9.3 Externally Guided Motion [689-1]

9.3.1 Introduction to EGM

9.3.1.1 Overview

Purpose

Externally Guided Motion (EGM) offers two different features:

- *EGM Position Guidance*:

  The robot does not follow a programmed path in RAPID but a path generated by an external device.
- *EGM Path Correction*:

  The programmed robot path is modified/corrected using measurements provided by an external device.

EGM Position Guidance

The purpose of *EGM Position Guidance* is to use an external device to generate position data for one or several robots. The robots will be moved to that given position.

Some examples of applications are:

- Place an object (for example a car door or a window) at a location (for example a car body) that was given by an external sensor.
- Bin picking. Pick objects from a bin using an external sensor to identify the object and its position.

EGM Path Correction

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.

Some examples of applications are:

- Seam tracking.
- Tracking of objects moving near a known path.

What is included

The RobotWare option *Externally Guided Motion* gives you access to:

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

Continues on next page

Limitations

Limitations for EGM Position Guidance

- It is not possible to perform linear movements using EGM Position Guidance, since EGM Position Guidance does not contain interpolator functionality. The actual path of the robot will depend on the robot configuration, the start position, and the generated position data.
- EGM Position Guidance does not support MultiMove.
- It is not possible to use EGM Position Guidance to guide a mechanical unit in a moving work object.
- If the robot ends up near a singularity, i.e. when two robot axis are nearly parallel, the robot movement will be stopped with an error message. In that situation the only way is to jog the robot out of the singularity.
-

Limitations for EGM Path Correction

- 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).

Common limitations for EGM

- EGM 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 movement has to start in a fine point.
- Only one external device can be used for each robot to provide correction data.

9.3.1.2 Introduction to EGM Position Guidance

What is EGM Position Guidance

EGM Position Guidance is designed for advanced users and provides a low level interface to the robot controller, by by-passing the path planning that can be used when highly responsive robot movements are needed. EGM Position Guidance can be used to read positions from and write positions to the motion system at a high rate. This can be done every 4 ms with a control lag of 10–20 ms depending on the robot type. The references can either be specified using joint values or a pose. The pose can be defined in any work object that is not moved during the EGM Position Guidance movement.

All necessary filtering, supervision of references, and state handling is handled by EGM Position Guidance. Examples of state handling are program start/stop, emergency stop, etc.

The main advantage of EGM Position Guidance is the high rate and low delay/latency compared to other means of external motion control. The time between writing a new position until that given position starts to affect the actual robot position, is usually around 20 ms.

EGM handles *Absolute Accuracy*.

What EGM Position Guidance does not do

EGM goes directly into the motor reference generation, i.e. it does not provide any path planning. This means that you cannot order a movement to a pose target and expect a linear movement. It is not possible either to order a movement with a specified speed or order a movement that is supposed to take a specified time.

For ordering such movements path planning is needed and we refer you to the standard movement instructions in RAPID, i.e. `MoveL`, `MoveJ`, etc.

WARNING

Since the path planning is by-passed by EGM in the robot controller, the robot path is created directly from user input. It is therefore important to make sure that the stream of position references sent to the controller is as smooth as possible. The robot will react quickly to all position references sent to the controller, also faulty ones.

9.3.1.3 Introduction to EGM Path Correction

What is EGM Path Correction

EGM Path Correction gives the user the possibility to correct a programmed robot path. The device or sensor that is used to measure the actual path has to be mounted on the tool flange of the robot and it must be possible to calibrate the sensor frame.

The corrections are performed in the path coordinate system, which gets its x-axis from the tangent of the path, the y-axis is the cross product of the path tangent, and the z-direction of the active tool frame and the z-axis is the cross product of x-axis and y-axis.

EGM Path correction has to start and end in a fine point. The sensor measurements can be provided at multiples of about 48 ms.

9.3.2 Using EGM

9.3.2.1 Basic approach

Basic approach for EGM Position Guidance

This is the general approach to move/guide a robot using an external device (sensor) to give the target for the movement.

	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 `EGM_STATE_DISCONNECTED`.
3	Call an EGM setup instruction to set up the position data source using signals or UdpUc protocol connection. The EGM state changes to `EGM_STATE_CONNECTED`.
4	Choose if the position is given as joint values or as a pose and give the position convergence criteria, i.e. when the position is considered to be reached.
5	If pose was chosen, define which frames are used to define the target position and in which frame the movement is to be applied.
6	Give the stop mode, an optional time-out and perform the movement itself. Now the EGM state is `EGM_STATE_RUNNING`. This is when the robot is moving.
7	The EGM movement will stop when the position is considered to be reached, i.e. the convergence criteria is fulfilled. Now the EGM state has changed back to `EGM_STATE_CONNECTED`.

Basic approach for EGM Path Correction

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 `EGM_STATE_DISCONNECTED`.
3	Call an EGM setup instruction to set up the position data source using signals or UdpUc protocol connection. The EGM state changes to `EGM_STATE_CONNECTED`.
4	Define the sensor correction frame, which always is a tool frame.
5	Perform the movement itself. Now the EGM state is `EGM_STATE_RUNNING`.
	At the next fine point EGM will return to the state `EGM_STATE_CONNECTED`.
6	To free an EGM identity for use with another sensor you have to reset EGM, which returns EGM to the state `EGM_STATE_DISCONNECTED`.

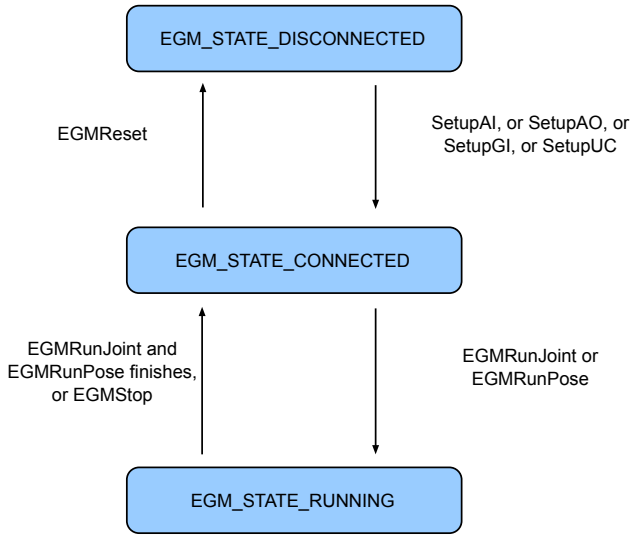
9.3.2.2 Execution states

Description

The EGM process has different states:

Value	Description
EGM_STATE_DISCONNECTED	The EGM state of the specific process is undefined. No setup is active.
EGM_STATE_CONNECTED	The specified EGM process is not activated. Setup has been made, but no EGM movement is active.
EGM_STATE_RUNNING	The specified EGM process is running. The EGM movement is active, i.e. the robot is moved.

Transitions between the different states are according to the figure below.



xx1400001082

The RAPID instructions `EGMRunJoint` and `EGMRunPose` start from `EGM_STATE_CONNECTED` and change the state to `EGM_STATE_RUNNING` as long as the convergence criteria for the target position have not been met or the timeout time has not expired. When one of these conditions is met, the EGM state is changed to `EGM_STATE_CONNECTED` again and the instruction ends, i.e. RAPID execution continues to the next instruction.

If EGM has the state `EGM_STATE_RUNNING` and RAPID execution is stopped, EGM enters the state `EGM_STATE_CONNECTED`. At program restart, EGM returns to the state `EGM_STATE_RUNNING`.

If the program pointer is moved using **PP to Main** or **PP to cursor**, the EGM state is changed to `EGM_STATE_CONNECTED`, if the state was `EGM_STATE_RUNNING`.

9.3.2.3 Input data

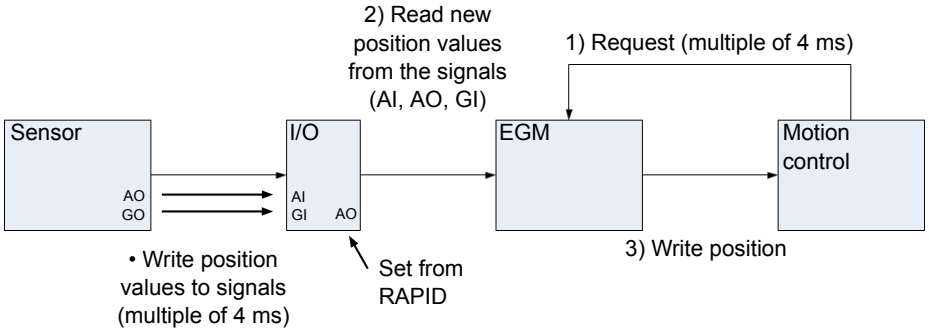
Input data for EGM Position Guidance

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 either as joints or as a pose, i.e. Cartesian position plus orientation.

The data flow for the signal interface is illustrated below:



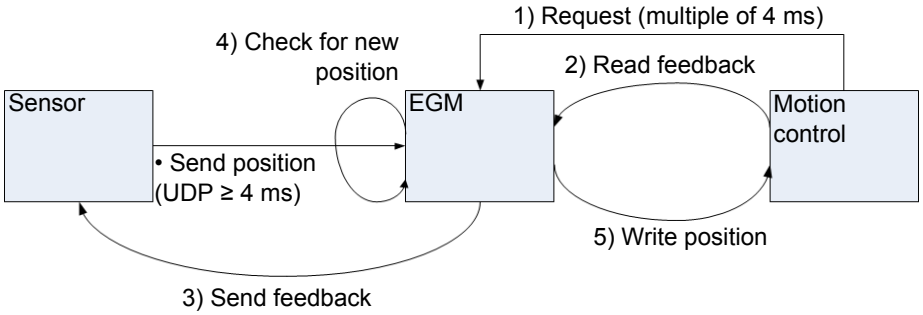
xx1400002016

1 Motion control calls EGM.
2 EGM reads the position values from the signals.
3 EGM writes the position data to motion control.
- The sensor writes position data to the signals.

If signals are used as data source, the input is limited to 6 for the robot, i.e. 6 joint values or 3 Cartesian position values (x, y, z) plus 3 Euler angle values (rx, ry, rz), and up to 6 values for additional axes.

Continues on next page

The data flow for the UdpUc interface is illustrated below:



xx1400002017

1 Motion control calls EGM.
2 EGM reads feedback data from motion control.
3 EGM sends feedback data to the sensor.
4 EGM checks the UDP queue for messages from the sensor.
5 If there is a message, EGM reads the next message and step 5 writes the position data to motion control. If no position data had been sent, motion control continues to use the latest position data previously written by EGM.
- The sensor sends position data to the controller (EGM). Our recommendation is to couple this to step 3. Then the sensor will be in phase with the controller.

The control loop is based on the following relation between speed and position:

*speed = k * (pos_ref – pos) + speed_ref*	*k* - factor *pos_ref* - reference position *pos* - desired position *speed_ref* - reference speed

For instructions on how to implement the UdpUc protocol for an external device, see *The EGM sensor protocol on page 339*. There you will also find a description of input data.

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*).

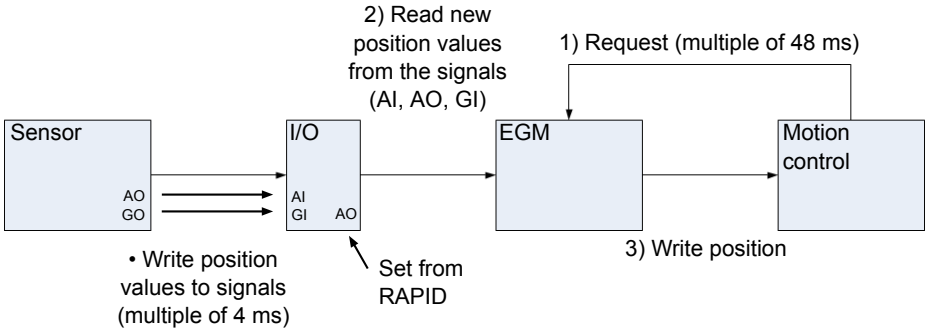
Instructions	Description
`EGMSetupAI`	Setup analog input signals for EGM
`EGMSetupAO`	Setup analog output signals for EGM
`EGMSetupGI`	Setup group input signals for EGM
`EGMSetupUC`	Setup the UdpUc protocol for EGM

Input data for EGM contain mainly position data.

Continues on next page

9.3.2.3 Input data
Continued

The data flow for the signal interface is illustrated below:



xx1400002016

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.

9.3.2.4 Output data

Description

Output data is only available for the UdpUc interface.

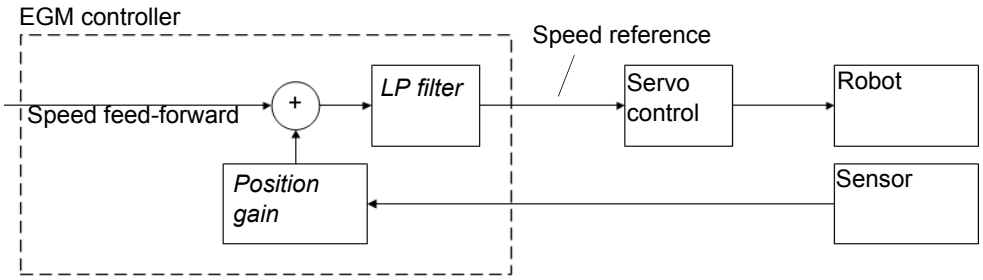
For instructions on how to implement the UdpUc protocol for an external device, see *The EGM sensor protocol on page 339*. There you will also find a description of output data.

9.3.2.5 Configuration

Configuration for EGM Position Guidance

EGM behavior can be influenced using the system parameters of type *External Motion Interface Data* topic *Motion*. For a description of all available EGM parameters, see *System parameters on page 343*.

Here follows a closer description of the two parameters that influence the EGM control loop. The figure shows a simplified view of the EGM control system.



xx1400001083

Default proportional Position Gain	The parameter *Position gain* in the figure influences the responsiveness moving to the target position, given by the sensor, in relation to the current robot position. The higher the value, the faster the response.
Default Low Pass Filter Bandwith Time	The parameter *LP Filter* in the figure is the default value used to filter the speed contribution from EGM.

9.3.2.6 Frames

Frames for EGM Position Guidance

EGM can be run in two different modes, joint mode and pose mode. The following section applies to the EGM pose mode only.

For the joint mode there is no need for reference frames, because both sensor values and position values are axis angles given in degrees relative to the calibration position of each axis. But for the pose mode reference frames are necessary. Measurements from the sensor and directions for position change can only be given relative to reference frames.

The RAPID instruction `EGMActPose` defines all frames that are available in EGM:

Frame	Description
Tool	The tool data to be used for the EGM process is defined with the optional `\Tool` argument.
Work object	The work object data used for the EGM process is defined with the optional `\Wobj` argument.
Correction	The frame to be used to give the final movement direction is defined by the mandatory `CorrFrame` argument.
Sensor	The frame to be used to interpret the sensor data is defined by the mandatory `SensorFrame` argument.

Tools and work objects

The tool and the work object may be defined in two combinations only:

1. If the tool is attached to the robot, the work object has to be fixed.
2. If the tool is fixed, the work object has to be attached to the robot.

Note

It is not possible to use a work object or tool that is attached to any other mechanical unit than the EGM robot.

Predefined frame types

For the frames `CorrFrame` and `SensorFrame` it is also necessary to know what they are related to. This information is specified using the predefined frame types in the data type egmframetype:

Value	Description
`EGM_FRAME_BASE`	The frame is defined relative to the base frame (pose mode).
`EGM_FRAME_TOOL`	The frame is defined relative to `tool0` (pose mode).
`EGM_FRAME_WOBJ`	The frame is defined relative to the active work object (pose mode).
`EGM_FRAME_WORLD`	The frame is defined relative to the world frame (pose mode).
`EGM_FRAME_JOINT`	The values are joint values (joint mode).

Frames for EGM Path Correction

EGM can be run in two different modes, joint mode and pose mode. The following section applies to the EGM pose mode only.

Continues on next page

9.3.2.6 Frames
Continued

For the joint mode there is no need for reference frames, because both sensor values and position values are axis angles given in degrees relative to the calibration position of each axis. But for the pose mode reference frames are necessary. Measurements from the sensor and directions for position change can only be given relative to reference frames.

The RAPID instruction `EGMActPose` defines all frames that are available in EGM:

Frame	Description
Tool	The tool data to be used for the EGM process is defined with the optional `\Tool` argument.
Work object	The work object data used for the EGM process is defined with the optional `\Wobj` argument.
Correction	The frame to be used to give the final movement direction is defined by the mandatory `CorrFrame` argument.
Sensor	The frame to be used to interpret the sensor data is defined by the mandatory `SensorFrame` argument.

Tools and work objects

The tool and the work object may be defined in two combinations only:

1. If the tool is attached to the robot, the work object has to be fixed.
2. If the tool is fixed, the work object has to be attached to the robot.

Note

It is not possible to use a work object or tool that is attached to any other mechanical unit than the EGM robot.

Predefined frame types

For the frames `CorrFrame` and `SensorFrame` it is also necessary to know what they are related to. This information is specified using the predefined frame types in the data type egmframetype:

Value	Description
`EGM_FRAME_BASE`	The frame is defined relative to the base frame (pose mode).
`EGM_FRAME_TOOL`	The frame is defined relative to `tool0` (pose mode).
`EGM_FRAME_WOBJ`	The frame is defined relative to the active work object (pose mode).
`EGM_FRAME_WORLD`	The frame is defined relative to the world frame (pose mode).
`EGM_FRAME_JOINT`	The values are joint values (joint mode).

9.3.3 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

9.3.3 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.

See the *egm.proto* file for a description of `EgmRobot` and `EgmSensor`, *UdpUc code examples on page 355*.

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: https://code.google.com/p/protobuf-csharp-port/.
2	Unpack the zip-file.
3	Copy the *egm.proto* file to a sub catalogue where protobuf-csharp was un-zipped, e.g. *~\protobuf-csharp\tools\egm*.
4	Start a Windows console in the tools directory, e.g. *~\protobuf-csharp\tools*.
5	Generate an EGM C# file (*egm.cs*) from the *egm.proto* file by typing in the Windows console: `protogen .\egm\egm.proto --proto_path=.\egm`
6	Create a C# console application in Visual Studio. Create a C# Windows console application in Visual Studio, e.g. *EgmSensorApp*.
7	Install NuGet, in Visual Studio, click **Tools** and then **Extension Manager**. Go to **Online**, find the *NuGet Package Manager extension* and click **Download**.
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, **Tools**, **Nuget Package Manager**, and **Package Manager Console**. Type *PM>Install-Package Google.ProtocolBuffers*
10	Add the generated file *egm.cs* to the Visual Studio project (add existing item).
11	Copy the example code into the Visual Studio Windows Console application file (*EgmSensorApp.cpp*) 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

9.3.3 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, `protoc --cpp_out=. egm.proto`
2	Create a win32 console application
3	Add Protobuf source as include directory.
4	Add the generated *egm.pb.cc* file to the project, exclude the file from precompile headers.
5	Copy the code from the *egm-sensor.cpp* file, see *UdpUc code examples on page 355*.
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
Name	The name of the transmission protocol. For example *EGMsensor*.
Type	The type of transmission protocol. It has to be *UDPUC*.
Serial Port	The name of the serial port that will be used for the sensor. This refers to the parameter *Name* in the type *Serial Port*. For IP based transmission protocols (i.e. *Type* has value TCP/IP, SOCKDEV, LTAPPTCP or UDPUC), *Serial Port* is not used and has the value N/A.
Remote Address	The IP address of the remote device.
Remote Port Number	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 to guide a robot. For more information, see *Using EGM Position Guidance with an UdpUc device on page 346*.

9.3.4 System parameters

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
Name	The name of the external motion interface data. This name is referenced by the parameter *ExtConfigName* in the RAPID instructions `EGMSetupAI`, `EGMSetupAO`, `EGMSetupGI`, and `EGMSetupUC`.
Level	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.
Do Not Restart After Motors Off	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.
Return to Programmed Position when Stopped	Determines if axes currently running external motion interface should return to the programmed position, when program execution is stopped. If *False*, axes will stop in their current position. If *True*, axes will move to the programmed finepoint.
Default Ramp Time	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 *Return to Programmed Position when Stopped* is *True*.
Default Proportional Position Gain	Defines the default proportional gain of the external motion interface position feedback control. For more information, see *Configuration on page 336*.
Default Low Pass Filter Bandwidth	Defines the default bandwidth of the low-pass filter used to filter the speed contribution from the external motion interface execution. For more information, see *Configuration on page 336*.

9.3.5 RAPID components

About the RAPID components

This is an overview of all instructions, functions, and data types in *Externally Guided Motion*.

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

Instructions

Instructions	Description
EGMActJoint	EGMActJoint activates a specific EGM process and defines static data for the sensor guided joint movement, i.e. data that is not changed frequently between different EGM movements.
EGMActMove	EGMActMove is used to activate a specific EGM process and defines static data for the movement with path correction, i.e. data that is not changed frequently between different EGM path correction movements.
EGMActPose	EGMActPose activates a specific EGM process and defines static data for the sensor guided pose movement, i.e. data that is not changed frequently between different EGM movements.
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.
EGMRunJoint	EGMRunJoint performs a sensor guided joint movement from a fine point for a specific EGM process (EGMid) and defines which joints will be moved.
EGMRunPose	EGMRunPose performs a sensor guided pose movement from a fine point for a specific EGM process (EGMid) and defines which directions and orientations will be changed.
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.

Continues on next page

Instructions	Description
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).

Data types

Data types	Description
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.

9.3.6 RAPID code examples

9.3.6.1 Using EGM Position Guidance with an UdpUc device

Description

The device which provides the input data for EGM, first has to be configured as an UdpUc device. See *Configuring UdpUc devices on page 342*.

Now the device can be used by EGM to guide a robot. A simple example is the following:

Example

```
MODULE EGM_test
VAR egmident egmID1;
VAR egmstate egmSt1;

! limits for cartesian convergence: +-1 mm
CONST egm_minmax egm_minmax_lin1:=[-1,1];
! limits for orientation convergence: +-2 degrees
CONST egm_minmax egm_minmax_rot1:=[-2,2];

! Start position
CONST jointtarget
     jpos10:=[[0,0,0,0,40,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
! Used tool
TASK PERS tooldata tFroniusCMT:=[TRUE,[[12.3313,-0.108707,416.142],
     [0.903899,-0.00320735,0.427666,0.00765917]],
     [2.6,[-111.1,24.6,386.6],[1,0,0,0],0,0,0.072]];
! corr-frame: wobj, sens-frame: wobj
TASK PERS wobjdata wobj_EGM1:=[FALSE,TRUE,"",
     [[150,1320,1140],[1,0,0,0]], [[0,0,0],[1,0,0,0]]];
! Correction frame offset: none
VAR pose corr_frame_offs:=[[0,0,0],[1,0,0,0]];

PROC main()
! Move to start position. Fine point is demanded.
MoveAbsJ jpos10\NoEOffs, v1000, fine, tFroniusCMT;
testuc;
ENDPROC

PROC testuc()
EGMReset egmID1;
EGMGetId egmID1;

egmSt1:=EGMGetState(egmID1);
TPWrite "EGM state: "\Num:=egmSt1;

IF egmSt1 <= EGM_STATE_CONNECTED THEN
! Set up the EGM data source: UdpUc server using device "EGMsensor:"
     and
```

Continues on next page

```
! configuration "default"
EGMSetupUC ROB_1, egmID1, "default", "EGMsensor:"\pose;
ENDIF

! Correction frame is the World coordinate system and the sensor
     measurements are relative
! to the tool frame of the used tool (tFroniusCMT)
EGMActPose egmID1\Tool:=tFroniusCMT, corr_frame_offs,
     EGM_FRAME_WORLD, tFroniusCMT.tframe, EGM_FRAME_TOOL
     \x:=egm_minmax_lin1 \y:=egm_minmax_lin1 \z:=egm_minmax_lin1
     \rx:=egm_minmax_rot1 \ry:=egm_minmax_rot1 \rz:=egm_minmax_rot1
     \LpFilter:=20;
! Run: the convergence condition has to be fulfilled during 2
     seconds before RAPID
! executeion continues to the next instruction
EGMRunPose egmID1, EGM_STOP_HOLD \x \y \z \CondTime:=2
     \RampInTime:=0.05;

egmSt1:=EGMGetState(egmID1);
IF egmSt1 = EGM_STATE_CONNECTED THEN
TPWrite "Reset EGM instance egmID1";
EGMReset egmID1;
ENDIF
ENDPROC
ENDMODULE
```

9.3.6.2 Using EGM Position Guidance with signals as input

Description

All signals that are used together with EGM has to be defined in the I/O configuration of the system. I.e. the signals that are set up with `EGMSetupAI`, `EGMSetupAO`, or `EGMSetupGI`. After that, the signals can be used by EGM to guide a robot.

The following RAPID program example uses analog output signals as input. The main reason for analog output signals is, that they are easier to simulate than analog input signals. In a real application group input signals and analog input signals might be more common.

In the example we also set the analog output signals to a constant value before the `EGMRun` instruction just for simplicity. Normally an external device will update the signal values to give the desired robot positions.

Example

```
MODULE EGM_test
VAR egmident egmID1;
VAR egmident egmID2;

CONST egm_minmax egm_minmax_lin1:=[-1,1];
CONST egm_minmax egm_minmax_rot1:=[-2,2];
CONST egm_minmax egm_minmax_joint1:=[-0.1,0.1];

CONST robtarget p20:=[[150,1320,1140],
     [0.000494947,0.662278,-0.749217,-0.00783173], [0,0,-1,0],
     [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget p30:=[[114.50,1005.42,1410.38],
     [0.322151,-0.601023,0.672381,0.287914], [0,0,-1,0],
     [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST jointtarget
     jpos10:=[[0,0,0,0,35,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST pose posecor:=[[1200,400,900],[1,0,0,0]];
CONST pose posesens:=[[12.3313,-0.108707,416.142],
     [0.903899,-0.00320735,0.427666,0.00765917]];

! corr-frame: world, sens-frame: world
VAR pose posecor0:=[[0,0,0],[1,0,0,0]];
VAR pose posesen0:=[[0,0,0],[1,0,0,0]];

TASK PERS tooldata tFroniusCMT:=[TRUE,[[12.3313,-0.108707,416.142],
     [0.903899,-0.00320735,0.427666,0.00765917]],
     [2.6,[-111.1,24.6,386.6],[1,0,0,0],0,0,0.072]];
TASK PERS loaddata load1:=[5,[0,1,0],[1,0,0,0],0,0,0];
! corr-frame: wobj, sens-frame: wobj
TASK PERS wobjdata
     wobj_EGM1:=[FALSE,TRUE,"",[[150,1320,1140],[1,0,0,0]],
     [[0,0,0],[1,0,0,0]]];
VAR pose posecor1:=[[0,0,0],[1,0,0,0]];
VAR pose posesen1:=[[0,0,0],[1,0,0,0]];
```

Continues on next page

```
TASK PERS wobjdata
     wobj_EGM2:=[FALSE,TRUE,"",[[0,1000,1000],[1,0,0,0]],
     [[0,0,0],[1,0,0,0]]];
VAR pose posecor2:=[[150,320,0],[1,0,0,0]];
VAR pose posesen2:=[[150,320,0],[1,0,0,0]];

PROC main()
MoveAbsJ jpos10\NoEOffs, v1000, fine, tFroniusCMT;
testAO;
ENDPROC

PROC testAO()
! Get two different EGM identities. They will be used for two
     different eGM setups.
EGMGetId egmID1;
EGMGetId egmID2;

! Set up the EGM data source: Analog output signals and
     configuration "default"
! One guidance using Pose mode and one using Joint mode
EGMSetupAO ROB_1,egmID1,"default" \Pose \aoR1x:=ao_MoveX
     \aoR2y:=ao_MoveY \aoR3z:=ao_MoveZ \aoR5ry:=ao_RotY
     \aoR6rz:=ao_RotZ;
EGMSetupAO ROB_1,egmID2,"default" \Joint \aoR1x:=ao_MoveX
     \aoR2y:=ao_MoveY \aoR3z:=ao_MoveZ \aoR4rx:=ao_RotX
     \aoR5ry:=ao_RotY \aoR6rz:=ao_RotZ;

! Move to the starting point - fine point is needed.
MoveJ p30, v1000, fine, tool0;
! Set the signals
SetAO ao_MoveX, 150;
SetAO ao_MoveY, 1320;
SetAO ao_MoveZ, 900;
! Correction frame is the World coordinate system and the sensor
     measurements are also relative to the world frame
! No offset is defined (posecor0 and posesen0)
EGMActPose egmID1 \Tool:=tFroniusCMT \WObj:=wobj0 \TLoad:=load1,
     posecor0, EGM_FRAME_WORLD, posesen0, EGM_FRAME_WORLD
     \x:=egm_minmax_lin1 \y:=egm_minmax_lin1 \z:=egm_minmax_lin1
     \rx:=egm_minmax_rot1 \ry:=egm_minmax_rot1 \rz:=egm_minmax_rot1
     \LpFilter:=20 \SampleRate:=16 \MaxPosDeviation:=1000;
! Run: keep the end position without returning to the start position
EGMRunPose egmID1,
     EGM_STOP_HOLD\x\y\z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p20, v1000, fine, tFroniusCMT;
! Set the signals
SetAO ao_MoveX, 150;
SetAO ao_MoveY, 1320;
SetAO ao_MoveZ, 1100;
```

Continues on next page

9.3.6.2 Using EGM Position Guidance with signals as input
Continued

```
! Run with the same frame definitions: ramp down to the start
     position after having reached
! the EGM end position
EGMRunPose egmID1,
     EGM_STOP_RAMP_DOWN\x\y\z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p30, v1000, fine, tool0;
! Set the signals
SetAO ao_MoveX, 50;
SetAO ao_MoveY, -20;
SetAO ao_MoveZ, -20;
! Correction frame is the Work object wobj_EGM1 and the sensor
     measurements are also
! relative to the same work object. No offset is defined (posecor1
     and posesen1)
EGMActPose egmID1 \Tool:=tFroniusCMT \WObj:=wobj_EGM1 \TLoad:=load1,
     posecor1, EGM_FRAME_WOBJ, posesen1, EGM_FRAME_WOBJ
     \x:=egm_minmax_lin1 \y:=egm_minmax_lin1 \z:=egm_minmax_lin1
     \rx:=egm_minmax_rot1 \ry:=egm_minmax_rot1 \rz:=egm_minmax_rot1
     \LpFilter:=20;
! Run: keep the end position without returning to the start position
EGMRunPose egmID1,
     EGM_STOP_HOLD\x\y\z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p20, v1000, fine, tFroniusCMT;
! Set the signals
SetAO ao_MoveX, 0;
SetAO ao_MoveY, 0;
SetAO ao_MoveZ, 0;
! Correction frame is the Work object wobj_EGM2 and the sensor
     measurements are also
! relative to the same work object. This time an offset is defined
     for the correction frame
! (posecor2), and for the sensor frame (posesen2)
EGMActPose egmID1 \Tool:=tFroniusCMT \WObj:=wobj_EGM2 \TLoad:=load1,
     posecor2, EGM_FRAME_WOBJ, posesen2, EGM_FRAME_WOBJ
     \x:=egm_minmax_lin1 \y:=egm_minmax_lin1 \z:=egm_minmax_lin1
     \rx:=egm_minmax_rot1 \ry:=egm_minmax_rot1 \rz:=egm_minmax_rot1
     \LpFilter:=20;
! Run: keep the end position without returning to the start position
EGMRunPose egmID1,
     EGM_STOP_HOLD\x\y\z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p20, v1000, fine, tFroniusCMT;
! Set the signals
SetAO ao_MoveX, 0;
SetAO ao_MoveY, 0;
SetAO ao_MoveZ, 0;
```

Continues on next page

```
! Correction frame is of tool type and the sensor measurements are
     relative to the work
! object wobj_EGM2. This time an offset is defined for the
     correction frame (posecor2), and
! for the sensor frame (posesen2)
EGMActPose egmID1 \Tool:=tFroniusCMT \WObj:=wobj_EGM2, posecor2,
     EGM_FRAME_TOOL, posesen2, EGM_FRAME_WOBJ \x:=egm_minmax_lin1
     \y:=egm_minmax_lin1 \z:=egm_minmax_lin1 \rx:=egm_minmax_rot1
     \ry:=egm_minmax_rot1 \rz:=egm_minmax_rot1 \LpFilter:=20;
EGMRunPose egmID1,
     EGM_STOP_HOLD\x\y\z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p20, v1000, fine, tFroniusCMT\TLoad:=load1;
! Set the signals
SetAO ao_MoveX, 150;
SetAO ao_MoveY, 1320;
SetAO ao_MoveZ, 1100;
! Same as last, but with tool0 and wobj0
EGMActPose egmID1, posecor2, EGM_FRAME_TOOL, posesen2,
     EGM_FRAME_WOBJ \x:=egm_minmax_lin1 \y:=egm_minmax_lin1
     \z:=egm_minmax_lin1 \rx:=egm_minmax_rot1 \ry:=egm_minmax_rot1
     \rz:=egm_minmax_rot1 \LpFilter:=20;
! Run: keep the end position without returning to the start position
EGMRunPose egmID1,
     EGM_STOP_HOLD\x\y\z\RampInTime:=0.05\PosCorrGain:=1;

! Move to the starting point - fine point is needed.
MoveJ p20, v1000, fine, tFroniusCMT\TLoad:=load1;
! Set the signals
SetAO ao_MoveX, 70;
SetAO ao_MoveY, -5;
SetAO ao_MoveZ, 0;
SetAO ao_RotX, 0;
SetAO ao_RotY, 0;
SetAO ao_RotZ, 0;
! Joint guidance for joints 2-6
EGMActJoint egmID2 \J2:=egm_minmax_joint1 \J3:=egm_minmax_joint1
     \J4:=egm_minmax_joint1 \J5:=egm_minmax_joint1
     \J6:=egm_minmax_joint1 \LpFilter:=20;
! Run: keep the end position without returning to the start position
EGMRunJoint egmID2, EGM_STOP_HOLD \J2 \J3 \J4 \J5 \J6 \CondTime:=0.1
     \RampInTime:=0.05 \PosCorrGain:=1;

EGMReset egmID1;
EGMReset egmID2;
ENDPROC
ENDMODULE
```

9.3.6.3 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

```
  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 "wg1sim",
  ! configuration "pathCorr", joint type 1 and at point sensor.
  EGMSetupLTAPP ROB_1, egmId1, "pathCorr", "wg1sim", 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

9.3.6.3 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
```

9.3.7 UdpUc code examples

File locations

The following code examples are available in the RobotWare distribution.

File	Description
egm-sensor.cs	Example using protobuf-csharp-port
egm-sensor.cpp	Example using Google protocol buffers C++
egm.proto	The *egm.proto* file defines the data contract between the robot and the sensor.

The files can be obtained from the PC or the IRC5 controller.

- **In the RobotWare installation folder in RobotStudio**: *...\RobotPackages\RobotWare_RPK_<version>\utility\Template\EGM*
- **On the IRC5 Controller**: *<SystemName>\PRODUCTS\<RobotWare_xx.xx.xxxx>\utility\Template\EGM*

Note

Navigate to the RobotWare installation folder from the RobotStudio **Add-Ins** tab, by right-clicking on the installed RobotWare version in the **Add-Ins** browser and selecting **Open Package Folder**.