The variable transferred when special function 24 is called, is filled with the value of the system counter. The variable value is given in milliseconds [ms]. The counter matrix corresponds to the clock time. Every time the control is booted, the real-time counter is reset to zero.
SPC_FCT: 28= belt_set (VALUE INTEGER: belt_no VALUE REAL: reset_value)	Set belt counter Special function 28 assigns a reset value to the belts. The RCL command "SYNC" can set the belt counter to its reset value (default= 0).

RCL Special Functions

Syntax	Description
SPC_FCT: 53 = belt_area (VALUE INTEGER: belt_no JC_POINT: @Base REAL: FullLength REAL: BeginLength)	Geometric position of the belt range with belt type 4 belt_no: Number of the belt for which the value is to be set or read (1..4). If belt_no is negative, the values are read out. @Base: JC_POINT of the beginning of the belt-synchronized workspace. FullLength: Total length of the belt-synchronized workspace in belt direction. The value must be greater than zero BeginLength: Latest start for belt-synchronized operation. The value must be greater than zero and less than or equal to the total length (FullLength).
SPC_FCT: 59 = getBlendMode (VALUE INTEGER : blMode)	Query blending mode blMode=0: The blending radius is limited to half the distance of the total distance (default) blMode=1: The blending radius can reach the entire distance of the shorter distance if possible
SPC_FCT: 60 = setBlendMode (VALUE INTEGER : blMode)	Set blending mode blMode=0: The blending radius is limited to half the distance of the total distance (default) blMode=1: The blending radius can reach the entire distance of the shorter distance if possible

Fig. 11-3: Special functions

11.3 Position-accurate Switching

11.3.1 Position-accurate Switching, General

The MLC allows switching on one path. During a travel motion, signals can be transferred from a RoCo program to the PLC in advance and with accurate position information using the KinData interface.

See also [RobotControl Functionality Programmed in the PLC, page 225](#).

Two special functions are available for this task.

- With special function 1, digital outputs can be enabled at the interface.
In the following, special function 1 is identified as I/O logic (switch on/switch off logic, IOL).
- With special function 2, integers can be output.
In the following, special function 2 is identified as PPO logic (process parameter output logic). Use these values to control your technology, e.g. measuring the amount of paint in painting applications or switching on or off the spray gun.

Position-accurate output Accurate position output means that a specific position of the tool center point (gripper position) is defined in the program that sets the output to the KinData interface at the PLC.

RCL Special Functions

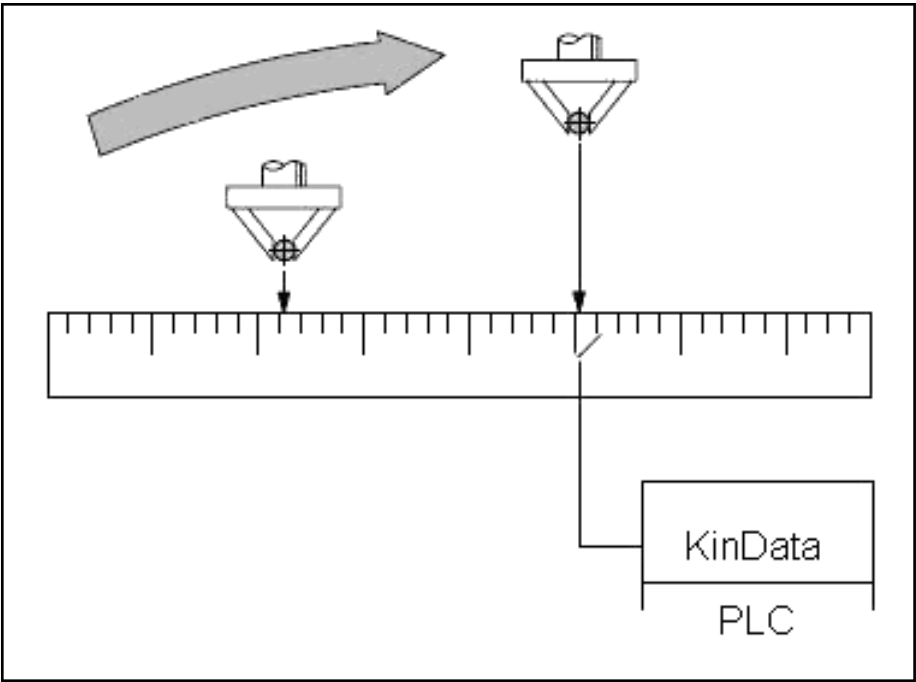


Fig.11-4: Accurate position output at the KinData interface

Programming Before the special functions are called, they have to be declared in the RoCo program declaration similar to variables. It must be ensured that the parameter list corresponds exactly to the following declaration.

11.3.2 SPC_FCT: 1, Position-accurate Switching of Digital Outputs on the Path

The declaration of the special function includes code 1 and the special function designation as well as the names and type designations of the function sizes used in the special function call to define which control value should be output when the corresponding axis is at a given position.

Declaration *Program:*

```
SPC_FCT:1=IOL(VALUE INTEGER: IONo
VALUE INTEGER: KinNo
VALUE INTEGER: CoordNo
VALUE REAL: SwitchPos
VALUE REAL: ParamValue
VALUE INTEGER: RateTime)
```

IONo	The KinData interface has 32 available outputs that can be enabled with special function 1 (OUTPUT BINARY 1..32).
KinNo	Number of the kinematics (1..16) to which the switching position is referred.
CoordNo	Number of the coordinate within the kinematics to which the switching position is referred.
SwitchPos	Position switched to, i.e. where the parameter value is output.

RCL Special Functions

ParamValue	The parameter value is preset as a decimal value where a value < 0.5 is output as logic 0 (low) and a value ≥ 0.5 is output as logic 1 (high).
RateTime	Time, in [ms], which elapses between the actual output of the parameter value and the reaching of the switching position. This means that the digital output is already switched before reaching the desired position (lead time). Thus, peripheral reaction times can be compensated.



No belt-synchronous coordinates may be used with this function!

Enabling signals *For the special function to operate correctly, the following states must be set:*

- Automatic,
- No emergency stop
- All axes are assigned to the kinematics with MB_AxisToGroup,
- In K-0-0241 feed halt cannot be set.

Program:

```
PROGRAM prog1
VAR:
SPC_FCT:1=ioout(VALUE INTEGER: chan,kin,coord
                VALUE REAL: po, va
                VALUE INTEGER: tm)
OUTPUT BINARY: 7= ToVaOn      ; corresponds to KinData.dwUserBits_i.6
BEGIN
  ioout(7,1,3,-300,1.0,50) ; 50ms (tm) before the Cartesian z-axis (coord)
                           ; reached position (po) -300, value (va) 1 is output on
                           ; output channel (chan) 7
MOVE LINEAR TO (10,20,-300)
PROGRAM_END
```

11.3.3 SPC_FCT: 2, Position-accurate Switching of Process Data on the Path

The declaration of the special function includes code 2 and the special function designation as well as the names and type designations of the function sizes used in the special function call to define which control value should be output when the corresponding axis is at a given position.

Declaration *Program:*

```
SPC_FCT:2=PPO(VALUE INTEGER: PPONo
              VALUE INTEGER: KinNo
              VALUE INTEGER: CoordNo
              VALUE REAL: SwitchPos
              VALUE REAL: ParamValue
              VALUE INTEGER: RateTime)
```

PPONo	The KinData interface is provided with 4 integer outputs that can be switched with special function 2 (OUTPUT INTEGER 1..4).
KinNo	Number of the kinematics (1..16) to which the switching position is referred.
CoordNo	Number of the coordinate within the kinematics to which the switching position is referred.
SwitchPos	Position switched to, i.e. where the parameter value is output.

RCL Special Functions

ParamValue	The parameter value is preset as decimal value.
RateTime	Time, in [ms], which elapses between the actual output of the parameter value and the reaching of the switching position. This means that the digital output is already switched before reaching the desired position (lead time). Thus, peripheral reaction times can be compensated.



No belt-synchronous coordinates may be used with this function!

Enabling signals *For the special function to operate correctly, the following states must be set:*

- Automatic,
- No emergency stop
- All axes are assigned to the kinematics with MB_AxisToGroup,
- In K-0-0241 feed halt cannot be set.

Program:

```
PROGRAM prog1
VAR:
SPC_FCT:2=ppo(VALUE INTEGER: chan,kin,coord
              VALUE REAL: po, va
              VALUE INTEGER: tm)
              OUTPUT INTEGER: 403=ToVaOn ; corresp. to KinData.adiUserDInt_i[3]
BEGIN
ioout(403,1,3,-300,42.0,50) ; 50ms (tm) before the Cartesian z-axis (coord)
                           ; reached position (po) -300; value (va) 42 is output on output channel
MOVE LINEAR TO (10,20,-310)
PROGRAM_END
```

11.4 SPC_FCT: 21, Belt Type

With special function 21, the type of belt synchronization is selected. After the control is booted, synchronization type 1 is active.



Currently, only synchronization type 4 is approved for use.

If the synchronization type is changed, this setting is valid (for all processes) until it is modified again or the control is rebooted.

Declaration *Program:*

```
SPC_FCT:21 = belt_kind (VALUE INTEGER: belt_no
                       VALUE INTEGER: kind_belt)
```

belt_no	Belt number (1 to 4)
kind_belt	Belt synchronous type: 4= belt type 4 with an axis parallel to the belt and with the possibility for a belt-synchronized PTP travel

Example *Program:*

```
PROGRAM SPZ21 ;Program name
              ;declaration of the special function
SPC_FCT : 21 = belt_kind (VALUE INTEGER: belt_no
                        VALUE INTEGER: kind_belt)
```

```
BEGIN belt_kind(1,4)      ;selecting the belt type 4 for belt 1
PROGRAM_END
```

11.5 SPC_FCT: 23, DateTime

With special function 23, the MLC system clock can be accessed from an RCL program. Day, month, year, hours and minutes are determined.

Declaration *Program:*

```
SPC_FCT:23 = DateTime(INTEGER: hour, minute,
                     day, month, year)
```

hour	hour
minute	minutes
day	day
month	month
year	year

Example *Program:*

```
PROGRAM SPZ23                      ;program name
VAR:
INTEGER: hr, mi, dy, mo, ye      ;declaration of the special function
SPC_FCT : 23 = DateTime(
INTEGER: hour,
                minute,
                day,
                month,
                year)
BEGIN
    DateTime(hr, mi, dy, mo, ye)   ;returns the current data and time in
                                ;hr, mi, dy mo, and ye
PROGRAM_END
```

11.6 SPC_FCT: 24, SysClk

The variable transferred when special function 24 is called is specified by the system counter value.

The variable value is given in milliseconds [ms]. The counter matrix corresponds to the MC cycle time (see C-0-0410). Every time the control is booted, the real-time counter is reset to zero.

Declaration *Program:*

```
SPC_FCT:24 = SysClk(INTEGER: clks)
```

clks	Number of motion cycles (MC) passed after the reboot
------	--

Example *Program:*

```
PROGRAM SPZ24                      ;program name
VAR:
INTEGER: clks                      ; declaration of special functions
SPC_FCT : 24 = SysClk(INTEGER: clks)
BEGIN S
    SysClk( clks )                 ;supplies number of motion cycles in clks
PROGRAM_END
```

RCL Special Functions

11.7 SPC_FCT: 28, Set Belt Counter

Special function 28 assigns a reset value to the belts.

The belt counter can be set using the RCL command "SYNC".

Declaration *Program:*

```
SPC_FCT:28=belt_set(VALUE INTEGER: belt_no  
                   VALUE REAL: reset_value)
```

belt_no	Belt number (1 to 4)
reset_value	Reset value in [mm] for the belt counter.

Enabling signals *For the special function to operate correctly, the following states must be set:*

- Automatic,
- No emergency stop
- All axes are assigned to the kinematics with MB_AddAxisToGroup
- In K-0-0241 feed halt cannot be set.

Example *Program:*

```
;;INCLUDE head           ;head contains compiler statements  
PROGRAM SPC28            ;program name  
SPC_FCT:28=belt_set(VALUE INTEGER: belt_no ;declar. of special function  
                   VALUE REAL: reset_value)  
  
BEGIN  
belt_set(2,200)  
PROGRAM_END
```

The reset value remains active until special function 28 is called again for the same belt. If the control is rebooted, the reset value is set to 0.

For the belt synchronization in an RCL program, the internal belt counters are also set to the reset value active at the time the function is called. An example is

Program:

```
SYNC band1, E1=1 ; belt 1 is set to the reset value
```

Program:

```
SYNC belt3 ≥ 200 ; 200 is once subtracted from the current belt counter value  
                ; of belt3.  
                ; if the current belt counter is < 200, the function waits  
                ; until the counter is increased to 200 and subsequently subtracts 200.
```

11.8 SPC_FCT: 53, Belt Synchronous Operating Range of Belt Type 4

Before using SPC_FCT: 53, belt-synchronized workspace, the kinematics must be defined with SPC_FCT: 21, belt type 4.

The working principle is shown in the following illustration.

RCL Special Functions

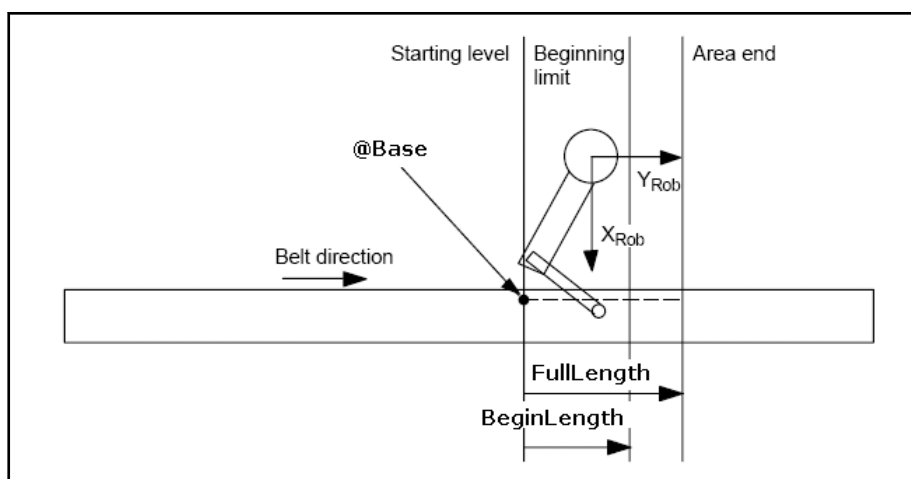


Fig. 11-5: Belt-synchronized workspace

The belt direction is defined by parameter K-0-0071, Coupling factors belt 1 (K-0-0072/belt 2... K-0-0074/belt 4).

In the figure the belt is rotated by +90 degrees to the robot coordinate system, i.e., Coord1=0, Coord2=1, Coord3=0. (A coordinate's value is the cosine (cos a) of the angle generated by the belt and the corresponding coordinates).

The starting level runs through the machine coordinate point @Base and is perpendicular to the belt direction.

The start and end of the area are parallel to the starting level.

A check is made during the first travel motion in the belt-synchronized section of belt type 4, to see whether the end point of the travel motion is located between the starting level and the beginning limit. If it has not yet reached the starting level, the travel motion is delayed until the end point has passed the starting level. If the end point of the first belt synchronized traveling motion has already exceeded the start limit, an error message is displayed (F12C00B4, belt sync.: 1. BS travel block activated too late).

When executing the belt-synchronized section of belt type 4, the tool center point (TCP) to the area end is monitored. If it is determined that the TCP will pass the end of the area or has already passed it, an error message is output (F12C00AE, Belt-sync.: travel block activated too late).

During the final belt-synchronized travel block (desynchronization and travel to fixed position), the area end is not monitored.

Declaration Program:

```
SPC_FCT:53 = belt_area (VALUE INTEGER : belt_no
                        JC_POINT: @Base
                        REAL: FullLength
                        REAL: BeginLength)
```

belt_no	Number of the belt for which the belt value is to be set (1 to 4) or to be read (-1 to -4).
@Base	JC_POINT of the start of the belt-synchronized workspace
FullLength	Full length of the belt synchronized workspace in the belt direction. The value must be greater than zero.
BeginLength	Latest start for belt-synchronized operation. The value must be greater than zero and less than or equal to the full length.

RCL Special Functions

After the control is booted, the initial values for the belt range are:

- @Base all components 0.0
- FullLength= 0.0
- BeginLength= 0.0

Belt type 4 cannot be used with these values.

They must be set to reasonable values at least once with SPC_FCT: 53 = belt_area (...) to reasonable values. They are globally valid for all processes until they are modified again.

If a kinematics has more than one belt, each belt has its own belt range.

The currently valid values for the belt ranges can be read out by calling SPC_FCT: 53 = belt_area (...) with negative belt numbers. The values are then written to the transfer variables.

Requirements for input data:

- FullLength must be greater than zero.
- BeginLength must be greater than zero and less than or equal to FullLength.
- All components of the machine coordinate point @Base must be defined.
- If there is a process in the belt-synchronized section with belt type 4 for the kinematics, belt_area should not be modified.
- The machine coordinate point @Base must be located within the work-space.
- Five equidistant machine coordinate points from the machine coordinate point @Base in the belt direction up to the end of the area are specified internally.

These five machine coordinate points must also be located within the work-space of the kinematics. The geometric dependence of belt motions on axis motions is determined based on this total of six machine coordinate points.

Example Program:

```

PROGRAM beltsyn
VAR:
  CONST :
    C_BELT_NO_1 = 1,           ; belt number
    C_BK_PTP    = 4           ; belt type PTP
  BELT :
    500 + C_BELT_NO_1 = belt1

                                ; declaration of special functions
SPC_FCT : 21 = belt_kind (VALUE INTEGER: belt_no
                                VALUE INTEGER: kind_belt)
SPC_FCT : 53 = belt_area (VALUE INTEGER: belt_no
                                JC_POINT: @Base
                                REAL: rTotalLength
                                REAL: rBeginLength)

BEGIN

                                ; belt type specification
                                ; specifying belt-synchronous operating range
belt_kind ( C_BELT_NO_1, C_BK_PTP )
rTotalLength = 560
rBeginLength = 400
base = (280, 140, -600, 0, 0, 0)
@Base = JC(Base)
belt_area ( C_BELT_NO_1, @Base, rTotalLength, rBeginLength)

```

Several belts within one kinematics

If a kinematics has several belts, the special function can be used with the same declaration for all of the kinematic belts. "belt_no" is used for distinguishing purposes.

Several kinematics



An RCL program can move exactly one kinematic. The RoCo is based on the following philosophy:

1. the RCL program is responsible for the robot's motion
2. the PLC program takes care of all other tasks (communication, coordination, system/process control...)

"JC_POINT" type variables are related to a kinematics. If this special function is to be used for several kinematics, it must be declared with different names, e.g. "belt_area1".

The number (53 = ...) has to be the same for all kinematics with kinematic-related "JC_POINT" type variables, e.g. Robi_1.JC_POINT.

Example:

Program:

```
SPC_FCT :53=belt_area1 (VALUE INTEGER: belt_no
                        Robi_1.JC_POINT: @Base
                        REAL: FullLength
                        REAL: BeginLength)

SPC_FCT :53=belt_area2 (VALUE INTEGER: belt_no
                        Robi_2.JC_POINT: @Base
                        REAL: FullLength
                        REAL: BeginLength)
```

11.9 RCL Programming "Blending Mode"

11.9.1 SPC_FCT: 60, Set Blending mode

The blending mode can be selected in the RCL program using SPC_FCT 60. This mode applies to all subsequent commands in the block buffer.

Program:

```
SPC_FCT: 60 = setBlendMode(VALUE INTEGER : blMode)
```

blMode=0: The blending radius is limited to half the distance of the total distance (default)

blMode=1: The blending radius can reach the entire distance of the shorter distance if this is possible

The programmed blending radius R is taken into consideration in both modes.

In case of **blMode = 0**

the effective blending radius R_w results from the minimum of the programmed radius R, half the distance s_1 and half the distance s_2 .

$$R_w = \min \left(R, \frac{1}{2}s_1, \frac{1}{2}s_2 \right)$$

Fig. 11-6: Effective radius when blending with blending mode 0

In case of **blMode = 1**

the effective blending radius R_w results from the minimum of the programmed radius R, the total distance s_1 and the total distance s_2 , if this is possible.

RCL Special Functions

$$R_w = (R, n_1 \cdot s_1, n_2 \cdot s_2)$$

Fig. 11-7: Effective radius when blending with blending mode 1

$n_{(1,2)} = 1$ or $1/2$

s_1 = distance 1

s_2 = Path2

$n_{(1,2)}$ is automatically selected by the control by means of the programmed travel commands.

If the travel command is a "MOVE TO" command, $n = 1$ for this distance.

If the travel command is a "MOVE VIA" command, n depends on the specified command.

11.9.2 SPC_FCT: 59, Query blending mode

The blending mode set in the block preparation can be queried in the RCL program using SPC_FCT 59.

Program:

```
SPC_FCT: 59 = getBlendMode(VALUE INTEGER : blMode)
```

blMode=0: The blending radius is currently limited to half the distance of the total distance (default)

blMode=1: The blending radius can reach the entire distance of the shorter distance if this is geometrically possible.

Example:

```
;;COMPILER = RCL3
;;INT = LINEAR
;;KINEMATICS: ( 1= Kin1)
;;Kin1.JC_NAMES = vAchse1,vAchse2,vAchse3,vAchse4,vAchse5,vAchse6
;;Kin1.WC_NAMES = kx,ky,kz,kc,kb1,kb2
```

```
PROGRAM halfFull
```

```
VAR:
SPC_FCT : 60 = setBlendMode (VALUE INTEGER : blMode)
INTEGER : blMode
POINT : P1, P2, P3, P4
BEGIN
```

```
V=200 ; 0.2 m/s
A=1000 ; 1 m/s^2
R=1000 ; 1 meter = control calculates the max. blend radius
```

```
; because the moved distances will be lower then 1 meter
```

```
BLOCK_SLOPE ; has no effect because this is the default
setBlendMode( 0 ) ; has no effect because this is the default
```

```
; points for the rectangle
P1 = (-300, 0, -650, 0, 0, 0)
P2 = (-300, 200, -650, 0, 0, 0)
P3 = ( 300, 200, -650, 0, 0, 0)
P4 = ( 300, 0, -650, 0, 0, 0)
```

```
; no blending
MOVE LINEAR TO P1
MOVE LINEAR TO P2
MOVE LINEAR TO P3
MOVE LINEAR TO P4
```

RCL Special Functions

```
; blending with half distances

MOVE LINEAR TO P1
MOVE LINEAR VIA P2
MOVE LINEAR TO P3
MOVE LINEAR TO P4

; blending with half distances

MOVE LINEAR TO P1
MOVE LINEAR VIA P2
MOVE LINEAR VIA P3
MOVE LINEAR TO P4

; blMode = 1
setBlendMode( 1 )

; no blending

MOVE LINEAR TO P1
MOVE LINEAR TO P2
MOVE LINEAR TO P3
MOVE LINEAR TO P4

; blending with complete distances of P1P2 and P2P3

MOVE LINEAR TO P1
MOVE LINEAR VIA P2
MOVE LINEAR TO P3
MOVE LINEAR TO P4

; blending with complete distance of P1P2
; and half distance of P2P3
; blending with half distance of P2P3
; and complete distance of P3P4

MOVE LINEAR TO P1
MOVE LINEAR VIA P2
MOVE LINEAR VIA P3
MOVE LINEAR TO P4

PROGRAM_END
```

12 RCL Motion Statements

12.1 General Information

Robot motions are initiated by motion statements. Motion statements describe the motion of a robot from a current position and orientation to a target point.

It is distinguished between:

- Direct motion statements (specification of **V**, **A** etc.)
- Motion-influencing motion statements

12.2 Direct Motion Statements

12.2.1 Overview

A direct motion statement consists of the following single statements:

- Motion command
- Interpolation method
- Additional information
- Abort condition (opt.)
- Approach method
- Target position

The following table provides an overview on possible entries. The possible combinations result from the following individual descriptions.

Motion statements				
Motion command	Interpolation method	Additional information	Abort condition	Approach method
MOVE	LINEAR	V	UNTIL	TO
MOVE_REL	CIRCULAR	A		VIA
	PTP	J		APPROX
		R		EXACT
		VFACTOR		
		AFACTOR		
		DFACTOR		
		V_PTP		
		R_PTP		

Fig. 12-1: Overview on motion statements

Programming an abort condition is optional.

Approaching method and target position have to be specified for each motion command.

Statement-specific information

The WITH command defines the additional information **V**, **A**, **J** and **R** only statement-specifically. The definition of the interpolation method is executed directly after the motion command without the WITH command.

RCL Motion Statements

These specifications do not apply anymore for the following motion commands.

It is not possible to program the **AFACTOR**, **VFACTOR** and **DFACTOR** statement-specifically.

Global information If no global statement-related additional information is commanded with the motion command, the global commands are valid. They include all additional information (V, A, J, R etc.) and the interpolation method (LINEAR, CIRCULAR and PTP).

At the beginning, the respective default values are valid. As soon as these are defined, the following motion commands apply.



If no additional information is specified for the motion command, the global additional commands apply.

12.2.2 Motion Commands

The control is familiar with the following motion commands:

- MOVE
- MOVE_REL

MOVE

Syntax:

```
MOVE[Interpolation method][additional information][abort condition]
[TO]target point[VIA target point[TO target point]
```

The control interprets all positions programmed after MOVE as absolute positions. The coordinate values refer to the zero point of the world coordinate system or joint coordinate system.



The motion command MOVE can be combined with all interpolation methods.

Program:

```
target=(10,10)
MOVE TO target
```

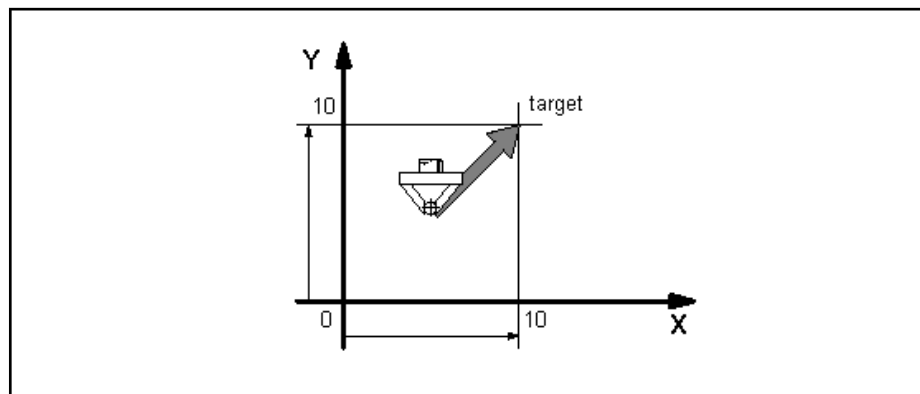


Fig.12-2: MOVE command

Additional information in the travel command are the decisive factor whether the robot travels exactly to the programmed points – i.e., within the defined tolerance - or whether it travels past the points without stopping.

RCL Motion Statements

When traveling in absolute position with MOVE, the TO and VIA specifications apply.

MOVE VIA ...(blending) The robot travels past the positions without stopping. See "R" and "blending".

Program:

```
MOVE VIA pos1, pos2, pos3
```

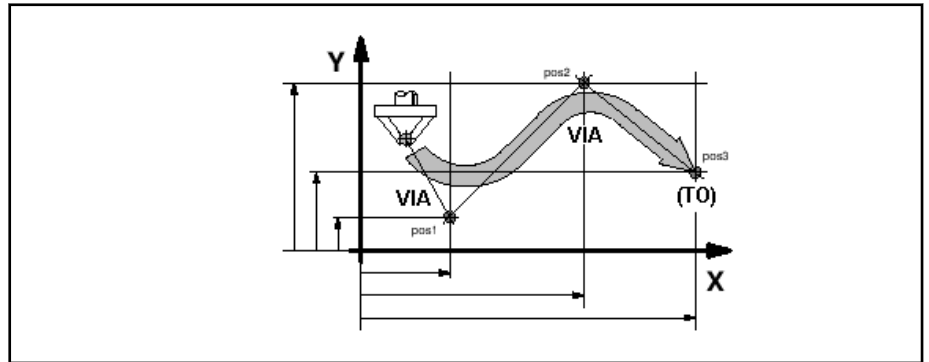


Fig. 12-3: MOVE VIA ...(blending)

MOVE TO ... The robot travels successively to the positions with intermediate stop. In this case, "TO" is selected as default addition.

Program:

```
MOVE TO pos1, pos2, pos3
```

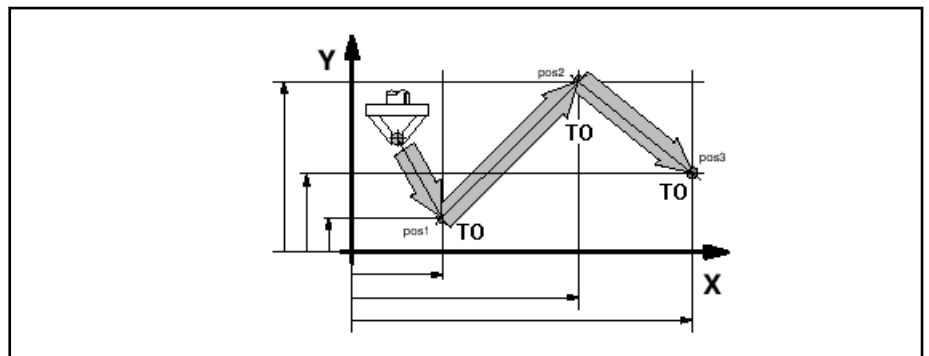


Fig. 12-4: MOVE TO ...

It is possible to link VIA and TO in a travel command.

MOVE VIA ... TO ... The robot travels past position 1 without stopping and then travels successively to the positions 2 and 3 with intermediate stop.

Program:

```
MOVE VIA pos1 TO pos2, pos3
```

RCL Motion Statements

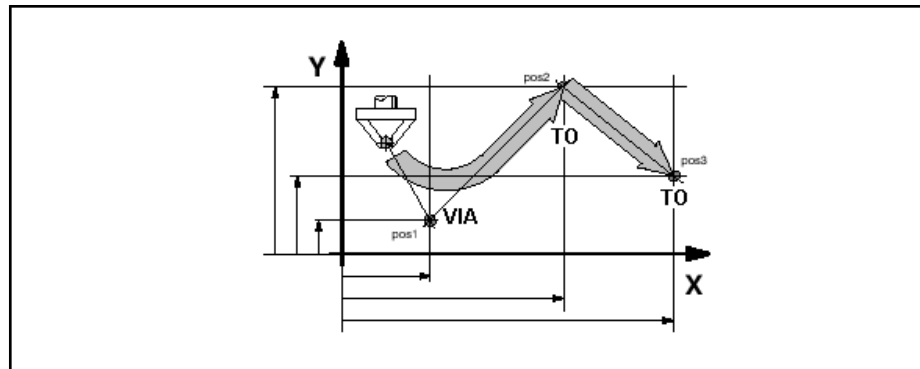


Fig.12-5: MOVE VIA ... TO ...

MOVE_REL*Syntax:*

```
MOVE_REL[Interpolation method][additional information][abort condition]
([EXACT]target point|APPROX target point[EXACT target point] )
```

The control interprets all positions programmed after MOVE_REL as incremental dimensions.

The coordinate values are distance values in the respective coordinate system and refer to the last commanded position of the robot.



The motion command MOVE_REL can be combined with all interpolation methods.

Program:

```
target=(10,10)
MOVE_REL target
```

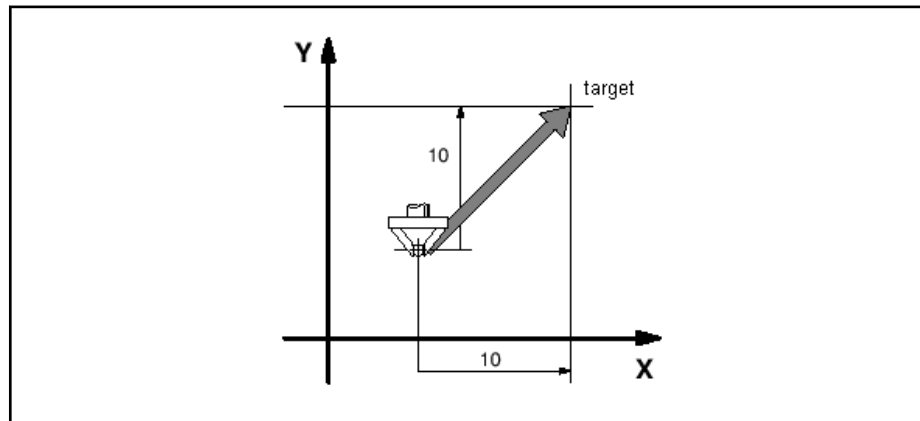


Fig.12-6: MOVE_REL

When traveling in incremental dimensions with MOVE_REL, the words "APPROX" and "EXACT" are used to specify an exact stop or traveling past.

MOVE_REL APPROX...

The robot travels past the positions defined in incremental dimensions without intermediate stop. See "R" and "blending".

Program:

```
MOVE_REL APPROX dis1,dis2,dis3
```

RCL Motion Statements

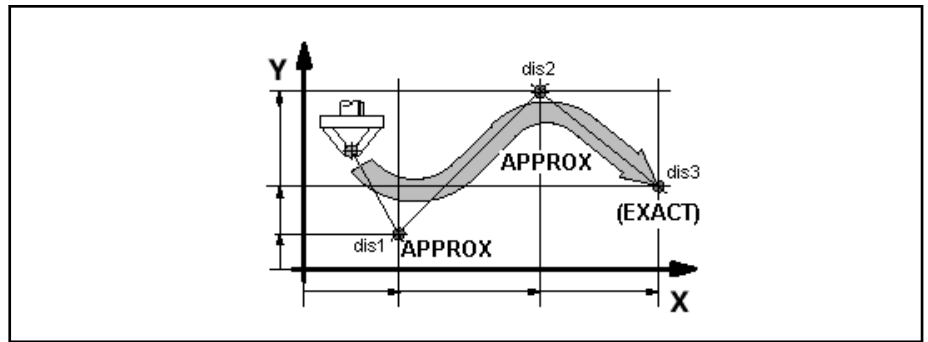


Fig. 12-7: MOVE_REL APPROX...

MOVE_REL EXACT...

The robot successively travels to the positions defined in incremental dimensions with intermediate stops. The word "EXACT" can be omitted when programming the travel command. The control automatically generates the command.

Program:

```
MOVE_REL EXACT dis1,dis2,dis3
```

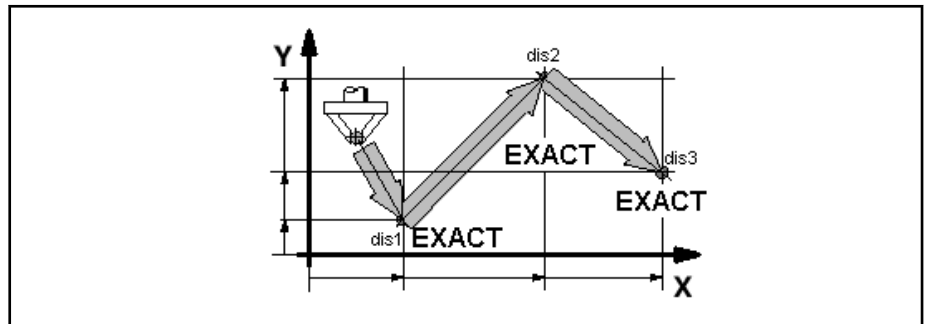


Fig. 12-8: MOVE_REL EXACT...

It is possible to link "APPROX" and "EXACT" in the travel command.

MOVE_REL APPROX ... EXACT ...

The robot travels past the first position without intermediate stop and then travels to the next positions 2 and 3 with intermediate stop.

Program:

```
MOVE_REL APPROX dis1 EXACT dis2,dis3
```

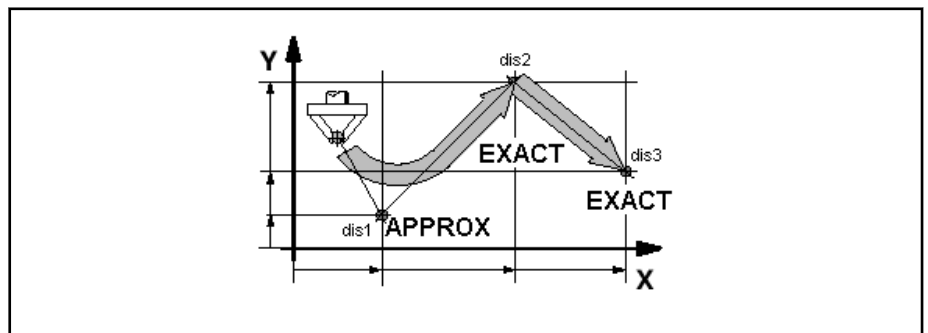


Fig. 12-9: MOVE_REL APPROX ... EXACT ...

RCL Motion Statements

Target point names The names of the point variables can be freely selected, see [Point Variables, page 58](#). For example, the names "palett1", "palett2" ... can be assigned to pallet points.

Due to reasons of clarity, the examples on the following pages, point specifications in absolute position are designated with "pos", e.g. "MOVE pos".

Point specifications in incremental dimensions are designated with "dis" (distance values), e.g. "MOVE_REL dis".

12.2.3 Interpolation Method

The MLC RobotControl supports one kinematic technique in three interpolation methods:

- LINEAR (straight line in space)
- CIRCULAR (circular path in space)
- PTP (synchronized point-to-point)

The interpolation methods refer to the interpolation in the space. Depending on the kinematics of the robot, the axis motions are executed after the inverse position kinematics (from space coordinates to axis coordinates).



Orientation axes of a kinematics are not transformed. For the interpolation, they are triggered in a fully synchronized (identical ramp times) way.

LINEAR interpolation method

The robot travels to the target point on a straight line. The straight line is geometrically defined by two points. Since the current robot position P1 is known to the control, it is sufficient to specify target point P2.

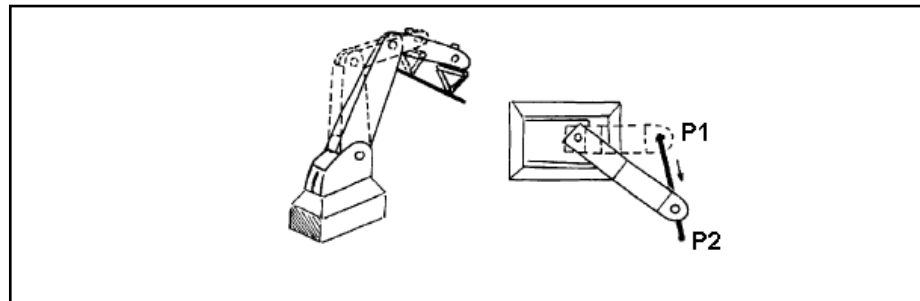


Fig. 12-10: LINEAR interpolation method

CIRCULAR interpolation method

The robot approaches the target point on a circular path in the space.

The circle is geometrically defined by three points.

Apart from specifying target point P2, an intermediate point has to be specified "PZ" so that the control can correctly calculate the circular path.

The control is familiar with point P1 as it was the last point approached. The intermediate point PZ is a point on the circular arc passed by the robot.

RCL Motion Statements

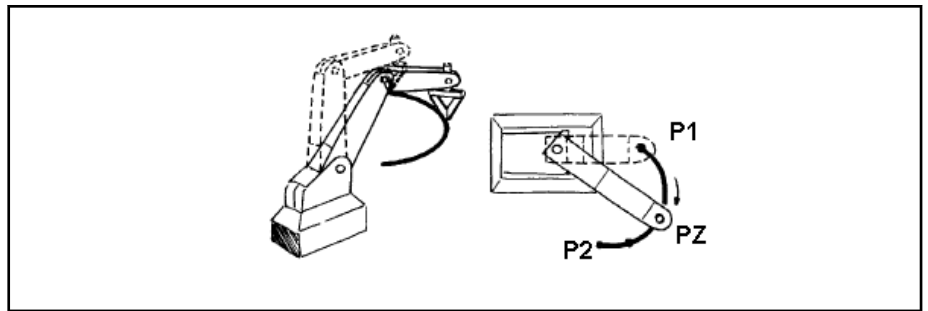


Fig. 12-11: CIRCULAR interpolation method

PTP interpolation method

PTP: Point-To-Point

The control calculates the motion of all axes so that they start and stop moving simultaneously. The same ramp times apply to all axes. In general, this is referred to as fully synchronous PTP. Depending on the lever ratio of the robot arms and the points P1 and P2, this can result in a spatial distance not defined in detail.

Specifying a target point is sufficient for the PTP method.

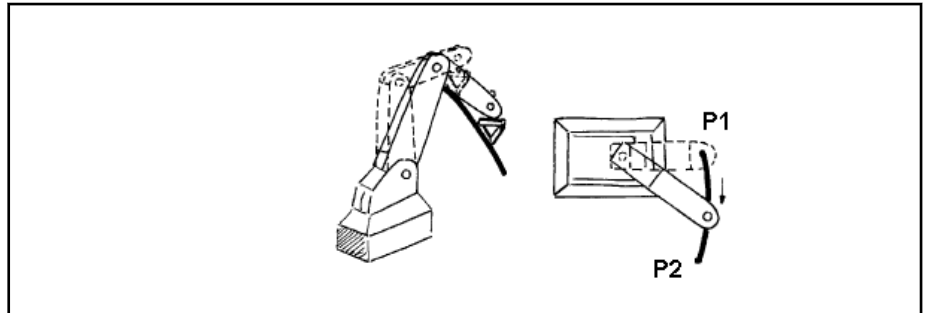


Fig. 12-12: PTP interpolation method

Statement-specific interpolation method

If a certain interpolation method is supposed to apply to only one motion statement, the interpolation method is programmed in the motion statement. The interpolation method is included in the travel command, directly following "MOVE" or "MOVE_REL".



Only one interpolation method can be in one motion function!

Example: LINEAR

Program:

```
MOVE_LINEAR TO pos1
MOVE_REL_LINEAR APPROX dis1 EXACT dis2
```

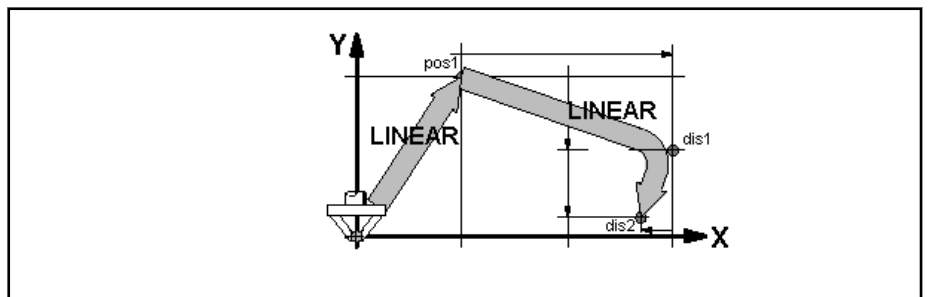


Fig. 12-13: Example: LINEAR

RCL Motion Statements

Example: CIRCULAR

The point specifications required for the CIRCULAR method, such as the intermediate point "pz" or "dz", end position "pos1" or dis1 have to be separated by a comma and placed in parentheses.

Program:

```
MOVE CIRCULAR TO (pz,pos1)
```

```
MOVE_REL CIRCULAR EXACT (dz,dis1)
```

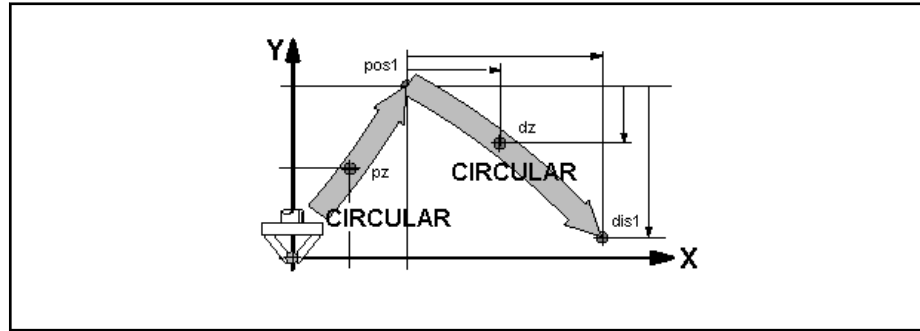


Fig. 12-14: Example: CIRCULAR

Example: PTP

Program:

```
MOVE PTP TO pos1
```

```
MOVE_REL PTP EXACT dis1
```

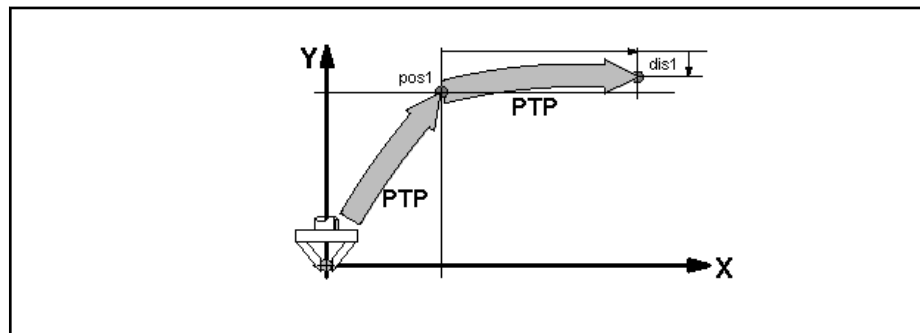


Fig. 12-15: Example: PTP

Global interpolation method

If an interpolation method is to apply to several motion statements, it can be specified as global interpolation method. The global interpolation method is specified by the following compiler statement.

Syntax:

```
;;INT= [LINEAR, CIRCULAR, PTP] ;Interpolation mode, all kinematics
```

This is integrated into the RCL program header by default. The defined interpolation method applies to all subsequent motion statements without any defined interpolation specification.

Example:

```
;;INT=LINEAR
MOVE pos1 ;The position pos1 and the point defined with the help of dis1,
           ;are approached by a robot on a straight line each.
MOVE_REL dis1
MOVE CIRCULAR(pz,pos2) ;Point pos2 is approached on a circular path.
```

RCL Motion Statements

The global interpolation method remains valid until it is replaced by another interpolation method.

Example:

```
;;INT=LINEAR

MOVE pos1          ;The robot approaches the axes of the kinematics 1 to the positions
MOVE pos2          ;pos1 and pos2 on a straight line each

;;INT=CIRCULAR

MOVE (pz1,pos3) ;It is approached towards the positions pos3 and pos4 and all remaining positions on a circle
MOVE (pz2,pos4) ;according to the interpolation specification CIRCULAR
```



If a CIRCULAR interpolation is specified as global method, the point values in the motion statement to which the interpolation specification is to apply, have to be valid point pairs.



If no interpolation method is specified and there is no global specification regarding an interpolation method, the control automatically selects PTP.



If several global interpolation methods are programmed, the specification of the interpolation method occurs in ascending line sequence and not according to the program sequence (subroutine, jumps, etc.).

Example:

PTP is used to approach "pos2" although the compiler statement ";;INT = PTP" was skipped in the program sequence.

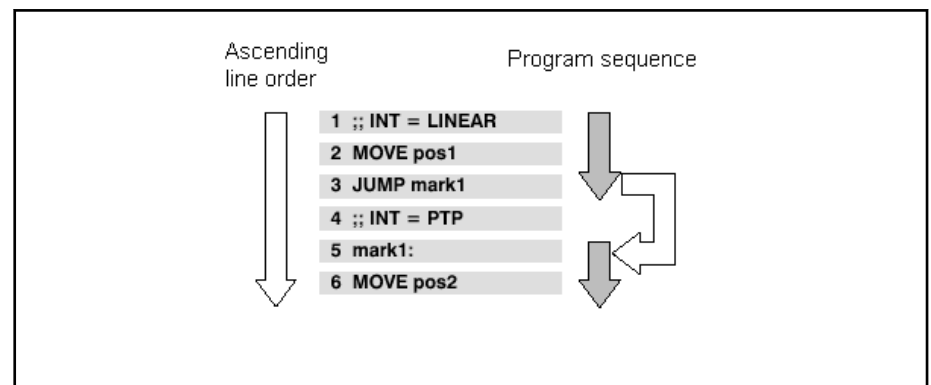


Fig. 12-16: Ascending line sequence vs. program sequence

RCL Motion Statements

12.2.4 Mode of Operation of Motion-Influencing RCL Statements and Parameters

Interpolation method: LINEAR/ CIRCULAR (RCL syntax)

	Explanation	Calculation	Effect
V	Path velocity		SV
VFACTOR	Path velocity override	$V_{eff} = V * VFACTOR$	SV
VFIX	Deactivates K-0-0240, RoCoFeedrate override		SV
A	Path acceleration		SV
J	Path jerk		SV
AFACTOR	Path acceleration override	$A_{eff} = A * AFACTOR$	SV
DFACTOR	Path deceleration override	$Deff = A * DFACTOR$	SV
AFIX	Deactivated K-0-0242, RoCoAcceleration override		SV
R	Blending radius		SV

SV

Block preparation

Fig. 12-17:

Mode of operation of motion-influenced RCL statements

K-parameter (PLC/ parameter editor)

	Explanation	Calculation	Effect
K-0-0240	RoCoFeedrate override	$V_{eff} = V_{eff} * (K-0-0240 / 100)$	SA
K-0-0242	RoCoAcceleration override	$A_{eff} = A_{eff} * (K-0-0242 / 100)$	SA
K-0-0243	RoCoDeceleration override	$Deff = Deff * (K-0-0243 / 100)$	SA
K-0-0014	Path acceleration, modification time	$Jeff = A_{eff} / K-0-0014$ (Acceleration in triangle mode) $Jeff = A_{eff} / t$ for $t \leq K-0-0014$ (Acceleration in trapezoidal mode)	SA
K-0-0015	Path deceleration, modification time	$Jeff = Deff / K-0-0014$ (Acceleration in triangle mode) $Jeff = Deff / t$ for $t \leq K-0-0014$ (Acceleration in trapezoidal mode)	SA

SA

Block execution, block processing

Fig. 12-18:

Mode of operation of motion-influenced parameters

Interpolation method: PTP (RCL syntax)

	Explanation	Calculation	Effect
V_PTP	decimal factor for the smaller value in A-0-0032/ A-0-0033 of the slowest axis participating in the motion	$V_PTPeff = (0.8 * A-0-0032/33) * V_PTP$	SV
AFACTOR	decimal factor for A-0-0034, Acceleration limit value bipolar, of the slowest axis participating in the motion	$A_PTPeff = (0.8 * A-0-0034) * AFACTOR$	SV
R_PTP	decimal factor for A-0-0230, Blending distance PTP	$R_PTPeff = A-0-0230 * R_PTP$	SV

SV Block preparation
Fig. 12-19: Mode of operation of motion-influenced RCL statements



A-0-0032/ A-0-0033/ A-0-0034: The axis limit values can be modified in P2 and BB.

The modification of the values becomes only effective when switching from P2 to BB.

12.2.5 Additional Information

Additional information, General

Apart from informing the control about the position of the target point and the path, information on the period or the velocity in which the robot has to approach the target point has to be specified.

The control also requires specifications on the motion period or the path velocity of the robot.

The value entered for acceleration indicates how fast the robot is supposed to reach this path velocity.

Additional Information: Velocity

The travel velocity is programmed in mm/s for LINEAR and CIRCULAR interpolation. In case of a PTP interpolation, programming is carried out as factor of the maximum axis velocity.



If no velocity specifications are programmed, the robot travels at default velocity values. These values are 10 mm/s or 10 inch/s (see K-0-0054) for path interpolation or 10% of the maximum axis velocity for synchronous PTP interpolation.

The velocity V_PTP is specified as a decimal factor of the maximum axis velocity for the synchronous PTP interpolation.

It is also possible to enter values in per cent, i.e. V_PTP = 80 % is equivalent to V_PTP = 0.8.

V

Description	V
Measuring unit	mm/s or inch/s (see K-0-0054)
Input range	1 to 99999
Switch-on state	25

Fig. 12-20: Velocity for LINEAR and CIRCULAR

RCL Motion Statements

V_PTP

Description	V_PTP
Measuring unit	Decimal factor
Input range	0.0001 to 9.9999 0.01% to 999.99%
Switch-on state	0.1

Fig. 12-21: Velocities for PTP

Motion optimization for orientation axes

The interpolator calculates the accelerations and velocities of the orientation axes in advance that result when the path velocity is not exceeded.

If these limit values are violated, the resulting path velocity and path acceleration reduce accordingly. This also applies to linear interpolations.



The motion specification is always carried out with the limit values of all orientation axes maintained!

Programming options

The velocity can be programmed as a

- global velocity specification,
- statement-specific velocity specification,
- statement-specific time specification,

Global velocity specification

If the path velocity remains constant for the entire program or for a larger section, a global velocity is to be specified.

The velocity value remains valid until it is modified by another global velocity specification.

Example:

```

V=750                                ;V = decimal expression. All the following
                                     ;motion functions with the interpolation method LINEAR and
MOVE LINEAR pos1                     ;CIRCULAR are provided with the specified velocity value of V = 750 mm/s
MOVE CIRCULAR (pz,pe)

V=750                                ;The velocity when approaching linearly towards the points pos1
MOVE LINEAR pos1                     ;and pos2 is V = 750 mm/s
MOVE LINEAR pos2

V=300                                ;It is approached towards the points place3 und place4 as well
MOVE LINEAR place3                   ;towards the remaining ones
MOVE LINEAR place4                   ;with V= 300

```

Statement-related velocity specification (WITH)

If the path velocity is supposed to apply to just one motion statement, the velocity specification is statement-specific, which means that it is specified with the motion statement.

Velocity designation and value assignment are part of the motion function to which they apply.

Programming occurs following the interpolation method with the keyword WITH.

Example: LINEAR:

```
MOVE_REL LINEAR WITH V=500 TO pos
```

RCL Motion Statements

Example: PTP:

```
MOVE PTP WITH V_PTP=70% TO pos
```

The global velocity specifications do not influence the statement-specific specifications.

Override

Via the kinematic parameters

K-0-0240, RoCoFeedrate override,

K-0-0241, RoCoFeedrate override control word,

K-0-0242, RoCoAcceleration override and

K-0-0243, RoCoDeceleration override,

globally valid override parameters can be set for the kinematics. They directly affect the motion specification.

VFIX, VFIX_PTP, AFIX: Disabling override

The RCL commands VFIX, VFIX_PTP and AFIX can disable the following override parameters valid for the kinematics:

Description	Disables parameter
VFIX, VFIX_PTP	K-0-0240, RoCoFeedrate override
AFIX	K-0-0242, RoCoAcceleration override

Fig. 12-22: VFIX, AFIX: Disable override

VFIX, AFIX and VFIX_PTP are only effective block by block. That means that these values can/may only be programmed together with the keyword WITH within a MOVE command. After this block, locking is automatically canceled. Influencing the velocity or acceleration is not effective/possible for this block via potentiometer.

Examples:

```
MOVE PTP WITH VFIX_PTP = 0.8 TO @pos1 ; potentiometer will have no effect
```

```
MOVE PTP WITH V_PTP = 0.8 TO @pos1 ; potentiometer will be considered
```

Additional Information: Velocity Factor

Velocity and time specifications can be modified using the programmed velocity factor "VFACTOR".

All velocity specifications are multiplied by the velocity factor.

Time specifications are divided by the "VFACTOR". The values determined apply to the subsequent motion functions.

The factor refers to the velocity and time specifications defined in the program.

The factor can also be specified in per cent, i.e. VFACTOR = 180 % is equivalent to VFACTOR = 1.8.

VFACTOR

Description	VFACTOR
Measuring unit	1 or %
Input range	0.0001 (0.01%) to 6 (600%)
Switch-on state	1.0 (100%)

Fig. 12-23: Velocity factor

RCL Motion Statements



When starting an RCL program, AFACTOR, DFACTOR and VFACTOR are set to 1.0 (100 %)

Example:

```
VFACTOR=180%
;The VFACTOR of 180% (= Factor 1.8) increases the
;the path velocity to the positions 1, 2,
;3 and 4. It is moved to 180% of Vmax to pos1
;(initial state 10%, Factor 1.8)

MOVE PTP TO pos1

V=200
MOVE LINEAR TO pos2
MOVE LINEAR WITH V=100 TO pos3
MOVE LINEAR WITH T=10 TO pos4
;It is moved towards pos 2 with a velocity of 360 mm/s
;It is moved towards pos3 with a velocity of 180 mm/s
;It is moved towards pos4 within 5.6 s
;(10/1.8= 5.6)
```

Effective velocity The actual effective velocity consists of the programmed velocity (V), the velocity factor (VFACTOR) and the velocity override (K-0-0240).

Example:

K-0-0054 Bit 0 = 0 (mm)

Programmed V = 1000.00 mm/s, VFACTOR = 80% and K-0-0240 = 100 %.
An effective

$V_{\text{eff}} = V * \text{VFACTOR} * \text{K-0-0240} = 800 \text{ mm/s}$ results

Additional Information: Acceleration

From the acceleration specifications, the control identifies how fast the robot has to reach its specified velocity.



Acceleration specifications can only be programmed for LINEAR and CIRCULAR interpolation.

In the PTP method, the robot accelerates based on the maximum values defined in the axis data set. The AFACTOR can be used to modify the acceleration.

Programming options for the acceleration specification

For LINEAR and CIRCULAR interpolation, the acceleration specification can be programmed as

- global specification
- statement-specific specification
- acceleration factor.

The acceleration factor is also effective for the PTP method.

Global acceleration specification (Global specification)

Interpolation method: LINEAR and CIRCULAR

If the acceleration remains constant for the entire program or for a large section, a global acceleration is to be specified.

Designation and value assignment are part of an individual statement at the beginning of the program or program section.

The acceleration value remains valid until it is modified by another global specification.

RCL Motion Statements

Example:

```

O=30          ;All the following motion functions with the
              ;interpolation method LINEAR and CIRCULAR are provided with an
              ;acceleration value of A=30 mm/s2
V=100         ;When moving towards the positions pos1 to pos3, the acceleration till a
              ;path velocity of V=100 is reached
              ;is A=30 mm/s2

MOVE CIRCULAR (pz,pos1)
MOVE LINEAR pos2
MOVE LINEAR pos3

O=15          ;The positions pos4, pos6 and all the remaining ones with LINEAR and
              ;CIRCULAR interpolations are provided with a specified value of
              ;A=15 mm/s2

MOVE LINEAR pos4
MOVE PTP pos5
MOVE LINEAR pos6
    
```



pos5 is approached with the PTP interpolation. The acceleration specification $A = 15 \text{ mm/s}^2$ does not apply. (K-0-0054 Bit 0 = 0 (mm))

Statement-related acceleration specification (WITH)

A

If a path acceleration is supposed to apply to just one motion command, the acceleration is specified locally.

Description	A
Measuring unit	mm/s ² or inch/s ² (see K-0-0054)
Switch-on state	10

Fig. 12-24: Acceleration for LINEAR and CIRCULAR interpolation



Local acceleration specifications can only be used for LINEAR and CIRCULAR interpolation.

In the PTP method, the robot accelerates based on the maximum values defined in the axis data set. The AFACTOR can be used to modify the acceleration.

Name and value assignment are part of the motion function for which the acceleration should apply.

Programming occurs after the interpolation method with the keyword WITH.

Example: LINEAR:

```
MOVE LINEAR WITH J=1000 TO pos
```

Example: CIRCULAR:

```
MOVE_REL CIRCULAR WITH J=2000 EXACT (dz,dis1)
```



The global acceleration specifications do not influence the statement-specific specifications.

It is possible to specify velocity and acceleration locally within a motion function. When programming, the specifications are separated by a comma. There is no specified order.

Program:

```
MOVE_REL CIRCULAR WITH V=120, A=35, J=1000 EXACT (dz,dis1)
```

Additional Information: Acceleration Factor

The acceleration factor "AFACTOR" can influence the acceleration specifications again. The acceleration factor is multiplied by all acceleration specifications. The values calculated this way apply to all subsequent motion functions.

 The AFACTOR is not effective for the statement-specific acceleration specification (in contrast to the DFACTOR).

Example:
K-0-0054 Bit 0 = 0 (mm)
Programmed A= 300.00 mm/s² and AFACTOR= 90%. Result
 $A_{\text{eff}} = A \cdot \text{AFACTOR} = 270 \text{ mm/s}^2$.

AFACTOR	Description	AFACTOR
	Measuring unit	%
	Input range	0.0001 (0.01%) to 1.2 (120%)
	Switch-on state	1.0 (100%)

Fig. 12-25: Acceleration factor

 The AFACTOR is not effective if the acceleration is programmed statement-specifically.

 After the program start, the AFACTOR, DFACTOR and VFACTOR are reset to 1.0 (100 %).

Example:

```
O = 1000 ;1000 [mm/s²]
V = 200 ;200 [mm/s]

AFACTOR = 50% ;The AFACTOR of 50 %
               ;(=Factor 0.5) increases the path acceleration

MOVE PTP TO pos1 ;The path acceleration has no influence on the PTP
                  ;interpolation
MOVE CIRCULAR TO (pz,pos2) ;Effective A = 500 mm/s² (0.5 * 1000)
MOVE LINEAR WITH O=100 TO pos3 ;Effective A = 100 mm/s², AFACTOR has no effect
```

Effective acceleration The actual effective acceleration consists of the programmed acceleration (A), the acceleration factor (AFACTOR) and the acceleration override (K-0-0242).

Example:
K-0-0054 Bit 0 = 0 (mm)
Programmed A = 300.00 mm/s², AFACTOR= 90% and K-0-0242 = 100 %.
An effective
 $A_{\text{eff}} = A \cdot \text{AFACTOR} \cdot \text{K-0-0242} = 270 \text{ mm/s}^2$.

Additional Information: Deceleration Factor

Analog to the acceleration factor, the deceleration factor can be set as impact on the deceleration via the command "DFACTOR".

RCL Motion Statements

The deceleration factor is multiplied with all deceleration specifications. The values calculated this way apply to all subsequent motion functions.

Example:

K-0-0054 Bit 0 = 0 (mm)

Programmed $A = 300.00 \text{ mm/s}^2$ and DFACTOR= 90%. Result: an effective

$$D_{\text{eff}} = A * \text{DFACTOR} = 270 \text{ mm/s}^2.$$

DFACTOR

Description	DFACTOR
Measuring unit	%
Input range	0.0001 (0.01%) to 1.2 (120%)
Switch-on state	1.0 (100%)

Fig. 12-26: Deceleration factor

Effective deceleration

The actual effective deceleration consists of the programmed acceleration (A), the deceleration factor (DFACTOR) and the deceleration override (K-0-0243).

Example:

K-0-0054 Bit 0 = 0 (mm)

Programmed $A = 300.00 \text{ mm/s}^2$, DFACTOR= 90% and K-0-0243 = 100 %. An effective

$$D_{\text{eff}} = A * \text{DFACTOR} * \text{K-0-0243} = 270 \text{ mm/s}^2$$



Acceleration and deceleration are based on the same **A** value.

The overlay occurs separately with the AFACTOR and the DFACTOR.

Example:

```

A = 1000                                ;1000 [mm/s²]
V = 200                                ;200 [mm/s]

DFACTOR = 50%                          ;The DFACTOR of 50 %
                                        ;(=Factor 0.5) increases the path deceleration

MOVE PTP TO pos1                       ;The path acceleration has no influence on the PTP
MOVE CIRCULAR TO (pz,pos2)              ;interpolation
MOVE LINEAR WITH O=100 TO pos3          ;Effective D = 500 mm/s² (0.5 * 1000)
                                        ;Effective D = 50 mm/s² (0.5 * 100)

```

Additional Information on the Jerk

From the jerk specifications, the control identifies how fast the robot has to reach its specified acceleration.

J

Description	J
Measuring unit	mm/s³ or inch/s³ (see K-0-0054)
Switch-on state	1000000

Fig. 12-27: Path jerk for LINEAR and CIRCULAR interpolation

RCL Motion Statements



Jerk specifications can only be programmed for LINEAR and CIRCULAR interpolation in the block transition modes BLOCK_SLOPE and CONT_SLOPE.

In the block transition mode PROGR_SLOPE, an acceleration change time can be set with the parameter K-0-0014 and a deceleration change time can be set with the parameter K-0-0015.

Programming options for the jerk specification

The jerk specification can result as

- global specification
- statement-specific specification

Global jerk specification (Global specification)

If the jerk remains constant for the entire program or for a large section, a global jerk is to be specified.

Designation and value assignment are part of an individual statement at the beginning of the program or program section.

The jerk value remains valid until it is modified by another global specification.

Example:

```
J=1000           ;All the following motion functions with the
                  ;interpolation method LINEAR and CIRCULAR are provided with a
                  ;jerk of J=1000 mm/s³.
O=30             ;The jerk until reaching the path acceleration of A=30 mm/s² is
                  ;When approaching the points pos1 to pos3 J=1000 mm/s³
MOVE CIRCULAR (pz, pos1)
MOVE LINEAR pos2
MOVE LINEAR pos3

J=500            ;the positions pos4, pos5 and all remaining motion functions
                  ;with LINEAR and CIRCULAR interpolation have a value specification of
                  ;J=500 mm/s³
MOVE LINEAR pos4
MOVE LINEAR pos5
MOVE PTP pos6    ;Pos6 is approached with PTP. Thus, the jerk specification is not effective.

PROGR_SLOPE      ;The block transition mode PROGR_SLOPE becomes ineffective for all following
                  ;motion functions.
MOVE LINEAR pos7 ;Pos7 is approached with the block transition mode PROGR_SLOPE.
                  ;Thus, the jerk specification is not effective.
```

Statement-related jerk specification (WITH)

If a path jerk is supposed to apply to just one motion command, the jerk is specified locally.

Name and value assignment are part of the motion function for which the jerk should apply.

Programming occurs after the interpolation method with the keyword WITH.

Example: LINEAR:

```
MOVE LINEAR WITH J=1000 TO pos
```

Example: CIRCULAR:

```
MOVE_REL CIRCULAR WITH J=2000 EXACT (dz, dis1)
```



The global jerk specifications do not influence the statement-specific specifications.

It is possible to specify velocity, acceleration and jerk locally within a motion function. When programming, the specifications are separated by a comma. There is no specified order.

Program:

```
MOVE_REL CIRCULAR WITH V=120, A=35, J=1000 EXACT (dz,dis1)
```

12.3 Motion-influencing Statements

12.3.1 General Information

In addition to the direct motion statements described above, the following statements also influence the motion sequences:

- Synchronization statements SYNC, SYNCHRON, SYNCHRON_END
- Acceleration and deceleration between motion statements (block transitions) BLOCK_SLOPE, PROGR_SLOPE and CONT_SLOPE

12.3.2 Belt Synchronization

With the "belt synchronization" function, the robot motion can be synchronized to an assembly or conveyor belt with respect to position and orientation.

In this case, the belt can travel either forward or backward, change velocity or stop.

The presettings are selected using the special functions [SPC_FCT: 21, belt type, page 100](#), and [SPC_FCT: 53, belt-synchronized workspace with belt type 4, page 102](#).

The belt must be a straight line. The line can be arbitrarily positioned in space; see MLC parameter "K-0-0070" and the following. The belt motion is registered by a distance measuring system or commanded by the MLC with "MC_MoveVelocity" for example.

The statements SYNC, SYNCHRON and SYNCHRON_END are available in the RCL for functionality programming.

Syntax:

```
SYNC belt variable[, variable] condition
SYNCHRON [kinematic name] belt variable
SYNCHRON_END [kinematic name] belt variable
```

The belt variable has to be declared in the program's declaration; see also [Communication Channels to PLC, General, page 64](#) and has to be assigned to a kinematics.

Example:

Declaration of belt variables of several belts to one kinematics

```
Kin1.belt:    501=belt1, 502=belt2
screwdr.belt: 503=screw_belt ;same belt like belt1
```



Altogether, up to 4 belts can be declared per kinematics. The belt names and belt numbers must be different, even if several kinematics use the same measuring system for one belt variable.

Synchronized motion can only occur on belts that have also been declared for this kinematics.

The component names and axis names must be specified accordingly for the assigned belt.

RCL Motion Statements



The belt synchronization with the block transition modes `PROGR_SLOPE` and `CONT_SLOPE` is currently not supported.

Programming belt synchronization

The belt variable of data type "REAL" contains the counter value of the belt measuring system.

The belt variable can only be queried with comparative operations `>=` and `<=` or used in the program with "WAIT UNTIL", "IF" or "SYNC".

Example:

```
WAIT UNTIL belt_1>=6 ;belt variable in WAIT UNTIL
```



The program waits at the SYNC command till the belt variable has met the criterion.

The "SYNC" command sets the belt variable to zero or to the reset value (SPC_FCT 28).

Zeroing can take place depending on a condition.¹⁾

The following example shows possibilities for using the SYNC command.

Example:

```
SYNC belt_1>=300 ;subtracts 300 from the belt counter if the current
                  ;counter reading is ≥ 300
SYNC belt_1, li_scr=1 ;resetting to zero is executed depending on a condition
```

The following example shows how to execute the non-waiting modulo processing.

Example:

```
IF belt_1>=300      ;initially checks if
                   ;counter reading is ≥ 300
THEN
BEGIN
  SYNC belt_1>=300 ;belt counter is reset in any case, as condition is correct.
END
```

The "SYNCHRON" command enables the belt synchronization. From now onwards, all programmed motions are synchronized with the belt in terms of position and orientation. The synchronization takes place as soon as the programmed target point (moved with the counter) is located within the belt-synchronous workspace.

"SYNCHRON_END" is used to switch off the synchronization.

Example:

```
SYNCHRON Kin1 belt_1 ;belt variable and SYNCHRON statement
MOVE LINEAR TO pos_1 ;(the pos_1 belt counter has a valid value > 0)
SYNCHRON_END belt_1
```

Example:

```
;;COMPILER= RCL3
;;KINEMATICS: (1=robi_1)      ;definition of kinematic names
;;robi_1.JC_NAMES=a1,a2,a3,b1 ;b1 is a dummy for belt variables
;;robi_1.WC_NAMES=k1,k2,k3,b_c1

PROGRAM beltsyn
VAR:
  CONST :
```

¹⁾ Subtracts a value (similar to modulo).

```

C_BELT_NO_1 = 1,          ; belt number
C_BK_PTP = 4              ; belt type PTP
BELT :
500 + C_BELT_NO_1 = belt1 ; channel / belt name

; declaration of special functions
SPC_FCT : 21 = belt_kind (VALUE INTEGER: belt_no
                           VALUE INTEGER: kind_belt)
SPC_FCT : 53 = belt_area (VALUE INTEGER: BeltNo
                           JC_POINT: @Base
                           REAL: rTotalLength
                           REAL: rBeginLength)

INPUT: 1=i1,2=i2

BEGIN ; belt type specification
belt_kind ( C_BELT_NO_1, C_BK_PTP ) ; specifying belt-synchronous operating range
rTotalLength = 560
rBeginLength = 400
Base = (280, 140, -600, 0, 0, 0)
@Base = JC(Base)
belt_area ( C_BELT_NO_1, @Base, rTotalLength, rBeginLength)

SYNC belt1,i1=1 ;waits until the PLC has set i1=1 to reset the
                ;belt counter belt1 (501)
SYNCHRON robi_1 belt1 ;synchronization of kinematics robi_1 to belt1

MOVE robi_1 LINEAR TO robi_1.POS
WAIT UNTIL belt1>=1000
SYNCHRON_END robi_1 belt1

PROGRAM_END

```

12.3.3 Block Transitions (BLOCK_SLOPE, PROGR_SLOPE, CONT_SLOPE)

Syntax:

BLOCK_SLOPE

PROGR_SLOPE

CONT_SLOPE

In the normal motion sequence, the triggered axes are accelerated to the programmed velocity with each MOVE statement. They travel at the programmed velocity and are then decelerated again to $V = 0$ to reach the programmed position.

The velocity change is referred to as "SLOPE". Accordingly, the acceleration phase is known as up-slope and deceleration as down-slope.

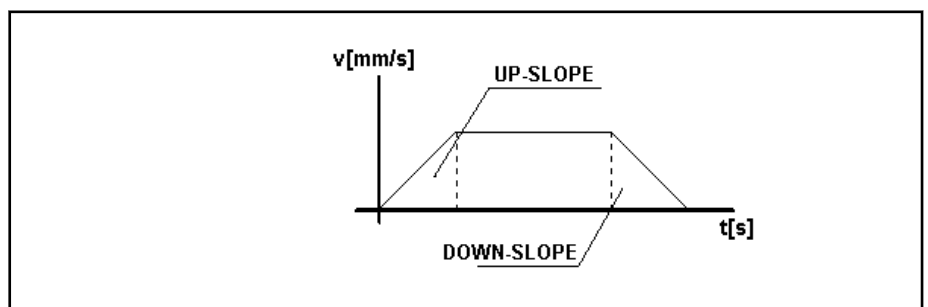


Fig. 12-28: Up-slope and down-slope

General Information

The block transition types PROGR_SLOPE and CONT_SLOPE are used to travel the blending motion commands altogether without intermediate stops. The block transitions are executed at a constant command velocity. If the blocks are programmed with different velocities, a ramp-shaped velocity change results between the motion commands.

RCL Motion Statements

- The block transition mode BLOCK_SLOPE is used for a block by block acceleration and deceleration.
- Enabling SLOPE mode** The initial setting for the slope mode is BLOCK_SLOPE.
- This slope mode can be switched in the RCL program. The RCL standard functions BLOCK_SLOPE, PROGR_SLOPE and CONT_SLOPE are defined for this purpose.
- When enabling a SLOPE type, the previous one is switched off.



For each slope mode change, the transition velocity is always 0 !

- BLOCK_SLOPE** The operation mode BLOCK_SLOPE handles all velocity specifications and acceleration specifications in a block-specific way. Acceleration and deceleration take place at the beginning and at the end of a motion command. Thus, there is no transition velocity between the two blocks. Via the blending radius, a velocity blending is achieved when the subsequent block was started too early.
- PROGR_SLOPE** The operation mode PROGR_SLOPE is a block-comprehensive velocity and acceleration specification. The robot accelerates at the start of a coherent motion sequence and decelerates at its end. At block transitions, the velocity is kept constantly if no velocity change has been programmed. Spatial blending via circular segments is executed via the blending radius.
- CONT_SLOPE** The operation mode CONT_SLOPE is a block-comprehensive velocity and acceleration specification. The robot accelerates at the start of a coherent motion sequence and decelerates at its end. At block transitions, the velocity is kept constantly if no velocity change has been programmed. Spatial blending via spline segments is executed via the blending radius.



For more information on the spatial blending via blending radius, see below: [chapter 12.3.6 "Spatial Blending" on page 134](#)

Programming the SLOPE mode is explained below using a few RCL program examples. Its effect is shown in the figures below.

Example 1:

```

PROGRAM ex_1
BEGIN
;;INT=LINEAR
V=800,A=1000
MOVE TO beg_point

MOVE TO point_center
MOVE TO end_point

PROGR_SLOPE
MOVE VIA beg_point
MOVE VIA point_center
MOVE TO end_point
MOVE VIA RAMP_1
MOVE VIA RAMP_2
MOVE VIA beg_point
MOVE VIA point_center
MOVE TO end_point

BLOCK_SLOPE
HALT
PROGRAM_END

```

;(1)
;(2)
;(3)
;(4)
;(5) Programmslope nicht aktiv, d.h. der Roboter wird in jedem Satz mit A=1000 mm/s² beschleunigt und am Ende des Satzes wieder abgebremst.
;(6)
;(7)
;(8) PROGR_SLOPE wird aktiviert
;(9)
;(10)
;(11)
;(12)
;(13)
;(14)
;(15)
;(16)
;(17)
;(18)
;(19)

In the example, bit 0 = 0 (mm) is configured in the parameter K-0-0054.

RCL Motion Statements

In the first part of the program, blocks 5 to 7, the default slope mode BLOCK_SLOPE is active. The robot accelerates in each block at $A = 1000 \text{ mm/s}^2$ and decelerates at the end of the block, see slope, Fig. 1.

In block 8, PROGR_SLOPE is switched on. As a result, the robot accelerates in block 9 at $A = 1000 \text{ mm/s}^2$.

The block transitions from 9 to 10 and 10 to 11 are executed at constant velocity.

At the end of block 11, the robot decelerates with the programmed deceleration of 1000 mm/s^2 ; see slope, Fig. 2.

If the travel distance in the MOVE TO block (block 11) is not sufficient to decelerate the robot with the slope function, the deceleration has been initiated in the previous block (block 10, slope, Fig. 3).

The acceleration phase can run across any number of blocks, e.g. block 12 to block 13, slope, Fig. 3. This also applies to the deceleration phase, e.g. block 15 to block 16, Fig. 3.

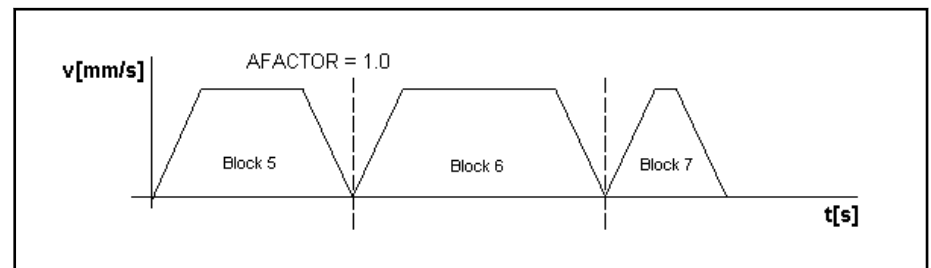


Fig. 12-29: Slope, Fig. 1 - BLOCK_SLOPE

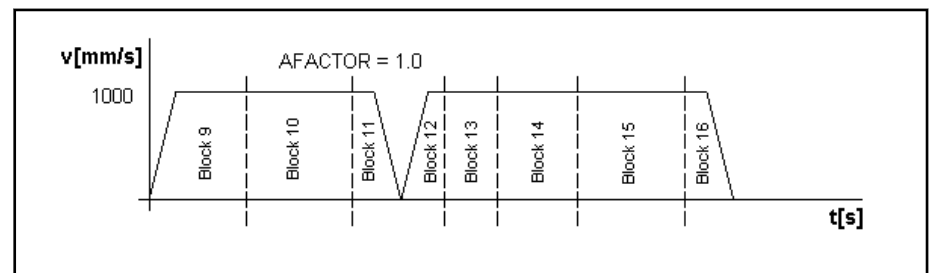


Fig. 12-30: Slope, Fig. 2 - PROGR_SLOPE

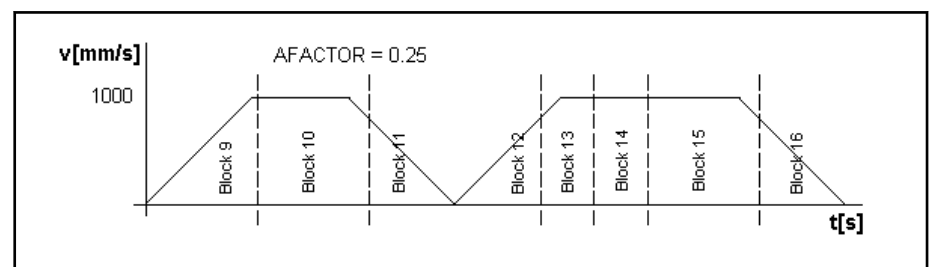


Fig. 12-31: Slope, Fig. 3 - PROGR_SLOPE

Motion abort

Abort of a coherent motion in the RCL program

A motion sequence with the slope modes PROGR_SLOPE and CONT_SLOPE is interrupted by the following RCL commands:

- Synchronous input prompt
- WAIT
- PAUSE
- HALT

RCL Motion Statements

- BLOCK_SLOPE
- Several blocks without travel information, e.g. calculations, variable assignments, setting of output signals

The number of possible blocks depends on the length of the preceding travel blocks and the type of the assignments or calculations performed.

Example 2:

```

PROGRAM ex_3
;;INT=LINEAR           ;Interpolation method LINEAR applies to the following motion commands
BEGIN
V=800, O=1000
PROGR_SLOPE           ;PROGR_SLOPE is switched on

MOVE LINEAR VIA beg_point ;End velocity = 0, since the next command is a synchronous
                        ;command
WAIT 1
MOVE LINEAR VIA point_center ;It is traveled in a velocity-blending manner.
                        ;The end velocity is calculated.

MOVE TO end_point
PROGRAM_END

```

Changing the interpolation method

The global acceleration and deceleration behavior can be enabled irrespective of the interpolation method. Block transitions without any change in the interpolation method are executed as described under [Block Transitions \(BLOCK_SLOPE, PROGR_SLOPE, CONT_SLOPE\)](#), page 129..

Switching between linear and circular interpolation

When switching from linear to circular interpolation and vice versa, the block transitions are executed as described in [Block Transitions \(BLOCK_SLOPE, PROGR_SLOPE, CONT_SLOPE\)](#), page 129.

Example 3:

```

PROGRAM ex_4
;;INT=LINEAR
BEGIN
V=800, A=1000
PROGR_SLOPE
MOVE VIA beg_point
MOVE CIRCULAR VIA (int_point1,circular_end1)
MOVE CIRCULAR TO (int_point2,circular_end2)
MOVE TO pnt_center
MOVE CIRCULAR TO (int_point3, circular_end3)
HALT
PROGRAM_END

```

Switching between LINEAR and PTP interpolation

Before switching from LINEAR to PTP interpolation and vice versa, the motion sequence has to be completed by a "MOVE TO" block so that a controlled transition can be executed.

If the switching occurs during a coherent motion, the velocity is modified in jumps.

Example 4:

```

PROGRAM ex_5
BEGIN
PROGR_SLOPE
V_PTP=1
MOVE PTP TO beg_point
MOVE LINEAR TO p1
MOVE PTP VIA p2
MOVE LINEAR VIA end_pnt

```

```
V_PTP=0.5
MOVE PTP TO end_pnt_1
HALT
PROGRAM_END
```

Exceeding axis limit values

Due to the coordinate transformation, it cannot be excluded that the limit values of individual machine axes are exceeded during the path operation. While the program is running, it can only be monitored.

If an error occurs, the monitoring function triggers one of the following error messages:

- Interpolator stop, axis X
- Axis velocity too high, axis X

X is the number of the respective machine axis.

In this case, the software engineer has to modify the program at the respective positions. Automatic velocity adjustment is not possible, as it would contradict the requirement for keeping the path velocity as constant as possible.

12.3.4 Look-Ahead Across Several Blocks at PROGR_SLOPE and CONT_SLOPE

In the slope modes PROGR_SLOPE and CONT_SLOPE, the calculation of possible block transition velocities is calculated before across several blocks. It is considered, that the robot is provided with sufficient distance to stop at the end of the travel motion.

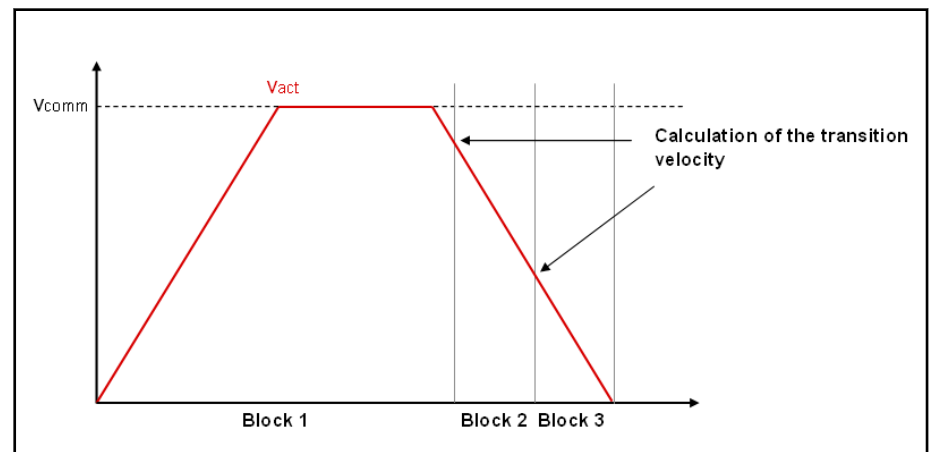


Fig. 12-32: Look-ahead across several blocks

The figure above shows the operating principle of an integrated look-ahead. The programmed command velocity is not reached in block 2 and block 3. The end velocity of the first block is adapted to the deceleration distance.

12.3.5 Late Commanding

The MLC RobotControl provides the option to transmit motion commands late. This is possible for PROGR_SLOPE and CONT_SLOPE as well as for BLOCK_SLOPE. If another block is commanded during block traveling, the end velocity of the current block might also be influenced.

RCL Motion Statements

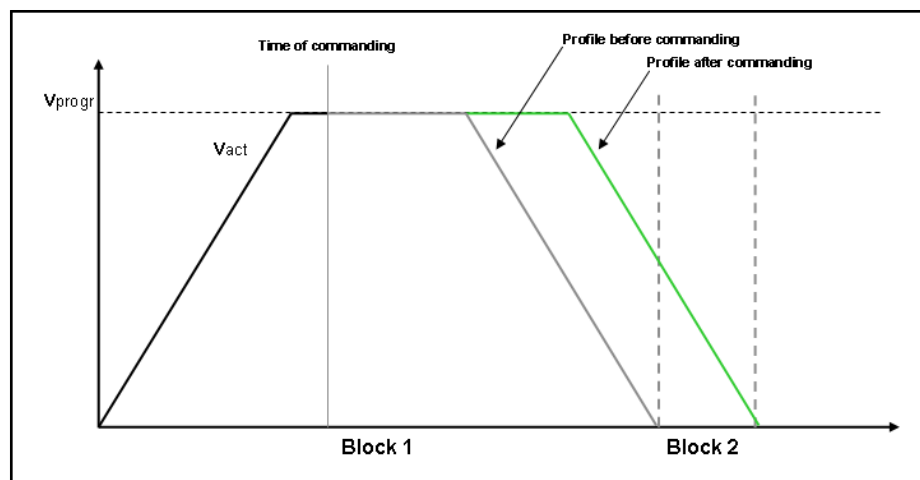


Fig. 12-33: Increasing end velocity by late commanding

The figure describes such a procedure. Block 1 starts with the acceleration to the programmed velocity. The block is interpolated so that the kinematics is in standstill after the programmed block. At the time "Time of commanding", block 2 is commanded while block 1 is active. This increases the possible end velocity of the first block, even though the block has already been started.

Adapting the end velocity includes several blocks beforehand. This mechanism is completely running in the background.

12.3.6 Spatial Blending

Overview

To carefully treat the mechanics of the robot and to decrease the time needed for the motion sequence at the same time, the programmed target points can be blended. The consequence is a continuous and harmonic motion at block transitions.

Definition of "blending": Ensuring uninterrupted velocity characteristics for block transitions using a defined deviation from the programmed path.

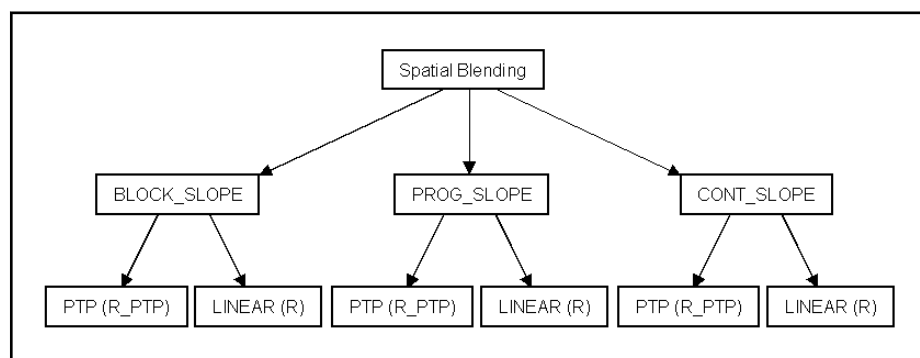


Fig. 12-34: Spatial blending

Blending radius

Depending on the interpolation method of the default variables "R" or "R_PTP", spatial blending is influenced in the RCL program.

The default variable "R_PTP" is programmed for the PTP interpolation. This factor can be programmed in decimals or in per cent. It is calculated with parameter A-0-0230, Blending distance PTP, of all axes.

RCL Motion Statements

The default variable "R" like in radius is used for the LINEAR interpolation. Absolute values are programmed for "R". The robot type selected defines the unit for "R".

R, R_PTP

Description	R	R_PTP
Measuring unit	mm or inch or ° (see K-0-0054)	-
Switch-on state	0	0
Effect	absolute	A-0-0230, Blending distance PTP

Fig. 12-35: Blending radius

Example MOVE statement with R or R_PTP

```
;;INT=LINEAR ;interpolation method LINEAR applies for the following motion statements
R = 50 ;R = 50 mm now applies to all blocks

MOVE LINEAR WITH R=20 VIA p1;R = 20 mm, as block-related
MOVE LINEAR TO p2 ;R = 50 mm

;;INT=PTP ;interpolation method presetting
R_PTP = 1.4 ;R_PTP = 140% of A-0-0230 of the participating axes
;now applies globally
MOVE PTP WITH R_PTP=1.2 VIA p3 ;R = 120%, since defined block-related
MOVE PTP TO p4 ;R = 140%
```



If R= 0 or R_PTP= 0 are programmed, spatial blending is switch-off.

In general, the following applies to BLOCK_SLOPE

- The spatial deviation in the blending range **depends on velocity**.
- The individual motion blocks are started earlier in terms of the preset blending distances. The path velocity results from the geometric addition of single axis velocities.

In general, the following applies to the PROGR_SLOPE and CONT_SLOPE

- The spatial deviation in the blending range **does not depend on the velocity**.
- In the blending range, the path is specified by a block generated by the control. In a related motion sequence, attempts are made to keep the velocity constant.



Points approached in the circular interpolation or from which approaching starts with the circular interpolation, are not spatially blended.

This applies to BLOCK_SLOPE as well as to PROGR_SLOPE and CONT_SLOPE.

Spatial Blending for PROGR_SLOPE and CONT_SLOPE

PTP Blending for PROGR_SLOPE and CONT_SLOPE

A blending distance in degrees or mm can be saved for each axis for its end position in parameter A-0-0230, Blending distance PTP. A blending range around the target point is defined based on these distances. These distances can be changed with a factor (R_PTP) in the RCL program by the user. All of the stored distances for the individual axes are equally influenced so that blending ranges are either compressed or extended.

RCL Motion Statements

A unique blending range can thus be defined around each target position. This range can be optimally adjusted based on the respective requirements.

If the tool working point enters this blending range during the travel motion, the robot travels on a path generated by the control ensuring that the individual axis velocities do not change in jumps. After leaving the blending range, the robot acts as in case of travel motions without programmed blending.

Each motion block contains information on velocity and distance to be traveled. If a different velocity is programmed in successive blocks, the velocity change is carried out at the block transition. If the defined blending range is relatively small and the difference of the programmed block velocity is relatively high, the difference velocity of the successive motions has to be balanced in a short period. The deviation from the originally programmed path, that is without blending, is limited to a small range. If large blending ranges are selected, the velocity transition is smoother, but the deviation occurs over a longer distance from the path.

The transitions between the programmed path and the blending path are tangential.

The following example tries to clarify the continuous velocity change in one block transition. It is based on a Cartesian system with two axes, one in x-direction and one in y-direction. The slope mode is `PROGR_SLOPE`. The blending range in the x- and y-direction is determined using the parameter `A-0-0230` of the respective axis. If it is traveled without blending, there is a velocity jump from $+V$ to $-V$ at the block transition for the y-axis.

If blending is enabled, the velocity is changed continuously from $+V$ to $-V$ within the blending range. Smoother block transitions are generated.

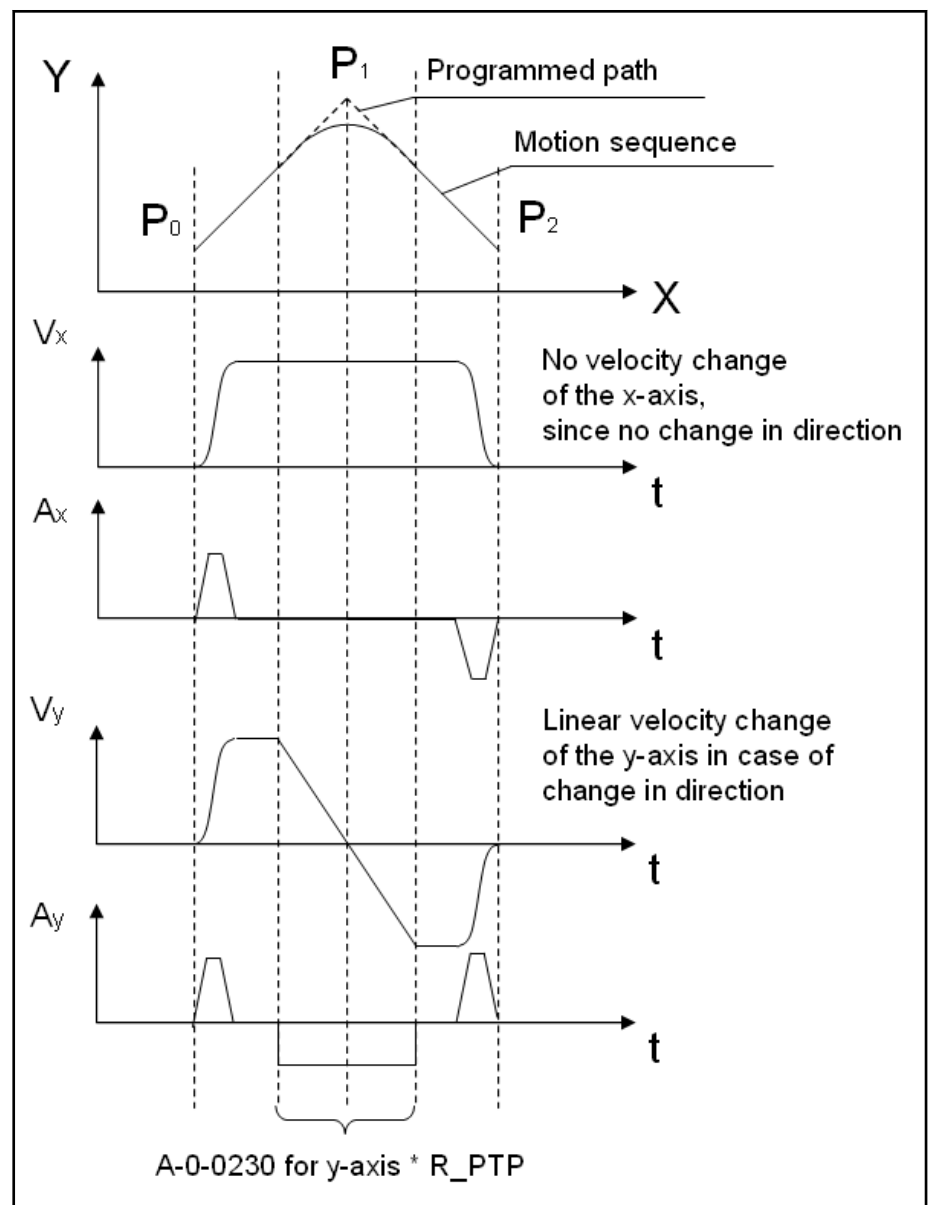


Fig. 12-36: Velocity characteristics of the x-axis and y-axis during PTP blending in *PROGR_SLOPE*

Linear Blending for *PROGR_SLOPE* and *CONT_SLOPE*

In path blending, a spatial sphere is defined around each point to be blended. The radius of this sphere is defined in the RCL program with the default variable *R*. Thus, for path blending, each programmed spatial point can be provided with its individual blending range. During entry and exit from this spatial sphere, the original path is exited for a constant path velocity. The motion within the sphere is carried out on a path generated by the control. Outside this sphere, the robot moves on the programmed path.

RCL Motion Statements

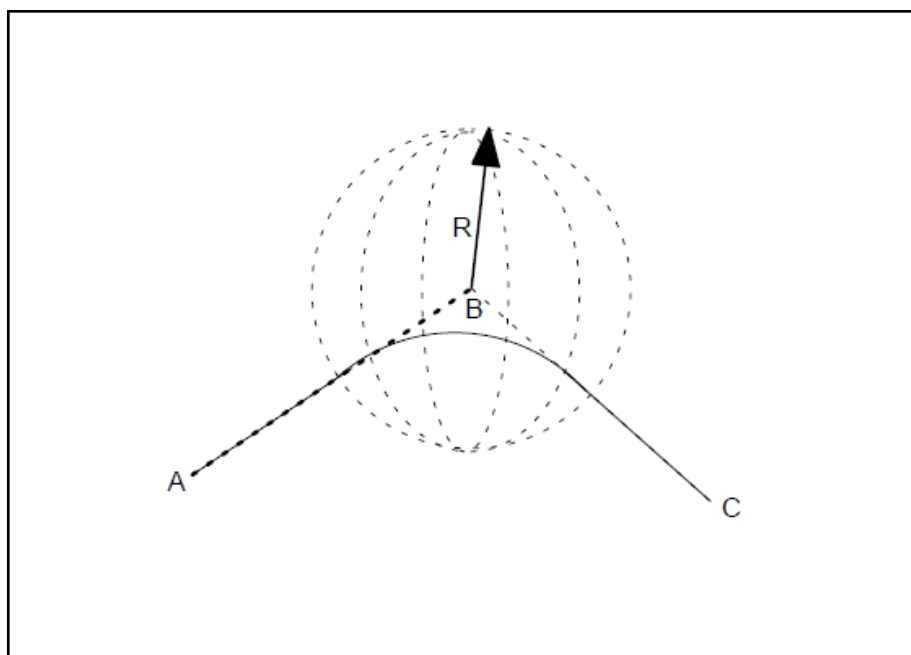


Fig. 12-37: Blending sphere for linear interpolation in *PROGR_SLOPE*

Blending radius: PROG_SLOPE

If the distances between the points to be blended do not meet the programmed blending criterion, the control automatically reduces to the maximum possible blending range. This is shown in

[fig. 12-38 "Overlapping blending ranges " on page 139](#)

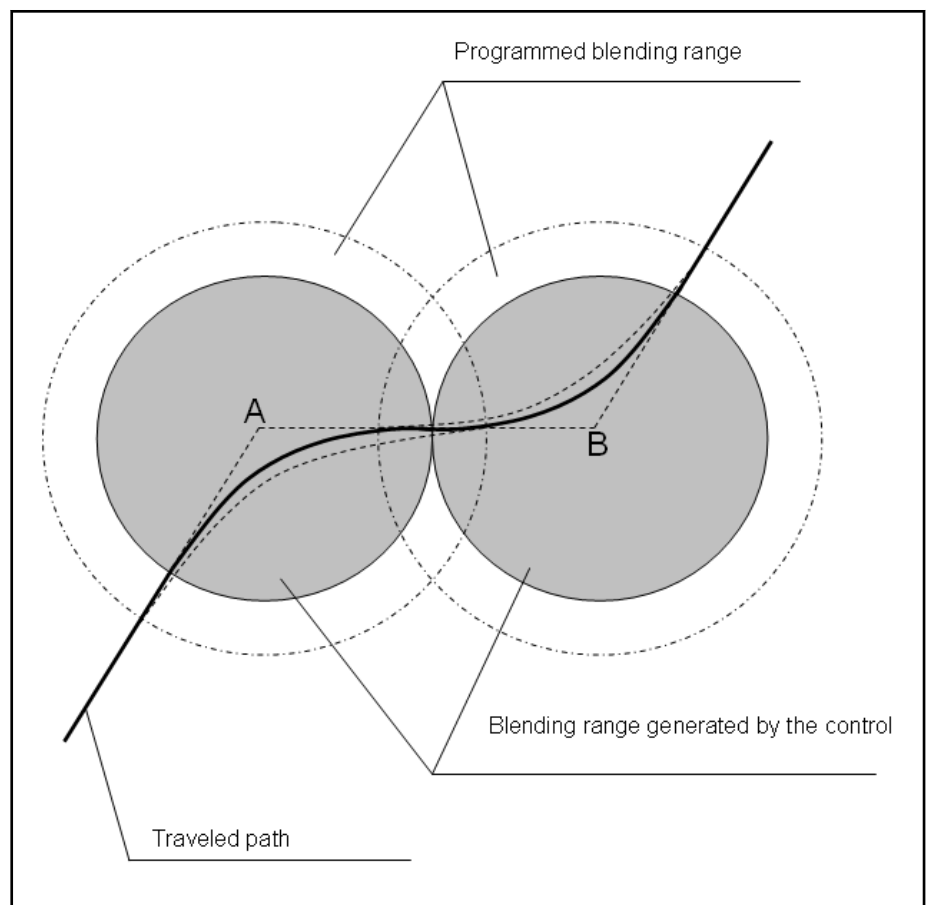


Fig. 12-38: Overlapping blending ranges

Explanation of the figure:

Overlapping blending ranges are automatically reduced by the control in such a way that the maximum possible blending range results for this case. It is shown by the continuous line. For more details on the effective radius while blending, please refer to [Blending mode, page 144](#).

Blending radius: CONT_SLOPE

The maximum radius possible for blending is half the distance between two points. If the distances between the points to be blended do not meet the programmed blending criterion, the control automatically reduces to the maximum possible blending range. In contrast to the slope type PROG_SLOPE, asymmetric blending radii are possible. The following applies: The maximum blending radius between several blocks to be blended is the respective half of the distance between two points. For start and end blocks of a motion, the complete distance can be used for blending.

RCL Motion Statements

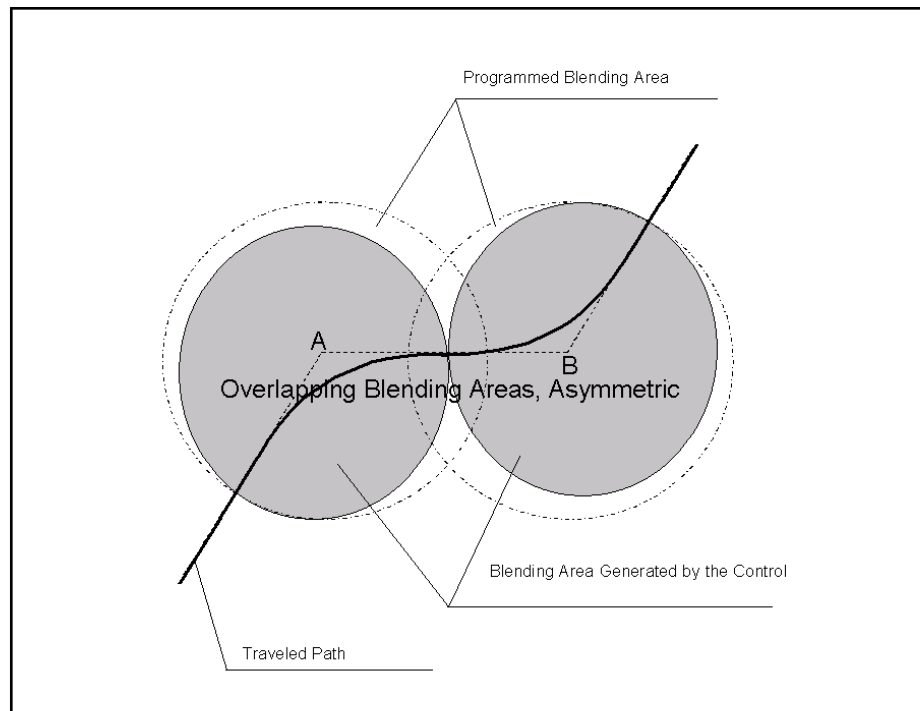


Fig.12-39: Overlapping blending ranges

Explanation of the figure:

Overlapping blending ranges are automatically reduced by the control in such a way that the maximum possible blending range results for this case. It is possible for CONT_SLOPE that asymmetric blending areas result. The continuous line represents the actual path. For more details on the effective radius while blending, please refer to [Blending mode, page 144](#).



The belt synchronization with the block transition modes PROGR_SLOPE and CONT_SLOPE is currently not supported.

Blending when Changing the Interpolation Method

Blending in case of PROGR_SLOPE active, is also effective when changing the interpolation methods. PTP can be used to move to a point and LINEAR can be used to move away from a point for example. This is also possible vice versa. Before entering and after leaving the blending envelope, the robot does not deviate from the originally programmed path. In this case, it is irrelevant whether it is changed from PTP to LINEAR or vice versa.



For the operation mode CONT_SLOPE, blending is only supported for the motion combinations Linear - Linear and PTP - PTP. When changing the interpolation method, no blending takes place.

Spatial Blending for BLOCK_SLOPE

PTP Blending for BLOCK_SLOPE

For the slope type BLOCK_SLOPE, the next motion block is started too early depending on A-0-0230, blending distance and R_PTP factor programmed in the RCL program. Distances is specified in degrees or mm for each axis with regard to its end position. From these distances, a common point in time is determined to start the following motion block. Both block are now parallelly

RCL Motion Statements

executed. The path velocity results from the geometric addition of single axis velocities. This ensures that after the robot's tool working point exited the blending range, it reaches again the path originally taken without blending. To prevent the simultaneous activation of more than two motion blocks, the control automatically limits the earliest starting point to the center of the previous motion. A started motion block is ended until the center of the subsequent block. Depending on the effective blending mode, the behavior described can deviate with regard to the mentioned distances. For more details on the effective radius while blending, please refer to [Blending mode, page 144](#),.

The following example tries to clarify the travel behavior.

It is based on a Cartesian system with 3 positioning axes, in which the z-axis will first travel upwards from the workspace. Then, the x-axis and the y-axis travel to emerge again with the z-axis in the workspace.

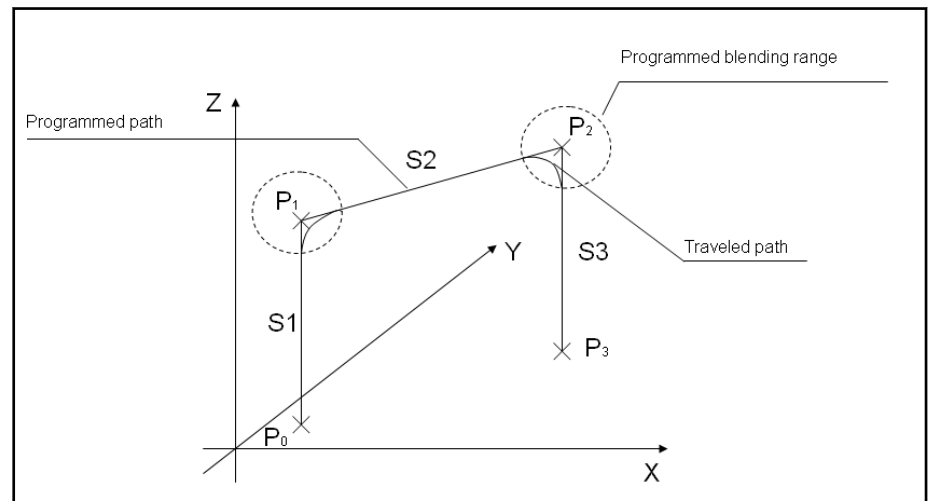


Fig. 12-40: Path characteristics of Pick&Place application

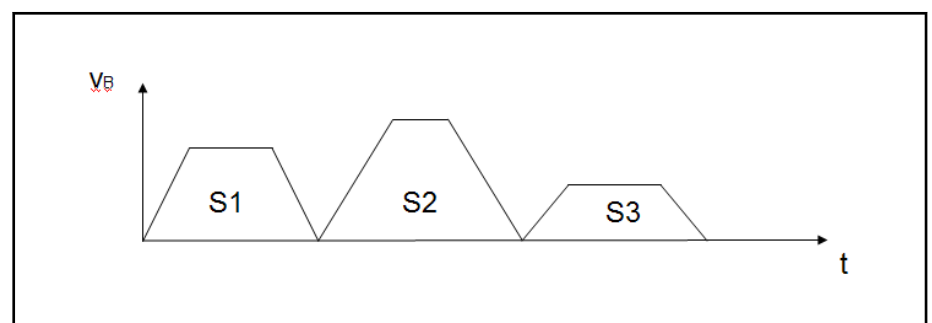


Fig. 12-41: Path velocity characteristics of Pick&Place application without blending

RCL Motion Statements

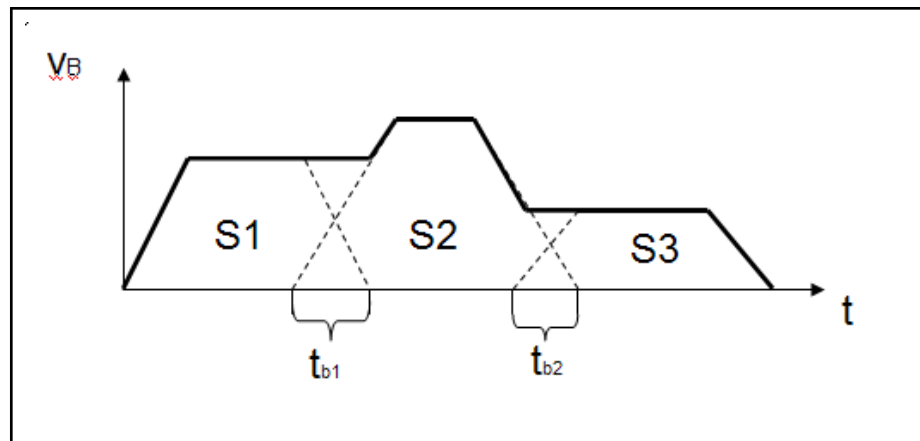


Fig.12-42: Path velocity characteristics of Pick&Place application with blending

Time saved: $t_{b1} + t_{b2}$.

Linear Blending for BLOCK_SLOPE

Path blending with BLOCK_SLOPE has, with regard to the path, principally the same effect as PTP blending with BLOCK_SLOPE.

Blending when Changing the Interpolation Method

In case of an active BLOCK_SLOPE, it is blended from LINEAR to PTP and vice versa when changing the interpolation method.

Late Blending

The mechanism of the "Late blending" refers to the spatial blending in BLOCK_SLOPE, CONT_SLOPE as well as in PROGR_SLOPE.

In practice, the target point of the travel motion is often posted at a later point in time. To optimize the motion sequence from a temporal point of view, the robot can already be commanded to an intermediate point before this point in time. If the target point becomes known on the way to the intermediate point, late blending starts. If necessary, the programmed radius is approximately reduced to the distance between the current position and the intermediate point.

RCL Motion Statements

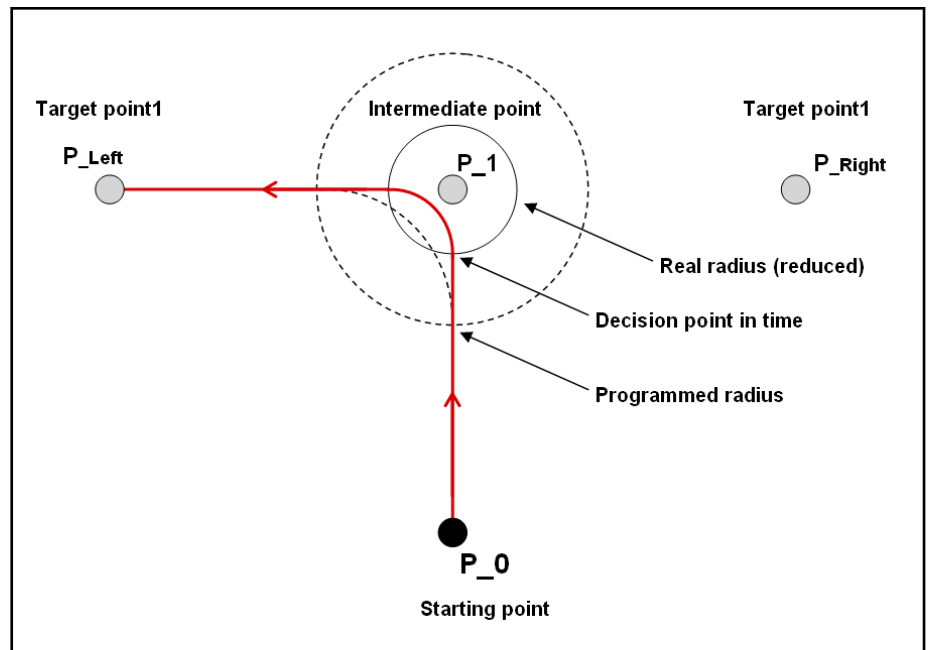


Fig.12-43: Late blending for PROGR_SLOPE

The example above shows a motion sequence that will be completely defined at a later point in time using the "Late blending". It is first traveled to the starting point P_0. The motion is started when traveling to the intermediate point P_1. While traveling, it is decided to approach the point "P_Left" at a later point in time. Since the motion is already in a progressed state and the initially programmed radius range has already been exceeded, the radius is reduced automatically.

Program:

```

PROGRAM LBlend                                     ;(1)
INPUT BINARY : 1001 = bAsyncGo                    ;(2) "Time of commanding" signal
INPUT BINARY : 1002 = bAsyncIn                    ;(3) Decision whether to turn left or right
POINT : P_0, P_1, P_Left, P_Right                 ;(4) Definition of sample points
BEGIN                                              ;(5)
;;INT=LINEAR                                       ;(6) Default MOVE is LINEAR

P_0 = (0,0)                                       ;(7) Initial Point
P_1 = (0,100)                                    ;(8) Intermediate Point
P_Left = (-100,100)                               ;(9) Turn Left End Point
P_Right = (100,100)                               ;(10) Turn Right End Point

PROGR_SLOPE                                       ;(11) Activate Program Slope

A = 10000                                         ;(12)
V = 2000                                          ;(13)
R = 20                                            ;(14)

MOVE TO P_0                                       ;(15) Move to the Initial Point
MOVE VIA P_1                                     ;(16) Move to the first intermediate point

; Decide whether to turn left or right
; By the time when this decision is done, switch on the bAsyncGo input.
; The program shall wait here for this plc signal.
WAIT UNTIL bAsyncGo                               ;(17)

; The decision was made. Check the result.

; 1: Left, 0: Right
IF bAsyncIn = 1
THEN
MOVE TO P_Left
ELSE
MOVE TO P_Right

```

RCL Motion Statements

PROGRAM_END

Blending Mode

General Information



This blending behavior applies to BLOCK_SLOPE and to PROGR_SLOPE. Blending with point-to-point movements (PTP) and blending with linear interpolation (LINEAR) are equally implemented. At CONT_SLOPE, this behavior only applies for the blending at linear interpolation (LINEAR).

For many applications, it is recommended not to limit the blending radius of the first and last motion by half the distance of the shortest distance (upper blue curve).

Optionally, the entire distance of the shortest distance can be used as blending radius from version MLC-FW 10V06 onwards if this is geometrically possible (lower curve shown in green).

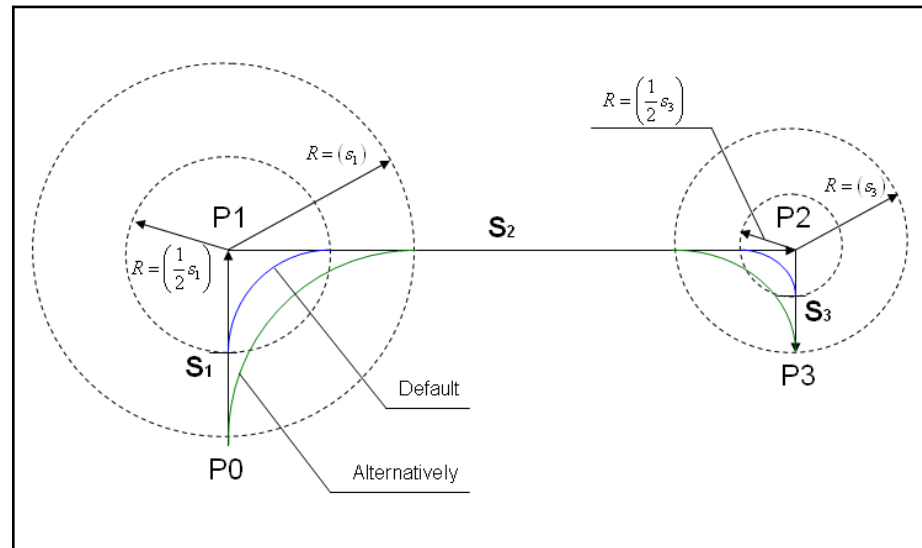


Fig. 12-44: Adapting the effective radius via blending mode



An overlapping of the blending radii is not allowed in both the cases and is impeded by the control. In this case, the effective radius

$$R_w = \min \left(R, \frac{1}{2} s_1, \frac{1}{2} s_2 \right)$$

us is limited to



The term **"half distance"** describes a little bit less than half the distance (1/2,3) due to technical reasons. This is also the case for the **"complete distance"** that is also a bit less (1/1,1).

Programming this function can take place in the PLC as well as in the RCL program. However, it is strongly recommended not to use both access options at the same time.

See also [RobotControl Functionality Programmed in the PLC, page 225](#).

RCL Motion Statements

PLC Programming "Blending Mode" see ML_Robot, Library Description:

- PLC function ML_KinGetDistanceBlendOption,
- PLC function ML_KinSetDistanceBlendOption.

RCL programming "Blending mode"

- [SPC_FCT: 59, Query blending mode, page 106,](#)
- [SPC_FCT: 60, Set blending mode, page 105](#)

13 Setup Mode

13.1 Setup Mode, General Information

An interface to the PLC to set up kinematics is provided.

Data is exchanged via a shared memory between PLC and Motion. The data are stored in the KinData array according to the control type for the maximum number of kinematics (axis groups).

The structure ML_KINEMATICSDATA_SM, ML_Robot.library, is available for each individual kinematics.

For the required elements to jog along the individual axes in this structure, refer to "Command interface to jog"

stJogCmd_q :ML_KIN_JOG; // x-x-xxxx -RW- Command interface for jog mode

See also [RobotControl Functionality Programmed in the PLC](#), page 225.

Setup Mode

13.2 Jogging in Different Coordinate Systems

13.2.1 Parameters to be Set to Jog in Axis Coordinates

Kinematic axes can be moved manually in the axis coordinates. To jog a single axis in the setup mode, the parameters have to be set according to the jog mode.

Incremental jogging, distance [mm]

Short:	A-0-0080, Jogging, short distance	
	Velocity [mm/s] or [inch/s]	
	Fast:	A-0-0083, Jogging, high velocity
	Acceleration [mm/s²] or [inch/s²]	
	Acceleration:	A-0-0084, Jogging, acceleration
	Deceleration:	A-0-0085, Jogging, deceleration
	Jerk [mm/s³] or [inch/s³]	
	Acceleration jerk:	A-0-0086, Jogging, acceleration jerk
	Deceleration jerk:	A-0-0087, Jogging, deceleration jerk
Long:	A-0-0081, Jogging, long distance	
	Velocity [mm/s] or [inch/s]	
	Fast:	A-0-0083, Jogging, high velocity
	Acceleration [mm/s²] or [inch/s²]	
	Acceleration:	A-0-0084, Jogging, acceleration
	Deceleration:	A-0-0085, Jogging, deceleration
	Jerk [mm/s³] or [inch/s³]	
	Acceleration jerk:	A-0-0086, Jogging, acceleration jerk
	Deceleration jerk:	A-0-0087, Jogging, deceleration jerk

Continuous jogging, Velocity = [mm/s]

Slow:	A-0-0082, Jogging, low velocity	
	Acceleration [mm/s²] or [inch/s²]	
	Acceleration:	A-0-0084, Jogging, acceleration
	Deceleration:	A-0-0085, Jogging, deceleration
	Jerk [mm/s³] or [inch/s³]	
	Acceleration jerk:	A-0-0086, Jogging, acceleration jerk
Fast:	A-0-0083, Jogging, high velocity	
	Acceleration [mm/s²] or [inch/s²]	
	Acceleration:	A-0-0084, Jogging, acceleration
	Deceleration:	A-0-0085, Jogging, deceleration
	Jerk [mm/s³] or [inch/s³]	
	Acceleration jerk:	A-0-0086, Jogging, acceleration jerk
	Deceleration jerk:	A-0-0087, Jogging, deceleration jerk

13.2.2 Parameter to Be Set to Jog in World Coordinates

Kinematic axes can be moved manually in the world coordinates. To jog along a single coordinate in the setup mode, the parameters have to be set according to the jog mode.

Incremental jogging, distance [mm] or [inch]

- Short: K-0-0080, Jogging, distance short in WC system
Velocity [mm/s] or [inch/s]
 Fast: K-0-0083, Jogging, velocity high in WC system
- Acceleration [mm/s²] or [inch/s²]**
 Acceleration: K-0-0084, Jogging, acceleration in WC system
 Deceleration: K-0-0085, Jogging, deceleration in WC system
- Jerk [mm/s³] or [inch/s³]**
 Acceleration jerk: K-0-0086, Jogging, acceleration jerk in WC system
 Deceleration jerk: K-0-0087, Jogging, deceleration jerk in WC system
- Long: K-0-0081, Jogging, distance long in WC system
Velocity [mm/s] or [inch/s]
 Fast: K-0-0083, Jogging, velocity high in WC system
- Acceleration [mm/s²] or [inch/s²]**
 Acceleration: K-0-0084, Jogging, acceleration in WC system
 Deceleration: K-0-0085, Jogging, deceleration in WC system
- Jerk [mm/s³] or [inch/s³]**
 Acceleration jerk: K-0-0086, Jogging, acceleration jerk in WC system
 Deceleration jerk: K-0-0087, Jogging, deceleration jerk in WC system

Continuous jogging, velocity [mm/s] or [inch/s]

- Slow: K-0-0082, Jogging, velocity low in WC system
Acceleration [mm/s²] or [inch/s²]
 Acceleration: K-0-0084, Jogging, acceleration in WC system
 Deceleration: K-0-0085, Jogging, deceleration in WC system
- Jerk [mm/s³] or [inch/s³]**
 Acceleration jerk: K-0-0086, Jogging, acceleration jerk in WC system
 Deceleration jerk: K-0-0087, Jogging, deceleration jerk in WC system
- Fast: K-0-0083, Jogging, velocity high in WC system
Acceleration [mm/s²] or [inch/s²]
 Acceleration: K-0-0084, Jogging, acceleration in WC system
 Deceleration: K-0-0085, Jogging, deceleration in WC system
- Jerk [mm/s³] or [inch/s³]**
 Acceleration jerk: K-0-0086, Jogging, acceleration jerk in WC system
 Deceleration jerk: K-0-0087, Jogging, deceleration jerk in WC system

Setup Mode

13.2.3 Jogging in World Coordinates

Kinematic axes can be moved manually in the world coordinates. To jog along a single coordinate in the setup mode, the parameters have to be set according to the jog mode.

Incremental jogging, distance [mm] or [inch]

Short:	K-0-0080, Jogging, distance short in WC system
	Velocity [mm/s] or [inch/s]
	Fast: K-0-0083, Jogging, velocity high in WC system
	Acceleration [mm/s²] or [inch/s²]
	Acceleration: K-0-0084, Jogging, acceleration in WC system
	Deceleration: K-0-0085, Jogging, deceleration in WC system
	Jerk [mm/s³] or [inch/s³]
	Acceleration jerk: K-0-0086, Jogging, acceleration jerk in WC system
	Deceleration jerk: K-0-0087, Jogging, deceleration jerk in WC system
Long:	K-0-0081, Jogging, distance long in WC system
	Velocity [mm/s] or [inch/s]
	Fast: K-0-0083, Jogging, velocity high in WC system
	Acceleration [mm/s²] or [inch/s²]
	Acceleration: K-0-0084, Jogging, acceleration in WC system
	Deceleration: K-0-0085, Jogging, deceleration in WC system
	Jerk [mm/s³] or [inch/s³]
	Acceleration jerk: K-0-0086, Jogging, acceleration jerk in WC system
	Deceleration jerk: K-0-0087, Jogging, deceleration jerk in WC system

Continuous jogging, velocity [mm/s] or [inch/s]

Slow:	K-0-0082, Jogging, velocity low in WC system
	Acceleration [mm/s²] or [inch/s²]
	Acceleration: K-0-0084, Jogging, acceleration in WC system
	Deceleration: K-0-0085, Jogging, deceleration in WC system
	Jerk [mm/s³] or [inch/s³]
	Acceleration jerk: K-0-0086, Jogging, acceleration jerk in WC system
	Deceleration jerk: K-0-0087, Jogging, deceleration jerk in WC system
Fast:	K-0-0083, Jogging, velocity high in WC system
	Acceleration [mm/s²] or [inch/s²]
	Acceleration: K-0-0084, Jogging, acceleration in WC system
	Deceleration: K-0-0085, Jogging, deceleration in WC system
	Jerk [mm/s³] or [inch/s³]
	Acceleration jerk: K-0-0086, Jogging, acceleration jerk in WC system
	Deceleration jerk: K-0-0087, Jogging, deceleration jerk in WC system

14 RCL Parameters

14.1 Axis-specific Parameters Used by the Kinematics of an MLC

14.1.1 General Information

Limit values and axis-specific data are stored in the A-parameters of the respective axis.

For details, refer to "IndraMotion MLC Parameter Description".

14.1.2 A-0-0028, Travel range limit switch

Function This parameter defines the signal behavior of the travel range limit switch inputs and the behavior of the drive with regard to exceeded travel ranges.

Structure

Bit	Function
0	Signal behavior travel range limit switch inputs (real axes, only) 0: not inverted, 24 V $\Rightarrow$ travel range is exceeded, 1: inverted, 0 V $\Rightarrow$ travel range is exceeded.
1	Activation of travel range limit switch (real axes, only) 0: no, 1: yes.
2	Handling when travel range exceeded 0: as error (acc. to error reaction), 1: as fatal warning (shutdown).
15..3	reserved

14.1.3 A-0-0029, Position polarities

Function For real axes applies:

The first four bits of this parameter can invert the polarities of the specified position data. The polarities are switched outside the controlled system. That is at the input and output of the controlled system. The parameter acts on the parameter S-0-0055 of the drive device.



The position polarity has to be specified at first commissioning of an axis before creating the machine zero point reference for measuring systems, since other actual position values result when changing the polarity.

For rotary (rotating) motors applies:

Rotation of the motor output shaft in clockwise direction at positive position command value difference and positive polarity.

For linear motors apply:

Motion of the primary component towards the cable connection side at positive position command difference and positive polarity.

See also: A-0-0048 and A-0-0049

Function The software position limit value, "A-0-0030, Position limit value, positive" or "A-0-0031, Position limit value, negative" are activated or deactivated using bit 4 of this parameter.

RCL Parameters

Structure

Bit	Name/function
0	Position command value 0: Positive polarity 1: Negative polarity
1	Position command value, additive 0: Positive polarity 1: Negative polarity
2	Actual position value 1 0: Positive polarity 1: Negative polarity
3	Actual position value 2 0: Positive polarity 1: Negative polarity
4	Position limit value 0: Not active, 1: Active.
15..5	Reserved

14.1.4 A-0-0030, Position limit value, positive

Function An axis's permitted travel range can be defined by the position limit values. The maximum allowed positive actual position value (positive traveling direction) is entered in this parameter.

The positive limit value is only active if all position data refers to the axis zero point, i.e. the axis is referenced "A-0-0021, Axis status, bit 10".

Use If the "A-0-2200, Command position" of the axis is set outside the positive position limit value, the error bit in "A-0-0021, Axis status, bit 9" is TRUE and generates the error "F2112053, Command target position outside of travel range".

If the "A-0-0100, Actual position value" exceeds the "A-0-0030, Position limit value, positive" or the "A-0-0031, Position limit value, negative", the axis sets the error bit in the "A-0-0021, Axis status, bit 9" and generates the error "F2116029, Positive position limit value exceeded" or "F2116030, Negative position limit value exceeded".

See also Functional Description: "Position Limit Values".

14.1.5 A-0-0031, Position limit value, negative

Function An axis's permitted travel range can be defined by the position limit values. The maximum allowed negative actual position value (negative travel direction) is entered in this parameter.

The negative limit value is only active if all position data refers to the axis zero point, i.e. the axis is referenced "A-0-0021, Axis status, bit 10".

Use If the "A-0-2200, Command position" of the axis is set outside the negative position limit value, the error bit in "A-0-0021, Axis status, bit 9" is TRUE and generates the error "F2112053, Command target position outside of travel range".

If the "A-0-0100, Position value" exceeds the "A-0-0030, Position limit value, positive" or the "A-0-0031, Position limit value, negative", the axis sets the

error bit in the "A-0-0021, Axis status, bit 9" and generates the error "F2116029, Positive position limit value exceeded" or "F2116030, Negative position limit value exceeded".

See also Functional Description: "Position Limit Values".

14.1.6 A-0-0032, Velocity limit value, positive

Function The parameter describes the maximum admissible velocity in the positive direction.

Use If the "A-0-0102, Actual velocity value" exceeds the absolute limit value "A-0-0032, Velocity limit value, positive" or "A-0-0033, Velocity limit value, negative", the axis sets the warning bit "A-0-0021, Axis status, bit 8" and generates the warning "E0112063, Velocity greater than limit A-0-0032 / A-0-0033".

The entered value is used as the maximum value for velocity variables in the positive direction.

See also Functional Description "Velocity Limitation".

14.1.7 A-0-0033, Velocity limit value, negative

Function The parameter describes the maximum admissible velocity in the negative direction.

Use If the "A-0-0102, Actual velocity value" exceeds the absolute limit value "A-0-0032, Velocity limit value, positive" or "A-0-0033, Velocity limit value, negative", the axis sets the warning bit "A-0-0021, Axis status, bit 8" and generates the warning "E0112063, Velocity greater than limit A-0-0032 / A-0-0033".

The entered value is used as the maximum value for velocity variables in the negative direction.

See also Functional Description "Velocity Limitation".

14.1.8 A-0-0034, Acceleration limit value, bipolar

Function The parameter describes the maximum admissible symmetric acceleration in both directions (acceleration and deceleration).

Use If the parameter "A-0-0103, Actual acceleration value" exceeds the parameter "A-0-0034, Acceleration limit value, bipolar", the axis sets the warning bit "A-0-0021, Axis status, bit 8" and generates the warning "E0112039, Maximum acceleration exceeded".

14.1.9 A-0-0036, Jerk limit value, bipolar

Function The parameter describes the maximum admissible acceleration change for each time interval (jerk) symmetrical in both directions (acceleration and deceleration).

This limit value counteracts the commanded jerk in the parameter "A-0-0216, Jerk".



The parameter limits the acceleration change for each time interval during "Drive hold" and for commands automatically generating position command values for themselves.

A value of "0" switches off the jerk limit value.

It is influenced by the parameter "A-0-0053, Scaling type for acceleration data".

RCL Parameters

14.1.10 A-0-0038, Torque/force limit value, bipolar

Function This parameter can be used to limit the torque/force command value to maximum admissible values to protect mechanical components in the drive train.

The parameter acts in a bipolar way. That means that positive and negative command values are limited to the entered value. The limit affects the total of all torque command values. That is the total of the command value parts of the velocity controller output and the command value parts from the acceleration-dependent feedforwards ("S-0-0348, Acceleration feedforward gain"; velocity control loop: "P-0-1126, Acceleration feedforward").

The unit of "A-0-0038" depends on the scaling set.

See also Parameter Description "A-0-0050, Scaling type for torque/force data"



The effective limit value is displayed in "P-0-0049, Effective torque/force command value". This value can be limited via "A-0-0038" due to a smaller value in "P-0-0109, Peak torque/force limitation" or due to a current limitation by a respective amplifier or motor load.

If the drive should be provided with a lower torque during stationary mode (constant speed) than during accelerations, this can be achieved with the respective values of "S-0-0082, Torque/force limit value, positive" and "S-0-0083, Torque/force limit value, negative" in connection with the acceleration feedforward (S-0-0348 or P-0-1126).

The torque/force limit value, bipolar also affects the "C1300 command: Travel to fixed stop".

The value of "A-0-0038" is a criterion for acknowledging "C1300".

See also Functional Description "Torque/Force Control".

14.1.11 A-0-0048, Velocity polarities

Function In this parameter, the polarity of the velocity data is switched with regard to the application.

The polarity is not switched within, but outside (at input and output) a controlled system.

Structure

Bit	Name/function
0	Velocity command value 0: Positive polarity 1: Negative polarity
1	Position command value, additive 0: Positive polarity 1: Negative polarity
2	Actual velocity value 0: Positive polarity 1: Negative polarity
15..3	Reserved



Bit 1 and bit 2 are copies of bit 0.

Changes only become effective in bit 0. Different settings of the individual bits are not permitted!

For **rotary (rotating) motors** applies: Rotation of the motor output shaft in clockwise direction causes a positive actual velocity value (positive polarity).

For **linear motors** apply: Motion of the primary component towards the cable connection side causes a positive actual velocity value and (positive polarity).

14.1.12 A-0-0049, Torque-/force polarities

Function In this parameter, the polarity of the specified torque/force data is inverted with regard to the application.

The polarity is switched outside the controlled system. That means at the input and the output of the controlled system.

Structure

Bit	Name/function
0	Torque command value 0: Positive polarity 1: Negative polarity
1	Torque command value, additive 0: Positive polarity 1: Negative polarity
2	Actual torque value 0: Positive polarity 1: Negative polarity
15..3	Reserved



Bit 1 and bit 2 are copies of bit 0.

Changes only become effective in bit 0. Different settings of the individual bits are not permitted!

For **rotary (rotating) motors** applies: Rotation of the motor output shaft in clockwise direction at positive torque and positive polarity.

For **linear motors** apply: Motion of the primary component towards the cable connection side at positive force and positive polarity.

14.1.13 A-0-0080, Jogging, short distance

Function The small distance for incremental jogging of the axis is stored in this parameter.



This parameter can also be written in phase 4. The new value is taking effect at the next jogging command. Changing the parameter is not possible for the active jog command.

Structure

- Unit:
for linearly scaled axes in mm,
for rotary scaled axes in degrees.
- Value range: > 0.0 (floating point number)

RCL Parameters

- Default: 1.0 mm or degrees.

Use This parameter is used by the Robot-Control for incremental jogging of the axis.

If the axis is linked to another axis (e.g. combined lifting and revolving spindle) the other axis is controlled in such a way that its logic position remains unchanged.

With the jog bits (KinData[1].stJogCmd_q) the PLC program can activate incremental jogging with small distance:

E.G. incremental jogging with small distance of the 3. axis in positive direction:

Program:

```
KinData[1].stJogCmd_q.enJogCmdCoordSys := KIN_CORD_SYS_JOINT_COORD;
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STEP_SMALL;
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*no axis in neg.*)
KinData[1].stJogCmd_q.wJogCmdPositive := 4; (*3. bit for 3.axis of kinematics or group*)
```

- The maximum velocity is listed in A-0-0083 (always fast for incremental jogging).
- The maximum acceleration is listed in A-0-0084 and
- the maximum deceleration is listed in A-0-0085.
- The maximum jerk for acceleration is listed in A-0-0086 and
- the maximum jerk for deceleration is listed in A-0-0087.

Premature stopping of the jog movement is possible by setting enJogMode = KIN_JOG_STOP.

Program:

```
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STOP;
```

14.1.14 A-0-0081, Jogging, long distance

Function The large distance for incremental jogging of the axis is stored in this parameter.



This parameter can also be written in phase 4. The new value is taking effect at the next jogging command. Changing the parameter is not possible for the active jog command.

- Structure**
- Unit:
for linearly scaled axes in mm,
for rotary scaled axes in degrees.
 - Value range: > 0.0 (floating point number)
 - Default: 10.0 mm or degrees.

Use This parameter is used by the Robot-Control for incremental jogging of the axis.

If the axis is linked to another axis (e.g. combined lifting and revolving spindle) the other axis is controlled in such a way that its logic position remains unchanged.

With the jog bits (KinData[1].stJogCmd_q) the PLC program can activate incremental jogging with large distance:

E.G. incremental jogging with large distance of the 4. axis in negative direction:

Program:

```
KinData[1].stJogCmd_q.enJogCmdCoordSys := KIN_CORD_SYS_JOINT_COORD;
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STEP_LARGE;
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*4. bit for 4. axis of kinematics or group*)
KinData[1].stJogCmd_q.wJogCmdPositive := 4; (*no axis in pos.*)
```

- The maximum velocity is listed in A-0-0083 (always fast for incremental jogging).
- The maximum acceleration is listed in A-0-0084 and
- the maximum deceleration is listed in A-0-0085.
- The maximum jerk for acceleration is listed in A-0-0086 and
- the maximum jerk for deceleration is listed in A-0-0087.

Premature stopping of the jog movement is possible by setting enJogMode = KIN_JOG_STOP.

Program:

```
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STOP;
```

14.1.15 A-0-0082, Jogging, low velocity

Function The low velocity for continuous jogging of the axis is stored in this parameter.



This parameter can also be written in phase 4. The new value is taking effect at the next jogging command. Changing the parameter is not possible for the active jog command.

- Structure**
- Unit:
for linearly scaled axes in mm/s,
for rotary scaled axes in degrees/s.
 - Value range: > 0.0 (floating point number)
 - Default: 10.0 mm/s or degree/s.

Use This parameter is used by the Robot-Control for continuous jogging of the axis.

If the axis is linked to another axis (e.g. combined lifting and revolving spindle) the other axis is controlled in such a way that its logic position remains unchanged.

With the jog bits (KinData[1].stJogCmd_q) the PLC program can activate continuous jogging with low velocity:

E.G. continuous jogging with low velocity of the 3. axis in positive direction:

Program:

```
KinData[1].stJogCmd_q.enJogCmdCoordSys := KIN_CORD_SYS_JOINT_COORD;
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_SLOW;
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*no axis in neg.*)
KinData[1].stJogCmd_q.wJogCmdPositive := 4; (*3. bit for 3.axis of kinematics or group*)
```

- The maximum acceleration is listed in A-0-0084 and
- the maximum deceleration is listed in A-0-0085.
- The maximum jerk for acceleration is listed in A-0-0086 and
- the maximum jerk for deceleration is listed in A-0-0087.

The continuous jogging can be completed by 2 different methods:

RCL Parameters

Program:

```
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STOP;
```

and / or

Program:

```
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*no axis in neg.*)
KinData[1].stJogCmd_q.wJogCmdPositive := 0; (*no axis in pos.*)
```

14.1.16 A-0-0083, Jogging, high velocity

Function The high velocity for continuous jogging of the axis is stored in this parameter.



This parameter can also be written in phase 4. The new value is taking effect at the next jogging command. Changing the parameter is not possible for the active jog command.

- Structure**
- Unit:
for linearly scaled axes in mm/s,
for rotary scaled axes in degrees/s.
 - Value range: > 0.0 (floating point number)
 - Default: 25.0 mm/s or degree/s.

Use This parameter is used by the Robot-Control for continuous **and incremental** jogging of the axis.

If the axis is linked to another axis (e.g. combined lifting and revolving spindle) the other axis is controlled in such a way that its logic position remains unchanged.

With the jog bits (KinData[1].stJogCmd_q) the PLC program can activate continuous jogging with high velocity:

E.G. continuous jogging with fast velocity of the 4. axis in negative direction:

Program:

```
KinData[1].stJogCmd_q.enJogCmdCoordSys := KIN_CORD_SYS_JOINT_COORD;
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_FAST;
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*4. bit for 4. axis of kinematics or group*)
KinData[1].stJogCmd_q.wJogCmdPositive := 4; (*no axis in pos.*)
```

- The maximum acceleration is listed in A-0-0084 and
- the maximum deceleration is listed in A-0-0085.
- The maximum jerk for acceleration is listed in A-0-0086 and
- the maximum jerk for deceleration is listed in A-0-0087.

The continuous jogging can be completed by 2 different methods:

Program:

```
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STOP;
```

and / or

Program:

```
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*no axis in neg.*)
KinData[1].stJogCmd_q.wJogCmdPositive := 0; (*no axis in pos.*)
```

14.1.17 A-0-0084, Jogging, acceleration

Function The maximum acceleration for the jogging (incremental and continuous) of the axis is stored in this parameter.



The parameter can only be written with MLC 04VRS in phase 2.

- Structure**
- Unit:
for linearly scaled axes in mm/s^2 ,
for rotary scaled axes in degrees/s^2 .
 - Value range: > 0.0 (floating point number, must be $\leq \text{A-0-0034}$)
 - Default: 370 mm/s^2 or degree/s^2 .

Use This parameter is used by the Robot-Control for the jogging of the axis. The jog acceleration is valid for all jog movements (incremental and continuous). the jog deceleration is listed in A-0-0085.

For jogging in the world coordinate system (WC), the deceleration and acceleration values are determined in K-0-0084 and K-0-0085.

14.1.18 A-0-0085, Jogging, deceleration

Function The maximum deceleration for the jogging (incremental and continuous) of the axis is stored in this parameter.



The parameter can only be written with MLC 04VRS in phase 2.

- Structure**
- Unit:
for linearly scaled axes in mm/s^2 ,
for rotary scaled axes in degrees/s^2 .
 - Value range: > 0.0 (floating point number, must be $\leq \text{A-0-0034}$)
 - Default: 100 mm/s^2 or degree/s^2 .

Use This parameter is used by the Robot-Control for the jogging of the axis. The jog deceleration is valid for all jog movements (incremental and continuous).

The jog acceleration is listed in A-0-0084.

For jogging in the world coordinate system (WC), the deceleration and acceleration values are determined in K-0-0084 and K-0-0085.

14.1.19 A-0-0086, Jogging, acceleration jerk

Function The maximum acceleration jerk for the jogging (incremental and continuous) of the axis is stored in this parameter (indirect definition of the acceleration change time)..



The parameter can only be written with MLC 04VRS in phase 2.

- Structure**
- Unit:
for linearly scaled axes in mm/s^3 ,
for rotary scaled axes in degrees/s^3 .
 - Value range: ≥ 0.0 (floating point number, $\leq \text{A-0-00364}$)
 - Default: 0.0 (no jerk limitation).

RCL Parameters

Use This parameter is used by the Robot-Control for the jogging of the axis. The jog acceleration jerk is valid for all jog movements (incremental and continuous).

The acceleration change time $\Delta a = j * t$ or $t = \frac{\Delta a}{j}$ can be calculated from

$$t = \frac{A-0-0084}{A-0-0086}$$

The maximum jerk j for deceleration is listed in A-0-0087.

The maximum jerk values for jogging within world coordinates are applied by means of K-0-0086 and K-0-0087.

14.1.20 A-0-0087, Jogging, deceleration jerk

Function In this parameter, the maximum jerk in case of deceleration for jogging (incremental and continuous) of the axis is stored (indirect definition of the deceleration change time).



With MLC 04VRS, the parameter can only be written in phase 2.

- Structure**
- Unit:
for translationally weighted axes in mm/s³,
for rotationally weighted axes in degree/s³.
 - Value range: ≥0.0 (floating-point number, ≤ A-0-0036)
 - Default: 0.0 (no jerk limitation).

Use The robot control uses this parameter for the jogging of the axis. The jog deceleration jerk applies to all jog movements (incremental and continuous).

Based on $\Delta a = j * t$, you can calculate the deceleration change time $t = \frac{\Delta a}{j}$

$$t = \frac{A-0-0085}{A-0-0087}$$

and/or

The maximum jerk in the acceleration is specified in A-0-0086.

The maximum jerk values for jogging in world coordinates are applied with K-0-0086 and K-0-0087.

14.1.21 A-0-0224, Rapid stop deceleration

Function In this parameter, the maximum deceleration value at which the axis is set to standstill at

- Emergency stop with ramp
is set.

The parameter unit is specified by "A-0-0053, Scaling type for acceleration data".

See also Functional Description: "Error Reaction"

14.1.22 A-0-0225, Rapid stop jerk

Function In this parameter, the maximum jerk value at which the axis

- Emergency stop with ramp

is set.

The parameter unit is specified by "A-0-0053, Scaling type for acceleration data".

See also Functional Description: "Error Reaction".



This parameter is only in effect if "Interpolation in the control" was selected for the axis.

14.1.23 A-0-0230, Blending distance PTP

Function This parameter is relevant only for axes which are proceeded within a kinematics with the command " MOVE PTP ".

With these axis movements the strength of the passing can be programmed with movement block transitions by the factor R_PTP.

The value of this parameter indicates the distance to which the factor R_PTP refers.

If the value is 0, no passing of the axis can take place.

14.1.24 A-0-0240, Feedrate override

Function The factor used to modify the axis velocity is set in this parameter.

A value of 100% means no modification.

If the factor is 0%, the warning "E0112055, Feedrate override A-0-0240 = 0" is output.

14.1.25 A-0-0241, Feedrate override control word

Function This parameter influences the feedrate override without modifying the Feedrate override parameter A-0-0240.

Structure

Bit	Designation / Function
0	Reserved
1	Feedrate override = 0
15-2	Reserved

Use **Bit 1:**

When bit 1 is set, a feedrate override of 0 becomes effective without modifying the actual A-0-0240 parameter value.



Feedrate override = 0 means "Halt", but in this case without the warning "E0112055, Feedrate override A-0-0240 = 0"

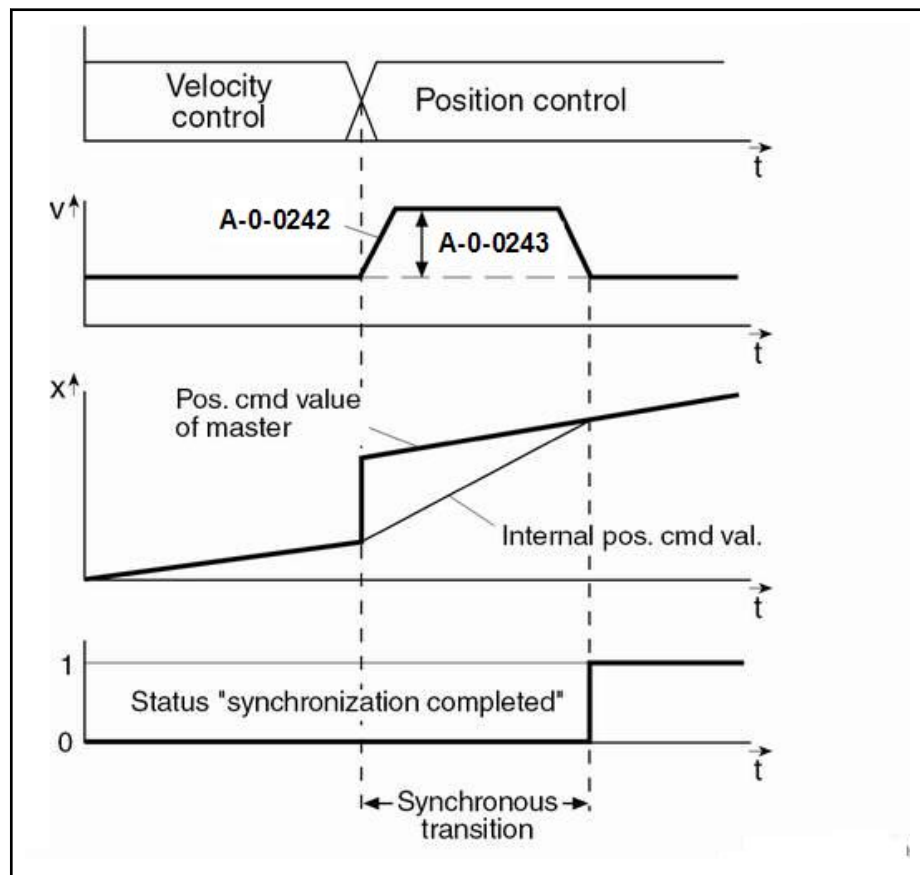
14.1.26 A-0-0242, Position control, synchronization acceleration

Function The parameter contains the acceleration value considered in a drive-controlled manner when switching the operation mode from velocity control to position control during synchronization.

The drive executes a synchronization when changing the operation mode to the cyclic position control. That means that it creates a smooth transition of the internal position command value from the actual position to the new command value characteristic specified by the control. This creation is based on the internal synchronization parameters (A-0-0242, A-0-0243) After the synchronization has been completed, the drive follows the position command values specified by the master in the Motion cycle

RCL Parameters

It is ensured drive-internally, that operation mode switching is also carried out in case of command value changes in jumps.



A-0-0242 Synchronization acceleration

A-0-0243 Synchronization velocity

Fig. 14-1: Time diagram "Drive-controlled operation mode switching"



The value of this parameter corresponds in its default state to the value of the parameter A-0-0034, Acceleration limit value, bipolar. Individual values can be configured if bit 4 is set in the parameter A-0-0007, Axis configuration.

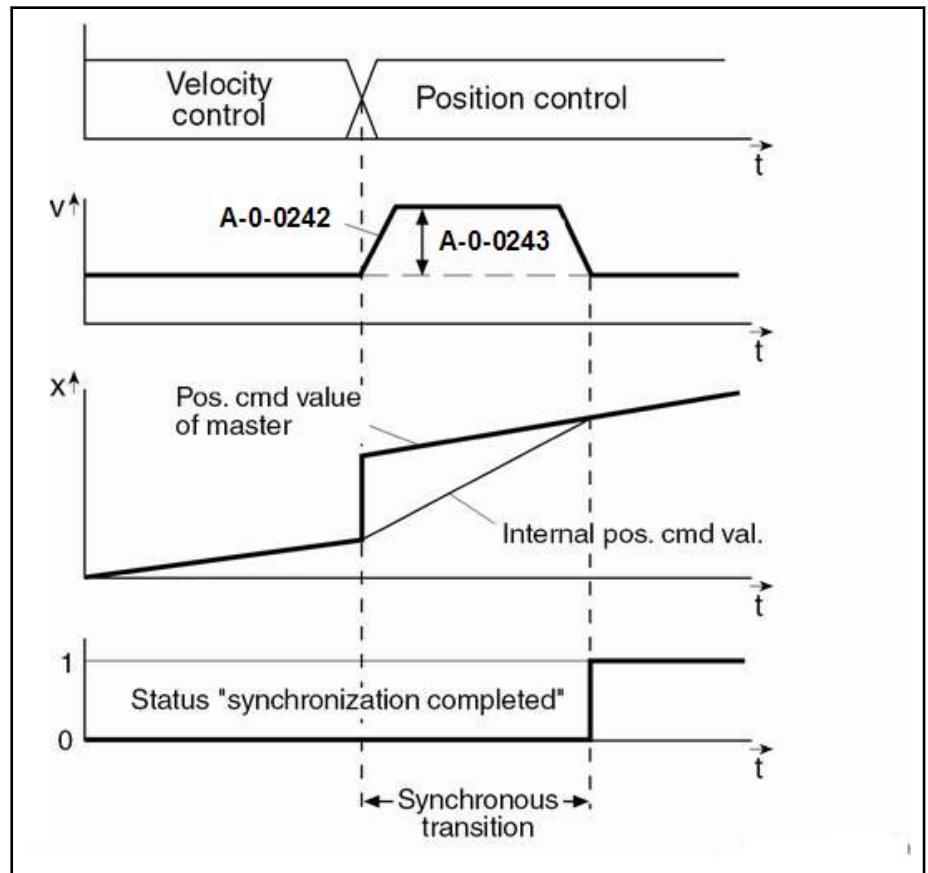
Use This value is synchronized to the drive when switching from P2 to BB.

14.1.27 A-0-0243, Position control, synchronization velocity

Function The parameter contains the velocity value considered in a drive-controlled manner when switching the operation mode from velocity control to position control during synchronization.

The drive executes a synchronization when changing the operation mode to the cyclic position control. That means that it creates a smooth transition of the internal position command value from the actual position to the new command value characteristic specified by the control. This creation is based on the internal synchronization parameters (A-0-0242, A-0-0243) After the synchronization has been completed, the drive follows the position command values specified by the master in the Motion cycle

It is ensured drive-internally, that operation mode switching is also carried out in case of command value changes in jumps.



A-0-0242 Synchronization acceleration

A-0-0243 Synchronization velocity

Fig. 14-2: Time diagram "Drive-controlled operation mode switching"



This parameter value corresponds in its default state to the maximum absolute value of the parameters A-0-0032/A-0-0033, Velocity limit value, positive/negative.

Individual values can be configured if bit 4 is set in the parameter A-0-0007, Axis configuration.

Use This value is synchronized to the drive when switching from P2 to BB.

14.2 Common C-Parameters of All Kinematics of an MLC

14.2.1 General Information

The following C-parameters are created to manage **all** kinematic data of one control.

For details, refer to "IndraMotion MLC Parameter Description".

14.2.2 C-0-0123, Maximum number of kinematics

Function This parameter indicates the maximum number of kinematics.

14.2.3 C-0-2480, Configured kinematic number

Function This parameter indicates the number of the currently configured kinematics.

RCL Parameters

14.2.4 C-0-2482, List of non-linked axes

- Function** An axis can be assigned only to one kinematics.
This lists parameter indicates the axes which are still not assigned to a kinematics.
If no kinematics is configured, this list corresponds to parameter "C-0-0483, List of the configured axes".
The allocation of the axes to the kinematics takes place via parameter "K-0-0008, Axis configuration list".

14.2.5 C-0-2483, List of configured kinematics

- Function** This parameter indicates the configured kinematics of the system.
Use Example:
3 kinematics (kinematics 1, 2 and 3) are configured in the system.
C-0-2483 indicates:

Element	Value	Comment
1	1	Kinematics 1 is configured
2	2	Kinematics 2 is configured
3	3	Kinematics 3 is configured

14.2.6 C-0-2484, Kinematic configuration list

- Function** This parameter defines the kinematics system. It configures the kinematic mechanisms. Only K-parameters of kinematic mechanisms configured in this parameter can be accessed.
Assignment of the axes to kinematics is carried out by means of parameter "K-0-0008, Axes configuration list" (Example K01: K-0-0008).

- Structure** Valid values of a kinematics configuration:

Value	Comment
0	No kinematic mechanism configured
1	Kinematic mechanism configured

- Use** The list must not contain any gaps with non-configured kinematic mechanisms. This is checked during switching after phase 4.
If this is not the case, parameter "C-0-2484, Kinematics configuration list" is entered into parameter "C-0-0111, List of all invalid C parameters" and an incorrect configuration is displayed (F0160024, Error at cross-check of C parameters).

Writing of parameter "C-0-2484, Kinematics configuration list" updates parameter "C-0-2480, Configured kinematics count" and parameter "C-0-2483, List of configured kinematics".

Example A:

A kinematic mechanism is to be configured.
C-0-2484 indicates:

RCL Parameters

Element	Value	Comment
1	1	Kinematic mechanism 1 is configured
2	0	No kinematic mechanism 2 used/required
3	0	No kinematic mechanism 3 used/required
4	0	No kinematic mechanism 4 used/required

Example B:

Three kinematic mechanisms are to be configured (with version 04V12).
C-0-2484 indicates:

Element	Value	Comment
1	1	Kinematic mechanism 1 is configured
2	1	Kinematic mechanism 2 is configured
3	1	Kinematic mechanism 3 is configured
4	0	No kinematic mechanism 4 used/required

14.3 Parameter of a Kinematics in the MLC

14.3.1 General Information

The following K-parameters are created to manage the data of **one** kinematics in the control. Each parameter exists once in each kinematics.

For details, refer to "IndraMotion MLC Parameter Description".

14.3.2 K-0-0001, Logic kinematic number

Function This parameter shows the actual logical kinematics number.

14.3.3 K-0-0002, Kinematic name

Function The name of the kinematics (e.g. Robi_1) is written in this parameter.

A global variable of type AXIS_GROUP_REF is automatically created with this name.



The following restrictions apply:

- maximum of 12 characters,
- A...Z, a...z, 0..9, '_'
- first character must be a letter,
- "Robi_1" is identical to "ROBI_1".

14.3.4 K-0-0003, Maximum path speed

Function Maximum programmable motion velocity in world coordinates (WC) at the Tool Center Point (TCP).

Depending on the parameter K-0-0054, the entry is made in mm/s or inch/s.

Determining a reasonable maximum path velocity depends on the resolution of the measuring system, the minimum programmed point distance, the robot mechanics and the sample time of the position control loop (clock starting time).

RCL Parameters

14.3.5 K-0-0004, reserved

Function This parameter is reserved for internal applications.

14.3.6 K-0-0005, Software limit switches, positive, WC

Function This parameter determines the values of the positive software limit switch in world coordinates.

Depending on the parameter K-0-0054, the entry is made in mm/s, inch/s or degrees.

Only in jog mode.

Due to the kinematics, an internal travel range limit can be reached when traveling in world coordinates in automatic or manual mode.

The motion is immediately stopped and "travel range limit reached" is displayed in the diagnostic error mode.

This parameter can also be written in phase 4. The new value is only applied after the next phase switching of the control from P2 to BB.

14.3.7 K-0-0006, Software limit switches, negative, WC

Function This parameter determines the values of the negative software limit switch in world coordinates.

Depending on the parameter K-0-0054, the entry is made in mm/s, inch/s or degrees.

Only in jog mode.

Due to the kinematics, an internal travel range limit can be reached when traveling in world coordinates in automatic or manual mode.

The motion is immediately stopped and "travel range limit reached" is displayed in the diagnostic error mode.

This parameter can also be written in phase 4. The new value is only applied after the next phase switching of the control from P2 to BB.

14.3.8 K-0-0007, Axis number of kinematics

Function This parameter contains the actual number of axes of the kinematics.

The value results from "K-0-0008, Axes configuration list".

It contains the sum of all there configured axes and belts.

Parameter "K-0-0070, Belt number of kinematics" contains only the sum of the belts.

14.3.9 K-0-0008, Axis configuration list

Function This parameter configures the axes for the kinematics.

Structure The parameter is a list parameter with the following triple structure:

- Element 1: Axis configuration of kinematic axis 1
- Element 2: Control number
- Element 3: Assigned axis for kinematic axis 1
- Element 4: Axis configuration of kinematic axis 2
- Element 5: Control number
- Element 6: Assigned axis for kinematic axis 2
- ...

The element number must be even.

RCL Parameters

Value	Comment
0	No axis configured
8	Positioning axis configured
16	Orientation axis configured
32	Belt axis configured

Fig. 14-3: Valid values of an axis configuration

Value	Comment
0	Individual control
1-64	Control number in C2C link

Fig. 14-4: Valid values for the control number

Use Structure of the triple list

The list may not contain any gaps.

One positioning axis has to be at least assigned to a kinematics.

Orientation axes must be located behind the positioning axes.

Belt axes must be located behind positioning axes and orientation axes.

Based on the set parameter "K-0-0031 Transformation scheme," a different minimum number of positioning and orientation axes are required (see desired transformation description).

A maximum of four belt axes may be defined.

The triple list may configure a maximum total of 16 axes.

Assignment assistance

Each axis may be assigned to at most one kinematics as positioning or orientation axis.

Parameter "C-0-2482, List of non-linked axes ", lists the axes not yet assigned to a kinematics.

Kinematic axes configured as belt axes do not affect the parameter "C-0-2482, List of non-linked axes."

Belt axes can be selected from "C-0-0483, List of configured axes" and can be assigned to several kinematics.

Influence on other parameters

The number of triple entries in the list determines the parameter "K-0-0007, Axis number of kinematics."

The number of configured belt axes in the list determines the parameter "K-0-0070, Belt number of kinematics."



When switching to phrase 4, it is checked whether at least one axis is configured for each configured kinematics and whether the configured axes are present.

If "A-0-0059, Scaling type for position data " is set to modulo format for kinematic axes, it is also checked whether the "A-0-0045, Modulo value" is greater than zero and an integer.

If this is not the case, the parameter "K-0-0008, Axis configuration list" is entered in the parameter "K-0-0011, List of all invalid K-parameters" and an incorrect configuration is displayed (F0160040, Error in cross-check of K-parameters).

RCL Parameters

Example:

The kinematics uses the transformation scheme 10 (SCARA) with four axes.

Axes 1, 2 and 3 are positioning axes.

Axis 4 is an orientation axis.

K-0-0008 displays:

Element	Value	Comment
1	8	Kinematic axis 1: Positioning axis
2	0	Control number 0: Individual control
3	1	Axis with logic axis number 1 assigned
4	8	Kinematic axis 2: Positioning axis
5	0	Control number 0: Individual control
6	2	Axis with logic axis number 2 assigned
7	8	Kinematic axis 3: Positioning axis
8	0	Control number 0: Individual control
9	3	Axis with logic axis number 3 assigned
10	16	Kinematic axis 4: Orientation axis
11	0	Control number 0: Individual control
12	4	Axis with logic axis number 4 assigned

The following axis types are available for positioning and orientation axes (see C-0-0484):

- real axis
- virtual axis
- controller axis.

For belt axes, the following are also available:

- encoder axis and
- link axis (cross link SERCOS 2).

14.3.10 K-0-0009, Osci: list of all K-parameters that can be oscilloscoped

Function This parameter contains the list of all K parameters which can be oscilloscoped.

14.3.11 K-0-0010, List of all K-parameters

Function This list includes all K-parameters. The entry is made in ID number format. The entries are sorted by meaning and how they relate to each other and not by ID number or alphabetically.

K-parameters are stored in the order of these entries.

The ID number format is as follows:

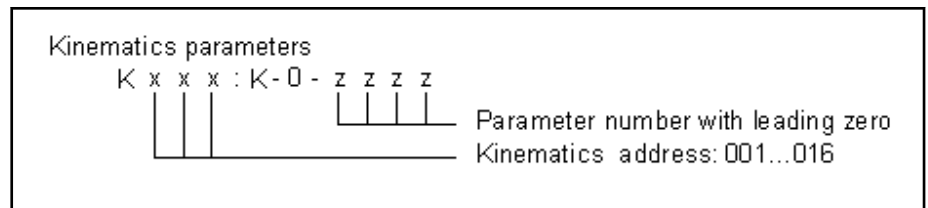


Fig. 14-5: Structure of a K-parameter

Example:

K001: K-0-0004 - cinematics 1; parameter K-0-0004.

14.3.12 K-0-0011, List of all invalid K-parameters

- Function** This list displays all invalid K-parameters in the result
- of a checksum test, e.g. after a firmware update, or
 - a plausibility test, e.g. in the result of an illegal operation mode.
- The entry is made in ID number format.

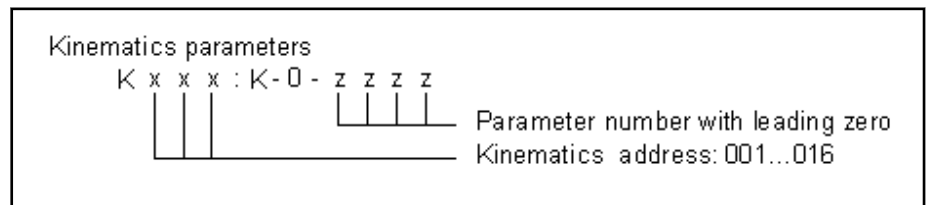


Fig. 14-6: Structure of a K-parameter

Use K001: K-0-0008 - cinematics 1; parameter K-0-0008.

14.3.13 K-0-0012, Checksum of all K-parameters

- Function** When reading this parameter the checksum is generated by all K parameter values the IDNs of which are listed in parameter "K-0-0013, IDN list of checksum K parameter".

14.3.14 K-0-0013, IDN list of checksum K-parameters

- Function** The parameter's ID numbers whose checksum is to be formed are entered in this parameter.
- The checksum formation is triggered when the "K-0-0012, Checksum of all C-parameters" is read out.
- In the default state, no ID numbers are entered in K-0-0013.

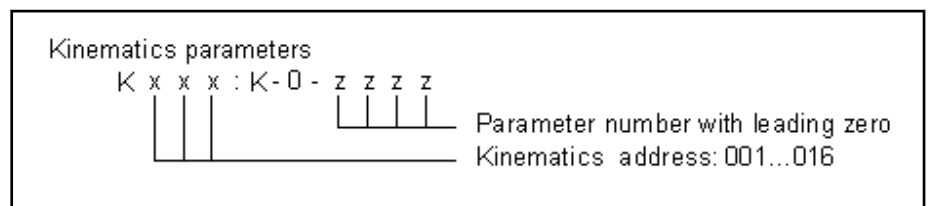


Fig. 14-7: Structure of a K-parameter

Use K001: K-0-0008 - cinematics 1; parameter K-0-0008.

14.3.15 K-0-0014, Path acceleration, change time

- Function** This parameter is only effective with the RCL commands MOVE LINEAR and MOVE CIRCULAR in the block transition mode PROGR_SLOPE.
- In the block transition mode BLOCK_SLOPE, a jerk limitation can be programmed using the RCL keyword "J".

RCL Parameters

The change time only affects the acceleration, but not the deceleration.

The deceleration change time is set using parameter K-0-0015.



The deceleration change time specified here is the maximum time until the acceleration/ braking deceleration/ programmed in the RCL program is reached.

We distinguish 3 movement forms:

- Acceleration in rectangle mode:
K-0-0014 and K-0-0015 are 0.

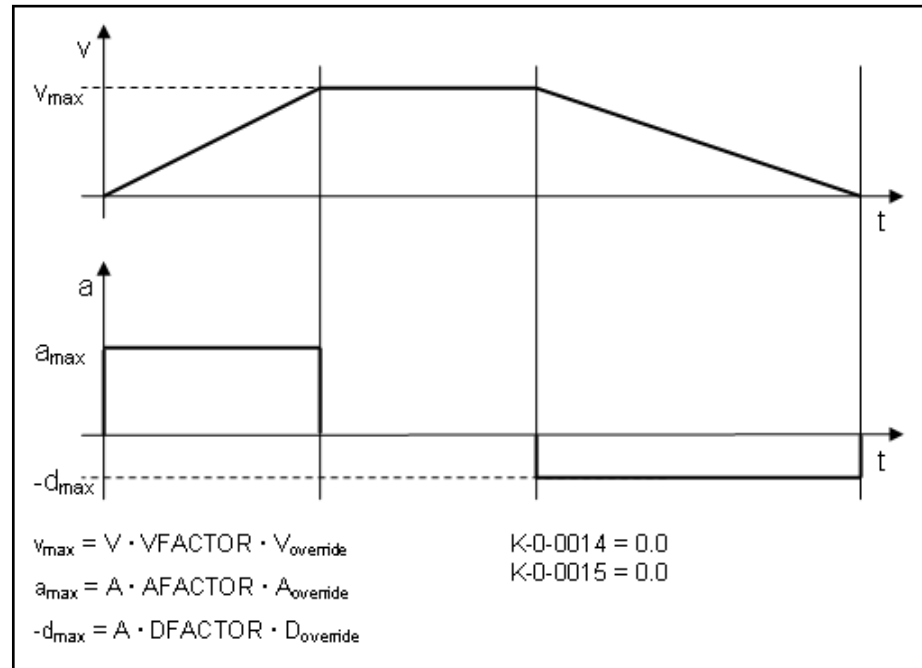


Fig. 14-8: Acceleration/deceleration in rectangle mode

The resulting jerk j is mathematically infinite.

- Acceleration in trapezoid mode:
K-0-0014 and/or K-0-0015 are greater or equal to the Sercos cycle time (see also C-0-0400). The programmed acceleration must be constantly available at a certain time to reach the programmed velocity.

RCL Parameters

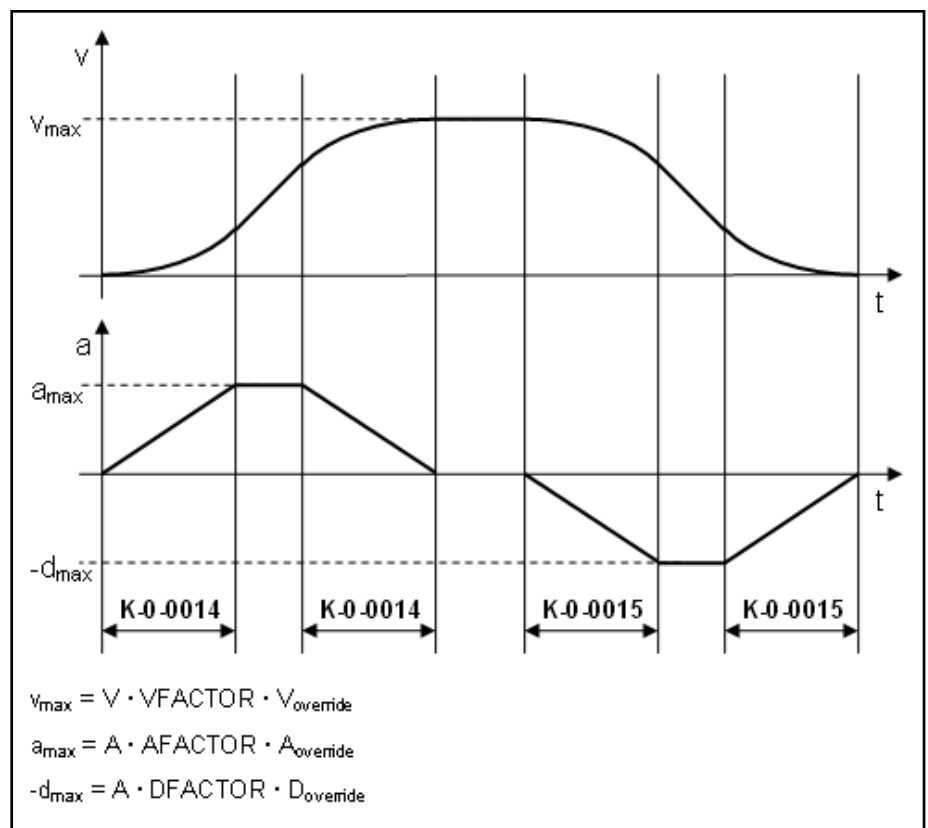


Fig. 14-9: Acceleration/deceleration in trapezoid mode

$$j = \frac{a_{\max}}{K-0-0014}$$

The resulting jerk j is

- Acceleration in triangle mode:

The times specified in K-0-0014 and K-0-0015 cannot be maintained to reach the programmed acceleration.

RCL Parameters

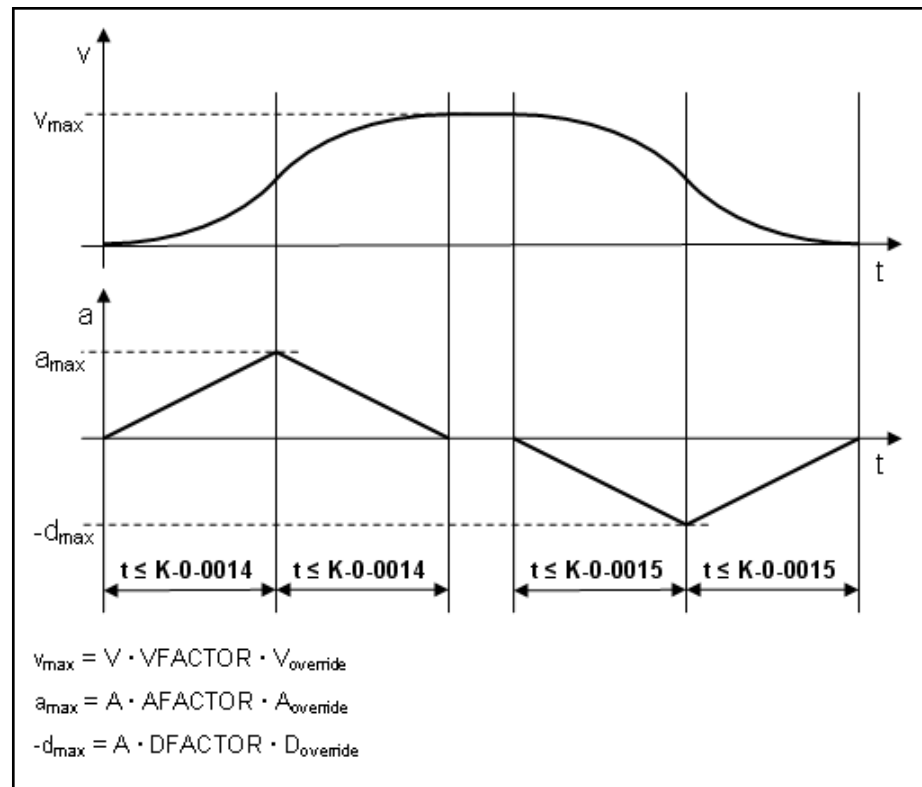


Fig. 14-10: Acceleration/deceleration in triangle mode

$$j = \frac{a_{\max}}{t}$$

The resulting jerk j is

Use The greater the change times, the smoother the motion profile. If the programmed acceleration/deceleration value is too large, the path planning is switched over to the aforementioned triangle mode.



The greater the programmed acceleration and the smaller the acceleration change time, the greater the jerk.

Low velocities can result in the very small acceleration change time t in triangle mode, which, in turn, produces a large jerk.

14.3.16 K-0-0015, Path deceleration, change time

Function This parameter is only effective with the RCL commands MOVE LINEAR and MOVE CIRCULAR in the block transition mode PROGR_SLOPE.

In the block transition mode BLOCK_SLOPE, a jerk limitation can be programmed using the RCL keyword "J".

The change time only affects the deceleration, but not the acceleration.

The acceleration change time is set using parameter K-0-0014.



The deceleration change time specified here is the maximum time until the acceleration/ braking deceleration/ programmed in the RCL program is reached.

We distinguish 3 movement forms:

- Acceleration in rectangle mode:
K-0-0014 and K-0-0015 are 0.

RCL Parameters

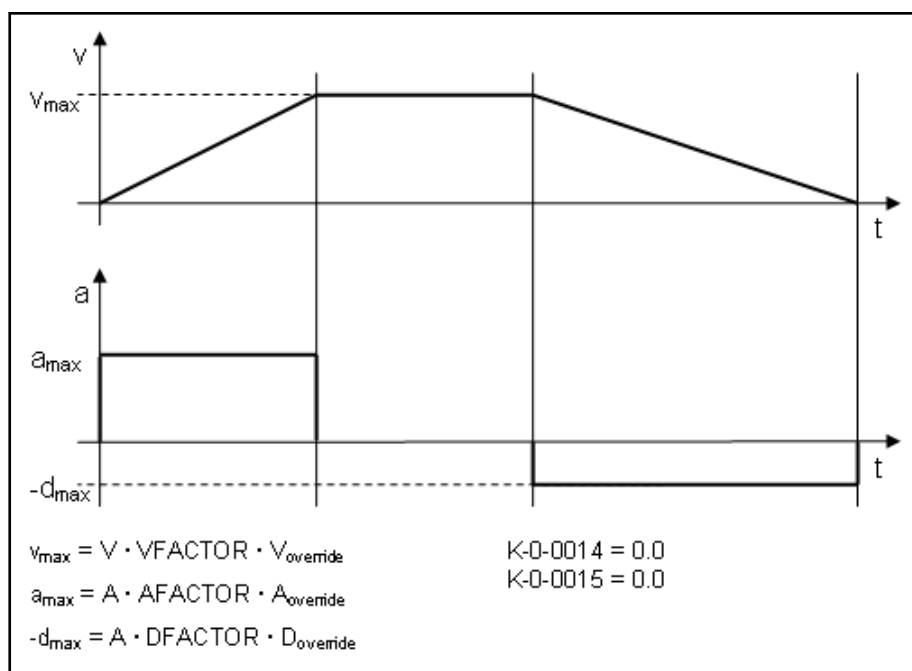


Fig. 14-11: Acceleration/deceleration in rectangle mode

The resulting jerk j is mathematically infinite.

- Acceleration in trapezoid mode:
K-0-0014 and/or K-0-0015 are greater or equal to the Sercos cycle time (see also C-0-0400). The programmed acceleration must be constantly available at a certain time to reach the programmed velocity.

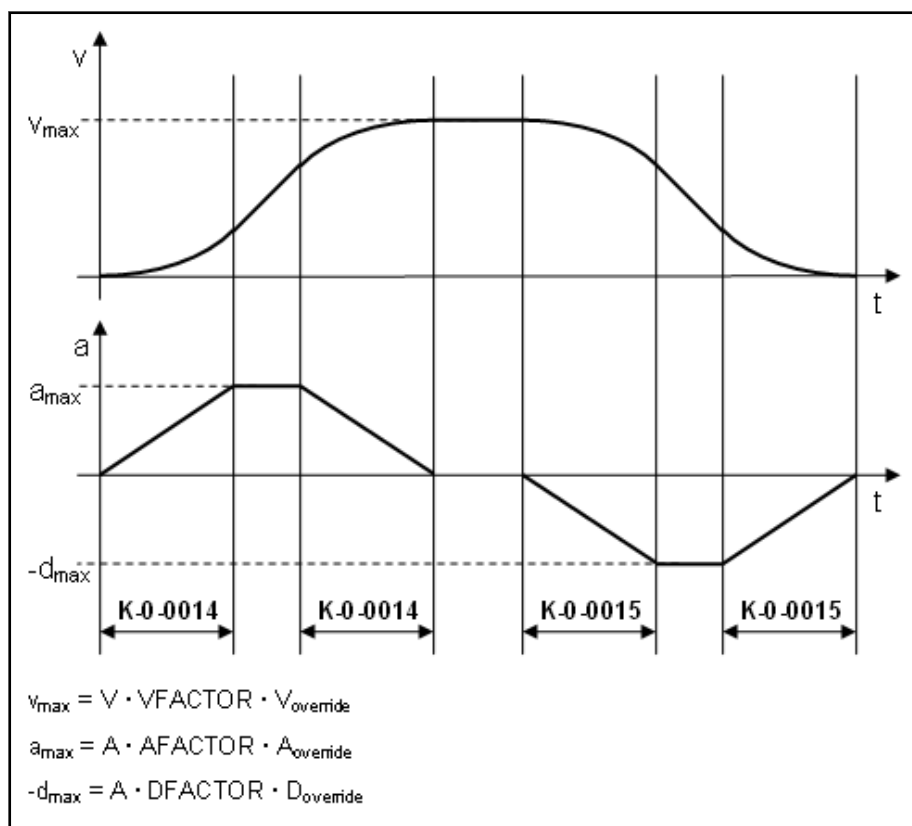


Fig. 14-12: Acceleration/deceleration in trapezoid mode

RCL Parameters

$$j = \frac{a_{\max}}{K-0-0015}$$

The resulting jerk j is

- Acceleration in triangle mode:

The times specified in K-0-0014 and K-0-0015 cannot be maintained to reach the programmed acceleration.

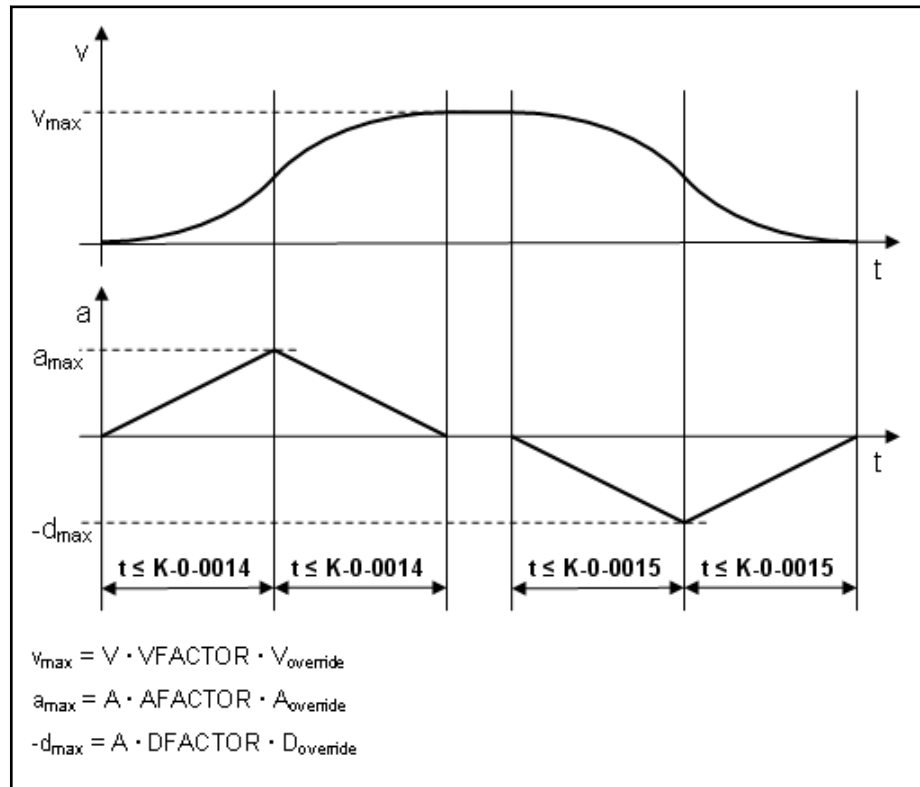


Fig. 14-13: Acceleration/deceleration in triangle mode

$$j = \frac{a_{\max}}{t}$$

The resulting jerk j is

Use The greater the change times, the smoother the motion profile. If the programmed acceleration/deceleration value is too large, the path planning is switched over to the aforementioned triangle mode.



The greater the programmed acceleration and the smaller the acceleration change time, the greater the jerk.

Low velocities can result in the very small acceleration change time t in triangle mode, which, in turn, produces a large jerk.

14.3.17 K-0-0016, Project identification number

Function IndraWorks entered a series of unique characters in this parameter to identify worldwide the project.

This series of characters is one to one for each project.

This parameter has no influence on the functions of the Motion Logic Control.



This parameter is for internal purposes only.

14.3.18 K-0-0020, Kinematic diagnostic message

Function These parameters contain the current diagnostic message for the kinematic (message, warning or error).

If bit 8 or bit 9 of the parameter "K-0-0021, Kinematics status" is set to 1, this parameter contains the warning (bit 8) or error (bit 9) text.

From the perspective of the PLC, this information is available to the kinematic "*" as

- KinData[*].dwKinStatus_i.8= TRUE, diagnostic message in K-0-0020 is a warning text or
- KinData[*].dwKinStatus_i.8= TRUE, diagnostic message in K-0-0020 is an error text.

14.3.19 K-0-0021, Kinematic status

Function This parameter describes the current status of the kinematic.

This is internal system information that is triggered via PLC function blocks, for example. This is not information on the state of the kinematics state machine (see also K-0-0022).

Structure

Bit	Content	Bit name
2...0	001 - Automatic 010 - Manual	Kin_OpMode_Auto Kin_OpMode_Manual
4..3	Jog coordinate system 00 - JC 01 - WC original 10 - WC workpiece 11 - WC tool	Kin_JogCoord1 (Bit 3) Kin_JogCoord2 (Bit 4)
5	All axes of the kinematics in standstill. (Belt axes are not considered).	Kin_StandStill
6	Command position = Actual position The programmed end position is reached.	Kin_InPosition
7	Belt-synchronous	Kin_BeltSynchronous
8	Warning	Kin_Warning
9	Error	Kin_Error
10	Homing required. Not all axes are referenced.	Kin_NotHomed
12	Transformed WCs are invalid	Kin_TrafoValuesInvalid
13	Operating mode MANUAL_JOG is enabled KinData[].stJogCom_q.enJogCmdVel KIN_JOG_STOP	Kin_JogEnabled
14	All axes ready for operation (AH or AF)	Kin_DrivesOn
15	Feed enabled Enabled for traversing motions.	Kin_FeedEnabled
16	Block processing enabled Processing the next block enabled	Kin_BlockEnabled

RCL Parameters

Bit	Content	Bit name
17	Error reaction activated	Kin_ErrorReaction
19..18	JOG-MODE 00 - small steps 01 - big steps 10 - slow 11 - fast	Kin_JogMode1 (Bit 18) Kin_JogMode2 (Bit 19)
21	Program selected	Kin_ProgSelected
23	Reset is active	Kin_ResetActive
24	Program ended	Kin_ProgEnded
25	Program started	Kin_ProgStarted
26	Program: PAUSE command active	Kin_ProgPaused

Fig.14-14: K-0-0021 Kinematic status



Bits 0..2 for the operation modes are evaluated as numerals (0..7).

Use Bit 5: All kinematics axes are stopped.

This bit signals that the axes positioned by the kinematics are at a standstill. This means that velocity is zero.

This bit differs from Kis_Standstill (see parameter K-0-0022) which refers to the kinematics state machine.

Bit 8: Warning

If an incorrect status is recognized, which correct operation nonetheless allows, a warning is generated.

Example:

The programmed path velocity cannot be reached due to the maximum velocity of one or more axes.



If they are ignored, some warnings lead to errors.

Bit 9: Error

If an incorrect status is recognized, which correct operation no longer allows, an error is generated.

Example:

The programmed position is located beyond the travel limit for one or more axes.

Bit 12: Transformed world coordinates are invalid

The current world coordinates for the kinematics indicated in KinData are invalid. For example, this is the case if not all of the axes are assigned to the kinematics or if there is an error in the transformation.

Bit 17: Error reaction is enabled

When errors with error reaction capability occur (F0 errors do not have error reaction capability), all of the axes assigned to the axes are stopped as quickly as possible within the programmed path.

Bit 17 is deleted when all axes are stopped.

Bit 26: Program: PAUSE command enabled

In the robot program, a PAUSE command is enabled (see PAUSE command).

14.3.20 K-0-0022, Extended kinematic status

Function This parameter uses several status bits to describe the current state of the kinematics state machine. Each state is assigned to a status bit combination in the parameter K-0-0022.

The abbreviation KIS in the bit designation stands for "Kinematics State Machine" in contrast to the KIN bits. The KIN bits describe individual kinematic status messages.

Structure The kinematic state machine is explained in the following figure:

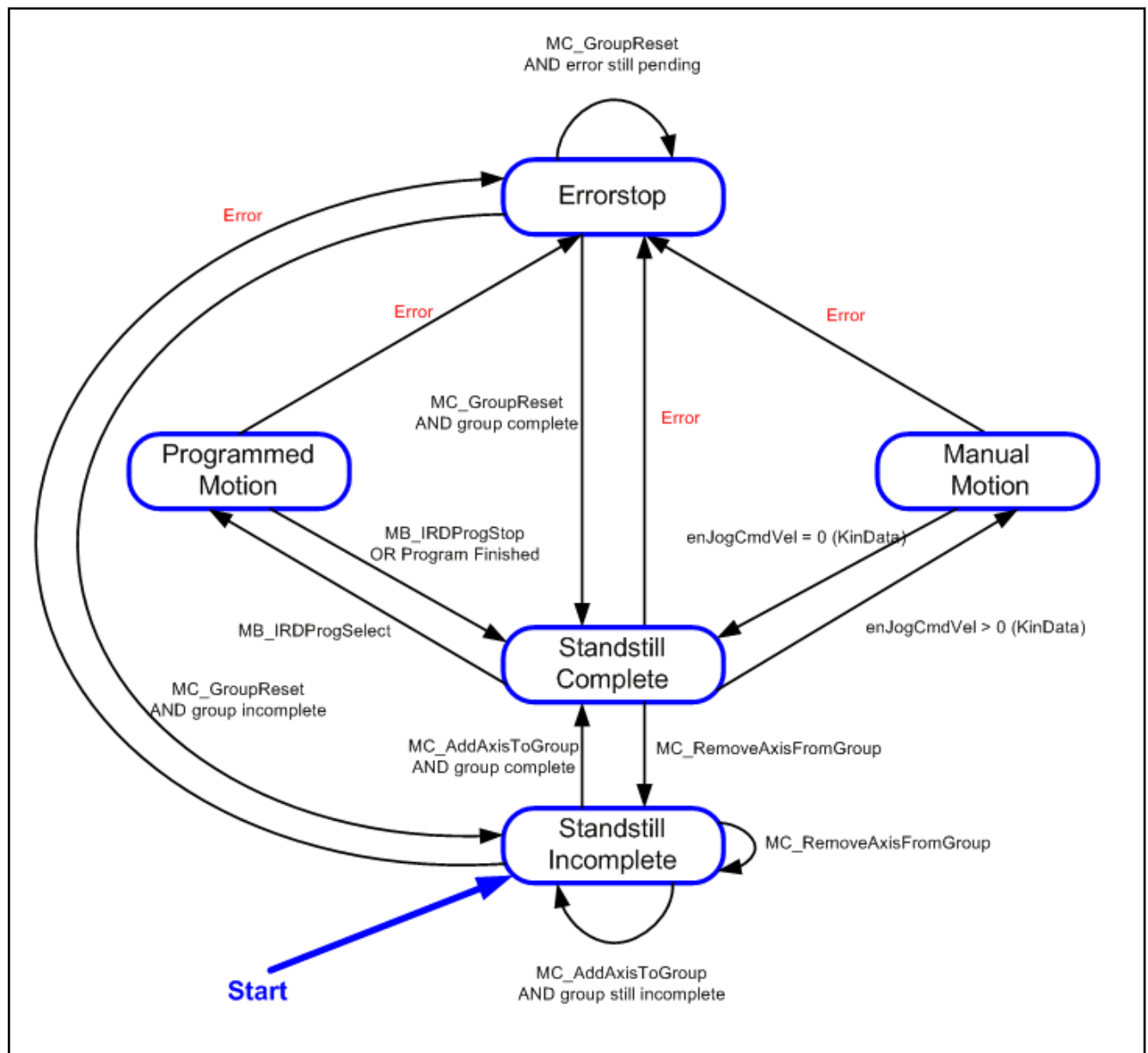


Fig. 14-15: Kinematic state machine

"Programmed Motion" has the following substates:

RCL Parameters

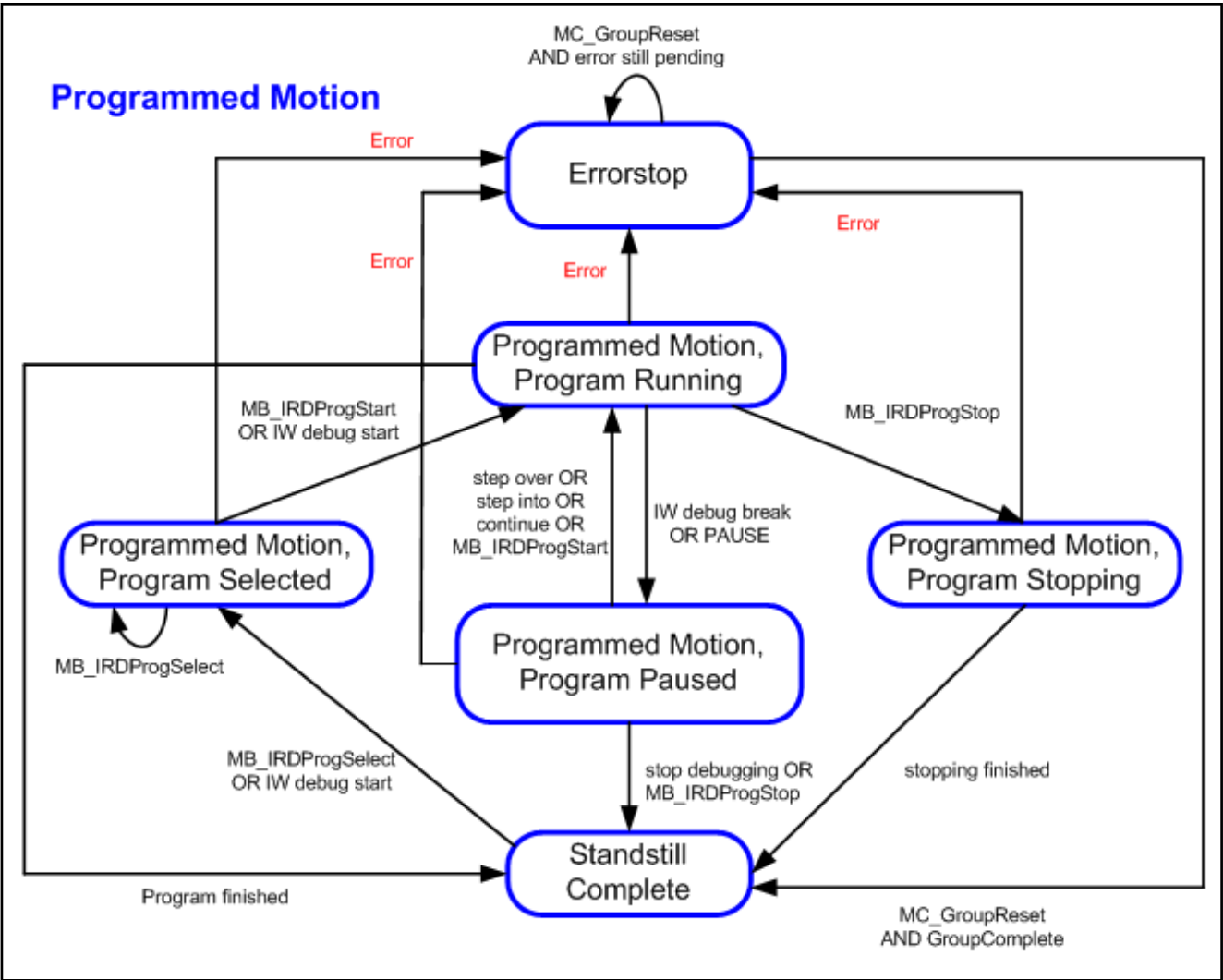


Fig. 14-16: "Programmed Motion" state machine

"Errorstop" state The "Errorstop" state is reached when an error occurs while the kinematic is running. The axes remain in the "Coordinated Motion" state and remain assigned to the kinematics. Except for axes that have caused an error or that are currently in error state.

The "Errorstop" state can be reset via the MB_GroupReset function block or the command parameter "C-0-1030, Command: Clear all control errors". Afterwards, it is switched to the "Standstill Complete" state for complete kinematics or to the "Standstill incomplete" state for incomplete kinematics.

Bit field definition In parameter K-0-0022, the following bits are available:

Bit	Content	Name of the bit
0	Kinematic is in "error stop"	Kis_Errorstop
1	Kinematic is "stopping"	Kis_Stopping
3	Not all axes are assigned to the kinematic (incomplete)	Kis_Incomplete
5	Kinematic is in "programmed motion"	Kis_ProgrammedMotion
7	Kinematic is in "StandStill"	Kis_StandStill

RCL Parameters

Bit	Content	Name of the bit
8	Program is selected	Kis_ProgramSelected
9	Kinematic is enabled (running)	Kis_Running
10	Kinematic is in jog status KinData[].stJogCom_q.enJogCmdVel KIN_JOG_STOP	Kis_ManualMotion
11	Kinematic is in "Pause"	Kis_ProgramPaused

Fig. 14-17: K-0-0022 Extended kinematics status

Bit assignment/state

The assignment between the outlined states and their respective bits is shown in the following table.

1 : Bit set

0 : Bit not set

x : Bit irrelevant for state

State	Bits 12...8	Bits 7...4	Bits 3...0
Standstill incomplete	00000	1x0x	1x00
Standstill complete	00000	1x0x	0x00
Programmed Motion, Program Selected	00001	0x1x	0x00
Programmed Motion, Program Running	00010	0x1x	0x00
Programmed Motion, Program Stopping	00000	0x1x	0x10
Programmed Motion, Program Paused	01000	0x1x	0x00
Errorstop	00000	0x0x	0x01
Manual Motion	00100	0x0x	0x00
GroupMoving	10000	0x0x	0x00

Fig. 14-18: Assigning kinematic state to status bits

14.3.21 K-0-0023, Kinematics diagnostic number

Function If bits 8 or 9 of parameter "K-0-0021, Kinematic status" are set, the current diagnostic number can be read this parameter.

14.3.22 K-0-0031, Transformation scheme

Function This parameter describes the scheme used to couple the kinematic axes mechanically. It identifies how pivot angles or axis positions in Cartesian coordinates are to be transformed.

Use The meaning of the individual values of parameter "K-0-0040, Transformation, lengths of the mechanics " and "K-0-0041, Transformation, factors" depends on this set transformation scheme.

More specific information can be found in the special transformation documentation.

Value	Comment
0	Robot type 00, Cartesian robot
9	Robot type 09, 3 axis Scara with 1 optional manual axis
10	Robot type 10, 4 axis Scara with 1 optional manual axis

RCL Parameters

Value	Comment
20	Robot type 20, 3 axis Delta with 1 optional manual axis
≥1000	User specified transformation

Fig. 14-19: Valid values for the transformation scheme



This parameter must be set when starting operation according to DCC service specifications.

When switching after phase 4, a check is made to see if enough axes are configured for the transformation scheme in parameter "K-0-0008, Axis configuration list."

If this not the case, this parameter is entered in parameter "K-0-0011, List of all invalid K-parameters" and

a configuration error is displayed ("F0160040, Error in cross-check K-parameters").

The kinematic wizard does not make any parameter checks for user-defined kinematic types.

14.3.23 K-0-0032, Referencing axes, command configuration

Function A presetting to reference all kinematic axes can be set for various robot types.

Referencing all axes:

- 0 - no, default setting
- 1 - yes, only allowed for specific transformations.



This parameter is to be set when commissioning according to DCC service specifications.

14.3.24 K-0-0033, IPOS, update when coordinate change

Function A presetting can be made for IPOS updater at coordinate change for various robot types.

IPOS, update when coordinate change:

- 0 - no, default setting,
- 1 - yes, only allowed for specific transformations.



This parameter must be set when commissioning according to DCC service specifications.

14.3.25 K-0-0036, WC axis monitoring

Function WC axis monitoring can be configured for various robot types (monitoring axis velocity).

Use *WC axis monitoring:*

- 0 = No monitoring
- 1 = Monitoring with movement abort.
- 2 = Only in Manual mode, monitoring with limitation of axis velocities. No monitoring in Automatic mode.
- 3 = Do not use this setting.

- 4 = In Manual and Automatic mode, monitoring with limitation of axis velocities. The limitation is effective for the complete motion.
- 5 = Continuous monitoring of the axis velocity during linear motion.

Please note the following:

- The fast JC jog velocities set in parameter A-0-0083 are monitored in Manual mode. The maximum axis velocities set in $\text{MIN}\{|A-0-0032| \vee |A-0-0033|\}$ are monitored in Automatic mode.
- If the axis limitation occurs during the motion, the velocity at the TCP does not match the programmed velocity.
- If the axis limitation occurs, there are path inaccuracies.

14.3.26 K-0-0040, Transformation, lengths of the mechanics

Function Lengths required to transform axis positions into Cartesian coordinates are configured in this parameter.

The meaning of the individual values depends on the set "K-0-0031, Transformation scheme".

More specific information can be found in the special transformation documentation.



When switching to phase 4, the values are checked depending on parameter "K-0-0031, Transformation scheme". (For transformation scheme SCARA, it is checked for example whether the first two lengths are not equal to zero.)

If an error occurs when the values are checked,

- this parameter is entered in the parameter "K-0-0011, List of all invalid K-parameters" and
- an incorrect configuration is displayed ("F0160040, Error in cross-check K-parameters").

14.3.27 K-0-0041, Transformation, factors

Function Factors required to transform axis positions to Cartesian coordinates are configured in this parameter.

The meaning of the individual values depends on the set "K-0-0031, Transformation scheme". More specific information can be found in the special transformation documentation.



When switching to phase 4, the values are checked depending on parameter "K-0-0031, Transformation scheme".

If an error occurs when the values are checked,

- the parameter "K-0-0041, Transformation factors" is entered in the parameter "K-0-0011, List of all invalid K-parameters" and
- an incorrect configuration is displayed (F0160040, Error in cross-check K-parameters).

14.3.28 K-0-0042, WC system offset

Function This parameter allows the coordinate system origin of the transformation scheme to be displaced so that it is possible to edit coordinate values that refer to a reference point in the editing cell.

RCL Parameters

Structure This parameter is a list parameter with six values.

Element	Meaning
1	Displacement in Cartesian x-direction [in mm or inch (see K-0-0054)]
2	Displacement in Cartesian y-direction [in mm or inch (see K-0-0054)]
3	Displacement in Cartesian z-direction [in mm or inch (see K-0-0054)]
4	Rotation about the Cartesian x-axis [in degrees] (see note)
5	Rotation about the Cartesian y-axis [in degrees] (see note)
6	Rotation about the Cartesian z-axis [in degrees] (see note)

Use The first three values indicate the position of the coordinate system origin from the perspective of the cell coordinate system. The elements 4 to 6 indicate the rotation of the coordinate system.



The rotation of the coordinate system is only possible for particular transformation schemes (see K-0-0031). This is documented in the corresponding descriptions (No information on the rotation of the coordinate system means that this option is not supported with this particular transformation scheme).

14.3.29 K-0-0043, Assignment list of workpiece coordinates

Function This parameter describes which robot coordinate affects which displacement/rotation of the workpiece coordinate system. If the kinematics is provided with less than six degrees of freedom, not all displacements/rotations of the workpiece coordinate system are compensated. Using this parameter, it is reported to the control

- which of the six general workpiece coordinates are considered at all from the mechanics
- which robot coordinate corresponds to which workpiece coordinate

Structure This parameter is a list parameter with six values.

Element	Meaning
1	Robot coordinate including the displacement in Cartesian x-direction
2	Robot coordinate including the displacement in Cartesian y-direction
3	Robot coordinate including the displacement in Cartesian z-direction
4	Robot coordinate including the rotation around the Cartesian x-axis
5	Robot coordinate including the rotation around the Cartesian y-axis
6	Robot coordinate including the rotation around the Cartesian z-axis



If a rotation is mechanically impossible, 0 has to be entered in the respective element.



The rotation of the coordinate system is only possible for particular transformation schemes (see K-0-0031). This is documented in the respective descriptions (if no reference is made on the rotation of the coordinate system, this option is not supported with this transformation scheme).

Use *Example:*

4-axis Scara

A 4-axis Scara can travel in x-, y-, z- direction and rotate around around the z-axis. Thus, the following assignment results for the parameter K-0-0043:

K-0-0043 assignment at a 4-axis Scara

Element	Value	Meaning
1	1	The first axis of the kinematics is used for displacement in x-direction
2	2	The second axis of the kinematics K-0-0042 is used for displacement in y-direction
3	3	The third axis of the kinematics K-0-0042 is used for displacement in z-direction
4	0	Rotation of the workpiece around the x-axis is mechanically not possible
5	0	Rotation of the workpiece around the y-axis is mechanically not possible
6	4	The fourth axis of the kinematics is used for rotating the workpiece coordinate system along z-direction

Fig. 14-20: K-0-0043 assignment at a 4-axis Scara

14.3.30 K-0-0050, Distance units WC

Function The distance and orientation units for each element of a list parameter in the world coordinate system are separated with a command and saved in this text parameter.

- Only legible entries are automatically determined.
- One entry (with comma separator) for each coordinate axis (x, y, z, alpha, beta, gamma),
- depending on the type of transformer K-0-0031 and the axes used K-0-0008.
 - For x, y, z-axes: mm
 - For alpha, beta, gamma: degrees

Use *Examples*

1. Cartesian robot with X, Y, Z, C
K-0-0050 = "mm,mm,mm,degree"
2. Scara with distance lathe spindle (X, Y, Z, C)
K-0-0050 = "mm,mm,mm,degree"
3. Cartesian robot with X, Y, Z, C, B
K-0-0050 = "mm,mm,mm,degree,degree"

RCL Parameters

4. Cartesian robot with X, Y, C
K-0-0050 = "mm,mm,degree"
5. Delta with C-axis and 2 belts
K-0-0050 = "mm,mm,mm,degree"



The belts are not included in the lists for the world coordinate system.

14.3.31 K-0-0051, Velocity units WC

Function The velocity units for each element of a list parameter in the world coordinate system are separated with a command and saved in this text parameter.

- Only legible entries are automatically determined.
- One entry (with comma separator) for each coordinate axis (x, y, z, alpha, beta, gamma),
- depending on the type of transformer K-0-0031 and the axes used K-0-0008.
 - For x, y, z-axes: mm/s
 - For alpha, beta, gamma: degree/s

Use *Examples*

1. Cartesian robot with X, Y, Z, C
K-0-0051 = "mm/s,mm/s,mm/s,degree/s"
2. Scara with distance lathe spindle (X, Y, Z, C)
K-0-0051 = "mm/s,mm/s,mm/s,degree/s"
3. Cartesian robot with X, Y, Z, C, B
K-0-0051 = "mm/s,mm/s,mm/s,degree/s, degree/s"
4. Cartesian robot with X, Y, C
K-0-0051 = "mm/s,mm/s,degree/s"
5. Delta with C-axis and 2 belts
K-0-0051 = "mm/s,mm/s,mm/s,degree/s"



The belts are not included in the lists for the world coordinate system.

14.3.32 K-0-0052, Acceleration units WC

Function The acceleration or deceleration units for each element of a list parameter in the world coordinate system are separated with a command and saved in this text parameter.

- Only legible entries are automatically determined.
- One entry (with comma separator) for each coordinate axis (x, y, z, alpha, beta, gamma),
- depending on the type of transformer K-0-0031 and the axes used K-0-0008.
 - For x, y, z-axes: mm/s²
 - For alpha, beta, gamma: degree/s²

Use *Examples*

1. Cartesian robot with X, Y, Z, C

- K-0-0052 = "mm/s^2,mm/s^2,mm/s^2,degree/s^2"
2. Scara with distance lathe spindle (X, Y, Z, C)
K-0-0052 = "mm/s^2,mm/s^2,mm/s^2,degree/s^2"
 3. Cartesian robot with X, Y, Z, C, B
K-0-0052 = "mm/s^2,mm/s^2,mm/s^2,degree/s^2,degree/s^2"
 4. Cartesian robot with X, Y, C
K-0-0052 = "mm/s^2,mm/s^2,degree/s^2"
 5. Delta with C-axis and 2 belts
K-0-0052 = "mm/s^2,mm/s^2,mm/s^2,degree/s^2"



The belts are not included in the lists for the world coordinate system.

14.3.33 K-0-0053, Jerk units WC

Function The jerk units for each element of a list parameter in the world coordinate system are separated with a command and saved in this text parameter.

- Only legible entries are automatically determined.
- One entry (with comma separator) for each coordinate axis (x, y, z, alpha, beta, gamma),
- depending on the type of transformer K-0-0031 and the axes used K-0-0008.
 - For x, y, z-axes: mm/s³
 - For alpha, beta, gamma: degree/s³

Use *Examples*

1. Cartesian robot with X, Y, Z, C
K-0-0053 = "mm/s^3,mm/s^3,mm/s^3,degree/s^3"
2. Scara with distance lathe spindle (X, Y, Z, C)
K-0-0053 = "mm/s^3,mm/s^3,mm/s^3,degree/s^3"
3. Cartesian robot with X, Y, Z, C, B
K-0-0053 = "mm/s^3,mm/s^3,mm/s^3,degree/s^3, degree/s^3"
4. Cartesian robot with X, Y, C
K-0-0053 = "mm/s^3,mm/s^3,degree/s^3"
5. Delta with C-axis and 2 belts
K-0-0053 = "mm/s^3,mm/s^3,mm/s^3,degree/s^3"



The belts are not included in the lists for the world coordinate system.

14.3.34 K-0-0054, Unit of measurement for kinematic data

Function This parameter determines the scaling unit of the kinematics.

Structure The following settings can be made:

RCL Parameters

Bit	Name/function
0	Unit 0: mm 1: inch
15..1	Reserved

14.3.35 K-0-0070, Belt number of kinematics

- Function** This parameter contains the current number of belts of the kinematics.
The value results from "K-0-0008, Axes configuration list". It only contains the sum of there configured belts.
The parameter "K-0-0007, Axis number of kinematics" contains the sum of all axes and belts.

14.3.36 K-0-0071, Coupling factors, belt 1

- Function** The belt coupling factors specify how belt position changes of the first belt affect coordinate changes in the program sections between SYNCHRON and SYNCHRON_END.
- Structure** These parameter values are only important if at least a value of 1 is written in the parameter "K-0-0070, Belt number of kinematics".
This is a list parameter. One element in the list is responsible for each coordinate.

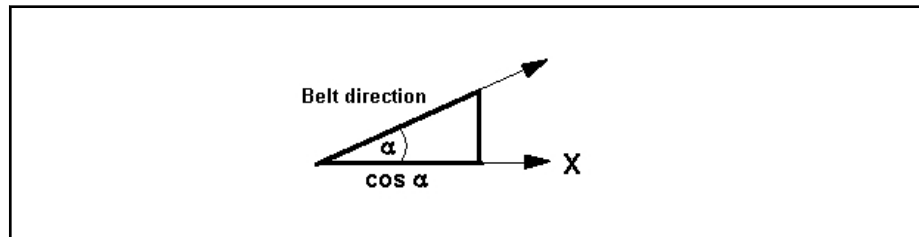


Fig. 14-21: Phase: Belt direction/axis

The value for a coordinate corresponds to the cosine value ($\cos \alpha$) of the phase between the belt and the respective coordinates.



Use the Windows calculator (View -> Scientific) with the setting DEG (not (new) Grads).

- Use** The assignment of the coordinates to the spatial direction does generally depend on the parameter "K-0-0031, Transformation scheme". Generally, the first coordinate is assigned to the x-direction, the second coordinate to the y-direction and the third coordinate to the z-direction. The following examples are based on this definition.

Example 1:

The belt is exactly parallel to the x-axis. Increasing belt values cause the part on the belt moving in positive x-direction.

K-0-0071 displays:

RCL Parameters

Element	Value	Comment
1	1.000000	Cos (0 degrees)
2	0.000000	Cos (90 degrees)
3	0.000000	Cos (90 degrees)

Example 2:

The belt is exactly parallel to the y-axis. Increasing belt values cause the part on the belt moving in negative y-direction.

K-0-0071 displays:

Element	Value	Comment
1	0.000000	Cos (90 degrees)
2	-1.000000	Cos (180 degrees)
3	0.000000	Cos (90 degrees)

Example 3:

The belt and the x-axis are in an angle of 30 degrees to each other; the belt and the y-axis in an angle of 60 degrees. Increasing belt values cause the part on the belt moving in positive x-direction and in positive y-direction.

K-0-0071 displays:

Element	Value	Comment
1	0.866025	Cos (30 degrees)
2	0.500000	Cos (60 degrees)
3	0.000000	Cos (90 degrees)



The root from the sum of the square of the cosine values has to be as exactly 1 as possible.

If the value is higher than 1, it corresponds to a length elongation in belt direction. If the value is smaller than 1, there is a length compression.

14.3.37 K-0-0072, Coupling factors, belt 2

Function The belt coupling factors specify how belt position changes of the second belt affect coordinate changes in the program sections between SYNCHRON and SYNCHRON_END.

These parameter values are only important if at least a value of 2 is written in the parameter "K-0-0070, Belt number of kinematics".

The parameter structure corresponds to the structure of the parameter "K-0-0071, Coupling factors, belt 1".

14.3.38 K-0-0073, Coupling factors, belt 3

Function The belt coupling factors specify how belt position changes of the third belt affect coordinate changes in the program sections between SYNCHRON and SYNCHRON_END.

These parameter values are only important if at least a value of 3 is written in the parameter "K-0-0070, Belt number of kinematics".

RCL Parameters

The parameter structure corresponds to the structure of the parameter "K-0-0071, Coupling factors, belt 1".

14.3.39 K-0-0074, Coupling factors, belt 4

Function The belt coupling factors specify how belt position changes of the fourth belt affect coordinate changes in the program sections between SYNCHRON and SYNCHRON_END.

These parameter values are only important if at least a value of 4 is written in the parameter "K-0-0070, Belt number of kinematics".

The parameter structure corresponds to the structure of the parameter "K-0-0071, Coupling factors, belt 1".

14.3.40 K-0-0080, Jogging, distance short in WC system

Function This parameter provides the short distance for the incremental jogging in the world coordinate system.



This parameter can also be written in phase 4. The new value is applied at the next jog command. It is not possible to change the parameter for the active jog command.

- Structure**
- List parameter:
One entry for every coordinate axis (e.g. x, y, z, alpha, beta, gamma),
 - Unit:
Depends on the transformation type K-0-0031 and the axes used (see K-0-0008).
The units are stored in K-0-0050.
Example:
for x, y, z in mm or in inch (see K-0-0054),
for alpha, beta, gamma in degrees.
 - Value range: >0.0 (floating point number)
 - Default: 1.0 mm or inch or degree.

Use The RobotControl uses this parameter for the incremental axis jogging in the world coordinate system.

The PLC program can use the jog bits (KinData[1].stJogCmd_q) to enable the incremental jogging in the world coordinate system (WC) with short distance:

E.g. incremental jogging in WC with small distance of the 3rd axis in positive direction:

Program:

```
KinData[1].stJogCmd_q.enJogCmdCoordSys:=KIN_CORD_SYS_ORIGIN_COORD;
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STEP_SMALL;
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*no axis in neg.*)
KinData[1].stJogCmd_q.wJogCmdPositive := 4; (*3rd bit for 3rd axis of the kinematics or group*)
```

- The short distance to jog the axis is written in K-0-0080.
- The maximum velocity is written in K-0-0083 (always fast for incremental jogging).
- The maximum acceleration is written in K-0-0084 and
- the maximum deceleration is written in K-0-0085.
- The maximum jerk for acceleration is written in K-0-0086 and

- the maximum deceleration jerk is written in K-0-0087.

The jog movement can be prematurely terminated by setting enJogMode = KIN_JOG_STOP.

Program:

```
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STOP;
```

14.3.41 K-0-0081, Jogging, distance long in WC system

Function This parameter stores the long distance for incremental jogging in the world coordinate system.



This parameter can also be written in phase 4. The new value is applied at the next jog command. It is not possible to change the parameter for the active jog command.

- Structure**
- List parameter:
One entry for every coordinate axis (e.g. x, y, z, alpha, beta, gamma),
 - Unit:
Depends on the transformation type K-0-0031 and the axis used (see K-0-0008).
The units are stored in K-0-0050.
Example:
for x, y, z in mm or in inch (see K-0-0054),
for alpha, beta, gamma in degrees.
 - Value range: >0.0 (floating point number)
 - Default: 10.0 mm or inch or degree.

Use The RobotControl uses this parameter for incremental axis jogging in the world coordinate system.

The PLC program can use the jog bits (KinData[1].stJogCmd_q) to enable the incremental jogging in the world coordinate system (WC) with large distance:

E.g. incremental jogging in WC with long distance of the 3rd axis in positive direction:

Program:

```
KinData[1].stJogCmd_q.enJogCmdCoordSys:=KIN_CORD_SYS_ORIGIN_COORD;  
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STEP_LARGE;  
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*no axis in neg.*)  
KinData[1].stJogCmd_q.wJogCmdPositive := 4; (*3rd bit for 3rd axis of the kinematics or group*)
```

- The large distance for jogging the axis is written in K-0-0081.
- The maximum velocity is written in K-0-0083 (always fast for incremental jogging).
- The maximum acceleration is written in K-0-0084 and
- the maximum deceleration is written in K-0-0085.
- The maximum jerk for acceleration is written in K-0-0086 and
- the maximum deceleration jerk is written in K-0-0087.

The jog movement can be prematurely terminated by setting enJogMode = KIN_JOG_STOP.

RCL Parameters

Program:

```
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STOP;
```

14.3.42 K-0-0082, Jogging, velocity low in WC system

Function This parameter provides the slow velocity for the continuous jogging in the world coordinate system.



This parameter can also be written in phase 4. The new value is applied at the next jog command. It is not possible to change the parameter for the active jog command.

- Structure**
- List parameter:
One entry for every coordinate axis (x, y, z, alpha, beta, gamma),
 - Unit:
Depends on the transformation type K-0-0031 and the axis used (see K-0-0008).
The units are stored in K-0-0051.
Example:
for x, y, z in mm/s or inch/s (see K-0-0054),
for alpha, beta, gamma in degrees/s.
 - Value range: >0.0 (floating point number)
 - Default: 10.0 mm/s or inch/s or degree/s.

Use The RobotControl uses this parameter is used for the continuous axis jogging in the world coordinate system.

The PLC program uses the jog bits (KinData[1].stJogCmd_q) to enable the continuous jogging in the world coordinate system (WC) with low velocity:

E.g. continuous jogging in WC with low velocity of the 3. axis in positive direction:

Program:

```
KinData[1].stJogCmd_q.enJogCmdCoordSys:=KIN_CORD_SYS_ORIGIN_COORD;
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STEP_SLOW;
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*no axis in neg.*)
KinData[1].stJogCmd_q.wJogCmdPositive := 4; (*3rd bit for 3rd axis of the kinematics or group*)
```

- The low velocity is written in K-0-0082,
- the maximum acceleration is written in K-0-0084 and
- the maximum deceleration is written in K-0-0085.
- The maximum jerk for acceleration is written in K-0-0086 and
- the maximum deceleration jerk is written in K-0-0087.

The continuous jogging can be completed with two different methods:

Program:

```
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STOP;
```

and/or

Program:

```
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*no axis in neg.*)
KinData[1].stJogCmd_q.wJogCmdPositive := 0; (*no axis in pos.*)
```

14.3.43 K-0-0083, Jogging, velocity high in WC system

Function The high velocity for continuous jogging in the world coordinate system is stored in this parameter.



This parameter can also be written in phase 4. The new value is applied at the next jog command. It is not possible to change the parameter for the active jog command.

- Structure**
- List parameter:
One entry for every coordinate axis (x, y, z, alpha, beta, gamma).
 - Unit:
Depends on the transformation type K-0-0031 and the axis used (see K-0-0008).
The units are stored in K-0-0051.
Example:
for x, y, z in mm/s or inch/s (see K-0-0054),
for alpha, beta, gamma in degrees/s.
 - Value range: >0.0 (floating point number)
 - Default: 25.0 mm/s or inch/s or degree/s.

Use The RobotControl uses this parameter is used for the continuous axis jogging in the world coordinate system.

The PLC program can use the jog bits (KinData[1].stJogCmd_q) to enable the continuous jogging in the world coordinate system (WC) with high velocity:

E.g. continuous jogging in WC with high velocity of the 3. axis in positive direction:

Program:

```
KinData[1].stJogCmd_q.enJogCmdCoordSys:=KIN_CORD_SYS_ORIGIN_COORD;  
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STEP_FAST;  
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*no axis in neg.*)  
KinData[1].stJogCmd_q.wJogCmdPositive := 4; (*3rd bit for 3rd axis of the kinematics or group*)
```

- The high velocity is written in K-0-0083,
- the maximum acceleration is written in K-0-0084 and
- the maximum deceleration is written in K-0-0085.
- The maximum jerk for acceleration is written in K-0-0086 and
- the maximum deceleration jerk is written in K-0-0087.

The continuous jogging can be completed with two different methods:

Program:

```
KinData[1].stJogCmd_q.enJogMode := KIN_JOG_STOP;
```

and/or

Program:

```
KinData[1].stJogCmd_q.wJogCmdNegative := 0; (*no axis in neg.*)  
KinData[1].stJogCmd_q.wJogCmdPositive := 0; (*no axis in pos.*)
```

RCL Parameters

14.3.44 K-0-0084, Jogging, acceleration in WC system

Function The acceleration to jog (incremental and continuous) in the world coordinate system is stored in this parameter.



With the MLC 04VRS, the parameter can only be written in phase 2.

- Structure**
- List parameter:
One entry for every coordinate axis (x, y, z, alpha, beta, gamma),
 - Unit:
Depends on the transformation type K-0-0031 and the axis, see K-0-0008.
The units are stored in K-0-0052.
Example:
for x, y, z in mm/s² or inch/s² (see K-0-0054),
for alpha, beta, gamma in Grad/s².
 - Value range: >0.0 (floating point number)
 - Default: 370 mm/s² or inch/s² or degree/s².

Use The RobotControl uses this parameter to jog (incremental and continuous) in the world coordinate system.

- The maximum deceleration is written in K-0-0085,
- the maximum jerk for acceleration is written in K-0-0086 and
- the maximum deceleration jerk is written in K-0-0087.

Acceleration A-0-0084 and deceleration A-0-0085 are valid for the axis jogging (JC).

14.3.45 K-0-0085, Jogging, deceleration in WC system

Function This parameter stores the deceleration to jog (incremental and continuous) in the world coordinate system.



With the MLC 04VRS, the parameter can only be written in phase 2.

- Structure**
- List parameter:
One entry for every coordinate axis (x, y, z, alpha, beta, gamma).
 - Unit:
Depends on the transformation type K-0-0031 and the axis used (see K-0-0008).
The units are stored in K-0-0052.
Example:
for x, y, z in mm/s² or inch/s² (see K-0-0054),
for alpha, beta, gamma in Grad/s².
 - Value range: ≥ 0.0 (floating point number)
 - Default: 370 mm/s² or inch/s² or degree/s².

Use The RobotControl uses this parameter to jog (incremental and continuous) in the world coordinate system.

- The maximum acceleration is written in K-0-0084,

- the maximum jerk for acceleration is written in K-0-0086 and
- the maximum deceleration jerk is written in K-0-0087.

Acceleration A-0-0084 and deceleration A-0-0085 are valid for the axis jogging (JC).

14.3.46 K-0-0086, Jogging, acceleration jerk in WC system

Function This parameter stores the maximum acceleration jerk to jog (incremental and continuous) in the world coordinate system (indirect definition of the acceleration change time).



With the MLC 04VRS, the parameter can only be written in phase 2.

- Structure**
- List parameter:
One entry for each coordinate axis (x, y, z, alpha, beta, gamma)
 - Unit:
Depends on the transformation type K-0-0031 and the axis used (see K-0-0008).
The units are stored in K-0-0053.
Example:
for x, y, z in mm/s³ or inch/s³ (see K-0-0054),
for alpha, beta, gamma in Grad/s³.
 - Value range: ≥ 0.0 (floating point number)
 - Default: 0.0 (no jerk limitation).

Use The RobotControl uses this parameter to jog (incremental and continuous) in the world coordinate system.

Off $\Delta a = j * t$ The acceleration change time can be calculated from $t = \frac{\Delta a}{j}$
or

$$t = \frac{K-0-0084}{K-0-0086}$$

The maximum deceleration jerk in the world coordinate system is written in K-0-0087

The jerk limit values for axis-related jogging are written in A-0-0086 and A-0-0087.

14.3.47 K-0-0087, Jogging, deceleration jerk in WC system

Function This parameter stores the maximum deceleration jerk to jog (incremental and continuous) in the world coordinate system (indirect definition of the deceleration change time).



With the MLC 04VRS, the parameter can only be written in phase 2.

- Structure**
- List parameter:
One entry for every coordinate axis (x, y, z, alpha, beta, gamma).
 - Unit:

RCL Parameters

Depends on the transformation type K-0-0031 and the axis used (see K-0-0008).

The units are stored in K-0-0053.

Example:

for x, y, z in mm/s³ or inch/s³ (see K-0-0054),

for alpha, beta, gamma in Grad/s³.

- Value range: ≥ 0.0 (floating point number)
- Default: 0.0 (no jerk limitation).

Use The RobotControl uses this parameter to jog (incremental and continuous) in the world coordinate system.

Off $\Delta a = j * t$ The deceleration time can be calculated from $t = \frac{\Delta a}{j}$ or

$$t = \frac{K-0-0085}{K-0-0087}$$

The maximum acceleration jerk in the world coordinate system is written in K-0-0086.

The jerk limit values for axis-related jogging are written in A-0-0086 and A-0-0087.

14.3.48 K-0-0100, Current position, Cartesian original coordinates

Function The parameter indicates the current position of the kinematics in the original coordinate system.

Depending on the type of kinematics, the current position consists of a list of Cartesian world coordinates, the tool orientation and the belt counters.

14.3.49 K-0-0102, Current path velocity

Function This parameter provides the current path velocity of the kinematics at the TCP.

Only the positioning axes and not the orientation and belt axes are considered.

The unit is mm/s or inch/s (see K-0-0054).

14.3.50 K-0-0103, Current path acceleration

Function This parameter provides the current path acceleration of the kinematics at the TCP.

Only the positioning axes and not the orientation and belt axes are considered.

The unit is mm/s² or inch/s².

14.3.51 K-0-0104, Current path jerk

Function This parameter provides the current path jerk of the kinematics at the TCP.

Only the positioning axes and not the orientation and belt axes are considered.

The unit is mm/s³ or inch/s³.

14.3.52 K-0-0105, Actual position, Cartesian workpiece coordinates

Function This parameter is reserved and has no functionality.

14.3.53 K-0-0106, Coordinate values of tool

Function The parameter indicates the coordinate values of the active tool, the three translations X, Y Z and the three rotations O1, O2, O3.

14.3.54 K-0-0107, Coordinate values of workpiece

Function This parameter is reserved and has no functionality.

14.3.55 K-0-0121 to K-0-0136, Coordinate Names

The parameters K-0-0121 ... 0136 contain the names of the individual coordinates. When an RCL program is created in the IndraWorks environment, the compiler statement

```
;;WC_NAMES
```

is applied.

Please refer to "K-0-0121, Coordinate name C1" for the standard, detailed description.

14.3.56 K-0-0121, Coordinate name C1

Function This parameter contains the name of the first coordinate of the world coordinate system of the kinematics.



The following restrictions apply:

- maximum of 12 characters
- A...Z, a...z, 0..9, '_'
- first character must be a letter
- "K_1" is identical to "k_1"; upper-case and lower-case letters are not differentiated.

14.3.57 K-0-0140, Tool name

Function The parameter indicates the name of the active tool coordinate system. "none active" is displayed after startup.

14.3.58 K-0-0141, Workpiece name

Function This parameter is reserved and has no functionality.

14.3.59 K-0-0150, Interpolated command position

Function This parameter is reserved and has no functionality.

14.3.60 K-0-0152, Interpolated path velocity

Function This parameter is reserved and has no functionality.

14.3.61 K-0-0153, Interpolated path acceleration

Function This parameter is reserved and has no functionality.

14.3.62 K-0-0154, Interpolated path jerk

Function This parameter is reserved and has no functionality.

14.3.63 K-0-0155, Safe zones on/off

Function This parameter specifies which of the 32 safe zones are enabled or disabled.

RCL Parameters

Structure	Bit	Content
	0	Enabling the 1st safe zone
	1	Enabling the 2nd safe zone
	...	...
	31	Enabling the 32nd safe zone

Fig. 14-22: K-0-0155 Safe zones on/off

Use Parameter K-0-0155 can be directly modified in P4 (BB RUN). Switching to P2 and back to BB is not necessary.

When running a RCL program, it must still be remembered that enabling/disabling the safe zones can be delayed, since safe zones are checked during the block preparation.

Example:

K-0-0155: 0000 0000 0000 0000 0000 0000 0000 0100 ; the 3rd safe zone is enabled.

14.3.64 K-0-0156, Safe zone 1 min. point

Function This parameter determines the minimum point of the 1st safe zone.

A cuboid safe zone is determined with the maximum point from K-0-0157.

Structure The parameter is a list parameter with the following structure:

- Element 1: Position of the 1st axis
- Element 2: Position of the 2nd axis
- Element 3: Position of the 3rd axis

Use The parameter can be directly modified in P4 (BB RUN). Switching to P2 is not necessary.

If an RCL program is running, note that the changes in the safe zone can be delayed since the safe zone check is executed in the block preparation.

In addition, the maximum point has to be greater than the minimum point, since the safe zone check is otherwise disabled.

14.3.65 K-0-0157, Safe zone 1 max. point

Function This parameter determines the maximum point of the 1st safe zone.

A cuboid safe zone is determined with the maximum point from K-0-0156.

Structure The parameter is a list parameter with the following structure:

- Element 1: Position of the 1st axis
- Element 2: Position of the 2nd axis
- Element 3: Position of the 3rd axis

Use The parameter can be directly modified in P4 (BB RUN). Switching to P2 is not necessary.

If an RCL program is running, note that the changes in the safe zone can be delayed since the safe zone check is executed in the block preparation.

In addition, the maximum point has to be greater than the minimum point, since the safe zone check is otherwise disabled.

14.3.66 K-0-0156/K-0-0157 Safezone1 min./max. Point till K-0-0218/K-0-0219 Safezone32 min./max. Point

Function: Safe zones can define areas blocked for the robot motion. 32 safe zones are available for each kinematics.

Functional description: A motion command in the RCL program consists of a start point A and the programmed end point B.

In the block preparation, a linear distance AB is interpolated from the points A and B. It is checked additionally, whether the distance AB has one intersection or one boundary point in common with one side of the safe zone. This is checked for each motion command and for all workspaces assigned to the kinematics.

If the block preparation identifies a violation of a safe zone in the next block, the kinematics is traveled up to the next start point of the incorrect block and then, the RCL program is aborted. Additionally, the error message „F12C0433 Workspace violation“ is output.

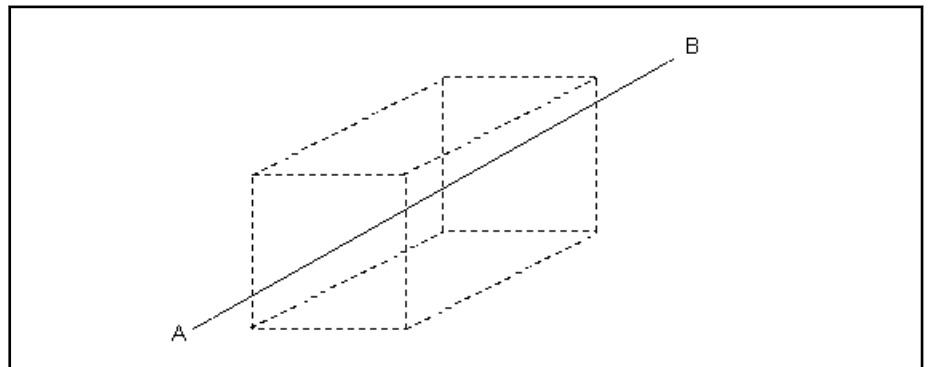


Fig. 14-23: Safe zone with minimum point (A) and maximum point (B)

Conditions: When programming the safe zones, the following has to be considered:

- Each kinematics considers only its individual safe zones.
- The safe zones consider only the TCP of the robot, not the mechanics.
- In case of the PTP interpolation, the monitoring of the safe zones is strictly restricted since the distance to be traveled is not defined. Deviations from the linear distance AB are not detected.
- In case of a CIRCULAR interpolation, the monitoring of the safe zones is strictly restricted. The checking of the safe zones is restricted on the linear distance AB as well. The circular path and the programmed interim point are not checked.
- When blending with VIA or APPROX, deviations from the linear distance AB are not detected. Deviations due to blending radii are not considered.
- The safe zone monitoring is only effective in the automatic mode. The safe zones are not monitored in the manual mode.
- In case of a belt-synchronous traveling motion, the belt-synchronous motion part is not considered when checking the safe zones.
- Since the safe zones are checked in the block preparation, it is not possible to dynamically adapt the safe zones. A change of the safe zones becomes only effective if the block buffer was processed. Therefore, a change or activation / deactivation of a safe zone can delay when the RCL program is running.

RCL Parameters

- The maximum point must be bigger than the minimum point, since the safe zone check would become inactive otherwise.

Use: The activation and adaptation of the safe zone is executed via the parameters K-0-0155 to K-0-0219.

The K parameters can be switched via the respective PLC FBs or directly via the parameter editor to P4 (BB RUN).

Switching to P2 is not necessary.

The parameter K-0-0155 activates the checking of the safe zones (from bit 1 for 1st safe zone up to bit 32 for 32nd safe zone).

Example: K-0-0155 0000 0000 0000 0000 0000 0000 0100 $\equiv$ decimal: 4 $\Rightarrow$ Safe zone 3 is active.

The following parameters K-0-0156 to K-0-0219 can be used to specify the maximum and minimum points by which cuboids are defined as safe zones.

K-0-0155 0000 0000 0000 0000 0000 0000 0100 decimal: 4 $\Rightarrow$ Safe zone 3 is active.
 K-0-0156 minimum point safe zone 1 (0, 0, 0),
 K-0-0157 maximum point safe zone 1 (20, 10, 10),
 K-0-0158 minimum point safe zone 2 (... , ... , ...),
 K-0-0159 maximum point safe zone 2 (... , ... , ...),

 K-0-0219 maximum point safe zone 32 (... , ... , ...).

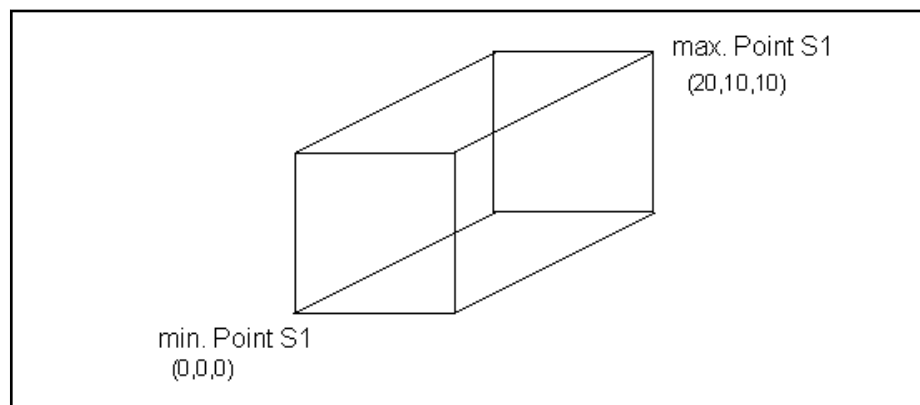


Fig. 14-24: Maximum and minimum point of safe zone 1

14.3.67 K-0-0220, File transfer name

Function The name of the RCL program is written in this parameter. It can be copied using the command parameters K-0-1060 and K-0-1061.

If the extension is .*, all files with the same program name - irrespective of the extension - are copied when executing one of the mentioned command parameters.

If the program name is specified as *.* , all files are copied when executing one of the mentioned command parameters.



The following restrictions apply:

- A maximum of 12 characters
- A...Z, a...z, 0..9, ' _ ' * *
- The first character must be a letter
- "Robi_1" is identical to "ROBI_1"

14.3.68 K-0-0221, File name to be compiled

Function The name of the QLL file to be compiled is in this parameter.
A suffix, i.e. .qll, is automatically removed. The suffix .QLL is always added internally.
If a compiler process is started using the command parameter K-0-1062, the QLL file specified in K-0-0221 is compiled.



The following restrictions apply:

- a maximum of 12 characters with suffix
- a maximum of 8 characters without suffix
- only A...Z, a...z, 0..9, '_' are permissible
- the first character must be a letter
- upper-case and lower-case letters are not differentiated, "Robi_1" and "ROBI_1" are identical

14.3.69 K-0-0240, Feedrate override

Function By means of this parameter, the target velocity can be changed as percentage of the runtime.
A value of 100% means that the axes are moved using the specified (programmed) velocity.
The percentage can be set in a range between 0% and 600%.



With 0%, the axes stop without changing the status of the kinematics status machine.

As long as the VFIX command is active in the RCL program, this parameter will have no effect

14.3.70 K-0-0241, Feedrate override control word

Function Using this parameter, the RoCoFeedrate override is controlled.
Structure

Bit	Designation/ function
0	reserved
1	Bit 1 = 0: RoCoFeed override in the K-0-0240 is effective, Bit 1 = 1: Feed halt, the value in K-0-0240 remains unchanged.
15..2	reserved

Use **Bit 1:**
By setting bit 1, the feed halt becomes effective.

14.3.71 K-0-0242, Acceleration override

Function By means of this parameter, the acceleration can be changed as percentage of the runtime.
A value of 100% means that the axes are moved using the specified (programmed) acceleration.
The percentage can be set in a range between 1% and 600%.

RCL Parameters



Setting of K-0-0242 takes effect in the block processing, i.e. not immediately.

As long as the AFIX command is active in the RCL program, this parameter will have no effect

14.3.72 K-0-0243, Deceleration override

Function By means of this parameter, the deceleration can be changed as percentage of the runtime.

A value of 100% means that the axes are moved using the specified (programmed) deceleration.

The percentage can be set in a range between 1% and 600%.



Setting of K-0-0243 takes effect in the block processing, i.e. not immediately.

Small values in K-0-0243 cause very long braking paths.

If the PROGR_SLOPE passing function is active in the RCL program, this may cause a jerky halt in case of a short braking path.

14.3.73 K-0-1060, Command: Load file from CF

Function This command parameter copies the file(s) specified in parameter K-0-0220 from the Compact Flash card to the control's internal memory.

The file source directories "<User-Partition> / RC_Files" are specified on the Compact Flash.

- Use**
- Enter the binary value "0000000000000011" (=0x0003) in this parameter. This triggers the copying process.
 - After the copy process is complete, the control automatically resets the parameter to "0000000000000000".

14.3.74 K-0-1061, Command: Save file to CF

Function This command parameter copies the file(s) specified in parameter K-0-0220 from the control's internal memory to the Compact Flash card.

The file target directories "<User-Partition> / RC_Files" are specified on the Compact Flash.

- Use**
- Enter the binary value "0000000000000011" (=0x0003) in this parameter. This triggers the copying process.
 - After the copy process is complete, the control automatically resets the parameter to "0000000000000000".

14.3.75 K-0-1062, Command: Compile file to MLC

Function This parameter compiles the QLL file specified in parameter K-0-0221.



The QLL file to be compiled must be saved in the SRAM/RAM.

The QLL file is loaded in the SRAM/RAM using parameter K-0-1060.

- Use**
- Enter the binary value "0000000000000011" (=0x0003) in this parameter. This triggers the compile process.
 - After the compile process is complete, the MLC automatically resets the parameter to "0000000000000000".

14.3.76 K-0-2200, Programmed position 1 (active)

Function This parameter is reserved and has no functionality.

14.3.77 K-0-2201, Programmed position 2 (active)

Function This parameter is reserved and has no functionality.

14.3.78 K-0-2202, Type of actively programmed position

Function This parameter is reserved and has no functionality.

14.3.79 K-0-2203, Interpolation type

Function This parameter is reserved and has no functionality.

14.3.80 K-0-2204, Currently active QLL file name

Function This parameter is reserved and has no functionality.

14.3.81 K-0-2205, Currently active line number

Function This parameter is reserved and has no functionality.

14.3.82 K-0-2300, PLC-Bit input (RCL)

Function The parameter indicates the bit inputs that are transferred from RCL toward the PLC.

14.3.83 K-0-2301, PLC-Integer input (RCL)

Function The parameter indicates the integer inputs that are transferred from RCL toward the PLC.

14.3.84 K-0-2302, PLC-Real input (RCL)

Function The parameter indicates the floating point inputs that are transferred from RCL toward the PLC.

15 Kinematic User Interface

15.1 General Information

IndraWorks provides specific user interfaces to implement a robot application. These consist of the following components:

- [RobotControl Menu Structure, page 203,](#)
The RobotControl menu structure is located in the MLC project tree. All toolbar icons are provided with help texts. These are displayed when holding the mouse cursor on the icon for a moment.
Additionally, information on the respective functions is shown in the status bar.
- **Kinematic wizard**
A kinematics is created via the kinematic wizard. Kinematic properties can be configured via this wizard.
- [RCL Editor, page 205,](#)
The RCL Editor creates and checks user programs in the RCL (Robot Control Language).
- [Compiler, page 210,](#)
The compiler compiles a source code to a machine-readable code. The output texts and the error messages are redirected to an output window and displayed. Double-click to jump to the position in the program.
- [Debugger, page 211,](#)
The debugger facilitates the finding of program errors. The values of previously declared variables can be displayed in the "Debug" mode and monitored during program execution.
- **Point editor**
The point editor enters, deletes or modifies points. The points are loaded from the point file (online or offline) and represented in a table. Points can also be declared as point arrays. DEF command are used for the declaration (see [chapter 7.3 "RCL Variable Declaration" on page 57](#)).
- **File transfer**
Files can be transferred to and deleted in the control.
- **Diagnostic overview**
All relevant status information on the respective kinematics is displayed in the "Status" dialog.

15.2 RobotControl Menu Structure

15.2.1 Menu Item: Robot

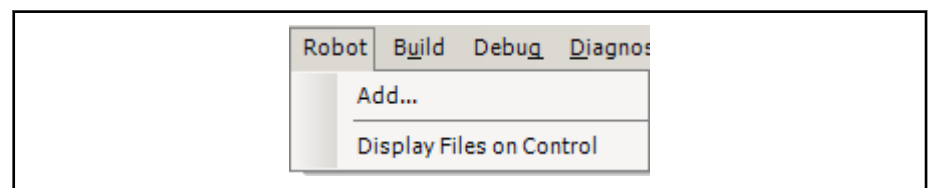


Fig. 15-1: Menu Item: Robot

Kinematic User Interface

The menu item **Robot** in the IndraWorks main menu is enabled by selecting the "Robot" node. Start the wizard to configure a new kinematics via **Add...** Display the RCL programs in the control via **Display Files on Control**. Open the previously described functions also via the context menu of the "Robot" node.

15.2.2 Menu Item: Kinematics

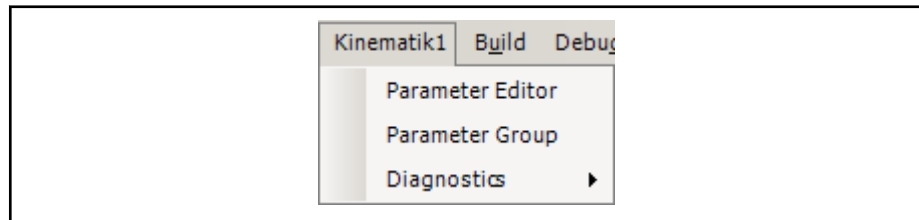


Fig. 15-2: Menu item: Kinematics

The "Kinematics" node is enabled after the selection of the kinematic name. The menu item in the IndraWorks main menu is then enabled.

15.2.3 Menu Item: Programs

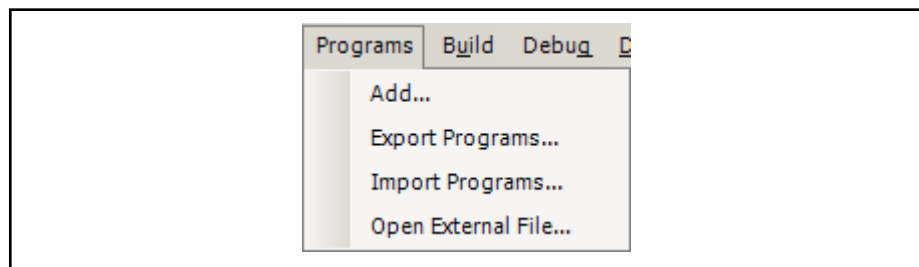


Fig. 15-3: Menu Item: Programs

Select the "Programs" node to enable the menu item **Programs** in the IndraWorks main menu.

Create a new RCL program via **Add...**

Store all RCL programs outside the project via **Export Programs...**

Import all RCL programs outside the project into the project via **Import Programs...**

Open an RCL program outside the project in the editor via **Open External File...**

15.2.4 Menu Item: Prog.QLL

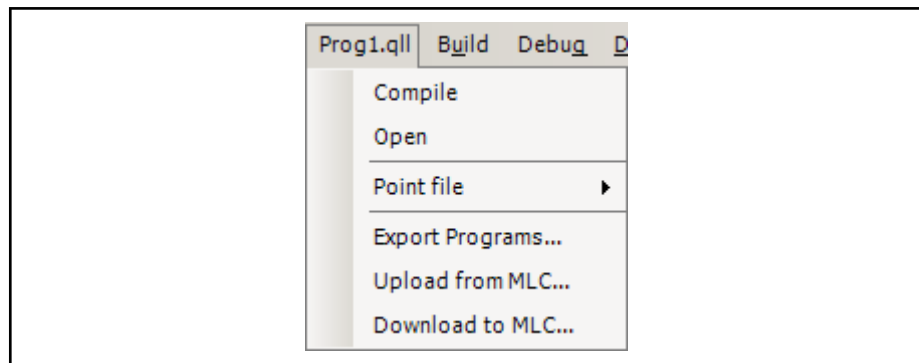


Fig. 15-4: Menu Item: Prog.QLL

Select the "Prog.QLL" node to enable the menu item **Prog.QLL** in the IndraWorks. The name of the created RCL program is displayed as name in the menu item or node.

Create an executable robot program in the source text (.QLL) via **Compile**,
Open the selected RCL program in the editor via **Open....**

Via **Point File**, open or store a point file using the submenus of a point file.

Store the selected RCL program outside the project via **Export Programs....**

Upload the selected RCL program from the control to the project via **Upload from MLC....**

Download the selected RCL program to the control via **Download to MLC....**

15.3 RCL Editor

15.3.1 General Information

The text editor creates, checks and commissions user programs in the programming language RCL (**R**obot **C**ontrol **L**anguage).

Options can set the behavior, appearance, colors, line numbering, tabulator widths, indentations, etc.

15.3.2 Characteristics

Special editor characteristics:

- Syntax highlighting (can be freely configured in the "Options" dialog)
- Copy, Paste, Find, Replace, etc.
- Bookmark management
- Line numbering
- Tooltip
- Automatic keyword specification
- Code folds
- Intellisense
- Code templates
- Debugging

Kinematic User Interface

15.3.3 Options

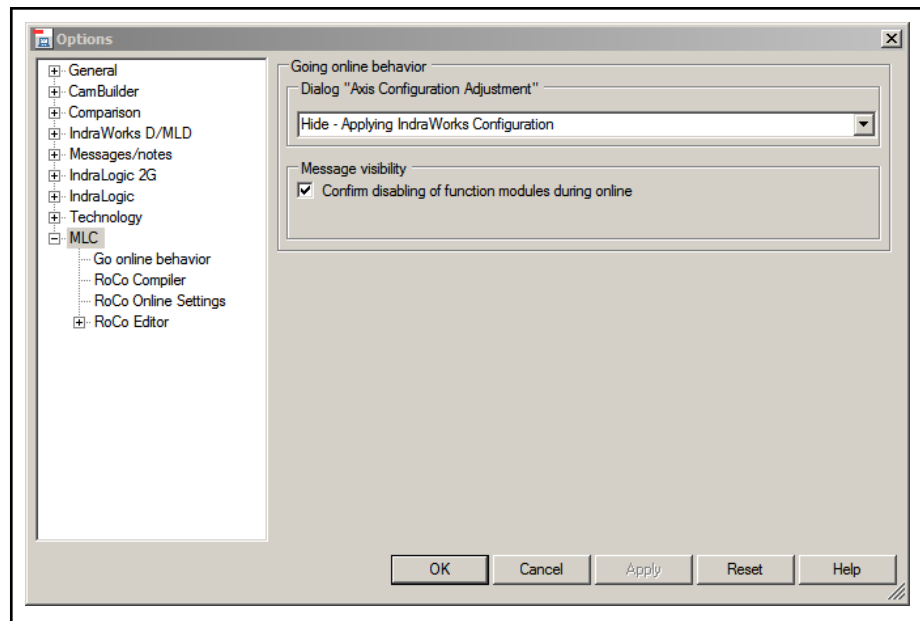


Fig. 15-5: RoCo Editor "Options" dialog

Functions to support the source text input are preset by default under the IndraWorks menu item **Tools ► Options** in the tree menu under "RoCo Editor". They can be changed by the user.

15.3.4 Creating a New RCL Program

Create a new RCL program via the following menus:

- In the IndraWorks main menu via **Programs ► Add**
- or -
- via the program context menu item **Add...** of the kinematics

The program name given by default can be overwritten. Confirm the entry with OK. The RCL Editor, containing the program body of the new RCL program, starts. The program is shown in the menu item after its initial saving.

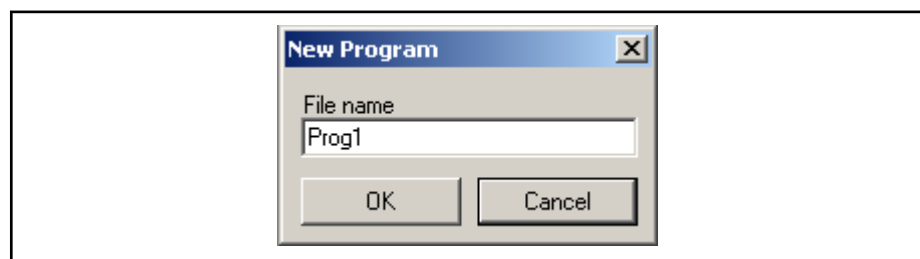


Fig. 15-6: "Add New Program" dialog

15.3.5 Working with the Editor

General editor functions

The RoCo Editor integrated useful techniques of common Windows editors:

- Find
- Find files
- Replace
- Block highlighting
- Match case

- Drag&drop

Find and replace

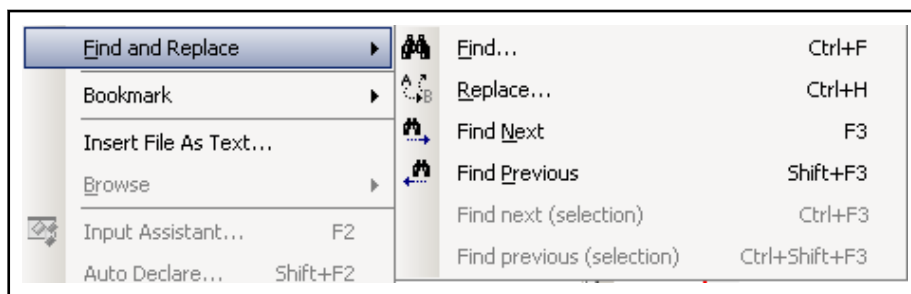


Fig. 15-7: Menu item: "Find and Replace"

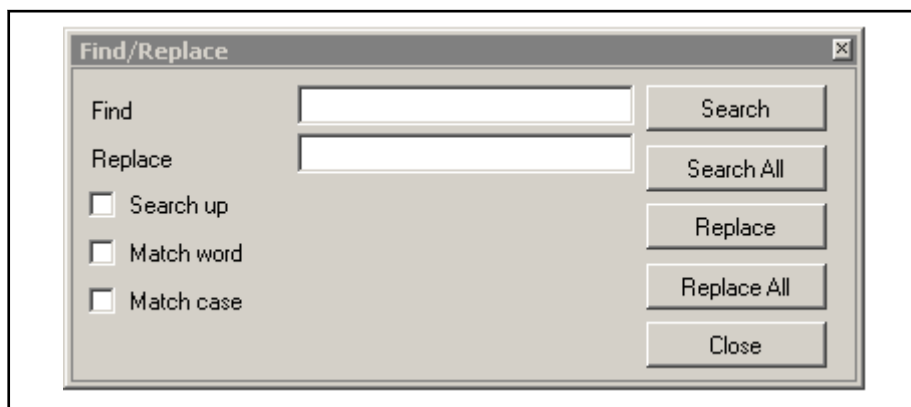


Fig. 15-8: Dialog "Find/Replace"

Bookmark

The text editor provides bookmark management. Add bookmarks to lines to facilitate navigation in long programs. The commands described below allow the setting and navigation of bookmarks. They are included by default in the "Bookmarks" submenu in the "Edit" menu. Additionally, they can be found via "Bookmarks" in the toolbar.

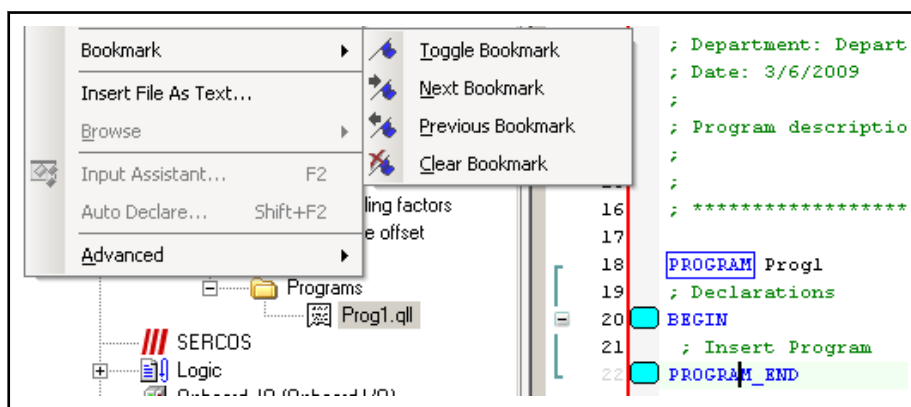


Fig. 15-9: Menu item: "Bookmark"

Comments

Comments can be inserted. The comment starts with ";" and ends at the end of the line.

Kinematic User Interface

Cut, copy, paste Highlight the area in the editor with the left mouse button or via the Shift cursor shortcut. The commands "Cut", "Copy", "Paste" can then be used in these areas.

Find the commands in the "Edit" menu or via the context menu of the editor.

Supporting source code input

- **Parameter information**

If the mouse pointer is moved across a subroutine, the transfer parameters are displayed.

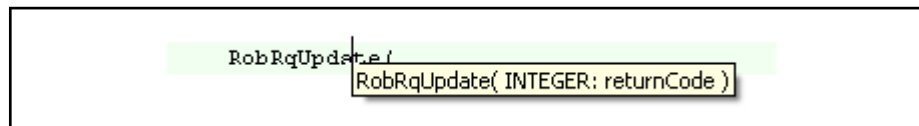


Fig. 15-10: Parameter information

If a subroutine name and an open parenthesis are entered, the transfer parameters are also displayed.

- **Code completion**

This function supplements the user input. The input options reduce with the progress and might also with the semantics of the user input. Enter the first character or press <CTRL> + <Space> to display a selection list containing the RoCo command block as well as the already declared variables. Apply the object selected in the list with <Return>.

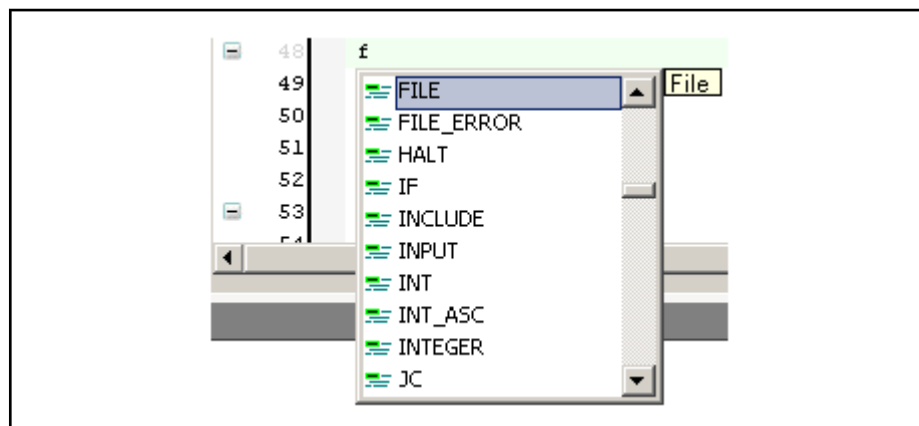


Fig. 15-11: Code completion

- **Element listing**

Enter a link variable and a subsequent point and the elements of the link variables are displayed for selection. Select the desired element using the arrow keys. Apply the selection with <Return>.

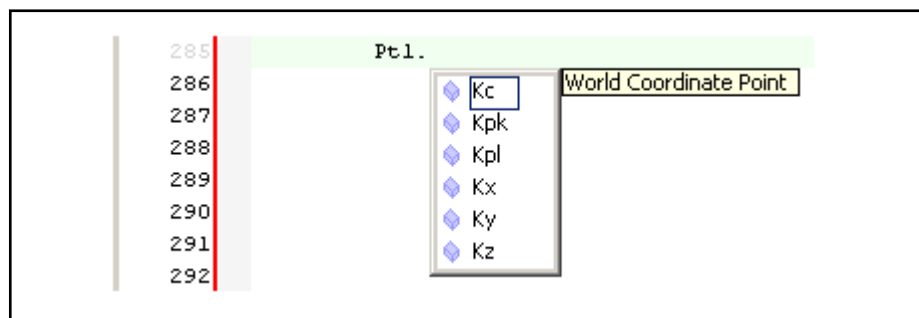


Fig. 15-12: Element listing

- **Type information**

Move the mouse cursor across the variable and the type information is shown as information.

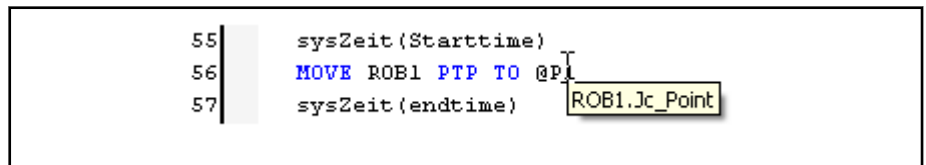


Fig.15-13: Type information

- **Indenting the cursor automatically**

When writing a source text, the indentation of the previous line is automatically applied to the next line.

- **Tabulators**

Set the length of the tabulator jumps in the "Options" dialog.

- **Syntax highlighting**

The keywords in the source text are highlighted. Change the initial specifications in the "Options" dialog.

- **Special functions/code templates**

Entering special functions for the code completion is supported.

Enter the RoCo command for special functions and it is selected in the drop-down list.

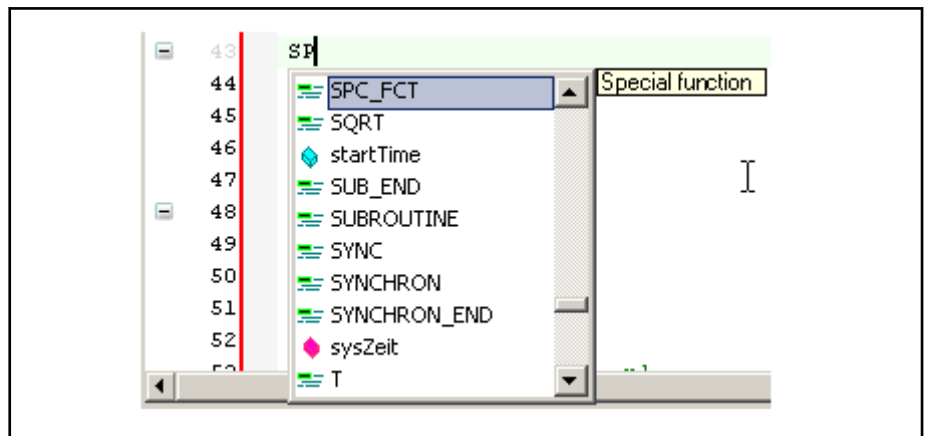


Fig.15-14: Selecting the "Special Function" command

Apply the object selected in the list with <Return>. A drop-down list displays the special functions available for selection.

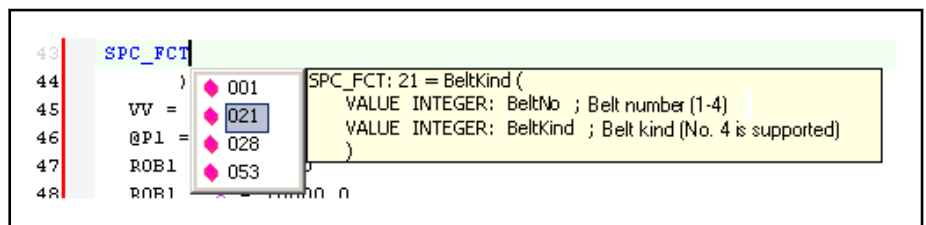


Fig.15-15: Selecting the special function

<Return> applies the special function selected in the list. The default transfer parameter names can be changed.

Kinematic User Interface

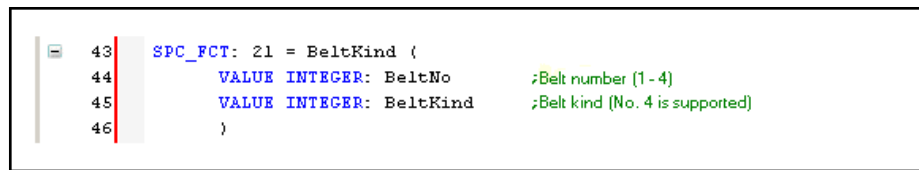


Fig.15-16: Inserted special function

15.4 Compiler

The compiler creates the program in a machine-readable form from the source text and checks each single line for a correct command syntax. The source text can be compiled in the editor if an RCL program is selected.

Start the compilation as follows:

- via the main menu

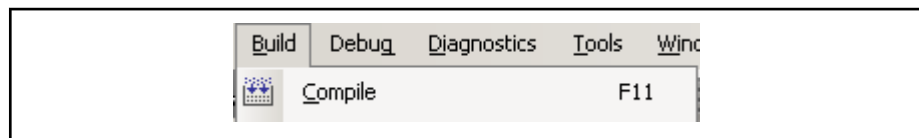


Fig.15-17: Selecting the compiler

- via the context menu in the editor

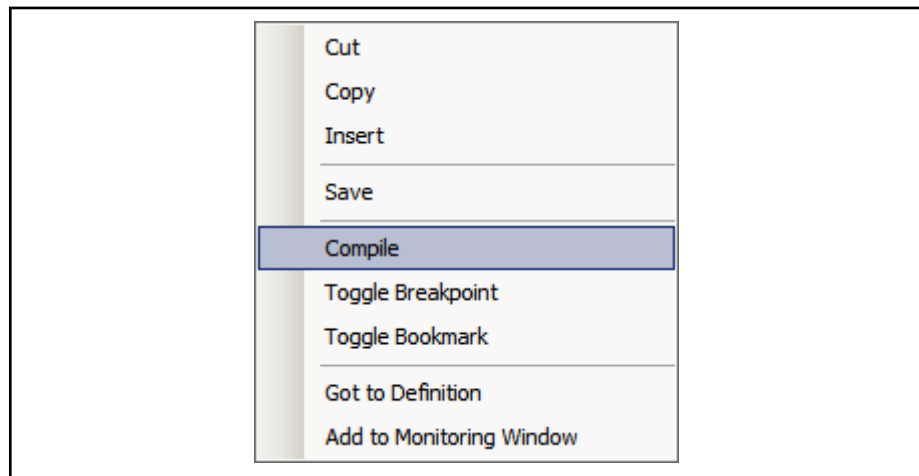


Fig.15-18: Selecting the compiler

- via <F11>

The selected file is compiled in machine language and saved in the project. Enable an additional immediate saving in the control via the "RoCo Compiler" option.

The output texts as well as error messages are redirected to the dockable "Output" window.

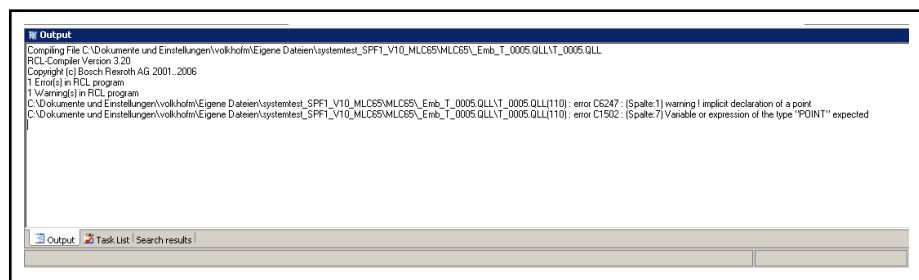


Fig.15-19: Output window of the compiler

Kinematic User Interface

If error messages are displayed in the output area, double-click on the error message in the **Task List** window to directly jump to the position in the program where the error was identified.

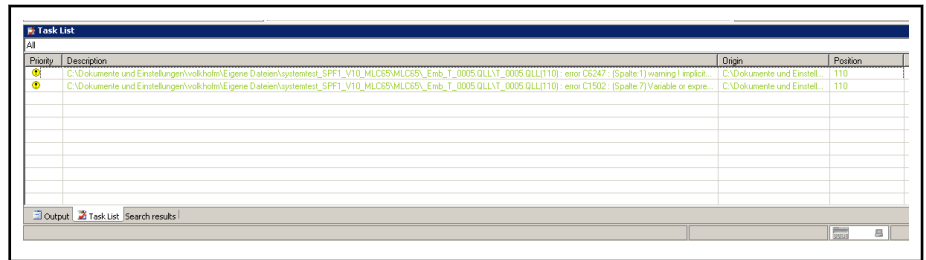


Fig. 15-20: Outputting compiler error messages

15.5 Debugger

15.5.1 General Information

Check the execution of a program in the Debug mode for logic programming errors. The content of the variables is displayed in the dockable "Monitoring" window.

15.5.2 Setting and Deleting Breakpoints

Set breakpoints to interrupt the running program at a specified location.

Set the cursor to the location in the program where the breakpoint should be set.

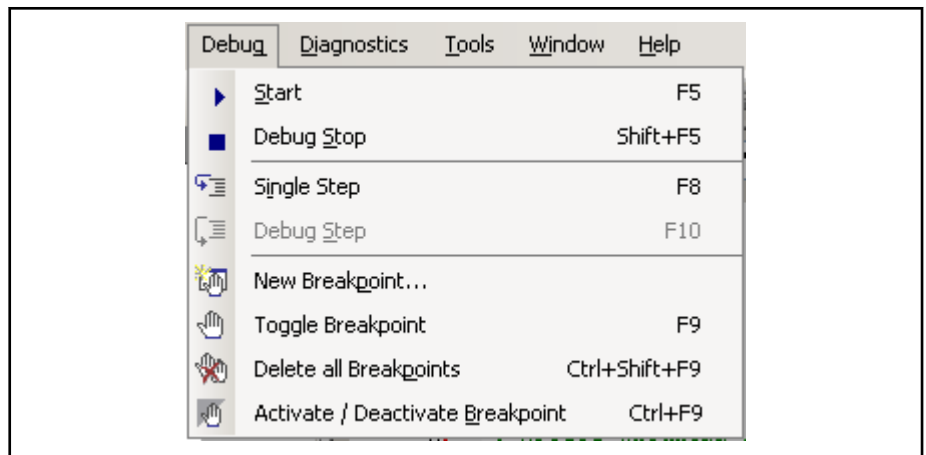


Fig. 15-21: "Debug" menu

Click on the **New Breakpoint...** function in the **Debug** menu to set the breakpoint or to delete it again. Set and delete a breakpoint using the **Toggle Breakpoint** function or <F9>. Breakpoints can also be set and deleted by clicking on the indicator bar (grey margin on the left of the source text window).

Kinematic User Interface

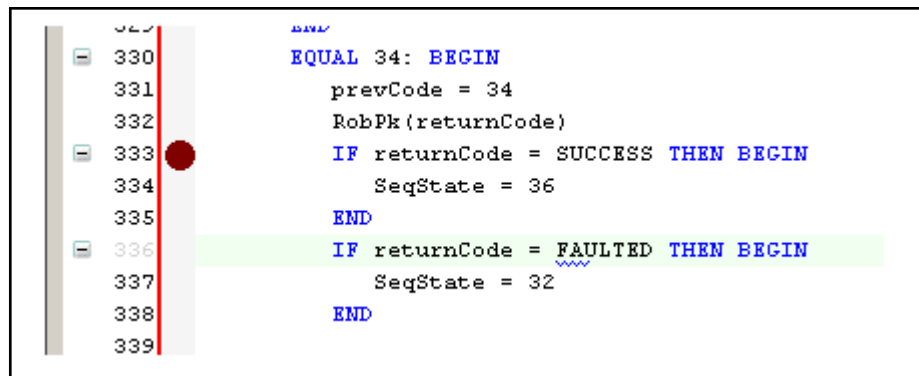


Fig. 15-22: Set breakpoint

Delete all breakpoints using the **Delete all Breakpoints** function in the **Debug** menu or the shortcut <CTRL>+<Shift>+<F9>.

15.5.3 Debugging a Program

Execute the program in the debug mode via **Debug ▶ Start** or <F5>.

⚠ WARNING

Directly start the program via Debug ▶ Start.

Take precautions to ensure that this is safe. Set a breakpoint to the first command in the RCL program for example.

It is checked before the start whether the program has been changed on the PC or control. If changes are detected in the point file for example, a message is displayed. Then, select either to transfer the point file to the control or to close the debugger.



Date and time of PC and control have to match to ensure a correct monitoring mode of operation.

Refer to the menu item **Diagnostics ▶ Device status** in the context menu of the control in the IndraWorks Project Explorer.

Add the debugger to an active program via the  icon in the RoCo toolbar.

The program sequence is interrupted at first breakpoint.

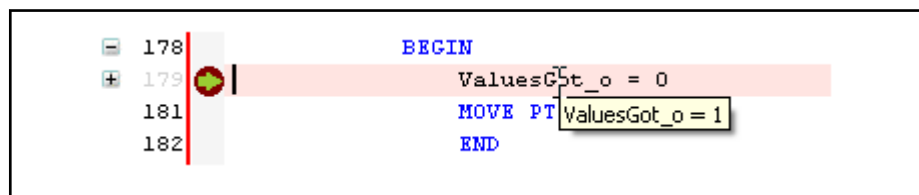


Fig. 15-23: Program sequence stops at first breakpoint

Continue the program after a stop at a breakpoint with <F5>. Test the program in single steps with <F8>. Close the debugger with <Shift> + <F9>.

Set the cursor on a variable to show the current variable value in the tooltip.



The variable values are only displayed if the program is on a breakpoint.

15.5.4 Monitoring Variables

Apply variables to the monitoring window in the "Debug" mode. Enter the variables directly entered into the window or store them there via drag&drop.

The  icon in the RoCo toolbar opens the monitoring window.

 The variable values are only displayed if the program is on a breakpoint.

16 Using the RoCo Multi-Kinematic Technology

16.1 General Information

- One MLC control supports up to 16 kinematics (robots).
- Different robot types can be controlled side by side in one MLC (one robot type, e.g. Cartesian, SCARA, Delta, etc., for each kinematics).
- One kinematics supports up to 16 axes (incl. belt axes).
- One kinematics supports up to 4 belt axes.
- Several robots can work synchronously on one belt (belt axes can be assigned to several kinematics).
- The operating mode can be selected individually for each kinematics (Programmed Motion, Manual Motion, Discrete/Synchronized Motion).
- Each kinematics has its own PLC shared memory interface (KinData).

16.2 Creating Kinematics

16.2.1 General Information

In the parameter C-0-2484, Kinematic configuration list, the kinematics are created via the menu items **Robot** ► **Add** when IndraWorks is offline.

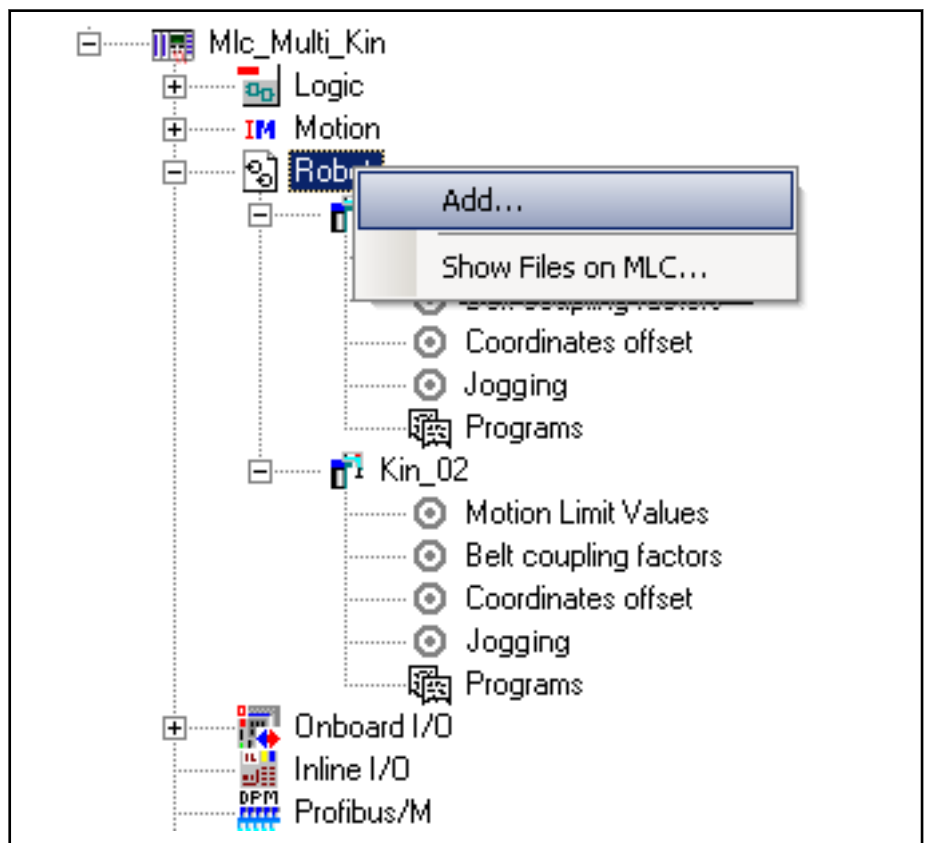


Fig. 16-1: Creating a kinematics

When creating a kinematics, a consecutive number is automatically assigned. Specify a unique symbolic identifier (device name) in the kinematic wizard. The wizard starts automatically when adding,

Using the RoCo Multi-Kinematic Technology

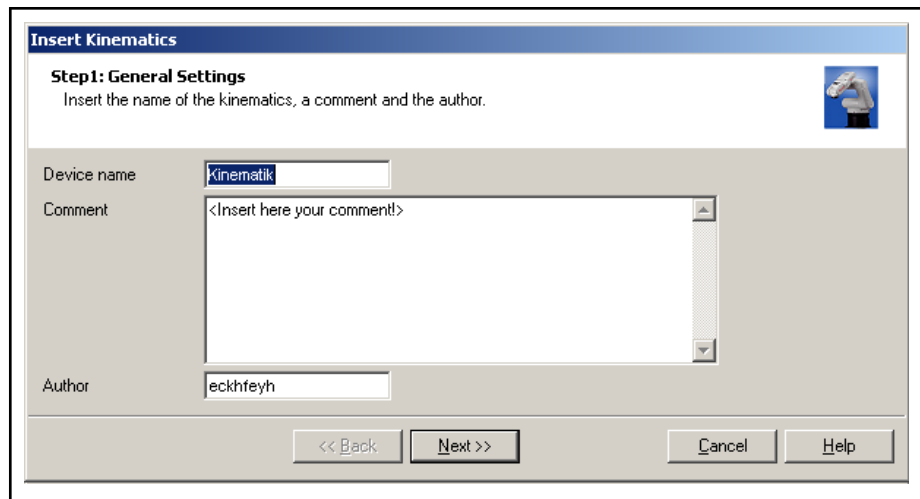


Fig. 16-2: Entering the kinematic name

The PLC program uses the kinematic name to access the kinematics irrespective of the assigned number. Furthermore, number and name of the kinematics are entered the RCL program head when an RCL program is created. This entry can ensure later on that only programs written for this kinematics can run on this kinematics.

16.2.2 Defining the Kinematic Type (Robot Type, K-0-0031)

Define the kinematic type via **Next**.

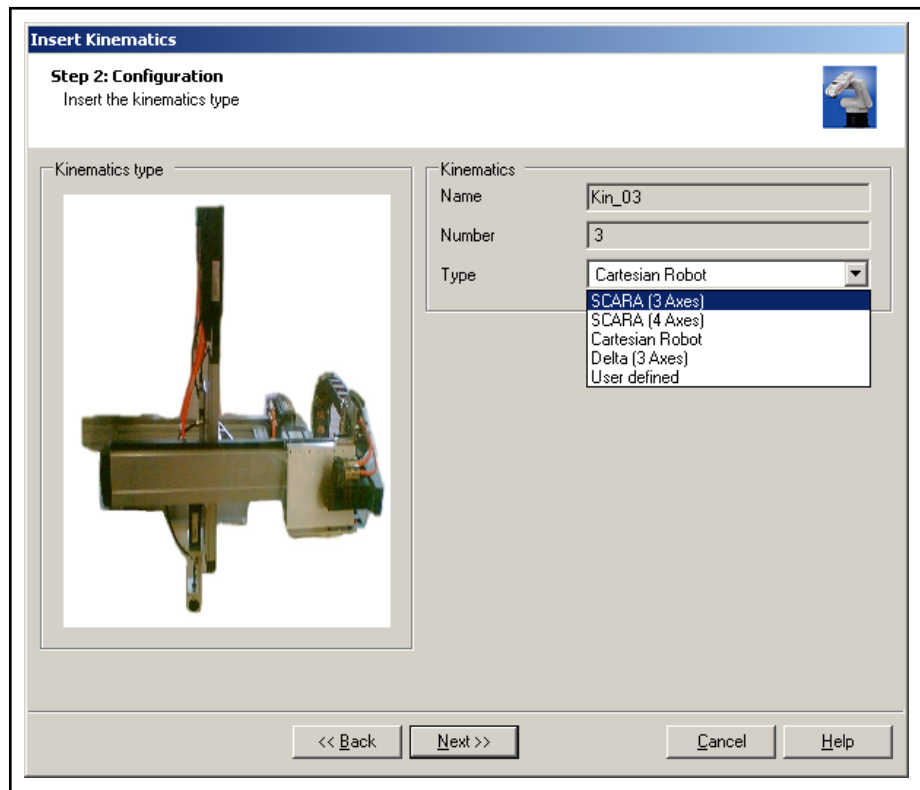


Fig. 16-3: Defining the kinematic type

All types (SCARA, Cartesian Robot, Delta, etc.) are always available here. Any robot types can be operated together in one MLC.

16.2.3 Defining the Axis Assignment (K-0-0008)

An MLC control can control up to 16 robots (kinematics) at a time with up to 16 axes each. Virtually, it can sum up to 256 axes. However, this theoretical number of axes is limited by the maximum number of axes of the control.

- On the L40.2, a maximum of 32 axes can be operated. This is a maximum of 16 kinematics with 2 axes each.
- On the L65, a maximum of 64 axes can be operated. This is a maximum of 16 kinematics with 4 axes each.

The axes available in the control are listed in the parameter C-0-0484, Axis configuration list. In the parameter K-0-0008, Axis configuration list, these axes can be assigned to their respective kinematics. The kinematic wizard provides only axes from C-0-0484, which are not yet assigned to a kinematics, as positioning and orientation axes. The wizard starts automatically when a kinematics is created.

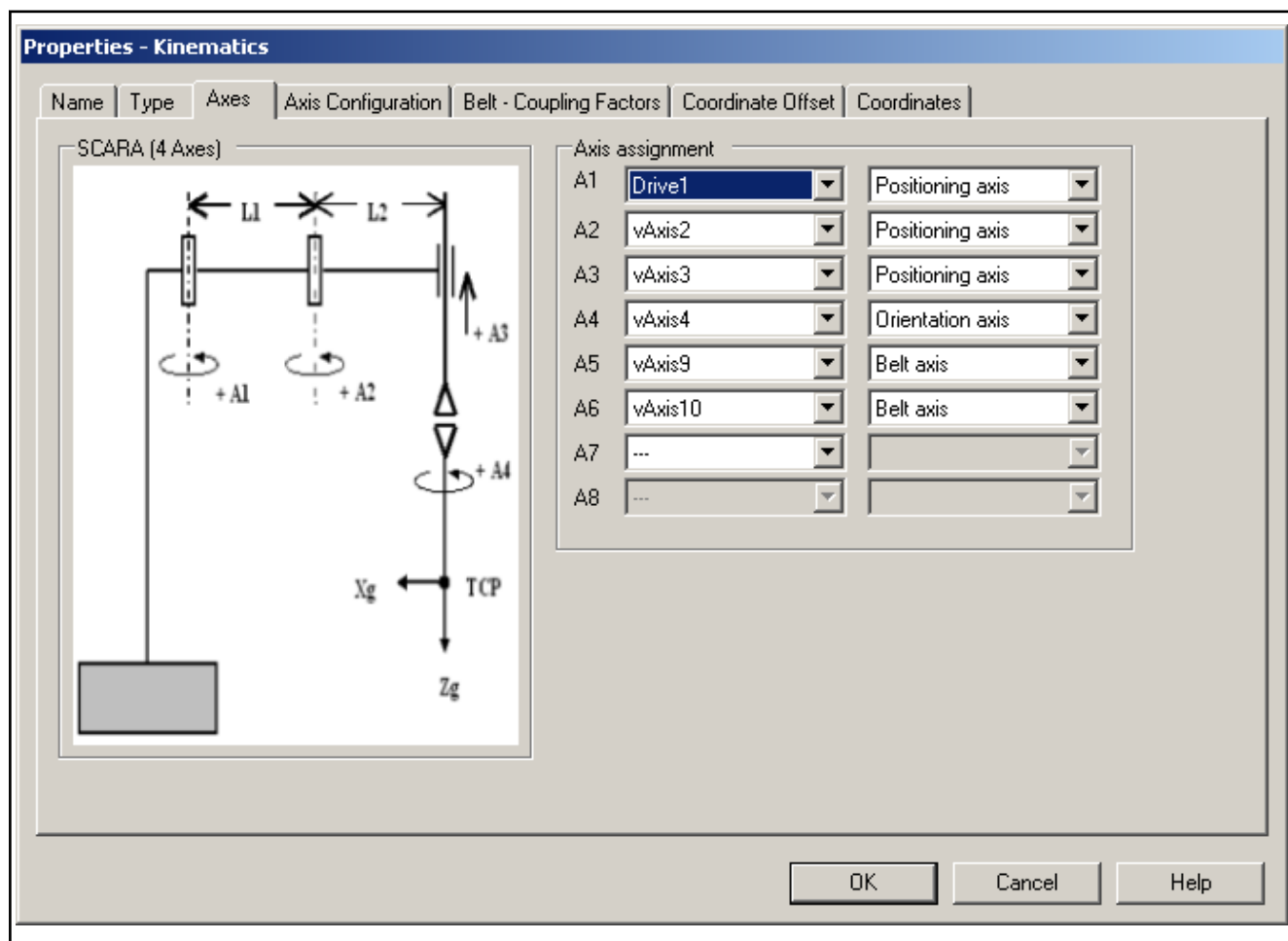


Fig. 16-4: Axis assignment for various kinematics (in this case, 4-axis Scara)

Using the RoCo Multi-Kinematic Technology

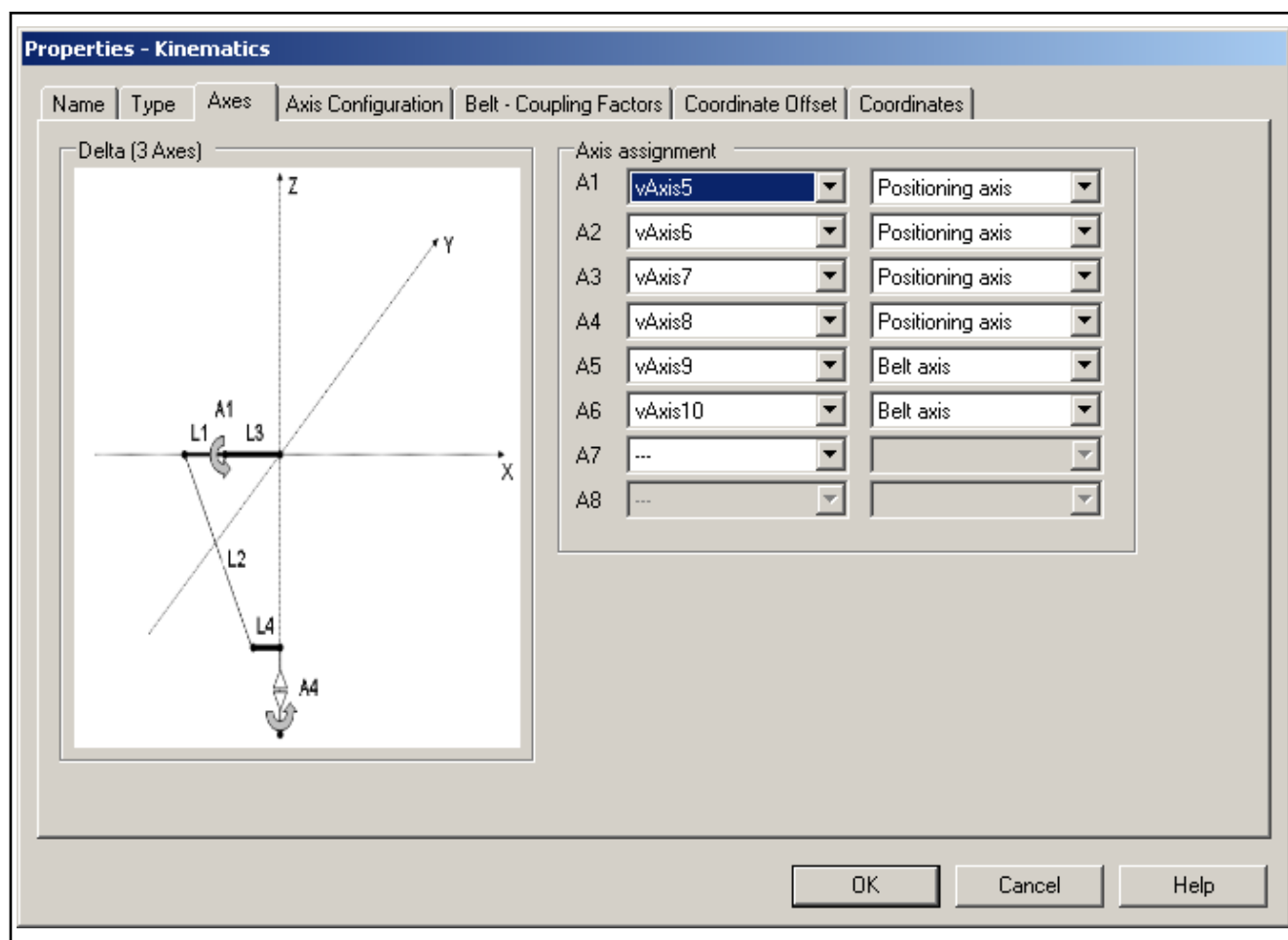


Fig. 16-5: Axis assignment for various kinematics (in this case, Delta pack)

This ensures that no axis (except for belt axes) can be assigned to more than one kinematics.

When switching from the parameter mode (P2) to the operation mode (BB or P4), the axis assignment in the K-0-0008 parameters is checked again. In case of an error, F0160040, Error in cross-check of K-parameters, is output. In the event of an error, the parameter K-0-0008 can be found in K-0-0011, List of all invalid K-parameters.

Belt axes can be simultaneously assigned to several kinematics. This feature allows several different robot types to move synchronously and independently on one or more belts. Independently means that each kinematics manages its own belt counter.

Thus, the belt blending can be operated separately for each kinematics. The other kinematics do not have to be stopped. However, to detect the product position, all belt counters for the kinematics always have to be latched - unless it is already determined by which robot the product is picked up.

16.2.4 Deleting Kinematics

A kinematics can also be deleted from the command configuration of the IndraWorks project.

Using the RoCo Multi-Kinematic Technology

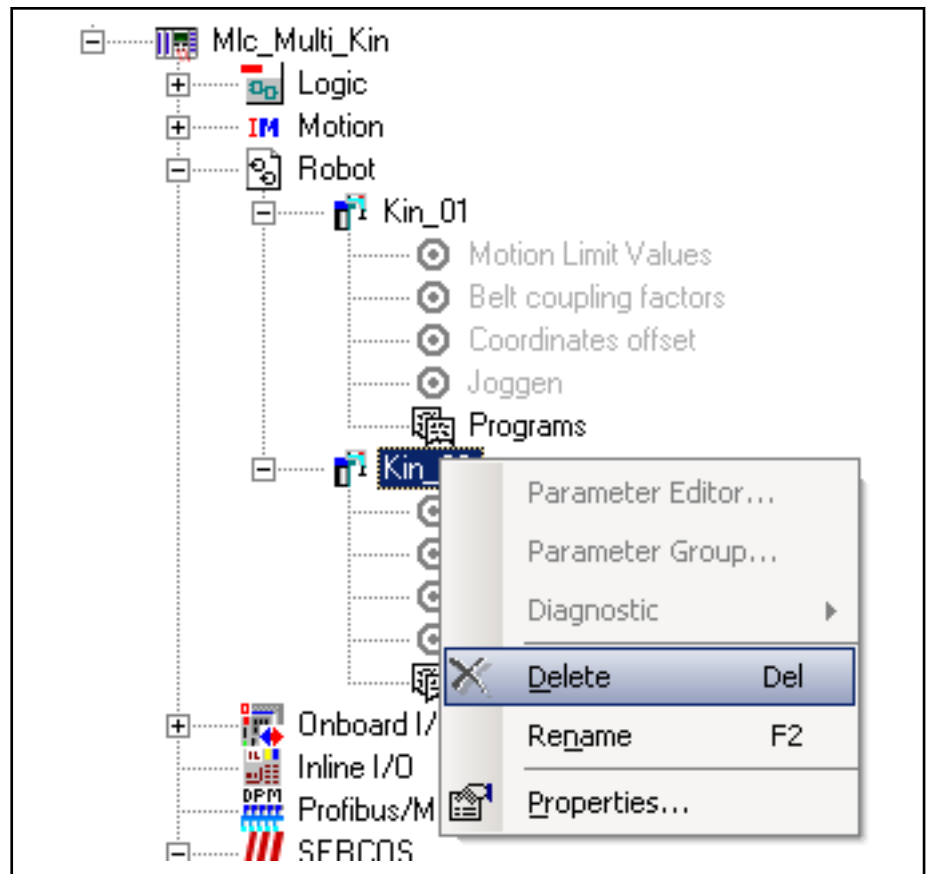


Fig. 16-6: Deleting a kinematics



In contrast to the axis configuration list C-0-0484, the list in C-0-2484 may not contain any gaps.

Thus, the kinematics are always continuously addressed from 1..n (n = number of kinematics used). This simplifies the work for the firmware and the PLC program.

To delete a kinematics from the command configuration in IndraWorks, reload and assign the previously saved parameters to the remaining kinematics accordingly.

16.3 Coordinated Motion of the Kinematics

The axes of a robot can be moved

- "directly" from the PLC program via the PLCopen function blocks such as MOVE_LINEAR **the PLCopen function blocks are still in the development phase**
- by executing an RCL program (one RCL program for each kinematics).
- by setting the JOG bits in the respective KinData interface (manual motion).

Using the RoCo Multi-Kinematic Technology

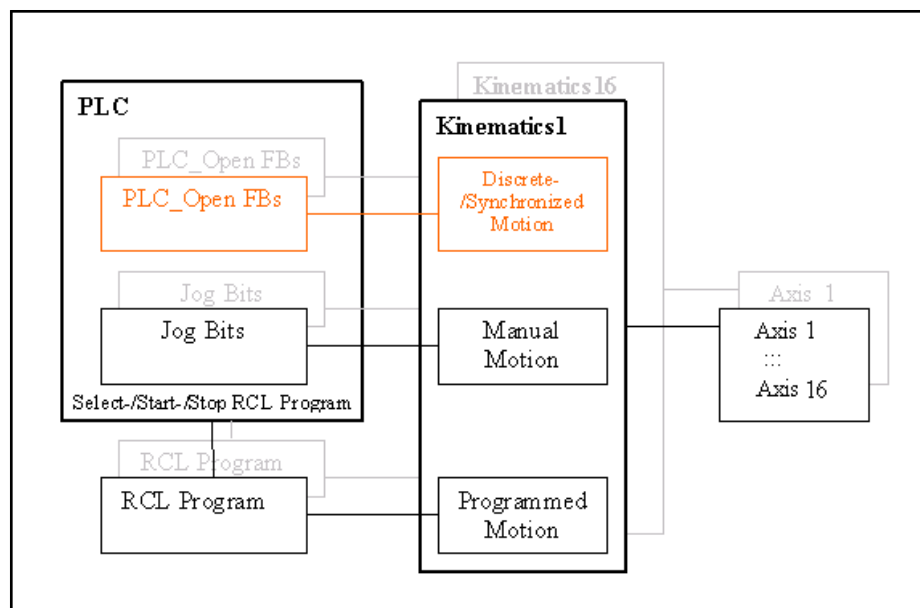


Fig. 16-7: Control options for axes in kinematics

All kinematics mostly run independently of each other, i.e. kinematics 1 can be in Manual Motion, kinematics 2 can be in Programmed Motion and kinematics 3 can be in Discrete Motion.

In other words: while one robot executes an RCL program and the motion of another is directly controlled by the PLC program with the PLCopen function blocks, a third robot can be jogged manually.

16.4 RCL Programs of the Kinematics



An RCL program can move exactly one kinematic. The RoCo is based on the following philosophy:

1. the RCL program is responsible for the robot's motion
2. the PLC program takes care of all other tasks (communication, coordination, system/process control...)

If a new RCL program is created using IndraWorks (**Programs** ► **Add**, etc.), head and body of the RCL program are automatically generated.

Using the RoCo Multi-Kinematic Technology

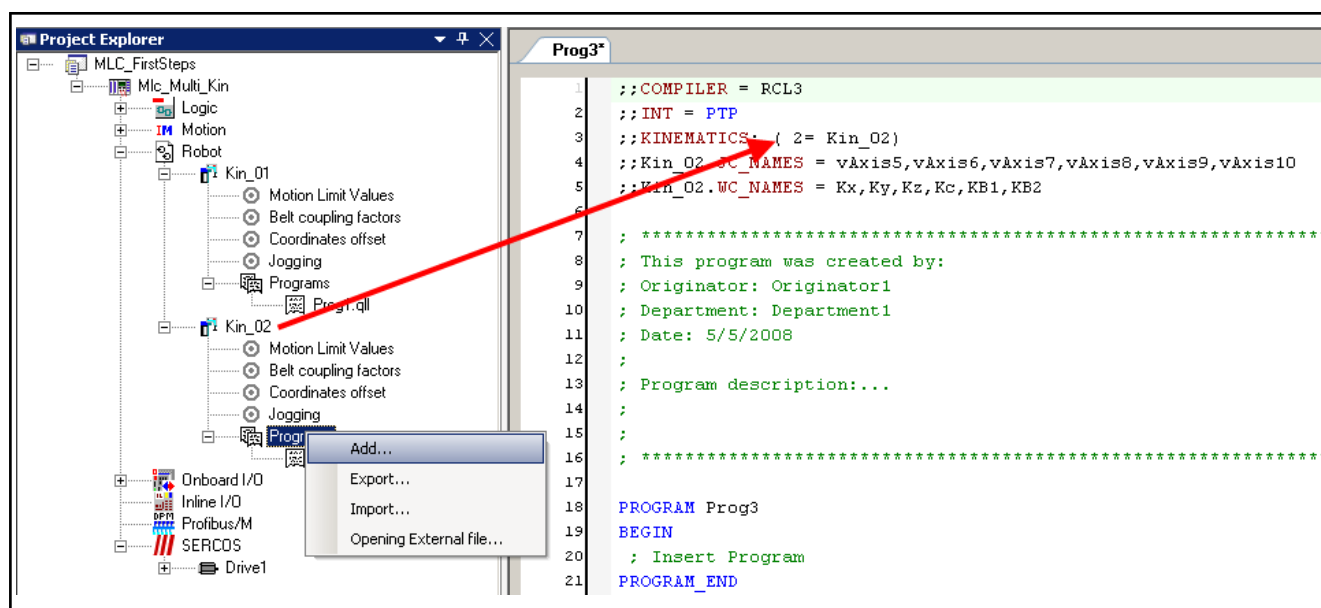


Fig.16-8: Creating a new RCL program

The entry KINEMATICS in the RCL compiler statement, e.g.

```
;;KINEMATICS: ( 2= Kinematic2)
```

defines the kinematics for which this program is written.

Subsequently, if it is tried to start an RCL program that was not written for this kinematics, the error F12C0072 is output and the kinematics state machine goes into the state KIS_Error. At first glance, this seems to be the correct response.

However, due to this response, an RCL program cannot run simultaneously on several robots. If there are several identical robots in a row, all of them can virtually run with the same RCL program. However, this can only be the case if a program is created for each kinematics and the source code is copied accordingly. Otherwise, it cannot be ensured that the RCL program can actually run on the kinematics.

In the MLC RoCo system, an RCL program can only be started once at a time. This is primarily the case due to the variable management. The variables are overwritten if the program starts more than once.

16.5 Point Tables of the Kinematics

16.5.1 General Information

One RCL program can have one point table.

From the perspective of block preparation, a point table is only another storage location for the point variables. An RCL program cannot be linked to a different (external) point table. Defines the names of the point variables. They are then permanently linked to one RCL program.

Using the RoCo Multi-Kinematic Technology

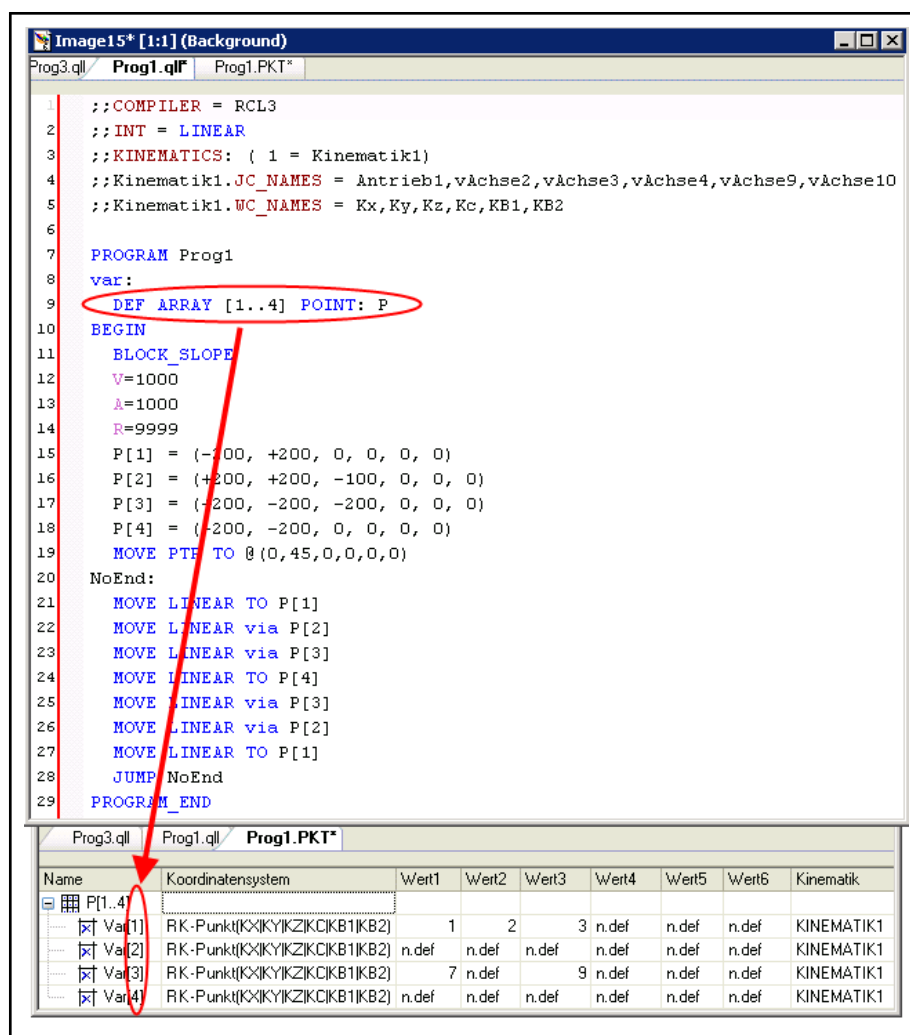


Fig.16-9: Assigning kinematics and point table

Definition of a point can be different in each kinematics (number of degrees of freedom = number of axes). The degrees of freedom for the points are defined when creating an RCL program using with the compiler statements JC_NAMES and WC_NAMES. Thus, the definition depends on the axis configuration in the parameter K-0-0008.

16.5.2 Point Tables, Assignment and Response

This results in the following strict assignment

- A point table is always permanently assigned to one RCL program.
An RCL program can have only one point table (not several!). The RCL compiler generates/modifies the point table and ensures that the IRD program can read or write the point variables from the point table.
- An RCL program cannot generate new points in the point table at run-time.
- An RCL program is assigned to one kinematics (see section [RCL programs, page 220](#)).
- One kinematics can have several RCL programs.
- The RCL program name has to be unique in the entire MLC RoCo system, i.e. the program "Prog1.qll" has to be the only Prog1.qll - a second Prog1.qll may thus not exist in different kinematics.

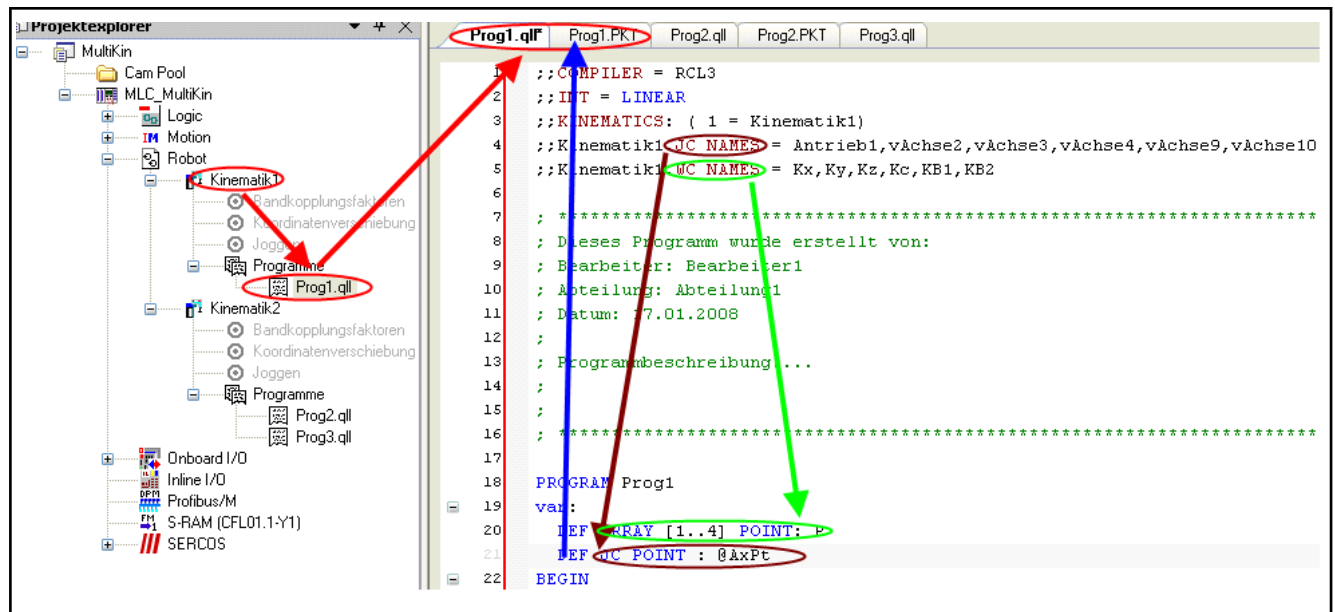


Fig. 16-10: Assigning kinematics, point table and coordinates

The "POINT" data type is defined by the compiler statement WC_NAMES (=structure with the elements Cx, Cy, Cz, etc., where each of the elements is of data type "REAL").

The compiler statement JC_NAMES defines the data type "JC_POINT" (= structure with the elements Drive1, vAxis2, etc., where each of the elements is of data type "REAL").

The number of elements in POINT and JC_POINT is always the same.

The RCL program "Prog1.qll" has exactly one point table, "prog1.pnt" with the name of the source program. The software engineer used the keyword DEF to define that these points are to be stored in the point table and not in the ird program. The program "Prog1" is permanently assigned to kinematics1. It exists only once in the entire MLC robot system.

16.6 PLC Shared Memory Interfaces (KinData) of the Kinematics

Each kinematics has its individual KinData interface to the PLC.

IndraWorks creates a variable of type "AXIS_GROUP_REF" for each kinematics under its name in the global variable list "GlobalMLCVariables" (as created for the axes).

This variable contains the GroupNo of the kinematics. Using this GroupNo, the KinData array can be indexed. This ensures that the PLC program always addresses the correct kinematics.

Using the RoCo Multi-Kinematic Technology

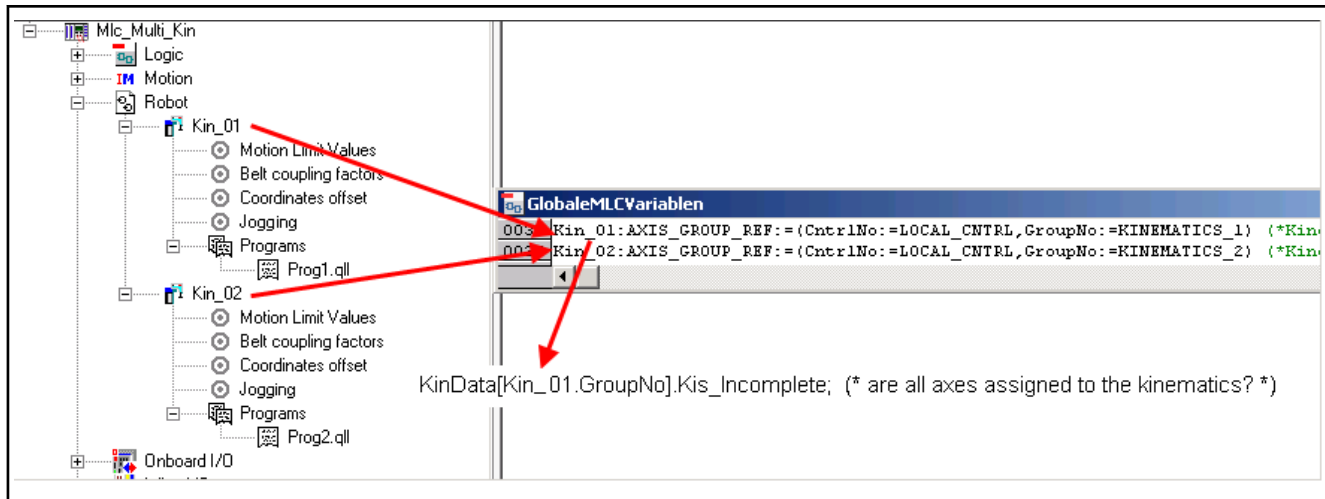


Fig.16-11: Access to data in the shared memory

17 RobotControl Functionality Programmed in the PLC

The RobotControl functionality can often be programmed in the PLC as well as in the RCL program.

However, it is strongly recommended not to use both access options at the same time.

PLC libraries

- ML_Robot, function blocks, functions and data types to implement the RobotControl functionality.
- ML_Base, data types for ML_Robot and ML_PLCOpen of the systems IndraMotion XLC and IndraMotion MLC
- RIL_CommonTypes, data types for all DC systems.

The libraries are all described in

Rexroth IndraLogic XLC IndraMotion MLC 12VRS PLCOpen Libraries, DOK-XLCMLC-FUNLIB**V12-L1xx-EN-P, R911334182.

Function modules

- S-RAM module (CFL01.1-Y1), memory extension for IndraMotion MLC

The function modules are all described in

Rexroth IndraControl Lxx 12VRS Function Modules, DOK-CONTRL-FM*LXX**V12-APxx-EN-P, R911333833.

Parameters (XLC, MLC)

- Overview: Parameters (XLC, MLC)
- Overview: RCL parameters
- A-parameters (axis parameters)
- C-parameters (control parameters)
- K-parameters (kinematic parameters)

The parameters are described in

Rexroth IndraLogic XLC IndraMotion MLC 12VRS Parameters, DOK-XLCMLC-PARAM***V12-RExx-EN-P, R911334186.

Diagnostics (XLC, MLC)

- Overview: Diagnostics (XLC, MLC)
- F1 error, axis group error

The diagnostics are described in

Rexroth IndraLogic XLC IndraMotion MLC 12VRS Diagnostics, DOK-XLCMLC-DIAG***V12-RExx-EN-P, R911334180.

...an example can be found in:

- First Steps - RobotControl Using MLC

Rexroth IndraMotion MLC 12VRS First Steps, DOK-MLC***-F*STEP**V12-COxx-EN-P, R911333846.

18 Transformation Rules

18.1 Transformation Rules, Overview

This chapter contains all transformation rules available for the programming language RCL (Robot Control Language).

- [Robot Type 00, Cartesian Robot, page 227,](#)
- [Robot Type 07, 2-Axis Delta with 1 Optional Manual Axis, page 231,](#)
- [Robot Type 09, 3-Axis Scara with 1 Optional Manual Axis, page 234,](#)
- [Robot Type 10, 4-Axis Scara with 1 Optional Manual Axis, page 236,](#)
- [Robot Type 12, 2-Axis Parallelogram, page 239,](#)
- [Robot Type 13, 2-Axis Parallelogram, page 241,](#)
- [Robot Type 18, Contact Kinematics, page 243,](#)
- [Robot Type 19, 2-Axis Parallelogram V1, page 244,](#)
- [Robot type 20, 3-Axis Delta with 1 Optional Manual Axis,, page 246,](#)
- [Robot Type 21, 2-Axis Parallelogram V2, page 251](#)
- [Robot Type 26, Robot with Offset Wrist with Six \(Seven\) Axes, page 254,](#)
- [Robot Type 27, Robot with Offset Wrist with Six Axes + Three Linear Axes, page 260,](#)
- [Robot Type 29, 2-Axis Delta with Rotary and Translatory Axis, page 267,](#)
- [Robot Type 30, Adabot, page 270,](#)
- [Robot Type 32, 2-Axis Delta with Asymmetric TCP, page 273,](#)
- [Robot Type 100, PLC, page 274](#)

18.2 Robot Type 00, Cartesian Robot

The transformation is switched off for robot type 0 (Cartesian robot). The axis coordinates (JC) are identical to the world coordinates (WC), i.e. in terms of the path motion, MOVE PTP is identical to MOVE LINEAR. All axes participating in the motion are started simultaneously and reach their target point simultaneously.

For MOVE LINEAR and CIRCULAR, the path velocity for the positioning axes in the RCL program is programmed with V and the velocities for the orientation axes are set so that all axes reach their target point at the same time.

For MOVE PTP motions, the velocity is determined based on the maximum axis velocities multiplied by the V_PTP programmed in RCL. The axis with the longest motion in terms of time determines the overall duration for the PTP motion. All other axes are adjusted to match this time.

Transformation Rules

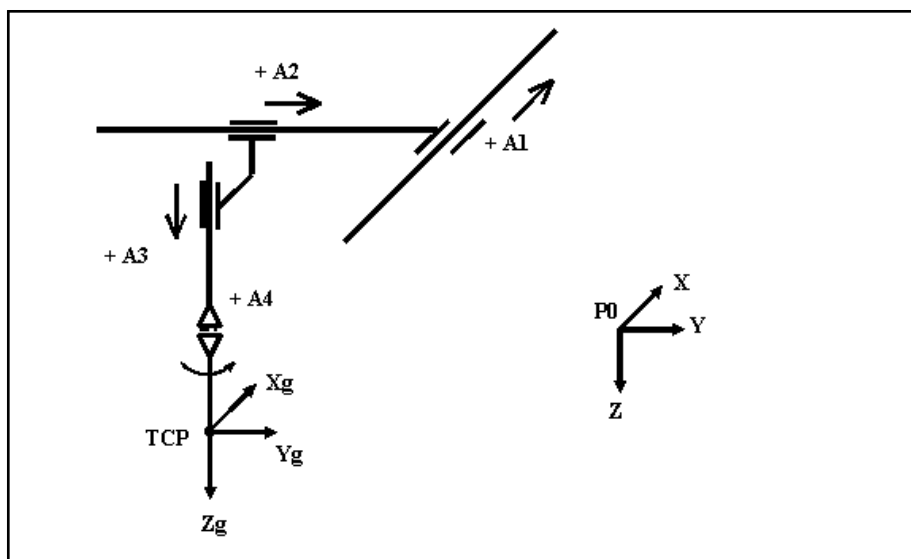


Fig. 18-1: Robot type 0, Cartesian robot, 3D perspective

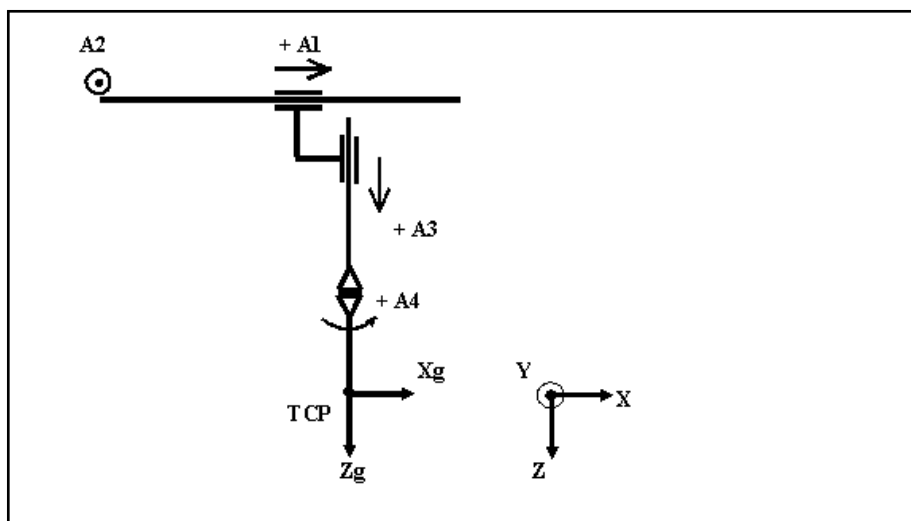


Fig. 18-2: Robot type 0, a typical X-Y-Z-C Cartesian system

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Translatory absolute	
A2:	Positioning axis (optional) Orientation axis (optional) Belt axis (optional)	Translatory absolute Rotary absolute/modulo Translatory absolute	Optional
A3:	Positioning axis (optional) Orientation axis (optional) Belt axis (optional)	Translatory absolute Rotary absolute/modulo Translatory absolute	Optional
A4:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A5:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional

Transformation Rules

A6:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A7:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A8:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A9:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A10:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A11:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A12:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A13:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A14:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A15:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional
A16:	Orientation axis (optional) Belt axis (optional)	Rotary absolute/modulo Translatory absolute	Optional

Fig. 18-3: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, Type = 0 (Cartesian robot)
- K-0-0032, Referencing all axes = 0
- K-0-0033, IPOS, update at coordinate change = 0
- K-0-0036, WC axis monitoring = 1
- K-0-0040, Transformation lengths, axis configuration, axis lengths not used
- K-0-0041, Transformation factors, axis configuration, axis coupling not used
- K-0-0121 .. K-0-0136, Coordinate names,

Coordinates: in the Cartesian coordinate system the robot has exactly as many degrees of freedom as axes in K-0-0008 were assigned (incl. belt axes).

Axes have to be assigned to all degrees of freedom.

Example of a 3-axis Cartesian system with an orientation axis and two belt axes

Name	Coordinate	Remark
K-0-0121	Coordinate 1:	Cx

Transformation Rules

K-0-0122	Coordinate 2:	Cy
K-0-0123	Coordinate 3:	Cz
K-0-0124	Coordinate 4:	Cc
K-0-0125	Coordinate 5:	Cb1
K-0-0126	Coordinate 6:	Cb2

Fig. 18-4: K-0-0121 .. K-0-0126, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- exact adjustment of the axis angle
- correct direction of rotation of the axes

If one of the these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

The following views are available for robot type 0 (Cartesian):

- [fig. 18-1 "Robot type 0, Cartesian robot, 3D perspective "](#) on page 228 and
- [fig. 18-2 "Robot type 0, a typical X-Y-Z-C Cartesian system"](#) on page 228.
- The zero positions for all axes can be freely selected for the Cartesian robot and can also be outside the travel range limits.
- The travel direction for up to 3 positioning axes always forms a right-handed Cartesian coordinate system.
- The zero points and the travel directions of the orientation axes are arbitrary.

18.3 Robot Type 07, 2-Axis Delta with 1 Optional Manual Axis

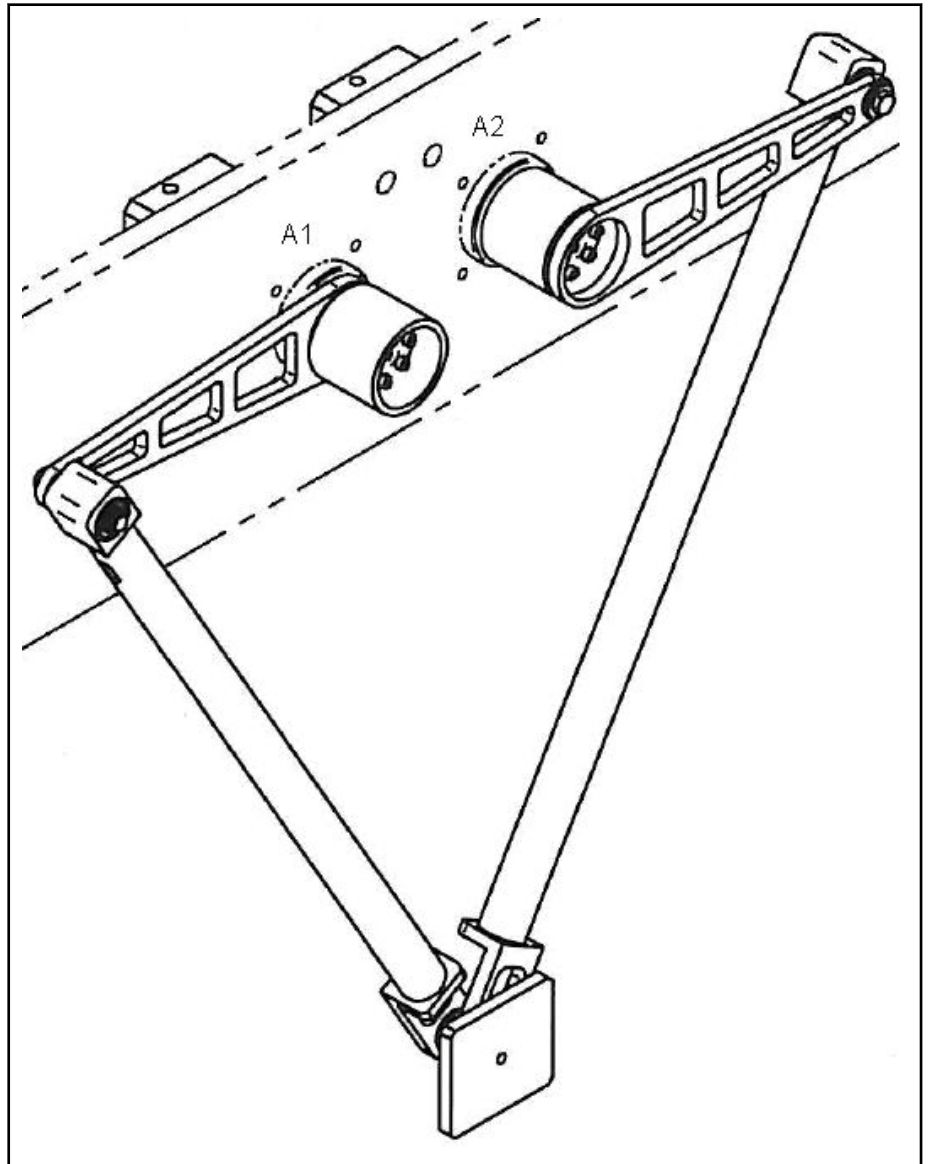


Fig. 18-5: Robot type 07, 2-axis delta

Transformation Rules

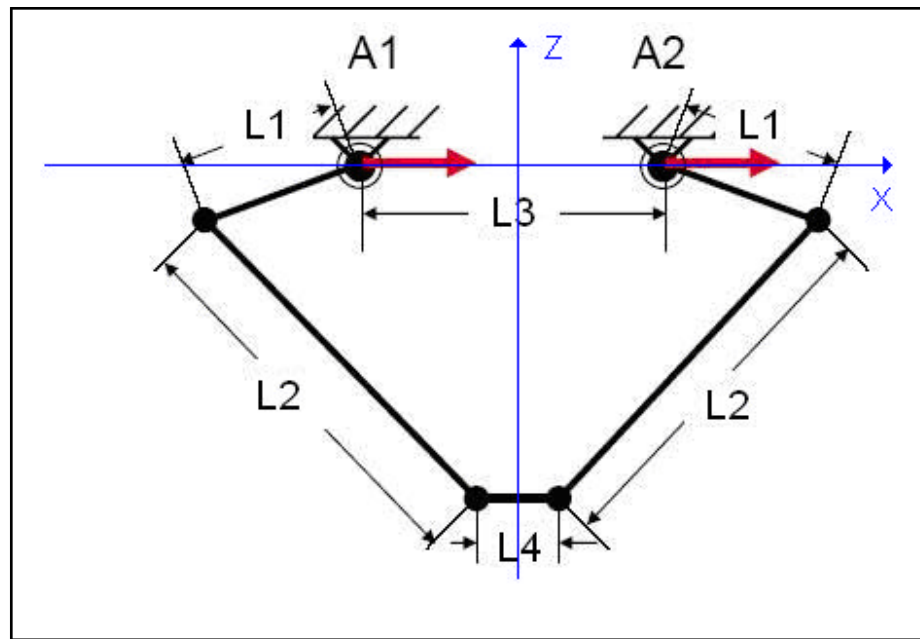


Fig. 18-6: Robot type 07, 2-axis delta (X-Z view)

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+ 90 .. 270 degrees
A2:	Positioning axis	Rotary absolute	+/- 90 degrees
A3:	Orientation axis	Rotary abs./modulo	Optional
A4:	Belt axis (optional)	Translatory, modulo	Optional
A5:	Belt axis (optional)	Translatory, modulo	Optional
A6:	Belt axis (optional)	Translatory, modulo	Optional
A7:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-7: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, transformation scheme, Type = 7 (2-axis delta, bipod)
- K-0-0032, Referencing all axes = 0
- K-0-0033, IPOS, update at coordinate change = 0
- K-0-0036, WC axis monitoring = 1

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of the arm at the motor	mm or inch (see K-0-0054)	>0
L2:	Length of rod to mold plate	mm or inch (see K-0-0054)	>0
L3:	Distance of the two rotation points of the arms	mm or inch (see K-0-0054)	>0
L4:	Distance of rods at base plate	mm or inch (see K-0-0054)	>0

Fig. 18-8: K-0-0040, Transformation lengths, axis configuration, axis lengths

K-0-0041, Transformation factors, axis configuration, axis coupling, is not used

Transformation Rules

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cz)	
Coordinate 3:	Name of the 3rd coordinate (e.g. Cc)	Optional
Coordinate 4:	Name of the 4th coordinate (e.g. Kbelt1)	Optional
Coordinate 5:	Name of the 5th coordinate (e.g. Kbelt2)	Optional
Coordinate 6:	Name of the 6th coordinate (e.g. Kbelt3)	Optional
Coordinate 7:	Name of the 7th coordinate (e.g. Kbelt4)	Optional

Fig. 18-9: K-0-0121 .. K-0-0136, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- Correct direction of rotation of the axes

If one of these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

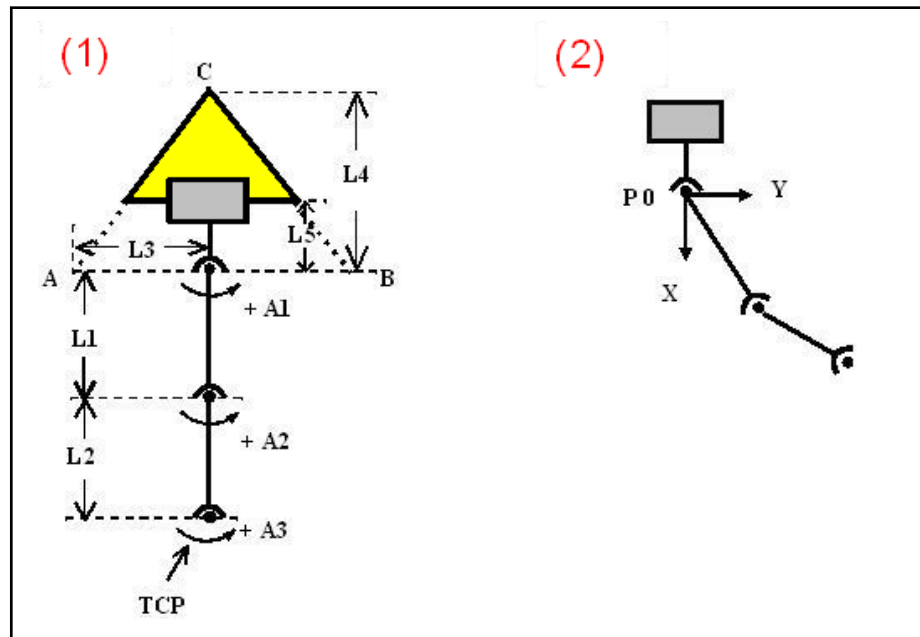
fig. 18-5 "Robot type 07, 2-axis delta" on page 231 shows a bipod robot in 3D with the axis positions $A1=180^\circ$ and $A2 = 0^\circ$.

In zero position, the axes 1 and 2 (positioning axes) are horizontal towards the x-axis (see red arrow in fig. 18-6 "Robot type 07, 2-axis delta (X-Z view)" on page 232).

- The zero point for the 3rd axis (orientation axis) is in the positive direction on the x-axis.
- The arm length L_1 is measured from the rotary center point to the center point of the rod retainer.
- The rod length L_2 is measured from the reference points on both sides.
- L_3 is the distance of the rotary center point of both arms.
- L_4 is the length of the mold plate measured at the rod mountings.
- Positive counterclockwise rotation of the positioning axes A1 and A2.
- The Cartesian zero point of the bipod robot is in the center of both arm rotation points A1 and A2.

Transformation Rules

18.4 Robot Type 09, 3-Axis Scara with 1 Optional Manual Axis



(1) Top view with zero positions for all axes; yellow: prohibited area
(2) Zero point reference of all axes

Fig. 18-10: Robot type 09, 3-axis Scara with 1 optional manual axis

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+/- 179 degrees
A2:	Positioning axis	Rotary absolute	+/- 179 degrees
A3:	Orientation axis (optional)	Rotary	Optional
A4:	Belt axis (optional)	Translatory, modulo	Optional
A5:	Belt axis (optional)	Translatory, modulo	Optional
A6:	Belt axis (optional)	Translatory, modulo	Optional
A7:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-11: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, Type = 09 (3-axis Scara)
- K-0-0032, Referencing all axes = 0
- K-0-0033, IPOS, update at coordinate change = 0
- K-0-0036, WC axis monitoring = 1

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of the 1st arm	mm or inch (see K-0-0054)	>0
L2:	Length of the 2nd arm	mm or inch (see K-0-0054)	>0
L3:	Distance from the zero point to the intersection with the line AC	mm or inch (see K-0-0054)	Optional, see ①

Transformation Rules

L4:	Distance from zero point to point C of the triangle	mm or inch (see K-0-0054)	Optional, see ①
L5:	Distance from zero point to the line AB	mm or inch (see K-0-0054)	Optional, see ①

Fig. 18-12: K-0-0040, Transformation lengths, axis configuration, axis lengths

Transformation factor	Axis configuration	Measuring unit	Axis coupling
C1:	Mechanical coupling between the axes 1, 2 and 3	Factor	Optional

Fig. 18-13: K-0-0041, Transformation factors, axis configuration, axis coupling

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cy)	
Coordinate 3:	Name of the 3rd coordinate (e.g. Cc)	Optional
Coordinate 4:	Name of the 4th coordinate (e.g. Cb1)	Optional
Coordinate 5:	Name of the 5th coordinate (e.g. Cb2)	Optional
Coordinate 6:	Name of the 6th coordinate (e.g. Cb3)	Optional
Coordinate 7:	Name of the 7th coordinate (e.g. Cb4)	Optional

Fig. 18-14: K-0-0121 .. K-0-0128, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- exact determination of the coupling factors (if present)
- Correct direction of rotation of the axes

If one of these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed. This can result in incorrect or unexpected motions.

fig. 18-10 "Robot type 09, 3-axis Scara with 1 optional manual axis" on page 234: (1) displays the zero positions of all axes in a top view.

- If the robot is in the position shown in the figures, all axes in the machine coordinates have to display zero.

This can be ensured by setting the actual values of the reference point accordingly.

- The figures also show arm lengths L_1 , L_2 and the positive rotary directions of the axes.
- A prohibited area can be defined with the lengths L_3 , L_4 , L_5 where the tool center point (TCP) may not enter.

This prevents collisions when the travel ranges of axes 1 and 2 are so large that the robot could run into its own support.

In ①, this prohibited area is shown in yellow.

- If the length $L_3=0$, range monitoring is not enabled.
- The lengths L_1 , L_2 , L_3 , L_4 , L_5 always have to be entered as positive number and L_1 , L_2 may not be equal to zero.

Transformation Rules

A2:	Positioning axis	Rotary absolute	+/- 179 degrees
A3:	Positioning axis (optional)	translatory	Optional
A4:	Orientation axis (optional)	Rotary	Optional
A5:	Belt axis (optional)	Translatory, modulo	Optional
A6:	Belt axis (optional)	Translatory, modulo	Optional
A7:	Belt axis (optional)	Translatory, modulo	Optional
A8:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-16: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, Type = 10 (2-axis Scara)
- K-0-0032, Referencing all axes = 0
- K-0-0033, IPOS, update at coordinate change = 0
- K-0-0036, WC axis monitoring = 1

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of the 1st arm	mm or inch (see K-0-0054)	>0
L2:	Length of the 2nd arm	mm or inch (see K-0-0054)	>0
L3:	Distance from the zero point to the intersection with the line AC	mm or inch (see K-0-0054)	Optional, see ①
L4:	Distance from zero point to point C of the triangle	mm or inch (see K-0-0054)	Optional, see ①
L5:	Distance from zero point to the line AB	mm or inch (see K-0-0054)	Optional, see ①

Fig. 18-17: K-0-0040, Transformation lengths, axis configuration, axis lengths

Transformation factor	Axis configuration	Measuring unit	Axis coupling
C1:	Mechanical coupling between axes 1, 2 and 4	Factor	Optional
C2:	Mechanical coupling between motors 4 and 3 (distance/lathe spindle)	mm/degree	Optional

Fig. 18-18: K-0-0041, Transformation factors, axis configuration, axis coupling

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cy)	
Coordinate 3:	Name of the 3rd coordinate (e.g. Cz)	Optional
Coordinate 4:	Name of the 4th coordinate (e.g. Cc)	Optional
Coordinate 5:	Name of the 5th coordinate (e.g. Cb1)	Optional
Coordinate 6:	Name of the 6th coordinate (e.g. Cb2)	Optional
Coordinate 7:	Name of the 7th coordinate (e.g. Cb3)	Optional
Coordinate 8:	Name of the 8th coordinate (e.g. Cb4)	Optional

Fig. 18-19: K-0-0121 .. K-0-0128, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

Transformation Rules

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- exact determination of the coupling factors (if present)
- Correct direction of rotation of the axes

If one of the these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

fig. 18-15 "Robot type 10, 4-axis Scara with 1 optional manual axis" on page 236: (1) displays the zero positions of all axes in top view and (2) in side view.

- If the robot is in the position shown in the figures, all axes in the machine coordinates have to display zero.
This can be ensured by setting the actual values of the reference point accordingly.

- In the zero position, axes 1, 2 and 4 are extended in forward direction.
- The zero point for the 3rd axis can be freely chosen.
- The figures also show the arm lengths L_1 , L_2 and the positive directions of rotations of the axes.
- A prohibited area can be defined with the lengths L_3 , L_4 , L_5 where the tool center point (TCP) may not enter.
- This prevents collisions when the travel ranges of axes 1 and 2 are so large that the robot could run into its own support.

In ①, this prohibited area is shown in yellow.

- If the length $L_3=0$, range monitoring is not enabled.
- The lengths L_1 , L_2 , L_3 , L_4 , L_5 always have to be entered as positive number and L_1 , L_2 may not be equal to zero.
- The transformation can consider a mechanical coupling of the axes 1 and 2 with axis 4.

Such a coupling exists if axis 4 moves along when axes 1 and 2 travel and the measuring system for the 4th axis does not detect this motion. This is generally the case if the motor for the 4th axis is not directly on the axis, but instead on its support.

Set this coupling using the parameter K-0-0041 (see above).

- The transformation also consider a mechanical coupling of the 4th axis with the 3rd axis.

Such a coupling exists if the 3rd world coordinate (z-axis) changes when axis 4 travels and the measuring system for the 3rd axis does not detect this motion.

This is for example the case if the axes 3 and 4 are a lifting spindle and a rotating spindle.

The coupling factor is defined as follows:

- $K_{43} = \Delta z / \Delta A_4$

ΔA_4 is the axis angle detected by the measuring system and Δz is the change in the world coordinate.

Transformation Rules



The maximum travel range of axes 1 and 2 is ± 179 degrees.
The travel range for axes 3 and 4 is unlimited.

The respective world coordinate system is identified by the 3 main axes (x, y, z).

The zero point for the world coordinates is at the base point P_0 of the robot.

The transformation can also be used for a 2- or 3-axis Scara.

18.6 Robot Type 12, 2-Axis Parallelogram

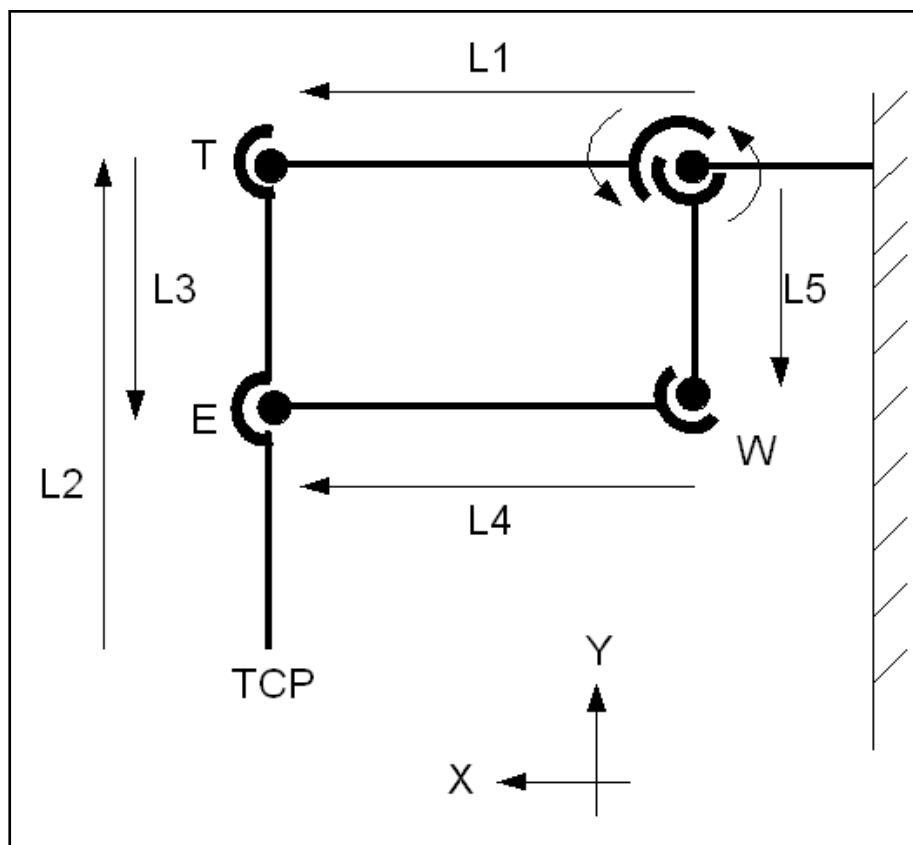


Fig. 18-20: Robot type 12, 2-axis parallelogram

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+/- 89 degrees
A2:	Positioning axis	Rotary absolute	+/- 89 degrees
A3:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-21: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, Type = 12 (2-axis parallelogram)
- K-0-0032, Referencing all axes = 0

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of the 1st arm	mm or inch (see K-0-0054)	>0

Transformation Rules

L2:	Length of the 2nd arm	mm or inch (see K-0-0054)	>0
L3:	Length of the 3rd arm	mm or inch (see K-0-0054)	>0
L4:	Length of the 4th arm	mm or inch (see K-0-0054)	>0
L5:	Length of the 5th arm	mm or inch (see K-0-0054)	>0

Fig. 18-22: K-0-0040, Transformation lengths, axis configuration, axis lengths

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cy)	

Fig. 18-23: K-0-0121 .. K-0-0128, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- correct determination of signs of individual lengths
- Correct direction of rotation of the axes

If one of the these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

[fig. 18-20 "Robot type 12, 2-axis parallelogram " on page 239](#) shows

- The schematic kinematic representation for axes in zero position.
- The figures also show the arm lengths L_1 , L_2 , L_3 , L_4 , L_5 and the positive directions of rotation of the axes.
- The travel range of the axes has to be limited mechanically so that the angle between both arms with the length name L_1 and L_5 is never equal to 180° . Otherwise, two different mathematical solutions are available depending on the mechanics. This can result in undesired motions.
- The lengths L_1 , L_2 , L_3 , L_4 , L_5 always have to be entered as a positive number and have to be unequal to zero.
- The respective world coordinate system is identified by the two main axes (x, y).
- The world coordinate zero point is located in the intersection of the two length specifications L_1 and L_5 .

18.7 Robot Type 13, 2-Axis Parallelogram

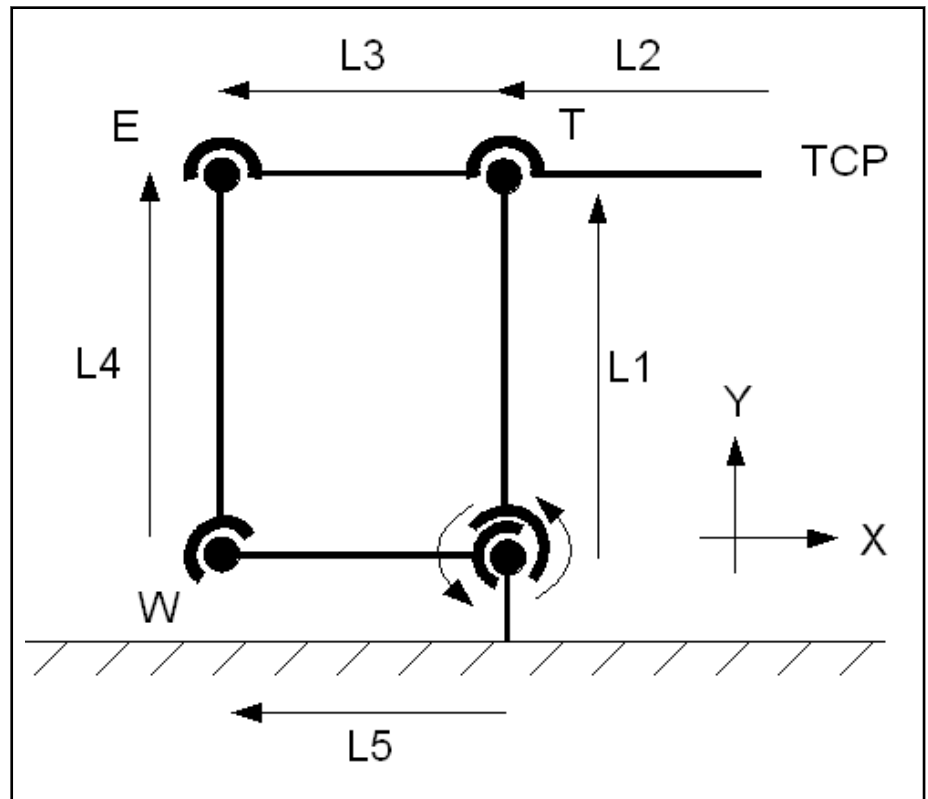


Fig. 18-24: Robot type 13, 2-axis parallelogram

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+/- 89 degrees
A2:	Positioning axis	Rotary absolute	+/- 89 degrees
A3:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-25: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, Type = 13 (2-axis parallelogram)
- K-0-0032, Referencing all axes = 0

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of the 1st arm	mm or inch (see K-0-0054)	>0
L2:	Length of the 2nd arm	mm or inch (see K-0-0054)	>0
L3:	Length of the 3rd arm	mm or inch (see K-0-0054)	>0
L4:	Length of the 4th arm	mm or inch (see K-0-0054)	>0
L5:	Length of the 5th arm	mm or inch (see K-0-0054)	>0

Fig. 18-26: K-0-0040, Transformation lengths, axis configuration, axis lengths

Transformation Rules

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cy)	

Fig. 18-27: K-0-0121 .. K-0-0128, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- correct determination of signs of individual lengths
- Correct direction of rotation of the axes

If one of these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

[fig. 18-24 "Robot type 13, 2-axis parallelogram "](#) on page 241 shows

- The schematic kinematic representation for axes in zero position.
- The figures also show the arm lengths L_1 , L_2 , L_3 , L_4 , L_5 and the positive directions of rotation of the axes.
- The travel range of the axes has to be limited mechanically so that the angle between both arms with the length name L_1 and L_5 is never equal to 180° . Otherwise, two different mathematical solutions are available depending on the mechanics. This can result in undesired motions.
- The lengths L_1 , L_2 , L_3 , L_4 , L_5 always have to be entered as a positive number and have to be unequal to zero.
- The respective world coordinate system is identified by the two main axes (x, y).
- The world coordinate zero point is located in the intersection of the two length specifications L_1 and L_5 .

18.8 Robot Type 18, Contact Kinematics

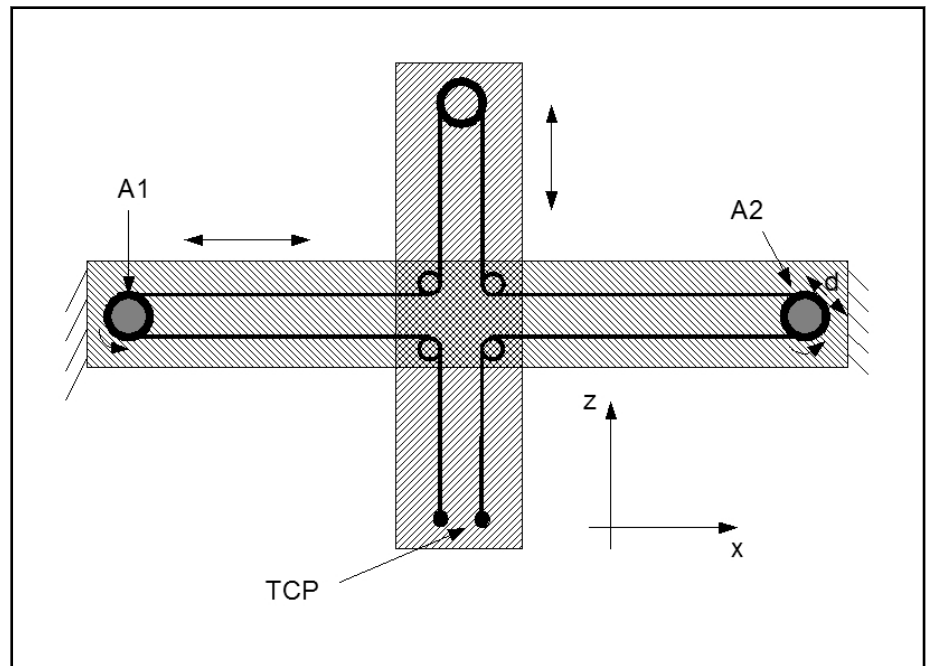


Fig. 18-28: Robot type 18, contact kinematics

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	Gear: the radius ($d/2$) has to be entered at load side
A2:	Positioning axis	Rotary absolute	Gear: the radius ($d/2$) has to be entered at load side

Fig. 18-29: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, type = 18 (Cartesian kinematics).
- K-0-0032, Referencing of all axes = 0.
- K-0-0040, Transformation lengths, axis configuration, axis lengths are not used.

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cz)	

Fig. 18-30: K-0-0121 .. K-0-0128, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- exact measurement of diameter d of driving axes
- Correct direction of rotation of the axes

If one of these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

Transformation Rules

fig. 18-28 "Robot type 18, contact kinematics" on page 243 shows the schematic kinematic representation for axes in zero position. Consequences for the application: the axes have to be travelled to their desired zero position. Subsequently, they have to be referenced.

- The positive directions of rotation of the axes are shown.
- The set travel range limits of the axes may not exceed the mechanical travel ranges.
- The respective world coordinate system is identified by the two main axes (x, y).
- The gear of both axes has to set so that the radius ($d/2$) is entered in mm on the load side (S-0-0121 = 1, S-0-0122 = $d/2$).

18.9 Robot Type 19, 2-Axis Parallelogram V1

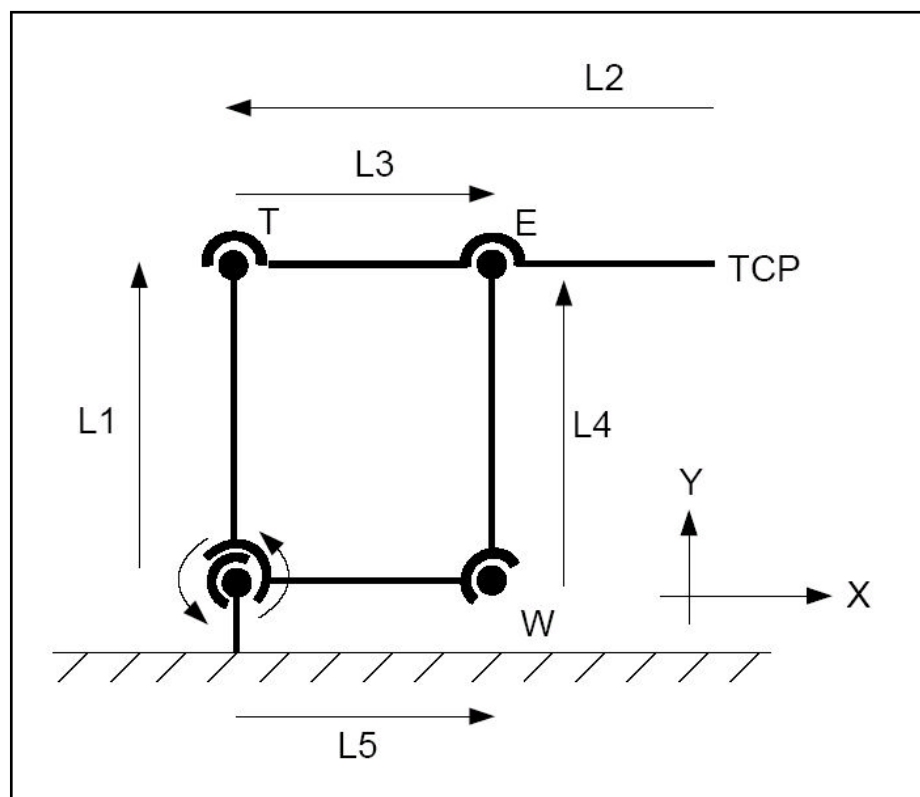


Fig. 18-31: Robot type 19, 2-axis parallelogram V1

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+/- 89 degrees
A2:	Positioning axis	Rotary absolute	+/- 89 degrees
A3:	Positioning axis	Translatory absolute	Optional
A4:	Orientation axis	Rotary mod./abs.	Optional
A5:	Belt axis (optional)	Translatory, modulo	Optional
A6:	Belt axis (optional)	Translatory, modulo	Optional

Transformation Rules

A7:	Belt axis (optional)	Translatory, modulo	Optional
A8:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-32: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, Type = 19 (2-axis parallelogram)
- K-0-0032, Referencing all axes = 0

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of the 1st arm	mm or inch (see K-0-0054)	>0
L2:	Length of the 2nd arm	mm or inch (see K-0-0054)	>0
L3:	Length of the 3rd arm	mm or inch (see K-0-0054)	>0
L4:	Length of the 4th arm	mm or inch (see K-0-0054)	>0
L5:	Length of the 5th arm	mm or inch (see K-0-0054)	>0

Fig. 18-33: K-0-0040, Transformation lengths, axis configuration, axis lengths

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cy)	
Coordinate 3:	Name of the 3rd coordinate (e.g. Cz)	
Coordinate 4:	Name of the 4th coordinate (e.g. Cb)	
Coordinate 5:	Name of the 5th coordinate (e.g. Cbelt)	

Fig. 18-34: K-0-0121 .. K-0-0128, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- correct determination of signs of individual lengths
- Correct direction of rotation of the axes

If one of the these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

fig. 18-31 "Robot type 19, 2-axis parallelogram V1" on page 244 shows

- The schematic kinematic representation for axes in zero position.
- The figures also show the arm lengths L_1 , L_2 , L_3 , L_4 , L_5 and the positive directions of rotation of the axes.
- The travel range of the axes has to be limited mechanically so that the angle between both arms with the length name L_1 and L_5 is never equal to 180° . Otherwise, two different mathematical solutions are available depending on the mechanics. This can result in undesired motions.
- The lengths L_1 , L_2 , L_3 , L_4 , L_5 always have to be entered as a positive number and have to be unequal to zero.

Transformation Rules

- The respective world coordinate system is identified by the two main axes (x, y).
- The world coordinate zero point is located in the intersection of the two length specifications L_1 and L_5 .

18.10 Robot type 20, 3-Axis Delta with 1 Optional Manual Axis,

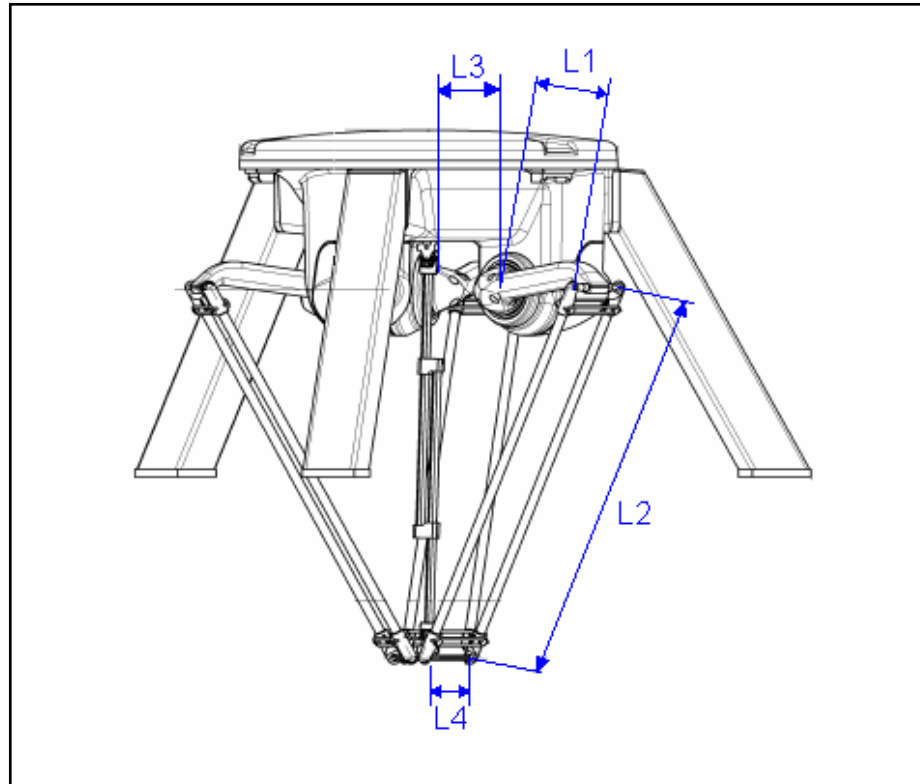
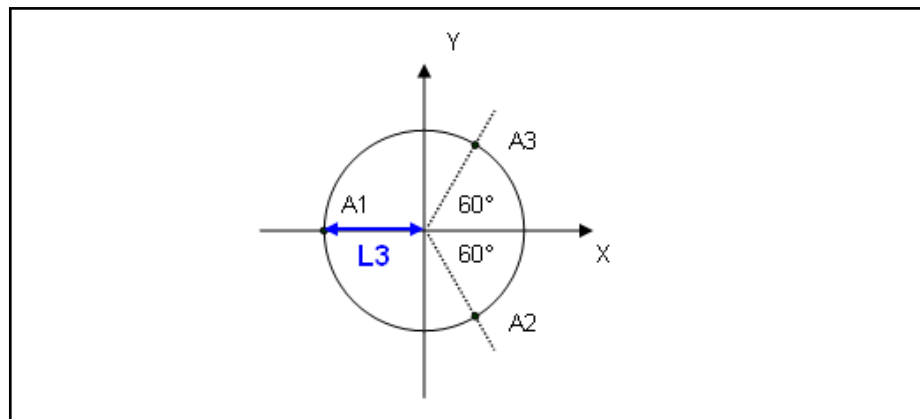


Fig.18-35: Robot type 20, 3-axis Delta with 1 optional manual axis

Fig.18-36: Radius $L3$ of the motor base and the coordinate system (X-Y view)

The three axes A1 to A3 are all displaced by 120° from each other.

Transformation Rules

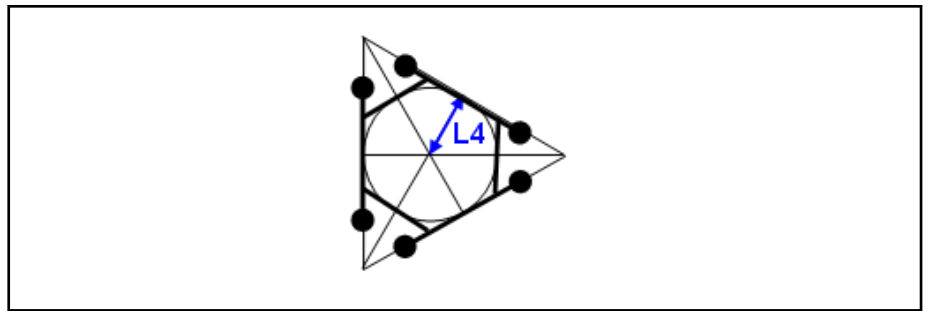
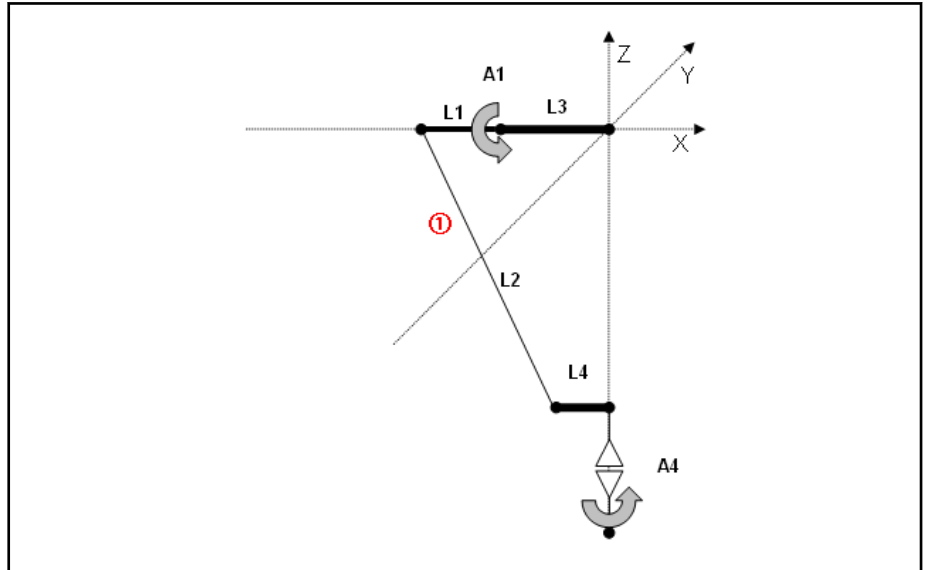


Fig. 18-37: Radius L4 of the tool plate



① Connection rod

Fig. 18-38: Representation of the zero position of the first axis

The zero position of the orientation axis A4 should be in positive x-direction.

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+/- 90 degrees
A2:	Positioning axis	Rotary absolute	+/- 90 degrees
A3:	Positioning axis	Rotary absolute	+/- 90 degrees
A4:	Orientation axis (optional)	Rotary abs./modulo	Optional
A5:	Belt axis (optional)	Translatory, modulo	Optional
A6:	Belt axis (optional)	Translatory, modulo	Optional
A7:	Belt axis (optional)	Translatory, modulo	Optional
A8:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-39: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, Type = 20 (3-axis Delta)
- K-0-0032, Referencing all axes = 0
- K-0-0033, IPOS, update at coordinate change = 0
- K-0-0036, WC axis monitoring = 1

Transformation Rules

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of the arm at the motor	mm or inch (see K-0-0054)	>0
L2:	Length of rod to mold plate	mm or inch (see K-0-0054)	>0
L3:	Radius of the circle defined by the three rotation points of the arms	mm or inch (see K-0-0054)	>0
L4:	Radius of the base plate	mm or inch (see K-0-0054)	>0
L5:	Has to be 0		=0
L6:	Has to be 0		=0

Fig. 18-40: K-0-0040, Transformation lengths, axis configuration, axis lengths



If L5 and L6 $\neq$ 0, translatorily scaled axes are used. Refer to [Translatorily-scaled positioning axes on page 249](#).

K-0-0041, Transformation factors, axis configuration, axis coupling, is not used

Name	Coordinate	Remark
K-0-0121	Coordinate 1:	Cx
K-0-0122	Coordinate 2:	Cy
K-0-0123	Coordinate 3:	Cz
K-0-0124	Coordinate 4:	Cc
K-0-0125	Coordinate 5:	Cb1
K-0-0126	Coordinate 6:	Cb2
K-0-0127	Coordinate 7:	Cb3
K-0-0128	Coordinate 8:	Cb4

Fig. 18-41: K-0-0121 .. K-0-0128, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- Correct direction of rotation of the axes

If one of the these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

fig. 18-35 "Robot type 20, 3-axis Delta with 1 optional manual axis" on page 246 shows a Delta robot at zero position of all axes, i.e. if the robot is in the position shown in the figure, the joint coordinates for all of the axes have to be zero. This can be ensured by setting the actual values of the reference point accordingly.

In the zero position, the axes 1, 2 and 3 (positioning axes) are horizontal.

Transformation Rules

The zero point for the 4th axis (orientation axis) is in the positive direction of the x-axis.

The arm length L_1 is measured from the rotary center point to the center point of the rod retainer.

The rod length L_2 is measured from the reference points on both sides.

L_3 is the radius of the perimeter formed by the three motor spindles (or gears) measured at the rotary center points of the arms (see [fig. 18-36 "Radius \$L_3\$ of the motor base and the coordinate system \(X-Y view\)" on page 246](#)).

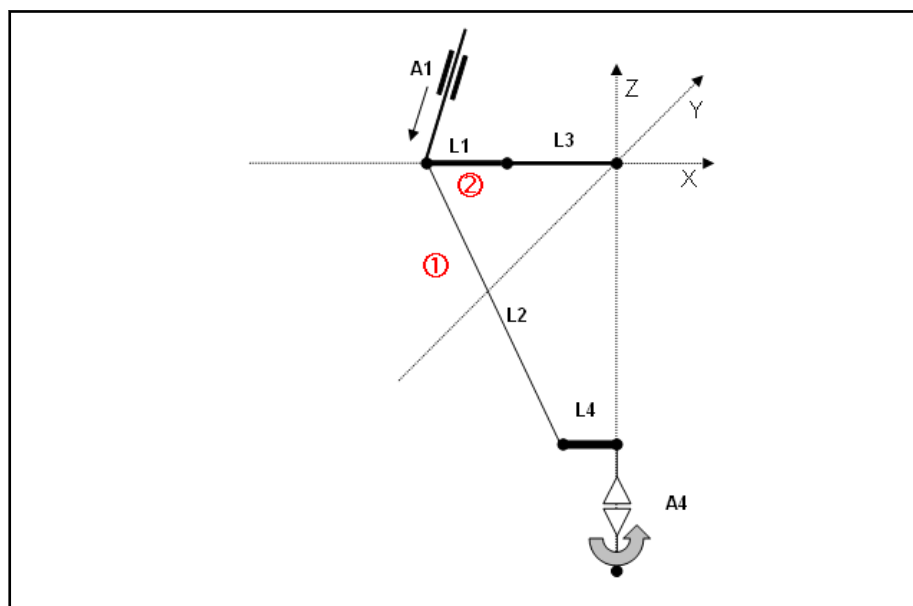
L_4 is the radius of the incircle of the equilateral triangle on the tool plate. The equilateral triangle is defined by the reference points of the rods. (see [fig. 18-37 "Radius \$L_4\$ of the tool plate" on page 247](#)).

The direction of rotation of the positioning axes A1 to A3 is positive if the arms move down into z-direction. The orientation axis A4 (rotation about the z-axis) rotates positively counterclockwise.

The Cartesian zero point for the Delta robot is at the center point of the perimeter defined by the three motor spindles (see also [fig. 18-36 "Radius \$L_3\$ of the motor base and the coordinate system \(X-Y view\)" on page 246](#)).

Translatorily-scaled positioning axes

The transformation can also be operated with translatorily-scaled positioning axes.



- ① Connection rod
- ② Arm

Fig. 18-42: Representation of the zero position for translatorily-scaled positioning axes

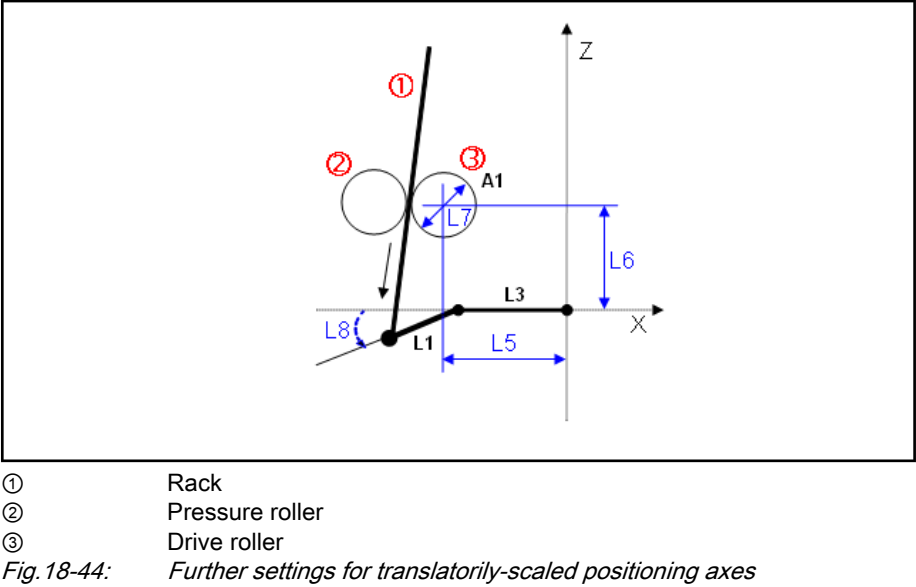
When the arms are in a horizontal position, the positions of the axes A1 to A3 have to be zero. The positive motion of the positioning axes pushes the corresponding arm down.

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Translatory absolute	
A2:	Positioning axis	Translatory absolute	
A3:	Positioning axis	Translatory absolute	

Fig. 18-43: K-0-0008, Axis configuration list, axis assignment

Transformation Rules

- The positioning axes A1 to A3 can also be translatory.
- K-0-0040, Transformation lengths, axis configuration, axis lengths; in addition to the K-0-0040 entries L1 to L4 described above, L5 to L8 have to be set.



Transformation length	Axis configuration	Measuring unit	Axis length
L5:	Horizontal distance from the rack drive roller to the origin of the coordinate system	mm or inch (see K-0-0054)	>0
L6:	Vertical distance from the drive roller to the x-y plane	mm or inch (see K-0-0054)	>0
L7:	Diameter of the drive roller	mm or inch (see K-0-0054)	>0
L8:	Phase correction	Degrees	-45 ... +45

Fig. 18-45: K-0-0040, Transformation lengths, axis configuration, axis lengths

 If L5 and L6 ≠ 0, translatorily-scaled axes are used.

18.11 Robot Type 21, 2-Axis Parallelogram V2

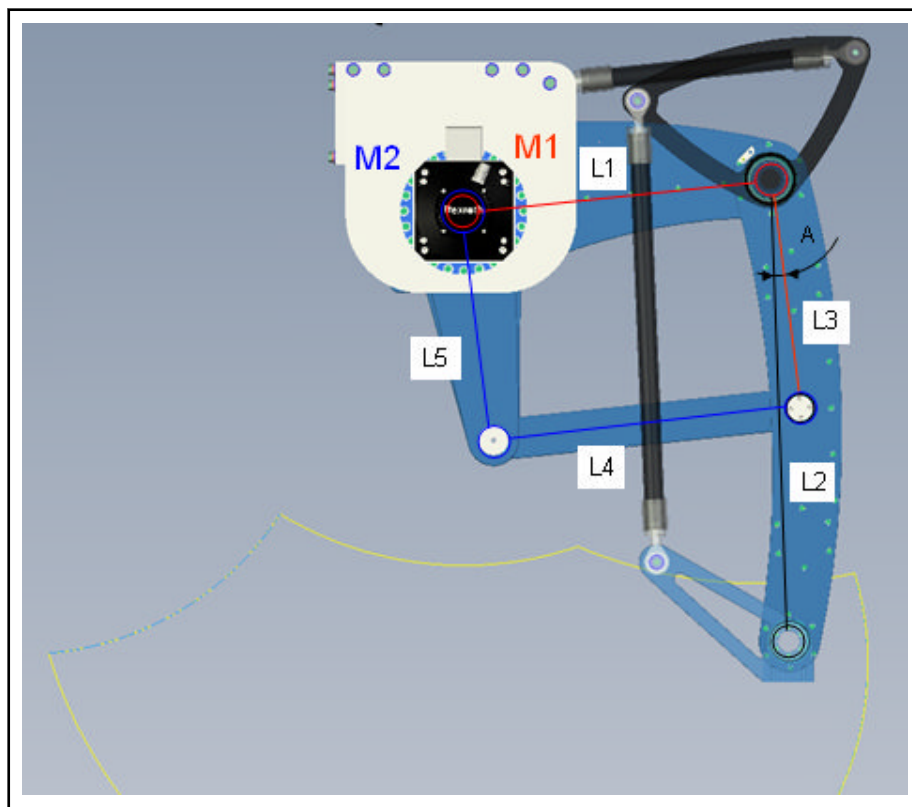


Fig. 18-46: Robot type 21, 2-axis parallelogram V2

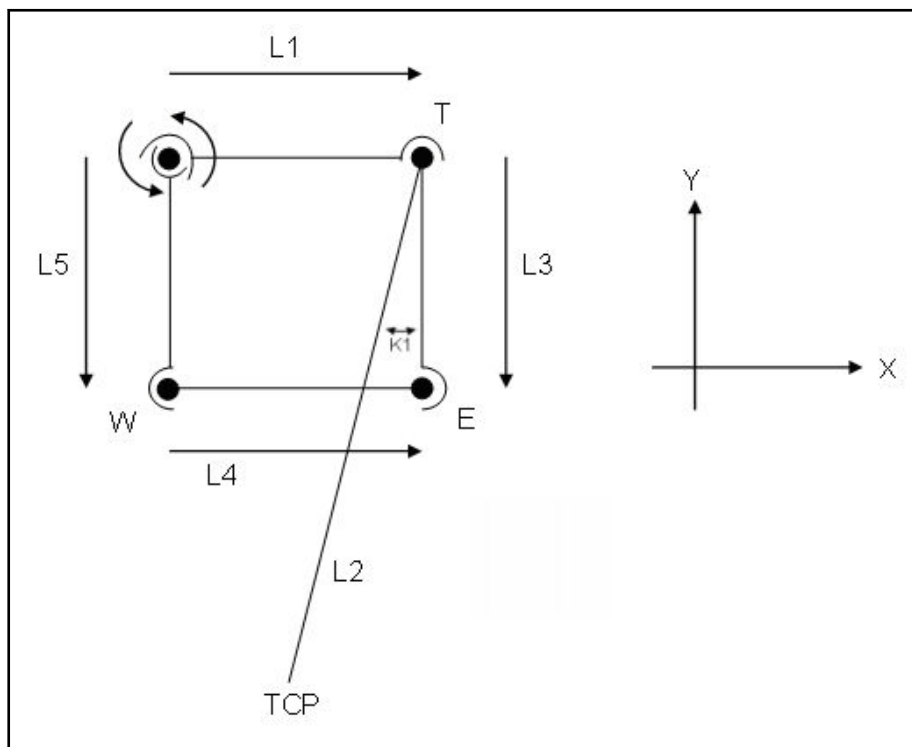


Fig. 18-47: Robot type 21, 2-axis parallelogram V2, in the zero position of the axes

Parameter settings

Transformation Rules

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+/- 89 degrees
A2:	Positioning axis	Rotary absolute	+/- 89 degrees
A3:	Positioning axis	Translatory absolute	Optional
A4:	Orientation axis	Rotary abs./mod.	Optional
A5:	Belt axis (optional)	Translatory, modulo	Optional
A6:	Belt axis (optional)	Translatory, modulo	Optional
A7:	Belt axis (optional)	Translatory, modulo	Optional
A8:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-48: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, Type = 21 (2-axis parallelogram)
- K-0-0032, Referencing all axes = 0

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of the 1st arm	mm or inch (see K-0-0054)	>0
L2:	Length of the 2nd arm	mm or inch (see K-0-0054)	>0
L3:	Length of the 3rd arm	mm or inch (see K-0-0054)	>0
L4:	Length of the 4th arm	mm or inch (see K-0-0054)	>0
L5:	Length of the 5th arm	mm or inch (see K-0-0054)	>0

Fig. 18-49: K-0-0040, Transformation lengths, axis configuration, axis lengths

Transformation factor	Configuration	Measuring unit	Remark
C1:	Angle between L ₂ and L ₃	Degrees	

Fig. 18-50: K-0-0041, Transformation factors

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cy)	
Coordinate 3:	Name of the 3rd coordinate (e.g. Cz)	
Coordinate 4:	Name of the 4th coordinate (e.g. Cb)	
Coordinate 5:	Name of the 5th coordinate (e.g. Cbelt)	

Fig. 18-51: K-0-0121 .. K-0-0128, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- correct determination of signs of individual lengths
- Correct direction of rotation of the axes

Transformation Rules

If one of the these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

fig. 18-47 "Robot type 21, 2-axis parallelogram V2, in the zero position of the axes" on page 251 shows the schematic kinematic representation for axes in zero position.

- The figures also show the arm lengths L_1 , L_2 , L_3 , L_4 , L_5 . The positive direction of rotations of the axes are counterclockwise.
- The travel range of the axes has to be limited mechanically so that the angle between both arms with the length name L_1 and L_5 is never equal to 180° . Otherwise, two different mathematical solutions are available depending on the mechanics. This can result in undesired motions.
- The lengths L_1 , L_2 , L_3 , L_4 , L_5 always have to be entered as a positive number and have to be unequal to zero.
- The respective world coordinate system is identified by the two main axes (x, y).
- The world coordinate zero point is located in the intersection of the two length specifications L_1 and L_5 .

Transformation Rules

18.12 Robot Type 26, Robot with Offset Wrist with Six (Seven) Axes

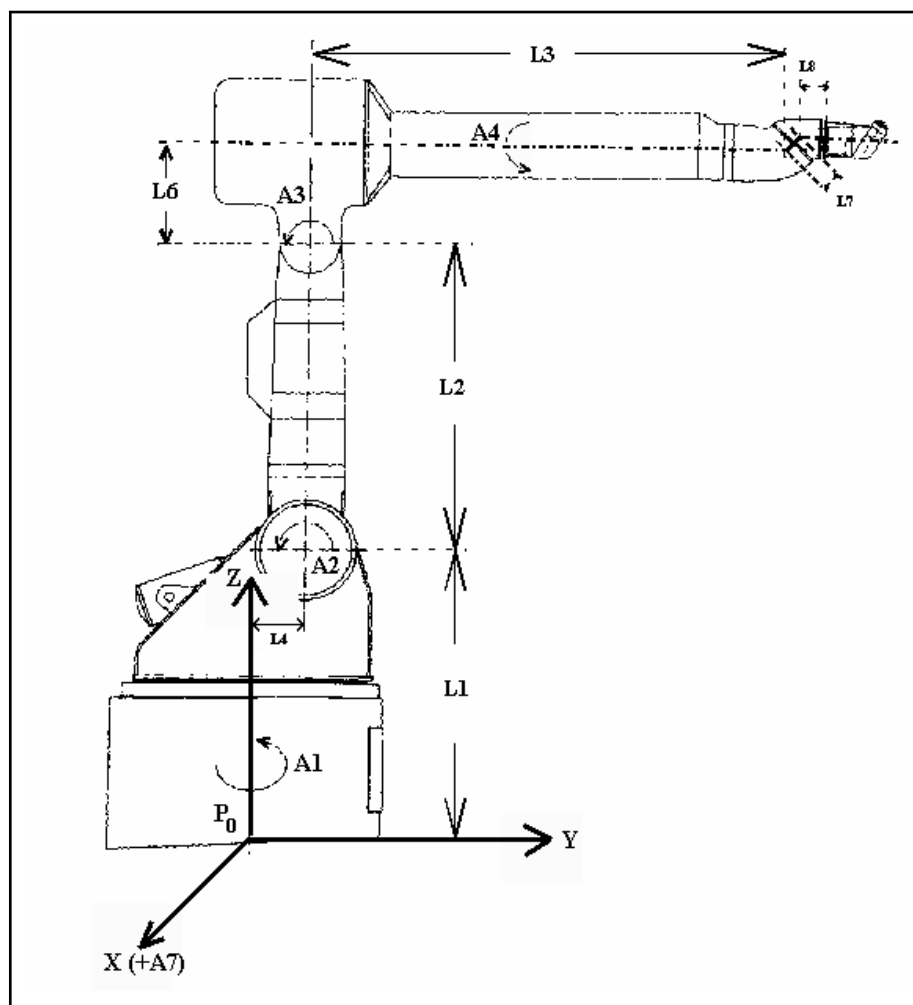


Fig.18-52: Robot type 26, robot with offset wrist with 6 (7) axes, zero position

Transformation Rules

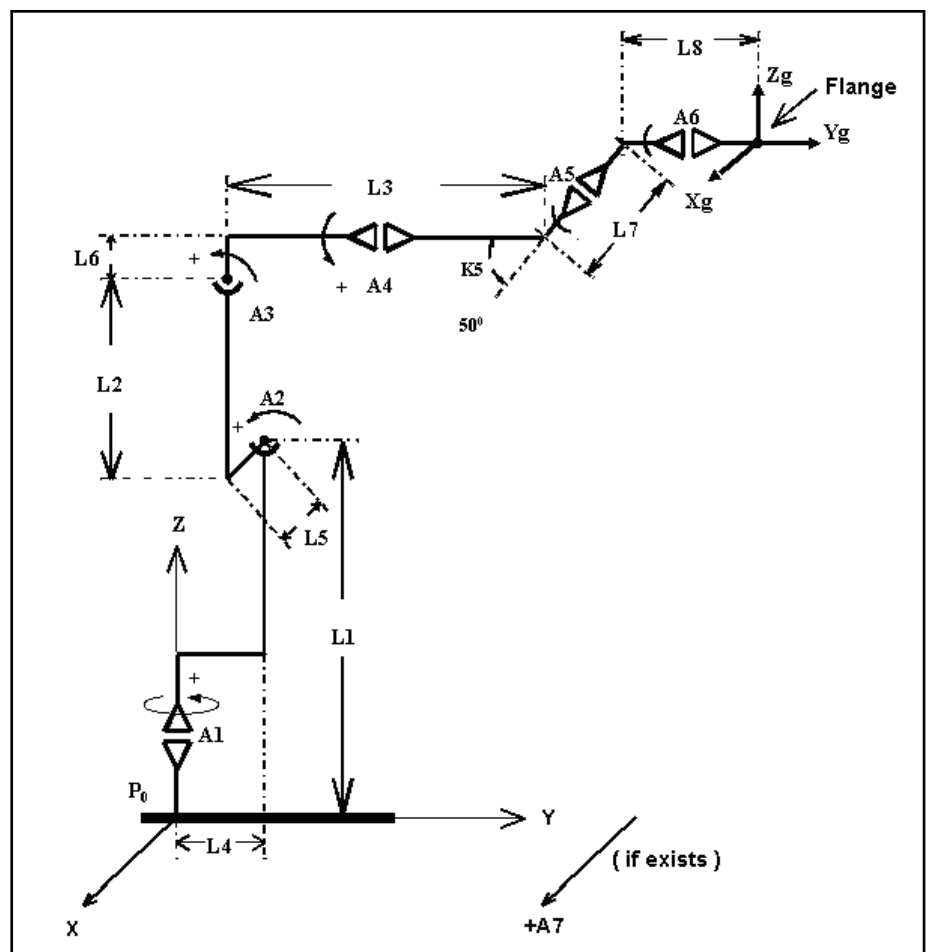


Fig. 18-53: Robot type 26, robot with offset wrist with 6 (7) axes, zero position, schematic

Transformation Rules

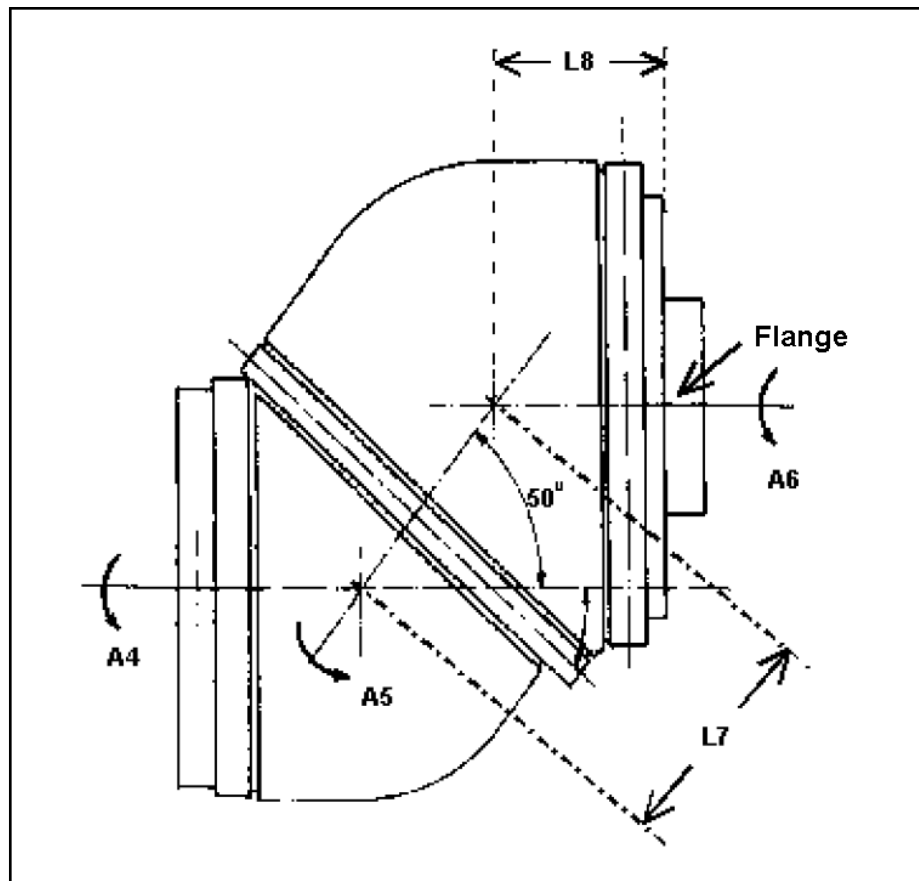


Fig. 18-54: Robot type 26, robot with offset wrist with 6 (7) axes, wrist

Zero position

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- exact determination of the coupling factors (if present)
- correct directions of rotation of the axes

⚠ CAUTION

If one of the these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed. This can cause incorrect and unexpected robot motions.

The respective world coordinate system is identified by the three main axes (x, y, z) in [fig. 18-53 "Robot type 26, robot with offset wrist with 6 \(7\) axes, zero position, schematic" on page 255](#). The zero point is at the base point P0 of the robot.



When using this transformation with `PROGR_SLOPE`, the path for spatial blending is not calculated by the control. The user has to program the tangential transitions at block switching.

Settings in the axis configuration list, axis assignment (K-0-0008)

The following settings have to be made in the kinematics and in the axis configuration:

Transformation Rules

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+/- 179 degrees
A2:	Positioning axis	Rotary absolute	+/- 179 degrees
A3:	Positioning axis	Rotary absolute	+/- 90 degrees
A4:	Orientation axis	Rotary absolute	Unlimited
A5:	Orientation axis	Rotary absolute	Unlimited
A6:	Orientation axis	Rotary absolute	Unlimited
A7:	Orientation axis	Translatory absolute	parallel to the belt (optional)

Fig. 18-55: K-0-0008, Axis configuration list, axis assignment

Transformation scheme, type
(K-0-0031)

26 has to be set as transformation scheme.

Referencing all axes (K-0-0032)

Select 1 as value for this parameter.

IPOS; update at coordinate
change (K-0-0033)

Select 1 as value for this parameter.

Transformation lengths, axis con-
figuration, axis lengths (K-0-0040)

Find all data in [fig. 18-53 "Robot type 26, robot with offset wrist with 6 \(7\) axes, zero position, schematic" on page 255:](#)

Transformation length	Meaning	Measuring unit	Restriction
L1:	Arm length L_1	mm or inch (see K-0-0054)	≥ 0
L2:	Arm length L_2	mm or inch (see K-0-0054)	> 0
L3:	Arm length L_3	mm or inch (see K-0-0054)	> 0
L4:	Eccentricity L_4	mm or inch (see K-0-0054)	Without restriction
L5:	Eccentricity L_5	mm or inch (see K-0-0054)	Without restriction
L6:	Eccentricity L_6	mm or inch (see K-0-0054)	Without restriction
L7:	Arm length L_7	mm or inch (see K-0-0054)	≥ 0
L8:	Arm length L_8	mm or inch (see K-0-0054)	≥ 0

Fig. 18-56: K-0-0040, Transformation lengths, axis configuration, axis lengths

The eccentricities L_4 , L_5 , L_6 can be positive or negative:

- $L_4 > 0$: Eccentricity in (+y) direction
- $L_4 < 0$: Eccentricity in (-y) direction
- $L_5 > 0$: Eccentricity in (+x) direction
- $L_5 < 0$: Eccentricity in (-x) direction
- $L_6 > 0$: Eccentricity in (+z) direction
- $L_6 < 0$: Eccentricity in (-z) direction

Transformation factors, axis con-
figuration, axis coupling
(K-0-0041)

Transformation Rules

Coupling factor	Meaning	Restriction
C1:	C ₄₅ mechanical coupling between the axes 4/5	Without restriction
C2:	C ₄₆ mechanical coupling between the axes 4/6	Without restriction
C3:	C ₅₆ mechanical coupling between the axes 5/6	Without restriction
C4:	C ₂₃ mechanical coupling between the axes 2/3	Without restriction
C5:	Bending angle of the 5th axis (50°)	Without restriction
C6:	Max. sum of the manual axes (1080°)	Without restriction
C7:	Abort accuracy for iteration	>= 0
C8:	Max. number of iteration steps + selection of the orientation definition + activation of the tensor interpolation	>= 0

Fig. 18-57: K-0-0041, Transformation factors, axis configuration, axis coupling

Coupling factors C1 to C4

If mechanical axis couplings exist, the respective coupling factors have to be determined. They are defined as follows:

$$C_{23} = \Delta J_3 / \Delta A_2$$

$$C_{45} = \Delta J_5 / \Delta A_4$$

$$C_{46} = \Delta J_6 / \Delta A_4$$

$$C_{56} = \Delta J_6 / \Delta A_5$$

A_i is used for the axis angle and J_i for the joint angle detected by the measuring system. That means that the coupling factor C_{ij} indicates the effect on the joint j when axis i changes without notifying the measuring system j about this change.

Coupling factor C5

Angle by which the robot hand (50°) is bended, [fig. 18-53 "Robot type 26, robot with offset wrist with 6 \(7\) axes, zero position, schematic" on page 255](#)

Coupling factor C6

Max. sum of the three manual axes (1080°) If the value is exceeded, there is a motion abort with the message "manual axis rotation exceeded"

Coupling factor C7

Abort accuracy in degrees to iterate the inverse position kinematics. If the value is smaller than 1 (e.g. 0.01), the same abort accuracy is applied to the position (1, 2, 3) and orientation axes (4, 5, 6). If the value is higher than 1, different abort criteria for position and orientation axes can be applied: The decimal values specifies the abort accuracy. The abort accuracy of the orientation axes results from the decimal value multiplied by the integer value (e.g. 20.01 => position accuracy = 0.01° orientation accuracy = 20 x 0.01° = 0.2°)

Coupling factor C8

The first and the second place of C8 (indicates the maximum number of iterations allowed (99 max., 8 recommended). If no convergence has been reached after the set number of iterations, the motion is aborted and the message "Iteration divergent" is output. The third and fourth place of C8 (WC orientation type) specifies the definition of the orientation angle (world coordinates 4, 5, 6). If the fifth place of C8 is 1, the tensor interpolation is enabled.

Example:

- C8 = 8
 - Maximum of eight iterations
 - WC orientation type 0
 - No tensor interpolation

Transformation Rules

- C8 = 312
 - maximum of twelve iterations
 - WC orientation type 3
 - No tensor interpolation
- C8 = 10807
 - Maximum of seven iterations
 - WC orientation type 8
 - With tensor interpolation

The following definition applies for the WC orientation type = 0, 4:

- WC[4] = rotation about the spatially fixed z-axis
- WC[5] = rotation about the resulting y'-axis
- WC[6] = rotation about the resulting x''-axis

The following definition applies for the WC orientation type = 1, 5:

- WC[4] = rotation about the spatially fixed z-axis
- WC[6] = rotation about the resulting x'-axis
- WC[5] = rotation about the resulting z''-axis

The following definition applies for the WC orientation type = 2, 6:

- WC[4] = rotation about the spatially fixed z-axis
- WC[5] = rotation about the resulting x'-axis
- WC[6] = rotation about the resulting y''-axis

The following definition applies for the WC orientation type = 3, 7:

- WC[4] = rotation about the spatially fixed y-axis
- WC[5] = rotation about the resulting x'-axis
- WC[6] = rotation about the resulting z''-axis

The following definition (Euler angle) applies for the WC orientation type = 8:

- WC[4] = rotation about the spatially fixed x-axis
- WC[5] = rotation about the resulting y'-axis
- WC[6] = rotation about the resulting z''-axis

The world coordinates 4, 5 and 6 are not restricted for the WC orientation types 0, 1, 2 and 3. The types 4, 5, 6 and 7 are subject to the following restrictions:

- $-90^\circ < Wc[4] < 90^\circ$
- $-180^\circ < Wc[5] < 180^\circ$
- $-180^\circ < Wc[6] < 180^\circ$

The following limitations apply for the "classic" Euler angles (type 8):

- $-180^\circ < Wc[4] < 180^\circ$
- $0^\circ < Wc[5] < 180^\circ$
- $-180^\circ < Wc[6] < 180^\circ$

The tensor interpolation is enabled via a 5-digit C₈. The spatial orientations (WC[4], WC[5], WC[6]) at linear motions are interpolated via a rotary sensor in automatic mode. For the tensor interpolation, one of the restricted WC ori-

Transformation Rules

entation types has to be selected via the coupling factor C8 (i.e. WC orientation type).

18.13 Robot Type 27, Robot with Offset Wrist with Six Axes + Three Linear Axes

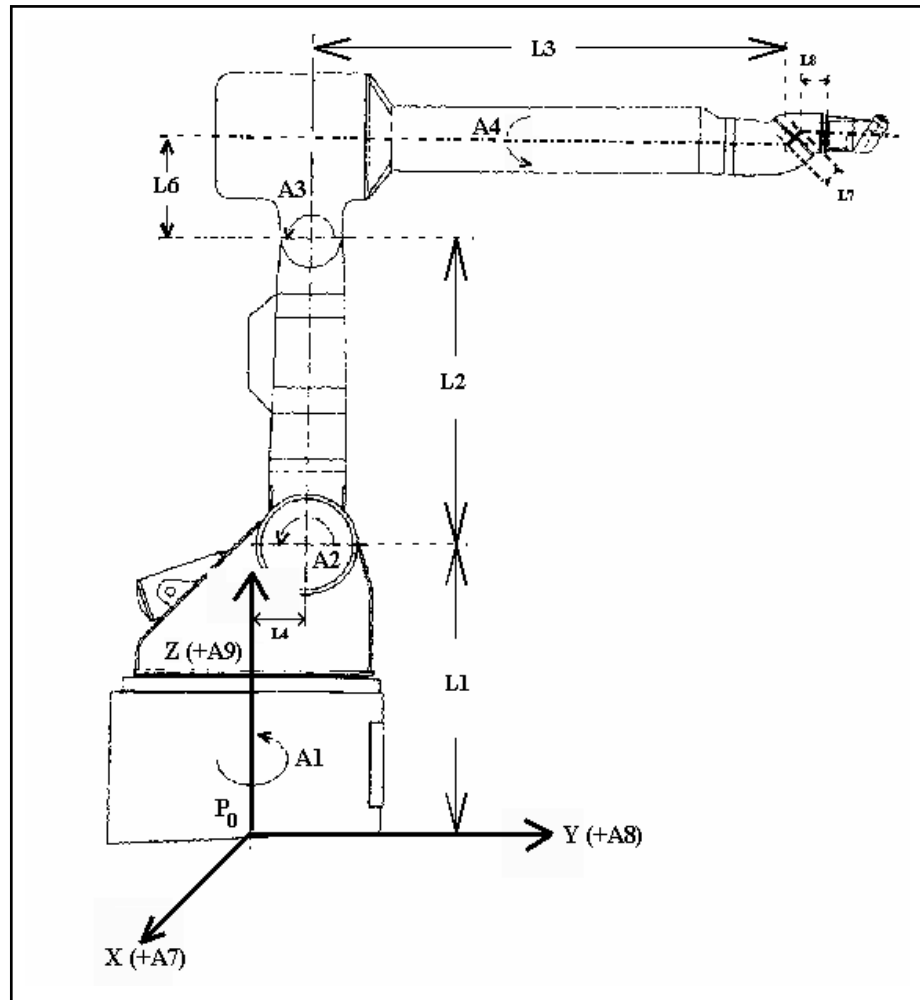


Fig. 18-58: Robot type 27, robot with offset wrist with 6 axes + 3 linear axes, zero position

Transformation Rules

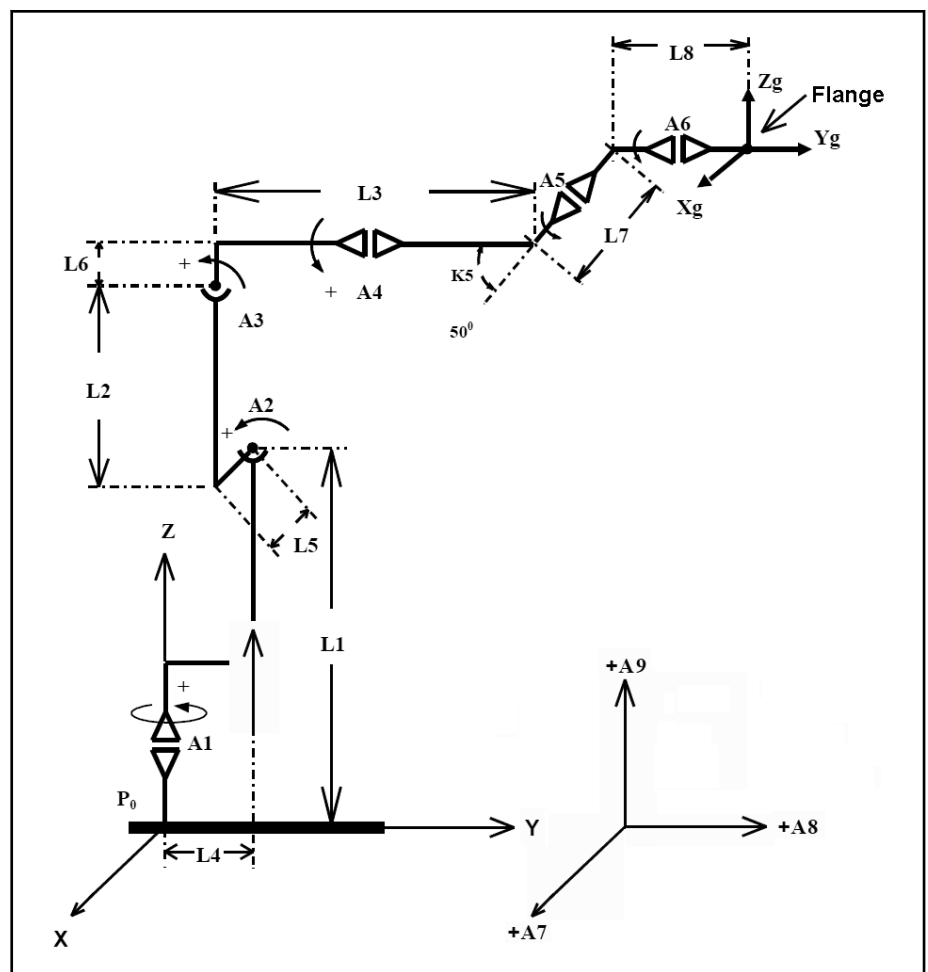


Fig. 18-59: Robot type 27, robot with offset wrist with 6 axes + 3 linear axes, zero position, schematic

Transformation Rules

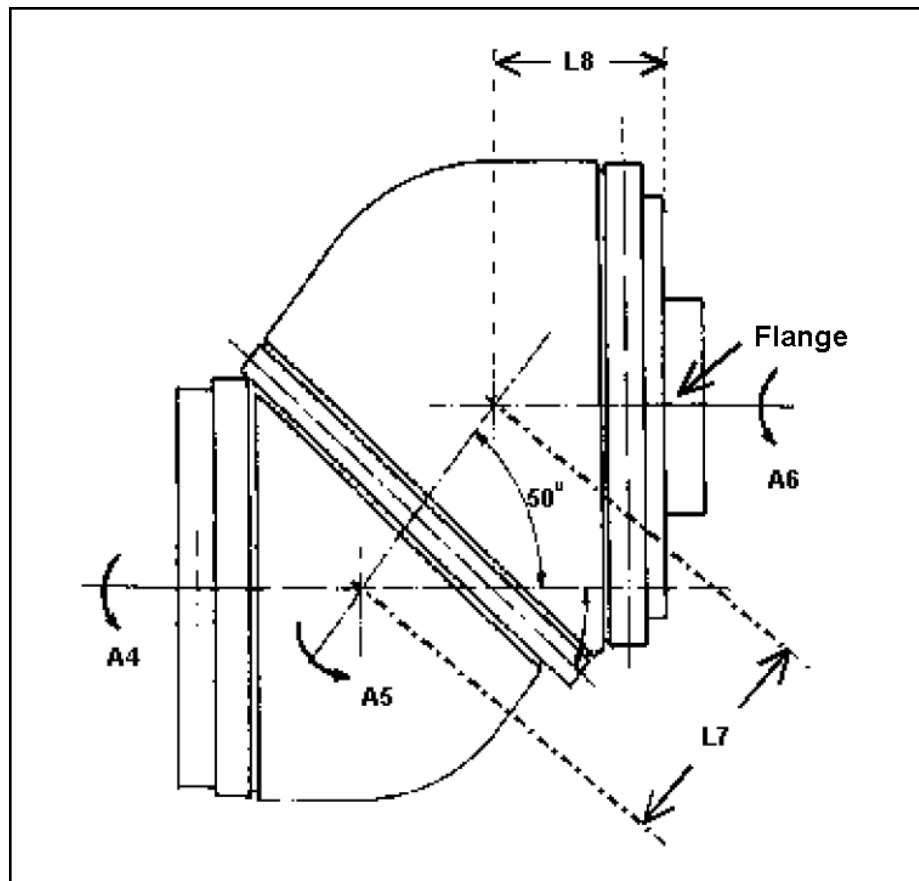


Fig. 18-60: Robot type 27, robot with offset wrist with 6 axes + 3 linear axes, zero position, wrist

Zero position

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- exact determination of the coupling factors (if present)
- correct directions of rotation of the axes

⚠ CAUTION

If one of these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed. This can cause incorrect and unexpected robot motions.

The respective world coordinate system is identified by the three main axes (x, y, z) in [fig. 18-59 "Robot type 27, robot with offset wrist with 6 axes + 3 linear axes, zero position, schematic" on page 261](#). The zero point is at the base point P0 of the robot.



When using this transformation with `PROGR_SLOPE`, the path for spatial blending is not calculated by the control. The user has to program the tangential transitions at block switching.

Settings in the axis configuration list, axis assignment (K-0-0008)

The following settings have to be made in the kinematics and in the axis configuration:

Transformation Rules

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+/- 179 degrees
A2:	Positioning axis	Rotary absolute	+/- 179 degrees
A3:	Positioning axis	Rotary absolute	+/- 90 degrees
A4:	Orientation axis	Rotary absolute	Unlimited
A5:	Orientation axis	Rotary absolute	Unlimited
A6:	Orientation axis	Rotary absolute	Unlimited
A7:	Orientation axis	Translatory absolute	Parallel to the x-axis
A8:	Orientation axis	Translatory absolute	parallel to the y-axis
A9:	Orientation axis	Translatory absolute	Parallel to the z-axis

Fig. 18-61: K-0-0008, Axis configuration list, axis assignment

Transformation scheme, type
(K-0-0031)

Referencing all axes (K-0-0032)

IPOS; update at coordinate
change (K-0-0033)

Transformation lengths, axis con-
figuration, axis lengths (K-0-0040)

27 has to be set as transformation scheme.

Select 1 as value for this parameter.

Select 1 as value for this parameter.

Find all data in [fig. 18-59 "Robot type 27, robot with offset wrist with 6 axes + 3 linear axes, zero position, schematic"](#) on page 261:

Transformation length	Meaning	Measuring unit	Restriction
L1:	Arm length L_1	mm or inch (see K-0-0054)	≥ 0
L2:	Arm length L_2	mm or inch (see K-0-0054)	> 0
L3:	Arm length L_3	mm or inch (see K-0-0054)	> 0
L4:	Eccentricity L_4	mm or inch (see K-0-0054)	Without restriction
L5:	Eccentricity L_5	mm or inch (see K-0-0054)	Without restriction
L6:	Eccentricity L_6	mm or inch (see K-0-0054)	Without restriction
L7:	Arm length L_7	mm or inch (see K-0-0054)	≥ 0
L8:	Arm length L_8	mm or inch (see K-0-0054)	≥ 0

Fig. 18-62: K-0-0040, Transformation lengths, axis configuration, axis lengths

The eccentricities L_4 , L_5 , L_6 can be positive or negative:

- $L_4 > 0$: Eccentricity in (+y) direction
- $L_4 < 0$: Eccentricity in (-y) direction
- $L_5 > 0$: Eccentricity in (+x) direction
- $L_5 < 0$: Eccentricity in (-x) direction
- $L_6 > 0$: Eccentricity in (+z) direction
- $L_6 < 0$: Eccentricity in (-z) direction

Transformation Rules

Transformation factors, axis configuration, axis coupling
(K-0-0041)

Coupling factor	Meaning	Restriction
C1:	C ₄₅ mechanical coupling between the axes 4/5	Without restriction
C2:	C ₄₆ mechanical coupling between the axes 4/6	Without restriction
C3:	C ₅₆ mechanical coupling between the axes 5/6	Without restriction
C4:	C ₂₃ mechanical coupling between the axes 2/3	Without restriction
C5:	Bending angle of the 5th axis (50°)	Without restriction
C6:	Variant of the additional linear axes	Without restriction
C7:	Abort accuracy for iteration	>=0
C8:	Max. number of iteration steps + selection of the orientation definition + activation of the tensor interpolation	>=0

Fig. 18-63: K-0-0041, Transformation factors, axis configuration, axis coupling

Coupling factors C1 to C4 If mechanical axis couplings exist, the respective coupling factors have to be determined. They are defined as follows:

$$C_{23} = \Delta J_3 / \Delta A_2$$

$$C_{45} = \Delta J_5 / \Delta A_4$$

$$C_{46} = \Delta J_6 / \Delta A_4$$

$$C_{56} = \Delta J_6 / \Delta A_5$$

A_i is used for the axis angle and J_i for the joint angle detected by the measuring system. That means that the coupling factor C_{ij} indicates the effect on the joint j when axis i changes without notifying the measuring system j about this change.

Coupling factor C5 Angle by which the robot hand (50°) is bended, [fig. 18-59 "Robot type 27, robot with offset wrist with 6 axes + 3 linear axes, zero position, schematic" on page 261](#)

Coupling factor C6 Variants of additional linear axes

Entry in C6	Number of axes	Additional linear axes
0	6	Without additional linear axis
-11	7	Additional linear axis in x-direction
-12	7	Additional linear axis in y-direction
-13	7	Additional linear axis in z-direction
-21	8	Additional linear axes in x- and y-direction
-22	8	Additional linear axes in x- and z-direction
-23	8	Additional linear axes in y- and z-direction
10	9	Additional linear axes in x-direction, y- and z-direction are virtual axes
11	9	Additional linear axes in x-direction, y- and z-direction are virtual axes

Transformation Rules

12	9	Additional linear axes in y-direction, x- and z-direction are virtual axes
13	9	Additional linear axis in z-direction, x- and y-direction are virtual axes
21	9	Additional linear axes in x- and y-direction, z-direction is a virtual axis
22	9	Additional linear axes in x- and z-direction, y-direction is a virtual axis
23	9	Additional linear axes in y- and z-direction, x-direction is a virtual axis
31	9	Additional linear axes in x-, y- and z-direction
Others	6	No additional axis available

Fig. 18-64: Variants of additional linear axes

Virtual axes have to be formally created. If points are specified, they have to be carried along as coordinates with a value of 0.

Coupling factor C7 Abort accuracy in degrees to iterate the inverse position kinematics. If the value is smaller than 1 (e.g. 0.01), the same abort accuracy is applied to the position (1, 2, 3) and orientation axes (4, 5, 6). If the value is higher than 1, different abort criteria for position and orientation axes can be applied: The decimal values specifies the abort accuracy. The abort accuracy of the orientation axes results from the decimal value multiplied by the integer value (e.g. 20.01 => position accuracy = 0.01° orientation accuracy = 20 x 0.01° = 0.2°)

Coupling factor C8 The first and the second place of C8 (indicates the maximum number of iterations allowed (99 max., 8 recommended). If no convergence has been reached after the set number of iterations, the motion is aborted and the message "Iteration divergent" is output. The third and fourth place of C8 (WC orientation type) specifies the definition of the orientation angle (world coordinates 4, 5, 6). If the fifth place of C8 is 1, the tensor interpolation is enabled.

Example:

- C8 = 8
 - Maximum of eight iterations
 - WC orientation type 0
 - No tensor interpolation
- C8 = 312
 - maximum of twelve iterations
 - WC orientation type 3
 - No tensor interpolation
- C8 = 10807
 - Maximum of seven iterations
 - WC orientation type 8
 - With tensor interpolation

The following definition applies for the WC orientation type = 0, 4:

- WC[4] = rotation about the spatially fixed z-axis
- WC[5] = rotation about the resulting y'-axis
- WC[6] = rotation about the resulting x"-axis

The following definition applies for the WC orientation type = 1, 5:

- WC[4] = rotation about the spatially fixed z-axis
- WC[6] = rotation about the resulting x'-axis
- WC[5] = rotation about the resulting z"-axis

Transformation Rules

The following definition applies for the WC orientation type = 2, 6:

- WC[4] = rotation about the spatially fixed z-axis
- WC[5] = rotation about the resulting x'-axis
- WC[6] = rotation about the resulting y''-axis

The following definition applies for the WC orientation type = 3, 7:

- WC[4] = rotation about the spatially fixed y-axis
- WC[5] = rotation about the resulting x'-axis
- WC[6] = rotation about the resulting z''-axis

The following definition (Euler angle) applies for the WC orientation type = 8:

- WC[4] = rotation about the spatially fixed x-axis
- WC[5] = rotation about the resulting y'-axis
- WC[6] = rotation about the resulting z''-axis

The world coordinates 4, 5 and 6 are not restricted for the WC orientation types 0, 1, 2 and 3. The types 4, 5, 6 and 7 are subject to the following restrictions:

- $-90^\circ < Wc[4] < 90^\circ$
- $-180^\circ < Wc[5] < 180^\circ$
- $-180^\circ < Wc[6] < 180^\circ$

The following limitations apply for the "classic" Euler angles (type 8):

- $-180^\circ < Wc[4] < 180^\circ$
- $0^\circ < WC[5] < 180^\circ$
- $-180^\circ < Wc[6] < 180^\circ$

The tensor interpolation is enabled via a 5-digit C_8 . The spatial orientations (WC[4], WC[5], WC[6]) at linear motions are interpolated via a rotary sensor in automatic mode. For the tensor interpolation, one of the restricted WC orientation types has to be selected via the coupling factor C8 (i.e. WC orientation type).

18.14 Robot Type 29, 2-Axis Delta with Rotary and Translatory Axes

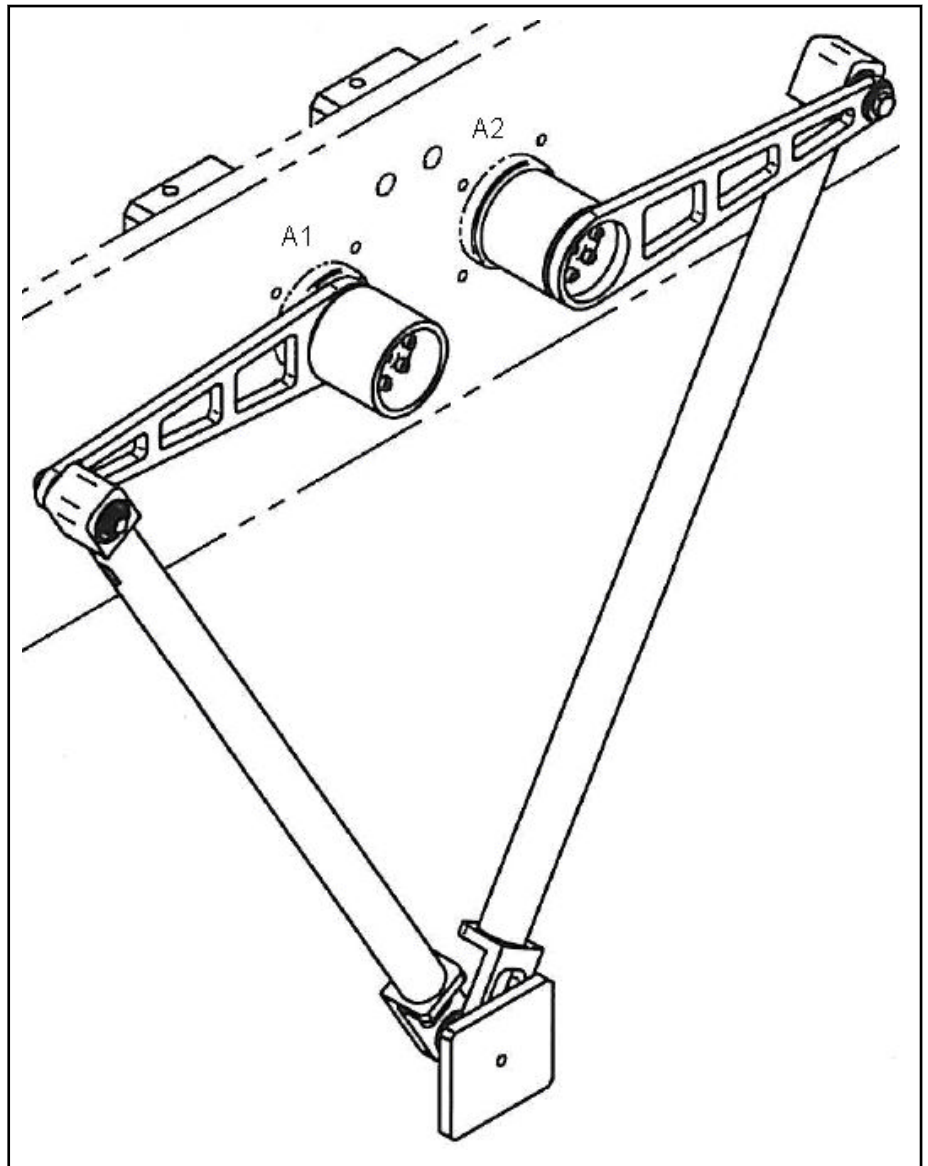


Fig. 18-65: Robot type 29, 2-axis delta

Transformation Rules

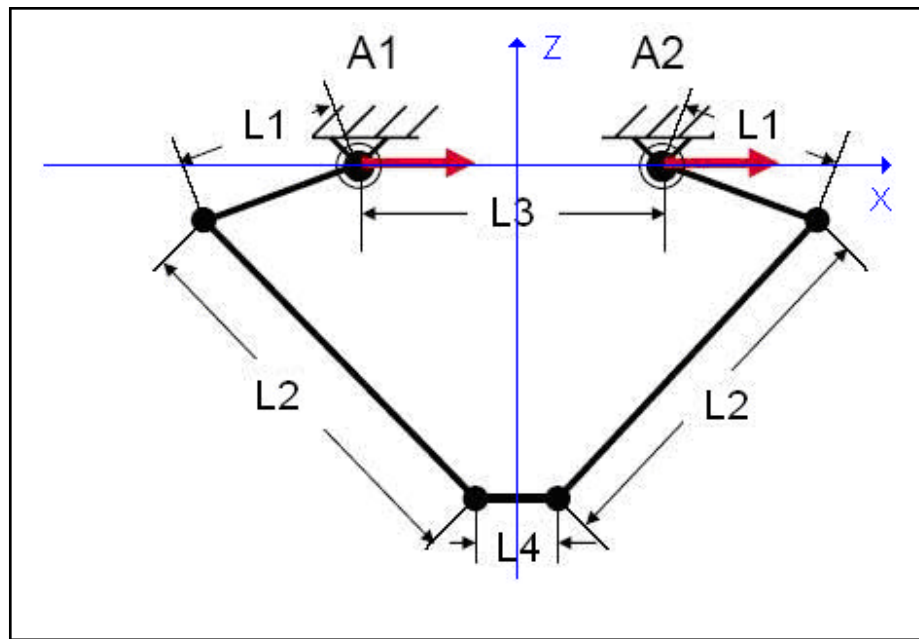


Fig. 18-66: Robot type 29, 2-axis delta (X-Z view)

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+ 90 .. 270 degrees
A2:	Positioning axis	Rotary absolute	+/- 90 degrees
A3:	Positioning axis	Translatory absolute	Optional
A4:	Orientation axis	Rotary abs./modulo	Optional
A5:	Belt axis (optional)	Translatory, modulo	Optional
A6:	Belt axis (optional)	Translatory, modulo	Optional
A7:	Belt axis (optional)	Translatory, modulo	Optional
A8:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-67: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, transformation scheme, Type = 29 (2-axis delta, bipod)
- K-0-0032, Referencing all axes = 0
- K-0-0033, IPOS, update at coordinate change = 0
- K-0-0036, WC axis monitoring = 1

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of the arm at the motor	mm or inch (see K-0-0054)	>0
L2:	Length of rod to mold plate	mm or inch (see K-0-0054)	>0
L3:	Distance of the two rotation points of the arms	mm or inch (see K-0-0054)	>0
L4:	Distance of rods at base plate	mm or inch (see K-0-0054)	>0

Fig. 18-68: K-0-0040, Transformation lengths, axis configuration, axis lengths

Transformation Rules

K-0-0041, Transformation factors, axis configuration, axis coupling, is not used

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cz)	
Coordinate 3:	Name of 3rd coordinate (e.g. Cy)	
Coordinate 4:	Name of the 4th coordinate (e.g. Cc)	
Coordinate 5:	Name of the 5th coordinate (e.g. Kbelt1)	Optional
Coordinate 6:	Name of the 6th coordinate (e.g. Kbelt2)	Optional
Coordinate 7:	Name of the 7th coordinate (e.g. Kbelt3)	Optional
Coordinate 8:	Name of the 8th coordinate (e.g. Kbelt4)	Optional

Fig. 18-69: K-0-0121 .. K-0-0128, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- Correct direction of rotation of the axes

If one of the these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

fig. 18-65 "Robot type 29, 2-axis delta" on page 267 shows a bipod robot in 3D with the axis positions $A1=180^\circ$ and $A2 = 0^\circ$.

In zero position, the axes 1 and 2 (positioning axes) are horizontal towards the x-axis (see red arrow in fig. 18-66 "Robot type 29, 2-axis delta (X-Z view)" on page 268).

- The arm length L_1 is measured from the rotary center point to the center point of the rod retainer.
- The rod length L_2 is measured from the reference points on both sides.
- L_3 is the distance of the rotary center point of both arms.
- L_4 is the length of the mold plate measured at the rod mountings.
- Positive counterclockwise rotation of the positioning axes A1 and A2.
- The Cartesian zero point of the bipod robot is in the center of both arm rotation points A1 and A2.

Transformation Rules

18.15 Robot Type 30, Adabot

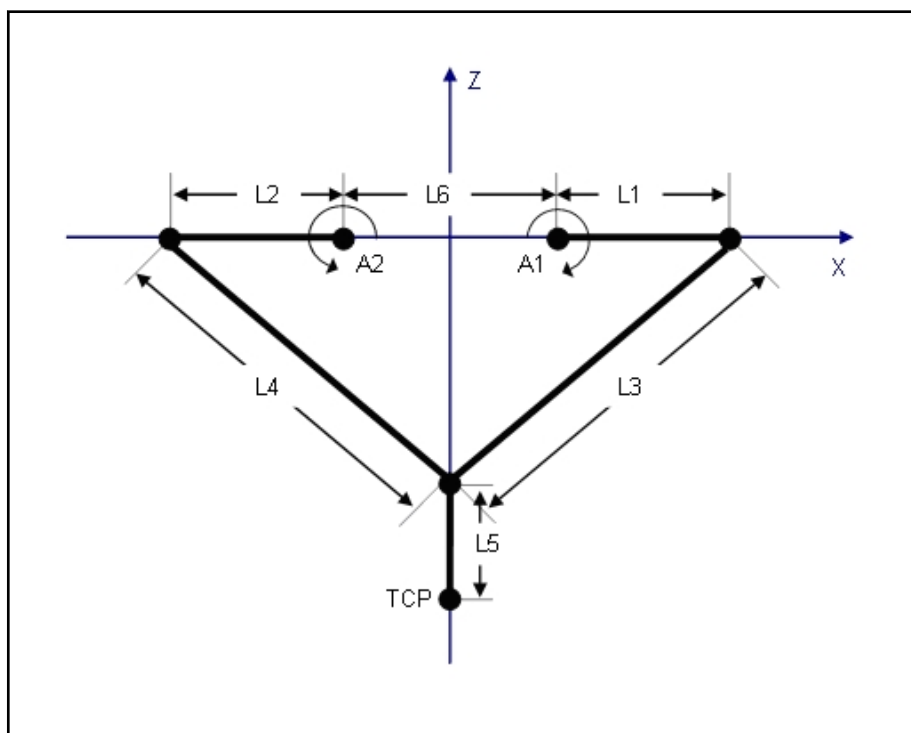


Fig. 18-70: Robot type 30, Adabot (X-Z view, axis zero positioning)

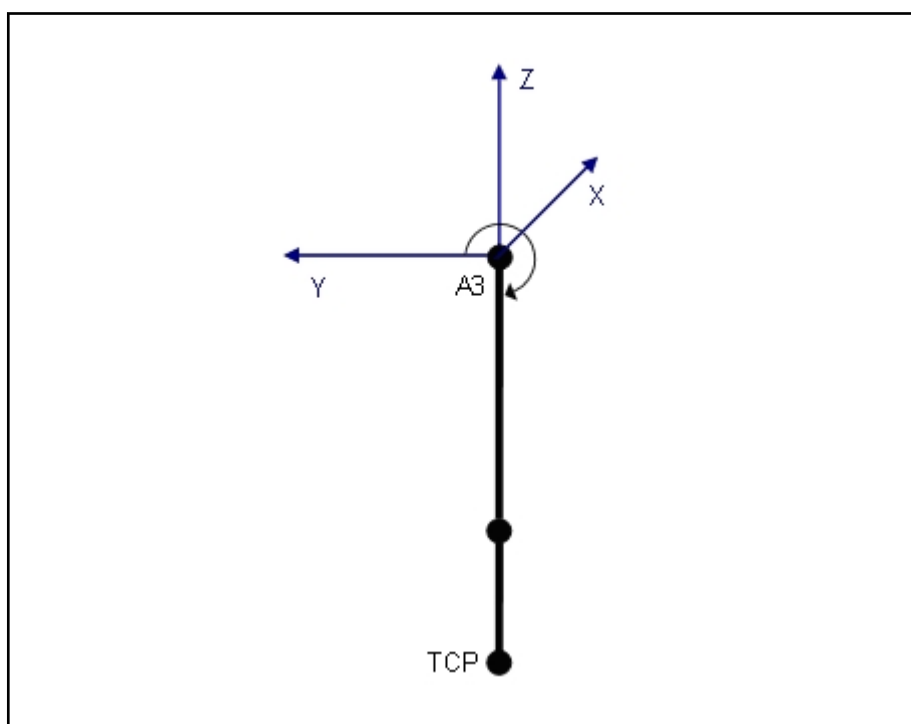


Fig. 18-71: Robot type 30, Adabot (X-Z view, axis zero positioning)

Transformation Rules

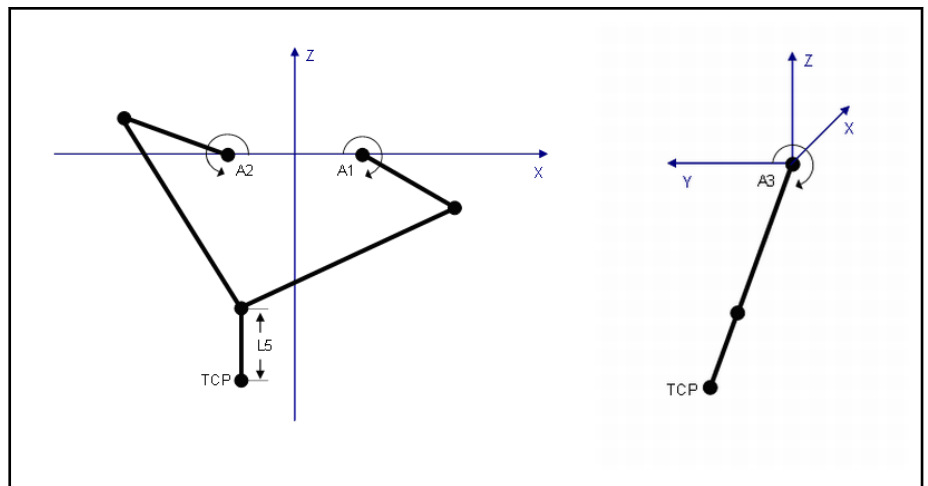


Fig. 18-72: Robot type 30, Adabot (behavior of rod L5)

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+/- 90 degrees
A2:	Positioning axis	Rotary absolute	+/- 90 degrees
A3:	Positioning axis	Rotary absolute	+/- 90 degrees
A4:	Orientation axis	Rotary abs./modulo	Optional
A5:	Belt axis (optional)	Translatory, modulo	Optional
A6:	Belt axis (optional)	Translatory, modulo	Optional
A7:	Belt axis (optional)	Translatory, modulo	Optional
A8:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-73: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, type = 30 (Adabot).
- K-0-0032, Referencing all axes = 0

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of arm at motor	mm or inch (see K-0-0054)	>0
L2:	Length of arm at motor (left)	mm or inch (see K-0-0054)	>0
L3:	Length of rod at arm L1	mm or inch (see K-0-0054)	>0
L4:	Length of rod at arm L2	mm or inch (see K-0-0054)	>0
L5:	Rod to TCP	mm or inch (see K-0-0054)	>=0
L6:	Distance of the two rotation points of the arms	mm or inch (see K-0-0054)	>0

Fig. 18-74: K-0-0040, Transformation lengths, axis configuration, axis lengths

K-0-0041, Transformation factors, axis configuration, axis coupling, is not used

Transformation Rules

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cy)	
Coordinate 3:	Name of the 3rd coordinate (e.g. Cz)	
Coordinate 4:	Name of the 4th coordinate (e.g. Cc)	Optional
Coordinate 5:	Name of the 5th coordinate (e.g. Kbelt1)	Optional
Coordinate 6:	Name of the 6th coordinate (e.g. Kbelt2)	Optional
Coordinate 7:	Name of the 7th coordinate (e.g. Kbelt3)	Optional
Coordinate 8:	Name of the 8th coordinate (e.g. Kbelt4)	Optional

Fig. 18-75: K-0-0121 .. K-0-0136, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- Correct direction of rotation of the axes

If one of these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

[fig. 18-70 "Robot type 30, Adabot \(X-Z view, axis zero positioning\)" on page 270](#) and [fig. 18-71 "Robot type 30, Adabot \(X-Z view, axis zero positioning\)" on page 270](#) show the schematic kinematic representation of the Adabot kinematics for zero positioning of all axes.

- The figures also show the arm lengths L_1 , L_2 , L_3 , L_4 , L_5 , L_6 .
- The positive directions of rotation of the axes A1, A2, A3 are also shown.
- The lengths L_1 , L_2 , L_3 , L_4 , L_6 always have to be entered as positive number and unequal to zero. L_5 always has to be specified as positive number or zero.

The rod to TCP (L_5) is always held vertically to the x-axis by the mechanics (not represented in the figure). The rod to TCP (L_5) is always in the plane spanned by the rods L_1 , L_2 , L_3 , L_4 (see [fig. 18-72 "Robot type 30, Adabot \(behavior of rod L5\)" on page 271](#)).

The Cartesian zero point of the Adabot kinematics is in the center of both arm rotation points A1 and A2 (see [fig. 18-70 "Robot type 30, Adabot \(X-Z view, axis zero positioning\)" on page 270](#)).

18.16 Robot Type 32, 2-Axis Delta with Asymmetric TCP

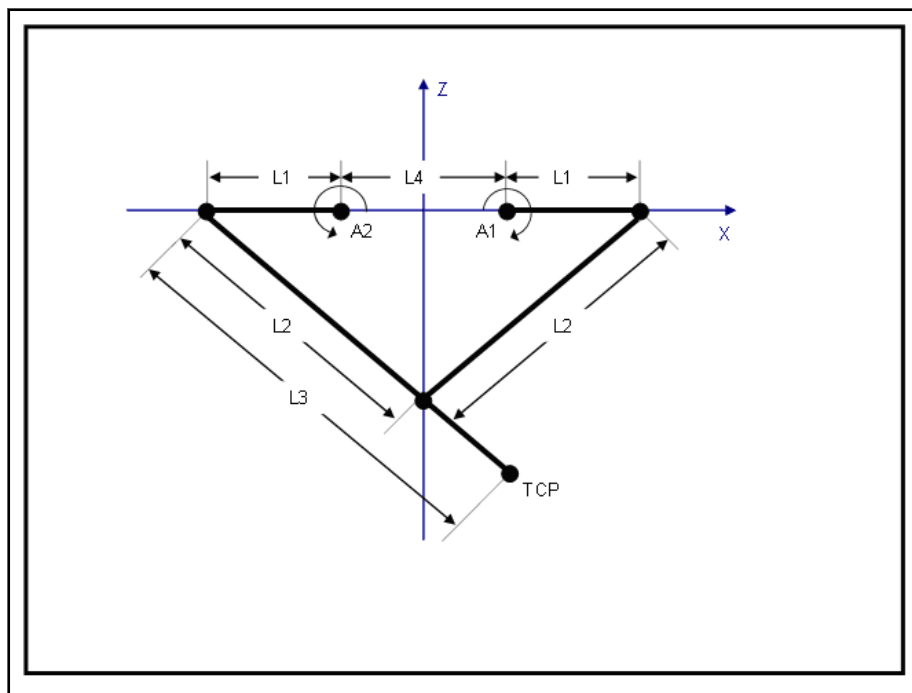


Fig. 18-76: Robot type 32, 2-axis delta with asymmetric TCP (X-Z view, axis zero positioning)

Parameter settings

Axis name	Axis type	Scaling	Remark
A1:	Positioning axis	Rotary absolute	+/- 90 degrees
A2:	Positioning axis	Rotary absolute	+/- 90 degrees
A3:	Orientation axis	rotary abs. /modulo	Optional
A4:	Belt axis (optional)	Translatory, modulo	Optional
A5:	Belt axis (optional)	Translatory, modulo	Optional
A6:	Belt axis (optional)	Translatory, modulo	Optional
A7:	Belt axis (optional)	Translatory, modulo	Optional

Fig. 18-77: K-0-0008, Axis configuration list, axis assignment

- K-0-0031, Transformation scheme, type = 32
- K-0-0032, Referencing all axes = 0

Transformation length	Axis configuration	Measuring unit	Axis length
L1:	Length of the arm at the motor	mm or inch (see K-0-0054)	>0
L2:	Length of short rod	mm or inch (see K-0-0054)	>0

Transformation Rules

L3:	Length of the long rod to the TCP	mm or inch (see K-0-0054)	>L2
L4:	Distance of the two rotation points of the arms	mm or inch (see K-0-0054)	>0

Fig. 18-78: K-0-0040, Transformation lengths, axis configuration, axis lengths

K-0-0041, Transformation factors, axis configuration, axis coupling, is not used

Name	Coordinate	Remark
Coordinate 1:	Name of the 1st coordinate (e.g. Cx)	
Coordinate 2:	Name of the 2nd coordinate (e.g. Cz)	
Coordinate 3:	Name of the 3rd coordinate (e.g. Cc)	Optional
Coordinate 4:	Name of the 4th coordinate (e.g. Kbelt1)	Optional
Coordinate 5:	Name of the 5th coordinate (e.g. Kbelt2)	Optional
Coordinate 6:	Name of the 6th coordinate (e.g. Kbelt3)	Optional
Coordinate 7:	Name of the 7th coordinate (e.g. Kbelt4)	Optional

Fig. 18-79: K-0-0121 .. K-0-0136, Coordinate names, coordinates

Axis directions of rotation and zero positions

The basic requirements to perform the coordinate transformation correctly are:

- Exact setting of the axis zero points
- Exact measurement of the arm lengths or center offsets
- Correct direction of rotation of the axes

If one of the these requirements is not completely fulfilled, a precise linear motion cannot be guaranteed.

This can result in incorrect or unexpected motions.

fig. 18-76 "Robot type 32, 2-axis delta with asymmetric TCP (X-Z view, axis zero positioning)" on page 273 shows the schematic representation of the 2-axis delta kinematics with asymmetric TCP when all axes are in zero position.

- The figures also show the lengths L_1 , L_2 , L_3 , L_4 .
- The positive directions of rotation of the axes A1, A2, A3 are also shown.
- The lengths L_1 , L_2 , L_3 , L_4 always have to be entered as positive number or unequal to zero. The value of L_3 always has to be greater than of L_2 .
- The Cartesian zero point is in the center of the two arm rotation points A1 and A2.

18.17 Robot Type 100, PLC

Function It often occurs in the application that the transformations provided by the system do not fit the kinematics. Therefore, the system provides the user-defined transformation type with the number 1100. The conversion rules from world to axis coordinates and vice versa are implemented in the PLC for this type. To define such a transformation, the interface MC_KIN_REF has to be implemented by a function block.

Transformation Rules

Then, the transformation has to be registered at the system. This is done using the function block MC_SetKinTransform.

The function pointers have to be transferred to the individual functions of the interface MC_KIN_REF as follows to the function block via the structure ML_KIN_AXES_CONFIG_DATA / ML_KIN_AXIS_CONFIG.

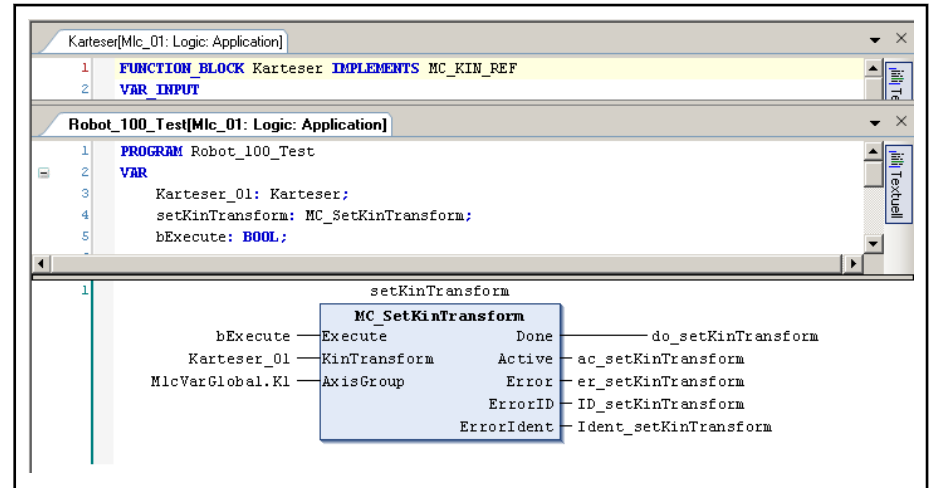


Fig. 18-80: Registering a transformation

After a transformation has been registered at the system, the transformation is used by the system.

Details The required function block, the interface and the structures are part of the library ML_Robot_xx.library.

The description of this library is part of the documentation:

Rexroth IndraLogic XLC IndraMotion MLC 12VRS PLCopen Libraries, DOK-XLCMLC-FUNLIB**V12-L1xx-EN-P, R911334182.

- Parameter settings**
- K-0-0031, Transformation scheme, type = 1100
 - K-0-0040, Transformation, lengths of the mechanics
 - K-0-0041, Transformation, factors

19 RCL Keywords

In the following, all the language symbols currently reserved for the RCL are listed. The language symbols listed may not be used in an RCL program as variables, file names or subroutine names.

Keyword	Remark
@	Joint coordinate points. See JC_POINT in Composed RCL Data Types , page 55..
@MPOS	Reserved
@POS	Current position in joint coordinates, see also Reading the actual position POS, @POS , page 61,
A	Path acceleration, see also A , page 123.
ABS	Function provides the value of a REAL or INTEGER number, ABS Function , page 89.
AFACTOR	Path acceleration factor, see also AFACTOR , page 124.
AFIX	AFIX , page 121.
ALSO	Reserved
AND	For logical AND operation bit by bit, see also Logic Expressions , page 83.
APPROX	See MOVE_REL APPROX... , page 112.
ARRAY	Working with Array Variables (Arrays) , page 62,
ASC_INT	Data conversion from CHAR to INTEGER, ASC_INT Function , page 91.
ASSIGN	File access, ASSIGN Function , page 90.
ATAN	ATAN Function , page 87,
BEGIN	Start of program code, see also Statement Part , page 40.
BELT	Belt Synchronization , page 127., Communication Channels to PLC, General , page 64,
BINARY	Overview , page 53,
BLOCK_SLOPE	BLOCK_SLOPE Block Transitions (BLOCK_SLOPE, PROGR_SLOPE, CONT_SLOPE) , page 129,
BRN_FILE	Reserved
BREAK	Reserved
CASE	Case decision, CASE Statement , page 80.
CASE_END	
CHAR	RCL Data Types , page 53,
CHR	Data conversion, calculates a CHAR from INTEGER MOD 256, CHR Function , page 89.
CIRCULAR	Interpolation Method , page 114,
CLOSE	See also ASSIGN Function , page 90.

RCL Keywords

Keyword	Remark
CLS	Reserved
COMPILER	Mode of the Compiler Operation, page 35,
CONDITION	Reserved
CONST	RCL Constant Declaration , page 51,
CONT_SLOPE	Block Transitions (BLOCK_SLOPE, PROGR_SLOPE, CONT_SLOPE) , page 129,
CONTROL	Reserved
COS	COS Function, page 87,
DEBUGINFO	DEBUGINFO+, DEBUGINFO-, compiler statement, see also Compiler Statements, page 36.
DEF	Point variable in point table, RCL Variable Declaration, page 57.
DEFAULT	CASE Statement, page 80,
DFACTOR	Deceleration factor, see also DFACTOR, page 125.
ELSE	IF-THEN-(ELSE) Statement, page 77,
END	Program Declaration, page 40,
END_OF_FILE	EOF Function, page 90,
EQUAL	CASE Statement, page 80,
ERROR	Maximum waiting time, page 71,
EVERY	Reserved
EXACT	See MOVE_REL EXACT..., page 113.
EXCLUSIVE	Reserved
EXCLUSIVE_END	Reserved
EXTERNAL	Main Program Call in Main Program, page 41,
FILE	File work, see FILE, page 55.
HALT	HALT Statement, page 73,
IF	IF-THEN-(ELSE) Statement, page 77,
INCLUDE	Including Files, page 37,
INPUT	Interface to PLC, PLC outputs - RoCo inputs, RCL Variable Declaration, page 57..
INT	Global interpolation method , page 116,
INT_ASC	Function converts an integer value to an array of CHAR, INT_ASC Function, page 90.
INTEGER	RCL Data Types, page 53,
J	Path jerk, see also A, page 125.
JC (..)	Coordinate transformation, conversion of world coordinates to joint coordinates, JC Standard Function, page 93.

RCL Keywords

Keyword	Remark
JC_NAMES	Compiler statements, Axis Names (JC Name Definition) , page 37 .
JC_POINT	Point Variables , page 58 ,
JUMP	JUMP Statement , page 75 ,
KINEMATICS	Compiler statement, Kinematic Definition , page 36 .
LIMIT_MAX	Workspace limitations, LIMIT_MIN , LIMIT_MAX , LIMIT_OFF Functions , page 93 .
LIMIT_MIN	
LIMIT_OFF	
LINEAR	Interpolation Method , page 114 ,
MAX_TIME	Maximum waiting time , page 71 , see also WAIT.
MCP	Reserved
MOD	Modulo division, Arithmetic Expressions , page 81 .
MOVE	Motion command, MOVE , page 110 .
MOVE TO	Motion command, MOVE TO ... , page 111 .
MOVE VIA	Motion command, MOVE VIA ... (blending) , page 111 .
MOVE VIA..TO	Motion command, MOVE VIA ... TO ... , page 111 .
MOVE_REL	Motion command, MOVE_REL , page 112 .
MOVE_REL APPROX	Motion command, MOVE_REL APPROX... , page 112 .
MOVE_REL EXACT...	Motion command, MOVE_REL EXACT... , page 113 .
MOVE_REL APPROX.. EXACT	Motion command, MOVE_REL APPROX ... EXACT ... , page 113 .
NOT	Bit-wise negation, Logic Expressions , page 83 .
OR	Bit by bit OR operation, Logic Expressions , page 83 .
ORD	Data conversion, INTEGER value of a variable from CHAR type, ORD Function , page 89 .
OUTPUT	Interface to PLC, RoCo outputs, PLC inputs, RCL Variable Declaration , page 57 .
PARALLEL	Reserved
PARALLEL_END	Reserved
PAUSE	PAUSE Statement , page 73 ,
PERMANENT	Reserved
PLC	Reserved
PLC_PROCESS	Reserved
PLC_TIME	Reserved
POINT	Point Variables , page 58 ,
POS	Current position in world coordinates, Reading the actual position POS , @POS , page 61 .
POSE	Reserved

RCL Keywords

Keyword	Remark
PRIOR	Reserved
PROCESS_KIND	Reserved
PROGR_SLOPE	Block Transitions (BLOCK_SLOPE, PROGR_SLOPE, CONT_SLOPE) , page 129,
PROGRAM	Program Declaration, page 40,
PROGRAM_END	
PTP	Interpolation Method , page 114,
PUBLIC	Reserved
R	R, page 135.
R_PTP	R_PTP, page 135.
READ	See also ASSIGN Function, page 90., FILE, page 55,
READ_BEGIN	See also ASSIGN Function, page 90.
REAL	Overview, page 53,
RECORD	Record Types, RECORDs, page 56,
RECORD_END	
REF_PNT	Reserved
REPEAT	n: number of repetitions REPEAT Statement, page 74,
REPEAT n TIMES	
REPEAT_END	
RETURN	Subroutine Declaration, page 44,
RHO_FCT	Reserved
SEMAPHORE	Reserved
SIN	SIN Function, page 87,
SIZEOF	SIZEOF Function, page 92,
SPC_FCT	RCL Special Functions, Listed as Table, page 96,
SQRT	SQRT Root Function, page 88,
START	Reserved
STOP	Reserved
SUB_END	Subroutine Declaration, page 44,
SUBROUTINE	
SYNC	SYNC, page 127,
SYNCHRON	SYNCHRON, page 127,
SYNCHRON_END	SYNCHRON_END, page 127,
TEXT	Overview, page 53,
THEN	IF-THEN-(ELSE) Statement, page 77,

RCL Keywords

Keyword	Remark
TIMES	See REPEAT Statement , page 74.
TO	See MOVE TO ... , page 111.
TOOL	Reserved
TRUNC	Data conversion that uses truncation to transform the argument from the REAL type into an INTEGER type value, TRUNC Function , page 89.
TTY	Reserved
TYPE	User-Defined RCL Types , page 55,
UNTIL	See WAIT UNTIL , Querying channels/signals , page 67.
V	V , page 119.
V_PTP	V_PTP , page 120.
VALUE	Subroutine call, call by value, Subroutine Sequence , page 46.
VAR	Beginning of the variable definition, see Simple RCL Data Types , page 54.
VERSION	Reserved
VFACTOR	VFACTOR , page 121.
VFIX	VFIX , page 121.
VIA	See MOVE VIA ...(blending) , page 111.
WAIT	WAIT Statement , page 69,
WAIT UNTIL	
WARNING	Reserved
WC (..)	Coordinate transformation, conversion of joint coordinates to world coordinates, WC Standard Function , page 93.
WC_CALCULATION	Reserved
WC_FRAME	Reserved
WC_NAMES	Compiler statement, World Coordinate Names (WC Name Definition) , page 36.
WC_SYSTEM	Reserved
WC_UR	See WC_UR , page 51.
WITH	Statement-related velocity specification (WITH) , page 120,
WRITE	See also ASSIGN Function , page 90., FILE , page 55,
WRITE_BEGIN	See also ASSIGN Function , page 90.
WRITE_END	See also ASSIGN Function , page 90.

Fig. 19-1: Keywords for RCL programs

RCL compiler statements RCL compiler statements begin with ";;", followed by the actual control statement.

RCL Keywords

Control statement	Remark
CONTROL	Reserved
COMPILER	General Information, page 36,
DEBUGINFO	Debug Information, page 38,
FILE_ERROR	Reserved
INCLUDE	Including Files, page 37,
INT	Global interpolation method , page 116,
JC_NAMES	Axis Names (JC Name Definition), page 37,
KINEMATICS	Kinematic Definition, page 36,
WARNING	Reserved
WC_NAMES	World Coordinate Names (WC Name Definition), page 36,

Fig. 19-2: Control statements for the RCL compiler

RCL standard functions

Standard functions	Remark
ABS	Function provides the value of a REAL or INTEGER number, ABS Function, page 89.
AND	For logical AND operation bit by bit, see also Logic Expressions, page 83.
ASC_INT	Data conversion from CHAR to INTEGER, ASC_INT Function, page 91.
ASSIGN	File access, ASSIGN Function, page 90.
ATAN	ATAN Function, page 87,
BNR_FILE	Reserved
CHR	Data conversion, calculates a CHAR from INTEGER MOD 256, CHR Function, page 89.
CONDITION	Reserved
COS	COS Function, page 87,
END_OF_FILE	Query whether the file end has been reached when reading a dat file, EOF Function, page 90.
INT_ASC	Function converts an integer value to an array of CHAR, INT_ASC Function, page 90.
JC	Coordinate transformation, conversion of world coordinates to joint coordinates, JC Standard Function, page 93.
MOD	Modulo division, Arithmetic Expressions, page 81.
NOT	For logical negation bit by bit, see also Logic Expressions, page 83.
OR	For logical OR operation bit by bit, see also Logic Expressions, page 83.
ORD	Data conversion, INTEGER value of a variable of type CHAR, ORD Function, page 89.
PLC_PROCESS	Reserved

RCL Keywords

Standard functions	Remark
PLC_TIME	Reserved
ROUND	Converts REAL into INTEGER. Rounding is to the closest integer. ROUND Function, page 89 ,
SIN	SIN Function, page 87 ,
SIZEOF	Determination of the memory space required for any sort of RCL variables or types, SIZEOF Function, page 92 .
SQRT	SQRT Root Function, page 88 ,
TRUNC	Uses truncation to convert REAL into INTEGER. TRUNC Function, page 89 ,
WC	Coordinate transformation, conversion of joint coordinates to world coordinates, WC Standard Function, page 93 .
WC_CALCULATION	Reserved

Fig. 19-3: RCL standard functions

RCL Special Functions

SPC_FCT	Brief functional description
1	IOL , page 98 , (accurate position switching of digital outputs on the path)
2	PPO , page 99 , (accurate position switching of process data on the path)
21	belt_kind , page 100 , (belt type)
23	DateTime , page 101 , (system date and time)
24	SysClk , page 101 , (system counter, real-time counter)
28	set_belt , page 102 , (set belt counter reset value) See SYNC
53	belt_area , page 102 , (belt range with belt type 4)
59	getBlendMode , page 106 , (query blending mode)
60	setBlendMode , page 105 , (set blending mode)

Fig. 19-4: RCL Special Functions

RCL standard constants

Standard constants are currently in limited use.

Standard constants	Remark
CLS	Reserved
VERSION	Reserved
WC_UR	See WC_UR , page 51 .

Fig. 19-5: RCL standard constants

RCL standard variables

Standard variables	Remark
@	Joint coordinate points See JC_POINT in Composed RCL Data Types, page 55..
@MPOS	Reserved

RCL Keywords

Standard variables	Remark
@POS	Current position in joint coordinates Reading the actual position POS, @POS, page 61,
A	A, page 123.
AFACTOR	AFACTOR, page 124.
AFIX	AFIX, page 121.
DFACTOR	DFACTOR, page 125
J	Path jerk, see also A, page 125.
LIMIT_MAX	Workspace limitations, LIMIT_MIN, LIMIT_MAX, LIMIT_OFF Functions, page 93.
LIMIT_MIN	
LIMIT_OFF	
MCP	Reserved
PLC	Reserved
POS	Current position in world coordinates Reading the actual position POS, @POS, page 61,
R	R, page 135.
R_PTP	R_PTP, page 135.
TTY	Reserved
V	V, page 119.
V_PTP	V_PTP, page 120.
VFACTOR	VFACTOR, page 121.
VFIX	VFIX, page 121.
WC_SYSTEM	Reserved

Fig.19-6: RCL standard variables

RCL standard data types

Information on the RCL standard data types can be found under [Overview, page 53.](#)

Standard data types	Remark
Simple data types	
BINARY	0, 1
CHAR	The ASCII (American Standard Code for Information Interchange) character set (7 bits) is used to represent letters and symbols used on a standard US keyboard. The ASCII character set corresponds to the first 128 characters (0-127) of the ANSI character set.
INTEGER	Variables of data type INTEGER (long integer) are saved as 32 bit numbers (4 bytes) with signs in the range from -2,147,483,648 to 2,147,483,647.
REAL	Variables of data type REAL are saved as 32 bit numbers (4 bytes) -3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values.

RCL Keywords

Standard data types	Remark
Structured data types	
ARRAY	Working with Array Variables (Arrays), page 62,
BELT	Reserved
FILE	Composed RCL Data Types, page 55,
JC_POINT	Point Variables, page 58,
POINT	Point Variables, page 58,
RECORD RECORD_END	Record Types, RECORDs, page 56,
TEXT	Working with Text Variables, page 61,
WC_FRAME	Reserved

Fig. 19-7: Standard data types

20 Service and Support

We provide a worldwide service network for an optimum and fast support of your needs. Our experts are there for you. You can contact us **24/7**.

Service Germany

Our technology-oriented Competence Center in Lohr, Germany, is responsible for all your service-related concerns for electric drive and controls.

Contact the **Service Helpdesk & Hotline** under:

Phone: **+49 9352 40 5060**
Fax: **+49 9352 18 4941**
E-mail: service.svc@boschrexroth.de
Internet: <http://www.boschrexroth.com>

Additional information on service, repair (e.g. delivery addresses) and training can be found on our internet sites.

Service worldwide

Outside Germany, please contact your service office first. For hotline numbers, refer to the sales office addresses on the internet.

Preparing information

We can help you more quickly and efficiently if you have the following information ready:

- Detailed description of malfunction and circumstances leading to the malfunction
- Type plate name of the affected products, in particular type codes and serial numbers
- Your contact data (phone and fax number as well as your email address)

Index

Symbols

@.....	55
@POS.....	61

A

A.....	123
A-0-0028, Travel range limit switch.....	151
A-0-0029, Position polarities.....	151
A-0-0030, Position limit value, positive.....	152
A-0-0031, Position limit value, negative.....	152
A-0-0032, Velocity limit value, positive.....	153
A-0-0033, Velocity limit value, negative.....	153
A-0-0034, Acceleration limit value, bipolar.....	153
A-0-0036, Jerk limit value, bipolar.....	153
A-0-0038, Torque/force limit value, bipolar.....	154
A-0-0048, Velocity polarities.....	154
A-0-0049, Torque-/force polarities.....	155
A-0-0080, Jogging, short distance.....	155
A-0-0081, Jogging, long distance.....	156
A-0-0082, Jogging, low velocity.....	157
A-0-0083, Jogging, high velocity.....	158
A-0-0084, Jogging, acceleration.....	159
A-0-0085, Jogging, deceleration.....	159
A-0-0086, Jogging, acceleration jerk.....	159
A-0-0087, Jogging, deceleration jerk.....	160
A-0-0224, Rapid stop deceleration.....	160
A-0-0225, Rapid stop jerk.....	160
A-0-0230, Blending distance PTP.....	161
A-0-0240, Feedrate override.....	161
A-0-0241, Feedrate override control word.....	161
A-0-0242, Position control, synchronization acceleration.....	161
A-0-0243, Position control, synchronization velocity.....	162
Abbreviations.....	14
About this documentation.....	9
Information representation	13
Validity of the documentation	9
ABS.....	89
Acceleration.....	122
Acceleration, effective.....	124
Acceleration factor.....	124
Acceleration specification.....	122
Global	122
Statement-related	123
Actual position reading.....	61
Additional information.....	119
Acceleration	122
Acceleration factor	124
Deceleration factor	124
Effective acceleration	124
Effective deceleration	125
Effective velocity	122
Global acceleration specification (global specification)	122

A

...Additional information	
Global jerk specification (global specifica- tion)	126
Global velocity specification	120
Jerk	125
Motion optimization for orientation axes	120
Override	121
Statement-related acceleration specifica- tion (WITH)	123
Statement-related jerk specification (WITH)	126
Statement-related velocity specification (WITH)	120
Velocity	119
Velocity factor	121
Additional Information	
Disable override	121
AFACTOR.....	124
AFIX.....	121
ALIAS name.....	55
AND.....	84
Arithmetic expressions.....	81
ARRAY.....	57, 62
Array variables.....	62
ASC_INT.....	91
ASSIGN.....	90
ATAN.....	87

B

Basic coordinate system (BCS).....	16
Basic coordinate system relative to the world coordinate system.....	21
BEGIN.....	40
BELT.....	127
Belt synchronization.....	127
BINARY.....	54
Blending mode.....	144
RCL programming	105
Blending when changing the interpolation method.....	140, 142
BLOCK_SLOPE.....	130
Block transitions (slope types).....	129

C

C-0-0123, Maximum number of kinematics.....	163
C-0-2480, Configured kinematic number.....	163
C-0-2482, List of non-linked axes.....	164
C-0-2483, List of configured kinematics.....	164
C-0-2484, Kinematic configuration list.....	164
Call by VALUE.....	41
CASE.....	80
Channel declaration.....	66
CHAR.....	54

Index

C

CHR.....	89
CIRCULAR.....	114
CLOSE.....	55
Comparative operations.....	83
Compiler - Mode of operation.....	35
Compiler statements.....	36
CONST.....	51
CONT_SLOPE.....	130
Coordinates.....	15
Coordinate systems.....	15
COS.....	87

D

DEBUGINFO.....	38
Deceleration, effective.....	125
Deceleration factor.....	124
DEF.....	57
DEFAULT.....	80
Define coordinate systems.....	21
DFACTOR.....	125
Direct motion statements.....	109
Additional information	119
Interpolation method	114
Motion commands	110
Disable override.....	121

E

ELSE.....	77
END_OF_FILE.....	90
EOF.....	90
EQUAL.....	80
err file.....	36
Extension for include files.....	38
EXTERNAL.....	41
EXTERNAL, with parameter transfer.....	43

F

FILE.....	55
-----------	----

G

General information.....	144
--------------------------	-----

H

HALT.....	43, 73
-----------	--------

I

IF.....	77, 83
INCLUDE.....	37
Information representation	
Names and abbreviations	14
Safety instructions	13
Symbols used	14
INPUT.....	57

I

INT.....	116
INT_ASC.....	90
INTEGER.....	54
Intermediate point.....	114
Interpolation method.....	114
CIRCULAR	114
Global	116
LINEAR	114
PTP	115
Statement-specific	115

J

J.....	125
JC.....	60, 93
JC_NAMES.....	37
JC_POINT.....	55, 59
Jerk.....	125
jerk specification.....	126
Jerk specification	
Global	126
Statement-related	126
Jog	
Axis coordinates	148
Joint coordinates	148
Jogging	
World coordinates	149, 150
Jog in different coordinate systems.....	148
Jog mode.....	147
Joint coordinate system (JCS).....	16
JUMP.....	75
Jump target.....	75

K

K-0-0001, Logic kinematic number.....	165
K-0-0002, Kinematic name.....	165
K-0-0003, Maximum path speed.....	165
K-0-0004, reserved.....	166
K-0-0005, Software limit switches, positive,	
WC.....	166
K-0-0006, Software limit switches, negative,	
WC.....	166
K-0-0007, Axis number of kinematics.....	166
K-0-0008, Axis configuration list.....	166
K-0-0008 Define axis assignment.....	217
K-0-0009, Osci: list of all K-parameters that	
can be oscilloscoped.....	168
K-0-0010, List of all K-parameters.....	168
K-0-0011, List of all invalid K-parameters.....	169
K-0-0012, Checksum of all K-parameters.....	169
K-0-0013, IDN list of checksum K-parameters..	169
K-0-0014, Path acceleration, change time.....	169
K-0-0015, Path deceleration, change time.....	172
K-0-0016, Project identification number.....	174
K-0-0020, Kinematic diagnostic message.....	175
K-0-0021, Kinematic status.....	175

K

K-0-0022, Extended kinematic status.....	177
K-0-0023, Kinematics diagnostic number.....	179
K-0-0031, Transformation scheme.....	179
K-0-0031 Define kinematic type.....	216
K-0-0032, Referencing axes, command configuration.....	180
K-0-0033, IPOS, update when coordinate change.....	180
K-0-0036, WC axis monitoring.....	180
K-0-0040, Transformation, lengths of the mechanics.....	181
K-0-0041, Transformation, factors.....	181
K-0-0042, WC system offset.....	181
K-0-0043, Assignment list of workpiece coordinates.....	182
K-0-0050, Distance units WC.....	183
K-0-0051, Velocity units WC.....	184
K-0-0052, Acceleration units WC.....	184
K-0-0053, Jerk units WC.....	185
K-0-0054, Unit of measurement for kinematic data.....	185
K-0-0070, Belt number of kinematics.....	186
K-0-0071, Coupling factors, belt 1.....	186
K-0-0072, Coupling factors, belt 2.....	187
K-0-0073, Coupling factors, belt 3.....	187
K-0-0074, Coupling factors, belt 4.....	188
K-0-0080, Jogging, distance short in WC system.....	188
K-0-0081, Jogging, distance long in WC system.....	189
K-0-0082, Jogging, velocity low in WC system.....	190
K-0-0083, Jogging, velocity high in WC system.....	191
K-0-0084, Jogging, acceleration in WC system.....	192
K-0-0085, Jogging, deceleration in WC system.....	192
K-0-0086, Jogging, acceleration jerk in WC system.....	193
K-0-0087, Jogging, deceleration jerk in WC system.....	193
K-0-0100, Current position, Cartesian original coordinates.....	194
K-0-0102, Current path velocity.....	194
K-0-0103, Current path acceleration.....	194
K-0-0104, Current path jerk.....	194
K-0-0105, Actual position, Cartesian workpiece coordinates.....	194
K-0-0106, Coordinate values of tool.....	195
K-0-0107, Coordinate values of workpiece.....	195
K-0-0121, Coordinate name C1.....	195
K-0-0121 to K-0-0136, coordinate names.....	195
K-0-0140, Tool name.....	195
K-0-0141, Workpiece name.....	195
K-0-0150, Interpolated command position.....	195
K-0-0152, Interpolated path velocity.....	195
K-0-0153, Interpolated path acceleration.....	195
K-0-0154, Interpolated path jerk.....	195
K-0-0155, Safe zones on/off.....	195

K

K-0-0156, Safe zone 1 min. point.....	196
K-0-0156/K-0-0157 Safezone1 min./max. Point till K-0-0218/K-0-0219 Safezone32 min./max. Point.....	197
K-0-0157, Safe zone 1 max. point.....	196
K-0-0220, File transfer name.....	198
K-0-0221, File name to be compiled.....	199
K-0-0240, Feedrate override.....	199
K-0-0241, Feedrate override control word.....	199
K-0-0242, Acceleration override.....	199
K-0-0243, Deceleration override.....	200
K-0-1060, Command: Load file from CF.....	200
K-0-1061, Command: Save file to CF.....	200
K-0-1062, Command: Compile file to MLC.....	200
K-0-2200, Programmed position 1 (active).....	201
K-0-2201, Programmed position 2 (active).....	201
K-0-2202, Type of actively programmed position.....	201
K-0-2203, Interpolation type.....	201
K-0-2204, Currently active QLL file name.....	201
K-0-2205, Currently active line number.....	201
K-0-2300, PLC-Bit input (RCL).....	201
K-0-2301, PLC-Integer input (RCL).....	201
K-0-2302, PLC-Real input (RCL).....	201
KINEMATICS.....	36
Kinematic user interface.....	203
Compiler	210
Debugger	211
General information	203
RCL Editor	205
RobotControl menu structure	203

L

Late blending.....	142
Late commanding.....	133
Late commanding	133
LIMIT_MAX.....	93
LIMIT_MIN.....	93
LIMIT_OFF.....	93
LINEAR.....	114
Linear blending for BLOCK_SLOPE.....	142
Linear blending for PROGR_SLOPE and CONT_SLOPE.....	137
Logic expressions.....	83
Look-ahead across several blocks at PROGR_SLOPE and CONT_SLOPE.....	133

M

Main program	
Declaration part	39
Statement part	40
Structure	38
MAX_TIME.....	71
MOD.....	82
Motion commands.....	110

Index

M

Motion-influencing RCL statements and parameters.....	118
Motion-influencing statements.....	127
Belt synchronization	127
Block transitions (BLOCK_SLOPE, PROGR_SLOPE, CONT_SLOPE)	129
Look-ahead across several blocks at PROGR_SLOPE and CONT_SLOPE	133
Motion optimization for orientation axes.....	120
MOVE.....	110
MOVE_REL	112
MOVE_REL APPROX.....	112
MOVE_REL APPROX ... EXACT	113
MOVE_REL EXACT.....	113
MOVE TO	111
MOVE VIA	111
MOVE VIA ... TO	111

N

NOT.....	84
----------	----

O

OPEN.....	55
Operations.....	81
Arithmetic expressions	81
Comparative operations	83
Logic expressions	83
Sequence of operations	84
Operators	
Arithmetic	81
Logic	83
OR.....	84
ORD.....	89
OUTPUT.....	57
Override.....	121
Overview	
Motion-influencing RCL statements and parameters	118
Travel range limits	33

P

PAUSE.....	73
pkt file.....	35
PLC communication channels.....	64
POINT.....	55, 59
Points and point file pkt.....	59
Point variables.....	58
Complete value assignment	59
Declaration	58
Value assignment for individual components	60
POS.....	61
Position accurate switching	
Belt type	101
Position-accurate switching.....	97

P

...Position-accurate switching	
Belt synchronous operating range of belt type 4	102
Belt type	100
digital outputs on the path	98
Set belt counter	102
Switching of process data on the path	99
SysClk	101
PROGR_SLOPE.....	130
PROGRAM.....	40
PROGRAM_END.....	40
PTP.....	115
PTP blending for BLOCK_SLOPE.....	140
PTP blending for PROGR_SLOPE and CONT_SLOPE.....	135

Q

qll file.....	35
---------------	----

R

R.....	135
R_PTP.....	135
RCL	
Program structure	35
RCL constants.....	51
Constant declarations	51
Predefined	51
RCL coordinate systems	
Basic coordinate system (BCS)	16
Joint coordinate system (JCS)	16
Workpiece coordinate system (WpCS)	19
World coordinate system (WCS)	18
RCL data types.....	53, 92
Composed	55
Simple	54
User-defined	55
RCL Debugger.....	211
Debug programs	212
Monitor variables	213
Sett and delete breakpoints	211
RCL Editor.....	205
Bookmark	207
Characteristics	205
Comments	207
Create new program	206
Cut, copy, paste	208
Editor functions	206
Find and replace	207
Options	206
Support source code input	208
RCL functions.....	87
ABS function	89
ASC_INT function	91
ASC function	90
ASSIGN function	90

R

...RCL functions	
ATAN function	87
CHR function	89
COS function	87
EOF function	90
JC function	93
LIMIT_MAX function	93
LIMIT_MIN function	93
LIMIT_OFF function	93
ORD function	89
ROUND function	89
SIN function	87
SIZEOF function	92
SQRT root function	88
TRUNC function	89
WC function	93
RCL keywords.....	277
RCL library functions.....	26
RCL motion statements.....	109
Direct	109
Motion-influencing	127
RCL parameters.....	151
Axis-specific	151
C-parameters of all kinematics	163
Parameters of a kinematics	165
RCL program call	
External programs	41
Main program	41
Subroutine	44
Subroutine sequence	46
RCL program control.....	69
CASE statement	80
HALT statement	73
IF-THEN--ELSE) statement	77
JUMP statement	75
PAUSE statement	73
REPEAT statement	74
WAIT statement	69
RCL programming	
Blending mode	105
Position-accurate switching	97
RCL program structure.....	35
Compiler	35
Compiler statements	36
Main program call	41
Main program structure	38
Program declaration	40
Subroutine declaration	44
Subroutine sequence	46
RCL special functions.....	95
Call	95
Position-accurate switching	97
RCL variable declaration.....	57
RCL variables.....	53
Array variables	62
PLC communication channels	64

R

...RCL variables	
Points and point file "pkt"	59
RCL data types	53
RCL point variables	58
RCL variable declaration	57
Text variables	61
READ.....	55
REAL.....	54
RECORD.....	56
RECORD_END.....	56
Record types.....	56
REPEAT.....	62, 74
REPEAT_END.....	48, 74
REPEAT n TIMES.....	48, 74
RETURN.....	44
RobotControl	
Compiler	210
Debugger	211
Editor	205
Menu item: Kinematics	204
Menu item: Prog.QLL	204
Menu item: Programs	204
Menu item: Robot	203
Menu structure	203
RoCo multi-kinematics.....	215
Coordinated motion	219
Create kinematics	215
Define axis assignment	217
Define kinematic type	216
Delete kinematics	218
Overview	215
PLC shared memory interfaces (KinData) .	223
Point tables	221
Point tables, assignment and response	222
RCL programs	220
RoCo programming environment.....	203
ROUND.....	89

S

Sequence of operations.....	84
Setup mode.....	147
SIN.....	87
SIZEOF.....	92
Source file.....	35
Spatial blending.....	134
Overview	134
Spatial blending for BLOCK_SLOPE.....	140
Spatial blending for PROGR_SLOPE and	
CONT_SLOPE.....	135
SPC_FCT	
List as table	96
SPC_FCT: 1	
Position-accurate switching of digital out-	
puts on the path	98

Index

S

SPC_FCT: 2	
Position-accurate switching of process data on the path	99
SPC_FCT: 21	
Belt type	100
SPC_FCT: 23	
DateTime	101
SPC_FCT: 24	
SysClk	101
SPC_FCT: 28	
Set belt counter	102
SPC_FCT: 53	
Belt synchronized workspace with belt type 4	102
SPC_FCT: 59	
Query blending mode	106
SPC_FCT: 60	
Set blending mode	105
SQRT.....	88
SUB_END.....	44
SUBROUTINE.....	44
Support	
See service hotline	287
sym file.....	36
SYNC.....	127
SYNCHRON.....	127
SYNCHRON_END.....	127

T

Target point names.....	114
TEXT.....	55
Text variables.....	61
THEN.....	77
Transformation rule	
Overview	227
Robot type 00, Cartesian robot	227
Robot type 09, 3-axis Scara with 1 optional manual axis	234
Robot type 10, 4-axis Scara with 1 optional manual axis	236
Robot type 100, PLC	274
Robot type 12, 2-axis parallelogram	239
Robot type 13, 2-axis parallelogram	241
Robot type 18, Contact kinematics	243
Robot type 19, 2-axis parallelogram V1	244
Robot type 2, 3-axis delta with 1 optional manual axis	231
Robot type 20, 3-axis Delta with 1 optional manual axis	246
Robot type 21, 2-axis parallelogram V2	251
Robot type 29, 2-axis delta with rotary and translatory axis	267

T

...Transformation rule	
Robot type 30, Adabot	270
Robot type 32, 2-axis delta with asymmetric TCP	273
Robot with offset wrist with six (seven) axes	254
Robot with offset wrist with six axes + three linear axes	260
Travel range limits.....	33
TRUNC.....	89
TYPE.....	55

U

UNTIL.....	83
Use coordinate systems.....	21

V

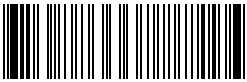
V.....	119
V_PTP.....	120
VALUE.....	46
Value assignments.....	81
VAR.....	54
Velocity.....	119
Velocity, effective.....	122
Velocity factor	
VFACTOR	121
Velocity specification	
Global	120
Statement-related	120
VFACTOR.....	121
VFIX.....	121
VFIX_PTP.....	121

W

WAIT.....	69
WC.....	60, 93
WC_FRAME.....	51
WC_NAMES.....	36
WC_SYSTEM.....	51
WC_UR.....	51
WITH.....	120
Workpiece coordinate editor.....	25
Workpiece coordinate file.....	24
Workpiece coordinate system (WpCS).....	19
Workpiece coordinate system relative to the world coordinate system.....	23
World coordinate system	
Basic coordinate system	21
Workpiece coordinate system	23
World coordinate system (WCS).....	18
WRITE.....	55

Notes

Bosch Rexroth AG
Electric Drives and Controls
P.O. Box 13 57
97803 Lohr, Germany
Bgm.-Dr.-Nebel-Str. 2
97816 Lohr, Germany
Tel. +49 9352 18 0
Fax +49 9352 18 8400
www.boschrexroth.com/electrics



R911333852

DOK-MLC***-RCL*PRO*V12-AP02-EN-P