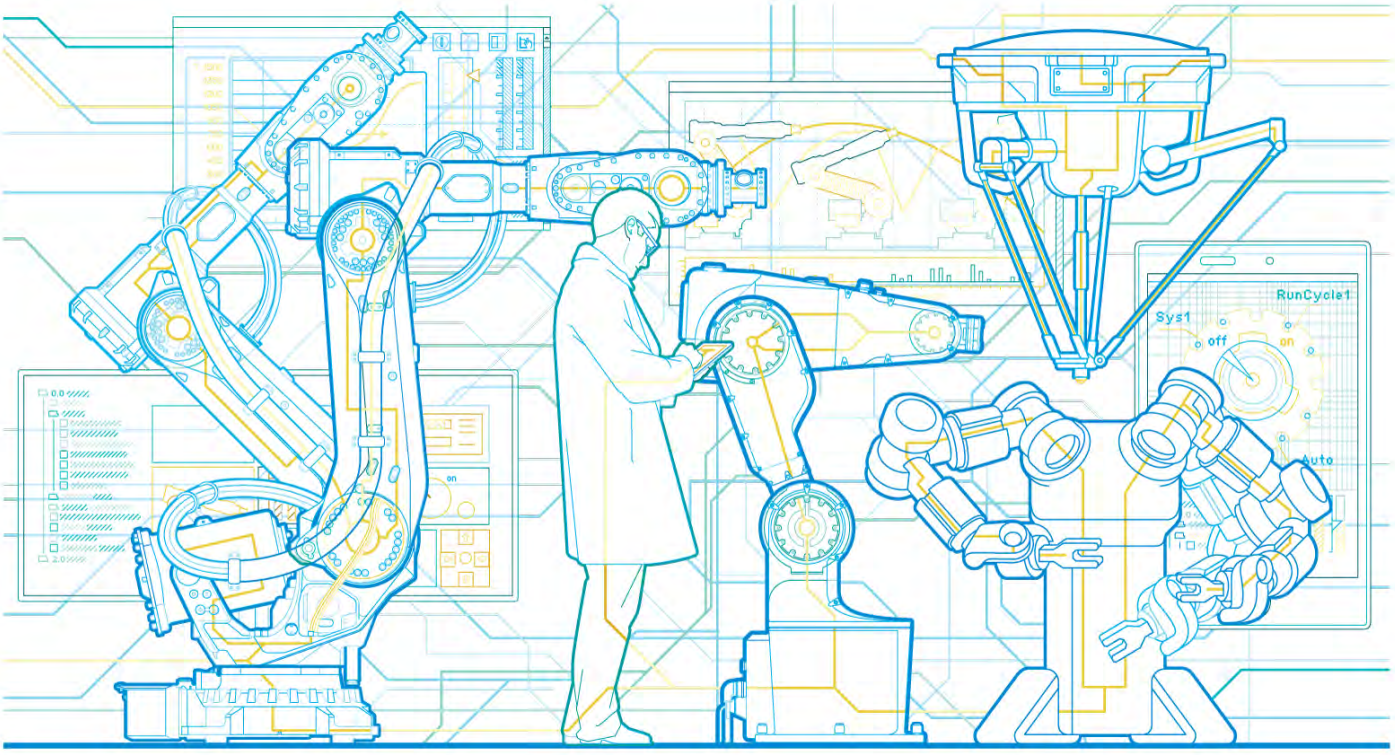


Application manual
Conveyor tracking

Trace back information:
Workspace R17-1 version a2
Checked in 2017-03-13
Skribenta version 5.1.011

Application manual

Conveyor tracking

RobotWare 6.05

Document ID: 3HAC050991-001

Revision: D

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

Table of contents

Overview of this manual

About this manual

This manual describes the options *Conveyor Tracking* and *Indexing Conveyor Control* for IRC5. Conveyor tracking, or line tracking, is when the robot follows a work object mounted on a moving conveyor.

The option *Indexing Conveyor Control* is used when the servo for an indexing conveyor is controlled by the IRC5 controller.

Usages

The manual describes how the options are installed, programmed, and operated.

References

The manual contains references to the following information products:

References	Document ID
Product manual - IRC5 IRC5 of design M2004	*3HAC021313-001*
Product manual - IRC5 IRC5 of design 14	*3HAC047136-001*
Product Manual - IRC5P	*3HNA008861-001*
Operating manual - IRC5 with FlexPendant	*3HAC050941-001*
Operator's Manual - IRC5P	*3HNA008861-001*
Operating manual - RobotStudio	*3HAC032104-001*
Technical reference manual - System parameters	*3HAC050948-001*
Technical reference manual - RAPID Instructions, Functions and Data types	*3HAC050917-001*
Technical reference manual - RAPID overview	*3HAC050947-001*
Application manual - Additional axes and stand alone controller	*3HAC051016-001*
Application manual - Controller software IRC5	*3HAC050798-001*
Application manual - MultiMove	*3HAC050961-001*
Application manual - DeviceNet Master/Slave	*3HAC050992-001*

Revisions

The following revisions of this manual have been released.

Revision	Description
-	Released with RobotWare 6.0.
A	Released with RobotWare 6.01. • Updated information about adding conveyor tracking to an existing robot system, see *Installing conveyor tracking software on page 44*.
B	Released with RobotWare 6.03. Added information about the system parameter *Supervise max_dist Off* in section *Defining the conveyor maximum and minimum distances on page 39*.

Continues on next page

Continued

Revision	Description
C	Released with RobotWare 6.04. • Updated the counter values from 16-bit signals to 32-bit signals, see *System parameters on page 71*. • Minor corrections.
D	Released with RobotWare 6.05. • Minor corrections.

Product documentation, IRC5

Categories for user documentation from ABB Robotics

The user documentation from ABB Robotics is divided into a number of categories. This listing is based on the type of information in the documents, regardless of whether the products are standard or optional.

All documents listed can be ordered from ABB on a DVD. The documents listed are valid for IRC5 robot systems.

Product manuals

Manipulators, controllers, DressPack/SpotPack, and most other hardware is delivered with a **Product manual** that generally contains:

- Safety information.
- Installation and commissioning (descriptions of mechanical installation or electrical connections).
- Maintenance (descriptions of all required preventive maintenance procedures including intervals and expected life time of parts).
- Repair (descriptions of all recommended repair procedures including spare parts).
- Calibration.
- Decommissioning.
- Reference information (safety standards, unit conversions, screw joints, lists of tools).
- Spare parts list with exploded views (or references to separate spare parts lists).
- Circuit diagrams (or references to circuit diagrams).

Technical reference manuals

The technical reference manuals describe reference information for robotics products.

- *Technical reference manual - Lubrication in gearboxes*: Description of types and volumes of lubrication for the manipulator gearboxes.
- *Technical reference manual - RAPID overview*: An overview of the RAPID programming language.
- *Technical reference manual - RAPID Instructions, Functions and Data types*: Description and syntax for all RAPID instructions, functions, and data types.
- *Technical reference manual - RAPID kernel*: A formal description of the RAPID programming language.
- *Technical reference manual - System parameters*: Description of system parameters and configuration workflows.

Continues on next page

Application manuals

Specific applications (for example software or hardware options) are described in **Application manuals**. An application manual can describe one or several applications.

An application manual generally contains information about:

- The purpose of the application (what it does and when it is useful).
- What is included (for example cables, I/O boards, RAPID instructions, system parameters, DVD with PC software).
- How to install included or required hardware.
- How to use the application.
- Examples of how to use the application.

Operating manuals

The operating manuals describe hands-on handling of the products. The manuals are aimed at those having first-hand operational contact with the product, that is production cell operators, programmers, and trouble shooters.

The group of manuals includes (among others):

- *Operating manual - Emergency safety information*
- *Operating manual - General safety information*
- *Operating manual - Getting started, IRC5 and RobotStudio*
- *Operating manual - IRC5 Integrator's guide*
- *Operating manual - IRC5 with FlexPendant*
- *Operating manual - RobotStudio*
- *Operating manual - Trouble shooting IRC5*

Safety

Safety of personnel

When working inside the robot controller it is necessary to be aware of voltage-related risks.

A danger of high voltage is associated with the following parts:

- Devices inside the controller, for example I/O devices, can be supplied with power from an external source.
- The mains supply/mains switch.
- The power unit.
- The power supply unit for the computer system (230 VAC).
- The rectifier unit (400-480 VAC and 700 VDC). Capacitors!
- The drive unit (700 VDC).
- The service outlets (115/230 VAC).
- The power supply unit for tools, or special power supply units for the machining process.
- The external voltage connected to the controller remains live even when the robot is disconnected from the mains.
- Additional connections.

Therefore, it is important that all safety regulations are followed when doing mechanical and electrical installation work.

Safety regulations

Before beginning mechanical and/or electrical installations, ensure you are familiar with the safety regulations described in *Operating manual - General safety information*[1].

[1] This manual contains all safety instructions from the product manuals for the manipulators and the controllers.

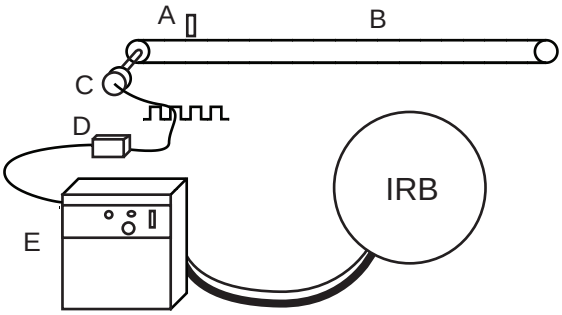
This page is intentionally left blank

1 Introduction to conveyor tracking

1.1 Physical components

Option Conveyor Tracking

A typical installation when using the option *Conveyor Tracking* includes the following components:



xx1200001074

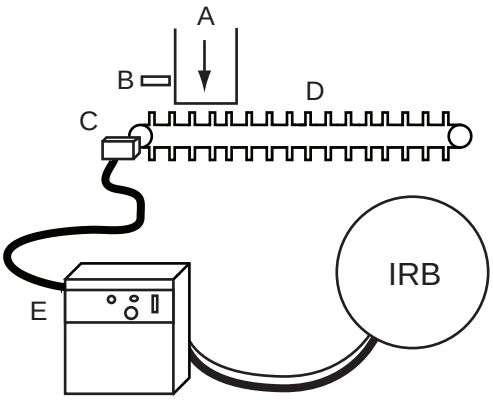
A	Synchronization switch
B	Conveyor
C	Encoder, 24 V
D	Encoder interface, DSQC 377B
E	IRC5 controller
IRB	Robot

The encoder and synchronization switch are connected to the encoder interface, DSQC 377B.

One encoder can be connected to several encoder interfaces. If more than one robot should track the conveyor, then each robot controller must have an encoder interface.

Option Indexing Conveyor Control

A typical installation when using the option *Indexing Conveyor Control* includes the following components:



xx1200001075

Continues on next page

A	Infeeder
B	Sensor
C	Motor unit
D	Indexed conveyor
E	IRC5 controller
IRB	IRB robot

An extra drive unit and SMB needs to be installed, see *Application manual - Additional axes and stand alone controller*.

1.2 Features

Accuracy

In automatic mode, at 150 mm/s constant conveyor speed, the tool center point (TCP) of the robot will stay within +/- 2 mm of the path as seen with no conveyor motion. This is valid as long as the robot is within its dynamic limits with the added conveyor motion. This figure depends on the calibration of the robot and conveyor and is applicable for linear conveyor tracking only.

Object queue

For the option *Conveyor Tracking*, the encoder interface will maintain a queue of up to 254 objects that have passed the synchronization switch. For the option *Indexing Conveyor Control*, the queue can contain up to 100 objects.

Start window

A program can choose to wait for an object that is within a window past the normal starting point, or wait for an object to pass a specific distance, or immediately take the first object in the object tracking queue. Objects that go beyond the start window are automatically skipped.

RAPID access to queue and conveyor data

A RAPID program has access to the number of objects in the object queue, and the current position and speed of the conveyor. A RAPID program can also remove the first object in the tracking queue or all objects in the queue.

Maximum distance

A maximum tracking distance can be specified to stop the robot from tracking outside of the working or safety area.

Track follows conveyor

If the robot is mounted on a linear track, then the system can be configured so that the track will follow the conveyor and maintain the relative position to the conveyor. The TCP speed relative the work object on the conveyor will still be the programmed speed.

Enter and exit conveyor tracking in corner zones

It is possible to enter and exit conveyor tracking via corner zones as well as via fine points. Use corner zones to achieve a minimum cycle time.

Exit and re-enter conveyor tracking to same object

It is possible to exit and re-enter to the same object on the conveyor unlimited times until the object moves outside the working area, reaches the maximum distance, or is explicitly dropped by the RAPID program.

Multiple conveyors

Up to six conveyors are supported with the option *Conveyor Tracking*. Each encoder must be connected to an encoder interface.

Continues on next page

Up to two indexed conveyors can be used with the option *Indexing Conveyor Control*.

Coordinated finepoint

A finepoint can be programmed while moving relative to the conveyor. This conveyor coordinated finepoint will ensure that the robot stops moving relative to the conveyor and will follow the conveyor while the RAPID program continues execution. The robot will hold the position within +/- 0.7 mm depending on calibration of the robot and conveyor.

Calibration of linear conveyors

A calibration method is provided for easy calibration of the position and direction of the conveyor motion in the robot work space. The linear conveyor may take any position and orientation.

1.3 Limitations

Small orientation error with SingArea\Wrist

There can be a small orientation error of the TCP while following the conveyor and make long motions with `SingArea\Wrist`. This error can be eliminated by using several short movements with `SingArea\Wrist`.

Robot mounted on track

If the robot is mounted on a track and the track is to be used to follow the conveyor motions, then the track and conveyor must be parallel.

The motion on the track and the motion on the conveyor must have the same direction of positive motion. See *Configuring a track motion to follow a conveyor on page 42*.

Calibrating circular conveyors

There are no built-in methods for calibrating circular conveyors. This can be evaded by calculating a quaternion orientation manually or with other tools during base frame calibration.

Additional axes

For the option *Conveyor Tracking*, each conveyor is considered an additional axis. Thus the system limitation of 6 active additional axes must be reduced by the number of active and installed conveyors.

The first installed conveyor will use measurement node 6 and the second conveyor will use measurement node 5. These measurement nodes are not available for additional axes and no resolvers should be connected to these nodes on any additional axes measurement boards.

For the option *Indexing Conveyor Control*, each conveyor is considered as two additional axes.

Object queue lost on restart or power failure

The object queue is kept on the encoder interface. If the system is restarted or if the power supply to either the IRC5 controller or the encoder interface fails, then the object queue will be lost.

Minimum and maximum speed

The minimum conveyor speed to maintain smooth and accurate motions depends on the encoder selection. It can vary from 4 mm/s to 8 mm/s. See *Minimum and maximum counts per second on page 26*.

There is no explicit maximum speed for the conveyor. Accuracy will degrade at speeds above the specification and with high speed robot motions or with very high conveyor speeds (> 500 mm/s) and the robot will no longer be able to follow the conveyor.

Continues on next page

WaitWObj after DropWObj

If a `WaitWObj` instruction is used immediately after a `DropWObj` instruction, it may be necessary to add a `WaitTime 0.1;` after the `DropWObj` instruction.

1.4 Principles of conveyor tracking

Description

In conveyor tracking, the robot Tool Center Point (TCP) will automatically follow a work object that is defined on the moving conveyor. While tracking the conveyor the IRC5 will maintain the programmed TCP speed relative to the work object even if the conveyor runs at different speeds.

The options *Conveyor Tracking* and *Indexing Conveyor Control* are built on the coordinated work object, as with coordinated motion with additional axes. See *Application manual - Additional axes and stand alone controller*.

Conveyor as a mechanical unit

In the option *Conveyor Tracking*, the conveyor is treated as a mechanical unit. It has all features of a mechanical unit except that it is not controlled by the IRC5 controller. In the option *Indexing Conveyor Control*, the conveyor is treated as a mechanical unit and controlled by the IRC5 controller.

As a mechanical unit, the conveyor can be activated and deactivated. The position of the conveyor is shown on the FlexPendant **Jogging** window, and in the robtarget when a position is modified (ModPos).

Conveyor coordinated work object

The robot movements are coordinated to the movements of a user frame connected to the conveyor mechanical unit. For example a user frame is placed on the conveyor and connected to its movements.

A movable work object is used with the name of the conveyor mechanical unit. The conveyor tracking coordination will be active if the mechanical unit is active and the conveyor coordinated work object is active. When the conveyor coordinated work object is used, in jogging or in a move instruction, the data in the uframe component will be ignored and the location of the user coordinate system will only depend on the movements of the conveyor mechanical unit. However the oframe component will still work giving an object frame related to the user frame and also the displacement frame can be used.

Waiting for a work object on the conveyor

The difference between a conveyor coordinated work object and a work object that is coordinated to another type of mechanical unit is that there is no work object for coordination until an object appears on the conveyor. There must be a work object present on the conveyor before the robot can coordinate the TCP positions to the conveyor.

For the option *Conveyor Tracking*, work objects on the conveyor are detected by the synchronization switch that is connected to the encoder interface. The unit tracks all objects that have past the synchronization switch and are within specified distances in the work area.

For the option *Indexing Conveyor Control*, the work object is created when the defined number of objects (that is, number of indexes) have passed. Before starting coordinated motion with the conveyor, the RAPID program must first check if there

Continues on next page

is a work object available on the conveyor. If an object is available then execution continues and the motions can use the coordinated work object. If there is no object, then the RAPID program waits until a work object is available.

Connecting to a work object

The RAPID instruction `WaitWObj` is used to wait for a work object on the conveyor before starting conveyor coordinated motion. When the `WaitWObj` instruction is successful then the conveyor work object is said to be connected to the RAPID program. See *WaitWObj - Wait for work object on conveyor on page 67*.

Once a RAPID program has connected to a work object on the conveyor then robot motion instructions and jogging commands can use this work object just as any other work object. When using the conveyor connected coordinated work object then all motions are relative to the work object on the conveyor.

The robot can switch between the conveyor coordinated work object and another coordinated work object in the system, but only one conveyor work object can be connected.

Disconnecting from a work object

The work object will remain connected until one of the following occurs:

- A `DropWObj` instruction is issued
- The maximum distance defined for the conveyor is reached
- Controller is restarted
- Power failure

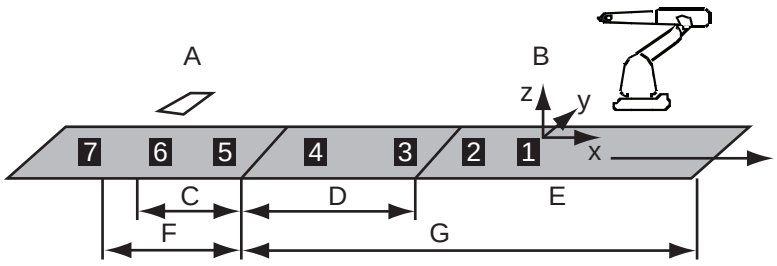
The connection to the work object will not be lost when deactivating the conveyor mechanical unit, and will return on re-activation.

The instruction `DropWObj` ends the connection before the maximum distance is reached.

After a `DropWObj` instruction is issued it is possible to immediately wait for and connect to the next work object in the conveyor object queue.

Start window and queue tracking distance

The object queue is based on a set of distances relative to the conveyor and synchronization switch. See the following figure.



xx1200001076

A	Synchronization switch
B	Work object user frame
C	Minimum distance (*minimum distance*)

Continues on next page

D	Start window width (*StartWinWidth*)
E	Working area
F	Queue tracking distance (*QueueTrckDist*)
G	Maximum distance (*maximum distance*)
1-7	Objects on conveyor

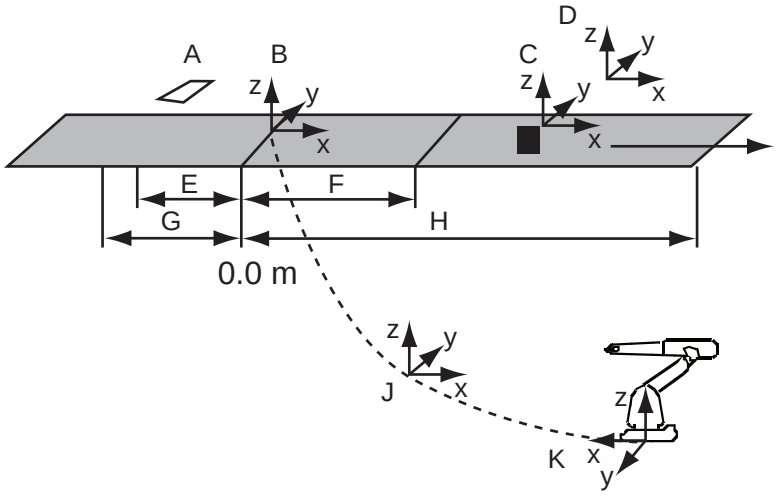
The conditions and states of objects 1 to 7 can be described as in the following table.

For descriptions of the system parameters *StartWinWidth* and *QueueTrckDist* see *Type Fieldbus Command on page 60*. For descriptions of the system parameters minimum distance and and maximum distance see *Type Conveyor systems on page 61*.

Objects	Description
1	This object is connected as indicated by the coordinate frame attached to the position of the object on the conveyor.
2	Object 2 is outside the start window and is no longer tracked. This object will be skipped and cannot be connected by a `WaitWObj` instruction. It is skipped because the conveyor speed is such that coordination with the object could not be completed before the object moves outside the maximum distance or work area of the robot.
3 and 4	These objects are within the start window and are tracked. If object 1 is and 4 dropped via a `DropWObj` instruction then object 3 is the next object to be connected when a `WaitWObj` instruction is issued. Because objects 3 and 4 were in the start window, the `WaitWObj` instruction will not wait but return immediately with object 3.
5 and 6	These objects lie within the queue tracking distance and are tracked. If objects and 6, 3, and 4 were not present, then a `WaitWObj` instruction would stop program execution until object 5 entered the start window.
7	This object has not yet passed the synchronization switch and has not yet been registered.

Continues on next page

Coordinate systems



xx1200001077

A	Synchronization switch
B	Base frame for conveyor
C	User frame
D	Object frame
E	Minimum distance
F	Start window width
G	Queue tracking distance
H	Maximum distance
J	World frame
K	Base frame for robot

The preceding figure shows the principle coordinate frames used in conveyor tracking. See the following table.

Coordinate system	Relative to
Base frame of robot	World frame
World frame	N.a.
Base frame of conveyor	World frame
User frame, coordinated to conveyor	World frame via base frame of conveyor
Object frame (not shown)	User frame

Key frames and positions

The two key frames in conveyor tracking are the base frame for the conveyor and the user frame in the conveyor coordinated work object. The position of user frame in the conveyor coordinated work object is determined from the position of the conveyor base frame and the linear position of the conveyor in meters.

The encoder interface provides the position of the conveyor relative the synchronization switch and the *Queue Tracking Distance*. When the encoder interface sends a value of 0.0 meters to the IRC5 controller, then the user frame

Continues on next page

for the conveyor coordinated work object coincides with the base frame of the conveyor. As the conveyor moves, then the user frame in the conveyor coordinated work object moves along the x-axis of the conveyor base frame.

See *Operating manual - IRC5 with FlexPendant*, and *Technical reference manual - RAPID overview*.

This page is intentionally left blank

2 Installation

2.1 Hardware connections

Introduction to hardware installation

For the option *Conveyor Tracking*, the conveyor is connected to the IRC5 controller using an encoder and an encoder interface (DSQC 377B). See *Installing the encoder for conveyor tracking on page 26*.

For the option *Indexing Conveyor Control*, the conveyor is controlled by a motor unit (ABB or other). See *Installing the additional axis for servo control on page 111*.

Related information

If the conveyor is used with IRC5P (for paint applications), see *Product Manual - IRC5P*.

2.2 Installing the encoder for conveyor tracking

Selecting encoder type

The encoder provides a series of pulses indicating the motion of the conveyor. This is used to synchronize the motions of the robot to the motion of the conveyor.

The encoder has two pulse channels, A and B, that differ in phase by 90°. Each channel will send a fixed number of pulses per revolution depending on the construction of the encoder.

The number of pulses per revolution for the encoder must be selected in relation to the gearing between the conveyor and the encoder. The pulse ratio from the encoder should be in the range of 1250 -2000 pulses per meter of conveyor motion. The pulses from channels A and B are used in quadrature to multiply the pulse ratio by 4 to get counts. This means that the control software will measure 5000 to 10000 counts per meter for an encoder with the pulse ratio given above. Reducing the number of measured counts below 5000 will reduce the accuracy of the robot tracking. Increasing the number of measured counts above 10000 will have no significant effect as inaccuracies in robot and cell calibration will be the dominating factors for accuracy.

The encoder must be of 2 phase type for quadrature pulses, to register reverse conveyor motion, and to avoid false counts due to vibration etc. when the conveyor is not moving.

Output signal	Open collector PNP output
Voltage	10 - 30 V (normally supplied by 24 VDC from DSQC 377B)
Current	50 - 100 mA
Phase	2 phase with 90 degree phase shift
Duty cycle	50%

An example encoder is the *Lenord & Bauer GEL 262*.

Technical specification for encoder interface

	DSQC 377B
Power supply	24 VDC, typically 50 mA, maximum 200 mA
Operating temperature	+5°C - +60°C
Encoder frequency	0 - 20 kHz
Encoder input current	6.8 mA at 24 VDC i
Digital input current	6.8 mA at 24 VDC i

i 18 VDC < '1' < 35 VDC
-35 VDC< '0' < 2 VDC

Minimum and maximum counts per second

Minimum speed	The lower limit on the number of counts per second before the encoder signals zero speed is 40 counts per second. If the speed of the conveyor is lower, zero speed will be indicated. At 10,000 counts per meter, the minimum conveyor speed is 4 mm/s.

Continues on next page

Maximum speed	The upper limit on the number of counts per second before the encoder can no longer keep track of the counts along the conveyor is 20,000 counts per second. At 10,000 counts per meter, the maximum conveyor speed is 2,000 mm/s.

2.3 Encoder location

Description

The encoder is normally installed on the conveyor drive unit. The encoder can be connected to an output shaft on the drive unit, directly or via a gear belt arrangement.

Prerequisites

If the encoder is connected directly to a drive unit shaft, a flexible coupling must be used to prevent applying mechanical forces to the encoder rotor. Do not use a coupling using a plastic or rubber hose. If the drive unit includes a clutch arrangement, the encoder must be connected on the conveyor side of the clutch.

If the conveyor drive unit is located a long distance away from the robot then the conveyor itself can be a source of inaccuracy as the conveyor will stretch or flex over the distance from the drive unit to the robot cell. In such a case it may be better to mount the encoder closer to the robot cell with a different coupling arrangement.

2.4 Connecting the encoder to the encoder interface

Encoder connections

One encoder can be connected to one or more encoder interfaces. The encoder interfaces can be connected to the same robot controller as in the case for two conveyors, or to different robot controllers, for example when two different robots follow the same conveyor.

Connecting to several controllers

An encoder that is connected to more than one robot controller can get power supply from the controllers or from an external power supply.

If the encoder has power supply from several robot controllers, a diode should be installed on each of the 24 VDC connections to the encoder to prevent parallel wiring of the power supplies.

Cables and capacitors

The encoder should be connected to the robot by a screened cable to reduce noise. If this cable is long, the inductance in the cable will produce spike pulses on the encoder signal that can, over a period of time, damage the opto couplers in the encoder interface.

The spike pulses can be removed by installing a capacitor between the signal wire and ground for each of the two phases. The capacitors should be connected on the terminal board where the encoder is connected and not on the encoder interface.

Typical capacitor values are 100 nF - 1 µF, depending on the length of the cable. The longer the cable, the larger the capacitor.

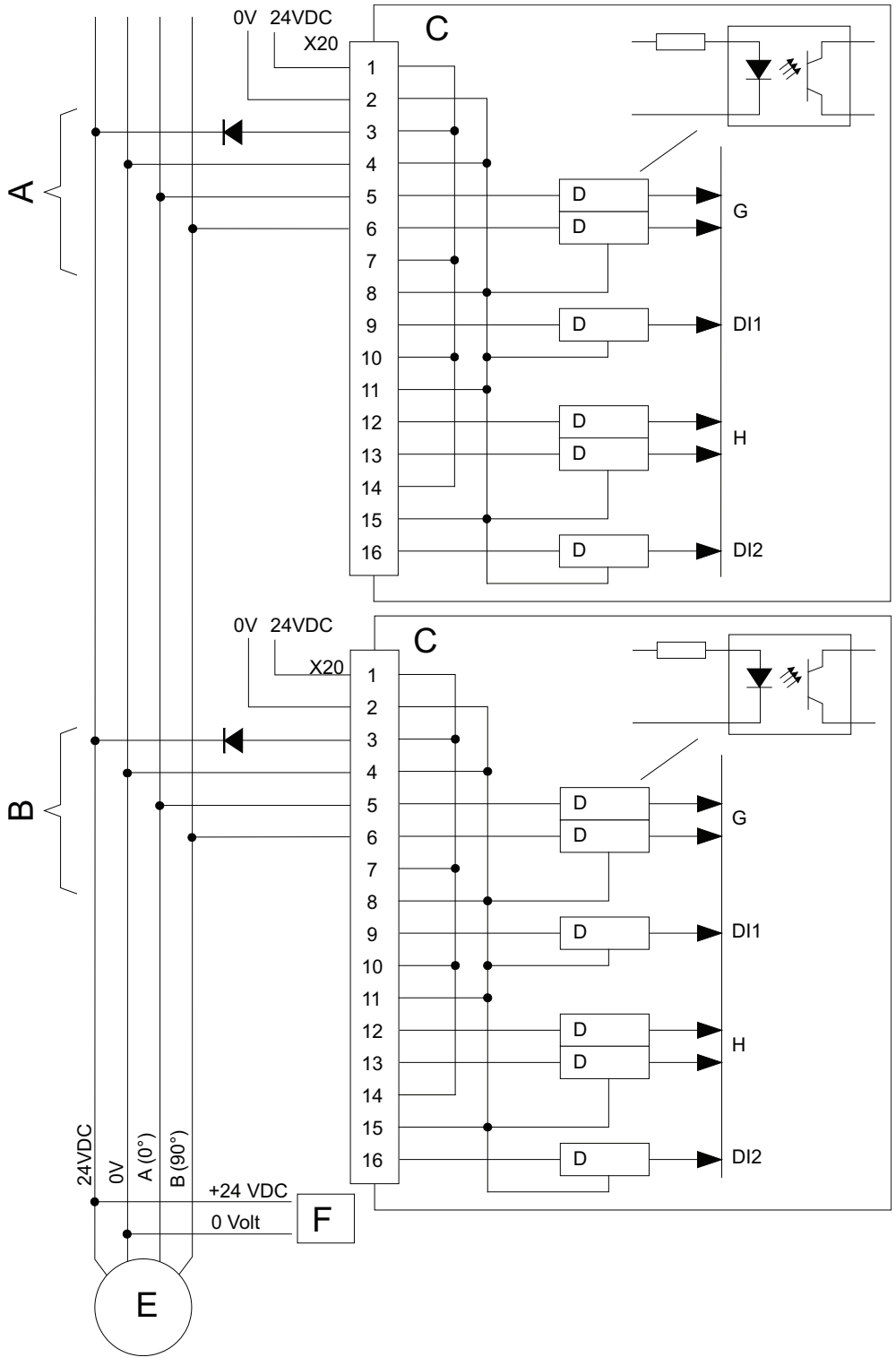
The correct capacitance value can be determined by viewing the encoder signal on an oscilloscope.

Continues on next page

2.4 Connecting the encoder to the encoder interface
Continued

Connection for PNP encoder

The below diagram shows a common encoder for two robots.



xx1200001078

A	Robot 1	
B	Robot 2	

Continues on next page

C	Encoder interface or queue tracking unit (with galvanic insulation)	DSQC 354, DSQC 377, DSQC 377A, DSQC 377B. The units support only PNP and 'push-pull' (E2) encoders.
D	Opto	
E	Encoder	
F	External power	Do not connect internal supply if external is used.
G	Encoder 1	
H	Encoder 2	Used for backup encoder only.
DI1	Digital input 1	
DI2	Digital input 2	Used for backup encoder only.

If digital input is used as common sync switch, use the same connection method as for encoder.

24 V initiators with PNP or push-pull outputs can be used.

Connecting the encoder interface to the robot controller

See *Product manual - IRC5*, for details on connecting the encoder to the encoder interface. See also *Application manual - DeviceNet Master/Slave*.

2.5 Synchronization switch

Description

The synchronization switch indicates that there are objects on the conveyor. Select a switch that provides a reliable and repeatable signal for objects on the conveyor regardless of conveyor speed.

Recommendations

If the conveyor can run backwards, then the switch should be constructed not to give a signal if an object runs backwards past the switch. If the synchronization switch can give multiple signals when an object passes it, then the parameter *SyncSeparation* can be set to so that only one signal is accepted as an object before a given distance is covered on the conveyor. See *Defining conveyor start window and sync separation on page 38*.

3 Configuration and calibration

3.1 About configuration and calibration

Introduction

This section describes how to configure and calibrate the conveyor and encoder (if used) with respect to the robot and the robot world frame.

The encoder descriptions in this chapter are only valid for the option *Conveyor Tracking*. Other descriptions apply also to the option *Indexing Conveyor Control*.

If the conveyor is used with IRC5P (painting), see *Operator's Manual - IRC5P*.

Direction of positive motion from the encoder

If an encoder is used, the encoder direction for positive motion can be checked after the correct installation of the conveyor tracking software.

Run an object past the synchronization switch while viewing the conveyor position on the FlexPendant, from the **Jogging** or the **I/O** window.

If the value is negative, reverse the A and B inputs from the encoder to the encoder interface.

3.2 Calibrating CountsPerMeter and QueueTrckDist

Description

If the exact gear ratio between the encoder and the conveyor is unknown (typically the case) then the system parameter *CountsPerMeter* must be calibrated using either a tape-measure or the robot TCP as a measuring device.

If the robot TCP is used as the measuring device then an accurately defined tool must be used. The encoder interface will keep track of all objects that have passed the synchronization switch but have not yet passed the 0.0 m point. It is not possible to connect to these objects. First configure the encoder interface for 10,000 *CountsPerMeter* (default) and the system parameter *QueueTrckDist* of 0.0 m.

Calibrating CountsPerMeter and QueueTrckDist

Use this procedure to modify *CountsPerMeter* and *QueueTrckDist* on the FlexPendant.

1. On the ABB menu, tap **Control panel** and **Configuration** and then tap **Topics** and **I/O**.
2. Tap the type **Fieldbus Command**.
3. Tap **CountsPerMeter** and change the value to 10000.
4. Tap OK.
5. Tap **QueueTrckDist** and change the value to 0.0.
6. Tap **OK**.
7. Measure and calculate the exact value as described below, *Calculating CountsPerMeter on page 34*.
8. Change the value of **CountsPerMeter** again to the calculated value.
9. Tap **OK**.

See *Type Fieldbus Command on page 60*.

Calculating CountsPerMeter

Move an object past the sync switch and stop the conveyor. Read the conveyor position from the **Jogging** window, `position_1`. Move the conveyor at least 1 meter and read `position_2`. The accuracy will be best if this distance is large as possible within the work space. Use a tape-measure (or differences in robot tool position) to find the exact distance between `position_1` and `position_2`. This is called `measured_meters`.

Use this formula to calculate *CountsPerMeter*:

$$CountsPerMeter = \frac{(position_2 - position_1) * 10000}{(measured_meters))}$$

3.3 Calibrating the conveyor base frame

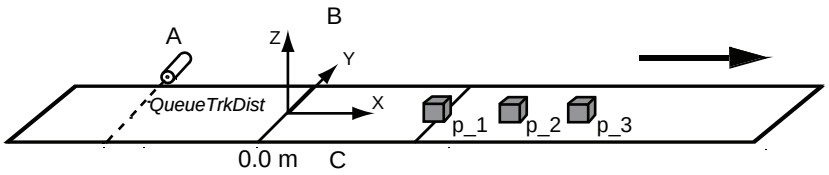
Introduction to the conveyor base frame

The accuracy of the conveyor tracking depends on the accuracy of the conveyor base frame calibration. For linear conveyors, use a method that uses the robot TCP to measure the position and orientation of the conveyor in the work space.

Prerequisites

Before calibrating the base frame of the conveyor the values for the *CountsPerMeter* and *QueueTrkDist* must be correct. See *Calibrating CountsPerMeter and QueueTrckDist on page 34*.

The conveyor base frame calibration method will use the measurement of 4 positions of the same object on the conveyor to determine the conveyor base frame (B), as shown in the following figure.



xx1200001098

A	Synchronization switch
B	Base frame
C	Conveyor distance

Before defining the 4 positions, an object must be defined on the conveyor.

Creating the work object

Use this procedure to create a work object (applicable for the option *Conveyor tracking*). To create a work object for a servo controlled indexing conveyor, see *Creating the work object on page 35*.

1. Step forward through a RAPID program containing the two instructions:
   ```
   ActUnit CNV1;
   WaitWObj wobjcnv1;
   ```
   Define the conveyor coordinated work object, see *Defining a conveyor coordinated work object on page 50*.
2. Run the conveyor until an object passes the sync switch and the 0.0 m point.
   The `WaitWObj` instruction will end execution.
3. Stop the conveyor.
4. Calibrate the base frame, see *Calibrating the base frame on page 36*.

Note

Once an object is on the conveyor and beyond the 0.0 m point, it is possible to use the base frame calibration method to define the conveyor position and orientation in the work space.

Continues on next page

Calibrating the base frame

Use this procedure to calibrate the conveyor base frame (applicable for both *Conveyor Tracking* and *Indexing Conveyor Control*).

1. On the FlexPendant, open the **Calibration** window and select the conveyor.
2. Tap **Base Frame**.
3. Tap **4 Point**.
4. Select the first point, **Point 1**. This point will be the origin for the user frame in the conveyor coordinated work object.
5. Point out Point 1 on the object on the conveyor with the robot TCP.
6. Modify the position by tapping **ModPos**.
7. Move the conveyor in the positive direction and repeat the above for the points 2, 3, and 4.
8. Tap **OK** to calculate the base frame for the selected conveyor mechanical unit.

   A dialog with the calibration result is shown. The calculation log shows the conveyor base frame expressed in the world coordinate system, see *Calculation result for the base frame on page 36*.
9. To save the calculation result in a separate file for later use in a PC:
   a. Tap **File**.
   b. Specify a name and a location where to save the result.
   c. Tap **OK**.
10. If the estimated error is:
    - Acceptable, tap **OK** to confirm the new user frame.
    - Not acceptable, tap **Cancel** and redefine.
11. Restart the controller and verify the results of the calibration. See *Verifying the base frame calibration on page 37*.

Calculation result for the base frame

Field	Description
Unit	The name of the mechanical unit for which the definition of base frame has be done.
List contents	Description
Method	Displays the selected calibration method.
Mean error	The accuracy of the robot positioning against the reference point.
Max error	The maximum error for one positioning.
Cartesian X	The x coordinate for the base frame.
Cartesian Y	The y coordinate for the base frame.
Cartesian Z	The z coordinate for the base frame.
Quaternion 1-4	Orientation components for the base frame.

Continues on next page

Verifying the base frame calibration

After restarting the controller, use this procedure to verify the conveyor base frame calibration.

1. Create a work object, see:
   - *Creating the work object on page 35*, for *Conveyor Tracking*.
   - *Calibrating the base frame on page 118*, for *Indexing Conveyor Control*.
2. Move the robot tool center point back to the previously chosen point 1 on the work object.
3. In the **Jogging** window, read the X, Y, Z position of the tool center point. Make sure to use the correct tool and wobjcnv1.
4. The robot TCP x, y, and z position in the work object coordinates should be 0.0 mm (or very close to that).
5. In the **Jogging** window, select the work object wobjcnv1 and coordinate system WObj and jog the robot in the x, y, and z directions of the conveyor.

   Verify that the x-direction is in the direction of positive motion of the conveyor.

3.4 Defining conveyor start window and sync separation

Start window

The start window is the length along the conveyor in which objects are tracked and are ready for connection. When a `WaitWObj` instruction is issued the system will connect to the first object inside the start window or wait otherwise.

If an object goes beyond the start window then it is no longer tracked and it is not available for connection. Such objects are automatically skipped. The purpose of the start window is to provide a buffer of objects for speed variations of the conveyor. If an object is connected within the start window then it should be certain that the motion coordinated to the object can be completed before the working area limit or maximum distance is reached.

Sync separation

The parameter *Sync Separation* is used to filter out unwanted sync signals from a synchronization switch. This parameter establishes a minimum distance that the conveyor must move after one sync signal before a new sync signal is accepted as a valid object.

Defining start window and sync separation

Use this procedure to define the start window and the sync separation on the FlexPendant.

1. On the ABB menu, tap **Control panel** and **Configuration** and select the topic **I/O**.
2. Tap type **Fieldbus Command**.
3. Select the parameters *StartWinWidth* and *SyncSeparation* and change values.
4. Tap **OK**.

See *Type Fieldbus Command on page 60*.

3.5 Defining the conveyor maximum and minimum distances

Maximum and minimum distances

It is possible to monitor the position of the conveyor and automatically drop any connected objects that move outside the maximum or minimum specified distance.

The purpose is to prevent coordination of motion beyond the work area of the robot for both forward and backward operation of the conveyor.

Defining maximum and minimum distances

Use this procedure to define the distances on the FlexPendant.

1. On the ABB menu, tap **Control panel** and **Configuration** and select the topic **Process**.
2. Tap type **Conveyor systems**.
3. Select **CNV1**.
4. Select the parameters *maximum distance* and *minimum distance* and change the values.
5. Tap **OK**.
6. Tap **Back** and select the type **Can interface**.
7. Select **CAN1**.
8. Select the parameter *Supervise max_dist Off* and set the value to **No**.
9. Tap **OK**.

See *Topic Process on page 61*.

3.6 Defining the robot adjustment speed

Adjusting the speed

When entering conveyor tracking, the robot must adjust its speed to the speed of the conveyor. The speed with which the robot catches up to the conveyor for the first motion is controlled by the parameter adjustment speed.

If the conveyor speed is higher than 200 mm/s then this parameter may have to be increased so the robot can quickly move to the first point on the conveyor.

Defining the robot adjustment speed

Use this procedure to define the robot adjustment speed on the FlexPendant.

1. On the ABB menu, tap **Control panel** and **Configuration** and select the topic **Process**.
2. Tap the type **Conveyor systems** and select **CNV1**.
3. Select the parameter *adjustment speed* and change the values.
4. Tap **OK**.

See *Type Conveyor systems on page 61*.

3.7 Additional adjustments

Description

There are more parameters that can be adjusted in the motion system.

Regulating CPU load and accuracy

The parameter *Path Resolution* specifies the period of the path planner in planning steps along the path (no units). Step calculations require lots of CPU time and if steps cannot be calculated in time to keep the robot on the path then error **50082 Deceleration Limit** may occur. As conveyor tracking increases the general CPU load then the parameter *Path Resolution* can be increased if this error occurs. See *Type Motion Planner on page 64*.

Defining the mechanical unit parameters

The mechanical unit parameters define the name used in RAPID, and the conditions for activation and deactivation.

These parameters can be changed to ensure activation of the conveyor. See *Type Mechanical Unit on page 64*.

Defining additional robot parameters

Some parameters in the type *Robot* (topic *Motion*) can need adjustments. See *Type Robot on page 64*.

3.8 Configuring a track motion to follow a conveyor

Track following a conveyor

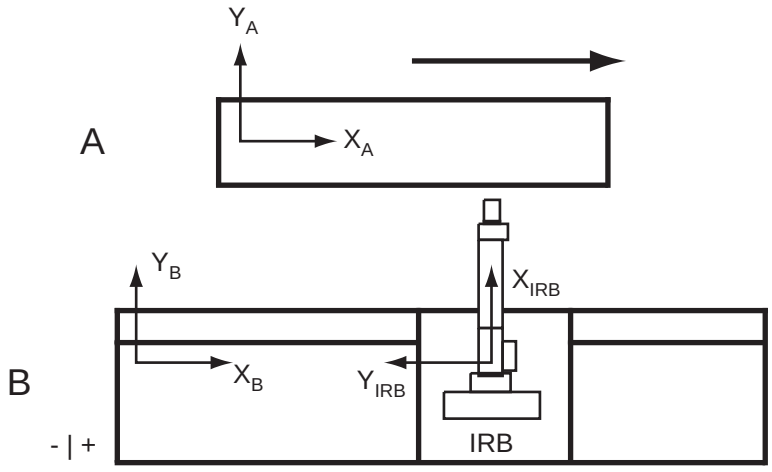
If the robot is mounted on a track, and the track is parallel to the conveyor, then the motion can be configured so that the track follows the conveyor. The robot and the track must be configured for *coordinated track motion*. See *Operating manual - IRC5 with FlexPendant* for information on configuring coordinated motion.

Once the robot and track are configured for coordinated motion, then conveyor tracking will automatically use the track to follow the conveyor. The track will maintain the same position relative to the object as the object moves on the conveyor as it was during programming.

The track and robot base frame must be defined so that positive motion of the track is in the same direction as the conveyor. In some installations this may require a re-definition of the track's direction of positive motion and calibration position.

Example configuration of track and conveyor directions

The following figure shows an example configuration.



xx1200001099

A	Conveyor The arrow shows the direction of movement for the conveyor
B	Track motion The -|+ shows the direction of movement for the track
Conveyor quaternion	1, 0, 0, 0
Robot base quaternion	0.7071, 0, 0, 0.7071
Track base quaternion	1, 0, 0, 0

Continues on next page

Recommendations for programming

Avoid moving the track when programming the conveyor coordinated instructions in the RAPID program. All motions of the track relative to the conveyor are saved and played back during conveyor tracking. Tracks typically have an acceleration ability that is far below that of the robot joints. If the track must move relative to the object then this will require an acceleration that will cause a reduction of the robot's TCP speed along the path in order to maintain coordination.

Tracking the conveyor with a robot instead of a track

If the robot base is not coordinated with the track axis, then the robot will do conveyor tracking without using the track. If the robot base frame is coordinated with the track, and conveyor tracking with the robot (instead of the track) is wanted, then change the parameter *Track Conveyor With Robot* in the type *Robot* (topic *Motion*). See *Type Robot on page 64*.

3.9 Installing conveyor tracking software

Installed software on delivery

The conveyor tracking functions and the RAPID instructions are specified in the license. The conveyor software is loaded on delivery and does not need to be reinstalled.

For *Indexing Conveyor Control*, see *Installing the software on page 111*.

Installing conveyor tracking in an existing system

Use this procedure to install the option *Conveyor tracking* in an existing system. Three configurations will be added:

- I/O for the DSQC 377B
- System parameters for the conveyor in the topic *Process*
- System parameters for the mechanical unit in the topic *Motion*

The conveyor tracking software will automatically install an external I/O configuration for the encoder interface on a virtual bus. Before the DSQC 377B can be used the configuration needs to be changed so that the correct bus and address for the card is specified.

1. Connect the DSQC 377B to the DeviceNet bus. Note the DeviceNet address on the unit.
2. In RobotStudio or FlexPendant, change the following system parameters:
   - Remove the flag *Simulated* from the board configuration. System parameter Simulated in the type *Industrial Network*.
   - Specify the correct address for the unit. System parameter *Address* in the type *Industrial Network*.
3. In RobotStudio or FlexPendant, open the **Configuration editor** and reload the calibration parameters. Load the original file cnv1_moc.cfg.
4. Restart the system.

   There should be no errors and the CNV1 mechanical unit should be available in the **Jogging** window on the FlexPendant.

Reloading saved *Motion* parameters

If the CNV1 mechanical unit does not appear on the FlexPendant then the *Motion* parameters must be reloaded manually in RobotStudio or the FlexPendant, from the IRC5 controller <system name>\PRODUCTS\RobotWare_6.XX.XXXX\options\cnv\cnv1_moc.cfg.

3.10 Installing additional conveyors for conveyor tracking

Additional conveyors

The option *Conveyor Tracking* installs one conveyor in the IRC5 configuration (that is, in the system parameters). If additional conveyors should be tracked with the same IRC5 controller then the parameters for the additional conveyor(s) must be loaded manually.

The second conveyor I/O configuration for the second DSQC 377B will use address 11 by default. The third conveyor board will use address 12, etc. Up to 6 conveyors can be installed on one controller.

For *Indexing Conveyor Control*, see *Installing the additional axis for servo control on page 111*.

Installing additional conveyors

Use this procedure to install additional conveyors for conveyor tracking.

1. Connect the second DSQC 377B to the DeviceNet bus. Note the address on the DeviceNet bus.
2. In RobotStudio, load the files cnv2_eio, cnv2_prc, and cnv2_moc. Repeat for additional conveyors.
3. Restart the system.
4. If needed, correct the address(es) for the Encoder2 to Encoder6 unit(s).

   The CNV1, CNV2 to CNV6 mechanical units should now be available in the **Jogging** window on the FlexPendant.

This page is intentionally left blank

4 Programming

4.1 Programming conveyor tracking

Prerequisites

To create a program that uses conveyor tracking and a conveyor coordinated work object, a work object must be present in the start window. An object must be moved into the start window. If there are several objects already on the conveyor, then it can be necessary to first clear the object queue and then move the conveyor.

Related information

If the conveyor is used with IRC5P (painting), see *Operator's Manual - IRC5P*.

4.2 Working with the object queue

I/O signals and the object queue

There are several I/O signals that allow a user or a RAPID program to monitor and control the object queue. The following table shows the I/O signals that impact the object queue.

I/O signal	Description
c1ObjectsInQ	Group input showing the number of objects in the object queue.
c1Rem1PObj	Remove first pending object from the object queue. Setting this signal will cause the first pending object to be dropped from the object queue. Pending objects are objects that are in the queue but are not connected to a work object.
c1RemAllPObj	Remove all pending objects. Setting this signal will empty all objects from the object queue. If an object is connected, then it is not removed.
c1DropWObj	Setting this signal will drop the tracked object and disconnect that object. The object is removed from the queue. This should not be set from RAPID, use a `DropWobj` instruction instead.

4.3 Activating the conveyor

Activation

As an additional axis and mechanical unit the conveyor must be activated before it can be used for work object coordination. The usual `ActUnit` instruction is used to activate the conveyor and `DeactUnit` can be used to deactivate the conveyor.

By default, the conveyor is installed non-active on start. The conveyor can be configured to always be active at start. See *Type Mechanical Unit on page 64*.

Automatic connection on activation

When a conveyor mechanical unit is activated, it first checks the state of the encoder interface to see if the object was previously connected. If the encoder interface, via the I/O signal *c1Connected*, indicates connection, then the object on the conveyor will automatically be connected on activation. The purpose of this feature is to automatically reconnect in case of a power failure with power backup on the encoder interface.

4.4 Defining a conveyor coordinated work object

Settings for wobjcnv1

Use these settings to define a new conveyor coordinated work object, wobjcnv1.

Data	Data type	Description
`ufprog` (user frame programmed)	bool	Defines if a fixed user coordinate system is used. Set to FALSE for a movable user coordinate system, that is, coordinated to conveyor.
`ufmec` (user frame mechanical unit)	string	The conveyor mechanical unit with which the robot movements are coordinated. Specified with the name that is defined in the system parameters, for example CNV1.

4.5 Waiting for a work object

Waiting for a work object

Motions that should be coordinated with the conveyor cannot be programmed until an object on the conveyor has been connected with a `WaitWObj` instruction.

If the object on the conveyor is already connected from a previous `WaitWObj` or if connection was established during activation, then execution of a second `WaitWObj` instruction will cause an error. This error can be handled in an error handler.

See *WaitWObj - Wait for work object on conveyor on page 67*.

4.6 Programming the conveyor coordinated motion

Programming the conveyor

1. Create a program with the following instructions:
   ```
   ActUnit CNV1;
   ConfL/Off;
   MoveL waitp, v1000, fine, tool;
   WaitWObj wobjcnv1;
   ```
2. Single-step the program past the `WaitWObj` instruction.

   If there is an object already in the start window on the conveyor then the instruction will return, else execution will stop while waiting for an object on the conveyor.
3. Run the conveyor until a work object is created.

   The program exits the `WaitWObj` and is now connected to the object. Stop the conveyor with the object in an accessible position on the workspace.
4. Program the `MoveL`, `MoveC`, `PaintL`, or `PaintC` instructions using the *wobjcnv1* conveyor coordinated work object.
5. End the coordinated motion with a fixed-frame work object (not coordinated to the conveyor). Note that this is the only way to end the use of the work object.
6. Add a `DropWObj wobjcnv1;` instruction.
7. If this is the end of the program, or the conveyor is no longer needed, then add a `DeactUnit CNV1;` instruction.

Example

The following program shows an example for a conveyor tracking program.

```
ConfL\Off;
MoveJ p0, vmax, fine, tool1;
ActUnit CNV1;
WaitWObj wobjcnv1;
MoveL p10, v1000, z1, tool1\Wobj:=wobjcnv1;
MoveL p20, v1000, z1, tool1\Wobj:=wobjcnv1;
MoveL p30, v500, z20, tool1\Wobj:=wobjcnv1;
MoveL p40, v500, fine, tool1;
DropWObj wobjcnv1;
MoveL p0, v500, fine;
DeactUnit CNV1;
```

4.7 Dropping a work object

Introduction

A connected work object can be dropped once conveyor coordinated motion has ended. Make sure that the robot motion is no longer using the conveyor positions when the work object is dropped. If motion still requires the positions then a stop will occur when the object is dropped.

It is not necessary to be connected in order to execute a `DropWObj` instruction. No error will be returned if there was no connected object.

Finepoints

Conveyor coordinated motion does not end in a finepoint. As long as the work object is coordinated to the conveyor, the robot motion will be coupled to the conveyor even in a finepoint. A fixed-frame or non-conveyor work object must be used in a motion instruction before dropping the conveyor work object.

Corner zones

Take care when ending coordination in a corner zone and executing the `DropWObj` instruction as the work object will be dropped before the robot has left the corner zone and the motion still requires the conveyor coordinated work object.

4.8 Entering and exiting conveyor tracking motion in corner zones

Enter and exit coordinated motion

Once a `WaitWObj` instruction has connected to a work object on the conveyor then it is possible to enter and exit coordinated motion with the conveyor via corner zones.

Example

```
MoveL p10, v1000, fine, tool1;
WaitWObj wobjcnv1
! enter coordination in zone
MoveL p20, v1000, z50, tool1;
MoveL p30, v500, z1, tool1\Wobj:=wobjcnv1;
MoveL p40, v500, z1, tool1\Wobj:=wobjcnv1;
MoveL p50, v500, z20, tool1\Wobj:=wobjcnv1;
MoveL p60, v1000, z50, tool1;
! exit coordination in zone
MoveL p70, v500, fine, tool1;
DropWObj wobjcnv1;
MoveL p10, v500, fine, tool1;
```

The move instruction ending coordination must be a fixed work object, for example ufprog is TRUE.

Take care when exiting coordination in a corner zone and executing a `DropWObj` instruction.

The following example shows an incorrect method for ending coordination in corner zones:

```
...
MoveL p50, v500, z20, tool1\Wobj:=wobjcnv1;
MoveL p60, v1000, z50, tool1;
! exit coordination in zone
DropWObj wobjcnv1;
```

This will cause an error, because the work object is dropped while robot is still in corner zone.

4.9 Information on FlexPendant

Available information on the FlexPendant

Information	Available in window
The conveyor position and speed The conveyor position is displayed in meters, and the conveyor speed in m/s. The speed will be 0 m/s until an object has passed the synchronization switch.	**Jogging**
The position in millimeters of the conveyor object This value will be negative if a Queue Tracking Distance is defined. When an object passes the synchronization switch then the position will be automatically updated.	**Jogging**
All signals that are defined for conveyor tracking	**I/O**

4.10 Programming considerations

Performance limits

Conveyor tracking will be lost if joint speed limits are reached, particularly in singularities. Ensure that the path during tracking does not exceed the acceleration, speed, and motion capabilities of the robot.

Motion commands

Only linear and circular motion instructions are allowed for conveyor tracking, that is `MoveL` and `MoveC`.

Do not start tracking with a `MoveC` instruction as the resulting circle diameter will depend on the conveyor position. Always start tracking with a `MoveL`.

Finepoints

Finepoints are allowed during conveyor tracking. The robot will hold the TCP still relative to the conveyor and RAPID execution will continue, see *Finepoint programming on page 57*.

ConfL

The RAPID instruction `ConfL\Off` must be executed before coordination with the conveyor. The purpose is to avoid configuration errors that would otherwise occur as the robot changes configuration during conveyor tracking.

Other RAPID limitations

The instructions `StorePath` and `RestoPath` do not function during conveyor tracking.

A `Search` instruction will stop the robot when hit or if the search fails. Make the search in the same direction as the conveyor moves and after the search stops continue with a move to a safe position. Use an error handler to move to a safe position if the search fails.

`EoffsSet`, `EoffsOn`, and `EoffsOff` have no effect for the conveyor additional axis, but can affect conveyor tracking with coordinated track. They also have effect on the sensor taught position.

Power fail restart is not possible with the synchronization option.

4.11 Finepoint programming

Example with SetDO

While tracking the conveyor it is possible to use a finepoint. The following program example shows how motion may be stopped with respect to the conveyor so that an I/O signal may be set:

```
WaitWObj wobjcnv1
MoveL p1, v500, z20, tool1\Wobj:=wobjcnv1;
MoveL p2, v500, fine, tool1\Wobj:=wobjcnv1;
SetDO release_gripper;
WaitTime 0.1;
MoveL p3, v500, z20, tool1\Wobj:=wobjcnv1;
MoveL p4, v500, fine, tool1;
DropWObj wobjcnv1;
```

In the above example the `SetDO` will be executed after the robot arrives at p2. The robot will then track the conveyor for 0.1 seconds while the `WaitTime` instruction is executed. It will then move to p3 and on to p4 via a corner zone before ending coordination with a fixed work object (`wobj0` in this case).

Example with stoppointdata

Finepoints can also be programed with `stoppointdata`. To make the robot follow the conveyor during 0.1 second, the following program can used.

```
VAR stoppointdata followtime:=[3,FALSE,[0,0,0,0],0,0.1, "",0,0];
WaitWObj wobjcnv1MoveL p1, v500, z20, tool1\Wobj:=wobjcnv1;
MoveL p2,v500,z1\Inpos:=followtime,tool1\WObj:=wobjcnv1;
SetDO release_gripper;
MoveL p3, v500, z20, tool1\Wobj:=wobjcnv1;
MoveL p4, v500, fine, tool1;
DropWObj wobjcnv1;
```

For coordinated movements the `\Inpos` event will not occur for instructions `Break` and `WaitTime`. Instead, use the data type `stoppointdata`.

4.12 Operating modes

Operation in manual reduced speed mode (<250 mm/s)

When the conveyor is not moving, then the forward and backward hard buttons on the FlexPendant can be used to step through the program. New instructions can be added and programmed positions can be modified (ModPos).

To simplify programming, the conveyor may be moved to new positions between instructions. The robot will return to the correct position when forward or backward button is pressed.

The robot will recover as normal if the enabling device is released during motion.

The robot cannot perform coordinated motions to the conveyor while in manual reduced speed mode and the conveyor is moving.

Operation in automatic mode

Once a `WaitWObj` instruction has been executed, then it is no longer possible to step through the program with forward and backward hard buttons while the conveyor is moving.

Action	Description
Start and stop	The robot will stop and no longer track the conveyor if the stop button is pressed, or a `Break` or `StopMove` instruction is executed, between the `WaitWObj` and `DropWObj` instructions. The conveyor coordinated work object will not be lost but if the conveyor is moving then the object will quickly move out of the working area. Restart from the current instruction is not possible and the program must be restarted from the Main routine or with a `WaitWObj` instruction.
Emergency stop	When the emergency stop is pressed the robot will stop immediately. If the program was and restart stopped after a `WaitWObj` then the coordinated work object will not be lost but if the conveyor is moving then the object will quickly move out of the working area. It is not possible to restart from the current instruction and the program must be restarted either from the Main routine or with a `WaitWObj` instruction.

Operation in manual full speed mode (100%)

Operation in manual mode (100%) is similar to operation in automatic mode. The program can be run by pressing and holding the start button, but once a `WaitWObj` instruction has been executed then it is no longer possible to step through the program with the forward and backward buttons while the conveyor is moving.

Action	Description
Stop and restart	When the start button is released, or emergency stop is pressed, the robot will stop immediately. If the program was stopped after a `WaitWObj` then the coordinated work object will not be lost but if the conveyor is moving then the object will quickly move out of the working area. It is not possible to restart from the current instruction and the program must be restarted either from the Main routine or with a `WaitWObj` instruction.

5 System parameters

5.1 Introduction

About system parameters

This chapter describes system parameters used for all conveyor tracking options. Some parameters that are specific for only one option are presented in procedure context if they need to be changed.

All basic system parameters and the system parameter principles are listed in *Technical reference manual - System parameters*.

System parameters are modified using the **Configuration Editor** in RobotStudio or on the FlexPendant.

5.2 Topic I/O

Type Fieldbus Command

The instance is named *Qtrack1*. See *Calibrating CountsPerMeter and QueueTrckDist on page 34*, and *Defining conveyor start window and sync separation on page 38*.

Parameter	Description
CountsPerMeter	Gives the number of quadrature pulses per meter of motion of the conveyor. Should be in the range of 5000-10000 for linear conveyors.
SyncSeparation	Sync signal separation (meters), this distance defines the minimum distance that the conveyor must move after a sync signal before a new sync signal is accepted as a valid object.
QueueTrckDist	The queue tracking distance (meters) defines the placement of the 0.0 meter point relative to the synchronization switch on the conveyor. All objects in this distance are tracked. The position returned for the object will be negative, relative to the 0.0 m point. All objects in this distance are tracked, but connection is not allowed until an object has passed 0.0 meters.
StartWinWidth	This distance defines the start window (meters). The start window defines the area that if a program starts using an object within the window, then all program coordination can end before the maximum distance or work area is reached. All objects within this window are tracked and are eligible for use in a coordinated work object. A `WaitWObj` instruction will connect to the first object in the window.
IIRFFP	Specifies the location of the real part of the poles in the left-half plane (in Hz). This is the break frequency for the speed filters in the encoder interface and regulates how hard the speed is filtered in the encoder interface. For stop and go conveyors this parameter should be set between 10 and 15 Hz to have a good accuracy during stop and start.

5.3 Topic Process

Type Conveyor systems

The instance is named *CNV1*. See *Defining the conveyor maximum and minimum distances on page 39*, and *Defining the robot adjustment speed on page 40*.

Parameter	Description
Adjustment speed	The speed (in mm/s) at which the robot should catch up to the conveyor. It must be higher than the conveyor speed or the robot may never catch up to the conveyor.The adjustment speed must not be higher than 130% of conveyor speed to reduce perturbations on robot speed.
minimum distance	The minimum distance (in millimeters) that a connected object can have before being automatically dropped. If an object is dropped during coordinated motion, then the motion is stopped and an error is produced. Note that if the minimum distance is set to a positive value, then all objects are dropped as soon as they are connected.
maximum distance	The maximum distance (in millimeters) that a connected object can have before being automatically dropped. If an object is dropped during coordinated motion, then the motion is stopped and an error is generated.
Stop ramp	The number of samples used to ramp down the correction when tracking is stopped. The default value of this parameter is 5, and this is a suitable setting for an IRB 360 with payload in the range 0-1 kg. For an IRB 360 with payload 1-8 kg, the value of this parameter should be increased to avoid vibrations and overload of the mechanical structure. A value of at least 10 should be chosen for a robot with 8 kg payload. When tracking with a track axis this parameter should be increased to 30. For a robot switching between conveyors increasing the stop ramp will increase the distance needed to reach accurate tracking on next conveyor. For example, for an IRB 360 10 steps of 12 ms at 5 m/s this means 0.6 meters.
Start ramp	The number of samples used to ramp up the correction when tracking is started. The default value of this parameter is 5, and this is a suitable setting for an IRB 360 with payload in the range 0-1 kg. For an IRB 360 with payload 1-8 kg, this parameter should be increased to avoid vibrations and overload of the mechanical structure. A value of at least 10 should be chosen for a robot with 8 kg payload. When tracking with a high speed conveyor this parameter can be increased. During the ramping the accuracy of tracking is not reached so be careful when increasing this parameter. For an IRB 360 the best choice is to use the default value (5) if the load is not higher than 1 kg. When switching between 2 conveyors with short distance between pick and place the use of time interpolation is the optimal solution. For example, put the robot in a wait position just above the pick conveyor with a very short distance to pick position. In this case a low speed or time interpolation should also be used.

Continues on next page

Parameter	Description
Adjustment accel	The maximum acceleration (in mm/s2) at which robot should catch up to the conveyor. By default no limitation. For big robots and heavy load or limited robot acceleration (like use of `AccSet` or `PathAccLim`) it can be necessary to set *Adjustment accel* according to the robot performances. With robot on track or trolley tracking with the track the performance of the track is automatically used. This parameter must be adjusted when the robot cannot continue its path but remains tracking the same position on the conveyor. For big robots like IRB 6600 *Adjustment accel* should be set around 1000 if conveyor speed is higher than 150 mm/s.
Speed filter length	The number of samples used for average filter of conveyor speed. Maximum value is 50. Default value 1, which equals no filter. Should be used only in case of high level of noise on conveyor speed and speed reduction on robot due to this noise.
Acc dependent filter	Specifies the acceleration dependent filter. Default value is 1 m/s^2. To value get good accuracy during acceleration this value should be set equal to the maximum acceleration of the conveyor. A low value means harder filtering. If there is a problem with noise this parameter should be reduced. If an IRB 360 is used for fast picking this parameter should be set to 0, this will turn off the filtering to improve the response times.
syncfilter ratio	Defines how fast the robot should adjust the speed to the conveyor speed. Default value is 0.8. For IRB 360 this can be reduced to 0.5 to improve the accuracy in fast pick and place applications with low payload (0-1 kg). A too low value might result in jerky movements.

Type Conveyor can sensor

The instance is named *CAN1*. The type *Conveyor can sensor* is used for the option *Conveyor Tracking*. The option *Indexing Conveyor Control* uses the type *Conveyor Internal*, see *Type Conveyor Internal on page 113*.

Parameter	Description
Eio unit name	Name of the I/O unit.
Connected signal	Name of the digital input signal for connection.
Position signal	Name of the analog input signal for conveyor position.
Velocity signal	Name of the analog input signal for conveyor speed.
Null_speed signal	Name of the digital input signal indicating zero speed on the conveyor.
Data ready signal	Name of the digital input signal indicating a poll of the encoder interface.
WaitWObj signal	Name of the digital output signal to indicate that a connection is desired to an object in the queue.
DropWObj signal	Name of the digital output signal to drop a connected object on the encoder interface.
ObjLost signal	Name of the digital input signal to indicate that an object has gone past the start window without being connected.
Supervise max_dist Off	Boolean to remove supervision of maximum and minimum distance.
Rem1PObj signal	Name of digital output signal to drop first pending object in encoder interface queue.

Continues on next page

Parameter	Description
RemAllPObj sig	Name of digital output signal to drop all pending objects in encoder interface queue.

5.4 Topic Motion

Type Mechanical Unit

The instance is named *CNV1*. See *Defining the mechanical unit parameters on page 41*, and *Activating the conveyor on page 49*.

Parameter	Description
Name	The name of the unit, usually *CNV1*, *CNV2* etc. This name is used in the **Jogging** window and from the program, for example when a unit should be activated.
Activate at Startup	Defines if the conveyor should be activated automatically at start.  **Note** This parameter should not be used when combining synchronized and un-synchronized mode, see *Combining synchronized and un-synchronized mode on page 143*.
Deactivation Forbidden	Defines if the conveyor is allowed to be deactivated.

Type Motion Planner

The instance is named *motion_planner*. See *Regulating CPU load and accuracy on page 41*.

Parameter	Description
Path Resolution	The parameter corresponds in some sense to the distance between two points in the path. Increasing path resolution means increasing the distance, which leads to a decrease in the resolution of the path.
Process Update Time	Determines how often the process path information is calculated.
Use spline parameters	Defines how long the robot waits when starting from a finepoint, that is how many positions will be calculated in advance by the motion planner. Default value is *motion_planner_1* for the first robot. Using *3steps mp1* will give a shorter time when starting from finepoint but with the risk that sometimes the robot will stop with warning 50024 (**Corner zone executed as finepoint**) on first move. mp1 stands for motion planner 1, that is, robot 1.

Type Robot

The instance is named *ROB_1*, *ROB_2* etc. See *Additional adjustments on page 41*, and *Configuring a track motion to follow a conveyor on page 42*.

Parameter	Description
Corvec correction level	Defines how often corrections of robot positions are done. Default is 1. Increasing the value gives corrections more often. Should be set to 2 or 3 to get good accuracy during acceleration. For IRB 360 with high payloads (6-8 kg), and for big robots like IRB 6600, it should be set to 1. Increasing it for big robots can lead to jerky movements.
Track Conveyor With Robot	If *Yes*, then the robot will track the conveyor without using the track axis even if robot is coordinated with the track. Default value is *No*.

Continues on next page

Parameter	Description
Max accel for conveyor tracking	This parameter should only be used when conveyor speed is higher than 600 mm/s, when the robot has to pick a part on the conveyor, and when increase of max_external_pos_adjustment and Stop_ramp is not an option. Defines the max acceleration allowed for the robot during conveyor tracking. The aim of this parameter is to reduce adjustment in servo task as big adjustment values can cause speed supervision errors or max_external pos adjustment error (50163). The drawback of too low acceleration is an increase of cycle time. Default value is 1 (limit not used), max value is 40 (m/s^2).
Max External Pos Adjustment	Defines the maximum position adjustment allowed in the servo task. Can be increased for big robots with heavy load and high conveyor speed if error 50163 occurs. First verify that *Adjustment speed* and *Adjustment accel* are correctly defined (see *Type Conveyor systems on page 61*). Default value 0.2, maximum 0.8, minimum 0.1. Defined in meters. If increased, *Start ramp* and *Stop ramp* should also be increased to 20 or 30 (see *Type Conveyor systems on page 61*).

Type Single

The instance is named *CNV1*. See *Calibrating the conveyor base frame on page 35*.

Parameter	Description
Base frame x Base frame y Base frame z	Defines the x, y, and z-direction of the base frame position in Base frame y relation to the world frame (in meters).
Base frame q1-q4	Defines the quaternions of the base frame orientation in relation to the world frame. That is, the orientation of the conveyor base coordinate system.

Type Single Type

The instance is named *CNV1*.

Parameter	Description
Mechanics	Defines what type of mechanics the single type uses.

Type Transmission

The instance is named *CNV1*.

Parameter	Description
Rotating Move	Defines if the conveyor is rotating (*Yes*) or linear (*No*).

This page is intentionally left blank

6 RAPID reference

6.1 Instructions

6.1.1 WaitWObj - Wait for work object on conveyor

Description

`WaitWObj` (*Wait Work Object*) connects to a work object in the start window on the conveyor mechanical unit.

Example

```
WaitWObj wobj_on_cnv1;
```

The program connects to the first object in the object queue that is within the start window on the conveyor. If there is no object in the start window then execution stops and waits for an object.

Arguments

```
WaitWObj WObj [\RelDist][\MaxTime][\TimeFlag]
```

`WObj`

Work Object

Data type: `wobjdata`

The moving work object (coordinate system) to which the robot position in the instruction is related. The mechanical unit conveyor is to be specified by the ufmec in the work object.

`[\RelDist]`

Relative Distance

Data type: `num`

Waits for an object to enter the start window and go beyond the distance specified by the argument. If the work object is already connected, then execution stops until the object passes the distance. If the object has already gone past the relative distance then execution continues.

`[\MaxTime]`

Maximum Time

Data type: `num`

The maximum period of waiting time permitted, expressed in seconds. If this time runs out before the sensor connection or relative distance is reached, the error handler will be called, if there is one, with the error code `ERR_WAIT_MAXTIME`. If there is no error handler, the execution will be stopped.

`[\TimeFlag]`

Timeout Flag

Data type: `bool`

Continues on next page

6.1.1 WaitWObj - Wait for work object on conveyor
Continued

The output parameter that contains the value TRUE if the maximum permitted waiting time runs out before the sensor connection or relative distance is reached. If this parameter is included in the instruction, it is not considered to be an error if the maximum time runs out. This argument is ignored if the `MaxTime` argument is not included in the instruction.

Program execution

If there is no object in the start window then program execution stops. If an object is present, then the work object is connected to the conveyor and execution continues.

If a second `WaitWObj` instruction is issued while connected then an error is returned unless the `\RelDistoptional` argument is used.

More examples

Example 1

```
WaitWObj wobj_on_cnv1\RelDist:=500.0;
```

If not connected, then wait for the object to enter the start window and then wait for the object to pass the 500 mm point on the conveyor. If already connected to the object, then wait for the object to pass 500 mm.

Example 2

```
WaitWObj wobj_on_cnv1\RelDist:=0.0;
```

If not connected, then wait for an object in the start window. If already connected, then continue execution as the object has already gone past 0.0 mm.

Example 3

```
WaitWObj wobj_on_cnv1;
WaitWObj wobj_on_cnv1\RelDist:=0.0;
```

The first `WaitWObj` connects to the object in the start window. The second `WaitWObj` will return immediately if the object is still connected, but will wait for the next object if the previous object had moved past the maximum distance or was dropped.

Example 4

```
WaitWObj wobj_on_cnv1\RelDist:=500.0\MaxTime:=0.1\Timeflag:=flag1;
```

The `WaitWobj` will return immediately if the object has passed 500 mm but otherwise will wait 0.1 seconds for an object. If no object passes 500 mm during this 0.1 seconds the instruction will return with `flag1=TRUE`.

Limitations

50 ms is required to connect to the first object in the start window. Once connected, a second `WaitWObj` instruction with `\RelDist` optional argument will take only normal RAPID instruction execution time.

Error handling

If following errors occur during execution of the `WaitWobj` instruction, the system variable `ERRNO` will be set. These errors can then be handled in the error handler.

ERR_CNV_NOT_ACT	The conveyor is not activated.

Continues on next page

ERR_CNV_CONNECT	The `WaitWobj` instruction is already connected.
ERR_CNV_DROPPED	The object that the instruction `WaitWobj` was waiting for has been dropped by another task (DSQC 377B: an object had passed the start window).
ERR_WAIT_MAXTIME	The object did not come in time and there is no `Timeflag`.

Syntax

```
WaitWObj
  [ WObj ':=']< persistent (PERS) of wobjdata> ';'
  [ '\' RelDist ':=' < expression (IN) of num > ]
  [ '\' MaxTime ':=' <expression (IN) of num>]
  [ '\' TimeFlag ':=' <variable (VAR) of bool>] ';'
```

Related information

For information about	See
`DropWObj`	*DropWObj - Drop work object on conveyor on page 70*
The data types `wobjdata`, `num`, and `bool`	*Technical reference manual - RAPID Instructions, Functions and Data types*

6.1.2 DropWObj - Drop work object on conveyor

Description

`DropWObj` (*Drop Work Object*) is used to disconnect from the current object and the program is ready for the next.

Example

```
MoveL *, v1000, z10, tool, \WObj:=wobj_on_cnv1;
MoveL *, v1000, fine, tool, \WObj:=wobj0;
DropWObj wobj_on_cnv1;
MoveL *, v1000, z10, tool, \WObj:=wobj0;
```

Arguments

```
DropWObj WObj
```

`WObj`

Work Object

Data type: `wobjdata`

The moving work object (coordinate system) to which the robot position in the instruction is related. The mechanical unit conveyor is to be specified by the ufmec in the work object.

Program execution

Dropping the work object means that the encoder interface not longer tracks the object. The object is removed from the object queue and cannot be recovered.

Limitations

If the instruction is issued while the robot is actively using the conveyor coordinated work object then the motion stops.

The instruction can be issued only after a fixed work object has been used in the preceding motion instructions with either a fine point or several (>1) corner zones.

Syntax

```
DropWObj
  [ WObj ':='] < persistent (PERS) of wobjdata> ';'
```

Related information

For information about	See
`WaitWObj`	*WaitWObj - Wait for work object on conveyor on page 67*
The data type `wobjdata`	*Technical reference manual - RAPID Instructions, Functions and Data types*

7 Advanced queue tracking

7.1 Introduction to advanced queue tracking

Introduction

The encoder interface supports queue tracking mode, that is, the job queue is external to the encoder interface, which means it can be handled by the main computer or RAPID code. If the queue tracking is disabled, then the queue is handled in the unit (old 354 mode).

Installation

The conveyor option automatically installs one conveyor in the IRC5 system parameters. Queue tracking is disabled by default in the conveyor I/O configuration, as the signal *c1PosInJob* is set to 0 (zero). To enable queue tracking, set *c1PosInJob* to 1.

System parameters

The following signals are used for advanced queue tracking and must be defined in the system parameters, topic *I/O*, type *Signal*.

I/O signal	Description
c1ObjectsInQ	Group input showing the number of objects in the object queue. These objects have passed the synchronization switch but have not gone outside the start window.
c1Rem1PObj	Remove first pending object from the object queue. Setting this signal will cause the first pending object to be dropped from the object queue. Pending objects are objects that are in the queue but are not connected to a work object.
c1RemAllPObj	Remove all pending objects. Setting this signal will cause the encoder interface to empty all objects from the object queue. If an object is connected, then it is not removed.
c1DropWObj	Setting this signal will cause the encoder interface to drop the tracked object and disconnect that object. The object is removed from the queue. This should not be set from RAPID, use the `DropWobj` instruction instead.
c1NewObjStrobe	DI new position from the encoder node to enter the job queue.
c1CntFromEnc	GI 32-bit counter value from encoder to main controller. **Note** The 32-bit signal is now default, but the old 16-bit signals are still supported: c1CntFromEnc1 (low word) and c1CntFromEnc2 (high word).
c1CntToEncStr	DO strobe for a 32-bit position to the encoder node from the job queue.
c1CntToEnc	GO 32-bit counter value from main controller to encoder. **Note** The 32-bit signal is now default, but the old 16-bit signals are still supported: c1CntToEnc1 (low word) and c1CntToEnc2 (high word).
c1ScaleEncPuls	DI scaled-down encoder pulses.

Continues on next page

I/O signal	Description
c1ForceJob	DO run this job even if the checkpoint should fail.
c1PosInJobQ	DO send the position to MC to be stored in the job queue. (0 = Queue-tracking disabled. Same mode as DSQC 354)
c1PassedStWin	DI notifies the main computer that an object has passed out of the start window (object lost). If the main process is waiting in a `WaitWObj` instruction, the program pointer will be moved to the nearest error handler, so appropriate action can be taken, for example pop the job queue.
c1EncSelected	DI: 0 = Encoder A selected, 1= Encoder B selected
c1EncAFault	DI Encoder A Fault
c1EncBFault	DI Encoder B Fault
c1DirOfTravel	DI direction of travel
c1Simulating	DI simulating mode
c1PowerUp-Status	DI power up status: Counters have been lost.
c1SimMode	DO puts the encoder interface in to simulation mode.
c1softSyncSig	DO soft-sync signal. Simulates Sync-input.
c1softCheckSig	DO soft-checkpoint signal. Simulates checkpoint input.
c1EncSelec	DO: • 0 = Encoder A selected • 1 = Encoder B selected

7.2 Working with the object queue

Signal values

The option *Conveyor Tracking* provides several I/O signals that allow a user or RAPID program to monitor and control the object queue. The table in *System parameters on page 71*, shows the I/O signals that impact the object queue. The counter values have to do with the queue tracking function. Positions detected on the encoder node are sent to the main computer to be stored in the job queue handled by the robot controller. Values are returned to the encoder when object is ready to be tracked.

Handling the object queue in RAPID

To handle the object queue in RAPID the program must store the counter value for each new object on the conveyor and write it to the conveyor board when the user wants to track this object.

The RAPID program needs the following elements:

```
SetDO c1PosInJobQ, 1;

! So the board will update the counters
CONNECT NewObj WITH NewObjOnConvey;
ISignalGI c1CntFromEnc, NewObj;

! To save the counters when a new object is detected
TRAP NewObjOnConvey
  ! Toggle has arrived; Read a new position from input group
  ObjectPosition := GInputDnum(c1CntFromEnc);
  RETURN;
ENDTRAP

TRAP TrackNewObj
  ! To track new obj: Write a new reference to output group
  SetGO c1CntToEnc, ObjectPosition;
  WaitTime 0.02;
  ! Validate the new reference
  PulseDO c1CountToEncStr;
  RETURN;
ENDTRAP
```

Checkpoint function

Checkpoint switch

Check Point Distance is used when running queue tracking. In addition to the normal sync switch you have a checkpoint switch just before the start window.

This switch will check if the object is within the limits set by the checkpoint window width. If not, the object will be discarded from the object queue, unless the signal *c1ForceJob* is active.

The checkpoint signal is connected on input 16.

Continues on next page

Checkpoint parameters

Parameter	Description
Check Point Distance	Distance to checkpoint from 0.0
Check Point Window Width	Tolerance window for the checkpoint

Note

Check Point Distance and *Check Point Window Width* must be set to zero when not used, or else this functionality can unintentionally drop objects from the queue.

Scale enc pulse

The *ScalingFactor* tells after how many counts the signal *c1ScaleEncPuls* should toggle. So the distance between scale encoder pulses = (*ScalingFactor* * 2)/*CountsPerMeter*

The maximum value of *ScalingFactor* is 25000. *ScalingFactor* and *CountsPerMeter* belong to the type *Fieldbus Command* in the topic *I/O*. See *Type Fieldbus Command on page 60*.

Passed start window signal

In some applications it is important to know if an object has gone through the start window without being connected. The encoder interface allows the IRC5 software to detect when an object has passed the start window without being connected and is thus lost. The detection of the lost object is done on the next `WaitWObj` instruction. The next `WaitWObj` instruction, following after an object has moved outside the start window, will return with the error `ERR_CNV_DROPPED`. This error can be handled in the RAPID error handler.

The encoder interface returns a new I/O signal, *c1PassStw*, on physical signal 44 from the encoder interface. This signal will go high when the next connect is attempted and one or more objects have left the start window without being connected.

If needed, the automatic use of the *c1PassStw* signal can be disabled and instead use this signal directly in the RAPID application.

Disabling the object lost feature in `WaitWObj`

The object lost feature is not wanted in all applications as it can limit backward compatibility or can complicate the application.

To disable the feature, remove the conveyor I/O signal *PassStartW*. This will stop the `ERR_CNV_DROPPED` from occurring on the next `WaitWObj` instruction. The *c1PassStw* signal from the DSQC 377B will still go high but the conveyor tracking process will no longer be looking for the signal.

Continues on next page

Simulation mode

The simulated encoder starts when the simulation signal is set. The simulation encoder counts is set to the real encoder counts when the simulation signal is set.

If the simulation signal is reset the encoder value returns to real encoder position.

The simulation speed is defined with the parameter *SimulationVel* in the type *Signal*, in the topic *I/O*.

This page is intentionally left blank

8 Circular conveyor tracking

8.1 Introduction to circular conveyor tracking

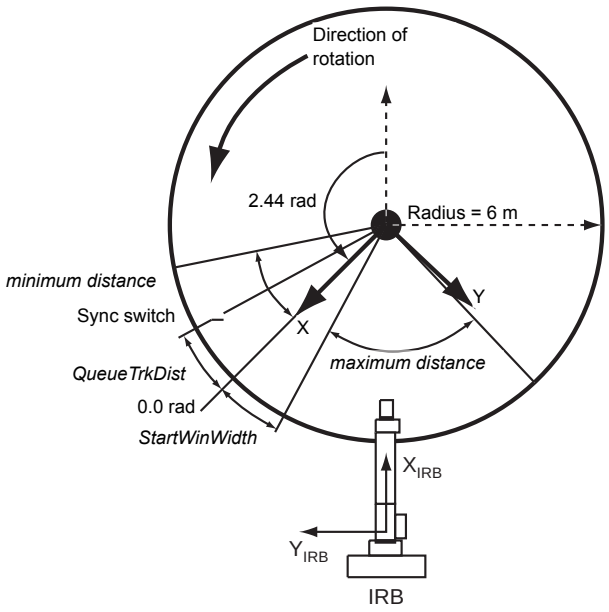
Introduction

Circular conveyors can be tracked with the option *Conveyor Tracking*. The principle for configuring circular conveyor tracking is to define values in radians instead of meters. Then configure as described in *Installation on page 25*, and *Configuration and calibration on page 33*.

This chapter will where there are differences in the configuration compared to linear conveyors.

Example

The figure below shows an example of circular conveyor tracking with example units and distances.



xx1200001101

CountsPerMeter	40000 counts per radian At 6 m radius, one count = 0.15 mm
minimum distance	-100 milliradians At 6 m radius, = -600 mm
Conveyor base frame	Base frame x = 8.0 m Base frame y = 0.0 m Base frame z = 0.0 m
The x-axis is rotated 2.44 rad from the world X (X_{IRB})	Base frame q1 = 0.3420 Base frame q2 = 0.0000 Base frame q3 = 0.0000 Base frame q4 = 0.9397

Continues on next page

SyncSeparation	0.005 rad At 6 m radius = 30 mm
QueueTrkDist	0.017 rad At 6 m radius = 100 mm
maximum distance	420 milliradians At 6 m radius = 2520 mm
StartWinWidth	0.017 rad At 6 m radius = 100 mm

8.2 Encoder type selection and location

Prerequisites

The goal in selecting an encoder for circular conveyor tracking is to have 0.1 mm to 0.2 mm resolution per count at the maximum radius of conveyor tracking.

Example

In the *Example on page 77*, following at a 6 meter radius in order to have 0.15 mm per count, we must have 40,000 counts per radian at the center of the table. The counts are quadrature encoded (four counts per pulse), thus the encoder must give 10,000 pulses per radian of circular conveyor movement. For a full revolution there are 2p radians per revolution, giving a requirement for 10000 × 2p = 62831.85 pulses per revolution of the circular conveyor.

Selecting gear ratio

If an encoder with 1000 pulses per revolution is selected, then we require a gear ratio of 1 to 62.83185 between the circular conveyor and the encoder shaft.

Note

The maximum value for *CountsPerMeter* in the encoder software is 50000. This should be considered when selecting gearing and encoder.

8.3 Installation and configuration

Software installation

The encoder interface and conveyor tracking software are connected and installed in the same way as for linear conveyors.

Direction of positive motion from encoder

See the description for linear conveyors, *Direction of positive motion from the encoder on page 33*.

Defining CountsPerMeter

The value of the parameter *CountsPerMeter* should be known from the selection of the encoder and the gear ratio between the circular conveyor and the encoder shaft. If the value is not known, then it is possible to measure the value following the same steps as outlined for a linear conveyor with extra equipment for measuring the change in angle of the conveyor between *position_1*, and *position_2*. See *Calibrating CountsPerMeter and QueueTrckDist on page 34*.

Defining the queue tracking distance

Before proceeding with conveyor setup and calibration it is necessary to define the desired queue tracking distance (*QueueTrckDist*). The queue tracking distance establishes the distance between the synchronization switch and the 0.0 rad point on the circular conveyor. The encoder interface will keep track of all objects that have passed the synchronization switch but have not yet passed the 0.0 rad point.

See *Calibrating CountsPerMeter and QueueTrckDist on page 34*.

8.4 Calibrating the conveyor base frame

Calibration options

The accuracy of the circular conveyor tracking depends on the accuracy in specifying the conveyor base frame. There are two methods to calibrate the base frame for a circular conveyor:

1. Enter the orientation and position of the base frame based on drawings of the robot installation and simple TCP measurements.
2. Use the robot TCP as a measuring tool and measure several points along the conveyor with some trigonometric calculations to calculate the conveyor base frame position and quaternion.

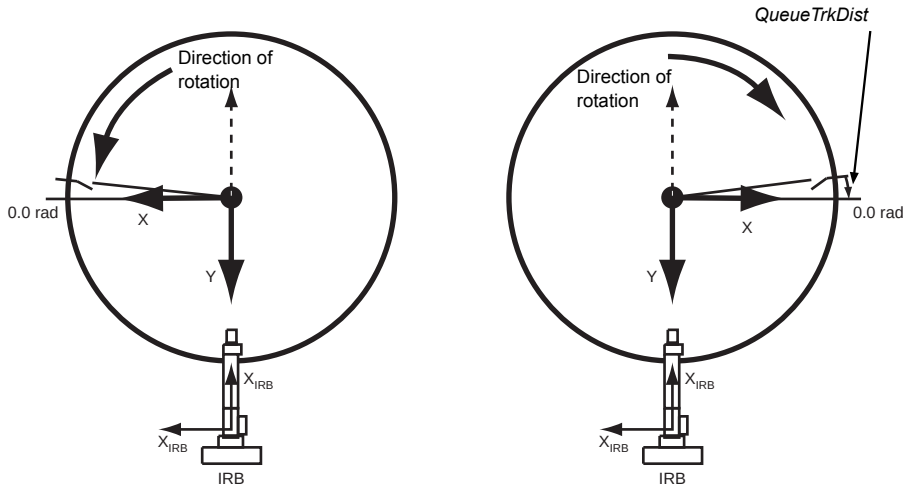
We recommend using the second method, with TCP as measuring tool. See *TCP measurement method on page 83*.

8.5 Manually calibrating the conveyor base frame

Orientation

The definition of the quaternion for conveyor orientation will also define the location of the 0.0 radian point on the circular conveyor. The direction of the x-axis will define the 0.0 radian point while the direction of the z-axis will define the direction of positive rotation using the right-hand-rule.

The graphic below shows two installations, one with clockwise rotation and the other with counterclockwise rotation and the corresponding quaternions. In cases where the 0.0 rad point is not an even multiple of 90° from the world frame, calculation of the conveyor orientation quaternion must be done using manual calculations of the quaternion. The TCP can be used to help make measurements, see *TCP measurement method on page 83*.



xx1200001102

Quaternion 0.7071, 0, 0, 0.7071	Quaternion 0, 0.7071, -0.7071, 0

Base frame position and start window start calibration

The conveyor base frame x, y, and z position must be specified relative to the world frame. This position must be calculated from the installation drawings or by using the robot as a measuring tool. Using the robot, one point can be marked on the edge of the circular conveyor and the TCP position is recorded for several points and the center point of the circle can be found. This is described in detail in the following section.

8.6 TCP measurement method

Recommendation

This section describes how to use TCP measurements and RAPID programs to calculate the conveyor base frame position and quaternion for a circular conveyor.

This method uses three measured points on the circular conveyor to calculate the center of rotation. The three points should be spaced as far apart as possible around the periphery.

Calculating the x and y positions for the base frame

Use this procedure to calculate the x and y positions for the base frame.

1. Use `Wobj0` on the FlexPendant. Pick out a reference point on the circular conveyor, jog the TCP to this point and record *p_0*.
2. Run the conveyor to another position. Jog the TCP to the reference point and record *p_1*.
3. Run the conveyor to a third position, jog the TCP to the reference point and record *p_2*.
4. Use the function `CNVUTL_cirCntr` with the points *p_0*, *p_1*, and *p_2*, to calculate the center of the circle, *p_centre*.

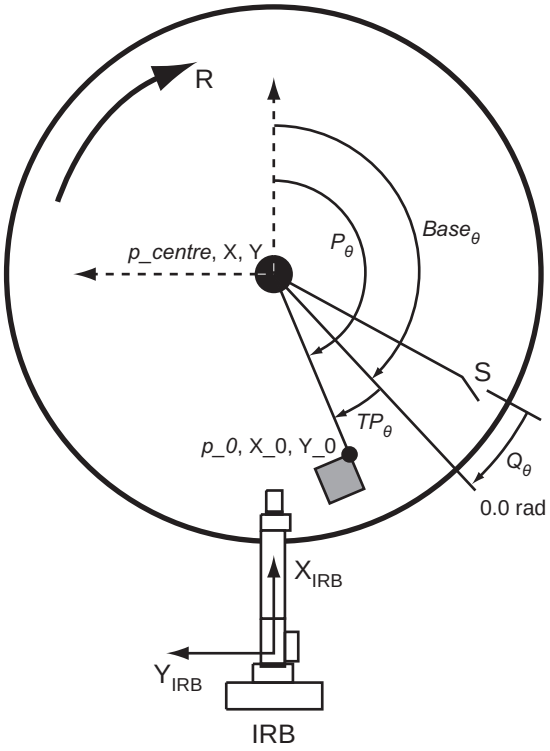
   The system module `cnv_utl.sys` can be found in Robotware.
5. Take the x and y values from *p_centre* and enter them into the base frame values for the conveyor, converting to meters, see *Topic Motion on page 64*. These are shown in *Orientation on page 82*. The z value will be entered later, once the work object zero position has been chosen.

Continues on next page

Defining the base frame orientation and start window start calibration

The base frame quaternion defines where the 0.0 rad point is for the robot motion.

The following figure shows an example of the angles that are used when defining the base frame orientation for the circular conveyor.



xx1200001103

R	Direction of rotation
S	Synchronization switch
Q_θ	Queue tracking distance angle
TP_θ	Angle shown on FlexPendant
P_θ	Angle calculated from p_0 position
$Base_\theta$	Base frame angle to be converted to a quaternion

Calculating the quaternion

Use this procedure to calculate the quaternion for the base frame orientation.

1. Define a temporary conveyor base frame quaternion as 1, 0, 0, 0.
2. Define a conveyor coordinated work object, *wobjcnv1*.
3. Step forward through a RAPID program containing the two instructions:
   ```
   ActUnit CNV1;
   WaitWObj wobjcnv1;
   ```
4. Run the conveyor until an object passes through the sync switch and beyond the queue tracking distance. The `WaitWObj` instruction will end execution. Stop the conveyor.

Continues on next page

5 Using *wobjcnv1*, move the robot TCP to the desired zero position on the work object and record this point, *p_0*. Write down the *X_0*, *Y_0*, and *Z_0* coordinates of the point *p_0* as shown on the FlexPendant (*wobjcnv1* must be selected as work object).

6 Write down the angle shown in the **Jogging** window for the *CNV1* conveyor. This is angle TP_θ, see example measurement points in *Defining the base frame orientation and start window start calibration on page 84*.

7 Calculate P θ from the *X_0* and *Y_0* coordinates of *p_0* and the atan function. X_0 and Y_0 should both be positive when using the atan function. Check the value, it may be necessary to add 90 degrees:

$$P\theta = \text{atan}\left(\frac{|Y\_0|}{|X\_0|}\right)$$

8 Calculate the value of Base.

$$Base_\theta = P_\theta - TP_\theta$$

9 Calculate the quaternion for the base frame taking into account the direction of rotation:

Counter clockwise rotation:

```
q1 = cos(Base θ /2)
q2 = 0.0
q3 = 0.0
q4 = sin(Base θ /2)
```

Clockwise rotation:

```
q1 = 0.0
q2 = cos(Base θ /2)
q3 = -sin(Base θ /2)
q4 = 0.0
```

10 Enter the value for z (in meters) from *p_0*, and the values for the quaternions, *q1*, *q2*, *q3*, and *q4*, into the base frame for the conveyor, see *Topic Motion on page 64*.

8.7 Additional motion settings

Conveyor start window and sync separation

For circular conveyor tracking these distances are defined in radians.

Conveyor maximum and minimum distances

Note

For circular conveyor tracking these distances are defined in milliradians.

Conveyor adjustment speed

The same as for linear conveyors.

Motion System parameters

The same as for linear conveyors.

Mechanical Unit parameters

The same as for linear conveyors.

Transmission and Single Type parameters

The motion configuration of the conveyor must be adjusted to account for a circular motion of the conveyor. There are two parameters that must be adjusted.

Parameter	Type	Description
Rotating Move	Transmission	Defines if the conveyor is rotating (*Yes*) or linear (*No*).
Mechanics	Single Type	Defines the mechanical structure of the conveyor. Select *EXT_ROT*.

9 Accelerating conveyors

9.1 Introduction to accelerating conveyors

Description

This section describes how to optimize the tracking performance of accelerating and decelerating conveyors for the option *Conveyor Tracking*. This might be needed for example if good accuracy is needed during start and stop of the conveyor. To get good accuracy during tracking of accelerating conveyors it is important that all system parameters in the system are defined correctly. This chapter describes the parameters that are important for accelerating and decelerating conveyors. See *System parameters on page 88*.

To further improve the accuracy it is possible to predict the speed change of the conveyor. This is done using a special RAPID function together with an I/O signal that is set just before the acceleration starts. See *Predicting speed changes on page 90*.

For indexing conveyors, see *Indexing conveyors on page 95*.

Prerequisites

The conveyor and encoder must be set up and calibrated correctly, see *Configuration and calibration on page 33*.

9.2 System parameters

Defining the system parameters

Use this procedure to define parameters that are important for an accelerating conveyor.

1. Set the speed filters.

   See *Speed filters on page 88*.
2. Change update rate of robot positions. See *Update rate of robot path on page 89*.
3. Verify the performance of the system.
4. If accuracy during acceleration still is not good enough, try changing the pollrate of the encoder interface(s).

   See *Encoder pollrate on page 89*.

Speed filters

There are two filter parameters that need to be changed. For both the filters there is a trade off between noise reduction and accuracy during acceleration. To get good accuracy during acceleration the filter values should be set according to the recommendations below. If there is too much noise in the system this might lead to disturbances in the robot movement and then the filter parameters should be decreased until this disturbances disappear.

This parameter belongs to the type *Fieldbus Command* in the topic *I/O*. The parameter should be changed for all units.

Parameter	Description
IIRFFP	Specifies the location of the real part of the poles in the left-half plane (in Hz). This is the break frequency for the speed filters in the encoder interface and regulates how hard the speed is filtered. For accelerating conveyors this parameter should be defined to 10-15 Hz to have good accuracy during start and stop.

This parameter belongs to the type *Conveyor Systems* in the topic *Process*.

Parameter	Description
Acc dependent filter	Specifies the setting of the acceleration dependent filter. Default value value is 1 m/s2. To get good accuracy during acceleration set it equal to the maximum acceleration of the conveyor. A low value gives harder filtering. If there is a problem with noise the value should be kept low. If IRB 360 is used for fast picking with low payload, this parameter should be set to 0. This will turn off the filtering to improve the response times.

Continues on next page

Update rate of robot path

It is possible to define how often the robot path should be updated due to new conveyor position.

This parameter belongs to the type *Robot* in the topic *Motion*.

Parameter	Description
Corvec correction level	Defines how often corrections of robot path shall be done. Default is 1. Should be set to 2 or 3 in order to get good accuracy during acceleration. Set to 1 if the prediction of speed changes functionality is used, see *Predicting speed changes on page 90*. For IRB 360 with high payloads (6-8 kg), and for big robots like IRB 6600, it should be set to 1. Increasing it for big robots can lead to jerky movements.

Encoder pollrate

The pollrate of the encoder unit defines how often the position and speed of the conveyor should be read, also known as cyclicity. A lower value (time) will make enable the robot to follow a speed change of the conveyor more accurately. However, a reduction of the pollrate will increase the load in the robot controller and on the DeviceNet bus. How much the pollrate can be reduced depends on the load of the system, for example number of robots, load in the I/O system, etc.

To change the pollrate, three system parameters must be changed.

This parameter belongs to the type *Unit Type* in the topic *I/O*. The instance is named *d377A*.

Parameter	Description
Connection 1 Interval	Defines the pollrate of the card. Default is 20 ms. The minimum value is 4 ms. A low value will give faster response on a conveyor speed change but also increase CPU load. We do not recommend a value lower than 10 ms. If more than one conveyor boards are used there is a risk that reducing pollrate value will create too high load on the DeviceNet bus. Therefore the default value is recommended.

This parameter belongs to the type *Fieldbus Command* in the topic *I/O*. This parameter should be changed for all units.

Parameter	Description
IIRFPeriod	Defines the period of the speed filter on the encoder unit, must be the same as *Connection 1 Interval* for d377A.

This parameter belongs to the type *CAN Interface* in the topic *Process*. The instance is named *CAN1*.

Parameter	Description
Pos Update Time	Defines how often the system reads the speed and position from the I/O system. For best performance it should be the same as *Connection 1 Interval*. It can be higher to reduce the system load.

9.3 Predicting speed changes

Introduction

It is possible to predict the speed change of a conveyor and use this prediction to improve the accuracy during tracking of an accelerating conveyor. The prediction is based on constant acceleration.

The prediction is setup from the RAPID instruction `UseAccProfile` and activated from an I/O signal. It is possible to have two independent profiles defined at the same time connected to two separate I/O signals. One could be used for starting and one for stopping the conveyor.

To access this RAPID instruction load the module Indexing_cnv.sys from the conveyor tracking option directory, that is, \ROBOTWARE_5.XX_XXXX\options\cnv\

Setting up the signals

One or two I/O signals must be defined (one for each profile). These signals activates the prediction and should be set a predefined time before the speed change occurs.

First define the digital input I/O signals, see *Technical reference manual - System parameters*.

The names of the signals are used in the parameters *Sensor start signal* and *Sensor stop signal*. These parameters belong to the type *Conveyor systems* in the topic *Process*.

Parameter	Description
Sensor start signal	Name of the digital input signal to synchronize the prediction and the speed change. The signal must be set a predefined time before the speed change of the conveyor. How far ahead the signal should be set is configured in the RAPID instruction `UseAccProfile`.
Sensor stop signal	Name of the digital input signal to synchronize the prediction and the speed change. The signal must be set a predefined time before the speed change of the conveyor. How far ahead the signal should be set is configured in the RAPID instruction `UseAccProfile`.

The system parameters that affect the accuracy during acceleration are described in *System parameters on page 88*.

9.4 UseAccProfile - Use acceleration profile

Description

`UseAccProfile` is used to predict conveyor movement with constant acceleration or deceleration.

The profile uses either the acceleration for the conveyor or the time that it takes for the conveyor to accelerate or decelerate. If two profiles are defined, they should use an acceleration value instead of a time value.

The prediction of the conveyor acceleration is started by setting the I/O signal configured in *Sensor start signal* or *Sensor stop signal*. For best result this signal must be set at least 150 ms before the conveyor is starting to accelerate or decelerate.

The settings for the acceleration can be changed during program execution.

Example

```
VAR intnum intno1;
VAR triggdata trigg1;
...
CONNECT intno1 With Acc_Dec;
TriggInt trigg1, 0.5\Time, intno1;

Resetset sensor_start_signal_DO;
UseACCProfile CNV1, 0.4, 0, 1\acc, \stop_sig;
SetDO STARTSTOP_CNV, 1;
TriggL p0, v20, trigg1, z10, tool1\Wobj:=wobjconv;
MoveJ p_start, v1000, fine, tool1;

TRAP Acc_Dec PulseDO \HIGH, sensor_start_signal_DO;
WaitTime 0.35;
SetDO STARTSTOP_CNV, 0;
ENDTRAP
```

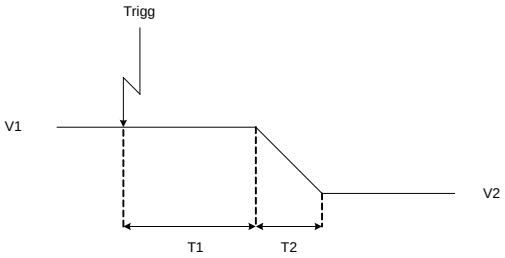
In this example the start and stop of the conveyor is controlled by the I/O signal *STARTSTOP_CNV*. The deceleration profile is setup with a `trigger_time` of 0.4 s, end velocity of 0 m/s and the deceleration is 0.2 m/s^2. This means that the conveyor will decelerate from the current speed down to zero speed with a deceleration of 0.2 m/s^2 and that the sensor stop signal is going to be set 0.4 s before the conveyor is starting to decelerate.

The stop is triggered from a `TriggL` instruction.

As seen in the trap routine the *sensor_start_signal* is set 0.35 s before the stop order to the conveyor. However in the setup of the profile it is said that this signal is coming 0.4 s before the stop. In this case it might be that there is a delay in the

Continues on next page

communication with the conveyor controller of 0.05 s and this is compensated in this way.



xx1200001104

In the preceding figure, the profile from the example is shown. *V1* is the speed before the deceleration and can in this case be for example 0.2 m/s. *V2* is the speed after deceleration, in this case it is 0 m/s. *T1* represents the time between the `Trigg` is coming and the conveyor is starting to decelerate, in this example 0.4 s. *T2* is the length, in time, of the deceleration.

Arguments

```
UseAccProfile MechUnit, Trigger_time, V_end, Acc_time[\acc | time],
     [\start_sig | stop_sig];
```

MechUnit

Mechanical Unit

Data type: `mechunit`

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

Trigger_time

Data type: `num`

The time between the *start_sensor_signal* is set and the time when the conveyor is starting to accelerate or decelerate. The time can be as big as possible but should not be smaller then 0.15 s. The value is given in seconds. In case of to small `Trigger_time` the profile might not be used.

V_end

Data type: `num`

Velocity to be reached at the end of acceleration or deceleration. In case of a stop this should be 0 m/s. The value is given in m/s.

Acc_time

Data type: `num`

Time from the start of the acceleration until the conveyor reaches the final speed (`V_end`). If `[\acc]` is set then this value is considered to be an acceleration value in m/s^2 describing the acceleration of the conveyor.

[\acc | time]

Data type: `switch`

Set to `acc` to use acceleration.

Continues on next page

Set to `time` to use acceleration time.

[\start_sig | stop_sig]

Data type: `switch`

Set to `start_sig` to use the signal configured as *Sensor start signal* to trigger the profile. Set to `stop_sig` to use the signal configured as *Sensor stop signal* to trigger the profile.

Program execution

To get the best possible accuracy during acceleration or deceleration it is important that the `Trigger_time` is the same as the time between setting the *Sensor start signal* and the time when the conveyor starts to accelerate or decelerate. The bigger the difference is between these two times the poorer accuracy will be achieved.

If two profiles are configured in the system at the same time it is very important that the `\acc` option is used. This is to secure a good behavior when for example there is a mix between a start and a stop profile. This could happen when the conveyor for example is stopping and a start order is given so that the stop ramp never is finished.

Limitations

Before `UseAccProfile` is executed, the *Sensor start signal* and *Sensor stop signal* must be reset.

This page is intentionally left blank

10 Indexing conveyors

10.1 Description of indexing conveyor options

Indexing conveyors

There are two ways to track indexing conveyors. If the conveyor is not controlled by IRC5 then recorded profiles must be used, see *Tracking indexing conveyors on page 96*. If the conveyor is controlled by IRC5 the option *Internal Conveyor Control* should be used to get the best possible accuracy. See *Indexing conveyor with servo control (Indexing Conveyor Control) on page 108*.

An indexing conveyor advances in steps instead of running continuously. One step is one index, and one or several indexes creates the work object. The conveyor belt can have pockets or magazines for the products, which makes them perfectly aligned for picking.

10.2 Tracking indexing conveyors

10.2.1 Setting up tracking for an indexing conveyor

Introduction

This section describes how to track an indexing conveyor using the option *Conveyor Tracking*. As the conveyor is not controlled by IRC5, is not possible to know exactly how the conveyor speed is changing. Instead, this method uses predicted speed changes. The repeatability is very important for the accuracy. Therefore, this method cannot be used if the conveyor movements are not repeatable. Then the option *Indexing Conveyor Control* with the conveyor controlled by the robot controller should be used.

To get good accuracy for indexing conveyors it must be possible to predict how the speed of the conveyor is changing. The prediction is based on a recorded profile of the conveyor during acceleration.

A new I/O signal must be defined and connected. RAPID instructions are used to handle prediction of conveyor position during speed changes.

Setting up tracking for an indexing conveyor

Use this procedure to set up tracking for an indexing conveyor.

1. Define a new I/O signal, topic *I/O*. See *Technical reference manual - System parameters*.
2. Connect the I/O signal to the conveyor system, topic *Process*. See *System parameters on page 97*.
3. Define the parameter *IIRFFP*. See *System parameters on page 97*.
4. Record the profile. See *RecordProfile on page 99*.
5. Store the profile. See *StoreProfile on page 101*.
6. Load and/or activate the profile for production. See *LoadProfile on page 102*, and *ActivateProfile on page 103*.

10.2.2 System parameters

Topic I/O

This parameter belongs to the type *Fieldbus Command* in the topic *I/O*.

Parameter	Description
IIRFFP	Specifies the location of the real part of the poles in the left-half plane (in Hz). This is the break frequency for the speed filters in the encoder interface and regulates how hard the speed is filtered in the encoder interface. For indexing conveyors this parameter should be set between 10 and 15 Hz to have a good accuracy during stop and start.

Topic Process

This parameter belongs to the type *Conveyor systems* in the topic *Process*.

Parameter	Description
Sensor_start_signal	Name of the digital input signal to synchronize recorded profile and new index movement. The signal must be set before start of conveyor movement. For example when a cam to move the conveyor the sensor can be placed to be triggered 100 ms before conveyor moves.

10.2.3 RAPID instructions

Introduction

There are two ways to use the indexing conveyor tracking functionality from RAPID. One is to use the instruction `CnvGenInstr`, the other to use the predefined RAPID functions located in the RAPID module named Indexing_cnv.sys. The RAPID instructions in Indexing_cnv.sys encapsulates the functionality in `CnvGenInstr` to make it easier to use. To access these RAPID instructions, load the module Indexing_cnv.sys from the conveyor tracking option directory, that is, \ROBOTWARE_5.XX_XXXX\options\cnv\

10.2.4 RecordProfile

Description

`RecordProfile` resets all profile data and records a new profile of the conveyor movement as soon as the *sensor_start_signal* is set.

To be able to make a recording it is important that a connection to a work object is made before the recording is started. This means that a `WaitWobj` instruction has to be executed before the recording starts.

Example

```
ActUnit CNV1;
WaitWobj wobj_on_cnv1;
RecordProfile CNV1, 1, "index_profile";
WaitTime 0.2;
PulseDO \HIGH sensor_start_signal_DO;
SetDO STARTSTOP_CNV 1;
```

A profile of the conveyor is recorded as soon as the *sensor_start_signal* is set. In this example, the signal *STARTSTOP_CNV* starts the conveyor movement.

Arguments

```
RecordProfile MechUnit, Record_duration, Profile_type
```

MechUnit

Data type: `mechunit`

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

Record_duration

Duration of speed

Data type: `num`

Specifies the duration of record in seconds. Must be between 0.1 and *pos_update_time* * 300.

Profile_type

Type of profile

Data type: `string`

Value	Description
index_profile	Recording is started by *sensor_start_signal*.
start_stop_profile	A start and stop movement can be recorded. *sensor_start_signal* is used to record start movement and *sensor_stop_signal* is used to record *stop_movement*.
stop_start_profile	Same as for `start_and_stop_profile` but the *sensor_stop_signal* is used first.
stop_move_profile	The recording is started with *sensor_stop_signal*.

10.2.5 WaitAndRecProf

Description

`WaitAndRecProf` resets all profile data and records a new profile of the conveyor movement as soon as the *sensor_start_signal* is set.

This instruction does the same as the instruction `RecordProfile` but it also handles the connection to a work object on the conveyor. This instruction is intended for use in PickMaster where the instruction `WaitWobj` is not available.

Example

```
WaitAndRecProf CNV1, 1, "index_profile";
```

A profile of the conveyor is recorded as soon as the *sensor_start_signal* is set.

Program execution

Use this procedure to execute the instruction `WaitAndRecProf`.

Arguments

```
WaitAndRecProf MechUnit, Record_duration, Profile_type
```

MechUnit

Data type: `mechunit`

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

Record_duration

Duration of speed

Data type: `num`

Specifies the duration of record in seconds. Must be between 0.1 and *pos_update_time* * 300.

Profile_type

Type of profile

Data type: `string`

Value	Description
index_profile	Recording is started by *sensor_start_signal*.
start_stop_profile	A start and stop movement can be recorded. *sensor_start_signal* is used to record start movement and *sensor_stop_signal* is used to record *stop_movement*.
stop_start_profile	Same as for `start_and_stop_profile` but the *sensor_stop_signal* is used first.
stop_move_profile	The recording is started with *sensor_stop_signal*.

10.2.6 StoreProfile

Description

`StoreProfile` activates and saves a recorded profile in a file.

Example

```
ActUnit CNV1;
WaitWobj wobj_on_cnv1;
RecordProfile CNV1, 1, "index_profile";
WaitTime 0.2;
PulseDO \HIGH sensor_start_signal_DO;
SetDO STARTSTOP_CNV 1;
WaitTime 2;
SetDO STARTSTOP_CNV 0;
StoreProfile CNV1, 0, "Profile.log";
```

A profile of the conveyor movement is recorded as soon as the *sensor_start_signal* is set and is stored in file profile.log.

Arguments

```
StoreProfile MechUnit, Delay, Filename
```

MechUnit

Data type: `mechunit`

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

Delay

Data type: `num`

The delay in seconds can be used to shift the record in time. It must be between 0.01 and 0.1. If the value is 0 (zero) no delay is added. The delay is not saved in the profile, it is only used for the activation. If the delay should be used together with a saved profile the delay has to be specified again in the instruction `LoadProfile`.

Filename

Data type: `string`

Name of the file where the profile is stored.

10.2.7 LoadProfile

Description

`LoadProfile` loads a recorded profile from a file.

Example

```
LoadProfile CNV1, 0, "profile.log";
WaitTime 0.2;
PulseDO \HIGH sensor_start_signal_DO;
SetDO STARTSTOP_CNV 1;
!
! Work against the conveyor
!
SetDO STARTSTOP_CNV 0;
```

A saved profile of the conveyor movement is loaded and used for prediction of conveyor movement as soon as `sensor_start_signal` is set. Error warning `SYS_ERR_MOC_CNV_REC_FILE_UNKNOWN` if the file is not found.

Arguments

```
LoadProfile MechUnit, Delay, Filename
```

MechUnit

Data type: `mechunit`

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

Delay

Data type: `num`

The delay in seconds can be used to shift the record in time. It must be between 0.01 and 0.1. If the value is 0 (zero) no delay is added. The delay is not saved in the profile, it is only used for the activation. If the delay should be used together with a saved profile the delay has to be specified again in the instruction `LoadProfile`.

Filename

Data type: `string`

Name of the file where the profile is stored.

10.2.8 ActivateProfile

Description

`ActivateProfile` activates a profile that was just recorded to use it without having to save it before.

If the system is restarted, all unsaved records are lost. Therefore, use `LoadProfile` after restarts.

Do not use `ActivateProfile` after `LoadProfile`.

Example

```
ActivateProfile CNV1, 0;
WaitTime 0.2;
PulseDO \HIGH sensor_start_signal_DO;
SetDO STARTSTOP_CNV 1;
!
! Work against the conveyor
!
SetDO STARTSTOP_CNV 0;
```

A profile of the conveyor is activated and used for prediction of conveyor movement as soon as the *sensor_start_signal* is set. Error warning `SYS_ERR_MOC_CNV_REC_NOT_READY` if record not finished.

Arguments

```
ActivateProfile MechUnit, Delay
```

MechUnit

Data type: `mechunit`

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

Delay

Data type: `num`

The delay in seconds can be used to shift the record in time. It must be between 0.01 and 0.1. If the value is 0 (zero) no delay is added.

10.2.9 DeactProfile

Description

`DeactProfile` deactivates a profile.

Example

```
DeactProfile CNV1;
```

A profile of the conveyor movement is deactivated and no longer used for prediction of conveyor movement.

Arguments

```
DeactProfile MechUnit
```

MechUnit

Data type: `mechunit`

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

10.2.10 CnvGenInstr

Description

`CnvGenInstr` sends a command to the conveyor process attached to the conveyor mechanical unit.

Example

```
CnvGenIstr CNV1, CNV_ACTIV_REC, mycnvdata;
```

The controller will activate the record.

Arguments

```
CnvGenInstr MechUnit, cnvcmd, Data
```

MechUnit

Data type: `mechunit`

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

cnvcmd

Command

Data type: `num`

List of possible commands:

- CNV_START_REC
- CNV_STOP_REC
- CNV_ACTIV_REC
- CNV_USE_FREC
- CNV_RESET_ALPROF
- CNV_DEACT_PROF
- CNV_STORE_PROF

Data

Data

Data type: `cnvgendata`

This structure is used to send `num` or `string` as parameters for different commands.

Program execution

All commands must be sent at least 0.2 seconds before start of conveyor movement.

More examples

Example1

```
VAR cnvgendata mycnvdata:=[0,0,0,0,"",""];
CnvGenInstr CNV2,CNV_START_REC,mycnvdata;
mycnvdata.value1:=1;
```

In this example, *data.value1* specifies the duration of recording in seconds. This value must be between 0.1 and *pos_update_time* * 300.

Continues on next page

10.2.10 CnvGenInstr
Continued

Example 2

```
CnvGenInstr CNV2,CNV_STOP_REC,mycnvdata;
```

This example can be used if CNV_START_REC has been sent with duration 0.

Example 3

```
CnvGenInstr CNV2,CNV_ACTIV_REC,mycnvdata;
mycnvdata.value1:=0;
```

In this example, data.value1 specifies a delay added to record in seconds. This value must be between 0.01 and 0.1 seconds.

If value1=0 default value: then signal delay is used. Ready for use of profile on next index movement. Error warning SYS_ERR_MOC_CNV_REC_NOT_READY if record not finished.

Example 4

```
CnvGenInstr CNV2,CNV_USE_FREC,mycnvdata;
```

mycnvdata.string1:="myprofile": string1 must contain the name of the file where to read the recorded profile.

The file must have been created by the command CNV_STORE_PROF. Ready for use of profile on next index movement.

Error warning SYS_ERR_MOC_CNV_REC_FILE_UNKNOWN if record file not found.

Example 5

```
CnvGenInstr CNV2,CNV_RESET_ALPROF,mycnvdata;
```

Reset all profile data, ready for a new START_REC.

Example 6

```
CnvGenInstr CNV2,CNV_DEACT_PROF,mycnvdata;
```

Stop using profile.

Example 7

```
CnvGenInstr CNV2,CNV_STORE_PROF,mycnvdata;
```

mycnvdata.string1:="myprofile";string1 must contain the name of the file to store the profile.

Limitations

As access to files can take a lot of time it is recommended not to use CNV_USE_FREC and CNV_STORE_PROF while robot is moving.

Repeatability error between record and real cycles must be less than 120 ms. A delay between *sensor_start_signal* and conveyor movement must not vary more than 120 ms.

Error handling

No error handling for this instruction. In case of emergency stop of robot or conveyor the command CNV_DEACT_PROF should be used before restarting the robot.

Syntax

```
CnvGenInstr
  [ MechUnit ':=']< var of mechanical unit > ';'
  [ Command ':=']< expression (IN) of num> ';'
```

Continues on next page

```
[ Data ':=']< var of cnvgendata> ';'
```

10.3 Indexing conveyor with servo control (Indexing Conveyor Control)

10.3.1 Introduction to indexing conveyors with servo control

Description of *Indexing Conveyor Control*

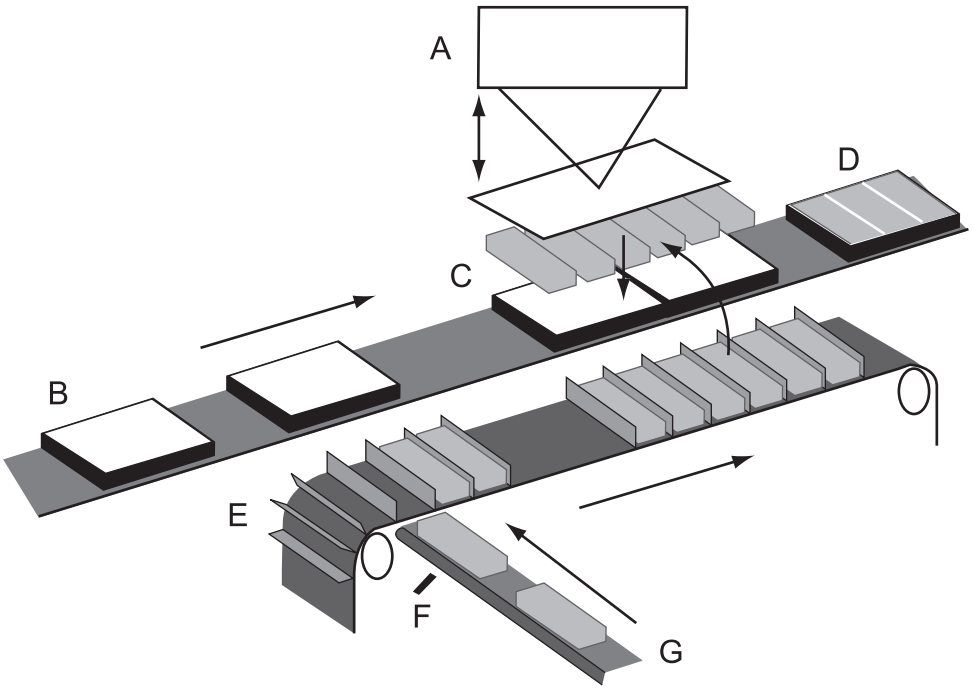
The option *Indexing Conveyor Control* includes RAPID instructions and one RAPID data type. A typical installation includes an infeed conveyor transporting the products in a row at high speed to the indexing conveyor. A photo eye (sensor) mounted on the infeeder detects products and sends a trig signal to the IRC5 controller. The controller starts the indexing movement after a specified time, that is, when the product has entered the pocket on the indexing conveyor. The FlexPicker picks the products from the indexing conveyor, even during indexing movement, and puts them in a box on the output conveyor.

The IRC5 controls the FlexPicker and the indexing conveyor. The output conveyor and the infeeder have their own drive systems.

Conveyor tracking will be used both on the indexing conveyor and the output conveyor, but no encoder and encoder interface is needed on the indexing conveyor since the position and speed are known as a part of the path planning in the IRC5 controller.

Continues on next page

Schematic overview



xx1000001425

A	Robot with gripper
B	Output conveyor with empty cartons, for example from carton erector
C	Robot picks products from indexing conveyor and places in cartons
D	Full cartons, for example to carton closer
E	Indexing conveyor with pockets
F	Photo eye
G	Product infeeder

Terminology

In context of controlling and moving the conveyor this is referred to as *M7* in this document.

In context of tracking the conveyor, it is referred to as *CNV1*. Hence, the conveyor will be configured as two different mechanical units, *M7* and *CNV1*.

Indexing mode is when the system listens (or waits) for a signal from the sensor that a product is available.

In indexing mode it is not possible to jog or use `Move` instructions. To disable indexing mode, execute `IndCnvReset` or move the program pointer to the routine Main (**PP to Main**).

Continues on next page

10.3.1 Introduction to indexing conveyors with servo control
Continued

Limitations

When using the option *Indexing Conveyor Control*, up to two indexing conveyors and two conventional (non-indexing) conveyors with encoder boards can be used per controller. Up to two IRB 360 robots can be configured in a MultiMove system. To be able to have two robots working on the same indexing conveyor, the options *Advanced Quetracking* or *PickMaster 3* must be used.

10.3.2 Setting up a servo controlled indexing conveyor

Installing the additional axis for servo control

For the option *Indexing Conveyor Control*, install the indexing conveyor as an IRC5 additional axis. See *Application manual - Additional axes and stand alone controller*. When installed, the indexing conveyor should be running as an IRC5 additional axis.

Installing the software

The conveyor tracking RAPID instructions, data types, and mechanical unit CNV1 are specified in the key string and do not need to be installed. If the system consists of more than one conveyor, three more files must be installed per conveyor. The files to install are stored on the controller.

The second conveyor to install is called CNV2 and the *Motion* configuration file is named cnv2_moc.cfg. The other two files to install for internally controlled conveyors are cnvint2_prc.cfg and cnvint2_eio.cfg.

10.3.3 System parameters and configuration files

Topic I/O

Type Unit

Verify that the I/O unit *Qtrack1* is defined as *Virtual* in the type *Unit*.

Parameter	Value
Connected To Bus	Virtual1

Type Unit Type

Verify that the I/O unit type used for the photo eye is defined as *Change of State*.

Parameter	Value
Connection 1 type	Change Of State (COS)

Type Signal

In the type *Signal*, configure a digital input signal for the photo eye which will trigger an indexing movement of the conveyor.

Parameter	Value
Name	DI_Eye
Type of Signal	Digital Input

Topic Controller

When running a MultiMove system and the indexing conveyor is running in a separate motion task, the following must be added.

Type Mechanical Unit Group

In the type *Mechanical Unit Group*.

Parameter	Value
Mechanical Unit Group	M7
Mech Unit 1	M7
Use Motionplanner	motion_planner_3

Type Tasks

In the type *Task*.

Parameter	Value
Use Mechanical Unit Group	M7

Type Motion System

In the type *Motion System*.

Parameter	Value
Name	motion_planner_3
dyn_ipol_type	1

Continues on next page

Note

Verify that the *motion_planner_3* has *dyn_ipol_type* defined as 1. To edit this parameter, create a backup and edit moc.cfg. This cannot be changed using RobotStudio.

Topic Process

Type Conveyor systems

In the type *Conveyor systems*, verify that the following parameters are set.

Parameter	Value
Syncfilter Ratio	0.0001
Acc Dependent Filter Value	0

Type Conveyor Internal

The instance is named *INTERNAL1*.

Parameter	Description
Eio unit name	Name of the simulated I/O unit.
Connected signal	Name of the digital input signal for connection.
Position signal	Name of the analog input signal for conveyor position.
Velocity signal	Name of the analog input signal for conveyor speed.
Null_speed signal	Name of the digital input signal indicating zero speed on the conveyor.
DropWObj signal	Name of the digital output signal to drop a connected object on the encoder interface.
ObjLost signal	Name of the digital input signal to indicate that an object has gone past the start window without being connected.
RemAllPObj sig	Name of the digital input signal to remove all Pobj.
Rem1PObj sig	Name of the digital input signal to remove one Pobj.
Pos Update time	Defines how often position and velocity output signals are updated.
Supervise max_dist Off	Boolean to remove supervision of maximum and minimum distance.
New object strobe	Name of digital output signal showing a new object in queue.
Objects in queue	Name of group output signal showing the number of objects in queue.
Count1 from encoder	Name of group output for new object position low word.
Count2 from encoder	Name of group output for new object position high word.
Single to track	Name of the single that is moving the indexing conveyor. (*Indexing Conveyor Control*.)

Continues on next page

10.3.3 System parameters and configuration files
Continued

Topic Motion

Type Arm

In the type *Arm*, enable *Independent Joint* and add joint limits.

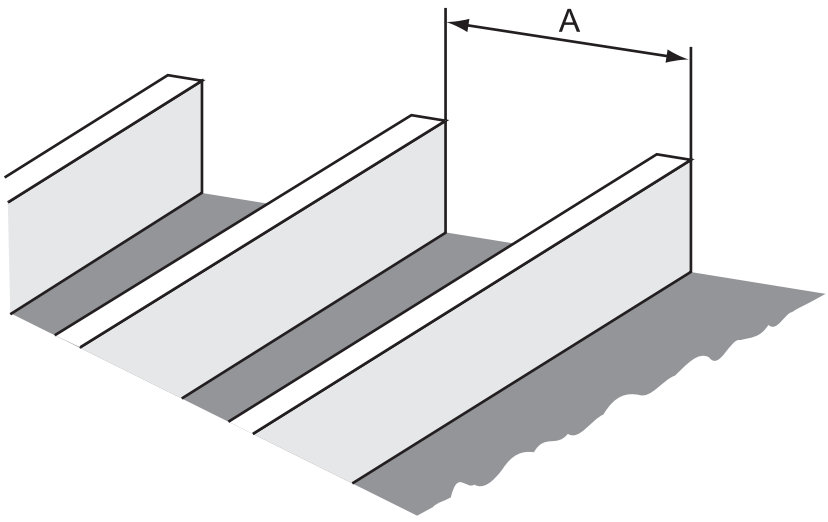
Parameter	Value
Independent Joint	On
Independent Upper Joint Bound	2E+07
Independent Lower Joint Bound	-2E+07

Type Single Type

In the type *Single Type*, the parameter *Pocket size* defines the distance the conveyor will move when triggered by the photo eye (the size of one pocket), see the following graphic. It is very important for conveyor tracking accuracy that the pocket size is given correctly. Use measuring tape and measure at least ten pockets to have a an average for one pocket.

Time before indexing move defines the time from the photo eye is triggered until the conveyor movement starts. The recommended value is 0.3 (300 ms). Depending on the robot payload and deceleration distance, this value can be decreased (might be needed to increase for some applications). If it is important to use a value as small as possible, see *Minimizing trigger time on page 127*.

Parameter	Value
Mechanics	FREE_ROT
Indexing move	Yes
Time before indexing move	0.3
Pocket size	0.05



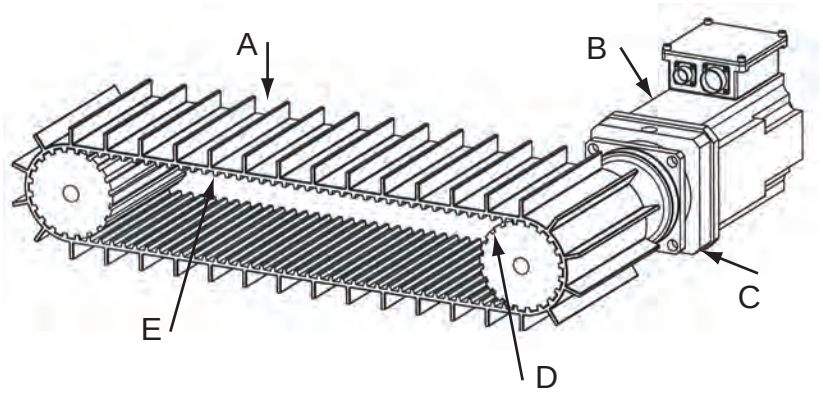
xx1200001105

A	Pocket size

Continues on next page

Type Transmission

The transmission must be represented by two integer parameters, *Transmission Gear High* and *Transmission Gear Low*. This is needed to avoid loosing accuracy after a very big number of indexing movements. These parameters are computed in a way that makes it possible for the controller to move the indexing conveyor exactly one pocket instead of for example 50 mm which could be the measured pocket size.



xx1200001106

A	Pockets
B	Motor
C	Gearbox
D	Gear wheel teeth
E	Belt teeth

See the following example for the calculation of the parameters *Transmission Gear High*, *Transmission Gear Low*, and *Transmission Gear Ratio*. With this setup, the parameter *Rotating Move* must be set.

Calculation example:

D = Number of gear wheel teeth

A = Number of pockets

Transmission Gear Low = D * A = 20 * 36 = 720

G = Gear box ratio

E = Number of belt teeth

Transmission Gear High = G * E * 360 = 10 * 108 * 360 = 388800

Transmission Gear Ratio = *Transmission Gear High* / *Transmission Gear Low* = 388800 / 720 = 540 / 1 = 540

In this example we use the gear box ratio 10, this means 10 motor revolutions correspond to one gear wheel revolution. The *Transmission Gear High* / *Transmission Gear Low* ratio can be given as 388800 / 720 but 540 / 1 is easier to comprehend.

Parameter	Value
Rotating move	YES

Continues on next page

10.3.3 System parameters and configuration files
Continued

Parameter	Value
Transmission Gear Ratio	540 (from example)
Transmission Gear High	540 (from example)
Transmission Gear Low	1 (from example)

Type Acceleration Data

For an indexing conveyor the acceleration data is given in m/s^2 even though the parameter *Rotating Move* is set. The indexing movement will be a symmetric triangular motion profile.

If different values are given for acceleration and deceleration, the smallest value will be used when creating the motion path.

To avoid vibrations and overload of the mechanical structure, the values for acceleration and deceleration should not be set higher than the robot capacity at the given payload.

Parameter	Value
Nominal Acceleration	25
Nominal Deceleration	25

10.3.4 Testing the indexing conveyor setup

Testing

Create a RAPID program using the code from the example below. Speed and acceleration are given in mm/s respectively mm/s^2. To avoid vibrations and overload of the mechanical structure, the values for acceleration and deceleration should not be set higher than the robot capacity at the given payload.

In this example an indexing movement will be performed each time the digital input signal *DI_Eye* is triggered.

RAPID example

```
MODULE MainModule
  VAR indcnvdata indcnvdata1:=[0,0,0,0,0,0,0];
  PROC main()
    indcnvdata1.speed := 2000;
    indcnvdata1.acceleration := 25000;
    indcnvdata1.productsperpick := 6;
    indcnvdata1.productsperindex := 1;

    ActUnit M7;
    IndCnvInit M7, DI_Eye, indcnvdata1;
    IndCnvEnable M7;

    WHILE TRUE DO
      WaitTime 1;
    ENDWHILE
  ENDPROC
ENDMODULE
```

10.3.5 Calibrating the base frame

Creating the work object

To calibrate the base frame, first create the work object. Then calibrate and verify the calibration.

Use this procedure to calibrate the base frame for CNV1.

1. Jog the mechanical unit M7 to the correct position according to the infeeder.
2. Perform a fine calibration of M7 in this position.
3. Activate the mechanical units M7 and CNV1.
   ```
   ActUnit M7;
   ActUnit CNV1;
   ```
4. Initialize the indexing conveyor by executing the following RAPID instruction where `indcnvdata1` is setup as described in *Testing the indexing conveyor setup on page 117*.
   ```
   IndCnvInit M7, DI_Eye, indcnvdata1;
   ```
5. To make sure there are no objects in the object queue execute the RAPID instruction:
   ```
   PulseDO \PLength:=0.1,c1RemAllPObj;
   ```
6. Add a new object to the object queue by executing the RAPID instruction:
   ```
   IndCnvAddObject M7;
   ```
7. To be able to jog M7 during calibration it is necessary to reset the indexing functionality by executing the RAPID instruction:
   ```
   IndCnvReset M7;
   ```

Calibrating the base frame

Start the base frame calibration routine of CNV1 and move the conveyor to the calibration positions by jogging the mechanical unit M7. See *Calibrating the base frame on page 36*.

Continue with verifying the calibration, see *Verifying the base frame calibration on page 37*.

10.3.6 indcnvdata

Description

The data type `indcnvdata` contains information about the indexing movement and the number of pockets that the work object holds.

Example

```
VAR indcnvdata indcnvdata1:=[0,0,0,0,0,0,0];
indcnvdata1.speed:= 2000;
indcnvdata1.acceleration := 25000;
indcnvdata1.productsperpick := 6;
indcnvdata1.productsperindex := 1;
indcnvdata1.accuracytuning := 0;
```

Components

speed

Conveyor speed in mm/s. Normally this parameter is set to a high value to create a motion profile that is triangular.

acceleration

Conveyor acceleration in mm/s^2. This value cannot be higher than what is configured in *Acceleration Data*, see *Type Acceleration Data on page 116*.

To avoid vibrations and overload of the mechanical structure, the values for acceleration and deceleration should not be set higher than the robot capacity at the given payload.

productsperpick

The number of pockets that offsets each work object. If there is one product in each pocket this parameter defines how many products the robot will pick in each robot cycle.

productsperindex

Use this parameter to set how many products that should enter each pocket before executing an indexing movement.

accuracytuning

This parameter can be used to tune the synchronization between robot and indexing conveyor and requires a high speed camera. The tuning value is default 0 and can be adjusted +-10ms.

10.3.7 IndCnvInit

Description

`IndCnvInit` used to set up the indexing conveyor functionality.

Example

```
IndCnvInit M7, DI_Eye, indcnvdata1;
```

Arguments

```
IndCnvInit MechUnit, Signal, indcnvdata1;
```

MechUnit

Mechanical Unit

Data type: `mechunit`

The name of the mechanical unit.

Signal

Signal

Data type: `signaldi`

The name of the digital input signal that triggers the indexing movement.

indcnvdata

Data type: `indcnvdata`

IndCnvData speed, acceleration, productsperpick, productsperindex, accuracytuning

Error handling

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

`ERR_INT_NOTVAL`	Not valid integer, decimal value
`ERR_NO_ALIASIO_DEF`	The signal variable is a variable declared in RAPID. It has not been connected to an I/O signal defined in the I/O configuration with instruction `AliasIO`.
`ERR_NORUNUNIT`	If there is no contact with the I/O unit

10.3.8 IndCnvEnable and IndCnvDisable

Description

`IndCnvEnable` is used to set the system in indexing mode. An indexing movement will be executed when the signal is triggered (indexing mode).

`IndCnvDisable` is used to stop listening to the digital input signal. No indexing movement will be performed even if the signal is triggered.

Note

It is not possible to jog or run `Move` instructions on the indexing conveyor until a `IndCnvReset` instruction has been executed.

Example

```
IndCnvEnable M7;
IndCnvDisable M7;
```

Arguments

```
IndCnvEnable MechUnit;
IndCnvDisable MechUnit;
```

MechUnit

Mechanical Unit

Data type: `mechunit`

The name of the mechanical unit.

Error handling

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

`ERR_INDCNV_ORDER`	An instruction requires execution of `IndCnvInit` before it is executed.

10.3.9 IndCnvReset

Description

`IndCnvReset` ends indexing mode and sets the system to normal mode which makes it possible to jog and run `Move` instructions.

Example

```
IndCnvReset M7;
```

Arguments

```
IndCnvReset MechUnit;
```

MechUnit

Mechanical Unit

Data type: `mechunit`

The name of the mechanical unit.

10.3.10 IndCnvAddObject

Description

To manually add an object to the object queue, the RAPID instruction `IndCnvAddObject` can be executed.

Example

```
IndCnvAddObject M7;
```

Arguments

```
IndCnvAddObject MechUnit;
```

MechUnit

Mechanical Unit

Data type: `mechunit`

The name of the mechanical unit.

10.3.11 RAPID programming example

Description

In this example an IRB 360 is picking a batch of products from an indexing conveyor (CNV1) and placing the products on the outfeed conveyor (CNV2). The outfeed conveyor is not controlled by IRC5.

Example code

```
MODULE MainModule
TASK PERS wobjdata wobj1:= [FALSE,FALSE,"CNV1",[[0,0,0],
     [1,0,0,0]],[[0,0,0],[1,0,0,0]]];
TASK PERS wobjdata wobj2:= [FALSE,FALSE,"CNV2",[[0,0,0],
     [1,0,0,0]],[[0,0,0],[1,0,0,0]]];
CONST robtarget WaitPos:= [[-129.82,1.57,-
     924.52],[0,0.999848,-0.0174063,0],
     [0,0,0,0],[0,9E+09,9E+09,9E+09,0,0]];
CONST robtarget Cnv1_Above:= [[24.81,40.73,34.42],
     [0.000115829,-0.753048,0.657965,-0.000112099],
     [0,1,0,0],[171.015,9E+09,9E +09,9E+09,277.57,400]];
CONST robtarget Cnv1_Below:= [[24.81,40.73,-9.77],
     [0.000115827,-0.75306,0.657952,-0.000112101],
     [0,1,0,0],[171.015,9E+09,9E +09,9E+09,277.57,400]];
CONST robtarget Cnv2_Above:= [[-23.46,-1.68,201.85],
     [1.72038E-05,0.997874,0.0651587,-0.000988565],
     [0,0,0,0],[158.015,9E+09,9E +09,9E+09,277.57,0]];
CONST robtarget Cnv2_Below:= [[-23.46,-1.68,55.28],
     [1.72335E-05,0.997876,0.0651288,-0.000988564],
     [0,0,0,0],[158.015,9E+09,9E +09,9E+09,277.57,0]];
PERS tooldata Tool_2:=[TRUE,[[0,0,56.5],[1,0,0,0]],
     [2,[0,0,26],[1,0,0,0],0,0,0.0016]];
VAR triggdata EaciPick;
VAR triggdata EaciPlace;
VAR speeddata speed_eaci;
VAR indcnvdata indcnvdata1:=[0,0,0,0,0,0,0];
VAR num prod_cnt;
CONST stoppointdata
     stoppoint_eaci:=[3,FALSE,[0,0,0,0],0,0.035,"",0,0];
PROC main()
  indcnvdata1.speed := 2000;
  indcnvdata1.acceleration := 25000;
  indcnvdata1.productsperpick := 6;
  indcnvdata1.productsperindex := 1;

  prod_cnt := 0;

  speed_eaci.v_tcp:= 5000;
  speed_eaci.v_ori:= 10000;

  ! Activate mechanical units
  ActUnit M7;
  ActUnit CNV1;
```

Continues on next page

```
ActUnit CNV2;

! Remove from queue and drop objects
PulseDO \PLength:=0.1,c1RemAllPObj;
PulseDO \PLength:=0.1,c2RemAllPObj;
DropWObj wobj1;
IF c2Connected = 1 THEN
  DropWObj wobj2;
ENDIF

! Set up pick and place trigg data
TriggEquip EaciPick, 2, 0.05 \DOp:=EaciSuck, 1;
TriggEquip EaciPlace, 2, 0.05 \DOp:=EaciSuck, 0;

! Set indexing conveyor CNV1 in indexing mode and start listening
     to the photo eye signals
IndCnvInit M7, DI_Eye, indcnvdata1;IndCnvEnable M7;
! Go to init position
MoveL WaitPos, v1000, fine, Tool_2;

WHILE TRUE DO

  ! CNV1 Pick
  WaitWObj wobj1\RelDist:=300;
  MoveL Cnv1_Above, speed_eaci, z20, Tool_2\WObj:=wobj1;
  TriggL Cnv1_Below, speed_eaci, EaciPick,
       z1\Inpos:=stoppoint_eaci, Tool_2\WObj:=wobj1;
  MoveL Cnv1_Above, speed_eaci, z20, Tool_2\WObj:=wobj1;
  IF prod_cnt >= 1 THEN
    DropWObj wobj2;
  ENDIF
  ! CNV2 Place
  WaitWObj wobj2\RelDist:=50;
  MoveL Cnv2_Above, speed_eaci, z20, Tool_2\WObj:=wobj2;


  TriggL Cnv2_Below, speed_eaci, EaciPlace,
       z1\Inpos:=stoppoint_eaci, Tool_2\WObj:=wobj2;
  MoveL Cnv2_Above, speed_eaci, z20, Tool_2\WObj:=wobj2;

  prod_cnt := prod_cnt + 1;
  TPErase;
  TPWrite " Number of products: "\Num:=prod_cnt;

  DropWObj wobj1;
ENDWHILE

! Move program pointer to the instructions below to enable e.g.
     jogging
! Stop listening to the photo eye signals
IndCnvDisable M7;
```

Continues on next page

10.3.11 RAPID programming example
Continued

```
    ! Set indexing conveyor in normal mode
    IndCnvReset M7;
ENDPROC
ENDMODULE
```

10.3.12 Minimizing trigger time

Trigger time

If the parameter *Time before indexing move* is set too short, the error **50423 IndCnv Time before indexing move too low** can occur.

To minimize the time between the trigger from the photo eye and start of the conveyor movement, follow the description below:

1. Set *Time before indexing move* to a high value, for example 0.4 s.
2. If trigger time is critical there is a chance to reduce the time by changing the parameters as suggested in *Possible solutions on page 127*.
3. Run the full application with the robot coordinated to CNV1 and with full robot payload.
4. Run the service routine `IndCnvOptimalTimeBefore` to get a proposed optimal time before.
5. Update *Time before indexing move* to match the value proposed by the service routine.

There is a risk that the value suggested by the service routine `IndCnvOptimalTimeBefore` is too small so some extra 10 ms margin might be needed.

Possible solutions

Change the following parameters in the type *Motion Planner*.

Parameter	Value
Dynamic Resolution	0.3333
Path Resolution	0.3333
Queue Time	0.032256
Group Queue Time	0.016128

Possible limitations

If the error **50082 Deceleration limit** occurs, return to the original setting of *Queue Time* (which is 0.064512), in the type *Motion Planner* (topic *Motion*).

If the error **50226 Motor reference error** occurs, return to the original setting of *Group Queue Time* (which is 0.032256), in the type *Motion Planner* (topic *Motion*).

This page is intentionally left blank

11 Conveyor tracking and MultiMove

11.1 About conveyor tracking and MultiMove

Note

The option *MultiMove* is not available in RobotWare 5.60.

Examples of use

Here are some examples of applications where conveyor tracking is combined with MultiMove:

- Several robots can work on the same object moving on a conveyor.
- Several robots can cooperate to pick objects on a conveyor.

Additional information

If the option *Indexing Conveyor Control* is used for a conveyor in a MultiMove system, then some additional parameters must be defined. See *System parameters and configuration files on page 112*. See also *Indexing conveyor with servo control (Indexing Conveyor Control) on page 108*.

Application manual - MultiMove

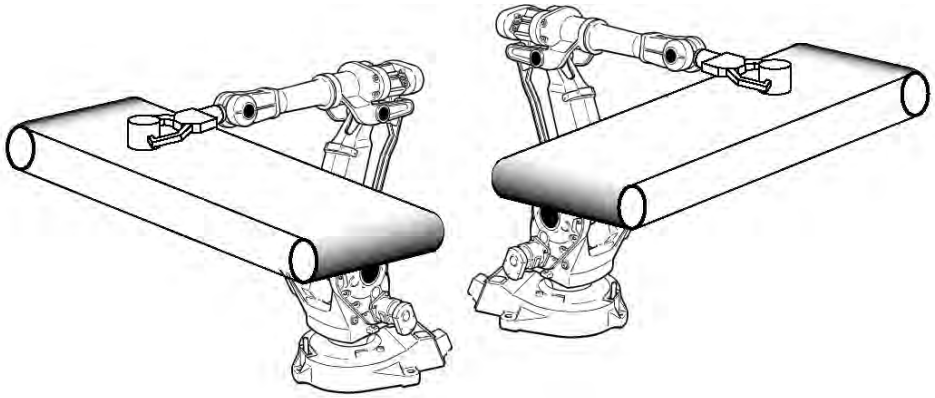
Continues on next page

Two application examples

This manual describes two examples of robot system setups to demonstrate how conveyor tracking can be combined with MultiMove. They are called *UnsyncCnv* and *SyncCnv*. See *Configuration example for UnsyncCnv on page 131*, and *Configuration example for SyncCnv on page 133*.

UnsyncCnv

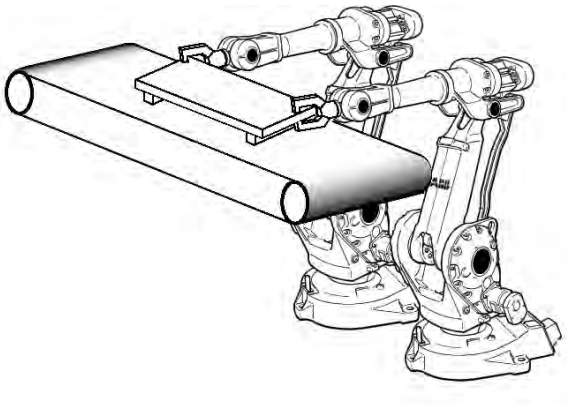
In the example UnsyncCnv, two robots work independently on one work piece for each robot. They do not cooperate in any way and do not have to wait for each other. There is one conveyor mechanical unit for each robot. Two encoder interfaces can be connected to the same encoder.



xx1200001107

SyncCnv

In the example SyncCnv, two robots arc weld on the same work piece. The work object is moved by a conveyor. One conveyor mechanical unit is used in a separate motion planner.



xx1200001108

11.2 Configuration example for UnsyncCnv

About this example

This section describes how to configure the example UnsyncCnv, with two independent robots. The robots are handled by one task each.

Configuration

Task

Task	Type	MotionTask	Use Mechanical Unit Group
T_ROB1	NORMAL	Yes	rob1
T_ROB2	NORMAL	Yes	rob2

Mechanical Unit Group

Name	Robot	Mech Unit 1	Use Motion Planner
rob1	ROB_1	CNV1	motion_planner_1
rob2	ROB_2	CNV2	motion_planner_2

Motion Planner

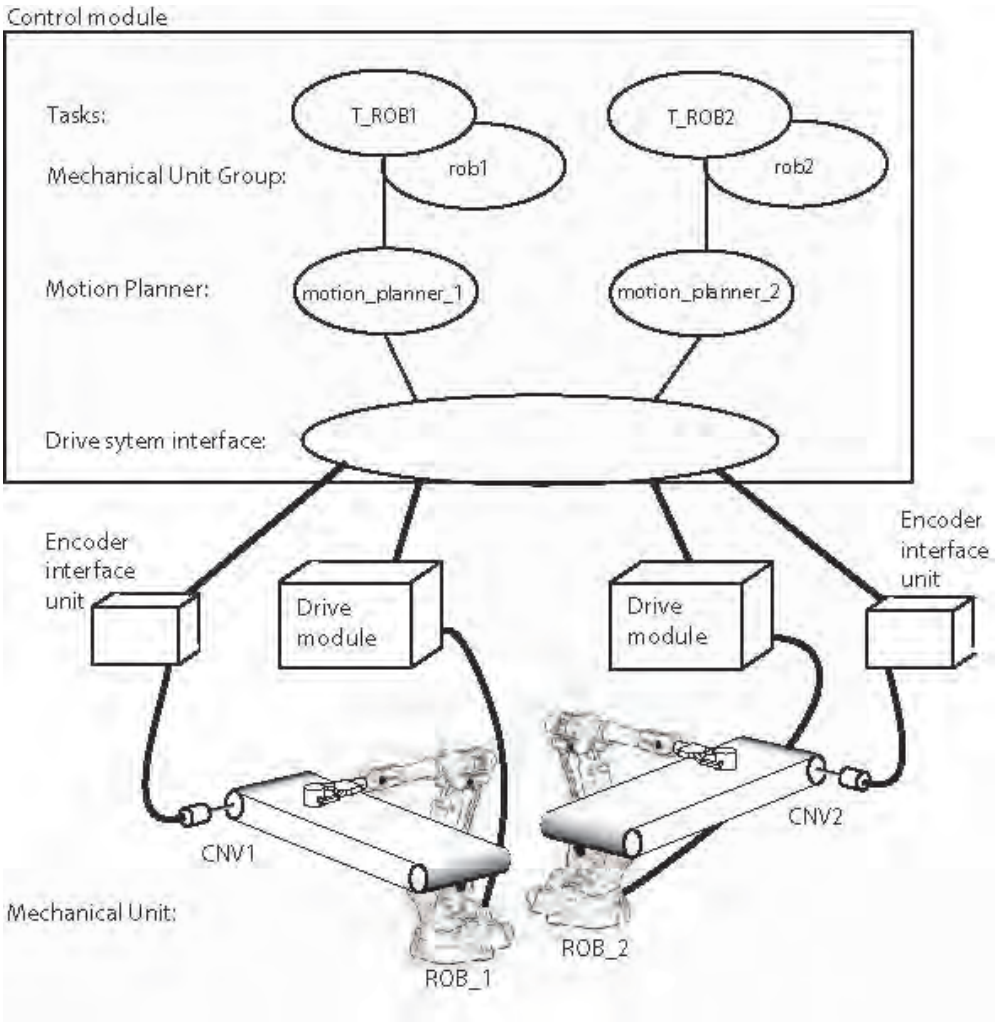
Name
motion_planner_1
motion_planner_2

Mechanical Unit

Name	Allow Move of User Frame	Activate at Start-up	Deactivation Forbidden	Use Brake Relay
ROB_1	No	Yes	No	
ROB_2	No	Yes	No	
CNV1	Yes	No	No	rob1_brake
CNV2	Yes	No	No	rob2_brake

Continues on next page

Illustration



xx1200001109

11.3 Configuration example for SyncCnv

About this example

This section describes how to configure the example SyncCnv, with two robots and one positioner. Each mechanical unit is handled by a separate task.

Configuration

Task

Task	Type	MotionTask	Use Mechanical Unit Group
T_ROB1	NORMAL	Yes	rob1
T_ROB2	NORMAL	Yes	rob2
T_CONV3	NORMAL	Yes	conv3

Mechanical Unit Group

Name	Robot	Mech Unit 1	Use Motion Planner
rob1	ROB_1		motion_planner_1
rob2	ROB_2		motion_planner_2
conv3		CNV3	motion_planner_3

Motion Planner

Name
motion_planner_1
motion_planner_2
motion_planner_3

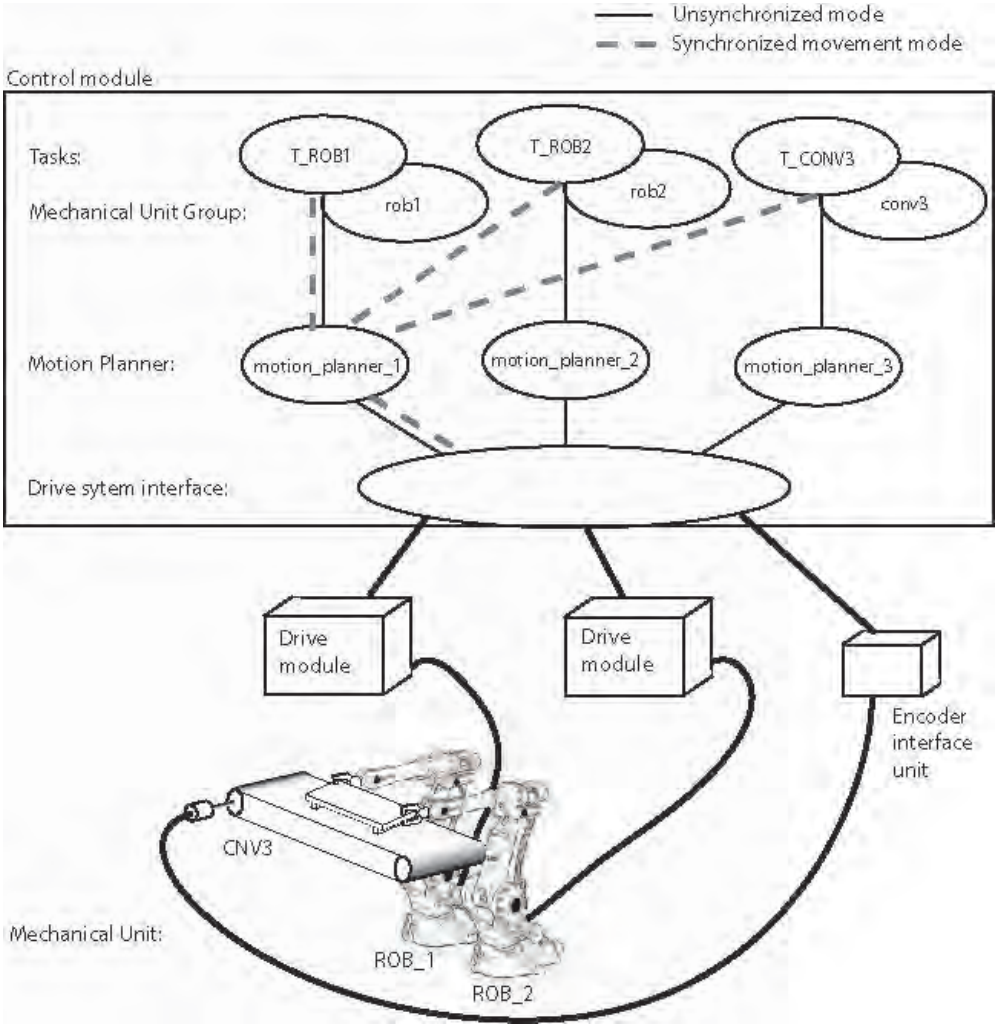
Mechanical Unit

Name	Allow Move of User Frame	Activate at Start-up	Deactivation Forbidden	Use Brake Relay
ROB_1	No	Yes	No	
ROB_2	No	Yes	No	
CNV3	Yes	Yes	No	rob1_brake

Continues on next page

11.3 Configuration example for SyncCnv
Continued

Illustration



xx1200001110

Calibration overview

For unsynchronized movements each conveyor must be calibrated with its motion group robot (after the base calibration of the robot):

- Cnv1 with Robot_1
- Cnv2 with Robot_2

For synchronized movements Cnv3 is calibrated with one robot only: Robot_1.

11.4 Tasks and programming techniques

Introduction to tasks

Each task program can handle the movements for one robot and up to 6 additional axes. Several tasks can be used, each containing a program quite similar to the program of the main task in a single robot application. For more information about tasks, see *Multitasking* in *Application manual - Controller software IRC5*.

One task program per robot

Each task program can only handle one TCP. This means that you must have one task for each robot.

Conveyor in separate tasks

Conveyors that move a work object can be handled by the same task program as one of the robots for un-synchronized movements. For synchronized movements where the conveyor should be able to move independently of the robots, it is best to have a separate task program for the conveyor.

Modifying positions

When modifying positions it is not possible to just stop the conveyor and modify the position. If this is done, there will be an offset in the position that corresponds to the length of the conveyor movement. This is since the conveyor and the robots belong to different mechanical unit groups, see *Configuration example for SyncCnv on page 133*, and the position of the conveyor is not known to the robots at this point.

To be able to modify positions, the conveyor must belong to the same mechanical unit group as the robot. This is done by creating "fake" conveyors that are only used for modifying positions.

Example

Below there is a configuration example for topic *System*. Conveyors `CNV1` and `CNV2` are "fake" conveyors, and conveyor `CNV3` is the real conveyor.

Task:

Task	Type	MotionTask	Use Mechanical Unit Group
T_ROB1	NORMAL	Yes	ROB1
T_ROB2	NORMAL	Yes	ROB2
T_CNV	NORMAL	Yes	GCNV

Mechanical Unit Group:

Name	Robot	Mech Unit 1	Use Motion Planner
ROB1	ROB_1	CNV1	motion_planner_1
ROB2	ROB_2	CNV2	motion_planner_2
GCNV		CNV3	motion_planner_3

Continues on next page

Mechanical Unit:

Name	Allow Move of User Frame	Activate at Start-up	Deactivation Forbidden	Use Brake Relay
CNV1	Yes	No	No	rob1_brake
CNV2	Yes	No	No	rob2_brake
CNV3	Yes	Yes	No	rob1_brake

In the configuration for topic *Process*, the "fake" conveyors `CNV1` and `CNV2` must be connected to the same encoder as `CNV3`, and must be configured using the same I/O signals as `CNV3`, see *Combining synchronized and un-synchronized mode on page 143*.

The last step is to create unsynchronized RAPID procedures for each task with move instructions where the positions can be modified.

	Action
1	Create new work objects for the "fake" conveyors, for example `wobj_CNV1` for `ROB_1` and `wobj_CNV2` for `ROB_2`. (In the synchronized production program both robots uses the same work object, for example `wobj_CNV3`.)

2	Copy the production procedures for robot 1, remove the synchronization, and replace the work objects.
	For example, the production procedure in synced task `T_ROB1`:
	`MoveL p101\ID:=10, v300, fine, tool1\WObj:=wobj_CNV3;`
	`MoveL p102\ID:=20, v300, fine, tool1\WObj:=wobj_CNV3;`
	`MoveL p103\ID:=30, v300, fine, tool1\WObj:=wobj_CNV3;`
	is copied to a new routine and modified:
	`MoveL p101, v300, fine, tool1\WObj:=wobj_CNV1;`
	`MoveL p102, v300, fine, tool1\WObj:=wobj_CNV1;`
	`MoveL p103, v300, fine, tool1\WObj:=wobj_CNV1;`

3	Copy the production procedures for robot 2, remove the synchronization, and replace the work objects.
	For example, the production procedure in synced task `T_ROB2`:
	`MoveL p201\ID:=10, v300, fine, tool1\WObj:=wobj_CNV3;`
	`MoveL p202\ID:=20, v300, fine, tool1\WObj:=wobj_CNV3;`
	`MoveL p203\ID:=30, v300, fine, tool1\WObj:=wobj_CNV3;`
	is copied to a new routine and modified:
	`MoveL p201, v300, fine, tool1\WObj:=wobj_CNV2;`
	`MoveL p202, v300, fine, tool1\WObj:=wobj_CNV2;`
	`MoveL p203, v300, fine, tool1\WObj:=wobj_CNV2;`

4	Modify the positions for each robot.

11.5 Independent movements, example UnsyncCnv

ROB1 task program

```
MODULE module1
  TASK PERS wobjdata wobj1 := [ FALSE, TRUE, "", [ [500, -200,
       1000], [1, 0, 0 ,0] ], [ [100, 200, 100], [1, 0, 0, 0] ]
       ];
  TASK PERS wobjdata wobjcnv1 := [ FALSE, FALSE, "CNV1", [ [0,0,
       0], [1, 0, 0 ,0] ], [ [0, 0, 0], [1, 0, 0, 0] ] ];
  TASK PERS tooldata tool1 := ...
  CONST robtarget p11 := ...
  ...
  CONST robtarget p14 := ...

  PROC main()
    ...
    IndependentMove;
    ...
  ENDPROC

  PROC IndependentMove()
    MoveL p11, v500, fine, tool1\WObj:=wobj1;
    WaitWObj wobjcnv1\RelDist:=10;
    MoveC p12, p13, v500, z10, tool1\WObj:=wobjcnv1;
    MoveC p14, p11, v500, fine, tool1\WObj:=wobj1;
  ENDPROC
ENDMODULE
```

ROB2 task program

```
MODULE module2
  TASK PERS wobjdata wobj2 := [ FALSE, TRUE, "", [ [500, -200,
       1000], [1, 0, 0 ,0] ], [ [100, 1200, 100], [1, 0, 0, 0] ]
       ];
  TASK PERS wobjdata wobjcnv2 := [ FALSE, FALSE, "CNV2", [ [0,0,
       0], [1, 0, 0 ,0] ], [ [0, 0, 0], [1, 0, 0, 0] ] ];
  TASK PERS tooldata tool2 := ...
  CONST robtarget p21 := ...
  ...
  CONST robtarget p24 := ...

  PROC main()
    ...
    IndependentMove;
    ...
  ENDPROC

  PROC IndependentMove()
    MoveL p21, v500, fine, tool2\WObj:=wobj2;
    WaitWObj wobjcnv2\RelDist:=10;
    MoveL p22, v500, z10, tool2\WObj:=wobjcnv2;
    MoveL p23, v500, z10, tool2\WObj:=wobjcnv2;
```

Continues on next page

```
    MoveL p24, v500, z10, tool2\WObj:=wobjcnv2;
    MoveL p21, v500, fine, tool2\WObj:=wobj2;
  ENDPROC
ENDMODULE
```

11.6 Coordinated synchronized movements, example SyncCnv

ROB1 task program

```
MODULE module1
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  PERS tasks all_tasks{3} := [["ROB1"],["ROB2"],["CONV3"]];
  PERS wobjdata wobjcnv3 := [ FALSE, FALSE, "CNV3", [ [0, 0, 0],
       [1, 0, 0 ,0] ], [ [0, 0, 0], [1, 0, 0, 0] ] ];
  TASK PERS tooldata tool1 := ...
  CONST robtarget p100 := ...
  ...
  CONST robtarget p199 := ...

  PROC main()
    ...
    SyncMove;
    ...
  ENDPROC

  PROC SyncMove()
    MoveJ p100, v1000, z50, tool1;
    WaitSyncTask sync1, all_tasks;
    MoveL p101, v500, fine, tool1\WObj:=wobj1;
    SyncMoveOn sync2, all_tasks;
    MoveL p102\ID:=10, v300, fine, tool1\WObj:=wobjcnv3;
    MoveC p103, p104\ID:=20, v300, z10, tool1\WObj:=wobjcnv3;
    MoveL p105\ID:=30, v300, z10, tool1\WObj:=wobjcnv3;
    MoveC p106, p101\ID:=40, v300, fine, tool1\WObj:=wobj1;
    SyncMoveOff sync3;
    MoveL p199, v1000, fine, tool1;
    UNDO
    SyncMoveUndo;
  ENDPROC
ENDMODULE
```

ROB2 task program

```
MODULE module2
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  PERS tasks all_tasks{3} := [["ROB1"],["ROB2"],["CONV3"]];
  PERS wobjdata wobjcnv3 := [ FALSE, FALSE, "CNV3", [ [0, 0, 0],
       [1, 0, 0 ,0] ], [ [0, 0, 0], [1, 0, 0, 0] ] ];

  TASK PERS tooldata tool2 := ...
  CONST robtarget p200 := ...
  ...
  CONST robtarget p299 := ...
```

Continues on next page

```
  PROC main()
    ...
    SyncMove;
    ...
  ENDPROC

  PROC SyncMove()
    MoveJ p200, v1000, z50, tool2;
    WaitSyncTask sync1, all_tasks;
    MoveL p201, v500, fine, tool2 \WObj:=wobj2;
    SyncMoveOn sync2, all_tasks;
    MoveL p202\ID:=10, v300, fine, tool2\WObj:=wobjcnv3;
    MoveC p203, p204\ID:=20, v300, z10, tool2\WObj:=wobjcnv3;
    MoveL p205\ID:=30, v300, z10, tool2\WObj:=wobjcnv3;
    MoveC p206, p201\ID:=40, v300, fine, tool2\WObj:=wobj2;
    SyncMoveOff sync3;
    MoveL p299, v1000, fine, tool2;
    UNDO
    SyncMoveUndo;
  ENDPROC
ENDMODULE
```

CONV3 task program

```
MODULE module3
  VAR syncident sync1;
  VAR syncident sync2;
  VAR syncident sync3;
  PERS tasks all_tasks{3} := [["ROB1"],["ROB2"], ["CONV3"]];
  PERS wobjdata wobjcnv3 := [ FALSE, FALSE, "CNV3", [ [0, 0, 0],
       [1, 0, 0 ,0] ], [ [0, 0, 0], [1, 0, 0, 0] ] ];
  CONST jointtarget angle_0 := [ [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9],
       [ 0, 9E9, 9E9, 9E9, 9E9, 9E9] ];
  ...
  CONST jointtarget angle_360 := [ [ 9E9, 9E9, 9E9, 9E9, 9E9, 9E9],[
       360, 9E9, 9E9, 9E9, 9E9, 9E9] ];
  PROC main()
    ...
    SyncMove;
    ...
  ENDPROC


  PROC SyncMove()
    MoveExtJ angle_neg20, vrot50, fine;
    WaitSyncTask sync1, all_tasks;
    ! Wait for the robots
    WaitWObj wobjcnv3;

    SyncMoveOn sync2, all_tasks;
    MoveExtJ angle_20\ID:=10, vrot100, fine;
```

Continues on next page

```
    WaitWObj wobjcnv3\RelDist:=100;
    MoveExtJ angle_160\ID:=20, vrot100, z10;
    MoveExtJ angle_200\ID:=30, rot100, z10;
    MoveExtJ angle_340\ID:=40, rot100, fine;
    SyncMoveOff sync3;
    DropWobj wobjcnv3;
    UNDO
    SyncMoveUndo;
  ENDPROC
ENDMODULE
```

11.7 Motion principles

Robot speeds

When the movements of several robots are synchronized, all robots adjust their speed to finish their movements simultaneously. This means that the robot movement that takes the longest time will determine the speed of the other robots.

11.8 Combining synchronized and un-synchronized mode

Introduction

For a combination of synchronized and un-synchronized mode is needed with a single conveyor there must be two mechanical units for un-synchronized mode. For example CNV1 and CNV2 can be connected to the same encoder. CNV3 can be configured using the same I/O signals as CNV1 in the topic *Process*. Replace default *C2xx* signals name with *C1xx* (that is, *position_signal c2position* becomes *position_signal c1position*).

Configuration

The configuration file Proc.cfg will look like this:

Conveyor

```
-name "CNV1" -sensor_type "CAN" -use_sensor "CAN1"\
-adjustment_speed 250 -min_dist -600 -max_dist 20000\
-correction_vector_ramp_length 10
#
-name "CNV2" -sensor_type "CAN" -use_sensor "CAN2"\
-adjustment_speed 250 -min_dist -600 -max_dist 20000\
-correction_vector_ramp_length 10
#
-name "CNV3" -sensor_type "CAN" -use_sensor "CAN3"\
-adjustment_speed 250 -min_dist -600 -max_dist 20000\
-correction_vector_ramp_length 10
#
```

Conveyor CAN sensor

```
-name "CAN1" -eio_unit_name "Qtrack1" -connected_signal
     "c1Connected"\
-position_signal "c1Position" -velocity_signal "c1Speed"\
-null_speed_signal "c1NullSpeed" -wait_wobj_signal "c1WaitWObj"\
-drop_wobj_signal "c1DropWObj" -data_timestamp "c1DTimestamp"\
-rem_all_pobj_signal "c1RemAllPObj"\
-rem_one_pobj_signal "c1Rem1PObj"
#
-name "CAN2" -eio_unit_name "Qtrack1" -connected_signal
     "c1Connected"\
-position_signal "c1Position" -velocity_signal "c1Speed"\
-null_speed_signal "c1NullSpeed" -wait_wobj_signal "c1WaitWObj"\
-drop_wobj_signal "c1DropWObj" -data_timestamp "c1DTimestamp"\
-rem_all_pobj_signal "c1RemAllPObj"\
-rem_one_pobj_signal "c1Rem1PObj"
#
-name "CAN3" -eio_unit_name "Qtrack1" -connected_signal
     "c1Connected"\
-position_signal "c1Position" -velocity_signal "c1Speed"\
-null_speed_signal "c1NullSpeed" -wait_wobj_signal "c1WaitWObj"\
-drop_wobj_signal "c1DropWObj" -data_timestamp "c1DTimestamp"\
-rem_all_pobj_signal "c1RemAllPObj"\
-rem_one_pobj_signal "c1Rem1PObj"
```

This page is intentionally left blank

Index

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

3HAC050991-001, Rev D, en