

VALTrack Software Version s7.4.1

Reference Guide

1	INTRODUCTION.....	5
2	INSTALLATION	5
2.1	HARDWARE.....	5
2.1.1	CS8C equipped with STARC board.....	5
2.1.2	CS8C equipped with STARC2 board.....	6
2.1.3	CS8/CS8M/CS8HP.....	7
2.1.4	Encoder Wiring	7
2.2	SOFTWARE	8
2.2.1	Install the license key	8
2.2.2	Install VAL3 software package.....	9
3	SOFTWARE DESCRIPTION	11
3.1	LIBRARIES OVERVIEW	11
3.2	ENCODER LIBRARY	12
3.3	DETECTION LIBRARY	13
3.4	CONVEYOR LIBRARY	15
3.5	MOTION LIBRARY	16
3.6	CALIBRATION WIZARD	16
4	PROGRAMMING STEPS	17
4.1	CREATE A LIBRARY FOR YOUR DETECTION DEVICE	17
4.1.1	Programming the detection library.....	17
4.1.2	Variable description.....	20
	- public num nMaxNumOfObject	20
	- public num nNumOfPart.....	20
	- private num nObjectID[]	20
	- private trsf trObjectPos[]	20
	- private num nLatchedValue[]	20
4.1.3	Programs description.....	20
	- public create(num x_nDetNum, string x_sTask).....	20
	- private detection().....	20
	- public getMaxNumOfPart(num& x_nMaxNumOfPart).....	20
	- public getObjectData(num x_nIdx, num& x_nObjectID, trsf& x_trObjectPos).....	20
	- public getObjectLatch(num x_nIdx, num& x_nLatchedVal).....	21
	- public init(num x_nDetNum, string x_sTask)	21
	- public isDetected(num& x_nHowMany).....	21
	- public kill().....	21
	- public resetTask()	21
4.2	CREATE A CONVEYOR LIBRARY	21
4.2.1	Select a conveyor library to work with.....	22
4.2.2	Select an encoder library	22
4.2.3	Select a detection library.....	23
4.2.4	Setup the conveyor library	24
	- General setup page.....	25
	- Encoder setup page.....	26
	- Detection setup page.....	26
	- Conveyor setup page.....	27
4.2.5	Teach the conveyor library.....	28
	- F1: Complete calibration sequence	28
	- F2: Define Scale factor.....	28
	- F3: Conveyor geometry	28
	- F4: Define frame for vision.....	29
	- F6: New/Load conveyor library.....	29
4.3	CREATE YOUR MAIN APPLICATION	29
4.3.1	Tracking libraries.....	30
4.3.2	Program example for single tracking.....	30
4.3.2.1	Used variables.....	30
4.3.2.2	Select conveyor library	32
4.3.2.3	Teach positions	32

4.3.2.4	Working tracking window	32
4.3.2.5	Cycle description	33
4.3.3	Program example for double tracking and queueing	37
4.3.3.1	Used variables	38
4.3.3.2	Select conveyor library	42
4.3.3.3	Teach positions	42
4.3.3.4	Working tracking window and conveyor speed control	42
4.3.3.5	Queueing	48
4.3.3.6	Cycle description	57
4.3.4	Work with more than 2 conveyor libraries	61
4.3.5	Initializing and launching all tracking tasks and variables.	62
4.4	USE THE MOTION LIBRARY	64
4.4.1	Programs Description	66
- public	create(string x_sConveyorList)	66
- public	trackOn(num x_nConvNum, point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)	66
- public	trackOnJ(num x_nConvNum, point x_pLoc, tool x_tTool, joint x_jConfig, mdesc x_mDesc, num& x_nMoveID, num& x_nError)	67
- public	moveL(point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)	67
- public	moveC(point x_pInter, point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)	68
- public	trackOff(point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)	68
- public	trackOffJ(joint x_jLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)	68
- public	trackStopMove()	69
- public	stopTrkMove()	69
- public	disableQueue(num x_nConvNum)	69
- public	enableQueue(num x_nConvNum)	69
- public	flushQueue(num x_nConvNum, num& x_nError)	70
- public	getConvFrame(num x_nConvNum, frame& x_fConveyor, frame x_fRefFrame, num& x_nError) ..	70
- public	getConvSpeed(num x_nConvNum, num& x_nSpeed, num& x_nError)	70
- public	getDetectFrame(num x_nConvNum, frame& x_fDetection, frame x_fRefFrame, num& x_nError) ..	70
- public	getDistance(num x_nConvNum, point x_pLoc, num x_nTime, num& x_Dist2Up, num& x_nDist2Down, num& x_nError)	71
- public	getErrorMessage(string& x_sMessage)	71
- public	getNextID(num x_nConvNum, num& x_nNextID)	71
- public	getObjectData(num x_nConvNum, trsf& x_trObject, num& x_nError)	72
- public	getObjectLatch (num x_nConvNum, num& x_nLatchedValue, num& x_nError)	72
- public	getRobotArea(num x_nConvNum, num& x_nAreaNumber)	72
- public	getStatus(num x_nConvNum, num& x_nStatus)	72
- public	getTrackStatus (num& x_nTrackStatus)	73
- public	getVersion(string& x_sVersion)	73
- public	kill()	73
- public	removePart(num x_nConvNum)	73
- public	reset(num x_nConvNum, num& x_nError)	73
- public	sendToNextRobot (sio x_sSocketClient, num x_nTimeout, num x_nID, trsf x_trObjectPos, num x_nLatchedValue, num& x_nError)	74
- public	setCalibrating(num x_nConvNum, bool x_bMode)	74
- public	teach(num x_nConvNum, tool x_tTool, frame x_fFrame, x_point& x_pLoc, num& x_nError)	75
4.5	SPEED CONTROL WHILE TRACKING	76
4.5.1	Synchronization / De-synchronization Moves	76
4.5.2	Synchronized Moves	76
- void	\$setCartTrkLim(string sParamName, num nValue)	76
5	FREQUENTLY ASKED QUESTIONS	77
5.1	HOW TO CREATE AN ENCODER LIBRARY?	77
5.1.1	Variable description	77
- num	nLatchPeriod	77
- num	nEncValue	78
- num	nEncLatchedVal	78
- dio	diEncLatched	78
- aio	aiEncCurrentPos	78

- aio aiEncLatchedPos	78
5.1.2 Program description	78
- public create(num x_nEncNum, string x_sTask)	78
- private init(num x_nEncNum, string x_sTask).....	78
- public getValue(bool x_bUseLatch, num& x_nEncValue)	78
- public isLatched (bool& x_bLatched).....	78
- private encoder().....	78
5.2 HOW TO DEFINE LOCATIONS RELATIVE TO A CONVEYOR?	79
5.3 HOW OBJECTS ARE PUSHED INTO THE CONVEYOR'S QUEUE?	80
5.4 HOW DOES THE CAPTURE OF THE ENCODER POSITION WORKS?	81
5.5 HOW TO GET THE STATUS OF THE LIBRARIES?	82
5.6 HOW TO USE THE VAL3 LIBRARY COGNEX1 PROVIDED WITH VALTRACK SOFTWARE?.....	84
5.6.1 Configuration	84
5.6.2 Description of the library	85
5.6.3 Program description	85
- public init()	85
- public login(sio skSocketName, string sUserName, string sPassword, num& nHandle, bool& bError)..	86
- public getOnline(num nHandle, bool& bError)	86
- public readNumValue(num nHandle, string sCol, num nRow, num& nValue, bool& bError)	86
- public getErrorMessage(num nHandle, string& sErrMessage).....	86
6 ANNEX	87
6.1 SCHEMA OF TASKS	87
6.2 MULTI-PICK	88
6.3 WIRING	89
6.3.1 STARC 2 and Camera Cognex.....	89

1 INTRODUCTION

The conveyor tracking feature allows a robot to perform operations on an object that is transported by a conveyor into its working area. The conveyor must be equipped with an encoder device which allows evaluating in real time the position and speed of the transported objects. A detection device (sensor, vision system...) placed upstream from the robot recognizes the object's type and its position on the conveyor.

Stäubli provides a customizable VAL3 software package to help setting up and managing the conveyor tracking feature. This software package is available as an option starting with VAL3 version s6.5. Please refer to the readme.pdf file provided with the software package to check software and hardware compatibility.

2 INSTALLATION

2.1 Hardware

Depending on the controller type, the encoder IO interface may be installed on the controller.

2.1.1 CS8C equipped with STARC board

STARC board is not supported starting from VAL3 s7.0.

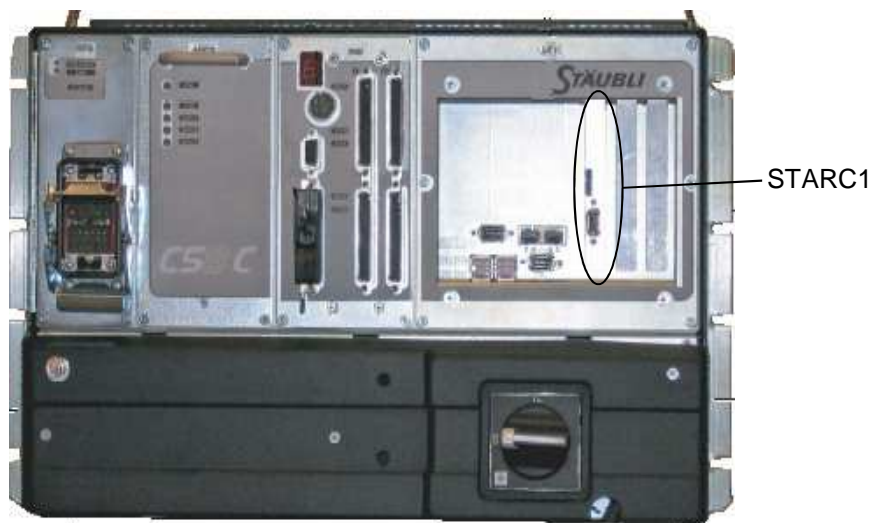
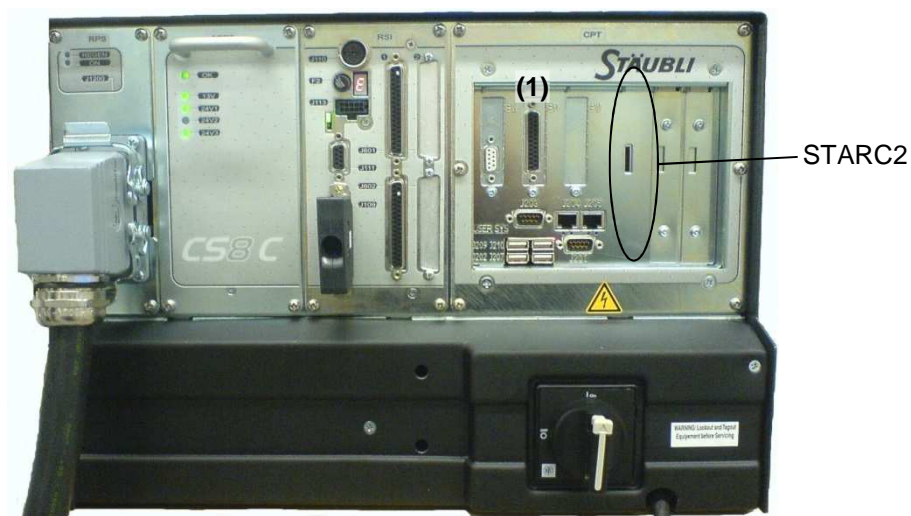


Figure 2.1

2.1.2 CS8C equipped with STARC2 board

The CS8C controller delivered with a STARC2 has an encoder interface through a D-Sub 25 female connector located on an optional DUAL ABZ encoder board (figure 2.2). A DUAL ABZ encoder boards has 2 encoders channels and up to 3 DUAL ABZ encoder boards can be installed on the controller. The encoder to be used with each channel must be a 5 volts incremental encoder with differential A, B, Z channels. The DUAL ABZ board provides the power supply for each encoder with a maximum available current of 250mA per channel.

One high speed trigger input for latching the encoder position is available for each channel. The trigger inputs are located on the same D-Sub 25 female connector of the DUAL ABZ encoder board. The reaction time of the position capture feature is < 1 ms.



Dual ABZ encoder board pinout channel 1										
Pin number	8	9	10	11	12	13	24	25	20	21
Signal name	Z-	Z+	B-	B+	A-	A+	Encoder GND	Encoder +VCC (5V)	GND Latch	LatchIn

Dual ABZ encoder board pinout channel 2										
Pin number	2	3	4	5	6	7	14	15	18	19
Signal name	Z-	Z+	B-	B+	A-	A+	Encoder +VCC (5V)	Encoder GND	GND Latch	LatchIn

Figure 2.2

For details wiring STARC2 see the annex 2.

2.1.3 CS8/CS8M/CS8HP

The encoder interface hardware used on these controllers is a fieldbus board such as Profibus (figure 2.3). Thus, it is possible to connect a wide variety of encoders. The high speed position capture is not available on these controllers, therefore the position capture feature must be available on the encoder interface. The encoder current position and the encoder latched value must be accessible at all times in two different variables, an encoder interface where the encoder current position and the encoder latched value are available under the same variable is not adapted for VALTrack.

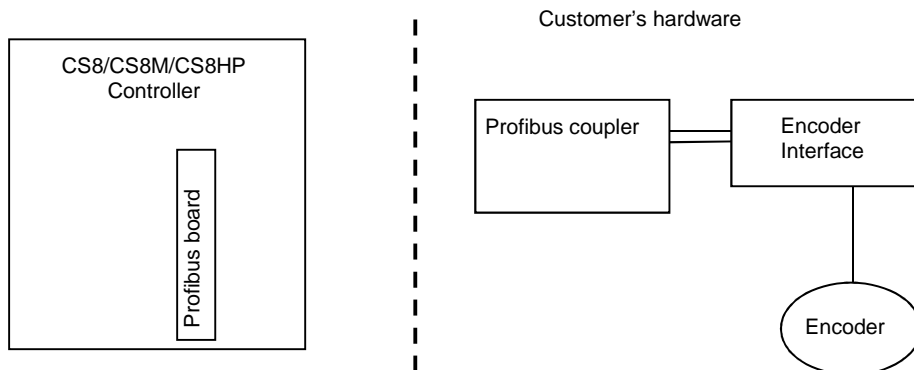


Figure 2.3

Note: The following devices can be used at the customer site as a proven hardware:

Wago Profibus coupler ref 750-333

Wago Incremental encoder interface ref 750-631

2.1.4 Encoder Wiring

For rollover management on the encoder value, VALTrack needs to see the encoder value increase when the conveyor is moving forward. If the encoder value is decreasing when the conveyor is moving forward, it is possible to change the direction of the encoder count by electrically swap the pair A and Inversed A with the pair B and Inversed B.

2.2 Software

The VAL3 software package delivered as the conveyor tracking option is composed of several VAL3 libraries. The main user's VAL3 project must have pointers to some of these libraries to be able to perform "conveyor tracking".

2.2.1 Install the license key

The use of the VAL3 software package requires a license to be installed on the CS8x controller. This license is a 16 digits key that must be entered by using Stäubli Robotic Suite (SRS). If the VALTrack option has been purchased at the same time as the robot, the license has been installed on the controller before it left the factory. Otherwise proceed as followed:

- Start Stäubli Robotic Studio
- Under the tab *Home*, click on the button *Remote Options*.
- A window called *Target Properties* appears.
- In the *IP* field, enter the IP address of the controller. (e.g. 192.168.0.254)
- Enter a valid login name in the field *User name* (e.g. default)
- Enter the password associated with the login name (e.g. no password for default profile)
- Press OK to continue
- A new tab document called *Options* is opened. It displays all possible options for the controller.
- Click on the *valTrack* option to display the pull down menu and select *Demo* or *Activated* button
- If *Demo* mode is selected, the use of VALTrack software package is limited to 2 hours. Once this period has expired, the controller needs to be rebooted to continue for the next 2 hours.
- If *Activated* mode is selected, enter the 16 digits key and press OK to confirm.
- Reboot the controller for the changes to take effect.

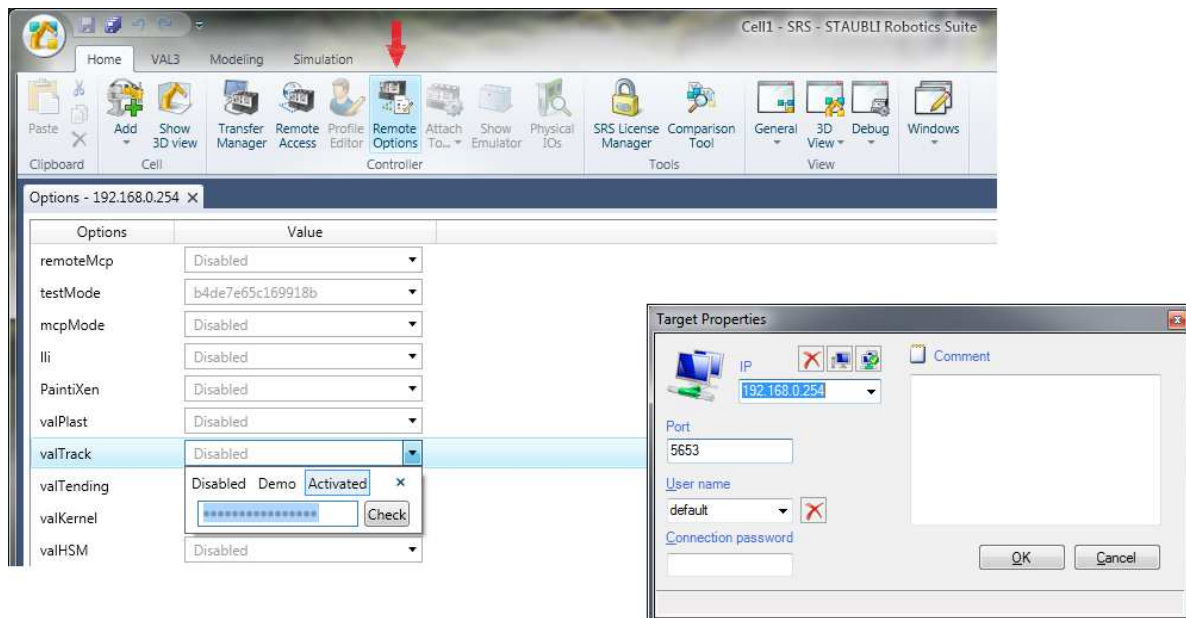


Figure 2.4

2.2.2 Install VAL3 software package

The installation requires an Ethernet connection between the controller and your PC. If the VALTrack option has been purchased at the same time as the robot, the software has been installed on the controller before it left the factory.

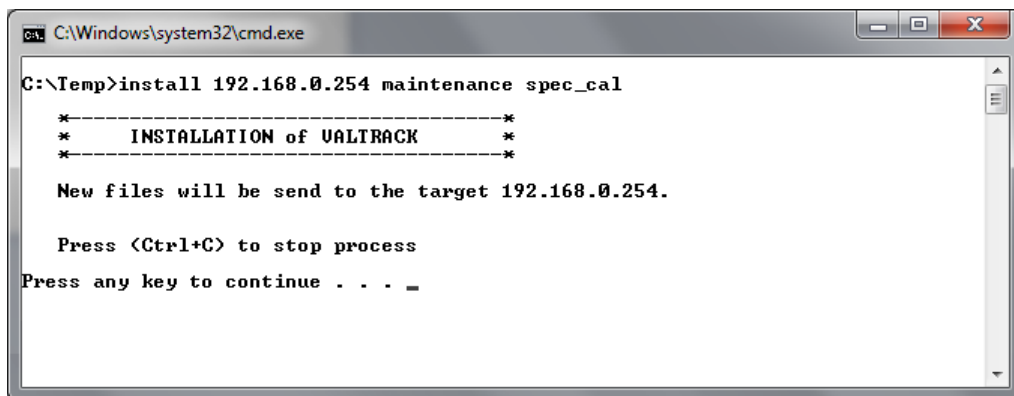
1. Copy the content of the directory Software from the CD delivered with the software package VALTrack to a destination folder on your PC.
2. Open the Microsoft® Windows command line: cmd.exe.
3. Navigate to the folder where you have the software package (see figure 2.5).
4. The installation tool is a basic batch file called *install.bat* that must be executed with the following syntax :

install IpAddress profile password

Parameters

<i>IpAddress</i>	IP address of the controller.
<i>profile</i>	One of the user profiles defined in the CS8; use "maintenance".
<i>password</i>	The network password of the considered profile; use "spec_cal" for the "maintenance" login

In the example below, the user changed the default folder to *C:\Temp*, in which the software package has been copied to.



```
C:\Windows\system32\cmd.exe

C:\Temp>install 192.168.0.254 maintenance spec_cal

*-----*
*  INSTALLATION of VALTRACK  *
*-----*

New files will be send to the target 192.168.0.254.

Press <Ctrl+C> to stop process
Press any key to continue . . . _
```

Figure 2.5

Once the installation is successfully performed, the following folders/files must be present in the USRAPP folder on the robot controller.

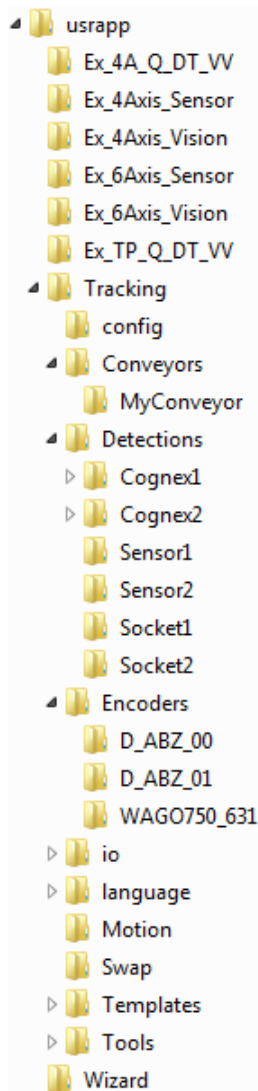


Figure 2.6

Note:

-The main folder is called Tracking. It appears as an application at the Application Manager but **it is not an application to be loaded in memory.**

-The folder Tracking\Conveyors contains all the conveyor's libraries available in the system. This folder is empty after installation of the VAL3 software package but it will be filled with the calibration files created by the Wizard application (e.g. file MyConveyor in the figure 2.6).

-The folder Tracking\Detections contains all the detection's libraries available in the system. These libraries are developed to match a specific detection device.

-The folder Tracking\Encoders contains the encoder's libraries available in the system. These libraries are developed to match an encoder reading device.

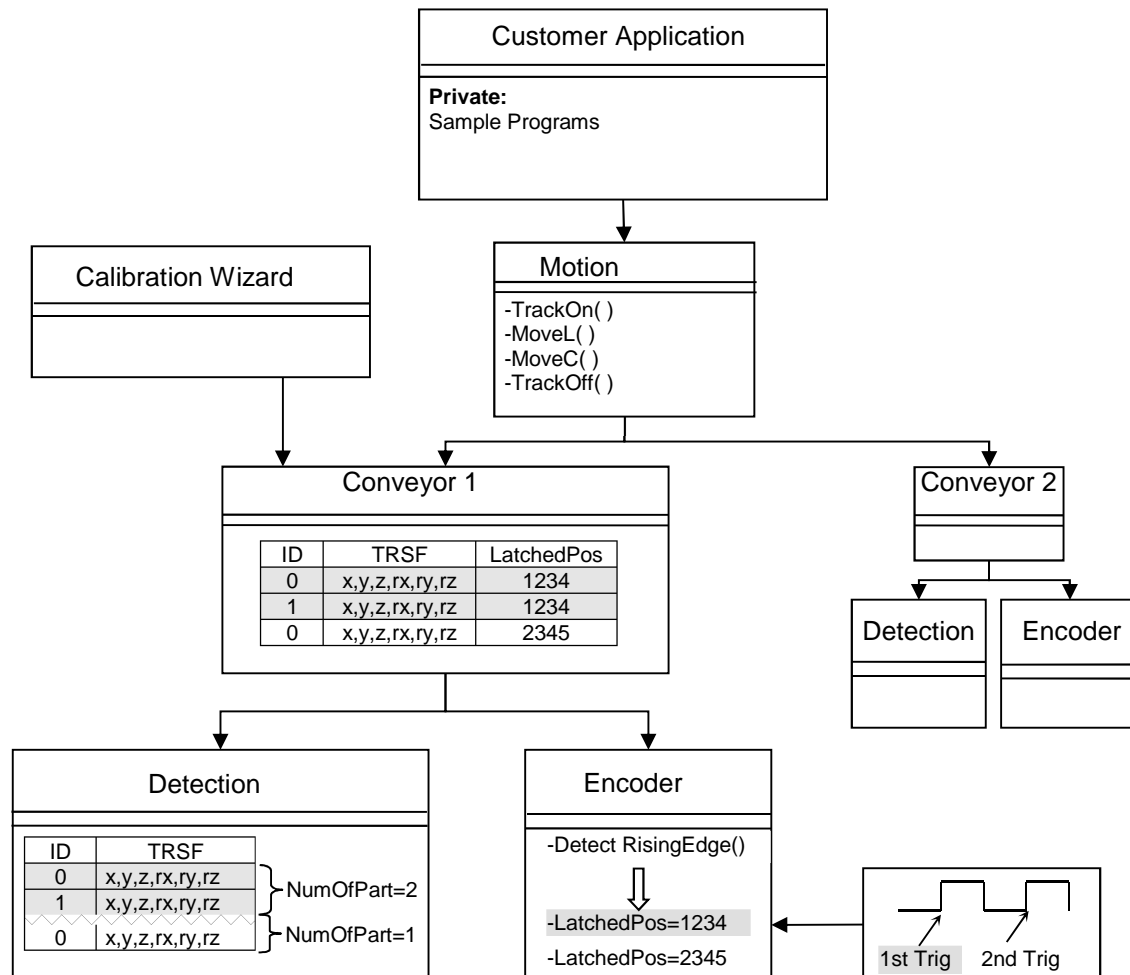
-The Wizard folder contains the application used to perform the calibration of a conveyor.

-4 examples for applications with 1 conveyor: *Ex_4axis_Sensor*, *Ex_4axis_Vision*, *Ex_6axis_Sensor*, *Ex_6axis_Vision*, containing each one an application example depending on the type of robot and the type of detection device used.

-2 examples for applications with 2 conveyors: *Ex_4A_Q_DT_VV* and *Ex_TP_Q_DT_VV*. These applications are developed for applications where there are 2 conveyors and multiple robots over the same conveyor. The detection device on both conveyors is of type vision system (the trsf of the objects is given by the detection library). If there is multiple robots working on the same conveyor, the objects can be transferred from one robot to the next robot installed downstream of the conveyor.

3 SOFTWARE DESCRIPTION

3.1 Libraries Overview



Schema of tasks (see annex 1)

3.2 Encoder Library

This library defines an interface for an encoder device. Since the encoder devices might be from different origins, the main goal of this library is to convert the data delivered by the encoder hardware into a VAL3 numerical variable. These variables will be accessed later by the [Conveyor Library](#).

VALTrack contains several encoder libraries that manage an encoder device connected either to a STARC2 board (CS8C) or to a Profibus board through a module WAGO 750-333 (CS8):

- *D_ABZ_00*: Manages an encoder connected to the first encoder input of the first DUAL ABZ board installed on a CS8C
- *D_ABZ_01*: Manages an encoder connected to the second encoder input of the first DUAL ABZ board installed on a CS8C
- *WAGO750_631*: Manages an encoder connected to a WAGO module 750_631. The module WAGO is read by the controller CS8 through a Profibus board installed on the controller.

In the case of an application with more than one conveyor, don't use the same Encoder Library for two Conveyor Libraries working at the same time.

It is also possible to write your own encoder library to adapt it to another encoder device. To that end, Stäubli provides a VAL3 template called GenericEnc with the following interface (public program and variables):

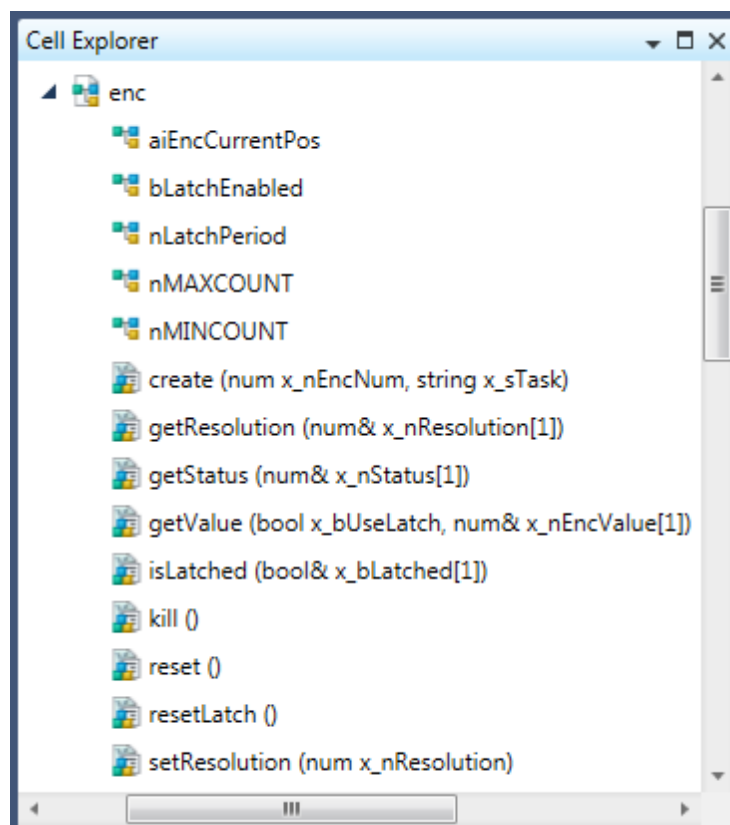


Figure 3.1

You will find more information about the structure and use of the encoder library in [chapter 5.1](#) of this document.

3.3 Detection Library

This library defines an interface for a detection device. The goal of this library is to identify the number of objects that have been simultaneously recognized, their type and their position on the conveyor. These three variables will be accessed later by the application programs through the [Motion Library](#).

VALTrack contains several detection libraries that can be used depending on the detection device used to detect the parts:

- *Cognex1* and *Cognex2*:
 - Used to read the parts detected by a Cognex camera using the Insight® Native Mode via a TCP/IP connection.
 - These two libraries are identical, *Cognex1* is meant to be used for the 1st conveyor and *Cognex2* is meant to be used for the 2nd conveyor.
 - The number on the name of the library is used as a reference to select the socket (client) used to read the information from the camera, thus the library *Cognex1* uses the socket *cognex1* and the library *Cognex2* uses the socket *cognex2*. If a 3rd or 4th conveyor is used, these libraries can be duplicated into *Cognex3* or *Cognex4*, these *Cognex3* or *Cognex4* libraries will use the socket *cognex3* and *cognex4* respectively.
- *Sensor1* and *Sensor2*:
 - Used to create parts detected with a simple on/off sensor.
 - These two libraries are identical, *Sensor1* is meant to be used for the 1st conveyor and *Sensor2* is meant to be used for the 2nd conveyor. If a 3rd or 4th conveyor is used, these libraries can be duplicated into *Sensor3* or *Sensor4*
- *Socket1* and *Socket2*:
 - Used to reads the parts detected by another Stäubli robot placed upstream over the same conveyor.
 - These two libraries are identical, *Socket1* is meant to be used for the 1st conveyor and *Socket2* is meant to be used for the 2nd conveyor.
 - The number on the name of the library is used as a reference to select the socket (server) used to receive the information from the other robot, thus the library *Socket1* uses the socket *fromRobot_sk1* and the library *Socket2* uses the socket *fromRobot_sk2*. If a 3rd or 4th conveyor is used, these libraries can be duplicated into *Socket3* or *Socket4*, these *Socket3* or *Socket4* libraries will use the socket *fromRobot_sk3* and *fromRobot_sk4* respectively.
 - These detections library can also be used to receive the information of the detected parts from any other CPU as long as this CPU sends the information via a TCP/IP socket respecting the format of the telegram.
 - They can be used as a reference to create a new detection library to work with any detection device sending the information of the detected objects by a TCP/IP socket. To that end, save the library with a name other than *Socket*, erease the instruction *sioLink()* on the socket section of the program *init()* and link the variable *skFromRobot* to the socket of your choise, modify the program *readSocket()* to adapt it to the format of the telegram send by the detection device, for example the encoder value should not be send by the detection device, and the handshake character \$ can be ereased if the detection device is not able to handle a comunication handshake.

The next table shows the summary of the sockets used by each detection library.

Library Name	Socket Name	Socket Type
Cognex1	cognex1	Client
Cognex2	cognex2	Client
Sensor1	---	---
Sensor2	---	---
Socket1	fromRobot_sk1	Server
Socket2	fromRobot_sk2	Server

Table 3.1

In the case of an application with more than one conveyor, don't use the same Detection Library for two Conveyor Libraries working at the same time.

It is also possible to write your own Detection Library to adapt another detection device. To that end, Stäubli provides a VAL template called *GenericDetect* with the following interface (public program and variables):

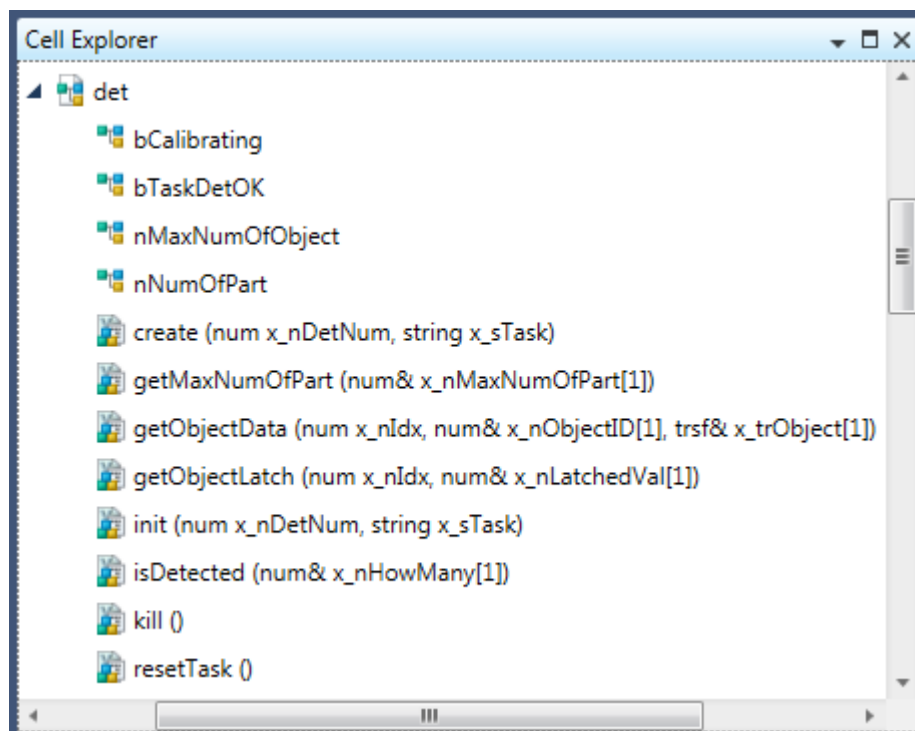


Figure 3.2

You will find more information about the structure and use of the detection library in [chapter 4.1](#) of this document.

3.4 Conveyor Library

This library defines the interface for a conveyor device. A Conveyor Library contains a link to one [Encoder Library](#) and to one [Detection Library](#) as shown in the figure 3.3, and features the following functionalities:

- Management of a queue of objects containing the type of the object (ID), the latched encoder position and the position within the conveyor for each object.
- The Conveyor Library is able to generate a periodic trigger signal on a selected digital output. This is a square wave signal which period corresponds to a displacement of the conveyor. For instance, it is possible to configure the library to have a rising edge on the selected output each time the conveyor travels 200mm. This feature is mainly used to trigger the image acquisition of a vision system at a fix interval.
- When the detection device is a vision system, the objects on the conveyor are moving across the field of view of the camera. If the camera's field of view is wider than the trigger's period, it is possible that one object is detected by the vision system in two consecutive pictures. The Conveyor Library compares the newly detected objects with the already queued objects from the previous image to avoid queuing two times the same object. The distance to evaluate if the already queued objects belong to the previous image is the trigger period plus the match tolerance ([MatchTol](#) + [ClkSignal](#))

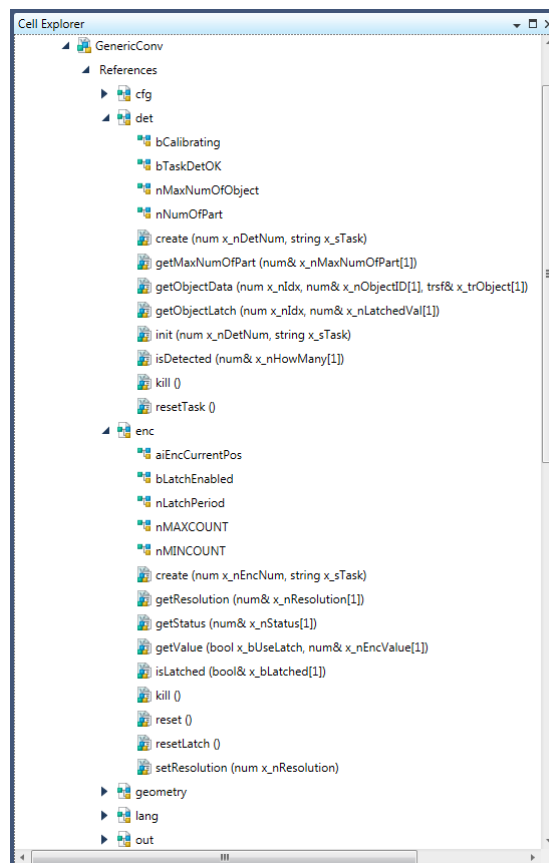


Figure 3.3

You can use the [Calibration Wizard](#) to create a conveyor library and also to define the properties of the conveyor. One conveyor library must be created for each conveyor connected to the controller.

You will find more information concerning the structure and use of the Conveyor Library in [chapter 5.2](#) of this document.

3.5 Motion Library

This library defines an interface to command robot movements synchronized with the conveyor. It must be declared in the main VAL3 project as shown in Figure 3.4.

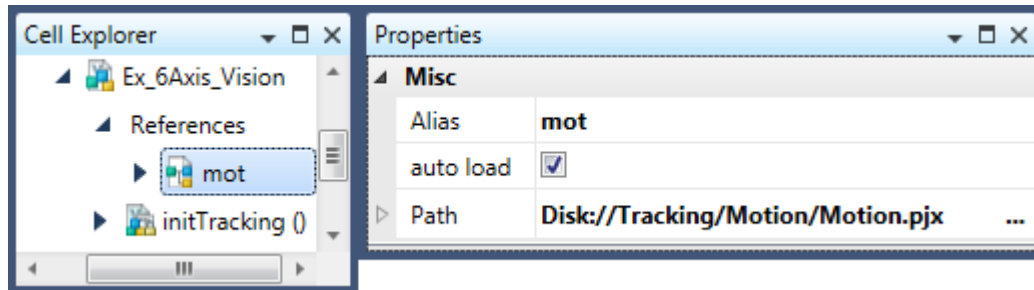


Figure 3.4

You will find more information concerning the structure and use of the Motion Library in [chapter 4.4](#) of this document.

3.6 Calibration Wizard

This is a standalone application that guides the user to define the following properties of the conveyor:

- The position and orientation of the conveyor in the reference frame of the robot. The orientation is used to define the direction in which the conveyor is moving.
- The scale factor between millimeters on the surface of the conveyor and the encoder counts (mm/count or mm/deg in the case of the DUAL ABZ board).
- The limits of the area in which the robot is allowed to work on the conveyor (Tracking area).
- The position of the detection device in the reference frame of the robot.

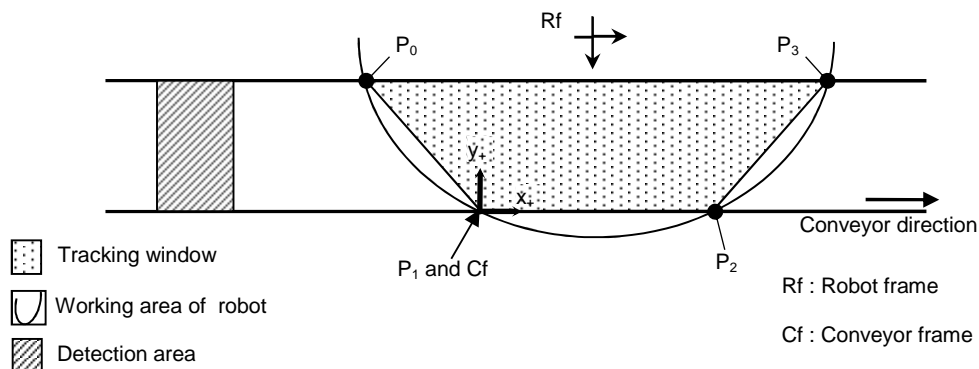


Figure 3.5

The use of the Wizard is explained step by step on [chapter 4.2](#).

4 PROGRAMMING STEPS

4.1 Create a Library for your Detection Device

Three type of detection libraries are included in VALTrack package, one type to work with a simple on/off sensor device, one type to work with a Cognex camera, and one more to be used when the detection device is another Stäubli robot placed upstream over the same conveyor. Most of the times, it is possible to use one of these detection libraries and make few changes to adapt it to a specific detection device. If none of those three types of libraries can be adapted to a detection device, a new detection library must be created starting from the detection library template.

4.1.1 Programming the detection library

- Create a new VAL3 application

If none of the already existing detection libraries is suitable for your detection device, a new detection library should be created. The new detection library must have the same interface (public programs and variables) as the default detection library, therefore the new detection library should be a copy of the *GenericDetect* library.

1. Open the library *GenericDetect* under the directory Tracking/Templates/.
2. Save it with a different name under the directory Tracking/Detections/.

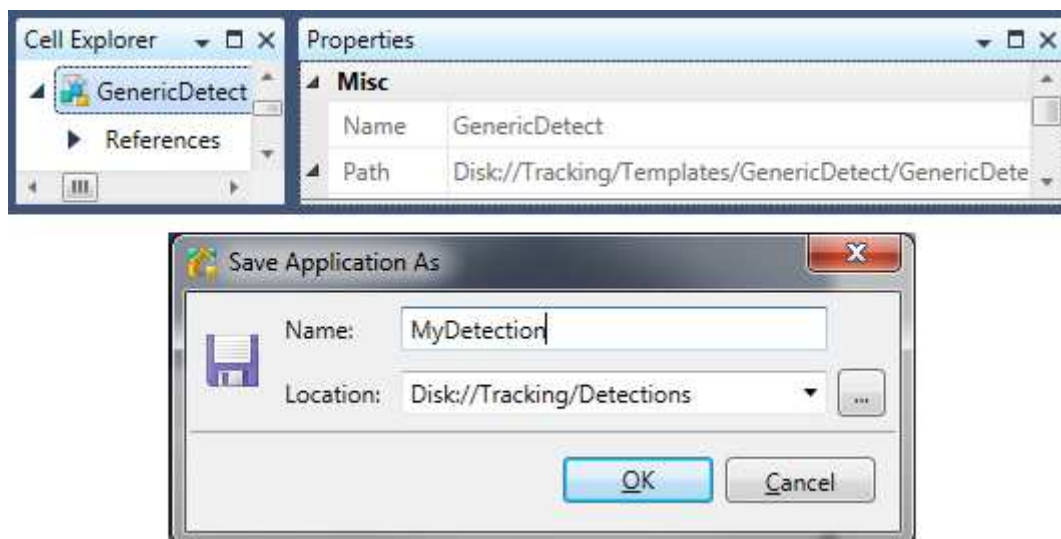


Figure 4.1

3. Modify the program *detection* according to the needs of the new detection device.

Note: Do not modify the library Tracking/Templates/GenericDetect. Any modification to the interface (names and number of public programs and variables) of this library will have to be copied to all detection libraries.

- Sequence to be follow by the detection library

1. Read the number of detected objects, the position and ID (identification number) of each object, from the detection device:

The program *detection()* is the program executed by the detection task. This program must define the information for each detected object, either the data is defined on the program or read from the detection device. The information needed is the number of objects detected on a single trigger, the ID of each detected object, and the position on the conveyor of each detected object. Place the ID and the position of the detected object(s) respectively into the variables *nObjectID[]* and *trObjectPos[]*. It is possible to sort the detected objects, for example by ID or by increasing X, reject object with negative Y etc...Use an internal counter variable, like *I_nNumOfPart*, to count the number of good objects.

2. Allow the conveyor library to read the object's data from your detection library:

Once all object's IDs and positions are written into the variables *nObjectID[]* and *trObjectPos[]*, copy the value of the internal counter to the global variable *nNumOfPart*. Once this variable is different than -1, the objects will be pushed into the conveyor's queue.

3. Wait until the conveyor library has pushed all objects into its queue:

Once the Conveyor Library has read the data of the objects, it sets the variable *nNumOfPart* back to -1.

- Save your library

You can find hereafter the main loop of the program provided in the template for the detection library:

```

36 do
37     // wait until camera has finished image treatment
38     call WaitUntilCameraReady()
39     //
40     // Read how many object have been detected.
41     call myRoutineToReadNumberOfDetectedObject(l_nObjDetected)
42     //
43     l_nNumOfPart=0
44     for l_i=0 to l_nObjDetected-1
45         // Read data from your detection device:
46         call myRoutineToReadObjectData (nObjectID[l_nNumOfPart],trObjectPos[l_nNumOfPart])
47         //
48         // increment l_nNumOfPart only if you want to bufferize the object.
49         // For instance, you bufferize object only with:
50         //     - ID=1
51         //     - positive Y coordinate
52         //
53         if (nObjectID[l_nNumOfPart]==1) and (trObjectPos[l_nNumOfPart].y>0)
54             l_nNumOfPart=l_nNumOfPart+1
55         endIf
56     endFor
57     //
58     // Set number of detected parts.
59     // nNumOfPart is watched by the conveyor library.
60     // Once nNumOfPart is set, the conveyor library will access
61     // the variables nObjectID[] and trObjectPos[].
62     // They must be fully updated before nNumOfPart is set.
63     nNumOfPart=l_nNumOfPart
64     //
65     // Once data have been read by conveyor library
66     // nNumOfPart is set to value -1
67     wait (nNumOfPart===-1)
68     //
69     delay(0)
70 until false

```

Figure 4.2



Warning

The program is an example. Stäubli can help the programmer to adapt the program, but the programmer is responsible.

4.1.2 Variable description

- public num *nMaxNumOfObject*

This variable is used to fix the maximum number of objects that can be detected in a single trigger. This variable is used to resize the variables described hereafter. This is the maximum number of objects that can be pushed into the buffer of the [Conveyor library](#) at the same time. The variable is modified by the Wizard application, it corresponds to the parameter *NbObject* on the Settings page for the Detection Library (Figure 4.11).

- public num *nNumOfPart*

Numerical value representing the number of objects detected. Typical value is 1 when you are using an on/off sensor as detection device. It should not be bigger than *nMaxNumOfObject*

- private num *nObjectID[]*

Numerical array -starting with item 0- containing the ID of each object detected. The ID must be a value greater than or equal to 0. The size of the array is given by the variable *nMaxNumOfObject* mentioned previously.

- private trsf *trObjectPos[]*

Array of transformation values -starting with item 0- containing the position of each object detected. The size of the array is given by the variable *nMaxNumOfObject* mentioned previously.

- private num *nLatchedValue[]*

Numerical array -starting with item 0- containing the latched encoder value of each object detected. The size of the array is given by the variable *nMaxNumOfObject* mentioned previously. This variable is used only by the detection library *Socket*. For any other detection library, the latched encoder value is given by the encoder library.

4.1.3 Programs description

- public *create*(num *x_nDetNum*, string *x_sTask*)

This program initializes the variables of the library and launches the program *detection* in a task called *detectX* where X is the conveyor number to which the detection library belongs to. It is called once by the [Conveyor Library](#).

- private *detection*()

This is the main program of the library. It is launched by the [Conveyor Library](#) in a VAL3 task called *detectX* where X is the conveyor number to which the detection library belongs to. It manages the communication with the detection device and update the variables [nObjectID\[\]](#), [trObjectPos\[\]](#), and [nNumOfPart](#) (and [nLatchedValue\[\]](#) in the case of the detection library *Socket*)

- public *getMaxNumOfPart*(num& *x_nMaxNumOfPart*)

This program returns the maximum number of objects that can be detected in a single latch. It is used by the [Conveyor Library](#) to resize internal variables.

- public *getObjectData*(num *x_nIdx*, num& *x_nObjectID*, trsf& *x_trObjectPos*)

This program returns the ID and the position of the *x_nIdx*th detected object. This program is called by the [Conveyor library](#).

- public *getObjectLatch*(num *x_nIdx*, num& *x_nLatchedVal*)

This program returns the encoder latch value of the detected object. This is useful only when the encoder latched value is given by the detection library and not by the encoder library, like in the case of the detection library *Socket*. This program is called by the [Conveyor library](#).

- public *init*(num *x_nDetNum*, string *x_sTask*)

This is the initialization program of the library. It is used to set the maximum number of objects that can be detected in a single latch by the detection device. It is called once when the library is started. *x_nDetNum* is the number of the conveyor using this detection library and *x_sTask+x_nDetNum* is the name of the task running the detection program.

- public *isDetected*(num& *x_nHowMany*)

This program returns the number of objects detected in a single latch. This program is called by the [Conveyor library](#).

- public *kill*()

This program terminates the task detect. This program is called by the [Conveyor library](#) to stop all the task of the conveyor.

- public *resetTask*()

This program terminates and re-launches the task detect without doing an initialization of variables. This program is called by the [Conveyor library](#) when the detection library *Socket* has to change from standard working mode to calibration mode.

4.2 Create a Conveyor Library

The following items are required to define the properties of a conveyor:

- A calibrated pointer mounted on the robot's flange (figure 4.3). The dimensions of the pointer must be entered in the menu Settings (F7) of the calibration wizard (figure 4.8).
- A calibration target that can be pointed to with the calibration pointer (figure 4.3). If you are using a vision system or equivalent: 1) make sure that the calibration target used to calibrate the vision system can be pointed to by the robot equipped with its calibrated pointer and 2) the calibration target is at the same high as the picking high of the parts.
- The encoder must be mounted on the conveyor and wired properly. VALTrack needs to see the encoder value increase when the conveyor is moving forward. If the encoder value is decreasing when the conveyor is moving forward, it is possible to change the direction of the encoder count by electrically swap the pair A and Inversed A with the pair B and Inversed B.

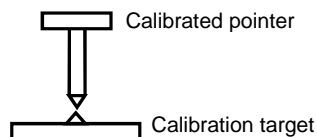


Figure 4.3

From the Application Manager, load and execute the application called Wizard.

4.2.1 Select a conveyor library to work with

The software displays the list of the libraries that have been already created (figure 4.4). You can scroll up and down with the arrow keys and press F4 or RETURN to select an existing library. You can also press F2 to create a NEW conveyor library, and type in its name.

Press F4 to select an existing library:

Press F2 to create a new conveyor library:

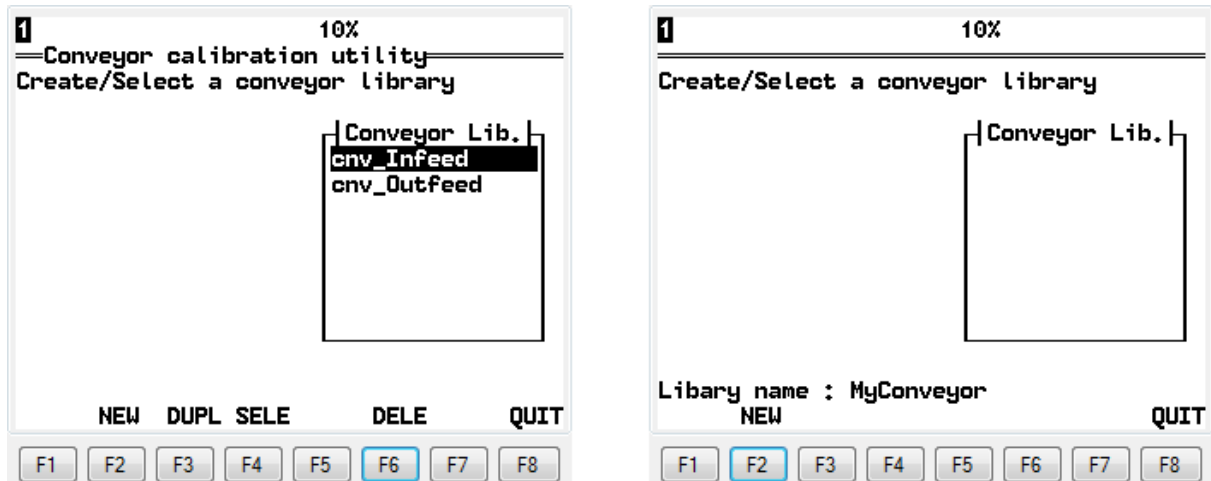


Figure 4.4

When creating a new conveyor library, its encoder and detection libraries must be defined.

4.2.2 Select an encoder library

The software displays the list of the available encoder libraries (figure 4.5). Each time you are creating a new conveyor library, you must select one encoder library to work with. Navigate in the list and press F4 or RETURN to select an encoder library. (Select the library that corresponds to your hardware: *D_ABZ_00* for channel 0, or *D_ABZ_01* for channel 1)

Press F4 or RETURN to select an encoder library:

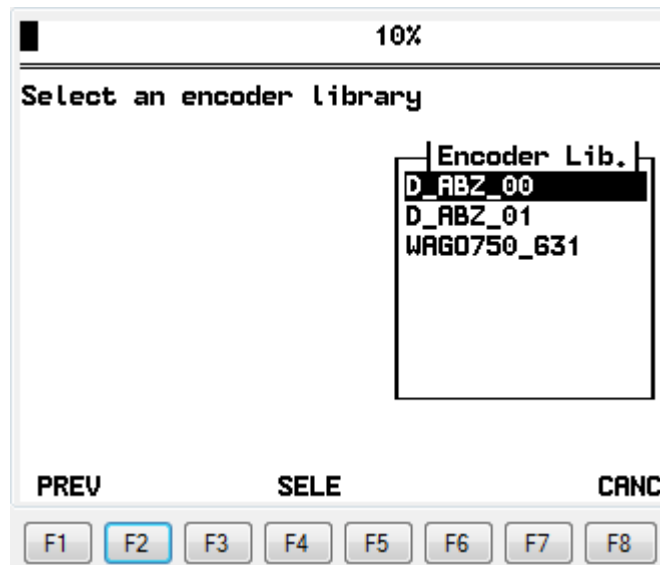


Figure 4.5

4.2.3 Select a detection library

The software displays the list of the available detection libraries (figure 4.6). Each time you are creating a new conveyor library, you must select one detection library to work with. Navigate in the list and press F4 or RETURN to select a detection library.

Press F4 or RETURN to select a detection library:

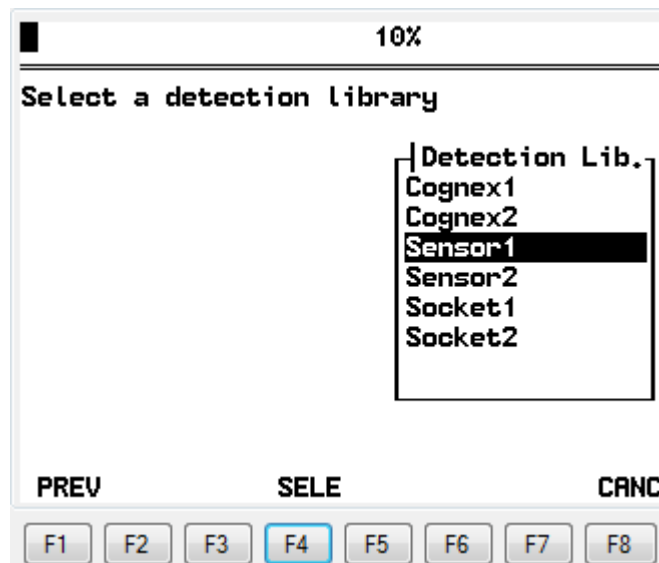


Figure 4.6

4.2.4 Setup the conveyor library

At this step, the conveyor library is created. Before proceeding with the teaching of the conveyor properties, you must setup different parameters. From the main menu (figure 4.7), press F7 to jump into the *Settings* menu. This menu is divided in 4 different pages. You can navigate between these pages by pressing the function keys:

- F2 for general parameters
- F4 for the parameters related to the encoder library
- F5 for the parameters related to the detection library
- F6 for the parameters related to the conveyor library.

Within any given page, you will use the following keys to navigate between the fields:

- Arrow UP : jump to previous field
- Arrow DOWN : jump to next field
- RETURN : Edit focused field
- ESCAPE : Leave the edited field without change or leave the current page and cancel any change on the various fields.
- BACKSPACE : During field edition, delete a character placed at the left of the cursor.

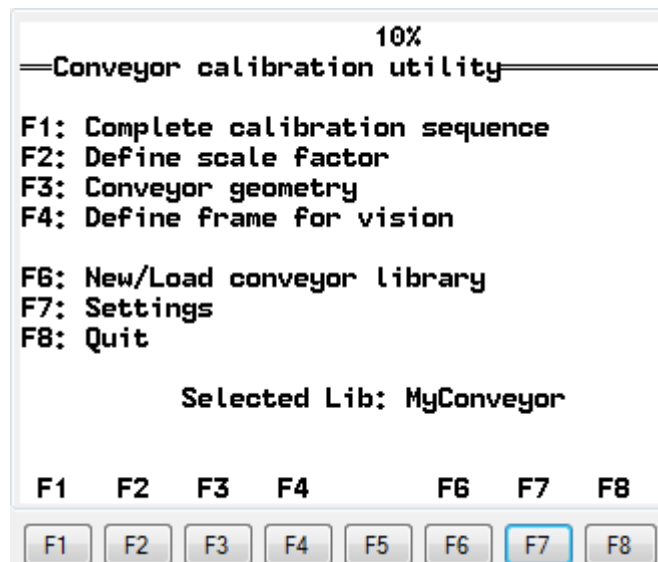


Figure 4.7

- General setup page

By pressing F2 you will access the page to setup general parameter as detailed below:

Calibrated pointer:

Enter the coordinates of the calibrated pointer used for teaching the locations in the various procedures.

Software Version:

This information should be provided for any contact to a Stäubli organization. This field is read only.

Event logger:

When this value is different to zero, debug messages are sent through the TCP/IP telnet port (default is 23) of the controller. You can read them by establishing a telnet connection to the controller. This integer value is interpreted as a bit field as detailed below:

10%

==Setup general parameter==

-Calibrated pointer-- -Software --

X= 0 Rx= 0 Version: s7.4.1

Y= 0 Ry= 0

Z= 0 Rz= 0 -Event Logger --

Enable Log: 0

-Approach --

X= 0

Y= 0

Z= -100

ENCO DETE CONV OK

F1 F2 F3 F4 F5 F6 F7 F8

Figure 4.8

Bit 1 (mask value =1)

When this bit is set, messages concerning the object's detection are sent through the Telnet Port.

Bit 2 (mask value =2)

When this bit is set, messages related to the motion of the robot while tracking the conveyor are sent through the telnet port.

Bit 3 (mask value =4)

When this bit is set, the queue is displayed on the telnet connection when it changes (add or remove parts).



Warning

If the Event logger parameter has a value greater than zero, a telnet client MUST be connected to the controller. Do not leave the Event Logger parameter set to any value other than zero if no one is debugging the installation, telnet messages use a lot of CPU time.

Approach:

During the various procedures for defining the properties of the conveyor, it is possible to move the robot to the locations that have been already taught. Instead of directly move to a taught location, the robot will depart from its current position and approach the destination by a transformation defined by these 3 fields.

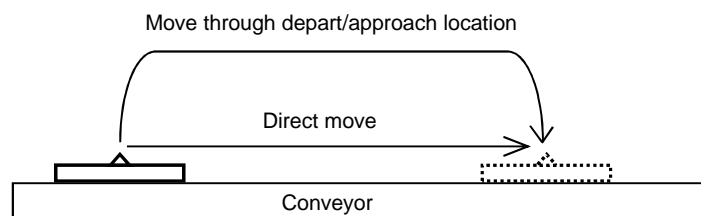


Figure 4.9

- Encoder setup page

By pressing F4 you will access the page to setup the parameters related to the encoder device as detailed below:

LatchPeriod:

Number of triggers that must occur on the position capture input before an object is considered as detected. See [chapter 5.1.1](#)

Resolution:

You can enter here the resolution of the encoder mounted on the conveyor. It is the number of pulses per encoder revolution.

Filter Length:

This parameter allows filtering the encoder input to guaranty smooth motions of the robot. The value represents the number of samples used by the filtering algorithm. The sampling rate of the encoder value is 250Hz (500Hz for the robot TP80)

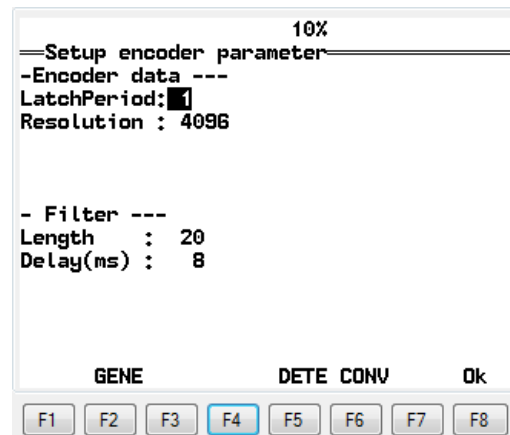


Figure 4.10

Filter Delay:

Delay of the system to read the current encoder value. It is the time that takes to the encoder value to pass throught the different hardware and software layers. Proportional to the sampling rate of the encoder value. This field is read only.

- Detection setup page

By pressing F5 you will access the page to setup parameters related to the detection device as detailed below:

NbObject:

Maximum number of objects detected on the same trigger. In the case of a vision camera it is the maximum number of objects to be found in one image.

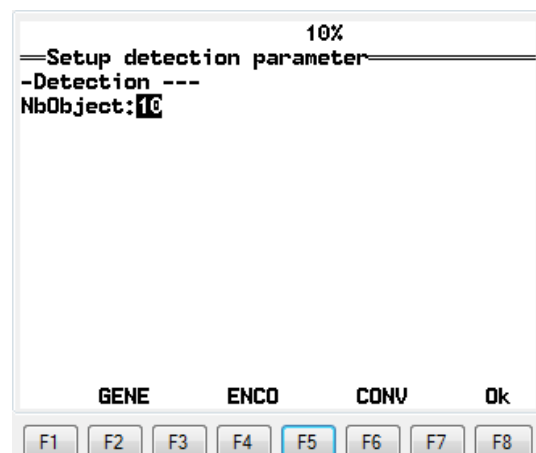


Figure 4.11

- Conveyor setup page

By pressing F6 you will access the page to setup parameters related to the conveyor as detailed below:

EncoderLib:

Name of the encoder library associated to the conveyor. When this field is focused, pressing ENTER allows the selection of an encoder library from a list.

DetectionLib:

Name of the detection library associated to the conveyor. When this field is focused, pressing ENTER allows the selection of a detection library from a list.

BufferSize:

Size of the conveyor's queue. It is the maximum number of objects that can be queued. Physically this represents the number of objects traveling on the conveyor between the area where they are detected and the tracking window of the robot.

Caution: If the queue is full and new objects are detected, the new objects will be lost. The message "CNV(fillQueue)>>Buffer overflow, object(s) might have been missed" is sent through the telnet connection. Increase the BufferSize value.

ScaleFact:

It is the ratio between the conveyor traveled distance and the encoder counts (mm/counts). It is computed during the procedures "Define scale factor" and "Complete calibration sequence" available from the main menu.

MatchTol:

Tolerance used to avoid placing several times the same object in the conveyor's queue. Two objects are considered to be the same when the distance in between them is smaller than *MatchTol*. Only the already queued object is kept, the new object is rejected. See [chapter 3.4](#) for further information.

ClkSignal:

Name of the output on which a square wave signal is generated. The conveyor library is able to generate a periodic trigger signal on the selected digital output. The period of this square wave signal corresponds to a displacement of the conveyor (see *ClkPeriod* field). For example, the library can be configured to have a rising edge on the selected output each time the conveyor travels 200mm. This feature is mainly used to trigger the image acquisition of a vision system at a fix interval. The trigger signal remains ON for half of the trigger period (*ClkPeriod*) but never more than 0.1sec.

ClkPeriod:

Distance in millimeter that the conveyor must travel between two rising edges of the trigger signal.

LockDist:

It is the distance (in millimeters) that the conveyor must travel before a second trigger is set and/or acknowledged. This is especially useful to filter rebounds on the trigger signal to latch the encoder position. It is also useful when the detection device is a on/off sensor and the shape of the detected object generates several triggers, in this case it is possible to go around this problem by entering a value that is larger than the length of the object.

Note: *ClkSignal* and *ClkPeriod* (trigger signal and trigger period) are used when the detection device is a vision system and *Lockdistance* is used when the detection device is an on/off sensor to avoid rebounds in the input trigger signal.

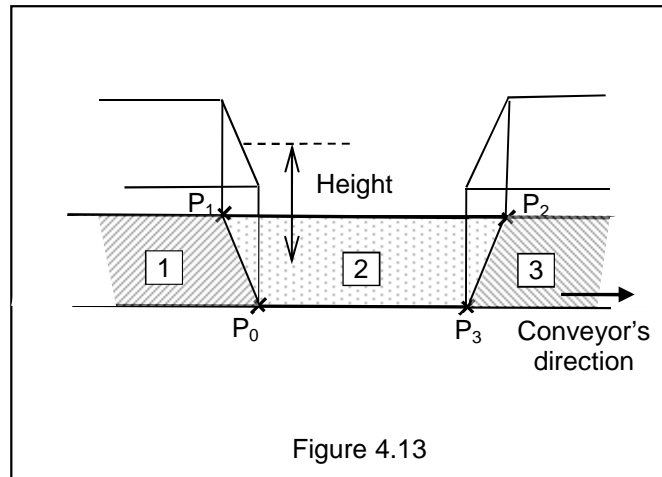
Figure 4.12

Height:

The tracking window for the robot is on top of the conveyor and it is considered as a volume. Labeled 2 on figure 4.13, the volume is defined by P0, P1, P2, P3 and the tracking window *Height*. Value expressed in mm.

ThruUpLim:

Name of the output signal that is set when the robot's TCP is inside the volume located upstream of the tracking window. Labeled 1 on figure 4.13, the volume is defined by P0, P1 and the tracking window *Height*.



ThruDwnLim:

Name of the output signal that is set when the robot's TCP is inside the area located downstream of the tracking window. Labeled 3 on figure 4.13, the volume is defined by P2, P3 and the tracking window *Height*.

Enable Log:

Enable/disable Detection and Queue telnet messages individually per conveyor. This is useful when the application has multiple conveyors. In the case of multiple conveyors, the type of messages to log for all conveyors is selected on the page "Setup general parameters", then on the page "Setup conveyor parameters", the messages of each conveyor are enable/disable so only the messages of one conveyor are logged at the same time.

4.2.5 Teach the conveyor library

Teaching the conveyor library means defining the environment of the robot. In the Wizard's main menu we will find almost all the tools to define the environment of the robot. The definition of the parameter that is not implemented on the Wizard is the distance that the conveyor has to travel from the moment an object is been detected, by a detection device such a on/off sensor, to the moment the object arrives to the tracking window of the robot. This parameter is defined at the same time while defining the picking position of each object in the main application, but remember this is only in the case of a detection device such an on/off sensor.

For all tracking applications, the robot needs to know: 1) where is the conveyor (F3), 2) the scale factor between millimeters on the conveyor and encoder counts (F2), and 3) the distance in between the detection device and the tracking window of the robot (F4 for vision, by program for an on/off sensor).

- F1: Complete calibration sequence

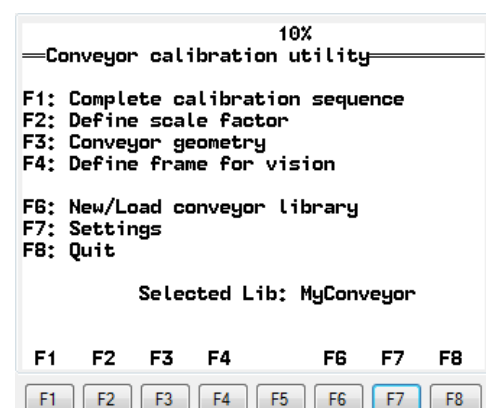
This procedure will guide the user through the steps to define the parameters of a conveyor. This menu does the same as menus F2 and F3, all at the same time.

- F2: Define Scale factor

This procedure is used to define the mechanical scale factor between the millimeter on the surface of the conveyor and the encoder counts.

- F3: Conveyor geometry

This procedure defines the position and orientation of the conveyor in the reference frame of the robot as well as the limits of the area in which the robot is able to track the



detected objects (tracking window).

- F4: Define frame for vision

This procedure allows teaching the position and orientation of the vision system in the reference frame of the robot.

- F6: New/Load conveyor library

This menu allows to create a new conveyor library or to load an existing one from the flashDisk.

Once all the above mentioned steps have been performed, the conveyor library is ready to be used and it is stored at the folder **Tracking\Conveyors**.

4.3 Create your Main Application

VALTrack is not a stand-alone application. VALTrack is a group of programs meant to be integrated as libraries of your main application. In order to help you building the structure (libraries and data) of your main application, some examples are provided with the software, the differences in between the examples are the type of robot used, the type of detection device used, the number of conveyors used. Open the application example that fits better to your installation and save it with a different name.

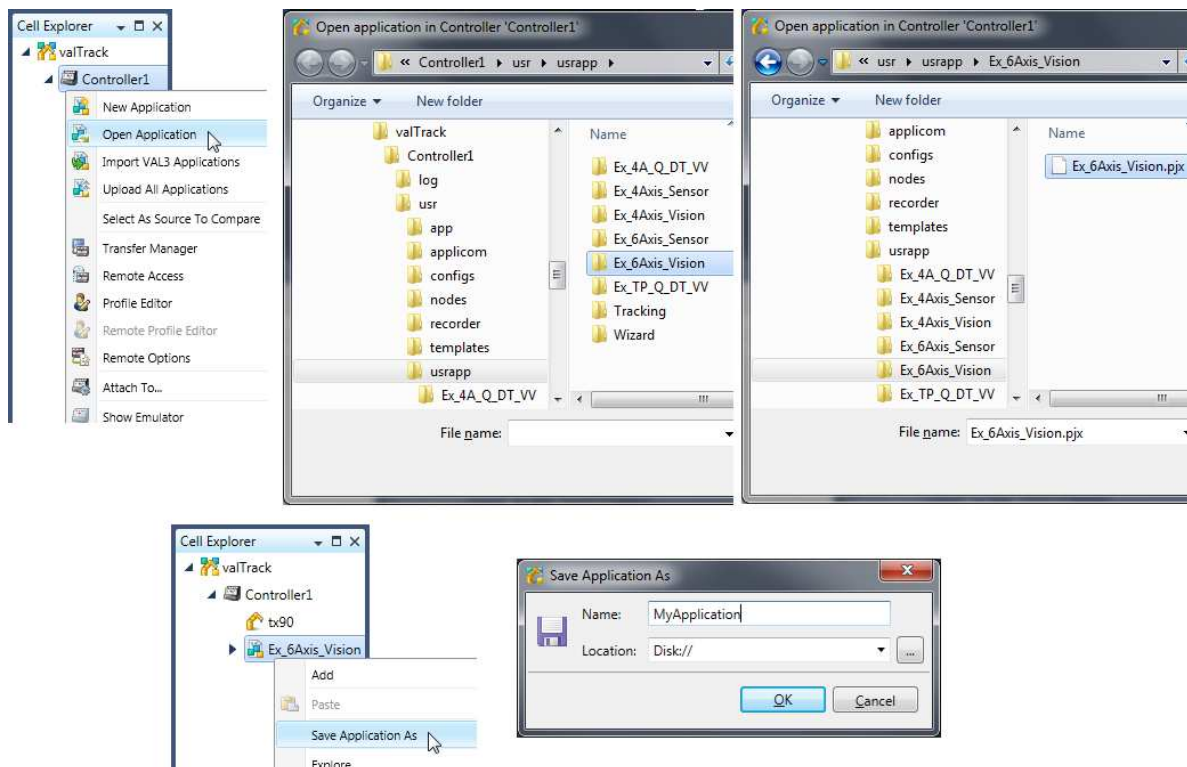


Figure 4.15

4.3.1 Tracking libraries

All tracking applications are located under the directory Tracking. The application Motion is the interface in between VALTrack and the main application. The application Motion must be declared as a library of your application.

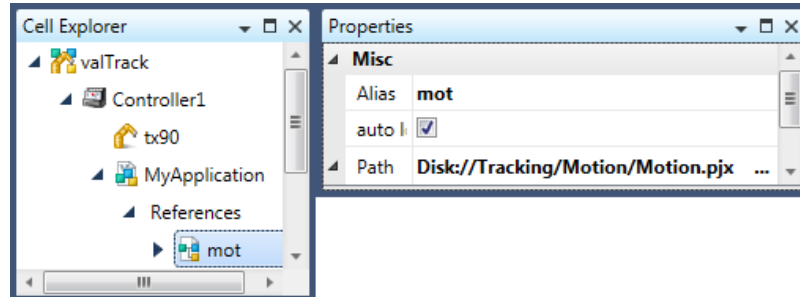


Figure 4.16

4.3.2 Program example for single tracking

An application where the robot works with only one conveyor it is called a Single Tracking Application. Four applications example are provided with VALTrack where the robot picks a moving part on a conveyor and places it on a fix position:

- *Ex_4axis_Sensor*: Example for a 4 axes robot. The detection device is a simple on/off sensor.
- *Ex_4axis_Vision*: Example for a 4 axes robot. The detection device is a complex sensor such as vision camera.
- *Ex_6axis_Sensor*: Example for a 6 axes robot. The detection device is a simple on/off sensor.
- *Ex_6axis_Vision*: Example for a 6 axes robot. The detection device is a complex sensor such as vision camera.

4.3.2.1 Used variables

- private tool *tGripper*

Definition of your tool. Distance from the center of the robot flange to the Tool Center Point.

- private joint *jHome*

Joint position used as a start position.

- private mdesc *mNomSpeed*

Motion descriptor containing standard speed/acceleration values for non-tracking movements.

- private mdesc *mTracking*

Motion descriptor containing standard speed and acceleration values for synchronized tracking movements. For movements when the robot is already synchronized with the conveyor.

- private mdesc *mTrackOnOff*

Motion descriptor containing standard speed and acceleration values for synchronization/de-synchronization tracking movements. For movements when the robot is moving from a fix point to a moving point over the conveyor or from a moving point to a fix point.

- private num *nWaitDistToUp*

Variable used to define the distance from the UPSTREAM limit from which the robot can start moving to the objects. A negative value defines a position upstream of the UPSTREAM limit, a positive value defines a position downstream of the UPSTREAM limit. Zero is a typical value for this variable, so the robot will go to the object as soon as the object has cross the UPSTREAM limit.

- private num *nWaitDistToDown*

Variable used to define the distance from the DOWNSTREAM limit from which the robot should not move anymore to an object. A negative value defines a position upstream of the DOWNSTREAM limit, a positive value defines a position downstream of the DOWNSTREAM limit. A negative value is a typical value for this variable, the value has to be big enough so the robot has time to go the object and de-synchronize from the object before the object leaves the tracking window.

- private num *nID*

Variable the get from the tracking libraries the type of object detected by the detection device.

- private string *sConveyorList*

List of conveyor libraries used in the application.

- private frame *fConveyor*

Variable used to get from the tracking libraries the position of the conveyor's frame taught with the Wizard during the conveyor calibration.

- private frame *fDetection*

Variable use to get from the tracking libraries the position of the vision's frame taught with the Wizard during the vision frame calibration.

- private point *pPick*

Variable used to define the position of an object on the conveyor. When the detection device is a simple on/off sensor, *pPick* is defined under the frame *fConveyor* and it must be taught with the program *teachLoc(x_nConvNum)* provided with the example. When the detection device is a complex system capable of defining the full position of an object on the conveyor, such as a vision system, *pPick* is defined under the frame *fDetection* and its value is returned by the program *mot:getObjectData* from the motion library.

- private point *pPlace*

Fix position taught by the user, position where the robot will place the parts after picking them over the conveyor.

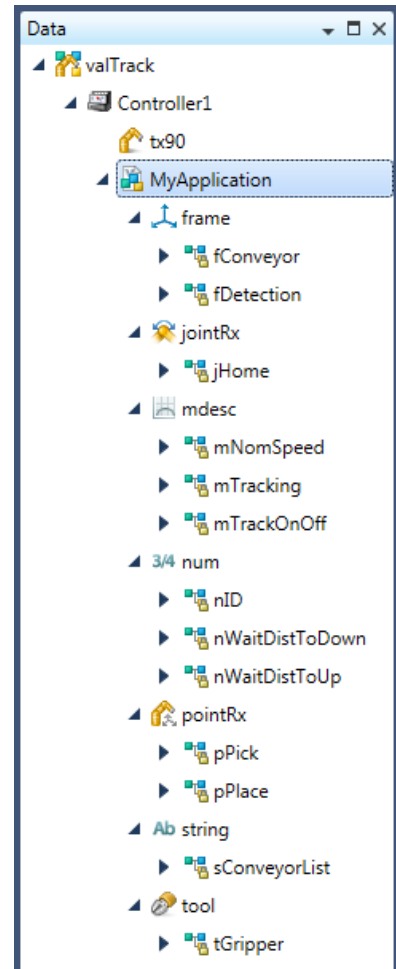


Figure 4.17

4.3.2.2 Select conveyor library

Select the variable *sConveyorList* and enter the name of the conveyor library created with the Wizard.

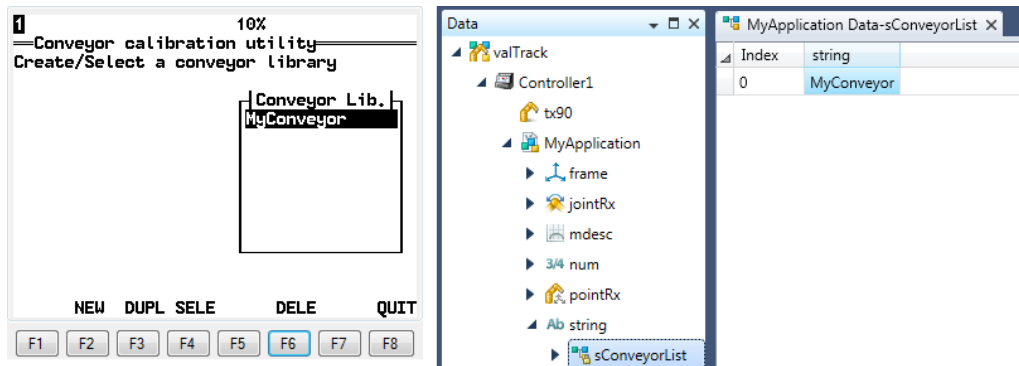


Figure 4.18

4.3.2.3 Teach positions

Define the tool and teach your points *jHome* and *pPlace*.

If the detection device is a simple on/off sensor, use the program *teachLoc(x_nConvNum)* provided with the example to teach *pPick*.

The program *teachLoc(x_nConvNum)* is used when the detection device, such as an on/off sensor, doesn't provide full information about the position of the object on the conveyor. All the information returned by this kind of detection device is that an object is present on the conveyor. Therefore, it is necessary to perform a specific procedure to define the coordinates of the point that the robot will have to track. On the contrary, if the detection device is able to provide the coordinates of the object on the conveyor (e.g. vision system), it is not necessary to perform any specific procedure. You must rather use the function *getObjectData()* to read these coordinates. For any further information, refer to [chapter 5.2](#)

4.3.2.4 Working tracking window

The tracking window is defined by the points P0, P1, P2 and P3 taught with the wizard, where P0 and P1 define the upstream limit, and P2 and P3 define the downstream limit. It is recommended to teach P0 through P3 at the maximum reach of the robot, therefore the tracking area covers the full reach of the arm.

Once the robot is on production, it is necessary to reduce the tracking window and work on a smaller working tracking window for two reasons:

1. It is not possible to send the robot to an object which is already at the limit of the downstream limit because the object will be out of reach before the robot arrives to the object. Therefore the limit from which the robot is no longer sent to an object is placed upstream from the downstream limit in order to have time to go to the object and de-synchronize from the object before the robot goes out of reach. Use the variable *nWaitDistToDown* to define the end of the working tracking window.
2. It is possible that the best cycle time is achieved when working only on a specific section of the tracking window. Use the variable *nWaitDistToUp* to define the beginning of the working tracking window and *nWaitDistToDown* to define the end of the working tracking window.

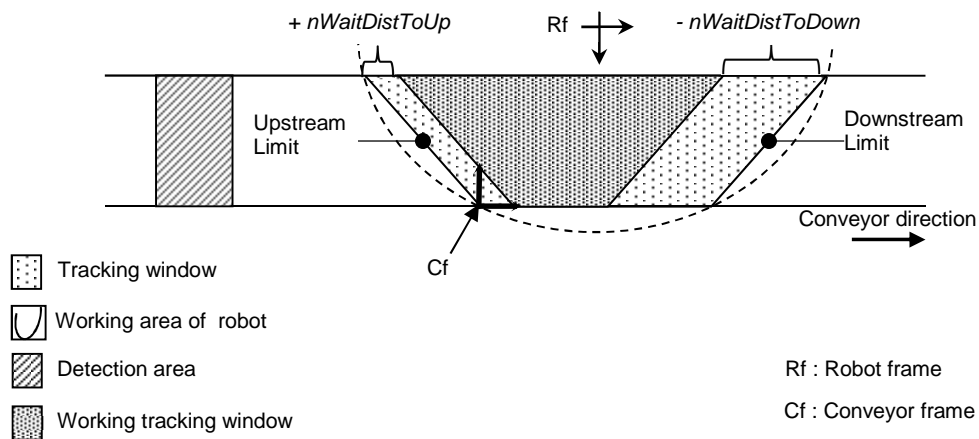


Figure 4.19

4.3.2.5 Cycle description

These examples applications, like any other application, start with the program *start()*. The first thing to do on the program *start()* is to launch all VALTrack tasks, each example has a program called *initTracking()* who does that. In the case of the examples made to work with a sensor as the detection device, the first time that the application is launched it will be to teach the *pPick* position, therefore the *start()* program looks as show on the next flowchart:

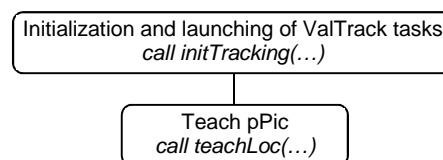


Figure 4.20

Once the all the teaching is finish, the programs *start()* will have the next flowchart, where the program *initTracking()* launches all VALTrack tasks and then the program *tracking()* executes the pick and place sequence.

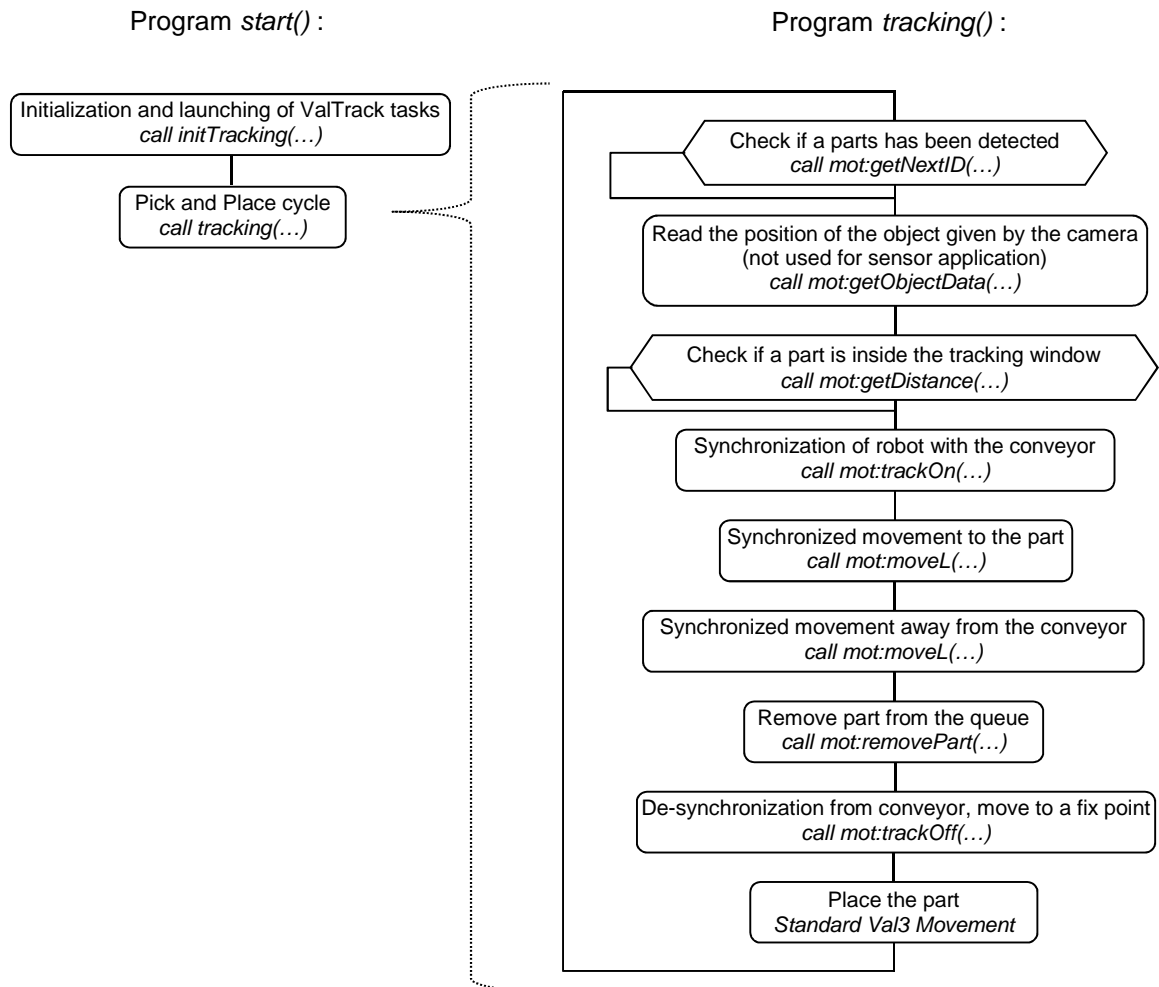


Figure 4.21

On the next figure we can find the program tracking() of the example *Ex_6Axis_Sensor*. All the programs with the prefix mot: are programs from the library Motion, for a full explanation on each one of these programs go to chapter 4.4

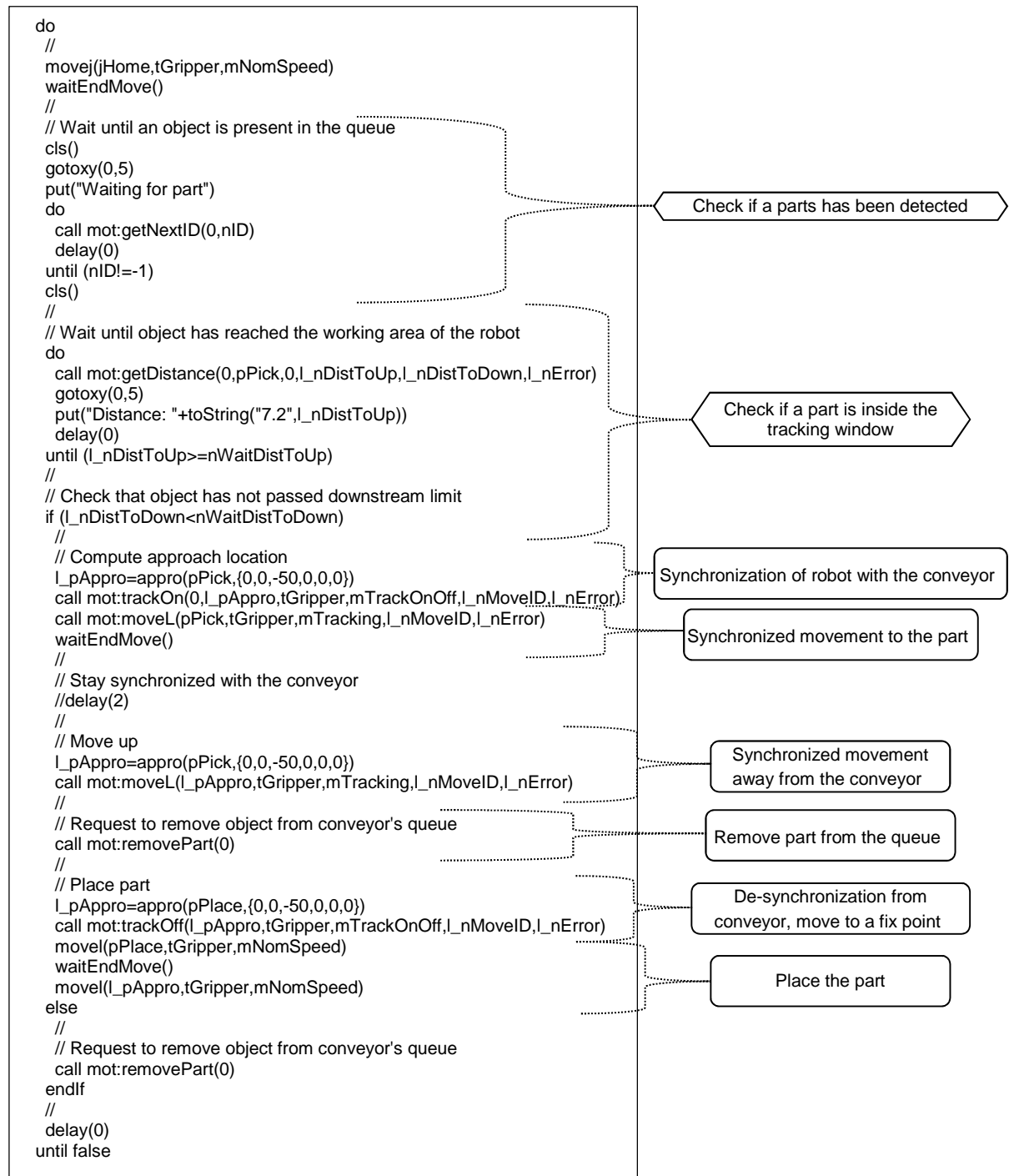


Figure 4.22

When the detection device is a camera, a call to the program *getObjectData()* must be used to recover the data of the point read from the camera. On the next figure, the call to the program *getObjectData()* is added to the program on the previous figure in between the loop where the program is waiting for an object to be detected and the loop where the program is waiting for the object to arrive into the tracking window.

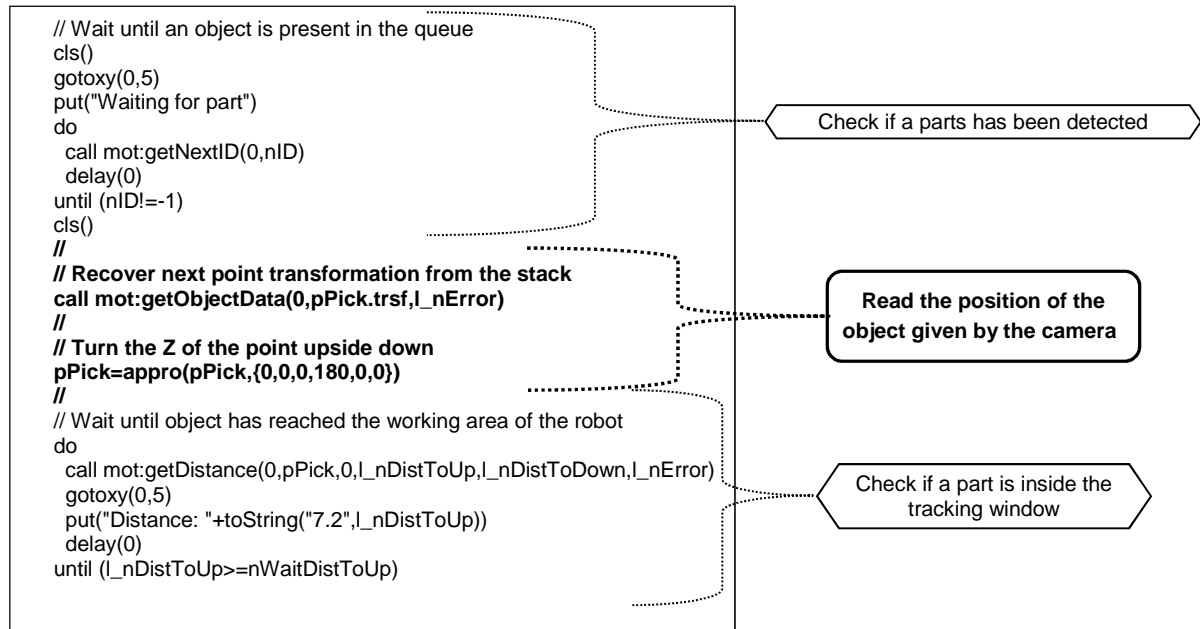


Figure 4.23

4.3.3 Program example for double tracking and queueing

An application where a robot works with two conveyors at the same time it is called a Double Tracking Application. An application where there are multiple robots and one detection device working over the same conveyor it is called a Queueing Application. Two applications example are provided with VALTrack where the robot picks a moving part on a infeed conveyor and it places it on a moving cavity on a outfeed conveyor:

- *Ex_4A_Q_DT_VV*: (Example_4Axis_Queueing_DoubleTracking_VisionVision). Example for a 4 axes robot. The detection device in both conveyors is a complex sensor such as vision camera. The robot can transfer not treated objects to the next robot placed downstream over the same conveyor. The robot can reduce the speed of the conveyor to avoid that an object goes out of reach of the robot, the speed is reduce gradually till a full stop of the conveyor. The robot can detect if it actually picked the part, if picks fails the robot picks again. The beginning of the working tracking window is defined as an offset from the upstream limit, and the end of the working tracking window is defined as an offset from the downstream limit.
- *Ex_TP_Q_DT_VV*: (Example_TP80_Queueing_DoubleTracking_VisionVision). This example is basicly the same application than the previous example. The only diferance is the definition of the working tracking window. Both, the beginning and the end of the working traking window are definied as an offset from the upstream limit.

Both examples have a small use interface to display some counters and the status of the cycle. The counters are:

- *Cycle Time (Last)*: Is it the cycle time of the last Pick&Place cycle executed by the robot. The value is in seconds and in Parts Per Minut.
- *Cycle Time (Average)*: Is it the average cycle time of the last 10 Pick&Place cycles executed by the robot. The value is in seconds and in Parts Per Minut.
- *Cycle Time (Best)*: Is it the cycle time of the fastest Pick&Place cycle executed by the robot. The value is in seconds and in Parts Per Minut.
- *Pick Done*: It is the number of parts picked by the robot.
- *Pick Transfer*: It is the number of parts not picked by the robot. The parts not picked by the robot can be simply removed from the conveyor's queue or transferred to the next robot before being removed from the queue.
- *Place Done*: It is the number of parts placed by the robot.
- *Place Transfer*: It is the number of cavities where the robot didn't place a part. The cavities not used by the robot can be simply removed from the conveyor's queue or transferred to the next robot before being removed from the queue.

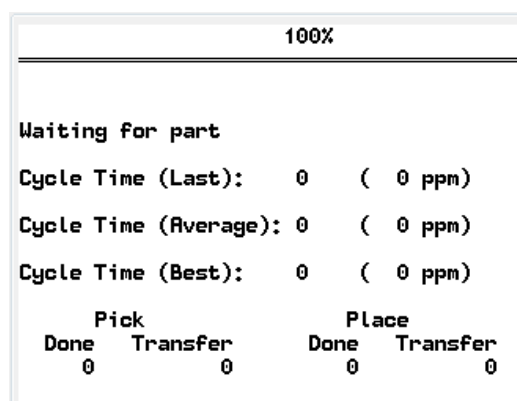


Figure 4.24

4.3.3.1 Used variables

Some of the variables on these applications are user variables used to setup the application and some are internal variables like counters and flag. To help the user to identify which are the user variables that will help him to setup the application, the name of the internal variables that are not meant to be modified by the user have the prefix “_” (underscore), in this way the user variables appear first on the list of variables. Find next the list of user variables:

- private aio *aConvSpeed[nCnvNum]*

Linked to an analog output controlling the speed of the conveyor. Array of two elements, one per conveyor.

- private bool *bPickDetection*

If set to true, the application checks if the part was really picked, if picks fails the robot picks again.

- private bool *bSetConvSpeed[nCnvNum]*

If set to true, the application controls the speed of the conveyor in order to avoid that an object leaves the tracking window. This variable should only be set to true on the last robot over the conveyor, it should remain to off for the rest of the robots. Array of two elements, one per conveyor. See the section “Working tracking window and conveyor speed control” for more details on how the speed is controlled.

- private bool *bTransferData[nCnvNum]*

If set to true, the application will send to the next downstream robot all the objects that were not treated by itself, either because the objects have crossed the downstream limit (overflow) or because they have been rejected by the conveyor strategy (ID Selection, Area Distribution or Bypass).

- private dio *dBlow*

Linked to a digital output, it is the signal to enable/disable the blow on the tool.

- private dio *dNoPartInTool*

Linked to the digital input detecting if vacuum has been created on the tool. In other words, detecting if a part has been taken. When the input is equal False, a part is been taken. This input is ignore if the boolean *bPickDetection* is not set to true.

- private dio *dTrigger[nCnvNum]*

Linked to the digital output used to trigger the conveyor. Array of two elements, one per conveyor. If one or both conveyors are triggered by an external signal, one or both elements of the variable should be erased, leaving only the elements which have the link to the CS8C outputs triggering the conveyors.

- private dio *dVacuum*

Linked to a digital output, it is the signal to enable/disable vacuum on the tool.

- private frame *fConveyor[nCnvNum]*

Conveyor frame taught with the wizard. It is defined in reference to world of the robot. The variable is updated with the values of the conveyor frame when calling the program *InitTracking()*. Array of two elements, one per conveyor.

- private frame *fDetection[nCnvNum]*

Detection frame taught with the wizard. It is defined in reference to world of the robot. The variable is updated with the values of the detection frame when calling the program *InitTracking()*. Array of two elements, one per conveyor.

- private joint *jConveyorHome[nCnvNum]*

Position used as start position in top of the conveyor and as waiting position when there is not objects on the conveyor's queue. This joint position defines the configuration of the arm (joint configuration) to work on the conveyor. Array of two elements, one per conveyor.

- private joint *jDropPart*

Position used while stopping the application to drop the part on the tool.

- private joint *jPark*

Park position, the robot is send to the park position at the end of the stop program.

- private mdec *mNomSpeed*

Speed used to move the robot to the waiting positions.

- private mdec *mSlow*

Speed used to move the robot to the start position, drop part position and park position.

- private mdec *mTracking*

Speed used to execute all the trackOn, trackOff and the depart movements from the pick and place positions.

- private mdec *mTrackingNB*

Speed used to execute the movements towards the pick and place positions. Blending for this movements is set to off because there is not waitEndMove instructions after these movements.

- private num *nBypass[nCnvNum]*

Number of objects to be transferred to the next robot after selecting one object. If the value of this variable is set to 0, the robot will select all objects. The Bypass count is done with the remaining objects selected by the *ID_Selection* or *Y_Selection* variables. This variable should be set to 0 on the last robot on the conveyor. Array of two elements, one per conveyor.

- private num *nID_Selection[nCnvNum,nNum]*

Objects's ID to be selected by the robot, any ID not present in this list is not selected. If the value of any element of the array is set to -1, the robot will select all objects regardless their ID. Array of two dimentions where the first index defines the conveyor number and the second index defines the ID of the objects to be selected by the robot, the second dimension should have as many elements as number of IDs to be selected by the robot.

- private num *nNomConvSpeed[nCnvNum]*

Nominal conveyor speed defined in mm/sec. When the managemnt of the conveyor speed is activated with the boolean *bSetConvSpeed[nCnvNum]*, this is the reference value to calculate the percentage of speed comanded on the conveyor, going from 100% to 0%. Array of two elements, one per conveyor. See the section "Working tracking window" for more details on the zone where the speed is managed.

- private num *nW8Dist2UpB4Byp[nCnvNum]*

Distance from the upstream limit used by the conveyor strategy program. The program waits until the object crosses this limit before removing an object from the conveyor's queue. The goal is to wait until the object has leave the vision area before removing it from the queue. Removing the object from the queue before it leaves the vision area could have as a consequence that the object is stacked again in the queue if the vision system sends to the robot two times the same object. Array of two elements, one per conveyor.

- private num *nWaitDistToDown[nCnvNum,nNum]* or *nWaitDistToDown[nCnvNum]*

Distance from the downstream limit used to define the end of the working tracking. On the example *Ex_4A_Q_DT_VV* it is a two dimentions array where the first index defines the conveyor number and the second index defines the working area on the conveyor. On the example *Ex_TP_Q_DT_VV* it is a one dimention array with two elements, one per conveyor. Fom more details on the use of this variable see the section "Working tracking window and conveyor speed control".

- private num *nWaitDistToUp[nCnvNum]* or *nWaitDistToUp[nCnvNum,nNum]*

Distance from the upstream limit used to define the begining of the working tracking. On the example *Ex_4A_Q_DT_VV* it is a one dimention array with two elements, one per conveyor. On the example *Ex_TP_Q_DT_VV* it is a two dimentions array where the first index defines the conveyor number and the second index defines the working area on the conveyor. Fom more details on the use of this variable see the section "Working tracking window and conveyor speed control".

- private num *nID_Selection[nCnvNum,nNum]* Defines the limits of the area on the conveyor where the robot selects objects. The value of reference is the value on Y of the TRSF of the point, which means that for a vision-like detection conveyor, the limits are defined in reference to the Y axis of the detection frame because the detection frame is comun to all robots working over the same conveyor, and for a sensor-like detection conveyor, the limits are defined in reference to the Y axis of the conveyor frame. The variable is a 2 dimension array, where the first dimension defines the conveyor number and the second dimension has two elements, the first with the minimum value and the second with the maximum value that the object has to have in order to be selected.

- private point *pPick*

Point used to pick the parts on the infeed conveyor. It is defained in reference to the frame *fDetection[0]*.

- private point *pPlace*

Point used to place the parts on the input conveyor. It is defained in reference to the frame *fDetection[1]*.

- private sio *skToRobot[nCnvNum]*

Linked to a TCP socket client. The TCP socket client has the address of the next robot on the conveyor, and it is used to transfer the objects from this robot to the next robot on the conveyor. Array of two elements, one per conveyor. Each TCP socket client must use an individual TCP port, therefore each conveyor uses a different communication channel. The next robot on the line should have a TCP socket server per conveyor, opening the communication on the same TCP port as the TCP client.

- private string *sConveyorList[nCnvNum]*

List of conveyor libraries created with the Wizard and used for the application. Array of two elements, one per conveyor.

- private tool *tGripper*

TCP of the tool used to pick and place the parts.

Ex_4A_Q_DT_VV

Ex_TP_Q_DT_VV

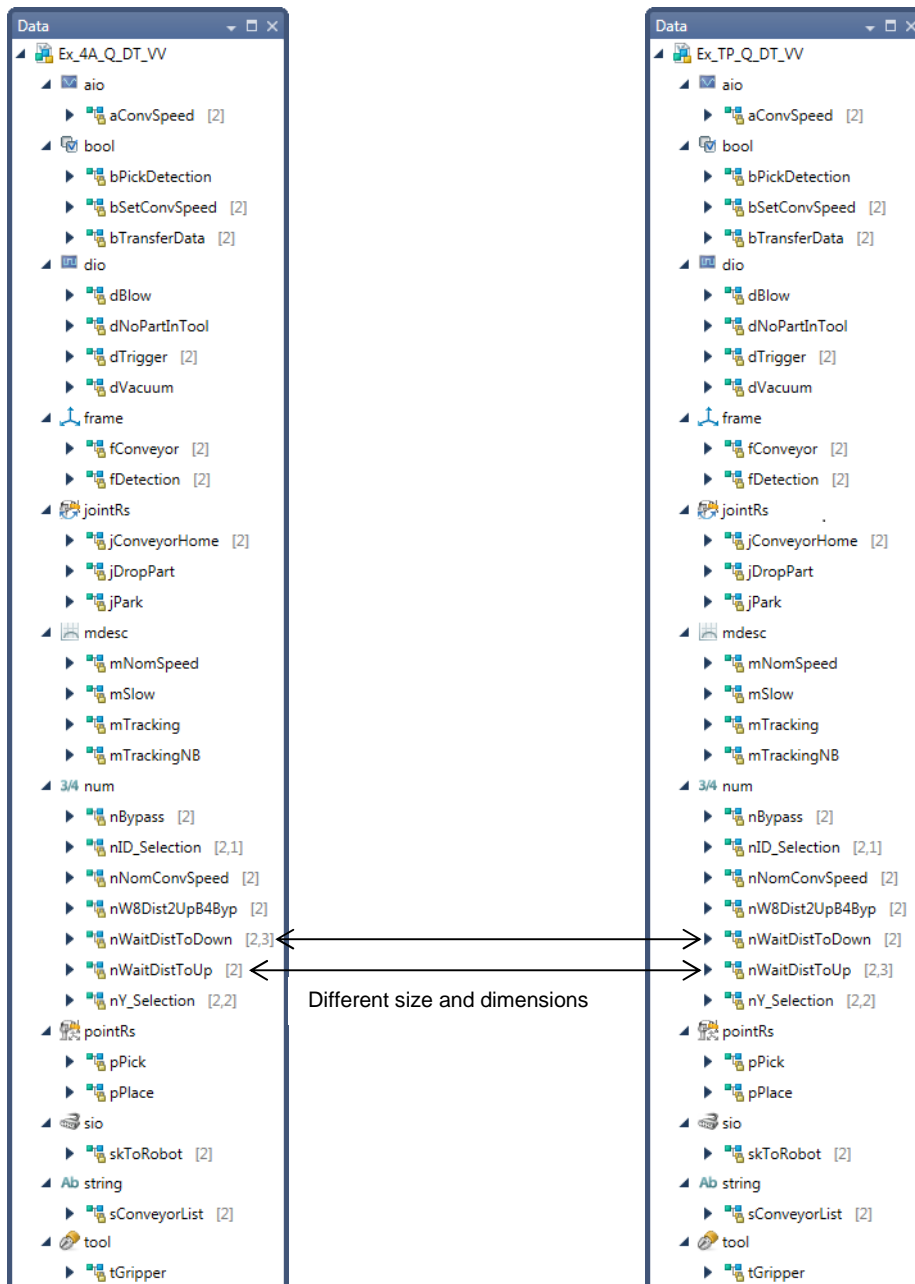


Figure 4.25

4.3.3.2 Select conveyor library

Select the variable *sConveyorList* and enter the names of the conveyors libraries created with the Wizard.

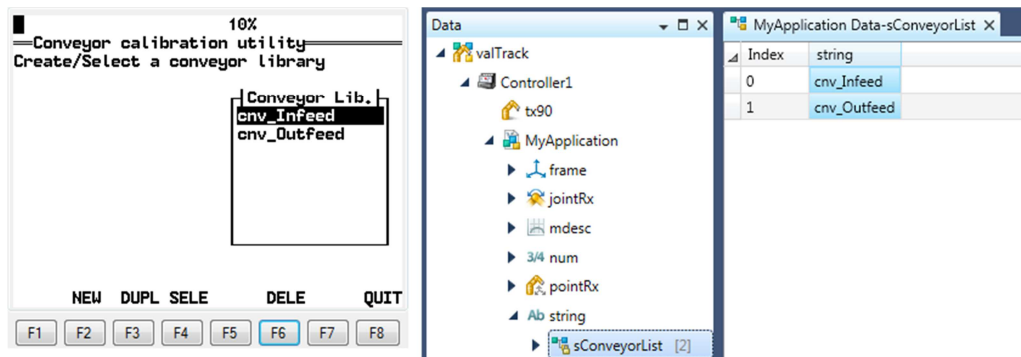


Figure 4.26

4.3.3.3 Teach positions

Define the tool and teach your points *jPark* and *jDropPark*, *jConveyorHome[0]* and *jConveyorHome[1]*. While teaching the positions *jConveyorHome[]* put special attention at the value of axes 4, remember that this joint position is used as a configuration of the arm to move to the objects on the conveyors.

4.3.3.4 Working tracking window and conveyor speed control

The working tracking window and the conveyor speed control window are defined with the variables *nWaitDistToUp* and *nWaitDistToDown* which are offsets from the upstream and downstream limits respectively.

The working tracking window defines the section of the tracking window where the robot is allowed to work on the conveyor. The size of the working tracking window changes depending on the status of the conveyor speed control, the working tracking window is bigger if the conveyor speed control is activated since the robot can work closer to the downstream limit without going out of reach considering that the conveyor will reduce its speed as the object approaches to the downstream limit.

The conveyor speed control window is inside the working tracking window, the conveyor speed control window is at the end of working tracking window. The speed of the conveyor is defined by the position of the next object on the conveyor's queue, the management is as follows (see figures 4.27 to 4.31 for a visual description):

- When the object is upstream of the upstream limit, the speed of the conveyor is at 100%
- When the object is inside of the working tracking window but upstream from the conveyor speed control window, the speed of the conveyor is at 100%.
- When the object is inside the conveyor speed control window, the speed of the conveyor is proportional to the position of the object inside conveyor speed control window. When the object is at the beginning of the conveyor speed control window the conveyor speed is at 100%, when the object is at the middle of the conveyor speed control window the conveyor speed is at 50%, and when the object is at the end of the conveyor speed control window the conveyor speed is at 0%.
- When the object is removed from the conveyor's queue, the conveyor speed is back to 100%.

Example Ex TP Q DT VV:

The limits management on this example is meant for a TP80 application where the robot is placed at the side of the conveyor and the robot can only work on the downstream half side of its full reach.

Also, on a TP80, the maximum speed of joint 2 is higher than the speed of joint 1, so the idea is to define a working tracking window where joint 2 works more than joint 1, then the working tracking window is defined by the work envelope of joint 2.

To avoid that the arm moves to the upstream side of its full working envelope, the lower limit on joint 1 is set to -15° and the lower limit on joint 2 is set to -0.1° .

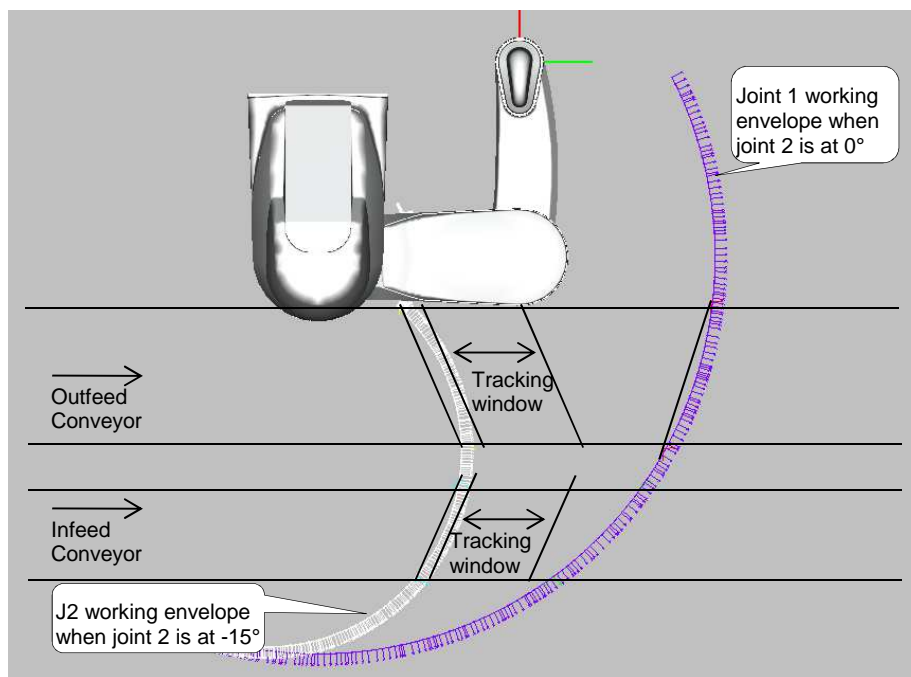


Figure 4.30

Example Ex TP Q DT VV without conveyor speed control:

The limits management on this example is meant for a TP80 application where the robot is placed at the side of the conveyor and the robot can only work on the downstream half side of its full reach.

The same management is applied in both conveyors, infeed and outfeed conveyors.

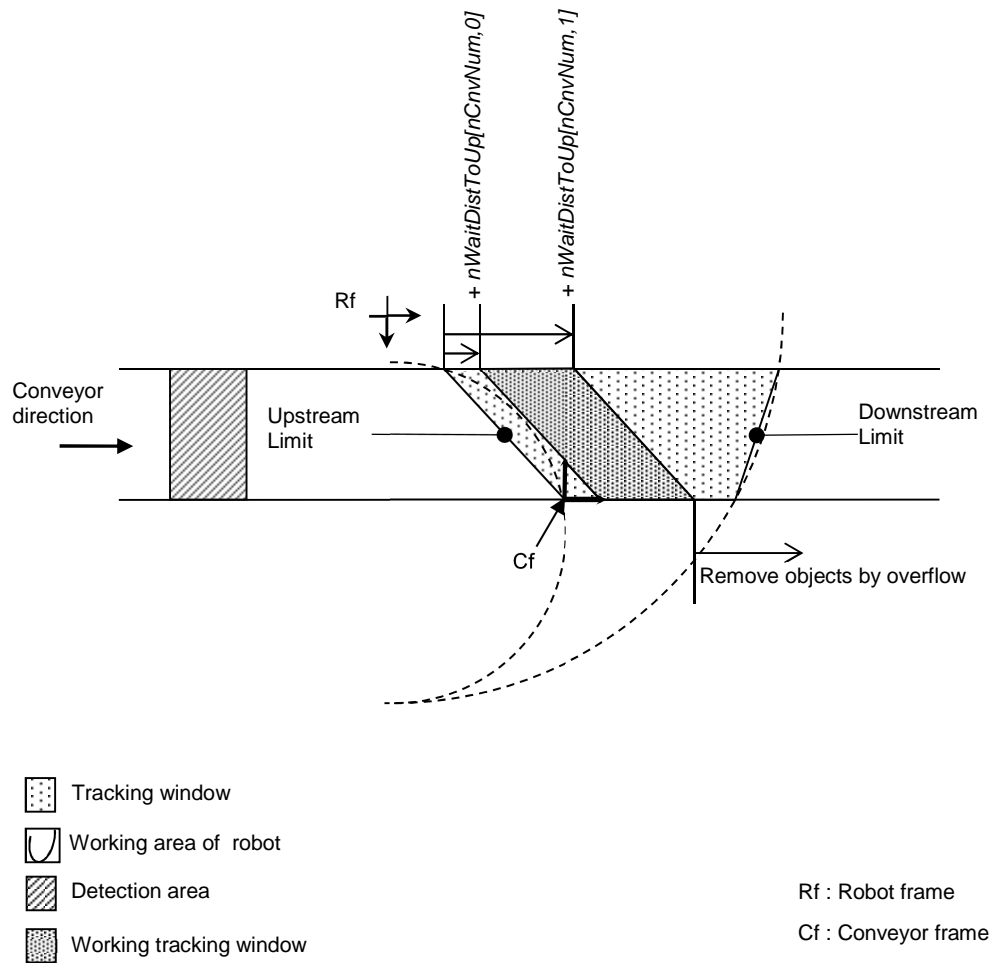
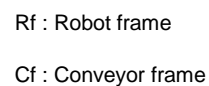


Figure 4.31

The limits management on this example is meant for a TP80 application where the robot is placed at the side of the conveyor and the robot can only work on the downstream half side of its full reach.

[illegible]

STÄUBLI

4.3.3.5 Queueing

Definition

- Multiple robots working over the same conveyor.
- Only one detection device.
- Only one encoder.
- The first robot recovers all the objects from the detection device.
- The first robot handles some of the objects and transfer the rest to the next robot.

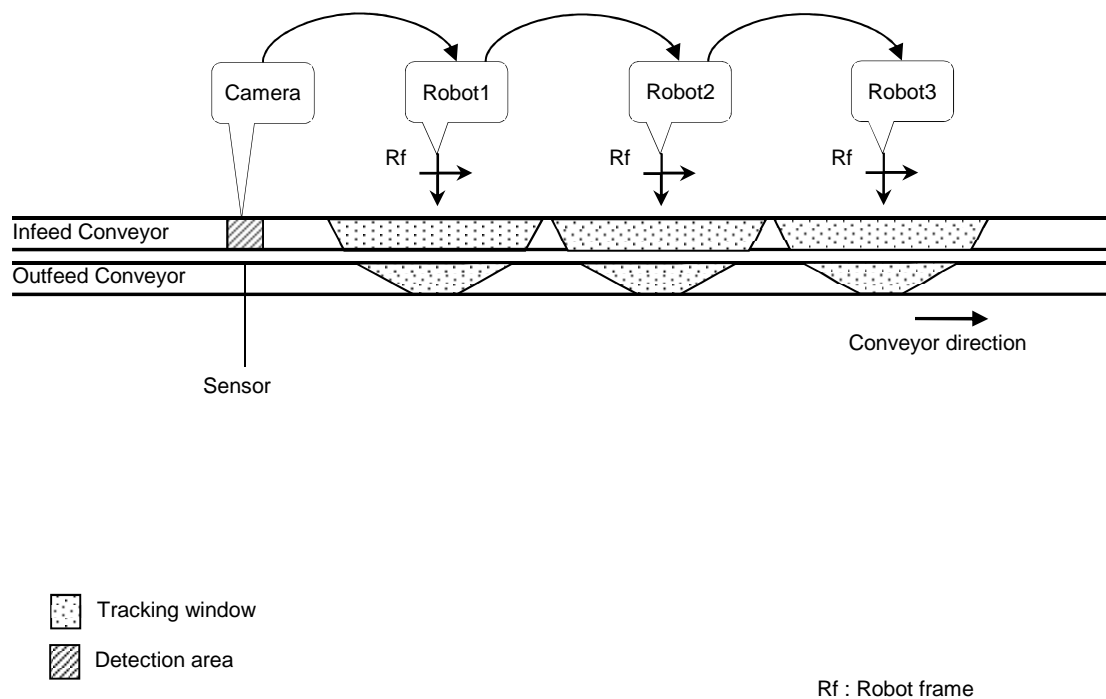


Figure 4.33

Communication Between Robots

- The objects are sent from one robot to the next by Sockets TCP/IP.
- Upstream robot sends the information via a socket client.
- Downstream robot receives the information via a socket server.
- The sockets are defined on the Control Panel of the robot.

On the next figure, we can see the communication between robots uses port 1001 for the first conveyor, and port 1002 for the second conveyor.

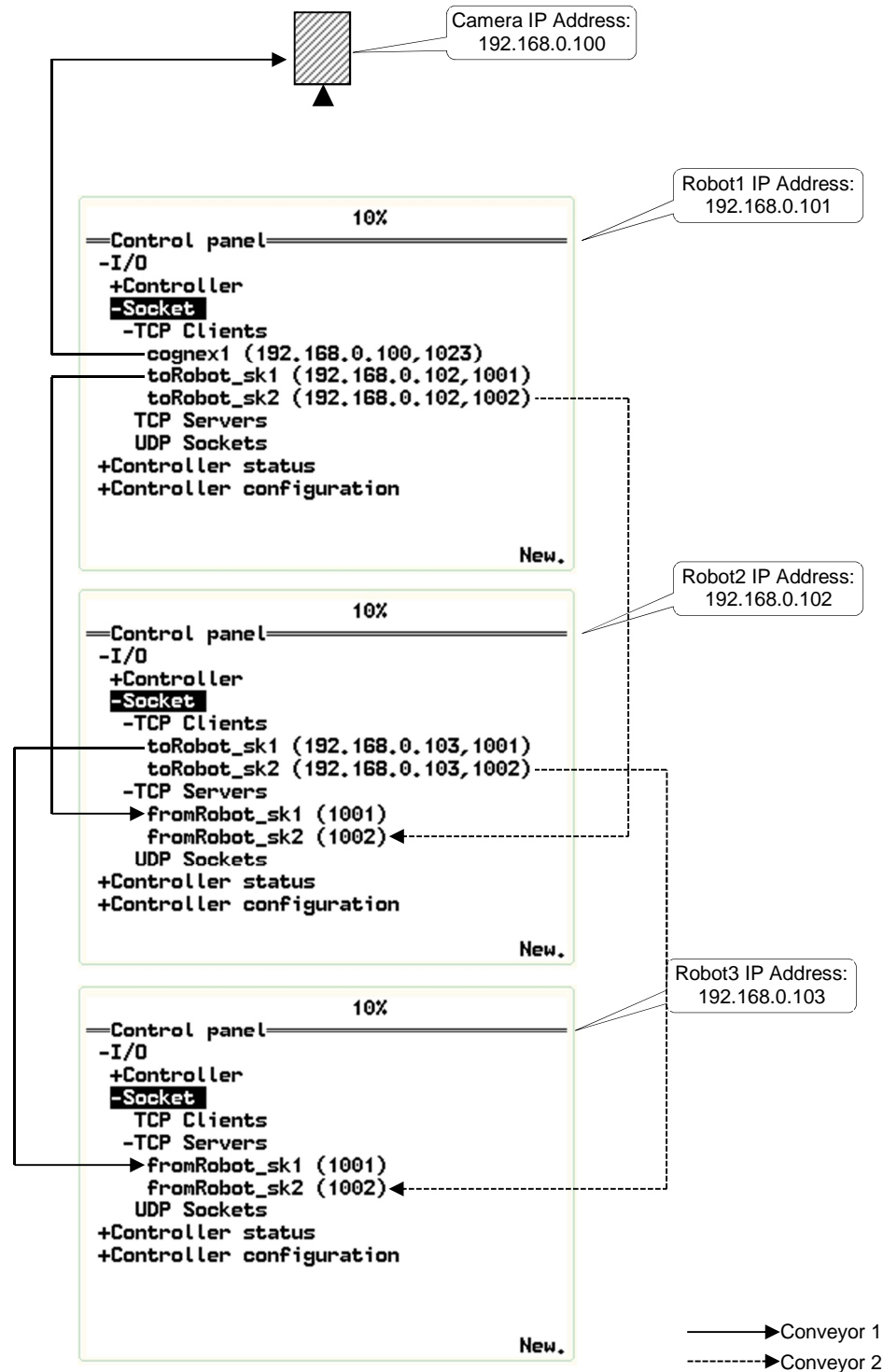


Figure 4.34

Conveyor libraries

The first robot on the line uses a detection library depending on the detection device used, like a vision camera or a sensor. In the case of the vision system, it is the first robot the only robot to generate the trigger signal for the camera, the rest of the robots should not generate a trigger signal.

The rest of the robots, starting from the second to the last robot on the line, they all use the detection library Socket1 for the first conveyor and Socket2 for the second conveyor. None of this robots should generate a trigger signal.

The scale factor on all the robots is the same. It is possible to use the average value of the scale factor calculated by each robot or it is possible to be calculated manually:

1. Place the calibration target at the beginning of the conveyor.
2. Set the encoder value to zero (ex: e00CurrPos for the first channel on the Dual ABZ board).
3. Start the conveyor and stop it when the calibration target arrives at the end of the conveyor.
4. Note the encoder value at this time.
5. Using a measuring device as precise as possible (like a laser beam), measure the distance that the calibration target has travel.
6. Calculate the scale factor: (Traveled distance in mm) / (Encoder Value).



Warning

When entering manually the value of the scale factor on the Wizard, use as many decimals as possible. An error of 1 thousandth on the scale factor value could be an error in millimeters on the position of the object at the end of the conveyor.

10%

==Setup conveyor parameter==

-Conveyor-- -Advanced--

EncoderLib: D_ABZ_00 MatchTol: 5

Detect Lib: Cognex1

BufferSize: 10 ClkSignal: fOut0

ScaleFact : 0.567563 ClkPeriod: 150

LockDist : 0

-TrackingWindow--

Height : 500

ThruUpLim : none -Event Logger --

ThruDwnLim: none Enable Log: TRUE

GENE ENCO DETE Ok

Robot1
First conveyor

10%

==Setup conveyor parameter==

-Conveyor-- -Advanced--

EncoderLib: D_ABZ_00 MatchTol: 5

Detect Lib: Socket1

BufferSize: 10 ClkSignal: none

ScaleFact : 0.567563 ClkPeriod: 0

LockDist : 0

-TrackingWindow--

Height : 500

ThruUpLim : none -Event Logger --

ThruDwnLim: none Enable Log: TRUE

GENE ENCO DETE Ok

From Robot2 to the last
First conveyor

Figure 4.35



Figure 4.36

Conveyors Calibration

Before doing any calibration, make sure that the encoders are connected to all the robots in the same way, and all the robots see the encoder value increasing when the conveyor is moving forward. The same for the trigger signal, if the trigger is generated by a robot (standard configuration for vision), the trigger signal must come back to this robot and send identically to all the robots, if the trigger signal is generated by a sensor, the sensor must be connected to the trigger input signal on each robot.

The calibration of the conveyor frame is done individually for each robot and each conveyor. There is not link in between the different conveyor frames

The calibration of the detection frame must be done at the same time for all the robots working on the same conveyor. To proceed, run the Wizard on all the robots at the same time:

1. Execute the step by step procedure of the function "Define frame for vision" until all robots are asking to start the conveyor (Figure 4.37).
2. Start the conveyor and stop it when the calibration target is inside the tracking window of the first robot.
3. Make sure all robots show the same Encoder Value and the same Latched Value. Start Over otherwise.
4. Finish the procedure on the first robot.
5. Start the conveyor and stop it when the calibration target is inside the tracking window of the second robot.
6. Finish the procedure on the second robot.
7. Repeat steps 5 and 6 for all other robots.

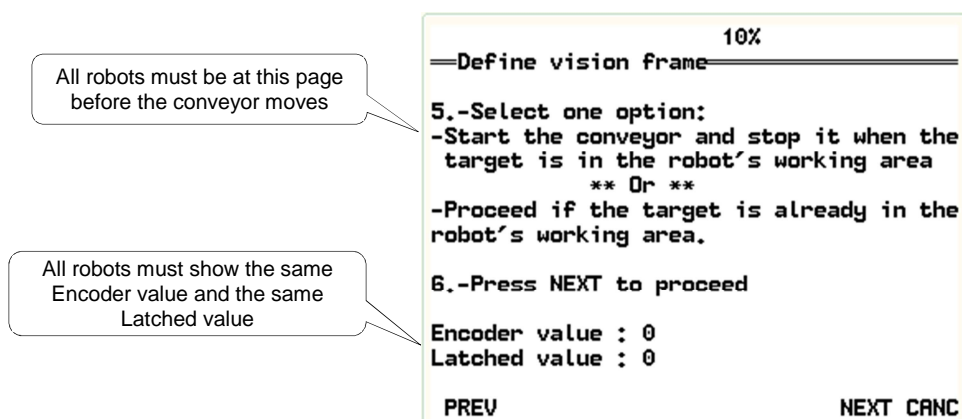


Figure 4.37

In the case of the detection by sensor of a single object, the point of the object is taught with the program `teachLoc()` provided with the sensor examples. The procedure must be done in all robots at the same time. Run the program `teachLoc()` on all the robots:

1. Execute the step by step procedure until all robots are ready to detect an object (Figure 4.38).
2. Start the conveyor and wait until an object is detected on the first robot.
3. Stop the conveyor when the object is inside the tracking window of the first robot.
4. Make sure all robots detected the object. Start over otherwise.
5. Finish the procedure on the first robot.
6. Start the conveyor and stop it when the object is inside the tracking window of the second robot.
7. Finish the procedure on the second robot.
8. Repeat steps 6 and 7 for all other robots.

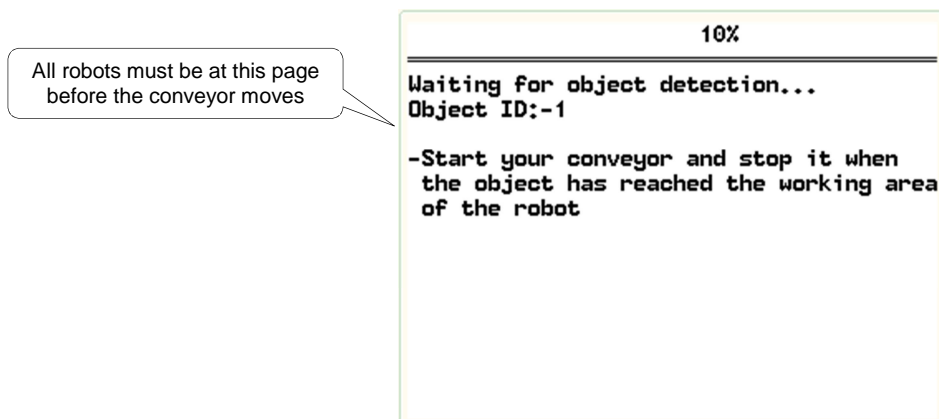


Figure 4.38

Queuing Strategies

1. Overflow

- The strategy consist on sending to the next robot all the objects that have crossed the downstream limit of the tracking window.
- Other strategies can be implemented but this strategy should remain activated all the time as a base of any other strategy.
- The strategy is implemented on the examples, to start transferring objects to the next robot set to true the variable `bTransferData[nCnvNum]`. Make sure the sockets `toRobot_sk1` and/or `toRobot_sk2` are created and correctly defined.

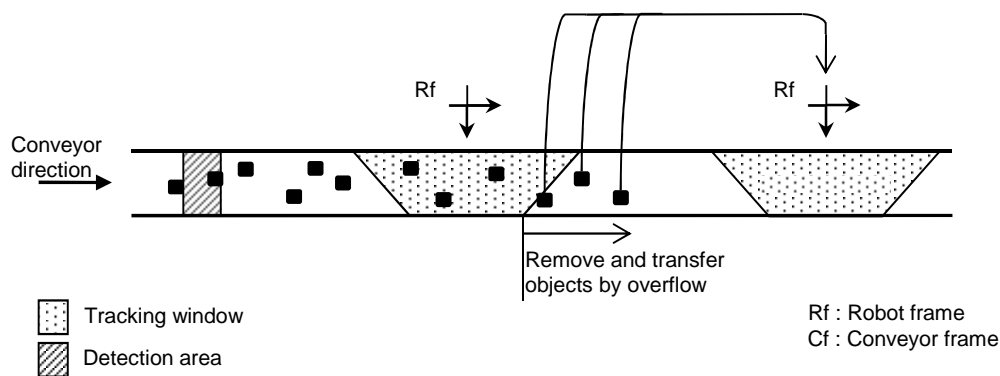


Figure 4.38

2. Bypass

- The strategy consist on selecting 1 object and then sending a number $nBypass$ of objects to the next robot.
- The Bypass count is done with the remaining objects selected by the ID Selection and Area Distribution strategies
- Overflow strategy remains activated.
- Real strategy implemented: Overflow + Bypass
- To transfer objects by Bypass, set to true the variable $bTransferData[nCnvNum]$ and define the number of objects to transfer by Bypass on the variable $nBypass[nCnvNum]$. Make sure the sockets toRobot_sk1 and/or toRobot_sk2 are created and correctly defined.
- In the case where the detection device detects twice the same object, the already queued objects cannot be removed from the conveyor's queue immediately after they have been queued because every time that a new object arrives from the detection device, the conveyor compares it with the already queued objects to check if the new object isn't already queued. The variable $nW8Dist2UpB4Byp[nCnvNum]$ is a distance from the downstream limit and the objects are not transferred to the next robot until the objects cross this limit.

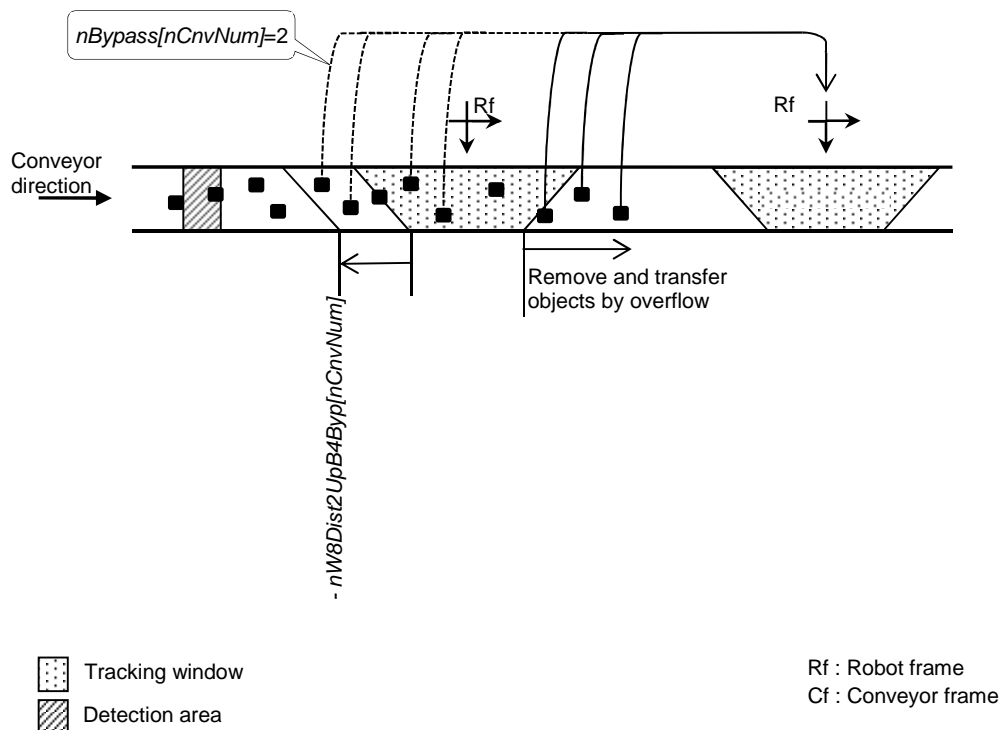


Figure 4.39

3. ID Selection

- The strategy consist on selecting the objects by their ID. The robot selects the objects with an specific ID and transfers to the next robot any object with a different ID.
- Overflow strategy remains activated.
- Real strategy implemented: Overflow + ID Selection.
- To transfer objects by ID Selection, set to true the variable $bTransferData[nCnvNum]$ and define the IDs to be selected by this robot on the variable $ID_Selection[nCnvNum, nNum]$, any object with an ID not in this list is transferred to the next robot. Make sure the sockets toRobot_sk1 and/or toRobot_sk2 are created and correctly defined.
- The objects cannot be removed from the conveyor's queue inmediately after they have been queued because every time a new object arrives from the detection device, the conveyor compares it with the already queued objects to check if the new object isn't already queued, in the case where the detection device detects twice the same object. The variable $nW8Dist2UpB4Byp[nCnvNum]$ is a distance from the downstream limit and the objects are not transferred to the next robot until the objects cross this limit.

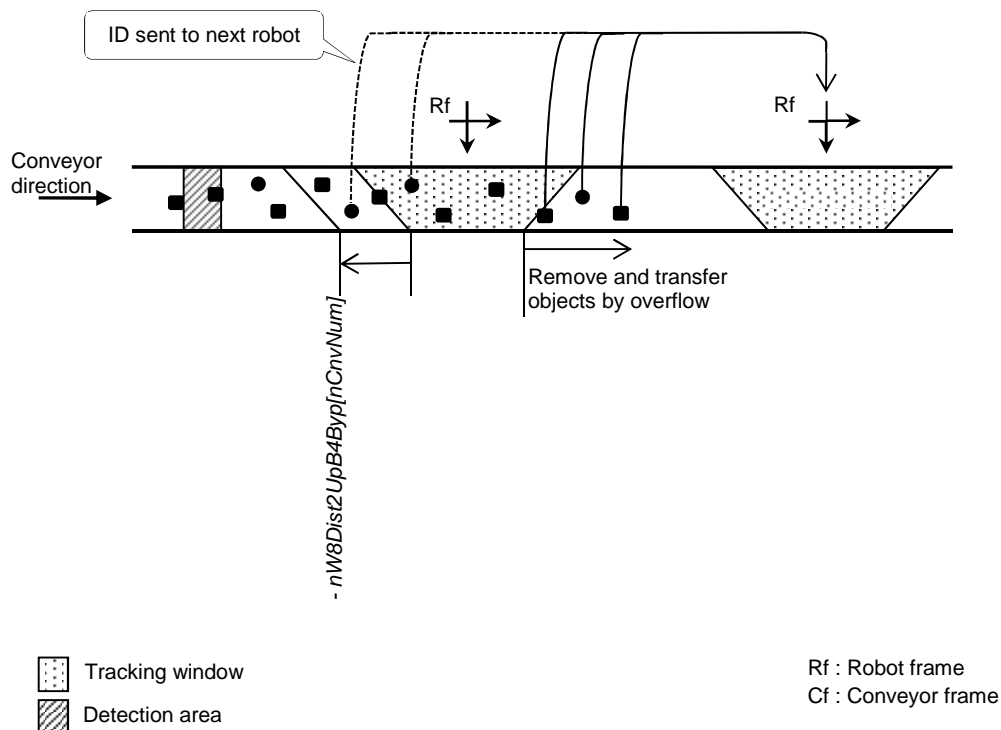


Figure 4.40

4. Area Distribution

- The strategy consist on selecting the objects by their position on the conveyor. The robot selects the objects located inside an specific area on the conveyor and transfers to the next robot any object outside this area.
- Overflow strategy remains activated.
- Real strategy implemented: Overflow + Area Distribution.
- To transfer objects by Area Distribution, set to true the variable *bTransferData[nCnvNum]* and define the limits where the robot should select objects on the variable *nY_Selection[nCnvNum,nNum]*, any object outside these limits is transferred to the next robot. Make sure the sockets toRobot_sk1 and/or toRobot_sk2 are created and correctly defined.
- The objects cannot be removed from the conveyor's queue inmediately after they have been queued because every time a new object arrives from the detection device, the conveyor compares it with the already queued objects to check if the new object isn't already queued, in the case where the detection device detects twice the same object. The variable *nW8Dist2UpB4Byp[nCnvNum]* is a distance from the downstream limit and the objects are not transferred to the next robot until the objects cross this limit.

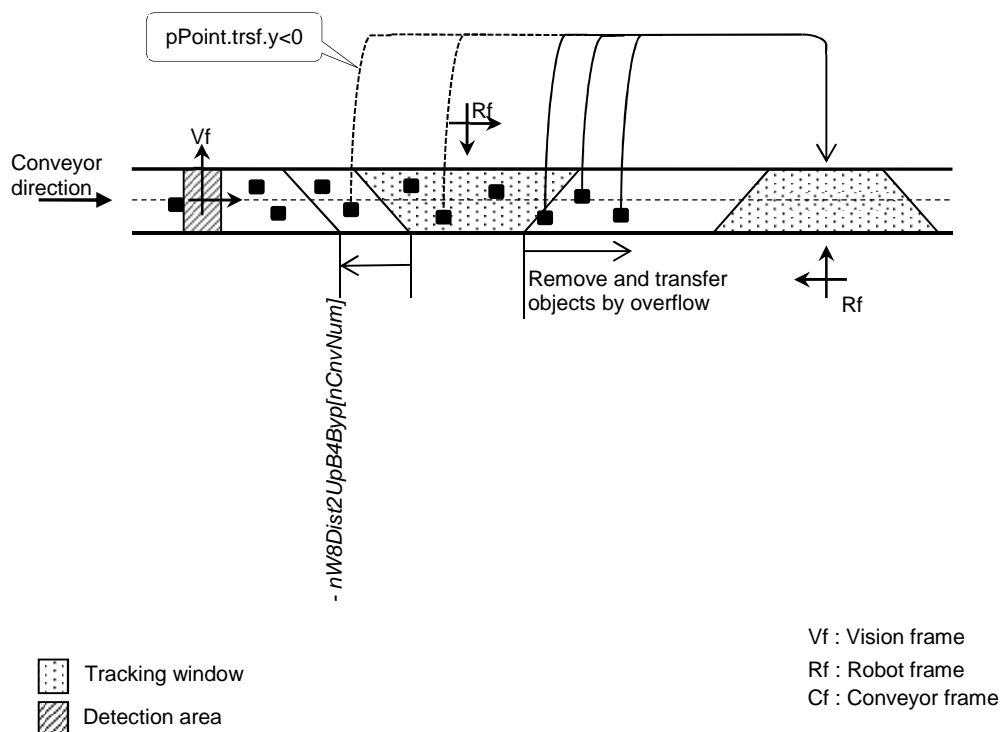


Figure 4.41

4.3.3.6 Cycle description

These examples applications, like any other application, start with the program *start()*. The only thing done in the program *start()* is calling the program *cellManager()*. The program *cellManager()* initialize all variables, launches all VALTrack tasks and launches all the tasks of the application. Once the application is running, the program *cellManager()* is testing if there is a tracking (motion) error, and if there is one, the robot is sent to the home position, all the tasks of the application are stopped, the variables are initialized and all the application tasks are launched again. All VALTrack tasks are kept running while a tracking error. The program *cellManager()* also displays the user page where it is shown the cycle status, the cycle time counters and the objects counter.

The tasks of the application launched by the program *cellManager()* are: *productionCycle*, *cnv0_StrategyT*, *cnv1_StrategyT*, *TransferQueue0*, *TransferQueue1* and *blow*.

- *productionCycle*: The task executes the program *productionCycle()*. The program *productionCycle()* commands the movements of the robot by calling the programs *trackingPick()* and *trackingPlace()* on an infinite loop. The program *trackingPick()* picks a part on the first conveyor and the program *trackingPlace()* places it on the second conveyor.

The task *productionCycle* and the tasks *cnv0_StrategyT* and *cnv1_StrategyT* work in parallel accessing the conveyors queues. A flag is used to avoid two tasks access the same conveyor queue at the same time.

The flowchart of the task *productionCycle* is as follow:

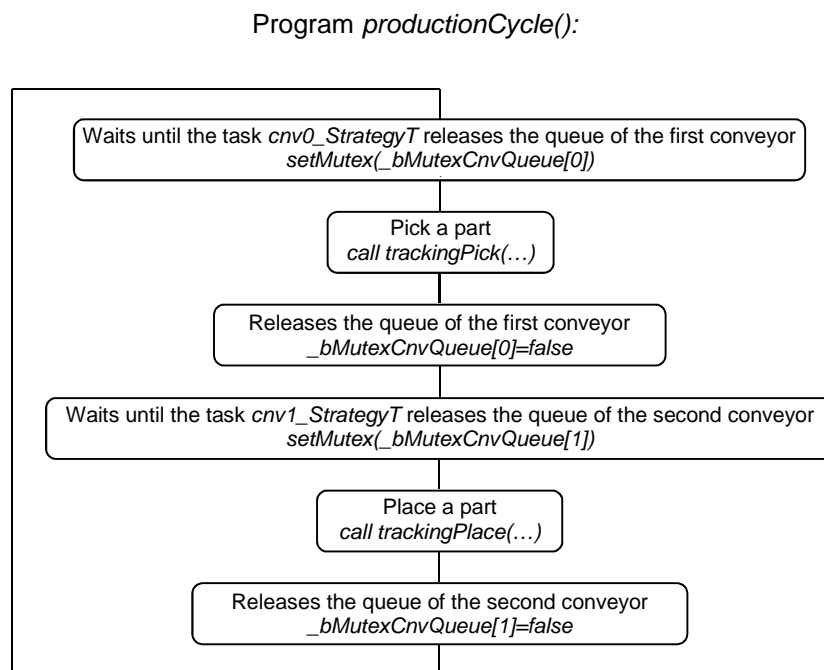


Figure 4.42

The cycle of the programs *trackingPick()* and *trackingPlace()* is basically the same, with the difference on the management of the tool. The flowchart of both is as follow:

Programs *trackingPick()* and *trackingPlace()* :

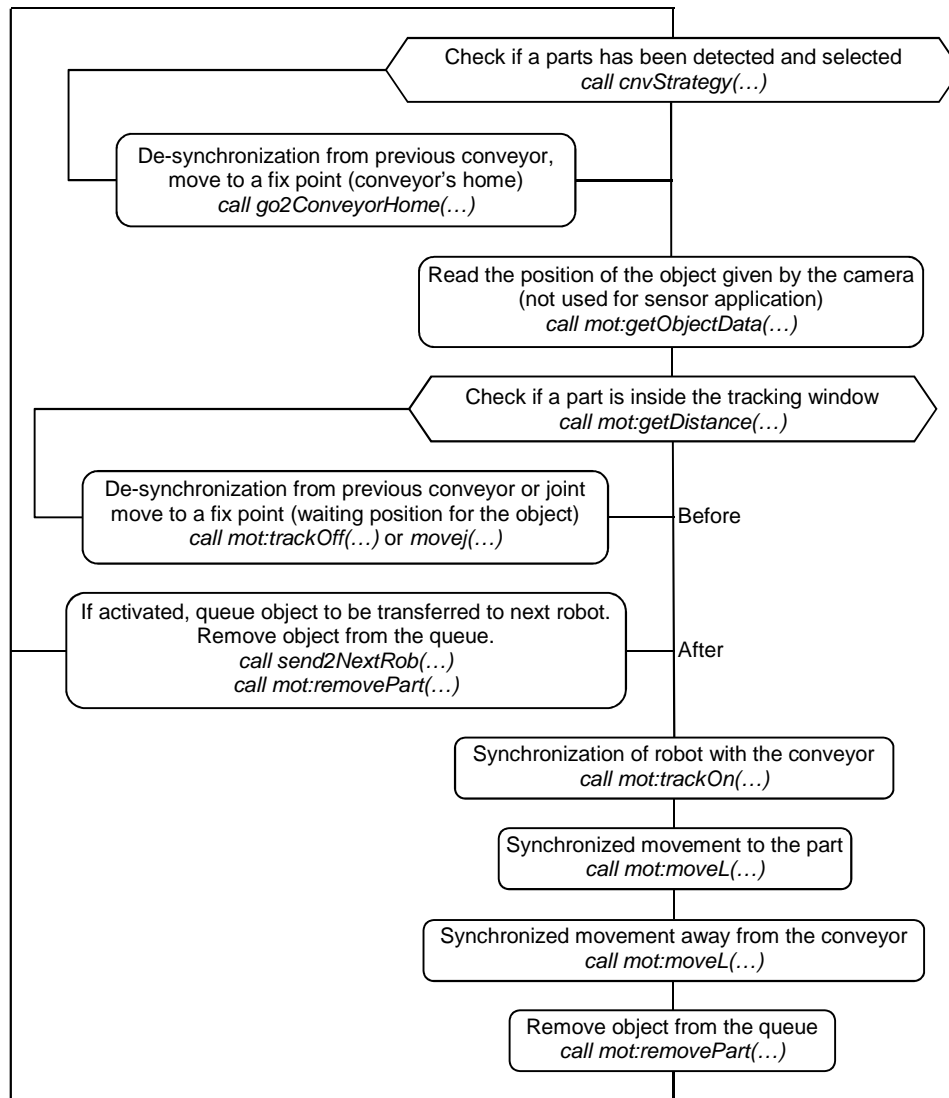


Figure 4.43

The previous flowchart allows the robot to have a fluid movement in between conveyors by never doing a trackOff to an intermediate position as long as there are objects present on the tracking window of each conveyor.

The program *cnvStrategy(nConvNum,pPoint)* is where the conveyor strategy is implemented. It checks if an object is already on the conveyor's queue, and decides if the object should be removed or selected. The strategies implemented are Overflow, ID Selection, Area Distribution and Bypass. It also controls the speed of the conveyor if the option is activated.

The flow chart of the program is as follows:

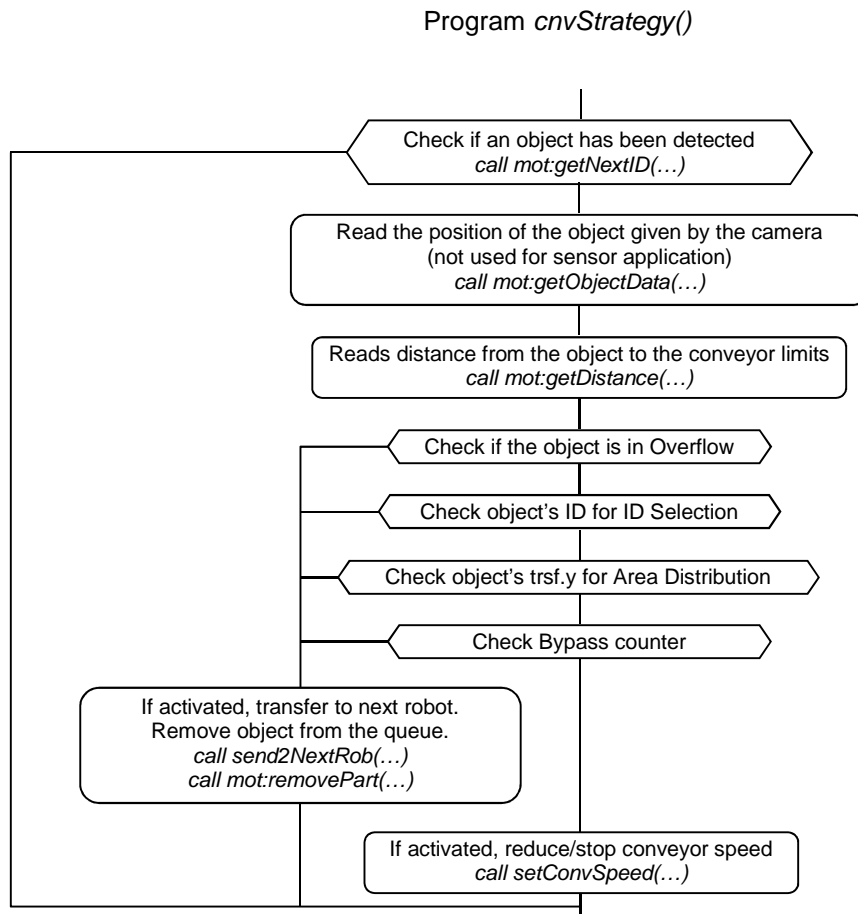


Figure 4.44

- *cnv0_StrategyT* and *cnv1_StrategyT*: Both tasks execute the same program *cnvStrategyTask(nCnvNum,pPoint)*, the task *cnv0_StrategyT* executes it with conveyor number 0 and the point *pPick*, and the task *cnv1_StrategyT* executes it with conveyor number 1 and the point *pPlace*. Their goal is to reduce the working load of the task *productionCycle*. Therefore, while the task *productionCycle* is executing the program *trackingPick()*, the task *cnv1_StrategyT* is removing all unwanted objects from the outfeed conveyor's queue, so when the task *productionCycle* executes the program *trackingPlace()* the outfeed conveyor's queue has been clear out from all unwanted objects. In the same way, while the task *productionCycle* is executing the program *trackingPlace()*, the task *cnv0_StrategyT* is removing all unwanted objects from the infeed conveyor's queue.

The program *cnvStrategyTask(nCnvNum,pPoint)* is very simple, it only calls the program *cnvStrategy(nCnvNum,pPoint)*, which is the same program called by the programs *trackingPick()* and *trackingPlace()*.

Program *cnvStrategyTask()* :

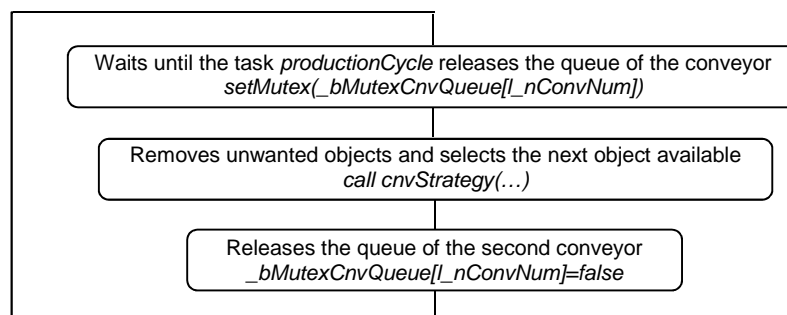


Figure 4.45

- *TransferQueue0* and *TransferQueue1*: Both tasks execute the program *transferQueue(num x_nCnvNum)* with different conveyor number. The task *TransferQueue0* is for the infeed conveyor and the task *TransferQueue1* is for the outfeed conveyor. The goal of these tasks is to transfer to the next robot the objects queued to be transferred by the tasks *productionCycle*, *cnv0_StrategyT* and *cnv1_StrategyT*.

- *blow*: The task execute the program *blow()*. Since vacuum is always on, only the blow is controlled on this application. When the task *productionCycle* sets the timer to a defined amount of time, the task *blow* activates the blow and waits until the time expires to turn it off and reset the timer.

4.3.4 Work with more than 2 conveyor libraries

VALTrack libraries are ready to work with up to two conveyors. If more than 2 conveyors will be used at the same time, besides adding as many elements as conveyors needed to the variables *sConveyorList[n]*, *fConveyor[n]* and *fDetection[n]* the library Motion will have to be modified to be ready to handle more than two conveyors:

1.- The library Motion has two libraries to be used as conveyor libraries: *cnv0* and *cnv1*. Add as many libraries as needed. In the figure 4.45 there is one more library added as *cnv2*.

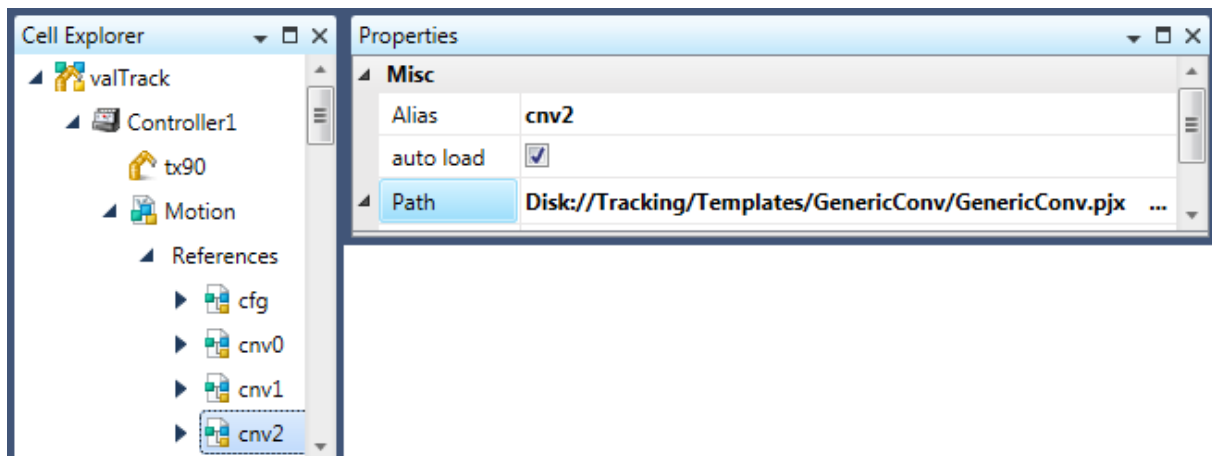


Figure 4.46

2.- All programs using the conveyor number as parameter (*x_nConvNum*) will have to be modified to be ready to handle the new libraries. In the figure 4.46 the program *getNextID* is modified to be ready to handle the new library added in figure 4.45. The modification for all programs is the same: add as many cases as conveyors libraries.

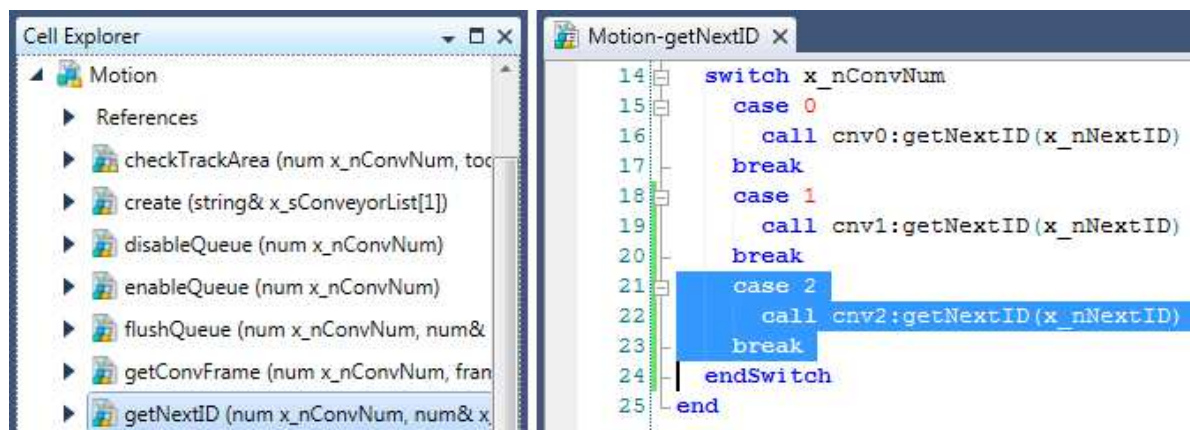


Figure 4.47

4.3.5 Initializing and launching all tracking tasks and variables.

In the initialization section of your application, program *initTracking()*, the library Motion is loaded under the identifier "mot" defined in step 4.3.1. Once loaded, the program *mot:create(sConveyorList)* is called, this program will launch one *tracking* task and three other tasks for each conveyor on the *sConveyorList* variable: *conveyor*, *encoder* and *detect*. Finally, the frames *fConveyor* and *fDetection* are initialized by respectively calling the functions *getConvFrame()* and *getDetectFrame()* of the [Motion Library](#).

The variable *fConveyor* can be an array of frames and it is aimed to contain the position and orientation of the conveyors declared with the string variable *sConveyorList*. This frame is defined by performing either the *complete Calibration sequence* or the *define Geometry* procedures available in the [Calibration Wizard](#). Changing this frame will not influence the way the robot will track the conveyor.

The variable *fDetection* can be an array of frames and is aimed to contain the position and orientation of the detection device associated with each conveyors declared in *sConveyorList*. This frame is defined by performing the *define Frame for Vision* procedure in the [Calibration Wizard](#). The detection device usually sends the coordinates of the detected object(s) with reference to that frame. It is possible to read these coordinates by using the program *getObjectData()* of the [Motion Library](#) and place them into a point variable. Thus, this point variable must be child of the *fDetection* frame.

The figure 4.47 shows a sample code of the Init section.



Note:

If you have not performed the procedure *Define frame for vision (F4)* from the [Calibration Wizard](#), the coordinates of the frame *fDetection* will remain 0, which would not be a problem if you are not working with a vision system.

```

begin
//
// Load motion library
l_nError=mot:libLoad("Tracking\Motion")
if l_nError!=0
    popUpMsg("Error while loading motion library: "+toString("",l_nError))
endif
//
// Start the libraries to perform conveyor tracking
call mot:create(sConveyorList)
//
// Waits until all tasks are up and running
for l_nConvNum=0 to size(sConveyorList)-1
do
    call mot:getStatus(l_nConvNum,l_nError)
    delay(0)
    until (l_nError==0)
endFor
//
//Reads the position of the frames defined with the Wizard,
//for all conveyors on the conveyor list
for l_nConvNum=0 to size(sConveyorList)-1
    // Get the conveyor frame
    call mot:getConvFrame(l_nConvNum,fConveyor[l_nConvNum],world,l_nError)
    if l_nError!=0
        call mot:getErrorMessage(l_sErrorMessage)
        popUpMsg(l_sErrorMessage)
    endif
    //
    // Get the detection frame
    call mot:getDetectFrame(l_nConvNum,fDetection[l_nConvNum],world,l_nError)
    if l_nError!=0
        call mot:getErrorMessage(l_sErrorMessage)
        popUpMsg(l_sErrorMessage)
    endif
endFor
//
end

```

Figure 4.48

4.4 Use the Motion Library

The motion library is a group of programs that allows interacting with the objects moving on the conveyor. These programs are meant to be called from the main application and allow for example to launch/kill all tracking tasks, check the position of an object on the conveyor, move from/to a fix position to/from a moving position on the conveyor, etc, etc...

Movements to/from a moving position, or conveyor tracking, have 3 main steps:

- 1.- A synchronization step in which the robot moves from a fix position to a moving position over the conveyor, during its trajectory the robot progressively integrates the speed vector of the conveyor to its own motion.
- 2.- A synchronized step in which the robot position is attached to the conveyor. If there isn't any movement commanded, the robot still moves at the same speed and direction than the conveyor. If a movement is commanded to a position moving with the conveyor, the final speed of the robot is the addition of the speed commanded for the movement and the speed of the conveyor.
- 3.- A de-synchronization step in which the robot is progressively zeroing the speed of the conveyor from its own motion.

When the robot is synchronized with the conveyor (step 2) a de-synchronization move (step 3) must be used before using a standard movement (movei, movej, movec)

For multiple picking on the conveyor, it's possible to use the trackOff only for the last pick. (cf. annex 1).

The next list is the list of programs available on the library Motion, each program is described on [chapter 4.4.1](#)

Motion

Public:

- create(string x_sConveyorList)
- trackOn(num x_nConvNum, point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)
- trackOnJ(num x_nConvNum, point x_pLoc, tool x_tTool, joint x_jConfig, mdesc x_mDesc, num& x_nMoveID, num& x_nError)
- moveL(point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)
- moveC(point x_plnter, point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)
- trackOff(point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)
- trackOffJ(joint x_jLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)
- resetTrkMotion()
- stopTrkMove ()
-
- disableQueue(num x_nConvNum)
- enableQueue(num x_nConvNum)
- flushQueue(num x_nConvNum, num& x_nError)
- getConvFrame(num x_nConvNum, frame& x_fConveyor, frame x_fRefFrame, num& x_nError)
- getConvSpeed(num x_nConvNum, num& x_nSpeed, num& x_nError)
- getDetectFrame(num x_nConvNum, frame& x_fDetection, frame x_fRefFrame, num& x_nError)
- getDistance(num x_nConvNum, point x_pLoc, num x_nTime, num& x_Dist2Up, num& x_nDist2Down, num& x_nError)
- getErrorMessage(string& x_sMessage)
- getNextID(num x_nConvNum, num& x_nNextID)
- getObjectData(num x_nConvNum, trsf& x_trObject, num& x_nError)
- getObjectLatch(num x_nConvNum, num& x_nLatchedValue, num& x_nError)
- getRobotArea(num x_nConvNum, num& x_nAreaNumber)
- getStatus(num x_nConvNum, num& x_nStatus)
- getTrackStatus (num& x_nTrackStatus)
- getVersion(string& x_sVersion)
- kill()
- removePart(num x_nConvNum)
- reset(num x_nConvNum, num& x_nError)
- sendToNextRobot(sio x_sSocketClient, num x_nTimeOut, num x_nID, trsf x_trObjectPos, num x_nLatchedValue, num& x_nError)
- setCalibrating(num x_nConvNum, bool x_bMode)
- teach(num x_nConvNum, tool x_tTool, frame x_fFrame, point& x_pLoc, num& x_nError)

4.4.1 Programs Description

- public create(string x_sConveyorList)

This program initializes and starts the motion library. According to the [libraries overview](#) , the motion library can be considered as the father library of the conveyor(s) library. Thus, starting the motion library simultaneously starts the conveyor library.

Input parameters

x_sConveyorList List of the conveyor to be initialized.

- public trackOn(num x_nConvNum, point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)

This program commands a linear motion to synchronize smoothly with a location moving with a conveyor.

Input parameters

x_nConvNum Index of the conveyor in the conveyor list. The list of conveyors known by the Motion library is the one passed as parameter to the program *create()*.
x_pLoc Location on the conveyor to move to.
x_tTool Tool to use for the motion.
x_mDesc Motion descriptor to use for the motion.

Output parameters

x_nMoveID Returns the ID number with which the movement is stored on the motion stack.
x_nError Error code.
 = 0 : No Error.
 = 1 : There is no object marked as "read" to synchronize on. This error is usually returned if program *getNextID()* has not been called previously.
 = 2 : Wrong sequence of commanded motion. This error code is returned, for instance, if the program is called twice.
 = 3 : Wrong conveyor number.

- public trackOnJ(num x_nConvNum, point x_pLoc, tool x_tTool, joint x_jConfig, mdesc x_mDesc, num& x_nMoveID, num& x_nError)

This program commands a linear motion to synchronize smoothly with a location moving with a conveyor.

Input parameters

<i>x_nConvNum</i>	Index of the conveyor in the conveyor list. The list of conveyors known by the Motion library is the one passed as parameter to the program <i>create()</i> .
<i>x_pLoc</i>	Location on the conveyor to move to.
<i>x_tTool</i>	Tool to use for the motion.
<i>x_jConfig</i>	Joint used to define the configuration of the arm to go to the point. This parameter is used instead of the configuration values of the point.
<i>x_mDesc</i>	Motion descriptor to use for the motion.

Output parameters

<i>x_nMoveID</i>	Returns the ID number with which the movement is stored on the motion stack.
<i>x_nError</i>	Error code.
= 0	: No Error.
= 1	: There is no object marked as "read" to synchronize on. This error is usually returned if program <i>getNextID()</i> has not been called previously.
= 2	: Wrong sequence of commanded motion. This error code is returned, for instance, if the program is called twice.
= 3	: Wrong conveyor number.

- public moveL(point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)

This program commands a linear motion that is synchronized with a location moving with a conveyor.

Input parameters

<i>x_pLoc</i>	Location on the conveyor to move to.
<i>x_tTool</i>	Tool to use for the motion.
<i>x_mDesc</i>	Motion descriptor to use for the motion.

Output parameters

<i>x_nMoveID</i>	Returns the ID number with which the movement is stored on the motion stack.
<i>x_nError</i>	Error code.
= 0	: No Error
= 1	: There is no object marked as "read" to perform a motion to.
= 2	: Wrong sequence of commanded motion. This error code is returned, for instance, if the program is called before the program TrackOn

- public moveC(point x_pInter, point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)

This program commands a circular motion that is synchronized with a location moving with a conveyor.

Input parameters

<i>x_pInter</i>	Intermediate location to perform a circular motion.
<i>x_pLoc</i>	Location on the conveyor to move to.
<i>x_tTool</i>	Tool to use for the motion.
<i>x_mDesc</i>	Motion descriptor to use for the motion.

Output parameters

<i>x_nMoveID</i>	Returns the ID number with which the movement is stored on the motion stack.
<i>x_nError</i>	Error code. = 0 : No Error. = 1 : There is no object marked as "read" to perform a motion to. = 2 : Wrong sequence of commanded motion. This error code is returned, for instance, if the program is called before the program TrackOn.

- public trackOff(point x_pLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)

This program commands a motion to desynchronize smoothly to a fixed cartesian location. The program waits that the motion is finished before the program interpretation continues.

Input parameters

<i>x_pLoc</i>	Fix cartesian location to move to.
<i>x_tTool</i>	Tool to use for the motion.
<i>x_mDesc</i>	Motion descriptor to use for the motion.

Output parameters

<i>x_nMoveID</i>	Returns the ID number with which the movement is stored on the motion stack.
<i>x_nError</i>	Error code. = 0 : No Error. = 1 : There is no object marked as "read" to desynchronize from. = 2 : Wrong sequence of commanded motion. This error code is returned, for instance, if the program is called before the program TrackOn.

- public trackOffJ(joint x_jLoc, tool x_tTool, mdesc x_mDesc, num& x_nMoveID, num& x_nError)

This program commands a motion to desynchronize smoothly to a fixed joint location. The program waits that the motion is finished before the program interpretation continues.

Input parameters

<i>x_jLoc</i>	Fix joint location to move to.
<i>x_tTool</i>	Tool to use for the motion.
<i>x_mDesc</i>	Motion descriptor to use for the motion.

Output parameters

<i>x_nMoveID</i>	Returns the ID number with which the movement is stored on the motion stack.
<i>x_nError</i>	Error code. = 0 : No Error. = 1 : There is no object marked as "read" to desynchronize from. = 2 : Wrong sequence of commanded motion. This error code is returned, for instance, if the program is called before the program TrackOn.

- public trackStopMove()

This instruction resets the motion stack, but keeps the synchronization with the conveyor.

- If called during a synchronized move (\$moveIt, \$moveCt or \$trackOn ended on waitEndMove), it stops the relative motion, keeps the synchronization and removes all the instructions from the motion stack. The following instruction must be either \$moveIt, or \$moveCt or \$trackOn or \$trackOff.
- If called in another context (\$trackOn before its end, moveJ, moveL, \$trackOff), it has the same behavior than resetMotion().
- If for any reason, the synchronisation is lost, it has the same effect than resetMotion(). Since the synchronization was lost, it's not possible to push directly a \$moveIt or a \$moveCt after it.
- To know if the synchronization is still active, you can use the function \$trackingState: it must return 2 or 3.
- \$trackOn = mot:trackOn() or mot:trackOnJ()
- \$moveIt = mot:moveL()
- \$moveCt = mot:moveC()
- \$trackOff = mot:trackOff() or mot:trackOffJ()
- \$trackingState = mot:getTrackStatus()

- public stopTrkMove()

This program stops the motion relative to the conveyor.

- If this program is called while the robot is executing a synchronized movement \$moveIt or \$moveCt, the robot will stop its trajectory but it will stay synchronized to the conveyor. Execute a restartMove() instruction to continue the synchronized movement.
- If this program is called while the robot is executing a synchronization movement \$trackOn, the robot will stop without synchronization to the conveyor. Executing a restartMove() will result on a \$trackOn() to the point over the conveyor.
- If this program is called while the robot is executing a de-synchronization movement \$trackOff, the robot will stop without synchronization to the conveyor. Executing a restartMove() will result on a moveJ() to the fix point.
- If this program is called while the robot is executing a standard move, the behavior of the arm will be exactly the same as with the instruction stopMove().
- \$trackOn = mot:trackOn() or mot:trackOnJ()
- \$moveIt = mot:moveL()
- \$moveCt = mot:moveC()
- \$trackOff = mot:trackOff() or mot:trackOffJ()

- public disableQueue(num x_nConvNum)

When this program is called, the queuing of detected objects is suspended until the program *enableQueue()* is called. But you must call the program *flushQueue()* before *enableQueue()* to flush the queue.

Input parameters

x_nConvNum Index of the conveyor in the conveyor list. The list of conveyors known by the Motion library is the one passed as parameter to the program *create()*.

- public enableQueue(num x_nConvNum)

When this program is called, the objects detected by the [Detection library](#) are pushed into the queue of the conveyor library. The queuing of the objects is enabled by default when the motion library is started by using the public program *create()*. This features remains active until the program *disableQueue()* is called. Before calling *enableQueue()* you must use the program *flushQueue()*.

Input parameters

x_nConvNum Index of the conveyor in the conveyor list. The list of conveyors known by the Motion library is the one passed as parameter to the program *create()*.

- public flushQueue(num x_nConvNum, num& x_nError)

This program flushes the queue of objects.

Input parameters

x_nConvNum Index of the conveyor in the conveyor list. The list of conveyors known by the Motion library is the one passed as parameter to the program *create()*.

Output parameters

x_nError Error code. 0 (zero) is returned if no error happened. You can get a clear text about the error by calling the program *getErrorMessage()*.

- public getConvFrame(num x_nConvNum, frame& x_fConveyor, frame x_fRefFrame, num& x_nError)

This program returns the position and orientation of the conveyor frame in reference to the frame *x_fRefFrame*.

Input parameters

x_nConvNum Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program *create()*.

x_fRefFrame Reference frame from which the coordinates of *x_fConveyor* should be returned.

Output parameters

x_fConveyor Coordinates of the conveyor's frame that have been taught during calibration.
x_nError Error code. 0 (zero) is returned if no error happened. You can get a clear text about the error by calling the program *getErrorMessage()*.

- public getConvSpeed(num x_nConvNum, num& x_nSpeed, num& x_nError)

This program returns the speed of the conveyor in mm/sec.

Input parameters

x_nConvNum Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program *create()*.

x_nSpeed speed's conveyor (mm/sec).

Output parameters

x_nError Error code. 0 (zero) is returned if no error happened. You can get a clear text about the error by calling the program *getErrorMessage()*.

- public getDetectFrame(num x_nConvNum, frame& x_fDetection, frame x_fRefFrame, num& x_nError)

This program returns the position and orientation of the detection device frame in reference to the frame *x_fRefFrame*.

Input parameters

x_nConvNum Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program *create()*.

x_fRefFrame Reference frame from which the coordinates of *x_fDetection* should be returned.

Output parameters

x_fDetection Coordinates of the detection device frame that have been taught during calibration.
x_nError Error code. 0 (zero) is returned if no error happened. You can get a clear text about the error by calling the program *getErrorMessage()*.

- public getDistance(num x_nConvNum, point x_pLoc, num x_nTime, num& x_Dist2Up, num& x_nDist2Down, num& x_nError)

This program returns the distance between the first object in the queue and the upstream/downstream limit of the tracking window. A negative value means that the object has not reached the considered limit of the tracking window. Once the object has passed the considered limit of the tracking window, the value of the returned parameters x_nDist2Up/ x_nDist2Down is positive.

Input parameters

<i>x_nConvNum</i>	Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program <i>create()</i> .
<i>x_pLoc</i>	Original position of the first object in the queue. It is the position given by the detection device or the position taught with the program <i>teach()</i> .
<i>x_nTime</i>	Time to look ahead to evaluate position of the object.

Output parameters

<i>x_nDist2Up</i>	Distance between location pLoc passed as parameter and the upstream limit of the tracking window. The value is positive if the object is downstream of the upstream limit of the tracking window.
<i>x_nDist2Down</i>	Distance between location pLoc passed as parameter and downstream end of the tracking window. The value is positive if the object is downstream of the downstream limit of the tracking window.
<i>x_nError</i>	Error code. = 0 : No error. = 1 : No object with "read" status. This happens when program <i>getNextID()</i> has not been called. = 2 : The object with current ID is not calibrated. The program <i>teach()</i> has not been performed for this object ID.

- public getErrorMessage(string& x_sMessage)

This program returns a clear text about the last error code.

Output parameters

<i>x_sMessage</i>	Clear text about last error code.
-------------------	-----------------------------------

- public getNextID(num x_nConvNum, num& x_nNextID)

This program returns the type (Identity) of the first object in the queue. A negative value means that there is no part in the queue. The positive value returned by this program is the one that have been pushed into the queue by the [Detection library](#). This object gets the "read" status and is considered as the object to which all motions will be relative to until it has been removed from the queue.

Input parameters

<i>x_nConvNum</i>	Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program <i>create()</i> .
-------------------	---

Output parameters

<i>x_nNextID</i>	Identity of the first object in the conveyor's queue.
------------------	---

- public getObjectData(num x_nConvNum, trsf& x_trObject, num& x_nError)

This program returns the position and orientation of the object that have been pushed into the queue by the [Detection library](#).

Input parameters

x_nConvNum Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program *create()*.

Output parameters

x_trObjectPos Position and orientation of the object.

x_nError Error code

= 0 : No error.

= 1 : No object with "read" status. This happens when program *getNextID()* has not been called.

- public getObjectLatch (num x_nConvNum, num& x_nLatchedValue, num& x_nError)

This program returns the encoder latched value of the object that have been pushed into the queue by the [Encoder library](#).

Input parameters

x_nConvNum Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program *create()*.

Output parameters

x_nLatchedValue Encoder latched value of the object.

x_nError Error code

= 0 : No error.

= 1 : No object with "read" status. This happens when program *getNextID()* has not been called.

- public getRobotArea(num x_nConvNum, num& x_nAreaNumber)

This program returns an integer value representing the state of the robot position during tracking: 1, 2 or 3.

Input parameters

x_nConvNum Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program *create()*.

Output parameters

x_nAreaNumber Refer to [chapter 4.2.4](#) to setup the conveyor library. See topic conveyor setup

- public getStatus(num x_nConvNum, num& x_nStatus)

This program returns an integer value representing the current state of the VAL3 libraries.

Input parameters

x_nConvNum Number of the conveyor from of which the status should be read.

Output parameters

x_nStatus Refer to [chapter 5.5](#) for detailed description.

- public getTrackStatus (num& x_nTrackStatus)

This instruction returns the state of the last tracking move command.

Output parameters

x_nTrackState state of the last tracking move command.
= 0 : there is no pending tracking move command.
= 1 : a trackOn command is being executed.
= 2 : a tracking move command (linear or circular) is being executed.
= 3 : the arm is tracking a fixed point moving with the conveyor.
= -1 : the tracking is in error.



Warning

This instruction gives a runtime error if used while tracking initialization. This instruction should not be used before execution of the program *create()* and after the program *kill()*.

- public getVersion(string& x_sVersion)

This program returns the VALTrack software version.

Output parameters

x_sVersion VALTrack software version.

- public kill()

This program stops all tracking tasks. (Tasks 2 to 5 in annex 1)

- public removePart(num x_nConvNum)

This program deletes the first object in the queue. The part will not be deleted until all command tracking movements associated to this part have been executed by the robot arm.

Input parameters

x_nConvNum Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program *create()*.

- public reset(num x_nConvNum, num& x_nError)

This program resets the encoder library.(resetting the VAL3 counter, resetting the digital output cRstErr) and flushes the queue.

Input parameters

x_nConvNum Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program *create()*.

Output parameters

x_nError Error code. 0 (zero) is returned if no error happened. You can get a clear text about the error by calling the program *getErrorMessage()*

- public sendToNextRobot (sio x_sSocketClient, num x_nTimeout, num x_nID, trsf x_trObjectPos, num x_nLatchedValue, num& x_nError)

This program sends a telegram via a TCP/IP socket client. The telegram contains all the information of a detected object, this information is meant to be used by the detection library *Socket* of a second robot.

Input parameters

<i>x_sSocketClient</i>	Socket client used to send the message.
<i>x_nTimeout</i>	Timeout for the socket client to read/write one character.
<i>x_nID</i>	ID of the detected object.
<i>x_trObjectPos</i>	Transformation of the detected object.
<i>x_nEncLatchVal</i>	Latched encoder value of the detected object.

Output parameters

<i>x_nError</i>	Error code:
= 0	: No error, message sent.
= 1	: The socket sent as a parameter is not working. The sio variable has a bad link or the device is in error.
= 2	: Message not sent. Timeout expired. The other robot is not present on the network.
= 3	: Timeout expired. The other robot is not working with the detection library <i>Socket</i> .
= 4	: Message sent but the server from the other robot didn't acknowledge reception before expiration of timeout.

- public setCalibrating(num x_nConvNum, bool x_bMode)

This program sets the detection library *Socket* in mode calibration. When the library isn't in calibration mode, the latched encoder value is given by detection library *Socket*. When the library is in calibration mode, the latched encoder value is given by the encoder library. The detection library *Socket* must be set in calibration mode when the detection of the parts is done with a simple detection device such as an on/off sensor and during calibration of the distance in between the sensor and the pick/place position on the conveyor.

This is also valid for any other detection library which name begins with *Socket*. For any other detection library, the latched encoder value is given by the encoder library and this program has not use.

Input parameters

<i>x_nConvNum</i>	Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program <i>create()</i> .
<i>x_bMode</i>	Defines if the conveyor library gets the latched encoder value from the detection library or from the encoder library.
= true	: Any detection library which name begins with <i>Socket</i> is re-launched in calibration mode.
= false	: Any detection library which name begins with <i>Socket</i> is set in normal operation mode.

- public teach(num x_nConvNum, tool x_tTool, frame x_fFrame, x_point& x_pLoc, num& x_nError)

This program returns the coordinates of a point, in the specified frame, with respect to the instantaneous location of the conveyor. It is usually used when the detection device doesn't provide all necessary information about the position of the object on the conveyor like an on/off sensor. A specific procedure must be performed to teach a location that way. Refer to [chapter 5.2](#) for further information.

Input parameters

<i>x_nConvNum</i>	Index of the conveyor in the list. The list of conveyor known by the Motion library is the one passed as parameter to the program <i>create()</i> .
<i>x_tTool</i>	Tool Center Point with which the location must be taught.
<i>x_fFrame</i>	The reference frame of the location to teach.

Output parameters

<i>x_pLoc</i>	The location to teach
<i>x_nError</i>	Error code:
= 0	: No error
=-1	: No object detected, not possible to teach anything.

4.5 Speed Control While Tracking

4.5.1 Synchronization / De-synchronization Moves

The meaning of the parameters of the mdesc used for a synchronization / de-synchronization move (trackOn / trackOff) is the same as the parameters of the mdesc used for a standard movement without tracking: Accel and Decel will define the acceleration ($^{\circ}/\text{sec}^2$ or mm/sec^2) of the axes and Vel will define the maximum speed of the axes ($^{\circ}/\text{sec}$ or mm/sec). The cartesian parameters Tvel and Rvel are not used. In manual mode, the 250mm/sec speed limit at the TCP is respected.

mdesc Commands:

- Accel = Percentage of the nominal acceleration of all axes, $^{\circ}/\text{sec}^2$ or mm/sec^2 .
- Decel = Percentage of the nominal deceleration of all axes, $^{\circ}/\text{sec}^2$ or mm/sec^2 .

mdesc Limits:

- Vel = Percentage of the nominal speed of all axes, $^{\circ}/\text{sec}$ or mm/sec .
- Tvel = NOT USED
- Rvel = NOT USED

4.5.2 Synchronized Moves

The meaning of the parameters of the mdesc used for a synchronized move (moveL / moveC) is NOT the same as the parameters of the mdesc used for a standard movement without tracking. Accel, Decel and Vel are now a percentage of the nominal cartesian values defined for each robot. Accel and Decel will define the cartesian acceleration (mm/sec^2 and $^{\circ}/\text{sec}^2$) of the TCP and the smallest of Vel, Tvel or Rvel will define the maximum speed, in reference to the conveyor, reached by the TCP ($^{\circ}/\text{sec}$ or mm/sec). In manual mode, the 250mm/sec speed limit at the TCP is respected.

mdesc Commands:

- Accel = Percentage of the nominal cartesian acceleration of the TCP in reference to the conveyor speed, mm/sec^2 and $^{\circ}/\text{sec}^2$.
- Decel = Percentage of the nominal cartesian deceleration of the TCP in reference to the conveyor speed, mm/sec^2 and $^{\circ}/\text{sec}^2$.

mdesc Limits:

- Vel = Percentage of the nominal cartesian speed of the TCP in reference to the conveyor speed, mm/sec and $^{\circ}/\text{sec}$.
- Tvel = Cartesian translation speed of the TCP in reference to the conveyor speed, mm/sec (x, y, z).
- Rvel = Cartesian rotation speed of the TCP in reference to the conveyor speed, $^{\circ}/\text{s}$ (rx, ry, rz).

The nominal speed of the TCP in reference to the conveyor speed is different for each robot. If this parameter should be changed, go the *init()* program of the Motion library and uncomment the instruction `$setCartTrkLim(string sParamName, num nValue)`

- void \$setCartTrkLim(string sParamName, num nValue)

This instruction sets nominal cartesian parameters for tracking moves. These nominal parameters are those used by the motion descriptor for accel, vel and decel (100%).

Input parameters

`sParamName` "tvel", "rvel", "tacc", "racc", "tdec" or "rdec".

nValue value of the parameter, in user units.

5 FREQUENTLY ASKED QUESTIONS

5.1 How to Create an Encoder Library?

If none of the already existing encoder libraries is suitable for your encoder module, a new encoder library should be created. The new encoder library must have the same interface as the default encoder library, therefore the new encoder library should be a copy of the *GenericEnc* library.

1. Open the library *GenericEnc* under the directory Tracking/Templates/.
2. Save it as the name you like under the directory Tracking/Encoders/.

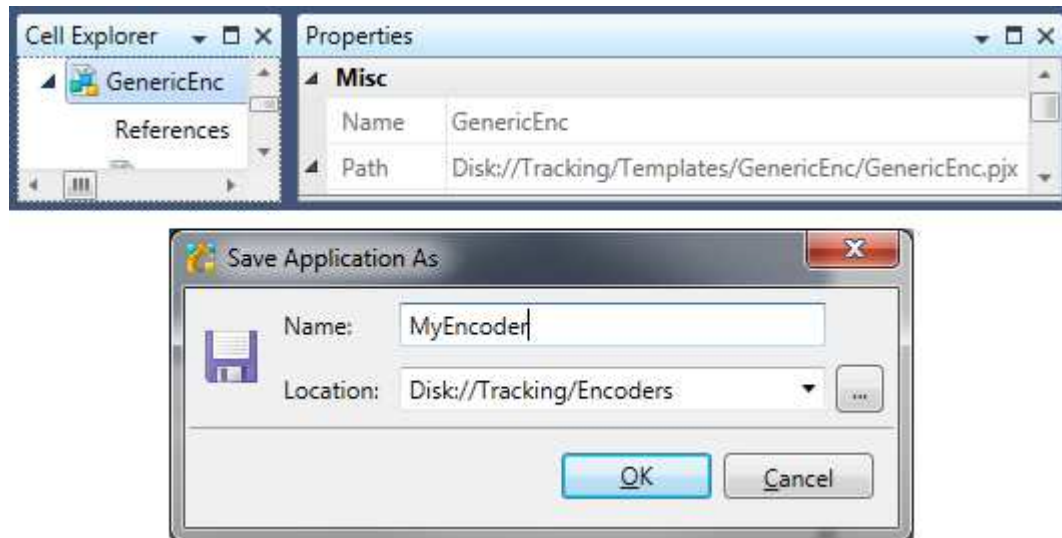


Figure 5.1

3. Modify the program *encoder* according to the needs of your encoder module.

Note: Do not modify the library Tracking/Templates/GenericDetect. Any modification to the interface (number and names of public programs and variables) of this library will have to be copied to all detection libraries.

5.1.1 Variable description

This section describes the list of the variables used in an Encoder library.

- num nLatchPeriod

This variable represents the number of triggers that must occur on the position capture input before an object is considered as detected. Usually this variable is set to 1; however the detection device may rather detect a carrier than the object itself. In these cases, this variable is set to a different value as shown in diagram below.

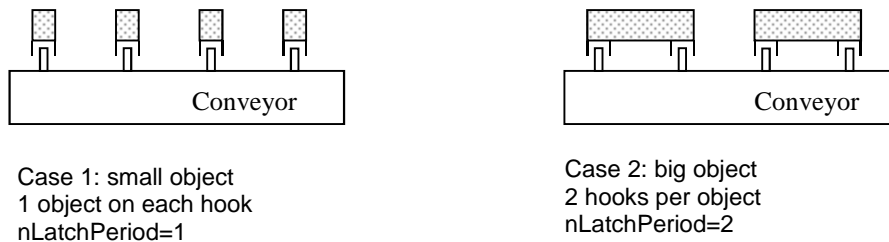


Figure 5.2

- num nEncValue

This variable represents the instantaneous position of the conveyor. Since the conveyor moves always in the same direction, this value must increase continuously.

- num nEncLatchedVal

This variable represents the position of the conveyor when the input defined for the position capture feature has been triggered.

- dio diEncLatched

This variable has to be linked to the physical Input used for the position capture feature.

- aio aiEncCurrentPos

This variable has to be linked to the physical Input that gives the instantaneous encoder position.

- aio aiEncLatchedPos

This variable has to be linked to the physical Input that gives the latched encoder position.

5.1.2 Program description

This section lists the programs that may be modified to adapt the template to your needs.

- public create(num x_nEncNum, string x_sTask)

This program initializes the variables of the library and starts the program called encoder in a task called encoder. It is called once by the [Conveyor Library](#).

- private init(num x_nEncNum, string x_sTask)

This program is used to link the Inputs/Outputs, defined for the encoder device, to the variables used in the library. The Init program is called once when the Encoder library is started.

- public getValue(bool x_bUseLatch, num& x_nEncValue)

This program returns either the instantaneous encoder position through the variable [nEncValue](#) or the latched encoder value through [nEncLatchedVal](#). This program is called by the [Conveyor Library](#).

- public isLatched (bool& x_bLatched)

This program informs if the digital signal used as position capture has been triggered. This program is continuously polled by the [Conveyor Library](#)

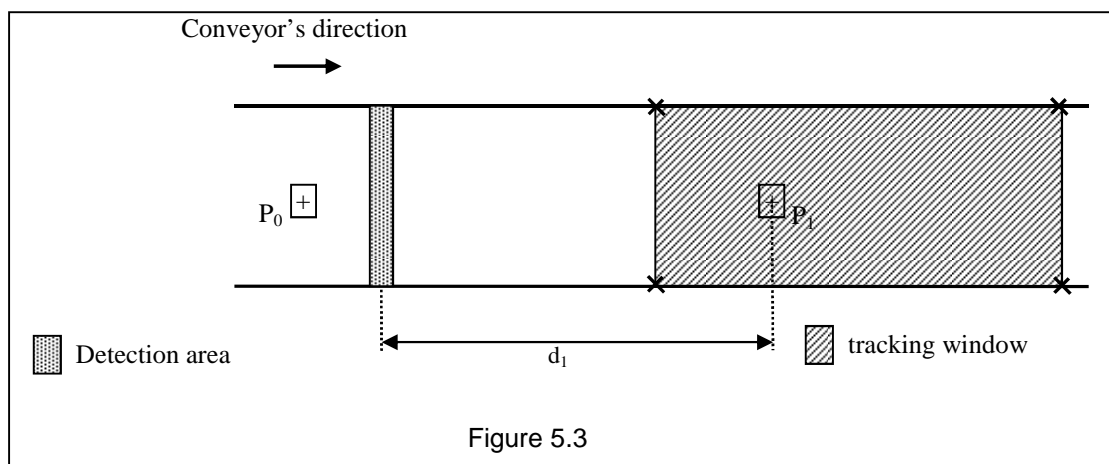
- private encoder()

This is the main program of an Encoder library. It should manage both the encoder signals to update the variable [nEncValue](#) and the trigger on the position capture digital input to setup the variable [nEncLatchedVal](#).

5.2 How to Define Locations Relative to a Conveyor?

A conveyor-relative location can be thought as a fix transformation defined in a frame which has a time-varying component. The transformation is defined as standard POINT type in VAL3 language where as the time-varying component is managed by the [Conveyor library](#). Thus, the teaching of a location relative to a conveyor requires a specific procedure that must use the *teach* function of the [Conveyor library](#). This function not only returns the coordinates of the location but memorizes as well the distance (d_1) that the object has travel from the detection device. This procedure must be performed for each different object type. This section described the procedure to follow.

Place object at location P_0 , upstream from the detection device
 Make sure both Detection and Encoder libraries are started
 Start conveyor until the object has reached the location P_1 in the tracking window
 Stop the conveyor
 Jog the robot gripper to the object
 Record the location using the program *teach* from the [Conveyor library](#)



To perform this procedure, you can use the sample program given hereafter.

```
begin
  userPage()
  cls()
  call up:type(0,0,"-Place the object to be detected on the")
  call up:type(0,1,"conveyor upstream of the sensor.")
  call up:type(0,6,"Press RETURN to proceed")
  wait(getKey()==270)
  // release Return Key
  wait(getKey()!=270)
  //
  cls()
  call up:type(0,0,"-Start your conveyor")
  call up:type(0,2,"Waiting for object detection...")
  call up:type(0,3,"Object ID: ")
  // Search ID 0 in the tracking stack
  do
    call mot:getNextID(0,l_nID)
    call up:type(8,3,toString("2",l_nID))
    delay(0)
  until (l_nID!=-1) // Ready to teach
  cls()
  call up:type(0,3,"You can teach your locations")
  call up:type(0,5,"Press RETURN to proceed")
  wait((getKey()==270))
  call mot:teach(0,tPointer,world,pPoint,l_nError)
  call mot:removePart(0)
end
```

5.3 How Objects Are Pushed Into the Conveyor's Queue?

This section describes the mechanism used to fill in the conveyor's queue with the objects that have been detected. The figure 5.4 shows the organization of the different libraries. The first goal of the [Conveyor library](#) is to link the objects that have been detected by the Detection library, with the position of the conveyor given by the Encoder library.

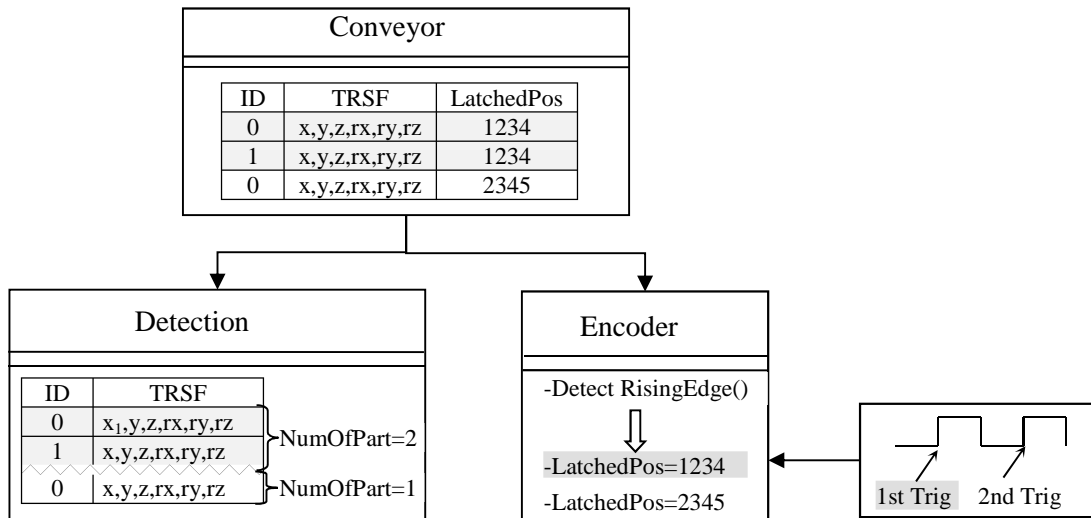


Figure 5.4

The Conveyor library has one task running. This task is polling the Encoder library until a trigger for the position capture has occurred. When this happens, the Detection library must provide the information (number, ID, coordinates) concerning the detected objects before the next trigger occurs, otherwise these objects will not be queued.

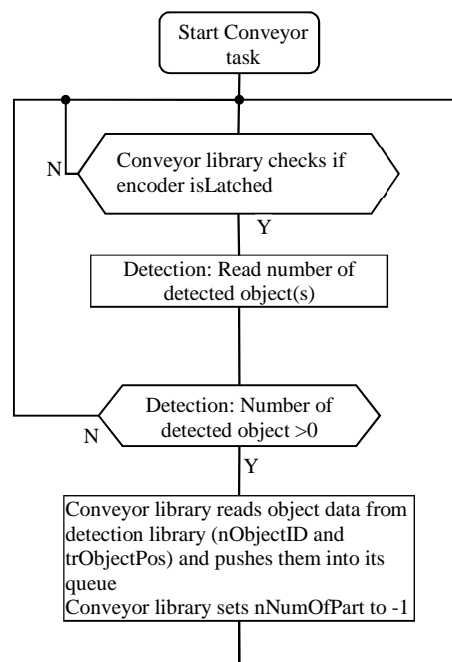


Figure 5.5

5.4 How Does the Capture of the Encoder Position Works?

The time diagram in figure 5.6 shows the state of the various input/output signals that are involved in the position capture of the encoder position. Depending on the hardware, the name of the signals may be different as described in the following table:

Hardware type	Trigger signal is :	Latch input is:
STARC2 board 1 channel 0	e00LatchSig	e00Latch
STARC2 board 1 channel 1	e01LatchSig	e01Latch
STARC2 board 2 channel 0	e10LatchSig	e10Latch
STARC2 board 2 channel 1	e11LatchSig	e11Latch

The *Trigger signal* is the input that triggers the high speed position capture of the encoder input.

The *Latch input* is the input that is set when an encoder position has been captured.

The *Enable Latch* signal is the output that allows enabling the high speed position capture feature. It is reset as soon as an encoder position is captured (*Trigger signal* is high) and it is set under the following conditions.

- Once a position is captured, the conveyor has travel the distance specified under SETTINGS→CONV→LockDist in the Wizard application if this value is non zero.

Or

- 100 milliseconds after an encoder position has been captured if the value specified under SETTINGS→CONV→LockDist is zero.

The situations marked ①, ③ and ⑤ are normal cases in which an object is detected and an encoder position is captured.

The situation marked ② is a case in which the input has been triggered but the encoder position is not captured. This happens because the conveyor has not travel the required distance since the previous object detection. This object will then not be pushed in the conveyor's queue.

The situation marked ④ is a case in which an encoder position is captured whereas this is not object detection. This happens with systems that capture encoder position on both rising and falling edges of the trigger signal. In the figure 5.6, the state of the trigger signal remained HIGH longer than the distance during which the position capture feature is disabled. To avoid that situation, you must increase the value of the LockDist parameter.

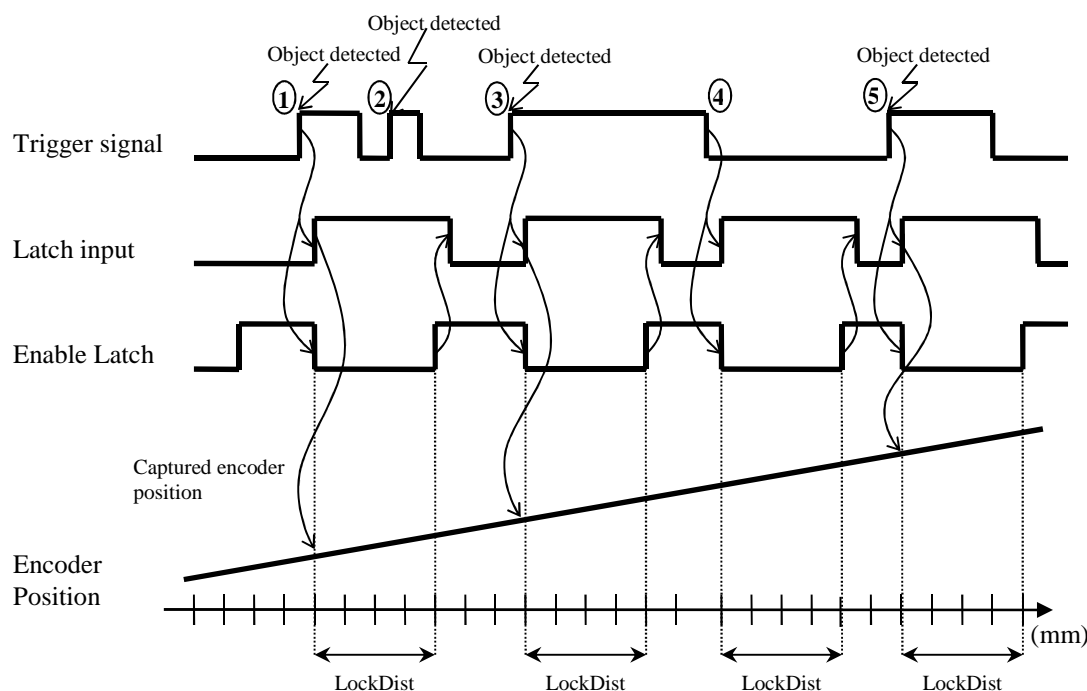


Figure 5.6

5.5 How to Get the Status of the Libraries?

The VALTRACK package runs 4 tasks which could report some standard VAL3 error code. The tasks are identified as followed:

Task ID	Task description
2000	Conveyor
3000	Encoder
4000	Detection
5000	Motion

The program *getStatus()* of the Motion library returns a positive value representing the status of all the libraries included in the VALTRACK package. The meaning of these values is given in the table below:

Status code	Description
W A R N I N G S	
1	The detection library attempted to push an object with negative ID into the conveyor's queue
2	Detection overrun. A new trigger for object detection occurs before the detection library delivers the data of the previous object(s).
V A L T R A C K E R R O R C O D E S	
101	A motion to a location that is out of reach has been commanded. The robot cannot execute the commanded motion because the motion descriptor is not valid (commanded accel/decel or speed are usually too low). An Internal error of the conveyor tracking algorithm. Check error logger and contact Stäubli customer support. (Cf note[1])
V A L T R A C K S T A T U S C O D E S	
-1	Not initialized. Check that you have successfully called the program <i>create()</i> of the motion library.
0	Running. No tracking move command pending.
201	A trackOn command is being executed.
202	A tracking move command (linear or circular) is being executed.
203	The arm is tracking a fixed point on a conveyor.
R U N T I M E E R R O R S	
TaskID + 0	No execution error
TaskID + 1	A task is running
TaskID + 10	Invalid numerical calculation (division by zero).
TaskID + 11	Invalid numerical calculation (e.g. ln(-1))
TaskID + 20	Access to an array with an index that is larger than the array size.
TaskID + 21	Access to an array with a negative index.
TaskID + 29	Invalid task name. See taskCreate instruction.
TaskID + 30	The specified name does not correspond to any VAL3 task.
TaskID + 31	A task with the same name already exists. See taskCreate instruction.
TaskID + 32	Only 2 different periods for synchronous tasks are supported. Change scheduling period.
TaskID + 40	Not enough memory space available.
TaskID + 41	Not enough memory space to run the task. See the run memory size.
TaskID + 60	Maximum instruction run time exceeded.
TaskID + 61	Internal VAL3 interpreter error
TaskID + 70	Invalid instruction parameter. See the corresponding instruction.
TaskID + 80	Uses data or a program from a library not loaded in the memory.
TaskID + 81	Incompatible kinematics: Use of a point/joint/config that is not compatible with the arm kinematics.
TaskID + 82	The reference frame or tool of a variable belongs to a library and is not accessible from the variable's scope (library not declared in the variable's project, or reference variable is

	private).
TaskID + 90	The task cannot resume from the location specified. See taskResume() instruction.
TaskID + 100	The speed specified in the motion descriptor is invalid (negative or too great).
TaskID + 101	The acceleration specified in the motion descriptor is invalid (negative or too great).
TaskID + 102	The deceleration specified in the motion descriptor is invalid (negative or too great).
TaskID + 103	The translation velocity specified in the motion descriptor is invalid (negative or too great).
TaskID + 104	The rotation velocity specified in the motion descriptor is invalid (negative or too great).
TaskID + 105	The reach parameter specified in the movement descriptor is invalid (negative).
TaskID + 106	The leave parameter specified in the movement descriptor is invalid (negative).
TaskID + 122	Attempt to write in a system input.
TaskID + 123	Use of a dio, aio or sio input/output not connected to a system input/output.
TaskID + 124	Attempt to access a protected system input/output
TaskID + 125	Read or write error on a dio, aio or sio (field bus error)
TaskID + 150	Cannot run this movement instruction: a previous movement request could not be completed (point out of reach, singularity, configuration problem, etc.)
TaskID + 153	Movement command not supported
TaskID + 154	Invalid movement instruction: check the movement descriptor.
TaskID + 160	Invalid flange tool coordinates
TaskID + 161	Invalid world tool coordinates
TaskID + 162	Use of a point without a reference frame. See Definition.
TaskID + 163	Use of a frame without a reference frame. See Definition.
TaskID + 164	Use of a tool without reference tool. See Definition.
TaskID + 165	Invalid frame or reference tool (global variable linked to a local variable)
TaskID + 250	No runtime license for this instruction, or demo license is over.
TaskID + 999	There is no task running for this library

5.6 How to Use the VAL3 Library Cognex1 Provided with VALTrack Software?

The VALTrack software package is delivered with a detection library that could be used as a base to communicate with a Cognex Insight® Camera. This section gives a brief description about the use of this library.

5.6.1 Configuration

The VAL3 library called *Cognex1* uses the Insight® Native Mode protocol to control the camera. This protocol is based on TCP/IP. As a precondition, a socket item must be defined on the robot side by performing the following operations:

From the main menu, select Control panel→I/O.

- Expand the node by pressing the right arrow.
- Select the *Socket* item and press New(F8) to create a new socket.
- Select Client as *socket Type* and press down arrow to edit the socket Name
- Enter **cognex1** as socket Name and press Ok (F8)
- In the *Parameters* window, setup the *Server address*. This address must match the IP address of the camera. You can find this address in the In-sight explorer software under menu Sensor→Network Settings.
- Configure the remaining parameters as shown in the figure 5.7

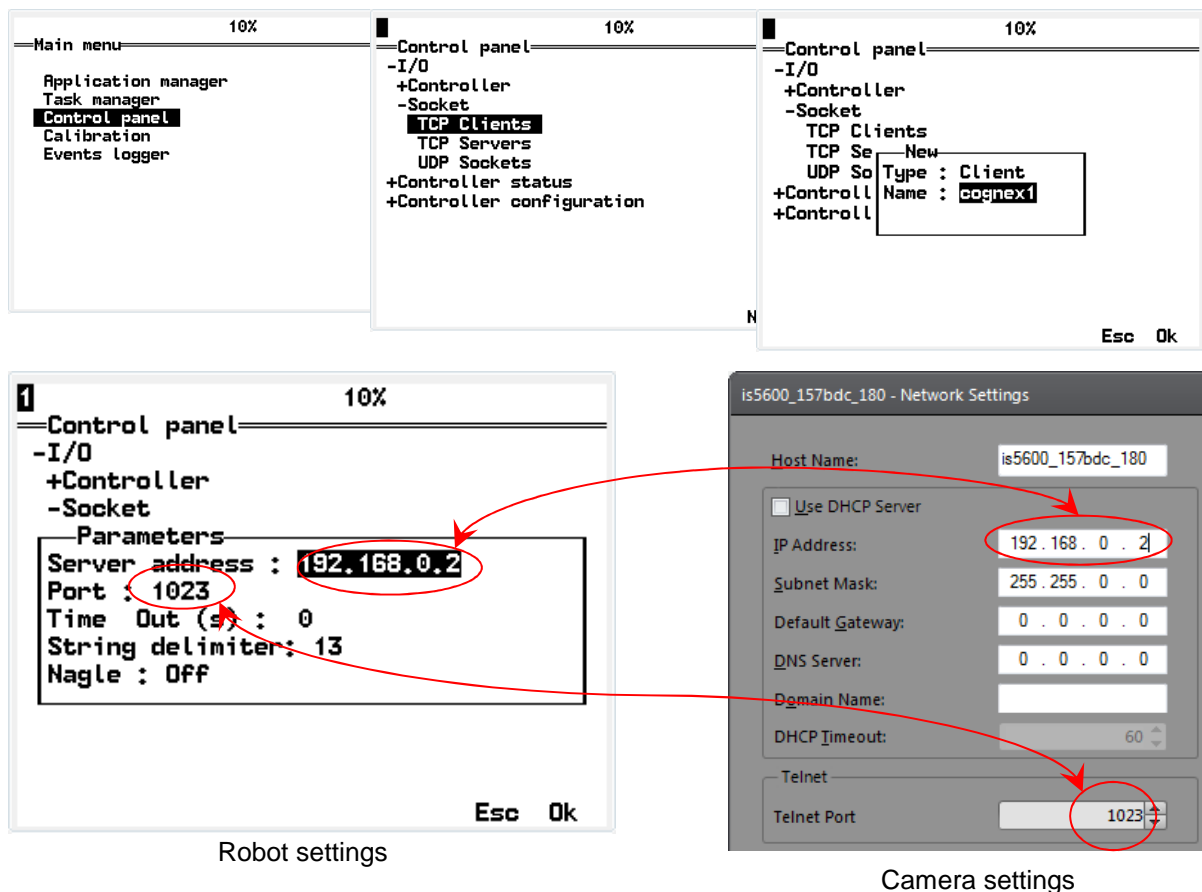


Figure 5.7



Warning

The default port number configured for the native mode protocol of the camera is 23 (Telnet). This is highly recommended to change this value because it could interact with the Telnet feature of the CS8 controller.



Warning

If there is no socket defined in the I/Os of the controller or if the name of the socket doesn't match to the name used in the VAL3 library Cognex, you will not be able to select this detection library from the Wizard application. In that case you will get the following error message:

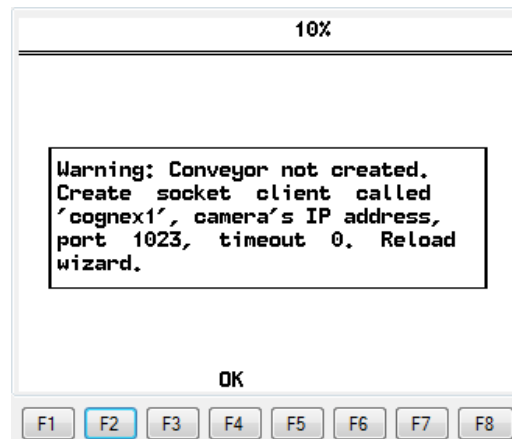


Figure 5.8

5.6.2 Description of the library

The application *Cognex1* is a detection library. This detection library has been extended with a set of programs to manage the communication in the native mode of a Cognex Insight® camera. These extra programs are collected in a library called *cognexComm* which is added to the detection library *Cognex1* and used with the identifier *cam* (figure 5.9).

You can refer to [chapter 4.1](#) for further information about a generic detection library.

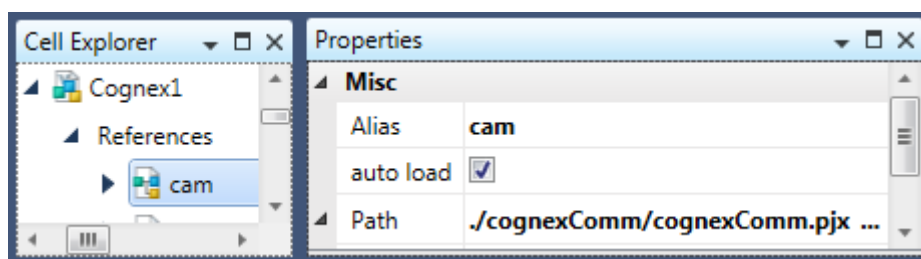


Figure 5.9

5.6.3 Program description

- public init()

This program initializes the library that handles the communication with the Cognex camera. It must be called once at the beginning

- public login(sio skSocketName, string sUserName, string sPassword, num& nHandle, bool& bError)

This program logs into the camera as user sUserName and password sPassword. Once logged in, this is possible to control the camera in native mode.

Parameters

<i>skName</i>	Name of the socket used to communicate with the camera. This socket must be defined as described in the section 5.6.1
<i>sUserName</i>	Name of the user to build up the TCP/IP connection (typically "admin")
<i>sPassword</i>	Password associated with the user name used previously (typically "")
<i>nHandle</i>	Handle number that will be used for further commands
<i>bError</i>	error code. If this variable is TRUE, the login was successful. Otherwise, you can use the program getErrorMessage to have a clear text about the error.

- public getOnline(num nHandle, bool& bError)

This program checks if the camera is Online or not.

Parameters

<i>nHandle</i>	Handle number returned by the login program
<i>bError</i>	error code. When this variable is set to TRUE, the camera is Online. Otherwise, the camera is Offline.

- public readNumValue(num nHandle, string sCol, num nRow, num& nValue, bool& bError)

This program reads the content of the cell located at sCol and nRow in the spreadsheet. This value must be numerical otherwise an error is returned.

Parameters

<i>nHandle</i>	Handle number returned by the login program
<i>nCol</i>	string value in range from 'A' to 'Z' to specify the column number of the cell to read in the spreadsheet
<i>nRow</i>	numerical value in range from '1' to '400' to specify the row of the cell to read in the spreadsheet
<i>nValue</i>	Content of the cell.
<i>bError</i>	This variable is TRUE if the cell contained a numerical value. Otherwise, you can use the program getErrorMessage to have a clear text about the error.

- public getErrorMessage(num nHandle, string& sErrMessage)

This program returns a clear text error message of the last operation that was executed.

Parameters

<i>nHandle</i>	Handle number returned by the login program.
<i>sErrMessage</i>	Clear text error message.

6 ANNEX

6.1 Schema of tasks

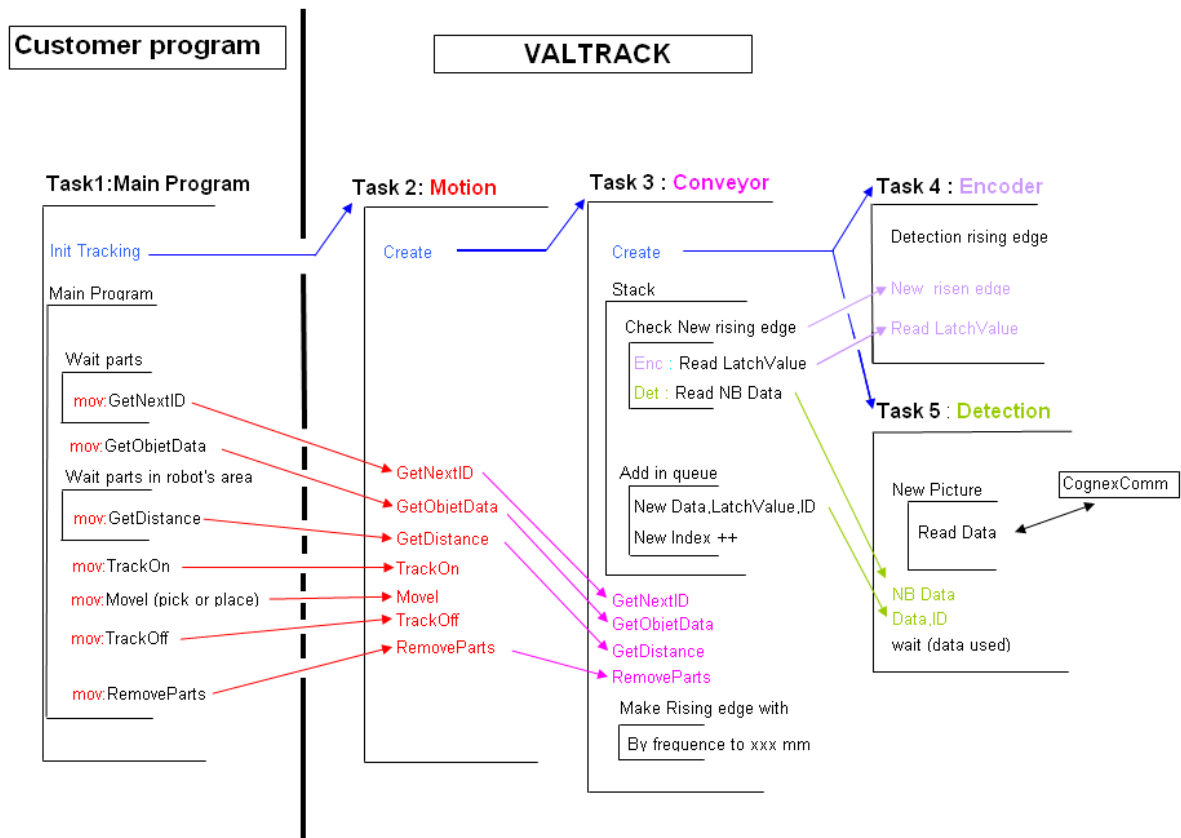


Figure 6.1

© Stäubli December 2015



6.3 Wiring

6.3.1 STARC 2 and Camera Cognex

Trigger output on rising edge, trigger input on falling edge:

- CS8C: Set encoder output e00LatchEdgFall = On

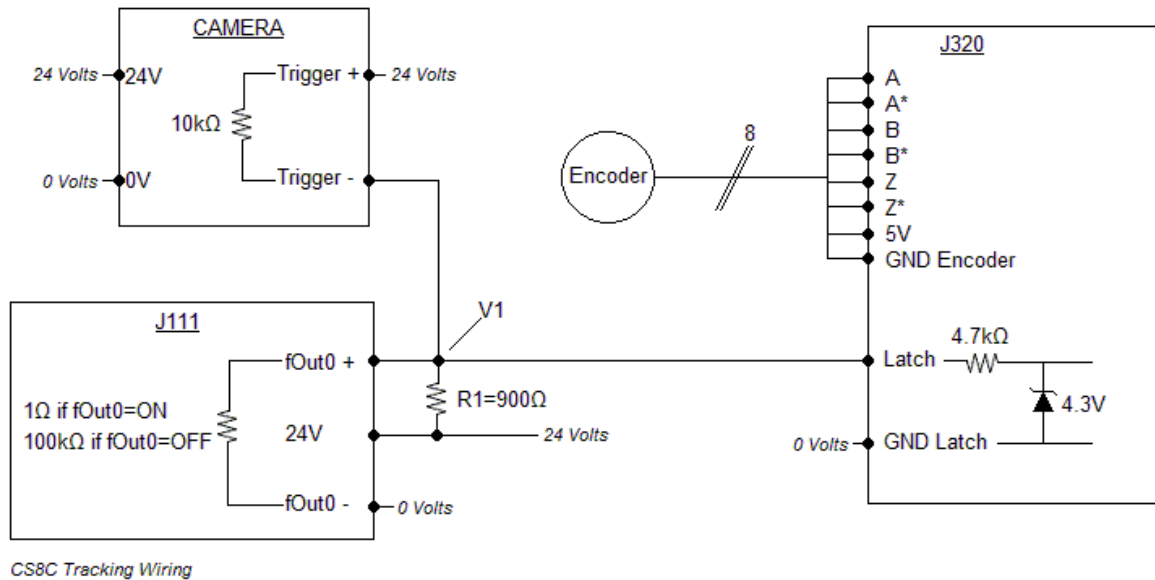


Figure 6.3

Multi-robot configuration

When multiple DUAL ABZ boards are triggered by the fast output of the same robot, the value of R1 changes in order to keep a minimum of 20V at V1 (figure 6.3). Then, the value of R1 is:

$$R1 = 900 / (\text{Num of DUAL ABZ boards}) \Omega$$
$$\text{Power of R1} = 0.639 * (\text{Num of DUAL ABZ boards}) W$$

The second use of R1 is to limit the current going through fOut0+ to a maximum of 250mAmps when the output is ON, therefore R1 cannot be smaller than 100Ω and a maximum of 9 DUAL ABZ boards can be triggered by the same fast output.