Università degli Studi di Padova

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DEPARTMENT OF INFORMATION ENGINEERING

—

MASTER DEGREE IN
CONTROL SYSTEMS ENGINEERING

Design of a Process Management System for a Conveyor Tracking Task with Computer Vision

Supervisor: Prof. Giovanni Boschetti
Co-Supervisor: Prof. Riccardo Minto

Student: Ettore Santoiemma
ID Number: 2096413

ACCADEMIC YEAR 2023 - 2024
Graduation Day 10/10/2024

Alla mia famiglia,
alla mia ragazza,
ai miei amici.

Contents

Sommario

I sistemi Conveyor Tracking sono molto diffusi nel campo dell'automazione industriale poiché consentono di migliorare la produzione di massa e ridurre i tempi di produzione in molti diversi tipi di processi industriali.
In queste applicazioni, la disciplina della Computer Vision ha un ruolo importante, poiché consente maggiore flessibilità e molte funzionalità che possono essere utili per l'implementazione dei sistemi di Conveyor Tracking.
Lo scopo di questo progetto è proporre una soluzione generale al problema del Conveyor Tracking con l'ausilio della Computer Vision, progettando il miglior sistema di gestione dei processi possibile.
Nel documento viene spiegata la soluzione con tutte le ipotesi e le scelte effettuate al fine di ridurre tutti gli errori e ottimizzare il comportamento del sistema.
Verrà mostrata anche l'implementazione di uno specifico problema di Conveyor Tracking, presentando tutte le parti utilizzate e tutti i vincoli che ne presentano l'applicazione reale. In particolare verranno illustrate le differenze nell'utilizzo di due diversi sistemi di visione, evidenziando le principali problematiche e le similitudini dei due approcci.

Abstract

The Conveyor Tracking systems are very diffused in the field of the industrial automation since they allow to improve the mass-production and reduce the production's time in many different type of industrial process.
In this applications, the Computer Vision discipline has an important role, due to the fact that allows more flexibility and a lot of features that can be useful for the implementation of the Conveyor Tracking systems.
The aim of this project is to propose a general solution to the Conveyor Tracking problem with the aid of Computer Vision, designing the best possible process management system.
This paper explains the solution with all the assumptions and the choices made in order to reduce all the errors and optimize the behavior of the system.
It will be showed also the implementation of a specific Conveyor Tracking problem, presenting all the parts used and all the constraints that has the real application. In particular, it will be shown the discrepancies in the exploiting of two different Vision Systems, highlighting the main differences, the problems and the similarities of this two approaches.

Introduction

The objective of this thesis is to propose a solution to the Conveyor Tracking problem, with the aid of a Computer Vision System. This case of study has been chosen for the wide use of this kind of Conveyor Tracking System in the nowadays industry, trying to develop the best possible solution with the given experimental setup.

The first Chapter presents all the necessary theoretical knowledge used for the implementation of the system and employed in the discussion of the solution. The main topic outspread in the first section is the Conveyor Tracking problem, in order to give the most complete theoretical background to the arguments that has been taken into account to better discuss the issues that may occur during its exploitation.
Then, it is also presented the Computer Vision argument and the Transformation Matrix problem, in order to give the main knowledge regarding the tools used in completing the solution.

After the general theoretical explanation, the second Chapter explains the formulated solution to the Conveyor Tracking problem with the utilization of Computer Vision. This sections discuss all the main components of the system, in particular it is looked into the implementation of the solution with two type of Camera System based on the exploitation of the matrix camera or the linear camera.
An another important aspect presented in this Chapter is the List Management tasks, utilized to keep memory of all the object to capture with the Industrial

Robot and the implementation of the Robot Movement task in two particular scenarios: with object *priority* or without object *priority*.
The solution has the scope of optimizing as much as possible the process management, in order to make all the subsystems working in the best possible conditions and to reduce to the minimum all the errors that can occur in the tracking problem.

The third Chapter illustrates all the experimental setup exploited for the implementation of the purposed solution. In particular it is discussed all the assumptions needed for the adaptation of the general solution to the real system. The contention goes through all the settings and the specific formulas and software necessary to implement correctly the solution. An important argument handled in this sections is about the specific transformation matrices computation for the two types of camera exploited in the solution and the actual Conveyor Tracking function used in the problem.

Finally, fourth Chapter presents all the results obtained with the real system, highlighting the problems and the limitation that the tests has reported. In particular, there are discussed the data obtained with the two different Camera System implemented, with the final objective in point out the difference between these two resolutions, and trying to figure out which could be more efficient in the described problem.

Chapter 1

Overview of the problem

This Chapter presents the basics of the Conveyor Tracking problem and the basics of Computer Vision in order to give a proper background to the arguments explained in the text.
The first section is dedicated to the Conveyor Tracking problem, which is the main argument of the treatment, to make sure that the problem is well defined and the purpose of the thesis is properly highlighted. In this section it will be given a general overview of the system that exploit the Conveyor Tracking with the focus on the components of the model and their role in the mechanism.
The second section is focused on the Computer Vision task, in order to give some general notations and conventions that are used during the following chapters of the treatment, with the focus on the basics of the Blob Analysis method and on its applications in the image analysis and the detection of objects for the industrial application.
Finally the last section talks about the Transformation Matrix argument, in order to recall all the main definitions and properties of the computation of a transformation matrix which have been used in the design of the solution of the problem and in particular in the implementation in the real system.

1.1 Conveyor Tracking

The Conveyor Tracking is a widely used approach for industrial scenarios to make the production more efficient, since it makes possible to the Industrial Robot to interact with moving objects, in order to manipulate them without stopping the production process, thus reducing the cycle time.
The general system that exploit this type of problem is composed by three main components [1]:

- the *Conveyor Belt* that provides the motion of the objects
- the *Vision System* to locate the objects
- the *Industrial Robot* to grab the objects

Conveyor Belt

The first important aspect to notice is that the position of the belt is known at each moment of time. This is crucial to know also the positions of the objects and to make the robot reach the target at the correct time.
One of the most common way to measure the position of the belt is with the use of an encoder. This component senses the movement of the conveyor belt and this information is used by the controller of the robot to reconstruct the location of the items at each time.

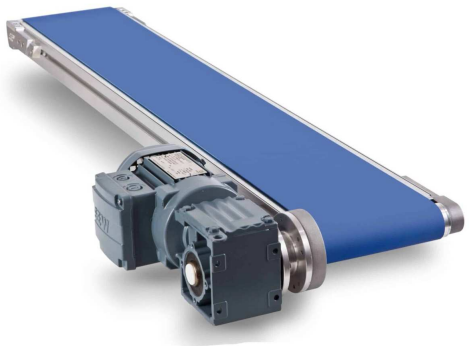


Figure 1.1: Conveyor Belt Example

In the Conveyor Tracking problem, it is known the position of sensing of the objects, which means that when a new object is detected on the belt, it is also known the pose in that time instant, and observing the new position of the belt the controller is able to correctly estimate the amount of space travelled by the piece, and hence its position with refer to the global reference system.
It is important to mention also the possible errors that can occurs in this branch of the system. In particular, the most common issues regards the tracking errors that can happen due to the mounting methods of the encoder on the belt.

Vision System

The main principle of the Conveyor Tracking problem is to be able to detect and track the objects that are travelling on a conveyor belt, in order to make the industrial robot grab the items within certain constraints and manipulate them. The simplest way to sense the object that are travelling on the belt is using one or more photocells, that detects the presence of an object that are passing through a certain position. The main drawbacks are that the objects must be placed always in the same position and orientation, since this kind of sensors only allow to know the location with refer to the axis of movement of the belt.
To extend the applications of this type of approach it has been introduced a vision system in order receive more information and to eliminate all the constraints on the position and orientation of the objects on the belt.

Figure 1.2: Example of Vision Systems

The vision system consist of some instrumentation that allows to obtain a picture of the belt and the objects that are moving on it. There are several way to acquire an image, the most used one is the implementation of an industrial camera. There are also different types of cameras that can be chosen according to the task that have to be performed.

The utilization of the Vision System can introduce also some issues in the detection task of the system. In particular, to exploit the artificial vision system it is necessary to satisfy certain conditions, in order to have good accuracy in the location task.
The main problem is about the light disturbances that can occurs during the sensing. External lights interference can produce errors in the estimation of the position of the objects or, in extreme case, make the objects no longer recognisable.

Industrial Robot

The task of the industrial robot, given the information from the vision system and the position of the conveyor belt, is to reach the location of the object on the belt.
It is important to notice that the robot has its own reference system, which is different from the one of the image taken by the vision system. Then it is important to give to the controller of the robot the correct coordinates, by means of the computation of the transformation matrix that translates the information given by the sensing system to the global reference system of the robot.

Figure 1.3: Industrial Robot Example: Adept Quattro s650H

The errors that may occurs due to the industrial robot are much smaller than the ones discussed in the previous branches of the system. The main problems arises due to the grasping device.
The most used way to activate the end-effector of the industrial robot is the

compressed air pressure. This kind of approach may introduce some issues since there could be some delays in the activation of the device, but also there could be pressure losses that can influence the activation time of the gripper in a negative sense.

1.2 Computer Vision

Given the *Vision System* from the previous section, the computer, that communicate with the controller of the robot, receives an image to analyze, in order to detect if there are pieces or not on the belt.
The aim of the Computer Vision (CV) System is the image analysis and, in particular for the problem of Conveyor Tracking, the location of the items in the image according to a reference system defined on the image.
In general, the CV algorithms are able to manipulate images in order to modify some parameters or to obtain several type of information for a vast number of uses, from the most simple application to very complex tasks.
It is important to mention that, talking about Computer Vision tasks, there are three main problems that the algorithms have to solve:

- *Recognition*: one or more preset objects can be assigned to generic classes, usually with their position in the frame
- *Identification*: a specific instance of a class is found
- *Detection*: the image is scanned in order to find a specific condition

Given that definitions, it can be notice that, for the execution of the Conveyor Tracking problem, the *Vision System* has to accomplish the *Recognition* task, to make sure that the items on the belt are correctly localized with refers to the global reference system.

1.2.1 Digital Image Acquisition

A digital image is defined as a two-dimensional function $f(x, y)$ where its value at the spacial coordinates (x, y) is a scalar quantity, which is proportional to the

energy radiated by the physical source captured in the picture.
The energy is adsorbed by the camera sensors and mapped into a scalar value. Typically, for a monochrome image, $f(x, y)$ is an integer between 0 and 255, starting from total black to total white.
A picture can be seen not only like a two dimensional function, but also like a matrix defined as:

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,n-1) \\ f(1,0) & f(1,1) & \dots & f(1,n-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(m-1,0) & f(m-1,1) & \dots & f(m-1,n-1) \end{bmatrix} \tag{1.1}$$

where it has been defined a $m \times n$ pixel image.
From the image digitization, knowing the parameters m and n, it is necessary to determine the parameter l which is the number of discrete intensity levels.
The only constraints in the choice of l is that it must be an integer number like m and n, and it is usually choosen as:

$$l = 2^k \tag{1.2}$$

where k is an integer. Then all the discrete levels are equally distributed along the range $[0, l-1]$.

1.2.2 Computer Vision Processing

There are several approaches to perform the inspection of pictures, in particular the *Blob Detection method* is widely used. This method allow to detects points and/or regions in which the parameters of the image differs from the surrounding environment.

Image Segmentation

The *Blob Detection* can be accomplished in many different ways, depending on the requirements that are needed for the specific application, like precision, com-

putation speed and complexity.

The most simple approach to the problem is the *Image Segmentation method* which is based on the principle of partitioning a digital image into multiple image segments, in order to simplify the representation of the image into something easier to analyze.

The result of the segmentation is a set of segments that cover all the image. In these regions the pixels have similar characteristics or computed properties, like color or intensity, while the adjacent regions are significantly different referring to the same characteristics.

The simplest segmentation that can be done, given the gray-scale image defined in the previous section, is imposing a **threshold** on the values of the matrix elements.

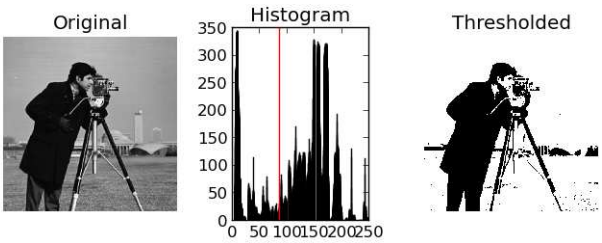


Figure 1.4: Monochrome Image Segmentation

The choice of the threshold is free, but there exist also a method to automatize this the segmentation called *Otsu's Method*. This method is useful to find the best threshold to apply in the segmentation since it maximizes the inter-class variance and minimizes the intra-class variance.

The main constraint of this approach is that, to identify correctly the objects, there must be an high contrast between the color of the object and of the ambient, to make the segmentation the more precise as possible and reduce the errors in the pose estimation.

1.3 Transformation Matrix

Talking about robot's tasks, it is important to know how to translate the pose of a rigid body from a reference system to another one, in order to obtain all the com-

ponents of the system that refers to the same triple of axes. In particular, given the Computer Vision program described in the previous section, the image analysis returns the *location* and *orientation* of the detected object with respect to the Camera Reference Frame which can be different from the global reference system.

Then given a *rigid body*, that in this application is the object to be captured by the Industrial Robot, it is completely described in the space by its *position* and its *orientation*, with respect to a reference. Putting together the position and the orientation of the rigid body it's obtained the *pose representation* [2].

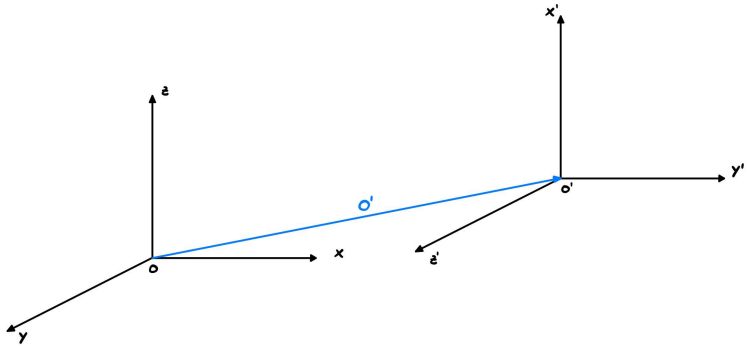


Figure 1.5: Transformation between two reference frames

With refer to Figure 1.5 it can be written the position and the orientation as follows:

$$O' = \begin{bmatrix} O'_x \\ O'_y \\ O'_z \end{bmatrix} = O'_x x + O'_y y + O'_z z \qquad v' = \begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} = v'_x x + v'_y y + v'_z z \quad (1.3)$$

The simplest transformation matrix that can be defined is the *Rotation Matrix*. Given the definition of the orientation above 1.3, the rotation matrix can be computed as:

$$R = \begin{bmatrix} x' & y' & z' \end{bmatrix} = \begin{bmatrix} x'_x & y'_x & z'_x \\ x'_y & y'_y & z'_y \\ x'_z & y'_z & z'_z \end{bmatrix} \quad (1.4)$$

It is also important to mention that the main property of the rotation matrix is that the inverse of the matrix R^{-1} is equal to the transpose of the matrix R^T. This implies also that $R^T R = I_3$ called *orthogonality condition*.
The matrix R belongs to the *special orthonormal* group $SO(m)$ of the real $m \times m$ matrices with orthonormal columns and determinant equal to 1.

1.3.1 Elementary Rotation Matrices

The simplest rotations matrices that can be written are the ones that describes the rotation of the reference frame around one of the coordinate frame. Given the reference system $O - xyz$ there can be defined three elementary rotation matrices, describing the rotations about the axis x, the axis y and the axis z [2]. Let $O' - x'y'z'$ be the rotate frame, the matrices obtained are:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & -sen\alpha \\ 0 & sen\alpha & cos\alpha \end{bmatrix} \tag{1.5}$$

$$R_y(\beta) = \begin{bmatrix} cos\beta & 0 & sen\beta \\ 0 & 1 & 0 \\ -sen\beta & 0 & cos\beta \end{bmatrix} \tag{1.6}$$

$$R_z(\gamma) = \begin{bmatrix} cos\gamma & -sen\gamma & 0 \\ sen\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.7}$$

recalling the convention that the positive rotation, i.e. the positive value of the angle, correspond to the rotation in counter-clockwise direction.

1.3.2 Composition of Rotation Matrices

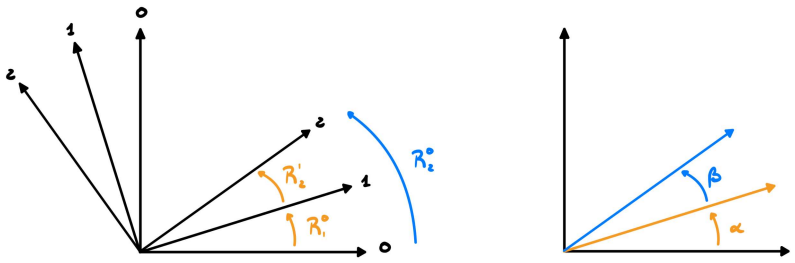
One important aspect of the rotation matrices and, in general, of the transformation matrices is that the product is not commutative, which means that the result of the composition $R_x(\alpha)R_y(\beta)$ is different from the one of $R_y(\beta)R_x(\alpha)$.

Consider the representation of a vector in different reference frames $O - x_i y_i z_i$ with $i = 0, 1, 2, \ldots$, then the matrix R_i^j denotes the rotation matrix of frame i with refers to the frame j.
Then, given the starting vector P^i, it can be defined:

$$P^j = R_i^j P^i \tag{1.8}$$

This relation can be interpreted as composition of subsequent rotations. There are two cases to analyze:

- Subsequent rotations w.r.t. **current frame**
- Subsequent rotations w.r.t. **fixed frame**



(a) *Current Frame* (b) *Fixed Frame*

Figure 1.6: Rotation of reference systems

Talking about the first case, the overall rotation is defined as the sequence of rotations where each rotation is defined with refers to the preceding one (*current frame*). Then the general formula is:

$$R_{i+k}^i = R_{i+1}^i R_{i+2}^{i+1} \ldots R_{i+k}^{i+k-1} \tag{1.9}$$

On the contrary, for the second case, the composition of successive rotations w.r.t. a fixed frame is obtained by **pre-multiplication** of the single rotation matrices in the order of the given sequence of rotation. Then, the general formula becomes:

$$R_{i+k}^i = R_{i+k}^{i+k-1} \ldots R_{i+2}^{i+1} R_{i+1}^i \tag{1.10}$$

For example, referring to the Figure 1.6b, the overall rotation for this specific case is given by:

$$R_{tot} = R_z(\beta)R_z(\alpha) \tag{1.11}$$

1.3.3 Euler's Angles

Rotation matrices are redundant description. Referring to the three-dimensional space, those matrices are composed of 9 elements, but have only 6 constraints, due to the orthogonality condition $R^T R = I_3$. This means that 3 parameters are enough to describe the orientation of a rigid body.

Then it is possible to define the **minimal representation** as the set of 3 independent parameters describing the orientation.
In general for $SO(m)$ the minimal representation requires $m(m-1)/2$ parameters.

Theorem 1.1 *Any two independent orthonormal coordinates frames can be rotated by a sequence of rotations (no more than 3) about* ***coordinate axes****, where no two successive rotations may be about the same axis.*

Euler's Rotation Matrix

Given the previous Theorem 1.1, it is possible to represent every possible rotation with a triplet of elementary rotations. The first notable case is defined as $ZY'Z''$ and is obtained as the composition of the following 3 elementary rotations:

1. Rotate the *current frame* by the angle φ about axis z: $R_z(\varphi)$
2. Rotate the *current frame* by the angle ϑ about axis y': $R_{y'}(\vartheta)$
3. Rotate the *current frame* by the angle ψ about axis z'': $R_{z''}(\psi)$

It is possible to observe that this approach consist of successive rotations w.r.t. the **current frame**, then the resulting matrix of these consecutive rotations is:

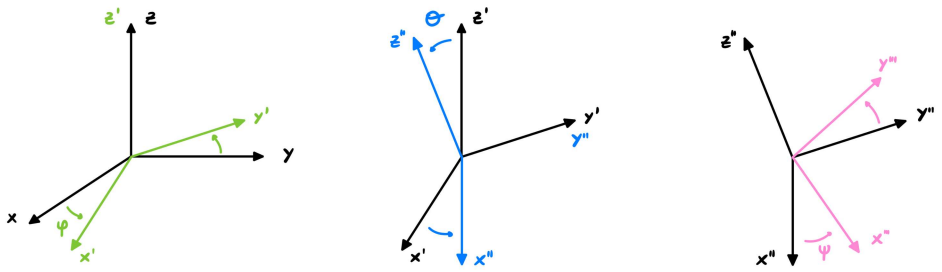


Figure 1.7: Euler's Frame Rotation $ZY'Z''$

$$R = R_z(\varphi)R_{y'}(\vartheta)R_{z''}(\psi) = \begin{bmatrix} c_\varphi c_\vartheta c_\psi - s_\varphi s_\psi & -c_\varphi c_\vartheta s_\psi - s_\varphi s_\psi & c_\varphi s_\vartheta \\ s_\varphi c_\vartheta c_\psi + c_\varphi s_\psi & -s_\varphi c_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta \\ -s_\vartheta c_\psi & s_\vartheta s_\psi & c_\vartheta \end{bmatrix} \quad (1.12)$$

where, to simplify the notation, it has been used $c_\phi = cos\phi$ and $s_\phi = sen\phi$.

Roll-Pitch-Yaw Rotation Matrix

The second important case of triplet of elementary rotations is the *Roll-Pitch-Yaw* and is defined as ZYX. This approach of composing elementary rotations is obtained as:

1. Rotate the *reference frame* by the angle ψ about axis x: $R_x(\psi)$
2. Rotate the *reference frame* by the angle ϑ about axis y: $R_y(\vartheta)$
3. Rotate the *reference frame* by the angle φ about axis z: $R_z(\varphi)$

It can be notice that in this method the rotations are all defined w.r.t. the **fixed reference frame**, then the result is:

$$R = R_z(\varphi)R_{y(\vartheta)}R_x(\psi) = \begin{bmatrix} c_\varphi c_\vartheta & c_\varphi s_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta c_\psi + s_\varphi s_\psi \\ s_\varphi c_\vartheta & s_\varphi s_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta c_\psi - c_\varphi s_\psi \\ -s_\vartheta & c_\vartheta s_\psi & c_\vartheta c_\psi \end{bmatrix} \quad (1.13)$$

1.3.4 Combining Translation and Rotation

After the discussion about the rotation matrices is possible to put together the *translation* and *rotation*, in order to obtain the complete *coordinate transformation* between two reference frames.

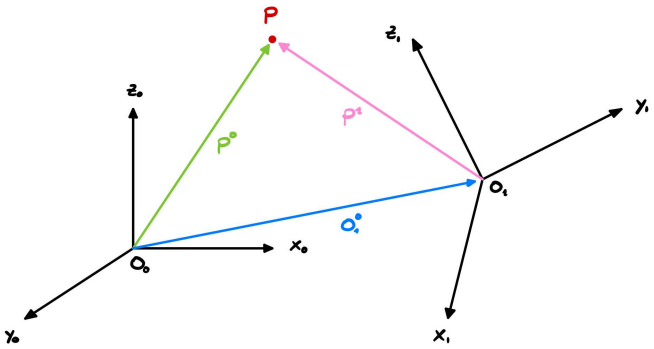


Figure 1.8: Transformation between two reference frames [2]

Referring to Figure 1.8 it must be defined the following parameters:

Symbol	Definition
P^i	Representation of P w.r.t. $O_i - x_i y_i z_i$
P^j	Representation of P w.r.t. $O_j - x_j y_j z_j$
O^i_j	Origin of frame j w.r.t. frame i
R^i_j	Rotation matrix of frame j w.r.t. frame i

Then, the formula to obtain the coordinates of the point w.r.t. the frame i is:

$$P^i = O^i_j + R^i_j P^j \tag{1.14}$$

It is possible to rewrite the formula in order to have a single matrix, instead of two different elements for translation and rotation. To do so, it is necessary to define the *Homogeneous Representation*, starting from the coordinate vector of the point P:

$$\hat{P}^i = \begin{bmatrix} P^i \\ 1 \end{bmatrix} \qquad \hat{P}^j = \begin{bmatrix} P^j \\ 1 \end{bmatrix} \tag{1.15}$$

Then it is possible to write the equation 1.14 as:

$$\hat{P}^i = A^i_j \hat{P}^j \qquad \text{with:} \quad A^i_j = \left[\begin{array}{c|c} R^i_j & O^i_j \\ \hline 0_{1\times 3} & 1 \end{array}\right] \tag{1.16}$$

where A^i_j is the *homogeneous transformation matrix*.

The homogeneous transformation matrix express the coordinate transformation between two frames in a compact way. If the frames have the same origin, it reduces to a rotation matrix.

A sequence of coordinate transformation can be write as:

$$\hat{P}^i = A^i_{i+1} A^{i+1}_{i+2} \dots A^{i+n-1}_{i+n} \hat{P}^{i+n} \tag{1.17}$$

Chapter 2

Proposed Solution

This Chapter illustrates the general solution and approach to the problem presented in the Chapter 1.
In particular it will be described how the *Total System* works and the inputs and the outputs that every part of the system has to manage.
It is going to be described also the *Camera System*, in order to present two possible type of cameras that can be used in this solution and it will be explained their working principle and how they are exploited.
Another topic that it is going to be outspread is the proposed solution to the problem of managing the memory storage of pieces. This mean that the pieces identified by the *Vision System* can't be grasped all at the same time, so it is necessary to record all the pieces that arrives.
Finally it will be shown the proposed solution for the motion of the robot, starting from the inputs that the controller receives from the *Conveyor Belt Encoder* and the *Camera System*.

2.1 Conveyor Tracking System

In Chapter 1 it has been introduced the three main components of the Conveyor Tracking System: the *Conveyor Belt*, the *Vision System* and the *Industrial Robot*. The target of the Conveyor Tracking problem is the perfect tracking of the objects that are travelling on the belt in order to make the Industrial Robot reach

and capture the items with the correct position and orientation. To accomplish this task it is necessary that all the three components works together, in order to reconstruct the real-time location of the pieces.
All the departments are independent from the others, which means that it is necessary to manage all the inputs and the outputs to make them working together, accomplishing the required tasks.
In the following figure it is shown the overall system proposed for the implementation of the problem:

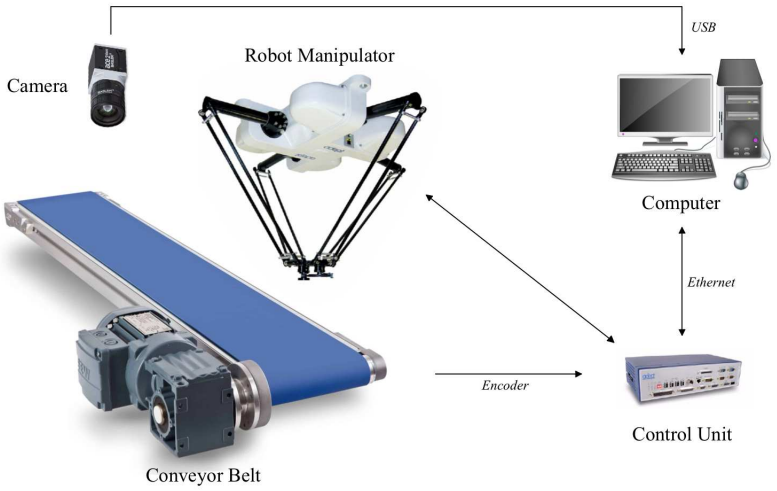


Figure 2.1: Total System Scheme

In the first step the items are loaded on the Conveyor Belt, which is running at a determined velocity. The real-time position of the belt is always known thanks to the exploitation of an *encoder*, which sends the signal to the Industrial Robot Controller. This information is used in two ways:

- to continuously sense the actual position of the belt
- to record the belt position when the picture is acquired

In order to use the information of the position of the belt derived from the *encoder*, it is necessary to define also the reference system of the Conveyor Belt, to know in which direction the objects are moving with refers to the Industrial Robot.

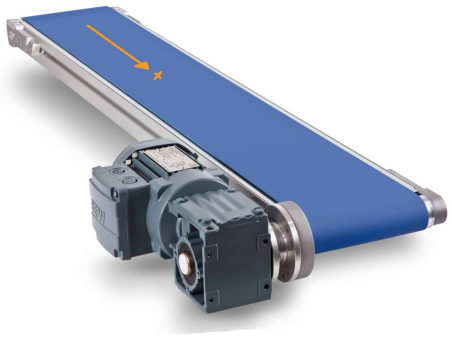


Figure 2.2: Conveyor Belt Reference System

In a specific area of the belt, there is the Camera Frame. The Vision System is mounted above the belt, such that a portion of it is included in the frame of the image.
At a certain instant of time, the Vision System acquire the image of the belt and the Computer Vision application detects if there are objects or not.
When an object is detected, the Computer Vision program sends to the Robot Controller the coordinates of the object, referring to the specific moment on which the photo has been taken.

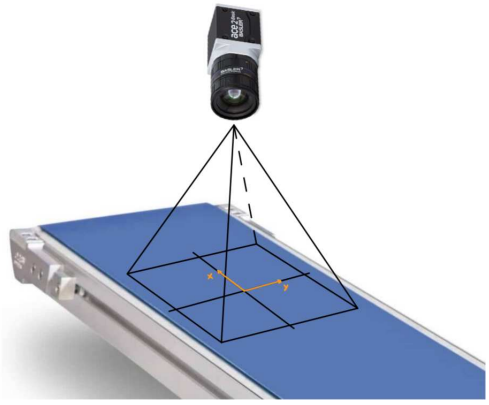


Figure 2.3: Camera Reference System on the Conveyor Belt

Then it is necessary to put together the information of the position of the Conveyor Belt and the information derived from the Computer Vision System in order to reconstruct the actual position of the item. When the image is acquired, the controller save the position of the belt and then it is able to continuously compute the amount of space that the item has travelled by making the difference

between the two positions:

$$\Delta s = s_{actual} - s_{image} \tag{2.1}$$

This means that, knowing the position of the Camera Reference Frame and the direction of the Conveyor Belt Reference Frame with refers to the Industrial Robot Reference Frame (i.e. the Global Reference Frame) it is possible to compute the amount of space travelled by the pieces and then their actual location in the Global Reference System.

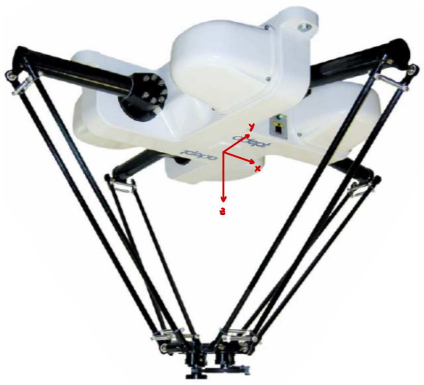


Figure 2.4: Industrial Robot Reference System

In conclusion, it is necessary to make all the three systems refers to the same reference frame in order to be able to put together all the information needed to reconstruct the location of the items at every moment of operation of the Conveyor Tracking System.

2.1.1 Design Problems

The first problem to face, as already mention in the previous section, is about the **different reference systems** of each department of the system, since each subsystem works referring to its own Cartesian axes.
It is necessary to make all the systems referring to the same coordinate axes, in order to make them working together.
To accomplish this task it has been chosen to proceed making the *Camera Frame* and the *Conveyor Belt Frame* refers to the *Industrial Robot Frame*, since the

Control Unit of Figure 2.1 receives the inputs from all the systems and sends outputs only to the robot manipulator.

The second problem to solve is about the memory of the pieces that are detected by the Vision System. This issue occurs in two main situations:

- the Vision System detects two objects in one image
- the Vision System detects a second object while the Industrial Robot has not already taken the first one

In these kind of situations the Industrial Robot controller has to keep track of all the pieces sensed by the Camera System that have not been taken by the robot manipulator.

An another possible error that can occur is that the computation of the travelled space of 2.1 is made with the information given by the *encoder*. This instrument has a finite resolution, which means that it can introduce an error on the position.
To avoid this problem it can be used an encoder with high resolution, to reduce the measuring error to the minimum.

2.2 Camera System

The first subsystem to take into account is the *Vision System*, which has a very important role in this solution for the *Conveyor Tracking* problem.
It is fundamental that the Computer Vision analysis sends the correct information to the controller of the robot, since all the calculation and the tracking of the pieces depends on these data.
In this solution it has been considered two types of camera, with two different behaviours and specification, in order to evaluate the advantages and disadvantages in both cases.

2.2.1 Matrix Camera

The first type of camera is called *Matrix Camera* and it is the most common kind of digital camera. The operation of this device is the creation of a $m \times n$ matrix, described in Chapter 1, representing the image captured in the camera frame.
The picture is acquired in a few microseconds so that it can be considered instantaneous. However, looking more in the details, when the image is taken, the matrix is composed one line of bits at time in this short period, with the result of a digital picture constituted by a matrix defined in 1.1.

The advantage in the utilization this type of camera is that the photo is quite instantly taken and the Computer Vision can directly analyze all the image. This type of cameras is also very reliable and it is possible to obtain the *trigger* perfectly synchronized with the acquire event of the device.

The main disadvantage is that this kind of cameras is not designed for the shooting of moving objects. This issue is translate by the fact that, when the belt is moving fast, the shapes of the objects may be distorted, since the camera needs a little of time to compose the all image. This occurrence can introduce some errors in the Computer Vision detection and in the results about the position and orientation of the pieces.

2.2.2 Linear Camera

The second type of camera is called *Linear Camera* from the fact that the frame of the camera is constituted by just a line of bits. This kind of camera captures only one line of bits at time, then is able to compose such lines of bits to obtain the overall image. This type of device is used especially within the cases where the camera or the items are moving along one direction, since to obtain a correct picture is necessary to scan the desired frame to make the camera gradually reconstruct the image.
In Figure 2.5 can be seen an example of the described behaviour of a linear camera. The scheme represents the area of the image that is travelling through the frame

of the camera, then the camera continuously capture lines of bits and attaches the new line to the previous ones that are saved in the camera buffer, obtaining a complete matrix image of the area that has passed through the camera frame.

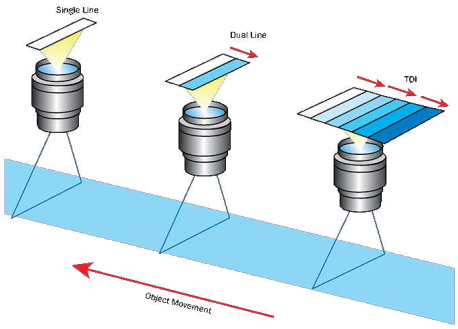


Figure 2.5: Example of Linear Camera Operation

The advantages of this kind of camera is that it is designed for the applications within there are moving objects, in particular for scanning of the Conveyor Belt. In fact, referring to this application, the camera is able to obtain images that are quite more clear with respect to the matrix cameras, since there is not the issue of the composition time.

The disadvantages are that, to obtain such precision, it is necessary to synchronize the speed of the objects with the camera rate. To do so, it is required to provide the encoder signal to the camera and to set properly the scaling factor of the encoder. This allows the camera to adapt the line rate to compose correctly the picture.

2.2.3 Computer Vision program

After the image is taken, the Computer Vision program analyzes the photo in order to detect the objects. In this solution it has been assumed an high contrast between the items and the belt, for example black objects on a white belt or vice versa, and the implementation of monochrome cameras. The only property imposed for distinguish the objects is the shape, for examples rectangular and circular items.
The information that the Computer Vision program is required to provide are:

- the position, to reach the object
- the orientation, to correctly grab the object
- the height and width, to be able to distinguish different objects

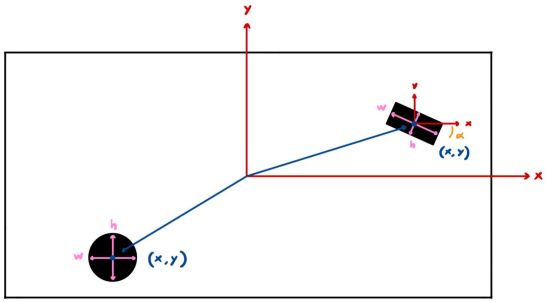


Figure 2.6: Example of Computer Vision Results

In Figure 2.6 is shown an example of how can be set the reference frame of the Computer Vision and an sample of the features that the program exploits for the application of the Conveyor Tracking.

The first important information that is needed for the realization of the solution is the position of the objects. This is the essential information for the implementation of the Conveyor Tracking problem.

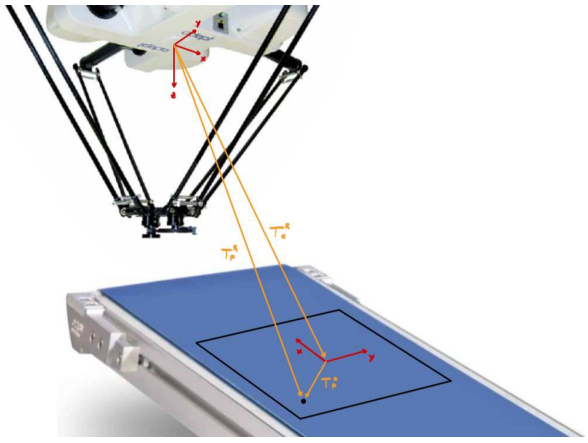


Figure 2.7: Transformation Matrix between Robot and Camera Frames

The main problem to face is that the location of the items, given by the Computer Vision program, are completely useless if the position and the orientation

of its reference frame with refers to the global reference system are unknown. Then, it is necessary to compute the *transformation matrix* between the Industrial Robot reference system and the Computer Vision reference system:

$$T_{piece}^{robot} = T_{camera}^{robot} \cdot T_{piece}^{camera} \tag{2.2}$$

Reversing the equation it can be derived the transformation matrix of the *camera* with refers to the *robot*:

$$T_{camera}^{robot} = T_{piece}^{robot} \cdot \left[T_{piece}^{camera}\right]^{-1} = \begin{bmatrix} R_{camera}^{robot} & O_{camera}^{robot} \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{2.3}$$

The other important information are the *orientation* and the *dimensions* of the objects. The *orientation* feature is used to allow the gripper to correctly catch the items, especially if the pieces are rectangle shaped, while for circles it is not important, since the gripper has to do the same work in all the possible orientation. As for the location, this information exploit his function only if the transformation matrix computed in 2.3 is known.

Last but not least, there are the *dimensions* of the objects, which are exploited for the identification of the different type of the items. In this solution it has been assumed that the objects are of two different shapes: rectangular and circular.
In Figure 2.6 can be notice that, with a particular function of the Computer Vision program, is possible to obtain the *width* and *height* of a detected item. Clearly, the rectangle shaped objects have this two dimension quite different from each other, while the circular shaped objects have the two dimensions equal, within a certain margin of error.

2.3 List Management

One important problem to solve is about the memorization of all the pieces that have been identified by the Computer Vision program, in order to continuously maintain the control over the items, to update their position, and to make sure that the Industrial Robot is always able to reach the objects and grasp them.

To accomplish this task, it has been chosen the utilization of a *list* to save the important data relative to each item identified by the Vision System.

Position x	Position y	Angle α
x_1	y_1	α_1
x_2	y_2	α_2
$\vdots$	$\vdots$	$\vdots$
x_n	y_n	α_n

Table 2.1: Example of the List of Items Detected by the C.V. System

The Table 2.1 is an example of the schedule that is necessary to implement for the purposed solution. In the table it is needful to save the coordinates x and y of the object and its orientation α.

2.3.1 Adding the objects

Dealing with this kind of listing, the first problem to take into account is the way to add the data in the list.
The main issue that can occur in this situation is the addition of the same object two or more times, making the robot manipulator trying to reach an already considered object and wasting time during the operation of the system.
To avoid this problem is necessary to exploit some comparison features in order to check if the new sensed item is already saved in the list or not. The suggested solution is the availment of the following rule:

$$x_{new} \stackrel{?}{=} x_i + \Delta x_i \quad \text{and} \quad y_{new} \stackrel{?}{=} y_i + \Delta y_i \tag{2.4}$$

where x_{new} and y_{new} are the coordinates of the new item detected by the Computer Vision program, x_i and y_i are the coordinates of the object already in the list, Δx_i and Δy_i are the components of the space travelled by the piece, starting from the first time that it has been inserted in the list to the time in which the comparison is made.
In this solution it has been assumed that the camera frame and the conveyor belt frame have one axis along the same direction, which means that the space

travelled by the items on the belt has only the component x or y, depending on which axes are parallel.

After all these considerations, and recalling 2.1, is clear that is necessary to record in the list also the position of the belt at the moment of the object detection by the Camera System. Then the previous table can be extended as:

Position x	Position y	Angle α	Belt Position t
x_1	y_1	α_1	t_1
x_2	y_2	α_2	t_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_n	y_n	α_n	t_n

Table 2.2: Extension of the List of Items 2.1

where the value t_i is the belt encoder position at the time instant in which the photo has been taken.
Then the formula 2.1 can be computed as:

$$\Delta x_i = (t_{new} - t_i) \cdot sf \tag{2.5}$$

where t_{new} is the actual value of the belt encoder position, t_i is the encoder position saved in the list and sf is the encoder *scaling factor* to convert the encoder position to the global unit of measure.

2.3.2 Removing the objects

This table keep track of all the pieces that have been located and that are not already been grabbed by the Industrial Robot, then it is important to implement a feature to remove an object from the list if it has already been taken.
In this solution it has been decided upon the simplest possible solution, to reduce as much as possible the computation time of the list management program. When and object is grabbed by the robot, the list is shifted up of one position starting from the object that has been manipulated, overwriting the data of the removed piece.

Let's suppose that the chosen piece is the one at the position j, then the updated list become:

Position x	Position y	Angle α	Belt Position t
x_1	y_1	α_1	t_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_{j-1}	y_{j-1}	α_{j-1}	t_{j-1}
x_{j+1}	y_{j+1}	α_{j+1}	t_{j+1}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_n	y_n	α_n	t_n
x_n	y_n	α_n	t_n

Table 2.3: Example of object removed from the list

where there are now $n - 1$ elements in the list.

It is important to mention that this method may be a little venturesome since during the action of overwriting the data there could be some errors that can corrupt the information. However, a more reliable method can introduce some delays in the list management task and there also could be some occasional computational errors.
As can be notice in Table 2.3, since the list is only shifted up gradually, the last element remains and overwrites all the exiting lines, so the list is never really empty. To avoid problems it is necessary to introduce a counter to keep track of the number of *real* element that are contained in the list, to allow the controller of the robot to know if there are object to grab or not.
It is important to mention that, knowing the number of *real* element saved in the list, it is possible to overwrite the last m lines of the table, which contain multiple copies of the last line and waste the controller's memory.

2.4 Robot Movement

The last argument to take into account about the purposed solution is the *Robot Movement* feature, in particular the parameters for the choice of the item to cap-

ture and the assumptions that must be taken into account.

The Industrial Robot controller checks continuously the list of objects defined in the previous section. From the list, it obtains the location and the orientation of the object to grab with refers to the global reference frame and makes the end-effector reach the item.

There are two main cases that can be exploited in the selection of the object to be grabbed:

- following the arriving order of the items
- setting a priority parameter on the different types of items

2.4.1 No Priority

The first way to manage the list is to maintain the arriving order, which means that the items saved in the list 2.2 follows the arriving order, i.e. the sequence in which they has been identified by the Computer Vision System.

There are some cases in which this statement could fail, mainly when there are multiple objects in one photo acquired by the Camera System. Then, it has been made the assumption such that the Computer Vision program returns the objects in the correct order. This means that, knowing the orientation of the camera frame with respect to the global reference system, it is possible to impose that the Computer Vision task returns the data starting from the item closer to the Industrial Robot to the farther one.
For example, taking into account the Figure 2.6 and assuming that the direction of travelling of the belt is parallel to the x axis of the camera frame, but in the opposite direction, the Computer Vision program have to send firstly the information relative to the circular object and then the data of the rectangular item.

There can be some issues in the analysis of the image and the program may return the data in the wrong order. However, this kind of error is not catas-

trophic, since the distance between two objects in the same photo is quite small with respect to the distance between two pieces identified in different images. This means that, even the items are grabbed in the wrong order, this is not a problem for the correct behaviour of the system.

At this point, assuming that the list is always ordered correctly, the controller of the robot has only to check the actual position of the first item in the list and catch it.

2.4.2 Adding Priority

The second case requires a more complex management of the list. The first addition to is to insert in the list also the data relative to the type of object.

Position x	Position y	Angle α	Belt Position t	Type of Item
x_1	y_1	α_1	t_1	type_1
x_2	y_2	α_2	t_2	type_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
x_n	y_n	α_n	t_n	type_n

Table 2.4: Extension of the List of Items 2.2

Then the controller has to check not only the position of the first element, but the position of all the elements in the list and grab firstly one item or another, with refers to the objects priority.
In this solution is assumed to have two types of items, and the new assumption is that rectangular pieces have higher priority with respect to circular pieces.

Recalling the Table 2.4, it is necessary to refine the controls, in order to correctly manage all the pieces. The first addition is to use also the *dimensions* data of the detected objects in the *addition task*. Using this information the program is able to distinguish the two type of objects detected by the Computer Vision software, as already discussed in the previous section, and to insert the parameter to differentiate the elements of the list.

At this point the controller checks the list, starting from the first element, to find the first object with higher priority, and grabs it if is inside the robot workspace. This mean that, if there are multiple objects inside the reachable space of the robot, it will catch firstly the ones with rectangular shape in the arriving order and then the circular items in the same order.

2.4.3 Mobile Reference System

When the object has been selected it is necessary to give the Industrial Robot the right coordinates in the global reference system to make the end-effector reach the item correctly.
As explained in the previous sections, by computing the transformation matrix 2.2 it is possible to know the position of the camera reference frame into the global reference system, so it is possible to set a reference system at the center of the camera reference frame at the moment when the image is taken by the camera. Then, knowing in real-time the amount of space travelled by the belt with the formula 2.5, it is possible to make the reference system *move* accordingly to the belt movement.

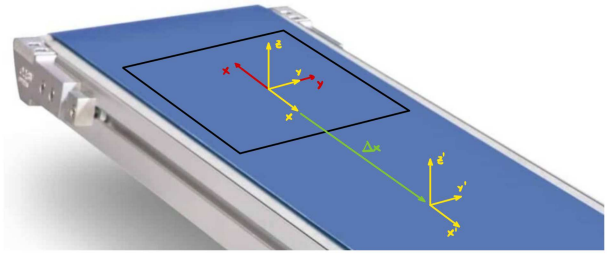


Figure 2.8: Mobile Reference Frame

This technique of *Mobile Reference System* is used to do not continuously update the information in the list of object and to avoid possible errors that can occur trying to modify in real-time the elements of Table 2.2.
It is important to notice that this method can only be implement if in the list is saved the *belt position* when the photo is taken, however it is not possible to

reconstruct the new position of the reference system.

When it is necessary to make the robot grab the object, it is computed the mobile reference system with refers to the belt position of the selected item saved in the list. At this point it is necessary to recall the information of position and orientation, computed by the Computer Vision System, since the method of the mobile reference system simulates the image frame in the actual position of the mobile frame.

Chapter 3

Experimental Setup

This Chapter is going to discuss the adaptation of the purposed solution, explained in the previous sections, to the specific case under consideration. In the preceding sections it has been exposed a possible general solution with some possible general problems and issues. Now it is going to be analyzed the system under consideration, with all the problems encountered during the implementation of the solution.
The discussion start from the description of the *Adept ACE* work environment, which is the main program utilized for the programming of the robot manipulator. Then it going to be presented the Industrial Robot itself with all the specifics and the limitations.
Furthermore are illustrated all the specifics of the Conveyor Belt and of its encoder, in order to show the relation with the global reference system. Finally are shown the specifications of the robot end-effector to explain the constraints related to the picking task and to the computation of the specific transformation matrices of the overall system.

3.1 Adept ACE 3.6

The first component to analyze is the main environment used for the coding of the most of the program. The *Adept Automatic Control Environment (ACE)* is a widespread software for the control of industrial robot manipulator. The *ACE*

program is a software package that contains a set of tools to control, configure, program and monitor *Adept* equipment [3] and it is based on the *Adept V+* programming code.

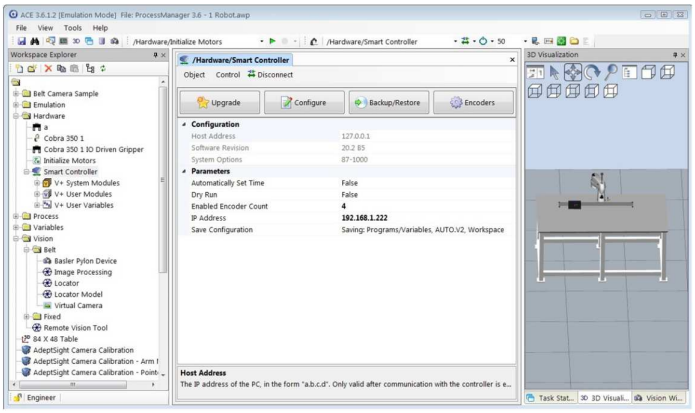


Figure 3.1: Adept ACE 3.6 Workspace [3]

In figure 3.1 it is shown the principal interface of the software. On the left band it can be notice the *Workspace Explorer*, in which is possible to navigate through the program files, the variables, and all the components that composes the system.
In the center there is the general editor, in which is possible to write the code or to change the parameters of a particular component connected to the system.
Finally, on the right side, there is the *3D Virtual Display*, which can be used to make 3D simulations of a specific system that may not be accessible in the reality.

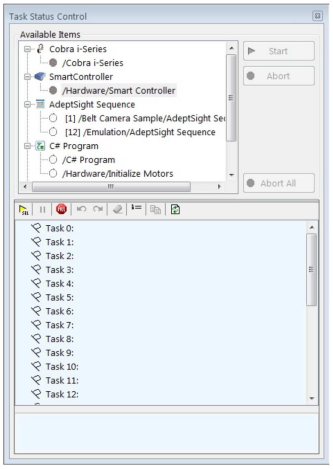
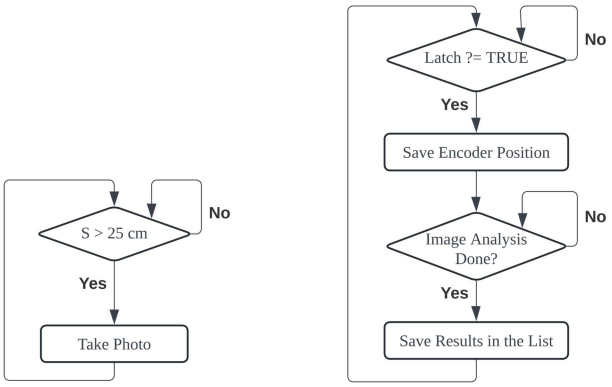


Figure 3.2: Task Manager in *ACE* Environment [3]

The main advantage of the *ACE* software is the possibility of running more than one program at the same time. This feature is crucial for the implementation of the solution presented in the previous Chapters.
To manage all the function needed for the accomplish of the Conveyor Tracking problem illustrated before, it has been divided the code in three main components:

- Photo Management program
- New Pieces Management program
- List Checking program



(a) (b)

Figure 3.3: Flow Charts of Photo Tacking and List Insert Segmentation

The first component is also the less complex one in term of concept and behaviour. As can be notice in Figure 3.3a, this code checks the amount of space travelled by the belt, in order to have a fixed distance between one photo and the next one.
In particular, this code is crucial with the implementation of matrix cameras. This is due to the fact that one object may be framed partly in the picture, causing errors in the recognition of the type of object and in the calculation of location and orientation. Then this code allows to have an overlap between two consecutive photos, in order to be sure to completely include the object in at least one photo.

The second code is a little more complex in term of syntax, but the behaviour remains quite simple. The aim of this component is the management of the new data that arrives to the controller when a new image is taken.
The feature is programmed in such way that it firstly waits for the rising of the *latch* signal, which is activated though the wire connection of the camera to the controller of the Industrial Robot. When a photo is acquired, an electric signal activate an input of the controller, activating the *latch* function, which has been previously set in the software.

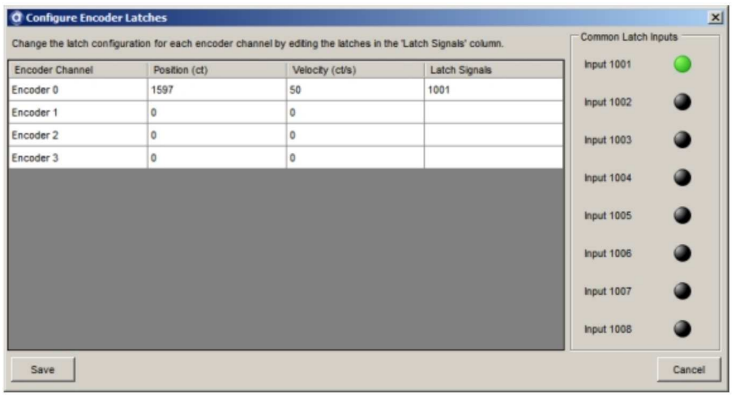


Figure 3.4: Latch Configuration in *ACE* [3]

Thanks to the activation of the electrical signal of the camera trigger, the position of the belt is saved at the exact moment when the photo is taken and it is inserted in the list with all the data of the objects detected in the current photo.
At this point, it is necessary to wait for the computation of the results from the Computer Vision program, in order to avoid the mian program to apply for results that are not already calculated. Then, when the C.V. system sends the acknowledge, the information about the detected pieces are recorded in the list.

The third part of the program is slightly more complex, because it has been developed as the main code that manages the overall system.
In Figure 3.5 can be noted that this part of the code has a more intricate behaviour, because it has to made a sequence of inspections to choose the piece and to verify that the selected item can actually be grabbed by the robot.

It is important to mention that the flowchart of the Figure 3.5 and the following explanation are focused on the case in which it is considered a priority on the type of pieces. It is important to mention that the code works anyway, if there is not such priority, by setting the corresponding variable to the same value for all the element in the list.

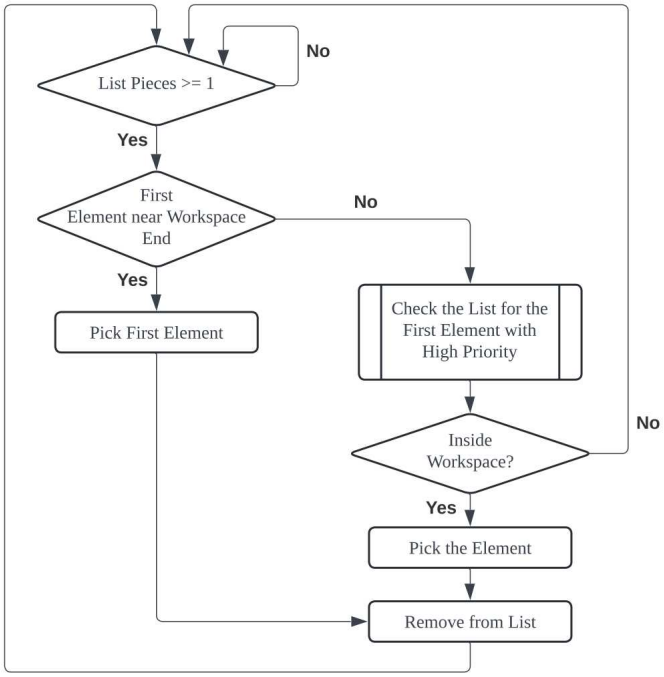


Figure 3.5: Flow Chart for the Main Program

The primary verification to do is the number of elements in the list. As already discussed in the previous sections, it is predisposed a variable to record the number of items in order to avoid miscounting.

When the number of objects is grater than 0, it is necessary to select the item to be catch by the robot, then the program check the list to find the first element with the higher priority. Then, it is necessary to wait for the item to enter the workspace before send the position to the Industrial Robot, in order to avoid mechanical errors and problems.
It can be notice that, if the object selected did not already enter the workspace, the program restarts from the beginning and all the checks are done again. This feature has been introduced to have a continuous control of the list since a position

is sent to the robot, to prevent the case in which an higher priority piece is sensed while the robot is waiting for the lower priority piece to enter the workspace.
It is important to notice that it has also been introduced a loophole that checks if the first element in the list is near to the end of the reachable space of the robot, i.e. if an object is close to exit the workspace. This control has been designed to avoid the case in which a lower priority object is near to the end of the belt and an higher priority object is sensed, making the system wait for the higher priority item and ignoring the first piece.

Finally, when the position of the selected object has been sent to the Industrial Robot, the element is removed from the list and the loop start again from the beginning.

3.1.1 Basler's Camera and ACE Sight

The first camera used for the realization of the proposed solution is a matrix camera. The chosen camera is a *Basler acA2500-14um*, which is completely configurable with the *Adept ACE* software.

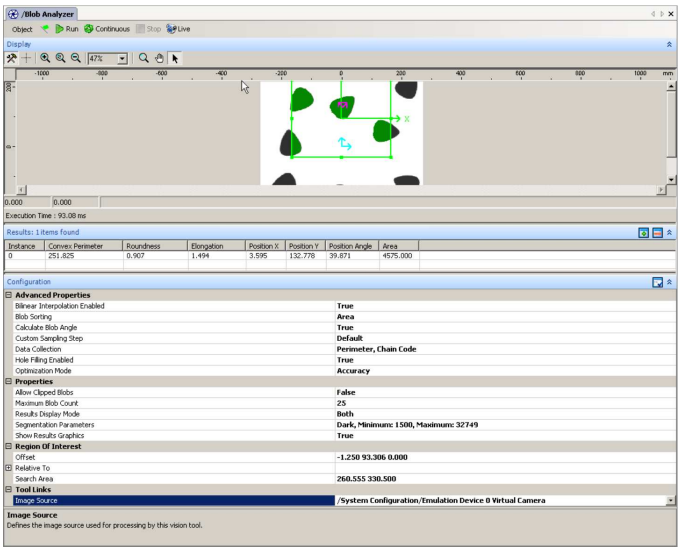


Figure 3.6: ACE Sight Blob Analyzer Workspace [3]

This matching with the camera allows to use the *ACE Sight* package, which permits to have an internal *Blob Analyzing* program without the necessity to im-

plement an external code or an additional function.
With this camera and the mentioned package, it has been implemented the code of Appendix A.1, which follows the flowcharts described in the previous section.

Firstly, it is necessary to configure the image analysis, before being able to exploit the camera and the Blob Analyzer. To do this it is crucial to calibrate the camera, in order to have a correct representation of the framed picture.

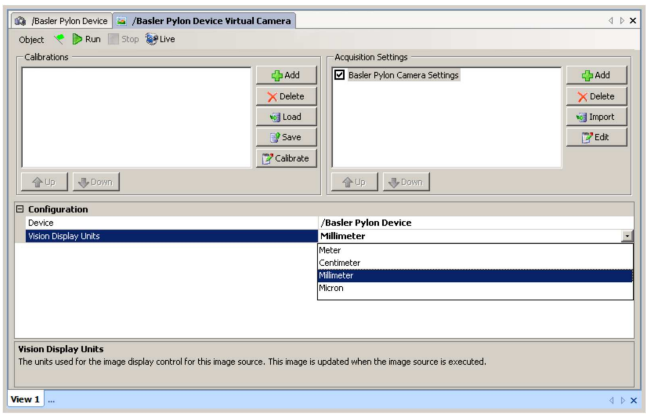


Figure 3.7: ACE Sight Camera Calibration [3]

The calibration that has been used is the *fixed pixel calibration* method, that is a manual way to declare the corresponding dimension of one pixel in the real world.

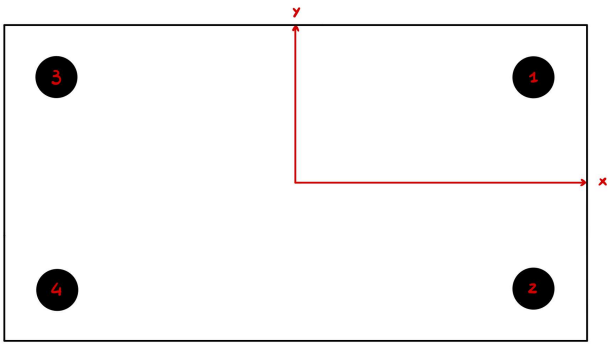


Figure 3.8: Camera Calibration Scheme

In order to calibrate the Vision System it has been positioned four pieces as shown in Figure 3.8. The measure of the y distance (between piece 1 and 2 or

between 3 and 4) has been done manually, while the measure of the x distance (between piece 1 and 3 or between 2 and 4) has been using the belt encoder.

Element	Position x [pixel]	Position y [pixel]
1	1121,031	740,807
2	1108,455	-839,170
3	-1067,260	806,048
4	-1057,873	-810,051

Table 3.1: Camera Calibration measurements

$$\Delta x_{13} = 325,25 \text{ mm} \quad \text{and} \quad \Delta x_{24} = 323,75 \text{ mm} \tag{3.1}$$

$$\Delta y_{12} = 240 \text{ mm} \quad \text{and} \quad \Delta x_{34} = 234 \text{ mm} \tag{3.2}$$

Given all the measurements listed in Table 3.1, in 3.1 and in 3.2 it is possible to compute the conversion factor between pixels and millimeters as:

$$x_{px2mm} = \frac{\Delta \text{mm}}{\Delta \text{pixel}} = 0,14863197 \div 0,14944644 \text{mm/px} \tag{3.3}$$

$$y_{px2mm} = \frac{\Delta \text{mm}}{\Delta \text{pixel}} = 0,14479311 \div 0,15190095 \text{mm/px} \tag{3.4}$$

This conversions allows to obtain the information of the Computer Vision program in the global unit of measure used by the Industrial Robot controller.

An another problem that has been faced in the implementation of the solution is that the camera reference frame it is not perfectly aligned with the conveyor belt travel axis. This error, even if the angle is very small, can cause problems to the correct localization of the detected objects and also in the controls that allows to not insert in the list the same object multiple times.

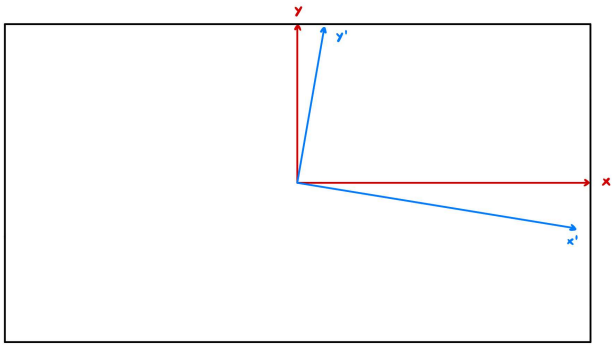


Figure 3.9: Representation of the Angle Error

In Figure 3.9 can be seen a representation of the angle error between the two reference frames. In *red* is plotted the camera reference system, while in *blue* is showed how should be the reference system of the camera if the x axis is perfectly parallel to the travel direction of the belt.
After the computation with trigonometric calculations the angle between the two reference frames results $\alpha = -1,42°$ and it is possible to correct the results of the Computer Vision program with the following formulas:

$$x = x_{CV} \cdot cos(\alpha) + y_{CV} \cdot sen(\alpha) \tag{3.5}$$

$$y = -x_{CV} \cdot sen(\alpha) + y_{CV} \cdot cos(\alpha) \tag{3.6}$$

Finally, it is necessary to impose the threshold for the *image segmentation* that the *Blob Analyzer* apply to the picture to find the blobs, i.e. to detect the objects.

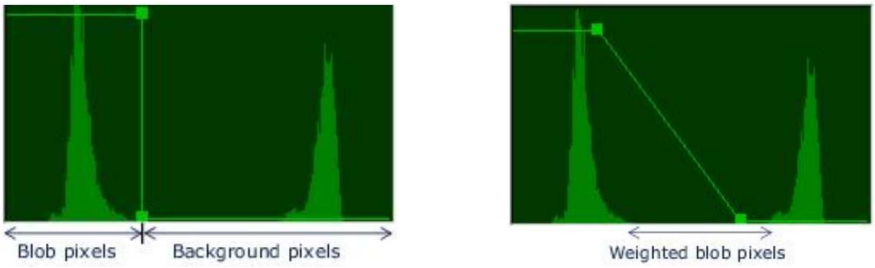


Figure 3.10: Representation of Hard *(left)* and Soft *(right)* Segmentation

As can be notice in Figure 3.10 the *ACE Sight* package offers two types of image segmentation:

- Hard Segmentation
- Soft Segmentation

Looking at the histograms, that represents the number of pixels for each value between 0 and 255, as described in Chapter 1, the main difference between this two types of segmentation is that the first one separates toughly the two sets of values, while the second type has a smoother approach and it weights the pixels that are inside the middle region to choose if assign them to the right side or left side.
For this application it has been chosen the hard approach, since there is an high contrast between pieces and belt. The correct threshold has been set by trial and error, due to its variability with refers to the light changes and disturbances.

3.1.2 Teledyne DALSA's Camera and Matlab

The second camera used to exploit the purposed solution of Chapter 2 is the *Teledyne DALSA Linea GigE*.
The main advantage of this camera, and of the linear cameras in general, is that it is designed for the application that implies moving objects. Unfortunately, the chosen camera is not configurable directly with the software of the controller, then it has been necessary to use *Matlab* to manage the camera and to perform the image to analyze.

Figure 3.11: Teledyne DALSA Linea GigE

The configuration of this kind of camera requires more steps with respect to the previous one. Firstly, it is necessary to synchronize the camera acquisition rate with the speed of the belt, to allow the correct reconstruction of the image while the objects are moving through the camera frame.
To implement this feature it is necessary to connect the belt encoder to the camera, providing the encoder signal as shown in Appendix D.

After the connection to the encoder signal, it is necessary to set the camera to acquire the picture following the rate of the belt encoder and analyze the images that the camera takes. To exploit these features it has been necessary to make use of a secondary software.

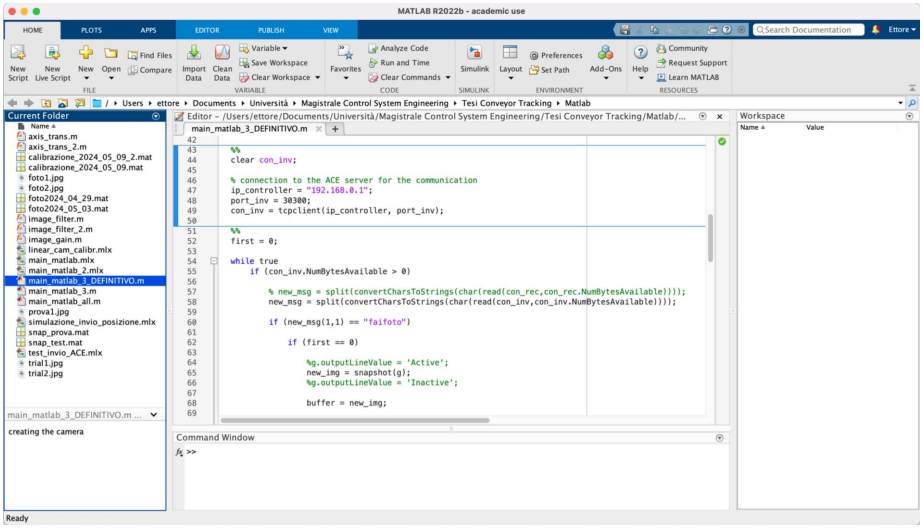


Figure 3.12: *Matlab* Software Workspace Example

The program exploited for the utilization of the linear camera is *Matlab*, for the high flexibility of the software and for the possibility of communication with the controller's software. With the *Matlab* software it has been possible to acquire the camera frames and analyze the image, in order to detect the objects and send the information to the controller's software as shown in Appendix B.

The next step is the calibration of the image frame and the computation of the conversion factor between the pixels and millimeters. To compute the calibration of the camera it has been used the following approach: firstly it has been taken

an image of an object and then it has been computed manually the conversion factor for the image information.

Figure 3.13: Image taken with the *Teledyne DALSA* camera

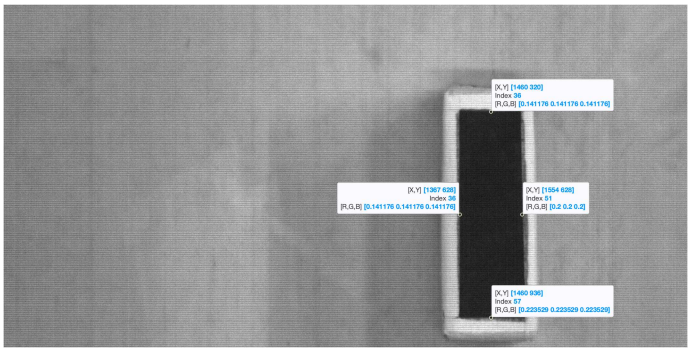


Figure 3.14: Image Elaborated with *Matlab*

With the information of Figure 3.14 it is possible to make the same computation of 3.3 obtaining:

$$x_{px2mm} = \frac{\Delta \text{mm}}{\Delta \text{pixel}} = 0.1044 \text{ mm/px} \tag{3.7}$$

$$y_{px2mm} = \frac{\Delta \text{mm}}{\Delta \text{pixel}} = 0.125 \text{ mm/px} \tag{3.8}$$

After the acquisition of the images by the linear camera, it is necessary to analyze the picture with the *Matlab* functions. The *Computer Vision* package of the software provides some useful features that permits the blob analysis [4].

To exploit this features of the *Computer Vision Toolbox* it is fundamental to create a `vision.BlobAnalysis` object with the command:

```
Hblob = vision.BlobAnalysis(Name,Value)
```

where the parameters `Name` and `Value` allows to activate or not some features of the blob analysis function.
For the current application it has been necessary to activate the `MajorAxisLengthOutputPort` feature, the `MinorAxisLengthOutputPort` feature and the
`OrientationOutputPort` feature, while the settings to obtain the location of the blobs are active by default. An another important setting it is the `MinimumBlobArea` value, in order to do not take under consideration the blobs with too small area. This last feature allows to do not compute the parameters of blobs that are not the objects to be capture, like background imperfections or in the case that one object is half framed in the image.
Then the complete command used in the solution is:

```
blob = vision.BlobAnalysis('BoundingBoxOutputPort',false,'MinimumBlobArea',
    area_trh,'OrientationOutputPort',true,'MajorAxisLengthOutputPort',true,'
    MinorAxisLengthOutputPort',true);
```

It is important to mention that, to use this blob analysis function, is necessary to elaborate the image to make it compatible with the feature.

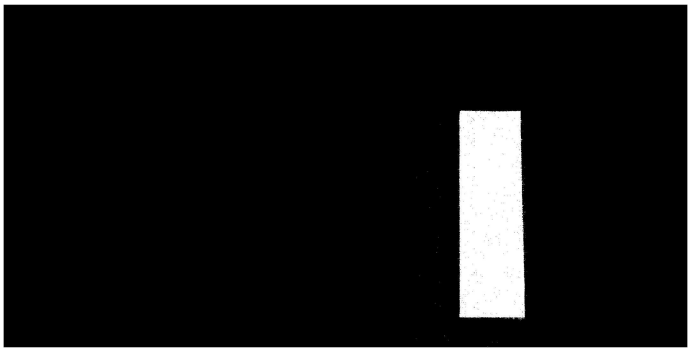


Figure 3.15: Image Elaborated for the Blob Analysis

The image in Figure 3.15 is an elaboration of the picture of Figure 3.14. Firstly

the image has been scanned to make zero all the components of the image matrix that are over the chosen threshold for the black level. Then, it has been applied the *Matlab* function `logical` to make the image matrix composed only by 0 for the background, and 1 for the blob.
With the picture translated in the `logical` format it is possible to apply the *Computer Vision* function to compute the desired parameters of the blobs.

After the blob identification by the *Matlab* code, it is crucial to create the communication to send the data between the *Matlab* environment and the *Adept ACE* software.
The first step is the creation of the communication between the two programs. It has been opted for a local communication based on the *TCP/IP* protocol, in which the *ACE* software works like the *master* of the communication and the *Matlab* software works like the *slave* component of the network.
Firstly it has been necessary to create the line with the proper *Adept ACE* function:

```
FOPEN (lun, 16) "/LOCAL_PORT number"
```

where the parameter `number` select the port in which the communication is established [5]. In the Appendix A.2 is reported the complete code used for the creation of the connection between the softwares.
After the execution of this function, it is possible to connect the *Matlab* program to the network with the following function:

```
con = tcpclient(ip, port);
```

where the parameter `ip` is used to select the IP of the master of the network and the variable `port` is used to chose the port of the communication channel [6].

3.2 Robot, Belt and End-Effector

This section is going to presents the mechanical part of the system, which are the Industrial Robot manipulator, the Conveyor Belt with its Encoder and the

End-Effector of the robot.
The attention is focused on the application of the general solution to the system and the *real* issues to face.

3.2.1 Adept Quattro s650H

The main mechanical component of the system is the Industrial Robot, which it is the element with the most number of constraints and limitations that have to be taken into account during the implementation of the purposed solution.
For the implementation of the solution it has been chosen the *Adept Quattro s650H* Industrial Robot.

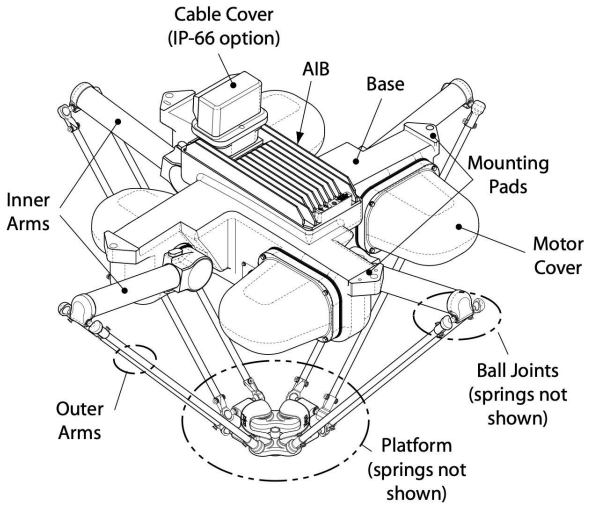


Figure 3.16: Adept Quattro s650H Scheme [7]

In Figure 3.16 it is reported a general scheme of the robot manipulator exploited for the solution. It is composed by four motors and each of them drives one of the four arm, which are connected to the below platform and allow it to move in three dimensions in the space.

Each motor of the Industrial Robot directly drives the first part of the arm shown in Figure 3.17 (*left*), making the inner arm rotate upward or downward. The three-dimensional movement and the rotation that the platform can achieve are the results of the composition of the actions of all the four inner arms.

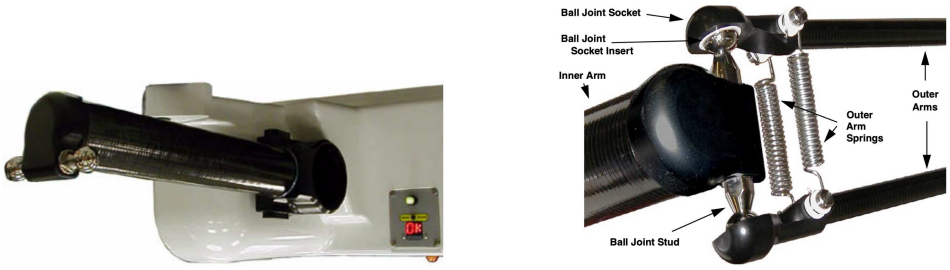


Figure 3.17: Adept Quattro s650H Arm Components

The industrial robot is directly connected to the controller which allows the user to control the motion of the robot by programming the code in *Adept ACE* software.

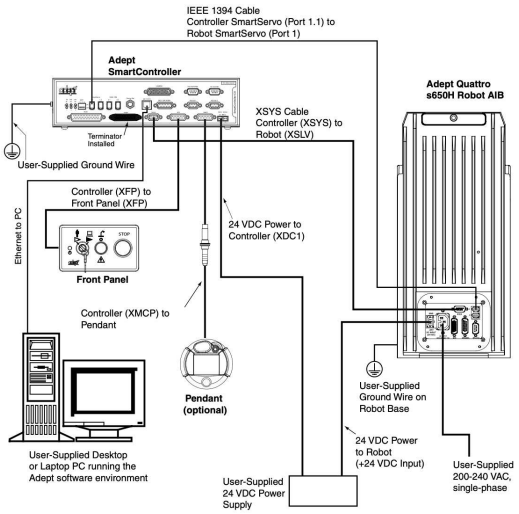


Figure 3.18: Adept Quattro s650H Connection Scheme [7]

In the system it is also present a *Front Panel*, which is necessary to enable the motion of the robot. This is a safety measure to have a double check when the user wants to provide power to the robot. In the system are also installed other safety measures, in fact the robot is boxed into a safety cage to avoid that external objects or people enters the workspace of the robot while it is moving. The cage is endowed with sensors that send a signal when the door opens, in order to stop instantly the robot motion to avoid breakages or injuries.

In Figure 3.19 are displayed the dimension of the industrial robot and the di-

mension an location of the reachable workspace of the robot, which is particularly important for the design and the implementation of the solution to the Conveyor Tracking problem.

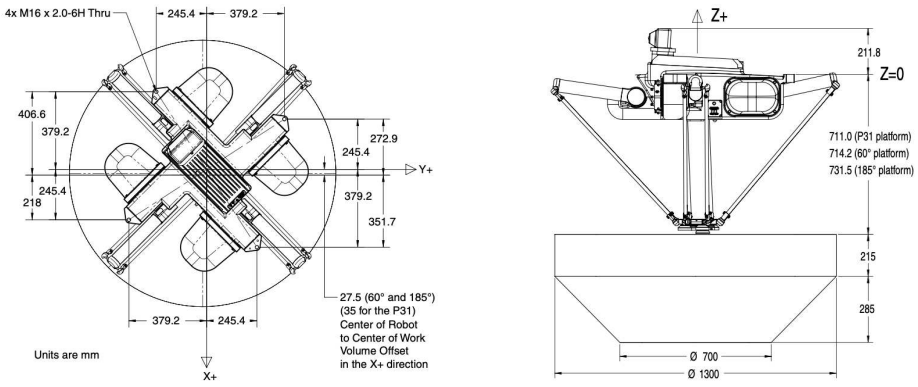


Figure 3.19: Adept Quattro s650H Dimensions and Workspace

3.2.2 Belt and Encoder

An another important component of the system is the Conveyor Belt, which is the means on which the pieces are transported.

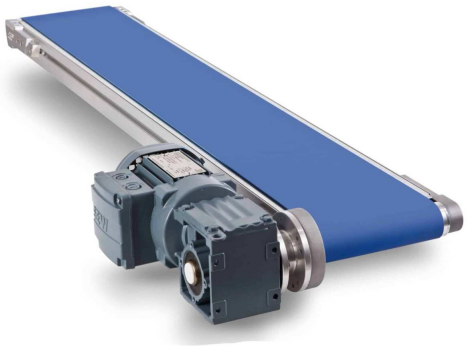


Figure 3.20: Conveyor Belt Example

As shown in Figure 3.20, the belt is powered by a motor which makes the belt surface slide in only one orientation with two possible directions. In the purposed solution it is assumed that the belt travel only forward, without going back.
To operate the belt it is used an industrial control panel where is possible to set

the direction of travelling and the speed.

The actual position of the belt is sensed through the use of an *incremental encoder* connected to the Industrial Robot controller. The encoder itself is only able to sense the variation of the belt position and it sends the signal to the controller, which keeps in memory the current position and continuously update the new position, according to the signals received by the encoder.
Recalling that the encoder position (and speed) has to be converted with the scaling factor defined in the previous Chapters, it is useful for the following sections to present a complete table that contains all the value of the belt speed:

Indicator Speed	Encoder Speed $[tick/s]$	Belt Speed $[mm/s]$
1	170	42,5
2	310	77,5
3	440	110
4	570	142,5
5	700	175
6	840	210
7	1000	250
8 (max)	1260	315

Table 3.2: Belt Speed Table

3.2.3 End-Effector

The last component is the End-Effector of the Industrial Robot. In this application it has been implied a *Pneumatic Gripper* for the grabbing task.
This kind of End-Effector is widely used in the industrial application for the simplicity of its behaviour and for its reliability against the malfunctioning. This characteristics are due to the fact that the gripper is drive with the use of compressed air, in fact the tongs are normally open and to make them close it is sufficient to inject the air into the body of the gripper.

The main issue about this kind of gripper is when pressure losses occurs. There are many causes that can produce this type of problem, mainly if the

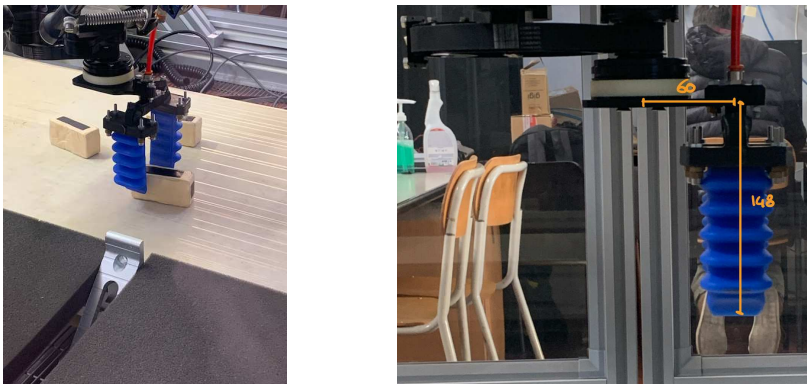


Figure 3.21: Industrial Pneumatic Gripper

gripper bumps into some obstacles causing the break of the tubes or the break of the surface of the gripper.
In Figure 3.21 is shown the gripper of the experimental setup. It is important to notice that the gripper adds an offset in two direction of the global reference system, that must be taken into account to obtain the position of the point where the item are grasped, starting from the point where the gripper is attached to the last joint of the robot.

3.3 Specific Transformation Matrices

After the explanation of all the system, it is necessary to clarify how all the parts are related. The most important connection is carried out with the *Transformation Matrices* between all the reference systems, that allows the uniform data representation across all the departments of the system.

Firstly it is necessary to present all the subsystems, in order to understand where the components are located and how it is defined each reference frame.
In Figure 3.22 is shown the complete setup from two different points of view. It can be noticed the position of the two different cameras on the top of the pictures and it is highlighted the end-effector of the robot. On the left image it is possible to observe the position of the Basler's Camera frame on the conveyor belt, while on the right one is displayed the Linear Camera frame.

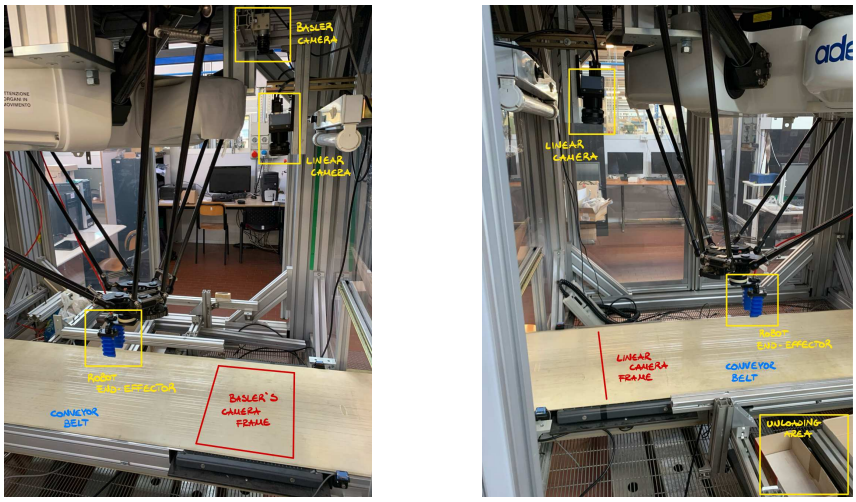


Figure 3.22: Global Overview of the Experimental Setup

The first component to analyze is the global reference system, which has been set as the reference frame of the Industrial Robot for the implementation of this solution. The main advantage of this choice is that the controller of the robot, i.e. the *Adept ACE* software, is the main component and it manages all the operation of the system, while the Camera System and the Conveyor Belt are used as secondary instruments to provide information.

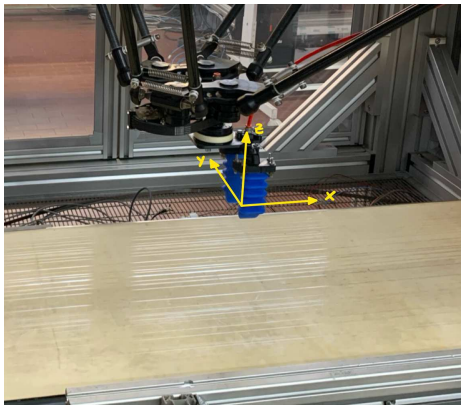


Figure 3.23: Industrial Robot Reference Frame at End-Effector

As shown in Figure 3.19, the center of the global reference system is set at the top of the main body of the robot. Then, as shown in Figure 3.23, it is possible to translate the reference system to every point inside the workspace of the robot, for example at the end-effector, since the controller always know the position of

the *platform*. In particular, for the end-effector used in this solution, it is necessary to apply the translations shown in Figure 3.21 to obtain the position of the end-effector with refers to the global reference system.

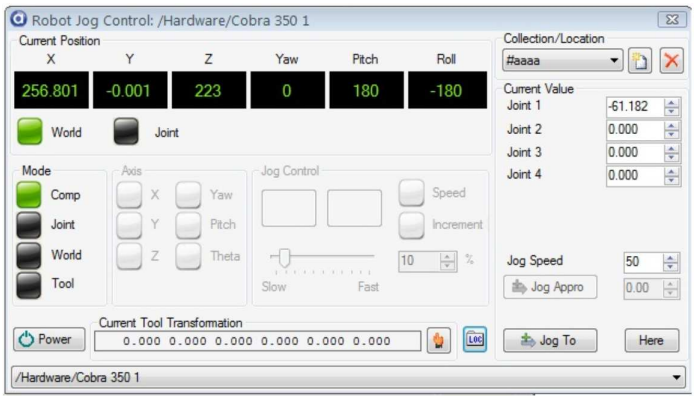


Figure 3.24: Adept ACE Jog Control [3]

In Figure 3.24 is shown the window of the *Jog Control* feature of the *Adept ACE software*, which allows to manually control the robot movements and returns the actual position of the platform or, if correctly set in the *Current Tool Transformation* bar, the actual position of the end-effector.

The next important element to explore is the reference frame of the Camera System, which is crucial to provide the correct location of the objects when detected by the Computer Vision System.

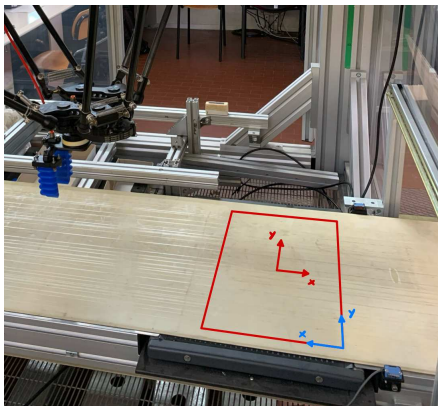


Figure 3.25: Camera System Reference Frame

In Figure 3.25 it shown the area of the frame of the Camera System with the two reference systems used for the Basler's Camera (*red*) and for the Linear Camera (*blue*).
It is important to notice that the two reference systems has the same orientation for the y axis while the opposite orientation for the x axis.

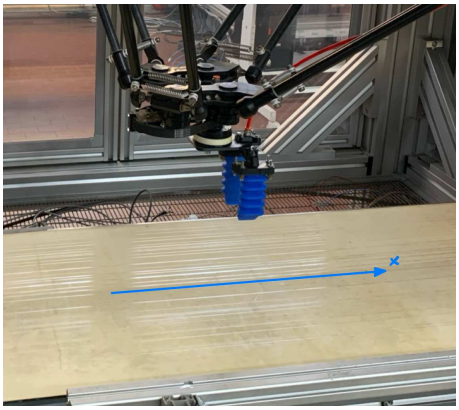


Figure 3.26: Conveyor Belt Reference Frame with Scaling Factor

The last reference frame to observe is the one belonging to the Conveyor Belt, which is also the one that provides less information.
In Figure 3.26 is displayed the reference system of the Conveyor Belt which is composed only by one axis, since the belt can only travel in one direction. It has be imposed the x axis since it is aligned with the x axes of the Industrial Robot and of the Camera System.
In this application the Conveyor Belt reference system is used only to measure the variation Δx of the belt position along its axis, adding this value to the location of the objects computed by the Computer Vision system.

At this point it is possible to compute the transformation matrices needed for the solution of the problem. Firstly it will be explained the computation for the Basler's Camera transformation matrix and then it will also shown the calculation of the transformation matrix for the Linear Camera.
Before starting, it is important to mention that, in the *Adept ACE* code, the transformation matrices are not used directly, and the computation is mainly done to derive the actual position of the Camera System reference frame with

refers to the global reference system. After knowing this position, the information of the Computer Vision program (position and orientation of the items) are used by means of specific *Adept eV+* commands in the code.

3.3.1 Basler's Transformation Matrix

The first step consist in placing an object, preferably a rectangular one, to compute also its orientation with the Computer Vision program, on a generic point of the Conveyor Belt reachable by the End-Effector of the Industrial Robot.
With the *jog control* shown in Figure 3.24, the End-Effector is moved to the *pick position* of the object, i.e. it is positioned in such way that, closing the gripper, it correctly grab the item. At this point the *jog control* returns the position of the object and its orientation with refers to the Industrial Robot reference system.

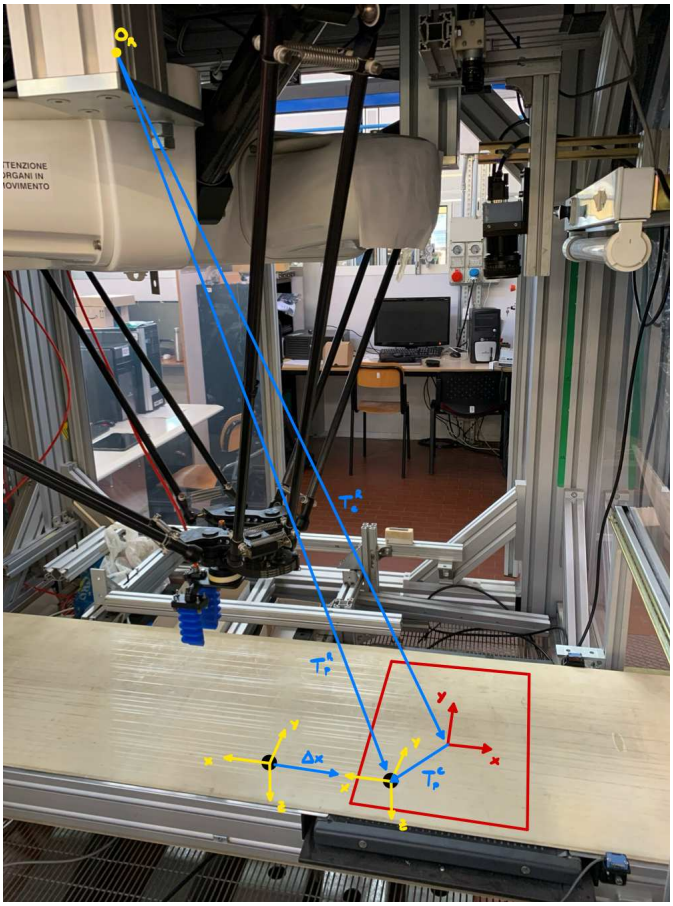


Figure 3.27: Real Transformation Matrix Calculation Scheme

Subsequently it is activated the Conveyor Belt backward, to make the object enter the image frame of the camera, in order to be able to compute its location and orientation with the Computer Vision program in a generic point of the picture frame.
Looking at Figure 3.27 it can be notice that, with the procedure discussed above, it is known the position of the object with refers to the global reference system, also after the translation, by adding the Δx component relative to the Conveyor Belt movement.
In the previous picture can also be notice that the reference frame of the object is rotated with refers to the global reference system. This is due to the settings of the Industrial Robot, which defines in such way the reference system at the End-Effector.
The information obtained with the *jog control* feature are given with refers to this reference frame, then the transformation matrix of the object with refers to the robot reference system can be written as:

$$T_P^R = \left[\begin{array}{ccc|c} 1 & 0 & 0 & x_R + \Delta x \\ 0 & -1 & 0 & y_R \\ 0 & 0 & -1 & -1050 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.9}$$

where the variables x_R and y_R are the coordinates of the object in the global reference system.
The top-left 3×3 matrix is computed with the *Euler's Rotation Matrix* formula 1.12:

$$R = R_z(0)R_{y'}(\pi)R_{z''}(\pi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

With an analogue reasoning it is possible to derive the transformation matrix of the object with refers to the camera frame. It is important to recall that, looking at the orientation of the x and y axes of the camera frame and using the *right hand rule* for the construction of the reference systems, the z axis of the

camera frame is orientated in the opposite direction of the one belonging to the item.
Then the transformation matrix results:

$$T_P^C = \left[\begin{array}{ccc|c} -1 & 0 & 0 & x_C + \Delta x \\ 0 & 1 & 0 & y_C \\ 0 & 0 & -1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.10}$$

where the variables x_C and y_C are the coordinates of the object in the camera reference system.
The top-left 3×3 matrix can be seen as a simple rotation matrix with refers to the y axis:

$$R = R_y(\pi) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

Finally, looking at the the global scheme of Figure 3.27, it is possible to derive the formula for the computation of the unknown matrix T_C^R. The formula is the same defined also in 2.2:

$$T_P^R = T_C^R \cdot T_P^C \tag{3.11}$$

It is possible to rewrite the formula as:

$$T_C^R = T_P^R \cdot T_C^P \tag{3.12}$$

where, recalling the properties of the transformation matrices, it is possible to write $T_C^P = (T_P^C)^{-1}$. Then, the final formula is:

$$T_C^R = T_P^R \cdot (T_P^C)^{-1} = \left[\begin{array}{ccc|c} -1 & 0 & 0 & -541.687 \\ 0 & -1 & 0 & -27.761 \\ 0 & 0 & 1 & -1050 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \tag{3.13}$$

where the top-right column vector is the position of the center of the camera frame with refers to the the global reference system, as shown in 2.3.
This position is used to define the location of the camera reference frame in the Adept ACE code with the command:

```
SET loccamera = TRANS(-541.687,-27.761,-1055,0,180,180)
```

where the first three parameters are the x, y and z coordinates of the center of the reference system, and the second three parameters defines three rotations that follows the Euler's Angle Formula [8].

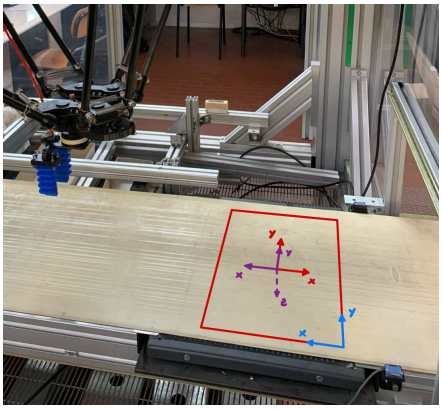


Figure 3.28: Basler's Camera Location Definition

In Figure 3.28 is shown the reference frame defined by the above command (*purple*). It can be notice that the orientation of the reference frame is not the same as the reference system of the camera frame. This difference in the orientation explains the negative and positive signs that needs to be set to the information returned by the Computer Vision program to correctly reach the target.

3.3.2 Teledyne DALSA's Transformation Matrix

For the second camera model it cannot be applied the same approach used for the matrix camera, due to the different operating mode. Recalling the Figure 3.22 and the discussion of the linear cameras behaviour in Chapters 2, it has been chosen a *manual* approach for the localization of the camera frame along the conveyor belt.

Using a *Matlab* function it has been possible to obtain a continuous frame capture of the line of bits of the linear camera frame. In this way it is possible to manually see when a dark object is under the frame of the camera and, moving by hand the item, it is possible to place it near to the camera frame, making the edge of the object almost coincide with the edge of the camera frame.
This method allows the computation of the position of the camera frame, instead of the location of the center of the image as done for the Basler's Camera.

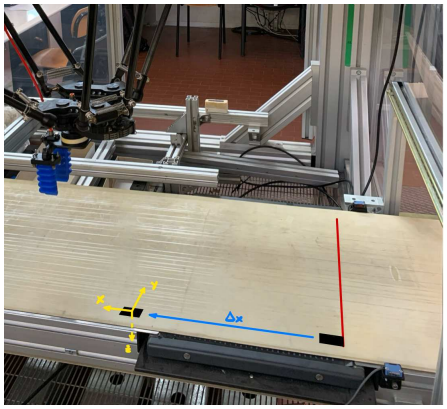


Figure 3.29: Linear Camera Location Calculation

In Figure 3.29 is shown the location of the object next to the camera frame. At this point the piece is moved with the aim of the Conveyor Belt and, thanks to the encoder, is possible to measure the amount of space Δx between the two locations of the item.
Then, using the *jog control* as done in the approach used for the matrix camera, is possible to move the robot end-effector over the item in the grabbing position, in order to obtain the location of the item after the movement.

Starting from the position obtained with the *jog control* and retracing the previous steps in reverse, it is possible to derive the location of the edge of the camera shown in Figure 3.30.
Knowing the position of the edge is possible to define the location of the camera with the command:

```
1 SET loccamera_2 = TRANS(-47.293-580, -0.433, -1055, 0, 180, 180)
```

The same definition of the camera frame is used in the *Matlab* program for the computation of the Computer Vision results. To do this it is necessary to

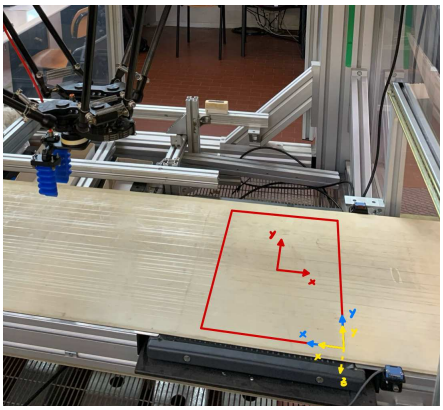


Figure 3.30: Linear Camera Location Definition

implement a change of coordinates in the *Matlab* picture, since the highlighted edge of the image frame is different from the edge of the picture visualized in *Matlab*.
Looking at 3.14, the edge of the previous image is the bottom-left, while in the *Matlab* software the coordinate frame is centered in the top-left edge.

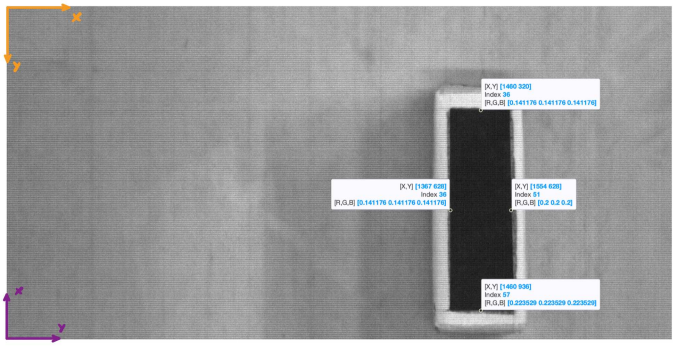


Figure 3.31: Image Frame Definition in *Matlab*

In Figure 3.31 can be notice that the two reference frames are orientated in different ways. The top-left frame (*orange*) is the default one of the *Matlab* software, while the bottom-left frame (*purple*) is the one defined for the Conveyor Tracking application.
Then, the Computer Vision program implemented in *Matlab* applies a change of coordinates and derives the information of location and orientation with refers to the bottom-left frame.

3.3.3 Specific Implementation of Conveyor Tracking

At this point the available information are: the location of the camera and the list of the objects with all the necessary information saved inside. Then, it is possible to exploit the actual Conveyor Tracking function, to make the system track and grab correctly the items that are travelling on the Conveyor Belt.
To accomplish this task is used the technique of the *Mobile Reference System* explained in the Chapter 2. To implement this method it is necessary to define a Conveyor Belt object into the *Adept ACE* software, in order to give the controller the necessary information to continuously reconstruct the actual position of the mobile reference frame.

The command used to define the belt is:

```
1 DEFBELT %locbelt = loccamera, 1, 1, sf
```

where the first parameter is the starting position of the belt, i.e. the starting position of the mobile reference frame, that coincides with the camera location. It is important to recall that the x axis of the frame in the first parameter defines also the direction of moving. The second parameter indicates the encoder that reads the location of the belt and the last parameter `sf` defines the encoder scaling factor, to have the measure in millimeters. The third parameter is not used but is necessary to set a value [5].

Then it is important to give the dimensions of the belt object, since the robot controller allows the Conveyor Tracking of the mobile reference system only inside the area of the belt object.
The code to define these parameters is:

```
1 WINDOW %locbelt = TRANS(600,500,0), TRANS(-250,-500,0)
```

where the two `TRANS` functions defines the limits of the Conveyor Tracking belt object. The window boundaries are planes that are perpendicular to the direction of belt travel and include the positions specified by the two transformations [5]. The values are in millimeters (the global measure reference) and refers to the center of the Industrial Robot reference system of Figure 3.19.

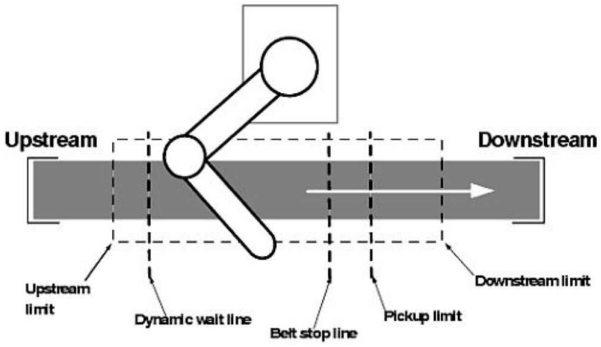


Figure 3.32: Conveyor Tracking Window Limits

It is important to mention that the values of the limits for the belt window has been chosen to avoid the robot exit the reachable workspace (first `TRANS` function) and to avoid the robot enters the Camera Frame area covering the belt during the image acquisition (second `TRANS` function).

When the belt is defined and the object is selected, it is necessary to make the robot track the piece and catch the object. To complete this task is crucial to firstly set the initial position of the belt, i.e. the position of the belt at the time of the photo acquisition, with the following command:

```
SETBELT %locbelt = pos_start
```

where the parameter `pos_start` is the trigger value of the selected piece, saved in the last column of items list.

Knowing the initial trigger value and sensing the actual position of the belt with the encoder signal, the controller of the industrial robot is able to compute the amount of space travelled by the belt since the photo has been taken. Then, it is possible to make the robot reach the mobile reference frame with the command:

```
APPRO %locbelt:TRANS(-x, y, 0, -alpha, 0, 0), dz_pick
```

where the `TRANS` function is used to add the information of the item computed by the Computer Vision program. In fact, this method allows to add a transformation matrix to make the robot translate and rotate starting from the mobile

reference frame defined with the previous code. Looking at Figure 3.28 the function add the translation and rotation with refers to the *purple* reference system, then it is necessary to add the negative sins to the `x` and `alpha` parameters due to the orientation of the reference frame with refers to the reference frame of the Camera System.

Finally, the `dz_pick` parameter is used to add a displacement in the approaching movement, to avoid the collision with the object before the actual grab task of the end-effector.

Chapter 4

Experimental Results

This Chapter is going to discuss all the experimental results and the problems highlighted during the tests of the two experimental setups.
The first system analyzed is the one with the Camera System exploited with the Basler's Matrix Camera, which is not the best possible instrument choice for the implementation of Conveyor Tracking problem.
Then it will be discussed the experimental setup exploited with the Teledyne DALSA Linear Camera, which is the best type of camera that can be used for the implementation of the Conveyor Tracking problem. However, the solution with the aid of the *Matlab* environment is not the most efficient.
In the last section will be presented a possible extension to the solution and the system, in order to improve the efficiency and avoid some problems that occurred in the behaviour of the experimental setup used in the case analyzed in this thesis.

4.1 Basler's Camera Results

The first part of this Chapter is dedicated to the *Basler's Camera* problems and results. The first important aspect to mention for the following discussion is that this kind of cameras (*matrix cameras*) are not designed to capture frames of moving objects. However, it has been decided to study the behavior of this device in the Conveyor Tracking problem, in order to analyze the limits of this type of instruments when applied in the Conveyor Tracking problem.

The main issue in the application of Vision Systems is the presence of external light disturbances, that causes problems to the setting of the blob analyzer threshold. As shown in the previous Chapters, the items to detect are marked with a black area on the top, which represents also the type of the object.

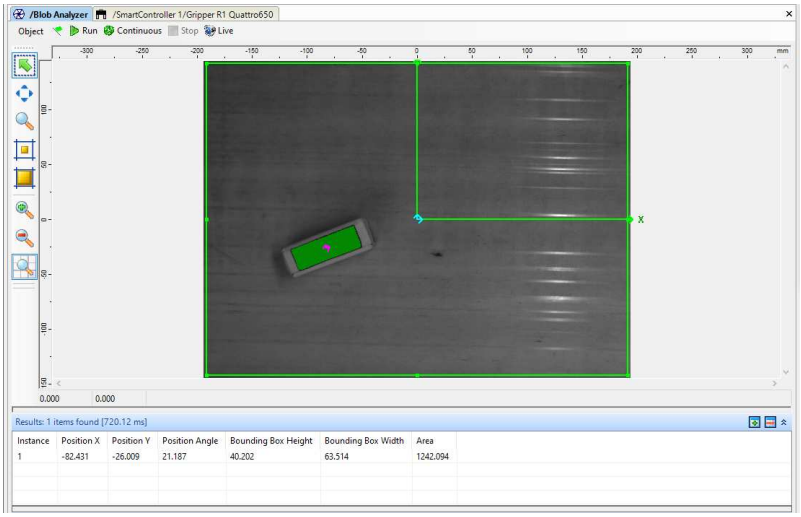


Figure 4.1: Basler Camera Blob Analyzer Result

In Figure 4.1 can be seen an example of a real image taken from the *Basler's Camera* and analyzed with the Blob Analyzer feature in the *Adept ACE* environment.
In this figure can be notice that the belt is not perfectly white, but presents some darker imperfections that could be sensed like other pieces, and some reflections of light on the right of the image, that can cause errors in the image analyzing.
In the picture is shown the reference system of the Blob Analyzer program and, on the bottom, the results obtained from the inspection of the image.

In order to avoid the detection of the darker imperfections on the belt it is necessary to set a threshold on the area of the blobs. Given the dimensions of the objects used in the experiments, it has been chosen the range between 1000 mm^2 and 2000 mm^2. This range allows to do not detect the small blemishes of the belt and discard objects too big to be captured by the Industrial Robot gripper.

The next problem to take into account is the setting of the threshold for the

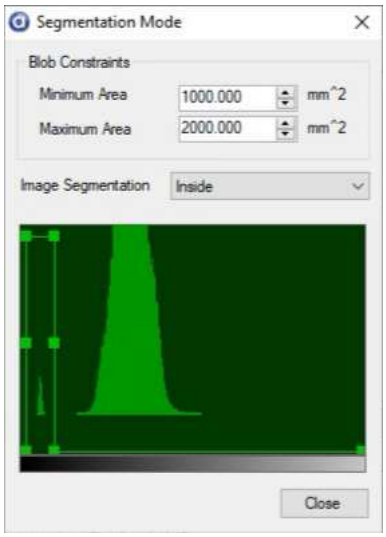


Figure 4.2: Basler Camera Blob Analyzer Threshold Definition

gray level in the blob analyzer. As already mentioned in Chapter 3, in the *ACE Sight Blob Analyzer* environment it is possible to set the threshold for the image segmentation.
As can be notice in Figure 4.2, the majority of the bits are the ones with gray level similar to the belt, while the smaller group are the bits similar to the color of the item markers.
Setting the *hard threshold* as shown in the previous figure is possible to make the Blob Analyzer detect only the markers of the objects that are travelling on the belt.

As already mention before, one of the main problem regarding the Camera Vision system is about the light disturbances, that can introduce errors in the analysis of the image. The changing in the light over the Conveyor Belt can be very problematic when applying the image segmentation for the detection of the items, since it introduces variations in the gray level values of the bits of the image.
To face with these variations it is necessary to modify the threshold about the gray level of the image segmentation. For example, if the light increase, also the marker of the items could be too bright for the threshold and the blob analyzer may not detect the items. In the opposite way, if the light decreases all image get

darker and the blob analyzer may detect dark areas of the Conveyor Belt, or the shadows of the items, as object to be taken.

An another issue that arises with this kind of cameras lies in the fact that these devices are not designed for the moving applications.
The explanation of this problem regards the concept of *exposure time* of the cameras. The exposure time is the amount of time in which the shutter remains open to allow light to enter the sensor, which means that, even if the photo is very fast, the image acquisition is not instantaneous.
The presence of this delay can cause issues in taking a photo of moving objects, because while the picture is taken, i.e. the shutter is open, the items are still moving. This fact can cause two main types of problems:

- wrong synchronization between the photo and the trigger signal
- distortion of the taken picture

Talking about the first problem, it is important to mention that the exposure time of the Basler's Camera has been set to $\simeq 20\ ms$. Tacking into account the speed of the Conveyor Belt defined in Table 3.2, it can be directly derived that the maximum amount of space travelled by the items during the exposure time is $\simeq 6\ mm$.
In other words, when the camera starts the procedure to capture the image of the belt, it sends the trigger signal to the controller of the Industrial Robot, which records the current position of the Conveyor Belt. However, due to the presence of the exposure time, the actual position of the belt when the image is sent to the Blob Analyzer is different, with a displacement of at maximum $\simeq 6\ mm$, in proportion to the speed of the belt.

Unlike the first issue, in which the presence of the exposure time creates problem in the actual position of the Conveyor Belt, the second one regards the Computer Vision software. The motion of the objects during the capturing of the image may creates some distortions in the resulting picture taken by the matrix camera.

Figure 4.3: Example of Motion Error during Exposure Time

Looking at Figure 4.3 can be seen an example of the frame acquisition of moving objects. The distortion of the moving items is perpendicular to the direction of travelling and it can cause some problems in the CV image analyzing.
In the previous picture can be notice that the shape of the moving items is elongated and the boundaries of the objects are faded. This two issues in the image causes the following troubles in the image segmentation:

- errors in area calculation
- errors in localization
- errors in the objects recognition

Recalling the principle of Image Segmentation and of threshold for the gray level it is possible to understand the origin of this two problems. Since the items in the picture are elongated, the objects are shown in a different shape with refers to the real one. This error implies that also the Blob Analyzer detects different shapes with refers to the real ones and may returns wrong results for location, orientation, and object recognition, when analyzing the image.

At this problems it is added also the fact that the edges of the items are faded which increases the errors in the Blob Analyzer task. The shades of the edges turns in different gray levels in the represented image, and this fact comes into conflict with the threshold imposed in Figure 4.2, since it may happen that a portion of the area of the item marker is not detected from the Blob Analyzer.

The most catastrophic event is that, due to the distortions of the items, the Blob Analyzer do not detect anymore the objects. This fact can happen if the shaded portion of the marker is too big, then the item results too small with refers to the area threshold imposed before, and the *CV* program do not detect the pieces.

In order to manage all these possible errors it has been necessary to take into account some solutions. Firstly it has been implemented one source of light above the area of the camera frame, in order to minimize the shadows and to emphasize the difference between the color of the item markers and the color of the Conveyor Belt.
Furthermore, it has been added a cover sheet allover the Industrial Robot working area, in order to minimize the external light disturbances for the Computer Vision program. In this way it has been feasible to set the best possible thresholds for the blob area and for the gray level, without modifying the values in relation to the available light.

After the implementation of all the solutions it has been possible to perform some experiments with the system in the best achievable conditions, with the minimum light disturbances, the most similar to the ambient of a real industrial application.
In Figure 4.4 is reported an example of the grabbing task in the Conveyor Tracking system realized with the Basler's Camera. The first thing that can be notice from the pictures is that the tracking task is correctly done, since the end-effector of the Industrial Robot is centered with refers to the longitudinal and transversal dimensions of the item.

The most important result is that, with refers to the direction of travelling of the object, the end-effector reach the item in the center. This fact support the outcome of a correct implementation of the Conveyor Tracking problem, since the actual errors that may occur in the tracking of the objects regards their position with refers to the direction of motion.

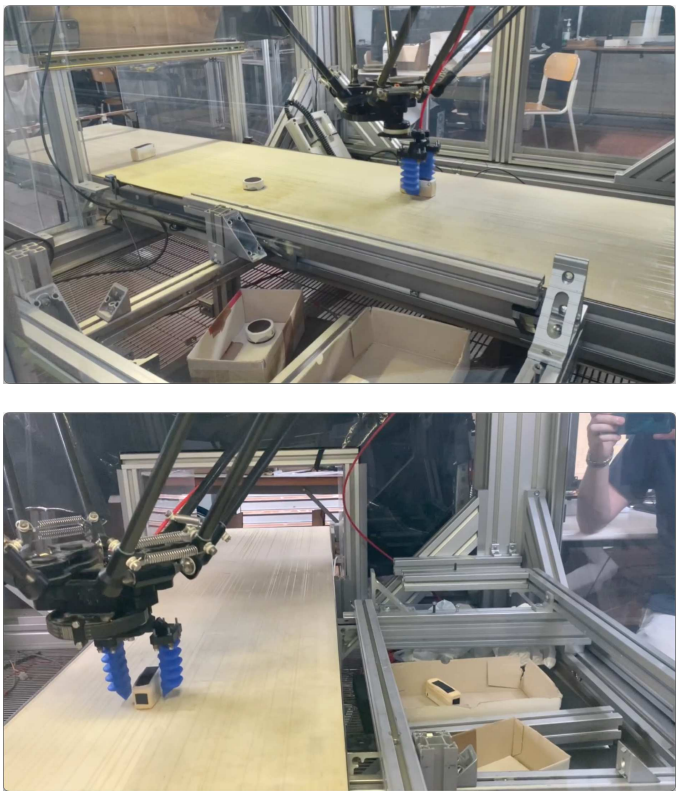


Figure 4.4: Example of Item Catching with Basler's Camera Vision

In the figure can also be seen that, with refers to the direction perpendicular to the motion of the belt, the end-effector is correctly orientate with the object, which means that also the computation of the transformation matrices between the different reference frames has been done correctly and the definition of the camera frame in the global reference frame is correct.

In order to analyze the performances of the solution, it has been chosen to test the system to derive the maximum possible precision of the detection task with refers to the speed of the Conveyor Belt. This study has been chosen since the core of this particular Conveyor Tracking problem lays in the identification and location of the objects and in the correctness of the information given to the Industrial Robot controller.

In Figure 4.5 is reported the plot that represents the maximum precision of the Computer Vision task in relation with the velocity of the Conveyor Belt. The

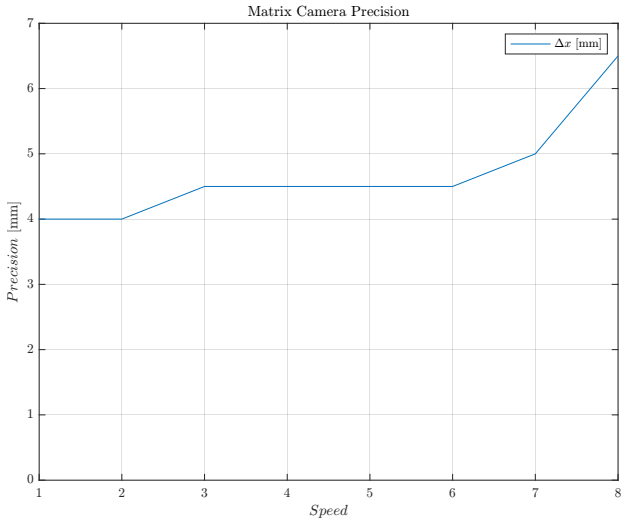


Figure 4.5: Basler Camera Precision Limit

data has been acquired by setting the speed of the belt and varying the threshold in the Adept ACE code used to distinguish if two pieces detected in two different photos are the same one or not.

The first thing that can be notice from the previous picture is that, with refers to the increasing of the belt speed, the loss of precision of the Basler's Camera do not grow fast, and it remains in the range between $\simeq 4\ mm$ and $\simeq 5\ mm$ almost to the maximum speed. On the contrary, if the speed increases too much, the Basler's Camera precision loss has a growth peak, as can be observed in the right side of the previous graph.
These results explains the fact that the matrix cameras are not designed for the application with moving objects. The peak in the loss of precision can be attributed to the image distortions and the faded boundaries of the items.
With refers to the information obtained it can be deduced that the solution implemented with the Basler's Camera is functional in relation to the requirements, but if the speed increase also the distortion and the nuance of the items increases. This fact implies that, even in the best possible conditions of light, the Computer Vision program may have some problems with small items.

4.2 Teledyne DALSA's Camera Results

After the test with the matrix camera it has been chosen to extend the study to the most appropriate kind of device for the Conveyor Tracking application. It has been chosen to implement the Teledyne DALSA's Linear Camera to acquire the images of the Conveyor Belt, since this type of cameras are designed for the applications with motion.

The first problem to solve, as for the Basler's Camera, it has been the light disturbance and the installation of a proper source of light, to make the Conveyor Belt correctly visible from the camera. Even with the presence of a light source, the images captured from the DALSA's camera are quite dark, but adding a simple *gain* to the picture with the Matlab program, it has been possible to have brighter and more contrasting images to analyze.

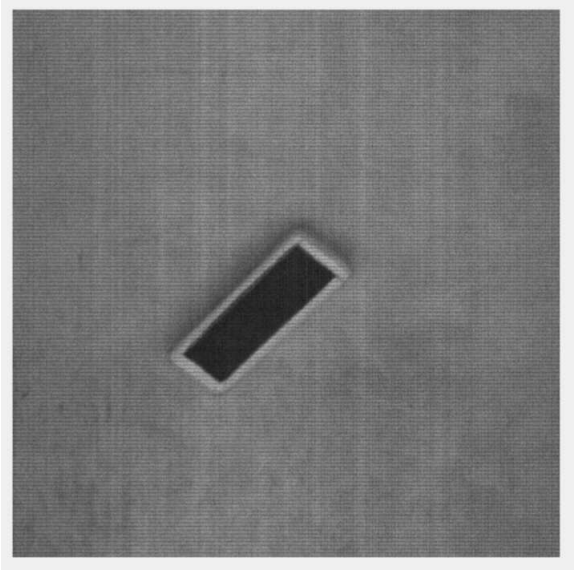


Figure 4.6: Linear Camera Captured Image

In Figure 4.6 can be see an example of a captured picture ready to be analyzed with the Matlab Blob Analyzer feature, in order to derive the necessary information of the item.

An another important problem to solve is that, unlike the matrix cameras

in which is possible to take two photo in sequence and overlap a portion of the frame, with the Teledyne Linear camera it has been necessary to continuously keep in memory the last image acquired by the linear camera. This solution is necessary since, after the acquisition of a certain portion of area that has passed through the linear frame, it is not possible to capture again that area, because it has already surpassed the line of bits that the camera can capture.
This memorization is necessary to avoid the problem of acquiring only a portion of the items in one photo and the other part in the successive picture. Then, saving the current photo is possible to chain the new acquired image to the old one, to make sure that all the items are properly captured in the image frame.
The Figure 4.6 is actually composed by two images concatenated and it represents exactly the described problem. In fact, cutting the photo in half horizontally, it is possible to notice that the item is also divided in the two parts.

After the Image Analysis performed by the Matlab code, it is necessary to send the information to the Industrial Robot controller, in which is contained the main program that manages the list of objects and the Industrial Robot movements.
Using the connection established between the Matlab program and the Industrial Robot controller, as described in Chapter 3, and after computing the message with the information, as shown in Appendix B, it is possible to send the results computed by the Matlab Blob Analyzer feature.

The next problem to solve is the synchronization between the two programs, since they has different computing times, different tasks to accomplish and different operating behaviour. For solving this issue it has been used the communication and the latch feature of the Industrial Robot controller.
In order to implement a solution to this problem, it is important to recall an important concept: the Teledyne Linear Camera is continuously capturing lines of bits and reconstructing pictures while the Conveyor Belt is running. If a picture is not acquired by the Matlab program, it is saved in the internal buffer of the camera, which can store at least one image. If a new picture is completed before

the old one has been send to the Matlab software, the buffer is overwritten by the DALSA camera and the old photo gets lost. Then, it is necessary to execute the `snapshot` command to acquire the image before the camera overwrites the buffer. It is also important to mention that, if the command is executed before the end of reconstruction of the image and the buffer is empty, the *Matlab* program waits the current image to be completed and the Teledyne Camera sends the picture immediately after its reconstruction.
On the contrary, if there is a picture in the buffer and the `snapshot` command is executed, the image in the buffer is sent immediately to the *Matlab* software.

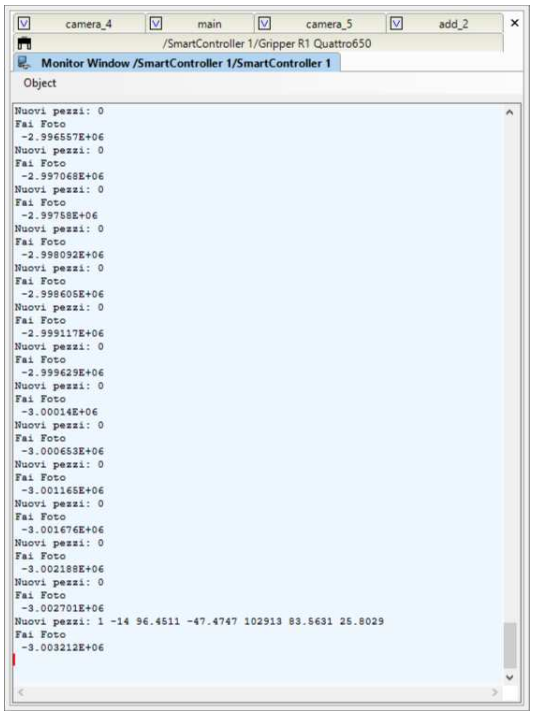


Figure 4.7: Linear Camera Messages of Results

In Figure 4.7 is reported the *Monitor Window* of the Adept ACE software in which are printed the messages sent between the Industrial Robot controller and the Matlab software.

Coming back to the starting problem, to synchronize the two program it has been necessary to send the request for the new information, from the Industrial Robot controller to the Matlab program, sending the string *Fai Foto*.

When the message is received from the Matlab program it starts the image acquisition with the command `snapshot` and it analyze the photo with the Blob Analyzer feature. When the image is sent from the linear camera to the computer, an electrical signal activates the latch feature of the Industrial Robot controller, in order to save the actual position of the belt in that specific moment.
After the submission of the message, the main program waits for the electrical trigger from the linear camera. When the latch is activated, the actual position of the belt is saved and printed in the *Monitor Window*, then the program waits to receive the information of the Blob Analysis from the communication network.

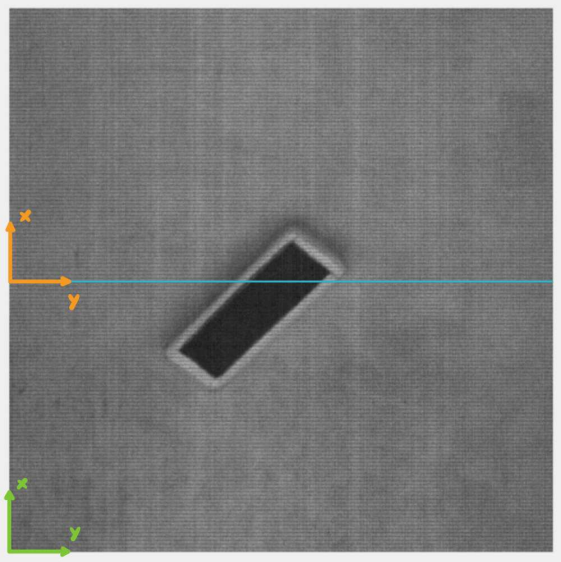


Figure 4.8: Linear Camera Expected Trigger Synchronization

In Figure 4.8 is shown the placements of the reference systems with refers to the operation described previously. The *orange* reference frame is related to the first portion of the image, while the *green* one is related to the second part of the photo. Recalling the explanation above, the reference system is placed when the trigger signal from the Linear Camera is sent to the Industrial Robot controller, which means that it is placed at the end of each portion of photo.

However, during the experiments it has been noticed that the actual behaviour of the two programs causes a different synchronization with respect to the one described above, so the trigger is shifted with refers to the previous explanation.

In fact, within the running of the system, the trigger used for the synchronization of the current photo is actually the one of the previous portion of photo.

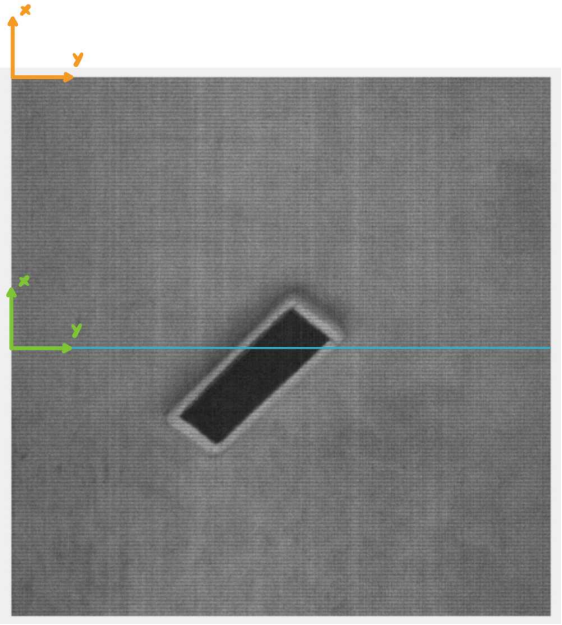


Figure 4.9: Linear Camera Real Trigger Synchronization

In Figure 4.9 is shown the real position of the reference systems placed in correlation of the latch signal received by the Industrial controller from the Linear camera. With refers to Figure 4.8 it can be noticed that the reference systems are shifted up by half of the image.

The real operation relies on the fact that, during the running of the Conveyor Belt, the Linear camera acquires lines of bits and, due to the addition of some computational delays, the buffer of the camera gets saturated before the Matlab program is able to execute the command `snapshot` and get the photo from the camera.
This mean that, when the `snapshot` command is executed, the photo is immediately sent from the buffer to the Matlab program and this means that also the trigger signal is sent to the Industrial Robot controller, activating the latch feature and saving the position of the belt. This position is actually the position of the end of the earlier image (which is also the beginning of the new photo) already analyzed by the Matlab program, and so the reference frame related to

the latch is placed at the beginning of the image.

Summarizing, the actual behaviour is that the Matlab program receives an image already scanned by the Linear camera and the Camera sends the trigger to the controller, which saves the triggered position of the belt and waits for the data from the Matlab software. After receiving the data and managing the list, the Industrial Robot controller sends the command *Fai Foto*, in order to obtain new data from the Matlab program. During these operation the Linear camera scans a new portion of the belt and, when the message is received from the Matlab software, a new image is already saved in the buffer of the Linear camera, and so on.

There are some considerations to take into account about this operation mode. The first one is that, as already mention in the previous Chapters, this is not the best implementation for the Linear camera, since the synchronization is not the best one. The communication between the two programs and the different computation time may cause other delays and other issues to the correct synchronization and the perfect tracking of the items.

The other important consideration regards the dimension of the photo that the Linear camera has to compute, since also this setting can affect the synchronization of the two programs. During the tests it has been noticed that the dimension of the image is related to the speed of the Conveyor Belt and it is necessary to chose the best dimension with refers to the velocity chosen for the belt.
The dimensions of the images of the Linear camera can be decided by changing the settings of the camera, to make the dimension fit in the best possible way and it has been deduced that the best dimensions are the ones defined in Table 4.1.

From the Table 4.1 can be observed that for each range of speed is related a certain value for the *Height* parameter. If is used a small value of dimension of the photo for high values of speed, for example $Height = 512$ and $Speed = 6$,

Image Height [*pixel*]	Belt Speed
512	*from* 0 *to* 3
1024	*from* 4 *to* 6
1536	*from* 7 *to* MAX

Table 4.1: Image Height w.r.t. Belt Speed

it happens that the Linear camera captures more than one photo between two executions of the command `snapshot`. Then, some portions of the Conveyor Belt scanned by the Linear camera may be lost, or there might be errors in the synchronization between the trigger and the data derived by the Matlab code.

On the contrary, if it is chosen an high value for the dimension parameter and a low one for the belt speed, for example $Height = 1536$ and $Speed = 2$, it may happen that the Linear camera do not have enough time for scanning an entire photo between two execution of the command `snapshot`. This fact creates problem for the synchronization between the two programs, since the code is designed for placing the reference frame at the beginning of the photo, with the operation mode described previously.
However, even if the program has been designed for placing the reference system at the end of the photo, there could be delays in the communication between the two programs, that may cause the wrong synchronization between the reference frames and the trigger signals.

After the managing of all the problems and issues described before, it has been tested the actual performances of the Conveyor Tracking system with the implementation of the Teledyne Linear Camera for the Computer Vision task.
In Figure 4.10 is reported an example of the grabbing task of the Conveyor Tracking system realized with the Teledyne DALSA Camera.
The first aspect that can be observed from the figures is that the object tracking is correctly done by the system, since the end-effector of the Industrial Robot is centered with refers to the longitudinal and transversal dimensions of the item.

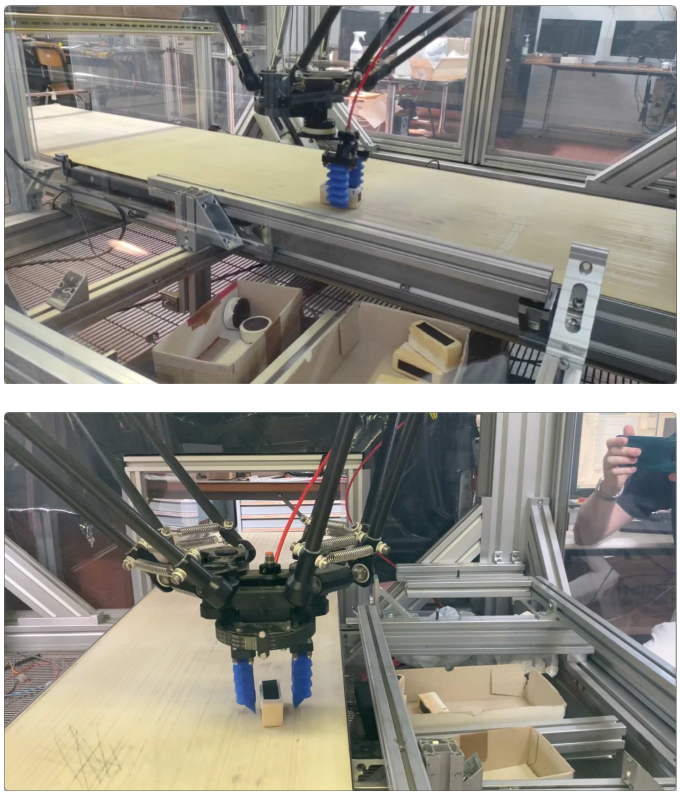


Figure 4.10: Example of Item Catching with Linear Camera Vision

The most important result is that, with refers to the direction of travelling of the object, the Industrial Robot reaches the item in the center. This fact support the outcome of a correct implementation of the Conveyor Tracking task, since the errors that may occur in the tracking of the items over the belt mainly regards the position with refers to the direction of travel.

In the picture can also be noticed that, with refers to the direction perpendicular to the motion of the belt, the end-effector is correctly orientate with the object, but there is a small error in the position. This issue can be related to the computation of the transformation matrices between the different reference frames, due to the manual measurements. This problem is also easily solvable with the addition of an offset to the camera frame definition, in order to match the real position of the image reference system.

As already done for the Basler's Camera, it has been chosen to test the system

to determine the maximum possible precision of the detection task in relation to the velocity of the Conveyor Belt.

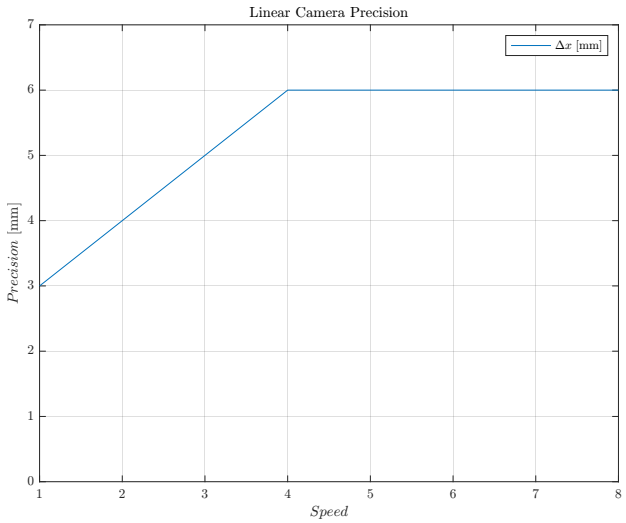


Figure 4.11: Teledyne DALSA Camera Precision Limit

In Figure 4.11 is shown the plot that represents the maximum precision reachable from the Computer Vision program with refers to the speed of the belt. As done for the Basler's Camera, the data has been recorded by choosing a constant velocity and varying the parameter related to the distinctiveness between two near objects.

The first aspect that can be observed from the previous picture is that the starting precision related to the minimum speed of the belt is smaller than the first value of the plot in Figure 4.5, and this means that the starting precision of the Linear camera is higher than the one of the Basler camera.
This fist fact can be related to the lower resolution of the Basler camera with refers to the resolution that can be obtained with the Linear camera, since at the minimum speed the problems related to the *exposure time* of the Matrix camera is negligible in term of the maximum precision achievable.

From Figure 4.11 it can be noticed that the precision of the Linear camera grows linearly in the range between 1 and 4 of the velocity, but then it remains constant till the maximum available speed of the Conveyor Belt.

The main aspect is that the precision remains constant after a certain points, which means that the system, and more specifically the camera, has reached the best operating regime. This fact relies on the predisposition of the linear cameras to the applications with moving objects or moving belts.
However, in the first part of the plot the precision grows linearly, due to the delays in the operation of the two program together. For the same reason it can be expected that, increasing the belt speed over the actual maximum velocity, the error may slowly grows, due to the imperfect implementation of the Teledyne camera with the system.

4.3 Problem of Overcrowding of Conveyor Belt

In the last Section of this Chapter it is discussed an another issue related to the Conveyor Tracking task, which creates problems either to the Computer Vision program and to the Industrial Robot moving task.
During the test it has been noticed that the system under consideration has some malfunctioning if over the belt are travelling too many pieces, or if the pieces are too close to each other.

4.3.1 Adding More Industrial Robots

Talking about the Industrial Robot itself, it has been observed that, if there are too many objects that arrives the Industrial Robot maximum speed may not be enough to catch all the piece. This problem is more likely to occur when the speed of the belt is high, but it can happen even if the belt velocity is smaller than the maximum speed.
In the first case, if the speed of the belt is near to the maximum, it is sufficient that two objects arrives together under the robot to make the grasping task difficult, because while the robot is grabbing the fist one, the other travel fast to the end of the reachable workspace and the robot may not have enough time to reach the second object before it exits the workspace.
On the contrary, if the speed of the belt is not too much high, the problem may arise when there are many objects that arrives together to the Industrial Robot.

In this case, while the Industrial Robot is grabbing some objects, the other items travels to the end of the reachable workspace, and it may happen that the robot is not fast enough to catch all the object in time.

Figure 4.12: Conveyor Tracking System with Multiple Industrial Robots

However, the problem described for the Industrial Robot speed is easily solvable with the addition of more robot in sequence, as shown in Figure 4.12.
This solution can be implemented in two main ways: the first one is making all the robots grab all the type of pieces, with priority or not, and if some pieces are too far ahead for one robot, the task is left to the next one in sequence.
An another way to solve this problem is making each robot grab only one type of items and ignore the other ones, in order to significantly reduce the amount of work for each Industrial Robot. However, with this last solution it may happen that on the belt travels clusters of the same kind of objects and, in this way, the problem arises again.

4.3.2 Computer Vision Delays

Coming back to the Computer Vision program, the presence of many objects may affect the correct behaviour of the vision task and image analysis.
During the tests of the system it has been noticed that, if there are too many items travelling on the belt, the Computer Vision program can be affected by some delays, due to the increase of the time needed for the computation of all the results with respect to the scenario where there are less items.

Basler's Camera

In the Vision system implemented with the Basler's Matrix Camera this issue is present when the number of items is actually high and only if the speed of the belt is near to the maximum. More specifically, this problem begins to appear when the speed of the belt is set to 7 or 8 (max) and when in the image frame of the camera are captured more than 4 objects, which means that the belt is very overcrowded.

Recalling Figure 3.3, in which are explained the two parallel programs for the implementation of the Basler's Camera, it can be observed that the problem described before is related to the fact that the acquisition and analysis of the photos is punctuated with refers to the amount of space travelled by the belt.
Then, if there are too many items, the Blob Analyzer of the Adept Sight package has no enough time to compute all the results of the current image before the acquisition of a new picture, since the computation needs more time and the belt is travelling at high speed.

Teledyne DALSA Camera

Talking about the case related to the implementation of the Teledyne Linear Camera, this issue arises with more probability than in the other case.
This fact is related to the communication between the two software and the difficulties in the synchronization between the two programs. The presence of multiple objects in the image frame introduces some delays in the computations of the results, as mentioned in the previous section.

Also in this case, the speed of the belt is related to the occurrence of this problem, but in different way with respect to the Matrix Camera. Recalling the synchronization between the two environments explained in the previous sections, the increasing of time in the computation of the result of the Blob Analyzer can affect the synchronization between the trigger signal and the actual position of the belt.
In these cases the information sent from the Matlab program to the Industrial

Robot controller are assigned to the wrong latch value, i.e. to the wrong belt position, making the tracking incorrect.

Conclusions

In conclusion, it can be said that the design of the management system for the Conveyor Tracking problem has been achieved. In this thesis it has been discussed this problem starting from a general purposed solution, passing through the implementation of the described solution into the given system, showing that the Conveyor Tracking system can be realized with the use of a Computer Vision system.
The results reported in Chapter 4 supports the thesis that, even with some minimal problems, the implementation of the solution is possible and the system can achieve considerable high levels of precision with both the two different type of cameras exploited.

Summarizing the discussion of Chapter 4, the main argument to recall is that at the minimum speed the Linear Camera has a better precision with refers to the Matrix Camera, but after the belt speed 2, the Matrix Camera precision remains better till the end of the graph, where there is the peak of precision loss of the Basler's Camera.
However, as discussed in the Chapter, the precision of the Linear Camera get saturated after a few incrementing of the belt speed, while the precision loss of the Basler's Camera is continuously growing.

Basler's Camera Conclusions

Recalling the results of Chapter 4 relative to the Basler's Camera it can be said that, even if the Matrix Camera is not designed for the implementation of Vision systems employed with moving objects or moving models, it is a valid choice for

the realization of the the solution for the Conveyor Tracking in the analyzed system.

In particular, it has been observed that, for the velocities that the used belt can reach, the maximum precision of this camera is very good, but for more fast applications it may not be the best choice.
Similarly, the perfect link between the Adept ACE environment with the Basler's Cameras allows a very good implementation of this camera with the system, and so it returns an optimal reliability of the system against disturbances and communication errors.

Teledyne DALSA's Camera Conclusions

Talking about the Teledyne DALSA Linear Camera, in general it is the best choice for the implementation of Vision systems employed with moving items, since this kind of cameras are designed for this applications.

Recalling the results of Chapter 4, the analysis of the Conveyor Tracking system has reported that the implementation of this camera, by using the Matlab environment for the acquisition of images and detection of the objects, has a very good precision. In particular, even if there is a precision loss at the beginning, the maximum precision saturates after few speed increments, which means that this particular camera can be exploited even for faster system than the one that has been analyzed.
However, the test also reported that this method is very sensible to the disturbances and the delays of the system, since it is based on the communication and the synchronization of two different programs. For these reasons, other problematic may be introduced if this solution is applied to more fast applications.

The Best Choice

Summarizing all the results and the discussion of the previous sections and chapters, it can be said that the best choice for this particular system and application is the Basler's Camera. However, it must be said that, with a better implemen-

tation and a better link with the *Adept ACE* environment, the Teledyne DALSA Camera is actually the best possible choice for the Conveyor Tracking problem.

Appendix A

Adept ACE Program

In this Appendix is reported the two codes for the *Adept ACE 3.6* environmet for each of the two different cameras implemented.
The basics of the programming code has been recalled from the *Industrial Robotics course* [8], while for the specific code it has been used the *Adept eV+ User's Guide* [5].

A.1 Basler Camera

```
.PROGRAM definitivo_4()

ATTACH ()

;define relevant locations
SET lochome = TRANS(0,0,-950,0,180,180)
SET locway = TRANS(50,-200,-950,0,180,180)
SET locplace_rect = TRANS(200,-400,-960,0,180,180)
SET locplace_sq = TRANS(-100,-400,-960,0,180,180)
SET loccamera = TRANS(-541.687,-27.761,-1055,0,180,180)
dz_pick = 40
dz_place = 40

;define belt parameters
sf = -0.25
ENABLE BELT
DEFBELT %locbelt = loccamera, 1, 1, sf
WINDOW %locbelt = TRANS(600,500,0), TRANS(-250,-500,0)

;resetting the position and speed
```

```
SPEED 100 ALWAYS

MOVE lochome
BREAK
OPENI

home = 1

DELAY 0.5

WHILE TRUE DO

    piece_to_pick = 1

    ; checks if there are pieces in the list
    IF piece_list >= 1 THEN
        IF (-list[piece_to_pick,0]+(-541.687)+((DEVICE(0,0,,1)-list[piece_to_pick,3])*sf)) > 300 THEN
            ; if first object is too much distant then we take it
            CALL definitivo_mov5(list[piece_to_pick,0], list[piece_to_pick,1], list[piece_to_pick,2], list[piece_to_pick,3], list[piece_to_pick,4])
            ; removing the first object of the list
            CALL definitivo_rem2(piece_to_pick)

        ELSE
            FOR p = 1 TO piece_list

                x_abs_p = -list[p,0]+(-541.687)+((DEVICE(0,0,,1)-list[p,3])*sf)
                x_abs_topick = -list[piece_to_pick,0]+(-541.687+correction_x)+((DEVICE(0,0,,1)-list[piece_to_pick,3])*sf)

                IF (list[p,4] > list[piece_to_pick,4]) AND ((x_abs_topick-x_abs_p) < 200) THEN
                    piece_to_pick = p
                END
            END

            ; check if the first object of the list to reach a safe distance, i.e. enters the robot working space
            IF (-list[piece_to_pick,0]+(-541.687)+((DEVICE(0,0,,1)-list[piece_to_pick,3])*sf)) > -200 THEN

                ; pick the first object of the list and put it in the "place" location
                CALL definitivo_mov5(list[piece_to_pick,0], list[piece_to_pick,1], list[piece_to_pick,2], list[piece_to_pick,3], list[piece_to_pick,4])
                ; remove the piece placed from the list
                CALL definitivo_rem2(piece_to_pick)
```

```
            END
        END
    END
END

DETACH ()
.END
```

```
.PROGRAM definitivo_add8(x, y, alpha, trigger, height, width, area)

    threshold_x = 7.5   ;threshold for the same piece
    threshold_y = 2
    threshold2 = 10     ;threshold for the type of piece
    threshold3 = 20     ;threshold to correct the error on the type of piece

    sim = ABS((area/height)-width)
    offset = ((ABS(DEVICE(0,0,,2)/1250))*15)

    ;flag used to check if there already is the new element
    check = TRUE

    x_temp = x*COS(-1.42)+y*SIN(-1.42)+offset
    y_temp = y*COS(-1.42)-x*SIN(-1.42)
    alpha_temp = alpha-(-1.42)

    IF piece_list == 0 THEN
        list[piece_list+1,0] = x_temp
        list[piece_list+1,1] = y_temp
        list[piece_list+1,3] = trigger
        list[piece_list+1,5] = area

        IF (ABS(height-width) <= threshold2) THEN
            IF (sim < threshold3) THEN
                ;adding circle element in the list
                list[piece_list+1,2] = 0
                list[piece_list+1,4] = 1
            ELSE
                ;adding a rectangular element
                list[piece_list+1,2] = alpha_temp
                list[piece_list+1,4] = 2
            END
        ELSE
            ;adding a rectangular element
            list[piece_list+1,2] = alpha_temp
            list[piece_list+1,4] = 2
        END

        ;updating the number of element in the list
        piece_list = piece_list+1
```

```
42      ELSE
43          FOR k = 1 TO piece_list
44              offset_x = -(x_temp-(list[k,0]-((trigger-list[k,3])*sf)))
45
46              IF (y_temp <= list[k,1]+threshold_y) AND (y_temp >= list[k,1]-
    threshold_y) THEN
47
48                  IF (x_temp <= list[k,0]-((trigger-list[k,3])*sf)+threshold_x)
    AND (x_temp >= list[k,0]-((trigger-list[k,3])*sf)-threshold_x) THEN
49
50                      check = FALSE
51
52                      IF (area > list[k,5]) THEN
53                          list[k,0] = list[k,0]+offset_x
54                          list[k,5] = area
55
56                          IF (ABS(height-width) <= threshold2) THEN
57                              IF (sim < threshold3) THEN
58                                  ;adding circle element in the list
59                                  list[piece_list+1,2] = 0
60                                  list[piece_list+1,4] = 1
61                              ELSE
62                                  ;adding a rectangular element
63                                  list[piece_list+1,2] = alpha_temp
64                                  list[piece_list+1,4] = 2
65                              END
66                          ELSE
67                              ;adding a rectangular element
68                              list[k,2] = alpha_temp
69                              list[k,4] = 2
70                          END
71                      END
72                  END
73              END
74          END
75
76          IF check THEN
77              list[piece_list+1,0] = x_temp
78              list[piece_list+1,1] = y_temp
79              list[piece_list+1,3] = trigger+offset
80              list[piece_list+1,5] = area
81
82              IF (ABS(height-width) <= threshold2) THEN
83                  IF (sim < threshold3) THEN
84                      ;adding circle element in the list
85                      list[piece_list+1,2] = 0
86                      list[piece_list+1,4] = 1
87                  ELSE
```

```
                    ;adding a rectangular element
                    list[piece_list+1,2] = alpha_temp
                    list[piece_list+1,4] = 2
                END
            ELSE
                ;adding a rectangular element
                list[piece_list+1,2] = alpha_temp
                list[piece_list+1,4] = 2
            END

            ;updating the number of element in the list
            piece_list = piece_list+1
        END
    END
.END
```

```
.PROGRAM definitivo_cam()

    ;define camera parameters
    $ip = "192.168.0.25"
    sequence_id = 1
    tool_id = 2
    instance_id = 1
    result_id = 1
    idx = 1612
    idy = 1613
    idangle = 1617
    idcount = 1610
    idround = 1623
    idheight = 1626
    idwidth = 1627
    idarea = 1611
    index_id = 1
    frame_id = 1

    ;to make the buffer empty
    WHILE LATCHED(-1) DO
    END

    trig_pos = 0
    piece_list = 0
    piece_new = 0

    WHILE TRUE DO
        event = LATCHED(-1)

        ;waiting for the latch
        WHILE NOT event DO
            WAIT
```

```
            event = LATCHED(-1)
        END

        trig_pos = DEVICE(0,0,,4)

        ;waiting for the results of blob analyzer
        WAIT VSTATE($ip, sequence_id) == 3

        piece_new = VRESULT($ip,sequence_id,tool_id,instance_id,idcount,index_id
    ,frame_id)

        IF piece_new > 0 THEN
            IF DEVICE(0,0,,2) <= 0 THEN
                FOR i = 1 TO piece_new
                    x_new = VRESULT($ip, sequence_id, tool_id, i, idx, index_id,
     frame_id)
                    y_new = VRESULT($ip, sequence_id, tool_id, i, idy, index_id,
     frame_id)
                    alpha_new = VRESULT($ip, sequence_id, tool_id, i, idangle,
    index_id, frame_id)
                    area_new = VRESULT($ip, sequence_id, tool_id, i, idarea,
    index_id, frame_id)
                    height_new = VRESULT($ip, sequence_id, tool_id, i, idheight,
     index_id, frame_id)
                    width_new = VRESULT($ip, sequence_id, tool_id, i, idwidth,
    index_id, frame_id)

                    ;adding the new pieces detected
                    CALL definitivo_add8(x_new, y_new, alpha_new, trig_pos,
    height_new, width_new, area_new)
                END
            END
        END
    END
.END
```

```
.PROGRAM definitivo_mov5(x, y, alpha, pos_start, type)

    ;setting the belt initial position as the position when the object has been
    individuated
    SETBELT %locbelt = pos_start

    IF (type == 1) THEN
        SET locplace = locplace_sq
    ELSE
        SET locplace = locplace_rect
    END

    ;movements to reach and the grip the current object
```

```
    ;signs in the "trans" fuction are due to the orientation of the frame-axis
    of the cam and the robot one's

    ;grabbing object
    IF home == 0 THEN
        MOVE locway
    END

    home = 0
    APPRO %locbelt:TRANS(-x,y,0,-alpha,0,0), dz_pick
    BREAK
    MOVE %locbelt:TRANS(-x,y,0,-alpha,0,0)
    CLOSEI
    DELAY 0.2
    DEPART dz_place
    MOVE locway
    APPRO locplace, dz_place
    MOVE locplace
    OPENI
    DELAY 0.2
    DEPART dz_place

    IF piece_list == 1 THEN
        MOVE locway
        MOVE lochome
        BREAK
        home = 1
    END
.END
```

```
.PROGRAM definitivo_ph_2()

    WHILE TRUE DO
        VRUN $ip, sequence_id
        WAIT ABS((DEVICE(0,0,,1)-trig_pos)*sf) >= 250
    END
.END
```

```
.PROGRAM definitivo_rem2(rem)

    FOR l = rem TO piece_list-1
        ;shifting up the list from the last picked piece
        list[l,0] = list[l+1,0]
        list[l,1] = list[l+1,1]
        list[l,2] = list[l+1,2]
        list[l,3] = list[l+1,3]
        list[l,4] = list[l+1,4]
        list[l,5] = list[l+1,5]
    END
```

```
12
13      ;updating the number of element in the list
14      piece_list = piece_list-1
15  .END
```

A.2 Teledyne DALSA Camera

```
1   .PROGRAM main()
2
3       ATTACH ()
4
5       ;define relevant locations
6       SET lochome = TRANS(0,0,-950,0,180,180)
7       SET locway = TRANS(50,-200,-950,0,180,180)
8       SET locplace_rect = TRANS(200,-400,-960,0,180,180)
9       SET locplace_sq = TRANS(-100,-400,-960,0,180,180)
10      SET loccamera_2 = TRANS(-47.293-580,-0.433,-1055,0,180,180)
11      dz_pick = 40
12      dz_place = 40
13
14      ;define belt parameters
15      sf = -0.25
16      ENABLE BELT
17      DEFBELT %locbelt = loccamera_2, 1, 1, sf
18      WINDOW %locbelt = TRANS(600,500,0), TRANS(-250,-500,0)
19
20      ;resetting the position and speed
21      SPEED 20 ALWAYS
22
23      MOVE lochome
24      BREAK
25      OPENI
26
27      home = 1
28
29      DELAY 0.5
30
31      WHILE TRUE DO
32          piece_to_pick = 1
33
34          ; checks if there are pieces in the list
35          IF piece_list >= 1 THEN
36              IF (list[piece_to_pick,0]+(-47.293-580)+((DEVICE(0,0,,1)-list[
        piece_to_pick,3])*sf)) > 300 THEN
37
38                  ; if first object is too much distant then we take it
```

```
39                CALL move_1(list[piece_to_pick,0], list[piece_to_pick,1], list[
    piece_to_pick,2], list[piece_to_pick,3], 2)
40                ; removing the first object of the list
41                CALL remove(piece_to_pick)
42
43            ELSE
44                FOR p = 1 TO piece_list
45
46                    x_abs_p = list[p,0]+(-47.293-580)+((DEVICE(0,0,,1)-list[p
    ,3])*sf)
47                    x_abs_topick = list[piece_to_pick,0]+(-47.293-580)+((DEVICE
    (0,0,,1)-list[piece_to_pick,3])*sf)
48
49                    IF (list[p,4] > list[piece_to_pick,4]) AND ((x_abs_topick-
    x_abs_p) < 200) THEN
50                        piece_to_pick = p
51                    END
52                END
53
54                ; check if the first object of the list to reach a safe distance
    , i.e. enters the robot working space
55                IF (list[piece_to_pick,0]+(-47.293-580)+((DEVICE(0,0,,1)-list[
    piece_to_pick,3])*sf)) > -150 THEN
56
57                    ; pick the first object of the list and put it in the "place
    " location
58                    CALL move_1(list[piece_to_pick,0], list[piece_to_pick,1],
    list[piece_to_pick,2], list[piece_to_pick,3], list[piece_to_pick,4])
59                    ; remove the piece placed from the list
60                    CALL remove(piece_to_pick)
61                END
62            END
63        END
64    END
65
66    DETACH ()
67.END
```

```
1 .PROGRAM add_2(x, y, alpha, trigger, area, major, minor)
2
3     threshold_x = 6
4     threshold_y = 2
5     threshold_axis = 20
6
7     ;flag used to check if there already is the new element
8     check = TRUE
9
10    IF piece_list == 0 THEN
11        list[piece_list+1,0] = x
```

```
12          list[piece_list+1,1] = y
13          list[piece_list+1,3] = trigger
14          list[piece_list+1,4] = area
15
16          IF (ABS(major-minor) > threshold_axis) THEN
17              list[piece_list+1,2] = alpha
18              list[piece_list+1,5] = 2
19          ELSE
20              list[piece_list+1,2] = 0
21              list[piece_list+1,5] = 1
22          END
23
24          ;updating the number of element in the list
25          piece_list = piece_list+1
26
27      ELSE
28          FOR k = 1 TO piece_list
29              IF (y <= list[k,1]+threshold_y) AND (y >= list[k,1]-threshold_y)
    THEN
30                  IF (x <= list[k,0]+((trigger-list[k,3])*sf)+threshold_x) AND (x
    >= list[k,0]+((trigger-list[k,3])*sf)-threshold_x) THEN
31
32                      check = FALSE
33                      offset_x = (x-(list[k,0]+((trigger-list[k,3])*sf)))
34
35                      IF (area > list[k,4]) THEN
36                          list[k,0] = list[k,0]+offset_x
37                          list[k,4] = area
38
39                          IF (ABS(major-minor) > threshold_axis) THEN
40                              list[k,2] = alpha
41                              list[k,5] = 2
42                          ELSE
43                              list[k,2] = 0
44                              list[k,5] = 1
45                          END
46                      END
47                  END
48              END
49          END
50
51          IF check THEN
52              list[piece_list+1,0] = x
53              list[piece_list+1,1] = y
54              list[piece_list+1,3] = trigger
55              list[piece_list+1,4] = area
56
57              IF ABS(major-minor) > threshold_axis THEN
```

```
                list[piece_list+1,2] = alpha
                list[piece_list+1,5] = 2
            ELSE
                list[piece_list+1,2] = 0
                list[piece_list+1,5] = 1
            END

            ;updating the number of element in the list
            piece_list = piece_list+1
        END
    END
.END
```

```
.PROGRAM remove(rem)

    FOR l = rem TO piece_list-1
        ;shifting up the list from the last picked piece
        list[l,0] = list[l+1,0]
        list[l,1] = list[l+1,1]
        list[l,2] = list[l+1,2]
        list[l,3] = list[l+1,3]
        list[l,4] = list[l+1,4]
        list[l,5] = list[l+1,5]
    END

    ;updating the number of element in the list
    piece_list = piece_list-1
.END
```

```
.PROGRAM move_1(x, y, alpha, pos_start, type)

    ;setting the belt initial position as the position when the object has been
    individuated
    SETBELT %locbelt = pos_start

    IF (type == 1) THEN
        SET locplace = locplace_sq
    ELSE
        SET locplace = locplace_rect
    END

    ;movements to reach and the grip the current object
    ;signs in the "trans" fuction are due to the orientation of the frame-axis
    of the cam and the robot one's

    ;grabbing object
    IF home == 0 THEN
        MOVE locway
    END
```

```
19
20      home = 0
21      APPRO %locbelt:TRANS(x,y,0,-alpha,0,0), dz_pick
22      BREAK
23      MOVE %locbelt:TRANS(x,y,0,-alpha,0,0)
24      CLOSEI
25      DELAY 0.2
26      DEPART dz_place
27      MOVE locway
28      APPRO locplace, dz_place
29      MOVE locplace
30      OPENI
31      DELAY 0.2
32      DEPART dz_place
33
34      IF piece_list == 1 THEN
35          MOVE locway
36          MOVE lochome
37          BREAK
38          home = 1
39      END
40  .END
```

```
1   .PROGRAM tcpopen()
2
3       ATTACH (lun, 4) "TCP"
4
5       TYPE "lun: ", lun
6
7       status = IOSTAT(lun) ;Check status of ATTACH
8       IF status < 0 THEN
9           TYPE "Error from ATTACH: ", $ERROR(status)
10          CALL tcpclose(lun, $port)
11      END
12
13      FOPEN (lun, 16) "/LOCAL_PORT 30300"
14
15      status = IOSTAT(lun) ;Check status of FOPEN
16      IF status < 0 THEN
17          TYPE "Error from FOPEN: ", $ERROR(status)
18          CALL tcpclose(lun, $port)
19      END
20  .END
```

```
1   .PROGRAM tcpclose()
2
3       FCLOSE (lun)
4       DETACH (lun)
5       TYPE "TCP port 30300 CLOSED"
```

```
    RETURN
.END
```

```
.PROGRAM camera_4()

    trig_pos = 0
    piece_list = 0

    ;to make the latch buffer empty
    WHILE LATCHED(-1) DO
    END

    CALL tcpopen()

    ; empty the communication buffer
    READ (lun, do_wait) $datas

    TYPE $datas

    p = 1
    DO
        $temp = $DECODE($datas," ",0)   ;take the first number
        data[p] = VAL($temp)            ;convert to real value
        $temp = $DECODE($datas," ",1)   ;delete the space between number
        p = p+1
    UNTIL $datas == ""

    WHILE TRUE DO

        ; take snapshot
        WRITE (lun) "faifoto"
        TYPE "Fai Foto"

        event = LATCHED(-1)
        ; waiting for the latch
        WHILE NOT event DO
            WAIT
            event = LATCHED(-1)
        END

        ; saving latch pos
        trig_pos = DEVICE(0,0,,4)
        TYPE trig_pos

        ; waiting for the results
        READ (lun, , do_wait) $datas
        TYPE "Nuovi pezzi: "+$datas

        p = 1
```

```
47          DO
48              $temp = $DECODE($datas," ",0) ;take the first number
49              data[p] = VAL($temp) ;convert to real value
50              $temp = $DECODE($datas," ",1) ;delete the space between number
51              p = p+1
52          UNTIL $datas == ""
53
54          IF data[1] >= 1 THEN
55              FOR i = 1 TO data[1]
56
57                  CALL add_2(data[2+((i-1)*6)], data[3+((i-1)*6)], data[4+((i-1)
    *6)], trig_pos, data[5+((i-1)*6)], data[6+((i-1)*6)], data[7+((i-1)*6)])
58
59              END
60          END
61      END
62  .END
```

Appendix B

Matlab Program

In this Appendix is reported the code of the *Matlab* program used in the implementation of the *Teledyne DALSA Linear Camera*.

```
% creating the camera
g = gigecam(1);

;g.Height = 512;
g.Height = 512*2;
;g.Height = 512*3;

g.rotaryEncoderMultiplier = 4;
g.rotaryEncoderDivider = 1;

g.LineSelector = 'Line3';
g.LineMode = 'Output';
g.LineFormat = 'OpenCollector';

g.outputLineSource = 'PulseOnStartofFrame';
g.outputLinePulseDuration = 50;
g.LineInverter = 'True';

% variables definition
new_msg = [];
data = "";

threshold = 60;
area_trh = 90000;

f_x = 0.1044;
f_y = 0.1226;

```

```
29 gain = 3;
30
31 blob = vision.BlobAnalysis('BoundingBoxOutputPort',false,'MinimumBlobArea',
       area_trh,'OrientationOutputPort',true,'MajorAxisLengthOutputPort',true,'
       MinorAxisLengthOutputPort',true);
32
33 clear con_inv;
34
35 % connection to the ACE server for the communication
36 ip_controller = "192.168.0.1";
37 port_inv = 30300;
38 con_inv = tcpclient(ip_controller, port_inv);
39
40 first = 0;
41
42 while true
43     if (con_inv.NumBytesAvailable > 0)
44         new_msg = split(convertCharsToStrings(char(read(con_inv,con_inv.
       NumBytesAvailable))));
45
46         if (new_msg(1,1) == "faifoto")
47             if (first == 0)
48                 new_img = snapshot(g);
49
50                 buffer = new_img;
51
52                 % analisi buffer
53                 dim_img = size(new_img);
54                 temp_blob = image_filter(new_img, threshold, dim_img(1), dim_img
       (2), gain);
55                 [area, center, major, minor, angle] = blob(temp_blob);
56                 blob_numb = size(area);
57
58                 data = string(blob_numb(1,1));
59
60                 if (blob_numb(1,1) >= 1)
61                     for k=1:1:blob_numb(1,1)
62                         [y_new,x_new,angle_new] = axis_trans(center(k,1), center
       (k,2), f_x, f_y, 0, 0, angle(k));
63
64                         % sending the number of blobs and datas
65                         data = string(data+" "+x_new+" "+y_new+" "+angle_new+"
       "+area(k)+" "+(major(k)*f_y)+" "+(minor(k)*f_y));
66                     end
67
68                     cmd = convertStringsToChars(data);
69                     write(con_inv, uint8(cmd));
70                 else
```

```
                    % sending the number of blobs is 0
                    cmd = convertStringsToChars(data);
                    write(con_inv, uint8(cmd));
                end

                first = 1;
            else
                new_img = snapshot(g);
                temp = vertcat(buffer, new_img);

                % analysis temp
                dim_img = size(temp);
                temp_blob = image_filter(temp, threshold, dim_img(1), dim_img(2)
    , gain);
                [area, center, major, minor, angle] = blob(temp_blob);
                blob_numb = size(area);

                data = string(blob_numb(1,1));

                if (blob_numb(1,1) >= 1)
                    for k=1:1:blob_numb(1,1)

                        [y_new,x_new,angle_new] = axis_trans(center(k,1), center
    (k,2), f_x, f_y, 0, g.Height, angle(k));

                        % sending the number of blobs and datas
                        data = string(data+" "+x_new+" "+y_new+" "+angle_new+"
    "+area(k)+" "+(major(k)*f_y)+" "+(minor(k)*f_y));
                    end

                    cmd = convertStringsToChars(data);
                    write(con_inv, uint8(cmd));
                else
                    % sending the number of blobs is 0
                    cmd = convertStringsToChars(data);
                    write(con_inv, uint8(cmd));
                end

                % new buffer
                buffer = new_img;
            end
        end
    end
end
```

```
function [filt_img] = image_filter(img, threshold, dim1, dim2, gain)
x = img;

    for i=1:1:dim1
```

```
        for j=1:1:dim2
            x(i,j) = x(i,j)*gain;
            if x(i,j) > threshold
                x(i,j) = 0;
            end
        end
    end

filt_img = logical(x);
end
```

```
function [x,y,alpha] = axis_trans(x_pix, y_pix, f_x, f_y, ph_center_x,
    ph_center_y, angle)

    x = (x_pix - ph_center_x)*(f_x);
    y = (y_pix - ph_center_y)*(-f_y);

    temp = rad2deg(angle);

    if (temp >= 0)
        alpha = temp - 90;
    else
        alpha = temp + 90;
    end

end
```

Appendix C

Camera Trigger Schematics

In this Appendix it is discussed how the *Camera Acquire Event* and the actual position of the Conveyor Belt can be properly synchronized.
It is very important to make sure that, when the camera takes an image, the controller records the position of the belt in that exact moment, in order to be able to reconstruct the location of the object with high accuracy.
To do so, it has been used a camera feature called *trigger*, that sends an electrical signal every time that a picture is acquired.

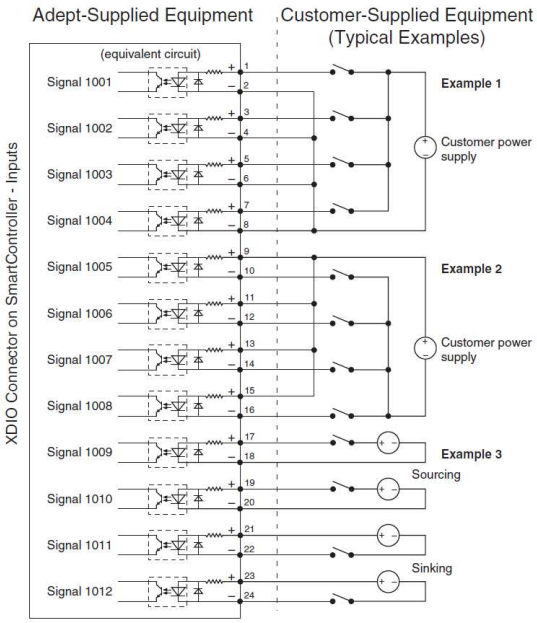


Figure C.1: Input Connectors of the Robot Controller [9]

In Figure C.1 it is shown the input's connection scheme of the Industrial Robot Controller from its manual [9].
It is important to notice that the input signals do not directly enter into the controller, but the signals are sensed by photosensitive transistors.

C.1 Basler's Camera Schematics

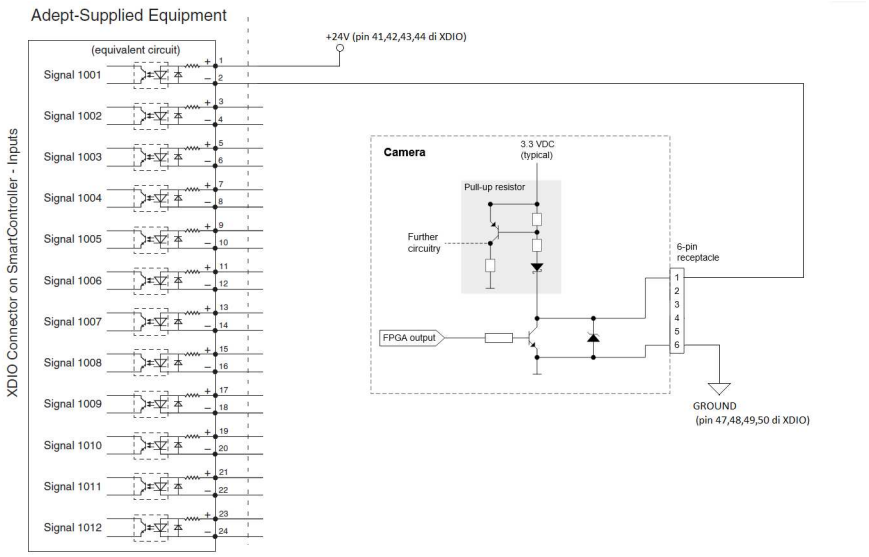


Figure C.2: Example of the Camera Connection to the Controller

The total system, composed by the input of the Industrial Robot Controller and the camera, can be seen as a switch. Looking at Figure C.2 it is possible to notice that the action of the camera trigger is used to close the circuit to the *ground*. This means that, when an image is acquired, the circuit is closed and the input of the controller detects a voltage. This activation of the input is used from the controller to record the current position of the belt.

C.2 DALSA's Camera Schematics

With the table of Figure C.3 it is possible to extract the correct line of the electric signal used for the controller's *latch*.
In particular, it has been chosen the *Line 3* for the connection of the *trigger*, by

Pin Number	Linea GigE	Direction	Definition
1	Line 1+	In	RS-422/Single ended Input Port 1+
2	Line 1-	In	RS-422 Input Port 1-
3	Line 2+	In	RS-422/Single ended Input Port 2+
4	Line 2-	In	RS-422 Input Port 2-
5	Signals Ground		Signals Ground
6	Line 3+	In/Out	RS-422/Single ended Input *or* Output Port 3+
7	Line 3-	In/Out	RS-422 Input *or* Output Port 3-
8	Line 4+	In/Out	RS-422/Single ended Input *or* Output Port 4+
9	Line 4-	In/Out	RS-422 Input *or* Output Port 4-
10	PWR-GND		Camera Power Ground
11	Line 5+	Out	RS-422/Single ended Output Port 5+
12	Line 5-	Out	RS-422 Output Port 5-
13	Line 6+	Out	RS-422/Single ended Output Port 6+
14	Line 6-	Out	RS-422 Output Port 6-
15	PWR-VCC	-	Camera Power – DC +12 to +24 Volts

Figure C.3: Camera HD15 Connector Pin Map

setting this feature with the proper *Matlab* function.

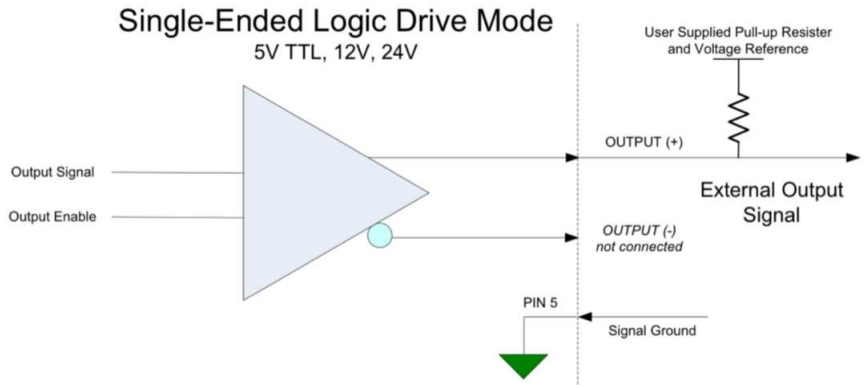


Figure C.4: Camera Output Connection Scheme

In Figure C.4 is shown the connection scheme of the camera output. In particular, since the signal used for the activation of the controller's *latch* works with *24 Volts*, it has been necessary to implement the *Open Collector* output mode, due to the fact that the camera can not manage such high voltages. With this method the camera is able to close the circuit to ground when a new photo is taken.

Appendix D

Belt Encoder Schematics

In this Appendix is illustrated the interconnection relative to the belt encoder signal. This schematics are useful for the utilization of the *Linear Camera*, since it is necessary to synchronize the camera with the belt encoder, to have the best behaviour and precision for the Conveyor Tracking scope.

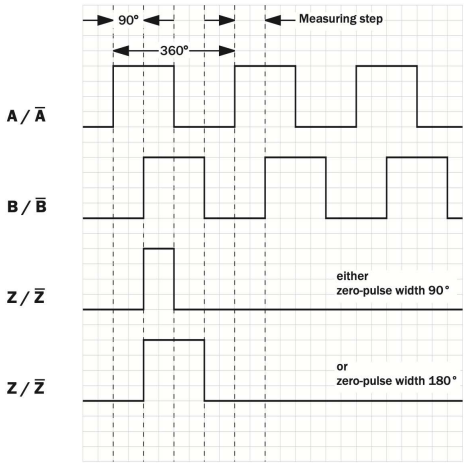


Figure D.1: Encoder's Signals Diagram [10]

Firstly, it is useful to look at Figure D.1, in order to understand the behaviour of the encoder signals during the system operation. This type of encoder presents two signals A and B, which have a constant phase difference of 90°. With this definition of the signals it is possible to understand in which direction the encoder is rotating.

It can also be noticed that this model of encoder presents an additional signal Z,

which allows to know when the encoder is orientated in a certain position [10].

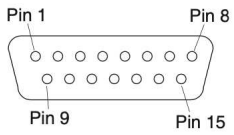
PIN and wire allocation/cable 11-core

PIN	Signal	Wire colour (Cable outlet)	Explanation
1	$\overline{B}$	black	Signal line
2	Sense +	grey	Connected internally to U_s
3	Z	lilac	Signal line
4	$\overline{Z}$	yellow	Signal line
5	A	white	Signal line
6	$\overline{A}$	brown	Signal line
7	N. C.	orange	Not connected
8	B	pink	Signal line
9	Screen		Housing potential
10	GND	blue	Zero volt connected to the encoder
11	Sense –	green	Connected internally to GND
12	U_s	red	Supply voltage [1)]

Figure D.2: Encoder's Pin Table [10]

In Figure D.2 is shown the total table of all the encoder pin signals that are required for the correct behaviour of the instrument.

Channel 1		Channel 2	
Signal	**Pin**	**Signal**	**Pin**
A+	15	A+	11
A–	7	A–	3
B+	14	B+	10
B–	6	B–	2
I+	13	I+	9
I–	5	I–	1
Encoder 5 V out	4	Encoder 5 V out	4
Encoder ground	12	Encoder ground	12
not used	8	not used	8



Belt Encoder Connector Pinout

Figure D.3: Belt Encoder Connector of the Robot Controller [9]

In Figure D.3 is possible to observe the scheme of the *Belt Encoder Connector* of the Industrial Robot Controller. In the purposed solution is used only one encoder, then the discussion will be focused on the *Channel 1* of the connector. The couples $A+/A$- and $B+/B$- are respectively the positive and negative pins of

the two channels of the encoder, the couple *I+/I-* are the positive and negative of the reference signal and finally there are the *Encoder 5V out* which supplies the voltage and the *Encoder ground*.

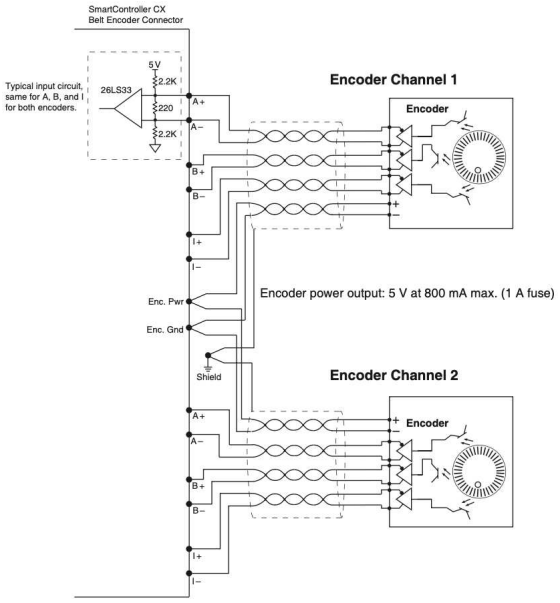


Figure D.4: Example of the Encoder Connection to the Controller [9]

In Figure D.4 is shown the connection of the two channels of the encoders. It can be notice that the encoder is directly connected to the controller, then is is necessary to split the cables, in order to send the signals also to the Linear Camera.

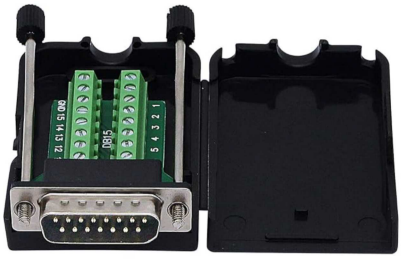


Figure D.5: Encoder Wire Splitter

To double the wires of the encoder can be used a configurable connector like the one in Figure D.5. Using the map of the pins of Figure D.3 is possible to extract the useful signals (the two channels A and B) from the encoder cable and connect them to the camera.

Bibliography

[1] F. Anton, T. Borangiu and others, "An evaluation of pick on the fly methods for high-speed part processing in low cost digital manufacturing," *Springer Nature Switzerland AG*, 2022.

[2] R. Carli, *Robotics and Control 1 Course*. University of Padova, 2022.

[3] Omron Adept Technologies, Inc., *Automation Control Environment User's Guide*, 2016.

[4] Matlab MathWorks. Computer Vision Toolbox Blob Analysis. [Online]. Available: https://it.mathworks.com/help/vision/ref/vision.blobanalysis-system-object.html

[5] Omron Adept Technologies, Inc., *eV+ Language Reference Guide*, 2016.

[6] Matlab MathWorks. TCP/IP Communication Client. [Online]. Available: https://it.mathworks.com/help/matlab/ref/tcpclient.html

[7] Omron Adept Technologies, Inc., *Omron Adept Quattro s650H Robot User's Guide*.

[8] G. Boschetti, *Industrial Robotics Course*. University of Padova, 2022.

[9] Omron Adept Technologies, Inc., *Adept SmartController User's Guide*, June 2009.

[10] CoreTech by Sick Segmann, *DRS60 / DRS61 Data Sheet*, February 2008.