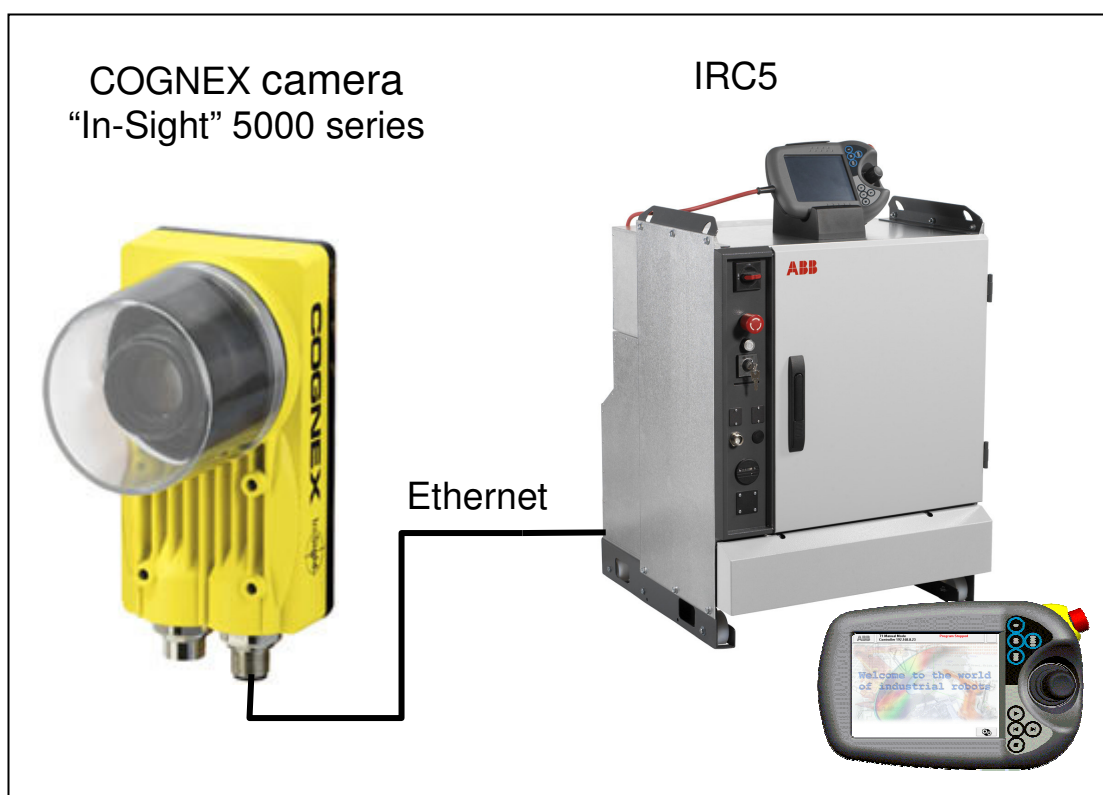




User Manual

COGNEX “In-Sight” vision system interface for IRC5 control system

IRC5



User Manual

COGNEX “In-Sight” vision interface for IRC5 control system

Document ID: RDNU0039_C_InterfaceVisionCOGNEX_ENG
Revision: C – Updated April 2009

The information contained in this document can be modified without prior warning and should not be considered as a commitment on the part of ABB France s.a.s. ABB France s.a.s refutes any responsibility pertaining to errors that may have occurred in this document.

Under no circumstances can ABB France s.a.s be held responsible for any incident that might result from the use of this document or the software and hardware described therein.

This document should not be reproduced, nor copied whether partly or in full, without prior permission from ABB France s.a.s. It should not be divulged to third parties, nor should it be used for any purposes other than those authorised. Any transgression will lead to the application of the legal proceedings laid down by the law in force.

Further copies of this document can be obtained from ABB France s.a.s at the current price.

© ABB France 2009

ABB France s.a.s – Robotics Division
Rue de l'Equerre
ZI des Béthunes
95310 Saint-Ouen l'Aumône
FRANCE

Contents

Overview	5
Product Documentation	7
1 Safety	8
1.1 About the Safety Section	8
1.2 Safety standards applicable to the IRC5 control system.....	9
1.3 Safety Terminology.....	10
2 Welcome	11
2.1 What is the COGNEX “In-Sight” vision interface for the IRC5 control system?	11
2.2 Description of terms and concepts	12
3 Software Organization	14
3.1 Generalities	14
3.2 Loading the software	15
4 Reference of the communication library instructions	16
4.1 General instructions for basic use	16
4.1.1 “CX_SetupCamera” instruction.....	17
4.1.2 “CX_InitComm” function.....	19
4.1.3 “CX_TriggerImage” function	20
4.1.4 “CX_GetVisionData” function	21
4.1.5 “CX_ShowErrStatus” instruction	24
4.1.6 “CX_CloseComm” function	25
4.1.7 “CX_SetLanguage” instruction.....	26
4.1.8 “CX_SetDebugMode” instruction	27
4.1.9 “CX_ShowVersion” instruction.....	28
4.2 Instructions to read and write data in the vision program.....	29
4.2.1 “CX_GetInfo” function.....	30
4.2.2 “CX_GetValue” function.....	32
4.2.3 “CX_SetFloat” function.....	34
4.2.4 “CX_SetInteger” function.....	36
4.2.5 “CX_SetString” function	38
4.3 Instructions to manage execution and “Online” mode.....	40
4.3.1 “CX_GetOnline” function	41
4.3.2 “CX_SendMessage” function	42
4.3.3 “CX_SetEvent” function.....	44
4.3.4 “CX_SetEventWait” function.....	45
4.3.5 “CX_SetOnline” function	46
4.4 File management instructions.....	47
4.4.1 “CX_GetFile” function	48
4.4.2 “CX_LoadFile” function.....	49
4.5 Robot-vision calibration instructions	50
4.5.1 “SetToolVision” instruction.....	51
4.5.2 “SetWObjVision” Instruction	53

5	Implementing the communication library instructions	55
5.1	Standard utilization sequences for library instructions.....	55
5.2	Using the vision transformation	57
5.3	Example programs provided with the communication library.....	61
5.4	IP address configuration for an IRC5 control system.....	62
5.5	IP address configuration for an “In-Sight” camera	65
6	Human-machine Interface “HMI”	66
6.1	Error messages generated by the ABB communication library.....	66
6.2	Error messages generated by the “In-Sight” camera.....	67
6.3	Information message generated by the communication library functions	68
6.4	“In-Sight” camera image display on “FlexPendant” unit.....	69
7	Robot-vision calibration procedures	70
7.1	Implementing a vision system for guiding a robot.....	70
7.2	Robot – vision calibration in the case of a fixed camera.....	71
7.3	Robot – vision calibration in the case of a camera mounted on the robot	75
8	Specific Procedures	76
8.1	Using the “Virtual Robot” to test a vision application	76
8.2	“Debug” mode.....	78
8.3	Image transfer from “In-Sight” camera to IRC5 control system.....	79
9	Release Notes	80
10	Index and tables	81
10.1	Index.....	81
10.2	Illustrations Table	81

Overview

About this manual

This manual offers detailed instructions for the implementation of the COGNEX “In-Sight” vision interface for an IRC5 control system with a “FlexPendant” unit.

This manual applies to version **1.0.3** of the interface.

Usage

This manual is intended to be used during programming and maintenance of a vision application involving COGNEX « In-Sight » cameras.

Who should read this manual?

This manual is intended for:

- Robot programmers,
 - Robot application developers,
 - Service technicians.
-

Pre-requisites:

The reader should:

- Be familiar with RAPID language,
 - Have been trained in the use of RobotStudio Online software and the "FlexPendant" unit,
 - Have been trained in the use of the COGNEX Company "In-Sight Explorer" software as well as in the operation of COGNEX "In-Sight" cameras.
-

References

Reference	Document ABB
Product Manual, procedures	IRC5 3HAC 021313-001
Product Manual, references	IRC5 3HAC 021313-001
Start up - IRC5 and RobotStudio Online	3HAC 021564-001
Operator manual - RobotStudio Online	3HAC 18236-1
RAPID Reference manual– RAPID Overview	3HAC 16580-1
RAPID Reference manual– Instructions	3HAC 16581-1
RAPID Reference manual– Data functions and types	3HAC 16581-1
RAPID Reference manual– RAPID Kernel	3HAC 16585-1

Reference	Document COGNEX
“In-Sight” Explorer 3.2 Reference Guide	Revision: 3.2.1219
“In-Sight” Explorer 4.1 Reference Guide & EasyBuilder	Revision: 4.1.0

How to read this manual?

To make it easier to read, text of specific types have specific formatting:

Type of text	Formatting	Example
User interface items such as menu, dialogue box, button	Bold	In the Open File dialogue box, click on OK
Keyboard keys	Capitals	For online help press F1.
Computer messages and RAPID instruction	Single space (mail)	MoveJ
Variables in RAPID Instruction messages	Italic single space (mail)	PROC <i>RoutineName</i> ()
Hypertext links:	Italic blue	See Manual RobotStudio Online Operator's Manual
References to other manuals or sources of information	Italic	See the <i>RAPID Reference Manual</i> for further information.

Revisions

Revision	Description	Prepared by
A	First edition – March 2007 (preliminary)	Philippe CHARLES
B	Second edition – January 2008 (preliminary)	Philippe CHARLES
C	Third edition – April 2009 (official release in English)	Philippe CHARLES
D		

Product Documentation

Generalities

The robot documentation is divided into different categories. This list is based on the type of information contained in every document, whether products are optional or not. This means that every robot product delivery does not contain all of the listed documents, but only those corresponding to the delivered equipment.

However, all listed documents can be ordered from ABB. The listed documents are valid for M2004 robot systems equipped with the IRC5 control system.

Manuals

The following manuals describe the robot software in general and contain relevant reference information:

- **RAPID General Overview:** overview of RAPID programming language.
- **RAPID Reference Manual:** description of all the RAPID instructions, functions, as well as the types of data.
- **Technical reference manual – System parameters:** description of the system parameters.

1 Safety

1.1 About the Safety Section

Introduction to Safety

This chapter describes safety principles and the procedures to be used when a robot or a robot system is in operation.

It covers neither the design nor the installation of equipment safety measures. These topics are covered in the Product Manuals delivered with the robot system.

Personnel Safety

A robot is heavy, extremely powerful and quick a pause or stopping of movement can be followed by an unpredictable quick movement. Even if a trajectory is envisaged, a modification during operation can intervene via an external signal resulting from an unexpected movement. It is therefore important to follow all rules in force when entering in the robot secure area.

Safety Regulation

Before beginning working with the robot, be sure that you know the safety regulations described in *Operator Manual - IRC5 with FlexPendant*.

1.2 Safety standards applicable to the IRC5 control system

1.2 Safety standards applicable to the IRC5 control system

Safety Standards

The robot complies totally with the safety standards specified in the European machine directives. The ABB robots controlled by the IRC5 control system comply with the following standards:

Standard	Description
EN ISO 12100-1	Safety of machinery, terminology
EN ISO 12100-2	Safety of machinery, technical specifications
EN 954-1	Safety of machinery, safety related parts of control systems
EN 775	Manipulating industrial robots, safety
EN 60204	Electrical equipment of industrial machines
EN 61000-6-4 (option)	EMC, generic emission
EN 61000-6-2	EMC, generic emission

Standard	Description
IEC 204-1	Electrical equipment of industrial machines
IEC 529	Degrees of protection provided by enclosures

Standard	Description
ISO 10218	Manipulating industrial robots, safety
ISO 9787	Manipulating industrial robots, coordinate systems and motions

Standard	Description
ANSI/RIA 15.06/1999	Safety requirements for industrial robots and robot systems
ANSI/UL 1740-1998 (option)	Safety standard for robots and robot equipment
CAN/CSA Z 434-03 (option)	Industrial robots and robot systems - General safety

1.3 Safety Terminology

1.3 Safety Terminology

General Layout

This section specifies all dangers that may arise from performing the work detailed in the manual. Each danger is detailed in its own section consisting of:

- A symbol per danger level (DANGER, WARNING or CAUTION) and the type of danger.
- A short description of what will happen if the service operator/personnel do not eliminate the danger.
- An instruction enabling elimination of the danger and manual completion of the task.

Levels of danger

The following table defines the warning symbols used in this manual.

Symbol	Description	Meaning
	Danger	Warns of risks that can cause serious bodily injury, even death and/or major damage to the product if recommendations are not followed.
	Warning	Warns of risks that could cause bodily harm or major damage to the product. Always comply with the recommendations that accompany this symbol.
	Electric shock	Warns of risks of electrical shock that could cause serious bodily harm, or even death. Always comply with the recommendations that accompany this symbol.
	Caution	Draws attention to the risk of damage to product if a certain action is not carried out or is carried out incorrectly.
	Electro-static discharge (ESD)	The ESD symbol indicates a risk of electro-static discharges that could lead to major damage to the product.
	Advice	Directs you towards specific instructions for further information, or tips to carry out the same operation in an easier way.
	Note	Information on important items.

2.1 What is the COGNEX “In-Sight” vision interface for the IRC5 control system?

2 Welcome

2.1 What is the COGNEX “In-Sight” vision interface for the IRC5 control system?

Description of the COGNEX “In-Sight” vision interface

The vision interface is available as a library of routines in “RAPID” language, which ensures communication functions between the IRC5 control system and one or more “In-Sight” COGNEX “5000” cameras connected to an Ethernet network. One camera can also communicate with several IRC5 control systems.

Main elements

Main elements of the COGNEX “In-Sight” vision interface:

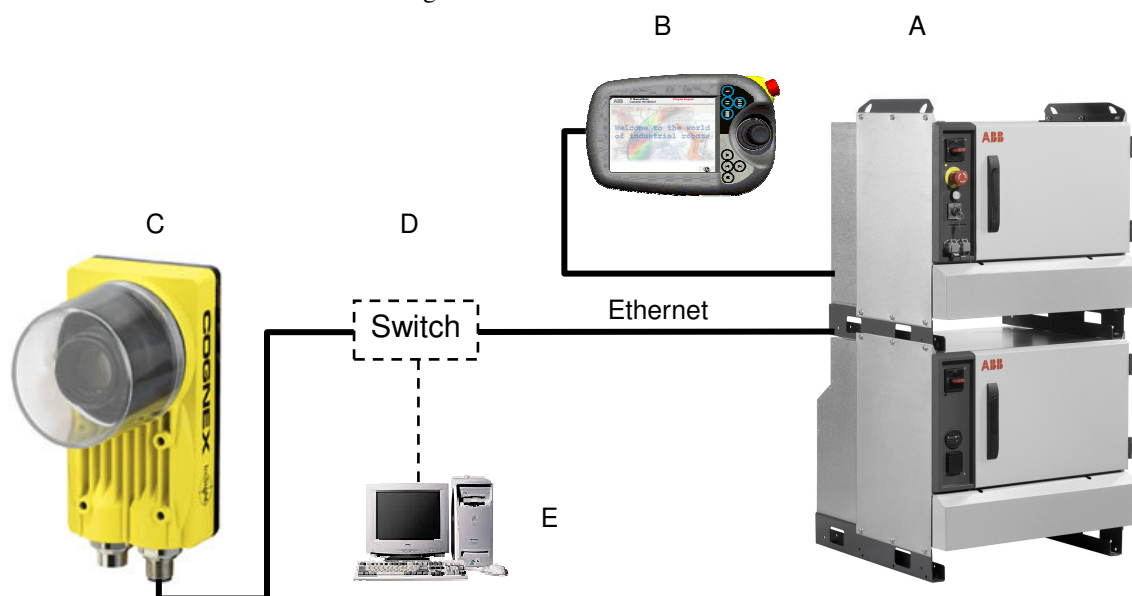


Figure 1. Hardware architecture with an IRC5 control system

A	IRC5 Control System
B	“FlexPendant” unit
C	“In-Sight” smart camera (5000 series)
D	Ethernet hub or “switch”
E	PC compatible computer

Note: the items (D) and (E) are only necessary when modifying the vision program using “In-Sight Explorer” as provided by the COGNEX Company.

2.2 Description of terms and concepts

Overview

This chapter contains the description of main concepts and words used in this manual. Additional information may also be found in all other chapters.

Robot Program

The robot program of an IRC5 control system is written in "RAPID" structured language.

This program can be viewed and edited on a PC compatible computer with "RobotStudio Online" software or on the robot's FlexPendant unit.

Vision program

The vision program for an "In-Sight" smart camera is called "job" and comes in the form of a spreadsheet with 400 lines (from 0 to 399) and 26 columns (from A to Z).

This program can be viewed and edited on a PC compatible computer using the "In-Sight Explorer" software provided by the COGNEX Company.

2D vision

A "2D" vision means image processing in 2 dimensions: the image obtained by the camera is processed in a plane. The coordinates obtained are 2 translations (X and Y) and a rotation around the Z normal of the image plane (rotation RZ).

In the case of the COGNEX image processing, these 3 coordinates are supplemented by an indicator of reliability of object recognition, called "score", expressed in the form of a percentage (between 0 and 100%).

Transformation

A transformation is a mathematical object that describes the combination of a translation and a rotation in the 3D space. Transformation is used to define the geometrical relation between two 3D frames.

Transformation can be expressed in matrix form ("homogeneous matrix" with a 4x4 dimension) or as a vector. It is the second formalism which is used in "RAPID" language.

"Socket", communication port and IP address

The "socket" is a communication mechanism that enables the exchange of messages between 2 computerized processes, generally located on different computers connected to a network. In this case, each "socket" uses one communication port and one "IP" address:

- The communication port is a logical port defined by an integer number encoded on 16 bits. Values generally reserved for the system are between 0 and 1023; values over 1023 are available for the user, in general for temporary connections.
 - The "IP" address is a logical address which enables identification of a computer linked to an "IP" network. The "IP" address is defined by an integer number 32 bits written as decimal, in the form of 4 bytes (from 0 to 255) separated by a point. Example: "10.33.50.105".
-

2.2 Description of terms and concepts

A "Socket" can be used according to 2 modes:

- "Connected" mode which uses the "TCP" transport protocol,
- "Non connected" mode which uses the "UDP" transport protocol,

The IRC5 control system enables the use of "sockets" in accordance with 2 modes. COGNEX "In-Sight" vision interface for the IRC5 control system only uses "connected" mode.

"Telnet" Application Protocol

The "Telnet" communication protocol is a standard Internet application protocol. This protocol, based on character strings, enables link up between a "client" (e.g. an input terminal) and a "server" (e.g. a remote computer).

This communication protocol is used on a very large number of computers, because it is available on most operating systems (e.g. UNIX, Windows) available on personal computers, as well as on industrial computer equipment: this is the case of COGNEX "In-Sight" cameras.

In the case of the IRC5 control system, the "Telnet" protocol implements communication "Sockets" linked to the "TCP" data transport protocol.

The default communication port for the "Telnet" protocol is number 23.

3 Software Organization

3.1 Generalities

Overview

The vision interface comes in the form of 7 modules written in RAPID language:

- INSIGHT General Module (mandatory)
- INSIGHT_DUT Module for localization in Dutch
- INSIGHT_ENG Module for localization in English (mandatory)
- INSIGHT_FRA Module for localization in French
- INSIGHT_GER Module for localization in German
- INSIGHT_ITA Module for localization in Italian
- INSIGHT_SPA Module for localization in Spanish
- INSIGHT_SWE Module for localization in Swedish

These modules are encrypted and cannot be edited by the user. These modules are available in 2 versions, either as “program module” or as “system module”: it is up to the user to choose the module version, function of the design of his application.

Software option required for the IRC5 control system

Operation of the COGNEX “In-Sight” vision interface requires the “Socket messaging” communication function with one of the following software options:

- RobotWare 5.06 to 5.09: “Socket Messaging” reference [672-1].
- RobotWare 5.10 and after: “PC Interface” reference [616-1].

Please ensure that your IRC5 control system is equipped with this software option.



Advice:

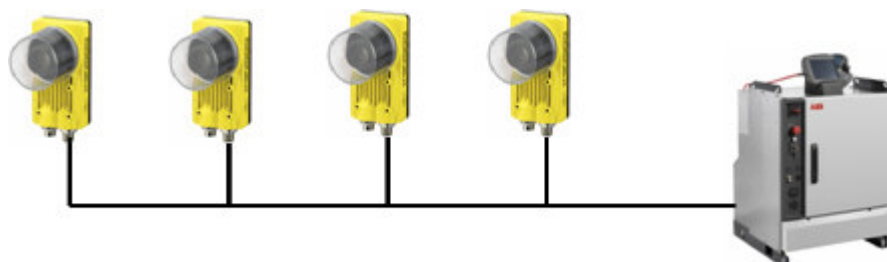
For more details on the Socket Messaging software option, please refer to:

- French documentation “Product characteristics – IRC5 control system software” reference 3HAC022349-001_rev3_FR, or
- English documentation “Product Specification marked points – Controller software IRC5” reference 3HAC022349-001_rev3_EN.

“Client/server” architecture

Communication between the IRC5 control system and the COGNEX “In-Sight” cameras is achieved by the Ethernet network and uses the “Native Mode” communication protocol designed for “In-Sight” cameras.

The camera is always used as the communication server and the robot as client. This means that the robot always initiates communication and sends requests to the camera, which then responds to the request. In this version, the robot can handle up to 4 cameras.



3.2 Loading the software

Overview

The software and documentation are delivered as a compressed ZIP file. Sequence of operation:

- Unzip this file: the product is delivered as library structured as several directories,
- Decide if you want to use the communication interface with “program modules” or “system modules” (recommended),
- Decide which HMI language you want to use in addition to English (which is mandatory),
- Select the modules in the library in accordance – e.g. “INSIGHT_FRA” for French. You have to load at least the modules “INSIGHT” and “INSIGHT_ENG” to be able to run the interface without error.

The required modules can be loaded into the IRC5 control system memory in 2 ways, by using:

- A PC compatible computer and the "RobotStudio Online" software, linked up with the IRC5 control system Ethernet on the "Service" or "LAN" port,
- The "FlexPendant" unit which can load files from an USB key.

4 Reference of the communication library instructions

4.1 General instructions for basic use

Overview

This chapter describes following instructions:

- CX_CloseComm
- CX_GetVisionData
- CX_InitComm
- CX_SetDebugMode
- CX_SetLanguage
- CX_SetupCamera
- CX_ShowErrStatus
- CX_ShowVersion
- CX_TriggImage

These instructions correspond with the operations most often used to manage communication between the robot and one or more cameras.

4.1.1 “CX_SetupCamera” instruction

CX_SetupCamera: configuration of data associated with a camera

The “CX_SetupCamera” instruction defines the configuration data for a camera concerning communication and the object recognition tolerances.

This instruction should be used first of all, before all other instructions of the communication library. The camera should have been pre-configured using the COGNEX utility software, in order to define its IP address.

Example

```
CX_SetupCamera 1, "10.33.50.100", 23, "admin", "", -80, 80, -60, 60, -90, 90, 70;
```

Arguments

CX_SetupCamera Identity SocketIP SocketPort UserName Password MinCoordX MaxCoordX
MinCoordY MaxCoordY MinCoordRZ MaxCoordRZ MinScore

Identity

Type of data: num

Integer value which defines the identity number of a camera, between 1 and 4.

SocketIP

Type of data: string

Character string defining the IP address of a camera: “xxx.xxx.xxx.xxx” (4 bytes between 0 and 255 separated by a point).

SocketPort

Type of data: num

Integer value which defines the communication port to dialogue with the camera.

Note: the default communication port for the “Telnet” protocol is number **23**.

UserName

Type of data: string

Character string that defines the user name associated with the camera during use in production.

Note: the default user name on “In-Sight” cameras is “**admin**”. One camera can manage up to 16 different users.

Password

Type of data: string

Character string that defines the password linked to the “UserName”.

Note: the password of the default user “admin” on the “In-Sight” cameras is an empty string

MinCoordX

Type of data: num

Real value that defines the lower threshold of the translation coordinate X of an object in the camera range. A lower coordinate will be considered as erroneous.

MaxCoordX

Type of data: num

Real value that defines the upper threshold of the translation coordinate X of an object in the camera range. A higher coordinate will be considered as erroneous.

MinCoordY

Type of data: num

Real value that defines the lower threshold of the translation coordinate Y of an object in the camera range. A lower coordinate will be considered as erroneous.

4 Reference of the communication library instructions

4.1 General instructions for basic use

MaxCoordY

Type of data: num

Real value that defines the upper threshold of the translation coordinate Y of an object in the camera range. A higher coordinate will be considered as erroneous.

MinCoordRZ

Type of data: num

Real value that defines the lower threshold of the rotation coordinate RZ of an object in the camera range. A lower coordinate will be considered as erroneous.

MaxCoordRZ

Type of data: num

Real value that defines the upper threshold of the rotation coordinate RZ of an object in the camera range. A higher coordinate will be considered as erroneous.

MinScore

Type of data: num

Real value that defines the lower "score" tolerance threshold of the view processing of an "In-Sight" camera. A lower value will be considered as erroneous.

Timeout

Type of data: num

Real value in the form of an optional parameter that defines the maximum waiting time for an "In-Sight" camera response. By default, the waiting time is 5 seconds.



Note:

The threshold parameters above are used by the boolean function "CX_GetVisionData"; if one of the parameters is erroneous, this function will return "FALSE".

Limits

The instruction checks the following parameters provided by the user:

- The "Identity" parameter must be an integer number between 1 and 4,
- The "SocketIP" parameter must be a string of four integer numbers between 0 and 255 separated by a point,
- The "SocketPort" parameter must be a positive integer number,
- The "MinScore" parameter must be a real value between 0 and 100,
- The optional "Timeout" parameter must be a real value higher than 0.

An error message is shown if one of the parameters is erroneous.

Syntax

CX_SetupCamera

```
[Identity' :='] <expression (IN) of num>','  
[SocketIP' :='] <expression (IN) of string>','  
[SocketPort' :='] <expression (IN) of num>','  
[UserName' :='] <expression (IN) of string>','  
[Password' :='] <expression (IN) of string>','  
[MinCoordX' :='] <expression (IN) of num>','  
[MaxCoordX' :='] <expression (IN) of num>','  
[MinCoordY' :='] <expression (IN) of num>','  
[MaxCoordY' :='] <expression (IN) of num>','  
[MinCoordRZ' :='] <expression (IN) of num>','  
[MaxCoordRZ' :='] <expression (IN) of num>','  
[MinScore' :='] <expression (IN) of num>','  
['\Timeout :='] <expression (IN) of num>' ;
```

4.1.2 “CX_InitComm” function

CX_InitComm : initialisation of communication with a camera

The Boolean function “CX_InitComm” enables initialisation of communication between the robot and a camera, previously configured using the instruction “CX_SetupCamera”.

This instruction should be used after the instruction “CX_SetupCamera” and before all the other instructions in the communications library (see the example of programming in section 5.2).

Example

```
VAR num nError;  
  
IF CX_InitComm(1,nError) THEN  
    ...  
ELSE  
    ! Error Management  
    ...  
ENDIF
```

The function returns value “**TRUE**” if communication with camera number 1 is correctly established. In the event of failure to initialise communication, the function takes value “**FALSE**” and variable *nError* receives the corresponding error code.

Arguments

CX_InitComm(Identity,ErrStatus)

CameraId

Type of data: num

Integer value which defines the identity number of a camera, between 1 and 4.

ErrStatus

Type of data: num

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.

An error message is shown if one of the parameters is erroneous.

Syntax

```
CX_InitComm '('  
[CameraId':=']<expression>(IN) of num>','  
[ErrStatus':=']<var or pers>(INOUT) of num>  
)'
```

4.1.3 “CX_TriggImage” function

CX_TriggImage: Triggers an image capture for a camera

The Boolean function “CX_TriggImage” enables an image capture to be triggered for a specific camera.

This instruction can only be used if the “CX_InitComm” function has taken place correctly.

Example

```
VAR num nError;  
  
IF CX_TriggImage(1,nError) THEN  
    ! Image processing  
    ...  
ELSE  
    ! Error management  
    ...  
ENDIF
```

The function returns value “**TRUE**” if the image capture with camera number 1 has been performed correctly. In the event of failure, it takes the value “**FALSE**” and the *nError* variable receives the corresponding error code.

Arguments

CX_InitComm(Identity,ErrStatus)

CameraId

Type of data: num

Integer value which defines the identity number of a camera, between 1 and 4.

ErrStatus

Type of data: num

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.

An error message is shown if one of the parameters is erroneous.

Syntax

```
CX_TriggImage'( '  
[CameraId':=']<expression>(IN) of num>', '  
[ErrStatus':=']<var or pers>(INOUT) of num>  
)'
```

4.1.4 “CX_GetVisionData” function

CX_GetVisionData: acquisition of vision data for a camera

The “CX_GetVisionData” function enables you to retrieve the 2D coordinates (X, Y, RZ and vision score) of an object in the vision program of an “In-Sight” camera. There are two 2D coordinates acquisition methods:

- Coordinate retrieval in a single cell using the ABB format,
- Coordinate retrieval into four contiguous cells.

The acquisition of 2D coordinates into a single cell is recommended to optimize the communication cycle time.

Example

```
PERS wobjdata wCam1:=[FALSE, TRUE, "", [[0,0,0],[1,0,0,0]], [[0,0,0],
[1,0,0,0]]];
VAR pose pVision:=[[0,0,0],[0,0,0,0]];
VAR num nScore;
VAR num nError;

IF CX_GetVisionData(1,"F",22\Read1Cell,pVision,nScore,nError) THEN
    ! Use vision offset
    wCam1.oframe:=pVision;
    ...
ELSE
    ! Error management
    ...
ENDIF
```

The function returns “**TRUE**” if the “F22” cell of the camera 1 vision program contains a character string using the ABB format:

- The *pVision* “pose” variable receives the 2D coordinates of the vision program,
- The *nScore* “num” variable receives the “score” value of the vision program.

In the event of failure, the function takes the value “**FALSE**” and the *nError* variable receives the corresponding error code.

Arguments

CX_GetVisionData(Identity,CellColumn,CellRow,[\Read1Cell][\Read4Cells],VisionTransform, VisionScore,ErrStatus)

CameraId

Type of data: num

Integer value which defines the identity number of a camera, between 1 and 4.

CellColumn

Type of data: string

Character string that defines the cell column in the vision program, between A and Z.

CellRow

Type of data: num

Integer value that defines the cell row in the vision program, between 0 and 399.

4 Reference of the communication library instructions

4.1 General instructions for basic use

\Read1Cell

Type of data: switch

Parameter that enables 2D coordinates to be read in just one cell of the vision program defined in ABB format, from the cell defined by (**CellColumn,CellRow**) and defined by the following syntax:

`[ ' <x> ' , ' <y> ' , ' <rz> ' , ' <score> ' ]`



Note:

- This method of reading 2D coordinates is recommended for optimizing communication cycle time.
- This format corresponds with a composite data item in RAPID language (identified by the term “aggregate” in the documentation).
- This parameter corresponds with the default behavior of the instruction when neither of these 2 optional parameters is specified by the user.

\Read4Cells

Type of data: switch

Parameter enabling 2D coordinated to be read in four contiguous cells of the vision program starting from the cell defined by (**CellColumn,CellRow**) up to (**CellColumn+3,CellRow**).

Every cell must contain a real or integer value. Otherwise, the function returns the value “FALSE”.

VisionTransform

Type of data: pose

“Pose” type variable that receives formatted 2D coordinates as a geometric transformation, directly usable as a displacement frame.

The transformation format is “[X,Y,0],[Q1,0,0,Q4]” where:

- X and Y are the translation coordinates of the vision program,
- Q1, Q2, Q3 and Q4 are the “quaternions” calculated using the RZ angle of rotation of the vision program. In this particular case, the quaternions Q2 and Q3 are always equal to 0.

VisionScore

Type of data: num

Real value that receives the vision recognition score set by the In-Sight vision.

ErrStatus

Type of data: num

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

Vision programming

The formatting of the vision program cell for reading all the 2D coordinates in just one cell can be achieved simply using the “Concatenate” function, available in the In-Sight camera programming. Example:

```
Concatenate ( " [ " , B22 , " , " , C22 , " , " , D22 , " , " , E22 , " ] " )
```

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.
- The “CellColumn” parameter must be a character between A and Z.
- The “CellRow” parameter must be an integer number between 0 and 399.

An error message is shown if one of the parameters is erroneous.

The 2D coordinates and “score” value should be within the tolerances defined in the *CX_SetupCamera* instruction.

4 Reference of the communication library instructions

4.1 General instructions for basic use

Syntax

```
CX_GetVisionData('
[CameraId':=']<expression>(IN) of num>',
[CellColumn':=']<expression>(IN) of string>',
[CellRow':=']<expression>(IN) of num>',
['\Read1Cell','] |
['\Read4Cell',']
[VisionTransform':=']<var or pers>(INOUT) of pose>',
[VisionScore':=']<var or pers>(INOUT) of num>',
[ErrStatus':=']<var or pers>(INOUT) of num>
')
```

4.1.5 “CX_ShowErrStatus” instruction

CX_ShowErrStatus: Displaying a vision error on the robot teach pendant

The “CX_ShowErrStatus” instruction enables an error message to be displayed on the “FlexPendant” unit. The error message is registered in the events log of the IRC5 control system.

Example

```
PERS wobjdata wCam1:=[FALSE,TRUE,"",[0,0,0],[1,0,0,0]],[[0,0,0],
[1,0,0,0]];
VAR pose pVision:=[0,0,0],[0,0,0,0];
VAR num nScore;
VAR num nError;

IF CX_GetVisionData(1,"F",22\Read1Cell,pVision,nScore,nError) THEN
    ! Use vision offset
    wCam1.oframe:=pVision;
    ...
ELSE
    ! Error Management
    CX_ShowErrStatus(nError);
    ...
ENDIF
```

If the result of the CX_GetVisiondata function is “**FALSE**”, an error message will be displayed on the teach pendant screen and registered in the IRC5 control system events log.

Arguments

CX_ShowErrStatus(ErrStatus)

ErrStatus

Type of data: num

Integer number that defines the error number to be displayed.

Limits

The instruction checks the following parameters provided by the user:

- The “ErrStatus” parameter must be an integer number.

In the event of an unknown error number, the displayed message will be in the following format:

- “Application error”: Unknown error code number <ErrStatus>“

Syntax

```
CX_ShowErrStatus
[ErrStatus':=']<var or pers>(INOUT) of num>';
```

4.1.6 “CX_CloseComm” function

CX_CloseComm: closing communication with a camera

The Boolean function “CX_CloseComm” enables closing of communication between the robot and a camera, previously configured using the instruction “CX_InitComm”.

Example

```
VAR num nError;  
  
IF CX_CloseComm(1,nError) THEN  
    ...  
ELSE  
    ! Error Management  
    ...  
ENDIF
```

The function returns value “**TRUE**” if communication with camera number 1 is correctly closed. In the event of erroneous camera number, the function takes the value “**FALSE**” and the *nError* variable receives the corresponding error code.

Arguments

CX_CloseComm(Identity,ErrStatus)

CameraId

Type of data: num

Integer value which defines the identity number of a camera, between 1 and 4.

ErrStatus

Type of data: num

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.

An error message is shown if one of the parameters is erroneous.

Syntax

```
CX_CloseComm'( '  
[CameraId':=']<expression>(IN) of num>', '  
[ErrStatus':=']<var or pers>(INOUT) of num>  
)'
```

4.1.7 “CX_SetLanguage” instruction

CX_SetLanguage: Definition of the man-machine interface language

The “CX_SetLanguage” instruction enables the definition of the man-machine interface and of all of the messages generated by the communication instructions (refer to chapter 8.2 for further display details).



Note:

English is the default language used if this instruction is not used.

Example

```
CX_SetLanguage \German;
```

Arguments

CX_SetLanguage Dutch | English | French | German | Italian | Spanish

Dutch	Type of data: switch
Parameter that enables the Dutch language.	
English	Type of data: switch
Parameter that enables the English language (default value).	
French	Type of data: switch
Parameter that enables the French language.	
German	Type of data: switch
Parameter that enables the German language.	
Italian	Type of data: switch
Parameter that enables the Italian language.	
Spanish	Type of data: switch
Parameter that enables the Spanish language.	
Swedish	Type of data: switch
Parameter that enables the Swedish language.	

Syntax

```
CX_SetLanguage  
['\Dutch','] |  
['\English','] |  
['\French','] |  
['\German','] |  
['\Italian','] |  
['\Spanish'] |  
['\Swedish'] ;'
```

4.1.8 “CX_SetDebugMode” instruction

CX_SetDebugMode : Enabling and disabling “debug” mode

The “CX_SetDebugMode” instruction enables or disables the “debug” mode to help debugging of a communication application. Enabling the “debug” mode displays all of the characters exchanged during the communication on the “FlexPendant” unit (refer to section 8.2 for further details on display).

Example

```
CX_SetDebugMode \On;
```

Arguments

CX_SetDebugMode On|Off

On

Type of data: switch

Parameter that enables the “debug” mode.

Off

Type of data: switch

Parameter that disables the “debug” mode.

Limits

This instruction should only be used **temporarily** during application development.



Note:

The debug mode should not be used during production, as there is a risk of deleting or interfering with the “FlexPendant” operator dialogue window created in “RAPID” language ; this is often used in production and is usually achieved with the instructions “TPWrite” or “TPReadFK”.

Syntax

```
CX_SetDebugMode  
['\On','] |  
['\Off','] ;'
```

4.1.9 “CX_ShowVersion” instruction

CX_ShowVersion: Displays the version of the communication library

The “CX_ShowVersion” instruction enables display of the communication library version on the “FlexPendant” unit.

This function is required to have access to the communication library version number due to the encryption of the source code. This utility function is not normally used during production, at least only in manual mode for information.

Example

```
CX_ShowVersion;
```

Arguments

CX_ShowVersion \Pause

Pause (optional argument)

Type of data: switch

Optional argument which prompts a pause during the robot program execution, through an acknowledgment request on the “FlexPendant” unit by the operator (press on “Ok” key).

If this optional argument optional is not used, the version is displayed in the “operator dialogue” window.

Syntax

```
CX_ShowVersion;  
['\Pause'] ;'
```

4 Reference of the communication library instructions

4.2 Instructions to read and write data in the vision program

4.2 Instructions to read and write data in the vision program

Overview

This chapter describes following instructions:

- CX_GetInfo
- CX_GetValue
- CX_SetFloat
- CX_SetInteger
- CX_SetString

These instructions give the possibility to read and write data in the vision program of an “In-Sight” camera.

Refer to the section on “Settings & Cell Values Commands” in the COGNEX documentation for details of the protocol used.

4.2 Instructions to read and write data in the vision program

4.2.1 “CX_GetInfo” function

CX_GetInfo: Read status of vision system

The “CX_GetValue” function enables you to read the vision system status data: the serial number, the identity number “MAC address», the versions and the date of the system software.



Note:

The “CX_GetInfo” function is not yet implemented in this version, and returns “FALSE”.

Example

```
IF NOT CX_GetInfo(1, strSN, strFWV, strMV, strMAC, strDoB, nError) THEN
...
ENDIF
```

Arguments

CX_GetInfo(CameraId, SerialNumber, FirmwareVersion, MonitorVersion, MacAddress, DateOfBuild, ErrStatus)

CameraId	Type of data: num
Integer value which defines the identity number of a camera, between 1 and 4.	
SerialNumber	Type of data: string
Character string that returns the camera serial number.	
FirmwareVersion	Type of data: string
Character string that returns the version of the system’s camera Firmware.	
MonitorVersion	Type of data: string
Character string that returns the version of the system’s Monitor software.	
MacAddress	Type of data: string
Character string that returns the MAC address of the camera’s Ethernet port.	
DateOfBuild	Type of data: string
Character string that returns the build date of the system’s camera Firmware.	
ErrStatus	Type of data: num
Integer value in which the function returns the error status: 0 if Ok or negative value as error code.	

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.

An error message is shown if one of the parameters is erroneous.

4 Reference of the communication library instructions

4.2 Instructions to read and write data in the vision program

Syntax

```
CX_GetInfo
[CameraId':=']<expression>(IN) of num>',
[SerialNumber':=']<var or pers>(INOUT) of string>',
[Firmware Version':=']<var or pers>(INOUT) of string>',
[MonitorVersion':=']<var or pers>(INOUT) of string>',
[MacAddress':=']<var or pers>(INOUT) of string>',
[DateOfBuild':=']<var or pers>(INOUT) of string>',
[ErrStatus':=']<var or pers>(INOUT) of num>
')
```

4 Reference of the communication library instructions

4.2 Instructions to read and write data in the vision program

4.2.2 “CX_GetValue” function

CX_GetValue: Reading a character chain in the vision program

The “CX_GetValue” function enables you to read the value of a cell in the vision program.

Example

```
VAR num nError;  
VAR string strValue;  
  
IF NOT CX_GetValue(1,"C",29,strValue,nError) THEN  
    CX_ShowErrStatus(nError);  
    RETURN;  
ENDIF
```

The function returns the value “**TRUE**” if the content of cell “C29” could be assigned to the “strValue” character string. Otherwise it returns “**FALSE**” with the error number in “nError”.

Arguments

CX_GetValue(Identity,CellColumn,CellRow,StringData,ErrStatus)

CameraId	Type of data: num
Integer value which defines the identity number of a camera, between 1 and 4.	
CellColumn	Type of data: string
Character chain that defines the vision program pointing column between A and Z.	
CellRow	Type of data: num
Integer value that defines the vision program line to be pointed, between 0 and 399.	
StringData	Type of data: string
Character chain that returns the value of the cell specified by the vision program.	
ErrStatus	Type of data: num
Integer value in which the function returns the error status: 0 if Ok or negative value as error code.	

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.
- The “CellColumn” parameter must be a character between A and Z.
- The “CellRow” parameter must be an integer number between 0 and 399.

An error message is shown if one of the parameters is erroneous.

4 Reference of the communication library instructions

4.2 Instructions to read and write data in the vision program

Syntax

```
CX_GetValue'(
[CameraId':=']<expression>(IN) of num>',
[CellColumn':=']<expression>(IN) of string>',
[CellRow':=']<expression>(IN) of num>',
[StringData':=']<expression>(INOUT) of string>',
[ErrStatus':=']<var or pers>(INOUT) of num>
')
```

4 Reference of the communication library instructions

4.2 Instructions to read and write data in the vision program

4.2.3 “CX_SetFloat” function

CX_SetFloat: writing a real value into the vision program

The “CX_SetFloat” function enables the writing of a real value into a cell of the vision program.

Example

```
VAR num nError;  
IF NOT CX_SetFloat(1,"C",29,451.32,nError) THEN  
    CX_ShowErrStatus(nError);  
    RETURN;  
ENDIF
```

The function will try to write the number **451,32** into the cell “C29” of the vision program. If the operation succeeds, the function returns the value “**TRUE**”.

The destination cell must be in the correct “Float” format, otherwise the function will return the value “**FALSE**” and the error number into “nError”.

Arguments

CX_SetFloat(Identity,CellColumn,CellRow,FloatData,ErrStatus)

CameraId	Type of data: num
Integer value which defines the identity number of a camera, between 1 and 4.	
CellColumn	Type of data: string
Character string that defines the vision program pointing column between A and Z.	
CellRow	Type of data: num
Integer value that defines the vision program line to be pointed, between 0 and 399.	
FloatData	Type of data: num
Real value that is written into the cell specified by the vision program.	
ErrStatus	Type of data: num
Integer value in which the function returns the error status: 0 if Ok or negative value as error code.	

Vision programming

The vision program cell must be defined as an interactive control of type “EditFloat”. This control has minimum and maximum values defined by default (respectively 0 and 100) which should be adjusted in relation to the real value written by the robot.

4 Reference of the communication library instructions

4.2 Instructions to read and write data in the vision program

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.
- The “CellColumn” parameter must be a character between A and Z.
- The “CellRow” parameter must be an integer number between 0 and 399.
- The “FloatData” parameter must be a real or integer number,

An error message is shown if one of the parameters is erroneous.

The “In-Sight” camera returns an error if the vision program cell does not contain an “EditFloat” interactive control or if the number returned by the robot is not within the minimum and maximum control limits.

Syntax

```
CX_SetFloat'(
[CameraId':=']<expression>(IN) of num>',
[CellColumn':=']<expression>(IN) of string>',
[CellRow':=']<expression>(IN) of num>',
[FloatData':=']<var or pers>(IN) of num>',
[ErrStatus':=']<var or pers>(INOUT) of num>
')
```

4.2.4 “CX_SetInteger” function

CX_SetInteger: writing an integer value into the vision program

The “CX_SetInteger” function enables the writing of an integer value into a cell of the vision program.

Example

```
IF NOT CX_SetInteger(1,"C",29,221,nError) THEN
  CX_ShowErrStatus(nError);
  RETURN;
ENDIF
```

The function will try to write the number **221** into the cell “C29” of the vision program. If the operation succeeds, the function returns the value “**TRUE**”.

The destination cell must be in the correct format, otherwise the function will return the value “**FALSE**” and the error number into “nError”.

Arguments

CX_SetInteger(Identity,CellColumn,CellRow,IntegerData,ErrStatus)

CameraId	Type of data: num
Integer value which defines the identity number of a camera, between 1 and 4.	
CellColumn	Type of data: string
Character string that defines the vision program pointing column between A and Z.	
CellRow	Type of data: num
Integer value that defines the vision program line to be pointed, between 0 and 399.	
IntegerData	Type of data: num
Integer value that is written into the specified vision program call, between 0 and 255.	
ErrStatus	Type of data: num
Integer value in which the function returns the error status: 0 if Ok or negative value as error code.	

Vision programming

The vision program cell must be defined as an interactive control of type «EditInt», «CheckBox» or «ListBox». These controls have minimum and maximum values defined by which should be adjusted in relation to the integer value written by the robot.

- “EditInt”: minimum 0 and maximum 255,
- “CheckBox”: minimum 0 and maximum 1,
- “ListBox”: minimum 0 and maximum depending on the number of elements in the list.

4 Reference of the communication library instructions

4.2 Instructions to read and write data in the vision program

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.
- The “CellColumn” parameter must be a character between A and Z.
- The “CellRow” parameter must be an integer number between 0 and 399.
- The “IntegerData” parameter must be an integer number,

An error message is shown if one of the parameters is erroneous.

The “In-Sight” camera returns an error if the vision program cell does not contain a valid interactive control or if the number returned by the robot is not within the minimum and maximum control limits.

Syntax

```
CX_SetInteger('
[CameraId':=']<expression>(IN) of num>',
[CellColumn':=']<expression>(IN) of string>',
[CellRow':=']<expression>(IN) of num>',
[IntegerData':=']<var or pers>(IN) of num>',
[ErrStatus':=']<var or pers>(INOUT) of num>
')
```

4 Reference of the communication library instructions

4.2 Instructions to read and write data in the vision program

4.2.5 “CX_SetString” function

CX_SetString: writing a character chain in the vision program

The “CX_SetString” function enables the writing of a character string into a cell of the vision program.

Example

```
IF NOT CX_SetString(1,"C",29,"ABB",nError) THEN
  CX_ShowErrStatus(nError) ;
  RETURN;
ENDIF
```

The function will try to write the string “ABB” into cell “C29” of the vision program. If the operation succeeds, the function returns the value “**TRUE**”.

The destination cell must be in the correct format otherwise the function will return the value “**FALSE**” and the error number into “nError”.

Arguments

CX_SetString(Identity,CellColumn,CellRow,StringData,ErrStatus)

CameraId	Type of data: num
Integer value which defines the identity number of a camera, between 1 and 4.	
CellColumn	Type of data: string
Character string that defines the vision program pointing column between A and Z.	
CellRow	Type of data: num
Integer value that defines the vision program line to be pointed, between 0 and 399.	
StringData	Type of data: string
Character string that is written into the cell specified by the vision program.	
ErrStatus	Type of data: num
Integer value in which the function returns the error status: 0 if Ok or negative value as error code.	

Vision programming

The vision program cell must be defined as an interactive control of type “EditString”. The maximum character string length by default (8) can be adjusted to comply with the string length written by the robot.

4 Reference of the communication library instructions

4.2 Instructions to read and write data in the vision program

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.
- The “CellColumn” parameter must be a character between A and Z.
- The “CellRow” parameter must be an integer number between 0 and 399.

An error message is shown if one of the parameters is erroneous.

The “In-Sight” camera returns an error if the vision program cell does not contain an “EditString” interactive control or if the number returned by the robot is too long (i.e. exceeds the maximum size defined in the control).

Syntax

```
CX_SetString'(  
[CameraId':=']<expression>(IN) of num>',  
[CellColumn':=']<expression>(IN) of string>',  
[CellRow':=']<expression>(IN) of num>',  
[StringData':=']<expression>(IN) of string>',  
[ErrStatus':=']<var or pers>(INOUT) of num>  
)'
```

4 Reference of the communication library instructions

4.3 Instructions to manage execution and “Online” mode

4.3 Instructions to manage execution and “Online” mode

Overview

This chapter describes following instructions:

- CX_GetOnline
- CX_SendMessage
- CX_SetEvent
- CX_SetEventWait
- CX_SetOnline

These instructions enable the management of events during the execution of the vision program, as well as the “Online” mode.

Refer to the section on “Execution & Online Commands” in the COGNEX documentation for details of the protocol used.

4 Reference of the communication library instructions

4.3 Instructions to manage execution and “Online” mode

4.3.1 “CX_GetOnline” function

CX_GetOnline: checks “online” status for a camera

The Boolean function “CX_ GetOnline”, enables checking of the online status of a specified camera.

This instruction can only be used if the “CX_InitComm” function has taken place correctly.

Example

```
VAR num nError;  
  
IF CX_GetOnline(1,nError) THEN  
    ! Image Processing  
    ...  
ELSE  
    ! Error management  
    ...  
ENDIF
```

The function returns the value “**TRUE**” if camera number 1 is online. Otherwise it returns “**FALSE**” with the error number in “nError”.

Arguments

CX_GetOnLine(Identity,ErrStatus)

CameraId

Type of data: num

Integer value which defines the identity number of a camera, between 1 and 4.

ErrStatus

Type of data: num

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.

An error message is shown if one of the parameters is erroneous.

Syntax

```
CX_GetOnLine :  
[CameraId':=']<expression>(IN) of num>','  
[ErrStatus':=']<var or pers>(INOUT) of num>  
)'
```

4.3.2 “CX_SendMessage” function

CX_SendMessage: sends a message to the vision program

The Boolean function “CX_SendMessage» enables a message to be sent and an optional event code to the vision program.

Example

```
VAR num nError;  
IF CX_SendMessage(1,"ABB",\EventNb:=1,nError) THEN  
    ! Processing  
    ...  
ELSE  
    ! Error Management  
    ...  
ENDIF
```

The function returns value “**TRUE**” if the event transmission with camera number 1 has been performed correctly. In the event of failure, it takes the value “**FALSE**” and the *nError* variable receives the corresponding error code.

Arguments

CX_SendMessage(Identity,ErrStatus)

CameraId	Type of data: num
Integer value which defines the identity number of a camera, between 1 and 4.	
Message	Type of data: string
Character string that defines the message to be sent to the vision program.	
EventNb	Type of data: num
Optional integer number that defines the event number to be sent to the vision program.	
ErrStatus	Type of data: num
Integer value in which the function returns the error status: 0 if Ok or negative value as error code.	

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.
- The “EventNb” parameter has to be an integer within the range recognised by RAPID.

An error message is shown if one of the parameters is erroneous.

4 Reference of the communication library instructions

4.3 Instructions to manage execution and “Online” mode

Syntax

```
CX_SendMessage'(
  [CameraId':=']<expression>(IN) of num>',
  [Message':=']<expression>(IN) of string>',
  [EventNb':=']<expression>(IN) of num>',
  [ErrStatus':=']<var or pers>(INOUT) of num>
  )'
```

4.3.3 “CX_SetEvent” function

CX_SetEvent: sends an event code to the vision program

The Boolean function “CX_ SetEvent” enables an event code to be sent to the vision program. The robot does **not** wait until the end of processing for this event in order to continue its program.

Example

```
VAR num nError;  
IF CX_SetEvent(1,101,nError) THEN  
    ! Processing  
    ...  
ELSE  
    ! Error Management  
    ...  
ENDIF
```

The function returns value “**TRUE**” if the event transmission with camera number 1 has been performed correctly. Otherwise it returns “**FALSE**” with the error number in “nError”.

Arguments

CX_SetEvent(CameraId,EventNb,ErrStatus)

CameraId

Type of data: num

Integer value which defines the identity number of a camera, between 1 and 4.

EventNb

Type of data: num

Integer number that defines the event number to be sent to the vision program.

ErrStatus

Type of data: num

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.
- The “EventNb” parameter has to be an integer within the range recognised by RAPID.

An error message is shown if one of the parameters is erroneous.

Syntax

```
CX_SentEvent(  
[CameraId':=']<expression>(IN) of num>',  
[EventNb':=']<expression>(IN) of num>',  
[ErrStatus':=']<var or pers>(INOUT) of num>  
)
```

4.3.4 “CX_SetEventWait” function

CX_SetEventWait: Sends an event code to the vision program and waits for the process end

The Boolean function “CX_SetEventWait” sends an event code to the vision program. The robot waits for this event to be completely processed before continuing its program.

Example

```
VAR num nError;  
IF CX_SetEventWait(1,101,nError) THEN  
    ! Processing  
...  
ELSE  
    ! Error management  
...  
ENDIF
```

The function returns value “**TRUE**” if the event transmission with camera number 1 has been performed correctly. Otherwise it returns “**FALSE**” with the error number in “nError”.

Arguments

CX_SendEventWait(CameraId,EventNb,ErrStatus)

CameraId

Type of data: num

Integer value which defines the identity number of a camera, between 1 and 4.

EventNb

Type of data: num

Integer number that defines the event number to be sent to the vision program.

ErrStatus

Type of data: num

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.
- The “EventNb” parameter has to be an integer within the range recognised by RAPID.

An error message is shown if one of the parameters is erroneous.

Syntax

```
CX_SendEventWait(''  
[Identity':=']<expression>(IN) of num>','  
[EventNb':=']<expression>(IN) of num>','  
[ErrStatus':=']<var or pers>(INOUT) of num>  
'')
```

4 Reference of the communication library instructions

4.3 Instructions to manage execution and “Online” mode

4.3.5 “CX_SetOnline” function

CX_SetOnline: Places camera at "online" status

The Boolean function “CX_GetOnline” enables the specified camera to be placed online or offline. This function can only be used if the “CX_InitComm” function has taken place correctly.

Example

```
VAR num nError;  
IF CX_SetOnline(1\On,nError) THEN  
    ! Image processing ...  
ELSE  
    ! Error Management ...  
ENDIF
```

The function returns value “**TRUE**” if the placing online of camera number 1 has been performed correctly. Otherwise it returns “**FALSE**” with the error number in “nError”.

Arguments

CX_SetOnline(Identity,\OnOff,ErrStatus)

CameraId	Type of data: num
Integer value which defines the identity number of a camera, between 1 and 4.	
On	Type of data: switch
Enables the camera to go online.	
Off	Type of data: switch
Enables the camera to go offline. Backing up the \On switch, this is the default value when neither switch is specified.	
ErrStatus	Type of data: num
Integer value in which the function returns the error status: 0 if Ok or negative value as error code.	

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.

An error message is shown if one of the parameters is erroneous.

Syntax

```
CX_SetOnLine :  
[CameraId':=']<expression>(IN) of num>','  
['\On',''] | ['\Off','']  
[ErrStatus':=']<var or pers>(INOUT) of num>  
)'
```

4.4 File management instructions

Overview

This chapter describes following instructions:

- CX_GetFile
- CX_LoadFile

These instructions enable management of an “In-Sight” camera’s program files (“job”).

Refer to the section on “File & Job Commands” in the COGNEX documentation for details of the protocol used.

4.4.1 “CX_GetFile” function

CX_GetFile: Reads the name of the vision program loaded onto the camera

The Boolean function “CX_GetFile” enables reading of the name of the vision program currently loaded onto the “In-Sight” cameras execution memory.

Example

```
VAR string strFileName;  
VAR num nError;  
IF CX_GetFile(1,strFileName,nError) THEN  
    ! Processing ...  
ELSE  
    ! Error Management ...  
ENDIF
```

The function returns the value “**TRUE**” if the vision program for camera 1 has been saved; the name of the vision program is then returned in the *strFileName* variable.

If no vision program has been loaded, the value will be “**FALSE**”. In the event of failure, it takes the value “**FALSE**” and the *nError* variable receives the corresponding error code.

Arguments

CX_GetFile(CameraId,FileName,ErrStatus)

CameraId

Type of data: num

Integer value which defines the identity number of a camera, between 1 and 4.

FileName

Type of data: string

Character string that returns the name of the vision program loaded onto the “In-Sight” camera.

ErrStatus

Type of data: num

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.

An error message is shown if one of the parameters is erroneous.

Syntax

```
CX_GetFile(''  
[CameraId':=']<expression>(IN) of num>','  
[FileName':=']<var or pers>(INOUT) of string>','  
[ErrStatus':=']<var or pers>(INOUT) of num>  
'')
```

4.4.2 “CX_LoadFile” function

CX_LoadFile: Loads a vision program onto the camera

The Boolean function “CX_LoadFile” enables the loading of a vision program (“job”) from the flash memory of the “In-Sight” camera to its execution memory.

**Note:**

This function can only be used if the camera has been previously set “Offline” by the robot using the “CX_SetOnline” function. The camera should be set “online” in the same way after running the “CX_LoadFile” function.

Example

```
VAR string strFileName;  
VAR num nError;  
IF NOT CX_SetOnline(1\Off,nError) Exit;  
IF NOT CX_LoadFile(1,strFileName,nError) THEN  
    ! Error Management  
    ...  
ENDIF  
IF NOT CX_SetOnline(1\On,nError) Exit;
```

Arguments

CX_LoadFile(Identity,ErrStatus)

CameraId

Type of data: num

Integer value which defines the identity number of a camera, between 1 and 4.

FileName

Type of data: string

Character string that defines the name of the vision program to be loaded onto the “In-Sight” camera. The “.job” extension can be omitted (the “In-Sight” camera then adds the extension).

ErrStatus

Type of data: num

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

Limits

The instruction checks the following parameters provided by the user:

- The “CameraId” parameter must be an integer number between 1 and 4.

An error message is shown if one of the parameters is erroneous.

Syntax

```
CX_LoadFile(''  
[CameraId':=']<expression>(IN) of num>','  
[FileName':=']<var or pers>(IN) of string>','  
[ErrStatus':=']<var or pers>(INOUT) of num>  
'')
```

4.5 Robot-vision calibration instructions

Overview

This chapter describes following instructions:

- SetToolVision
- SetWObjVision

These instructions enable management of robot-vision calibration in the event of a fixed camera and a camera mounted on the robot.

Refer to section 7 “Robot-vision calibration procedures” of this documentation for details of the implementation of these instructions.

4.5.1 “SetToolVision” instruction

SetToolVision: calculating a tool frame for a camera mounted on the robot

The “SetToolVision” instruction enables calculation of the robot-vision transformation calibration in case of a camera mounted on the robot, and returns a tool frame associated with the camera.



Note:

The “SetToolVision” function is not yet implemented in this version of the interface. Contact ABB France Robotics Division for technical support.

Example

```
VAR robtarget RobotPos{4};
VAR pose VisionPos{4};
PERS tooldata toolVision;
VAR num nError;

SetToolVision RobotPos, VisionPos, 4, \FRAME_2D, toolVision, nError;
```

The instruction returns the tool frame *toolVision* associated with the camera if the calibration calculation is correct. In the event of failure, the *nError* variable received the corresponding error code.

Arguments

SetToolVision RobotPos, VisionPos, NbCalPos, \TCP, toolVision, ErrStatus

RobotCalPos

Type of data: robtarget

Array of robot positions that defines the calibration positions learnt in the robot base frame (or the world frame).

VisionCalPos

Type of data: pose

Array of vision transformations that define the calibration positions learnt in the vision frame.

NbCalPos

Type of data: num

Integer value that defines the number of calibration positions, between 4 and 20.

FRAME_2D (optional argument)

Type of data: switch

Optional argument that specifies that the tool frame will be calculated in 2D based on a nominal tool frame, i.e. components X/Y/RZ in the vision frame; 4 positions are required. This method is used by default if no optional argument is programmed.

FRAME_3D (optional argument)

Type of data: switch

Optional argument that specifies that the complete tool frame will be calculated; this argument requires a specific calibration plate (argument not yet implemented).

Tool

Type of data: tooldata

Tool frame returned by the instruction (transformation associated with the “In-Sight” camera).

ErrStatus

Type of data: num

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

4 Reference of the communication library instructions

4.5 Robot-vision calibration instructions

Limits

The instruction checks the following parameters provided by the user:

- The “NbCalPos” parameter must be an integer number between 4 and 20.
- The “RobotCalPos” and “VisionCalPos” arrays should contain the required number of elements (4 or up to 20 depending on the optional argument selected).

An error message is shown if one of the parameters is erroneous.

Syntax

```
SetToolVision('
[RobotCalPos:=']<expression>(IN) of robtarget>',
[VisionCalPos:=']<expression>(IN) of pose>',
[NbCalPos:=']<expression>(IN) of num>',
['\FRAME_2D',]
['\FRAME_3D',]
[Tool:=']<pers>(INOUT) of tooldata>',
[ErrStatus:=']<VAR or PERS>(INOUT) of num>
')
```

4.5.2 “SetWObjVision” Instruction

SetWObjVision: calculate robot-vision calibration for fixed camera

The “SetWObjVision” instruction enables calculation of the robot-vision calibration in case of a fixed camera and returns a work object frame (“wobjdata”) associated with the camera.

Example

```
VAR robtargget RobotPos{9};
VAR pose VisionPos{9};
PERS wobjdata wobjVision;
VAR num nError;

SetWObjVision RobotPos,VisionPos,9,wobjVision,nError;
```

The instruction returns the work object frame *wobjvision* associated with the camera if the calibration calculation is correct. In the event of failure, the *nError* variable received the corresponding error code.

Arguments

SetWObjVision RobotPos,VisionPos, NbCalPos,wobjVision,ErrStatus

RobotCalPos	Type of data: robtargget
Array of robot positions that defines the calibration positions learnt in the robot base frame (or world frame).	
VisionCalPos	Type of data: pose
Array of vision transformations that define the calibration positions learnt in the vision frame.	
NbCalPos	Type of data: num
Integer value that defines the number of calibration positions, between 3 and 20.	
UFrame (optional argument)	Type of data: switch
Optional argument that returns the calibration transformation in the “uframe” component of the work object frame. This argument is used by default if no optional argument is programmed.	
OFrame (optional argument)	Type of data: switch
Optional argument that returns the calibration transformation in the “oframe” component of the work object frame.	
WObj	Type of data: wobjdata
Work object frame (type “wobjdata”) into which the instruction returns the transformation associated with the “In-Sight” camera.	
MaxError (optional argument)	Type of data: num
Optional argument that returns the maximum calibration calculation error.	
MeanError (optional argument)	Type of data: num
Optional argument that returns the mean calibration calculation error.	
ErrStatus	Type of data: num

4 Reference of the communication library instructions

4.5 Robot-vision calibration instructions

Integer value in which the function returns the error status: 0 if Ok or negative value as error code.

Limits

The instruction checks the following parameters provided by the user:

- The “NbCalPos” parameter must be an integer number between 3 and 10.
- The “RobotCalPos” and “VisionCalPos” arrays must contain at least the number of elements specified in the parameter “NbCalPos”.

An error message is shown if one of the parameters is erroneous.

Syntax

```
SetWObjVision'(  
[RobotCalPos':=']<expression>(IN) of robtarget>',  
[VisionCalPos':=']<expression>(IN) of pose>',  
[NbCalPos':=']<expression>(IN) of num>',  
['\UFrame',]  
['\OFrame',]  
[WObj:=']<pers>(INOUT) of wobjdata>',  
['\MaxError':=']<expression>(INOUT) of num>',  
['\MeanError':=']<expression>(INOUT) of num>',  
[ErrStatus':=']<VAR or PERS>(INOUT) of num>  
)'
```

5 Implementing the communication library instructions

5.1 Standard utilization sequences for library instructions.

Overview

The simplest use of the communication library is provided by the following sequence of instructions, detailed in section 4:

- | | | |
|----|------------------|---------------------------------------|
| 1. | CX_SetLanguage | Defines user language |
| 2. | CX_SetupCamera | Camera configuration |
| 3. | CX_InitComm | Initializes communication with camera |
| 4. | CX_TriggImage | Requests a picture from camera |
| 5. | CX_GetVisionData | Reads coordinates in the camera |
| 6. | CX_CloseComm | Closes communication with camera |

This sequence is specific to a camera and should be repeated for every camera used with the IRC5 control system.

Case of a single camera

In case of a single camera, only the triggering of an image capture and the reading of vision data are used in the production loop.

```
CX_SetLanguage \French;    Defines the user language
CX_SetupCamera 1,...      Camera configuration
CX_InitComm 1,...         Initializes communication with camera
...
WHILE <Production> DO
  ...
  CX_TriggImage 1,...      Requests an image capture from camera
  ...
  CX_GetVisionData 1,...   Reads coordinates in the camera
  ...
ENDWHILE
...
CX_CloseComm 1,...        Closes communication with camera
```

This elementary instruction sequence can be extended with instructions to read and write data in the vision program, typically to manage the part type being produced. The most often used additional instructions are:

- “CX_SetFloat” Writing a real value into the vision program
- “CX_SetInteger” Writing an integer value into the vision program
- “CX_SetString” Writing a character string in the vision program

Other services can also be added, for example to read the camera status before starting up production (“CX_GetOnline”). See the program example supplied.

5 Implementing the communication library instructions

5.1 Standard utilization sequences for library instructions.

Case of several cameras

In the case of several cameras, a limitation of the current version of the library imposes a different instruction sequence; the initializing and closing of communication should be located in the production loop, with the triggering of image capture and reading of vision data:

```
CX_SetLanguage \French;    Defines user language
CX_SetupCamera 1,...       Camera 1 configuration
CX_SetupCamera 2,...       Camera 2 configuration
...
WHILE <Production> DO
    ...
    CX_InitComm 1,...       Initializes communication with camera 1
    CX_TriggImage 1,...     Requests image capture from camera 1
    ...
    CX_GetVisionData 1,...  Reads coordinates in camera 1
    CX_CloseComm 1,...     Closes communication with camera 1
    ...
    CX_InitComm 2,...       Initializes communication with camera 2
    CX_TriggImage 2,...     Requests image capture from camera 2
    ...
    CX_GetVisionData 2,...  Reads coordinates in camera 2
    CX_CloseComm 2,...     Closes communication with camera 2
    ...
ENDWHILE
...
```



Note:

This restriction will be removed in a future version.

5.2 Using the vision transformation

Overview

The instruction “CX_GetVisionData” returns a transformation calculated from the 2D coordinates provided by the “In-Sight” camera. A transformation can be used directly in the “RAPID” programming language of the IRC5 control system as follow:

- In a program displacement instruction (case of a fixed camera),
- In a work object frame data (case of a fixed camera),
- In a transformation calculation defined by the user.

Camera frame and vision transformation

Vision transformation is expressed in the camera frame. To be correctly used in the robot program, you must have done previously:

- The calibration procedure of the image plane: this is handled by the vision system, to take into account the pixels/mm ratio and any possible optical deformations. See the COGNEX user manual, especially the new “EasyBuilder” wizard which now includes an automatic procedure to deal with this calibration,
- The robot/vision calibration procedure (see details in section 7). The robot/vision calibration operation enables a correlation between the camera frame and a robot user frame (object frame or tool frame according to whether the camera is fixed to the ground or is mounted on the robot).

Chain of geometrical frames used during operation (case of a fixed camera)

The chain of geometrical frames used during operation is defined by the following figure:

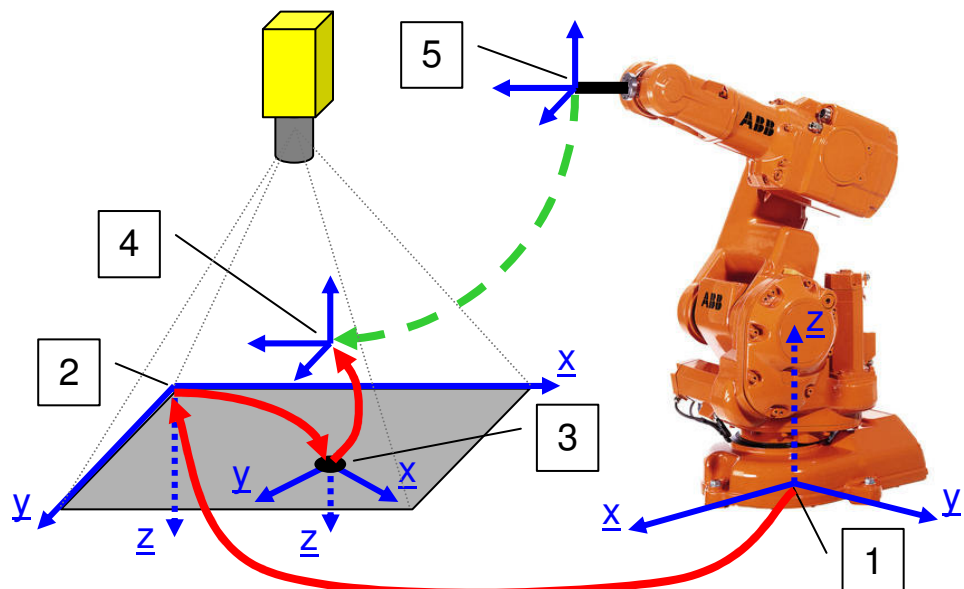


Figure 2. Chain of geometrical frames used in operation

1	Robot base frame (or world frame)
2	Work object frame (aligned with the camera frame after robot/vision calibration)
3	Program displacement frame, defined by vision transformation
4	Position frame to be reached by the robot tool, defined by the movement instruction
5	Robot tool frame

Implementation in a program displacement instruction (case of a fixed camera),

The program displacement instruction is “PDispSet” (refer to the “RAPID” language reference manual for further details).

Example of using vision transformation in a program displacement:

```
VAR pose pVision;  
...  
IF CX_GetVisionData(1,"F",22,pVision,nScore,nError) THEN  
    ! Utilization in a program displacement  
    PDispSet pVision;  
    ...  
    ! Robot movements in relation to object recognized by vision  
    ...  
    ! Re-initialization of program displacement  
    PDispOff;  
ELSE  
    ! Error Management  
    ...  
ENDIF
```

The “PDispSet” instruction is used to achieve a temporary geometrical shift of the program and is therefore well adapted to the use of an object’s coordinates provided by a camera, which will in principle change with every production cycle. .



Note:

A program displacement is reinitialized to a null transformation in the following situations:

- Execution of a “PDispOff” instruction,
- Re-initialization of the robot program (for example, with a manual displacement of the execution pointer at the main robot program or the loading of a new program).



Warning:

You should be careful when using program displacement, as it can easily be reset, especially during operations in manual mode. The reset of a program displacement can provoke unexpected robot movements.

Implementation in a work object frame (case of a fixed camera),

A work object frame is defined in “RAPID” language by the type “wobjdata”. This type of structured data includes several components, including two transformations “uframe” and “oframe”:

- The “uframe” component will be used to store the transformation defined at the end of robot/vision calibration (see details in section 7),
- The “oframe” component will be used to store the vision transformation associated with the current position of the object.

Example of using vision transformation in an object frame data item:

```
VAR pose pVision;  
! Declaration of a null transformation  
CONST pose pTRANSF0:=[[0,0,0],[1,0,0,0]];  
! Declaration of the vision transformation  
CONST pose pTRANSF0:=[[0,0,0],[1,0,0,0]];  
...  
IF CX_GetVisionData(1,"F",22,pVision,nScore,nError) THEN  
    ! Utilization of a work object frame data  
    wobjCamera.oframe:=pVision;  
    ...  
    ! Robot movements in relation to object recognized by vision  
    ...  
    ! Object frame reset  
    wobjCamera.oframe:=pTRANSF0;  
ELSE  
    ! Error Management  
    ...  
ENDIF
```



Note:

This method can only be used if the object frame is not associated with the movement of a mechanical unit (for example an external rotating axis) or a conveyor, otherwise the component “uframe” is reserved by the control system for real time storage of the work object frame coordinates displaced by the mechanical unit or the external axis. In this case use the program displacement method.

Implementation in a transformation calculation defined by the user (case of a fixed camera)

**Note:**

This third method is aimed at **experienced programmers**. An understanding of the following calculation functions in RAPID language is essential for the implementation of this method:

- “PoseInv”,
- “PoseMult”,
- “PoseVect”.

A possible utilization consists of recalculating the position of taking up the part by the robot as from the nominal take up position of the part (i.e. without shifting the object). Example of using the vision transformation in a transformation calculation defined by the user:

```
! Declaration of the vision transformation
VAR pose pVision;
! Declaration of the temporary calculation transformation
VAR pose pTemp;
! Declaration of take up position
CONST robtarget pPrise:=[[...], [...], [...], [...]];
VAR robtarget pPriseVision:=[[...], [...], [...], [...]];
...
IF CX_GetVisionData(1, "F", 22, pVision, nScore, nError) THEN
    ! Initialization of modified grip position
    pPriseVision:=pPrise;
    ! Initialization of temporary transformation
    pTemp.trans:=pPrise.trans;
    pTemp.trans:=pPrise.trans;
    ! Calculation of modified grip position
    pTemp:=PoseMult(pVision, pTemp);
    pPriseVision.trans:=pTemp.trans;
    pPriseVision.rot:=pTemp.rot;
    ! Robot movements in relation to object recognized by vision
    MoveL pPriseVision, v100, fine, tool1, wobj1;
    ...
ELSE
    ! Error Management
    ...
ENDIF
```

5 Implementing the communication library instructions

5.3 Example programs provided with the communication library

5.3 Example programs provided with the communication library

Overview

The communication library is equipped with 2 example files:

- One example robot program “TESTVISION.MOD”,
- One example vision program “TESTVISION.JOB”,

These 2 example files are designed to operate together for a quick implementation of communication between the robot and the “In-Sight” camera.

Example of robot program

The example robot program tests all of the functions in the communication library. The result of the communication exchanges is displayed in an interactive manner on the “FlexPendant” unit:

- Launch routine “TestProduction” to run without user interaction,
- Launch routine “TestProduction2» to run with user interaction.

Example of vision program

The example vision program “TESTVISION.JOB” operates with the example robot program, the assembly enables confirmation of the library functions.

The objects recognized by the camera are 3 x 6 sided nuts, size “M8” or “M10”; this object has voluntarily been selected for its simplicity and the robustness of the vision processing, and because they are easily found anywhere. Object detection should be on a light background (e.g. on a white paper).

Index	X pixels	Y pixels	Angle RZ	Score	Echelle	Etat	Status
0							
1	0.000	283.022	323.551	-3.007	99.850	100.005	1.000
2	1.000	133.707	164.403	-0.133	99.444	100.012	1.000
3	2.000	89.527	431.818	-17.715	99.188	99.948	1.000
4							
5							
6							
7							
8							
9							
10	0.000	283.022	323.551	-3.007	99.850	100.005	1.000
11	1.000	133.707	164.403	-0.133	99.444	100.012	1.000
12	2.000	89.527	431.818	-17.715	99.188	99.948	1.000
13							
14							
15							
16	0.000	283.022	323.551	-3.007	99.850	100.005	1.000
17	1.000	133.707	164.403	-0.133	99.444	100.012	1.000
18	2.000	89.527	431.818	-17.715	99.188	99.948	1.000
19							
20							
21							
22	0.000	283.022	323.551	-3.007	99.850	100.005	1.000
23	1.000	133.707	164.403	-0.133	99.444	100.012	1.000
24	2.000	89.527	431.818	-17.715	99.188	99.948	1.000
25							
26							
27							
28							
29							
30							
31							

5.4 IP address configuration for an IRC5 control system

Overview

IP address configuration for an IRC5 control system is necessary to communicate by Ethernet with the “In-Sight” camera. This configuration can only be carried out using the IRC5 “FlexPendant” unit (this operation cannot be carried out on the “Virtual Robot”).

The IP address can be consulted at any time, in two ways:

- Either by using the “FlexPendant” unit,
- Or by using the PC software “RobotStudio Online” which comes with the IRC5 control system.

Ethernet port to be used and default IP address configuration

The IRC5 control system comes equipped with 2 Ethernet ports available for the user:

- The “Service” port dedicated to maintenance, installation and programming operations,
- The “LAN” port (typically for production network).



Caution:

Only the “LAN” port should be used with a vision system (or any other external computer), as the “Service” port implements a “DHCP” server in the control system, which automatically allocates IP addresses. A connection to the “Service” port of the production network could make it totally inoperable!



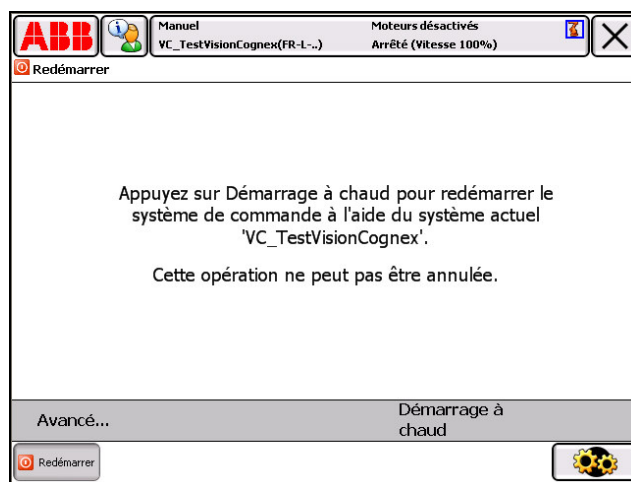
Note:

The IRC5 control system does not have a default IP address for the “LAN” port, an IP address must be defined in order to communicate with the camera.

IP address configuration using the “FlexPendant” unit

Procedure:

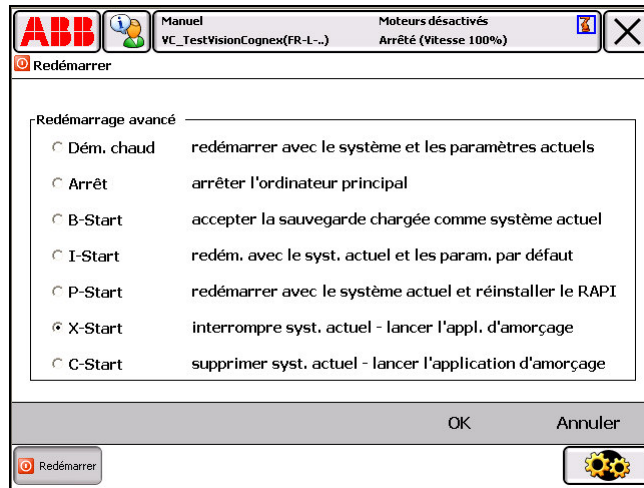
- Select the main “ABB” menu, and then the “Reboot” menu,
- The “Reboot” window opens. Select the “Advanced” menu.



5 Implementing the communication library instructions

5.4 IP address configuration for an IRC5 control system

- The “Advanced restart” window opens. Select the “X-Start” menu,



The “X-Start” command launches the IRC5 control system load application (“ABB Robotics Boot Application”). This window has four buttons:

- “Install System”,
- “Settings”,
- “Select System”,
- “Restart Controller”.

Press on the “Settings” button to access IP address configuration. The “Network Connection” window opens the target button “Use no IP address” is enabled by default.

- Select the “Use the following IP settings” target button to configure your IP address using the numeric keypad displayed on the left.
- Example of address configuration:
 - IP address: 10.33.48.250
 - Subnet mask: 255.255.255.0
 - Default gateway: 10.33.48.255

Press on the “Select system” button to select a system installed in flash memory.

Press on the “Restart Controller” button to boot the control system.

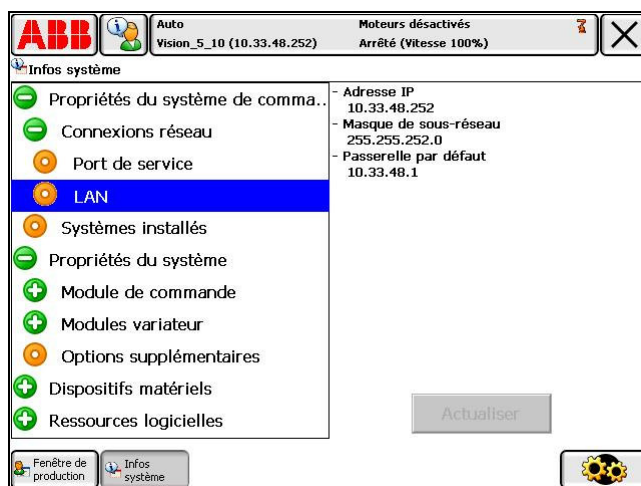
5 Implementing the communication library instructions

5.4 IP address configuration for an IRC5 control system

Consulting IP addresses using the “FlexPendant” unit

Procedure:

- Select the main “ABB” menu, and then the “System Info” menu,
- The “System Info” window opens. Select “Control system properties”, then “network connections” to view the IP addresses currently in use.



Note:

From the RobotWare 5.08 version, the main IP address is constantly on display on the system status bar. In the screen shot example above:

- System name: “Vision_5_10”
- IP address: 10.33.48.252.

This information is not available when using the virtual robot (only on a real robot controller).

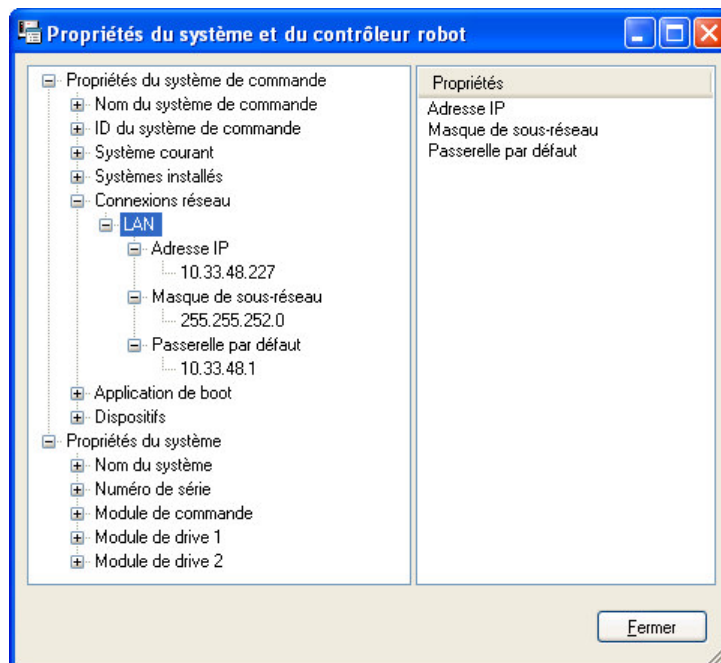
5 Implementing the communication library instructions

5.5 IP address configuration for an “In-Sight” camera

Consulting IP addresses using the “RobotStudio Online” PC software

Procedure:

- In the “Robot View” tree structure, select “Control Systems” and then right click and select “Add a control system” in the popup menu.
- Select your control system from the list, and it is added to the “Robot View” tree structure”. Its main IP address is displayed to the right of the system name.
- To access all of the IP address definition, select “Properties” in the popup menu, then “System and robot controller properties”, which will bring up the following dialogue box:



Note:

This procedure is not valid for a “Virtual Robot” which does not have an IP address on its own, as it uses that of the host computer running the “Virtual IRC5” or “RobotStudio 5” software.

5.5 IP address configuration for an “In-Sight” camera

Overview

IP address configuration for an “In-Sight” camera is achieved using the “In-Sight Connection Manager” utility supplied by the COGNEX company (please refer to corresponding user manual).



Note:

The selected IP address must be compatible with that of the robot. In the case of small networks, only the last byte of the IP address changes; this is what the “In-Sight Connection Manager” software suggests by default.

6.1 Error messages generated by the ABB communication library

6 Human-machine Interface “HMI”

Overview

The communication library does not implement a vision specific man-machine interface on the “FlexPendant” unit. The warning and error messages use the standard man-machine interface of the “FlexPendant” unit. There are two types of error message:

- Error messages generated by the ABB communication library,
- Error messages generated by the “In-Sight” camera.

6.1 Error messages generated by the ABB communication library

In this version, the error messages generated by the communication library are defined in accordance with the 3 topics shown below.

“Socket messaging” communication errors

Code	Description	Routine
-100	The socket initialization is not possible	CX_InitComm
-101	Writing data is not possible.	All routines
-102	Reading data is not possible.	All routines
-103	Communication timeout.	All routines

General vision application errors

Code	Description	Routine
-200	The camera number is incorrect (range: 1-4).	All routines using “CameraId”
-201	The cell data format of the vision program is incorrect (ABB format).	CX_GetVisionData
-202	The object bounds are out of range. Note: the value is outside of the minimum and maximum tolerances defined in the “CX_Setup_Camera” instruction.	
-203	The object score is too low. Note: the value is lower than the tolerance defined in the “CX_Setup_Camera” instruction.	
-204	The requested function not yet implemented in this version.	CX_GetInfo

Vision application errors concerning robot-vision calibration

Code	Description	Routine
-250	Incorrect number of calibration positions.	SetWObjVision
-251	Insufficient number of robot calibration positions.	SetWObjVision
-252	Insufficient number of vision calibration positions.	SetWObjVision
-253	Error computing vision frame; inaccurate or inconsistent calibration positions.	SetWObjVision

6.2 Error messages generated by the “In-Sight” camera

Vision application errors specific to the “In-Sight” camera

Code	Description	Routine
-300	The code returned by the camera is incorrect.	All routines
-301	The row identifier is incorrect (range: 0-399).	CX_GetValue CX_GetVisionData
-302	The column identifier is incorrect (range: A-Z).	CX_SetFloat CX_SetInteger CX_SetString

6.2 Error messages generated by the “In-Sight” camera

In this version, the error messages generated by the “In-Sight” camera are as follows:

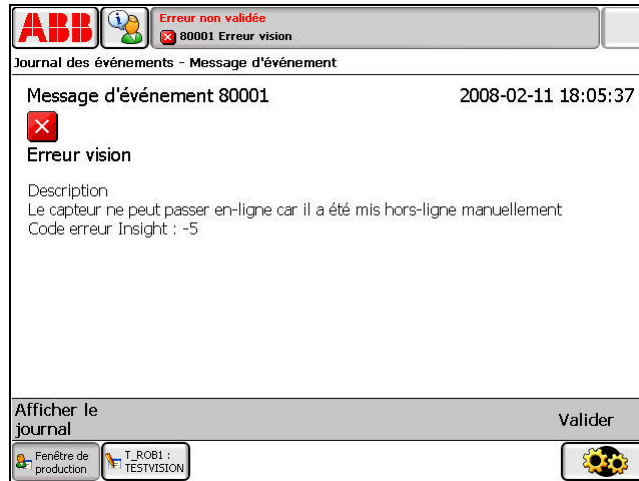
Code	Description	Routine
0	Unknown command.	All routines
-1	The number is either out of range (0 to 8) or not an integer.	CX_SendMessage CX_SetEvent CX_SetEventWait CX_SetOnline CX_TriggerImage
	The cell ID is invalid.	CX_GetValue
	The cell ID is invalid, or the specified value is not an integer.	CX_SetInteger
	The cell ID is invalid, or the specified value is not a float number.	CX_SetFloat
	The job filename is missing.	CX_LoadFile
-2	The command could not be executed, or the vision system is offline.	CX_GetInfo CX_GetValue CX_SendMessage CX_SetEvent CX_SetEventWait CX_SetFloat CX_SetInteger CX_SetOnline CX_SetString CX_TriggerImage
	The active job has not been saved, thus the command could not be executed	CX_GetFile
	The job failed to load, the system is online or file was not found.	CX_LoadFile
-4	The job to be loaded was not found, or the sensor is out of memory.	CX_LoadFile
-5	The sensor cannot go online as it has been taken offline manually.	CX_SetOnline
-6	The user does not have full access to execute the requested command.	CX_LoadFile CX_SendMessage CX_SetEvent CX_SetEventWait CX_SetFloat CX_SetInteger CX_SetOnline CX_SetString CX_TriggerImage

**Note:**

The error codes <-1> and <-2> returned by the “In-Sight” camera take on a different meaning depending on the communication routines used.

6.3 Information message generated by the communication library functions

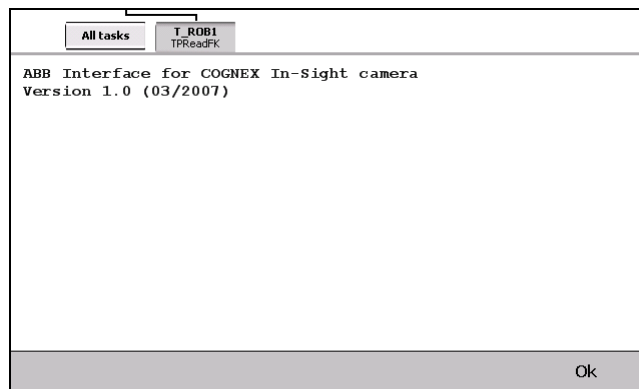
Example of display in the event of a camera that has stayed in “offline” mode (HMI configured in French):



6.3 Information message generated by the communication library functions

Only the “CS_ShowVersion” instruction generates an information display on the “FlexPendant” unit in the operator window (see section 4.1.9).

Result of this display on the “FlexPendant” unit:



6.4 “In-Sight” camera image display on “FlexPendant” unit

6.4 “In-Sight” camera image display on “FlexPendant” unit

The main utilization of such feature is to display defective objects so that the operator can easily monitor any unanticipated unknowns in production and take appropriate action.



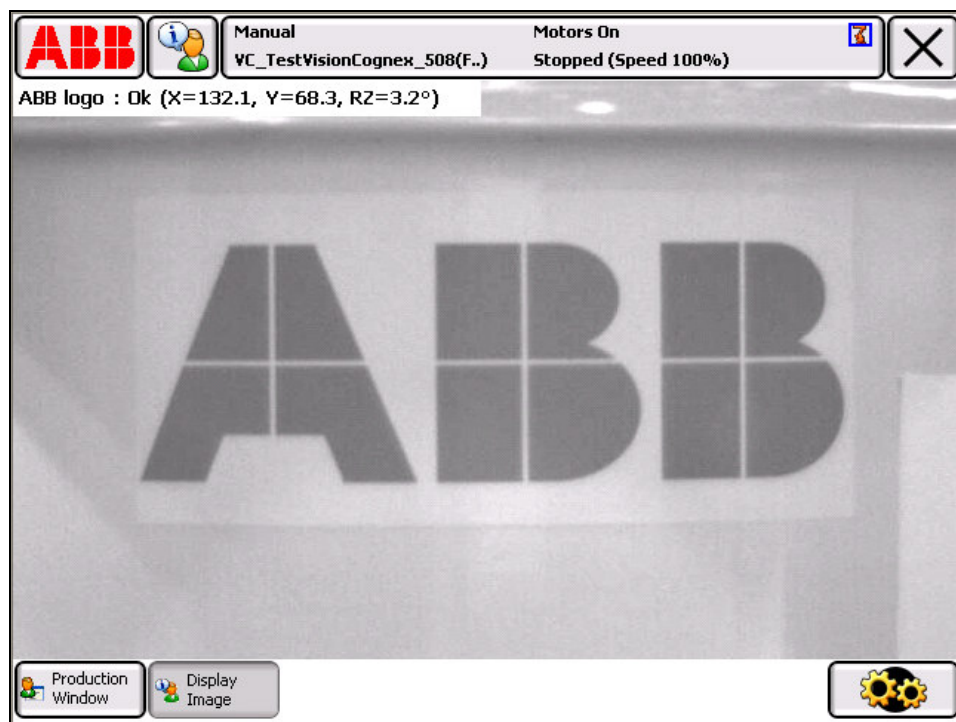
Note:

This feature is not yet available as a product, this is mentioned here as an example of what is technically possible.

The main implementation of this function is as follows:

- The robot program requests an image capture,
- The vision program detects an anomaly in the object and saves the error to memory,
- The vision program sends the image to the IRC5 control system with the instruction “WriteImageFTP” (see section 8.3),
- The robot program detects the anomaly when requesting the object coordinates,
- The robot program requests display of the image transmitted by the vision program on the “FlexPendant” unit.

Example of a preliminary display, created for a demonstration during a trade fair:



7 Robot-vision calibration procedures

7.1 Implementing a vision system for guiding a robot

Image surface calibration

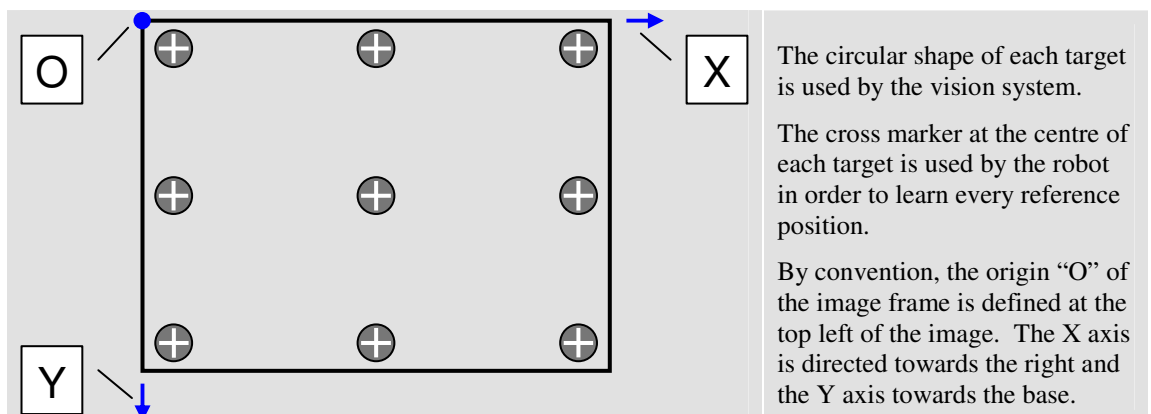
The utilization of a vision system to carry out a robot guiding operation requires the position coordinates of an object expressed in metric units:

- The native position of an object is always measured in pixel units in the sensor frame.
- A “scaling” operation in the metric system is required in order to get the Cartesian coordinates expressed in mm that can be used by the robot control system. This operation is an **image plane calibration**, and depends on several factors:
 - The geometry of the camera sensor (number and shape of pixels, etc.),
 - The field angle of the lens (defined by its focal for a given sensor size),
 - The lens focusing distance,
 - Any correction to optical distortion by the lens.
- This image surface calibration operation must be carried out **before** the robot – vision calibration operation.
- This documentation does not deal with achieving this image plane calibration, please refer to the COGNEX documentation for the “In-Sight Explorer” software:
 - For In-Sight Explorer version 3.2 and lower: check the following vision functions: “Calibrate” (which implements 4 points of reference) and “CalibrateAdvanced” (which implements up to 9 points of reference),
 - For In-Sight Explorer version 4.0 and higher: check the new “EasyBuilder” wizard that has an integrated automatic calibration function.

In all cases, this image surface calibration operation requires the implementation of a **calibration test pattern** which will be processed by the vision system:

- The size of the test pattern should be appropriate to the area of the image surface,
- The test pattern is composed of several targets that are recognized by the vision system. These targets are often circular and their centre materializes a reference point,
- The position of each target on the test pattern has to be known (or measured) in a frame on the image surface,
- The calibration algorithm of the image surface uses a pair of coordinates for each target to calculate the calibration parameters:
 - 2 camera coordinates (X, Y) in pixels,
 - 2 coordinates (X, Y) in the image surface in mm.
- In practice, this test pattern should also be usable to teach positions with the robot.

Example of a 9 target calibration test pattern with display of the (O, X, Y) frame of the image plane



7.2 Robot – vision calibration in the case of a fixed camera

Overview

The robot – vision calibration procedure is essential in order to achieve precise robot movements in relation to the coordinates transmitted by the vision system. This procedure enables precise definition of the position and direction of the camera frame in relation to the robot base frame (or world frame).

Hardware and geometrical frames implemented in the case of a fixed camera

In the case of a fixed camera, the procedure requires the implementation of a **calibration test pattern** that will be learned simultaneously by the robot and the vision system. The calibration pattern is placed in the field under camera observation at the nominal distance for lens focusing. The test pattern is made up of several geometrical references where the coordinates are known (see detailed definition in next section).

The hardware and the frames implemented by the robot/vision calibration procedure illustrated in the following figure:

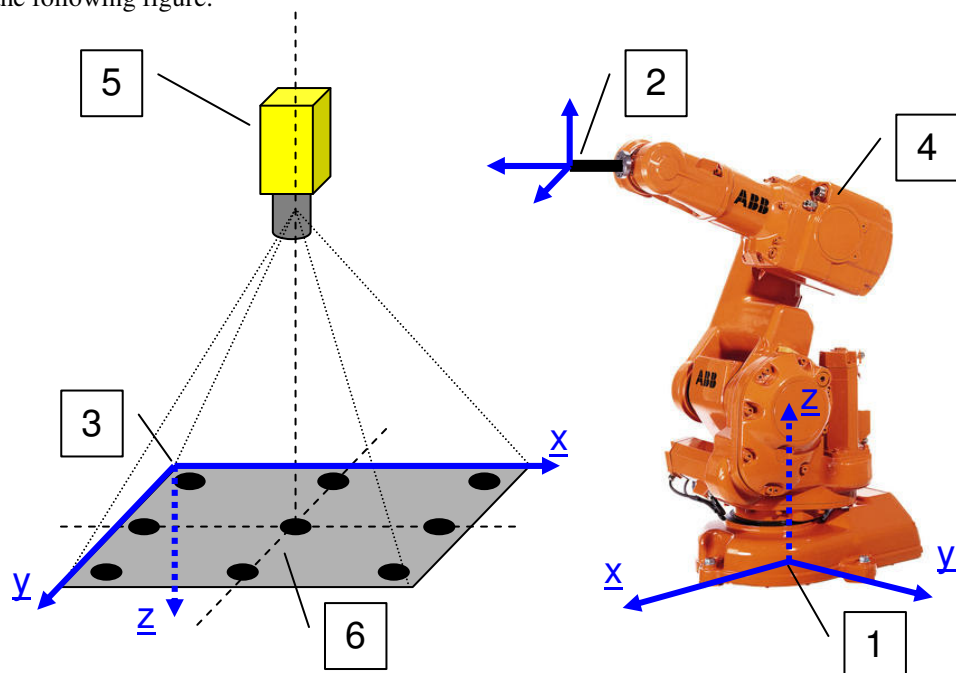


Figure 3. Geometrical frames implemented in the case of a fixed camera

1	Robot base frame
2	Robot tool frame
3	Robot work object frame (this frame should coincide with the camera frame)
4	Robot
5	“In-Sight” camera “5000” series with lens
6	Calibration test pattern with multiple geometrical references (9 targets in this example) located in the image surface of objects to be processed.

7.2 Robot – vision calibration in the case of a fixed camera

Implementing the robot – vision calibration calculation in the case of a fixed camera

The aim of robot – vision calibration in the case of a fixed camera is to compute a **work object frame** that coincide with the camera frame, in such a way that the 2D coordinates (X, Y, RZ) supplied by the vision system can be directly used as displacement in the work object frame.

Robot – vision calibration requires the implementation of a **calibration test pattern** as described in the previous section. Two methods are possible:

- Definition of the work object frame directly on the calibration test pattern by learning 3 points, as long as the origin and the X and Y axes are materialized on the test pattern:
 - Point 1 defines the origin of the object frame,
 - Point 2 defines the X axis of the object frame,
 - Point 3 defines the Y axis of the object frame,
- Definition of the object frame by several **pairs** of coordinates for each target (3 minimum):
 - 3 coordinates (X, Y, Z) in mm of the object position in the robot base frame,
 - 2 coordinates (X, Y) in mm of the object position in the vision frame.



Note:

The second method is recommended: it is easier to implement in practice and it is more accurate.

Example of learning routine of calibration positions in the case of a fixed camera

Example with pairs of coordinates, provided in the “TESTVISION.MOD” robot program.

```
! Number of calibration positions
CONST num nbCALPOS:=9;
! Declaration of calibration positions as an array
CONST robtaraget pCalRobot{nbCALPOS}:=[ ... ];
! Declaration of vision measurement transformations as an array
PERS pose pCalVision{nbCALPOS}:=[ ... ];
VAR num nError;
VAR num nError;
VAR bool bOk;
...
PROC Calibration()
    ! Movement on calibration positions
    MoveL pCalRobot{1},v20,fine,tool1;
    Stop;
    MoveL pCalRobot{2},v20,fine,tool1;
    Stop;
    ...
    MoveL pCalRobot{9},v20,fine,tool1;
    Stop;
    ...
    ! Vision coordinates read
    bOk:=CX_GetVisionData(1,"F",21,pCalVision{1},nScore,nError);
    bOk:=CX_GetVisionData(1,"F",22,pCalVision{2},nScore,nError);
    ...
    bOk:=CX_GetVisionData(1,"F",29,pCalVision{9},nScore,nError);
    !
```

7 Robot-vision calibration procedures

7.2 Robot – vision calibration in the case of a fixed camera

```
! Robot-vision calibration calculation
SetWobjVision pCalRobot,pCalVision,nbCALPOS,wobjVision,nError;
ENDPROC
```

7 Robot-vision calibration procedures

7.2 Robot – vision calibration in the case of a fixed camera



Note:

The “SetWobjVision” instruction is provided by the “INSIGHT.SYS” module.



Advice:

The interest of separating image surface calibration from robot – vision calibration is to make for easier maintenance of the installation in the event of intervention on the camera:

- The camera and/or lens can be changed without having to re-run robot – vision calibration.
- You just need to re-position the calibration test pattern with the required degree of precision and re-start vision processing.

7.3 Robot – vision calibration in the case of a camera mounted on the robot

7.3 Robot – vision calibration in the case of a camera mounted on the robot

Overview

The aim of the robot – vision calibration procedure, in the case of a camera mounted on a robot, is identical to the procedure described in the previous section, but implementation is different, especially regarding the calibration equipment and frames used. This procedure enables precise definition of the position and direction of the camera frame **in relation to the robot tool flange** (“tool0”).

Hardware and geometrical frames implemented

In the case of a camera mounted on the robot, the previously implemented test pattern will be replaced by a spherical reference to enable a wide range of camera rotation movements around the centre of the sphere.

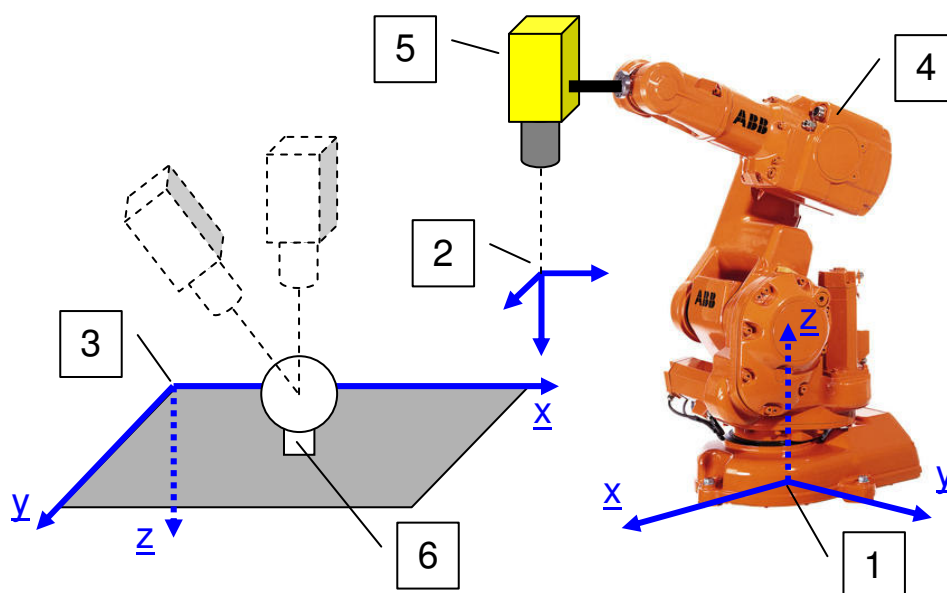


Figure 4. *Geometrical frames implemented in the case of a camera mounted on the robot*

1	Robot base frame
2	Robot tool frame (this frame should coincide with the camera frame)
3	Robot work object frame
4	Robot
5	“In-Sight” smart camera with lens
6	Calibration plate with spherical reference

Implementing the robot – vision calibration calculation in the case of a camera mounted on the robot

The aim of robot – vision calibration in the case of a camera mounted on the robot is to compute a **tool frame** aligned with the camera sensor frame. There are 2 possible calibration methods:

- A simplified method which requires a calibration plate with a spherical reference; this has similarities with the tool frame calibration procedure supplied with the IRC5 control system.
- An advanced method which requires a specific calibration plate.

The implementation of this procedure is more complex than in the case of a fixed camera. Contact ABB France Robotics Division for technical support relating to this operation.

8.1 Using the “Virtual Robot” to test a vision application

8 Specific Procedures

8.1 Using the “Virtual Robot” to test a vision application

Overview

The “Virtual Robot” enables testing of a communication application between the IRC5 control system and an “In-Sight” camera on a PC compatible computer, without having to have the actual robot in hand.

This type of offline test is especially interesting for the integrator who must prepare a new project for a customer, or an end user who wishes to test a modification without stopping production.

Implementing the “Virtual Robot”

The “Virtual Robot” for the IRC5 Control System is available with the software packages:

- “Virtual IRC5”, or
- “RobotStudio 5.0”.

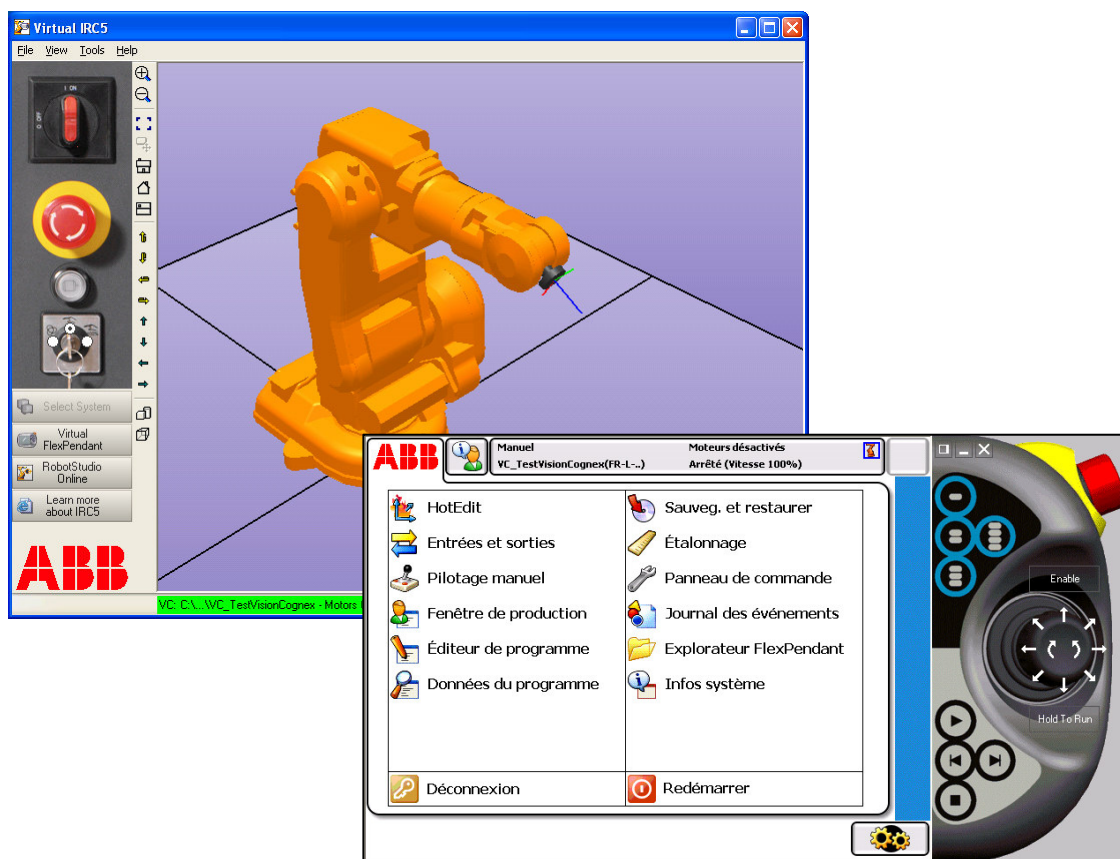


Figure 5. “Virtual IRC5” software



Advice:

You can get further information on the “Virtual Robot” and the ABB software products on the following Internet site:

<http://www.abb.com/roboticssoftware>

8.1 Using the “Virtual Robot” to test a vision application

Sequence of operations to be carried out to operate the “Virtual Robot”:

- Connect the Ethernet cable of the “In-Sight” camera to the PC compatible computer,
- Connect the “In-Sight” camera’s power lead to the extension module linked to a 24 V power supply,
- Configure an instance of the “Virtual Robot” that matches the IRC5 control system configuration to be simulated, either with the using the “RobotStudio Online” software (up to RobotWare version 5.10) or with the “RobotStudio” software (from RobotWare version 5.11 and higher),
- Configure the “In-Sight” camera IP address (the address must be compatible with the PC setup) using the COGNEX “In-Sight Connection Manager” software,
- Start the “Virtual Robot” using the “Virtual IRC5” or “RobotStudio 5” software,
- Load and run the robot / vision communication program in the “Virtual Robot”.

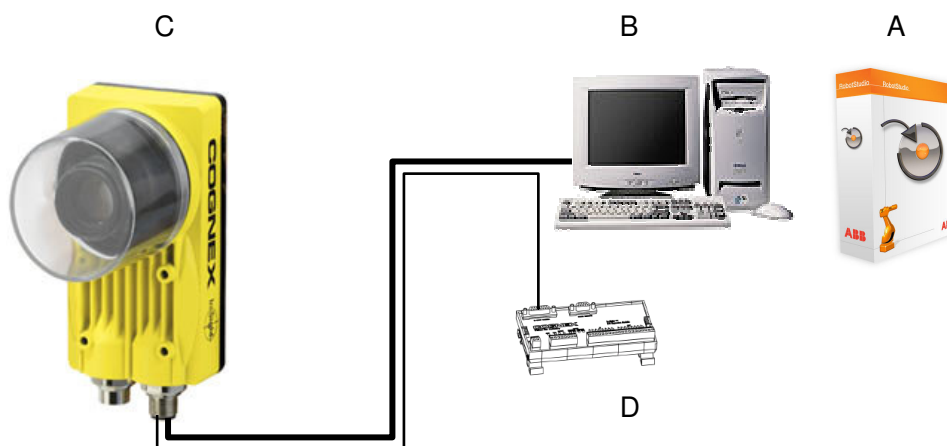


Figure 6. *Hardware architecture with a PC and the “Virtual Robot”*

A	Virtual Robot – “Virtual IRC5” or “RobotStudio 5.0” software
B	PC compatible computer
C	“In-Sight” smart camera
D	“In-Sight” smart camera extension module for 24 V power supply

Usage restrictions

The “Virtual Robot” simulates the IRC5 control system to be simulated, in a directory defined by the user. Due to this, this directory cannot be the root of the “Virtual Robot” installation disk, certain operations on the files created on the real robot cannot be run as such by the “Virtual Robot” without adaptations.



Note:

In the event of a communications application between an IRC5 control system and an “In-Sight” camera, a standard example will be the writing of an image by the camera into the flash memory of the IRC5 control system using the “WriteImageFTP” function: In this case, you must use a modified destination folder to write the file, in relation to the directory selected by the user for the IRC5 control system to be simulated.

8.2 “Debug” mode

Overview

The “Debug” mode facilitates the debugging of a vision application by displaying all of the communication frames exchanged between the robot and the “In-Sight” smart camera on the teach pendant, in a text format operator dialogue window.

Implementation

The “Debug” mode is implemented with the instruction “CX_SetDebugMode” and one of the 2 mutually exclusive parameters “On” | “Off” (see reference description of instruction in section 4.1.7).

Display in the “operator dialogue” window

Example of display on the “FlexPendant” unit in “debug” mode (HMI configured in English) :

- Open a communication session with an “in-Sight” camera, with user name (“admin” by default) and password (blank by default),
- Read command for name of program running (“GF” for “GetFile”), the camera returns “1” (for positive acknowledgement), then program name (“TestCameraInsight.job”).

All tasks	
T_ROB1->	Socket start client done...
T_ROB1->	Com. vision : Welcome to In-Sight(tm) 5400 Session 0
User:	
T_ROB1->	Com. robot : admin
T_ROB1->	Telnet user name done...
T_ROB1->	Com. vision : Password:
T_ROB1->	Com. robot :
T_ROB1->	Telnet password done...
T_ROB1->	Com. vision : User Logged In
T_ROB1->	Com. robot : GF
T_ROB1->	Com. vision : 1
TestCameraInsight.job	
Effacer	Ne pas afficher les journaux

8.3 Image transfer from “In-Sight” camera to IRC5 control system

8.3 Image transfer from “In-Sight” camera to IRC5 control system

Overview

The transfer of images from the “In-Sight” camera to the IRC5 control system is achieved with the “FTP” communication protocol. This opportunity is mainly used to view and save the image of defective objects on an Ethernet network computer.

**Note:**

Image display on the “FlexPendant” unit is not yet available as a standard product, but is technically possible.

Viewer programming

The transfer of images from the “In-Sight” camera to the IRC5 control system is achieved with the instruction “WriteImageFTP” (refer to “In-Sight Explorer” software). List of parameters to handle:

Host name	Use IP address of IRC5 control system for Ethernet “LAN” port.
User name	By default: no user is configured for the IRC5 control system.
Password:	By default: not used.
Image structure	By default: \$A\$0 (coordinate of the image acquisition cell in the vision program).
File Name	The image is saved into the “flash” memory of the IRC5 control system. Disk unit “hd0a”. Add directory name and image file name.
Max value	Use “0” to always write into the same image, otherwise limit the number of images to avoid saturating the “flash” memory.
Reset	Always use this option to avoid writing an unlimited number of images.
Data format	Use “JPEG” format (select: JPG) to optimize memory used.

is5400_016285 - Feuille de propriétés - WriteImageFTP

Edition Insérer Aide

Nom d'hôte 192.168.100.250

Nom d'utilisateur anonymous

Mot de passe

Structure Image \$A\$0 = Image

Nom du fichier hd0a/IRB6620/Image

Valeur max 0

Remise à zéro ☒

Format des données JPG

Capture d'écran ☐

Nom du fichier
Chemin et nom de fichier.

OK Annuler

8.3 Image transfer from “In-Sight” camera to IRC5 control system

9 Release Notes

Overview

This section lists all of the software changes achieved on the “Vision interface for COGNEX In-Sight for IRC5 control system”.

Version 1.0.0 (April 2007)

New functions:

- First version available as official product.
- Addition of Dutch interface (in addition to French and English).

Version 1.0.1 (May 2007)

New functions:

- Addition of German interface.

Corrections:

- Correction of an index error that led to the display of an error message about the camera number (number out of range 1 to 4).

Version 1.0.2 (February 2008)

New functions:

- Addition of Spanish interface.
- Addition of Italian interface.
- Addition of “CX_SetFile” instruction.
- Addition of “SetWObjVision» instruction.

Corrections:

- Correction of a display error for error number -103 and -300.

Documentation:

- Update in accordance to changes above.

Version 1.0.3 (April 2009)

New functions:

- Addition of Swedish interface.
- Possibility to load selected localization modules (instead of all modules in earlier releases).

Documentation:

- Update in accordance to changes above.
- Update of documentation format,
- Correction of the description of the “SetToolVision” instruction.
- Added references to new version of the “In-Sight Explorer” software with the “EasyBuilder” wizard.

10 Index and tables

10.1 Index

Section not used in this version of documentation.

10.2 Illustrations Table

Figure 1. *Hardware architecture with an IRC5 control system* 11

Figure 2. *Chain of geometrical frames used in operation*..... 57

Figure 3. *Geometrical frames implemented in the case of a fixed camera*..... 71

Figure 4. *Geometrical frames implemented in the case of a camera mounted on the robot* 75

Figure 5. *“Virtual IRC5” software* 76

Figure 6. *Hardware architecture with a PC and the “Virtual Robot”* 77