



# Operating manual

## Tracking and searching with optical sensors

Trace back information:  
Workspace R17-1 version a7  
Checked in 2017-03-22  
Skribenta version 5.1.011

# **Operating manual**

## **Tracking and searching with optical sensors**

**RobotWare 6.05**

**Document ID: 3HAC055269-001**

**Revision: A**

The information in this manual is subject to change without notice and should not be construed as a commitment by ABB. ABB assumes no responsibility for any errors that may appear in this manual.

Except as may be expressly stated anywhere in this manual, nothing herein shall be construed as any kind of guarantee or warranty by ABB for losses, damages to persons or property, fitness for a specific purpose or the like.

In no event shall ABB be liable for incidental or consequential damages arising from use of this manual and products described herein.

This manual and parts thereof must not be reproduced or copied without ABB's written permission.

Keep for future reference.

Additional copies of this manual may be obtained from ABB.

Original instructions.

© Copyright 2016-2017 ABB. All rights reserved.

ABB AB, Robotics  
Robotics and Motion  
Se-721 68 Västerås  
Sweden

# Table of contents

Overview of this manual .....	7
<b>1 Introduction to optical tracking</b> .....	<b>9</b>
1.1 About optical tracking .....	9
1.2 Sensors .....	11
1.3 RobotWare options .....	13
1.3.1 About RobotWare options .....	13
1.3.2 Introduction to Sensor Interface .....	14
1.3.3 Introduction to Optical Tracking CAP .....	15
1.3.4 Introduction to Optical Tracking Arc .....	16
1.3.5 Introduction to Externally Guided Motion (EGM) .....	17
<b>2 Installation</b> .....	<b>19</b>
2.1 Installing the sensor .....	19
2.2 Software installation .....	22
<b>3 Configuration</b> .....	<b>25</b>
3.1 Introduction .....	25
3.2 Configuring Sensor Interface .....	27
3.2.1 About the sensors .....	27
3.2.2 Configuring sensors on serial channels .....	28
3.2.3 Configuring sensors on Ethernet channels .....	29
3.3 Configuring CAP .....	30
3.4 Configuring Optical Tracking Arc .....	31
3.5 Configuring EGM .....	33
<b>4 Calibration</b> .....	<b>35</b>
<b>5 Programming</b> .....	<b>37</b>
<b>6 Tracking</b> .....	<b>39</b>
6.1 Sensor functionality .....	39
6.2 Tracking with CAP .....	40
6.3 Tracking with Arc .....	41
6.4 Tracking with Arc MultiPass .....	42
6.5 Tracking with EGM .....	43
6.6 Pre Process Tracking .....	45
<b>7 Searching</b> .....	<b>49</b>
<b>8 Adaptivity with optical sensors</b> .....	<b>51</b>
<b>9 Reference information</b> .....	<b>53</b>
9.1 Relationship between coordinate systems .....	53
9.2 Protocols .....	54
9.2.1 Introduction .....	54
9.2.2 LTAPP .....	55
9.2.3 LTAPPTCP .....	61
9.2.4 Sockdev .....	62
9.2.5 The EGM sensor protocol .....	63
9.2.6 Using EGM Path Correction with different protocol types .....	67
<b>10 RAPID reference</b> .....	<b>71</b>
10.1 RAPID components for optical tracking with Sensor Interface .....	71
10.2 RAPID components for optical tracking with CAP .....	74

## Table of contents

---

10.3	RAPID components for optical tracking with Arc .....	75
10.4	RAPID components for optical tracking with EGM .....	76
10.5	RAPID toolbox for searching .....	78
10.5.1	Instructions .....	78
10.5.1.1	ArcSearchLStart - Searching with optical sensors for arc welding .....	78
10.5.1.2	CapSensorScan - Searching with a laser tracking sensor .....	82
10.5.1.3	OptSearch_1D - Searching with optical sensors for Arc .....	85
10.5.2	Data types .....	88
10.5.2.1	optscandata - Scan data .....	88
<b>11</b>	<b>System parameters .....</b>	<b>91</b>
11.1	System parameters for Arc .....	91
11.2	System parameters for EGM .....	94
<b>Index</b>		<b>95</b>

---

# Overview of this manual

## About this manual

This manual contains instructions for installing and configuring a RobotWare system using different types of optical sensors for tracking or searching.

## Prerequisites

The installation/maintenance/repair engineer working with an ABB Robot must be trained by ABB and have the knowledge required for mechanical and electrical installation/maintenance/repair work.

## References

References	Document ID
<i>Application manual - Arc and Arc Sensor</i>	3HAC050988-001
<i>Application manual - Continuous Application Platform</i>	3HAC050990-001
<i>Application manual - Controller software IRC5</i>	3HAC050798-001
<i>Operating Manual - ArcWelding PowerPac</i>	3HAC028931-001
<i>Operating manual - IRC5 with FlexPendant</i>	3HAC050941-001
<i>Operating manual - RobotStudio</i>	3HAC032104-001
<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>	3HAC050917-001
<i>Technical reference manual - RAPID overview</i>	3HAC050947-001
<i>Technical reference manual - System parameters</i>	3HAC050948-001



### Note

The document numbers that are listed for software documents are valid for RobotWare 6. Equivalent documents are available for RobotWare 5.

## Revisions

Revision	Description
-	Published with RobotWare 6.04.01. First release.
A	Published with RobotWare 6.05. <ul style="list-style-type: none"><li>Added section <a href="#">Pre Process Tracking on page 45</a>.</li><li>Minor corrections.</li></ul>

This page is intentionally left blank



# 1 Introduction to optical tracking

## 1.1 About optical tracking

---

### Introduction

Optical sensors can be used in ABB robot systems to:

- search geometrical properties on an object
- track along a seam

---

### Hardware prerequisites

- Complete robot system with manipulator, IRC5 controller, and a FlexPendant of type SxTPU3.
- An optical sensor that is supported by RobotWare, see Limitations.
- A calibration plate for sensor calibration.
- PC.
- Ethernet cable for connecting the PC to the IRC5 controller.

---

### Software prerequisites

RobotWare 6.0 or later, with at least one of the following options enabled:

- *Sensor Interface*
- *Optical Tracking CAP*
- *Optical Tracking Arc*
- *Externally Guided Motion (EGM)*



#### Note

Some functionality, for example *Sensor Interface*, is embedded into the other options and does not have to be ordered separately, see [RobotWare options on page 13](#).

For more information about RobotWare options, see *Product specification - Controller software IRC5* and *Application manual - Controller software IRC5*.

---

### Limitations

The following limitations apply:

#### Supported optical sensors

The hardware is acquired from optical tracking sensor suppliers, for example ServoRobot, Scansonic, Meta/Scout, etc.

The sensors have to support one of the application protocols supported by the ABB controller software.

#### Supported transport protocols

The following protocols are supported by RobotWare:

- RTP1: is a protocol for serial communication interfaces.

*Continues on next page*

# 1 Introduction to optical tracking

---

## 1.1 About optical tracking

*Continued*

Note that the serial interface is only available if the controller is equipped with a serial interface card, see [Installation on page 19](#).

- TCP/IP: for socket communication over Ethernet.
- UDP: for UDP-socket communication over Ethernet.

### Supported application protocols

The following application protocols are supported by RobotWare:

- LTAPP: is a protocol for serial communication interfaces. Note that the serial interface is only available if the controller is equipped with a serial interface card, see [Installation on page 19](#).
- LTAPPTCP: This protocol is an encapsulated LTAPP for Ethernet.
- SOCKDEV: This is a SERVO-ROBOT proprietary protocol (RoboCom Light) used by that company's sensors. This protocol is available for Ethernet only. Only functionality for tracking and data according to the constants listed in section [Constants on page 71](#) are implemented.
- UdpUc: This protocol is based on the open source Google Protobuf protocol. It is available for EGM only. A description of the protocol is found in the section EGM in *Application manual - Controller software IRC5*.

### High accuracy

For applications with a high demand of accuracy it is recommended to use robots with the option *Absolute Accuracy* together with the optical sensors.

### MultiMove

Up to two robots using optical sensors for tracking simultaneously are supported in a MultiMove system.



#### Note

It may be possible, but not supported by ABB, to install and run more than two tracking robots in the same system. Such an application must be individually tested by the customer under the customer's own responsibility.

---

## Glossary

Term	Explanation
EGM	Externally Guided Motion
LTAPP	Laser Tracker Application Protocol
CAP	Continuous Application Platform
RW	RobotWare
TCP	Tool Center Point
LTC	Laser Tracker Calibration

## 1.2 Sensors

---

### Introduction

Two different types of laser sensors are supported:

- 2.5D sensors
- 6D sensors

To use the sensor for searching the sensor can be mounted on the axis 6 of the robot without restrictions. However, when using the sensor for tracking the sensor mounting possibilities are limited as described below.

To use the sensor for tracking the sensor must be mounted on the robot in front of the tool/TCP on the path that is to be corrected. The sensor shall be mounted in such a way that, on average, the tracked path is at the optimal distance for the used sensor.

---

### Sensor look ahead

The sensor look ahead depends on the robot speed and the sensor frequency. A typical distance is 30 mm or more.

- Both high TCP speed and low sensor frequency give a lower limit for the look ahead. This lower limit is also influenced by internal (buffering) delays in the controller. The delay is about 0.5 s, which gives a starting point, but it is necessary to adjust the sensor mounting in each specific installation to be sure the tracking works correctly.

$$speed * ((1 / sensor\ frequency) + delay) < look\ ahead$$

- With low TCP speed and/or high sensor frequency and/or large look ahead the robot controller's internal measurement buffer (max. 200 measurements) might give an upper limit for the look ahead.

$$look\ ahead < ((200 * speed) / sensor\ frequency)$$

*Continues on next page*

# 1 Introduction to optical tracking

---

## 1.2 Sensors

*Continued*

---

### 2.5D sensors

The 2.5D sensor is the most common type. These sensors have one laser stripe to measure, and can measure y and z and the orientation around x.

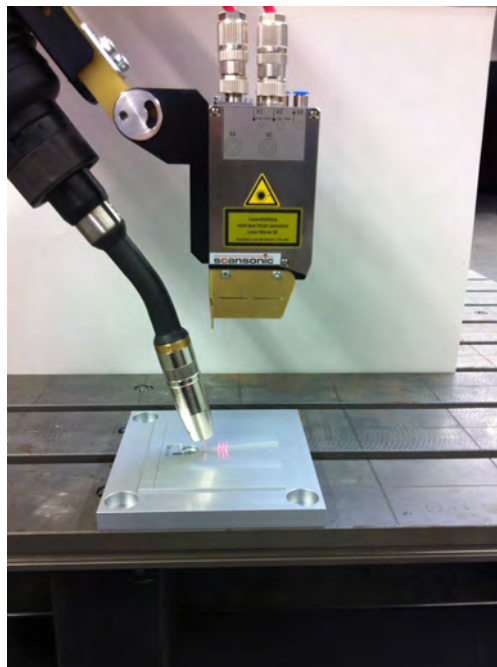


xx1500001675

---

### 6D sensors

The 6D sensors can measure both position (x, y, z) and orientation (three Euler angles).



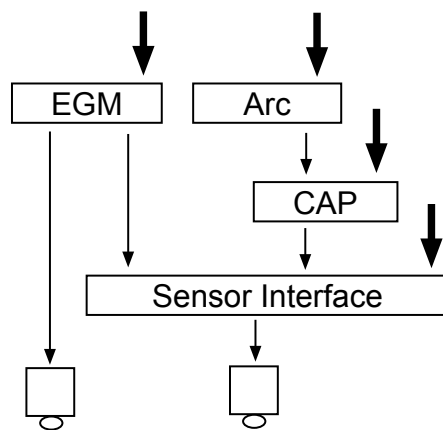
xx1500001676

## 1.3 RobotWare options

### 1.3.1 About RobotWare options

#### Introduction

ABB offers a set of RobotWare options for communicating with optical sensor on different abstraction levels according to the below picture. When selecting a lower level RobotWare option, then more functionality must be developed by the end user.



xx1500001684

The most basic and low level functionality for communicating with optical sensor is the RobotWare option *Sensor Interface*.

The functionality of *Sensor Interface* is embedded into the RobotWare option *Optical Tracking CAP*, and the functionality of *Optical Tracking CAP* is embedded into *Optical Tracking Arc*.

The RobotWare option *Sensor Interface* is not included in *Externally Guided Motion (EGM)*, but may be required depending on the desired communication protocol.

### 1.3.2 Introduction to Sensor Interface

---

#### Purpose

The option Sensor Interface is used for communication with external sensors via a serial or Ethernet channel.

The sensor may be accessed using a package of RAPID instructions that provide the ability to read and write raw sensor data.

An interrupt feature allows subscriptions on changes in sensor data.



#### Tip

The communication provided by Sensor Interface is integrated in arc welding instructions for seam tracking and adaptive control of process parameters. These instructions handle communication and corrections for you, whereas with Sensor Interface you handle this yourself. For more information, see *Application manual - Arc and Arc Sensor* and *Application manual - Continuous Application Platform*.

#### What is included

The RobotWare option Sensor Interface gives you access to:

- ABB supported sensor protocols.
- Instruction used to connect to a sensor device: `SenDevice`.
- Instruction used to set up interrupt, based on input from the `sensor:IVarValue`.
- Instruction used to write to a sensor: `WriteVar`.
- Function for reading from a sensor: `ReadVar`.
- Laser Tracker Calibration (LTC) functionality for optical sensor calibration.

### 1.3.3 Introduction to Optical Tracking CAP

---

#### Purpose

The option *Optical Tracking CAP* is used to integrate path correction with CapL/CapC instructions. These instructions handle communication and correction.

---

#### What is included

The RobotWare option *Optical Tracking CAP* gives access to:

- All functionality from the option *Sensor Interface*
- Instruction used to connect to an optical tracking sensor: CapLATRSetup
- Switch to enable tracking in CapL and CapC: \Track
- Data type to specify how the tracking should be performed: captrackdata

# 1 Introduction to optical tracking

---

## 1.3.4 Introduction to Optical Tracking Arc

### 1.3.4 Introduction to Optical Tracking Arc

---

#### Purpose

The option *Optical Tracking Arc* is used to integrate seam tracking with the welding instructions `ArcL`, `ArcC`, etc. These instructions handle communication and correction.

---

#### What is included

The RobotWare option *Optical Tracking Arc* gives access to:

- All functionality from the option *Sensor Interface*
- All functionality from the option *Optical Tracking CAP*
- Switch to enable tracking in the welding instructions (`ArcL`, `ArcC`, etc.):  
`\Track`
- Data type to specify how the tracking should be performed: `trackdata`
- System parameters to configure sensor and sensor properties.



### 1.3.5 Introduction to Externally Guided Motion (EGM)

---

#### Purpose

The purpose of *EGM Path Correction* is to use external robot mounted devices to generate path correction data for one or several robots. The robots will be moved along the corrected path, which is the programmed path with added measured corrections.

---

#### What is included

The RobotWare option *EGM* gives access to both path correction and position guidance, but in this context position guidance is irrelevant:

- Instructions to set up, activate, and reset *EGM Path Correction*.
- Instructions to perform *EGM Path Correction* movements.
- A function to retrieve the current *EGM* state.
- System parameters to configure *EGM* and set default values.

#### Limitations

- Only 6-axis robots are supported.
- The external device has to be robot mounted.
- Corrections can only be applied in the path coordinate system.
- Only position correction in y and z can be performed. It is not possible to perform orientation corrections, nor corrections in x (which is the path direction/tangent).
- EGM Path Correction can only be used on 6-axis robots.
- EGM can only be used in RAPID tasks with a robot, i.e. it is not possible to use it in a task that contains only additional axis, i.e. in robtargets there are values in the `pose` portion of the data.
- An EGM path correction has to start and end in a fine point.
- Only one external device can be used for each robot to provide correction data.

**This page is intentionally left blank**

## 2 Installation

### 2.1 Installing the sensor

---

#### Interfaces

The IRC5 controller supports both Ethernet and serial communication with optical sensors.

What to choose depends on the availability on the sensor side. If possible, use an Ethernet based sensor, which gives more flexibility and it is not necessary to install a serial board on the controller.

---

#### Ethernet communication

It is possible to connect the sensor to the following ports on the IRC5 main computer:

- X4 (LAN2)
- X5 (LAN3)
- X2 (SER)

It is recommended to use either X4 or X5 if possible, because this keeps the service port (X2) free and the sensor communication is kept local to the controller.

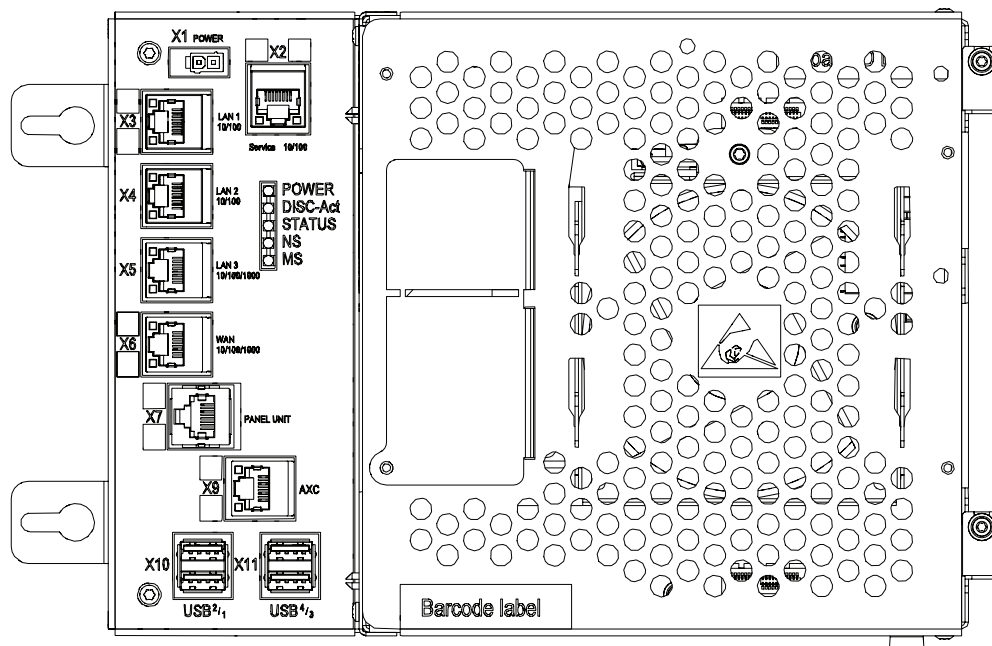
*Continues on next page*

## 2 Installation

### 2.1 Installing the sensor

*Continued*

The following illustration shows an overview of the IRC5 main computer.



xx1300000608

X1	Power supply
X2 (yellow)	Service (connection of PC).
X3 (green)	LAN1 (connection of FlexPendant).
X4	LAN2 (connection of Ethernet based options).
X5	LAN3 (connection of Ethernet based options).
X6	WAN (connection to factory WAN).
X7 (blue)	Panel unit
X9 (red)	Axis computer
X10, X11	USB ports (4 ports)

For more information about the basic configuration of the Ethernet ports, see section *Communication in Technical reference manual - System parameters*.

#### Serial communication

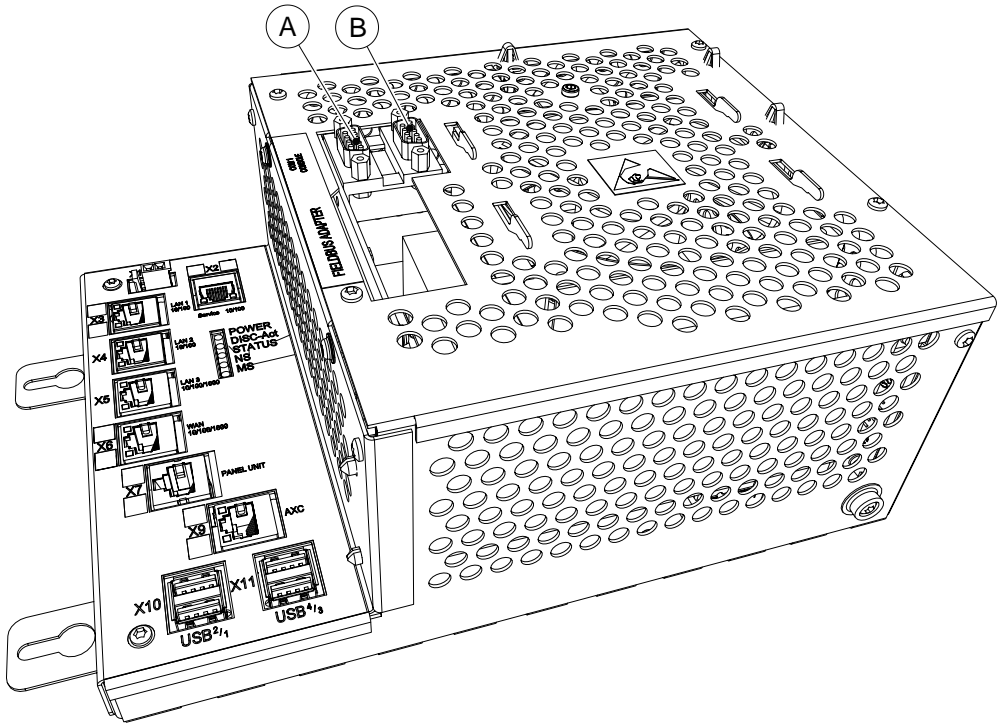
The serial channel is an option. To be able to connect a serial channel to the controller, the main computer needs to be equipped with the expansion board DSQC1003.

The expansion board has one RS232 serial channel, COM1, which can be used to communicate with process equipment.

It is not possible to use more than one sensor with serial interface together with the controller.

*Continues on next page*

The expansion board also enables the connection of a fieldbus adapter. For more information on how to connect a fieldbus adapter, see *Product manual - IRC5*.



xx1300000610

A	COM1
B	CONSOLE



**Note**

The CONSOLE connector is used for debugging purposes only.

## 2 Installation

### 2.2 Software installation

### 2.2 Software installation

#### Installing RobotWare to the IRC5 controller

RobotWare 6.0 or later, with at least one of the following options enabled is required to use optical sensors for tracking or searching:

- *Sensor Interface*
- *Optical Tracking CAP*
- *Optical Tracking Arc*
- *Externally Guided Motion (EGM)*

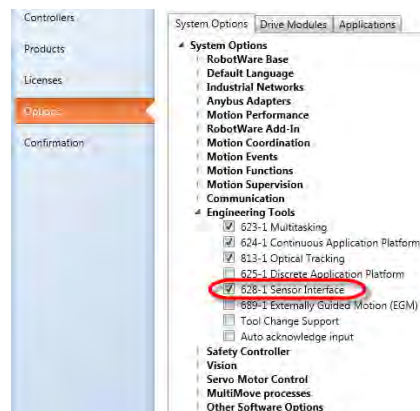
Use the Installation Manager in RobotStudio to configure, build, and download a RobotWare system to the IRC5 controller.

For more information, see *Operating manual - RobotStudio*.

#### Installing Sensor Interface, Optical Tracking CAP, or Externally Guided Motion

Make sure that the corresponding check box is selected in Installation Manager. When the option is installed it must also be configured, see [Configuration on page 25](#).

##### Sensor Interface



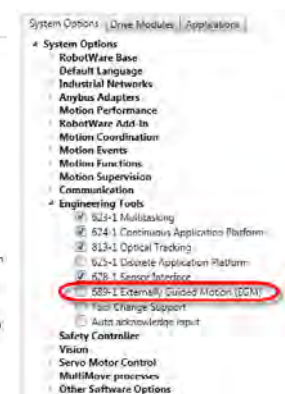
xx1600002013

##### Optical Tracking CAP



xx1600002014

##### Externally Guided Motion



xx1600002015

Continues on next page

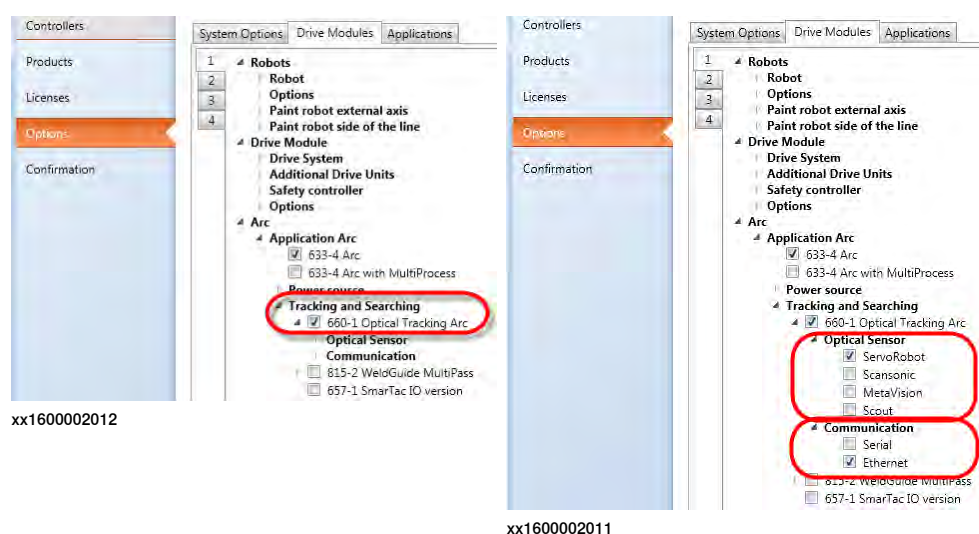
#### Installing Optical Tracking Arc

Make sure that the check box *Optical Tracking Arc* is selected in Installation Manager.

For *Optical Tracking Arc* it is also possible to select the most common sensor configurations during installation. When using a sensor brand that is not available, select the one that comes closest and modify the configuration afterwards.

The *Communication* selection, *Serial* or *Ethernet*, shall always be made.

When the option is installed it must also be configured, see [Configuration on page 25](#).



**This page is intentionally left blank**



## 3 Configuration

### 3.1 Introduction

#### General

The options *Sensor Interface* and *Optical Tracking CAP* do not provide ready to use configurations for the optical sensors. The configuration of communication and sensor must be done manually using RobotStudio or the FlexPendant.

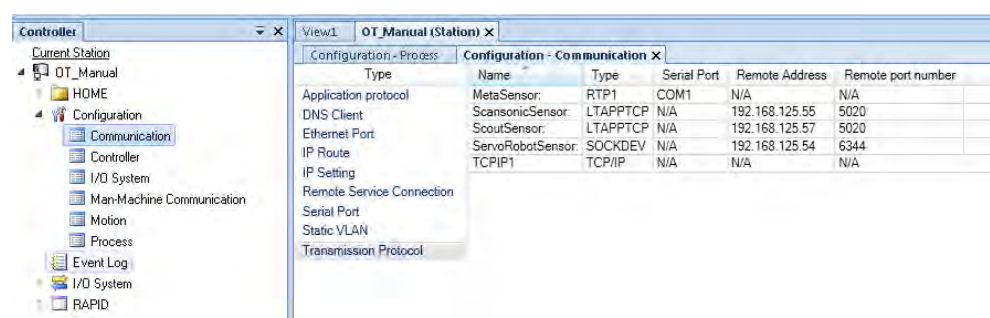
The option *Optical Tracking Arc* includes a default configuration for sensor communication, sensor, and sensor properties, see [Installing Optical Tracking Arc on page 23](#). This default configuration must be modified to match the actual installation, see [Configuring Optical Tracking Arc on page 31](#).

#### Supported sensors and protocols

For information about the supported sensors and protocols see [Limitations on page 9](#).

#### Configuration with RobotStudio

The following picture shows where to find the configuration settings in RobotStudio.



xx1500001677

For more information about the settings see [Configuring sensors on serial channels on page 28](#) or [Configuring sensors on Ethernet channels on page 29](#).

*Continues on next page*

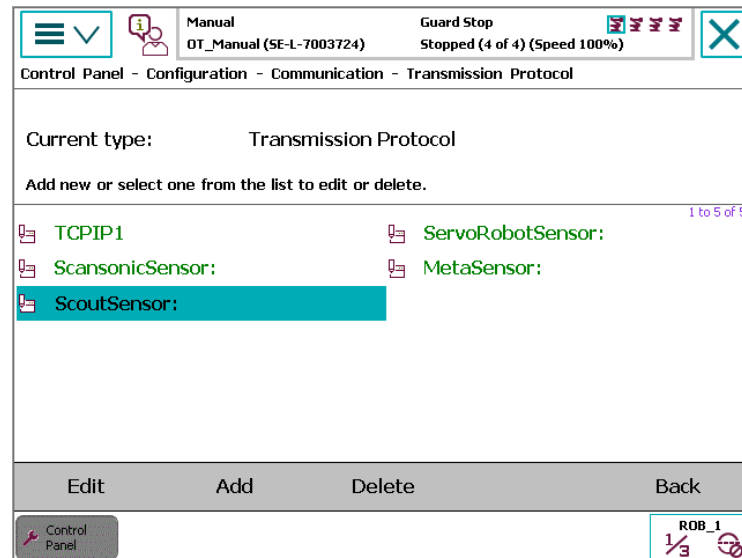
## 3 Configuration

### 3.1 Introduction

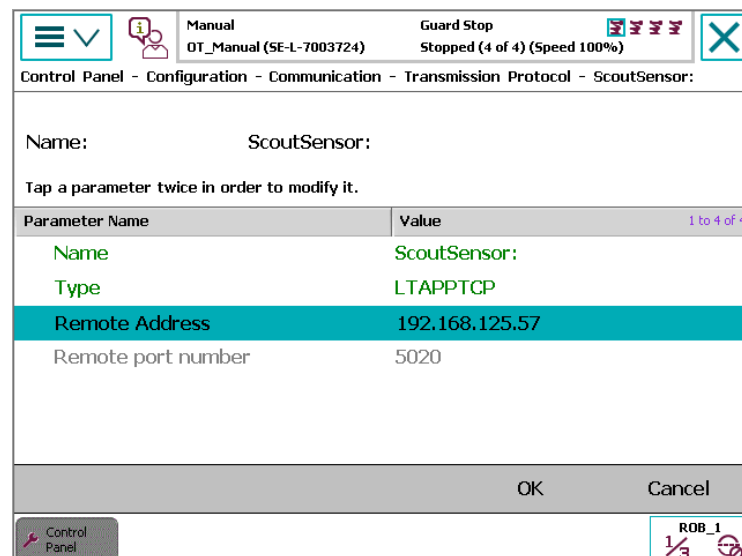
*Continued*

#### Configuration with FlexPendant

The following pictures show where to find the configuration settings on the FlexPendant.



xx1500001678



xx1500001679

For more information about the settings see [Configuring sensors on serial channels on page 28](#) or [Configuring sensors on Ethernet channels on page 29](#).

#### Configuration of the sensor controller

For configuration of the sensor controller, see the sensor supplier's installation and user manual for that sensor.

## 3.2 Configuring Sensor Interface

### 3.2.1 About the sensors

---

#### Supported sensors

Sensor Interface supports:

- Sensors connected via serial channels using the RTP1 protocol. For configuration, see [Configuring sensors on serial channels on page 28](#).
- Sensors connected to Ethernet using the RoboCom Light protocol from Servo-Robot Inc or LTAPP protocol from ABB. For configuration, see [Configuring sensors on Ethernet channels on page 29](#).

## 3 Configuration

### 3.2.2 Configuring sensors on serial channels

### 3.2.2 Configuring sensors on serial channels

#### Overview

Sensor Interface communicates with a maximum of one sensor over serial channels using the RTP1 protocol.

#### System parameters

This is a brief description of the parameters used when configuring a sensor. For more information about the parameters, see *Technical reference manual - System parameters*.

These parameters belong to the type *Transmission Protocol* in the topic *Communication*.

Parameter	Description
Name	The name of the transmission protocol. For a sensor the name must end with ":". For example "laser1:" or "swg:".
Type	The type of transmission protocol. For a sensor using serial channel, it has to be "RTP1".
Serial Port	The name of the serial port that will be used for the sensor. This refers to the parameter <i>Name</i> in the type <i>Serial Port</i> . For information on how to configure a serial port, see <i>Technical reference manual - System parameters</i> .

#### Configuration example

This is an example of how a transmission protocol can be configured for a sensor. We assume that there already is a serial port configured with the name "COM1".

Name	Type	Serial Port
laser1:	RTP1	COM1

### 3.2.3 Configuring sensors on Ethernet channels

#### Overview

Sensor Interface can communicate with a maximum of six sensors over Ethernet channel using the RoboCom Light protocol version E04 (from Servo-Robot Inc) or the LTAPP protocol (from ABB). RoboCom Light is an XML based protocol using TCP/IP.

The sensor acts as a server, the robot controller acts as a client. I.e. the robot controller initiates the connection to the sensor.

RoboCom Light has the default TCP port 6344 on the external sensor side, and LTAPPTCP has the default TCP port 5020.

#### System parameters

This is a brief description of the parameters used when configuring a sensor. For more information about the parameters, see *Technical reference manual - System parameters*.

These parameters belong to the type *Transmission Protocol* in the topic *Communication*.

Parameter	Description
Name	The name of the transmission protocol. For a sensor the name must end with ":". For example "laser1:" or "swg:".
Type	The type of transmission protocol. For RoboCom Light the protocol type SOCKDEV has to be configured, and for LTAPPTCP it is LTAPPTCP.
Serial Port	The name of the serial port that will be used for the sensor. This refers to the parameter <i>Name</i> in the type <i>Serial Port</i> . For information on how to configure a serial port, see <i>Technical reference manual - System parameters</i> . For IP based transmission protocols (i.e. <i>Type</i> has value TCP/IP, SOCKDEV, LTAPPTCP or UDPUC), <i>Serial Port</i> is not used and has the value N/A.
Remote Address	The IP address of the sensor. This refers to the type <i>Remote Address</i> . For information on how to configure Remote Address, see <i>Technical reference manual - System parameters</i> .
Remote Port	Remote Port specifies the port number on the network node identified by Remote Address with which the connection shall be established. The default value for SOCKDEV is 6344, and for LTAPPTCP it is 5020.

#### Configuration examples

These are examples of how a transmission protocol can be configured for a sensor.

Name	Type	Serial Port	Remote Address	Remote Port
laser2:	SOCKDEV	N/A	192.168.125.101	6344
laser3:	LTAPPTCP	N/A	192.168.125.102	5020

## 3 Configuration

---

### 3.3 Configuring CAP

### 3.3 Configuring CAP

---

#### Description

CAP is not configured by using the system parameters, instead CAP has a dynamic configuration by means of the RAPID instruction `CapLATrSetup`. See [RAPID components for optical tracking with CAP on page 74](#) and *Technical reference manual - RAPID Instructions, Functions and Data types*.

#### Configuration example

The device name of the sensor, which is defined in the topic *Communication*, is used as the sensor identifier. For example to use the `ServoRobotSensor:` for robot 1, as it is configured in [Configuration with RobotStudio on page 25](#), the sensor can be setup for usage with CAP in the RAPID task `T_ROB1` in this following way:

```
CapLATrSetup "ServoRobotSensor:", LTC__Scout_1_CalPose,  
            LTC__Scout_1_CalPos  
            \SensorFreq:=20\CorrFilter:=5\MaxBlind:=100\MaxIncCorr:=2;
```

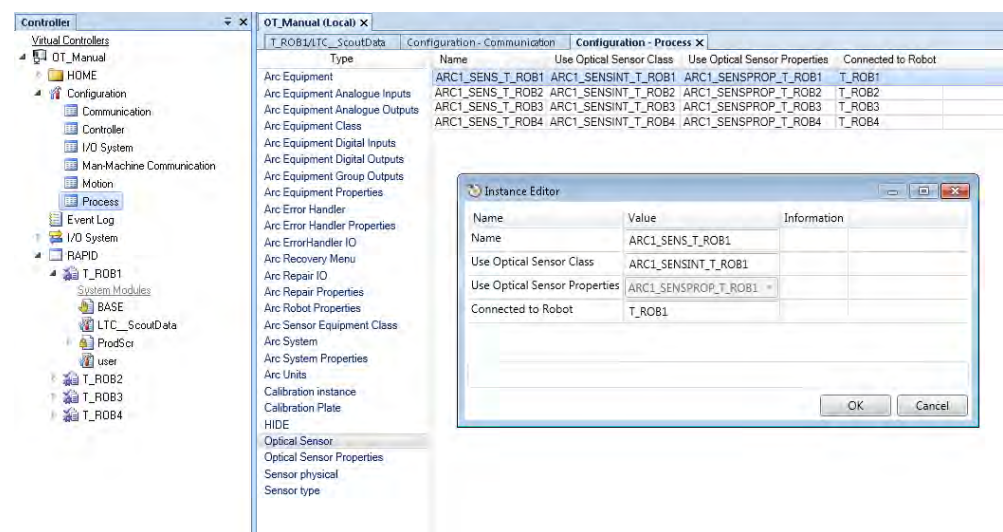
## 3.4 Configuring Optical Tracking Arc

### Introduction

In RobotWare Arc the sensors are configured statically using the configuration types *Optical Sensor* and *Optical Sensor Properties*. This means that the controller needs to be restarted to activate any changes.

The *Optical Sensor* data establishes the connection between the *Optical Sensor Properties*, the *Arc Sensor Equipment Class*, and the RAPID task of the robot that the sensor is to be used with.

In the example below the sensor properties that define the *ServoRobotSensor*: are connected to robot 1 in the controller.



xx1500001680

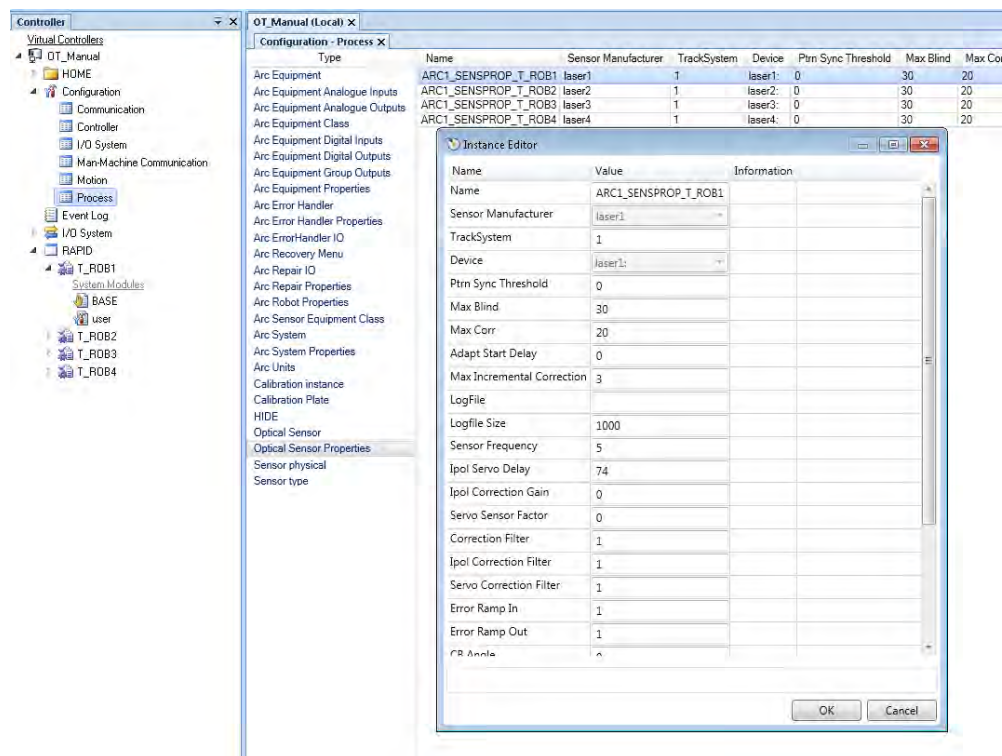
*Continues on next page*

## 3 Configuration

### 3.4 Configuring Optical Tracking Arc

*Continued*

The *Optical Sensor Properties* specify the physical device, where the sensor is connected. That device must have been set up in the “Communication” configuration before. In addition to that it defines behavior and limits for the sensor.



xx1500001681

For a detailed description of the configuration parameters see *Application manual - Arc and Arc Sensor*, section *The type Optical Sensor*.



## 3.5 Configuring EGM

---

### Introduction

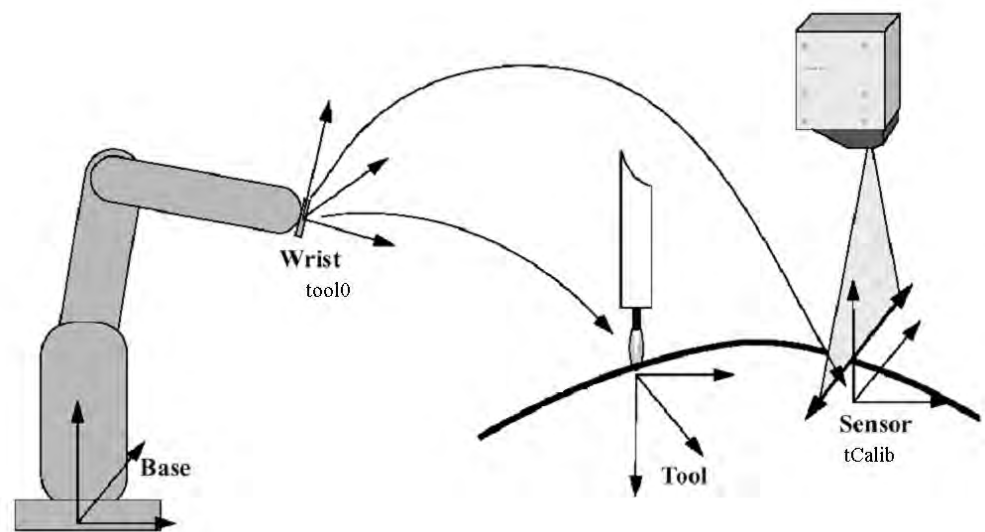
For a detailed description of the configuration see *Application manual - Controller software IRC5*, section *Externally Guided Motion [689-1]*.

This page is intentionally left blank

## 4 Calibration

### Introduction

ABB provides a calibration wizard on the FlexPendant: *Laser Tracker Calibration (LTC)*. By means of this calibration wizard, a relationship is established between the sensor coordinate system and the robot wrist (*tool0*). During LTC calibration the robot tool is used as a measurement tool, which requires that the used tool has to be accurate otherwise the robot tool error is propagated to the sensor calibration. LTC generates a `tooldata` for the sensor, which is stored in a system module.



xx1500001682

For more information, see *Application manual - Laser Tracker Calibration Interface*.

### Calibration plate

To calibrate an optical sensor with the FlexPendant application LTC you need a calibration plate.

The calibration plate shall be placed within the reach of the robot. The best results are achieved if the plate is as near and in the same direction as the actual tracked path. The robot will then most probably have the same axis configuration during tracking as it had during calibration, thus the error due to different axis configuration is eliminated. On the other hand the plate should be out of reach for welding spatter.

You can purchase the ABB plate, which can be used with all supported sensor types. If you are going to use a different calibration plate you have to parametrize it correctly in the *Process* configuration, see *Application manual - Laser Tracker Calibration Interface*.

This page is intentionally left blank

## 5 Programming

### Accessing an optical sensor from RAPID

This functionality requires at least the RobotWare option *Sensor Interface [628-1]*.

The communication with optical sensors is based on the concept of sensor variables. Each sensor supports a number of variables that give access to different functionality in the sensors. Not all predefined variables might be supported by all sensor suppliers.

A list of sensor variables can be found in *Application manual - Controller software IRC5*, section *Sensor Interface [628-1]*.

The RAPID interface to the sensor is independent of the actual sensor protocol, i.e. it is the same for all sensors. But if there is a need to add new variables to get access to special functionality in a sensor, the sensors with the protocol sockdev differ from all the other ones:

- For a sensor using the protocols LTAPP or LTAPPTCP it is sufficient that the sensor supplier implements the new functionality and makes sure that its variable number is unique.
- For a sensor using the protocol sockdev, ABB has to implement the new functionality together with the sensor supplier. I.e. both have to do work.

It is possible to connect to a sensor, control the sensor, and read sensor data using the following RAPID instructions:

RAPID instruction/function	Description
SenDevice	Connect to a sensor.
WriteVar	Change the value of a sensor variable in the sensor.
ReadVar	Read the value of a sensor variable from the sensor.

For more information see *Application manual - Controller software IRC5*, section *Sensor Interface [628-1]*.

This page is intentionally left blank

## 6 Tracking

### 6.1 Sensor functionality

#### Description

Path tracking is fully integrated into the options *Optical Tracking CAP*, *Optical Tracking Arc*, and *Externally Guided Motion (EGM)*.

The sensors that are used for tracking, using *Optical Tracking CAP* or *Optical Tracking Arc*, has to provide the following functionality:

#### LTAPP and LTAPPTCP

LTAPP variable	Unit	Description
LTAPP__AGE	ms	Measurement age
LTAPP__GAP	mm	Gap
LTAPP__X	mm	X-value
LTAPP__Y	mm	Y-value
LTAPP__Z	mm	Z-value
LTAPP__SUSPEND	-	Laser suspend
LTAPP__JOINT	-	Activation of sensor joint
LTAPP__UNIT	-	Resolution selection

#### sockdev

Robo-Com command	Robo-ComLight command
AcknowledgeAll	ack
BeginJointFinding	bjf
EndMode	emo
GetCorrection	gcol
GetError	ger
GetParameter	gpa
GetStatus	gst
GetVision (no filter)	gvil
SensorDisable	sdi

### 6.2 Tracking with CAP

---

#### Description

This functionality requires the RobotWare options *Continuous Application Platform [624-1]* and *Optical Tracking [813-1]*.

To be able to use tracking with CAP it is necessary to set up the sensor with the instruction `CapSetupLATR`. For more information see *Application manual - Continuous Application Platform*.

Tracking with CAP is activated by adding the optional argument `\Track` to the RAPID instructions `CapL` or `CapC`. The `captrackdata` specified with that argument defines the tracking behavior. The component device determines which type of tracker is to be used.

The components `device` and `filter` of `captrackdata` cannot be changed within a sequence of `CapL/CapC` instructions. Those components are set by the first `CapL/CapC` instruction in the sequence. It will not have any effect if the data is different in the remaining `CapL/CapC` instructions in the sequence.

If the `\Track` argument is missing in the first `CapL/CapC` instruction no correction will be applied, regardless if the argument is present in the following instructions in the sequence.



## 6.3 Tracking with Arc

---

### Description

This functionality requires the RobotWare options *Arc [633-4]* and *Optical Tracking Arc [660-1]*.

To be able to use tracking with Arc welding instructions, it is first necessary to configure the sensor in the Arc configuration.

Tracking with Arc is activated by adding the optional argument `\Track` to the arc welding RAPID instructions `ArcLStart`, `ArcL`, `ArcLEnd`, etc. The `trackdata` specified with that argument defines the tracking behavior.

For more information about arc welding instructions see *Application manual - Arc and Arc Sensor*.

#### 6.4 Tracking with Arc MultiPass

---

##### Description

This functionality requires the software options *Arc [633-4]*, *Optical Tracking Arc [660-1]*, and *WeldGuide MultiPass [815-2]*.

The following actions are necessary to activate optical tracking with MultiPass:

- *Weave Sync On* has to be set in the arc system properties.
- *WgLeftSync* and *WgRightSync* have to be set in the optical sensor properties. (*WgTrack* is not used.)
- *Ptrn Sync Threshold* has to be set in the optical sensor properties.
- *store\_path* has to be set to TRUE in `trackdata`.
- Weaving has to be defined and active, and `\SeamName` must be used in the `ArcX` instruction.

For more information about MultiPass see *Operating manual - Seam tracking with Weldguide IV and MultiPass*.

## 6.5 Tracking with EGM

### Description

This functionality requires the RobotWare option *Externally Guided Motion [689-1]*.

This is the general approach to correct a programmed path with EGM Path Correction.

	Action
1	Move the robot to a fine point.
2	Register an EGM client and get an EGM identity. This identity is then used to link setup, activation, movement, deactivation etc. to a certain EGM usage. The EGM state is still <code>EGM_STATE_DISCONNECTED</code> .
3	Call an EGM setup instruction to set up the position data source using signals or UdpUc protocol connection. The EGM state changes to <code>EGM_STATE_CONNECTED</code> .
4	Define the sensor correction frame, which always is a tool frame.
5	Perform the movement itself. Now the EGM state is <code>EGM_STATE_RUNNING</code> .
	At the next fine point EGM will return to the state <code>EGM_STATE_CONNECTED</code> .
6	To free an EGM identity for use with another sensor you have to reset EGM, which returns EGM to the state <code>EGM_STATE_DISCONNECTED</code> .

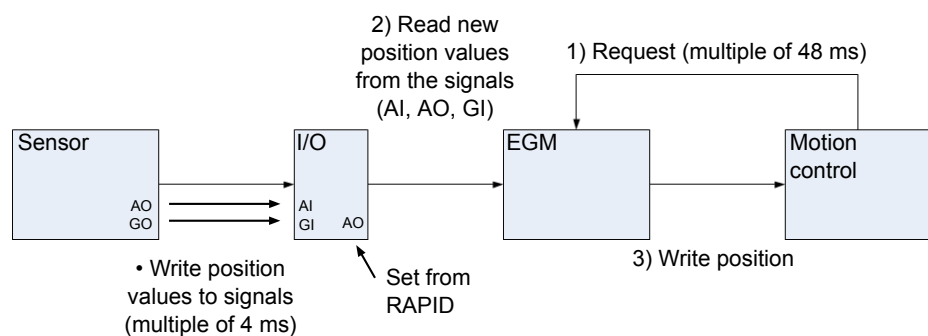
### Input data for EGM Path Correction

The source for input data is selected using the EGM setup instructions. The three first instructions select a signal interface and the last instruction a UdpUc interface (*User Datagram Protocol Unicast Communication*).

Instructions	Description
EGMSetupAI	Setup analog input signals for EGM
EGMSetupAO	Setup analog output signals for EGM
EGMSetupGI	Setup group input signals for EGM
EGMSetupUC	Setup the UdpUc protocol for EGM

Input data for EGM contain mainly position data.

The data flow for the signal interface is illustrated below:



xx1400002016

*Continues on next page*

## 6 Tracking

---

### 6.5 Tracking with EGM

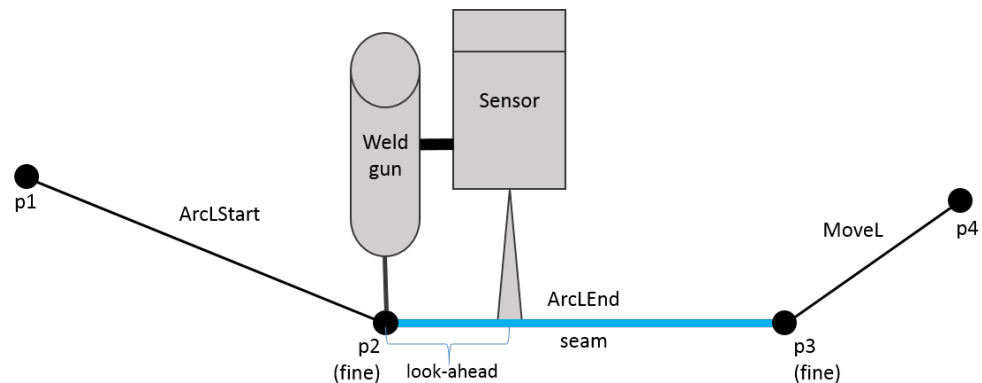
*Continued*

- 1 Motion control calls EGM.
- 2 The measurement data (y- and z-values) are read from the signals or fetched from the sensor at multiples of about 48 ms.
- 3 EGM calculates the position correction and writes it to motion control. If the UdpUc protocol is used, feedback is sent to the sensor.

## 6.6 Pre Process Tracking

### Description

Optical seam tracking sensors are look-ahead trackers, which means that the sensor cannot start to track at the seam start because the process is started when the weld gun (TCP) is at the seam start. The consequence is that the sensor is a distance of length "look-ahead" into the seam.



xx1700000361

This fact may, in the worst case, make it impossible to use tracking with an optical sensor. One example is if the seam length is of the same magnitude as the look-ahead distance. Another example is if the start position for the seam is not the same for all work pieces and it is not possible to search it.

With CAP and Arc it is possible to avoid this problem by using *Pre Process Tracking*.

### Pre Process Tracking with Arc

Compared to programming tracking without *Pre Process Tracking* you have to add one extra position ( $p_S$ ) that is used in the `ArcXStart` instruction and one extra `ArcX` instruction before the position where you want the process to start ( $p_2$ ). The position  $p_S$  should be chosen in a way that the sensor measurements start before  $p_2$  on the programmed path. The reason is that the robot should stop at the corrected  $p_2$  position for ignition.

Place  $p_2$ , the welding start, at a position where the sensor has the possibility to get valid measurements before  $p_2$ . If the weld start cannot be placed as described above, you have to search the weld start position and use the searched position

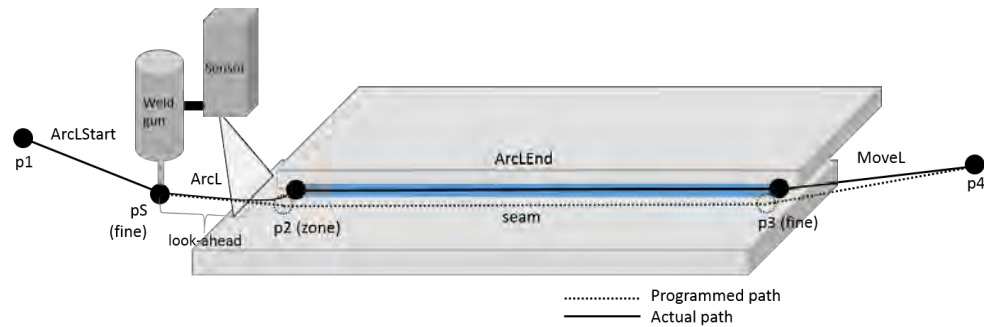
*Continues on next page*

## 6 Tracking

### 6.6 Pre Process Tracking

Continued

instead of p2. Otherwise the ignition will be done at the programmed p2 position which might be off the seam.



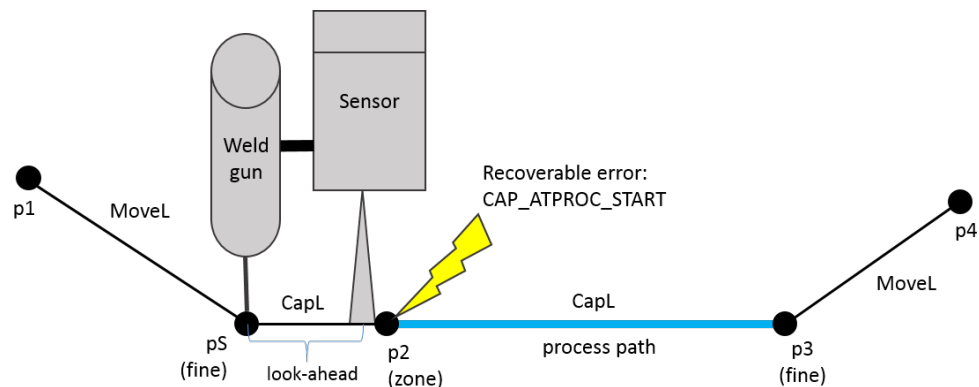
xx1700000362

To activate *Pre Process Tracking* you have to specify the optional argument `\PreProcessTracking` for the RAPID instruction `ArcXStart`. Below is an RAPID code example for optical seam tracking with *Pre Process Tracking*.

```
PROC LA_Arc()  
  MoveJ p1, v1000, z50, tFroniusMTB400i_WR;  
  ArcLStart pS, v1000, mySeam, weld10, fine,  
    tFroniusMTB400i_WR\Track:=track4\PreProcessTracking;  
  ArcL p2, v1000, mySeam, weld10, z50,  
    tFroniusMTB400i_WR\Track:=track4;  
  ArcLEnd p3, v1000, mySeam, myWeld, fine,  
    tFroniusMTB400i_WR\Track:=track4;  
  MoveL p4, v1000, z50, tFroniusMTB400i_WR;  
ENDPROC
```

#### Pre Process Tracking with CAP

Compared to programming tracking without *Pre Process Tracking* you have to add one extra position (pS) as the new starting point for the `CapX` sequence and one extra `CapX` instruction before the position where you want the process to start (p2). The position pS should be chosen in a way that the sensor can see the "path with process" (p2 to p3) during the movement from pS to p2.



xx1700000363

To activate *Pre Process Tracking* you have to specify the optional argument `\PreProcessTracking` for the extra `CapX` instruction, which is the first in the

Continues on next page

sequence. `first_instr` is set to `TRUE`. When the movement has arrived position `p2`, the movement stops and a recoverable error `CAP_ATPROC_START` is generated. In the error handler you have the possibility to start the process during error recovery.

Below is an RAPID code example for optical seam tracking with *Pre Process Tracking*.

```
PROC LA()  
  MoveJ p1, v1000, z50, tFroniusMTB400i_WR;  
  MoveL pS, v1000, fine, tFroniusMTB400i_WR;  
  CapL p2, v200, capdata1, capwvst1, capwv1, z10,  
    tFroniusMTB400i_WR\Track:=captrack1\PreProcessTracking;  
  CapL p3, v200, capdata2, capwvst1, capwv1, fine,  
    tFroniusMTB400i_WR\Track:=captrack1;  
  MoveL p4, v1000, fine, tFroniusMTB400i_WR;  
ERROR  
  TEST ERRNO  
  CASE CAP_ATPROC_START:  
    ! Here the process can be started  
    StartMoveRetry;  
  DEFAULT:  
    RETRY;  
  ENDTEST  
ENDPROC
```

This page is intentionally left blank



## 7 Searching

---

### Description

The existing functionality for searching with optical sensors is only a rudimentary toolbox for advanced users. There is the basic CAP instruction `CapSenScan` and the Arc instructions `OptSearch_1D` and `ArcSearchLStart`, which are slightly more user friendly. The Arc instructions uses the data type `optscandata`.



#### Note

These instructions are mentioned as they exist in the system and show up in the pick lists on the FlexPendant, but they are not an official product.

For more information on the RAPID instructions, see [RAPID toolbox for searching on page 78](#).


**This page is intentionally left blank**

## 8 Adaptivity with optical sensors

### Description

Adaptive welding has to be programmed in RAPID. The instruction `IValValue` connected to a `TRAP` routine can be used to supervise changes of sensor variables. To only influence the travel speed, it is sufficient to use `IVarValue` without a `TRAP` routine.

The instruction `IVarValue` that is included in the option *Sensor Interface* has the following optional arguments:

Argument	Description
<code>\ReportAtTool</code>	If this argument is present the change of the monitored sensor variable is not checked when the sensor measurement is performed, but when the active tool will reach the position where the measurement was performed. This optional argument has to be used together with optical sensors, otherwise the interrupts will arrive too early.
<code>\SpeedAdapt</code>	<p>This optional argument is effective only if <code>\ReportAtTool</code> is present at the same time. It specifies a speed adaption factor:</p> $speed(new[m/s]) = speed(old[m/s]) + (var * scale * speed\_adapt)[m/s]$ <div>  <b>Note</b> </div> <p>Depending on the CPU load and the load on the interpolator, it may take up to 0.5 s until the speed change takes effect.</p>

**This page is intentionally left blank**

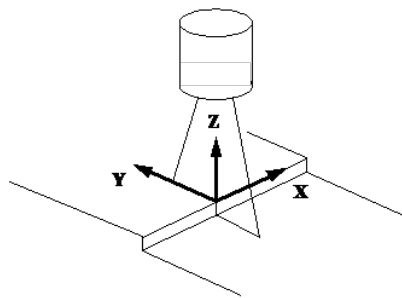
## 9 Reference information

### 9.1 Relationship between coordinate systems

#### Description

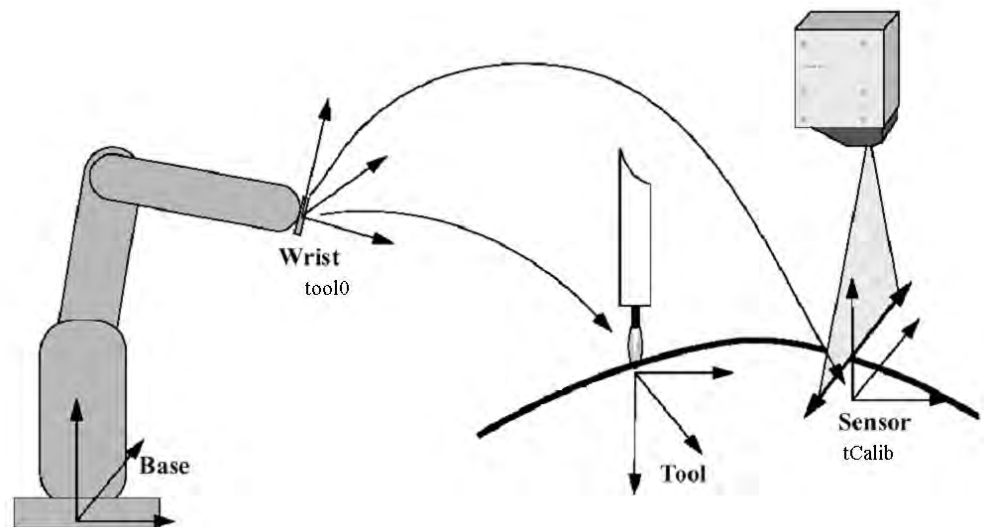
The sensor system shall provide information given in a rectangular right-hand coordinate system defined as follows:

- x-axis - in the direction of movement
- y-axis - horizontal direction
- z-axis - vertical direction "into" the sensor



xx1600001575

The sensor coordinate system has the following relation to the robot base coordinate system:



xx1600001576

## 9 Reference information

---

### 9.2.1 Introduction

## 9.2 Protocols

### 9.2.1 Introduction

---

#### Description

The communication is divided into two layers, the link protocol and the application protocol.

The task of the link protocol is to safely transfer messages between robot controller and sensor. The contents of the messages is of no concern for the link protocol. It should transfer any type of messages.

The task of the application protocol is to interpret the contents of the messages. Depending on the contents various actions shall be executed.

The link protocol is specified as a master/slave protocol, where robot controller is the master and sensor is the slave. This makes the protocol very efficient. There is no need for initial negotiating to set up the link. A complete communication session with a request message down to the slave, and a response message back to the master, will be transferred in only two telegrams with a minimum of overhead.

The protocols listed here are application protocols.

## 9.2.2 LTAPP

### Message structure

Structure of a message:

`<message type> <header><data>`

Content	Description
<code>&lt;message type&gt;</code>	Indicates the type of message, including if the message is a request or a response. <code>&lt;message type&gt;</code> is always present in a message.
<code>&lt;header&gt;</code>	Contains a user number 0-15 to make it possible to communicate with several sensors on the same serial link. Head specifies if a response is required and if more data will be sent. Head has also bits for future use.
<code>&lt;data&gt;</code>	Additional data depending on the type of message. The number of characters could be 0..128.



#### Note

Note the difference between message and telegram. The telegram is the container for the message during the transmission over the link. The message is the real information sent from robot controller to sensor.

### Requests and responses

A request is a message from robot controller to sensor, demanding an action to take place. Requests are indicated with `<message type> = 1..126`.

A response is a message from sensor to robot controller, indicating the result of a request. Responses are indicated with `<message type> = 129..254`.

The response codes are calculated as corresponding request code added to hex 80. If a request has `<message type> = 3`, the corresponding response has `<message type> = hex 83`.

### Data types

Data in messages are always of a specific data type. The following data types are used:

Name	Bytes	Type of data	Range
unsigned byte	1	integers	0..255
signed byte	1	integers	-128..+127
unsigned word	2	integers	0..65535
signed word	2	integers	-32768..+32767

Where data are of a type using more than one byte, the most significant part is transmitted first, the least significant part transmitted last.

*Continues on next page*

## 9 Reference information

### 9.2.2 LTAPP

*Continued*

#### Status

In a response message there is always a `<status>` byte included. This indicates the overall status of the action requested:

Value	Description
+64..+127	OK with an additional specific indication.
+1..+63	OK with an additional generic indication.
0	OK
-1..-63	Not OK, generic error codes.
-64..-128	Not OK, specific error codes.

If `<status>` is = 0 the task has been performed as requested.

If `<status>` is < 0 some error has occurred. The value will indicate what type of error. Generic error codes are defined below. Specific error codes may be defined specially for each implementation.

If `<status>` is > 0 no error has occurred but there is need to return some additional information. For example calibration has started but is not yet fulfilled.

#### Generic status codes

Following generic status codes can be returned in `<status>` in a response message:

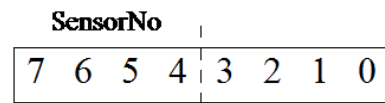
Error code	Description
+2	No measurement.
+1	Not ready yet.
0	OK
-1	General error has occurred.
-2	Busy, try later.
-3	Unknown command.
-4	Illegal variable or block.
-5	External alarm.
-6	Camera alarm.
-7	Temperature alarm.
-8	Value out of range.
-9	CAMCHECK failed.

If `<status>` is = -1, a general error has occurred which is not specifically connected to the requested action. Read the block *Error log* if this function is available to get complete information.

*Continues on next page*



## Header



xx1600001577

Bits	Request messages	Response messages
7-4	Not used.	Not used.
3	0: No more data to send 1: More data to send.	Not used.
2-0	Not used.	Not used.

## Functions

The following chapter describes all application protocol messages regarding:

- Function
- Syntax of request message
- Syntax of response message
- Parameters in request and response
- Description

## Read variable

	Description
<b>Function:</b>	To read one or more not consecutive variables in the sensor.
<b>Message type:</b>	1
<b>Request:</b>	01<header><number><variable1><variable2>...<variable <sub>n</sub> >
<b>Response:</b>	81<header><status><data1><data2>...<data <sub>n</sub> >
<b>Parameters:</b>	<header>, see <a href="#">Header on page 57</a> . <number>, unsigned word: number of variables to read. <status>, signed byte: status of the response. <variable <sub>j</sub> >, unsigned word: indicates which variable to read, 0, 1, 2..., 65535. A list of sensor variables can be found in <i>Application manual - Controller software IRC5</i> , section <i>Sensor Interface</i> . <data <sub>j</sub> >, unsigned words: data for the variable <i>j</i> to read.
<b>Description:</b>	A number of variables may be defined in the sensor. By this function it is possible for robot controller to read these variables. Each variable has the data type <code>unsigned word</code> .

## Write variable

	Description
<b>Function:</b>	To read one or more not consecutive variables in the sensor.
<b>Message type:</b>	2
<b>Request:</b>	02<header><number><variable1><data1>...<variable <sub>n</sub> ><data <sub>n</sub> >

Continues on next page

## 9 Reference information

### 9.2.2 LTAPP

Continued

	Description
<b>Response:</b>	81<header><status>
<b>Parameters:</b>	<header>, see <a href="#">Header on page 57</a> . <number>, unsigned word: number of variables to write. <variable <sub>j</sub> >, unsigned word: indicates which variable to write, 0, 1, 2..., 65535. A list of sensor variables can be found in <i>Application manual - Controller software IRC5</i> , section <i>Sensor Interface</i> . <data <sub>j</sub> >, unsigned words: data for the variable <i>j</i> to write. <status>, signed byte: status of the response.
<b>Description:</b>	A number of variables may be defined in the sensor. By this function it is possible for robot controller to write these variables. Each variable has the data type <code>unsigned word</code> .

Telegram examples for RTP1 transport protocol

All values are hexadecimal.

<header>=00 in all examples.

<status>=00 if Vision could execute the request (OK).

Acronym	Value
ETX	0x03
DLE	0x10
STX0	0x02
STX1	0x82
ENQ	0x05
NAK	0x15
BCC	Check sum CRC16, see <a href="#">CRC16 algorithm on page 59</a> .
T	0x00

A list of sensor variables can be found in *Application manual - Controller software IRC5*, section *Sensor Interface*.

Selection of joint number 3:

**Robot:** DLE STX<sub>n</sub> T 02 <head> 01 10 03 DLE ETX BCC

**Vision:** DLE STX<sub>n</sub> T 82 <head>< status> DLE ETX BCC

Turn on the camera with no laser power:

**Robot:** DLE STX<sub>n</sub> T 02 <head> 02 06 01 07 01 DLE ETX BCC

**Vision:** DLE STX<sub>n</sub> T 82 <head>< status> DLE ETX BCC

Activate the camera laser power (measure on):

**Robot:** DLE STX<sub>n</sub> T 02 <head> 01 07 00 DLE ETX BCC

**Vision:** DLE STX<sub>n</sub> T 82 <head><status> DLE ETX BCC

Request data and data not available:

**Robot:** DLE STX<sub>n</sub> T 01 <head> 03 09 0A 0B DLE ETX BCC

**Vision:** DLE STX<sub>n</sub> T 81 <head><status> 00 00 00 DLE ETX BCC

<status> = 02 (No measurement)

Continues on next page

**Request data and data is available:**

**Robot:** DLE STXn T 01<head> 03 09 0A 0B DLE ETX BCC

**Vision:** DLE STXn T 81<head><status><data1><data2><data3>DLE ETX BCC

**Turn the laser power down (measure off):**

**Robot:** DLE STXn T 02 <head> 01 07 01 DLE ETX BCC

**Vision:** DLE STXn T 82 <head><status> DLE ETX BCC

**Turn off the camera:**

**Robot:** DLE STXn T 02 <head> 01 06 00 DLE ETX BCC

**Vision:** DLE STXn T 82 <head><status> DLE ETX BCC

**CRC16 algorithm**

```
int crc16 (char *bufp, int count)
{
    unsigned int crc = 0;
    while (count--)
    {
        crc = (crc >> 8) ^ xtab [ (unsigned char) (crc ^ *bufp++) ];
    }
    return (crc);
}

static unsigned short xtab[256] = {
0x0000, 0xc0c1, 0xc181, 0x0140, 0xc301, 0x03c0, 0x0280, 0xc241,
0xc601, 0x06c0, 0x0780, 0xc741, 0x0500, 0xc5c1, 0xc481, 0x0440,
0xcc01, 0x0cc0, 0x0d80, 0xcd41, 0x0f00, 0xcfc1, 0xce81, 0x0e40,
0x0a00, 0xcac1, 0xcb81, 0x0b40, 0xc901, 0x09c0, 0x0880, 0xc841,
0xd801, 0x18c0, 0x1980, 0xd941, 0x1b00, 0xdbc1, 0xda81, 0x1a40,
0x1e00, 0xdec1, 0xdf81, 0x1f40, 0xdd01, 0x1dc0, 0x1c80, 0xdc41,
0x1400, 0xd4c1, 0xd581, 0x1540, 0xd701, 0x17c0, 0x1680, 0xd641,
0xd201, 0x12c0, 0x1380, 0xd341, 0x1100, 0xd1c1, 0xd081, 0x1040,
0xf001, 0x30c0, 0x3180, 0xf141, 0x3300, 0xf3c1, 0xf281, 0x3240,
0x3600, 0xf6c1, 0xf781, 0x3740, 0xf501, 0x35c0, 0x3480, 0xf441,
0x3c00, 0xfcc1, 0xfd81, 0x3d40, 0xff01, 0x3fc0, 0x3e80, 0xfe41,
0xfa01, 0x3ac0, 0x3b80, 0xfb41, 0x3900, 0xf9c1, 0xf881, 0x3840,
0x2800, 0xe8c1, 0xe981, 0x2940, 0xeb01, 0x2bc0, 0x2a80, 0xea41,
0xee01, 0x2ec0, 0x2f80, 0xef41, 0x2d00, 0xedc1, 0xec81, 0x2c40,
0xe401, 0x24c0, 0x2580, 0xe541, 0x2700, 0xe7c1, 0xe681, 0x2640,
0x2200, 0xe2c1, 0xe381, 0x2340, 0xe101, 0x21c0, 0x2080, 0xe041,
0xa001, 0x60c0, 0x6180, 0xa141, 0x6300, 0xa3c1, 0xa281, 0x6240,
0x6600, 0xa6c1, 0xa781, 0x6740, 0xa501, 0xa5c0, 0x6480, 0xa441,
0x6c00, 0xacc1, 0xad81, 0x6d40, 0xaf01, 0x6fc0, 0x6e80, 0xae41,
0xaa01, 0x6ac0, 0x6b80, 0xab41, 0x6900, 0xa9c1, 0xa881, 0x6840,
0x7800, 0xb8c1, 0xb981, 0x7940, 0xbb01, 0x7bc0, 0x7a80, 0xba41,
0xbe01, 0x7ec0, 0x7f80, 0xbf41, 0x7d00, 0xbdc1, 0xbc81, 0x7c40,
0xb401, 0x74c0, 0x7580, 0xb541, 0x7700, 0xb7c1, 0xb681, 0x7640,
0x7200, 0xb2c1, 0xb381, 0x7340, 0xb101, 0x71c0, 0x7080, 0xb041,
0x5000, 0x90c1, 0x9181, 0x5140, 0x9301, 0x93c0, 0x5280, 0x9241,
0x9601, 0x96c0, 0x9780, 0x9741, 0x9500, 0x95c1, 0x9481, 0x9440,
0x9c01, 0x9cc0, 0x9d80, 0x9d41, 0x9f00, 0x9fc1, 0x9e81, 0x9e40,
```

*Continues on next page*

## 9 Reference information

---

### 9.2.2 LTAPP

*Continued*

```
0x5a00, 0x9ac1, 0x9b81, 0x5b40, 0x9901, 0x59c0, 0x5880, 0x9841,  
0x8801, 0x48c0, 0x4980, 0x8941, 0x4b00, 0x8bc1, 0x8a81, 0x4a40,  
0x4e00, 0x8ec1, 0x8f81, 0x4f40, 0x8d01, 0x4dc0, 0x4c80, 0x8c41,  
0x4400, 0x84c1, 0x8581, 0x4540, 0x8701, 0x47c0, 0x4680, 0x8641,  
0x8201, 0x42c0, 0x4380, 0x8341, 0x4100, 0x81c1, 0x8081, 0x4040};
```

### 9.2.3 LTAPPTCP

#### Description

The LTAPPTCP protocol is using LTAPP as body, see [LTAPP on page 55](#). A header and a footer is added. The complete message is sent in network order.

	Description
<b>Header:</b>	4 bytes Byte length of body
<b>Body:</b>	Body, see <a href="#">LTAPP on page 55</a> .
<b>Footer:</b>	2 bytes 0xabb

## 9 Reference information

---

### 9.2.4 Sockdev

#### 9.2.4 Sockdev

---

##### Description

The sockdev protocol is an implementation of the ServoRobot proprietary XML based protocol Robo-Com Light®. For more information contact Servo-Robot Inc.

## 9.2.5 The EGM sensor protocol

### Description

The EGM sensor protocol is designed for high speed communication between a robot controller and a communication endpoint with minimum overhead.

The communication endpoint is typically a sensor, so *sensor* will be used from now on instead of communication endpoint. Sometimes the sensor is connected to a PC, and the PC then transfers the sensor data to the robot. The purpose of the sensor protocol is to communicate sensor data frequently between the robot controller and sensors. The EGM sensor protocol is using Google Protocol Buffers for encoding and UDP as a transport protocol. Google Protocol Buffers has been selected due to its speed and its language-neutrality. UDP has been chosen as a transport protocol since the data sent is *real-time* data sent with high frequency and if packets get lost it is useless to re-send the data.

The EGM sensor protocol data structures are defined by the EGM proto file. Sensor name, IP-address and port number of sensors are configured in the system parameters. A maximum of eight sensors can be configured.

The sensor is acting as a server and it cannot send anything to the robot before it has received a first message from the robot controller. Messages can be sent independently of each other in both directions after that first message. Applications using the protocol may put restrictions on its usage but the protocol itself has no built-in synchronization of request responses or supervision of lost messages.

There are no special connect or disconnect messages, only data which can flow in both directions independently of each other. The first message from the robot is a data message. One has also to keep in mind, that a sender of an UDP message continues to send even though the receiver's queue may be full. The receiver has to make sure, that its queue is emptied.

By default, the robot will send and read data from the sensor every 4 milliseconds, independently of when data is sent from the sensor. This cycle time can be changed to a multiple of 4 ms using the optional argument `\SampleRate` of the RAPID instructions `EGMActJoint` or `EGMActPose`.

### Google Protocol Buffers

Google Protocol Buffers or *Protobuf*, are a way to serialize/de-serialize data in a very efficient way. Protobuf is in general 10-100 times faster than XML. There is plenty of information on the Internet about Protobuf and the *Google overview* is a good start.

In short, message structures are described in a *.proto* file. The *.proto* file is then compiled. The compiler generates serialized/de-serialized code which is then used by the application. The application reads a message from the network, runs the de-serialization, creates a message, calls serialization method, and then sends the message.

It is possible to use Protobuf in most programming languages since Protobuf is language neutral. There are many different implementations depending on the language.

*Continues on next page*

## 9 Reference information

### 9.2.5 The EGM sensor protocol

*Continued*

The main disadvantage with Protobuf is that Protobuf messages are serialized into a binary format which makes it more difficult to debug packages using a network analyzer.

#### Third party tools

Except for the *Google C++* tool, we have also verified the following third party tools and code:

- *Nanopb*, generates C-code and it does not require any dynamic memory allocations.
- *Protobuf-net*, a Google Protobuf .NET library.
- *Protobuf-csharp*, a Google Protobuf .NET library, the C# API is similar to the Google C++ API.



#### Note

Note that the code mentioned above is open source, which means that you have to check the license that the code is allowed to be used in your product.

#### EGM sensor protocol description

The EGM sensor protocol is not a request/response protocol, the sensor can send data at any frequency after the sensor gets the first message from the robot.

The EGM sensor protocol has two main data structures, *EgmRobot* and *EgmSensor*. *EgmRobot* is sent from the robot and *EgmSensor* is sent from the sensor. All message fields in both the data structures are defined as optional which means that a field may or may not be present in a message. Applications using *Google Protocol Buffers* must check if optional fields are present or not.

The *EgmHeader* is common for both *EgmRobot* and *EgmSensor*.

```
message EgmHeader
{
  optional uint32 seqno = 1; // sequence number (to be able to find
    lost messages)
  optional uint32 tm = 2; // time stamp in milliseconds

  enum MessageType {
    MSGTYPE_UNDEFINED = 0;
    MSGTYPE_COMMAND = 1; // for future use
    MSGTYPE_DATA = 2; // sent by robot controller
    MSGTYPE_CORRECTION = 3; // sent by sensor
  }

  optional MessageType mtype = 3 [default = MSGTYPE_UNDEFINED];
}
```

Variable	Description
seqno	Sequence number. Applications shall increase the sequence number by one for each message they send. It makes it possible to check for lost messages in a series of messages.

*Continues on next page*



Variable	Description
tm	Timestamp in milliseconds. (Can be used for monitoring of delays).
mtype	Message type. Shall be set to MSGTYPE_CORRECTION by the sensor, and is set to MSGTYPE_DATA by the robot controller.

The Google protobuf data structure can include the *repeated* element, i.e. a list of elements of the same type. The *repeated* element count is a maximum of six elements in the EGM sensor protocol.

#### How to build an EGM sensor communication endpoint using .Net

This guide assumes that you build and compile using Visual Studio and are familiar with its operation.

Here is a short description on how to install and create a simple test application using *protobuf-csharp-port*.

	Action
1	Download protobuf-csharp binaries from: <a href="https://code.google.com/p/protobuf-csharp-port/">https://code.google.com/p/protobuf-csharp-port/</a> .
2	Unpack the zip-file.
3	Copy the <i>egm.proto</i> file to a sub catalogue where protobuf-csharp was un-zipped, e.g. <code>~\protobuf-csharp\tools\egm</code> .
4	Start a Windows console in the tools directory, e.g. <code>~\protobuf-csharp\tools</code> .
5	Generate an EGM C# file ( <i>egm.cs</i> ) from the <i>egm.proto</i> file by typing in the Windows console: <code>protogen .\egm\egm.proto --proto_path=.\egm</code>
6	Create a C# console application in Visual Studio. Create a C# Windows console application in Visual Studio, e.g. <i>EgmSensorApp</i> .
7	Install NuGet, in Visual Studio, click <b>Tools</b> and then <b>Extension Manager</b> . Go to <b>Online</b> , find the <i>NuGet Package Manager extension</i> and click <b>Download</b> .
8	Install protobuf-csharp in the solution for the C# Windows Console application using NuGet. The solution has to be open in Visual Studio.
9	In Visual Studio select, <b>Tools</b> , <b>Nuget Package Manager</b> , and <b>Package Manager Console</b> . Type <code>PM&gt;Install-Package Google.ProtocolBuffers</code>
10	Add the generated file <i>egm.cs</i> to the Visual Studio project (add existing item).
11	Copy the example code into the Visual Studio Windows Console application file ( <i>EgmSensorApp.cpp</i> ) and then compile, link and run.

#### How to build an EGM sensor communication endpoint using C++

When building using C++ there are no other third party libraries needed.

C++ is supported by Google. It can be a bit tricky to build the Google tools in Windows but here is a guide on how to build protobuf for Windows.

*Continues on next page*

## 9 Reference information

### 9.2.5 The EGM sensor protocol

*Continued*

Use the following procedure when you have built *libprotobuf.lib* and *protoc.exe*:

	Action
1	Run Google protoc to generate access classes, <code>protoc --cpp_out=. egm.proto</code>
2	Create a win32 console application
3	Add Protobuf source as include directory.
4	Add the generated <i>egm.pb.cc</i> file to the project, exclude the file from precompile headers.
5	Copy the code from the <i>egm-sensor.cpp</i> file.
6	Compile and run.

#### Configuring UdpUc devices

UdpUc communicates with a maximum of eight devices over Udp. The devices act as servers, and the robot controller acts as a client. It is the robot controller that initiates the connection to the sensor.

#### System parameters

This is a brief description of the parameters used when configuring a device. For more information about the parameters, see *Technical reference manual - System parameters*.

These parameters belong to the type *Transmission Protocol* in topic *Communication*.

Parameter	Description
<i>Name</i>	The name of the transmission protocol. For example <i>EGMsensor</i> .
<i>Type</i>	The type of transmission protocol. It has to be <i>UDPUC</i> .
<i>Serial Port</i>	The name of the serial port that will be used for the sensor. This refers to the parameter <i>Name</i> in the type <i>Serial Port</i> . For IP based transmission protocols (i.e. <i>Type</i> has value TCP/IP, SOCKDEV, LTAPPTCP or UDPUC), <i>Serial Port</i> is not used and has the value N/A.
<i>Remote Address</i>	The IP address of the remote device.
<i>Remote Port Number</i>	The IP port number that the remote device has opened.

#### Configuration example

The device which provides the input data for EGM, has to be configured as an UdpUc device in the following way:

Name	Type	Serial Port	Remote Address	Remote Port Number
UCdevice	UDPUC	N/A	192.168.10.20	6510

After this configuration change, the controller has to be restarted. Now the device can be used by EGM Path Correction. For more information, see [Using EGM Path Correction with different protocol types on page 67](#).

## 9.2.6 Using EGM Path Correction with different protocol types

## Description

This example contains examples for different sensor and protocol types. The basic RAPID program structure is the same for all of them and they use the same external motion data configuration.

## Example

```

MODULE EGM_PATHCORR
! Used tool
PERS tooldata tEGM:=[TRUE,[[148.62,0.25,326.31],
    [0.833900724,0,0.551914471,0]], [1,[0,0,100],
    [1,0,0,0],0,0,0]];
! Sensor tool, has to be calibrated
PERS tooldata
    tLaser:=[TRUE,[[148.619609537,50.250017146,326.310337954],
    [0.390261856,-0.58965743,-0.58965629,0.390263064]],
    [1,[-0.920483747,-0.000000536,-0.390780849],
    [1,0,0,0],0,0,0]];
! Displacement used
VAR pose PP:=[[0,-3,2],[1,0,0,0]];
VAR egmident egmId1;

! Protocol: LTAPP
! Example for a look ahead sensor, e.g. Laser Tracker
PROC Part_2_EGM_OT_Pth_1()
    EGMGetId egmId1;
    ! Set up the EGM data source: LTAPP server using device "Optsim",
    ! configuration "pathCorr", joint type 1 and look ahead sensor.
    EGMSetupLTAPP ROB_1, egmId1, "pathCorr", "OptSim", 1\LATR;
    ! Activate EGM and define the sensor frame.
    ! Correction frame is always the path frame.
    EGMActMove egmId1, tLaser.tframe\SampleRate:=50;
    ! Move to a suitable approach position.
    MoveJ p100,v1000,z10,tEGM\WObj:=wobj0;
    MoveL p110,v1000,z100,tEGM\WObj:=wobj0;
    MoveL p120,v1000,z100,tEGM\WObj:=wobj0;
    ! Activate displacement (not necessary but possible)
    PDispSet PP;
    ! Move to the start point. Fine point is demanded.
    MoveL p130, v10, fine, tEGM\WObj:=wobj0;
    ! movements with path corrections.
    EGMMoveL egmId1, p140, v10, z5, tEGM\WObj:=wobj0;
    EGMMoveL egmId1, p150, v10, z5, tEGM\WObj:=wobj0;
    EGMMoveC egmId1, p160, p165, v10, z5, tEGM\WObj:=wobj0;
    ! Last path correction movement has to end with a fine point.
    EGMMoveL egmId1, p170, v10, fine, tEGM\WObj:=wobj0;
    ! Move to a safe position after path correction.
    MoveL p180,v1000,z10,tEGM\WObj:=wobj0;
    ! Release the EGM identity for reuse.

```

*Continues on next page*

## 9 Reference information

---

### 9.2.6 Using EGM Path Correction with different protocol types

*Continued*

```
    EGMReset egmId1;
ENDPROC

! Protocol: LTAPP
! Example for an at point sensor, e.g. Weldguide
PROC Part_2_EGM_WG_Pth_1()
    EGMGetId egmId1;
    ! Set up the EGM data source: LTAPP server using device "wglsim",
    ! configuration "pathCorr", joint type 1 and at point sensor.
    EGMSetupLTAPP ROB_1, egmId1, "pathCorr", "wglsim", 1\APTR;
    ! Activate EGM and define the sensor frame,
    ! which is the tool frame for at point trackers.
    ! Correction frame is always the path frame.
    EGMActMove egmId1, tEGM.tframe\SampleRate:=50;
    ! Move to a suitable approach position.
    MoveJ p100,v1000,z10,tEGM\WObj:=wobj0;
    MoveL p110,v1000,z100,tEGM\WObj:=wobj0;
    MoveL p120,v1000,fine,tEGM\WObj:=wobj0;
    ! Activate displacement (not necessary but possible)
    PDispSet PP;
    ! Move to the start point. Fine point is demanded.
    MoveL p130, v10, fine, tEGM\WObj:=wobj0;
    ! movements with path corrections.
    EGMMoveL egmId1, p140, v10, z5, tEGM\WObj:=wobj0;
    EGMMoveL egmId1, p150, v10, z5, tEGM\WObj:=wobj0;
    EGMMoveC egmId1, p160, p165, v10, z5, tEGM\WObj:=wobj0;
    ! Last path correction movement has to end with a fine point.
    EGMMoveL egmId1, p170, v10, fine, tEGM\WObj:=wobj0;
    ! Move to a safe position after path correction.
    MoveL p180,v1000,z10,tEGM\WObj:=wobj0;
    ! Release the EGM identity for reuse.
    EGMReset egmId1;
ENDPROC

! Protocol: UdpUc
! Example for an at point sensor, e.g. Weldguide
PROC Part_2_EGM_UDPUC_Pth_1()
    EGMGetId egmId1;
    EGMSetupUC ROB_1, egmId1, "pathCorr", "UCdevice"\PathCorr\APTR;
    EGMActMove egmId1, tEGM.tframe\SampleRate:=50;
    ! Move to a suitable approach position.
    MoveJ p100,v1000,z10,tEGM\WObj:=wobj0;
    MoveL p110,v1000,z100,tEGM\WObj:=wobj0;
    MoveL p120,v1000,fine,tEGM\WObj:=wobj0;
    ! Activate displacement (not necessary but possible)
    PDispSet PP;
    ! Move to the start point. Fine point is demanded.
    MoveL p130, v10, fine, tEGM\WObj:=wobj0;
    ! movements with path corrections.
    EGMMoveL egmId1, p140, v10, z5, tEGM\WObj:=wobj0;
    EGMMoveL egmId1, p150, v10, z5, tEGM\WObj:=wobj0;
```

*Continues on next page*

### 9.2.6 Using EGM Path Correction with different protocol types

*Continued*

```
EGMMoveC egmId1, p160, p165, v10, z5, tEGM\WObj:=wobj0;
! Last path correction movement has to end with a fine point.
EGMMoveL egmId1, p170, v10, fine, tEGM\WObj:=wobj0;
! Move to a safe position after path correction.
MoveL p180,v1000,z10,tEGM\WObj:=wobj0;
! Release the EGM identity for reuse.
EGMReset egmId1;
ENDPROC
ENDMODULE
```

**This page is intentionally left blank**

## 10 RAPID reference

### 10.1 RAPID components for optical tracking with Sensor Interface

#### About the RAPID components

This is an overview of the instructions, functions, and data types in *Sensor Interface* that can be used for optical tracking.

For an overview of all instructions, functions, and data types in *Sensor Interface*, see *Application manual - Controller software IRC5*.

For more information on the RAPID components, see *Technical reference manual - RAPID Instructions, Functions and Data types*.

#### Instructions

Instructions	Description
SenDevice	SenDevice is used, to connect to a physical sensor device.
IVarValue	IVarVal (Interrupt Variable Value) is used to order and enable an interrupt when the value of a variable accessed via the serial sensor interface is changed.
WriteVar	WriteVar is used to write a variable to a device connected to the serial sensor interface.

#### Functions

Function	Description
ReadVar	ReadVar is used to read a variable from a device connected to the serial sensor interface.

#### Data types

There are no data types for *Sensor Interface*.

#### Modules

The option *Sensor Interface* includes one system module, *LTAPP\_\_Variables*. This module contains the variable numbers defined in the protocol LTAPP. It is automatically loaded as SHARED and makes the variables (CONST num) available in all RAPID tasks.



#### Note

A copy of the module is placed in the robot system directory *HOME/LTC*, but the copy is not the loaded module.

#### Constants

Name	Number	Read/write (R/W)	Description
LTAPP__VERSION	1	R	A value that identifies the sensor software version.

*Continues on next page*

## 10 RAPID reference

### 10.1 RAPID components for optical tracking with Sensor Interface

*Continued*

Name	Number	Read/write (R/W)	Description
LTAPP__RESET	3	W	Reset the sensor to the initial state, regardless of what state it is currently in.
LTAPP__PING	4	W	Sensor returns a response indicating its status.
LTAPP__CAMCHECK	5	W	Start camera check of the sensor. If this cannot be done within the time limit specified in the link protocol a <i>Not ready yet</i> status will be returned.
LTAPP__POWER_UP	6	RW	Turn power on (1) or off (0) for the sensor and initialize the filters. (Power on can take several seconds!)
LTAPP__LASER_OFF	7	RW	Switch the laser beam off (1) or on (0) and measure.
LTAPP__X	8	R	Measured X value, unsigned word. The units are determined by the variable <i>Unit</i> .
LTAPP__Y	9	R	Measured Y value, unsigned word. The units are determined by the variable <i>Unit</i> .
LTAPP__Z	10	R	Measured Z value, unsigned word. The units are determined by the variable <i>Unit</i> .
LTAPP__GAP	11	R	The gap between two sheets of metal. The units are determined by the variable <i>Unit</i> , -32768 if not valid.
LTAPP__MISMATCH	12	R	Mismatch, unsigned word. The units are determined by the variable <i>Unit</i> . -32768 if not valid.
LTAPP__AREA	13	R	Seam area, units in mm <sup>2</sup> , -32768 if not valid.
LTAPP__THICKNESS	14	RW	Plate thickness of sheet that the sensor should look for, LSB=0.1mm.
LTAPP__STEPIR	15	RW	Step direction of the joint: Step on left (1) or right (0) side of path direction.
LTAPP__JOINT_NO	16	RW	Set or get active joint number.
LTAPP__AGE	17	R	Time since profile acquisition (ms), unsigned word.
LTAPP__ANGLE	18	R	Angle of the normal to the joint relative sensor coordinate system Z direction - in 0.1 degrees.
LTAPP__UNIT	19	RW	Units of X, Y, Z, gap, and mismatch. 0= 0.1mm, 1= 0.01mm.
-	20	-	Reserved for internal use.
LTAPP__APM_P1	31	R	Servo robot only! Adaptive parameter 1
LTAPP__APM_P2	32	R	Servo robot only! Adaptive parameter 2

*Continues on next page*



## 10.1 RAPID components for optical tracking with Sensor Interface

*Continued*

Name	Number	Read/write (R/W)	Description
LTAPP__APM_P3	33	R	Servo robot only! Adaptive parameter 3
LTAPP__APM_P4	34	R	Servo robot only! Adaptive parameter 4
LTAPP__APM_P5	35	R	Servo robot only! Adaptive parameter 5
LTAPP__APM_P6	36	R	Servo robot only! Adaptive parameter 6
LTAPP__ROT_Y	51	R	Measured angle around sensor Y axis
LTAPP__ROT_Z	52	R	Measured angle around sensor Z axis A
LTAPP__X0	54	R	Scansonic sensors only. Measured X value line 1, unsigned word. The units are determined by the variable <code>Unit</code> .
LTAPP__Y0	55	R	Scansonic sensors only. Measured Y value line 1, unsigned word. The units are determined by the variable <code>Unit</code> .
LTAPP__Z0	56	R	Scansonic sensors only. Measured Z value line 1, unsigned word. The units are determined by the variable <code>Unit</code> .
LTAPP__X1	57	R	Scansonic sensors only. Measured X value line 2, unsigned word. The units are determined by the variable <code>Unit</code> .
LTAPP__Y1	58	R	Scansonic sensors only. Measured Y value line 2, unsigned word. The units are determined by the variable <code>Unit</code> .
LTAPP__Z1	59	R	Scansonic sensors only. Measured Z value line 2, unsigned word. The units are determined by the variable <code>Unit</code> .
LTAPP__X2	60	R	Scansonic sensors only. Measured X value line 3, unsigned word. The units are determined by the variable <code>Unit</code> .
LTAPP__Y2	61	R	Scansonic sensors only. Measured Y value line 3, unsigned word. The units are determined by the variable <code>Unit</code> .
LTAPP__Z2	62	R	Scansonic sensors only. Measured Z value line 3, unsigned word. The units are determined by the variable <code>Unit</code> .

## 10 RAPID reference

### 10.2 RAPID components for optical tracking with CAP

### 10.2 RAPID components for optical tracking with CAP

#### About the RAPID components

This is an overview of the instructions, functions, and data types in *Continuous Application Platform* that can be used for optical tracking.

For an overview of all instructions, functions, and data types in *Continuous Application Platform*, see *Application manual - Continuous Application Platform*

For more information on the RAPID components, see *Technical reference manual - RAPID Instructions, Functions and Data types*.

#### Instructions

Instructions	Description
CapC <sup>i</sup>	CapC is used to move the tool center point (TCP) along a circular path to a given destination and at the same time control a continuous process.
CapL <sup>i</sup>	CapL is used to move the tool center point (TCP) linearly to a given destination and at the same time control a continuous process.
CapLATrSetup	CapLATrSetup ( <i>Set up a Look-Ahead-Tracker</i> ) is used to set up a Look-Ahead-Tracker type of sensor, for example, Laser Tracker.

<sup>i</sup> captrackdata provides the CapL/CapC instructions with all data necessary for path correction with a Look-Ahead- or At-Point-Tracker. The data is passed to the CapL/C instructions with use of the optional argument \Track.

#### Functions

There are no functions for *Continuous Application Platform*.

#### Data types

Data types	Description
caplatrackdata	caplatrackdata contains data, with which the user can influence how the CapL/CapC instructions incorporate the path correction data generated by a Look-Ahead-Tracker (for example, Laser Tracker). caplatrackdata is part of the captrackdata.
captrackdata	captrackdata provides the CapL/CapC instructions with all data necessary for path correction with a Look-Ahead- or At-Point-Tracker. The data is passed to the CapL/C instructions with of the optional argument \Track.

## 10.3 RAPID components for optical tracking with Arc

### About the RAPID components

This is an overview of the instructions, functions, and data types in *Arc* that can be used for optical tracking.

For more information on all instructions, functions, and data types in *Arc*, see *Application manual - Arc and Arc Sensor*.

### Instructions

Instructions	Description
ArcC <sup>i</sup>	ArcC is used to weld along a circular path.
ArcCEnd <sup>i</sup>	ArcCEnd is used to weld along a circular path.
ArcCStart <sup>i</sup>	ArcCStart is used to weld along a circular path.
ArcL <sup>i</sup>	ArcL is used to weld along a straight seam.
ArcLEnd <sup>i</sup>	ArcLEnd is used to weld along a straight seam.
ArcLStart <sup>i</sup>	ArcLStart is used to weld along a straight seam.

<sup>i</sup> `trackdata` provides the `Arc` instructions with all data necessary for path correction with a Look-Ahead- or At-Point-Tracker. The data is passed to the `Arc` instructions with use of the optional argument `\Track`.

### Functions

There are no functions for *Optical Tracking Arc*.

### Data types

Data types	Description
<code>trackdata</code>	<p><code>trackdata</code> is used to control path corrections during the weld phase.</p> <p><code>trackdata</code> provides the <code>Arc</code> instructions with all data necessary for path correction with a Look-Ahead- or At-Point-Tracker. The data is passed to the <code>Arc</code> instructions with use of the optional argument <code>\Track</code>.</p>

## 10 RAPID reference

### 10.4 RAPID components for optical tracking with EGM

#### 10.4 RAPID components for optical tracking with EGM

##### About the RAPID components

This is an overview of the instructions, functions, and data types in *Externally Guided Motion* that can be used for optical tracking.

For an overview of all instructions, functions, and data types in *Externally Guided Motion*, see *Application manual - Controller software IRC5*.

For more information on the RAPID components, see *Technical reference manual - RAPID Instructions, Functions and Data types*.

##### Instructions

Instructions	Description
EGMGetId	EGMGetId is used to reserve an EGM identity (EGMId). That identity is then used in all other EGM RAPID instructions and functions to identify a certain EGM process connected to the RAPID motion task from which it is used.  An egmident is identified by its name, i.e. a second or third call of EGMGetId with the same egmident will neither reserve a new EGM process nor change its content.
EGMMoveC	EGMMoveC is used to move the tool center point (TCP) circularly to a given destination with path correction. During the movement the orientation normally remains unchanged relative to the circle.
EGMMoveL	EGMMoveL is used to move the tool center point (TCP) linearly to a given destination with path correction. When the TCP is to remain stationary then this instruction can also be used to reorient the tool.
EGMReset	EGMReset resets a specific EGM process (EGMId), i.e. the reservation is canceled.
EGMSetupAI	EGMSetupAI is used to set up analog input signals for a specific EGM process (EGMId) as the source for position destination values to which the robot (plus up to 6 additional axis) is to be guided.
EGMSetupAO	EGMSetupAO is used to set up analog output signals for a specific EGM process (EGMId) as the source for position destination values to which the robot, and up to 6 additional axis, is to be guided.
EGMSetupGI	EGMSetupGI is used to set up group input signals for a specific EGM process (EGMId) as the source for position destination values to which the robot, and up to 6 additional axis, is to be guided.
EGMSetupLTAPP	EGMSetupLTAPP is used to set up an LTAPP protocol for a specific EGM process (EGMId) as the source for path corrections.
EGMSetupUC	EGMSetupUC is used to set up a UdpUc device for a specific EGM process (EGMId) as the source for position destination values to which the robot, and up to 6 additional axis, are to be guided. The position may be given in joints, for EGMRunJoint, or in cartesian format for EGMRunPose.
EGMStop	EGMStop stops a specific EGM process (EGMId).

##### Functions

Functions	Description
EGMGetState	EGMGetState retrieves the state of an EGM process (EGMId).

*Continues on next page*

**Data types**

<b>Data types</b>	<b>Description</b>
egmframetype	egmframetype is used to define the frame types for corrections and sensor measurements in EGM.
egmident	egmident identifies a specific EGM process.
egm_minmax	egm_minmax is used to define the convergence criteria for EGM to finish.
egmstate	egmstate is used to define the state for corrections and sensor measurements in EGM.
egmstopmode	egmstopmode is used to define the stop modes for corrections and sensor measurements in EGM.

## 10 RAPID reference

---

### 10.5.1.1 ArcSearchLStart - Searching with optical sensors for arc welding

## 10.5 RAPID toolbox for searching

### 10.5.1 Instructions

#### 10.5.1.1 ArcSearchLStart - Searching with optical sensors for arc welding

---

##### Usage

`ArcSearchLStart` is used to search with optical sensors.

This instruction uses the functionality from the instruction `OptSearch_1D` and adds the Arc process control to it. It can be used to find the start of a Seam and start the welding process in a controlled way.

---

##### Basic examples

The following example illustrates the instruction `ArcSearchLStart`.

##### Example 1

```
MoveJ ...  
ArcSearchLStart p1, p2, v100, scan1, seam1, weld1, fine, gun1,  
    track1;  
ArcL p3, v100, seam1, weld1, fine, gun1;  
ArcLEnd p4, v100, seam1, weld1, fine, gun1;  
MoveJ ...
```

---

##### Arguments

```
ArcSearchLStart p1 p2 \ID Speed Scan Seam Weld [\Weave] Zone Tool  
    [\Wobj] Track \SeamName
```

`p1`

**Data type:** `robtarget`

The starting point for the search. This position is defined using the active robot tool, not the sensor tool.

`p2`

**Data type:** `robtarget`

The end point for the search. This position is defined using the active robot tool, not the sensor tool.

`\ID`

(*Sync identity*)

**Data type:** `identno`

Synchronization identity for RAPID movement instructions in a MultiMove system in synchronized mode.

`Speed`

**Data type:** `speeddata`

The speed data that applies to movements.

*Continues on next page*

---

### 10.5.1.1 ArcSearchLStart - Searching with optical sensors for arc welding

*Continued*

Scan

**Data type:** `optscandata`

Data that defines how the search is performed.

Seam

**Data type:** `seamdata`

Seam data describes the start and end phases of a welding process. The argument `Seam` is included in all arc welding instructions so that, regardless of the position of the robot when the process is interrupted, a proper weld end and restart is achieved. Normally the same seam data is used in all instructions of a seam.

Weld

**Data type:** `welddata`

Weld data describes the weld phase of the welding process.

Weld data is often changed from one instruction to the next along a seam.

[`\Weave`]

**Data type:** `weavedata`

Weave data describes the weaving that is to take place during the heat and weld phases. Welding without weaving is obtained by not specifying any `weavedata` in the instruction.

Zone

**Data type:** `zonedata`

Zone data defines how close the axes must be to the programmed position before they can start moving towards the next position.

Tool

**Data type:** `tooldata`

The tool in use when the robot moves.

`\WObj`

**Data type:** `wobjdata`

The work object (object coordinate system) to which the robot position in the instruction is related.

Track

**Data type:** `trackdata`

Track data is used and is only applicable when the system is configured for seam tracking with a serial weld guide system or with a Laser Tracker system. Seam tracking is activated when this argument is included in the `ArcL` instruction, but deactivated if it is omitted. The optional `trackdata` argument must be used during the whole weld seam, that is, from the `ArcXStart` to the `ArcXEnd` instruction.

`\SeamName`

**Data type:** `string`

*Continues on next page*

## 10 RAPID reference

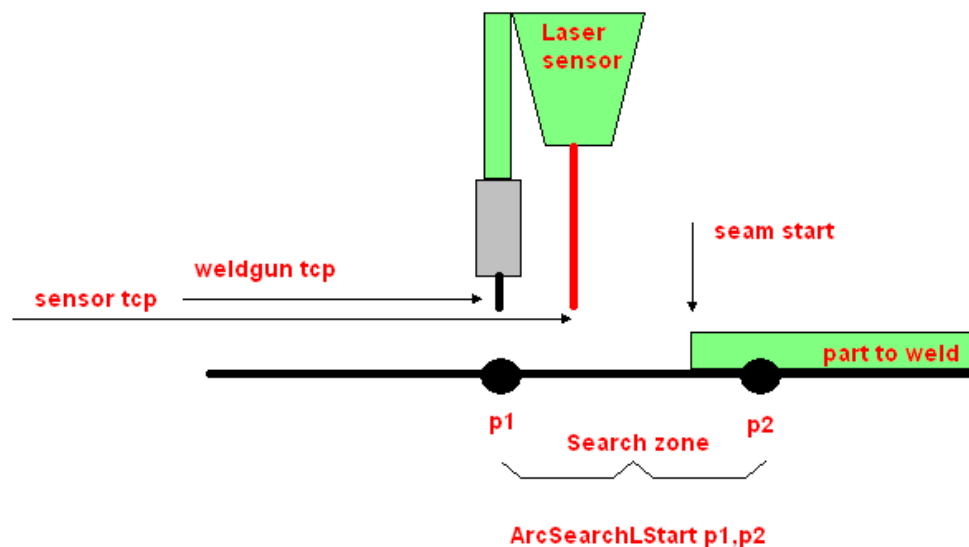
### 10.5.1.1 ArcSearchLStart - Searching with optical sensors for arc welding

*Continued*

The seam name is a string which is added to error logs if an error occurs during the welding sequence. \SeamName is only applicable together with the ArcSearchLStart instruction.

#### Program execution

The search will be performed with the weld gun TCP in the search zone between the positions  $p1$  and  $p2$ . That means that the sensor will be searching a region that is shifted, because the sensor TCP is shifted against the weld gun TCP due to mounting on the robot.



xx1500001683

#### Error handling

The following recoverable errors are generated and can be handled in an error handler. The system variable `ERRNO` will be set to:

AW_START_ERR	Start condition error; torch, gas or water supervision
AW_IGNI_ERR	Ignition error; arc supervision
AW_WELD_ERR	Weld error; arc supervision
AW_EQIP_ERR	Weld equipment error; voltage, current, water or gas supervision during welding
AW_WIRE_ERR	Wire stick error; wire stick supervision
AW_STOP_ERR	Welding interrupted using the stop process input

#### Syntax

```
ArcSearchLStart
[ p1 ':' '=' ] < expression (IN) of robtarg > ','
[ p2 ':' '=' ] < expression (IN) of robtarg >
[ '\ ' ID ':' '=' < expression (IN) of identno > ] ','
[ Speed ':' '=' ] < expression (IN) of speeddata > ','
[ Scan ':' '=' ] < expression (IN) of optscandata > ','
[ Seam ':' '=' ] < persistent (PERS) of seamdata > ','
[ Weld ':' '=' ] < persistent (PERS) of welddata >
```

*Continues on next page*



### 10.5.1.1 ArcSearchLStart - Searching with optical sensors for arc welding

*Continued*

```
[ '\ ' Weave ' := ' < persistent (PERS) of weavedata > ] ', '
[ Zone ' := ' ] < expression (IN) of zonedata > ', '
[ Tool ' := ' ] < persistent (PERS) of tooldata >
[ '\ ' WObj ' := ' < persistent (PERS) of wobjdata > ] ', '
[ Track ' := ' < persistent (PERS) of trackdata > ]
[ '\ ' SeamName ' := ' ] < expression (IN) of string > ';'

```

#### Related information

For information about	See
RobotWare option <i>Arc</i> .	<i>Application manual - Arc and Arc Sensor.</i>

## 10 RAPID reference

---

### 10.5.1.2 CapSensorScan - Searching with a laser tracking sensor

#### 10.5.1.2 CapSensorScan - Searching with a laser tracking sensor

---

##### Usage

CapSensorScan is used together with a laser tracking sensor to search for the start or end of a joint.

This RAPID instruction is a tool from the CAP toolbox. The instruction is a logical RAPID instruction and not a movement instruction, which means that the scanning is started and runs until it either has found the joint, is stopped by a new CapSensorScan with \Reset, or an error occurs.

The signal status of the output signal is neither checked by the instruction nor reset. That has to be done in the RAPID program.

CapSensorScan is designed to be used together with SearchL/C.

---

##### Basic examples

The following example illustrates the instruction CapSensorScan.

##### Example 1

```
! cross-connect doSeamFound and diSeamTrig
CapSensorScan nJointNo, posFound, doSeamFound \nValid:=5
    \MaxDeviation:=0.5;
MoveJ ...
SearchL\Stop,diSeamTrig,pSearchStop,p2,v10,gun1;
```

---

##### Arguments

```
CapSensorScan joint_no, corrpos, sigdo, [\nValid], [\maxDeviation],
[\Reset], [\Inverted]
```

joint\_no

**Data type:** num

Sensor joint number to be used for scanning.

corrpos

**Data type:** pos

Deviation of the found position, pos only, in sensor tool coordinates of the used sensor.

sigdo

**Data type:** signaldo

This signal is set when the scanning criteria is met. Either after one measurement, or a series of measurements in sequence if \nValid is used.

The measurements must be valid if \Inverted is used, and invalid if \Inverted is not used.

[\nValid]

**Data type:** num

Specifies how many sensor measurements must be valid if \Inverted is used, and invalid if \Inverted is not used.

*Continues on next page*

[\maxDeviation]

Data type: num

Not implemented.

[\Reset]

Data type: switch

If this switch is present an ongoing scan is stopped.

[\Inverted]

Data type: switch

The measurements must be valid if \Inverted is used, and invalid if \Inverted is not used.

### Program execution

CapSensorScan is a modal RAPID instruction. That means that the system starts the sensor scanning process but it does not carry out any movement itself. A Search instruction must be used for the movement. When the scanning criteria is satisfied, the digital output signal specified in the instruction is set.

### Error handling

The following recoverable errors are generated and can be handled in an error handler. The system variable `ERRNO` will be set to:

CAP_SEN_GENERRO	A general sensor error occurred.
CAP_SEN_BUSY	The sensor is busy and cannot answer the request.
CAP_SEN_UNKNOWN	The command sent to the sensor is unknown to sensor.
CAP_SEN_ILLEGAL	The variable or block number sent to the sensor is illegal.
CAP_SEN_EXALARM	An external alarm occurred in the sensor.
CAP_SEN_CAALARM	A camera alarm occurred in the sensor.
CAP_SEN_TEMP	The sensor temperature is out of range.
CAP_SEN_VALUE	The value sent to the sensor is out of range.
CAP_SEN_CAMCHECK	The camera check failed.

### Syntax

CapSensorScan

```
[ joint_no ':' '=' ] < expression (IN) of num > ','
[ corpos ':' '=' ] < expression (INOUT) of pos > ','
[ sigdo ':' '=' ] < variable (VAR) of signaldo > ','
[ '\ ' nValid ] ','
[ '\ ' maxDeviation ] ','
[ '\ ' Reset ] ','
[ '\ ' Inverted ] ';'

```

*Continues on next page*

## 10 RAPID reference

---

### 10.5.1.2 CapSensorScan - Searching with a laser tracking sensor

*Continued*

---

#### Related information

For information about	See
RobotWare option <i>Continuous Application Platform (CAP)</i> .	<i>Application manual - Continuous Application Platform</i>
RAPID components for <i>CAP</i> .	<i>Technical reference manual - RAPID Instructions, Functions and Data types</i>

### 10.5.1.3 OptSearch\_1D - Searching with optical sensors for Arc

#### Usage

OptSearch\_1D is used to search with optical sensors.

This instruction uses the CAP instruction CapSensorScan to get measurements from the sensor and combines it with the SearchL functionality as to provide a ready-to-use instruction for searching with optical sensors.

#### Basic examples

The following example illustrates the instruction OptSearch\_1D.

##### Example 1

```
Scan.v_search:=10;
Scan.valid_readings:=5;
Scan.x_offs:=0;
Scan.y_offs:=0;
Scan.z_offs:=0;

OptSearch_1D resultPose \SearchStop:=pSeamStart, pStart, pSearch,
    v100, gun1 \SchSpeed:=v10, Scan, Track;
ArcLStart pSeamStart, v10, Seam, Weld, fine, gun1\track:=Track;
```

#### Arguments

```
OptSearch_1D \NotOff Result \SearchStop, StartPoint, SearchPoint,
    Speed, Tool \WObj \PrePDisp \Limit \SchSpeed \SearchName,
    Scan, Track \TLoad
```

\NotOff

**Data type:** switch

The digital signal that is used to trigger SearchL when the search was successful, is set low after the search if \NotOff is present.

Result

**Data type:** pose

Returned displacement of seam.

\SearchStop

**Data type:** robtarget

Returned seam start position.

StartPoint

**Data type:** robtarget

Start point of searching.

SearchPoint

**Data type:** robtarget

Search point, nominal point of seam.

*Continues on next page*

## 10 RAPID reference

---

### 10.5.1.3 OptSearch\_1D - Searching with optical sensors for Arc *Continued*

Speed

**Data type:** speeddata

The speed data that applies to movements.

Tool

**Data type:** tooldata

The tool in use when the robot moves.

\WObj

**Data type:** wobjdata

The work object (object coordinate system) to which the robot position in the instruction is related.

\PrePDisp

**Data type:** pose

Optional displacement of both StartPoint and SearchPoint.

\Limit

**Data type:** num

Not used.

\SchSpeed

**Data type:** num

Special search speed.

\SearchName

**Data type:** string

Not used.

Scan

**Data type:** optscandata

Data that defines how the search is performed.

Track

**Data type:** trackdata

trackdata is used to control the optical sensor.

[\TLoad]

**Total load**

**Data type:** loaddata

The \TLoad argument describes the total load used in the movement. The total load is the tool load together with the payload that the tool is carrying. If the \TLoad argument is used, then the loaddata in the current tooldata is not considered.

If the \TLoad argument is set to load0, then the \TLoad argument is not considered and the loaddata in the current tooldata is used instead.

For a complete description of the argument [\TLoad], see the instruction MoveL in *Technical reference manual - RAPID Instructions, Functions and Data types*.

*Continues on next page*

**Program execution**

OptSearch\_1D starts the sensor for scanning and moves the robot along the path that was programmed for scanning for the configured joint type. When the specified criteria (optscandata) is met, the movement is stopped and the robtarget of the position that was found is returned.

**Syntax**

```
OptSearch_1D
[ '\ ' NotOff ]
[ Result ':= ' ] < expression (INOUT) of pose >
[ '\ ' SearchStop ':= ' < var or pers (INOUT) of robtarget > ] ', '
[ StartPoint ':= ' ] < var or pers (INOUT) of robtarget > ', '
[ SearchPoint ':= ' ] < var or pers (INOUT) of robtarget > ', '
[ Speed ':= ' ] < expression (IN) of speeddata > ', '
[ Tool ':= ' ] < persistent (PERS) of tooldata > ', '
[ '\ ' WObj ':= ' < persistent (PERS) of wobjdata > ]
[ '\ ' PrePDisp ':= ' < expression (INOUT) of pose > ]
[ '\ ' Limit ':= ' < expression (IN) of num > ]
[ '\ ' SchSpeed ':= ' < expression (IN) of num > ]
[ '\ ' SearchName ':= ' < expression (IN) of string > ]
[ Scan ':= ' ] < expression (IN) of optscandata > ', '
[ Track ':= ' ] < expression (IN) of trackdata >
[ '\ ' TLoad ':= ' < persistent (PERS) of loaddata > ] ';'

```

**Related information**

For information about	See
RobotWare option Arc.	<i>Application manual - Arc and Arc Sensor.</i>

## 10 RAPID reference

---

### 10.5.2.1 optscandata - Scan data

## 10.5.2 Data types

### 10.5.2.1 optscandata - Scan data

---

#### Usage

`optscandata` defines all data that are necessary to perform a seam start search with `OptSearch_1D` or `ArcSearchLStart`.

---

#### Components

`v_search`

**Data type:** num

Search speed [mm/s].

`valid_readings`

**Data type:** num

Number of valid readings from sensor.

`max_dev`

**Data type:** num

Max deviation. Not implemented.

`offset_x`

**Data type:** num

Seam start position offset x [mm].

`offset_y`

**Data type:** num

Seam start position offset y [mm].

`offset_z`

**Data type:** num

Seam start position offset z [mm].

`showUserMessage`

**Data type:** bool

Show user message on/off. A user message is shown before the search is started. This parameter is used in `ArcSearchLStart` only.

---

#### Basic examples

The following example illustrates the data type `optscandata`.

##### Example 1

```
Scan.v_search:=10;  
Scan.valid_readings:=5;  
Scan.x_offs:=0;  
Scan.y_offs:=0;  
Scan.z_offs:=0;
```

*Continues on next page*



```

OptSearch_1D resultPose \SearchStop:=pSeamStart, pStart, pSearch,
v100, gun1 \SchSpeed:=v10, Scan, Track;
ArcLStart pSeamStart, v10, Seam, Weld, fine, gun1\track:=Track;

```

**Limitations**

The components `offset_x`, `offset_y`, and `offset_z` must be within  $\pm 50$  mm.

**Structure**

```

< data object of optscandata >
  < v_search of num >
  < valid_readings of num >
  < max_dev of num >
  < offset_x of num >
  < offset_y of num >
  < offset_z of num >
  < showUserMessage of bool >

```

**Related information**

For information about	See
Searching with optical sensors for arc welding	<a href="#">OptSearch_1D - Searching with optical sensors for Arc on page 85</a>
Searching with optical sensors for arc welding	<a href="#">ArcSearchLStart - Searching with optical sensors for arc welding on page 78</a>

**This page is intentionally left blank**

# 11 System parameters

## 11.1 System parameters for Arc

### About the system parameters

This is a description of the system parameters used by *Arc*. For information about other system parameters, see *Technical reference manual - System parameters*.

### Type Optical Sensor

The type *Optical Sensor* holds parameters for the option *Optical Sensor* and belongs to the topic *Process*.

Parameter	Data type	Note
Name	string	The name of the <i>Optical Sensor</i> .
Use Optical Sensor Class	string	The <i>Optical Sensor Class</i> used by the <i>Arc Sensor Class</i> .
Use Optical Sensor Properties	string	The optical sensor properties used by the <i>Optical Sensor</i> . Two sensor properties are available: MSPOT90 and SCOUT.
Connected to Robot	string	The robot to which the sensor is connected.

### Type Optical Sensor Properties

The type *Optical Sensor Properties* holds parameters for the optical sensor and belongs to the topic *Process*.

Parameter	Data type	Note
Name	string	The name of the <i>Arc Sensor Class</i> .
Sensor Manufacturer	string	The name of the sensor manufacturer.
Track System	num	Defines the TrackSystem type. 1 for LaserTrack
Device	string	The device name used for the tracker. Device must match the transmission protocol name configured in SIO.cfg. "Laser1:" for lasertracker.
Pattern Sync Threshold	num	The coordination position at the extents of the weaving pattern. It is specified as a percentage of the width on either side of the weaving center. When weaving is carried out beyond this point, a digital output signal is automatically set to one. This type of coordination is intended for seam tracking using Through-the-Arc Tracker.

*Continues on next page*

## 11 System parameters

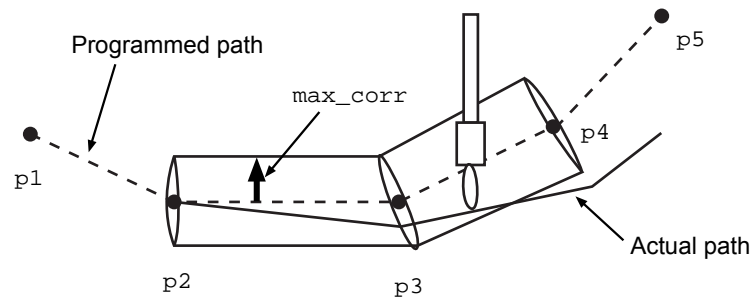
### 11.1 System parameters for Arc

Continued

Parameter	Data type	Note
Max Blind	num	The max_blind component defines the maximum distance the robot is allowed to continue moving under the assumption that the last reported position error is still valid. The parameter should be tuned to match the maximum expected tack lengths used, or the length of other features.  For example, clamps that may prevent the sensor from accurately detect the actual position and geometry of the seam. If the max_blind distance has been exceeded with no new position data from the sensor an error will be reported and program execution is stopped.
Max Corr	num	Not used. The max_corr component in trackdata is used instead.
Adapt Start Delay	num	Not used.
Max Incremental Correction	num	Max incremental correction for the arc tracking system. If the incremental TCP correction is bigger than <i>Max Incremental Correction</i> and <i>Max Correction Warning</i> was set, the robot will continue its path but the applied incremental correction will not exceed <i>Max Incremental Correction</i> . If <i>Max Correction Warning</i> was not set, a track error is reported and program execution is stopped. Default value is 3 mm.
Log File	string	The name of the log file created during tracking.
Sensor Frequency	num	Defines the sample frequency of the sensor used. (e.g. M-Spot-90 has 5Hz sampling frequency)
Ipol Servo Delay	num	Defines the robot controller internal time delay between ipol task and servo task. Use default value: 74 ms
Ipol Correction Gain	num	Defines the gain factor for the correction imposed on ipol. Use default value: 0
Servo Sensor Factor	num	Defines the number of servo corrections per sensor readings. Use default value: 0
Correction Filter	num	Defines filtering of the correction calculated, using mean value over corr filter values. Use default value: 1
Ipol Correction Filter	num	Defines filtering of the ipol correction, using mean value over path filter values. Use default value: 1
Servo Correction	num	Defines filtering of the servo correction, using mean value Filter over path servo filter values. Use default value: 1
Error Ramp In	num	Defines during how many sensor readings ramp in is done after an error caused by sensor reading.
Error Ramp Out	num	Defines during how many sensor readings ramp out is done after an error caused by sensor reading.
CB Angle	num	Defines the angle between a 3D sensor beam and the sensor z-axis. Use default value: 0 for M-Spot-90 and 25 for SCOUT.

Continues on next page

Parameter	Data type	Note
Calib Variable Name	num	The name of the calibration variable name found in the calibration programs. RAPID data type: pose
Calib Variable Offset Name	num	The name of the calibration variable offset name found in the calibration programs. RAPID data type: pos
Max Correction Warning	bool	If this parameter is enabled, program execution is not interrupted, when the limit for maximum correction, specified in the trackdata, is exceeded. Only a warning will be sent. Default value: FALSE
WgLeftSynch	string	Digital output signal for left syncpulse.
WgRightSynch	string	Digital output signal for right syncpulse.
Wg track	string	Not used.



xx1200000686

#### Max correction

## 11 System parameters

### 11.2 System parameters for EGM

### 11.2 System parameters for EGM

#### About the system parameters

This is a brief description of the system parameters used by *Externally Guided Motion*. For more information about the parameters, see *Technical reference manual - System parameters*.

#### Type External Motion Interface Data

The system parameters used by *Externally Guided Motion* belong to the type *External Motion Interface Data* in topic *Motion*.

Parameter	Description
<i>Name</i>	The name of the external motion interface data. This name is referenced by the parameter <i>ExtConfigName</i> in the RAPID instructions <i>EGMSetupAI</i> , <i>EGMSetupAO</i> , <i>EGMSetupGI</i> , and <i>EGMSetupUC</i> .
<i>Level</i>	External motion interface level determines the system level at which the corrections are applied. Level 0 corresponds to raw corrections, added just before the servo controllers. Level 1 applies extra filtering on the correction, but also introduces some extra delays and latency. Level 2 has to be used for path correction.
<i>Do Not Restart After Motors Off</i>	Determines if the external motion interface execution should automatically restart after the controller has been in the motors off state, for instance after emergency stop.
<i>Return to Programmed Position when Stopped</i>	Determines if axes currently running external motion interface should return to the programmed position, when program execution is stopped. If <i>False</i> , axes will stop in their current position. If <i>True</i> , axes will move to the programmed finepoint.
<i>Default Ramp Time</i>	Defines the default total time for stopping external motion interface movements when external motion interface execution is stopped. The value will be used to determine how fast the speed contribution from external motion should be ramped to zero when program execution is stopped, and how fast axes return to the programmed position if <i>Return to Programmed Position when Stopped</i> is <i>True</i> .
<i>Default Proportional Position Gain</i>	Defines the default proportional gain of the external motion interface position feedback control. For more information.
<i>Default Low Pass Filter Bandwidth</i>	Defines the default bandwidth of the low-pass filter used to filter the speed contribution from the external motion interface execution.

# Index

## A

ArcC, 75  
 ArcCEnd, 75  
 ArcCStart, 75  
 ArcL, 75  
 ArcLEnd, 75  
 ArcLStart, 75  
 ArcSearchLStart, 78  
 Arc system parameters, 91

## C

C# API, 64  
 CAP, 10  
   configuring, 30  
 CapC, 74  
 CapL, 74  
 caplatrackdata, 74  
 CapLATrSetup, 74  
 CAP RAPID components, 74–75  
 CapSensorScan, 82  
 captrackdata, 74  
   configuring  
     sensors, 27  
 constants  
   Sensor Interface, 71

## D

data types  
   trackdata, 75

## E

EGM, 10  
   configuring, 33  
 EGM Path Correction, 17  
 EGM RAPID components, 71, 76  
 EGM sensor protocol, 63  
 EGM system parameters, 94  
 External Motion Interface Data, 94

## G

glossary, 10  
 Google C++, 64  
 Google C++ API, 64  
 Google overview, 63  
 Google Protocol Buffers, 63

## I

Installation Manager, 22  
 instructions  
   ArcC, 75  
   ArcCEnd, 75  
   ArcCStart, 75  
   ArcL, 75  
   ArcLEnd, 75  
   ArcLStart, 75  
 interrupt, 71  
 IVarValue, 71

## L

limitations, 9  
 LTAPP, 10, 29  
 LTC, 10

## M

modules  
   Sensor Interface, 71

## N

Name, Transmission Protocol type, 28–29  
 Nanopb, 64

## O

Optical Sensor, 91  
 Optical Tracking Arc  
   configuring, 31  
 optscandata, 88  
 OptSearch\_1D, 85

## P

Protobuf, 63  
 Protobuf-csharp, 64  
 Protobuf-net, 64  
 protocols  
   Ethernet, 29  
   serial channels, 28

## R

ReadVar, 71  
 Remote Address, 29  
 Remote Port, 29  
 RoboCom Light, 29  
 RobotStudio, 22  
 RTP1 protocol, 28  
 RW, 10

## S

SenDevice, 71  
 sensor, 14  
 Sensor Interface, 14  
 sensors  
   configuring, 27  
 Serial Port, Transmission Protocol type, 28–29  
 system parameters  
   Sensor Interface, 28–29

## T

TCP, 10  
 term list, 10  
 trackdata, 75  
 Transmission Protocol, type, 28–29  
 Type, Transmission Protocol type, 28–29

## U

UDP, 63  
 UdpUc, 43  
 Udp Unicast Communication, 43

## W

WriteVar, 71







# Contact us

**ABB AB, Robotics**  
**Robotics and Motion**  
S-721 68 VÄSTERÅS, Sweden  
Telephone +46 (0) 21 344 400

**ABB AS, Robotics**  
**Robotics and Motion**  
Nordlysvegen 7, N-4340 BRYNE, Norway  
Box 265, N-4349 BRYNE, Norway  
Telephone: +47 22 87 2000

**ABB Engineering (Shanghai) Ltd.**  
**Robotics and Motion**  
No. 4528 Kangxin Highway  
PuDong District  
SHANGHAI 201319, China  
Telephone: +86 21 6105 6666

**ABB Inc.**  
**Robotics and Motion**  
1250 Brown Road  
Auburn Hills, MI 48326  
USA  
Telephone: +1 248 391 9000

[www.abb.com/robotics](http://www.abb.com/robotics)