


UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Università degli studi di Padova
Department of Information Engineering

Master Degree in

Automation Engineering

# Design of a workcell incorporating a gripper-based SCARA robot

Advisors
Prof. Luca Schenato
Dr. Richard Kavanagh

Author
Daniele Fadanelli

Anno Accademico 2015/2016

*To my grandparents,*
*that would certainly be proud of my achievements.*

**Abstract**

This Master Thesis dealt with the development of an operative gripping system, which has to be mounted on a SCARA manipulator.
The design concerned both electrical and mechanical part, with focus on the choices of each selected component making up the overall system. The final tool is a pneumatic gripper whose opening/closing actions are controlled by a pneumatic valve driven via Arduino and a suitable electric circuit.
To prove the reliability of the system, a camera-based application was developed and therefore implemented through MATLAB software. Image Processing Toolbox allowed to elaborate an initial acquired snapshot (got via a fixed webcam) to obtain a suitable final image that can be used for an interest object detection.
Final application deals with a sort of palletizing operation of eight parallelepipeds, placed on two levels (four objects above other four).

# Acknowledgements

Firstly I feel to thank my irish supervisor, Dr. Richard Kavanagh for his support and kindness and for having allowed me to work on the project that most I liked. I also want to show gratitude to my italian advisor, Dr. Luca Schenato who immediately accepted to follow my project and tried to advise me when possible.

Moreover, I greatly appreciated the help of the UCC's technicians whose technical support allowed me to overcome practical issues faced during this six months period. Thank you so much Michael O'Shea, Timothy Power, Hilary Mansfield, James Griffiths.
I can not forget even Ralph O'Flaherty who, with his sympathy and availability, always trying to make you feel in a good mood.
Finally, I like to thank Haiyang Lee and my labmate Fabiana who, in different ways, kept me company during these six months at UCC Mechatronics Laboratory.
As well as I would like to thank Luca Ferrari and Sean McSweeney for initially having helped me to get to know the laboratory instrumentation.

Many other people deserve my thanks, as far I have always believed that the joys of life are more beautiful if shared.
Massive thanks to Gaia, the person who was been closest to me in recent years and has never failed to lend her support and her love. I hope to share together many other achievements. Thanks to all the people I met during these last five years in Padova, each of them has contributed to make pleasant this demanding path. Thanks to all the longtime friends in Trento, with whom I shared and am enjoying much of my free time. I hope that our friendship can continue much longer.
Last but not least, heartfelt thanks to my parents, Stefano and Nella, who taught me so many thing during this years, supporting my efforts and always urging to achieve than initially planned. Your contribution was essential to complete my studies.

# Contents

# List of Figures

# List of Tables

# Listings

# Chapter 1

# Introduction

Nowadays industrial automation plays a key role in the development of many Companies.
Generally the word *Automation* is associated with the technology that allows human workers to be replaced by machines, not only to perform physical operations, but also for intelligently decide which operations have to be carried out and how to.
An other important advantage concerns the fact that robots are able to act with greater efficiency than men, even ensuring higher productivity and precision.
The robots which substitute for human arms are the robotic manipulators, also known as robotic arms, i.e. a set of rigid bodies interconnected by mechanical joints, which together form a kinematic chain that allows a certain degree of articulation.
There are a many possible configuration for the kinematic chain, each of which represents a different type of manipulator such as:

- cartesian;
- cylindrical;
- spherical;
- SCARA;
- anthropomorphic.

The manipulator ends with a mechanical wrist on which terminal devices are mounted to permit the manipulation of objects or to perform useful operations on workpieces. Those tools are the parts that allow the robotic arm to interact with the environment.

The terminal device or *end-effector* can be classified in two main categories: *Robotics tools* and *Robotic grippers*.
The tools are generally used for specific function as soldering, spray painting, milling and drilling, laser cutting etc.
The robotic grippers for industrial manipulators have the task of grasping objects and allow a movement, a processing or an assembly operation; there are commonly four types of grippers:

1. electrical / servo gripper;
2. pneumatic gripper;
3. suction cups;
4. magnetic gripper.

Each of these presents its own advantages and disadvantages related to the costs and the efficiency.

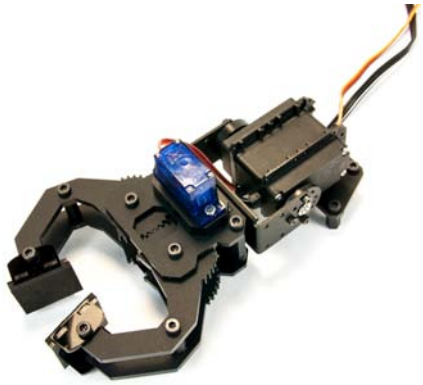


Figure 1.1: Example of gripper

## 1.1 Background studies on work-cell design

Historically, technical systems such as machines have been designed, built and further developed for a particular application. This specialization always led to the problem of limited operating flexibility. Therefore, aim of manufacturers is to integrate electronic controls, new materials and sensors to increase

machinery flexibility, although they are faced with the problem of meeting the requirements for all applications. As a result, complex solutions for various components, which again are special high-tech elements, are demanded in order to meet the requirements of each application as well as possible [1].
However, in industrial field, the process of handling component parts or work-pieces is often underrated as technically simple or even trivial.
From the production point of view, it is obvious that the work-piece itself does not increase in value during the handling process. Since a gripper makes a great contribution to practical success of using an automated and/or robotized solution, a proper design may be of fundamental importance. The design of a gripper must take into account several aspects of the system design, together with the peculiarities of a given application or a multi-task purpose. Strong constraints on the gripping system can be considered for lightness, small dimensions, rigidity, multi-task capability, simplicity and lack of maintenance. These design characteristics can be achieved by considering specific end-effectors or grippers [2, 3].
Moreover, if present, sensors are represented only by elementary devices and specialized for a single task. Depending on the specific work-cell position, special encoders, which monitor the movements of manipulator end-effector, or transducers able to test the compactness of an object, such as contact sensors, and to measure the values relating to interactions of work-pieces with the current application, can be found (for instance, in the case of a pneumatic gripper, it is good practice to monitor the values of applied contact force, the pressure provided to drive a specific force, the current/voltage required to power the robot movements etc.).
The manipulators can be therefore considered as “blind” robots, and they are generally programmed to constantly carry out the same sequence of movements.
In these conditions robotic arms are forced to operate in an specifically built environment.
On the other hand, it is not always easy to produce every time, especially for small and medium-sized companies, suitable robotic configurations, since each component adjustment could lead to an increase of the costs.
For this reason the flexibility of a work-cell can be significantly improved by the use of a suitable sensor that allows the robot to have the perception of its surrounding environment; typical devices are cameras, thereby requiring the study of a proper vision system [4, 5].

## 1.2 Project and Thesis goal descriptions

The Mechatronics Laboratory of the UCC (*University College Cork*) Electrical and Electronic Engineering Department was provided with four *SCARA Sankyo SR8408* robots in September 2013; these manipulators were previously used for product assembly, based on electronic components. This scenario had the aim of giving the opportunity to the engineering students to experiment real industrial problems, allowing them to perform practical research in the field of robotics.
Some work, supervised by Dr. Richard Kavanagh, has already been made; in particular the following tasks were performed

- **Task 1**: Development of an interpreter for controlling the robot through MATLAB;
- **Task 2**: Design of a vacuum gripping system to allow pick and place operations using the end-effector (vacuum based);
- **Task 3**: Implementation of an off-line camera calibration algorithm so that image processing routines can be developed to control the robot/vacuum system;
- **Task 4**: Design of a *Simulink* virtual model of the manipulator, so that its simulated movements can be compared with the ones of the real robot; the model could be used for training tests.

## 1.3 Proposed contribution

To increase the reliability of these robots, a robotic gripping system (pneumatic based) was designed in this Thesis in order to make the manipulator suitable for common manufacturing processes, such as grasping objects, assembling, palletizing etc.
This procedure not only focused on the choice of the mechanical structure of the gripper, but also analysed what were the best circuit solutions and assembly components (choice of actuators, sensors, transducers, etc.) that would provide an operative gripping system.
In particular, the control of the opening/closing actions is carried out by a pneumatic valve, which appositely regulate the air flow able to actuate the gripper tool. An electric circuit, with suitably selected components, makes the control operation working.
Using an Arduino microcontroller board, it was also possible to separately handle the gripping action, so that the movements of the manipulator were

managed by the Sankyo Controller, while the task of grasping an object was run by simple Arduino Sketch code.
The hardware synchronization was basically implemented thanks to a relay circuit which was well adaptable to the output ports of the manipulator controller.
Once designed and tested, the gripping system mounted on the manipulator was used to demonstrate a useful application for the SCARA robot, such as pick and place operation able to autonomously position eight similar parallelepipeds into a pallet configuration (see Figure 1.2 for an illustrative example).
To perform such a task, a simple webcam was mounted on the top of the manipulator and then calibrated. A camera calibration procedure was fundamental to relate a pixel point inside an image with respect to the same position measured (in [mm]) in the workplace by a manipulator encoder. A mathematical approach, exploiting a maximum likelihood estimation of the coefficients linking the two coordinates systems, is performed, proving to be satisfying and improving a previous offline technique able to work only on a small area of the workspace.
Several image processing techniques were then applied to the initial acquired snapshot to properly detect the interest objects to be grasped.
This vision process, implemented via MATLAB, allowed to make available a black and white image, suitable for the extraction of those features which are necessary to distinguish a meaningful object from the background.
The implemented technique cleans also the shapes, allowing to deal with rectangles instead of 3D parallelepiped. This simplification makes the application design much easier, especially with regards to the alignment of the end-effector tool with the object to be grasped.

## 1.4 Thesis outline

The remainder of this Thesis is organized in other six Chapters. All of them are briefly described in the following points.

- **Chapter 2** deals with structure and mathematical analysis of the SCARA manipulator, with a detailed Section describing the features of model installed in the Mechatronics Laboratory, Robot SR840.

- **Chapter 3** introduces the design of gripping system, presenting in detail all the electric and mechanical components used for the development of the operative gripper tool.

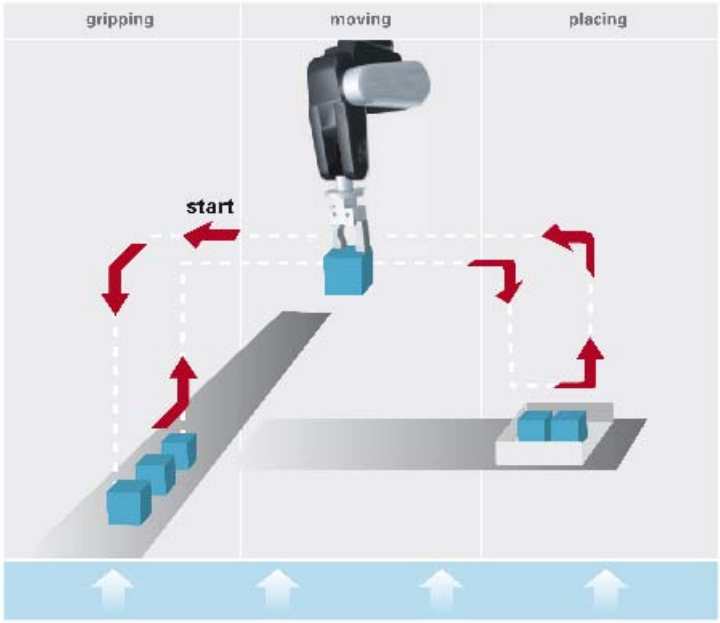



Figure 1.2: Example of industrial application: pick and place

- **Chapter 4** focuses on the overall system control, describing the programming fundamentals used to control the robotic arm and regulate the gripping action. In this section of the Thesis, MATLAB functions, used to manage robot movements, are introduced, as well as Arduino Sketches required to drive and monitor gripper actions (in particular explaining the hardware and software solutions implemented to synchronize the manipulator controller and the Arduino board).

- **Chapter 5** deals with the vision system designed to make the robot autonomous. Techniques and algorithms, provided by MATLAB Toolboxes and required for application purposes, are described by the help of some pictures. In the end, the camera calibration procedure (extremely important for the relation between image informations with robot movements), based on a parameter identification process, is explained from a mathematical point of view.

- **Chapter 6** concerns a vision-based application, developed using both the webcam and the gripping system.

- **Chapter 7** concludes the Thesis with a brief summary of project achievements. Last Section presents possible future developments.

# Chapter 2

# SCARA robot

The SCARA, which stands for "Selective Compliance Assembly Robot Arm", was firstly designed in 1979 by Hiroshi Makino, a professor of Yamanashi University (Japan). Since that time, it became one of the most used robotic arms in the world.
Its success was due to many factors, such as precision, compact dimension, simple structure, small backlash between the components and easy assembling.
This robotic arm is well suited for factory assembly lines, mainly due to its capacity of performing not only movements on an horizontal plane, but also onto the vertical one, allowing, for example, to grasp a part and move it into another position of the workspace.
It can be thereby used in many typical industrial application as pick and place, palletizing, assembling and packaging.


Figure 2.1: Example of SCARA robot

## 2.1 Mechanical structure

Despite the differences introduced by each company that produces this robot, the main mechanical structure remain almost the same (see Figure 2.2).

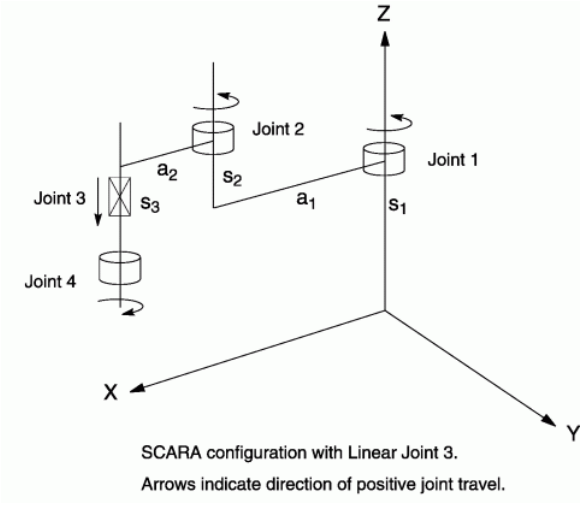



Figure 2.2: Basic mechanical structure of the manipulator

It has 4 degrees of freedom (DOFs), provided by two links and four axes. In practice, this is guaranteed by two parallel revolute joints and one prismatic joint, which allow the motion into the X-Y-Z space (3 DOFs), and by the rotational motion of the end-effector along the vertical axes (1 DOFs).
The manipulator is well fixed by an heavy base. See Figure 2.3 for the structure of SCARA installed in the Mechatronics Lab.

## 2.2 Mathematical analysis: Kinematics

Kinematics is the branch of mechanics that studies the motion of a body or a system of bodies without consideration given to its mass or the forces acting on it.
Even if its study is not essential for the basic operation of the arm, it represents a fundamental mathematical tool to solve problems such as obstacle avoidance or finding the orientation of mounted vision systems with respect to an object.

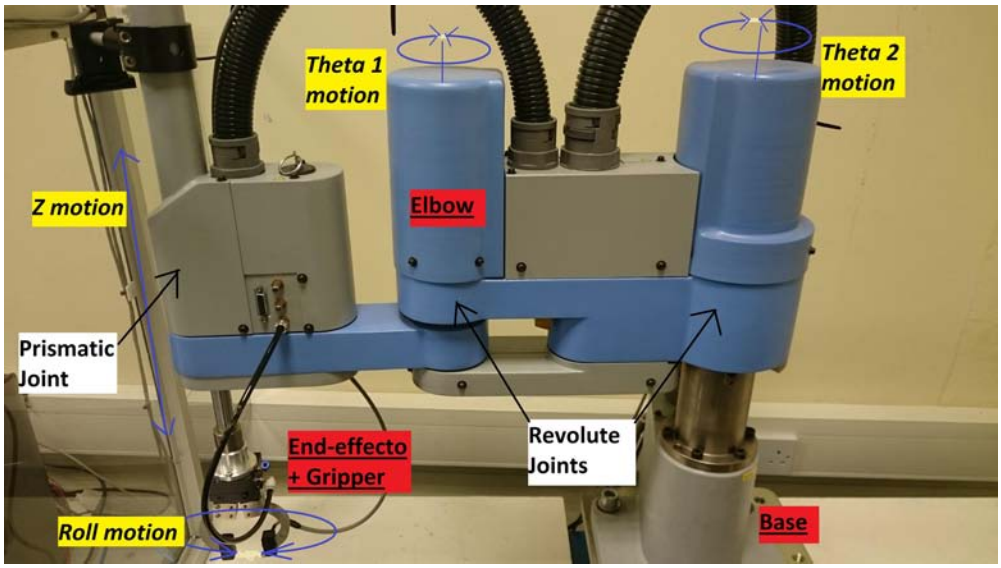



Figure 2.3: SCARA Sankyo SR8408

### 2.2.1 Direct Kinematics

The main goal of direct kinematics it to calculate the position and the orientation of the end-effector, i.e. the pose, with the respect to the main frame, commonly labeled as based frame, knowing the joint variables of the manipulator [6, 7].
Assuming that the joint axis for a revolute joint and for a prismatic joint are respectively around the rotational axis and along the positive direction of motion, it is possible to introduce four main parameters which describe each link of the manipulator; they are shown in the example of Figure 2.4 and described below:

- $a_i$: common normal distance between joint axes, measured from the axis of the current joint $i$ to the axis of the following joint $i+1$;
- $\alpha$: angle by which the axis of the current joint $i$ must be twisted to bring it into alignment with the axis of the following joint $i+1$ when looking along $a_i$. It is commonly assumed that the sign of the angle corresponds to the clockwise positive;
- $d_i$: distance between the two normals $a_{i-1}$ and $a_i$ measured along the

joint axis from $a_{i-1}$ to $a_i$;

- $\Theta_i$: joint angle from $a_{i-1}$ to $a_i$ in the plane normal to the joint axis.

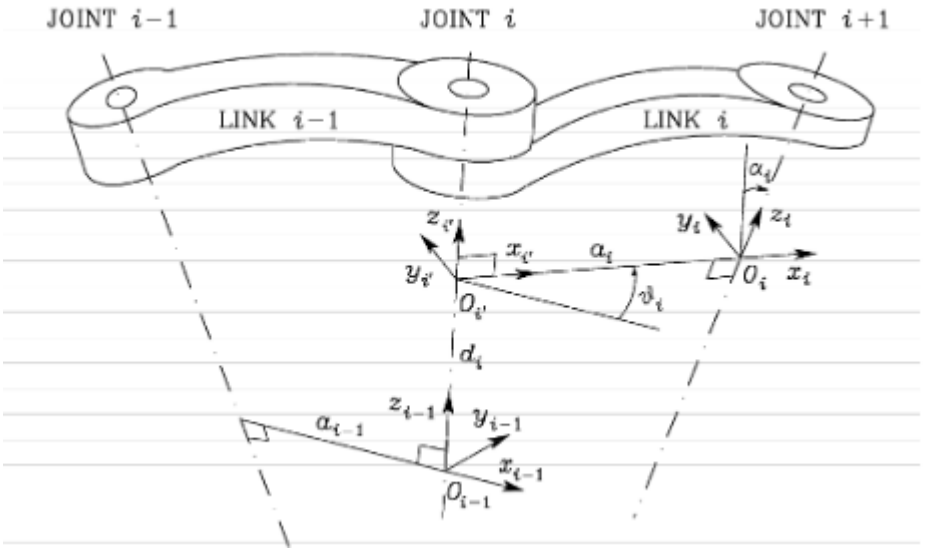



Figure 2.4: Scheme showing *Kinematics Parameters*

These parameters can be calculated only after having defined a reference frame for each link of the mechanical structure; one of the most common method is represented by the Denavit-Hartenberg (DH) convention.
Hence, in order to get the $i$-th frame link of Figure 2.4 the following steps have to be performed [6].

- Choose axis $z_i$ along the axis of the Joint $i+1$;
- Locate the origin $O_i$ at the intersection of axis $z_i$ with the common normal to axes $z_{i-1}$ and $z_i$. In addition, locate the $O_{i'}$ at the intersection of the common normal with axis $z_{i-1}$;
- Choose axis $x_i$ along the common normal to axes $z_{i-1}$ and $z_i$ with the direction from Joint $i$ to Joint $i+1$;
- Choose axis $y_i$ so as to complete a right-handed frame.

Once the link frames are determined, the kinematics parameters can be calculated for each link.
This procedure allows an easy way to find the mathematical transformation

between successive frames, which now is represented by a 4 matrix having the following standard structure:

$$A_i^{i+1} = \begin{bmatrix} \cos\Theta_i & -\sin\Theta_i\cos\alpha_i & \sin\Theta_i\sin\alpha_i & a_i\cos\Theta_i \\ \sin\Theta_i & \cos\Theta_i\cos\alpha_i & -\cos\Theta_i\sin\alpha_i & a_i\sin\Theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

After the calculation of all the matrices $A_i^{i+1}$, it is easy to find the final mathematical expression that links the pose of the end effector with the respect to a base frame, performing two simple steps:

1. calculation of the pose of the last $n$-th link with the respect to the first one $T_n^0 = A_1^0 \cdots A_n^{n-1}$;
2. calculation of the pose of the end-effector $e$ with the respect to the base frame $b$: $T_e^b = T_0^b T_n^0 T_e^n$.

The frame placement, for the SCARA robot used in this project, is showed in Figure 2.5.

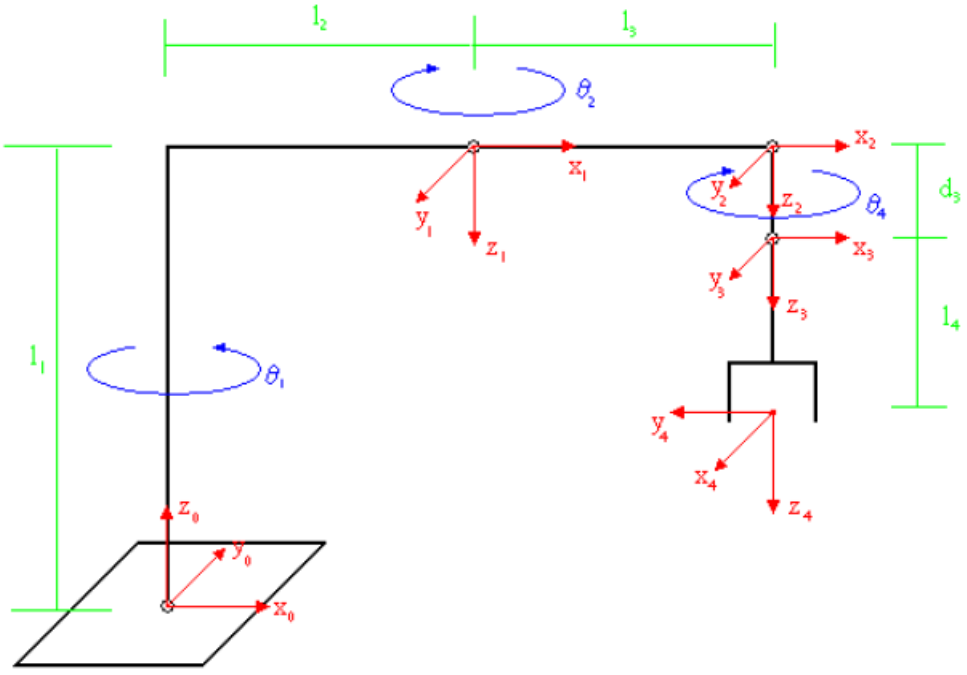



Figure 2.5: Link frames placement for a SCARA type manipulator

Due to SCARA structure, only four parameters are not fixed, i.e. $\Theta_1$, $\Theta_2$, $d_3$

and $\Theta_4$.
Following the rules explained previously, these values were calculated and carried into Table 2.1:

Table 2.1: Link and joint parameters

| Link | $a_i$ | $\alpha_i$ | $d_i$ | $\Theta_i$ | Home |
|---|---|---|---|---|---|
| 1 | $l_2$ | 180° | $l_1$ | $\Theta_1$ | 0° |
| 2 | $l_3$ | 0° | 0 | $\Theta_2$ | 0° |
| 3 | 0 | 0° | $d_3$ | 0° | $d_{max}$ |
| 4 | 0 | 0° | $l_4$ | $\Theta_4$ | 90° |

The substitution of these values into the expression (2.1) and further calculations lead to the matrix linking the end-effector frame to the base frame:

$$T_b^e = \begin{bmatrix} \cos\Theta_{1-2-4} & \sin\Theta_{1-2-4} & 0 & l_2\cos\Theta_1 + l_3\cos\Theta_{1-2} \\ \sin\Theta_{1-2-4} & -\cos\Theta_{1-2-4} & 0 & l_2\sin\Theta_1 + l_3\sin\Theta_{1-2} \\ 0 & 0 & -1 & l_1 - d_3 - l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

where

$$\Theta_{1-2-4} = \Theta_1 - \Theta_2 - \Theta_4 \tag{2.3}$$

$$\Theta_{1-2} = \Theta_1 - \Theta_2. \tag{2.4}$$

### 2.2.2 Inverse Kinematics

The purpose of Inverse Kinematics is the determination of the joint variables corresponding to a given end-effector pose, i.e. position and orientation. The solution to this problem is really important in common applications since transforms the motion specifications, assigned to the end-effector, into the corresponding joint space motions that allow execution of the desired action [6].
For a SCARA manipulator means finding the right values for parameters $\Theta_1$, $\Theta_2$, $d_3$ and $\Theta_4$.
The relation between the end-effector and the manipulator base is described by the matrix:

$$T_{base}^{tool} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & p_x \\ R_{21} & R_{22} & R_{23} & p_y \\ R_{31} & R_{32} & R_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.5}$$

. Observing Figure 2.6, it easy to obtain the following expressions:

$$\begin{aligned} p_x &= l_2 \cos\Theta_1 + l_3 \cos(\Theta_1 + \Theta_2) \\ p_y &= l_2 \sin\Theta_1 + l_3 \sin(\Theta_1 + \Theta_2) \end{aligned} \tag{2.6}$$

Let's also define the orientation of the end-effector $\phi$ as:

$$\phi = \theta_1 + \theta_2 + \Theta_4 \tag{2.7}$$

Looking Figure 2.5, the first unknown parameter $d_3$ results:

$$d_3 = l_1 - l_4 - p_z \tag{2.8}$$

, where the negative sign of $p_z$ is due to the fact that the end-effector has always a straight down approach.
By straightforward trigonometric expressions, $\Theta_2$ can be easily calculated with respect to Figure 2.6:

$$p_x^2 + p_y^2 = l_2^2 + l_3^2 + 2l_2 l_3 \cos\Theta_2 \tag{2.9}$$

Since the position of the end-effector is well known, i.e. $p_x$ and $p_y$, $\Theta_2$ is given by:

$$\Theta_2 = \pm \arccos\left(\frac{p_x^2 + p_y^2 - l_2^2 - l_3^2}{2l_2 l_3}\right). \tag{2.10}$$

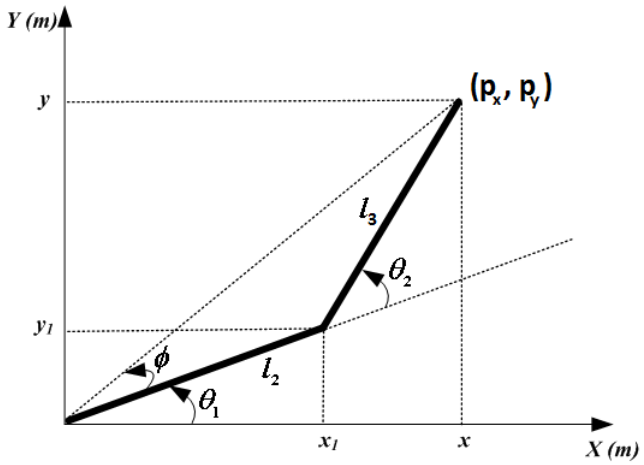



Figure 2.6: Schematic Top view of SCARA robot

The positive and the negative solution corresponds to the possibility to set

left-handed or right-handed mode to the manipulator, respectively.
To find $\Theta_1$ it is necessary to substitute the expression of $\Theta_2$ into 2.6:

$$\begin{aligned} p_x &= l_2 \cos\Theta_1 + l_3 \cos(\Theta_1 + \Theta_2) = (l_2 + l_3 \cos\Theta_2)\cos\Theta_1 + l_3 \sin\Theta_2 \sin\Theta_1 \\ p_y &= l_2 \sin\Theta_1 + l_3 \sin(\Theta_1 + \Theta_2) = (l_2 + l_3 \cos\Theta_2)\sin\Theta_1 - l_3 \sin\Theta_2 \cos\Theta_1 \end{aligned} \tag{2.11}$$

Isolating $\cos\Theta_1$ and $\sin\Theta_1$, it results:

$$\begin{bmatrix} \cos\Theta_1 \\ \sin\Theta_1 \end{bmatrix} = \begin{bmatrix} l_2 + l_3\cos\Theta_2 & l_3\sin\Theta_2 \\ -l_3\sin\Theta_2 & l_2 + l_3\cos\Theta_2 \end{bmatrix}^{-1} \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{2.12}$$

$$\begin{aligned} \Rightarrow \Theta_1 &= \arctan 2( \quad l_3\sin\Theta_2 p_x + (l_2 + l_3\cos\Theta_2)p_y \quad , \quad (l_2 + l_3\cos\Theta_2)p_x - l_3\sin\Theta_2 p_y \quad ) \\ &= atan2(\sin\Theta_1, \cos\Theta_1) \end{aligned} \tag{2.13}$$

In the end, the value of $\Theta_4$ is calculated:

$$\Theta_4 = \phi - \Theta_1 - \Theta_2 \tag{2.14}$$

## 2.3 The Sankyo SR8048

This Section deals with a brief description of the SCARA model used for Thesis purposes, i.e. the NIDEC SANKYO Robot SR8408 (further details are reported in [8]).
Table 2.2 shows the main features of the manipulator.
These parameters are extremely important because able to constrain design choices for any application to be performed by the robot.
The manipulator has got a fixed bas which permits safe and stable movements.
Any change of position is allowed by 4 servo motors, one for every degree of freedom.
$\Theta_1$ and $\Theta_2$ motions are allowed by the two servo motors directly mounted at the top of revolute joints, while the roll motion of the end-effector is possible due to the servo motor placed in the middle of the first arm and to a system of belts and pulley. Through the same mechanism, the Z-axis motion is transmitted up to the prismatic joint by a motor installed in the middle of the second arm.
Electromagnetic brakes are used to regulate the movements.

Table 2.2: Main features of SCARA SR8408

| | | |
|---|---|---|
| Weigth | | 40 [Kg] |
| Max couple | | 3 [Nm] |
| Pay-load | | 3 [Kg] |
| Arm lengths | First arm<br>Second arm<br>Total | 330 [mm]<br>250 [mm]<br>550 [mm] |
| Workspace constraints | $\Theta_1$<br>$\Theta_2$<br>Z axis travel<br>$\Theta_4$ | $\pm$ 120 °<br>$\pm$ 120 °<br>150 [mm]<br>$\pm$ 360 ° |
| Speed limits | Composite speed<br>Z axis<br>Rotational speed | 5000 [mm/s]<br>1000 [mm/s]<br>730 [°/s] |
| Repeatability | X-Y<br>Z<br>Rotation | 0.1 [mm]<br>0.02 [mm]<br>0.05 [°] |

The *Sankyo SR8048* is suitably commanded by the SC3150 Controller produced by the same company, the NIDEC SANKYO Corporation.
The controller has been developed for the purpose of controlling the robot system,equipped with AC servomotors and absolute encoders (ABS encoders) backed-up by battery. Therefore, the operation of returning to home position (which was needed in the case of conventional controllers) is no more required at the time of operation start and/or in the case of servo power down (further details in [9]).

# Chapter 3

# Gripping System Design

This thesis dealt with not only the design and the implementation of a single end-effector tool, but also an entire operational gripping work-cell.
Generally, a robotic work-cell, or simply robotic cell, is defined as a complete system that includes the robot, controller, and other peripherals such as sensors, transducers (used for monitoring the apparatus) and actuators able to feed the manipulator in suitable ways (see Figure 3.1).
The design and the implementation of a gripping work-cell represented a major part of this project.
This chapter introduces all the components (mechanical and electric) that constitutes the experimental apparatus, describing in details every choice made during the development of the system.

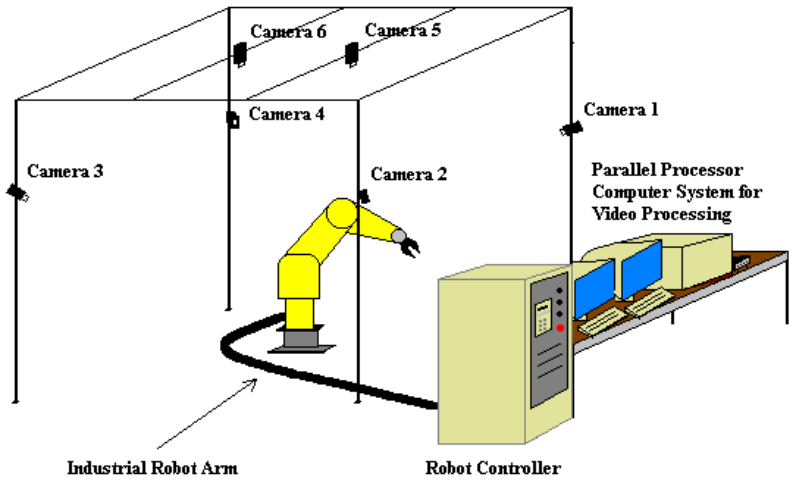



Figure 3.1: Typical layout of robotic work-cell

## 3.1 Gripper Selection

The main part of a gripping system is obviously represented by the gripper device.
As mentioned in Chapter 1, there are several possibilities.
The work environment, the work-pieces (a priori knowledge of the object to be grasped is often suggested), the model of the robot on which the gripper has to be mounted, assembling and costs are all parameter that affect this fundamental choice.
Thanks to the fact that the Mechatronics Laboratory is provided with a compressed air supply system, the implementation of a pneumatic gripper proved to be feasible.
A previous UCC project [10] involved the use of this kind of end-effector, thus, it was decided to adapt the old version of the gripper for the current robot installed in the Lab.
Figure 3.2 shows the mechanical configuration of the gripper used for the experiment.
This device is a fair trade-off between construction costs, project time constraints, capabilities and complexity of the mechanical structure.
It is important to underline the limits imposed by the choice of this simple configuration: the use of this actuator does not allow the grasping of a large workpiece due to the limited distance between the two tips, making the device suitable only for particular applications.
To overcome this constraint, partially mobile tips were inserted, in order to allow the handling of objects having different shapes.
In order to mount the gripper on the robotic arm, it was necessary to adjust the Z-axis shaft, since the previous one was suitable for an older manipulator model; the compressed air supply is provided through a hose which brings the air up to the parallelepiped block placed above the clamps (see Figure 3.3).
The compressed air causes displacement of a circular inner bearing which leads to the closing action; the presence of springs allow also for the correct opening of the gripper, as soon as the air feed is turned off.

## 3.2 Pneumatic feed system

In this section, the pneumatic feed system and each of its individual components are described.

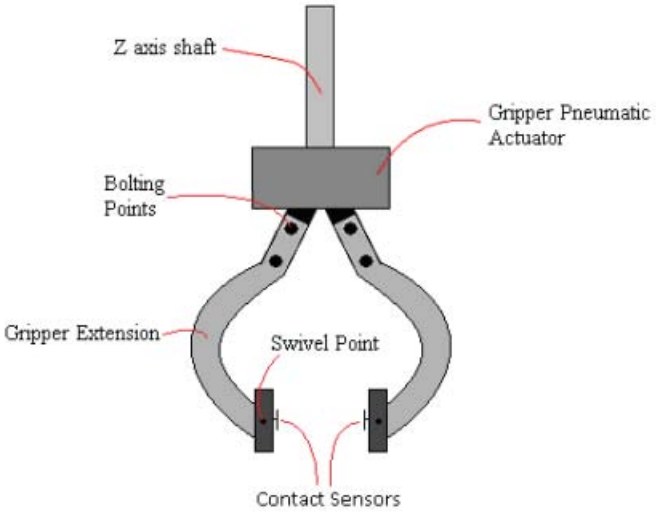



Figure 3.2: Gripper configuration

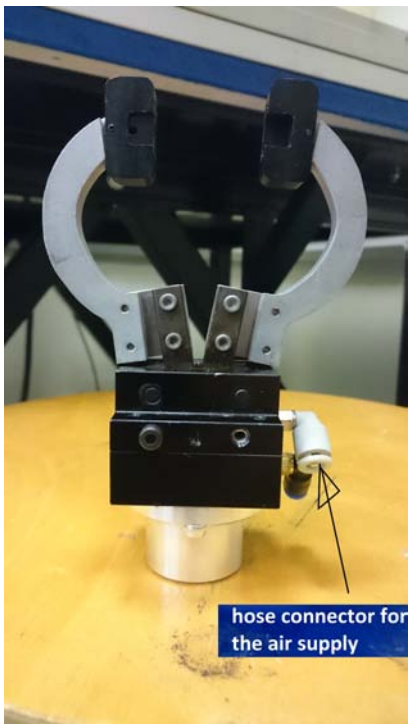



Figure 3.3: Pneumatic gripper used in the project

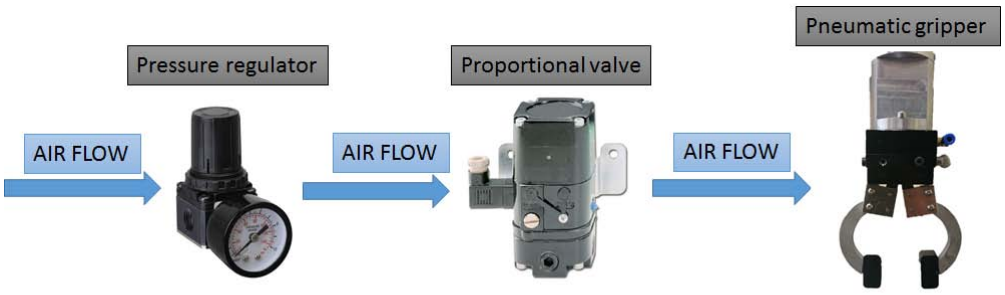



Figure 3.4: Scheme of the pneumatic feed system implemented in the project

### 3.2.1 Adjustable pressure regulator

The compressed air supply is provided by special tubes installed in the Mechatronics Laboratory. The air flow is therefore adjusted via a pressure regulator (shown in Figure 3.5) which allows the setting of the right value of pressure to be supplied to the whole system.
The pressure was kept set to 100 [PSI] (Pounds per Square Inch)[1], equal to about 69 [MPa].

Figure 3.5: Air Flow pressure regulator

### 3.2.2 I/P converter

Figure 3.6 shows the Current to Pressure (I/P) converter used in this project.

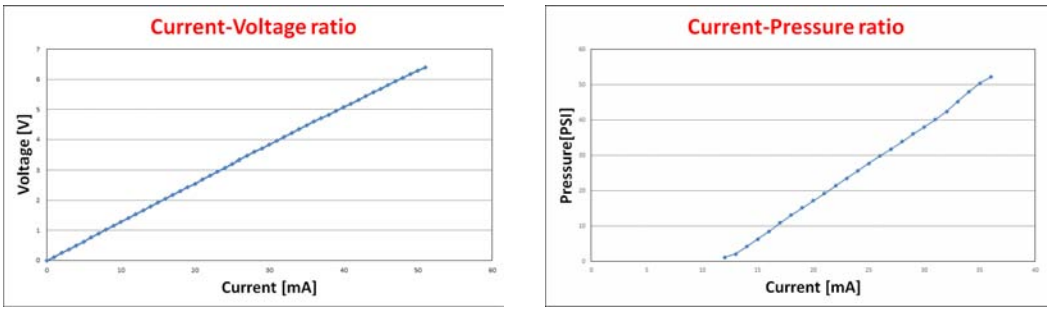
The I/P converter is a force balance device in which a coil is suspended in the field of a magnet by a flexure. Current flowing through the coil generates

[1]This is the pressure unit of measurement in the imperial system of units, *British Weights and Measures Act*


Figure 3.6: I/P converter

axial movement of the coil and flexure.
The flexure moves against the end of a nozzle, and creates a back-pressure in the nozzle by restricting air flow through it. This back-pressure acts as a pilot pressure to an integral booster relay.
Consequently, as the input signal increases (or decreases, for reverse acting), output pressure increases proportionally (it works as a proportional valve).
Due to the advanced age of the device (impossible to find its own datasheet), it was necessary to spend some time finding typical values, such as impedance and sensitivity.
It was rationally thought to model the valve as a series circuit with a resistor and an inductor; however, since the switching frequencies used to stress the device involved only few Hz, it was decided to neglect the second component.
It was therefore possible to estimate a static model for the I/P converter simply providing a voltage input to the I/P inverter and measuring the flowing current through it. By Ohm's Law, this results in a resistance $R$ equal to about 127 $\Omega$ (in Figure 3.7a are shown the data that allowed the estimation).
Finally measuring the output pressure and comparing these values with the input current, an estimated value for the sensitivity $S$ was found.
Concerning this parameter, it was noted that the device responded only to those input signals having amplitude exceeding a predetermined threshold; after this, sensitivity had an almost constant value, i.e. $S \approxeq 2.145[\frac{\text{mA}}{\text{PSI}}]$ (see Figure 3.7b, where is plotted only the linear trend on purpose).
This phenomenon can be attributed to the friction components inside the transduction circuit.

The choice of using an I/P converter, used as a proportional valve (rather than a simple directional solenoid valves), is justified by the ability of changing arbitrarily the compressed air flow. In fact, the output flow from the valve drives the closing action of the gripper: the possibility to appropriately controlling this operation enabled a greater versatility in the operation to be carried out (abrupt closure operations may damage the work-piece to be grasped).




(a) Relation between Current and Voltage (b) Relation between Current and Pression

Figure 3.7

## 3.3 Sensor system

As dealt with in Chapter 1, in order to improve the flexibility and the efficiency of the system, some sensors are generally added to the experimental apparatus. In the following sections the devices used to improve the capabilities of the pneumatic gripper are described, explaining in details the advantages that those bring to the system.

### 3.3.1 Gauge Pressure Transducer

The characteristics and the values in this section can be found in the datasheet [11, 12].

A common approach for monitoring a pneumatic system involves the constant measurement of the air pressure flowing inside the hoses.
Due due its availability inside the Mechatronics Lab, a gauge pressure transducer was mounted on the system.
This device (shown in Figure 3.8) allows measurement of the pressure variations which are detected and therefore converted into an analogue voltage

signal. To do this, the transducer uses a capacitive sensor that translates a physical phenomenon into an electric one.
In this project, the sensor supply was set to a rated voltage of 24 [VDC].
The specification found directly on the front side of the sensor are reported in the Table 3.1.

Table 3.1: Setra 280$E$ specifications

| | max value | min value |
|---|---|---|
| input $[PSIA]$ | 0 | 100 |
| excit $[V]$ | 15 | 32 |
| output $[V]$ | 0.03 | 5.03 |

Figure 3.8: Setra 280E Pressure Transducer

The *Setra 280 E*, i.e. model of project transducer, was mounted just after the exit port of the proportional valve with the goal of having a permanent pressure feedback, useful for monitoring the air flow system, and for providing a feedback control signal.
Since the datasheet was lacking, the sensitivity, which links current and pressure, of the device needed to be obtained via multiple experiments. Some tests provided the linear expected expression (see Figure 3.9 for an example of acquired data):

$$V = K_1 P + K_0 \tag{3.1}$$

, where $V$ represents the output voltage and $P$ the input pressure, while $K_1$ and $K_0$ are the coefficients of the straight line, whose estimated values are

shown below:

$$\begin{cases} \hat{K}_1 = 0.05[\frac{V}{PSIA}] \\ \hat{K}_0 = 0.03[V] \end{cases}$$

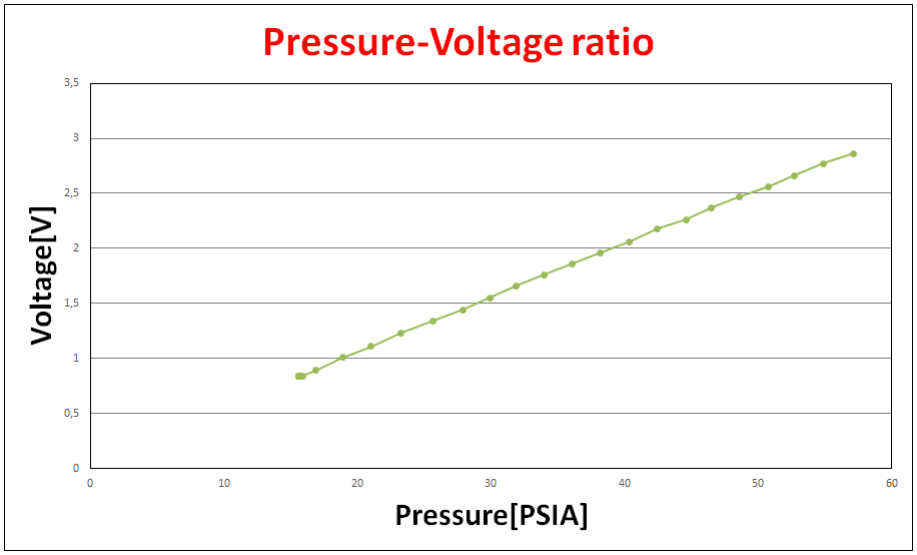



Figure 3.9: Relation between pressure and voltage

The pressure of this device refers to the unit [PSIA] [2] (Pounds per square inch absolute), which caused a suitable pressure unit conversion in the project (either in [PSI] or in [MPa] or [bar] ), when needed.

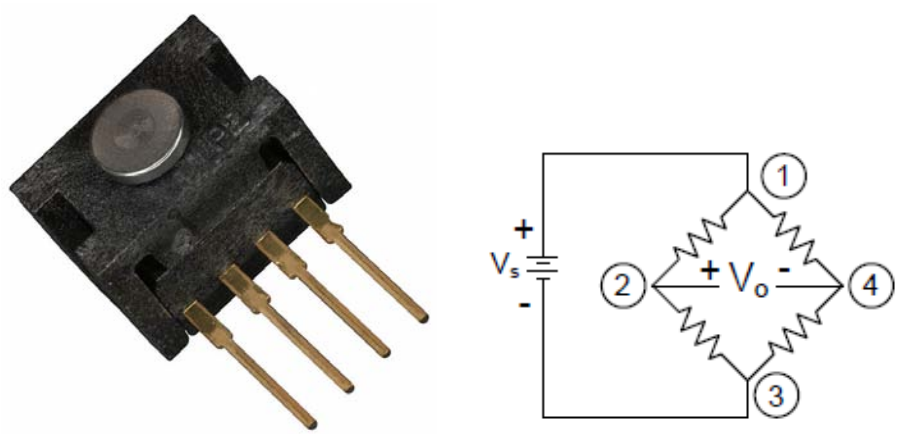
### 3.3.2 Force Sensor

The characteristics and the values in this section can be found in the datasheet [13, 14].

An other important parameter for a pneumatic gripper is the force applied by the clamps of the gripper on the grasped work-piece.
In order to have this important feedback signal for the purposes of contact sensing, it was decided to install sensors just upon the tips of the gripper (see Figure 3.2).

[2] Pounds per square inch absolute (PSIA) is used to make it clear that the pressure is relative to a vacuum rather than the ambient atmospheric pressure. Since atmospheric pressure at sea level is around 14.7 [PSI], this will be added to any pressure reading made in air at sea level

Despite the possibility of mounting a single sensor for each tip, it was decided to use only one force transducer (to minimize cost and to simplify construction), filling the hole present on the other opposite tip: this choice was sufficient for the applications used in this project (despite a greater amount of required force).
The *FSG15N1A* (the device used for the project and shown in Figure 3.10*a*) features a proven sensing technology that utilizes a specialized piezoresistive micro-machined silicon sensing element. The low power, unamplified, noncompensated Wheatstone bridge circuit design provides inherently stable millivolt [mV] outputs over the force range.




(a) FSG15N1A Honeywell Force Sensor (b) Wheatstone bridge sensor configuration

Figure 3.10

The resistance of silicon implanted piezoresistors will increase when the resistor flex under under an applied force. The amount of resistance changes in proportion to the amount of force being applied. This change in circuit resistance results in a corresponding output voltage range level.
For the purpose of the Thesis, the sensor supply was set to a rated voltage of 10 [VDC].
For this sensor, an analysis of the static relation between the applied force and the supplied pressure of the gripper was carried out.
Assuming that the air pressure measured at the output of the valve was reasonably equal to that supplied to the gripper actuator, it was possible to get the relation between pressure and force.
Given the sensitivity of the device

$$S_{force} = 0.24 \frac{[\text{mV}]}{[\text{gforce}]} = 24.473189 \frac{[\text{mV}]}{[\text{N}]} \tag{3.2}$$

the measured force expressed in Newton [N] could be measured at a given time.
Finally, performing grasping test on a suitable work-piece, a linear relation between [MPa][3] and [N] was obtained (see Figure 3.11).

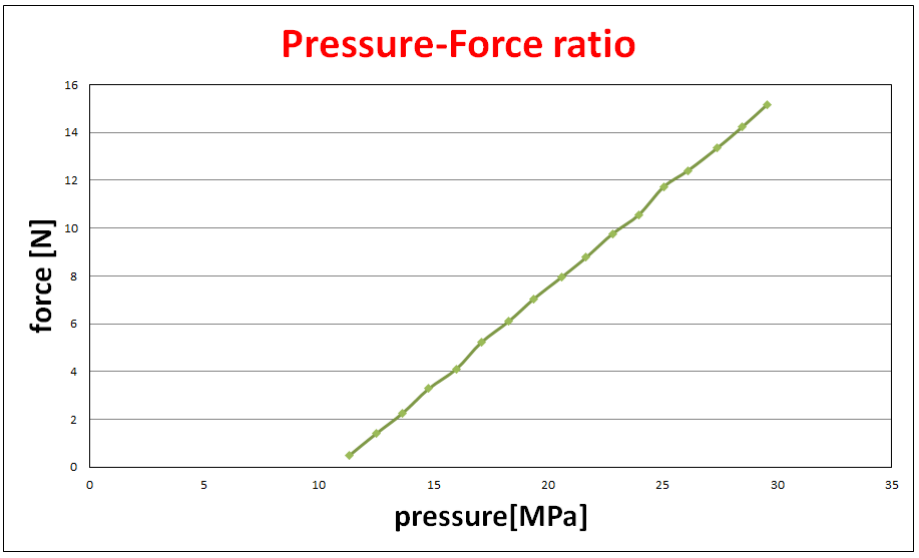



Figure 3.11: Relation between pressure and force

[3]the pascal is the official unit of measurement of pressure for the International System of Units

## 3.4 Gripping action control: use of Arduino

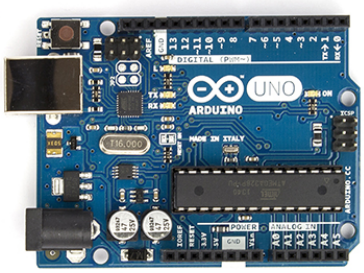



Figure 3.12: Arduino Uno

This section includes informations found in the official website [15].

In order to manage all the electric signals involved in the experiments, an Arduino board was used.
The choice of using Arduino was due to the need of a having a compact system, which could be also easy to control, especially for inexperienced users. Arduino has many other advantages, including:

- Nearly instantaneous start (plug in a USB cord and load an example program require few time);
- Large assortment of included libraries for interfacing to a wide range of hardware;
- Ease of use. The Arduino Uno has built in pin-outs for providing 5 [V], 3.3 [V], ground, analog input, digital output, SPI, I2C, which is useful;
- Lots of available support over the web and offline;
- Free IDE (Integrated development environment) available in all popular PC operating systems, provides an efficient debugger.

In this project, the microcontroller has carried out two main tasks:

**1**) data acquisition system;

**2**) driver of the gripping action.

### 3.4.1 Data acquisition system

Due to the need to acquire analogue signals coming from more sources (input driving current of the valve and output voltages from the pressure and force sensors), it was necessary to provide a data acquisition device to the system, which allowed an easy data collection, making the interfacing with the PC as simple as possible .
Given the impossibility of finding a suitable NI (National Instruments) DAQ (Data AcQuisition device),which represents one of the most common used method [16], it was chosen to employ a simple and efficient solution, i.e. use of Arduino (this project concerned an Arduino Uno version, whose layout is shown in Figure 3.12).
The Uno is a microcontroller board equipped of 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button [15].
Through the analog inputs (maximum input value equal to 5 [V] ), labeled A0 up to A5, each of which provided by an 10 bit Analog to Digital Converter (ADC), i.e. with 1024 different possible measurable values (49 [mV] per unit), and the simple Arduino sketch, shown in 3.1, a series of datas can be easily collected.
The acquired data were then handled and processed by the spreadsheet Microsoft Excel.

Listing 3.1: **Sketch Arduino**: Data acquisition code

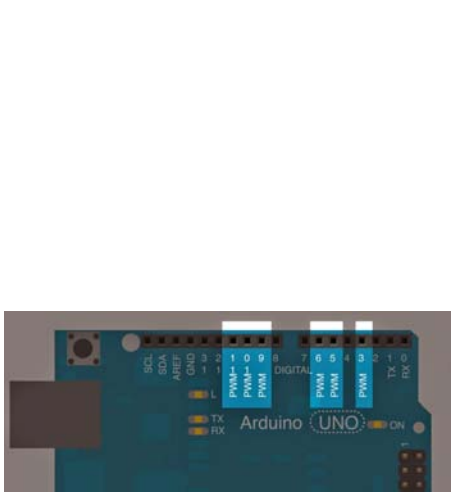
```
1
2  /*
3  ReadAnalogData
4  Reads analog inputs on pin 0,1 and 2;
5  Converts them into voltage;
6  Prints the result on the serial monitor.
7  */
8  // define a variable as an iteration counter
9  int n=0;
10
11 // the setup routine runs once when you press reset:
12 void setup() {
13
14 // initialize serial communication at 9600 bits per second:
15 Serial.begin(9600);
16
17 }
18
19 // the loop routine runs over and over again forever:
```

```
20 void loop() {
21
22 // read the input on analog pin 0:
23 int inputVoltage = analogRead(A0);
24
25 // read the input on analog pin 1:
26 int PressureSensorValue = analogRead(A1);
27
28 // read the input on analog pin 2:
29 int ForceSensorValue = analogRead(A2);
30
31 // Convert the read digital value to analog voltage:
32 // digital range = 0-1023 & voltage range = 0-5 V
33 float supply = inputVoltage * (5.0 / 1023.0);
34 float voltagePressure = PressureSensorValue * (5.0 / 1023.0);
35 float voltageForce = ForceSensorValue * (5.0 / 1023.0);
36
37 // print out the values you read:
38 Serial.print(',');
39 Serial.print(supply,4);
40 Serial.print(',');
41 Serial.print(voltagePressure,4);
42 Serial.print(',');
43 Serial.println(voltageForce,4);
44
45 delay(1);
46 }
47
48 (Note: The value printed in the serial monitor have to be
49        therefore translated into propper unit, i.e. Current,
50        Pressure, Force, by the knowledge of sensitivity)
```

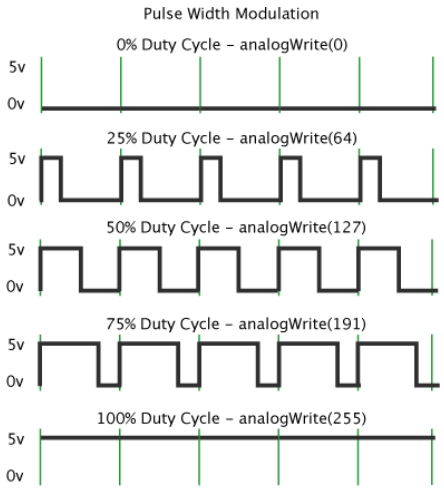
### 3.4.2 Gripping action driver

In order to have an experimental apparatus as compact as possible, it was decided to use the Arduino board even for controlling the system input, which is represented by the proportional valve, i.e. I/P converter.
Unfortunately this version of Arduino is only able to read analog signals, using a 10-bit ADC, and not to generate them as output (remember that the valve has to be fed with analogue current).
Despite this limitation, the board provides an alternative: it lets the user to generate Pulse Width Modulation (PWM) signals as output, which represents a smart technique for getting analog results with digital means.
To realize this task the Uno has specific output ports; in fact, there are 13 digital I/0 pins, six of which can be set to produce a PWM output signal(see Figure 3.13a).

Digital control is used to create a square wave, i.e. a signal switched between ON and OFF. In Arduino, this on-off pattern can simulate voltages between full ON (5 Volts) and OFF (0 Volts) by changing the proportion of time the signal spends ON (whose percentage represent the *duty cycle*) versus the time the signal spends off [15]. The duration of "ON time" is called pulse width, which has to be modulated in order to get various analog values.
The frequency of the PWM on Arduino is based on 8-bit configuration and can vary between 2 set values, 490 [Hz] or 980[Hz].
For this reason the sketch Arduino instruction **analogWrite()** is on a scale of 0-255, such that **analogWrite(255)** request a 100% duty cycle, i.e. 5 [V] as output, and **analogWrite(127)** is a 50% duty cycle, i.e. 2.5 [V] as output (for example, see Figure 3.13*b*).




(a) Digital Arduino PWM pins




(b) Examples of how to generate PWM with Arduino instruction

Figure 3.13

Using Arduino both for acquiring analog measurable values and generating control output signals allowed to get a compact and easy tool for monitoring and handling the gripping action at the same time, using only one device.

## 3.5 Signal Processing and Conditioning

The characteristics and the values in this section can be found in the datasheet [17, 18, 19].

This section of the project describes the circuit connections established to make the whole operating system working, hence allowing each device to have the correct interface inside the experimental apparatus.
Firstly some signal conditioning had to be done in order to adapt the acquired values (introduced in 3.4.1) to the input range of Arduino (0-5 [V]).
Fortunately, the values coming from the pressure transducer had already a proper output range (as shown in Table 3.1), hence the device was able to be connected directly to the analog pin $A1$ of Arduino board.
On the other hand the output voltage from the force sensor, mounted on the gripper tips required the use of an amplifier, since the measured values in millivolts were too small to be read and preprocessed by the 10-bit ADC.
This idea is graphically explained through the electric circuit of Figure 3.14, which exploits a simple differential amplifier, i.e. an op amp along with four resistors.

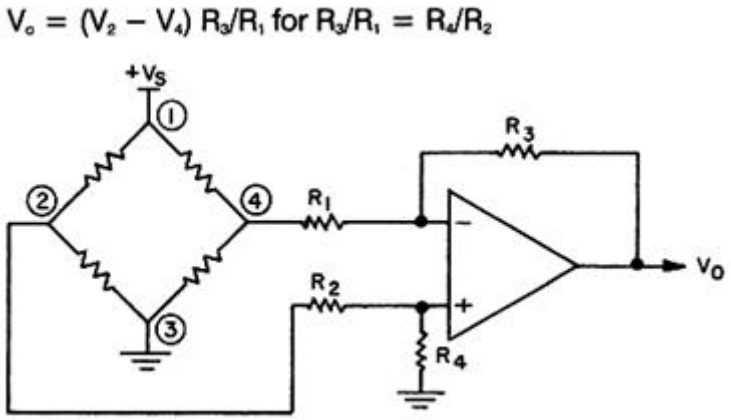
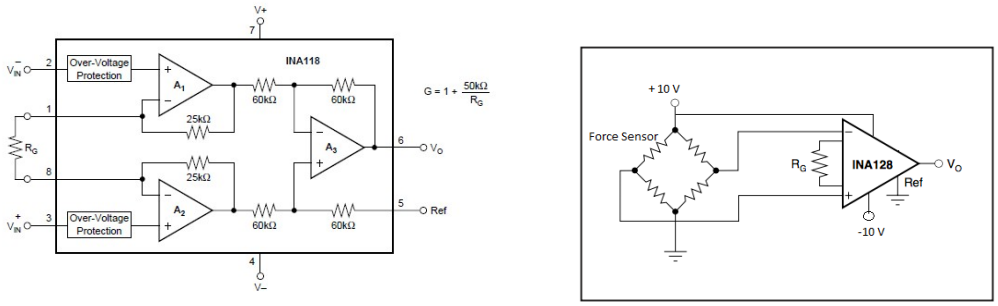



Figure 3.14: Differential amplifier for Wheatstone bridge devices

Besides this easy solution, the amplification circuit could be improved, since the voltage output coming from most sensors, is coarse and prone to common mode voltage as well as noise due to the wiring carrying the voltage from the sensor back to the motherboard, which the simple 741 op amp used in isolation is bad equipped to deal with.
As such, in any real world application the use of instrumentation amplifier is advised in order to offset the null point (i.e. zero voltage reference) and such that the sensor sees infinite input impedance, thus avoiding stray sensor currents. Another benefit, that the use of an instrumentation amplifier gives for a sensor measurement, is its CMRR (Common Mode Rejection Ratio), a

measure of the amplifiers ability to reject common voltage in inputs terminals, which is quite high [20].
The Instrumentation Amplifier chosen in this project was an $INA118P$, whose basic scheme is shown in Figure 3.15a.




(a) INA118P Instrumentatio amplifier (b) Amplification circuit

Figure 3.15

The connection shown in Figure 3.15b allows to amplify at will the differential output of the force sensor (simply changing the value of $R_G$,where the gain expression is given by $G = 1 + \frac{50k\Omega}{R_G}$).
It was chosen a gain resistance $R_G = 5.6K\Omega$ which is the best one that approximated as good as possible a gain of 10; this choice was made in order to constantly have a feedback of what really the force sensor is measuring, even if Arduino is actually reading a value coming from the output of the instrumentation amplifier.
Choosing INA118P, which can can operate with a supply voltage of $\pm 10[V]$, allowed to use efficiently one of the power supply of the Laboratory: with the same setted voltage it was easy to feed the amplifier and the force sensor at the same time, remembering that $V^{+}_{s,Fsensor} = 10[V]$, $V^{-}_{s,Fsensor} = GND$.

As introduced in Section 3.4.2, a PWM output signal was used to control the system, i.e. the air flow passing through the proportional valve.
Some electromechanical devices, as motors, are able to handle a PWM signal, managing easily the quite high PWM switching frequency and the associated analog values, making them suitable for application purpose.
However many cases, like the one in this project, need a real Digital to Analog Converter (DAC) which has to be interposed between the PWM source and the load/device to be driven.
The simple way to design such DAC is assembling a simple low pass filter, which represents a specific configuration of RC-filter, composed of a resistor and a capacitor respectively in series and in parallel with the load.

For this Thesis, suitable values for resistance and capacitance were chosen, advised by the fact that the valve could switch between closing and opening only to low frequency; a 10 [Hz] cutting frequency seemed to be enough for the project purpose:

$$\begin{cases} R &= 15k\Omega \\ C &= 1\mu F \end{cases} \tag{3.3}$$

The driving circuit required two important additional component: a buffer amplifier and a bipolar bipolar junction transistor (BJT).
Given the low impedance of the valve (see Section 3.2.2), the use of a buffer amplifier, designed via an $\mu$A 741 Op-amp, is generally advised to decouple the input (Arduino) and the load (valve); this device integration guaranteed that the output voltage coming from the microcontroller almost exactly fits the one at the buffer output.
The use of the BJT was necessary (used as current gain) to overcome the current feed problem introduced by the buffer: the valve needs a certain amount of current in order to operate, which could not be provided by the buffer (the Op-amp shows only few milliAmps as output current [18]).
A proper connection between a BJT npn 2N3053A and the buffer amplifier showed the expected behaviour, i.e. the right amount of current related to the voltage supplied by Arduino pin.
Figure 3.16 shows an overview of the adjusted driving circuit with all its components.

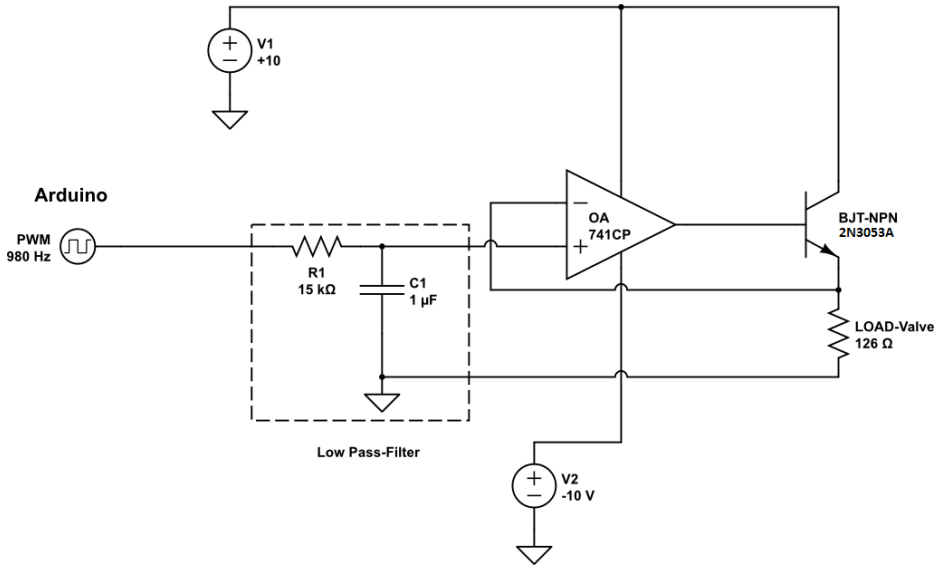



Figure 3.16: Electric scheme solution for the driving circuit

## 3.6 HD Webcam

For characteristics and specifications see [22].

An other important component of the work-cell is represented by a vision sensor, a camera in this project.
In order to realize the vision application (explained in detail in Chapter 6) an HD 720p Webcam was used to acquire, via snapshots, images of the work-pieces placed inside the workspace.
The model used in the project is the *Creative Live! Cam Chat HD* (see Figure 3.17).


Figure 3.17: Creative Live! Cam Chat HD Webcam

As can be see in Figure **??**, the Webcam was mounted, by the use of an aluminium extension, at the top of the the the prismatic joint. The position turned out to be the best for a good view of the worktop.
Moreover, the necessity of keeping the camera well fixed during the movement of the manipulator suggested to install a simple mechanical tool (shown in Figure 3.19) which allowed to maintain fixed the orientation and the position of the camera, but also to let these last parameters be adjustable, if necessary, by a couple of screws.

The communication between the "host" PC and the vision sensor was established through USB cord, which was simply fixed around the structure of the manipulator in order to avoid unexpected obstacles during movements.
The main technical specifications of the device are summarized as following:

- Video resolution: 1280 pixels.
- Picture resolution: 5.7 megapixels.
- Frame rate: Up to 30 fps.
- Fixed focus.

(a) Frontal view of the Camera (b) Side view of the Camera

Figure 3.18: Camera position with respect to the manipulator


Figure 3.19: Mechanical tool used to keep the camera fixed

- Cable length: 1.5 meters.
- USB 2.0 HI-Speed.

Figure 3.20 shows an overview of the system with all its components.

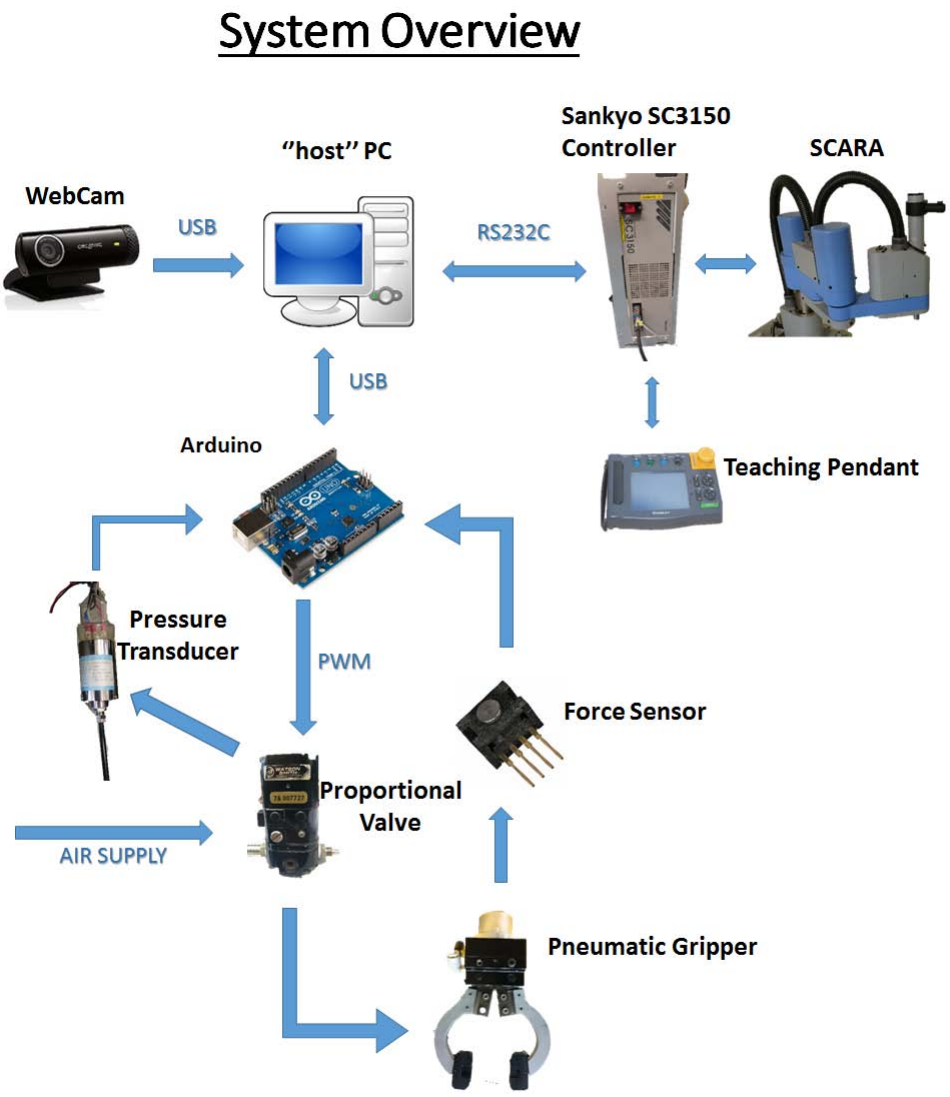



Figure 3.20: Gripper work-cell Overview

# Chapter 4

# Robotic work-cell control

The entire experimental apparatus needed a proper set-up in order to be easily controlled by remote with one of the Lab PCs.
The use of Arduino for handling the gripping action was already introduced in Section 3.4 , while no explanations were made about how to deal with the movements control of the manipulator.
This issue was fortunately already solved in a previous project [23], which allowed a complete control of the SCARA through the simple use of MATLAB software.
Some of the available MATLAB functions were exploited in this Thesis, especially for properly implementing the synchronization between the robot and the gripping system, which is indispensable for any kind of grasping application.

## 4.1 Use of MATLAB for controlling the Robot

This section has the aim of introducing and explaining to the reader the basics of the communication (both hardware and software part) between the manipulator controller and the PC installed in the Laboratory.

### 4.1.1 Software part

The use of MATLAB for the purpose of controlling the manipulator increased the flexibility and the functionality of the SCARA, when combined with the original programming language provided by the robot manufacturer, Sankyo Corporation, called *SSL/E Language* ( Sankyo Structured Language/Enhanced ). This language runs inside a SANKYO Robot Application Development Software named *Buzz2*, that supports the writing, compiling or building,

editing, monitoring and debugging of the user application programs which thereby has to be written inside *Buzz2* and then every time downloaded inside the Controller.
Despite these tools provided by the SANKYO Company, the use of *SSL/E Language* presents some limitations such as:

- limited possibility to operate modular programming;
- weak mathematical tools;
- programming is not straightforward and comfortable due use unfamiliar language;
- hardware limitations in terms of I/0 communications ports.

To overcome these issues, and to allow the user to easily control the robot actions through MATLAB code, an interpretation was necessary in order to translate a MATLAB script into the original *SSL/E* code: this operation was carried out by the *interpreter*, a program written in the robot native language, which is running on the robot controller [23].

### 4.1.2 Hardware part

The Hardware Configuration used for control the manipulator is shown in Figure 4.1.

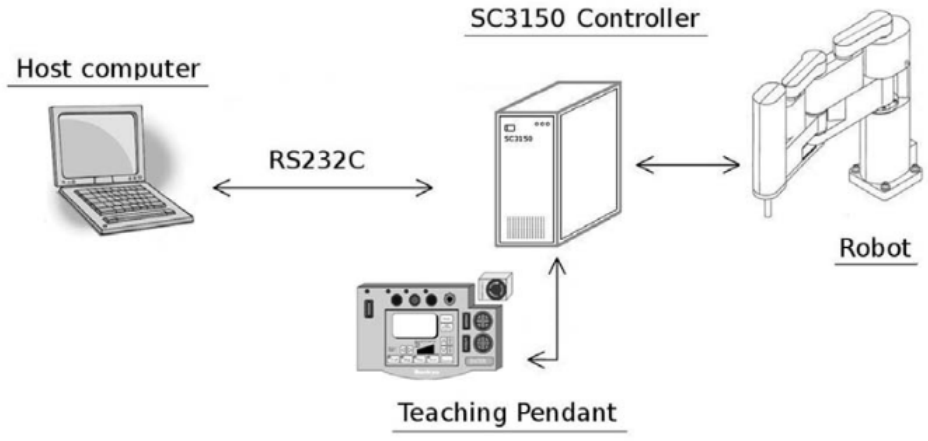



Figure 4.1: Robot control hardware configuration

MATLAB is installed on a PC connected to the SC3150 Controller, through

a RS232 cable (maximum rate: 115.200 [bps]): this communication was previously established by simple MATLAB instruction thanks to the *Matlab Communications System Toolbox*.
The communication arrangement is a standard 9 pin RS-232 connection:

- Data bits=8
- Stop bits=2
- Parity=Even
- Speed=Up to 19200 baud (the controller operates at 9600 baud[4]).

The robot controller is connected to the SCARA in order to provide power to the motors and to Absolute Home Position (ABS) encoders, which are position sensors able to detect the position of the joints, and to receive the most recent encoder position feedback signals from this last ones.
The Teaching Pendant allows many operations, the most important of which is the starting and stopping of the interpreter execution. It is also possible to manually move the manipulator via a joypad.

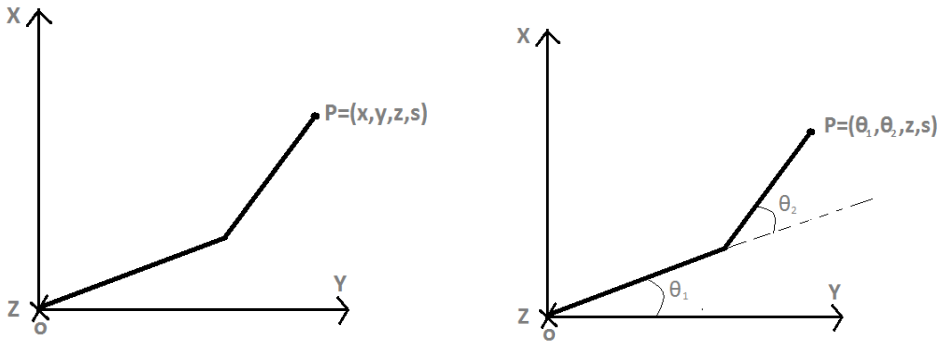
## 4.2 Matlab Robot Functions

For the purpose of this project, only a few of the translated MATLAB robot functions were used: the main ones are described in the following Subsection. For completeness, it is important to underline the possibility of referring to two different coordinate systems: the Cartesian Coordinate System and the Joint Coordinate System (see Figure 4.2 to explain).

In this Thesis only the first one, which describes the position of the end-effector with respect to the Cartesian frame relative to the base of the manipulator, was used for the application purpose described in Chapter 6.
For instance, defining a position $p$=[$x$ $y$ $z$ $s$] in the Cartesian Coordinate System means:

- $x$ represents the $X - axis$ position measured in [mm];
- $y$ represents the $Y - axis$ position measured in [mm];
- $z$ represents the $Z - shaft$ position measured in [mm];
- $s$ represents the $Roll - axis$ position of the the last joint measured in [deg].

[4] *baud* is the unit for symbol rate and represents the number of distinct symbol changes that occur per second




(a) Cartesian Coordinate System (b) Joint Coordinate System

Figure 4.2: SCARA available coordinate systems

### 4.2.1 Motion in the Cartesian Coordinate System: point to point

***MOVE***

This function allows the manipulator to move to a precise point, previously defined.
As **input** it needs a $N$x4 matrix, where $N = 1, ..., 8$; this means that up to 8 different positions can be reached in succession. The **output** value is none.

***SMOVE***

This function allows the manipulator to move only one of the Cartesian Coordinates, i.e. $x, y, z, s$, and to specify the absolute displacement. As **input** it needs two parameter:

- n: number to select the coordinate to change (x=1 or y=2 or z=3 or s=4);
- d: value of the absolute displacement expressed in [mm] or [deg].

The **output** value is none.

### 4.2.2 Speed/acceleration set

***SPEED***

Due to the fact that point to point functions are built to let the robot to operate at the maximum speed, a function that permits to arbitrarily change this parameter is extremely useful.
With this function, which as an integer number as **input** representing the set percentage of the speed limit, all the velocities of each axis are modified at the same time. When not specified the default percentage is 10%. The **output** is the previous set value.

***WEIGHT***

Sometimes is wise to know the payload for a precise application. For this value, it is also common habit to set a precise acceleration and deceleration to the movement of the manipulator. In fact when the *speed* function is used, the imposed value of velocity is not reached immediately, but takes a while. This time could therefore be set in relation with the weight of the workpiece grasped by the gripper.
As **input** it needs a real number between 0 and the maximum payload weight expressed in [Kg].
The **output** value is none.

***AUTOACL***

This function enables or disables the automatic optimum acceleration and deceleration settings for the point to point motion. The automatic acceleration and deceleration settings depend, as previously explained, on the payload handled by the *weight* function.
The **input** parameter can be chosen between:

- 0: Disabling the automatic acceleration and deceleration settings;
- 1: Enabling the automatic acceleration and deceleration settings.

The **output** value is none.

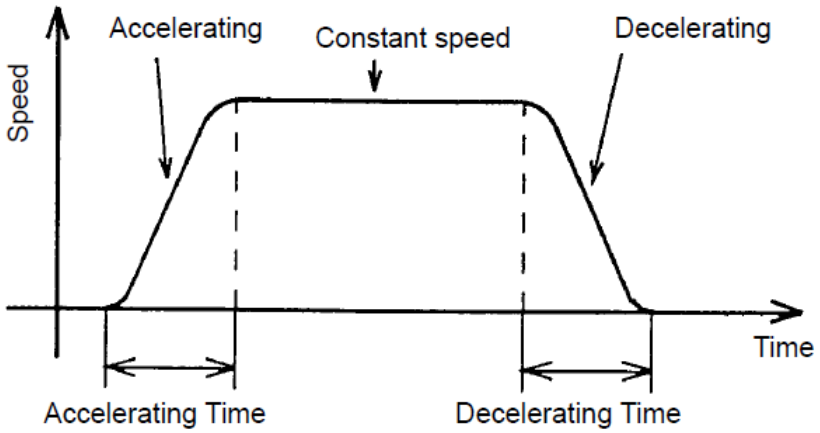



Figure 4.3: Trapezoidal trajectory for Robot movements

### 4.2.3 Mark function

***MARK***

This function is extremely useful, since it provides the current position of the end-effector in the Cartesian Coordinate System. The **input** is none, whereas the **output** is a 1x4 position vector.
Since the values are calculated by reverse-conversion of position pulse (not command pulse), a small error may be introduced between the real position and the one obtained through the use of the function.

### 4.2.4 I/O functions

***OUT***

The aim of this function is simply to enable or disable a digital port of the Robot controller. As **input** it needs two parameter:

- number representing an output port;
- 0(OFF) or 1(ON).

The **output** value is none

### 4.2.5 Palletizing functions

***SETPLT***

This function permits the definition of a pallet configuration. The example, shown in Figure 4.4, allows one to understand the meaning of the **input** parameters to be provided with this function. In order to define a pallet, it is necessary to know its main corner positions and the number of objects that could stand between these set positions. It is also possible, by the use of this function, to label the designed pallet with a number which will prove extremely useful for the next function.
The **output** value is none.

***PLT***

This function calculates the position corresponding to a point number on an user-defined pallet previously designed through use of **SETPLOT**. Thereby as **input** it needs two parameter:

- n: pallet number (positive integer);
- a: point number configuration (positive integer).

The **output** value is none.

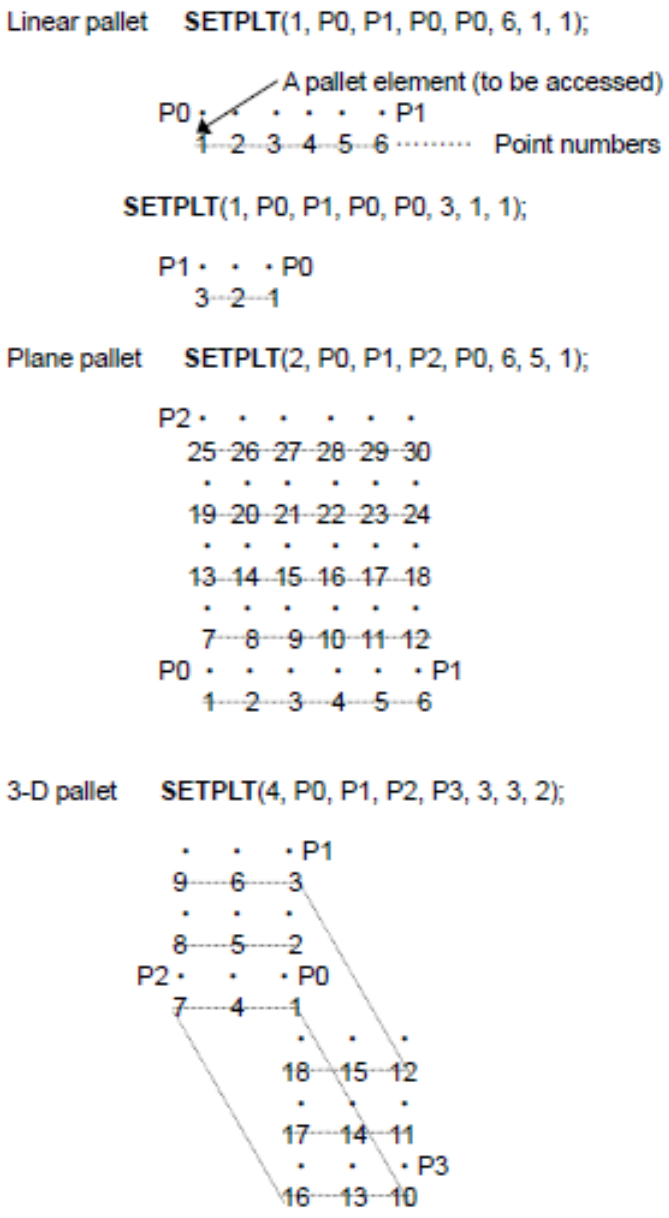



Figure 4.4: Example of pallet definition

## 4.3 Gripping action control

In order to make the system easier to control, it was decided to implement a technique allowing synchronization between the movements of the SCARA (handled by the *SANKYO Controller* via MATLAB) and the gripping action (handled by *Arduino Uno*).

This task required the design of a simple signal communication between the two controllers.
Given the output configuration of the robot controller (shown in Figure 4.5), a relay circuit was included in order to insulate Arduino with the SCARA controller (for safety purposes).

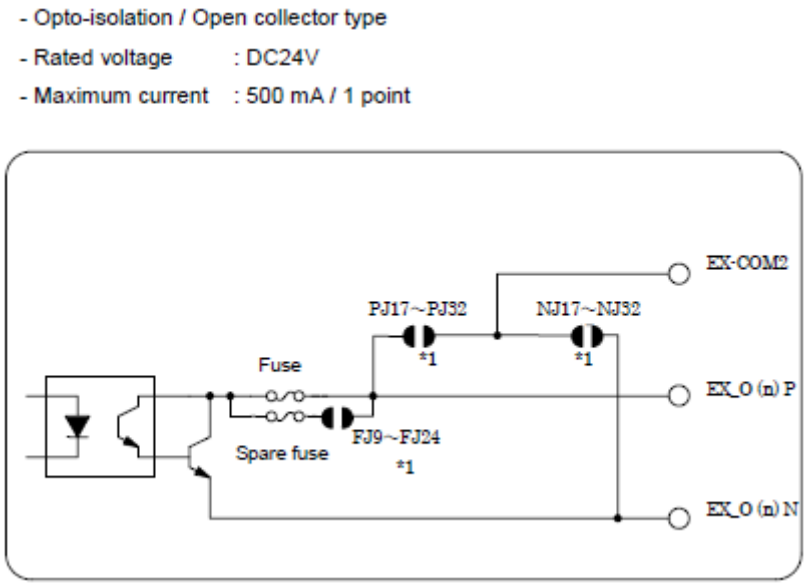



Figure 4.5: Electric configuration of a external output of the SCARA controller

Figure 4.6 shows the simple circuit used to implement the communication process.

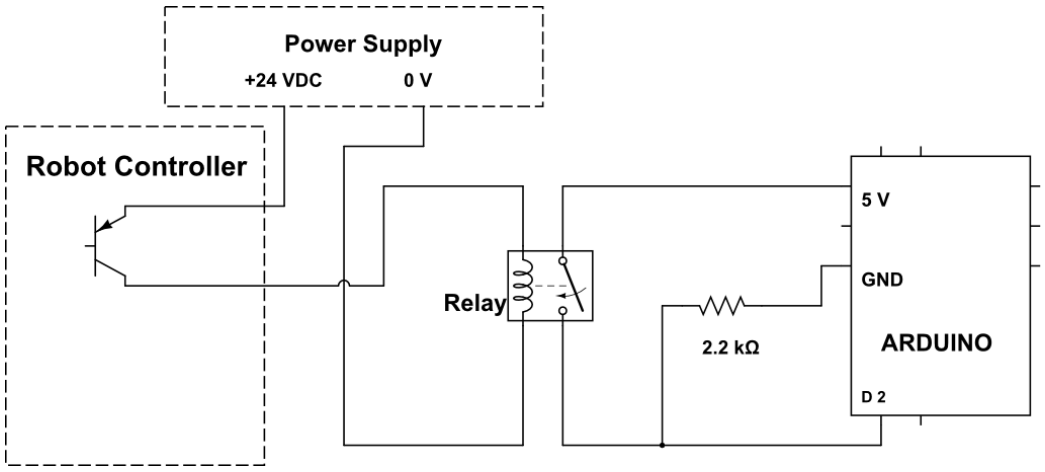



Figure 4.6: Electric circuit for driving the gripper

The MATLAB robot function *out* was used to switch ON and OFF the *BJT* placed inside the output port of the robot controller: this required action permitted the feeding of relay.
Hence, the use of relay allowed, through an `if` statement of Arduino sketch, to arbitrarily turn ON or OFF the gripping action, an operation completely handled by the microcontroller board. This concept is explained in detail in the following Subsection.

### 4.3.1 Communication procedure

When the relay switches from OFF state to ON state, it works as switcher for any other circuit connected to it ( in this case the simple circuit at the right of the relay in Figure 4.6).
Observing the electric configuration of the relay used in this project (shown in Figure 4.7), the procedure just explained above is implemented by applying 24 [VDC] between pin 1 and 8.
To optimize the procedure and thus decrease the number of electric components involved, it was decided to use the power supply of the Arduino (5 V pin) as a driving signal.

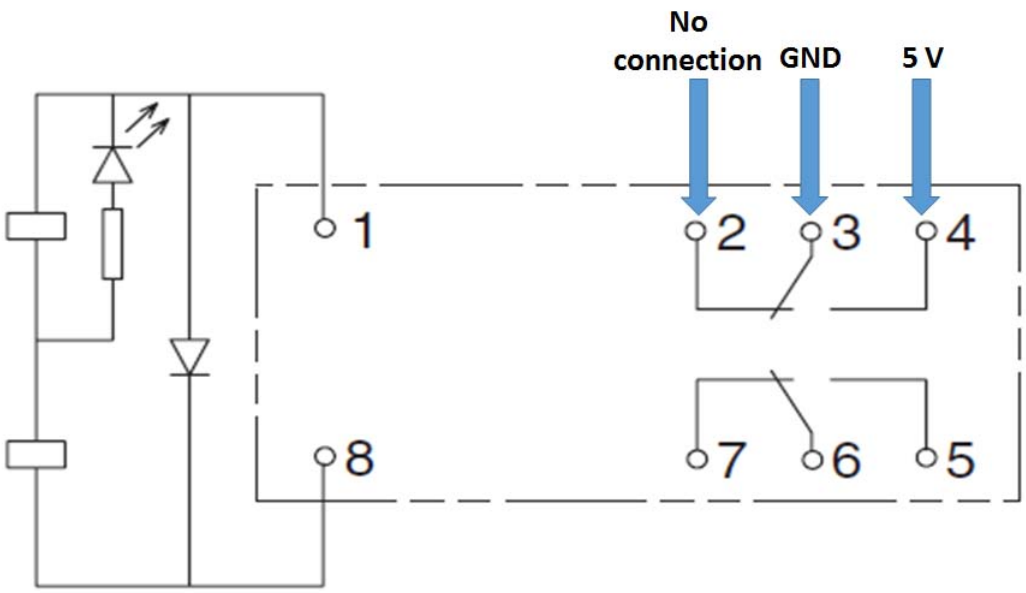



Figure 4.7: Electric configuration of relay: Double Pole Double Throw(DPDT)

One of the Arduino digital pins ($D2$ for the project) was used instead, for

reading sensor: signals above 3 V are considered as HIGH logic value, i.e. 1, while inputs below 1.3 V are associated with LOW logic value, i.e. 0 [15]. This simple circuit solution allowed to parallely manage the two controllers of the main system: when the pin $D2$ reads HIGH (pins 3 and 4 of the relay connected), then Arduino operates the closing action for the gripper through the use of PWM pin (explained in Subsection 3.4.2), while when the read value is LOW (pins 2 and 3 of the relay connected), the microcontroller stops supplying power, causing the opening of the jaws of the gripper (which is spring based).
Table 4.1 summarized the overall method:

Table 4.1: Enabling/Disabling of gripping action

| **MATLAB** | **status relay** | **status D2** | **status gripper** |
|---|---|---|---|
| out(938,0) | OFF | LOW | CLOSING |
| out(938,1) | ON | HIGH | OPENING |

Arduino sketch 4.1 represents an example of code used to run the Arduino control on the gripping action.

Listing 4.1: **Sketch Arduino**: Example of gripping action control performed

```
1
2  /*
3  Gripping action control
4  Use of digital pin 2 & 6;
5  */
6  // Define variable for level signal HIGH and LOW
7  int High=1;
8  int Low=0;
9
10 // Define number of the digital pin used as interrupt operator
11 int interrupt = 2;
12
13 // the setup routine runs once when you press reset:
14 void setup() {
15
16 // initialize serial communication at 9600 bits per second:
17 Serial.begin(9600);
18
19 // set digital pin 2 as input
20 pinMode(interrupt, INPUT);
21 }
```

```
22
23 // the loop routine runs over and over again forever:
24 void loop() {
25
26 // define the if statement use for control the gripping action
27 int switchOperator = digitalRead(interrupt);
28 if (switchOperator == High) {
29
30 analogWrite(6, 180);
31 delay(1);
32 }
33 else {
34
35 analogWrite(6, 0);
36 delay(1);
37 }
```

# Chapter 5

# Vision System

The development and the integration of vision systems inside the industrial automation had a big increase in the last decade, largely due to low component costs and to the presence of image processing algorithms more efficient and robust. [24].
The advantages and the limits of a vision system are various, and mostly depend on the work environment in which they are installed.
Despite this peculiarity, it is generally possible to define the main aspects that distinguish a vision system from a human being, pointing the advantages and disadvantages of their use.
The pros of using a vision system are summarized as follows:

- Repeatability. A human operator is not able to provide a constant performance, while a robot can easily do it. Even if all members of a team-workers collaborate for the same operation purpose, each of them has a way of working which will be slightly different from the others causing possible problems.

- The possibility to operate in hostile work environments. There are many areas inside industrial environment, where an operator can not work under safe conditions.

- Speed of check/efficiency. A vision system is able to perform monitoring operations in fractions of a second, even on objects moving fast, as on conveyor belts.

- Generation and elaboration of process data. During a process could be extremely important the detection and the storing of potential defects in data form, allowing the human operator to apply suitable corrections.

At the same time there are still many technical limitations for application purposes, especially if the system is designed and installed without following well predefined specifications.
Operations that the human brain is able to carry out in fractions of a second require, instead, thousands lines of code and powerful calculators.

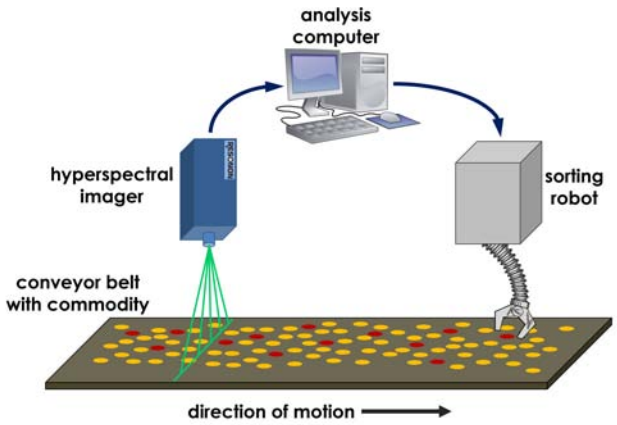



Figure 5.1: typical configuration of a vision-based robotic work-cell

Typical examples are the detection of complex or 3D shapes: each of us is able to recognize hundreds of faces in a fraction of a second or to find a screwdriver in a box of very messy tools. Our eye is also able to adapt to illumination changes, to variations of distance and to keep the interest object permanently inside the field of view, focusing on those important aspects useful for specific purposes.
All these aspect are not obvious for a vision system (see Figure 5.1 for an example).
). Typical examples are the detection of complex or 3D shapes: each of us is able to recognize hundreds of faces in a fraction of a second or to find a screwdriver in a box of very messy tools. Our eye is also able to adapt to illumination changes, to variations of distance and to keep the interest object permanently inside the field of view, focusing on those important aspects useful for specific purposes.
All these aspect are not obvious for a vision system (see Figure **??** for an example).
). Some limitations belong to such a system, especially when has to work with:

- various and different object shapes to be detected;

- work-pieces with complex structures which have to be checked from different point of views;
- illumination variations on the interest field of view;
- dirty and dusty work environment conditions.

The vision system designer's task is to emphasize the advantages resulting from that specific system, in particular arranging the layout of the cell in order to make it as easy as possible the analysis on interest parts and to overcome the issues just introduced.

For application purposes dealt with in Chapter 6, a vision system had to be appositely designed.
The vision sensor used in this project was already introduced in Section 3.6 and it is represented by a simple HD Webcam. Its use regards the detection of interest objects inside its field of view through the acquisition of an image frame. Afterwards, the acquired image is processed in order to store those useful features for application task.

## 5.1 Vision approach through MATLAB

The capture of an image frame and its elaboration are generally known as *Image acquisition* and *Image processing*.
Both of them are quite straightforward to implement thanks to the use of two MATLAB Toolboxes:

- *Image Acquisition Toolbox* [25];
- *Image Processing Toolbox* [26].

### 5.1.1 Image Acquisition

Image acquisition performed in this project is carried out by the use of *Image Acquisition Toolbox*.
Through a simple MATLAB code (see Listing 5.1), it is possible to switch ON the camera, configure the frame acquisition and return the video input object, which could therefore be used to store a snapshot image for the processing application.
The MATLAB function has to be called every time it is necessary to acquire an image for application purposes.

At the end of the Image acquisition process the camera has to be turned OFF by `close_cam` function (see Appendix A).

Listing 5.1: **start camera function**: enabling video storing and previewing

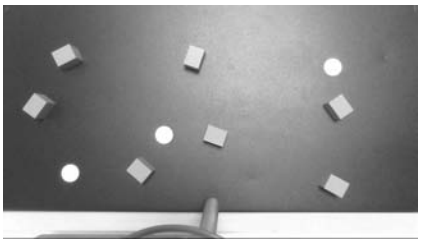
```
1
2  %% this function allows the operator to switch on the camera
3  % and set the parameter for the acquisition of a frame
4
5  function [o] = start_cam( a )
6
7  persistent vid;
8
9  if (strcmp(a,'open')) % if the two strings perfectly match each
10                       % others, it returns TRUE
11
12     pause on;
13
14     % The video input object is saved as variable vid
15     vid =videoinput('winvideo',1);
16
17     % Settings to acquire a single snapshot
18     set(vid, 'FramesPerTrigger', 1);
19
20     set(vid, 'TriggerRepeat', Inf);
21
22     triggerconfig(vid, 'manual');
23
24     start(vid);
25
26     % preview of the video
27     preview(vid);
28
29 else
30
31     if(strcmp(a,'retrieve')) % if a=='retrieve', the vid variable
32                              % is saved in the workspace
33         o=vid
34
35     end
36
37 end
38
39 end
```

### 5.1.2 Image Processing

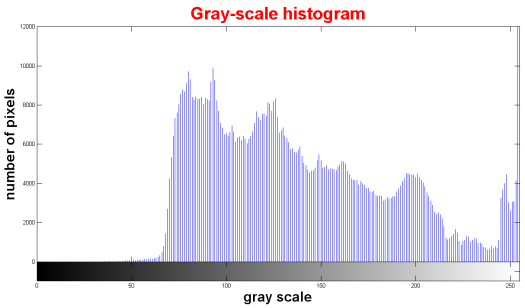
Image processing is a computational process that transforms, via algorithms, one or more input images into an output image.
Image processing is frequently used to enhance an image for human viewing or interpretation ( for instance, to improve contrast).
Alternatively, and of more interest to robotics, it is the foundation for the process of feature extraction, which results essential for an operative vision-based application.
Other possible applications for image processing are:

- Image display and printing;
- Image editing and manipulation;
- Image compression;

*Image Processing Toolbox* offers a big set of algorithms and functions for image processing, analysis, visualization, and code development.
The remaining of this Subsection discusses the main techniques ( used in this project) that provided a suitable image to be used for application purposes.
The first step dealt with the conversion of the Red-Green-Blue (RGB) acquired image into a gray-scale image which results to be more easy to process.
An RGB image, sometimes referred to as a truecolor image, is stored as an m-by-n-by-3 data array that defines red, green, and blue color components for each individual pixel. RGB images do not use a palette. The color of each pixel is determined by the combination of the red, green, and blue intensities stored in each color plane at the pixels location. Graphics file formats store RGB images as 24-bit images, where the red, green, and blue components are 8 bits each. This yields a potential of 16 million colors. The precision with which a real-life image can be replicated has led to the nickname "*truecolor image*" [26].
On the other hand a gray-scale image is a representation where each pixel corresponds to an integer value (stored in a matrix) corresponding to the brightness of the pixel. For instance an 8-bit image has a gray-scale of 256 ($2^8$) possible values, where **0** represents **black pixel** while **255** stands for **white pixel**.
The distribution of gray levels within image can be shown by an histogram which indicates the number of times each gray pixel value occurs.
Figure 5.2 shows an example of histogram.



(a) Gray-scale acquired image




(b) Gray-scale histogram

Figure 5.2

After the conversion into a gray-scale image, it was possible to proceed with the *image segmentation* process, which represents the operation by which items in an image are separated from each other and from the background, making them meaningful regions.
This is one of the earliest approaches to scene understanding and while conceptually straightforward, it is a very challenging problem.
A key requirement is robustness which evaluates how gradually the method degrades as soon as specific assumptions are not followed, for example changing scene illumination or viewpoint.
Generally the **first step** of *image segmentation* involves the use of a **classification** method which most of the time presents itself as binary classification ($c \in C = \{0, 1\}$): the pixels have been classified as object (c=1) or not-object (c=0) which are displayed as white or black pixels respectively [7].
If regions are homogeneous with respect to some pixel characteristic, an ideal classification is possible. In practice, it is common rule to accept that this stage is imperfect and that pixels may be misclassified: next processing steps will have to deal with this.
The decision of labelling a gray pixel as white or black depends on a set brightness value $t$, called *threshold*. If the brightness of a pixel is less than $t$ then that pixel will be classified as background (c=0), otherwise it will be a component of the object of interest (c=1).

$$c[u, v] = \begin{cases} 0 & I[u, v] < t \\ 1 & I[u, v] \geq t \end{cases} \qquad \forall (u, v) \in I \tag{5.1}$$

There are several ways to implement a binary classification through thresholding techniques. The easiest one is represented by a trial and error approach for choosing the best value of $t$, once a gray-scale histogram is given: the

aim is to manually select a threshold that divides the brightness scale in two distinct sets as best as possible.
Figure 5.3 shows an example of thresholding.

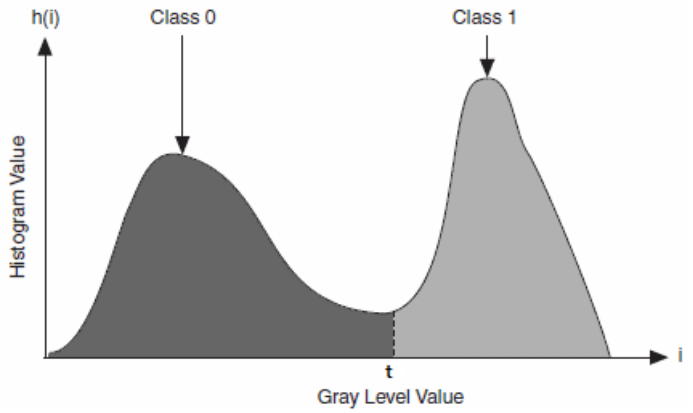



Figure 5.3: Choice of image theshold

The optimal global threshold can be selected through the Otsu's method [27](implementable with a single MATLAB function) which separates an image into two classes of pixels in a way that minimizes the intra-class variance (the variance within the class) while maximizes the inter-class variance (the variance between different classes).
The algorithm needs the strong assumption of bimodal distribution of the brightness.
Unfortunately the performances of these first two techniques deteriorate quite fast if the histogram does not present two distinct peaks. This circumstance could happen, for example, when the illumination is not uniform (see Figure 5.4 for an example of such case).
An alternative is to choose a local threshold rather than a global one. This solution can be implemented, for example, through the Niblack's algorithm [28] which computes a pixel-wise threshold by sliding a rectangular window $W$ over the gray-scale image.
The computation of threshold is based on the local mean $\mu(\cdot)$ and the standard deviation $\sigma(\cdot)$ of all the pixels in the window $W$ and is given by the following equation:

$$t(u,v) = \mu(u,v) + k\sigma(u,v) \qquad (u,v) \in I \tag{5.2}$$

where $k$ is a parameter used to control and adjust the effect of standard deviation, which depends on object features (defaults values are 0.2 for bright objects and -0.2 for dark objects).
The size of window $W$ is a critical parameter and has to be set as similar as

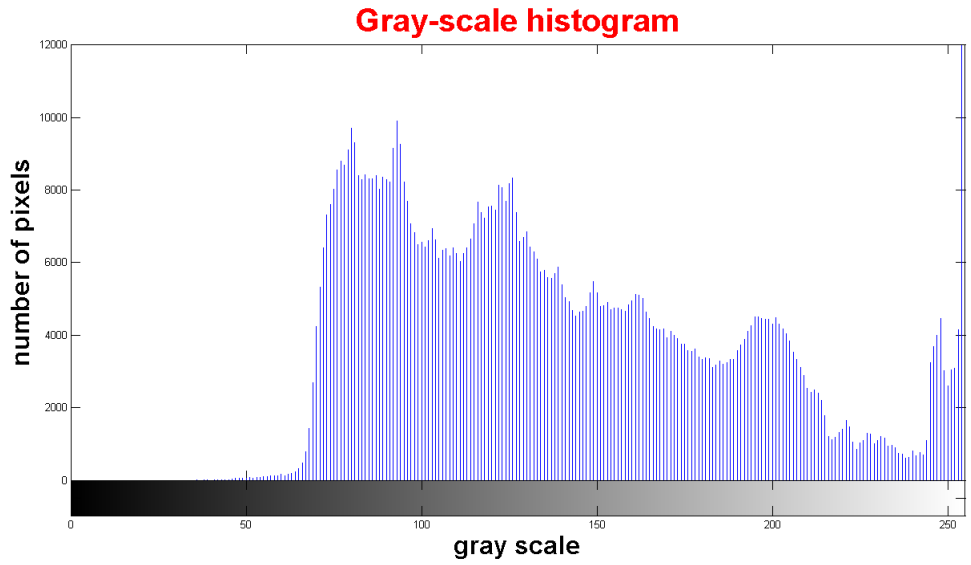
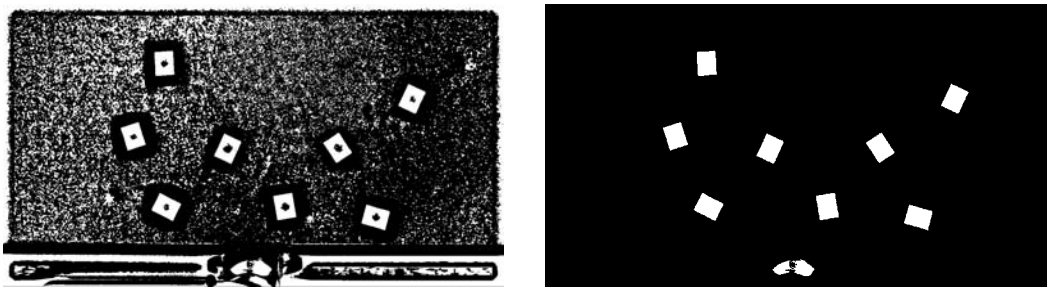



Figure 5.4: Gray-scale distribution due to non uniform illumination

possible to the size to the objects to be detected.
Despite these common rules, the choices of the size of $W$ and the parameter $k$ are generally given by a trial and error process.
Once a binary image is available, the **second step** of *image segmentation*, i.e. the **representation** process, can be performed [7]. This operation includes all those techniques which provide a final image where all adjacent pixels of the same class, i.e c=0 or c=1, are connected forming a *blob* (a spatially contiguous region of pixels of the same class).
Figure 5.5 shows the same scenario, before and after the implementation of representation process: it is clear how, in the processed image, the white blobs are clearly detectable and all other "impure" pixels are vanished.



(a) Binary image before representation (b) Binary image after represention

Figure 5.5

The main techniques, implemented in the Thesis via MATLAB functions, are briefly described in the following list.

- Noise removal through opening and closing morphological operations (see [7] for details).
- Enhancement through the use of suitable filter.
- Filling of the undesired holes.
- Clearing of the image borders.
- Deleting of meaningless detected objects.

Once it is known which pixels belong to each single objects in the scene ( thanks to the representation step), it is therefore possible to extract the most significant features of the blobs. Their knowledge allows to distinguish objects of interest with the remaining ones and the background.
This procedure represents the **third step** of `image segmentation` and it is commonly known as **description**.
In this project, thanks to the MATLAB function `regionprops`, a set of useful features for each labeled region were stored.

- **Area**: the actual number of pixels in the region [26];
- **Centroid**: centre of mass of the region [26];
- **Orientation**: the angle (in degrees ranging from -90 to 90) between the x-axis and the major axis of the ellipse that has the same second moments as the region [26];
- **Perimeter**: the distance between each adjoining pair of pixels around the boundary of the region [26];
- **Major Axis Length**: the length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region [26];
- **Minor Axis Length**: the length (in pixels) of the minor axis of the ellipse that has the same normalized second central moments as the region [26].

The direct use of this features could lead to misdetections, since none of them is simultaneously invariant to translation, rotation and scaling concerning observed objects.
In Chapter 6 the alternatives introduced to solve this limitations are described in details.

## 5.2 Camera Calibration

One of the most important aspect of a vision system regards camera calibration.
This procedure has the aim to determine all those parameters that allow a correct mapping between the (pixels) image coordinate frame and the main reference frame of the manipulator.
The efficiency of this operation results strictly important for application purposes.
In fact, an accurate calibration allows to control the robot movements through out the informations detectable in a simple image snapshot.
Before explaining the procedure, it is important to define and describe the two reference frames dealt with.

- Manipulator Reference Frame.
- Vision Reference Frame.

The acquired image is constituted by a matrix in which each element is the smallest unit of information on the pixel's brightness of the scene at that point.
The vision coordinate system is therefore given by row-column indices of the matrix, i.s. a discrete system based on the resolution of the camera (in this project there are 1280 x-axis and 720 y-axis possible values).
While the vision system is limited to 2-D coordinates ($x_v$,$y_v$), the manipulator Reference Frame is defined inside the real space, thus allowing 3-D coordinates ($x_r$,$y_r$,$z_r$).
Figure 5.6 shows a scheme for both coordinate systems.
It appears obvious the need to find a map that links the two reference systems, especially if the application is camera-based. A suitable and well-designed map is able to reduce the problems due to the perspective of the scene and to optical distortions related to the lenses present inside the camera (typical issues for a vision system).
The SANKYO SCARA (4 DOFs) allows to place the end-effector exactly above a precise point of the camera field of view, assuming a perpendicular angle between the end-effector z axis and the plane framed by the Webcam.
Following this assumption, the mapping function between pixel coordinates and manipulator coordinates can be expressed by:

$$\begin{cases} x_r = f_x(x_v, y_v) \\ y_r = f_y(x_v, y_v) \\ z_r = z_0 \end{cases} \tag{5.3}$$

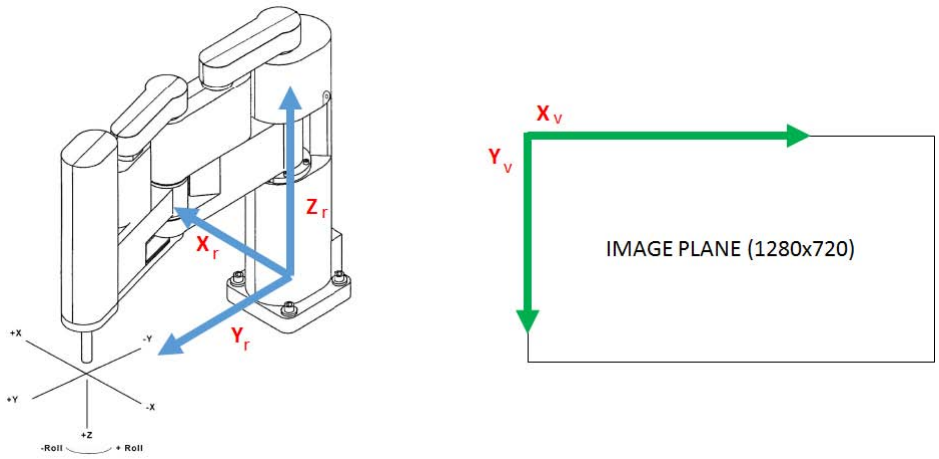



Figure 5.6: Manipulator and vision coordinate systems

where $z_0$ is a constant and known parameter.
A common approach for camera calibration would provide an estimation of extrinsic and intrinsic parameters of the camera, which are fundamental parameters to pass firstly from the robot reference system to the one of the camera and finally from the latter to the two-dimensional image plane frame [29].
The use of a simple Webcam as a vision sensor makes the implementation of this procedure really complex, suggesting to search for an alternative.
An ideal calibration can be seen as a particular rototranslation between the two different reference systems:

$$\begin{aligned} \begin{bmatrix} x_r \\ y_r \\ z_r \end{bmatrix} &= \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & a_0 \\ \sin(\alpha) & \cos(\alpha) & b_0 \\ 0 & 0 & z_0 \end{bmatrix} \cdot \begin{bmatrix} x_v \\ y_v \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} a_1 & a_2 \\ b_1 & b_2 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_v \\ y_v \end{bmatrix} + \begin{bmatrix} a_0 \\ b_0 \\ z_0 \end{bmatrix} \end{aligned} \tag{5.4}$$

from which is possible to get polynomial expressions for $f_x$ and $f_y$:

$$\begin{cases} f_x(x_v, y_v) = a_1 \cdot x_v + a_2 \cdot y_v + a_0 \\ f_y(x_v, y_v) = b_1 \cdot x_v + b_2 \cdot y_v + b_0 \end{cases} \tag{5.5}$$

Despite its simplicity, this simple linear model often is not robust for vision applications because not able to handle the problems concerning perspective and optical distortions.
These optical issues can however be compensated with good accuracy using

a non-linear model. This can be achieved increasing the degree of the polynomials describing $f_x$ and $f_y$ (in the project a second-order polynomial was enough to get satisfactory results):

$$\begin{cases} f_x = a_1 \cdot x_v + a_2 \cdot y_v + a_3 \cdot x_v^2 + a_4 \cdot y_v^2 + a_0 \\ f_y = b_1 \cdot x_v + b_2 \cdot y_v + b_3 \cdot x_v^2 + b_4 \cdot y_v^2 + b_0 \end{cases} \tag{5.6}$$

Once this model is chosen, the camera calibration process can be seen as an parameter identification problem where the coefficients $\theta_x = [a_1\ a_2\ a_3\ a_4\ a_0\ ]^T$ and $\theta_y = [b_1\ b_2\ b_3\ b_4\ b_0\ ]^T$ are the parameters to be estimated belonging to the two following linear statistical models (particular *Gauss model*):

$$\begin{cases} \mathbf{x_r} = S \cdot \theta_x + \mathbf{w_x} \\ \mathbf{y_r} = S \cdot \theta_y + \mathbf{w_y} \end{cases} \tag{5.7}$$

, where $\mathbf{x_r} = [\ x_r(1) \cdots\ x_r(n)]^T$ and $\mathbf{y_r} = [\ y_r(1) \cdots\ y_r(n)\ ]^T$ can be seen as the result of n measurements performed by placing the robot end effector just above different points of the workspace plane and $S$ represents a matrix that includes their corresponding coordinates in the vision reference frame

$$S = \begin{bmatrix} x_v(1) & y_v(1) & x_v(1)^2 & y_v(1)^2 & 1 \\ & & \cdots & & \\ x_v(n) & y_v(n) & x_v(n)^2 & y_v(n)^2 & 1 \end{bmatrix} \in \mathbb{R}^{nxp} \tag{5.8}$$

and $\mathbf{w_x}$ and $\mathbf{w_y}$ modelize measurement noises.
Model 5.7 is typically used in the description of digital communication channels, or for measurements made sequentially in time by numeric sensors of control systems [30].
In order to complete the camera calibration procedure, it necessary to calculate the maximum likelihood (ML) estimator of the parameters $\theta_x$ and $\theta_y$ $\in \mathbb{R}^p$ for the linear statistic model. [31].
Assuming $n > p$ (that means more measurements than number of parameters) and $\mathbf{w} = \sigma\mathbf{v}$, where $\mathbf{v} \sim \mathcal{N}(0, I)$, $I \in \mathbb{R}^{nxn}$ and $\sigma$ unknown.
A noise covariance matrix $R$ can be derived by $\mathbf{w}$ model:

$$\begin{aligned} R &= E\left[\mathbf{w} \cdot \mathbf{w}^T\right] \\ &= \sigma^2 I \end{aligned} \tag{5.9}$$

Thus because $\mathbf{w_x}$ and $\mathbf{w_y}$ are assumed independent and identically distributed (i.i.d.).
It can be also proved that, for *Rothenberg Theorem*, the parameter $\theta$ is globally identifiable (i.e. vector $\theta$ uniquely identifiable) if and only if $S$ is full rank

($rank(S) = p$), i.e. if and only if $S$ has $p$ columns linearly independents.
In the project $p = 5$, meaning 5 independent measurements had to be taken for the camera calibration operation, each of which is able to provide 5 linearly independent vectors such as $[x_v \; y_v \; x_v^2 \; y_v^2 \; 1]$ .
It is finally possible to define the expression of ML estimators for coefficient vectors $\theta_x$ and $\theta_y$:

$$\begin{aligned}\hat{\theta}_x &= \left[S^T R^{-1} S\right]^{-1} S^T R^{-1} \cdot \mathbf{x_r} \\ &= \left[S^T \left(\frac{1}{\sigma^2} I\right) S\right]^{-1} S^T \left(\frac{1}{\sigma^2} I\right) \cdot \mathbf{x_r} \\ &= \left[S^T S\right]^{-1} S^T \cdot \mathbf{x_r}\end{aligned} \tag{5.10}$$

$$\begin{aligned}\hat{\theta}_y &= \left[S^T R^{-1} S\right]^{-1} S^T R^{-1} \cdot \mathbf{y_r} \\ &= \left[S^T \left(\frac{1}{\sigma^2} I\right) S\right]^{-1} S^T \left(\frac{1}{\sigma^2} I\right) \cdot \mathbf{y_r} \\ &= \left[S^T S\right]^{-1} S^T \cdot \mathbf{y_r}\end{aligned} \tag{5.11}$$

For an identification process is always useful to test its efficiency.
To do that, it is compulsory to calculate the variance of the estimator. For a ML estimation, this parameter assumes the following expression:

$$Var(\hat{\theta}(\cdot)) = \sigma^2 \left[S^T R^{-1} S\right]^{-1} \tag{5.12}$$

It can be proved that the variance expression coincides with Cramer-Rao bound, thus resulting minimum variance estimator, i.e. fully efficient (the theory of Cramer-Rao can be applied since the estimator is unbiased) [30].

# Chapter 6

# Camera-based application

"The characteristic design of a SCARA offers compliance in the horizontal plane and high rigidity in vertical direction which is perfectly suited for Pick and Place Applications" [8].
As final part of the project, a vision-based application was designed and implemented.
It dealt with a grasping operation where eight identical small parallelepipeds had to be picked from a random position and placed into a predefined one. The latter represents a specific point of a pallet configuration ( resulted straightforward due to the MATLAB function described in Subsection 4.2.5). The manipulator has to perform this operation autonomously by using the mounted webcam and the pneumatic gripping system.
Figure 6.1 shows the workspace at the beginning and the end of the application execution.

(a) Random initial configuration (b) Final pallet configuration

Figure 6.1

The worktop is represented by a black metal base on which Eight manufactured plastic parallelepipeds are placed on a black metal base, representing the worktop. They are uniformly varnished with gray opaque paint in order to

avoid reflection issues.
Table 6.1 stores the main characteristics of a workpiece [4] (see Figure 6.2 for an example).

Table 6.1: Work-piece features

| | |
|---|---|
| color | gray |
| material | plastic |
| weight | 35 [g] |
| eight (a) | 35.48 [mm] |
| width (b) | 33.32 [mm] |
| thickness (c) | 26.31 [mm] |

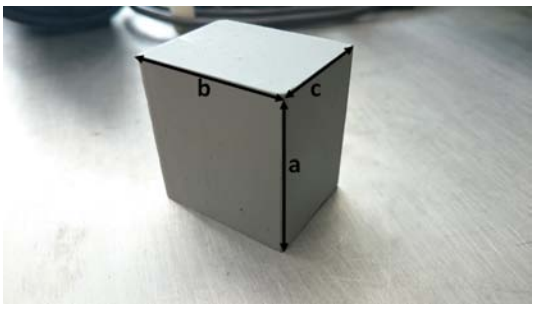
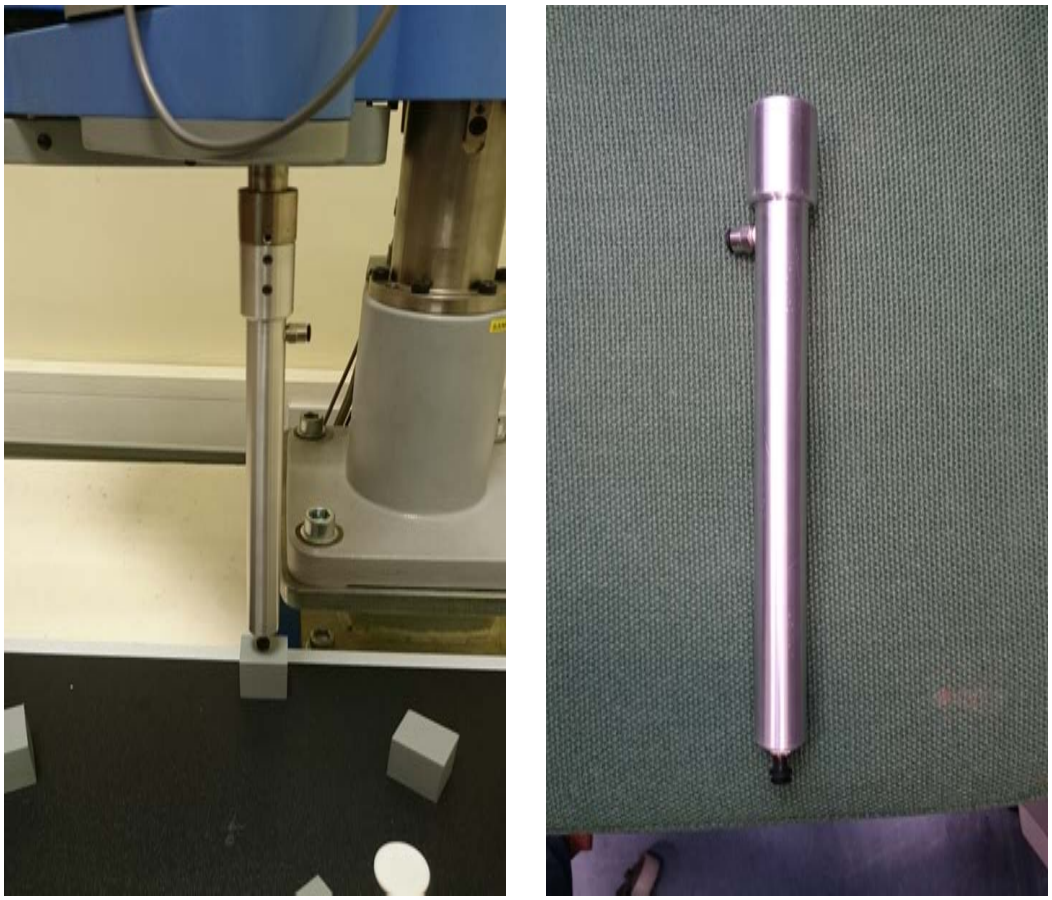



Figure 6.2: Sample of parallelepiped used for the application

## 6.1 Camera-calibration: practical setup

Section 5.2 already introduced camera calibration procedure from a theoretical point of view.
On the other hand this section provides an explanation about the practical implementation, describing, in details, the approach used to get an efficient map between image coordinates and manipulator coordinates.
First, the pneumatic gripper was replaced with an end-effector previously used in another project (see Figure 6.3b).
Given the like-pointer nature of this tool, it was possible to obtain a good precision regarding the acquisition of precise spatial coordinates within the

[4] the dimension values are related to a workpiece sample. The same parameters could be slightly different if measured on another parallelepiped, due to possible errors in manufacturing process.



(a) (b)

Figure 6.3: Practical implementation of camera calibration

robot workspace.
The procedure begins with the placement of five parallelepipeds in as many random positions within the workspace. These positions need to be included within the same field of view of the camera.
The operator must then manually position, through the teaching pendant, the end-effector exactly above the centroid of the upper rectangular face of the interest objects as precisely as possible, and thus acquire spatial coordinates relative to the manipulator coordinate system $[x_r(i),\ y_r(i),\ z_r(i)]$[5].
It is necessary to accurately point the centroid of the rectangle, since it represents also the grasping centre of the pneumatic gripper, which is fundamental

[5]The Z-axis coordinate $z_r(i)$ is not important since the system separately handles the z-shaft displacement and not affect the mapping between the two coordinate systems.

for a correct grasping (see Figure 6.3a).
After this stage, the robot must be moved to a position such that the five objects can simultaneously be observed. Successively, thanks to image processing techniques introduced in the previous Chapter, centroid positions in pixels coordinates $[x_v(i)\ y_v(i)]$ can be detected.
Knowing the centroid coordinate both in the worktop and in the image allows to determine the estimators of the coefficients that map the two coordinate systems (this calculation was possible due to the use of MATLAB codes `posx` and `posy`, which are listed in Appendix A).
These coefficient vectors are then passed to another MATLAB function (see Listing 6.1) which receives an object centroid position in pixels as input parameter, and outputs the same position measured in the manipulator workspace.

Listing 6.1: **Cartesian position detector function**

```
1
2  %% This function translates the pixel coordinates into Cartesian
3  %  coordinates
4  % INPUT: - a=vector containing centroid coordinates [pixell] of
5  %           interest objects;
6  %         - theta_x and theta_y vectors containing the estimated
7  %           coefficients with ML
8  %           estimator;
9  % OUTPUT: - vector containing the x-axis and y-axis position of
10 %            the object inside the manipulator coordinate system.
11 function [ o ] = position_Cartesian(a,theta_x,theta_y)
12
13 X=theta_x;
14 Y=theta_y;
15
16 px=X(1,1)*a(1,1)+X(1,2)*a(1,2)+X(1,3)*a(1,1)^2+X(1,4)*a(1,2)^2
17   +X(1,5);
18 py=Y(1,1)*a(1,1)+Y(1,2)*a(1,2)+Y(1,3)*a(1,1)^2+Y(1,4)*a(1,2)^2
19   +Y(1,5);
20
21 o=[px,py];
22
23 end
```

## 6.2 Parallelepipeds detection

A suitable MATLAB function was developed for parallelepiped detection, i.e. `OBJ_detection_niblack()` (see Appendix A for the code).

This function exploits the Niblack's thresholding method which proved to be the best for application purposes [6].
Function `OBJ_detection_niblack` exploits two other MATLAB local functions, both developed in the same m-file: `optimum_threshold_niblack` and `Image_processing_niblack`.
The first one has the aim to detect the best combination between window size $W$ and k gain, which are the two variables that set and eventually modify Niblack's threshold (see formula 5.2). This solution proved to be robust even for lighting changes. It is important to remind that if a threshold is not correct, an interest object could be not recognized.
The function `Image_processing_niblack` uses those techniques (using Matlab Image Processing Toolbox functions) allowing to reduce noise, clear borders, and make the shape of detectable interest object placed in the workspace as clear as possible.
Figure 6.4, Figure 6.5 and Figure 6.6 show the initial acquired image, the same snapshot after thresholding and the processed image, respectively.

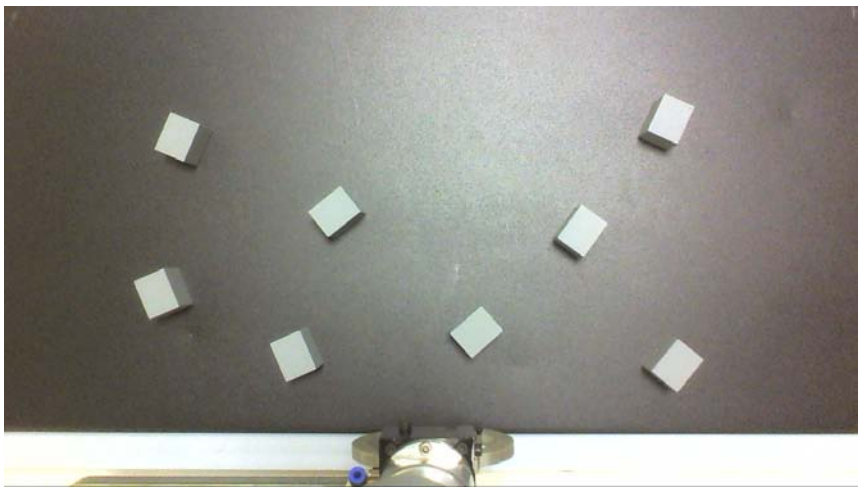


Figure 6.4: Initial acquired image snapshot

Figure 6.6 shows optimal result. Issues due to perspective disappeared, since in the final image only two-dimensional rectangular shapes were plotted (all the remaining meaningless objects have been assimilated to the black background).

[6] Some application tests were performed also using different thresholding techniques, such as "trial and error" and Otsu's method. To improve their efficiency, a MATLAB function for correcting non-uniform illumination was developed. Despite this solution, Niclack's approach remained the most reliable.

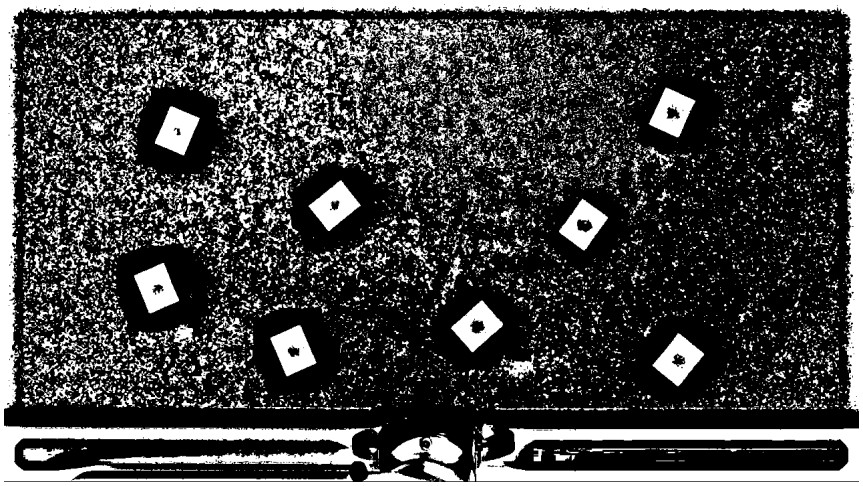


Figure 6.5: Image obtained through Niblack's thresholding method

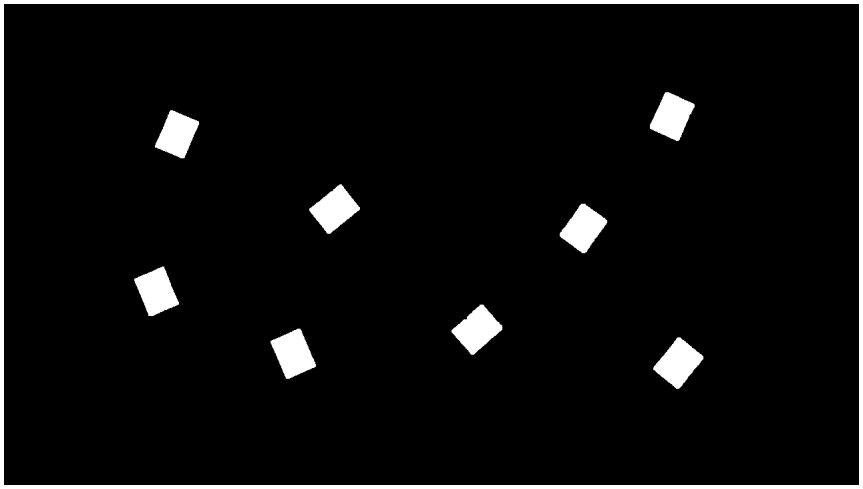
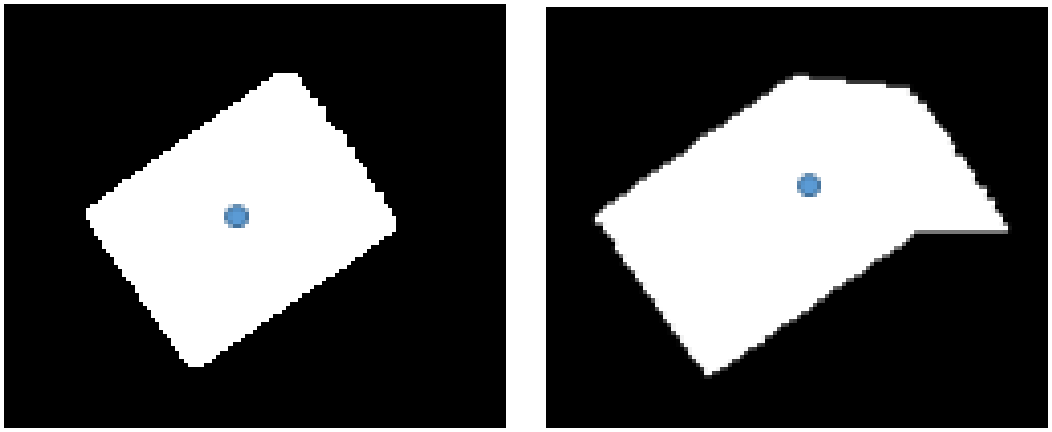


Figure 6.6: Image processed by `Image_processing_niblack` function

This aspect is crucial for a correct development of the application execution. In fact, the centre of grasping must be associated to the centroid of the upper rectangular face (see Figure 6.7a) and not to the global gray object viewed from the top in the initial acquired image (see again 6.4). In the last undesired case, which may result from a wrong threshold choice (see Figure 6.7b), the gripper could not be able to properly grab the parallelepiped, causing damages to the system and to the gripper itself (unexpected collision with one of work-piece edge).

From the processed image, the final step of detection process is implemented.



(a) (b)

Figure 6.7: Centroid detection: possible results

This deals with extraction of useful features that can distinguish between meaningful and meaningless objects. Two functions, provided by MATLAB Image Processing Toolbox, helps to achieve suitable results: `bwlabel` and `regionprops` [26].
Some descriptors are already been introduced in Subsection 5.1.2, underlining their weaknesses (possible misdetection).
For application purposes, other two common descriptors were used: aspect ratio Ar (or eccentricity) and circularity $\rho$. The first one represents the ratio of major to minor ellipse axis lengths. The second one is defined as:

$$\rho = \frac{4\pi \mathrm{A}}{\mathrm{p}^2} \tag{6.1}$$

where A is the Area of the region and p is its perimeter length. Circularity has a maximum value of $\rho = 1$ for circle, is $\rho = \pi/4$ for a square and $\rho = 0$ for an infinite long line.
These two descriptors turn out to be extremely robust since are invariant to translation, rotation and scale.
Listing 6.2 shows their computation.

Listing 6.2: **Circularity and aspect ratio computation**

```
1
2  % features extraction
3  stats = regionprops(L,'Area', 'Centroid',
4   'Orientation','Perimeter','MajorAxisLength','MinorAxisLength');
5
6  % store useful features
7  area=cat(1, stats.Area);
8  perimeter=cat(1,stats.Perimeter);
9  MajorAxis_elippse=cat(1, stats.MajorAxisLength);
10 MinorAxis_elippse=cat(1, stats.MinorAxisLength);
11
12 % Circularity
13 Circularity=cat(1,4*pi*area ./ perimeter.^2);
14
15 % Aspect Ratio
16 Ar=MajorAxis_elippse ./ MinorAxis_elippse;
```

Through an empirical approach, standard range values for $\rho$ and Ar were found. These limits were appositely set in order to deal with slight shape differences between each parallelepiped, non-uniform illumination and perspective point of view.
Table 6.2 reports this range values.

Table 6.2: Range value for image descriptors

| **Descriptor** | **range** |
|---|---|
| circularity $\rho$ | [0.5-0.85] |
| aspect ratio Ar | [1.10-1.35] |

Once these two descriptors are computed for each possible interest object, a simple `if` statement allows to store the interest rectangles.
If only a detected objects respect the range values then their orientation and centroid features are stored inside a matrix and outputted.
Figure 6.8 plots the detected rectangles with respective centroids.
All the detection procedure is summarized in the flowchart scheme of Figure 6.9.

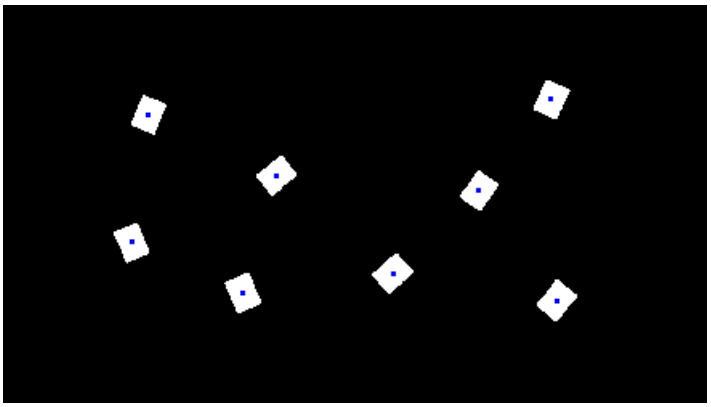


Figure 6.8: Detected rectangles with associated centroids (blue points)

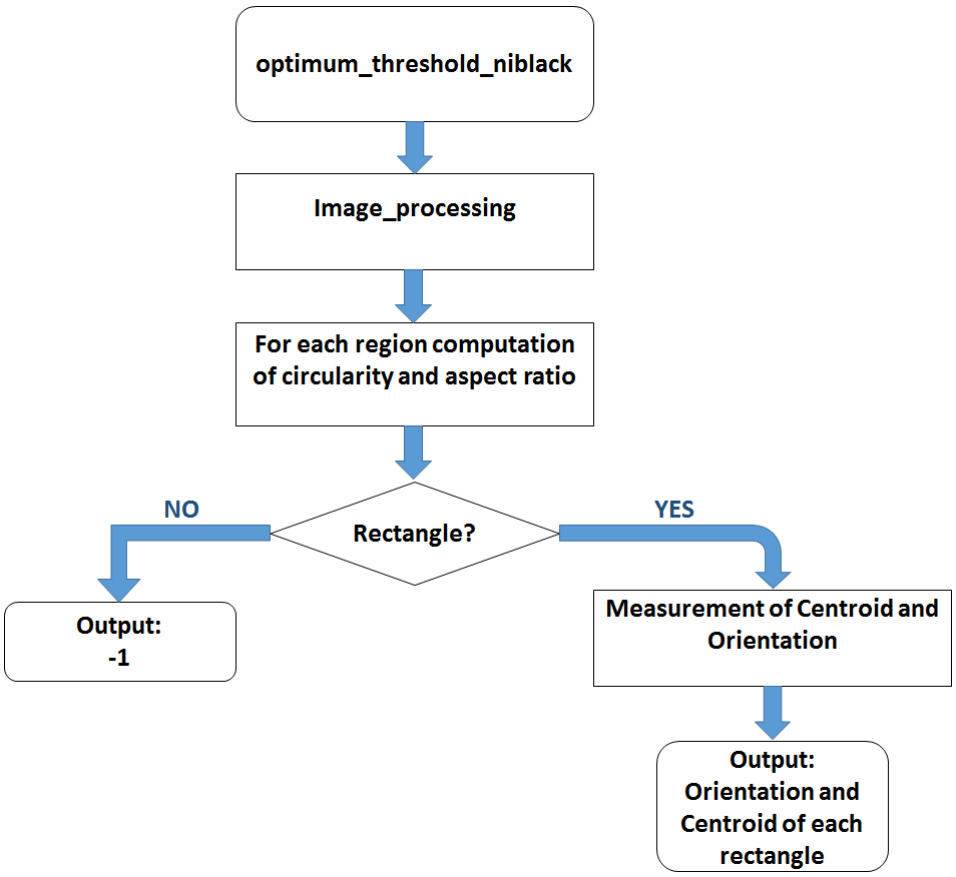



Figure 6.9: Flowchart describing MATLAB approach for work-piece detection

## 6.3 Application execution

The application is on the whole controlled through MATLAB function (see Figure 6.11 for the flowchart) `Camera_based_Palletizing`), which includes interest object detection and all those commands necessary to perform an operative application.
Preliminary steps deal with the setting of speed and the automatic acceleration/deceleration time related to the payload of a parallelepiped. Afterwards, relevant points for a pallet configuration are defined as well as the initial position point from where an image of the worktop is acquired and passed to the `OBJ_detection_niblack` (see previous Section).
If one or more rectangles are detected, then an operation of picking and placing of work-pieces into pallet points is performed.
An important aspect of the application concerned a suitable calibration between the effective angle measured by the end-effector and the orientation of object to be picked. It is essential to underline that the gripper has to align its axis in such a way that the object is grasped from the shorter side of rectangle surface (longer side could not properly fit the distance between gripper tips).
An empirical approach leads to the calibration showed in piece of code of Listing 6.3, where the mapping between angles changes according to the sign of detected 'Orientation' feature ( this one is stored in the first column of variable `OBJ`, while `s` variable represents gripper orientation).

Listing 6.3: **Calibration of gripper orientation**

```
1
2  % Calibraion of gripper's orientation
3  if(OBJ(i,1) >= 0)
4
5      s = -OBJ(i,1)+175;
6
7  else
8
9      s=-OBJ(i,1)-5;
10
11 end
```

Once the operation is completed, the manipulator moves to initial position and the camera is turned off.
The following points summarize the whole procedure, briefly describing each performed step:

1. Preliminary settings as: speed, acceleration/deceleration time, pallet

configuration and initial position.

2. Turning on the camera and image acquisition.

3. Object detection through `OBJ_detection_niblack` and storing of 'Centroid' and 'Orientation' features.

4. Starting of `while` cycle where each iteration corresponds to one pick and place action:

   **a**) `position_Cartesian` function allows to get centroid coordinates with respect to the main reference frame of the manipulator;

   **b**) moving the manipulator up to the centroid position;

   **c**) change of the end-effector orientation in order to allow a correct grasping (gripper calibration);

   **d**) lowering of the gripper and grasp of the parallelepiped through switch on the pneumatic feed system (`out(938,1)`);

   **e**) lifting of the gripper and move the manipulator up to a pallet point, and release, when arrived, the work-piece (`out(938,0)`);

   **f**) lifting of the gripper and go ahead with the procedure if pallet configuration is still not complete (return to point **a**)).

5. Initial position return.

6. Turning off the camera.

The complete code of the application can be found in A, while Figure 6.10 shows some snapshots of the application progress.

### 6.3.1 Results and comments

For a correct execution, few initial adjustments and constraints had to be applied. These actions turned out to be useful and helpful solutions for a feasible implementation of the pick&place operation.

- Due to delay time between an input signal, used to switch the air flow by the valve, and the real response of the gripper status (opening or closing), a manual time pause was set, after the call of MATLAB functions `out(938,0/1)`, in order to let a correct grasp and release.

- The pallet configuration required a precise space, which had to be left available and known a priori (in order to avoid unexpected collisions).

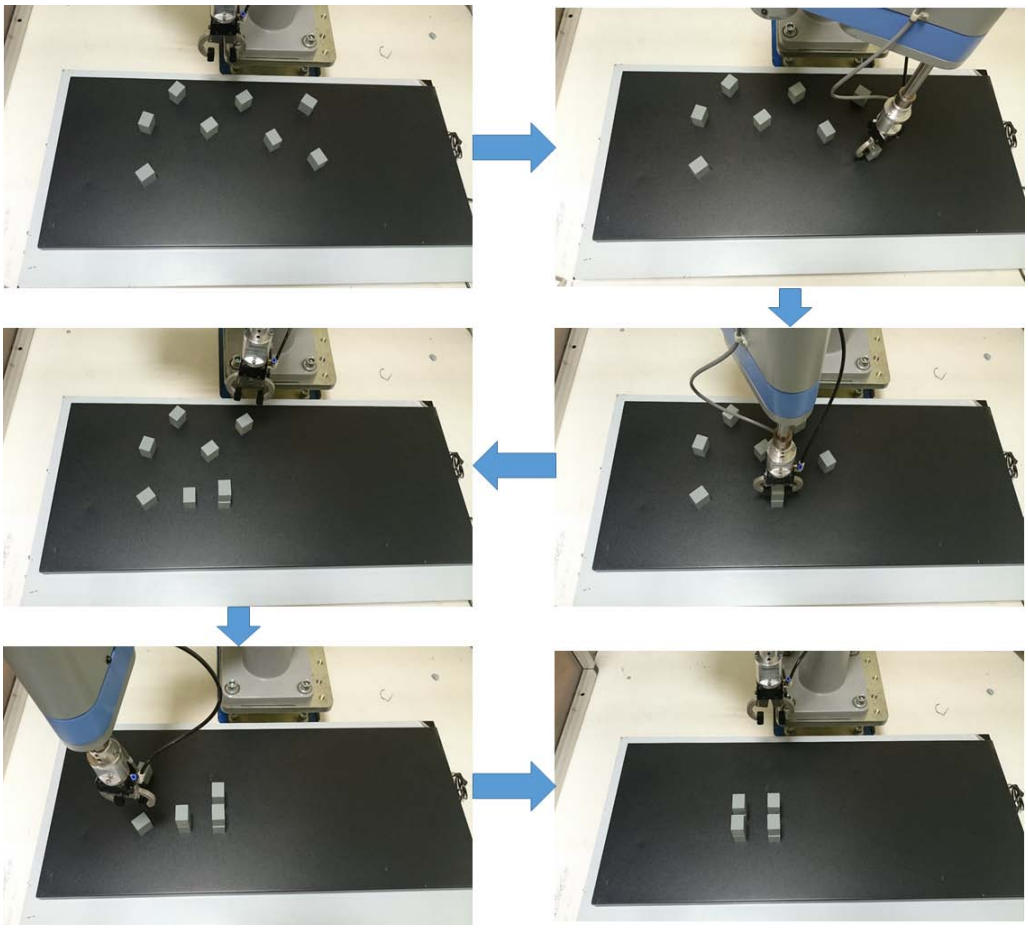


Figure 6.10: Example of application iterations implemented via MATLAB code

- Two adjacent objects were placed not too near, allowing the gripper to focus only on the object to be picked without dealing with, during that operation, other parts in the worktop.
- Time necessary to correctly detect the eight parallelepipeds is quite high due to the while cycle used to find the best possible combination between the parameters of Niblack's thresholding. This represents a trade-off between efficiency and execution velocity.
- The colors of work-piece and worktop board were chosen in order to help the detection, hence improving the contrast between background and interest objects and partially solving non-uniform illumination and reflection issues.

Despite these necessary constraints, the final application showed optimal

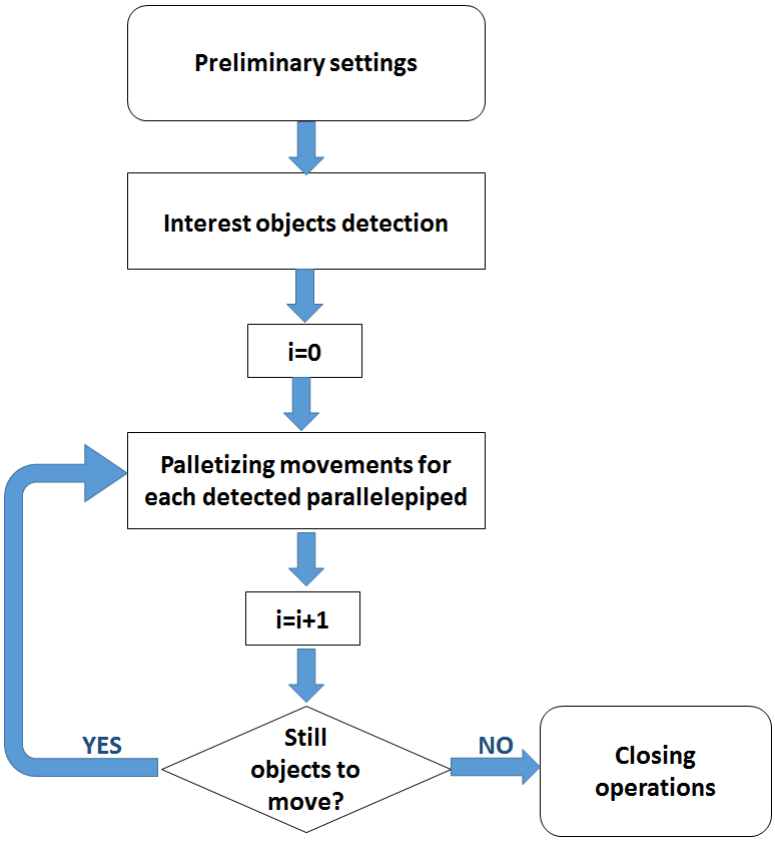



Figure 6.11: Flowchart describing MATLAB control application

result, especially in terms of robustness and reliability with respect of the initial random placement of the eight parallelepipeds: the perspective problem were successfully solved allowing a correct control of the grasping actions, irrespective from initial position of an object.
Also the execution speed was high, with a fixed limit value only to avoid damage of the manipulator structure.

# Chapter 7

# Conclusions

This thesis work involved the development of a robotic work cell for gripper-based SCARA manipulator.
The first part of the project dealt with the development of a suitable gripping system (both pneumatic and electrical design). This study also concerned an appropriate mounting of proper sensors and a microcontroller board used for acquisition and control of the gripping action (this work exploits an Arduino Uno board).
A crucial aspect of this section is represented by the use of MATLAB as control software to handle the overall system: a previous project allowed to translate many functions written in the manipulator own language into feasible MATLAB code.
An important study involved a suitable synchronization between Arduino signals and MATLAB commands.
On the other hand, the second part described the vision system connected to the manipulator, describing in detail common algorithms used to manage the acquisition and elaboration of images through MATLAB Toolboxes.
As natural consequence, the final part dealt with the design of a camera-based application of automated pick and place, which allows to incorporate all studies presented during this work.
The final result is appreciable due to good efficiency and reliability of such application, made mainly possible by a careful calibration procedure via parametric identification.
Moreover, the developed gripping system allowed to handle 3-D object, improving a previous installed vacuum end-effector able only to suck planar parts, such keys or poker fiches.

## 7.1 Future works

Despite satisfactory achievements, few improvements can be applied to the project.
Future works may include the following tasks.

- Development and design of an electrically based gripper, easier to control. Regulate the air pressure is a complex procedure due to sponginess of air.
- Modelling the overall system allowing the implementation of controller such as PID or state-space controller.
- Improvement of the robotic work-cell adding dynamic devices such as a conveyor belt. This inclusion forces the operator to develop a quick image detector system.
- Employment of C++ language to control the manipulator which speed a lot the communication between PC and robot, allowing real-time controller.
- Use a camera with better performances able to make the vision system more efficient. The vision sensor can be mounted on a fixed position above the worktop, with the addition of a uniform lighting system performed by suitably installed lamps.

# Appendix A

# Code

Listing A.1: **close camera function**: disabling video storing

```
1
2  %% This function allows the operator to turn off the camera
3
4  function [ ] = close_cam(  )
5
6  vid=start_cam('retrieve');
7
8
9  stop(vid);
10 delete(vid);
11
12 end
```

Listing A.2: **estimate of $\theta_x$ and $\theta_y$**

```
1
2  %% theta_x axis parameter estimation through ML estimator
3  % (use of 5 positioning points)
4
5  % Matrix including image centroid O coordinates [pixel]
6  S=[ O(1,2) O(1,3)  O(1,2)^2 O(1,3)^2 1;
7  O(2,2) O(2,3)  O(2,2)^2 O(2,3)^2 1;
8  O(3,2) O(3,3)  O(3,2)^2 O(3,3)^2 1;
9  O(4,2) O(4,3)  O(4,2)^2 O(4,3)^2 1;
10 O(5,2) O(5,3)  O(5,2)^2 O(5,3)^2 1];
11
12 % vector inccluding y centroid coordinates [mm]
13 X_r=[P1(1,1) P2(1,1) P3(1,1) P4(1,1) P5(1,1)];
14
15
```

```
16
17 % Maximum Likelihood estimator, using Markov model
18 if(rank(S)==size(S))
19     theta_x=inv(S'*S)*S'*X_r'
20 end
21
22 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
23 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
24
25 %% theta_y axis parameter estimation through ML estimator
26 % (use of 5 positioning points)
27
28 % Matrix including image centroid O coordinates [pixel]
29 S=[ O(1,2) O(1,3)  O(1,2)^2 O(1,3)^2 1;
30 O(2,2) O(2,3)  O(2,2)^2 O(2,3)^2 1;
31 O(3,2) O(3,3)  O(3,2)^2 O(3,3)^2 1;
32 O(4,2) O(4,3)  O(4,2)^2 O(4,3)^2 1;
33 O(5,2) O(5,3)  O(5,2)^2 O(5,3)^2 1];
34
35
36 % vector including y coordinates [mm]
37 Y_r=[P1(1,2) P2(1,2) P3(1,2) P4(1,2) P5(1,2)];
38
39
40
41 % Maximum Likelihood estimation, using Markov model
42 if(rank(S)==size(S))
43     theta_y=inv(S'*S)*S'*Y_r'
44 end
```

Listing A.3: **Interest objects detection**

```
1
2 %% This function detects objects of interest and stores their
3 %% orientation and centre of mass inside OBJ variable
4 % INPUT=none;
5 % OUTPUT= - '-1' if no objects are detected;
6 %          - Nx3 matrix where N is the number of detected objects
7 %            first column: orientaion (degree)
8 %            second column: x-position centroid (pixel)
9 %            third column: y-postion centroid (pixel)
10
11 function [o]=OBJ_detection_niblack()
12
13 % save the video as variable and get a snapshot of it in order to
14 % process the acquired image
15 vid_obj=start_cam('retrieve');
16 Acquired_Image=getsnapshot(vid_obj);
17
```

```
% conversion of the colored image into a gray-scale image
I_gray=rgb2gray(Acquired_Image);

% Image Processing of the gray image
BW_final=Image_processing_niblack( I_gray );

% The next statements returns a matrix L (same size BW_final)
% containing labels for the connected objects.
% n is the number of connected objects.
[L, n] = bwlabel(BW_final);


% save the most importan feature of each labelled region
stats = regionprops(L,'Area', 'Centroid',
 'Orientation','Perimeter','MajorAxisLength','MinorAxisLength');

% store useful features
area=cat(1, stats.Area);
perimeter=cat(1,stats.Perimeter);
MajorAxis_elippse=cat(1, stats.MajorAxisLength);
MinorAxis_elippse=cat(1, stats.MinorAxisLength);

% Circularity
Circularity=cat(1,4*pi*area ./ perimeter.^2);

% Aspect Ratio
Ar=MajorAxis_elippse ./ MinorAxis_elippse;


i = 1; % Iteration variable
j = 1; % Position pointer in 'OBJ'

OBJ=[0,0,0,0];

while(i <= n)
    % range values were obtained from multiple experiments
    if( Circularity(i,1) >= 0.5 & Circularity(i,1) <= 0.85 &
        Ar(i,1)>= 1.10 & Ar(i,1)<=1.35)

        % counting the detected objects
        OBJ(j,1)=i;

        % store the orientation [deg]
        OBJ(j,2)=stats(i).Orientation;



        %store the centroids
        OBJ(j,3)=(stats(i).Centroid(1)); %x position [pixel]
        OBJ(j,4)=(stats(i).Centroid(2)); %y position [pixel]

```

```
69          % update iteration variable
70          j=j+1;
71      end
72
73      % update iteration variable
74      i=i+1;
75  end
76
77
78  % if no parallelepiped is detected retrun -1, otherwise
79  % a vector including features
80  if(OBJ(1,1)==0 & OBJ(1,2)==0 & OBJ(1,3)==0 & OBJ(1,4)==0)
81      o=-1;
82  else
83      o=[OBJ(:,2), OBJ(:,3), OBJ(:,4)];
84  end
85  end
86
87  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
88  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
89
90  %% This function converts the gray image into a binary BW image
91  %% through the use of niblack's thresholding method
92  % INPUT= - gray-scale image;
93  % OUTPUT= - optimum threshold values represetend by  a 1x3
94  %           vector including size of the window W and value of
95  %           constant k;
96  %         - Binary BW processed image
97
98  function [ o, opt_threshold] = Image_processing_niblack(I_gray)
99
100 % plot and show the gray-scale histogram
101 figure
102 imhist(I_gray)
103
104
105 % adaptive niblack algorithm
106 opt_threshold=optimum_threshold_niblack(I_gray)
107 BW_start=niblack(I_gray,[opt_threshold(1,1)
108  opt_threshold(1,2)],opt_threshold(1,3));
109
110 % noise removal=opening + closing
111 % SE has to be smaller than objects shape
112 SE=strel('square',2);
113
114 BW_start=imopen(BW_start,SE);
115 BW_start=imclose(BW_start,SE);
116
117 % EDGE DETECTION
118 BW_sobel = edge(BW_start,'sobel');
119 BW_canny = edge(BW_start,'canny');
```

```
120 BW_LoG = edge(BW_start, 'log');
121
122
123 % enhancement filter
124 h = fspecial('unsharp',0.3);
125 BW_filter = imfilter(BW_start,h,'conv');
126
127 % filling the holes
128 BW_fill = imfill(BW_filter,'holes');
129
130 % clear borders
131 BW_fill = bwmorph(BW_fill,'clean',200);
132 BW_border = imclearborder(BW_fill,26);
133
134 BW_final = bwareaopen(BW_border, 2400);
135 o=BW_final;
136 end
137
138 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
139 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
140
141 %% This function is able to find the best variable combination
142 that has to be pass to Niblack's thresholding formula.
143 % INPUT= gray-scale image;
144 % OUTPUT=   - 1x3 array where
145 %           first element: width of window W (pixel)
146 %           second element: height of wondow W (pixel)
147 %           third element: k gain parameter
148
149 function [ o] = optimum_threshold_niblack( I )
150
151 opt_thr = [0 0 0]; %optimum BW array threshold
152
153 opt_thr_n_obj = 0; % Number of objects that match rectangular
154                    % shape by using 'opt_thr'
155 current_n_obj = 0;
156
157 % iteration variable for window size W
158 p=0
159
160 % iteration variable for k gain parameter
161
162 % double loop cycle to detect best combination between
163 % window size and k gain parameter
164 for p=50:5:70
165
166     for l=0.2:0.05:0.8
167
168         BW = niblack(I,[p  p],l);
169
170         SE=strel('square',2);
```

```
171
172         BW=imopen(BW,SE);
173         BW=imclose(BW,SE);
174
175         h = fspecial('unsharp',0.3);
176         BW = imfilter(BW,h,'conv');
177
178         BW = imfill(BW,'holes');
179         BW = bwareaopen(BW, 800);
180
181         [L, n] = bwlabel(BW);
182
183         stats = regionprops(L,'Area', 'Perimeter',
184         'MajorAxisLength','MinorAxisLength');
185
186         area=cat(1, stats.Area);
187
188         perimeter=cat(1,stats.Perimeter);
189
190         MajorAxis_elippse=cat(1, stats.MajorAxisLength);
191         MinorAxis_elippse=cat(1, stats.MinorAxisLength);
192
193         % Circularity
194         Circularity=cat(1,4*pi*area ./ perimeter.^2);
195
196         % Aspect Ratio
197         Ar=MajorAxis_elippse ./ MinorAxis_elippse;
198         i=1;
199
200         while(i <= n)
201
202         if( Circularity(i,1) >= 0.5 & Circularity(i,1) <= 1
203         & Ar(i,1)>= 1.10 &  Ar(i,1)<=1.35)
204             current_n_obj = current_n_obj +1 ;
205         end
206             i=i+1;
207         end
208
209         if(current_n_obj > opt_thr_n_obj)
210
211             opt_thr_n_obj = current_n_obj;
212
213             opt_thr = [p p l];
214         end
215
216             current_n_obj = 0;
217
218         end
219
220     end
221
```

```
o = opt_thr;

end
```

Listing A.4: **Camera-based application**

```

%% CAMERA BASED PALLETIZING OPERATION

clear all
close all
clc

% Set the speed
speed(15);

% Set the acceleration and deceleration time
% (depending on the weight)
autoacl(1);
weight(0.035); %the weight of a parallelepiped is 35 g

% Open camera
start_cam('open')

% wait 2 s to give the camera the time to switch on
pause(2);


% Define the pallet points for the configuration
Ap=[124.1679  400.9497 56 85];
Bp=[124.1679  400.9497 20  85];
Cp=[200.1679 400.9497 56 85];
Dp=[124.1679 350.9497 56 85];


j=0; % work-piece placement position iteration variable



% define the initial position of the manipulator
pos_iniz = [130.0586  176.8722   25.0000   85.0000];

% move to the initial position if not already there
while(mark()~= pos_iniz )
move(pos_iniz);
end

% detect the centroids of the parallelipeds (in the image frame)
OBJ = OBJ_detection_niblack();
```

```
44
45 % define a pallet configuration knowing the main points
46 setplt(4,Ap,Bp,Cp,Dp,2,2,2);
47
48 % size of the vector inluding the centroid coordinates,
49 % allowing to know the number of detected work-pieces
50 [a,b] = size(OBJ); % Returns size of OBJ dimensions
51
52 i = 1; % Iteration variable
53
54
55 % Sequential pick and place operations
56 while(i <= a)
57
58     % selection of the i-th work-piece
59     p=OBJ(i,:);
60
61     % find the centroid position in Cartesian Coordinates
62     pos=position_Cartesian([p(1,2) p(1,3)], X,Y);
63
64     % move the gripper above the work-piece to be grasped
65     move([pos(1,1) pos(1,2) 25 85]);
66
67
68
69     % detection of the orientation of the parallelepipeds/Gripper
70     % calibration
71     if(OBJ(i,1) >= 0)
72
73         s = -OBJ(i,1)+175;
74
75     else
76
77         s=-OBJ(i,1)-5;
78
79     end
80
81     % orienting of the gripper to allign the jaws axis to the
82     % work-piece orientation
83     smove(4,s);
84
85     % lower down th gripper in order to reach a right height
86     % to grab the object
87     smove(3,130)
88
89     % activate the pneumatic feed =>closing action of the gripper
90     out(938,1);
91
92     % wait a while in order to let the correct grasping
93     pause(0.7)
94
```

```
95      % raise up the gripper: start of the placing movement
96      smove(3,40)
97
98
99      % recall the pallet configurationg and
100     % save one of its point as variable
101     pallet_point=plt(4,i);
102
103     % move the grasped object above pallet position
104     move(pallet_point)
105
106     % lower the gripper
107     srmove(3,75)
108
109     % opening action of the gripper
110     out(938,0);
111
112     % wait a while in order to let the correct object release
113     pause(0.5);
114
115     % raise the gripper again
116     smove(3,56);
117
118     % update variable
119     i=i+1;
120
121     % update variable
122     j=j+1;
123
124 end
125
126 % move back to the initial position
127 move(pos_iniz)
128
129 % desable the automatic setting of acceleration and deceleration
130 autoacl(0);
131
132 % close the camera
133 close_cam();
```

# Appendix B

# Image processing steps

This appendix shows some pictures describing how the image processing procedure affects, step by step, the acquired initial snapshot.

Following points describes each of the seven pictures showed in Figure B.1.

1. Initial acquired coloured snapshot. This represent the first image which is passed to MATLAB as soon as after its acquisition via webcam.

2. Gray-scale converted image. This element is provided by the use of MATLAB function `rgb2gray()`.

3. Thresholded black&white image (via Niblack's method). The application of Niblack's thresholding provides a black and white image which is easier to process with MATLAB tools.

4. Image after noise-removal technique. This operation starts to clean the b&w image, getting rid of the smaller white dots via morphological techniques.

5. Image after application of fill-the-holes procedure. The result is due to the use of MATLAB function `imfill()` which closes the holes placed in the rectangle's centroids.

6. Image with cleared borders. `imclearborder()` function makes the borders black to avoid misdetection.

7. Final processed image. `bwareaopen()` function provides the final b&w image, removing those white regions with pixel area smaller than the one associated with interest rectangles.

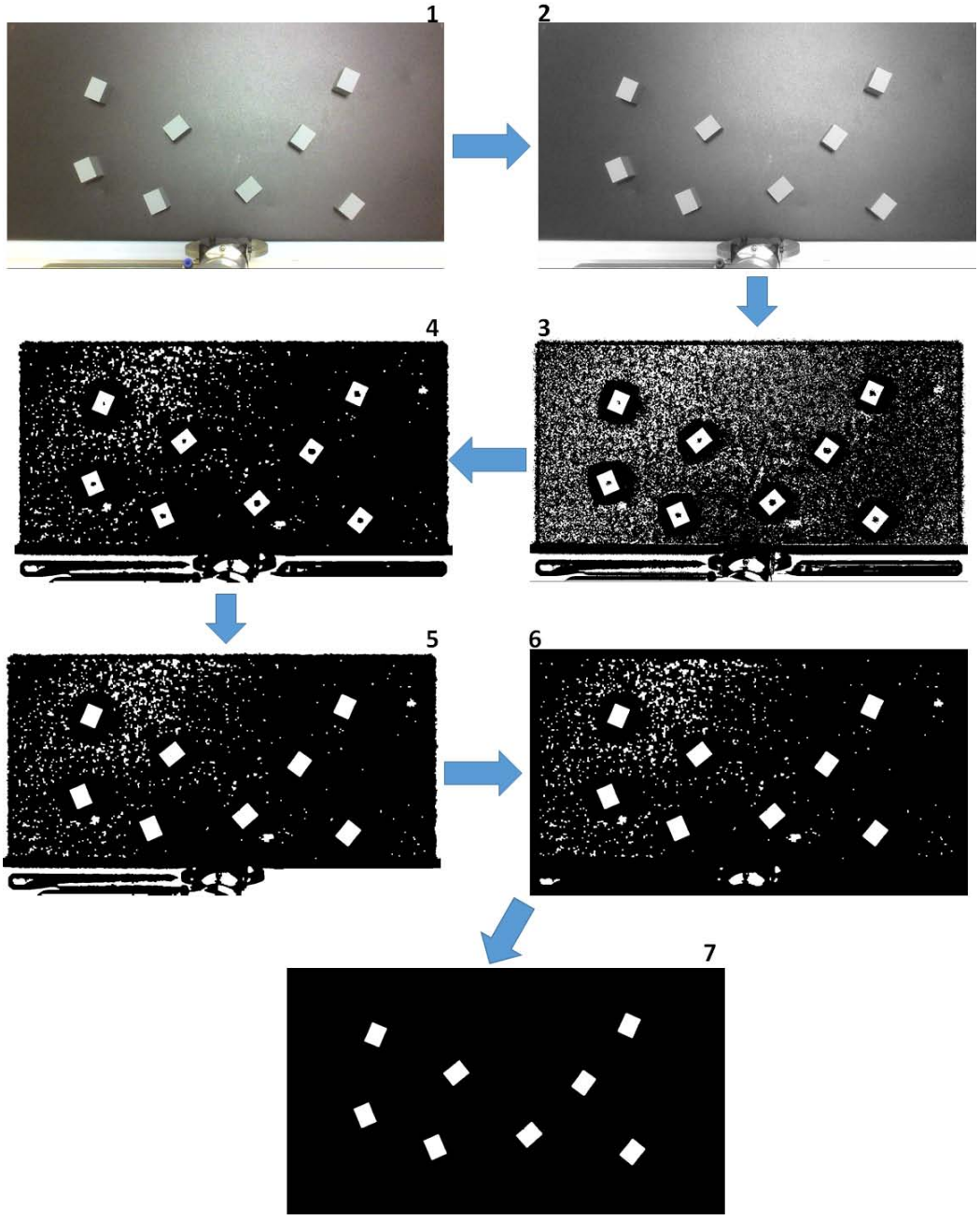



Figure B.1: Image processing steps performed via MATLAB functions

# Bibliography


[1] A. Wolf, R. Steinmann, and H. Schunk, "Gripper in Motion: The Fascination of Automated Handling Tasks", *Springer*, p.p. 16-31, 2005.

[2] G. Fantoni, S. Capiferri, J. Tilli, "Method for supporting the selection of robot grippers", *24th CIRP Design Conference*, 2014.

[3] V. D. Latake, Dr. V. M. Phalle, "A survey paper on a factors affecting on selection of mechanical gripper", *International Journal of Engineering Development and Research.*

[4] A. Glaser, "Industrial robotics: how to implement the right system for your plant", *Industrial Press*, 2008.

[5] E. Malamas, E. Petrakis, L. Petit, "A survey on Industrial Vision Systems, Applications and Tools, *Image and Vision Computing 21*, 2008.

[6] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo "Robotics: Modelling, Planning and Control", *Springer*, 2009.

[7] P. Corke, "Robotics,Vision and Control, Fundamental Algorithms in MATLAB", *Springer*, 2011.

[8] *SR8000 Series Hardware Manual,SC3000 Robot Systems*, NIDEC SANKYO.

[9] *SCARA Series Hardware Manual,SC3000 Robot Systems*, NIDEC SANKYO.

[10] S. Mc Sweeney, "Construction of a Robot Manufacturing Cell", *Master Thesis*, University College Cork, 2007.

[11] Setra, Models 280E/C280E Pressure Transducer and Pressure Transmitter Three-Wire, Voltage Output—Two Wire, Current Output, 280E Datasheet.

[12] Setra, Model 280 Gauge, Compound & Absolute Pressure Transducer.

[13] Honeywell Sensing and Control, FSG15N1A Datasheet.

[14] Honeywell Installation Instructions for FSG Series Force Sensor, Rev.A 50090349.

[15] Online: https://www.arduino.cc/.

[16] M. Abdallah, O. Elkeelany, "A Survey on Data Acquisition Systems DAQ", *Computing, Engineering and Information, 2009. ICC '09. International Conference on*, p.p. 240-243, 2-4 April 2009, Fullerton, CA.

[17] Burr Brown, Precision, Low Power INSTRUMENTATION AMPLIFIER, INA118 Datasheet.

[18] Texas Instrument, $\mu$A 741 General-Purpose Operational Amplifiers, $\mu$A 741 Op-amp Datasheet.

[19] Multicomp, Bipolar Transistor, BJT npn 2N3053A Datasheet.

[20] J. Karki, "Signal Conditioning Wheatstone Resistive Bridge Sensors", *Application Report SLOA034*, September 1999.

[21] B. Siciliano,O. Kathib, "Handbook of Robotics", *Springer*, 2008.

[22] Creative, LIVE! CAM Chat HD manual, 2013.

[23] L. Ferrari, "Matlab-based Control of a SCARA Robot.", *Master Thesis*, Università degli Studi di Padova, 2014.

[24] M. Spong, S. Hutchinson, M. Vidyasagar, "Robot Dynamics and Control", Second Edition„ 2004

[25] *Image Acquisition Toolbox User's Guide*, MathWorks, 2014.

[26] *Image Processing Toolbox User's Guide*, MathWorks, 2014.

[27] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms", *IEEE transactions on Systems, Man, and Cybernetics*, Vol.SMC-9, No.1, January 1979.

[28] W. Niblack, "An introduction to Digital Image Processing", *Prentice-Hall*, 1986.

[29] R. Kavanagh, "Industrial Automation and Control Course Notes", UCC (Cork), 2013.

[30] G. Picci, "Metodi statistici per l'identificazione di sistemi lineari", Dispense, Padova, 2007.

[31] T. Sodestrom, P. Stoica, "System Identification", *Pretince Hall*, 1989.