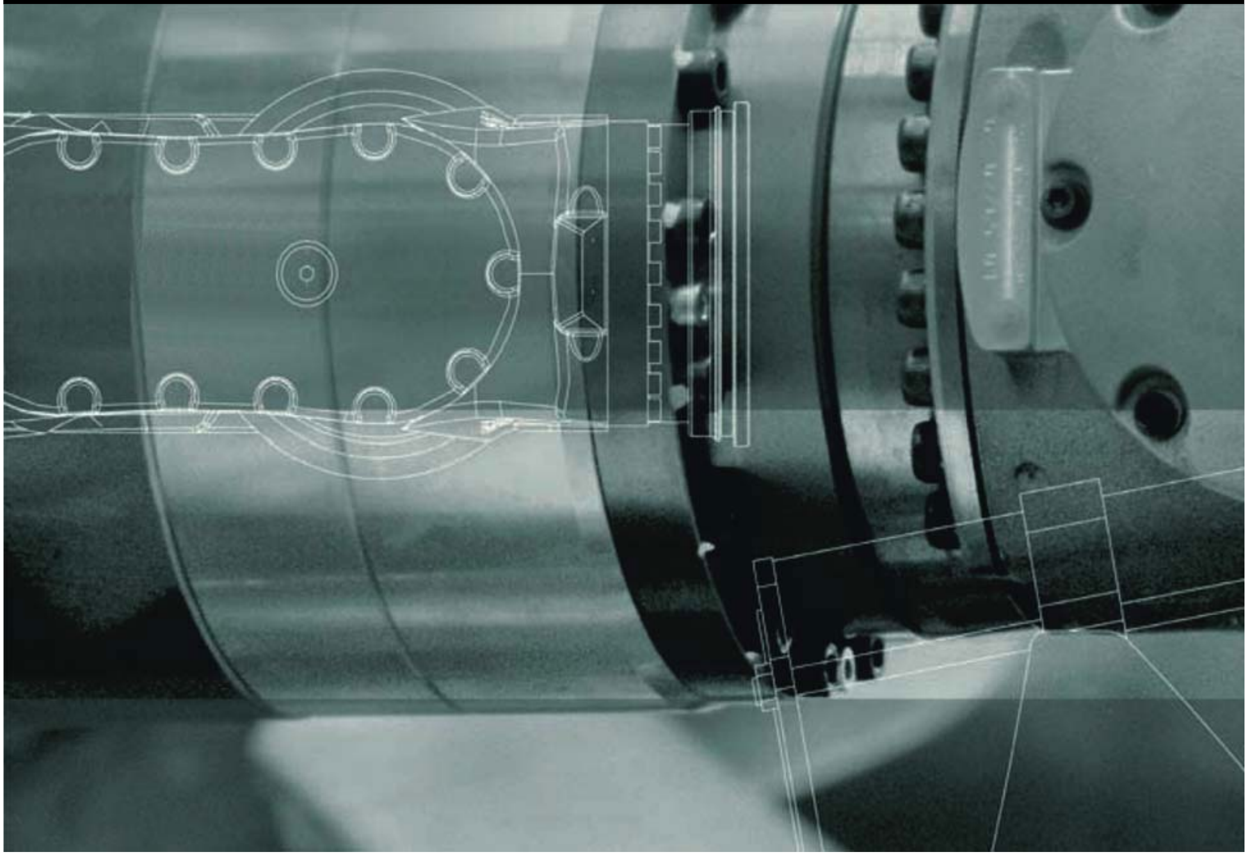
Software Option

KUKA Roboter GmbH

KUKA Sunrise.Connectivity Servoing 1.5 KUKA Sunrise.Connectivity Rendering 1.5

For KUKA Sunrise.Workbench 1.5

For KUKA Sunrise.OS 1.5



Issued: 30.01.2015

Version: KUKA Sunrise.Connectivity Servoing 1.5 Rendering 1.5 V2

Publication:	Pub KUKA Sunrise.Connectivity Servoing 1.5 Rendering 1.5 (PDF) en
Book structure:	KUKA Sunrise.Connectivity Servoing 1.5 Rendering 1.5 V1.1
Version:	KUKA Sunrise.Connectivity Servoing 1.5 Rendering 1.5 V2

Contents

1 Introduction 7

1.1 Industrial robot documentation 7
1.2 Representation of warnings and notes 7
1.3 Terms used 8
1.4 Trademarks 8

2 Purpose 9

2.1 Target group 9
2.2 Intended use 9

3 Product description 11

3.1 Overview of KUKA Sunrise.Connectivity 11
3.1.1 Comparison of motion characteristics (PTP, SmartServo, DirectServo) 11
3.1.2 DirectServo and SmartServo in the RoboticsAPI 12
3.2 KUKA Sunrise.Connectivity Rendering (option) 13

4 Safety 15

5 Installation 17

5.1 System requirements 17
5.2 Installing the SmartServo command library 17
5.3 Installing the SmartServo catalog with application examples 17
5.4 Installing the DirectServo command library 17
5.5 Installing the DirectServo catalog with application examples 17
5.6 Installing the Rendering interface 18
5.6.1 Integrating V-REP PRO into the Rendering interface 18

6 Operating the Rendering interface 21

6.1 “Rendering VREP” view 21
6.2 Configuring the Rendering interface for V-REP PRO 22
6.3 Starting live view 23
6.4 Synchronizing frames 23

7 Programming 25

7.1 Servo motion in an application context 25
7.2 Simple destination setting with setDestination(…) 26
7.3 Cyclical destination setting with setDestination(…) 26
7.4 Seconds hand effect 27
7.5 IServoRuntime class 28
7.5.1 Handling the real-time system 28
7.5.1.1 Polling time stamps 28
7.5.1.2 Polling the state of the precision interpolator 29
7.5.1.3 Reading out and displaying debug information 29
7.5.2 Axis-specific interface 30
7.5.2.1 Specifying an axis-specific end position 30
7.5.2.2 Specifying an axis-specific end position with a target velocity 30
7.5.2.3 Reading out the axis-specific actual position 30
7.5.3 Cartesian interface 31
7.5.3.1 Setting a Cartesian destination in the robot base coordinate system 31

7.5.3.2 Specifying a Cartesian destination in the reference coordinate system 31
7.5.3.3 Reading out a Cartesian actual position in the robot base coordinate system 32
7.5.3.4 Reading out the Cartesian actual position in the reference coordinate system 33
7.5.4 Terminating a servo motion 33
7.6 Destination reached and timeouts 34
7.7 Receiving the GoalReached event (SmartServo only) 34
7.8 Changing the path parameters 35
7.9 Time recording 36
7.9.1 Measuring the application cycle time 36
7.10 Impedance control 37
7.10.1 Validating the load model for impedance control 37
7.10.2 Activating impedance control 38
7.10.3 Changing controller parameters during the runtime 38
7.10.4 Transition between control types 39
7.11 Feedback effect for servo motions 40
7.12 Step response of the systems 42
7.12.1 SmartServo vs. DirectServo – comparison of axis positions over time 43
7.12.1.1 Step response comparison of SmartServo vs. DirectServo, A1 44
7.12.1.2 Step response comparison of SmartServo vs. DirectServo, A2 45
7.12.1.3 Step response comparison of SmartServo vs. DirectServo, A3 46
7.12.1.4 Step response comparison of SmartServo vs. DirectServo, A4 47
7.12.1.5 Step response comparison of SmartServo vs. DirectServo, A5 48
7.12.1.6 Step response comparison of SmartServo vs. DirectServo, A6 49
7.12.1.7 Step response comparison of SmartServo vs. DirectServo, A7 50
7.12.2 SmartServo vs. DirectServo – comparison of velocity, acceleration, jerk 50
7.12.2.1 Step response comparison of SmartServo vs. DirectServo, A1 (advanced) ... 52
7.12.2.2 Step response comparison of SmartServo vs. DirectServo, A2 (advanced) ... 53
7.12.2.3 Step response comparison of SmartServo vs. DirectServo, A3 (advanced) ... 54
7.12.2.4 Step response comparison of SmartServo vs. DirectServo, A4, advanced 55
7.12.2.5 Step response comparison of SmartServo vs. DirectServo, A5 (advanced) ... 56
7.12.2.6 Step response comparison of SmartServo vs. DirectServo, A6 (advanced) ... 57
7.12.2.7 Step response comparison of SmartServo vs. DirectServo, A7 (advanced) .. 58

8 Application examples 59

8.1 “Simple Tutorials for SmartServo” catalog 59
8.1.1 SmartServoSampleSimpleJointMotion application 59
8.1.2 SmartServoSampleSimpleCartesian application 59
8.1.3 SmartServoSampleInteractionControl application 59
8.2 “Simple Tutorials for DirectServo” catalog 60
8.2.1 DirectServoSampleSimpleJointMotion application 60
8.2.2 DirectServoSampleSimpleCartesian application 60
8.2.3 DirectServoSampleInteractionControl application 60

9 Command reference 61

9.1 SmartServo command reference 61
9.1.1 Motion parameters 61
9.1.2 Runtime parameters 62
9.1.3 Setting a destination 62
9.2 DirectServo command reference 63
9.2.1 Motion parameters 63

9.2.2 Runtime parameters 63
9.2.3 Setting a destination 64

10 Exception handling routines 65

10.1 Example of an exception handling routine 65
10.2 Overview of exceptions – runtime error messages 65
10.3 CommandInvalidException error causes 66

11 Troubleshooting 69

11.1 TCP/IP traffic from Windows is too slow 69
11.2 Robot makes jerky motions/whistles – avoiding the seconds hand effect 70
11.3 Impedance control cannot be activated 71
11.4 Feedback effect 71

12 KUKA Service 73

12.1 Requesting support 73
12.2 KUKA Customer Support 73

Index **81**

1 Introduction

1.1 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.

1.2 Representation of warnings and notes

Safety

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

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

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

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

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

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

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

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

Hints

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

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

1.3 Terms used

Term	Description
API	Application Programming Interface Interface for programming applications.
Exception	Exception or exceptional situation An exception describes a procedure for forwarding information about certain program statuses, mainly error states, to other program levels for further processing.
KUKA Sunrise Cabinet	Control hardware for operating the LBR iiwa industrial robot
KUKA Sunrise.OS	KUKA Sunrise.Operating System System software for industrial robots which are operated with the robot controller KUKA Sunrise Cabinet
KUKA Sunrise. Workbench	Development environment for the start-up of a robot cell (station) and for the development of robot applications
KUKA LBR iiwa	KUKA lightweight robot intelligent industrial work assistant
HRC	Human-robot cooperation
System reaction time	Time between command specification and effective robot motion
V-REP PRO	Virtual Robot Experimentation Platform Simulation software for testing the behavior of robots in a virtual environment

1.4 Trademarks

V-REP PRO is a trademark of COPPELIA ROBOTICS MARC FREESE.

2 Purpose

2.1 Target group

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

- Advanced system knowledge of robotics
- Advanced Java programming skills
- Advanced programming knowledge (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.

2.2 Intended use

Use

KUKA Sunrise.Connectivity is designed exclusively for use in the laboratory and has not been released for direct interaction with humans (HRC).

KUKA Sunrise.Connectivity may be used exclusively in accordance with the specified system requirements for the purpose of investigating new application fields, applications and algorithms for the LBR iiwa.

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. KUKA Roboter GmbH is not liable for any damage resulting from such misuse. The risk lies entirely with the user.

Examples of such misuse include:

- Use of KUKA Sunrise.Connectivity not in accordance with the specified system requirements
- Use in an industrial environment
- Use in medical applications
- Allowing humans in the robot’s workspace.

3 Product description

3.1 Overview of KUKA Sunrise.Connectivity

KUKA Sunrise.Connectivity is a software option for the implementation of soft real-time applications and provides the following interfaces for this purpose:

Interfaces

- KUKA Sunrise.Connectivity SmartServo
- KUKA Sunrise.Connectivity DirectServo
- KUKA Sunrise.Connectivity Rendering (option)

SmartServo, DirectServo

The SmartServo and DirectServo interfaces are realized through motion classes of the same name which each provide their own PTP motion type.

The SmartServo and DirectServo motion classes are installed subsequently via the Sunrise catalog. This adds the corresponding class libraries to the RoboticsAPI.

The SmartServo and DirectServo motion types can be used to set new end points for the runtime of a motion.

SmartServo and DirectServo, servo motions for short, 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 last commanded point is directly addressed.

- The motion type SmartServo enables jerk-limited paths with fast destination setting.
- The DirectServo motion type 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.

Rendering

(>>> 3.2 "KUKA Sunrise.Connectivity Rendering (option)" Page 13)

3.1.1 Comparison of motion characteristics (PTP, SmartServo, 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 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	Continuous path motion, collision-free path planning	Visual servoing, camera-based collision avoidance	Haptics
Destination interval (recommended)	Long (>100 ms)	Medium (>20 ms)	Very short (2 …19 ms)
Quantity of control points	Low	High	Very high

	PTP	SmartServo	DirectServo
Destination	Any point in the work envelope	Any point in the work envelope	At a maximum distance of ±5° from the axis-specific actual position of the robot
Path characteristics	■ Jerk-limited ■ Can be approximated	■ Jerk-limited ■ Cannot be approximated	■ Not jerk-limited ■ Cannot be approximated
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 during destination specification while an end point is being addressed	Every point is addressed.	Current end point is discarded when a new destination is specified.	
Configurable path parameters	■ Velocity ■ Acceleration	■ Velocity ■ Acceleration ■ Jerk	■ Velocity
Planning	■ Polynomial	■ 3rd-order polynomial	■ Digital filter ■ Planning in application
System reaction time		~22 ms	~12 ms

3.1.2 DirectServo and SmartServo in the RoboticsAPI

SmartServo and DirectServo are motion classes used by the robot application to start motions. In turn, these provide special runtime classes with which destinations can be changed during a motion.

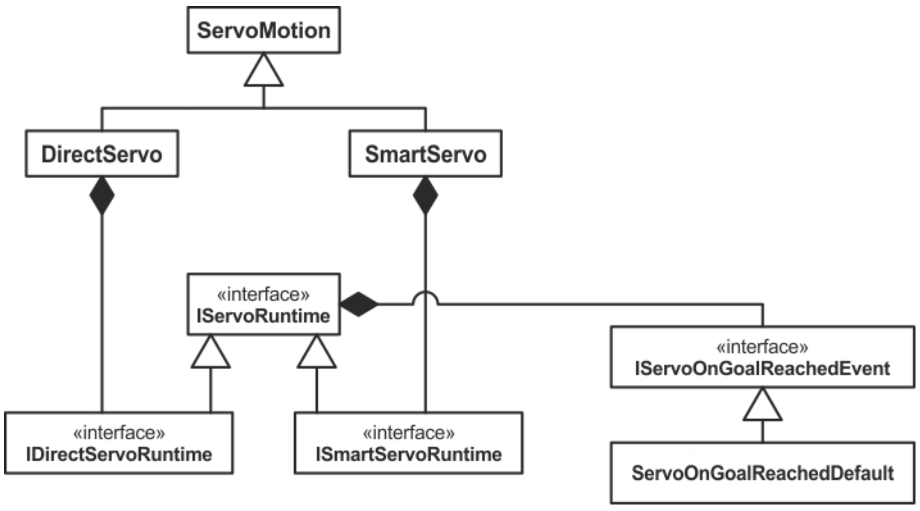

ServoMotion
DirectServo
SmartServo
«interface» IServoRuntime
«interface» IServoOnGoalReachedEvent
«interface» IDirectServoRuntime
«interface» ISmartServoRuntime
ServoOnGoalReachedDefault

Fig. 3-1: DirectServo and SmartServo in the RoboticsAPI

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

Each of the motion classes has its own interface:

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

These interfaces contain the specific characteristics of the precision interpolators for each of the motion classes.

3.2 KUKA Sunrise.Connectivity Rendering (option)

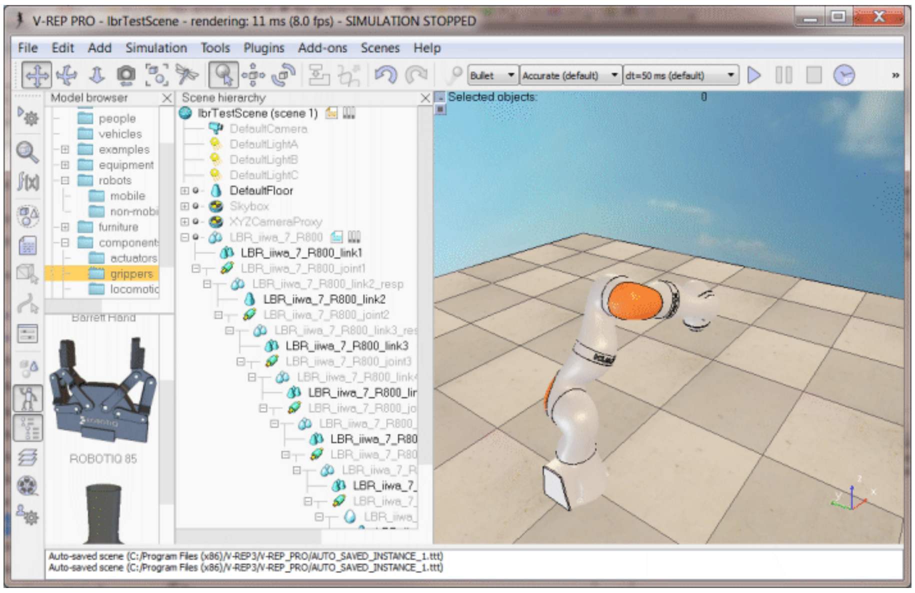
Overview

The Rendering interface allows 3D visualization tools to be connected to the KUKA Sunrise Cabinet robot controller.

The interface was realized as an example using the simulation software V-REP PRO and is supplied together with Java sources.

V-REP PRO can be obtained from http://www.coppeliarobotics.com/. The license terms governing the commercial use of the simulation software must be observed.

This documentation describes V-REP PRO only to the extent necessary for understanding the interface. Further information is contained in the simulation software operating instructions.



Auto-saved scene (C:/Program Files (x86)/V-REP3/V-REP_PRO/AUTO_SAVED_INSTANCE_1.ttt)
Auto-saved scene (C:/Program Files (x86)/V-REP3/V-REP_PRO/AUTO_SAVED_INSTANCE_1.ttt)

Fig. 3-2: Scene with LBR iiwa in V-REP PRO (example)

Functions

The following functionalities are available in the Rendering interface:

- Connection between V-REP PRO and the Sunrise project on the robot controller
- Configuration and starting of the visualization in V-REP PRO
- Activation of a live view
- Synchronization and visualization of the frame tree in V-REP PRO

Live view

The runtime data on the robot controller are available through the Rendering interface in Sunrise.Workbench and are displayed in V-REP PRO.

4 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 operating instructions or assembly instructions and in the operating and programming instructions must be observed. Death to persons, severe injuries or considerable damage to property may otherwise result.

⚠ The operator is responsible for performing his own risk and hazard analysis when using the robot under laboratory conditions. This applies in particular if humans must enter the robot’s work envelope. 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 SmartServo and DirectServo interfaces enable the implementation of closed control loops in the context of applications. 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.

⚠ The SmartServo and DirectServo motion types 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.

⚠ WARNING In the SmartServo and DirectServo motion types, the path planning is carried out by the application. Due to their fast destination setting changes, the actual system response will differ from the response specified by the manufacturer. The stopping distances and times specified in the operating and assembly instructions of the LBR iiwa are not valid. The user is responsible for determining the application-specific stopping distances and times.

5 Installation

5.1 System requirements

Hardware

- KUKA LBR iiwa
- KUKA Sunrise Cabinet

Software

- KUKA Sunrise.Workbench 1.5
- KUKA Sunrise.OS 1.5

5.2 Installing the SmartServo command library

Procedure

1. Open the station configuration of the Sunrise project and select the **Software** tab.
2. Select the **Smart Servo Motion Extension** software package for installation.
   - Set the check mark in the **Install** column.
3. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.

   The command library of the SmartServo motion class is inserted in the Sunrise project and is available for robot programming.
4. Install the system software on the robot controller (always required if the station configuration has been changed).

5.3 Installing the SmartServo catalog with application examples

Procedure

1. Open the station configuration of the Sunrise project and select the **Software** tab.
2. Select the **Simple Tutorials for SmartServo** catalog for installation.
   - Set the check mark in the **Install** column.
3. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.

   The **com.kuka.roboticsAPI.smartServo.samples** package is created and inserted in the **Samples** folder of the Sunrise project.

5.4 Installing the DirectServo command library

Procedure

1. Open the station configuration of the Sunrise project and select the **Software** tab.
2. Select the **Direct Servo Motion Extension** software package for installation.
   - Set the check mark in the **Install** column.
3. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.

   The command library of the DirectServo motion class is inserted in the Sunrise project and is available for robot programming.
4. Install the system software on the robot controller (always required if the station configuration has been changed).

5.5 Installing the DirectServo catalog with application examples

Procedure

1. Open the station configuration of the Sunrise project and select the **Software** tab.

2. Select the **Simple Tutorials for DirectServo** catalog for installation.
   - Set the check mark in the **Install** column.
3. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.

   The **com.kuka.roboticsAPI.directServo.samples** package is created and inserted in the **Samples** folder of the Sunrise project.

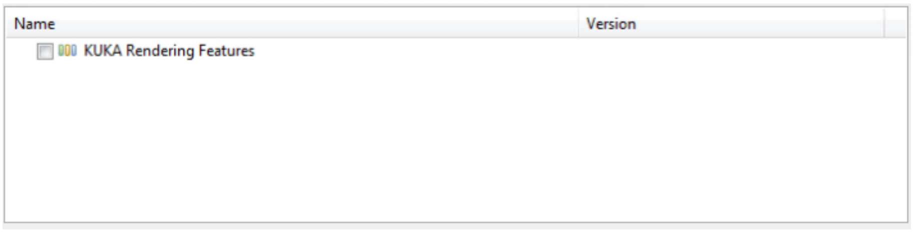
5.6 Installing the Rendering interface

Precondition

- V-REP PRO 3.1.3 or higher is installed.
- The ZIP archive **com.kuka.rendering.repository-release** is available.

Procedure

1. Select the menu sequence **Help** > **Install new software...**. The **Install** window is opened.
2. Click on the **Add ...** button. The **Add Repository** window is opened.
3. Click on the **Archive…** button. The **Repository archive** window is opened.
4. Navigate to the directory in which the ZIP archive **com.kuka.rendring.repository-release** is saved.
5. Select the ZIP archive and click on **Open**. The path to the archive is displayed in the **Add Repository** window.
6. Confirm the path with **OK**. The entry **KUKA Rendering Features** is displayed in the **Install** window.



Name	Version
☐ KUKA Rendering Features	

7. Set the check mark at **KUKA Rendering Features** and click on **Next >** klicken.
8. The component selected for installation is displayed. Confirm with **Next >**.
9. Accept the conditions of the license agreement and click on **Finish**. The installation is started.
10. If a security warning is displayed, confirm it with **OK**.
11. Sunrise.Workbench must be rebooted in order to complete the installation. This is indicated in a notification message. Click on the **Restart now** button.
12. Sunrise.Workbench is rebooted and the **Workspace Launcher** window is opened. Confirm the workspace by pressing **OK**.
13. If installation is successful, the **Rendering VREP** view is newly available in Sunrise.Workbench.

    Select the menu sequence **Window** > **Show View** > **Other...** and open the **Sunrise** folder. The installation is successful if the entry **Rendering VREP** is displayed.

5.6.1 Integrating V-REP PRO into the Rendering interface

Description

In order to integrate the simulation software V-REP PRO into the Rendering interface, the following files must be manually copied into the V-REP PRO installation directory.

(Default installation directory for version V-REP PRO V3.x under Windows 7 64 bit: C:\Program Files (x86)\V-REP3\V-REP_PRO)

- The file **remoteApiConnections.txt** (communication configuration) into the main directory …\V-REP_PRO
- The file **vrepAddOnScript_RenderingConnectorScript.lua** (add-on script) into the main directory …\V-REP_PRO

KUKA provides the required files in a catalog which can be installed in Sunrise.Workbench.

Procedure

1. Open the station configuration of the Sunrise project and select the **Software** tab.
2. Select the **Tools and Models for Rendering.VREP** catalog for installation.
   - Set the check mark in the **Install** column.
3. Save the station configuration. The system asks whether the modifications to the project should be accepted. Click on **Save and apply**.

   The **Rendering.VREP** folder is created in the Sunrise project. The folder contains the files required for using V-REP PRO.

6 Operating the Rendering interface

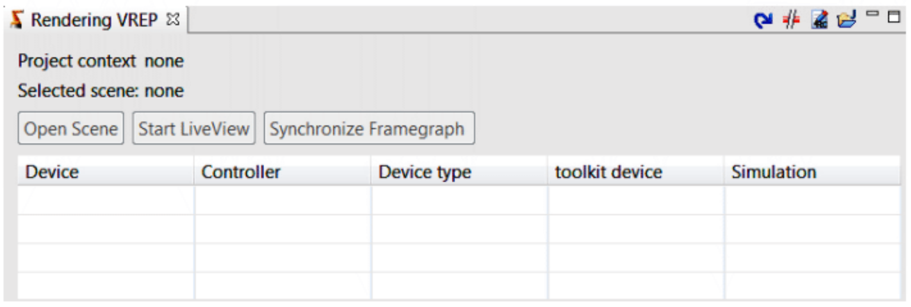
6.1 “Rendering VREP” view

Procedure

Open the **Rendering VREP** view in Sunrise.Workbench:

1. Select the menu sequence **Window** > **Show View** > **Other...** and open the **Sunrise** folder.
2. Select the entry **Rendering VREP** and confirm with **OK**.

Overview



Rendering VREP

Project context none

Selected scene: none

Device	Controller	Device type	toolkit device	Simulation

Fig. 6-1: “Rendering VREP” view (without link and project reference)

The following buttons are available:

Icon	Name / description
⟳	**Synchronize configuration** Refreshes the configuration of the interface and activates the configuration. Required after the following events: ■ The station configuration in Sunrise.Workbench has been changed. ■ Robots have been reconfigured or the robot configuration has been changed in V-REP PRO. ■ The connection to the real robot has been interrupted.
≠	**Stop LiveView** Ends the live view in V-REP PRO
▨	**Select scene** Opens the **Select scene** window. A scene projected in V-REP PRO can be selected.
▨	**Select project** Opens the **Select project** window. A Sunrise project can be selected.

The following buttons are available:

Button	Description
Load scene	Loads the selected scene in V-REP PRO. The button is only active if a Sunrise project has been selected.

Button	Description
Start LiveView	Starts the live view in V-REP PRO. (>>> 6.3 "Starting live view" Page 23)
Synchronize frame graph	Sends the frames created in the application data to V-REP PRO and displays them there. (>>> 6.4 "Synchronizing frames" Page 23)

6.2 Configuring the Rendering interface for V-REP PRO

Precondition

- V-REP PRO has been started.
- At least one scene has been configured with an LBR iiwa in V-REP PRO.
- Network connection to the robot controller

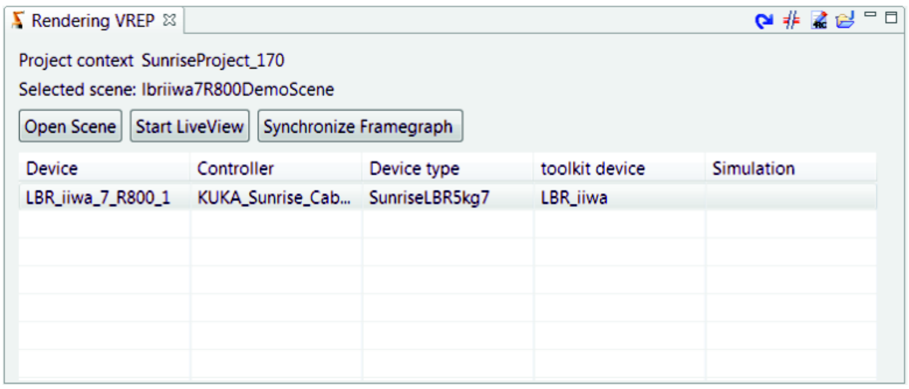
Procedure

1. Select the Sunrise project.
   a. Click on the **Select project** button. The **Select project** window with the available Sunrise projects opens.
   b. Select the desired Sunrise project and confirm with **OK**. The name of the selected Sunrise project is displayed in the **Rendering VREP** view.
2. Select a scene.
   a. Click on the **Select scene** button. The **Select scene** window opens.
   b. Select the **Open** button and navigate to the directory where the desired scene is located.
   c. Select the desired scene and and load it with **Open**. The path to the scene is displayed in the **Select scene** window.
   d. Close the window by clicking on **OK**. The name of the selected scene is displayed in the **Rendering VREP** view.
3. Click on the **Load scene** button. The selected scene is loaded in V-REP PRO.
4. The robots configured for the scene are available in the **Toolkit device** column.

   Select the robot here which is to be displayed in the live view later.

The settings are automatically saved in the Sunrise project.

Description



Project context SunriseProject_170
Selected scene: lbriiwa7R800DemoScene

Device	Controller	Device type	toolkit device	Simulation
LBR_iiwa_7_R800_1	KUKA_Sunrise_Cab...	SunriseLBR5kg7	LBR_iiwa	

Fig. 6-2: “Rendering VREP” view (with link and fully configured)

If there is no longer a connection to the real robot, the row containing this device is displayed in red text.

6.3 Starting live view

Precondition

- The Rendering interface is configured.
- V-REP PRO has been started.
- Network connection to the robot controller

Procedure

- Click on the **Start LiveView** button to start the live view in V-REP PRO.

6.4 Synchronizing frames

Precondition

- The Rendering interface is configured.
- V-REP PRO has been started.

Procedure

- Click on the **Synchronize Framegraph** button to send the frames created in the application data to V-REP PRO.

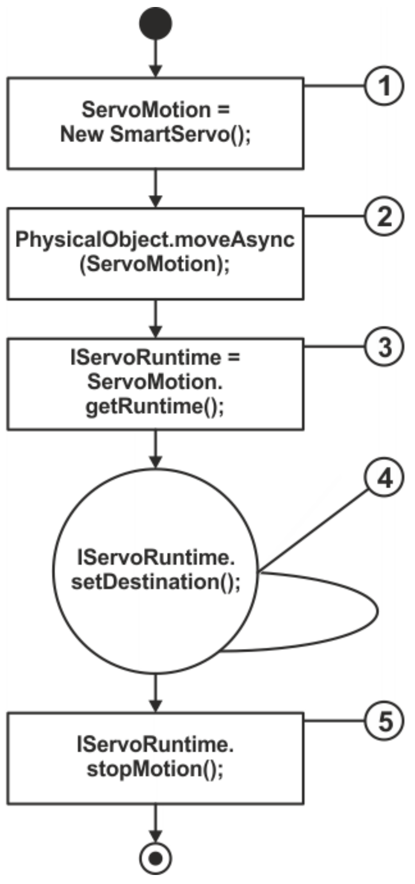
The newly created frames can be used in V-REP PRO for further modeling.

7 Programming

7.1 Servo motion in an application context

Overview

The figure shows the basic program sequence when executing a servo motion.



ServoMotion = New SmartServo();
PhysicalObject.moveAsync (ServoMotion);
IServoRuntime = ServoMotion. getRuntime();
IServoRuntime. setDestination();
IServoRuntime. stopMotion();
1
2
3
4
5

Fig. 7-1: 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 provided by the runtime instance of IServoRuntime. These data, e.g. the current robot position, velocity or acceleration, are available for further calling.

Item	Description
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. Once this time has elapsed, the servo motion is automatically ended (timeout after the time defined in the variable timeoutAfterGoalReach), provided that no new end point has been set or that the servo motion is ended with stopMotion(). (>>> 7.6 "Destination reached and timeouts" Page 34)
5	End the servo motion with stopMotion(). The robot application continues.

7.2 Simple destination setting with setDestination(…)

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

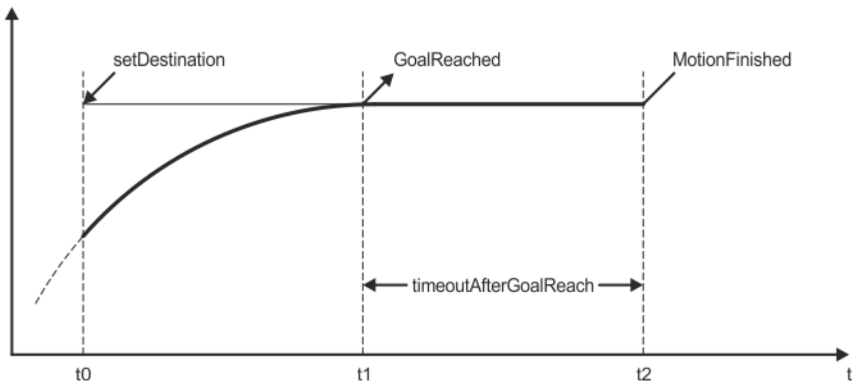


Fig. 7-2: Time diagram in an application context, basic behavior

The current velocity and position are applied in the application planning at the initial time **t0**. This results in a jerk-limited path being planned to the end position, with the specified maximum velocity being taken into account. With the SmartServo motion type, the maximum acceleration and maximum jerk are additionally taken into consideration.

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

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.

7.3 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.

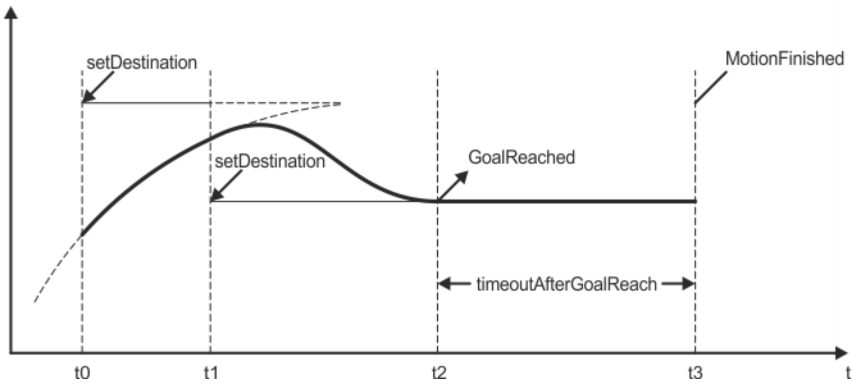


Fig. 7-3: Time diagram in an application context, replacing a destination setting prematurely

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. 7-3) 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**.

7.4 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.

If the seconds hand effect occurs, this 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.

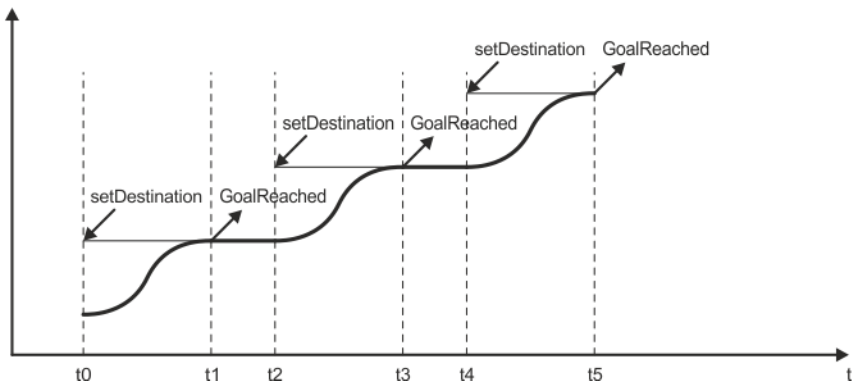


Fig. 7-4: Servo motion with seconds hand effect

Measures for avoiding the seconds hand effect can be found here:
(>>> 11.2 "Robot makes jerky motions/whistles – avoiding the seconds hand effect" Page 70)

7.5 IServoRuntime class

The IServoRuntime runtime environment is available as soon as the motion itself has been activated by the moveAsync(…) method.

The methods available in the runtime environment are divided into the following groups:

- Handling the real-time system
  - Time stamp
  - State of the precision interpolator
- Axis-specific interface
  - Reading out of axis positions
  - Setting of axis-specific setpoint positions
- Cartesian interface
  - Reading out of Cartesian positions
  - Setting of Cartesian setpoint positions

7.5.1 Handling the real-time system

When using time stamps, it must be ensured that the Java system time and real-time system time are not synchronized. The runtime environment provides process data which are updated, either when setting the destination with the setDestination(…) method or when the updateWithRealtimeSystem() method is explicitly called.

7.5.1.1 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.getTimeStampOfSetRealtimeDestina-
  tion()
  ```

  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()	
Instantaneous 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()	

7.5.1.2 Polling the state of the precision interpolator

Description In order to obtain current information when polling the precision interpolator, an update must first be carried out.

(>>> 7.5.1 "Handling the real-time system" Page 28)

The SmartServo motion type provides an event mechanism which signals that the destination has been reached.

(>>> 7.7 "Receiving the GoalReached event (SmartServo only)" Page 34)

Syntax

- Poll of whether the robot has reached the set destination:

  `boolean IServoRuntime.hasFineIpoGoalReached()`

- Poll of how long (in seconds) the robot still requires to reach the current set destination:

  `double IServoRuntime.getRemainingTime()`

7.5.1.3 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.toString());
```

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
levelOfVerbosity	Amount of detail with which the debug information is displayed ■ **1**: Standard information ■ **2**: Information which extends beyond the standard ■ **3**: All available information

7.5.2 Axis-specific interface

7.5.2.1 Specifying an axis-specific end position

Description

An axis-specific end position can be specified 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

`long IServoRuntime.setDestination(JointPosition` *newDestination*`)`

Explanation of the syntax

Element	Description
newDestination	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.

7.5.2.2 Specifying an axis-specific end position with a target velocity

Description

In the SmartServo motion type, it is possible to additionally specify the axis-specific velocity with which the end position is to be addressed using setDestination(…).

Syntax

`long ISmartServoRuntime.setDestination(JointPosition` *newDestination*`, JointSpeed` *newSpeed*`)`

Explanation of the syntax

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

7.5.2.3 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.

7.5.3 Cartesian interface

7.5.3.1 Setting a Cartesian destination in the robot base coordinate system

Description A new Cartesian destination can be set in the robot base coordinate system with setDestination(…).

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

Explanation of the syntax

Element	Description
destination	Cartesian destination in the robot base coordinate system. The robot flange or a frame subordinated to the flange is moved in the robot base coordinate system, e.g. the TCP of a tool.

Return value Time stamp of the command registration by the robot controller.

Example

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

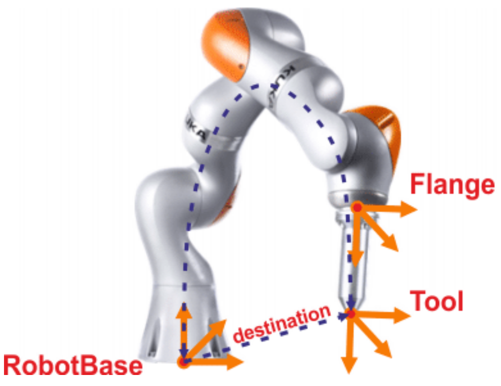


Fig. 7-5: Destination in the robot base

7.5.3.2 Specifying a Cartesian destination in the reference coordinate system

Description A new Cartesian destination can optionally be specified in an additional reference coordinate system with setDestination(…).

Syntax `long IServoRuntime.setDestination(AbstractFrame` *destination*`, AbstractFrame` *reference*`);`

Explanation of the syntax

Element	Description
destination	Cartesian destination in the reference coordinate system. The robot flange or a frame subordinated to the flange is moved in the reference coordinate system, e.g. the TCP of a tool.
reference	Reference coodinate system with a static connection to the robot base.

Return value Time stamp of the command registration by the robot controller.

Example

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

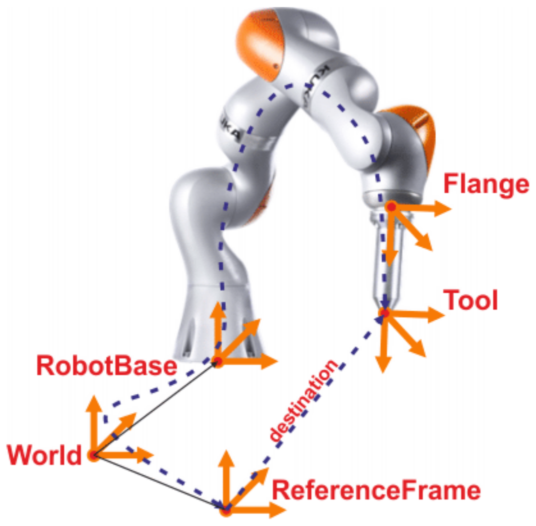


Fig. 7-6: Destination in the reference coordinate system

7.5.3.3 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().

(>>> 7.5.1 "Handling the real-time system" Page 28)

(>>> 7.5.1.1 "Polling time stamps" Page 28)

Example

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

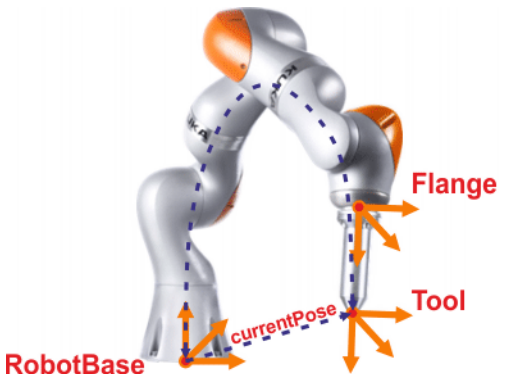


Fig. 7-7: Actual position in the robot base

7.5.3.4 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 world coordinate system.

Return value

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

(>>> 7.5.1 "Handling the real-time system" Page 28)

(>>> 7.5.1.1 "Polling time stamps" Page 28)

Example

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

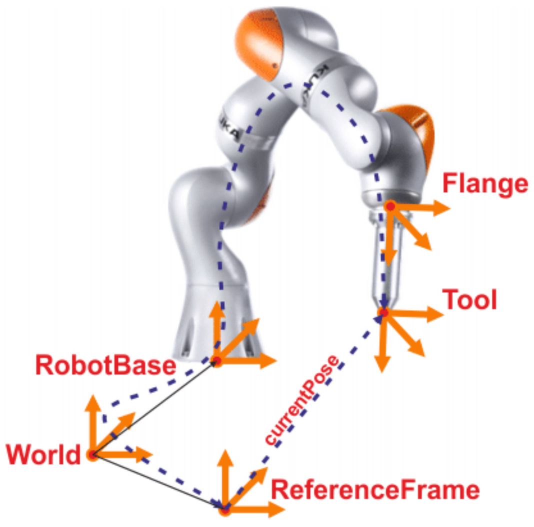


Fig. 7-8: Actual position in the reference coordinate system

7.5.4 Terminating 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()
```

7.6 Destination reached and timeouts

A state diagram illustrates the relationship between the reaching of the destination and the timeout. The state diagram shows the inner state machine of the precision interpolator. The isDestinationReached() method contained in the IServoRuntime class returns a state value of the data type Boolean when polled. Other state information is not significant for the application, as this information is too short-lived.

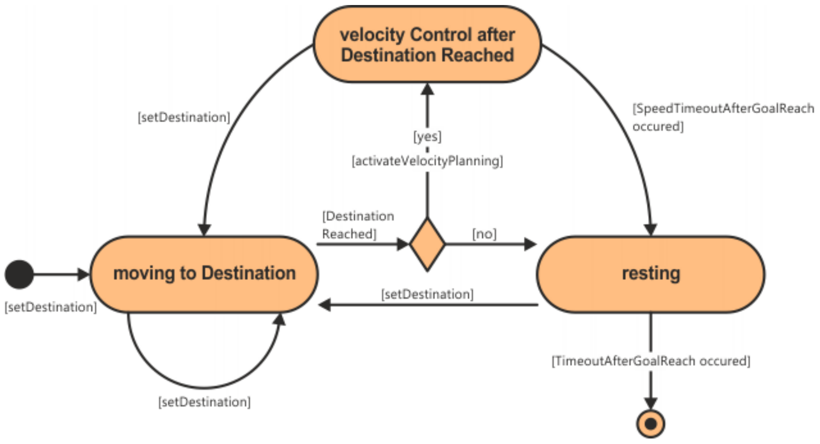

velocity Control after Destination Reached
[setDestination]
[SpeedTimeoutAfterGoalReach occured]
[yes]
[activateVelocityPlanning]
[Destination Reached]
[no]
moving to Destination
resting
[setDestination]
[setDestination]
[TimeoutAfterGoalReach occured]
[setDestination]

Fig. 7-9: Destination reached and timeouts (state diagram)

- If a new destination setting is sent with the setDestination(…) method, the precision interpolator jumps to the “moving” state.
- As long as a motion remains to be carried out, the precision interpolator of the real-time controller is in the “moving” state.
- Once the destination has been reached, the velocity of the destination state remains in the “velocityAfterGoalReach” state.
    - This state is only reached if the velocity planning is activated, i.e. if the variable activateVelocityPlanning has been set to “true”.
    - When using the SmartServo motion type and specifying a new axis-specific end position, the destination velocity can be determined with the setDestination(…) method.
    - The duration of the velocity to be maintained in the “velocityAfterGoalReach” state is defined with the setSpeedTimeoutAfterGoalReach(…) method.
- Following the “VelocityAfterGoalReached” or “moving” state, the precision interpolator goes into the “resting” state.
- The “resting” state is terminated after the time defined in the variable timeoutAfterGoalReach has elapsed. The default value is 120 seconds.
- Every state can be terminated with the stopMotion() command at any time.

7.7 Receiving the GoalReached event (SmartServo only)

In order to receive notification by means of the onGoalReachedEvent(…) event that the destination has been reached, the SampleGoalReachedEventListener listener must be registered at the IServoRuntime interface. This is done by calling the setGoalReachedEventHandler(…) method. The SampleGoalReachedEventListener listener must be derived from the IServoOnGoalReachedEvent class.

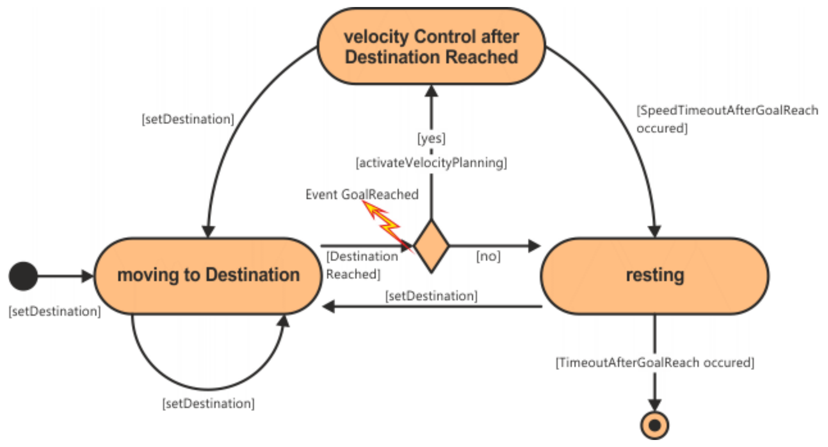


Fig. 7-10: Receiving the GoalReached event (SmartServo only)

```
/**
 * 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");
        }
    }
}
```

The listener must implement the onGoalReachedEvent(…) method.

7.8 Changing the path parameters

The path parameters velocity, acceleration and jerk can be set before a servo motion is activated. 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 acceleration and jerk can optionally be increased with SmartServo. The overrideJointAcceleration(…) and setJointJerk(…) methods are used for this. 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 SmartServo and DirectServo:

Method	**Description**	**SmartServo**	**DirectServo**
ServoMotion.setJointVelocityRel(…)	Maximum axis velocity (type: double, unit: %) ■ **0 … 1**	✓	✓
ServoMotion.setJointAccelerationRel(…)	Acceleration (type: double, unit: %) ■ **0 … 1**	✓	✗
SmartServo.overrideJointAcceleration(…)	Acceleration magnification (type: double, unit: factor) ■ **0 … 20**	✓	✗
SmartServo.setJointJerk(…)	Jerk (type: double, unit: %) ■ **0 … 1500**	✓	✗

7.9 Time recording

7.9.1 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.

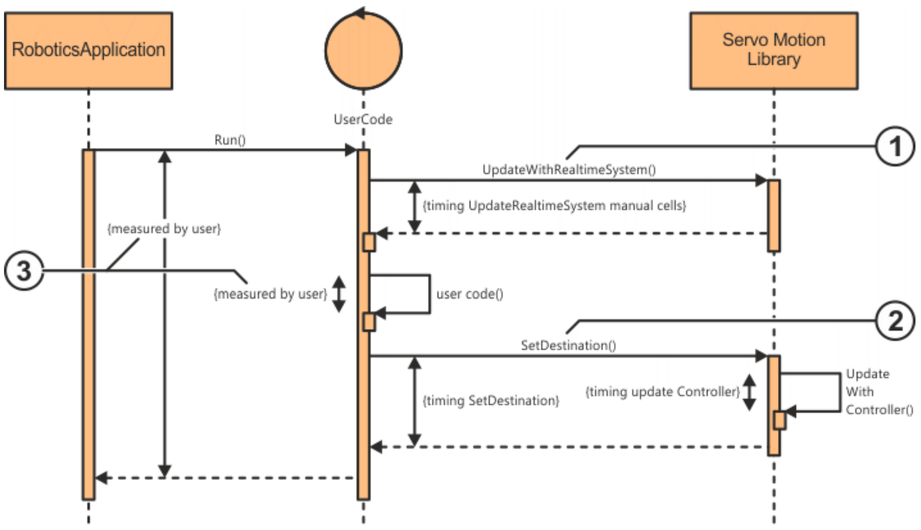


Fig. 7-11: Time measurement

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 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.

7.10 Impedance control

⚠ WARNING In impedance control, incorrectly selected parameters (e.g. incorrect load data, incorrect tool) or incorrect information (e.g. from defective torque sensors) can be interpreted as external forces, resulting in unpredictable motions of the robot.

⚠ WARNING In order to avoid hazards caused by unpredictable motions, 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.

7.10.1 Validating the load model for impedance control

Description

In order to use servo motions with a robot under impedance control, a validation must be carried out once during the runtime of the application.

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.

Example

The impedance control is validated with SmartServo.validateForImpedanceMode(…). If the validation is invalid, a message is displayed on the smartHMI.

```
if (SmartServo.validateForImpedanceMode(_toolAttachedToLBR) != true)
{
    getLogger().info("Validation of Torque Model failed");
}
```

7.10.2 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));
```

7.10.3 Changing controller parameters during the 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(IMotion-ControlMode` *controlMode*`)`

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(…)

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(aTransStiffVal);
    cartImp.parametrize(CartDOF.ROT).setStiffness(aRotStiffVal);
```

⚠ WARNING 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);
}
```

⚠ WARNING 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).setMaxPathDeviation(50.0);
cartImp.parametrize(CartDOF.ROT).setMaxPathDeviation(1.0);
```

7.10.4 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, the motion sections switch over seamlessly 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(…). (>>> 8.1.3 "SmartServoSampleInteractionControl application" Page 59)

```
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.

7.11 Feedback effect for servo motions

Description

When using the SmartServo and DirectServo motion types, feedback may occur as a control-related secondary effect. This effect is comparable to the feedback from an overloaded microphone through the loudspeaker to which it is connected.

Robot applications using standard motions send destination settings to the robot controller without including position feedback in their destination setting calculation.

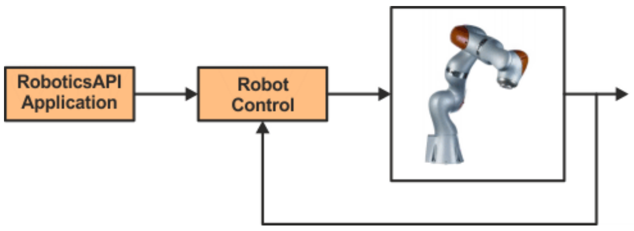


Fig. 7-12: Destination setting for standard motions

For servo motions, by contrast, position feedback is 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

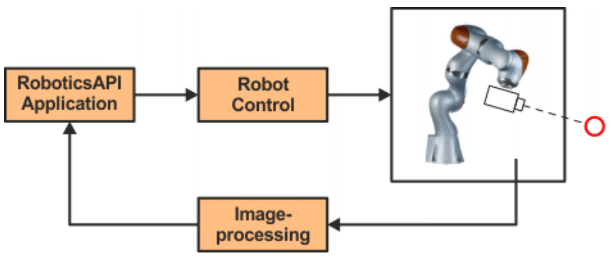


Fig. 7-13: Destination setting for servo motions, visual servoing example

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

Figure (>>> Fig. 7-14) illustrates the signal flow of the control loop.

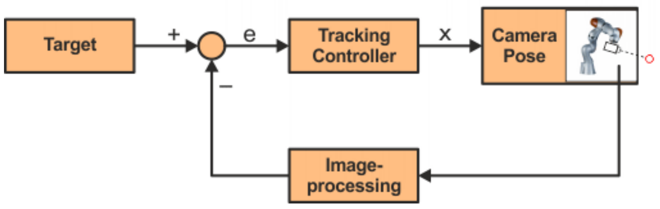


Fig. 7-14: Control loop for signal flow, visual servoing example

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

Figure (>>> Fig. 7-15) 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.

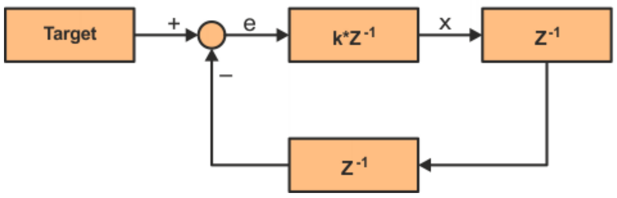


Fig. 7-15: p gain, visual servoing example

This results in a transfer function of: $G(z) = 1 / (1 + kz^3)$

The roots of the characteristic polynomial thus depend on the gain factor k of the actual controller, the “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 gain k < 1 (tracking controller) must be selected for stabilization.
  - Reduction of the robot performance (maximum velocity) in the camera pose.
  - Adaptation of the tracking controller.

- Suppression of the pixel noise in image processing.

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.

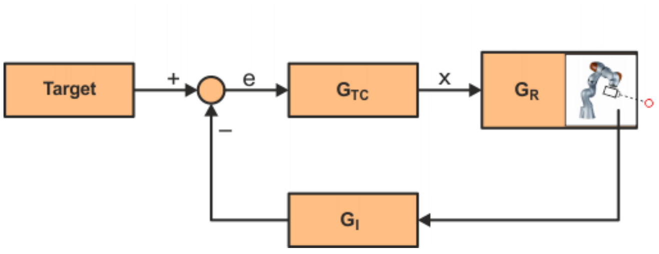


Fig. 7-16: Adaptation, visual servoing example

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 motion types SmartServo and DirectServo

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

7.12 Step response of the systems

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°

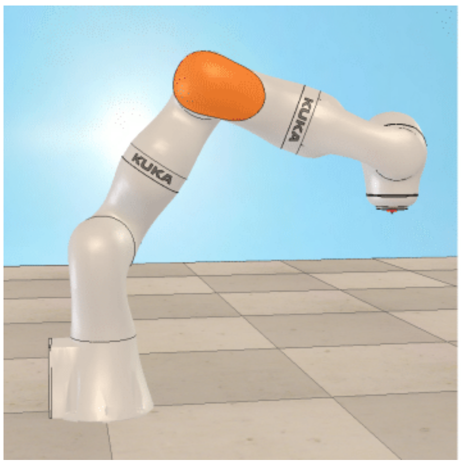


Fig. 7-17: Start position

7.12.1 SmartServo vs. DirectServo – comparison of axis positions over time

The following diagrams show the measured actual position of each axis over time. The blue line illustrates the commanded step (“tgt”).

The green line represents the path of the SmartServo motion type with the default setting. The acceleration values are rated very conservatively here to ensure that the robot can still be operated safely in the worst case (extended position and load).

The red line represents the path of the DirectServo motion type.

Axis acceleration and jerk can be changed in the SmartServo motion type. The light blue line represents the SmartServo motion type with the following settings:

```
aSmartServoMotion.setJointJerk(500);
aSmartServoMotion.overrideJointAcceleration(20);
```

7.12.1.1 Step response comparison of SmartServo vs. DirectServo, A1

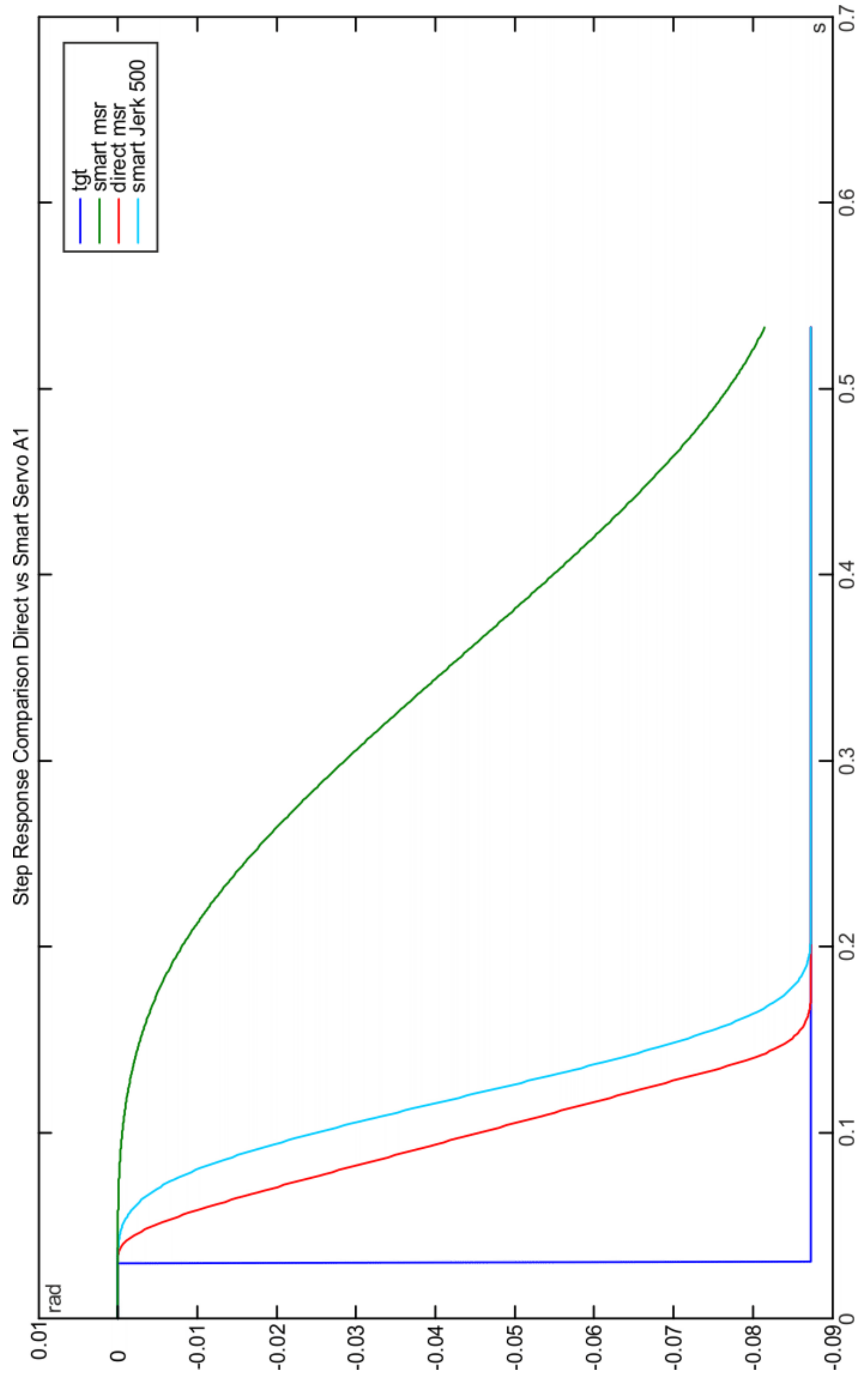


Fig. 7-18: Step response comparison of SmartServo vs. DirectServo, A1

7.12.1.2 Step response comparison of SmartServo vs. DirectServo, A2

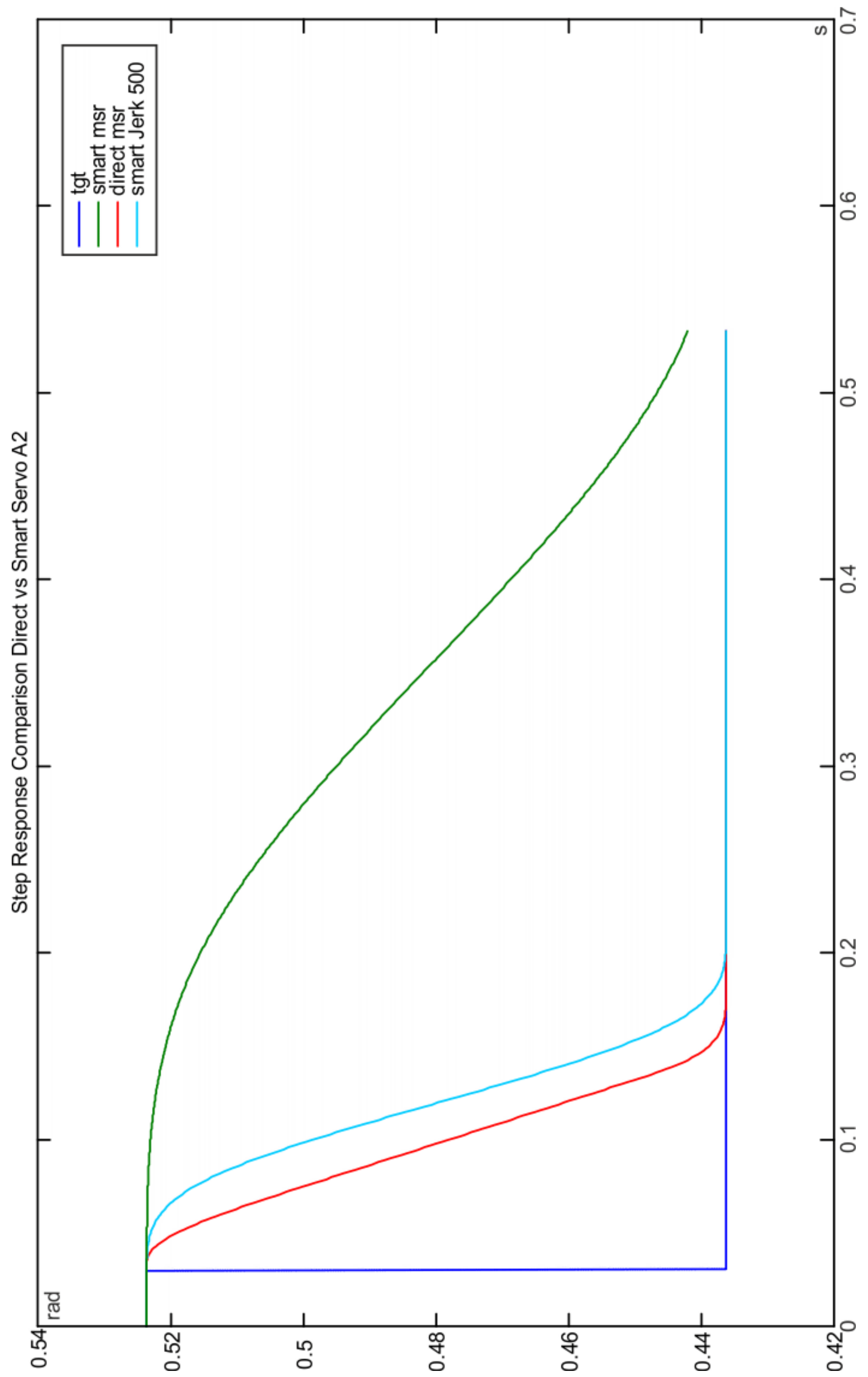


Fig. 7-19: Step response comparison of SmartServo vs. DirectServo, A2

7.12.1.3 Step response comparison of SmartServo vs. DirectServo, A3

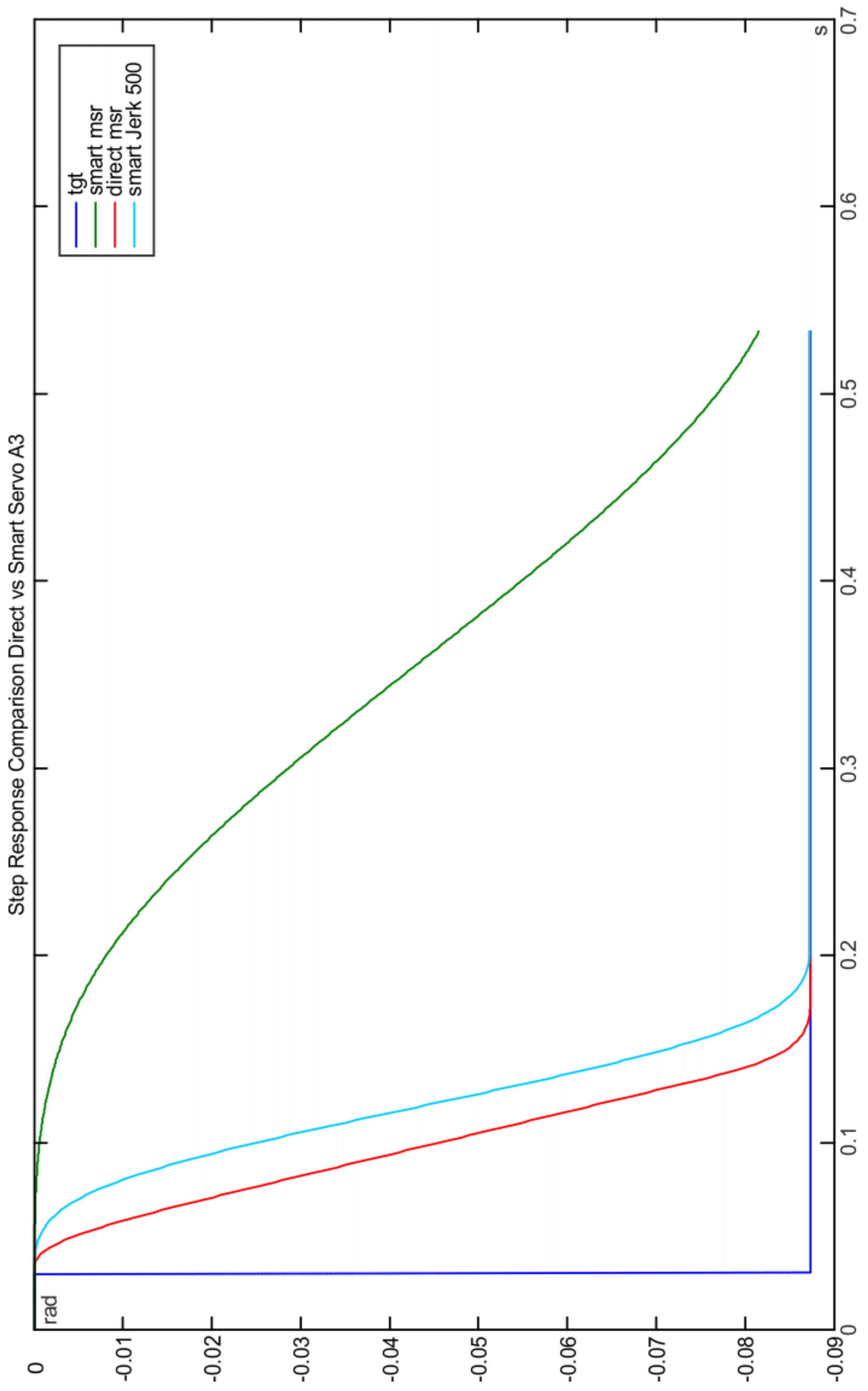


Fig. 7-20: Step response comparison of SmartServo vs. DirectServo, A3

7.12.1.4 Step response comparison of SmartServo vs. DirectServo, A4

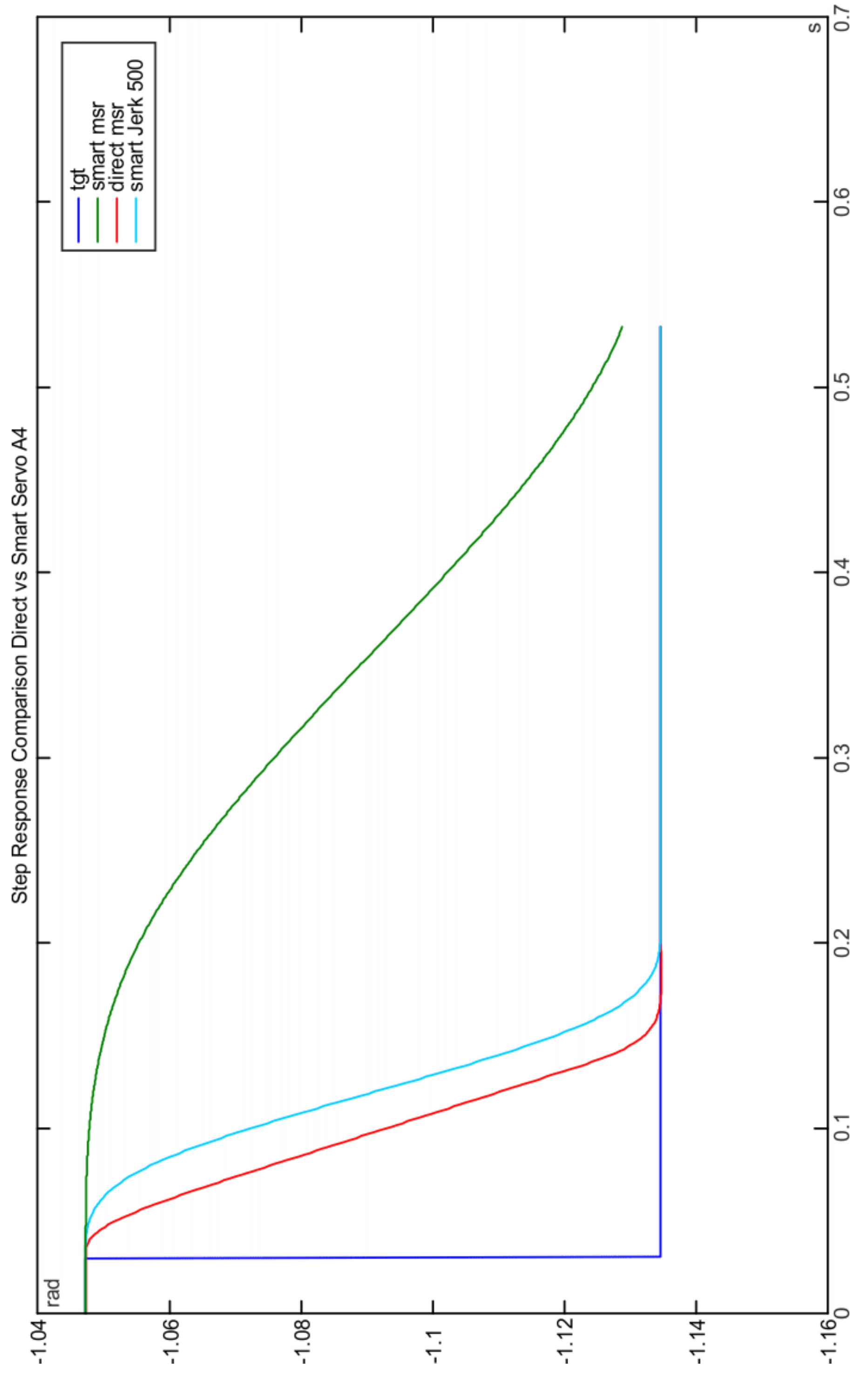


Fig. 7-21: Step response comparison of SmartServo vs. DirectServo, A4

7.12.1.5 Step response comparison of SmartServo vs. DirectServo, A5

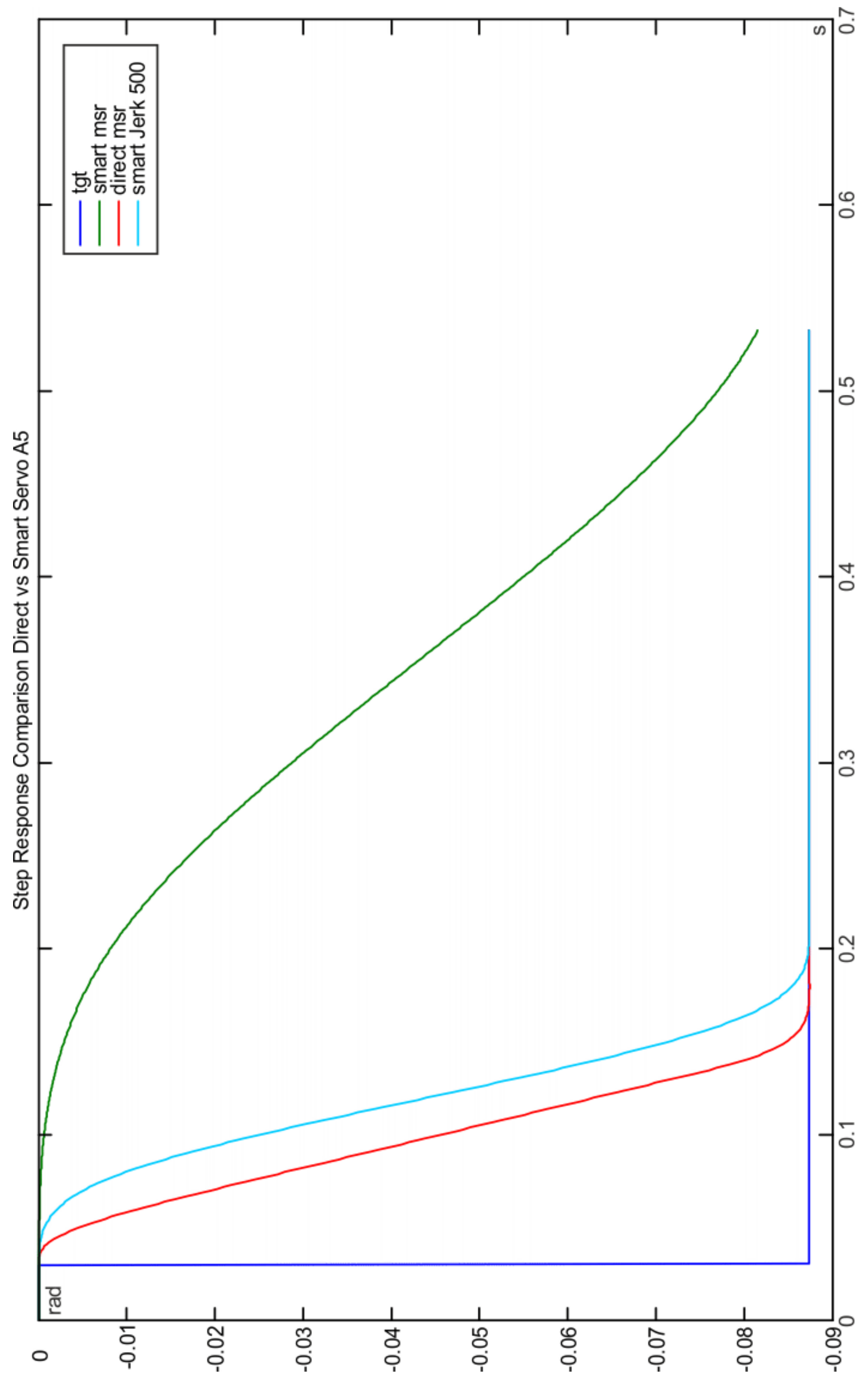


Fig. 7-22: Step response comparison of SmartServo vs. DirectServo, A5

7.12.1.6 Step response comparison of SmartServo vs. DirectServo, A6

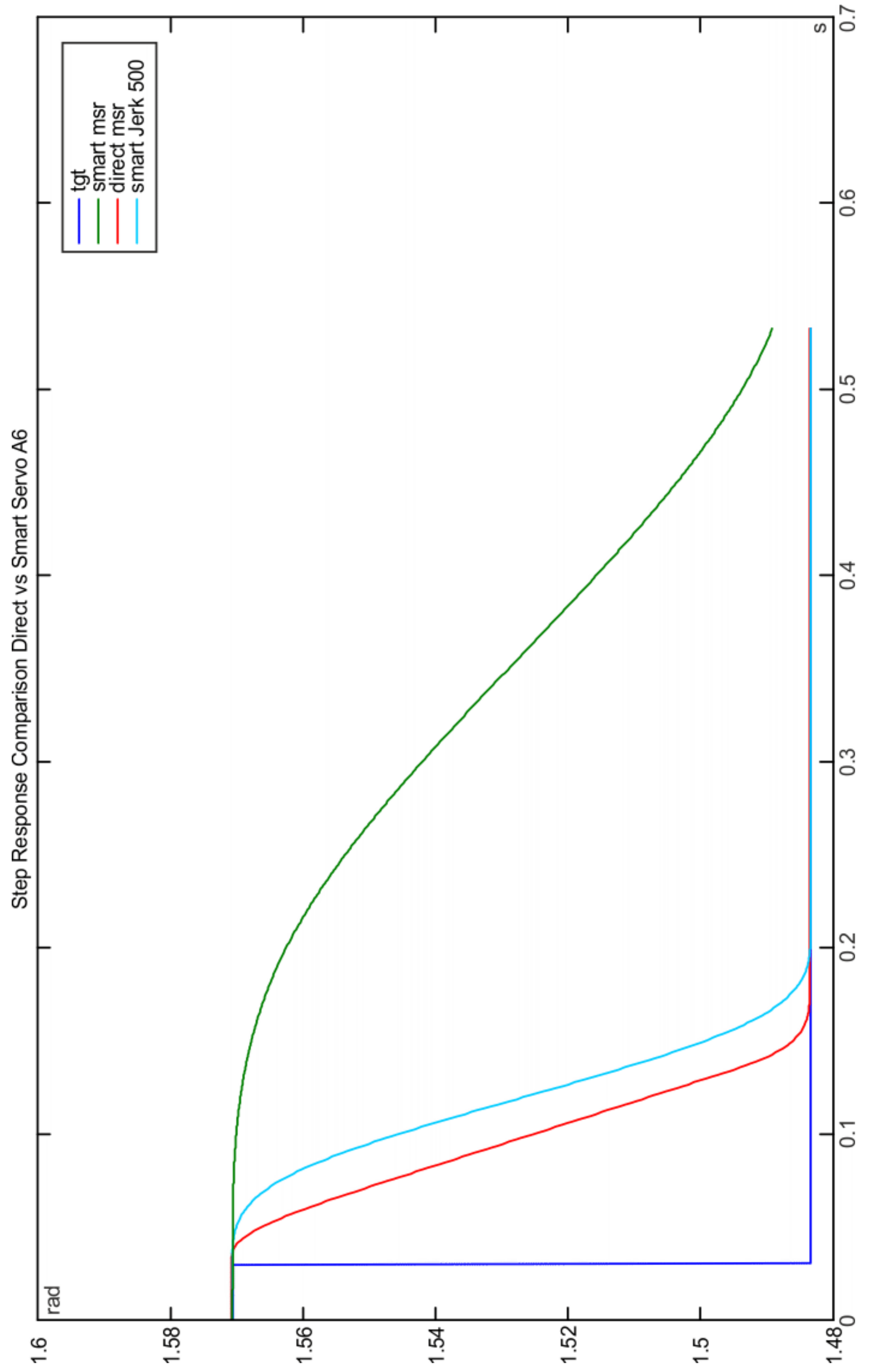


Fig. 7-23: Step response comparison of SmartServo vs. DirectServo, A6

7.12.1.7 Step response comparison of SmartServo vs. DirectServo, A7

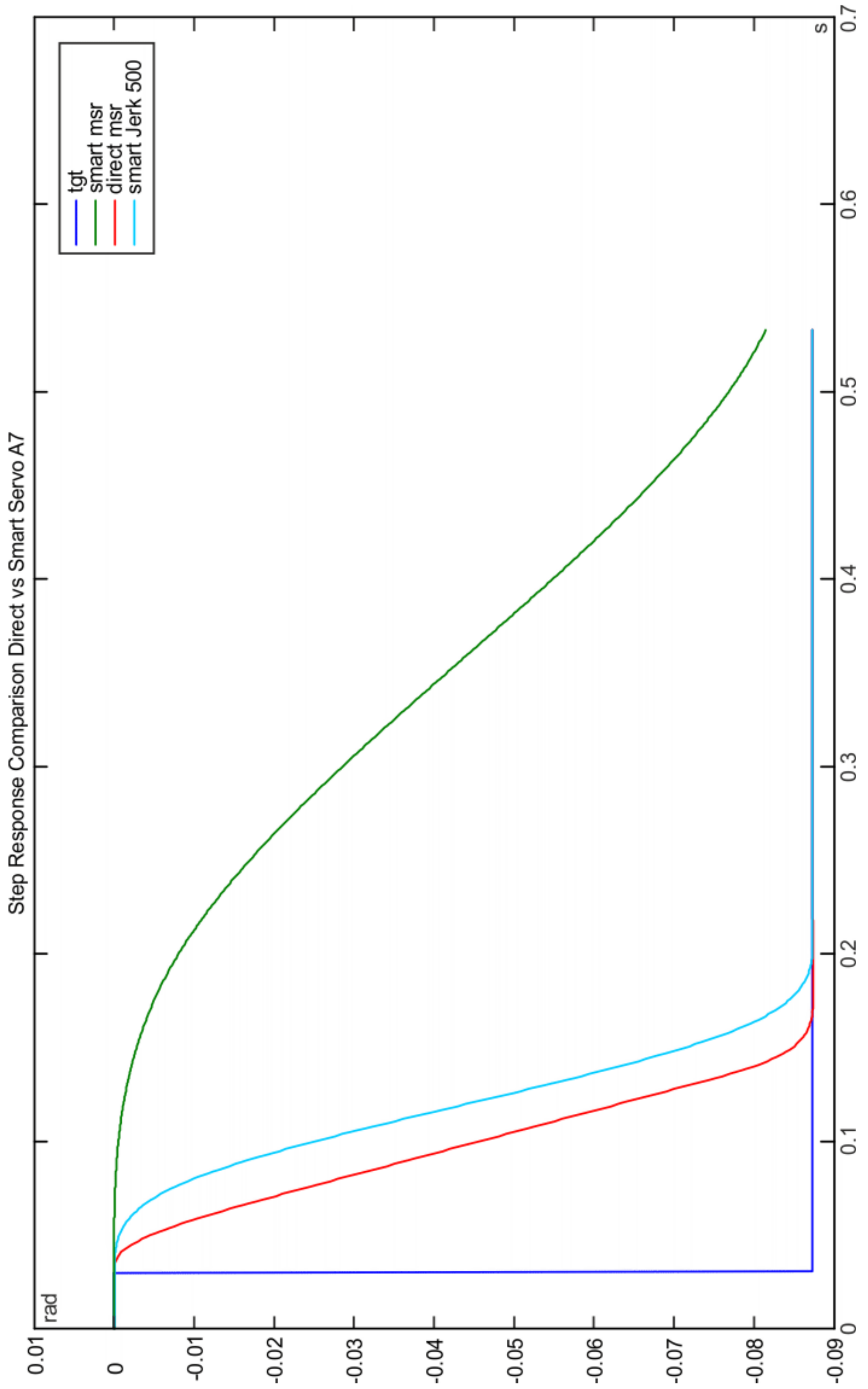


Fig. 7-24: Step response comparison of SmartServo vs. DirectServo, A7

7.12.2 SmartServo vs. DirectServo – comparison of velocity, acceleration, jerk

In addition to the position, the following diagrams show the velocity, acceleration and jerk of each axis.

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 of < 10 ms and exceeded regulator limit. If this occurs and the regulator reserves are exhausted by aSmartServoMotion.setJointJerk(500) and aSmartServo-Motion.overrideJointAcceleration(20), the DirectServo motion type is recom-mended.

7.12.2.1 Step response comparison of SmartServo vs. DirectServo, A1 (advanced)

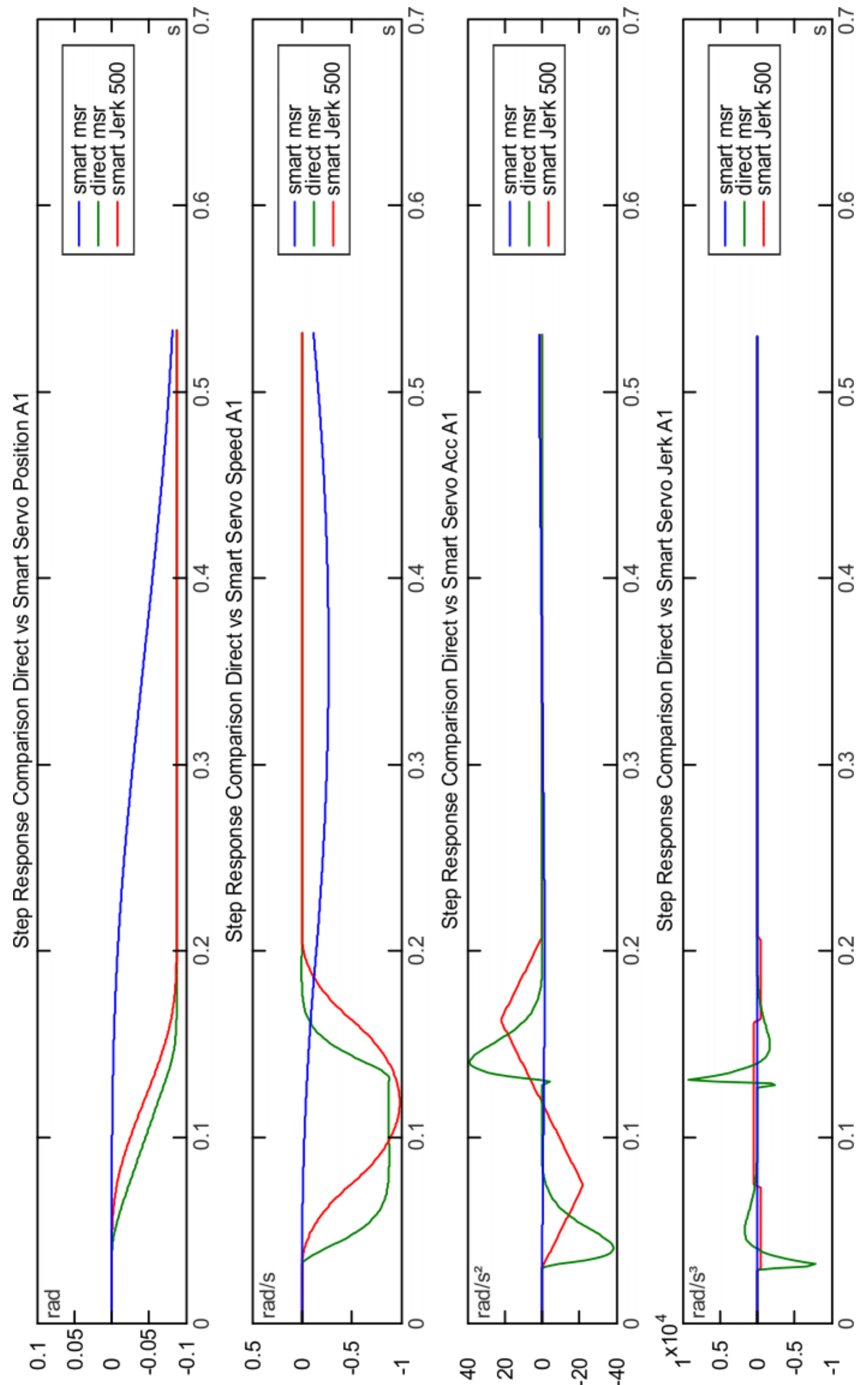


Fig. 7-25: Step response comparison of SmartServo vs. DirectServo, A1, advanced

7.12.2.2 Step response comparison of SmartServo vs. DirectServo, A2 (advanced)

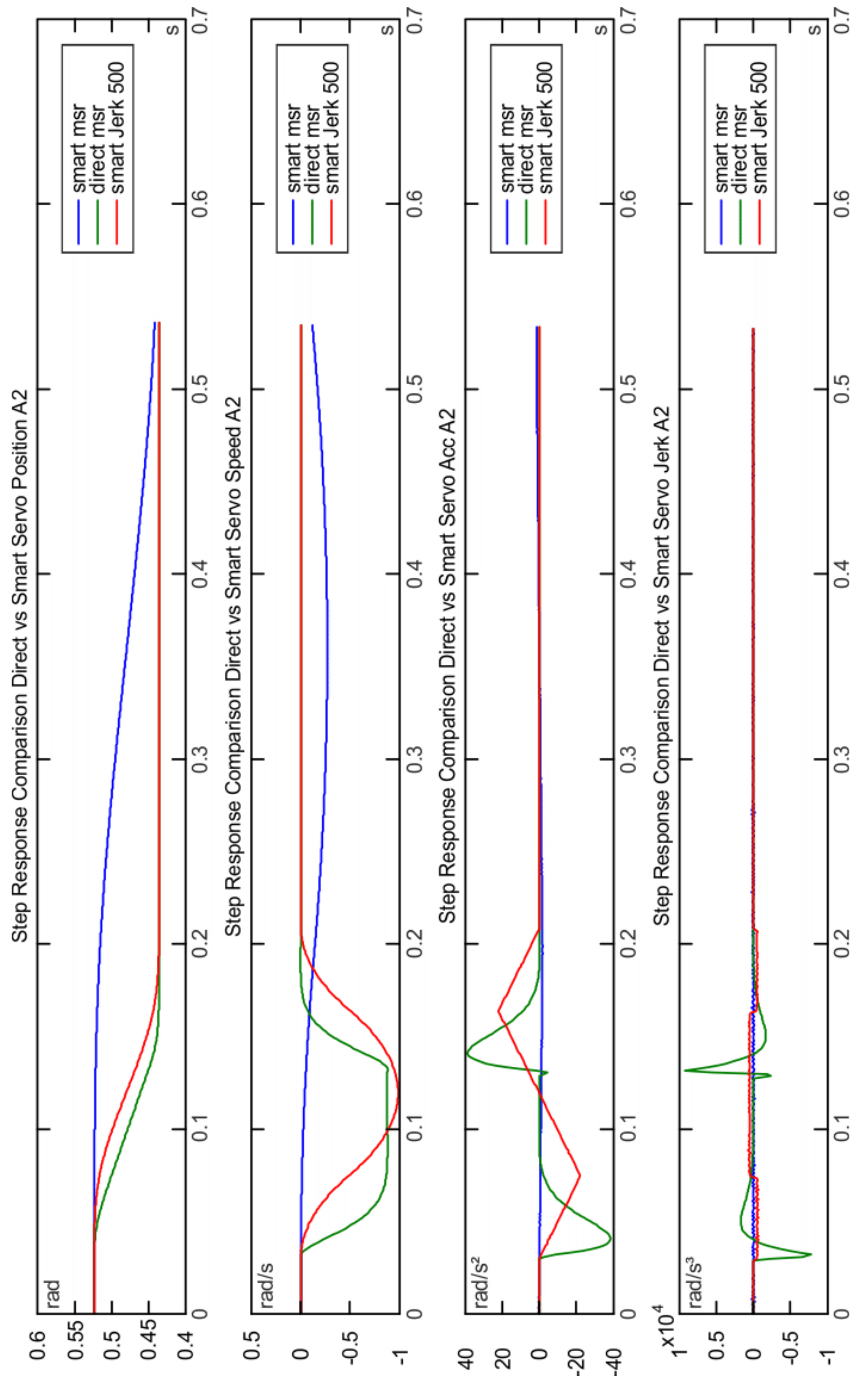


Fig. 7-26: Step response comparison of SmartServo vs. DirectServo, A2, advanced

7.12.2.3 Step response comparison of SmartServo vs. DirectServo, A3 (advanced)

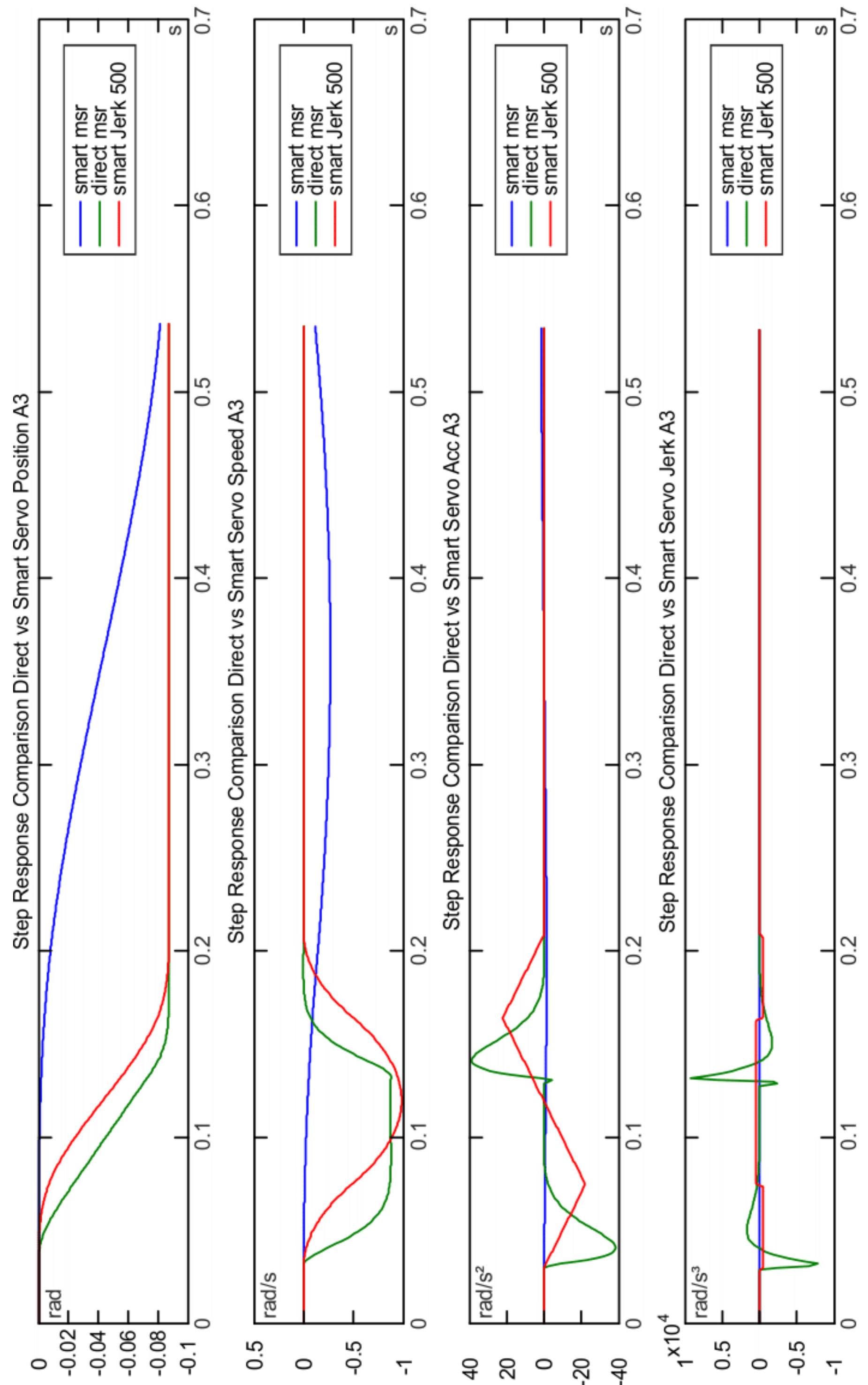


Fig. 7-27: Step response comparison of SmartServo vs. DirectServo, A3, advanced

7.12.2.4 Step response comparison of SmartServo vs. DirectServo, A4, advanced

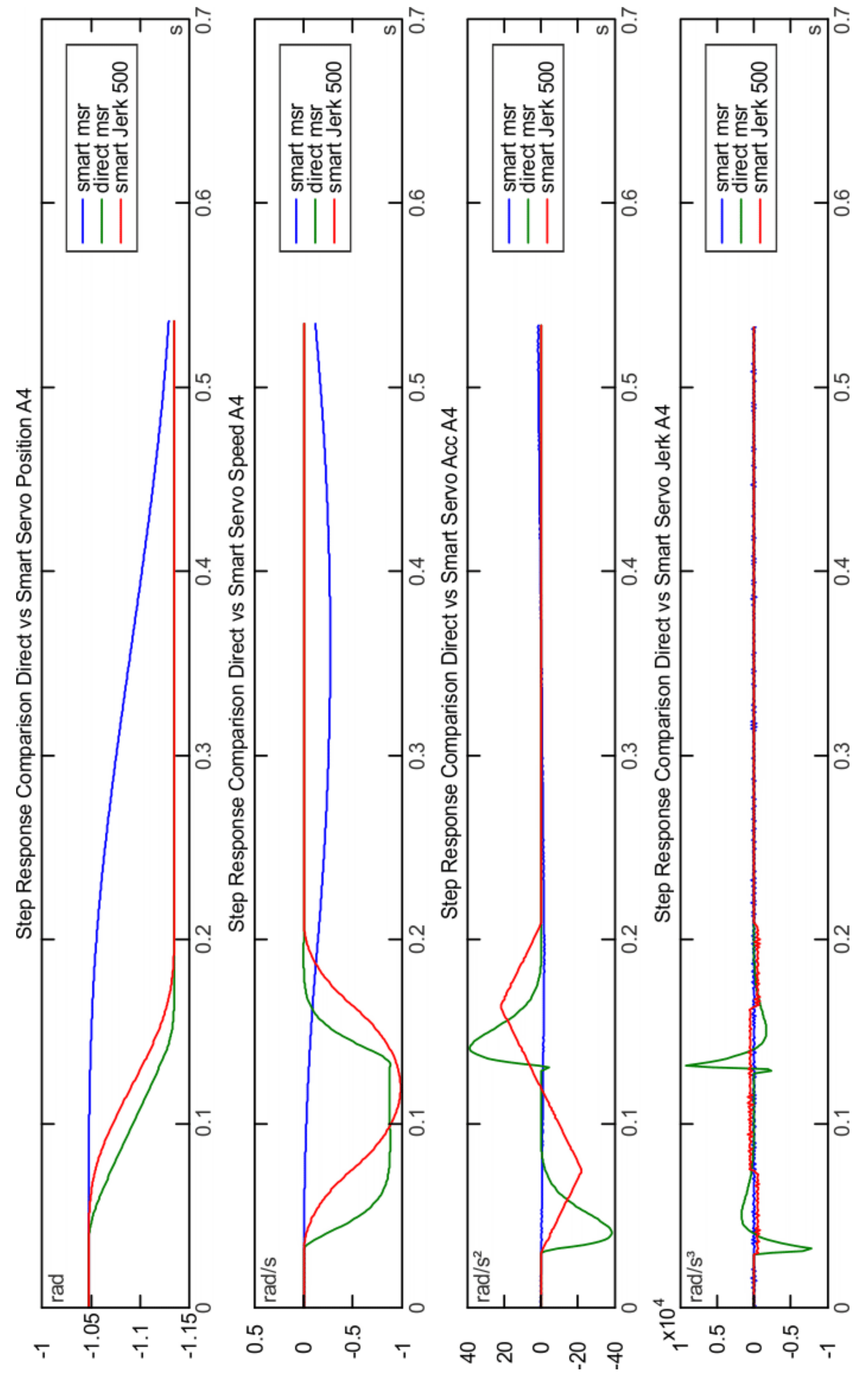


Fig. 7-28: Step response comparison of SmartServo vs. DirectServo, A4, advanced

7.12.2.5 Step response comparison of SmartServo vs. DirectServo, A5 (advanced)

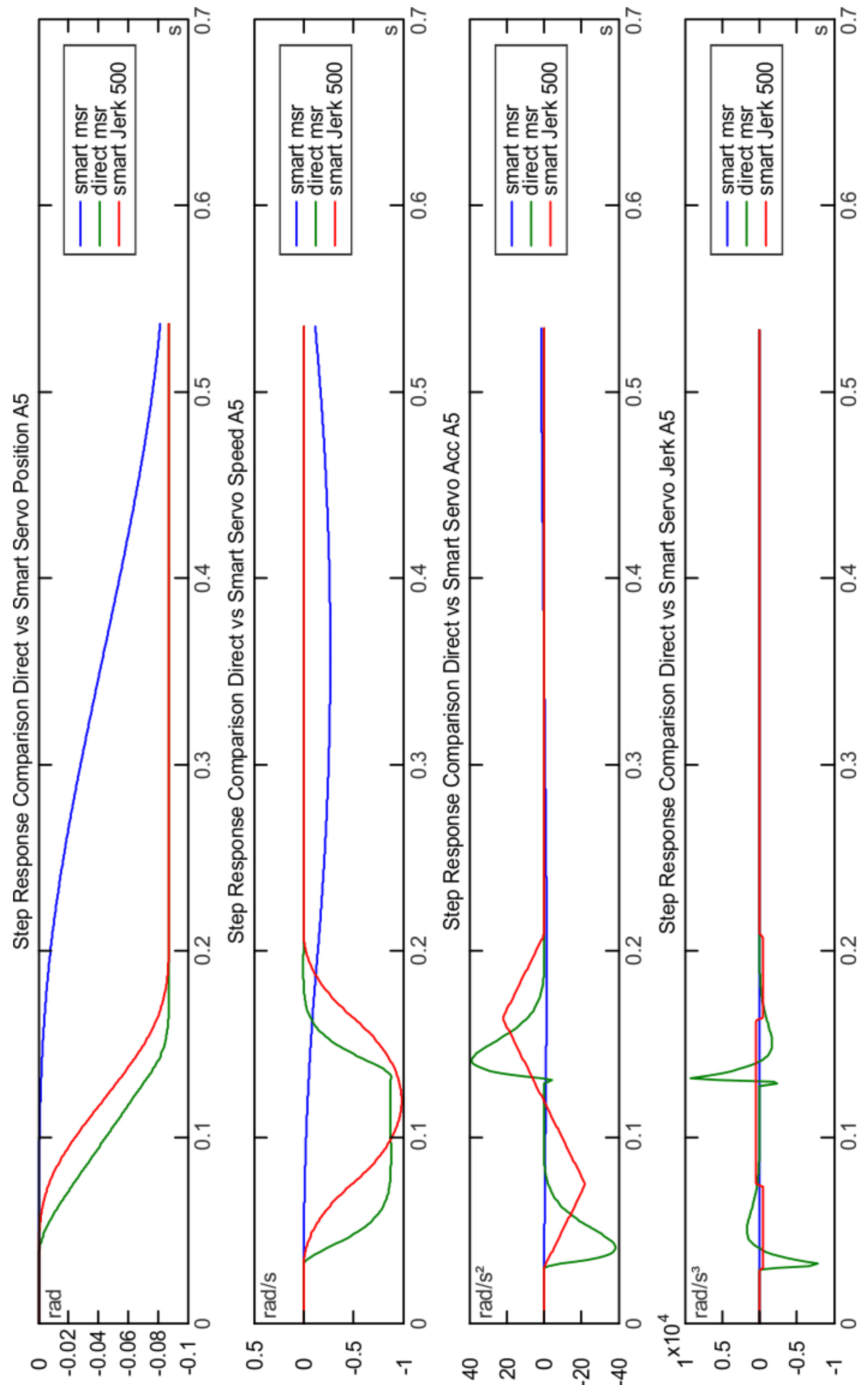


Fig. 7-29: Step response comparison of SmartServo vs. DirectServo, A5, advanced

7.12.2.6 Step response comparison of SmartServo vs. DirectServo, A6 (advanced)

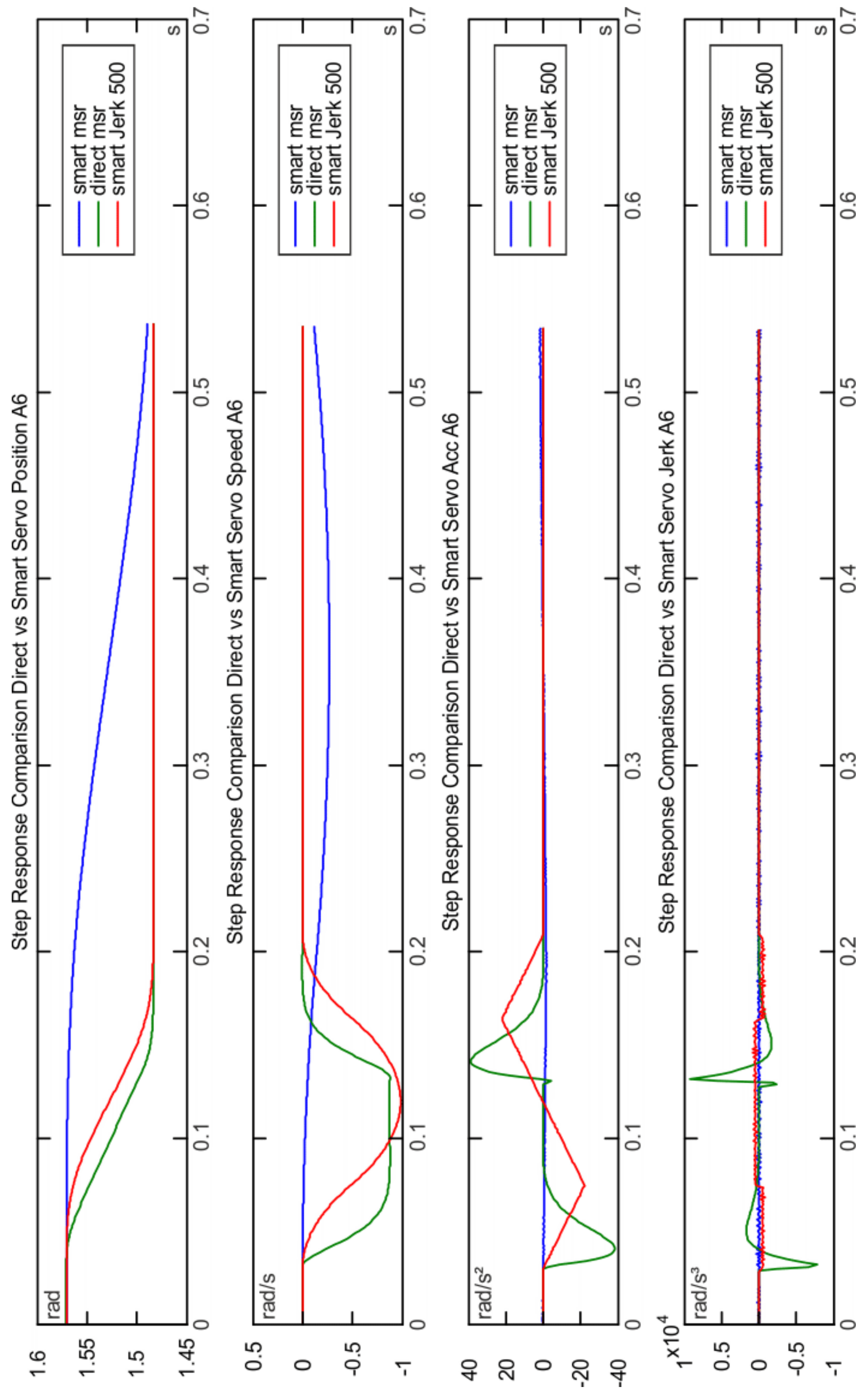


Fig. 7-30: Step response comparison of SmartServo vs. DirectServo, A6, advanced

7.12.2.7 Step response comparison of SmartServo vs. DirectServo, A7 (advanced)

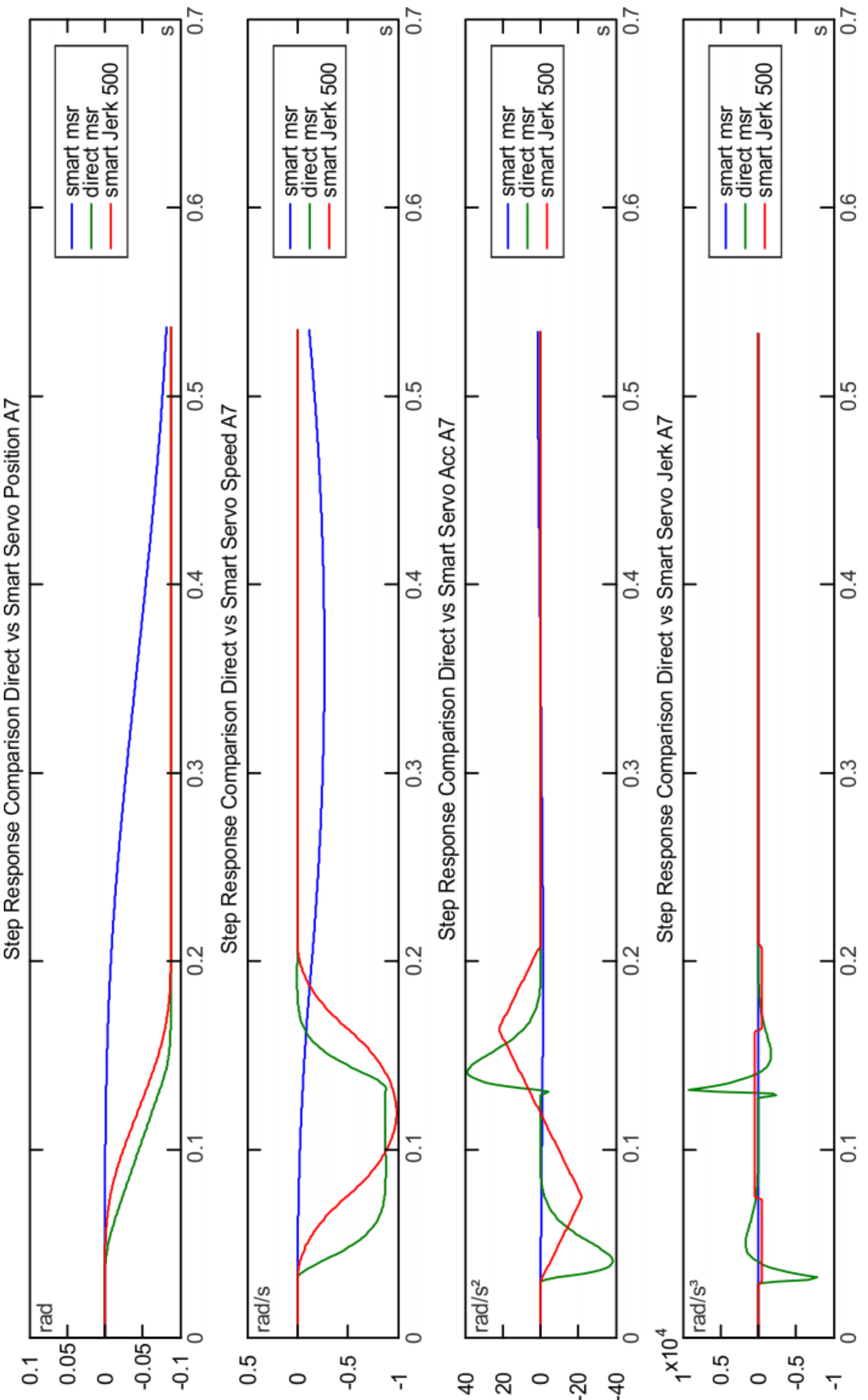


Fig. 7-31: Step response comparison of SmartServo vs. DirectServo, A7, advanced

8 Application examples

Programming examples are contained in the “Simple Tutorials for SmartServo” and “Simple Tutorials for DirectServo” catalogs provided.

We recommend reading them in the following order.

8.1 “Simple Tutorials for SmartServo” catalog

8.1.1 SmartServoSampleSimpleJointMotion application

Functionality Simple axis-specific destination setting.

Objectives

- Activation of the motion type SmartServo
  (>>> 7.1 "Servo motion in an application context" Page 25)
- Working with the IServoRuntime runtime environment
  (>>> 7.5 "IServoRuntime class" Page 28)
- Axis-specific destination setting
  (>>> 7.5.2 "Axis-specific interface" Page 30)
- Time measurement
  (>>> 7.9 "Time recording" Page 36)

Precondition Knowledge of application programming with RoboticsAPI.

8.1.2 SmartServoSampleSimpleCartesian application

Functionality Simple Cartesian destination setting.

Objectives

- Cartesian destination setting
  (>>> 7.5.3 "Cartesian interface" Page 31)

Precondition Programming knowledge in the SmartServoSampleSimpleJointMotion application.

8.1.3 SmartServoSampleInteractionControl application

Functionality Impedance control with the motion type SmartServo.

Objectives

- Validating the load model for impedance control
  (>>> 7.10.1 "Validating the load model for impedance control" Page 37)
- Activating impedance control
  (>>> 7.10.2 "Activating impedance control" Page 38)
- Changing controller parameters during runtime
  (>>> 7.10.3 "Changing controller parameters during the runtime" Page 38)
- Transition between control types
  (>>> 7.10.4 "Transition between control types" Page 39)

Precondition Programming knowledge in the applications SmartServoSampleSimpleJointMotion and SmartServoSampleSimpleCartesian.

8.2 “Simple Tutorials for DirectServo” catalog

Knowledge of the “Simple Tutorials for SmartServo” catalog is a prerequisite for the safe use of the “Simple Tutorials for DirectServo” catalog.

8.2.1 DirectServoSampleSimpleJointMotion application

Functionality Simple axis-specific destination setting.

Objectives

- Activating the motion type DirectServo
- Working with the IServoRuntime runtime environment
- Axis-specific destination setting
- Time measurement

Precondition Programming knowledge in the SmartServoSampleSimpleJointMotion application.

8.2.2 DirectServoSampleSimpleCartesian application

Functionality Simple Cartesian destination setting.

Objectives

- Cartesian destination setting

Precondition Programming knowledge in the applications SmartServoSampleSimpleCartesian and DirectServoSampleSimpleJointMotion.

8.2.3 DirectServoSampleInteractionControl application

Functionality Impedance control with the motion type DirectServo.

Objectives

- Validating the load model for impedance control
- Activating impedance control
- Changing controller parameters during runtime
- Transition between control types

Precondition Programming knowledge in the applications SmartServoSampleSimpleJointMotion, SmartServoSampleSimpleCartesian and SmartServoSampleInteractionControl.

9 Command reference

9.1 SmartServo command reference

9.1.1 Motion parameters

Method	Description
setJointVelocity-Rel(…)	Axis-specific relative velocity (type: double, unit: %) ■ **0.0 … 1.0** Default value: **1** Refers to the maximum value of the axis velocity in the machine data. **Note:** Further information on the programming of axis-specific motion parameters is contained in the operating and programming instructions for the System Software.
setJointAcceleration-Rel(…)	Axis-specific relative acceleration (type: double, unit: %) ■ **0.0 … 1.0** Default value: **1** Refers to the maximum value of the axis acceleration in the machine data. **Note:** Further information on the programming of axis-specific motion parameters is contained in the operating and programming instructions for the System Software.
setMinimumTrajecto-ryExecutionTime(…)	Minimum execution time for a trajectory (type: double, unit: seconds) ■ **≥ 0** Default value: **0.01** Sets the minimum execution time for a trajectory. Changes the path.
setSpeedTime-outAfterGoal-Reach(…)	Timeout for the velocity profile after the destination is reached (type: double, unit: seconds) ■ **≥ 0** Default value: **0** Sets the timeout, specifying the point up to which the interpolator should continue to plan with a velocity profile.
setTimeoutAfterGoal-Reach(…)	Timeout for the motion command after the destination has been reached (type: double, unit: seconds) ■ **≥ 0** Default value: **120** If no new destination is set after the specified time has elapsed, the motion command is terminated.
setMode(…)	Setting of the motion type. **Note:** Further information on programming is contained in the operating and programming instructions for the System Software.
overrideJointAcceler-ation(…)	Increase of the axis-specific acceleration (type: double, unit: factor) ■ **0 ... 20** Default value: **1**

9.1.2 Runtime parameters

Method	Description
setGoalReachedE-ventHandler(…)	Setting of the event handler when the destination is reached. The parameter must be an implementation of the type IServoOnGoal-ReachedEvent.
setDetailedOutput(…)	Setting of the amount of detail of the debug information ■ **1, 2, 3** Default value: **1** (>>> 7.5.1.3 "Reading out and displaying debug information " Page 29)
setMinimumTrajecto-ryExecutionTime(…)	Minimum execution time for a trajectory (type: double, unit: seconds) ■ **≥ 0** Default value: **0.01** Sets the minimum execution time for a trajectory. Changes the path.
activateVelocityPlan-ning(…)	Activates the velocity planning once an end point has been reached (type: Boolean) Default value: **false**
activateWaitFor-PauseDuringSetDes-tination(…)	Activates the pausing of a motion in T1/T2 as soon as the Start key or an enabling switch has been released (type: Boolean) Default value: **true** If this setting is deactivated, the application continues running and is responsible for pausing and starting up again.
changeControlMode-Settings(…)	Modification of controller parameters during impedance control. (>>> 7.10.3 "Changing controller parameters during the runtime" Page 38)

9.1.3 Setting a destination

Method	Description
setDestination(…)	Setting of a new axis-specific destination (>>> 7.5.2.1 "Specifying an axis-specific end position" Page 30)
	Setting of a new axis-specific destination and the corresponding destination velocity of the individual axes (>>> 7.5.2.2 "Specifying an axis-specific end position with a target velocity" Page 30)
	Setting of a new Cartesian destination (>>> 7.5.3.1 "Setting a Cartesian destination in the robot base coordinate system" Page 31)
	Setting of a new Cartesian destination and a reference coordinate system in which the servo motion is to be carried out (>>> 7.5.3.2 "Specifying a Cartesian destination in the reference coordinate system" Page 31)

9.2 DirectServo command reference

9.2.1 Motion parameters

Method	Description
setJointVelocity-Rel(…)	Axis-specific relative velocity (type: double, unit: %) ■ **0.0 … 1.0** Default value: **1** Refers to the maximum value of the axis velocity in the machine data. **Note:** Further information on the programming of axis-specific motion parameters is contained in the operating and programming instructions for the System Software.
setJointAcceleration-Rel(…)	**Note:** This method has no effect on the DirectServo motion type.
setMinimumTrajecto-ryExecutionTime(…)	Minimum execution time for a trajectory (type: double, unit: seconds) ■ **≥ 0** Default value: **0.01** Sets the minimum execution time for a trajectory. Changes the path.
setSpeedTime-outAfterGoal-Reach(…)	Timeout for the velocity profile after the destination is reached (type: double, unit: seconds) ■ **≥ 0** Default value: **0** Sets the timeout, specifying the point up to which the interpolator should continue to plan with a velocity profile.
setTimeoutAfterGoal-Reach(…)	Timeout for the motion command after the destination has been reached (type: double, unit: seconds) ■ **≥ 0** Default value: **120** If no new destination is set after the specified time has elapsed, the motion command is terminated.
setMode(…)	Setting of the motion type. **Note:** Further information on programming is contained in the operating and programming instructions for the System Software.

9.2.2 Runtime parameters

Method	Description
setGoalReachedE-ventHandler(…)	Setting of the event handler when the destination is reached. The parameter must be an implementation of the type IServoOnGoal-ReachedEvent.
setDetailedOutput(…)	Setting of the amount of detail of the debug information ■ **1, 2, 3** Default value: **1** (>>> 7.5.1.3 "Reading out and displaying debug information " Page 29)

Method	Description
setMinimumTrajectoryExecutionTime(…)	Minimum execution time for a trajectory (type: double, unit: seconds) ■ **≥ 0** Default value: **0.01** Sets the minimum execution time for a trajectory. Changes the path.
activateWaitForPauseDuringSetDestination(…)	Activates the pausing of a motion in T1/T2 as soon as the Start key or an enabling switch has been released (type: Boolean) Default value: **true** If this setting is deactivated, the application continues running and is responsible for pausing and starting up again.
changeControlModeSettings(…)	Modification of controller parameters during impedance control (>>> 7.10.3 "Changing controller parameters during the runtime" Page 38)

9.2.3 Setting a destination

Method	Description
setDestination(…)	Setting of a new axis-specific destination (>>> 7.5.2.1 "Specifying an axis-specific end position" Page 30)
	Setting of a new Cartesian destination (>>> 7.5.3.1 "Setting a Cartesian destination in the robot base coordinate system" Page 31)
	Setting of a new Cartesian destination and a reference coordinate system in which the servo motion is to be carried out (>>> 7.5.3.2 "Specifying a Cartesian destination in the reference coordinate system" Page 31)

10 Exception handling routines

10.1 Example of an exception handling routine

Description

The motion types SmartServo and DirectServo 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(initialPosition);
    _toolAttachedToLBR.getDefaultMotionFrame().
     moveAsync(aSmartServoMotion);
    // Fetch the Runtime of the Motion part
    IServoRuntime theServoRuntime = aSmartServoMotion.getRuntime();
    // do repetitive loop
    do
    {
        // Do some computations
        //
        theServoRuntime.setDestination(destination);
    } while (notFinished);
}
catch (Exception e)
{
    getLogger().error(e.getMessage());
    throw(new Exception(e));
}
```

10.2 Overview of exceptions – runtime error messages

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

Exception / Source	Message	Cause / remedy
Type: IllegalStateException Source: ■ IServoMotion.getRuntime()	“waitForTransferred timeout to become activated in realtime-system”	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.
Type: IllegalStateException Source: ■ IServoMotion.getRuntime() ■ IServoRuntime.stopMotion()	“waitForTransferred called prematurely - RealtimeMotion is not active”	The servo motion has not yet been activated with moveAsync(…). Check the order of the programming.
Type: CommandInvalidException Source: ■ IServoRuntime.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. (>>> 10.3 "CommandInvalidException error causes" Page 66)

10.3 CommandInvalidException error causes

errorReason	Cause	Remedy
“tracking error”	The difference between the setpoint position and the actual position is too great. The 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 loads on the robot. ■ 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(…).

errorReason	Cause	Remedy
“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.
“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.

errorReason	Cause	Remedy
"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.

11 Troubleshooting

11.1 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.

(>>> 7.5.1.1 "Polling time stamps" Page 28)

```
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.

11.2 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.
  (>>> 11.1 "TCP/IP traffic from Windows is too slow" Page 69)

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.

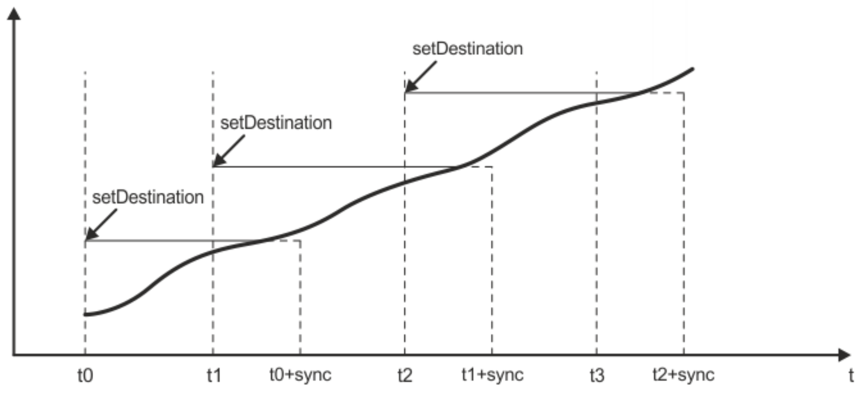


Fig. 11-1: Servo motion without the seconds hand effect

  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.

  (>>> 7.9 "Time recording" Page 36)
- **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.

11.3 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.

11.4 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.

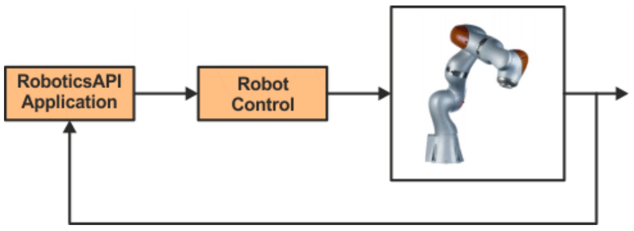


Fig. 11-2: Application is part of the control loop

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

(>>> 7.11 "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 overrideJoin-tAcceleration(…).
- Increase the maximum jerk used for planning with setJointJerk(…).

(>>> 7.12.1 "SmartServo vs. DirectServo – comparison of axis positions over time" Page 43)

(>>> 7.12.2 "SmartServo vs. DirectServo – comparison of velocity, accelera-tion, jerk" Page 50)

12 KUKA Service

12.1 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

12.2 KUKA Customer Support

Availability

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

Argentina

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

Australia

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

Belgium	KUKA Automatisering + Robots N.V. Centrum Zuid 1031 3530 Houthalen Belgium Tel. +32 11 516160 Fax +32 11 526794 info@kuka.be www.kuka.be
Brazil	KUKA Roboter do Brasil Ltda. Travessa Claudio Armando, nº 171 Bloco 5 - Galpões 51/52 Bairro Assunção CEP 09861-7630 São Bernardo do Campo - SP Brazil Tel. +55 11 4942-8299 Fax +55 11 2201-7883 info@kuka-roboter.com.br www.kuka-roboter.com.br
Chile	Robotec S.A. (Agency) Santiago de Chile Chile Tel. +56 2 331-5951 Fax +56 2 331-5952 robotec@robotec.cl www.robotec.cl
China	KUKA Robotics China Co., Ltd. No. 889 Kungang Road Xiaokunshan Town Songjiang District 201614 Shanghai P. R. China Tel. +86 21 5707 2688 Fax +86 21 5707 2603 info@kuka-robotics.cn www.kuka-robotics.com
Germany	KUKA Roboter GmbH Zugspitzstr. 140 86165 Augsburg Germany Tel. +49 821 797-4000 Fax +49 821 797-1616 info@kuka-roboter.de www.kuka-roboter.de

France	KUKA Automatisme + Robotique SAS Techvallée 6, Avenue du Parc 91140 Villebon S/Yvette France Tel. +33 1 6931660-0 Fax +33 1 6931660-1 commercial@kuka.fr www.kuka.fr
India	KUKA Robotics India Pvt. Ltd. Office Number-7, German Centre, Level 12, Building No. - 9B DLF Cyber City Phase III 122 002 Gurgaon Haryana India Tel. +91 124 4635774 Fax +91 124 4635773 info@kuka.in www.kuka.in
Italy	KUKA Roboter Italia S.p.A. Via Pavia 9/a - int.6 10098 Rivoli (TO) Italy Tel. +39 011 959-5013 Fax +39 011 959-5141 kuka@kuka.it www.kuka.it
Japan	KUKA Robotics Japan K.K. YBP Technical Center 134 Godo-cho, Hodogaya-ku Yokohama, Kanagawa 240 0005 Japan Tel. +81 45 744 7691 Fax +81 45 744 7696 info@kuka.co.jp
Canada	KUKA Robotics Canada Ltd. 6710 Maritz Drive - Unit 4 Mississauga L5W 0A1 Ontario Canada Tel. +1 905 670-8600 Fax +1 905 670-8604 info@kukarobotics.com www.kuka-robotics.com/canada

Korea	KUKA Robotics Korea Co. Ltd. RIT Center 306, Gyeonggi Technopark 1271-11 Sa 3-dong, Sangnok-gu Ansan City, Gyeonggi Do 426-901 Korea Tel. +82 31 501-1451 Fax +82 31 501-1461 info@kukakorea.com
Malaysia	KUKA Robot Automation (M) Sdn Bhd South East Asia Regional Office No. 7, Jalan TPP 6/6 Taman Perindustrian Puchong 47100 Puchong Selangor Malaysia Tel. +60 (03) 8063-1792 Fax +60 (03) 8060-7386 info@kuka.com.my
Mexico	KUKA de México S. de R.L. de C.V. Progreso #8 Col. Centro Industrial Puente de Vigas Tlalnepantla de Baz 54020 Estado de México Mexico Tel. +52 55 5203-8407 Fax +52 55 5203-8148 info@kuka.com.mx www.kuka-robotics.com/mexico
Norway	KUKA Sveiseanlegg + Roboter Sentrumsvegen 5 2867 Hov Norway Tel. +47 61 18 91 30 Fax +47 61 18 62 00 info@kuka.no
Austria	KUKA Roboter CEE GmbH Gruberstraße 2-4 4020 Linz Austria Tel. +43 7 32 78 47 52 Fax +43 7 32 79 38 80 office@kuka-roboter.at www.kuka.at

Poland	KUKA Roboter Austria GmbH Spółka z ograniczoną odpowiedzialnością Oddział w Polsce Ul. Porcelanowa 10 40-246 Katowice Poland Tel. +48 327 30 32 13 or -14 Fax +48 327 30 32 26 ServicePL@kuka-roboter.de
Portugal	KUKA Sistemas de Automatización S.A. Rua do Alto da Guerra n° 50 Armazém 04 2910 011 Setúbal Portugal Tel. +351 265 729780 Fax +351 265 729782 kuka@mail.telepac.pt
Russia	KUKA Robotics RUS Werbnaja ul. 8A 107143 Moskau Russia Tel. +7 495 781-31-20 Fax +7 495 781-31-19 info@kuka-robotics.ru www.kuka-robotics.ru
Sweden	KUKA Svetsanläggningar + Robotar AB A. Odhners gata 15 421 30 Västra Frölunda Sweden Tel. +46 31 7266-200 Fax +46 31 7266-201 info@kuka.se
Switzerland	KUKA Roboter Schweiz AG Industriestr. 9 5432 Neuenhof Switzerland Tel. +41 44 74490-90 Fax +41 44 74490-91 info@kuka-roboter.ch www.kuka-roboter.ch

Spain	KUKA Robots IBÉRICA, S.A. Pol. Industrial Torrent de la Pastera Carrer del Bages s/n 08800 Vilanova i la Geltrú (Barcelona) Spain Tel. +34 93 8142-353 Fax +34 93 8142-950 Comercial@kuka-e.com www.kuka-e.com
South Africa	Jendamark Automation LTD (Agency) 76a York Road North End 6000 Port Elizabeth South Africa Tel. +27 41 391 4700 Fax +27 41 373 3869 www.jendamark.co.za
Taiwan	KUKA Robot Automation Taiwan Co., Ltd. No. 249 Pujong Road Jungli City, Taoyuan County 320 Taiwan, R. O. C. Tel. +886 3 4331988 Fax +886 3 4331948 info@kuka.com.tw www.kuka.com.tw
Thailand	KUKA Robot Automation (M)SdnBhd Thailand Office c/o Maccall System Co. Ltd. 49/9-10 Soi Kingkaew 30 Kingkaew Road Tt. Rachatheva, A. Bangpli Samutprakarn 10540 Thailand Tel. +66 2 7502737 Fax +66 2 6612355 atika@ji-net.com www.kuka-roboter.de
Czech Republic	KUKA Roboter Austria GmbH Organisation Tschechien und Slowakei Sezemická 2757/2 193 00 Praha Horní Počernice Czech Republic Tel. +420 22 62 12 27 2 Fax +420 22 62 12 27 0 support@kuka.cz

Hungary	KUKA Robotics Hungaria Kft. Fö út 140 2335 Taksony Hungary Tel. +36 24 501609 Fax +36 24 477031 info@kuka-robotics.hu
USA	KUKA Robotics Corporation 51870 Shelby Parkway Shelby Township 48315-1787 Michigan USA Tel. +1 866 873-5852 Fax +1 866 329-5852 info@kukarobotics.com www.kukarobotics.com
UK	KUKA Automation + Robotics Hereward Rise Halesowen B62 8AN UK Tel. +44 121 585-0800 Fax +44 121 585-0900 sales@kuka.co.uk

Index

A
activateVelocityPlanning(…) 62
activateWaitForPauseDuringSetDestination(…) 62, 64
API 8
Application cycle time, measuring 36
Application examples 59
Axis-specific actual position, reading out 30
Axis-specific end position, specification 30
Axis-specific end position, specifying 30
Axis-specific interface 30

C
Cartesian actual position, reading out 32, 33
Cartesian destination, setting 31
Cartesian destination, specification 31
Cartesian interface 31
Catalog, Simple Tutorials for DirectServo 60
Catalog, Simple Tutorials for SmartServo 59
changeControlModeSettings(…) 62, 64
Command reference 61
CommandInvalidException, error causes 66
Controller parameters, changing 38

D
Debug information, reading out and displaying 29
Destination reached 34
DirectServo, characteristics 11
DirectServo, command reference 63
DirectServo, destination setting 64
DirectServo, installing 17
DirectServo, motion parameters 63
DirectServo, runtime parameters 63
DirectServoSampleInteractionControl 60
DirectServoSampleSimpleCartesian 60
DirectServoSampleSimpleJointMotion 60
Documentation, industrial robot 7

E
Exception 8
Exception handling routines 65
Exceptions, overview 65

F
Feedback effect 40
Feedback effect, troubleshooting 71
Frames, synchronizing 23

H
Handling the real-time system 28
Hardware 17
HRC 8

I
Impedance control 37
Impedance control cannot be activated 71
Impedance control, activating 38
Installation 17
Interfaces 11
Introduction 7
IServoRuntime class 28

K
Knowledge, required 9
KUKA Customer Support 73
KUKA Sunrise.Connectivity, overview 11
KUKA Sunrise.Workbench 8
KUKA timers 36
KUKA Sunrise Cabinet 8
KUKA Sunrise.OS 8

L
LBR iiwa 8
Live view 13
Live view, starting 23
Load model, validating 37

M
Misuse 9

O
onGoalReachedEvent(…) 34
overrideJointAcceleration(…) 61

P
Path parameters, changing 35
Precision interpolator, polling 29
Product description 11
Programming 25
PTP, characteristics 11
Purpose 9

R
Reference coordinate system 31, 33
Rendering interface, configuring 22
Rendering interface, installing 18
Rendering interface, integration of V-REP PRO 18
Rendering interface, operation 21
Rendering VREP (view) 21
Robot base coordinate system 31, 32
RoboticsAPI, DirectServo and SmartServo 12
Runtime error messages 65

S
Safety 15
Safety instructions 7
Second hand effect, troubleshooting 70
Seconds hand effect 27
Service, KUKA Roboter 73
Servo motion in an application context 25
Servo motion, terminating 33
setAmplitude(…) 38
setBias(…) 38
setDamping(…) 38

setDestination(…) 62, 64
setDestination(…), destination setting, cyclical 26
setDestination(…), destination setting, simple 26
setDetailedOutput(…) 62, 63
setFallTime(…) 38
setForceLimit(…) 38
setFrequency(…) 38
setGoalReachedEventHandler(…) 34, 62, 63
setHoldTime(…) 38
setJointAccelerationRel(…) 61, 63
setJointVelocityRel(…) 61, 63
setMaxControlForce(…) 39
setMaxPathDeviation(…) 39
setMinimumTrajectoryExecutionTime(…) 61, 62, 63, 64
setMode(…) 61, 63
setNullSpaceDamping(…) 38
setNullSpaceStiffness(…) 38
setPhaseDeg(…) 38
setPositionLimit(…) 38
setRiseTime(…) 38
setSpeedTimeoutAfterGoalReach(…) 61, 63
setStiffness(…) 38
setTimeoutAfterGoalReach(…) 61, 63
SmartServo vs. DirectServo 43, 50
SmartServo, characteristics 11
SmartServo, command reference 61
SmartServo, destination setting 62
SmartServo, installing 17
SmartServo, motion parameters 61
SmartServo, runtime parameters 62
SmartServoSampleInteractionControl 59
SmartServoSampleSimpleCartesian 59
SmartServoSampleSimpleJointMotion 59
Software 17
Step response of the systems 42
Step response, SmartServo vs. DirectServo, A1 44, 52
Step response, SmartServo vs. DirectServo, A2 45, 53
Step response, SmartServo vs. DirectServo, A3 46, 54
Step response, SmartServo vs. DirectServo, A4 47, 55
Step response, SmartServo vs. DirectServo, A5 48, 56
Step response, SmartServo vs. DirectServo, A6 49, 57
Step response, SmartServo vs. DirectServo, A7 50, 58
stopMotion() 33
Sunrise.Connectivity DirectServo 11
Sunrise.Connectivity Rendering (option) 11, 13
Sunrise.Connectivity SmartServo 11
Support request 73
System reaction time 8
System requirements 17

T
Target group 9
Target velocity, specification 30
TCP/IP traffic, slow 69
Terms used 8
Time recording 36
Time stamps, polling 28
Timeouts 34
Trademarks 8
Training 9
Transition between control types 39
Troubleshooting 69

U
Use, intended 9
User timer 36

V
V-REP PRO 8, 13

W
Warnings 7