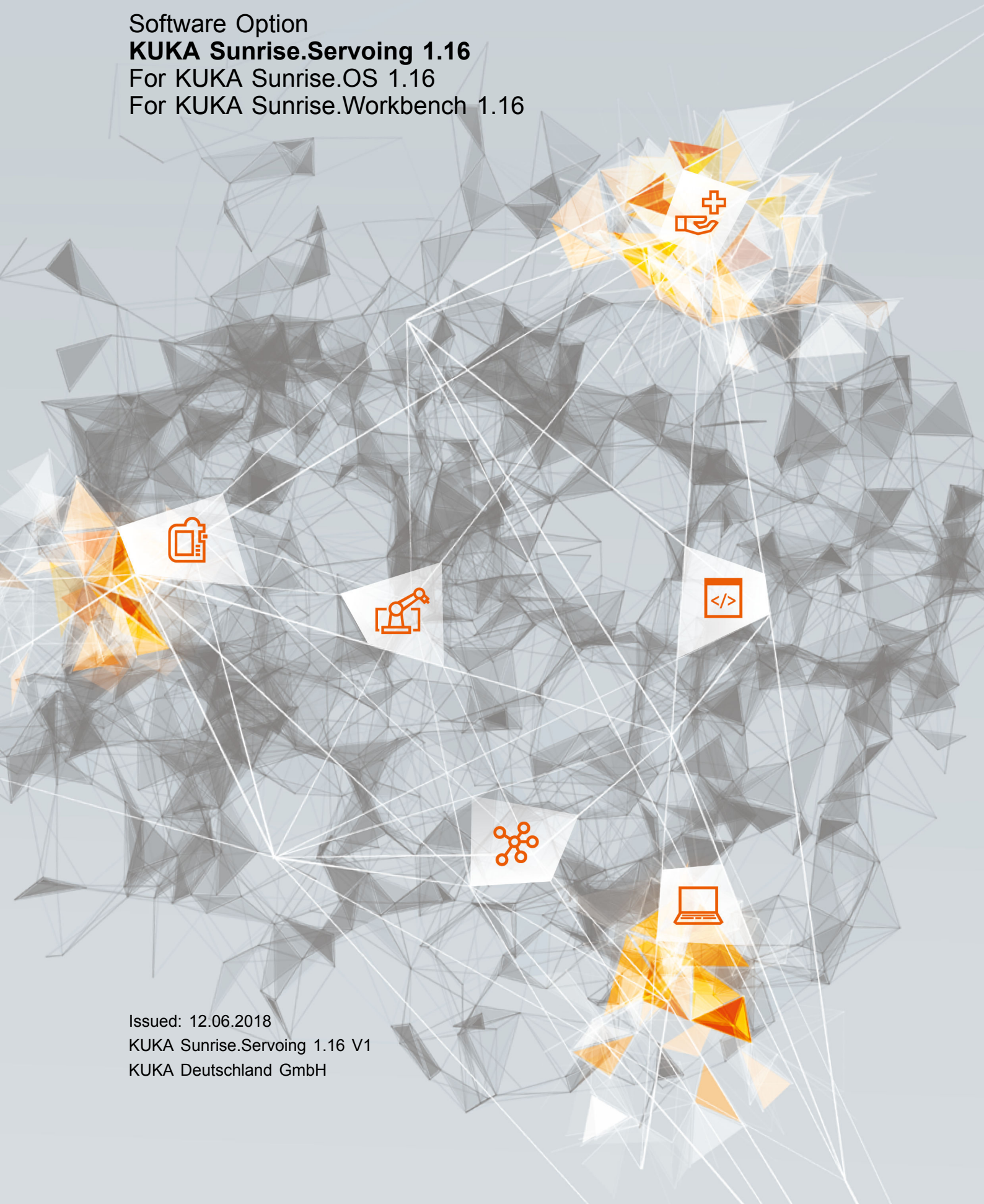
Software Option

KUKA Sunrise.Servoing 1.16

For KUKA Sunrise.OS 1.16

For KUKA Sunrise.Workbench 1.16



Issued: 12.06.2018

KUKA Sunrise.Servoing 1.16 V1

KUKA Deutschland GmbH

© Copyright 2018
KUKA Deutschland GmbH
Zugspitzstraße 140
D-86165 Augsburg
Germany

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

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

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

Subject to technical alterations without an effect on the function.

KIM-PS5-DOC

Translation of the original documentation

Publication:	Pub KUKA Sunrise.Servoing 1.16 (PDF) en
	PB10955
Book structure:	KUKA Sunrise.Servoing 1.16 V1.2
	BS10119
Version:	KUKA Sunrise.Servoing 1.16 V1

Contents

1 Introduction

1.1 Target group

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

- Advanced system knowledge of robotics
- Advanced Java programming skills
- Advanced knowledge of programming with KUKA RoboticsAPI
- Advanced knowledge of control technology and stability

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

1.2 Industrial robot documentation

The industrial robot documentation consists of the following parts:

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

Each of these sets of instructions is a separate document.

1.3 Representation of warnings and notes

Safety

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

DANGER

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

WARNING

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

CAUTION

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

NOTICE

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

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

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

SAFETY INSTRUCTION
The following procedure must be followed exactly!

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

Notices

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

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

2 Product description

2.1 Overview of Sunrise.Servoing

Sunrise.Servoing is an option package which provides the following new motion classes for programming:

- SmartServo
- DirectServo
- SmartServoLIN (part of the SmartServo option)

These motion classes enable non-deterministic, soft real-time applications to be implemented which are used to correct a robot path quickly. For example, the original robot path can be corrected using the cyclical evaluation and processing of external sensor signals.

The motion classes each contain a motion type of the same name and are referred to as servo motions.

The servo motions can be used to set new end points during the runtime of a motion:

- SmartServo and DirectServo set axis-specific and Cartesian end points. As in the case of a PTP motion, the end points are addressed along the fastest path.
- SmartServoLIN sets Cartesian end points. The end points are addressed with a linear motion.

Servo motions are executed as asynchronous motions. If a new end point is set while an end point is being addressed, the previous end point is discarded and the new end point is directly addressed.

- SmartServo and SmartServoLIN enable jerk-limited paths with fast destination setting.
- DirectServo enables velocity-limited paths with constant updating of the destination. This allows the realization of user-specific planning and interpolation algorithms.

When the DirectServo motion type is used, the destinations may be at a maximum distance of 5° from the current actual position.

2.2 Axis-specific motions (comparison of PTP, SmartServo and DirectServo)

NOTICE

The characteristics of the DirectServo motion type (digital filter, no jerk limitation) allow the implementation of high-frequency destination specifications. This can result in high-frequency oscillations on the robot (whistling, buzzing) which can cause damage to the robot. KUKA cannot be held liable for any property damage resulting from this.

If there is a question of whether to use the SmartServo or DirectServo motion type, SmartServo is recommended.

	PTP	SmartServo	DirectServo
Areas of application	Axis-specific continuous path motion, col-	“Visual servoing”, camera-based collision avoidance	Haptics

	PTP	SmartServo	DirectServo
	lision-free path planning		
Destination setting interval (recommended)	Long (>100 ms)	Medium (>20 ms)	Very short (2 … 19 ms)
Quantity of control points	Low	High	Very high
Destination	Any point in the workspace	Any point in the workspace	At a maximum distance of ±5° from the axis-specific actual position of the robot
Path characteristics	• Jerk-limited • Can be approximated (inter-motion)	• Jerk-limited • Can be approximated (intra-motion with new destination setting)	• Not jerk-limited • Can be approximated (intra-motion with new destination setting)
Path	Always remains the same, irrespective of the override setting, current velocity or current acceleration	Depends on the override setting, current velocity and current acceleration	
Response to destination specification while an end point is being addressed	Every point is addressed.	Current end point is discarded when a new destination is specified.	
Motion transition without exact positioning (on termination of a standard motion)	—	Fluid transition to servo motion programmable	—
Configurable path parameters	• Velocity • Acceleration • Jerk	• Velocity • Acceleration	• Velocity
Planning	• Polynomial	• 3rd-order polynomial	• Digital filter • Planning in application
System reaction (step response)	—	Dynamic (parameterizable)	Highly dynamic

2.3 Cartesian motions (comparison of LIN and SmartServoLIN)

	LIN	SmartServoLIN
Areas of application	Cartesian path motion, collision-free path planning	“Visual servoing”, camera-based collision avoidance
Destination setting interval (recommended)	Long (>100 ms)	Medium (>20 ms)
Quantity of control points	Low	High

	LIN	SmartServoLIN
Destination	Any pose in the workspace, no singularity treatment	Any pose in the workspace (with orientation restrictions), no singularity treatment
Path characteristics	• Linear path in Cartesian space • Jerk-limited • Can be approximated (inter-motion)	• Virtually linear path in Cartesian space • Jerk-limited • Can be approximated (intra-motion with new destination setting)
Path	Always remains the same, irrespective of the override setting, current velocity or current acceleration	Depends on the override setting, current velocity and current acceleration
Response to destination specification while an end point is being addressed	Every point is addressed.	Current end point is discarded when a new destination is specified.
Motion transition without exact positioning (on termination of a standard motion)	—	Fluid transition to servo motion programmable
Configurable path parameters	• Axis velocity • Axis acceleration • Axis jerk • Cartesian velocity (translational, rotational) • Cartesian acceleration (translational, rotational) • Cartesian jerk (translational, rotational)	• Cartesian velocity (translational, rotational) • Cartesian acceleration (translational, rotational)
Planning	• Polynomial	• 3rd-order polynomial

2.4 Class structure of servo motions

The SmartServo, DirectServo and SmartServoLIN motion classes contain methods for the starting and parameterization of the respective servo motion. Each motion class also provides a specific runtime class which can be used to continuously update the end positions and runtime parameters of the servo motion.

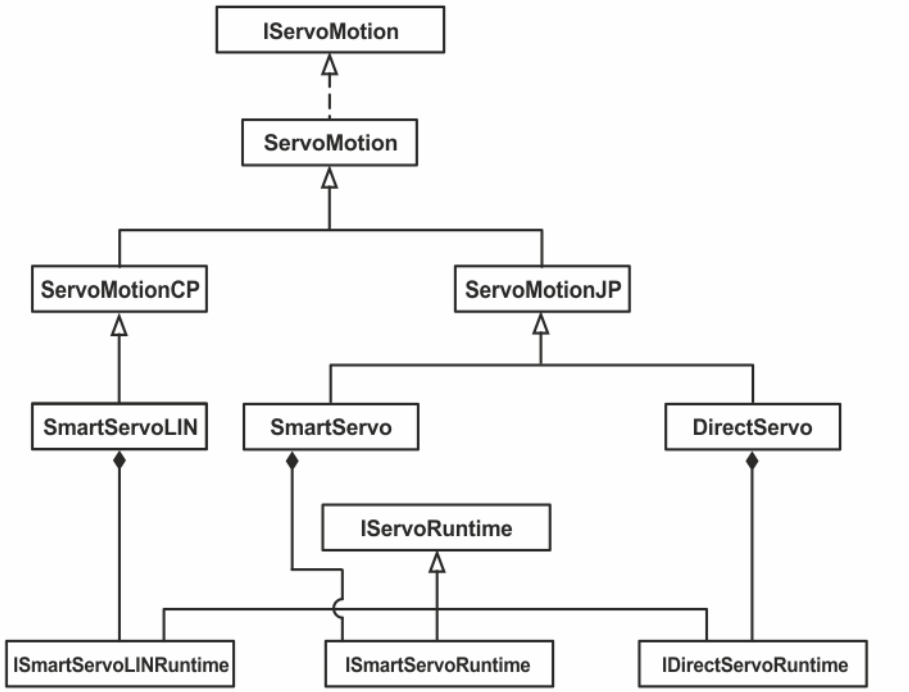


Fig. 2-1: Class structure of servo motions

The IServoRuntime interface provides the methods used jointly by all motion classes. It also saves their characteristics and values, such as destination setting, position detection, time stamp management, etc.

Each of the motion classes also has its own interface with the class-specific methods and characteristics:

- The ISmartServoRuntime interface is part of the SmartServo motion class.
- The IDirectServoRuntime interface is part of the DirectServo motion class.
- The ISmartServoLINRuntime interface is part of the SmartServoLIN motion class.

2.5 Intended use

Use

KUKA Sunrise.Servoing is designed for applications with rapid path corrections. A main area of application is “visual servoing” in which the evaluation of external sensors alters the set path of the robot in short cycles.

During start-up of the system, the provisions of monitored operation shall apply.

The user is responsible for ensuring that the provisions of monitored operation are adhered to. This particularly includes:

- Only trained personnel and personnel instructed in system use may be used.
- It must be possible at all times to bring the robot to a standstill within the reach of the operator by means of safety-oriented equipment.

The user must perform a risk analysis for use outside monitored operation.

Operation in accordance with the intended use also involves continuous observance of the following documentation:

- Assembly and operating instructions of the industrial robot
- Assembly and operating instructions of the robot controller
- Operating and programming instructions for the System Software

Misuse

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

3 Safety

This documentation contains safety instructions which refer specifically to the product described here. The fundamental safety information for the industrial robot can be found in the “Safety” chapter of the following documentation:

- Operating or assembly instructions of the robot
- Operating and programming instructions for the System Software

The “Safety” chapter in the robot operating or assembly instructions or in the operating and programming instructions for the system software must be observed. Death to persons, severe injuries or considerable damage to property may otherwise result.

WARNING

KUKA Sunrise.Servoing enables closed control loops to be implemented. The user is responsible for ensuring their stability. Unstable control loops can cause the overall system to vibrate and the robot to move otherwise than expected, although the robot addresses its destination correctly. KUKA is not liable for personal injury and damage to property resulting from operation of the robot with unstable control loops.

The user is responsible for performing his own risk and hazard analysis for use of the robot. This applies in particular if personnel have to enter the robot’s workspace. We recommend having a separate supervisor with a suitable EMERGENCY STOP device. The risk analysis must give special attention to injuries caused by crushing, collision and sharp objects.

The stopping distances and times specified in the operating and assembly instructions of the LBR iiwa are not valid in applications that use servo motions. Users must determine the stopping distances and times for these applications themselves.

Servo motions enable destinations to be set online from the application. Changing the path parameters, the override setting or switching to T1 mode changes the robot path; the actual path cannot be predicted. Applications with online destination setting must always be carefully tested first in Manual Reduced Velocity mode (T1).

When servo motions are executed in T1 mode, only the velocity of the robot flange is monitored at 250 mm/s.

4 Installation

4.1 System requirements

Hardware

- KUKA LBR iiwa
- KUKA Sunrise Cabinet

Software

- KUKA Sunrise.Workbench 1.16
- KUKA Sunrise.OS 1.16

4.2 Installing Sunrise.Servoing in Sunrise.Workbench

Description

If the option package is supplied together with KUKA Sunrise.Workbench, it is automatically installed during installation of Sunrise.Workbench. If Sunrise.Workbench is already installed, the option package must be installed subsequently.

Option packages are provided as ZIP archives for installation. The archive names have the following composition:

- *Article number*;*revision number (2-digit)*;*product name*;*build version*

The following files can be selected for installation (delivered in the **options** folder):

- ZIP archive of the SmartServo option (with SmartServoLIN)
- ZIP archive of the DirectServo option
- ZIP archive with the corresponding documentation

Precondition

- Local administrator rights
- Sunrise.Workbench is installed.
- The option package is available as a ZIP archive.

Procedure

1. Select the menu sequence **Help** > **Install new software ...**. The **Install** window is opened.
2. To the right of the **Work with** box, click on **Add**. The **Add repository** window is opened.

   Alternatively: Drag the ZIP archive of the option into the window, then continue with step 5.
3. Click on **Archive …**, navigate to the directory in which the ZIP archive of the option is located and select the archive.
4. Confirm your selection with **Open**. The **Position** box now displays the installation path. Confirm the path with **OK**.
5. In the **Install** window, the installation path is adopted in the **Work with** box.

   The window now also displays the **KUKA Connectivity** check box. Activate the check box.
6. Leave the other settings in the **Install** window as they are and click on **Next >**.

7. An installation details overview is displayed. Click on **Next >**.
8. A license agreement is displayed. In order to be able to install the software, the agreement must be accepted. Then click on **Finish**. The installation is started.
9. A safety warning concerning unsigned contents is displayed. Confirm with **OK**.
10. A message indicates that Sunrise.Workbench must be restarted in order to apply the changes. Click on **Restart now**.
11. Sunrise.Workbench restarts. This completes installation in Sunrise.Workbench.

4.3 Installing class libraries

Description

The SmartServo, DirectServo and SmartServoLIN motion classes can be installed individually via the Sunrise.Workbench software catalog. The installation adds the corresponding class libraries to the RoboticsAPI.

The following entries for installation can be selected in the software catalog:

- **Smart Servo Motion Extension**
  Expansion for SmartServo
- **Direct Servo Motion Extension**
  Expansion for DirectServo
- **Smart Servo Linear Motion Extension**
  Expansion for SmartServoLIN

Precondition

- The SmartServo and DirectServo option packages are installed in Sunrise.Workbench.
- Sunrise project has been created.

Procedure

1. Open the station configuration of the Sunrise project and select the **Software** tab.
2. Select the desired software expansions for the installation:
   - Set the check mark for the corresponding entry in the **Install** column.
3. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.
   The class libraries are inserted in the Sunrise project and are available for robot programming.

The software expansions installed in the Sunrise project must be transferred to the robot controller. The System Software must be reinstalled on the robot controller for this.

4.4 Installing the application examples

Description

A package with example applications is available for each of the SmartServo, DirectServo and SmartServoLIN motion classes. These packages can be installed individually via the Sunrise.Workbench software catalog.

The following entries for installation can be selected in the software catalog:

- **SmartServo Motion Example Applications**
  Example applications for SmartServo
- **DirectServo Motion Example Applications**
  Example applications for DirectServo
- **SmartServo Linear Motion Example Applications**
  Example applications for SmartServoLIN

Precondition

- The SmartServo and DirectServo option packages are installed in Sunrise.Workbench.
- Sunrise project has been created.

Procedure

1. Open the station configuration of the Sunrise project and select the **Software** tab.
2. Select the desired packages for the installation:
   - Set the check mark for the corresponding entry in the **Install** column.
3. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.
   The application examples are inserted into the Sunrise project (**examples** folder).

The software expansions installed in the Sunrise project must be transferred to the robot controller. The System Software must be reinstalled on the robot controller for this.

5 Programming

5.1 Servo motion in an application context

The figure shows the basic program sequence when executing a servo motion using the example of the SmartServo motion type.

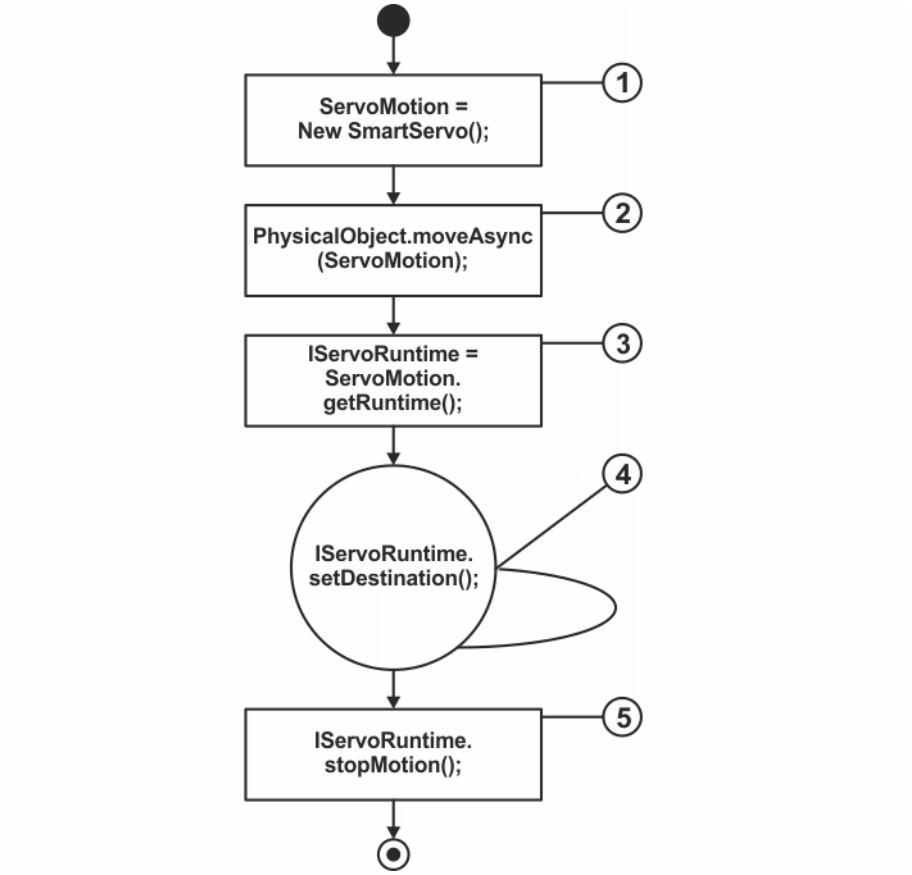


Fig. 5-1: Servo motion in an application context

Item	Description
1	Create an instance of the servo motion, here an instance of the SmartServo motion class.
2	Activate the servo motion with moveAsync(…). The real-time system on the robot controller executes the servo motion. The robot application continues.
3	Call the runtime environment with getRuntime(). The runtime environment is provided as soon as the robot is available. The application data are updated with the real-time system data that are provided by the runtime instance of IServoRuntime. These updated data, e.g. the current robot position, velocity or acceleration, are available for further calling.
4	Set new end points with setDestination(…) (cyclically during the runtime of the robot motion). Once an end position has been reached, the robot maintains this position for a defined time (timeoutAfterGoalReach). When this time has elapsed, the servo motion is automatically ended, provided that no new end point has been set or that the servo motion is ended with stopMotion().

Item	Description
	(>>> *5.6 "Destination reached and timeouts" Page 29*)
5	End the servo motion with stopMotion(). The robot application continues.

5.2 Simple destination setting with setDestination(…)

The figure schematically illustrates the fine interpolation of the robot over time:

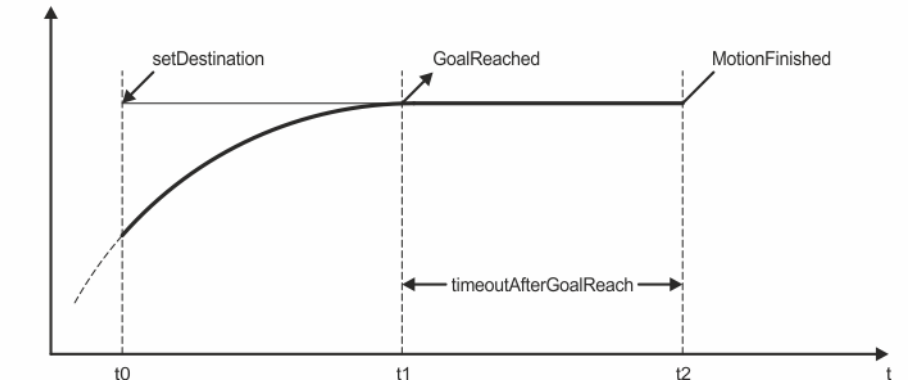


Fig. 5-2: Time diagram in an application context, basic behavior

The current velocity and position are applied in the path planning at the initial time **t0**. This results in a jerk-limited path being planned to the end position. The following motion parameters are taken into consideration during planning:

- Maximum permissible velocity (for all servo motions)
- Maximum permissible acceleration (for SmartServo and SmartServoLIN)
- Maximum permissible jerk (for SmartServo and SmartServoLIN)

At time **t1**, which is a result of the constraints, the precision interpolator reaches the specified destination state. The event IServoOnGoalReachedEvent(…) is triggered during the SmartServo and SmartServoLIN motion types.

The precision interpolator then remains in the destination state for the time period specified in the variable “timeoutAfterGoalReach”. Once this time period has elapsed, the servo motion is deactivated.

The maximum Cartesian acceleration and velocity can be set during a SmartServoLIN motion. If the selected values for the maximum Cartesian acceleration and velocity are too high, this can result in exceeding the maximum permissible axis-specific values for acceleration, velocity and jerk.
In such a case, the robot stops moving and an error message is generated. Smaller values must then be selected for the maximum Cartesian acceleration and velocity.

5.3 Cyclical destination setting with setDestination(…)

The IServoRuntime interface provides the functionality which allows the destination for the runtime of the motion command moveAsync(…) to be changed:

```
IServoRuntime theRuntime = ServoMotion.getRuntime();
...
theRuntime.setDestination(destination);
```

The updateWithRealtimeSystem() method is executed by calling the setDestination(…) method. This updates the application data with the real-time system data. A manual update is therefore not required.
By calling the stopMotion() method, the motion can be stopped at any time.

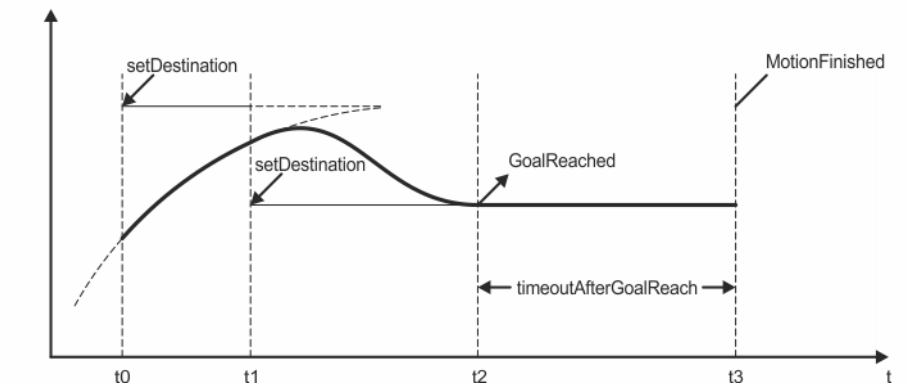


Fig. 5-3: Time diagram in an application context, replacing a destination setting prematurely

A destination is set at time **t0**. By calling the setDestination(…) method again at time **t1** while the destination set in **t0** is being addressed, the precision interpolator takes destination **t1** into its planning and discards the destination from **t0**. When calculating the new end point from the destination for **t1**, the precision interpolator includes the current velocity.

The result is a smooth path motion to the new destination.

The event GoalReached() is only triggered when the interpolator has reached the destination state. The figure (>>> *Fig. 5-3*) shows that the destination set at time **t0** is not reached, as the new destination setting at time **t1** has overwritten the original destination setting. In this way, the event IServoOnGoalReachedEvent(…) refers to the reaching of the end position from the call of the setDestination(…) method at time **t1**.

5.4 Seconds hand effect

Destinations set serially with the setDestination(…) method can achieve very fluid motion. However, if the specified end point is reached before the next destination is set, this leads to a “seconds hand” effect, in spite of the jerk-limited path motion.

NOTICE

The robot may be damaged if the seconds hand effect occurs. KUKA is not liable for damage resulting from the continuous operation of the robot with the seconds hand effect.

The seconds hand effect is manifested by buzzing/whistling and jerky movement of the robot. This effect can generally be attributed to incorrect time coordination between the robot application and the robot controller.

In the case of fast destination settings, text outputs on the smartHMI should be avoided, as these require computing time.

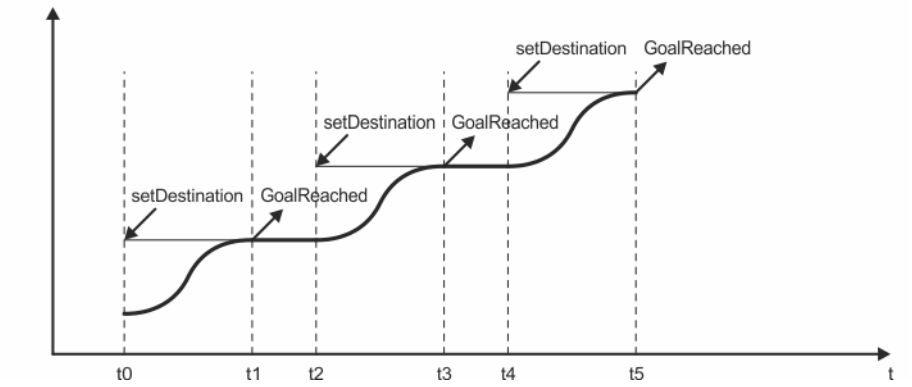


Fig. 5-4: Servo motion with seconds hand effect

Measures for avoiding the seconds hand effect can be found here: (>>> *8.2 "Robot makes jerky motions/whistles – avoiding the seconds hand effect" Page 53*)

5.5 IServoRuntime interface

Overview

The IServoRuntime interface provides the methods that are jointly used by the motion classes. The runtime environment is available as soon as the servo motion has been activated by the moveAsync(…) method.

Depending on the motion class used, a class-specific runtime environment is also made available:

- SmartServo: ISmartServoRuntime
- DirectServo: IDirectServoRuntime
- SmartServoLIN: ISmartServoLINRuntime

IServoRuntime

The methods made available by the IServoRuntime runtime environment are jointly used by the motion classes:

- Polling of data from the real-time system
  - Polling time stamps
    (>>> *5.5.1.1 "Polling time stamps" Page 22*)
  - Polling whether the end position has been reached
    (>>> *5.5.1.2 "Polling the reaching of the end position" Page 23*)
  - Polling the time still required to reach the end position
    (>>> *5.5.1.2 "Polling the reaching of the end position" Page 23*)
  - Polling and providing debug information
    (>>> *5.5.1.3 "Reading out and displaying debug information " Page 23*)
- Axis-specific interface
  - Reading the axis-specific actual position
    (>>> *5.5.2.3 "Reading out the axis-specific actual position" Page 24*)
- Cartesian interface
  - Reading the Cartesian actual position in the robot base coordinate system

(>>> *5.5.3.3 "Reading out a Cartesian actual position in the robot base coordinate system" Page 27*)

  - Reading the Cartesian actual position in the reference coordinate system

    (>>> *5.5.3.4 "Reading out the Cartesian actual position in the reference coordinate system" Page 28*)

- Stopping the servo motion

  (>>> *5.5.4 "Stopping a servo motion" Page 29*)

ISmartServo Runtime

The methods made available by the ISmartServoRuntime runtime environment are used by the SmartServo motion class:

- Axis-specific interface
  - Setting an axis-specific setpoint position

    (>>> *5.5.2.1 "Setting an axis-specific end position" Page 23*)
  - Setting a Cartesian setpoint position with command velocity at the end point

    (>>> *5.5.2.2 "Setting an axis-specific end position with a target velocity" Page 24*)
- Cartesian interface
  - Setting a Cartesian setpoint position

    (>>> *5.5.3.1 "Setting a Cartesian end position" Page 25*)

IDirectServo Runtime

The methods made available by the IDirectServoRuntime runtime environment are used by the DirectServo motion class:

- Axis-specific interface
  - Setting an axis-specific setpoint position

    (>>> *5.5.2.1 "Setting an axis-specific end position" Page 23*)
- Cartesian interface
  - Setting a Cartesian setpoint position

    (>>> *5.5.3.1 "Setting a Cartesian end position" Page 25*)

ISmartServoLINRuntime

The methods made available by the ISmartServoLINRuntime runtime environment are used by the SmartServoLIN motion class:

- Cartesian interface
  - Setting a Cartesian setpoint position

    (>>> *5.5.3.1 "Setting a Cartesian end position" Page 25*)
  - Setting a Cartesian setpoint position with command velocity at the end point

    (>>> *5.5.3.2 "Setting a Cartesian end position with a target velocity" Page 26*)

5.5.1 Polling of data from the real-time system

The runtime environment provides process data. To receive current data during polling, synchronization with the real-time system must be carried out.

The synchronization can occur as follows:

- Explicitly, by calling the updateWithRealtimeSystem() method
- Implicitly, by setting a new end position using setDestination(…)

5.5.1.1 Polling time stamps

Description

Each interaction with the real-time system returns a time stamp. The following time stamps are available and can be polled:

- Time stamp of the last setpoint command:
  `long IServoRuntime.getTimeStampOfSetRealtimeDestination()`
  This time stamp refers to the arrival of the last command set to the robot controller by calling the setDestination(…) method.
- Time stamp of the last update:
  `long IServoRuntime.getTimeStampOfUpdate()`
  This time stamp refers to the last update time by calling the updateWithRealtimeSystem() method.

Overview

The overview shows the relationship between runtime data and the respective time stamp.

Runtime data	Method	Time stamp
Axis-specific destination setting with setDestination(…)	getCurrentJointDestination()	getTimeStampOfSetRealtimeDestination()
Cartesian destination setting with setDestination(…)	getCurrentCartesianDestination()	
Interpolator state	getFineIpoState()	getTimeStampOfUpdate()
	isDestinationReached()	
Expected time until destination is reached	getRemainingTime()	
Current setpoint position in the kernel system	getCommandResultOfInterpolation()	
Actual position	getAxisQMsrOnController()	
Variance of the force estimate	getVariance()	
Interaction force on the flange	getExtForceMsr()	
	getExtForceVector()	
Axis-specific interaction torque	getExtTorqueVector()	
	getAxisTauExtMsr()	

5.5.1.2 Polling the reaching of the end position

Description

To receive current information, synchronization with the real-time system must be carried out.

(>>> *5.5.1 "Polling of data from the real-time system" Page 22*)

The SmartServo and SmartServoLIN motion types provide a notification service which signals that the end position has been reached.

(>>> *5.6.1 "Notification upon achievement of target" Page 30*)

Syntax

- Poll of whether the robot has reached the current end position:
  `boolean IServoRuntime.isDestinationReached()`
- Poll of how long (in seconds) the robot still requires to reach the current end position:
  `double IServoRuntime.getRemainingTime()`

5.5.1.3 Reading out and displaying debug information

Description

All values which are read out, e.g. the setpoint and actual position, the access timing statistics or the current interpolator state, can be displayed as debug information on the smartHMI.

The text output is executed with the IServoRuntime.toString() method:

```
getLogger().info("aTextForGood " + RuntimeInstance.to-
String());
```

The amount of detail with which this debug information is displayed can be defined with the setDetailedOutput(…) method.

Syntax

`void IServoRuntime.setDetailedOutput(int` *levelOfVerbosity*`)`

Explanation of the syntax

Element	Description
levelOf Verbosity	Amount of detail with which the debug information is displayed • **1**: Standard information • **2**: Information which extends beyond the standard • **3**: All available information

5.5.2 Axis-specific interface

5.5.2.1 Setting an axis-specific end position

Description

For SmartServo and DirectServo motions, an axis-specific end position can be set with setDestination(…).

When the DirectServo motion type is used, the destinations may be at a maximum distance of 5° from the current actual position.

Syntax

SmartServo:

`long ISmartServoRuntime.setDestination(JointPosition` *destination*`)`

DirectServo:

`long IDirectServoRuntime.setDestination(JointPosition` *destination*`)`

Explanation of the syntax

Element	Description
destination	Axis-specific end position which is to be addressed The axis angles of axes A1 … A7 must be specified (type: double; unit: rad).

Return value

Time stamp of the command registration by the robot controller.

5.5.2.2 Setting an axis-specific end position with a target velocity

Description

For SmartServo motions, it is also possible to set the axis-specific velocity with which the end position is to be addressed using setDestination(…).

Syntax

`long ISmartServoRuntime.setDestination(JointPosition` *destination*`, JointPosition` *jointSpeed*`)`

Explanation of the syntax

Element	Description
destination	Axis-specific end position which is to be addressed The axis angles of axes A1 … A7 must be specified (type: double; unit: rad).
jointSpeed	Axis-specific velocity with which the end position is to be addressed (unit: rad/s) **Note**: This parameter is only available for the SmartServo motion type.

Return value

Time stamp of the command registration by the robot controller.

5.5.2.3 Reading out the axis-specific actual position

Description

The current axis-specific actual position can be read out with getCurrentJointDestination().

Syntax

```
JointPosition IServoRuntime.getCurrentJointDestination()
```

Return value

Current axis-specific actual position.

5.5.3 Cartesian interface

5.5.3.1 Setting a Cartesian end position

Description

A Cartesian end position can be specified with setDestination(…). The end position must refer to one of the following coordinate systems:

- The robot base coordinate system.

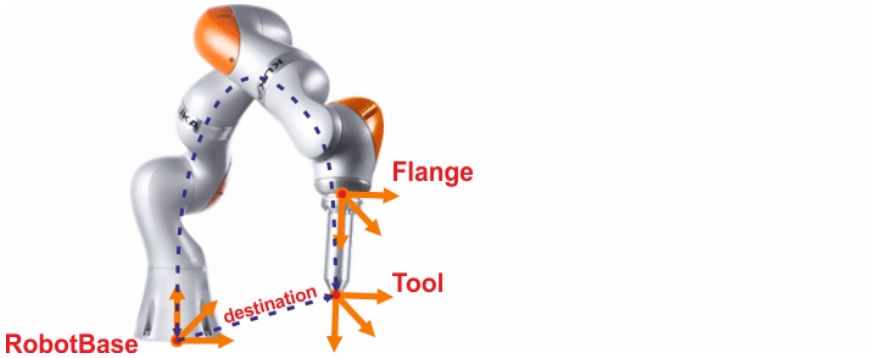


Fig. 5-5: Destination in the robot base coordinate system

- A robot base coordinate system statically connected to a reference coordinate system

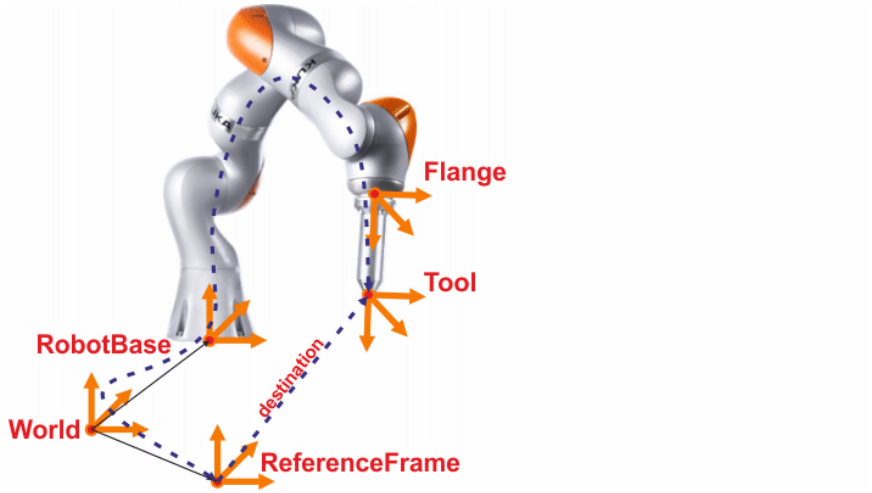


Fig. 5-6: Destination in the reference coordinate system

Syntax

```
long IServoRuntime.setDestination(AbstractFrame destination);
```

Explanation of the syntax

Element	Description
destination	Cartesian end position which is to be addressed. The end position must be defined in the robot base coordinate system or a statically connected reference coordinate system. The reference coordinate system corresponds to the parent frame of the end position and can be polled using the getParent() method of the AbstractFrame class. The end position is addressed with the robot flange or a frame subordinated to the flange, e.g. the TCP of a tool.

For SmartServoLIN, the change of orientation about Euler angle B (= rotation about the Y axis) is limited to ± 60°. This reduces the risk of the occurrence of a “gimbal lock” – i.e. a singularity that can particularly occur in connection with Euler angles – to a minimum.
The permitted change of orientation is relative to the initial orientation when starting the SmartServoLIN motion.

Return value

Time stamp of the command registration by the robot controller.

Example

```
theRuntime.setDestination(XYZ_Frame);
```

5.5.3.2 Setting a Cartesian end position with a target velocity

Description

For SmartServoLIN motions, it is also possible to set the translational velocity with which the end position is to be addressed using setDestination(…).

Syntax

`long ISmartServoLINRuntime.setDestination(AbstractFrame` *destination*`, double[x, y, z]` *targetVel*`)`

Explanation of the syntax

Element	Description
destination	Cartesian end position which is to be addressed. The end position must be defined in the robot base coordinate system or a statically connected reference coordinate system. The reference coordinate system corresponds to the parent frame of the end position and can be polled using the getParent() method of the AbstractFrame class. The end position is addressed with the robot flange or a frame subordinated to the flange, e.g. the TCP of a tool.
targetVel	Translational Cartesian destination velocity, relative to the reference coordinate system (unit: mm/s) • **≥ 0.0**

Element	Description
	The translational velocity must be entered for each of the 3 Cartesian degrees of freedom. **Note**: This parameter is only available for the SmartServoLIN motion type.

Return value

Time stamp of the command registration by the robot controller.

5.5.3.3 Reading out a Cartesian actual position in the robot base coordinate system

Description

The current Cartesian actual position of the robot relative to the robot base coordinate system can be read out with getCurrentCartesianPosition(…).

Syntax

```
Frame IServoRuntime.getCurrentCartesianPosition(AbstractFrame frameOnFlange);
```

Explanation of the syntax

Element	Description
frameOnFlange	The robot flange or a frame subordinated to the flange whose Cartesian position is polled, e.g. the TCP of a tool.

Return value

Cartesian actual position of the frame *frameOnFlange* relative to the robot base coordinate system. The value is updated with updateWithRealtimeSystem().

(>>> *5.5.1 "Polling of data from the real-time system" Page 22*)

(>>> *5.5.1.1 "Polling time stamps" Page 22*)

Example

```
Frame currentPose =
 theRuntime.getCurrentCartesianPosition(XYZ_Frame);
```

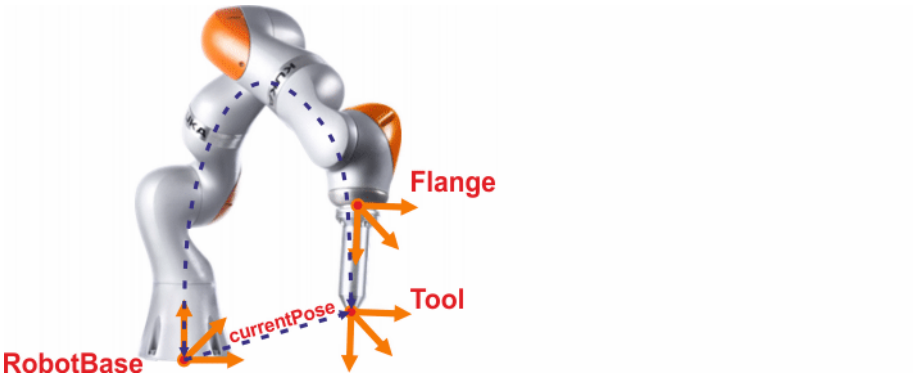


Fig. 5-7: Actual position in the robot base coordinate system

5.5.3.4 Reading out the Cartesian actual position in the reference coordinate system

Description

The current Cartesian actual position of the robot relative to a certain reference coordinate system can be read out with getCurrentCartesianPosition(…).

Syntax

```
Frame IServoRuntime.getCurrentCartesianPosition(AbstractFrame frameOnFlange, AbstractFrame reference);
```

Explanation of the syntax

Element	Description
frameOnFlange	The robot flange or a frame subordinated to the flange whose Cartesian position is polled, e.g. the TCP of a tool.
reference	Reference coordinate system relative to which the Cartesian position is polled. If no reference coordinate system is specified, the Cartesian position refers to the robot base coordinate system.

Return value

Cartesian actual position of the frame *frameOnFlange* relative to the reference coordinate system. The value is updated with updateWithRealtimeSystem().

(>>> *5.5.1 "Polling of data from the real-time system" Page 22*)

(>>> *5.5.1.1 "Polling time stamps" Page 22*)

Example

```
Frame currentPose =
 theRuntime.getCurrentCartesianPosition(XYZ_FrameTool,
XYZ_FrameRef);
```

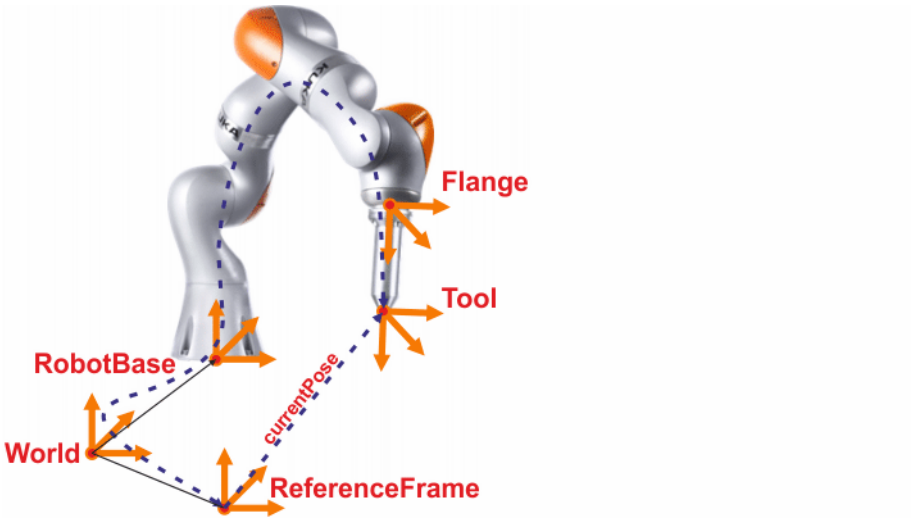


Fig. 5-8: Actual position in the reference coordinate system

5.5.4 Stopping a servo motion

Description

A servo motion can be terminated at any time using the stopMotion() method, irrespective of the status of the application or the robot.

The current robot motion is braked and no further destination set with setDestination(…) is applied. If a subsequent motion is programmed with move(…)/moveAsync(…) in the robot application, the robot motion is continued.

Syntax

```
boolean IServoRuntime.stopMotion()
```

5.6 Destination reached and timeouts

Description

The IServoRuntime class contains the isDestinationReached() method. It can be used to poll whether the current end position has been reached. The return value is of type boolean.

Once the end position has been reached, the following timeouts are relevant:

- Only for SmartServo and SmartServoLIN: Timeout for velocity planning ("speedTimeoutAfterGoalReach")
  - Precondition: Velocity planning is activated.
    - ISmartServoRuntime.activateVelocityPlanning(true)
    - ISmartServoLINRuntime.activateVelocityPlanning(true)

  The velocity reached at the end position can be retained for a brief time (≤ 100 ms). The duration of the velocity to be maintained is defined for the corresponding servo motion using the setSpeedTimeoutAfterGoalReach(…) method.
- For all servo motions: Standard timeout ("timeoutAfterGoalReach")

  By default, the wait for a new destination setting is 30 seconds. If no new end position is set in this time, the servo motion is canceled.

State diagram

A state diagram illustrates the relationship between the reaching of the end position and the timeouts. The state diagram shows the inner state machine of the precision interpolator:

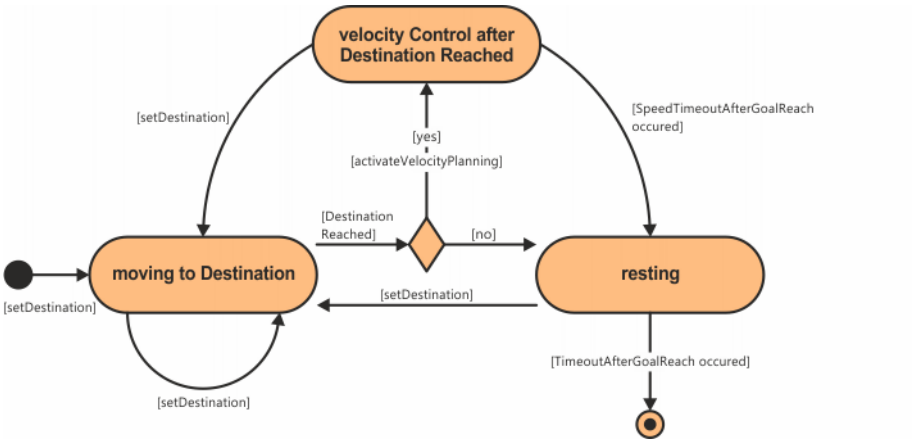


Fig. 5-9: Destination reached and timeouts (state diagram)

- If a new destination setting is sent with the setDestination(…) method, the precision interpolator changes to the “moving” state.
- As long as a motion remains to be carried out, the precision interpolator is in the “moving” state.
- Only for SmartServo and SmartServoLIN: Once the destination has been reached, the velocity of the destination state is maintained in the “velocityAfterGoalReach” state for a defined period of time:
  - The “velocityAfterGoalReach” state is only reached if the velocity planning has been activated (activateVelocityPlanning(true)).
  - If a new axis-specific end position is set during a SmartServo motion, the destination velocity can be also determined with the setDestination(…) method.
  - If a new Cartesian end position is set during a SmartServoLIN motion, the destination velocity can be also determined with the setDestination(…) method.
- Following the “velocityAfterGoalReached” or “moving” state, the precision interpolator changes to the “resting” state.
- The “resting” state is terminated after the time defined in the variable “timeoutAfterGoalReach” has elapsed. The default value for this timeout is 30 seconds.
- Every state can be terminated with the stopMotion() command at any time.

5.6.1 Notification upon achievement of target

Description

For SmartServo and SmartServoLIN motions, the listener SampleGoalReachedEventListener is available. It signals that the destination setting has been reached using the event onGoalReachedEvent(…).

The listener must be derived from the class IServoOnGoalReachedEvent and be registered on the interface IServoRuntime by calling the method setGoalReachedEventHandler(…).

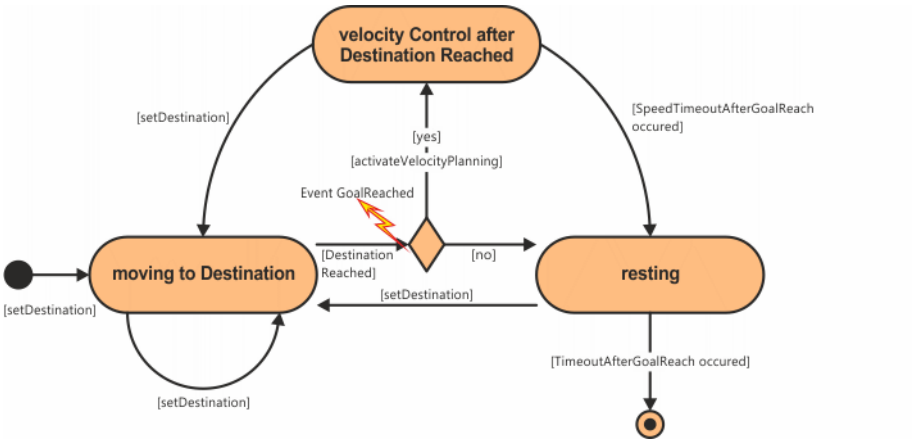


Fig. 5-10: OnGoalReachedEvent (SmartServo, SmartServoLIN)

Example

```
/**
 * Simple Implementation of an
 * GoalReachedEventHandler.Increase a local counter
 * and print an error message if a goal was reached.
 */
private class SampleGoalReachedEventListener
 implements IServoOnGoalReachedEvent
{
     @Override
     public void onGoalReachedEvent(String state,
      double[] currentAxisPos, int[] osTimestamp)
     {
          _count++;
          if (theSmartServoRuntime.isDestinationReached())
          {
               getLogger().info("ok");
          }
     }
}
```

5.7 Changing the path parameters

Description

The path parameters velocity and acceleration can be defined before a servo motion is activated.

- SmartServo and DirectServo: Standardized values from 0 … 1
- SmartServoLIN: Absolute values

The preset performance parameters refer to a full load in the extended position. In order to fully exploit the performance reserves of the robot, the following motion parameters can optionally be increased with SmartServo:

- SmartServo.overrideJointAcceleration(…): Axis-specific acceleration factor

NOTICE

An excessive increase in the axis-specific acceleration can result in damage to the machine. It is therefore advisable to only increase the acceleration factor if the robot is not reacting in a sufficiently dynamic manner.
The recommended acceleration factor can be determined using the ratio between the rated payload and the actual payload.
Example: An LBR iiwa 7 R800 has a rated payload of 7 kg. The actual payload is 3.5 kg. This means that the recommended acceleration factor is 7/3.5 = 2.

The performance reserves are directly dependent on application-specific parameters, e.g. the load and the robot configuration.

Under certain circumstances, depending on the load and the robot configuration, the robot can no longer follow the desired path. It stops moving and an error message is generated.

Overview

The following configurable path parameters are available for servo motions:

Method/Description	SmartServo	DirectServo	SmartServoLIN
ServoMotion.setJointVelocityRel(…) Axis-specific relative velocity (type: double or double[]) • **0.0 … 1.0**	✓	✓	✗
ServoMotion.setJointAccelerationRel(…) Axis-specific relative acceleration (type: double or double[]) • **0.0 … 1.0**	✓	✗	✗
ServoMotion.overrideJointAcceleration(…) Axis-specific acceleration factor (type: double) • **0.0 … 10.0**	✓	✗	✗
ServoMotion.setMaxTranslationVelocity(…) Maximum translational velocity for each Cartesian degree of freedom (type: double[x, y, z], unit: mm/s) • **> 0.0**	✗	✗	✓
ServoMotion.setMaxTranslation Acceleration(…) Maximum translational acceleration for each Cartesian degree of freedom (type: double[x, y, z], unit: mm/s^2) • **> 0.0**	✗	✗	✓
ServoMotion.setMaxOrientationVelocity(…)	✗	✗	✓

Method/Description	SmartServo	DirectServo	SmartServoLIN
Maximum rotational velocity for each Cartesian degree of freedom (type: double[a, b, c], unit: rad/s) • **> 0.0**			
ServoMotion.setMaxOrientation Acceleration(…) Maximum rotational acceleration for each Cartesian degree of freedom (type: double[a, b, c], unit: rad/s^2) • **> 0.0**	✗	✗	✓
ServoMotion.setMaxNullSpaceVelocity(…) Maximum velocity for the redundancy degree of freedom (type: double, unit: rad/s) • **> 0.0**	✗	✗	✓
ServoMotion.setMaxNullSpace Acceleration(…) Maximum acceleration for the redundancy degree of freedom (type: double, unit: rad/s^2) • **> 0.0**	✗	✗	✓

5.8 Transition to servo motion without exact positioning

Description

If a standard motion (e.g. PTP, LIN, CIRC, SPL) is terminated prematurely, the robot stops with exact positioning. Approximation into a subsequent motion is not possible here.

SmartServo and SmartServoLIN motions are capable of already taking over the robot during the motion. This allows terminated motions to be superseded without forcing exact positioning. Instead, a fluid motion transition similar to approximation can be achieved in order, for example, to react to an external event with immediate execution of an evasive movement.

In order to achieve this fluid transition, the preceding standard motion must be programmed as follows:

- Motion command must be executed asynchronously.
- Motion command must be linked to a condition and terminated with a .breakWhen(…) command.

The motion transition without exact positioning cannot be achieved by approximation into a servo motion. The regular approximation commands setBlendingCart() and setBlendingRel() have no effect on subsequent servo motions, which means that exact positioning is always executed at the end point of the standard motion. A response similar to approximation can only be achieved with premature termination of the standard motion and immediate takeover by a servo motion.

Example

Motion transition with exact positioning:

Exact positioning is always executed in the case of a break condition between 2 standard motions.

```
1 MotionPathCondition pathCondition =
2 MotionPathCondition.createFromDistance(ReferenceType.START,
  100);
3
4 _lbr.moveAsync(lin(P1).breakWhen(pathCondition));
5 _lbr.move(lin(P2));
```

Line	Description
1, 2	Path-related condition is used as a criterion for terminating an asynchronous linear motion.
4, 5	Synchronous linear motion follows asynchronous linear motion (transition with exact positioning).

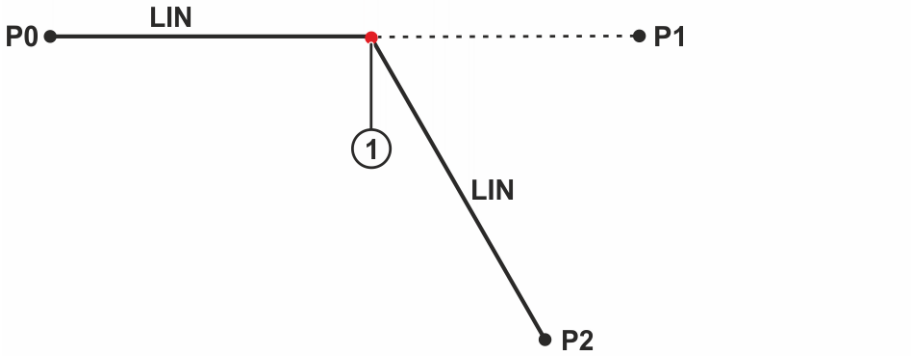


Fig. 5-11: LIN-LIN transition with exact positioning

1 Termination of LIN motion and exact positioning

Motion transition without exact positioning:

To prevent termination of a standard motion from resulting in exact positioning, a servo motion (SmartServo or SmartServoLIN) is required as the subsequent motion. This leads to a fluid transition between the motions.

```
1 MotionPathCondition pathCondition =
2 MotionPathCondition.createFromDistance(ReferenceType.START,
  100);
3
4 _lbr.moveAsync(lin(P1).breakWhen(pathCondition));
5 _lbr.move(new SmartServoLIN(P2));
```

Line	Description
1, 2	Path-related condition is used as a criterion for terminating an asynchronous linear motion.
4, 5	SmartServoLIN motion follows asynchronous linear motion (fluid transition without exact positioning).

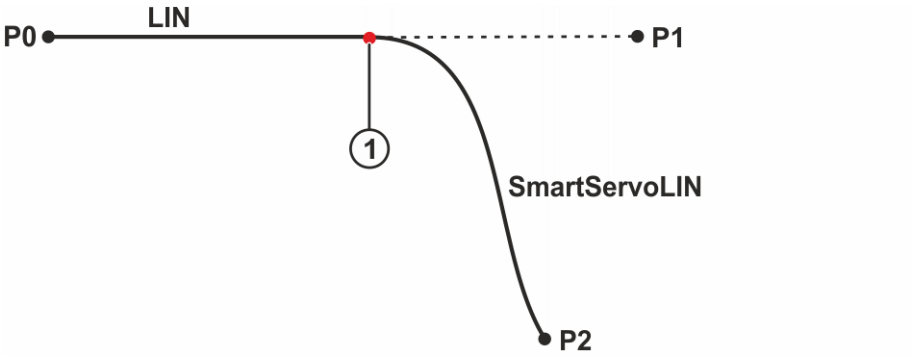


Fig. 5-12: LIN-SmartServoLIN transition without exact positioning

1 Termination of LIN motion without exact positioning

5.9 Time recording

5.9.1 Measuring the application cycle time

Two timers integrated by KUKA and a freely usable timer for the user are available for time measurement. The timers can be used for logging recurring events.

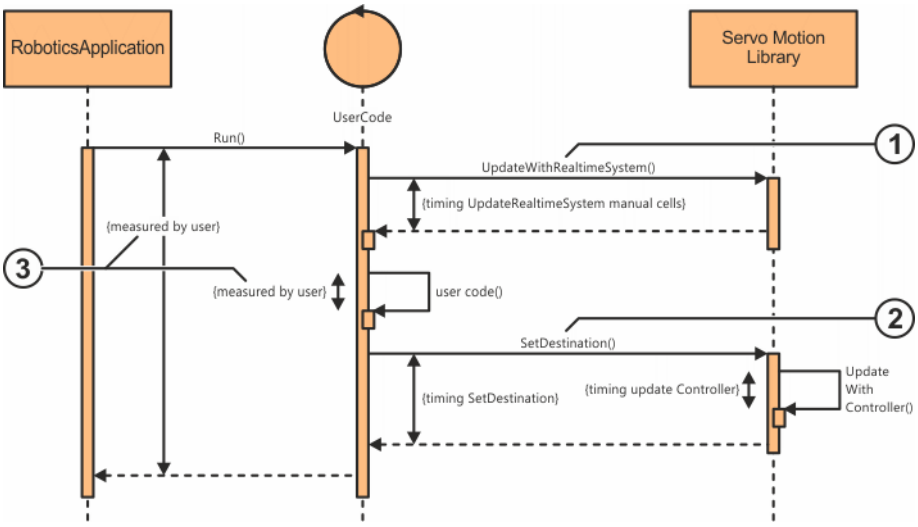


Fig. 5-13: Time measurement

1 Timer UpdateWithRealtimeSystem() (predefined)
2 Timer SetDestination() (predefined)
3 User-defined timer (StatisticTimer class)

KUKA timers

The two predefined timers are executed by calling the updateWithRealtimeSystem() method, and indirectly by calling the setDestination(…) method.

The statistics of the internal timers can be output using the toString() method of the IServoRuntime class.

User timer

The freely usable timer is provided by the StatisticTimer class. This saves the number of measurements as well as the shortest, average and longest measured cycle times.

For the StatisticTimer class to be available, it must itself be instanced:

```
import com.kuka.common.StatisticTimer;
import com.kuka.common.StatisticTimer.OneTimeStep;
...
StatisticTimer timing = new StatisticTimer();
```

Starting an individual measurement:

```
OneTimeStep aStep = timing.newTimeStep();
```

Stopping the measurement:

```
aStep.end();
```

Displaying the memory content of the StatisticTimer class on the smartHMI:

```
getLogger().info("Statistic Timing of Overall Loop " +
 timing.toString());
```

The display contains the number n of measurements and the statistic T of the cycle times. Statistic T contains the shortest, average and longest measured cycle times.

```
Statistic Timing of Overall Loop n(20687) T(0.91, 1.92, 6.36
msec)
 (stdDev 0.36)
```

More information about the StatisticTimer class is contained in Javadoc.

5.10 Impedance control

CAUTION

In impedance control, inaccurate sensor information or incorrectly selected parameters (e.g. incorrect load data, incorrect tool) can be interpreted as external forces, resulting in unpredictable motions of the robot.

CAUTION

In order to prevent injuries or damage to property caused by unpredictable motions of the robot under impedance control, a robot's maximum permissible range of motion and maximum permissible velocity must be limited under impedance control. The limit values for the maximum deviation and velocity must be selected as appropriate for the application and must be low enough.

5.10.1 Validating the load model for impedance control

Description

In order to use servo motions with a robot under impedance control, the load model for impedance control must be validated once during the runtime of the application using the validateForImpedanceMode(…) method.

A precondition for the validation is that the robot is in a well-conditioned position, e.g. it must not be in a singularity position.

It is advisable to carry out a position-controlled motion (e.g. a PTP motion) directly before validation. This ensures that the system uses the correct load data for validation.

Example

The load model for impedance control is validated for a SmartServo motion. If the validation is invalid, a message is displayed on the smartHMI.

```
double startPos[]={0.0, 0.2, 0.0, 1.5, 0.0, 1.5, 0.0};
_toolAttachedToLBR.move(ptp(startPos));
if (SmartServo.validateForImpedanceMode(_toolAttachedToLBR) !
= true)
{
     getLogger().info("Validation of Torque Model failed");
}
```

5.10.2 Activating impedance control

When activating a servo motion with moveAsync(…), the control type used must be transferred to the motion as a parameter. The setMode(…) method is used for this purpose.

```
_toolAttachedToLBR.getDefaultMotionFrame().moveAsync(
aSmartServoMotion.setMode(controlMode));
```

5.10.3 Changing controller parameters during runtime

Description

With the changeControlModeSettings(…) method, it is possible to change the controller parameters with which the servo motion was started. The control type itself cannot be changed during the servo motion.

Precondition

- The load model for impedance control has been validated.
- The control type was set when the servo motion was activated.

Syntax

`void IServoRuntime.changeControlModeSettings(IMotionControlMode` *controlMode*`)`

Explanation of the syntax

Element	Description
controlMode	Controller object which contains the controller parameters to be changed during the runtime of the servo motion.

Overview

The following parameters for each control type can be changed with the changeControlModeSettings(…) method during the runtime:

Controller	Changeable parameters	Method
Cartesian impedance controller	Data type: CartesianImpedanceControlMode	
	Spring stiffness	setStiffness(…)
	Spring damping	setDamping(…)
	Spring stiffness of the redundancy degree of freedom	setNullSpaceStiffness(…)
	Spring damping of the redundancy degree of freedom	setNullSpaceDamping(…)
Cartesian impedance controller with overlaid force oscillation	Data type: CartesianSineImpedanceControlMode	
	Amplitude of the force oscillation	setAmplitude(…)
	Frequency of the force oscillation	setFrequency(…)
	Phase offset of the force oscillation	setPhaseDeg(…)
	Constant force overlaid in addition to the force oscillation	setBias(…)
	Force limitation of the force oscillation	setForceLimit(…)
	Maximum deflection due to the force oscillation	setPositionLimit(…)
	Rise time of the force oscillation	setRiseTime(…)
	Hold time of the force oscillation	setHoldTime(…)
	Fall time of the force oscillation	setFallTime(…)
Axis-specific impedance controller	Data type: JointImpedanceControlMode	
	Spring stiffness	setStiffness(…)
	Spring damping	setDamping(…)

Example

New spring stiffness values are defined for a Cartesian impedance controller:

```
if (controlMode instanceof CartesianImpedanceControlMode)
{
     ...
     final CartesianImpedanceControlMode cartImp =
      (CartesianImpedanceControlMode) controlMode;
     final double aTransStiffVal =
      Math.max(100. * (i / (double) _numRuns + 1), 1000.);
     final double aRotStiffVal =
      Math.max(10. * (i / (double) _numRuns + 1), 150.);
     cartImp.parametrize(CartDOF.TRANSL).setStiffness(aTrans-
StiffVal);
     cartImp.parametrize(CartDOF.ROT).setStiffness(aRotStiffV-
al);
```

CAUTION

If the selected spring stiffness is insufficient, the robot cannot follow the destination setting correctly due to internal friction. This can result in spontaneous motions if the internal friction is overcome as a result of subsequent destination settings.

The new stiffness values are set in the runtime environment with the changeControlModeSettings(…) method:

```
    ...
    theSmartServoRuntime.changeControlModeSettings(cartImp);
}
```

CAUTION

The robot can move unexpectedly under impedance control. The maximum permissible Cartesian deviation from the path and the maximum permissible Cartesian force on the TCP must be restricted to such an extent that no injuries or damage to property can result.

The following methods are available for setting the Cartesian limitation:

- setMaxPathDeviation(…):
  Limitation of the maximum Cartesian deviation from the path
- setMaxControlForce(…):
  Limitation of the maximum force on the TCP

```
final CartesianImpedanceControlMode cartImp =
 new CartesianImpedanceControlMode();
cartImp.parametrize(CartDOF.TRANSL).setStiffness(1000.0);
cartImp.parametrize(CartDOF.ROT).setStiffness(100.0);
cartImp.setNullSpaceStiffness(100.0);
cartImp.parametrize(CartDOF.TRANSL).setMaxPathDevia-
tion(50.0);
cartImp.parametrize(CartDOF.ROT).setMaxPathDeviation(1.0);
```

5.10.4 Transition between control types

Description

With servo motions, it is not possible to change the control type within a motion instance. For this reason, the motion must be replaced by a new motion instance. It is only possible to change to the new control type when the running motion instance has ended.

A new servo motion is instanced with moveAsync(…). If the first motion is still in progress and is then terminated by a defined condition or the stopMotion() command, for example, there is a seamless transition between the motion sections due to the internal functionalities in the real-time area.

The same technique must be used for all motion-specific parameters which cannot be changed during the runtime. This applies, for example, to load reversals (mass parameters change with gripping) or the like.

The correct load parameters must be observed when superimposing. These are motion-specific.

Example

A first SmartServo motion is active. A second SmartServo motion is sent to the real-time system with moveAsync(…). (>>> *6.1.3 ""SmartServoSampleInteractionControl" application" Page 43*)

```
final SmartServo secondSmartServoMotion =
 newSmartServo(initialPosition);
...
_toolAttachedToLBR.getDefaultMotionFrame().
 moveAsync(secondSmartServoMotion);
...
final ISmartServoRuntime theSecondRuntime =
 secondSmartServoMotion.getRuntime();
```

The actual transfer occurs in the following instruction in which the SmartServo motion to be replaced (theFirstRuntime) is terminated.

```
theFirstRuntime.stopMotion();
```

Starting at this point in time, the second runtime environment (theSecondRuntime) is activated and carries out motion commands in the system.

5.11 Feedback effect for servo motions

Description

When using servo motions, feedback may occur as a control-related secondary effect. This effect is comparable to the feedback that occurs when a microphone is overloaded by a coupled loudspeaker.

Robot applications using standard motions (e.g. LIN or PTP) send destination settings to the robot controller without including position feedback in their destination setting calculation.

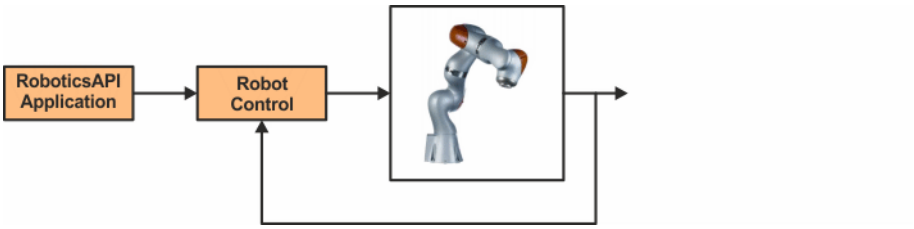


Fig. 5-14: Destination setting for standard motions

For servo motions, by contrast, position feedback can be included in the destination setting calculation. Depending on various parameters, there may be overloaded destination settings on the robot controller, and these may gradually oscillate. This can be compared to the low input attenuation of a microphone which is too close to a coupled loudspeaker.

Example 1: “visual servoing”

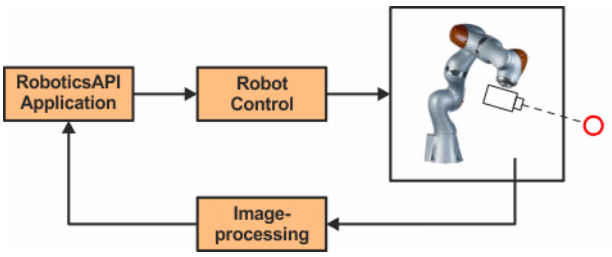


Fig. 5-15: Destination setting for servo motions, “visual servoing” example

For a “visual servoing” application, the robot is to keep the red circle in the center of the camera screen. The camera image is then recorded and evaluated. In the tracking application, a control signal is calculated to con-

trol the robot. This control signal changes the recording point of the camera.

Figure (>>> *Fig. 5-16*) illustrates the signal flow of the control loop.

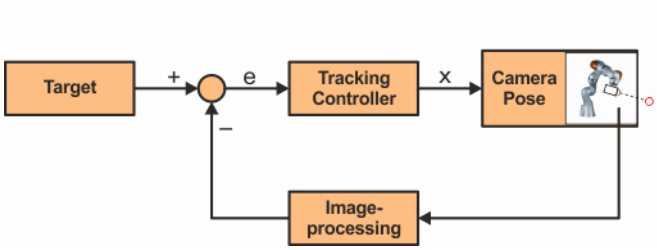


Fig. 5-16: Control loop for signal flow, “visual servoing” example

Here the activity of the robot and the robot controller are abstracted as a camera pose.

Figure (>>> *Fig. 5-17*) models both the camera pose and the image processing as ideal dead time elements Z^{-1}. The “tracking controller” is modeled as a dead time element with p gain.

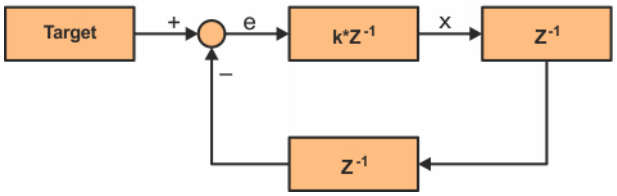


Fig. 5-17: p gain, “visual servoing” example

This results in the following transfer function: $G(z) = (k \cdot z) / (k + z^3)$

The roots of the characteristic polynomial thus depend on the gain factor k of the actual controller (“tracking controller”).

For k = 1, the control loop is critically stable.

This gives rise to the following conclusions:

- In practice, it is this control loop (k= 1) which is unstable.
  The pixel noise of the camera already leads to sustained oscillation of the overall system.
- Each subsystem is fully functional in itself.
- The loop gain of the overall system must be reduced for stabilization:
  - Select the gain k < 1 for the controller (“tracking controller”).
  - Reduce the robot performance (maximum velocity).

Example 2: adaptation

This example assumes a system with low performance which has a stable setting. The robot velocity has been reduced in order to achieve this stability.

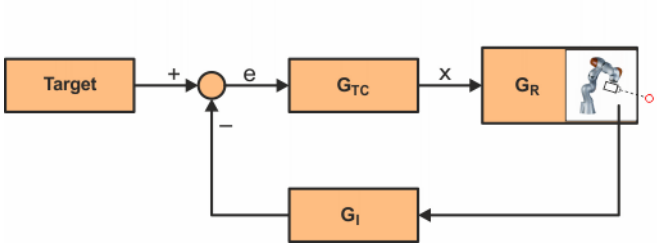


Fig. 5-18: Adaptation, “visual servoing” example

The aim is nevertheless to fully utilize the capacity of the robot. However, if the robot velocity is increased, the overall system can become unstable although each subsystem in itself is fully functional.

The following measures affect the stability of the overall system:

- Changing the maximum velocity or acceleration
- Changing between position control and impedance control
- Changing the impedance control
- Changing between the SmartServo and DirectServo motion types

Stabilization of the system can only be achieved if the application programming is adapted accordingly.

6 Application examples

The example applications build on each other. It is recommended for them to be studied in the order documented here.

6.1 Example applications for SmartServo

6.1.1 “SmartServoSampleSimpleJointMotion” application

Precondition

- Advanced knowledge of programming with KUKA RoboticsAPI

Functionality

- Simple axis-specific destination setting

Learning objectives

- Activating the SmartServo motion type
  (>>> *5.1 "Servo motion in an application context" Page 17*)
- Working with the IServoRuntime runtime environment
  (>>> *5.5 "IServoRuntime interface" Page 20*)
- Axis-specific destination setting
  (>>> *5.5.2 "Axis-specific interface" Page 23*)
- Time measurement
  (>>> *5.9 "Time recording" Page 35*)

6.1.2 “SmartServoSampleSimpleCartesian” application

Precondition

- Programming knowledge in the SmartServoSampleSimpleJointMotion application

Functionality

- Simple Cartesian destination setting

Learning objectives

- Cartesian destination setting
  (>>> *5.5.3 "Cartesian interface" Page 25*)

6.1.3 “SmartServoSampleInteractionControl” application

Precondition

- Programming knowledge in the SmartServoSampleSimpleJointMotion application
- Programming knowledge in the SmartServoSampleSimpleCartesian application

Functionality

- Impedance control with the SmartServo motion type

Learning objectives

- Validating the load model for impedance control
  (>>> *5.10.1 "Validating the load model for impedance control" Page 36*)
- Activating impedance control
  (>>> *5.10.2 "Activating impedance control" Page 37*)
- Changing controller parameters during runtime
  (>>> *5.10.3 "Changing controller parameters during runtime" Page 37*)
- Transition between control types
  (>>> *5.10.4 "Transition between control types" Page 39*)

6.2 Example applications for DirectServo

6.2.1 “DirectServoSampleSimpleJointMotion” application

Precondition

- Programming knowledge in the SmartServoSampleSimpleJointMotion application

Functionality

- Simple axis-specific destination setting

Learning objectives

- Activating the DirectServo motion type
  (>>> *5.1 "Servo motion in an application context" Page 17*)
- Working with the IServoRuntime runtime environment
  (>>> *5.5 "IServoRuntime interface" Page 20*)
- Axis-specific destination setting
  (>>> *5.5.2 "Axis-specific interface" Page 23*)
- Time measurement
  (>>> *5.9 "Time recording" Page 35*)

6.2.2 “DirectServoSampleSimpleCartesian” application

Precondition

- Programming knowledge in the SmartServoSampleSimpleCartesian application
- Programming knowledge in the DirectServoSampleSimpleJointMotion application

Functionality

- Simple Cartesian destination setting

Learning objectives

- Cartesian destination setting
  (>>> *5.5.3 "Cartesian interface" Page 25*)

6.2.3 “DirectServoSampleInteractionControl” application

Precondition

- Programming knowledge in the SmartServoSampleSimpleJointMotion application
- Programming knowledge in the SmartServoSampleSimpleCartesian application
- Programming knowledge in the SmartServoSampleInteractionControl application

Functionality

- Impedance control with the DirectServo motion type

Learning objectives

- Validating the load model for impedance control
  (>>> *5.10.1 "Validating the load model for impedance control" Page 36*)
- Activating impedance control
  (>>> *5.10.2 "Activating impedance control" Page 37*)
- Changing controller parameters during runtime
  (>>> *5.10.3 "Changing controller parameters during runtime" Page 37*)
- Transition between control types
  (>>> *5.10.4 "Transition between control types" Page 39*)

6.3 Example applications for SmartServoLIN

6.3.1 “SmartServoLINSimpleMotion” application

Precondition

- Advanced knowledge of programming with KUKA RoboticsAPI

Functionality

- Simple Cartesian destination setting

Learning objectives

- Activating the SmartServoLIN motion type
  (>>> *5.1 "Servo motion in an application context" Page 17*)
- Working with the IServoRuntime runtime environment
  (>>> *5.5 "IServoRuntime interface" Page 20*)
- Cartesian destination setting
  (>>> *5.5.3 "Cartesian interface" Page 25*)
- Time measurement
  (>>> *5.9 "Time recording" Page 35*)

6.3.2 “SmartServoLINInteractionControl” application

Precondition

- Programming knowledge in the SmartServoLINSimpleMotion application

Functionality

- Impedance control with the SmartServoLIN motion type

Learning objectives

- Validating the load model for impedance control
  (>>> *5.10.1 "Validating the load model for impedance control" Page 36*)
- Activating impedance control
  (>>> *5.10.2 "Activating impedance control" Page 37*)
- Changing controller parameters during runtime
  (>>> *5.10.3 "Changing controller parameters during runtime" Page 37*)

7 Exception handling routines

7.1 Example of an exception handling routine

Description

The servo motion classes provide exceptions which allow runtime errors to be transferred to the application.

These exceptions can be dealt with by the application with the Java-compliant try/catch block. Exceptions which are not dealt with by the application cause the application to terminate.

Example

In the following programming example, all runtime exceptions are intercepted.

If an error occurs in the try block, execution of the try block is terminated and the catch block is then executed immediately.

The error is intercepted in the catch block and an error-specific message is generated.

```
try
{
     // Create the motion
     SmartServo aSmartServoMotion = new SmartServo(initialPo-
sition);
     _toolAttachedToLBR.getDefaultMotionFrame().
      moveAsync(aSmartServoMotion);
     // Fetch the Runtime of the Motion part
     IServoRuntime theServoRuntime = aSmartServoMotion.ge-
tRuntime();
     // do repetitive loop
     do
     {
          // Do some computations
          //
          theServoRuntime.setDestination(destination);
     } while (notFinished);
}
catch (Exception e)
{
     getLogger().error(e.getMessage());
     throw(new Exception(e));
}
```

7.2 Overview of exceptions – error messages

7.2.1 Exceptions of the IServoRuntime interface

Exception/Source	Message	Cause/Remedy
Type: IllegalArgumentException Source: • getCurrentCartesianDestination(…)	“frameOnFlange is null”	The frame *frameOnFlange* has not been specified.
	“base is null”	No reference coordinate system has been specified.

Exception/Source	Message	Cause/Remedy
• getCurrentCartesianPosition(…)	“frameOnFlange is not frame below or equal to the flange of this robot”	The specified frame *frameOnFlange* is not connected to the robot flange.
	“base is not static relative to the base of this robot”	The reference coordinate system is connected to the robot flange and can therefore be moved.
Type: IllegalArgumentException Source: • setMinimumTrajectoryExecutionTime(…)	“The value of the minimum synchronization time is > 0.1”	The value of the minimum synchronization time is > 0.1. • Select a value between 0.0 … 0.1.
Type: IllegalStateException Source: • stopMotion()	“Servo Motion can't stop, because no servo motion is active”	Servo motion cannot be stopped as there is no servo motion active.
Type: IllegalStateException Source: • waitForTransfered()	“The realtime system didn’t respond”	The servo motion has been requested with moveAsync(…). The time until activation has been exceeded. Possible causes: • The application is paused. • Preceding motions in the application take too long.
	“No servo motion is active”	The servo motion has not yet been activated with moveAsync(…). • Check the order of the programming.
Type: CommandInvalidException Source: • updateWithRealtimeSystem()	“Command is no longer active, because” + *errorReason*	An error occurred during the update of the application data with the real-time system data. The cause of the error (= *errorReason*) is specified in the message. (>>> *7.2.4 "CommandInvalidException error causes" Page 49*)

7.2.2 Exceptions of the ISmartServoLINRuntime interface

Exception/Source	Message	Cause/Remedy
Type: IllegalArgumentException Source: • setDestination(…)	“Current destination is null”	Cartesian end position is missing. Enter Cartesian end position.
	“The vector of the target velocity must have a length of 3”	Enter the translational destination velocity for each of the 3 Cartesian degrees of freedom for the reference coordinate system.
	“The value of the target velocity is less than 0”	For at least one Cartesian degree of freedom, the value for the translational destination velocity is < 0.0.

Exception/Source	Message	Cause/Remedy
		• Select values ≥ 0.0.
Type: IllegalArgumentException Source: • setMaxNullSpaceVelocity(…)	"The value of the maximum null space velocity is equal or less than 0"	The value of the maximum velocity for the redundancy degree of freedom is ≤ 0.0. • Select a value > 0.0.
Type: IllegalArgumentException Source: • setMaxOrientationVelocity(…)	"The vector of the target velocity must have a length of 3"	Enter the maximum rotational velocity for each of the 3 Cartesian degrees of freedom.
	"The value of the maximum orientation velocity is equal or less than 0"	For at least one Cartesian degree of freedom, the value for the maximum rotational velocity is ≤ 0.0. • Select values > 0.0.
Type: IllegalArgumentException Source: • setMaxTranslationVelocity(…)	"The vector of the target velocity must have a length of 3"	Enter the maximum translational velocity for each of the 3 Cartesian degrees of freedom.
	"The value of the maximum translation velocity is equal or less than 0"	For at least one Cartesian degree of freedom, the value for the maximum translational velocity is ≤ 0.0. • Select values > 0.0.

7.2.3 Exceptions of the IServoMotion interface

Exception/Source	Message	Cause/Remedy
Type: IllegalStateException Source: • validateforImpedance-Mode(…)	"Load Mass Validation failed"	The validation of the load model for the impedance control has failed since incorrect load data were used. • Respecify load data and perform validation with the correct load data.
Type: IllegalStateException Source: • getRuntime()	"Servo Motion is not activated – call moveAsyc/move before access to runtime"	The servo motion has been requested with moveAsync(…). The time until activation has been exceeded. Possible causes: • The application is paused. • Preceding motions in the application take too long.

7.2.4 CommandInvalidException error causes

errorReason	Cause	Remedy
"tracking error"	The difference between the setpoint position and the actual position is too great. The	• Check the loads on the robot.

errorReason	Cause	Remedy
	robot cannot follow the command path. Possible causes: • The robot is overloaded, e.g. because the tool used is too heavy. Attention must also be paid to the load of a gripped workpiece. • Load data have been entered incorrectly. • Command velocity or command acceleration is too high.	• Check the load data and correct if required. • Check the setpoint inputs for velocity and acceleration in setDestination(…) and reduce these if necessary.
“speed limit exceeded”	Destination interval is too high.	Reduce the interval on sequential destination settings with setDestination(…).
“checkAxisTorqueLimit”	The torque specification of the servo control is too high. Possible causes: • The robot is overloaded, e.g. because the tool used is too heavy. Attention must also be paid to the load of a gripped workpiece. • Load data have been entered incorrectly.	• Check the loads on the robot. • Check the load data and correct if required.
“msr T1 illegal speed limit detected”	The measured Cartesian velocity in T1 mode is too high.	• In impedance control: check the external force acting on the robot. • Correct the destination setting in setDestination(…).
“cmd T1 illegal speed limit detected”	The Cartesian command velocity in T1 mode is too high.	Correct the destination setting in setDestination(…).
“directServo: Target Outside accepted Delta”	In the DirectServo motion type, the command position may be at a maximum distance of 5° from the current actual position.	Correct the destination setting in setDestination(…).
“directServo: Target Outside Joint Limit”	The destination setting is outside the permitted axis range (DirectServo).	Correct the destination setting in setDestination(…).
“interpolation error: RML_ERROR_POSITIONAL_LIMITS”	The destination setting is outside the permitted axis range (SmartServo).	Correct the destination setting in setDestination(…).
“interpolation error: ERROR_INVALID_INPUT_VALUES”	Internal error	Contact KUKA Service.
“interpolation error: ERROR”	Internal error	Contact KUKA Service.

errorReason	Cause	Remedy
“interpolation error: ERROR_EXECUTION_TIME_CALCULATION”	Internal error	Contact KUKA Service.
“interpolation error: ERROR_SYNCHRONIZATION”	Internal error	Contact KUKA Service.
“interpolation error: ERROR_NUMBER_OF_DOFS”	Internal error	Contact KUKA Service.
“interpolation error: ERROR_NO_PHASE_SYNCHRONIZATION”	Internal error	Contact KUKA Service.
“interpolation error: ERROR_NULL_POINTER”	Internal error	Contact KUKA Service.
“interpolation error: ERROR_EXECUTION_TIME_TOO_BIG”	Internal error	Contact KUKA Service.
“interpolation error: RML_ERROR_USER_TIME_OUT_OF_RANGE”	Internal error	Contact KUKA Service.
“interpolation error: RML_ERROR_OVERRIDE_OUT_OF_RANGE”	Internal error	Contact KUKA Service.

8 Troubleshooting

8.1 TCP/IP traffic from Windows is too slow

Description

The default setting for the TCP/IP network adapter under Windows results in update intervals which are too long (>100 ms). This has a negative effect on the command specification (seconds hand effect).

The update of the real-time system data with the application data should require no more than 5 ms on average.

(>>> *5.5.1.1 "Polling time stamps" Page 22*)

```
timing update Controller n(24212) T(0.89, 1.85, 6.30 msec)
 (stdDev 0.33)
timing SetDestination n(24212) T(0.91, 1.92, 6.36 msec)
 (stdDev 0.34)
```

If the numeric time statistic T(min, mean, max) records a minimum and average value of >100 ms, the cause may be the default optimization in Windows of the TCP/IP setting for high data throughput. This causes small individual TCP/IP packets to be bundled into larger TCP/IP packets, which in turn leads to delays in communication with the real-time system. The process prevents short-interval feedback from the real-time system.

In order to send unbundled, individual TCP/IP packets, 2 new keys must be created in the Windows registry database.

Precondition

- Local administrator rights

When making changes in the registry database, attention must be paid to uppercase and lowercase letters, as Windows makes a distinction here!

Procedure

1. Start **regedit** via the command prompt.
2. Navigate to the path **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\Tcpip\Parameters\Interfaces**.
3. Open the corresponding subfolder containing a key value with the IP address of the desired network adapter.
4. Right-click on the folder of the network adapter which was located beforehand.
   a. Select: **New** > **DWORD (32-bit) Value**
   b. Name the newly created entry **TcpAckFrequency**.
   c. Right-click on the newly created entry **TcpAckFrequency**.
   d. Select: **Modify**
   e. Select: **Hexadecimal**
   f. Enter the value: **1**
5. Right-click on the folder of the network adapter which was located beforehand.
   a. Select: **New** > **DWORD (32-bit) Value**
   b. Name the newly created entry **TCPNoDelay**.
   c. Right-click on the newly created entry **TCPNoDelay**.
   d. Select: **Modify**

e. Select: **Hexadecimal**
f. Enter the value: **1**

6. Check that:
   - The two newly created keys **TcpAckFrequency** and **TCPNoDelay** are located in the folder of the network adapter with the correct IP address and are correctly named.
   - Both keys are of the type **REG_DWORD**.
   - Both keys have the value **0×00000001**.
7. Reboot the Windows operating system.

8.2 Robot makes jerky motions/whistles – avoiding the seconds hand effect

NOTICE

The robot may be damaged if the seconds hand effect occurs. KUKA is not liable for damage resulting from the continuous operation of the robot with the seconds hand effect.

Precondition

- TCP/IP traffic from Windows flows quickly enough.
  (>>> *8.1 "TCP/IP traffic from Windows is too slow" Page 52*)

Remedy

The following measures can be taken to avoid the seconds hand effect:

- **Specify larger motion sections.**
  Specify larger motion sections so that the next setDestination(…) can be called on time. Since the precision interpolator is not able to reach the destination in this case, the seconds hand effect does not occur.
  A typical application of this is “visual servoing”, in which a camera/image processing specifies end poses which are far away in “slow” succession.
- **Reduce the maximum velocity and acceleration.**
  Reduce the maximum velocity and acceleration according to the expected motion pattern to the point where a new destination is set before the previous motion section is completed.
- **Specify the minimum synchronization time.**
  With setMinimumTrajectoryExecutionTime(…), set the value for the minimum synchronization time high enough so that the next destination setting can be expected before the end point is reached.

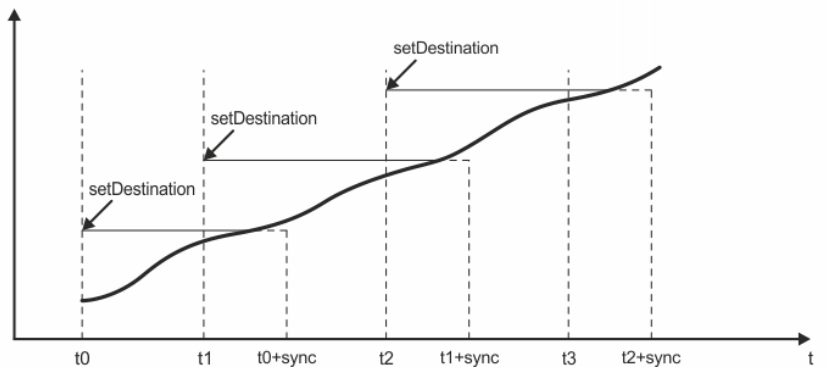


Fig. 8-1: Servo motion without seconds hand effect

As a rule of thumb, the minimum synchronization time should be 1.5 to 2 times the cycle time of the application context. The cycle time can be measured using the StatisticTimer class.

(>>> *5.9 "Time recording" Page 35*)

- **Use the SmartServo motion type.**

  In the DirectServo motion type, the jerk-limited planning reaches its limits because the recommended destination interval is very short.

8.3 Impedance control cannot be activated

Cause

The validation of the load model for the impedance control has failed.

Remedy

1. With a PTP motion, move the robot into a non-singularity pose.
2. Carry out the validation again.

8.4 Feedback effect

Description

When servo motions are used, the robot position is commanded very quickly and the current robot position is incorporated into the command.

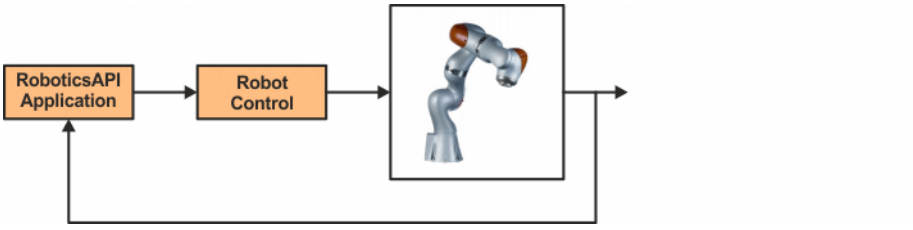


Fig. 8-2: Application is part of the control loop

This can result in a feedback effect, causing the robot to buzz and oscillate. The robot and application are in themselves fully functional.

(>>> *5.11 "Feedback effect for servo motions" Page 40*)

Remedy

In the Java application (recommended):

- Check and analyze the control law.

- Reduce amplification in the application.

On the robot (at the expense of performance):

- Reduce the robot performance:
  - Reduce the program override.
  - Reduce the maximum velocity and acceleration.
  - For impedance control: Increase the compliance by modifying the spring stiffness and spring damping.

On the robot (increasing the performance, only for SmartServo):

- Increase the maximum acceleration used for planning with overrideJointAcceleration(…).

NOTICE

An excessive increase in the axis-specific acceleration can result in damage to the machine. It is therefore advisable to only increase the acceleration factor if the robot is not reacting in a sufficiently dynamic manner.
The recommended acceleration factor can be determined using the ratio between the rated payload and the actual payload.
Example: An LBR iiwa 7 R800 has a rated payload of 7 kg. The actual payload is 3.5 kg. This means that the recommended acceleration factor is 7/3.5 = 2.

(>>> *9.4.1 "Comparing the axis positions over time" Page 63*)

(>>> *9.4.2 "Expanded comparison with velocity, acceleration and jerk" Page 70*)

9 Appendix

9.1 SmartServo command reference

9.1.1 Parameters of the SmartServo motion

Method	Description
setJointVelocityRel(…)	Axis-specific relative velocity (type: double or double[]) • **0.0 … 1.0** Default: 1.0 Refers to the maximum value of the axis velocity in the machine data.
setJointAccelerationRel(…)	Axis-specific relative acceleration (type: double or double[]) • **0.0 … 1.0** Default: 1.0 Refers to the maximum value of the axis acceleration in the machine data.
overrideJointAcceleration(…)	Increase of the axis-specific acceleration factor (type: double) • **0.0 … 10.0** Default: 1.0 **Note:** An excessive increase in the axis-specific acceleration can result in damage to the machine. It is therefore advisable to only increase the acceleration factor if the robot is not reacting in a sufficiently dynamic manner. The recommended acceleration factor can be determined using the ratio between the rated payload and the actual payload. Example: An LBR iiwa 7 R800 has a rated payload of 7 kg. The actual payload is 3.5 kg. This means that the recommended acceleration factor is 7/3.5 = 2.
setMinimumTrajectory ExecutionTime(…)	Minimum length of time from the start to the end position of the servo motion (= minimum synchronization time) (type: double, unit: s) • **0 … 0.1** Default: 0.01 **Note:** The seconds hand effect can be avoided by increasing the minimum length of time to reach the end point.
setSpeedTimeoutAfter GoalReach(…)	Timeout for the velocity planning after the end position of the servo motion is reached (type: double, unit: s) Specifies for how long the velocity reached in the end position is to be kept. • **0.0 … 0.1** Default: 0.0 **Note:** The timeout is only relevant if the velocity planning is activated: ISmartServoRuntime.activateVelocityplanning(true)
setTimeoutAfterGoal Reach(…)	Timeout after the end position of the servo motion is reached (type: double, unit: s)

Method	Description
	Specifies how long to wait for a new destination setting after the end position is reached. If no new end position is set in this time, the servo motion is canceled. • **> 0.0** Default: 30.0
setMode(…)	Control type A servo motion can be executed under impedance control. **Note:** Further information on the programming of robots under impedance control is contained in the operating and programming instructions for the System Software.

9.1.2 Parameters of the ISmartServoRuntime interface

Method	Description
changeControlMode Settings(…)	Modification of controller parameters during impedance control. (>>> *5.10.3 "Changing controller parameters during runtime" Page 37*)
setDetailedOutput(…)	The amount of detail when reading debug information • **1**: Standard information • **2**: Information which extends beyond the standard • **3**: All available information Default: 1 (>>> *5.5.1.3 "Reading out and displaying debug information " Page 23*)
setGoalReachedEvent Handler(…)	Notification service that signals the end position has been reached (type: IServoOnGoalReachedEvent) (>>> *5.6.1 "Notification upon achievement of target" Page 30*)
setMinimumTrajectory ExecutionTime(…)	Minimum length of time from the start to the end position of the servo motion (= minimum synchronization time) (type: double, unit: s) • **0 … 0.1** Default: 0.01 **Note:** The seconds hand effect can be avoided by increasing the minimum length of time to reach the end point.
activateVelocityPlanning(…)	Activation of velocity planning once the set end position has been reached (type: Boolean) • **true**: Velocity planning is activated. • **false**: Velocity planning is deactivated. Default: **False**

9.1.3 Setting the end position on the ISmartServoRuntime interface

Method	Description
setDestination(…)	Setting of a new axis-specific end position (>>> *5.5.2.1 "Setting an axis-specific end position" Page 23*)
	Setting of a new axis-specific end position and the corresponding destination velocity of the individual axes (>>> *5.5.2.2 "Setting an axis-specific end position with a target velocity" Page 24*)
	Setting of a new Cartesian end position (>>> *5.5.3.1 "Setting a Cartesian end position" Page 25*)

9.2 DirectServo command reference

9.2.1 Parameters of the DirectServo motion

Method	Description
setJointVelocityRel(…)	Axis-specific relative velocity (type: double or double[]; unit: %) • **0.0 … 1.0** Default: 1.0 Refers to the maximum value of the axis velocity in the machine data.
setMinimumTrajectory ExecutionTime(…)	Minimum length of time from the start to the end position of the servo motion (= minimum synchronization time) (type: double, unit: s) • **0 … 0.1** Default: 0.01 **Note:** The seconds hand effect can be avoided by increasing the minimum length of time to reach the end point.
setTimeoutAfterGoal Reach(…)	Timeout after the end position of the servo motion is reached (type: double, unit: s) Specifies how long to wait for a new destination setting after the end position is reached. If no new end position is set in this time, the servo motion is canceled. • **> 0.0** Default: 30.0
setMode(…)	Control type A servo motion can be executed under impedance control. **Note:** Further information on the programming of robots under impedance control is contained in the operating and programming instructions for the System Software.

9.2.2 Parameters of the IDirectServoRuntime interface

Method	Description
changeControlMode Settings(…)	Modification of controller parameters during impedance control (>>> *5.10.3 "Changing controller parameters during runtime" Page 37*)
setDetailedOutput(…)	The amount of detail when reading debug information • **1**: Standard information • **2**: Information which extends beyond the standard • **3**: All available information Default: 1 (>>> *5.5.1.3 "Reading out and displaying debug information " Page 23*)
setGoalReachedEvent Handler(…)	Notification service that signals the end position has been reached (type: IServoOnGoalReachedEvent) (>>> *5.6.1 "Notification upon achievement of target" Page 30*)
setMinimumTrajectory ExecutionTime(…)	Minimum length of time from the start to the end position of the servo motion (= minimum synchronization time) (type: double, unit: s) • **0 … 0.1** Default: 0.01 **Note:** The seconds hand effect can be avoided by increasing the minimum length of time to reach the end point.

9.2.3 Setting the end position on the IDirectServoRuntime interface

Method	Description
setDestination(…)	Setting of a new axis-specific end position (>>> *5.5.2.1 "Setting an axis-specific end position" Page 23*)
	Setting of a new Cartesian end position (>>> *5.5.3.1 "Setting a Cartesian end position" Page 25*)

9.3 SmartServoLIN command reference

9.3.1 Parameters of the SmartServoLIN motion

Method	Description
setReferenceFrame(…)	The reference coordinate system for the interpolation of Cartesian destination settings (type: AbstractFrame). Can be set in order to adapt the internal reference coordinate system used for motion planning. Interpolated in the robot base coordinate system by default. The method has no influence on the Cartesian destination setting. It can, however, if selected appropriately, lead to more precise linear motions. It is advisable to align the refer-

Method	Description
	ence coordinate system so that the axes correspond to the main direction of motion for the specific task.
setMaxTranslationVelocity(…)	Maximum translational velocity for each Cartesian degree of freedom (type: double[x, y, z], unit: mm/s) • **> 0.0** If no velocity is specified, the motion is executed with the fastest possible velocity.
setMaxTranslation Acceleration(…)	Maximum translational acceleration for each Cartesian degree of freedom (type: double[x, y, z], unit: mm/s^2) • **> 0.0** If no acceleration is specified, the motion is executed with the fastest possible acceleration.
setMaxOrientationVelocity(…)	Maximum rotational velocity for each Cartesian degree of freedom (type: double[a, b, c], unit: rad/s) • **> 0.0** If no velocity is specified, the motion is executed with the fastest possible velocity.
setMaxOrientation Acceleration(…)	Maximum rotational acceleration for each Cartesian degree of freedom (type: double[a, b, c], unit: rad/s^2) • **> 0.0** If no acceleration is specified, the motion is executed with the fastest possible acceleration.
setMaxNullSpaceVelocity(…)	Maximum velocity for the redundancy degree of freedom (type: double, unit: rad/s) • **> 0.0** If no velocity is specified, the motion is executed with the fastest possible velocity.
setMaxNullSpace Acceleration(…)	Maximum acceleration for the redundancy degree of freedom (type: double, unit: rad/s^2) • **> 0.0** If no acceleration is specified, the motion is executed with the fastest possible acceleration.
setMinimumTrajectory ExecutionTime(…)	Minimum length of time from the start to the end position of the servo motion (= minimum synchronization time) (type: double, unit: s) • **0 … 0.1** Default: 0.01 **Note:** The seconds hand effect can be avoided by increasing the minimum length of time to reach the end point.
setSpeedTimeoutAfter GoalReach(…)	Timeout for the velocity planning after the end position of the servo motion is reached (type: double, unit: s) Specifies for how long the velocity reached in the end position is to be kept. • **0.0 … 0.1**

Method	Description
	Default: 0.0 **Note:** The timeout is only relevant if the velocity planning is activated: ISmartServoLINRuntime.activateVelocityplanning(true)
setTimeoutAfterGoal Reach(…)	Timeout after the end position of the servo motion is reached (type: double, unit: s) Specifies how long to wait for a new destination setting after the end position is reached. If no new end position is set in this time, the servo motion is canceled. • **> 0.0** Default: 30.0
setMode(…)	Control type A servo motion can be executed under impedance control. **Note:** Further information on the programming of robots under impedance control is contained in the operating and programming instructions for the System Software.

9.3.2 Parameters of the ISmartServoLINRuntime interface

Method	Description
changeControlMode Settings(…)	Modification of controller parameters during impedance control. (>>> *5.10.3 "Changing controller parameters during runtime" Page 37*)
setDetailedOutput(…)	The amount of detail when reading debug information • **1**: Standard information • **2**: Information which extends beyond the standard • **3**: All available information Default: 1 (>>> *5.5.1.3 "Reading out and displaying debug information " Page 23*)
setGoalReachedEvent Handler(…)	Notification service that signals the end position has been reached (type: IServoOnGoalReachedEvent) (>>> *5.6.1 "Notification upon achievement of target" Page 30*)
setMinimumTrajectory ExecutionTime(…)	Minimum length of time from the start to the end position of the servo motion (= minimum synchronization time) (type: double, unit: s) • **0 … 0.1** Default: 0.01 **Note:** The seconds hand effect can be avoided by increasing the minimum length of time to reach the end point.
activateVelocityPlanning(…)	Activation of velocity planning once the set end position has been reached (type: Boolean) • **true**: Velocity planning is activated. • **false**: Velocity planning is deactivated.

Method	Description
	Default: **False**
setMaxTranslationVelocity(…)	Maximum translational velocity for each Cartesian degree of freedom (type: double[], unit: mm/s) • **> 0.0**
setMaxOrientationVelocity(…)	Maximum rotational velocity for each Cartesian degree of freedom (type: double[], unit: rad/s) • **> 0.0**
setMaxNullSpaceVelocity(…)	Maximum velocity for the redundancy degree of freedom (type: double, unit: rad/s) • **> 0.0**

9.3.3 Setting the end position on the ISmartServoLINRuntime interface

Method	Description
setDestination(…)	Setting of a new Cartesian destination (>>> *5.5.3.1 "Setting a Cartesian end position" Page 25*)
	Setting of a new Cartesian end position and the corresponding translational destination velocity (>>> *5.5.3.2 "Setting a Cartesian end position with a target velocity" Page 26*)

9.4 Step response of the systems – Comparison of SmartServo and DirectServo

The following diagrams show the step responses for the motion types SmartServo and DirectServo.

The robot's start position is as follows at the beginning of the recording:

- A1 = 0°
- A2 = 30°
- A3 = 0°
- A4 = 40°
- A5 = 0°
- A6 = 90°
- A7 = 0°

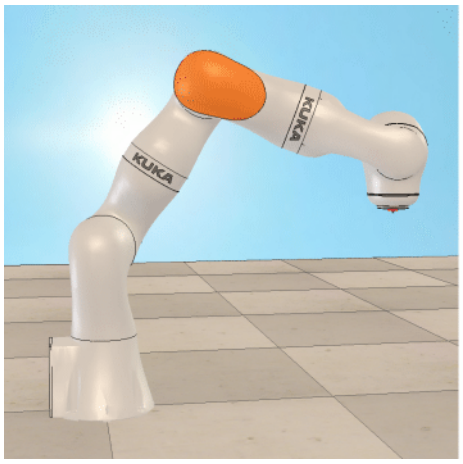


Fig. 9-1: Start position

9.4.1 Comparing the axis positions over time

The following diagrams show the commanded step and the measured actual position of the individual axis over time.

Key to the diagrams:

- Dark blue line (“tgt”): Commanded step
- Green line (“smart msr”): Path of the SmartServo motion using default motion parameters

  The acceleration values are rated very conservatively to ensure that the robot can still be operated safely in the worst case (extended position and load).
- Red line (“direct msr”): Path of the DirectServo motion using default motion parameters
- Light blue line (“smart OvAcc”): Path of the SmartServo motion at excessive acceleration

  The axis acceleration has been increased using the overrideJointAcceleration(…) method.

9.4.1.1 Step response of SmartServo/DirectServo (A1)

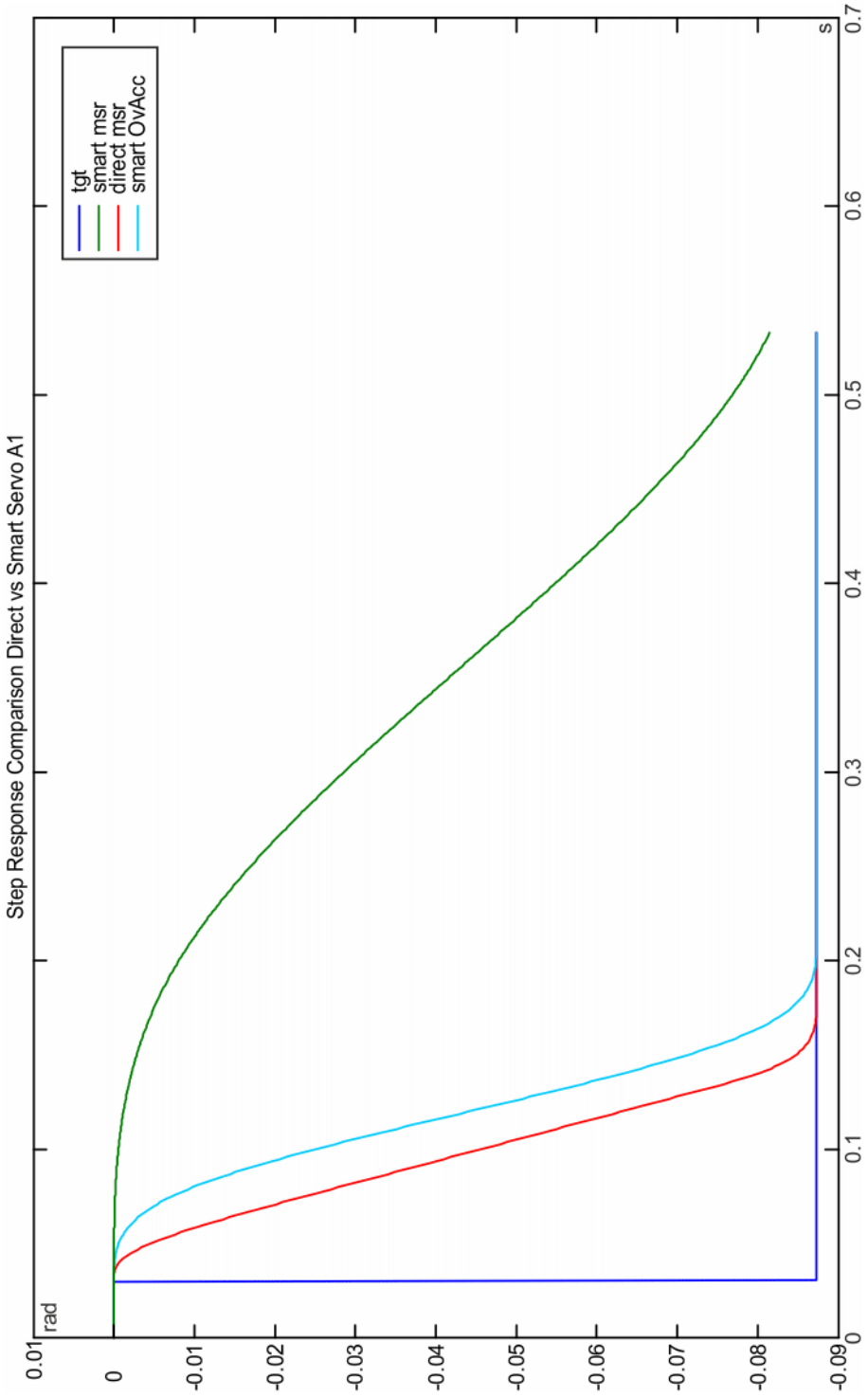


Fig. 9-2: Step response of SmartServo/DirectServo (A1)

9.4.1.2 Step response of SmartServo/DirectServo (A2)

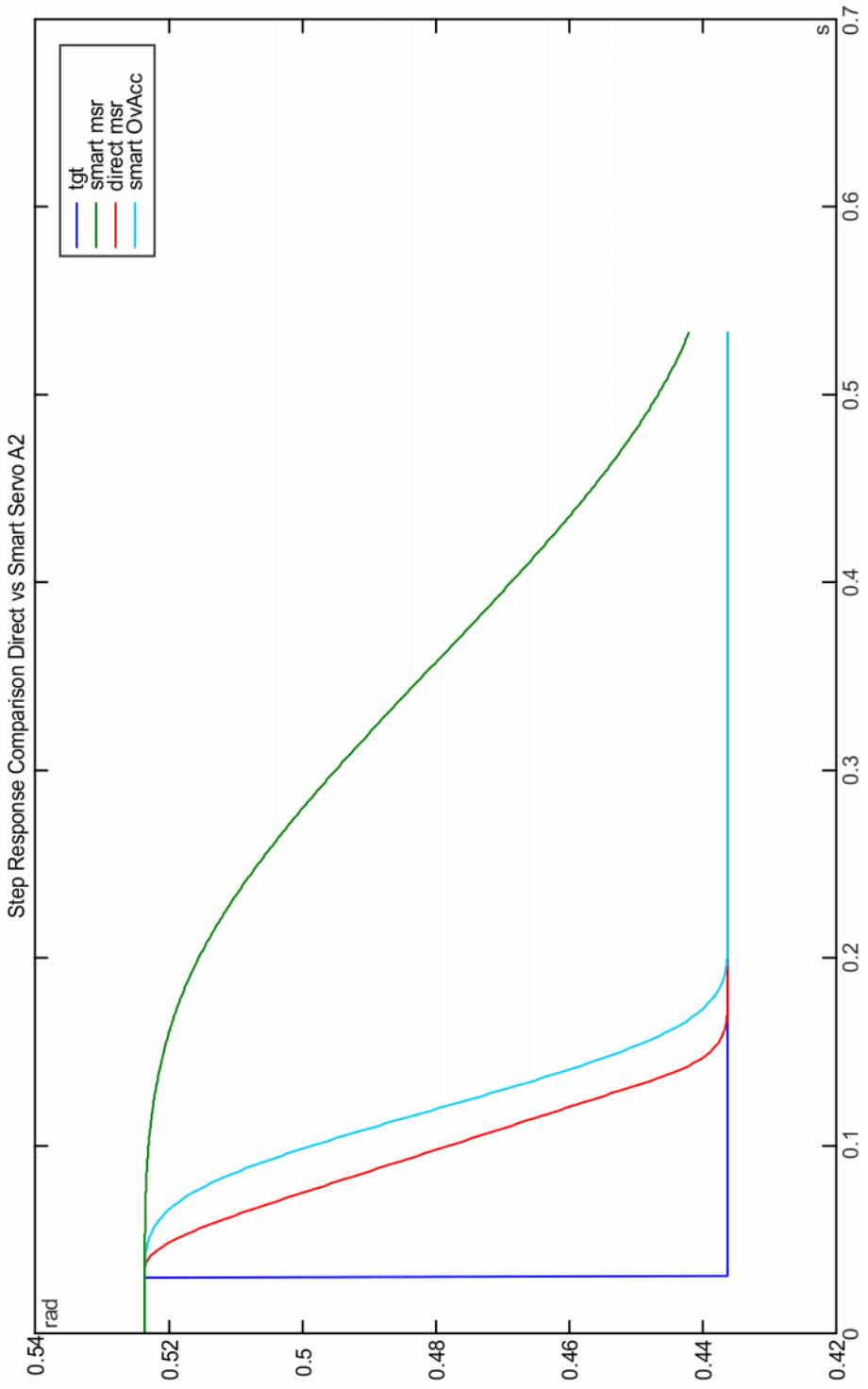


Fig. 9-3: Step response of SmartServo/DirectServo (A2)

9.4.1.3 Step response of SmartServo/DirectServo (A3)

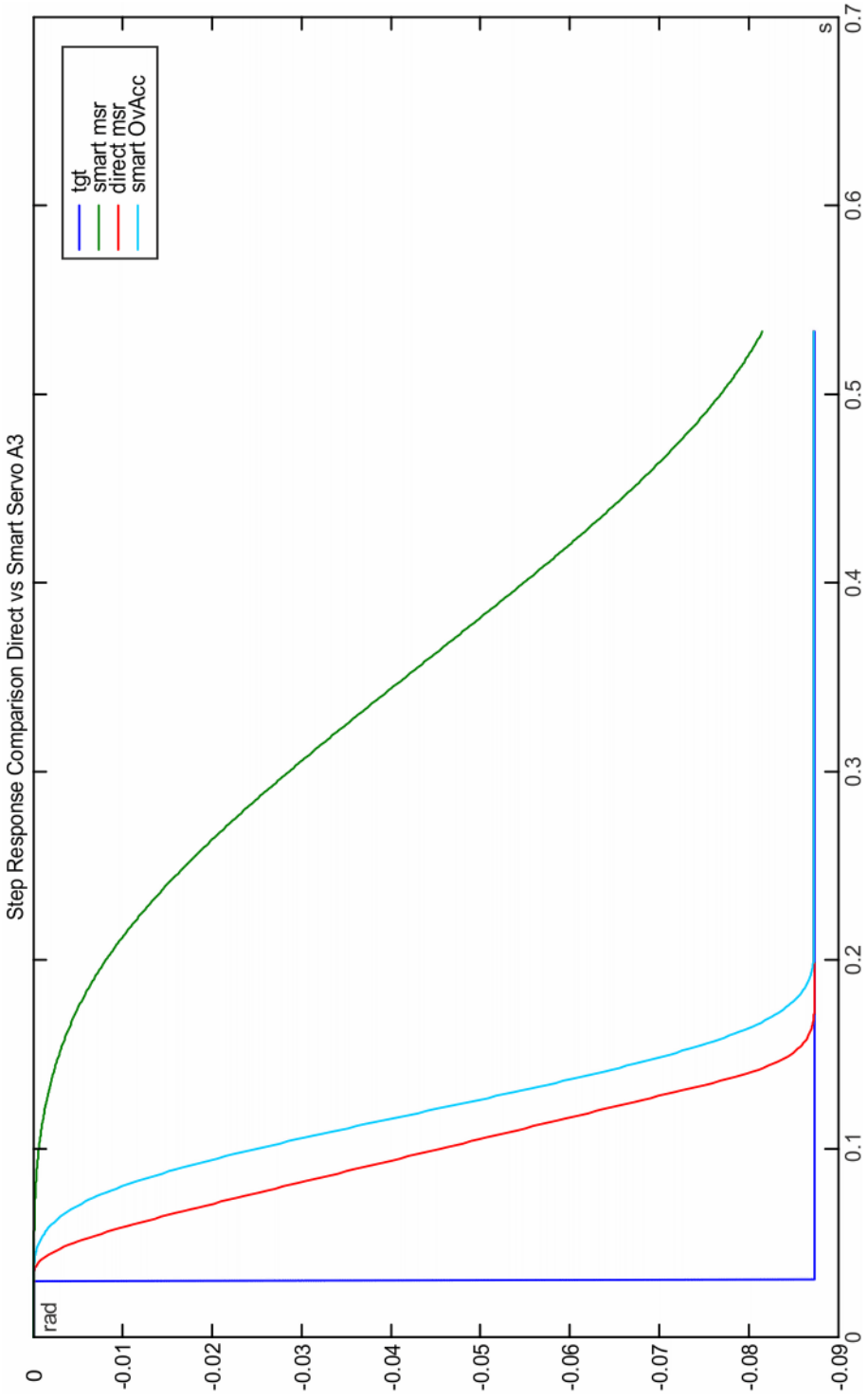


Fig. 9-4: Step response of SmartServo/DirectServo (A3)

9.4.1.4 Step response of SmartServo/DirectServo (A4)

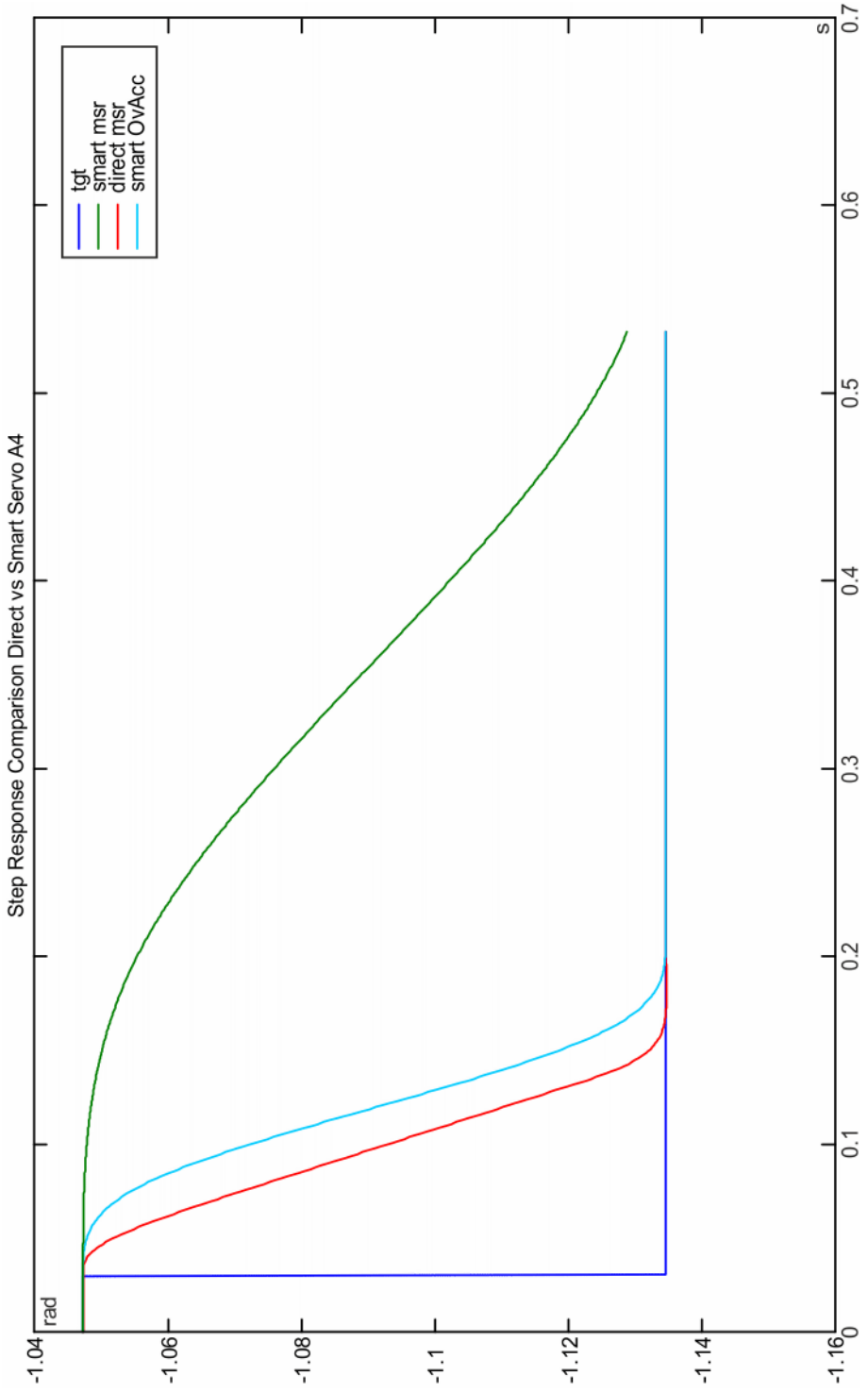


Fig. 9-5: Step response of SmartServo/DirectServo (A4)

9.4.1.5 Step response of SmartServo/DirectServo (A5)

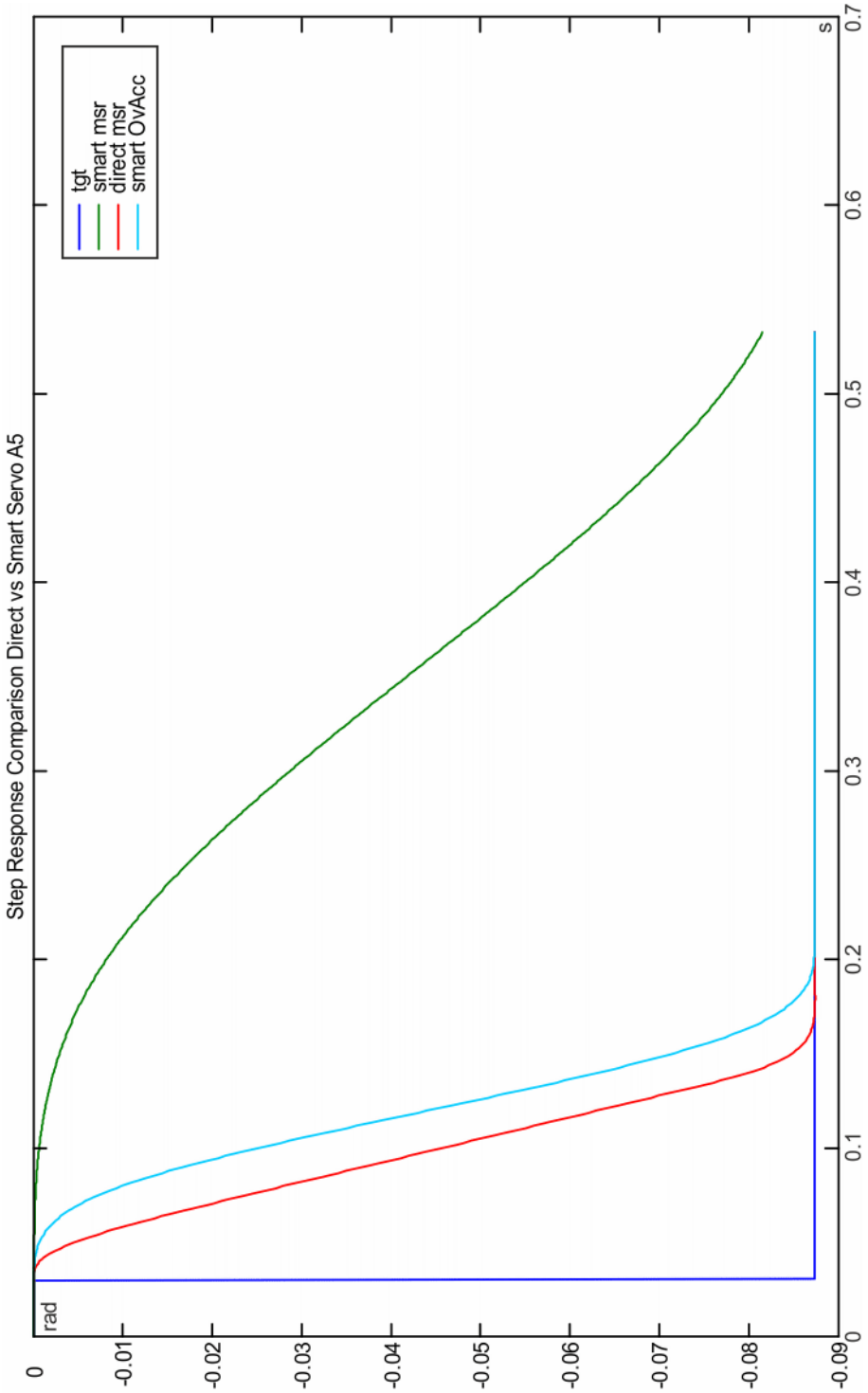


Fig. 9-6: Step response of SmartServo/DirectServo (A5)

9.4.1.6 Step response of SmartServo/DirectServo (A6)

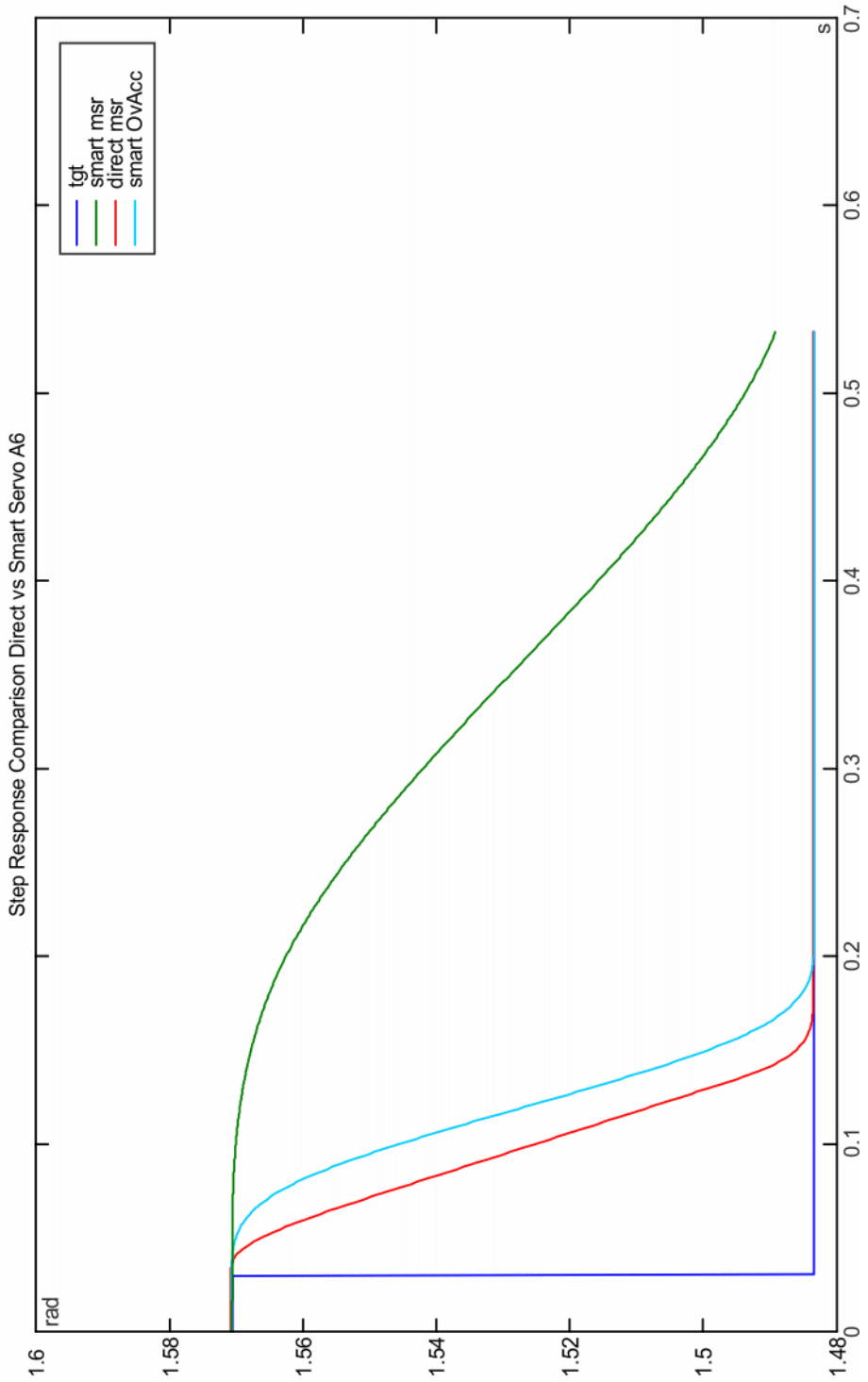


Fig. 9-7: Step response of SmartServo/DirectServo (A6)

9.4.1.7 Step response of SmartServo/DirectServo (A7)

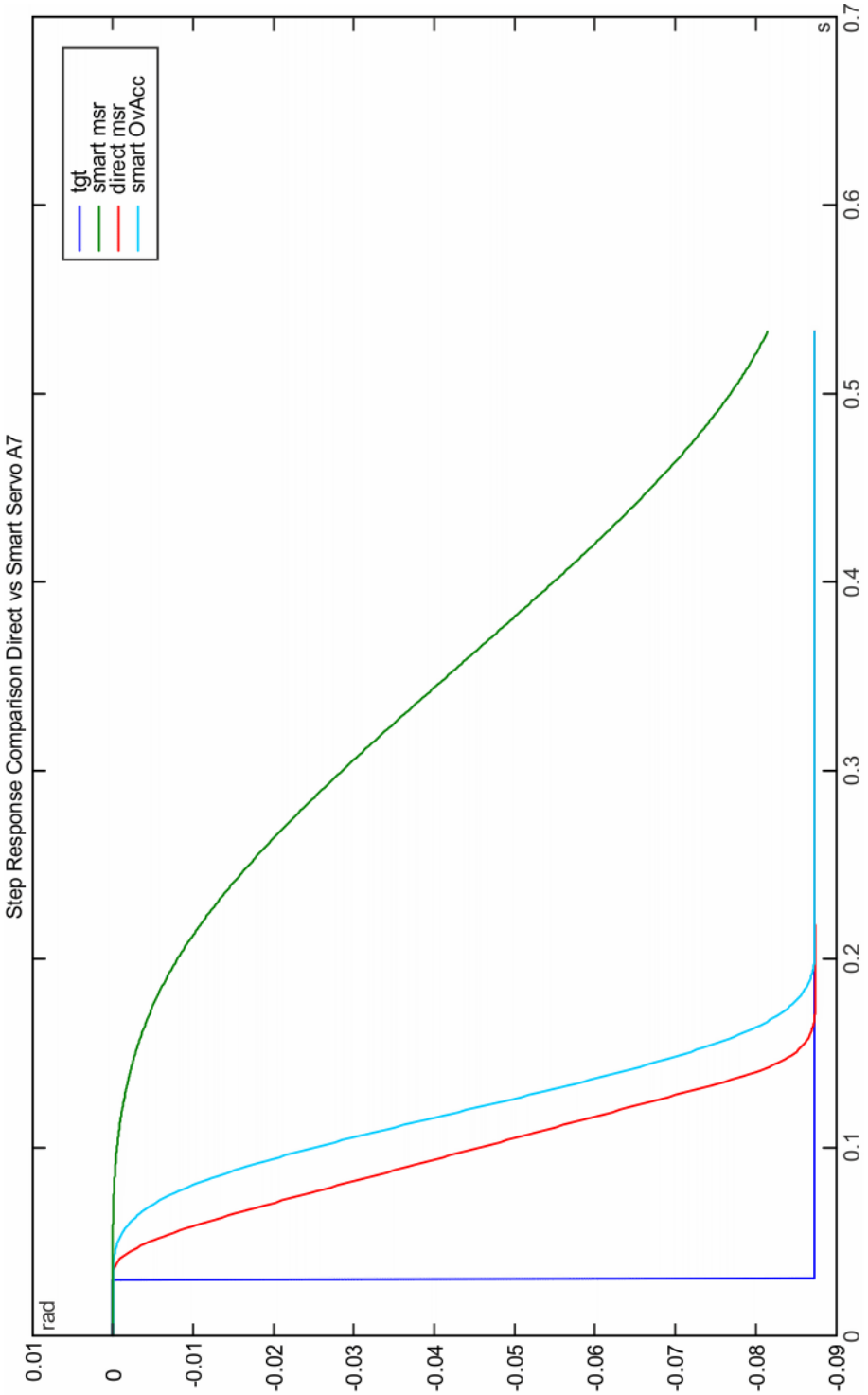


Fig. 9-8: Step response of SmartServo/DirectServo (A7)

9.4.2 Expanded comparison with velocity, acceleration and jerk

In addition to the measured actual positions, the following diagrams show the velocity, acceleration and jerk of the individual axes over time.

The following insights can be gained from these recordings:

- Both motion types offer the opportunity to fully exploit the robot performance.

- Smoother paths can be realized with the SmartServo motion type than with the DirectServo motion type.
- Path planning must be carried out in the application for the DirectServo motion type. The jerk is applied directly to the drives.
- The response time for the DirectServo motion type (12 ms) is somewhat faster than for the SmartServo motion type (22 ms).

Due to the polynomial solution, the SmartServo motion type tends to oscillate in unfavorable configurations due to extremely fast destination settings (< 10 ms) and exceeded regulator limit. If this occurs and the regulator reserves are exhausted by overrideJointAcceleration(…), the DirectServo motion type is recommended.

Key to the diagrams:

- Dark blue line (“smart msr”): Path of the SmartServo motion using default motion parameters

  The acceleration values are rated very conservatively to ensure that the robot can still be operated safely in the worst case (extended position and load).
- Green line (“direct msr”): Path of the DirectServo motion using default motion parameters
- Red line (“smart OvAcc”): Path of the SmartServo motion at excessive acceleration

  The axis acceleration has been increased using the overrideJointAcceleration(…) method.

9.4.2.1 Step response of SmartServo/DirectServo (A1, advanced)

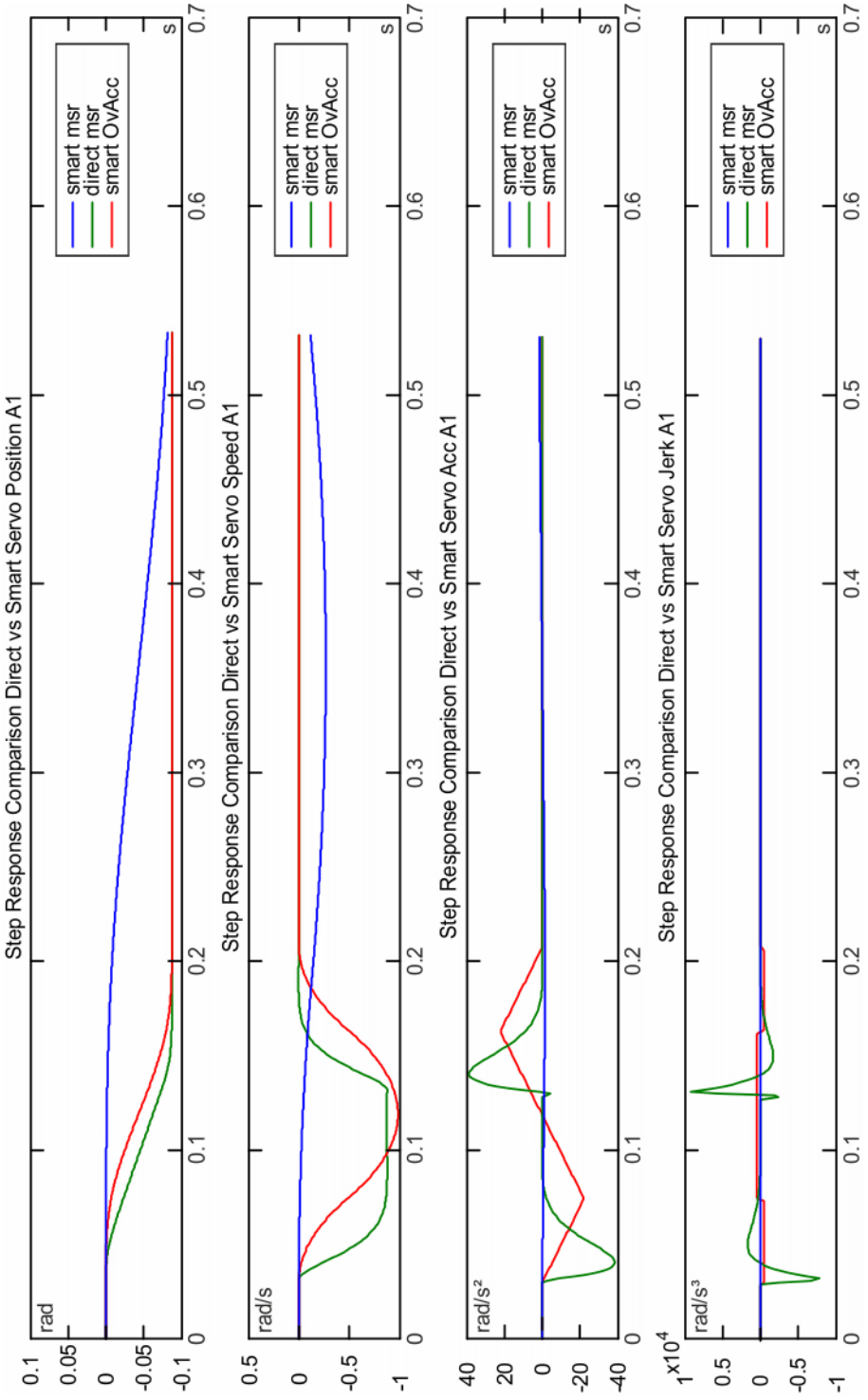


Fig. 9-9: Step response of SmartServo/DirectServo (A1, advanced)

9.4.2.2 Step response of SmartServo/DirectServo (A2, advanced)

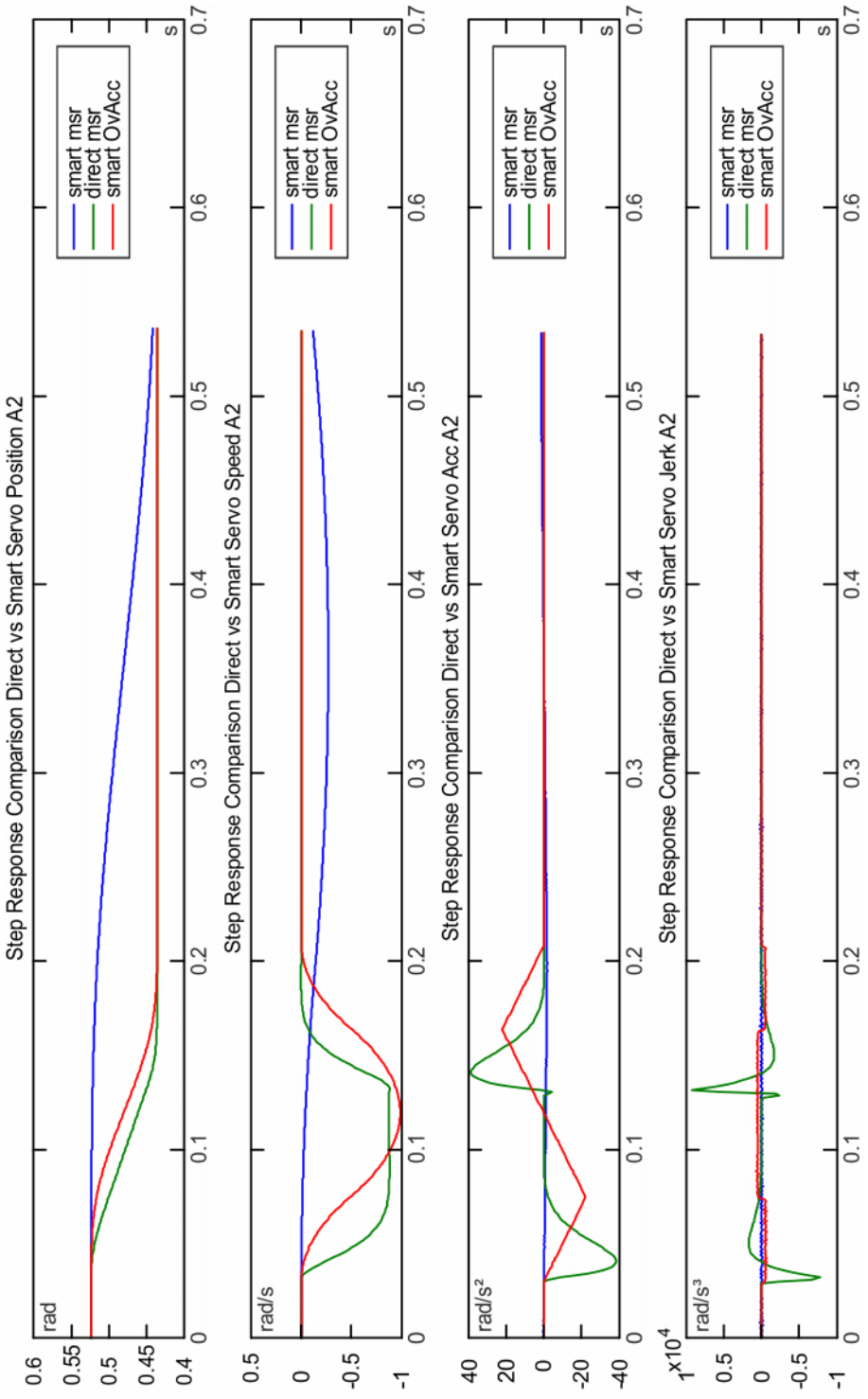


Fig. 9-10: Step response of SmartServo/DirectServo (A2, advanced)

9.4.2.3 Step response of SmartServo/DirectServo (A3, advanced)

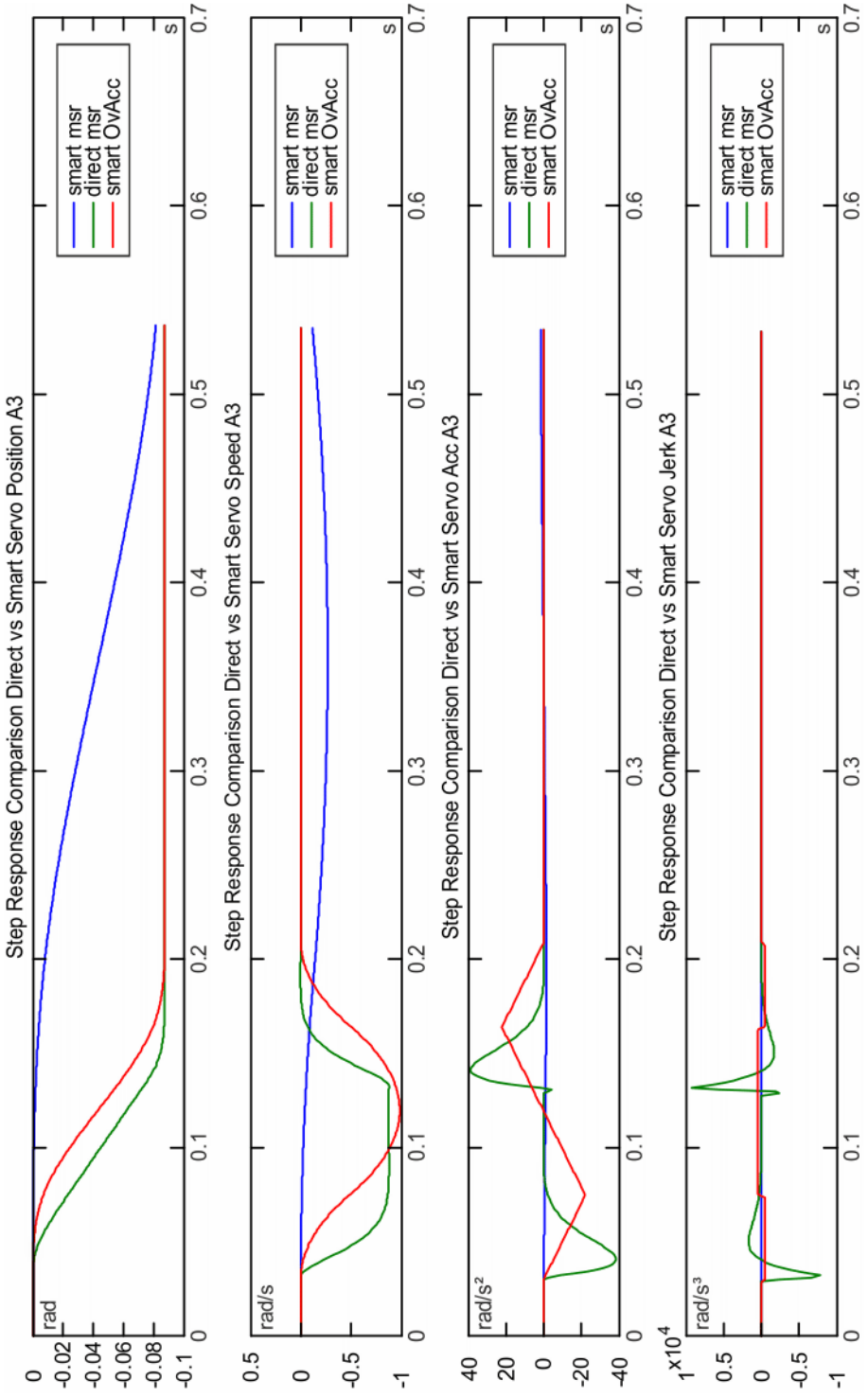


Fig. 9-11: Step response of SmartServo/DirectServo (A3, advanced)

9.4.2.4 Step response of SmartServo/DirectServo (A4, advanced)

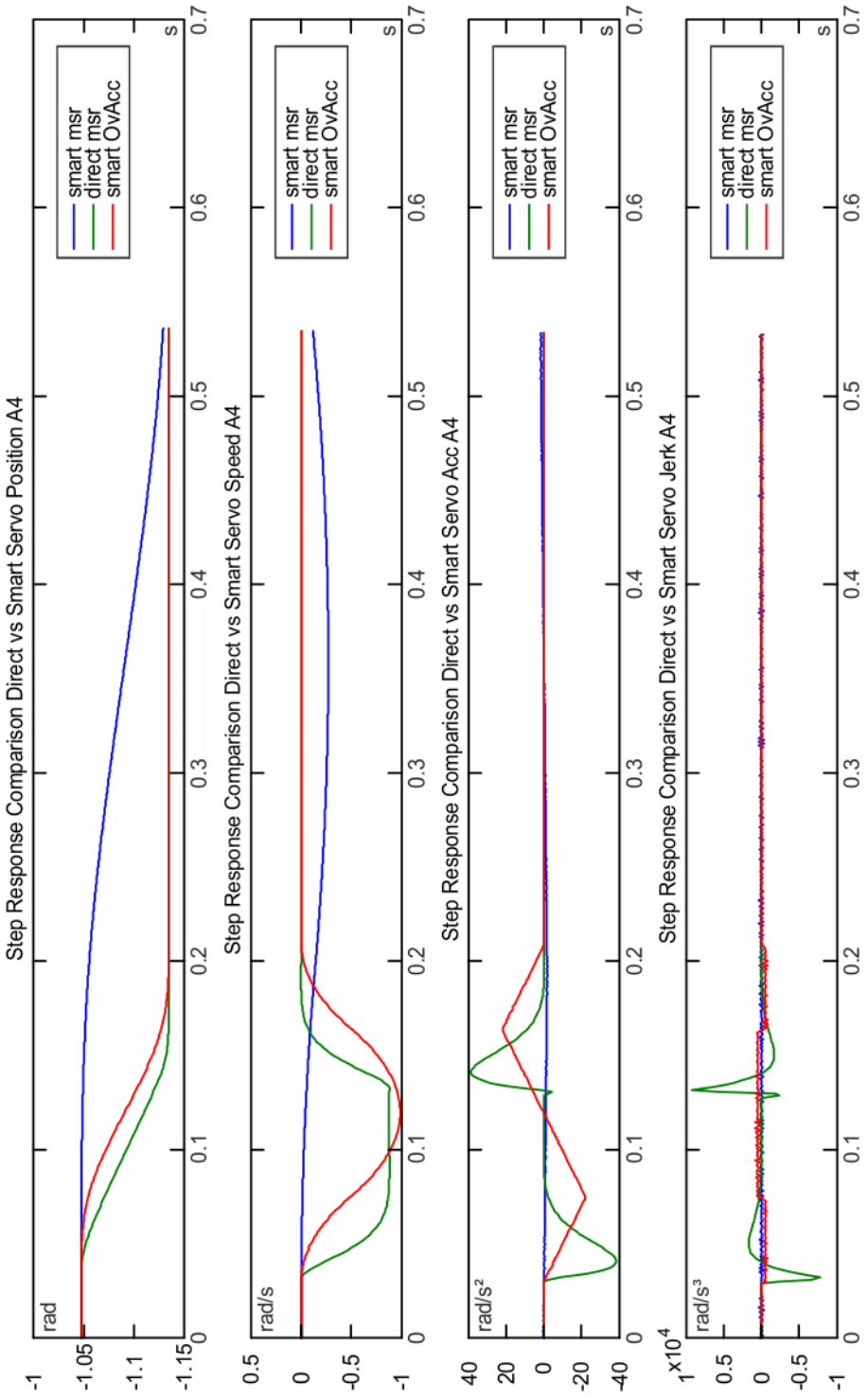


Fig. 9-12: Step response of SmartServo/DirectServo (A4, advanced)

9.4.2.5 Step response of SmartServo/DirectServo (A5, advanced)

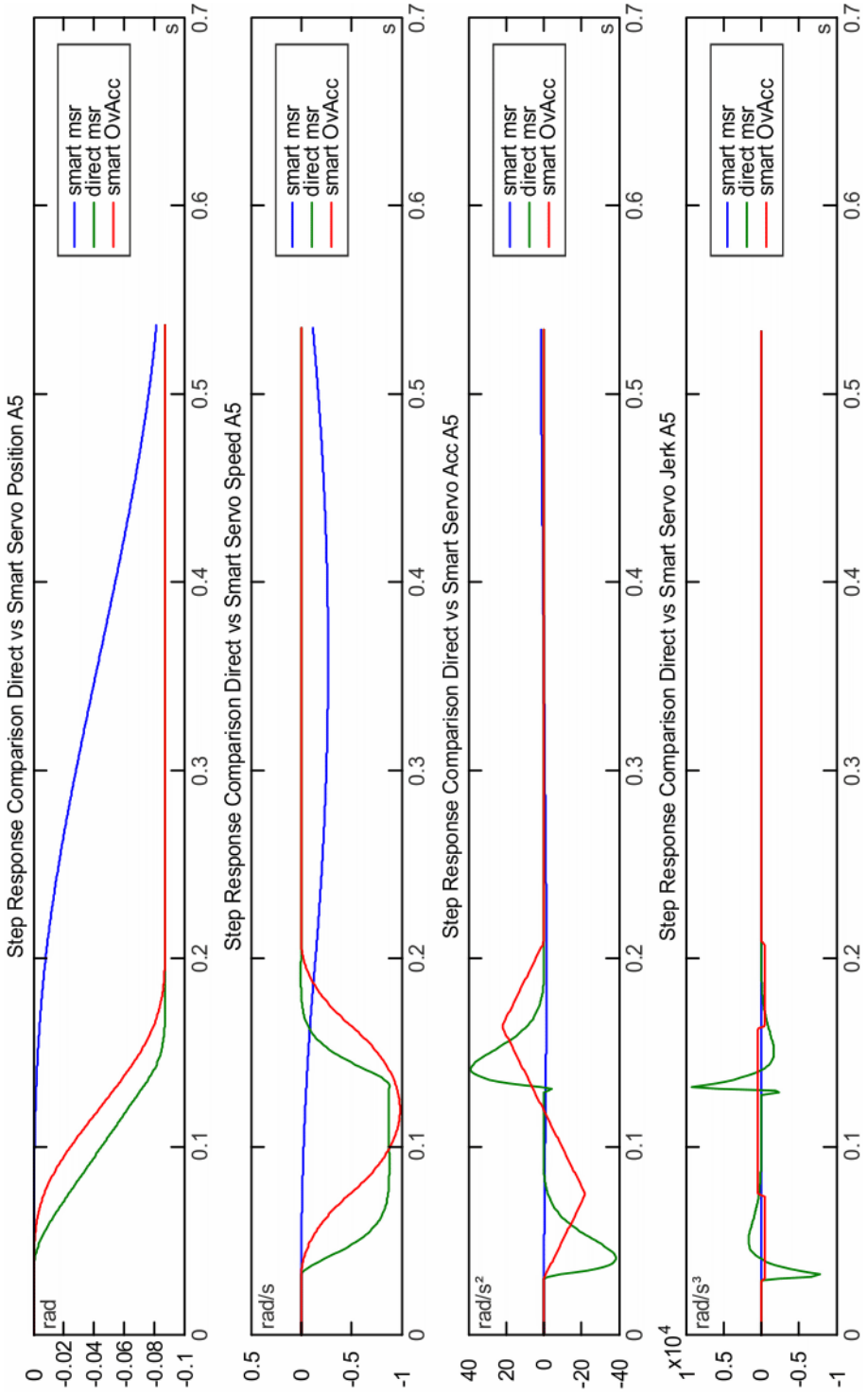


Fig. 9-13: Step response of SmartServo/DirectServo (A5, advanced)

9.4.2.6 Step response of SmartServo/DirectServo (A6, advanced)

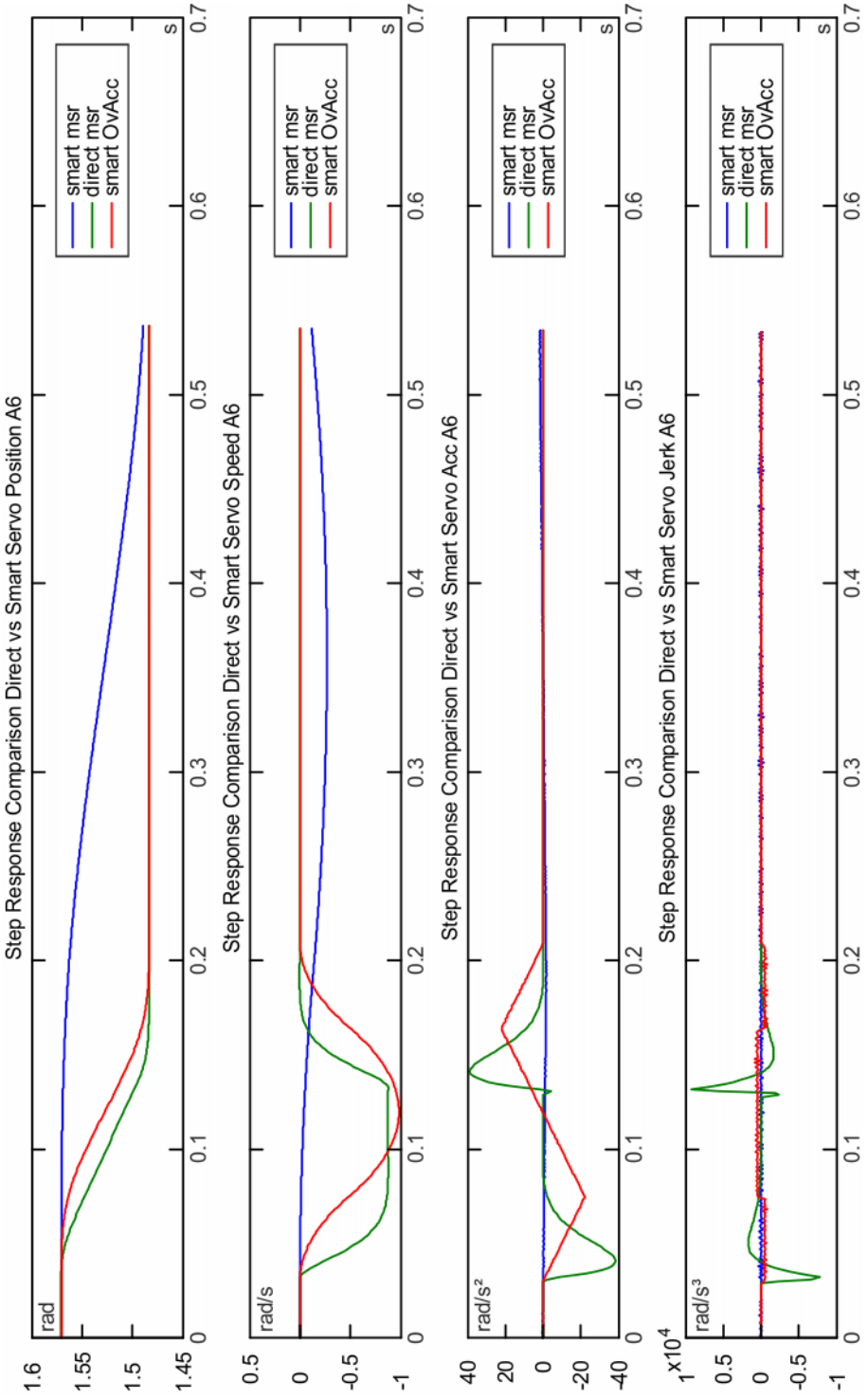


Fig. 9-14: Step response of SmartServo/DirectServo (A6, advanced)

9.4.2.7 Step response of SmartServo/DirectServo (A7, advanced)

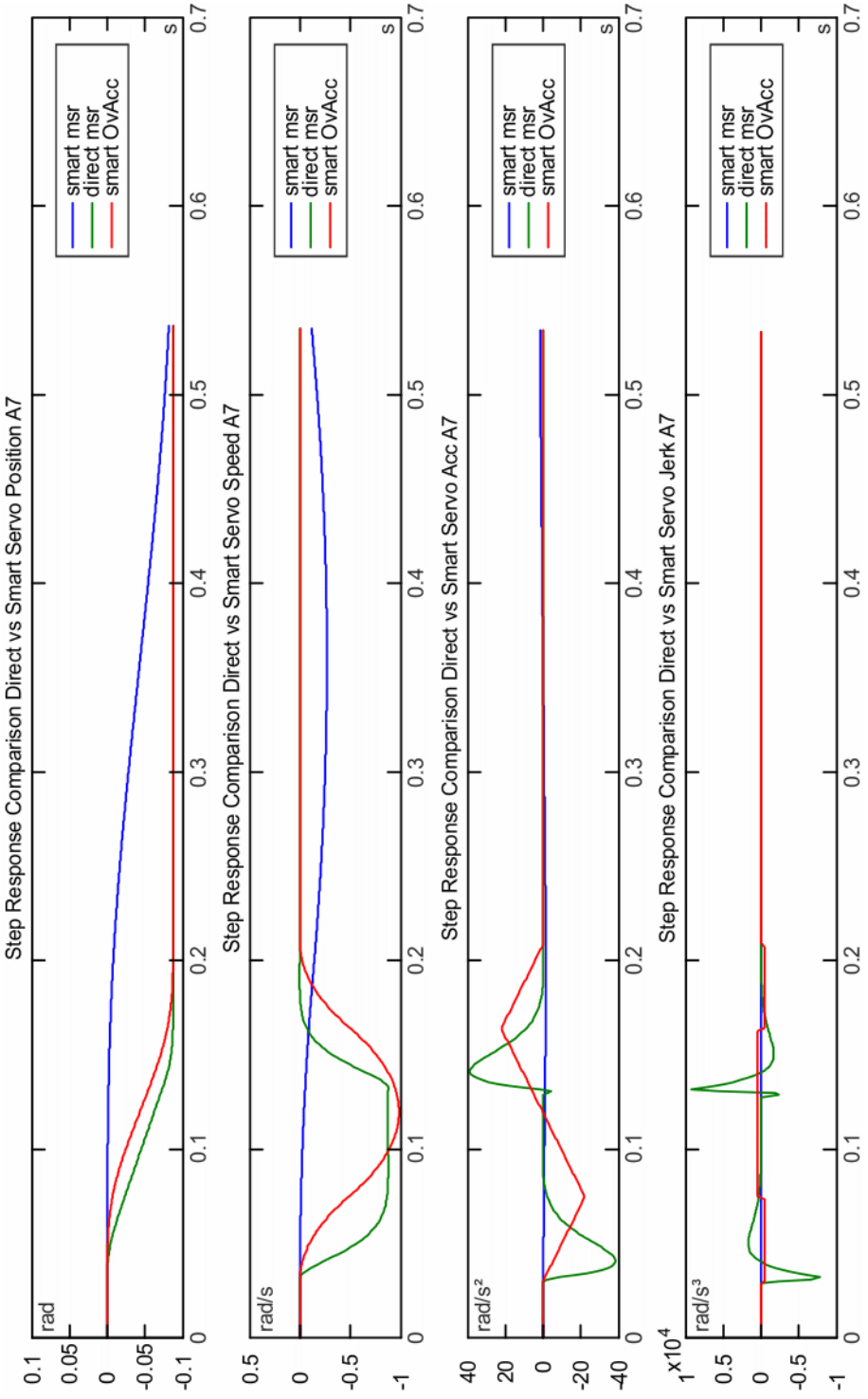


Fig. 9-15: Step response of SmartServo/DirectServo (A7, advanced)

10 KUKA Service

10.1 Requesting support

Introduction

This documentation provides information on operation and operator control, and provides assistance with troubleshooting. For further assistance, please contact your local KUKA subsidiary.

Information

The following information is required for processing a support request:

- Description of the problem, including information about the duration and frequency of the fault
- As comprehensive information as possible about the hardware and software components of the overall system

  The following list gives an indication of the information which is relevant in many cases:

  – Model and serial number of the kinematic system, e.g. the manipulator
  – Model and serial number of the controller
  – Model and serial number of the energy supply system
  – Designation and version of the system software
  – Designations and versions of other software components or modifications
  – Diagnostic package KRCDiag

    Additionally for KUKA Sunrise: Existing projects including applications

    For versions of KUKA System Software older than V8: Archive of the software (KRCDiag is not yet available here.)
  – Application used
  – External axes used

10.2 KUKA Customer Support

Availability

KUKA Customer Support is available in many countries. Please do not hesitate to contact us if you have any questions.

Argentina

Ruben Costantini S.A. (Agency)
Luis Angel Huergo 13 20
Parque Industrial
2400 San Francisco (CBA)
Argentina
Tel. +54 3564 421033
Fax +54 3564 428877
ventas@costantini-sa.com

Australia

KUKA Robotics Australia Pty Ltd
45 Fennell Street
Port Melbourne VIC 3207
Australia
Tel. +61 3 9939 9656
info@kuka-robotics.com.au
www.kuka-robotics.com.au

Belgium

KUKA Automatisering + Robots N.V.
Centrum Zuid 1031
3530 Houthalen
Belgium
Tel. +32 11 516160
Fax +32 11 526794
info@kuka.be
www.kuka.be

Brazil

KUKA Roboter do Brasil Ltda.
Travessa Claudio Armando, nº 171
Bloco 5 - Galpões 51/52
Bairro Assunção
CEP 09861-7630 São Bernardo do Campo - SP
Brazil
Tel. +55 11 4942-8299
Fax +55 11 2201-7883
info@kuka-roboter.com.br
www.kuka-roboter.com.br

Chile

Robotec S.A. (Agency)
Santiago de Chile
Chile
Tel. +56 2 331-5951
Fax +56 2 331-5952
robotec@robotec.cl
www.robotec.cl

China

KUKA Robotics China Co., Ltd.
No. 889 Kungang Road
Xiaokunshan Town
Songjiang District
201614 Shanghai
P. R. China
Tel. +86 21 5707 2688
Fax +86 21 5707 2603
info@kuka-robotics.cn
www.kuka-robotics.com

Germany

KUKA Deutschland GmbH
Zugspitzstr. 140
86165 Augsburg
Germany
Tel. +49 821 797-1926
Fax +49 821 797-41 1926
Hotline.robotics.de@kuka.com
www.kuka.com

France

KUKA Automatisme + Robotique SAS
Techvallée
6, Avenue du Parc
91140 Villebon S/Yvette
France
Tel. +33 1 6931660-0
Fax +33 1 6931660-1
commercial@kuka.fr
www.kuka.fr

India

KUKA India Pvt. Ltd.
Office Number-7, German Centre,
Level 12, Building No. - 9B
DLF Cyber City Phase III
122 002 Gurgaon
Haryana
India
Tel. +91 124 4635774
Fax +91 124 4635773
info@kuka.in
www.kuka.in

Italy

KUKA Roboter Italia S.p.A.
Via Pavia 9/a - int.6
10098 Rivoli (TO)
Italy
Tel. +39 011 959-5013
Fax +39 011 959-5141
kuka@kuka.it
www.kuka.it

Japan

KUKA Japan K.K.
YBP Technical Center
134 Godo-cho, Hodogaya-ku
Yokohama, Kanagawa
240 0005
Japan
Tel. +81 45 744 7531
Fax +81 45 744 7541
info@kuka.co.jp

Canada

KUKA Robotics Canada Ltd.
6710 Maritz Drive - Unit 4
Mississauga
L5W 0A1
Ontario
Canada
Tel. +1 905 670-8600
Fax +1 905 670-8604
info@kukarobotics.com
www.kuka-robotics.com/canada

Korea

KUKA Robotics Korea Co. Ltd.
RIT Center 306, Gyeonggi Technopark
1271-11 Sa 3-dong, Sangnok-gu
Ansan City, Gyeonggi Do
426-901
Korea
Tel. +82 31 501-1451
Fax +82 31 501-1461
info@kukakorea.com

Malaysia

KUKA Robot Automation (M) Sdn Bhd
South East Asia Regional Office
No. 7, Jalan TPP 6/6
Taman Perindustrian Puchong
47100 Puchong
Selangor
Malaysia
Tel. +60 (03) 8063-1792
Fax +60 (03) 8060-7386
info@kuka.com.my

Mexico

KUKA de México S. de R.L. de C.V.
Progreso #8
Col. Centro Industrial Puente de Vigas
Tlalnepantla de Baz
54020 Estado de México
Mexico
Tel. +52 55 5203-8407
Fax +52 55 5203-8148
info@kuka.com.mx
www.kuka-robotics.com/mexico

Norway

KUKA Sveiseanlegg + Roboter
Sentrumsvegen 5
2867 Hov
Norway
Tel. +47 61 18 91 30
Fax +47 61 18 62 00
info@kuka.no

Austria

KUKA Roboter CEE GmbH
Gruberstraße 2-4
4020 Linz
Austria
Tel. +43 7 32 78 47 52
Fax +43 7 32 79 38 80
office@kuka-roboter.at
www.kuka.at

Poland

KUKA Roboter CEE GmbH Poland
Spółka z ograniczoną odpowiedzialnością
Oddział w Polsce
Ul. Porcelanowa 10
40-246 Katowice
Poland
Tel. +48 327 30 32 13 or -14
Fax +48 327 30 32 26
ServicePL@kuka-roboter.de

Portugal

KUKA Robots IBÉRICA, S.A.
Rua do Alto da Guerra n° 50
Armazém 04
2910 011 Setúbal
Portugal
Tel. +351 265 729 780
Fax +351 265 729 782
info.portugal@kukapt.com
www.kuka.com

Russia

KUKA Russia OOO
1-y Nagatinskiy pr-d, 2
117105 Moskau
Russia
Tel. +7 495 665-6241
support.robotics.ru@kuka.com

Sweden

KUKA Svetsanläggningar + Robotar AB
A. Odhners gata 15
421 30 Västra Frölunda
Sweden
Tel. +46 31 7266-200
Fax +46 31 7266-201
info@kuka.se

Switzerland

KUKA Roboter CEE GmbH
Linz, Zweigniederlassung Schweiz
Heinrich Wehrli-Strasse 27
5033 Buchs
Switzerland
Tel. +41 62 837 43 20
info@kuka-roboter.ch

Slovakia

KUKA Roboter CEE GmbH
organizačná zložka
Bojnická 3
831 04 Bratislava
Slovakia
Tel. +420 226 212 273
support.robotics.cz@kuka.com

Spain

KUKA Iberia, S.A.U.
Pol. Industrial
Torrent de la Pastera
Carrer del Bages s/n
08800 Vilanova i la Geltrú (Barcelona)
Spain
Tel. +34 93 8142-353
comercial@kukarob.es

South Africa

Jendamark Automation LTD (Agency)
76a York Road
North End
6000 Port Elizabeth
South Africa
Tel. +27 41 391 4700
Fax +27 41 373 3869
www.jendamark.co.za

Taiwan

KUKA Automation Taiwan Co. Ltd.
1F, No. 298 Yangguang ST.,
Nei Hu Dist., Taipei City, Taiwan 114
Taiwan
Tel. +886 2 8978 1188
Fax +886 2 8797 5118
info@kuka.com.tw

Thailand

KUKA (Thailand) Co. Ltd.
No 22/11-12 H-Cape Biz Sector Onnut
Sukhaphiban 2 road, Prawet
Bangkok 10250
Thailand
Tel. +66 (0) 90-940-8950
HelpdeskTH@kuka.com

Czech Republic

KUKA Roboter CEE GmbH
organizační složka
Pražská 239
25066 Zdiby
Czech Republic
Tel. +420 226 212 273
support.robotics.cz@kuka.com

Hungary

KUKA HUNGÁRIA Kft.
Fö út 140
2335 Taksony
Hungary
Tel. +36 24 501609
Fax +36 24 477031
info@kuka-robotics.hu

USA

KUKA Robotics Corporation
51870 Shelby Parkway
Shelby Township
48315-1787
Michigan
USA
Tel. +1 866 873-5852
Fax +1 866 329-5852
info@kukarobotics.com
www.kukarobotics.com

UK

KUKA Robotics UK Ltd
Great Western Street
Wednesbury West Midlands
WS10 7LL
UK
Tel. +44 121 505 9970
Fax +44 121 505 6589
service@kuka-robotics.co.uk
www.kuka-robotics.co.uk

Index

T

U

W