Università degli studi di Padova

Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Magistrale in

Ingegneria Elettronica

Matlab-based Control of a SCARA Robot

Relatore

Prof. Alessandro Beghi

Candidato

Luca Enrico Ferrari

Correlatore

Dr. Richard Kavanagh

Anno Accademico 2014/2015

Abstract

This master's thesis shows how it is possible to increase the flexibility and the functionality of a SCARA robot by introducing an interpreter in order to control the robot through Matlab, a very versatile and powerful programming language. It is explained how a Matlab control of the robot opens interesting scenarios and how the Matlab control has been implemented.
A SCARA robot is a widely used industrial manipulator with three axes and four degrees of freedom. Common applications of this robot are pick and place operations, assembling, palletizing, and packaging.

Acknowledgements

First of all, I sincerely would like to thank my supervisor Doctor Richard Kavanagh who helped me in all the phases of the project with advice and ideas which have been crucial for the fulfillment of this thesis.
I also would like to thank Professor Alessandro Beghi for all the precious advice and support that he gave to me during the project.
Furthermore, I greatly appreciated Michael O'Shea, Hilary Mansfield, Timothy Power, James Griffiths, Ralph O'Flaherty for the impeccable technical support at UCC laboratories.
Finally, I would like to extend my appreciation to Milind Rodake with whom I shared this experience at the UCC Mechatronic Laboratory.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

This master's thesis shows how it is possible to increase the flexibility and the functionality of a SCARA robot by introducing an interpreter in order to control the robot through Matlab, a very versatile and powerful programming language.
A SCARA robot is a widely used industrial manipulator with three axes and four degrees of freedom. Common applications of this robot are pick and place operations, assembling, palletizing, and packaging. The Scara robot involved in this project is the Robot SR8408, produced by The NIDEC SANKYO Corporation. This robot and the hardware configuration in which it is usually employed has to satisfy strict policies in terms of safety and reliability. It leads hardware and software rigidities, clashing with the university research approach which is more focused on prototyping and on the development of new solutions.
This thesis provides an explanation of how a Matlab control of the robot opens interesting scenarios and how the Matlab control has been implemented.
Chapter 1, after a brief introduction to the overrall project, concerns the objectives of the thesis and the reasons why the software Matlab has been chosen to control the robot.
Chapter 2 deals with the structure and the mathematical analysis of the SCARA robot.
Chapter 3 is about the Robot SR8408, produced by The NIDEC SANKYO Corporation.
Chapter 4 shows the hardware configuration used in the laboratory. Furthermore, the chapter deals with the structure of the interpreter and its communication with Matlab.
Chapter 5 describes the Matlab functions for robot control.
Chapter 6 concerns two vision-based applications developed using an HD camera. In this chapter, the vacuum gripping system is also described.
Chapter 7 deals with the development of some GUIs (Graphical User Interface).

Chapter 8 concerns the Simulink virtual Model developed by the UCC student Milind Sudhir Rokade.
Chapter 9 concludes the thesis and describes possible future developments.
Appendix A includes part of the code developed in the project.

1.1 UCC SCARA robots project

Some work has been done on the Matlab control of industrial robots [10, 12, 13], however the overall project in which this thesis is involved makes the Matlab control the base for further interesting developments.
The mechatronic laboratory of the UCC (*University College Cork*) Electrical and Electronic Engineering Department was provided with four SCARA Sankyo SR8408 robots in September 2013. Under the supervision of Dr. Richard Kavanagh a research project has been undertaken. This project has many potential tasks:

- Task 1: develop an interpeter for controlling the robot through Matlab
- Task 2: design a Simulink virtual model of the robot, so that its motion can be compared with the motion of the actual robot and it can be used as a training method;
- Task 3: develop a vacuum gripping system in order to allow the use of a pick and place end effector (vacuum based);
- Task 4: design a gripper (pneumatic-based) so that the robot can pick up a part.
- Task 5: construct a conveyor-based work cell to demonstrate the operation of a SCARA-based work-cell;
- Task 6: design a software/hardware based system so that the robot can be controlled remotely (via the Internet);
- Task 7: implement a camera-based part identification algorithm so that image processing routines can be developed to control the robot/gripper;
- Task 8: use two robots to operate cooperatively on a task.

This thesis concerns Task 1, Task 3, and Task 7. Another student, with whom I collaborated, is working on Task 2. A brief overview of that work is shown in chapter 8.

1.2 Objectives of the thesis

The objectives of this thesis are:

- develop an interpreter written in the robot native programming language, in order to control the robot through Matlab;
- develop Matlab functions that allow robot control. Some of these functions are in one to one correspondence with the native robot language functions, but also new functions has to be developed in order to increase its functionality;
- develop Matlab GUIs (Graphical User Interfaces), so that non expert users can perform some basic operation with the robot;
- mantain safety conditions for the user;
- develop a vacuum gripping system, so that the robot can pick up an object;
- develop two vision-based applications;
- provide some program examples.

1.3 Why choose Matlab?

The language provided by Sankyo (SSL/E language), is a very specific language with many limitations. For instance, the possibility of modular programming are very limited, the mathematics tools are not so powerful, and programming is quite uncomfortable. Furthermore, the robot controller has harware limitations in terms of I/0 communications ports. These aspects can be resolved if the user could program the robot from a PC where a more structured programming language like C, C++, Java, or Matlab is installed. In this case, an interpreter should perform a translation operation. As will be explained in this document, the interpreter is a program written in the robot native language (SSL/E language), which is running on the robot controller. The user who wants to control the robot, will write an operative program in Matlab that is installed on a PC of the lab.
Matlab [8, 19], is a very versatile software environment developed by MathWorks used in many fields and it is known by almost every engineering student. The reasons that led the choice of Matlab instead of other high-level programming languages are:

- easy communication with external devices via all the main communication protocols (GPIB, serial, TCP/IP, and UDP) by using the Matlab *Instrument Control Toolbox* functions;
- easy implementation of GUIs (Graphical User Interfaces);

- possibility of developing a virtual model of the robot by using the Matlab integrated software *Simulink*;
- easy image and video acquisition and processing by using the Matlab *Image Acquisition Toolbox* and *Image Processing Toolbox*;
- control, simulation, and visual control integrated in the same software;
- Matlab Help, MathWorks on-line support, and many examples of code on the Internet make Matlab programming suitable for didactic applications with the robot.

Chapter 2

The SCARA Robot

2.1 Introduction

An *industrial robot* is a mechanical device that can be programmed to perform a variety of tasks of manipulation and locomotion under automatic control [2]. The industrial robots can be classified in five typologies [1, 2]:

- cartesian robot;
- cylindrical robot;
- spherical robot;
- SCARA robot;
- parallel robot.

The SCARA robot was introduced in Japan in 1979 [2, 3] and has since been adopted by numerous manufacturers. In Figure 2.1 the Hirata AR-300, one of the first SCARA robot, is shown. In the 1980's the SCARA Robot contributed largely to the flexibility and efficiency of Japanese assembly systems, due to its adaptability and functionality with its comparative decrease in overall production costs vis-a-vis competitors [3]. The prices of products decreased and in particular electronic products became more affordable in a worldwide market place.
Despite the continuous evolution in robotics, the SCARA is still a very widely used machine with widespread applications. The success of this robot was possible due in the main to the following factors:

- precision;
- high speed, due to its light structure;
- small dimensions;

- smooth motion;
- simple and reliable structure;
- ease of installation and use;
- very small backlash.

This robot is used in different sizes in all kinds of industries such as automotive, electronics, and pharmaceutical. The most common applications are:

- pick and place operations;
- assembling products;
- palletizing;
- packaging applications.

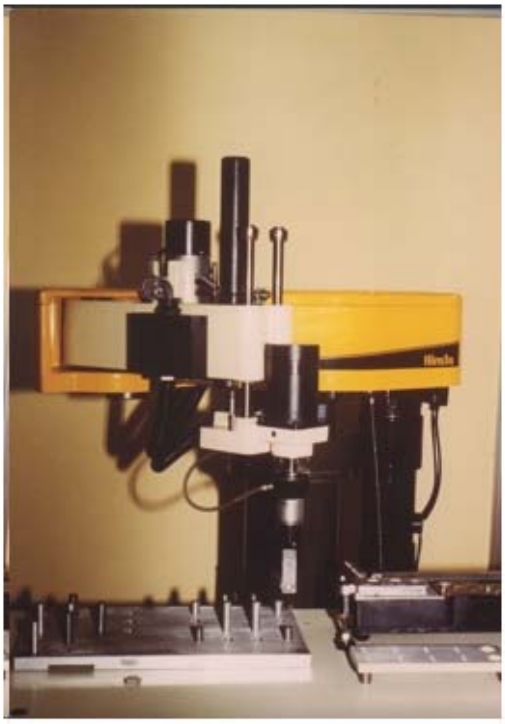


Figure 2.1: The Hirata AR-300, one of the first model of a SCARA.

2.2 Structure and Mathematical Analysis

Each company produces SCARA robots with different features, but the basic structure is pretty much the same. It has similarities to a human arm with a shoulder,

an elbow, and a wrist. The two links and 4 axes structure allows four degrees of freedom. Two parallel rotary joints and a linear vertical joint allow freedom in the X-Y-Z space. The fourth degree of freedom is given by the rotational motion of the end effector along the vertical axis. A heavy base is used to make the structure stable. See figure 2.2.

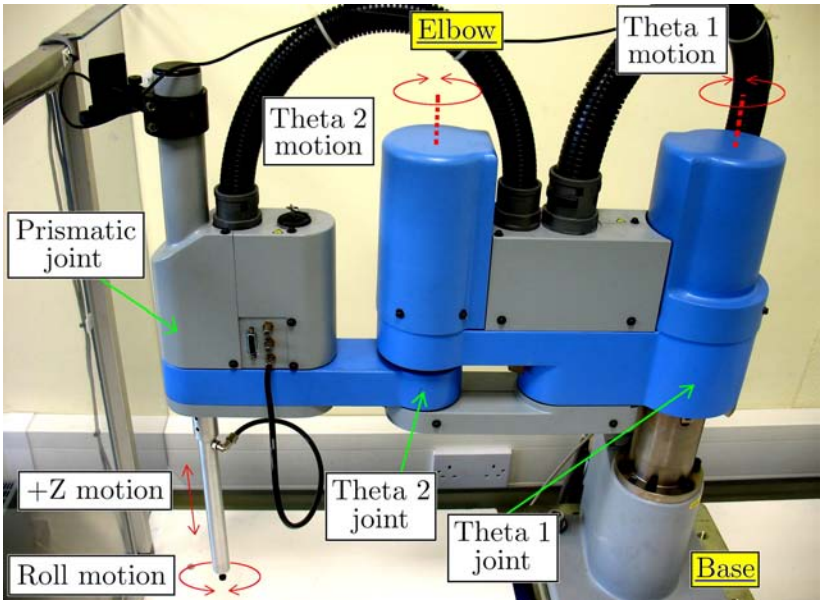


Figure 2.2: SCARA Sankyo SR8408.

2.2.1 Forward Kinematics

The aim of the *Forward Kinematics* [4, 5, 6, 22] is to determine the position and orientation of the end effector in reference to the main frame (base frame). Underlining the fact that the *Joint Axis* for a rotational joint and for a linear joint are respectively around the axis of rotation and along the positive direction of motion, one can introduce four *Kinematic Parameters*. The relative position and orientation of a link joint axes is specified by two *Link Parameters*: the *Link Length* (a_i) and the *Link Twist* (α_i). In detail:

- a_i is the common normal distance between the joint axes, measured from the axis of joint i to axis of joint $i+1$;
- α_i is the angle by which axis i must be twisted to bring it into alignment with axis $i+1$ when looking along a_i. It is assumed the sign of the angle corresponds to "clockwise positive".

The relative position and orientation of a link referring to the successive link is specified by two *Joint Parameters*: the *Joint Distance* (d_i) and the *Joint Angle* (Θ_i).

Detail:

- d_i is the distance between the two normals a_{i-1} and a_i, measured along the joint axis from a_{i-1} to a_i;
- Θ_i is the angle from a_{i-1} to a_i in a plane normal to the joint axis.

An example of the four Kinematic Parameters is shown in Figure 2.3. In order to determine these parameters the Denavit and Hartenberg (D-H) representation is used [4, 5]. Basically, according to this representation a link frame for each link has to be assigned. Let n be the number of links and L_k the frame of the *Link* k with $0 \leq k \leq n$. The rules that must to be followed are:

1) the z_k-*axis* is in the direction of the *joint axis*;
2) the x_k-*axis* is parallel to the common normal: $x_k = z_{k-1} \times z_k$. The direction of the x_k-*axis* is from z_{k-1} to z_k;
3) the y_k-*axis* follows from the x- and z-*axis* by choosing it to be a right-handed coordinate system.

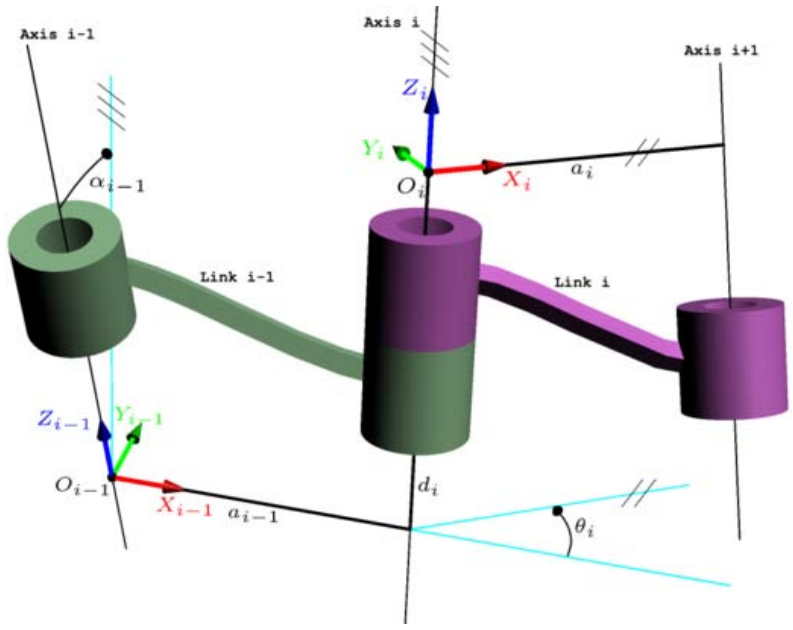


Figure 2.3: Example of the four Kinematic Parameters θ_i,d_i,a_i,α_i. With those four parameters, the coordinates can be translated from O_i to O_{i-1}.

Once all the frames are positioned, the parameters a_k, α_k, d_k and Θ_k are calculated for $0 \leq k \leq n$. The positive aspect of this approach is that transformations between successive frames are represented by a simple 4×4 matrix, with the same structure for each transformation.

In the case of simple serial links, the matrix that relates the *Link k* to the *Link k-1* is:

$$T_{k-1}^{k} = \begin{bmatrix} \cos\Theta_k & -\sin\Theta_k\cos\alpha_k & \sin\Theta_k\sin\alpha_k & a_k\cos\Theta_k \\ \sin\Theta_k & \cos\Theta_k\cos\alpha_k & -\cos\Theta_k\sin\alpha_k & a_k\sin\Theta_k \\ 0 & \sin\alpha_k & \cos\alpha_k & d_k \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.1}$$

The placement of frames on the SCARA robot is shown in figure 2.4. Due to the structure of the SCARA, only four parameters are variable: $\Theta_1, \Theta_2, \Theta_4, d_3$. The parameter values are listed in the table 2.1.

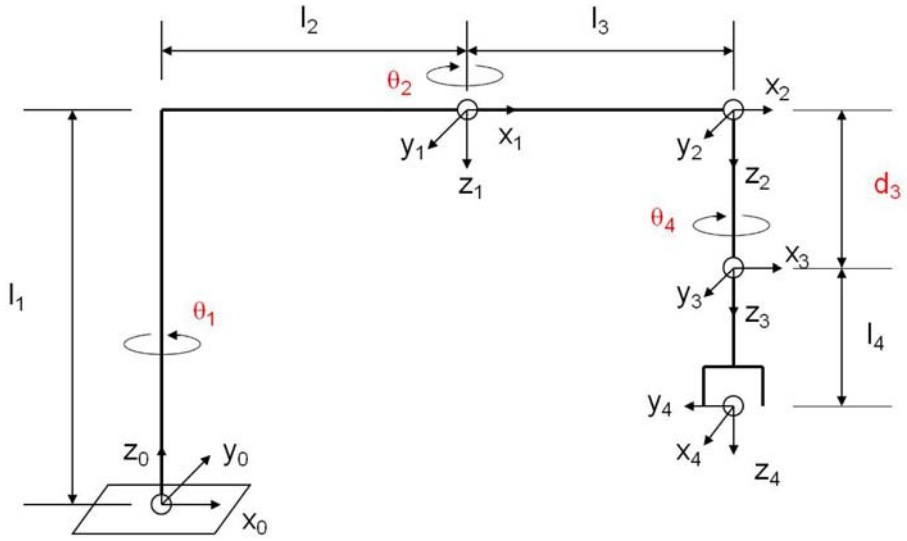


Figure 2.4: SCARA frames placement.

Table 2.1: Link and joint parameters.

Axis	Θ	d	a	α	Home
1	Θ_1	l_1	l_2	180°	0°
2	Θ_2	0	l_3	0°	0°
3	0°	d_3	0	0°	d_{max}
4	Θ_4	l_4	0	0°	90°

The matrix relating tool position to the base frame is T_{base}^{tool}:

$$T_{base}^{tool} = T_0^1 \, T_1^2 \, T_2^3 \, T_3^4 \tag{2.2}$$

with

$$T_0^1 = \begin{bmatrix} \cos\Theta_1 & -\sin\Theta_1 & 0 & l_2\cos\Theta_1 \\ \sin\Theta_1 & -\cos\Theta_1 & 0 & l_2\sin\Theta_1 \\ 0 & 0 & -1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1^2 = \begin{bmatrix} \cos\Theta_2 & -\sin\Theta_2 & 0 & l_3\cos\Theta_2 \\ \sin\Theta_2 & \cos\Theta_2 & 0 & l_3\sin\Theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^4 = \begin{bmatrix} \cos\Theta_4 & -\sin\Theta_4 & 0 & 0 \\ \sin\Theta_4 & \cos\Theta_4 & 0 & 0 \\ 0 & 0 & 1 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Finally:

$$T_{base}^{tool} = \begin{bmatrix} \cos\Theta_{1-2-4} & \sin\Theta_{1-2-4} & 0 & l_2\cos\Theta_1 + l_2\cos\Theta_{1-2} \\ \sin\Theta_{1-2-4} & -\cos\Theta_{1-2-4} & 0 & l_2\sin\Theta_1 + l_3\sin\Theta_{1-2} \\ 0 & 0 & -1 & l_1 - d_3 - l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

where

$$\Theta_{1-2-4} = \Theta_1 - \Theta_2 - \Theta_4 \tag{2.4}$$

$$\Theta_{1-2} = \Theta_1 - \Theta_2. \tag{2.5}$$

Inverse Kinematics

Often the matrix T_{base}^{tool} and the approach vector of the tool are known. The aim of the *Inverse Kinematics* [6, 22] calculation is to find the values of the variable parameters that allow a specific position to be reached. In the case of the SCARA, the parameters are $\Theta_1, \Theta_2, \Theta_4, d_3$. The approach vector of this robot is $(0, 0, -1)^T$,

which means that the approach direction of the end effector is always straight down.

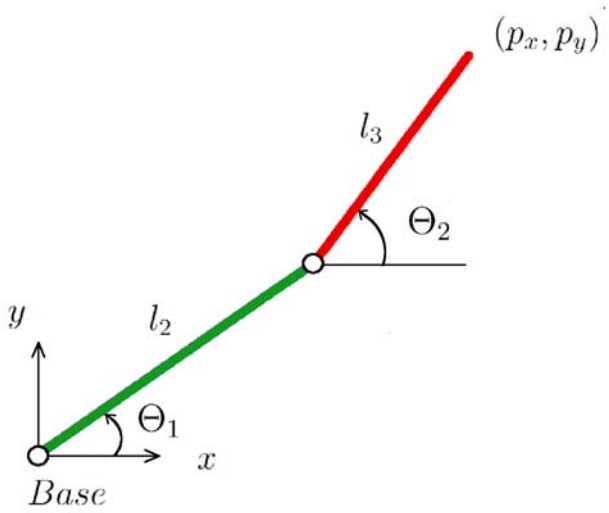


Figure 2.5: SCARA top view scheme.

Given the numerical matrix:

$$T_{base}^{tool} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & p_x \\ R_{21} & R_{22} & R_{23} & p_y \\ R_{31} & R_{32} & R_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.6}$$

the value of d_3 can be easily found:

$$d_3 = l_1 - l_4 - p_z. \tag{2.7}$$

To find Θ_2:

$$\begin{aligned} p_x^2 + p_y^2 &= l_2^2 + l_3^2 - 2l_2l_3\cos(180° - \Theta_2) \\ &= l_2^2 + l_3^2 + 2l_2l_3\cos\Theta_2 \end{aligned} \tag{2.8}$$

.

Because p_x and p_y are known, it is obtained:

$$\Theta_2 = \pm\arccos\Big(\frac{p_x^2 + p_y^2 - l_2^2 - l_3^2}{2l_2l_3}\Big). \tag{2.9}$$

The two possible solutions (one positive and one negative) are coherent with the fact that the robot can reach the target point in the right arm mode (positive solution)

and in the left arm mode (negative solution).
To find Θ_1:

$$p_x = l_2C_1 + l_3C_{1-2} = (l_2 + l_3C_2)C_1 + l_3S_2S_1 \tag{2.10}$$

$$p_y = l_2S_1 + l_3S_{1-2} = -l_3S_2C_1 + (l_2 + l_3C_2)S_1 \tag{2.11}$$

$$\begin{bmatrix} C_1 \\ S_1 \end{bmatrix} = \begin{bmatrix} l_1 + l_2C_2 & l_2S_2 \\ -l_2S_2 & l_1 + l_2C_2 \end{bmatrix}^{-1} \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{2.12}$$

$$\Theta_1 = \arctan 2(\quad l_2S_2p_x + (l_1 + l_2C_2)p_y \quad , \quad (l_1 + l_2C_2)p_x - l_2S_2p_y \quad) \tag{2.13}$$

If it is necessary Θ_4 can be found:

$$R_{21} = S_{1-2-4} \tag{2.14}$$

$$R_{11} = C_{1-2-4} \tag{2.15}$$

$$\begin{aligned} \Theta_4 &= \Theta_1 - \Theta_2 - \Theta_{1-2-4} \\ &= \Theta_1 - \Theta_2 - \arctan 2(R_{21}, R_{11}). \end{aligned} \tag{2.16}$$

Chapter 3

The Sankyo SR8408

In this chapter, the structure, the main features and components present on board the Robot SR8408, produced by The NIDEC SANKYO Corporation are described [34]. The main operative features of this robot are listed in Table 3.1.

Table 3.1: Main operative features of Sankyo SR8408.

Arm length	Total	550 mm
	First arm	300 mm
	Second arm	250 mm
Operative area	Θ_1	± 120 °
	Θ_2	± 120 °
	Z axis travel	150 mm
	Θ_4	± 360 °
Maximum speed	Composite speed	5000 mm/s
	Z axis travel	1000 mm/s
	Rotational speed	730 °/s
Repeatability	X-Y	0.1 mm
	Z	0.02 mm
	Rotation	0.05°
Load-carrying capacity		3 Kg
Max couple		3 Nm
Weight		40 Kg

3.1 Mechanical structure

Only functional parts pertaining the motion are described in this section.
The base of the robot is shown in Figure 3.1. It allows the fixing of the robot in a safe and stable way.

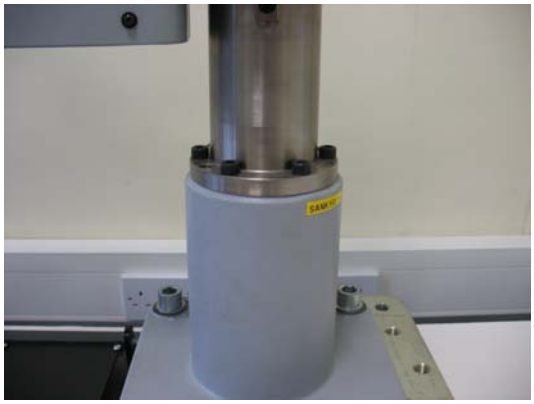


Figure 3.1: Base of the robot.

Each degree of freedom of the robot is driven by its own servo motor. Two motors are fixed on the rotational joints and allow the user to change the values of Θ_1 and Θ_2. The roll motion motor along the Z axis is fixed inside the first arm. This position makes the centre of mass of the entire arm closer to the first rotational joint, and gives space to the motor positioned at the edge of the arm, that moves the prismatic joint. In the table 3.2 all the indicated parts of Figure 3.2 are listed. The position of some important components listed in the tables 3.3 and 3.4 is shown in Figure 3.3 and 3.4 respectively.

Table 3.2: Item list for Figure 3.2.

Item	Parts name
A1	Θ_1 motor
B1	Θ_2 motor
C1	Z axis motor
D1	Roll axis motor
E1	Θ_1 harmonic drive (inside)
F1	Θ_2 harmonic drive (inside)
G1	Roll axis reduction gear unit
H1	Z axis shaft
I1	Z axis brake unit
J1	Connector panel
K1	Flexible cable hose connection and serial port
L1	Θ_1 arm
M1	Θ_2 arm
N1	Z axis pulley
O1	Z axis belt

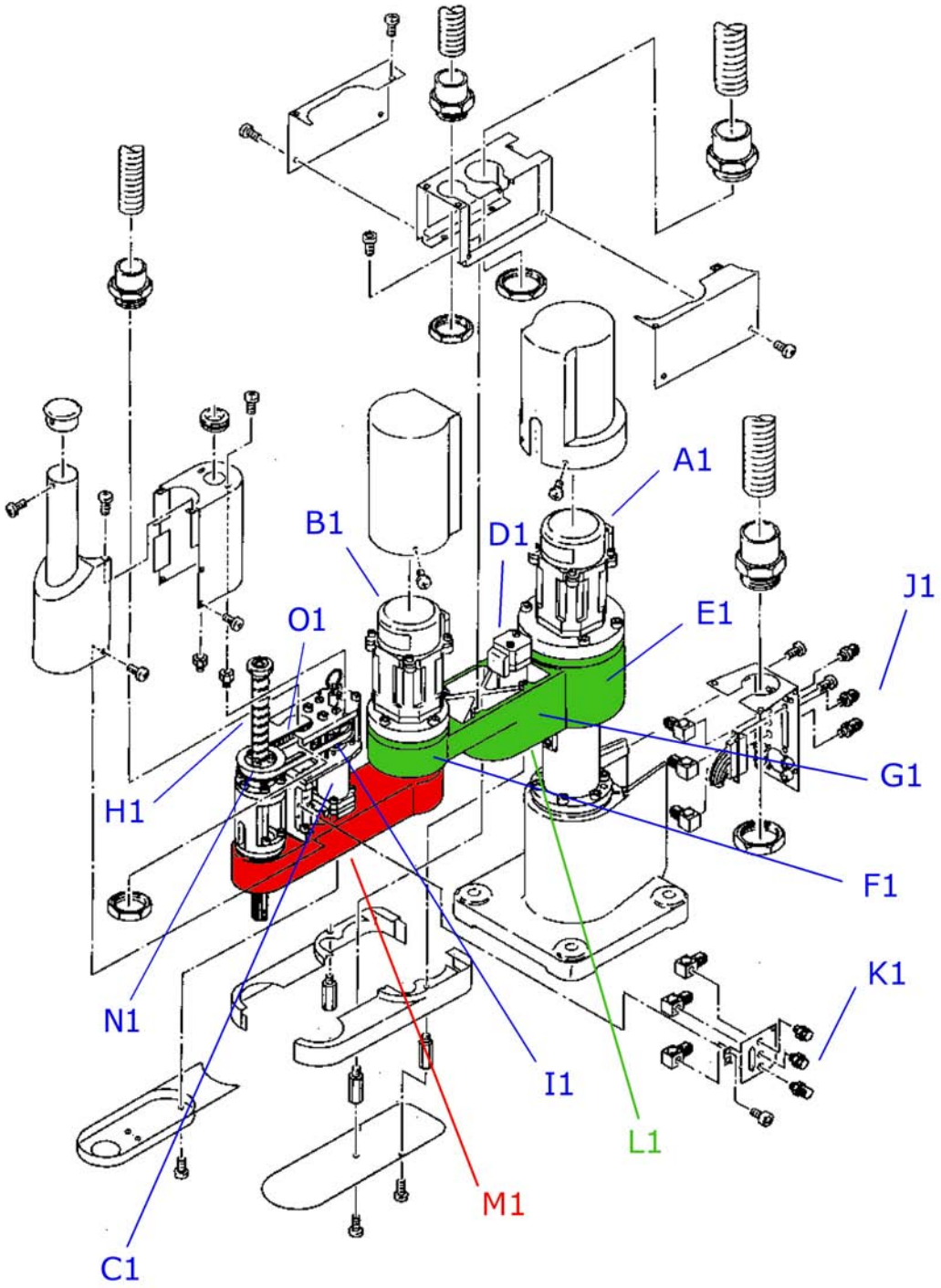


Figure 3.2: General assembly.

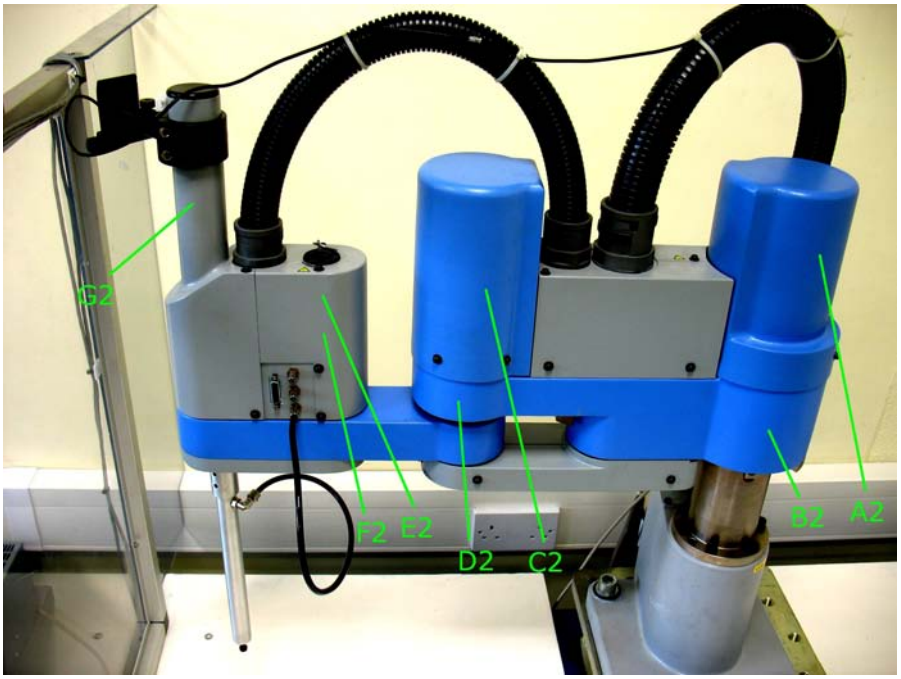


Figure 3.3: Showing the Robot with the covers in place.

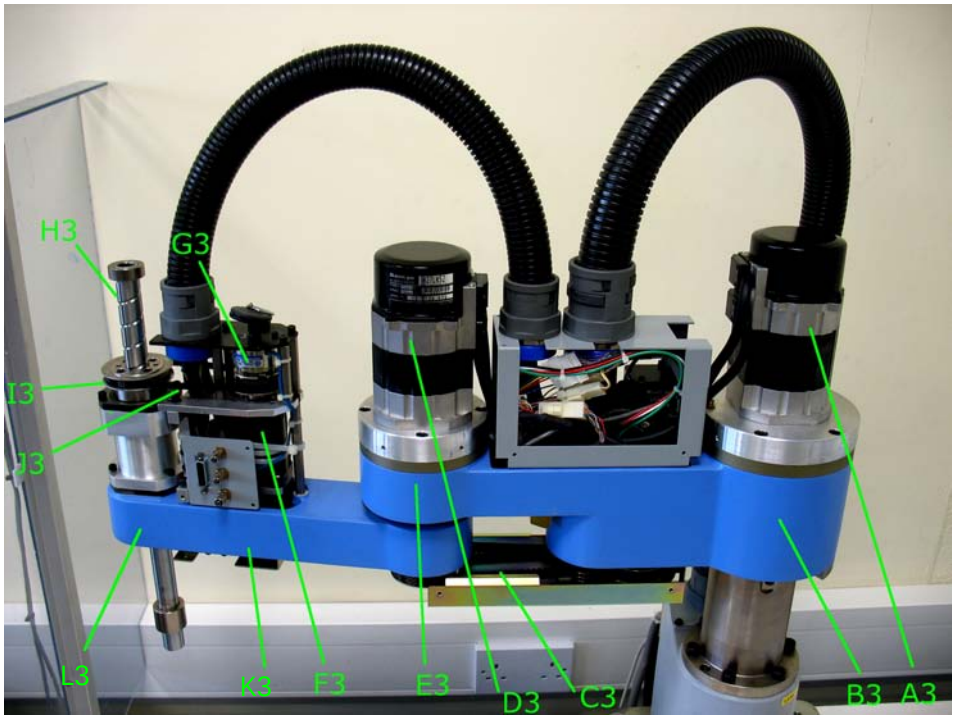


Figure 3.4: Robot partially uncovered.

Table 3.3: Item list for Figure 3.3.

Item	Parts name
A2	Θ_1 motor
B2	Θ_1 harmonic drive (inside)
C2	Θ_2 motor
D2	Θ_2 harmonic drive (inside)
E2	Z axis brake unit
F2	Roll axis motor
G2	Z axis shaft

Table 3.4: Item list for Figure 3.4.

Item	Parts name
A3	Θ_1 motor
B3	Θ_1 harmonic drive
C3	Roll axis belt 1
D3	Θ_2 motor
E3	Θ_2 harmonic drive
F3	Roll axis motor
G3	Z axis brake unit
H3	Z axis shaft
I3	Z axis pulley
J3	Z axis belt
K3	Roll axis belt 2 (inside)
L3	Roll axis pulley

3.2 Motors and motion transmission

Rotational joints motion

Both Θ_1 and Θ_2 motors are AC servo motors. See table 3.5 for their specifications.

Table 3.5: Θ_1 and Θ_2 motor specifications.

Motor	Power (W)	Speed (rpm)
Θ_1	366	4000
Θ_2	267	4000

The two motors are shown in Figure 3.5. In Figures 3.6, 3.7, and 3.8, Θ_2 motor is shown in detail. As it can be seen in Figure 3.6(b), it has two wire connections: a connection with four pins that provides power (LINE 1, Line 2, N/C, GROUND), and another connection for powering the encoder and for acquiring signals from it. In Figure 3.9 the electronic board attached to the encoder is shown .

Harmonic drive

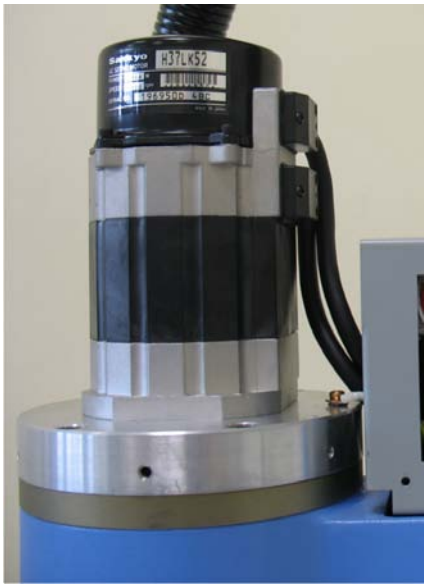
Θ_1 and Θ_2 joints employ a *harmonic drive* in order to increase the torque delivered by the servo motor. Developed over 50 years ago, primarily for aerospace applications, harmonic drives are compact transmission systems which increase torque of electric motors [7]. They are reduction drive with very low backlash, compactness, good resolution, excellent repeatability, and high torque capability. It allows a very smooth motion and it is made up of three main components: the *Circular Spline*, the *Wave Generator*, and the *Flexspline*. See Figure 3.10. The Circular Spline is a rigid steel ring with teeth on the inner surface. The Flexspline is a steel cylinder with flexible walls with teeth, but a quite rigid closed side. It is fixed to the load. The Generator is a thin elliptical ball bearing assembly, fixed to the rotor of the motor. To understand how the Harmonic Drive works, please see Figure 3.11. The zone of the tooth Wave engagement between the Flexspline and the Circular Spline moves with the Wave Generator major axis. The Flexspline has normally two teeth less than the Circular Spline due to its shorter diameter. Because of that, when the Wave Generator has turned 180 deg clockwise, the Flexspline has regressed by one tooth relative to the Circular Spline. After a complete revolution of the Wave Generator, the Flexspline has regressed by two teeth relative to the Circular Spline.

Roll motion and Z motion

The roll motion along the Z axis is activated by a servo motor inside the Θ_1 arm (118 W, 4000 rpm). Two belts and two pulleys ensure the motion transmission up to the Z axis. See Figures 3.12, 3.13. The prismatic joint is driven by another servo motor (118 W, 4000 rpm) through a pulley and a belt. An *electromagnetic clutch* is used as a break unit. See Figures 3.14, 3.15.

3.2.1 Closed loop control and repeatability

The Sankyo SR8404 is controlled by the SC3150 Controller produced by NIDEC SANKYO corporation. This Controller uses the ABS (absolute) encoders backed up by battery. Therefore, the *Home position operation* doesn't have to be carried out each time the robot is powered because the positional data is stored in the encoders back-up memory. During the Home position operation, this position is detected by 4 *Home sensors*. The mechanical structure and the feedback control allow a repeatability of 0.1 mm in the X-Y plane, 0.02 mm in Z positioning, and 0.05° in rotation.



(a) Θ_1 *motor.*

(b) Θ_2 *motor.*

Figure 3.5: Θ_1 and Θ_2 motors.

Figure 3.6: Θ_2 motor.

(a) Θ_2 *motor.*

(b) *Connectors.*

Figure 3.7: Θ_2 motor and connectors.

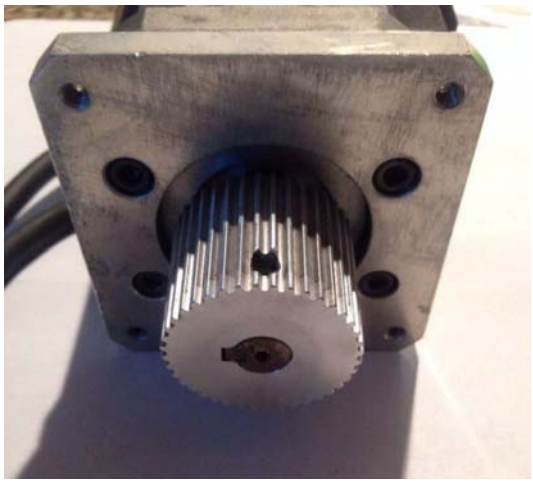


Figure 3.8: Rotor of Θ_2 motor.

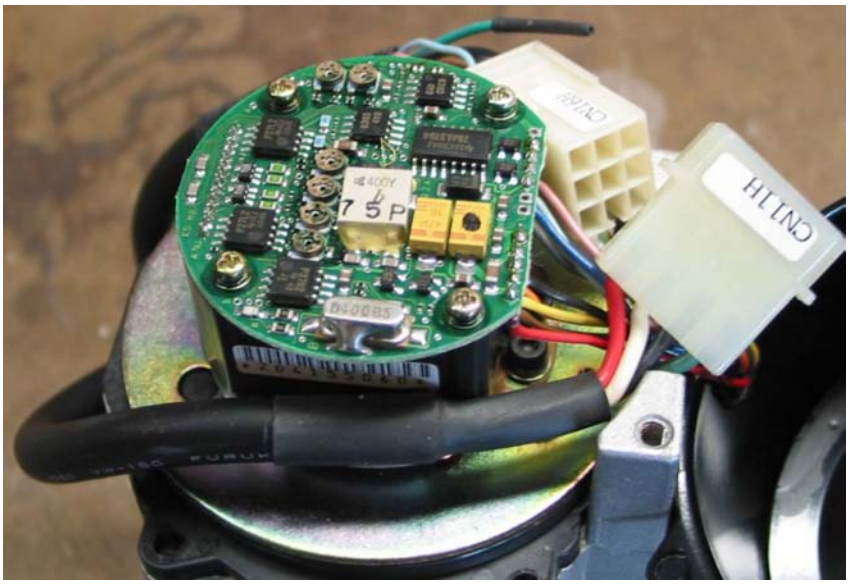
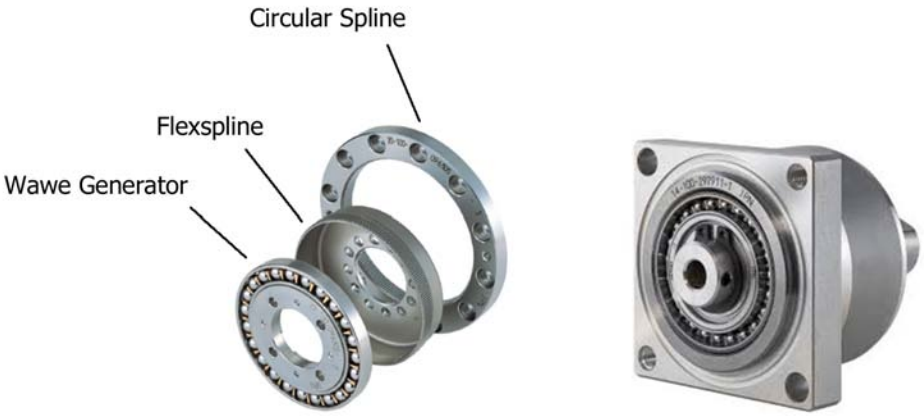


Figure 3.9: Encoder electronic board.



(a) *Disassembled Harmonic Drive.* (b) *Assembled Harmonic Drive.*

Figure 3.10: Harmonic Drive.

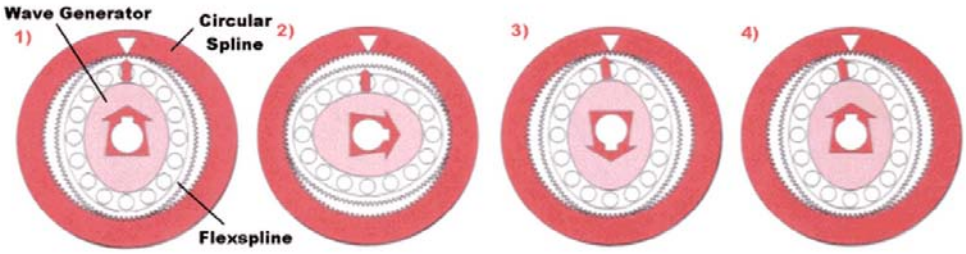


Figure 3.11: Harmonic Drive functioning.

Figure 3.12: Roll motor.

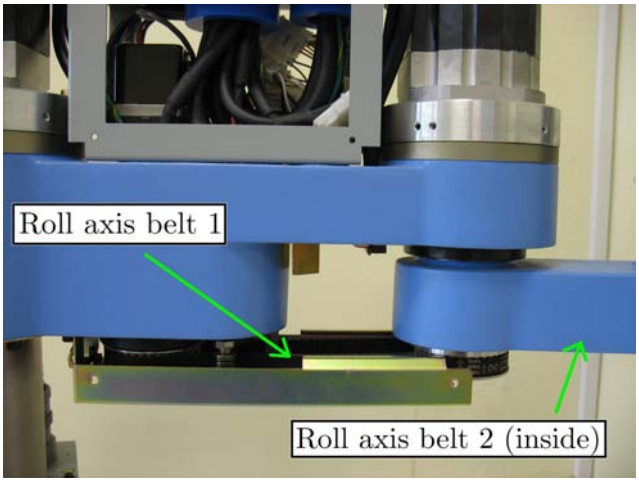
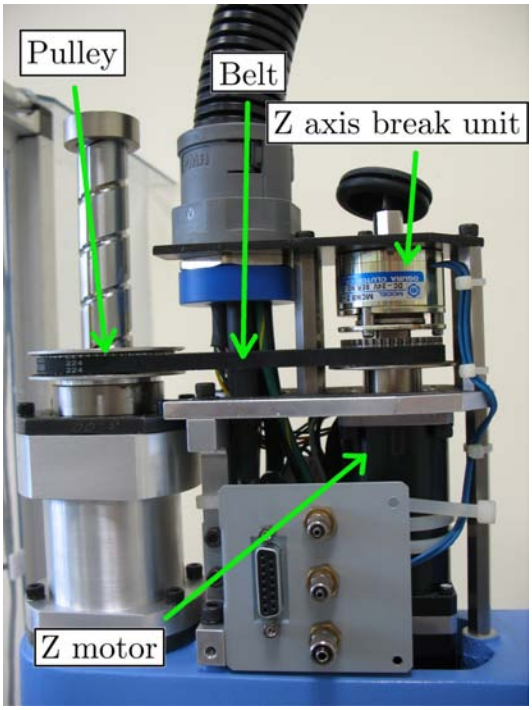


Figure 3.13: Roll motor belts.

(a) *Z motor and electromagnetic clutch.*

(b) *Z motor.*

Figure 3.14: Z motor and electromagnetic clutch.



(a) *Z axis motion unit and break unit.*

(b) *Electromagnetic clutch.*

Figure 3.15: Z axis motion unit and break unit.

Chapter 4

HW configuration and Interpreter

4.1 Hardware configuration

The robot manufacturer, the Sankyo Corporation, provides a robot controller and a programming language called *SSL/E Language* (*Sankyo Structured Language/Enhanced*). The controller includes a CPU board, the power electronics for driving the arm motors, the control electronics for managing the feedback loop, a mother board, some digital I/O ports, and two serial ports. Since the robot controller supports only the language provided by Sankyo (SSL/E language), an interpretation is necessary in order to convert a Matlab robot function in a SSL/E function.
The Hardware Configuration used is shown in Figure 4.1. Matlab is installed on a PC connected to the SC3150 Controller, through a RS232 cable. The Interpreter, as it will be explained, is a program written in the *SSL/E Language* (*Sankyo Structured Language/Enhanced*) and runs on the controller. The controller is connected to the Robot in order to provide power to the motors and to the ABS encoders, and to receive the encoder position feedback signals. The Teaching Pendant OP3000 allows many operations, but most importantly the operator can start and stop the Interpreter execution by using it.

4.2 Interpreter

4.2.1 Matlab functions: distinction into families

In order to make clear the content of the next sections, Matlab functions are divided in three families:

- *Matlab Native Commands and Functions*: these commands and functions are

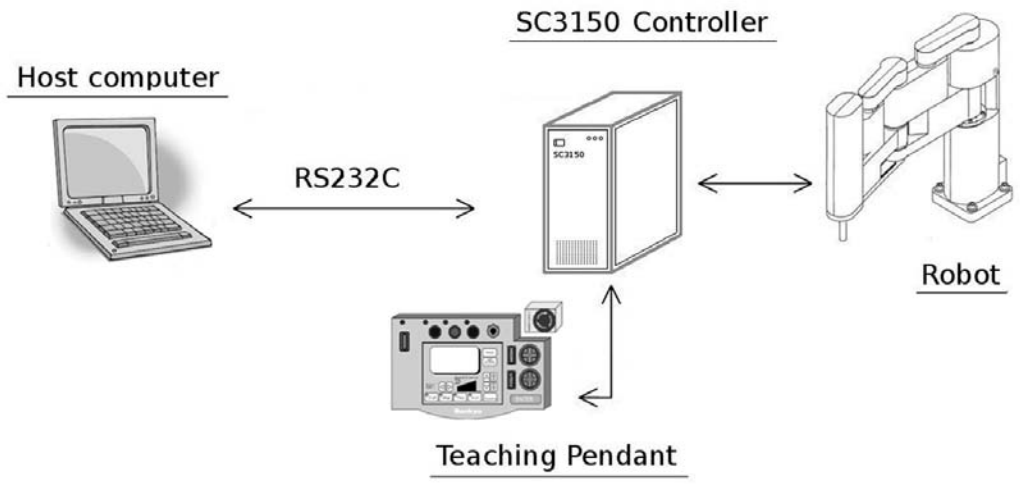


Figure 4.1: Hardware configuration.

provided by Matlab itself;

- *Matlab Robot Functions*: these functions have been developed in this project and are provided to the user without the possibility to see the inner code. They allow Robot control;

- *Matlab Application Functions*: these functions have been developed in this project and concern two applications. See chapter 6;

- *Matlab Auxiliary Functions*: these functions have been developed in this project. They carry out crucial operations to allow the correct execution of Matlab Robot Functions.

4.2.2 Programming with and without Matlab

Sankyo provides a Robot Application Development Software named *Buzz2*, that supports the writing, compiling or building, editing, monitoring and debugging of the user application programs for the Sankyo SC3000 series Robot Controllers. Thus, without the Matlab Interpreter developed in this project, the user has to write a program in Buzz2 and download it to the controller. The Task can be started from the Pendant or entering in the Buzz2 Debug Mode. See the diagram in Figure 4.2. On selecting the Interpreter, the user, can write a program in Matlab by using Matlab native commands, and functions that this project has made available. Matlab also allows debugging and variables monitoring. A function can be launched from the *Command Window*, this allows a very quick check about the effect of the function itself. See the diagram in Figure 4.3. However, a program is usually written

as a *Script*. The Listing 4.1 is an example of a simple application written in Matlab. It is for picking up 6 pieces individually, and moving them to a different position.

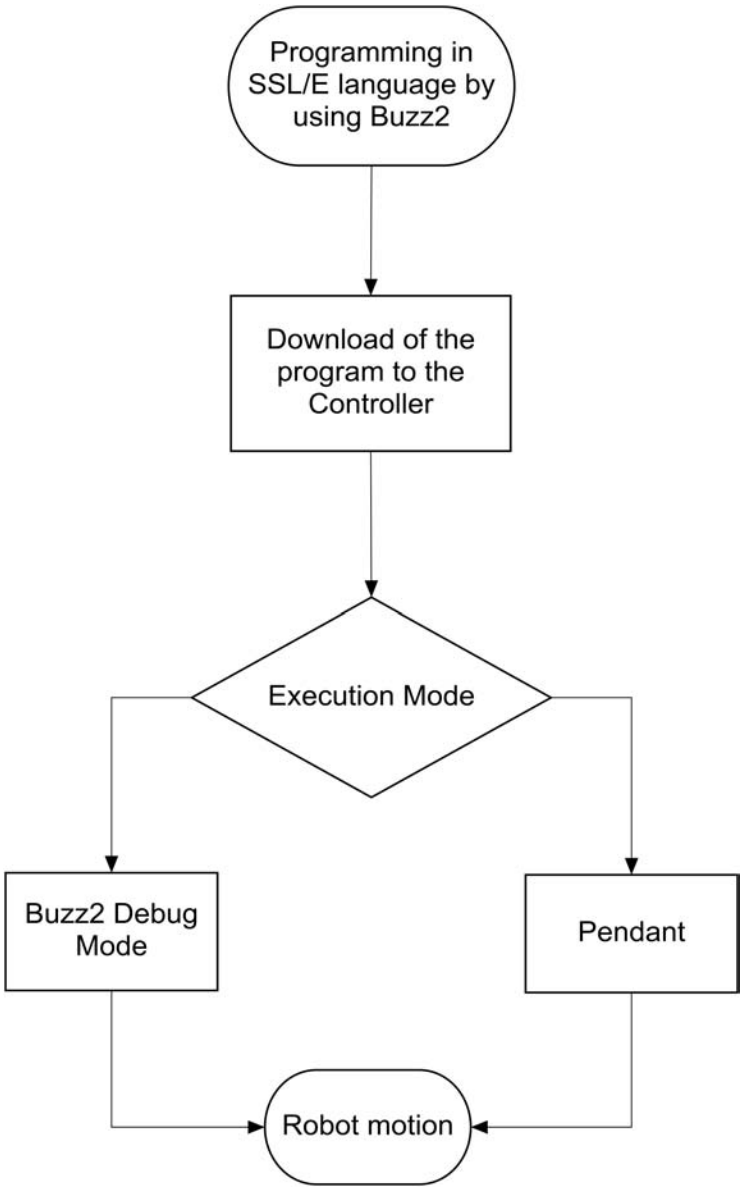


Figure 4.2: Programming, and program execution in Buzz2.

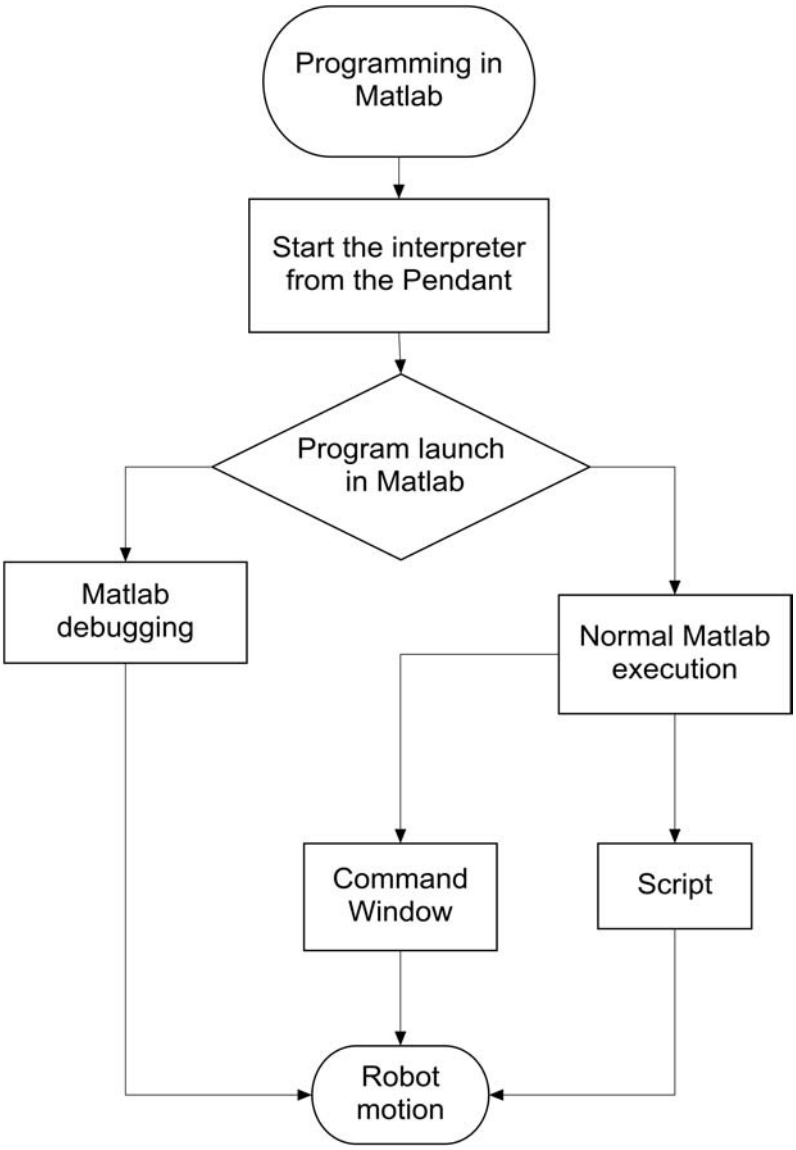


Figure 4.3: Programming, and program execution with Matlab.

Listing 4.1: Matlab program: six items are moved from position P1 to a position P2.

```
speed(4);  % Sets the speed (4% of the maximum speed)

PIECE_POS=[280,280,10,112];  % Cartesian Position of a piece
                             % [X (mm),Y (mm),Z (mm), rotation
                             % along the Z-axis (deg)]

RELEASE_POS=[280,280,10,112];    % Cartesian Position of the
                                 % release position [X (mm),
                                 % Y (mm),Z (mm), rotation
                                 % along the Z-axis (deg)]

```

```
12 move(PIECE_POS);  % Moves to PIECE_POS
13
14 i=1;    % Iteration variable
15
16 % Cycle for picking up 6 pieces in the workspace (It has
17 % been considered all the pieces in the same position)
18
19 while(i<7)
20
21    move(PIECE_POS);  % Moves to PIECE_POS
22
23    out(937,1);       % Activates vacuum device in order
24                      % to pick up a  piece by using a sucker
25
26    smove(3,60);  % Moves only the third axis (z-axis) straight
27                  % down in order to reach the piece
28
29    smove(3,-60);  % Moves only the third axis (z-axis)
30                   % straight up
31
32    move(RELEASE_POS);  % Moves to RELEASE_POS
33
34    out(937,0);  % Deactivates vacuum device, and releases
35                 % the piece
36
37    pause(0.2);  % Delay for allowing piece release (s)
38
39    i=i+1;       % Iteration variable updating
40
41 end
```

4.2.3 The Interpreter: what it is, and how it works

Actually, what has been developed is not exactly a true interpreter. Indeed, it does not translate a Matlab program to a SSL/E program. An example is used in order to explain this. A simple Matlab program is considered. See Listing 4.2. This program, after defining two positions in the workspace (P1 and P2), and setting the speed to the 10% of the maximum speed, moves the robot over P1 and P2 waiting one second after positioning.

Listing 4.2: Matlab code for point to point motion.

```
1 P1=[280,280,10,112];     % Cartesian Position of a piece
2                          % [X (mm),Y (mm),Z (mm), rotation
3                          % along the Z-axis (deg)]
4
5 P2=[-280,280,10,112];    % Cartesian Position of a piece
6                          % [X (mm),Y (mm),Z (mm), rotation
7                          % along the Z-axis (deg)]
8
9 speed(10);  % Sets the speed (10% of the maximum speed)
10
11 i=1;        % Iteration variable
```

```
12
13 while(i<11)
14
15    move(P1);  % Moves to P1
16
17    pause(1);  % Waits 1 second
18
19    move(P2);  % Moves to P2
20
21    pause(1);  % Waits 1 second
22
23    i=i+1;     % Iteration variable updating
24
25 end
```

Moving inside the function `move` function that performs point to point motion, the function `serial_out1` is called.

```
1 function [] = move( A )
2
3 % This function performs point to point motion
4
5 x=serial_out1(1000,A);
6
7 end
```

This function is extremely important. It has two input parameters: the first one is the number 1000, the unambiguous code that identifies the function `move`, the second one is the argument of the function `move`, i.e. a generic position A which is a vector of four numbers. The function `serial_out1` sends the code and the parameter through a serial cable to the controller, on which the "Interpreter" runs. After that, Matlab waits for the *Feedback Execution Confirmation Code* from the controller that confirms the correct execution of the statement by the Interpreter. Then, the Matlab program continues with the next statements.

What needs to be understood, is that this operation involves only a set of functions made available to the user. These functions will be called *Matlab Robot Functions*. They have been developed during this project and their inner code is not accessible to the user. It's easy to understand that, in the example of Listing 4.2, only the function `move` and the function `speed`, that are Matlab Robot Functions, are interpreted by the interpreter on the controller.

Therefore, the "Interpreter" is a program in the Robot programming language (SSL/E). It associates a Matlab robot function with a SSL/E function. It is a black box for the Matlab programmer. It is downloaded to the controller only once, and it doesn't get changed.

The programmer can use every Matlab Native Command. When a Matlab robot function (such as `move` or `speed`) occurs in the program flow, Matlab sends its code and argument(s) to the controller that executes the corresponding SSL/E Function. Then, the Matlab program execution continues with the next statement.

The basic functioning of the Interpreter program is shown in the diagram of Figure 4.4. Once the interpreter is started from the pendant, it polls the serial port waiting data from Matlab, i.e. the code that identifies the function, and its argument(s). Then a sequence of IF statements recognises which statement has to be executed. Finally, the program starts a new polling phase after sending the *Feedback Execution Confirmation Code* to Matlab.

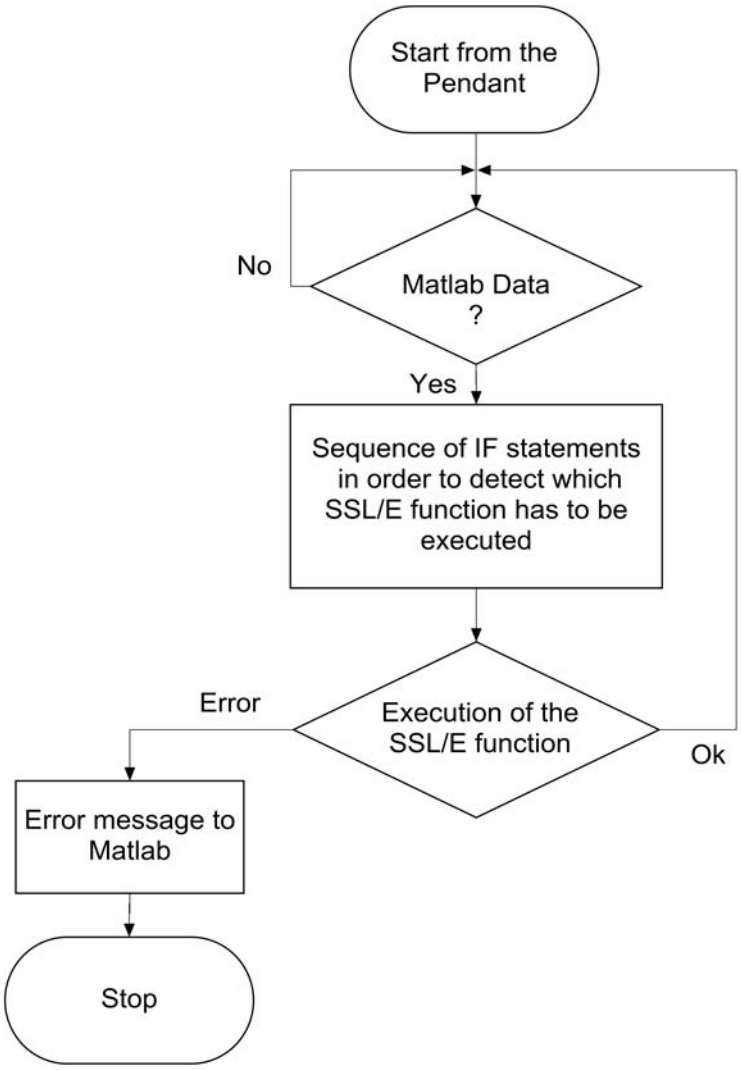


Figure 4.4: Interpreter functioning

4.3 Serial communication and synchronization

4.3.1 Serial communication

Data exchange between Matlab and the controller is made by using a RS232 cable shown in Figure 4.5. Both SSL/E language and Matlab are provided with some user-friendly statements that configure communication settings of the connection and read and send data through a RS232 serial port. See the Matlab example of Listing 4.3 and the examples in SSL/E language of Listing 4.4. For more details, refer to the Sankyo *SSL/E Reference Manual* [29] and Matlab *Communications System Toolbox Documentation* [24].

Figure 4.5: RS232 cable

Listing 4.3: Matlab program that configures communication settings of the connection and reads, and sends data through a RS232 port.

```
A=25.5;              % Real variable

S1='ABCD';             % String

S2='XYZ';              % String

s = serial('COM1');    % Creates a serial port object

% The next statement  opens the RS232  communication
% port and configures communication settings

set(s,'BaudRate',115200,'Parity','even','StopBits',2,
          'DataBits',8,'Terminator','CR/LF','Timeout',1);

fopen(s);       % Connects the RS232 port object to the device

A=num2str(A)    % Converts the integer variable A into
                % a string
```

```
fprintf(s,A);       %  Sends the string A to the RS232 port

I=fscanf(s);        %  Stores a string read from the RS232
                    %  port  into the variable I

I=str2num(I);       % Converts the string I into an integer

if(I==0)

   fprintf(s,S1);      % Sends the string S1 to the
                       % RS232 port
else

   fprintf(s,S2)    % Sends the string S2 to  the
                    % RS232 port
end

fclose(s);          % Removes the serial port object from memory

delete              % Closes the RS232 communication port

end
```

Listing 4.4: SSL/E program that configures communication settings of the connection and for reads, and sends data through a RS232 port.

```
INT I;          // Integer variable

REAL A=25.5;    // Real variable

STRING S1="ABCD";    // String

       S2="XYZ";     // String

PROG SUB()


   // The next statement opens the RS232 communication port and
   // configures communication settings
   // BAUD RATE: 115200
   // DATA LENGTH: B8
   // PARITY bits: PE
   // STOP bits: S2
   // BUFFER length (bytes): L512
   // DELIMITER characters: CRLF



   RSOPEN(1, "115200 B8 PE S2 L512 CRLF");

   RSOUT(1,A);  // Sends the string A to the RS232 port

   RSIN(1,I)    // Receives data from the RS232 port and
                // assign it into the variable I

   IF(I==0)
      RSOUT(1,S1);   // Sends the string S1 to the RS232 port
```

```
32     ELSE
33        RSOUT(1,S2);    // Sends the string S1 to the RS232 port
34
35
36     RSCLOSE(1);         // Closes the RS232 communication port
37
38  END
```

4.3.2 Synchronization

Interpreter side

As it has been shown above, the communication is possible by using a few simple statements. A simple protocol has been developed in order to synchronize Matlab and the Interpreter. In Figure 4.6 pseudocode explains the synchronization protocol between Matlab and the Interpreter from the Interpreter side. When the Start key on the pendant is pressed, the Interpreter starts running. First, it sends the *Feedback Execution Error Code* '7777' to Matlab. This provides a feedback to Matlab in case of error in the statement execution, see subsection 4.3.2 about the *Feedback Execution Error Code*. Then, a polling operation of the serial port is started. The Interpreter waits for the *Matlab Communication Initialization Code* '0000'. After receiving this code, the interpreter enters in a loop and it polls the port again waiting for the *New Matlab Robot Function Notification Code* '1111'. Once it gets this code, the interpreter reads from the port the *Function Code* of the Matlab robot function, and reads and stores the argument(s) of the Matlab robot function itself. Then, it executes the corresponding SSL/E function and returns to the polling operation of the serial port thanks to a jump statement. Before polling the port a *Feedback Execution Confirmation Code* is sent to Matlab in order to confirm the correct execution of the function.

Matlab side

After opening Matlab, the Matab user must execute the function `prog`. This function is for opening and configuring the serial port from the Matlab side. Furthermore, this function sends the *Matlab Communication Initialization Code* '0000'. As has been explained above, a Matlab robot function calls the Matlab function `serial_out1`. If `prog` hadn't been executed before executing the Matlab robot function, `serial_out1` stops the program and outputs an error message on the Matlab Command window. Otherwise, it sends the New Matlab Robot Function Notification Code '1111' to the Interpreter. Then, it sends the Function Code and the argument(s) of the Matlab robot function. After that, `serial_out1` waits for the Feedback Execution Confirmation Code.

```
        SERIAL_WRITE  (#7777) ;          // Sends the Feedback Execution
                                         // Error Code to Matlab.
                                         // This code is taken in account
                                         // only when the Intepreter is restarted
                                         // from the pendant after an Error
                                         // occurance

        X = SERIAL_READ()                // Polling of the serial port until
                                         // the Matlab Communication
                                         // Initialization Code  '0000' is read,
        WHILE ( X != # 0000 ) {          // i.e.  Matlab prog() function has been
            X = SERIAL_READ ()           // executed
        }                                //

        I=1;

LOOP :  IF(I == 0) {
            I=0
            SERIAL_WRITE  (FEEDBACK CODE)   // Sends the Feedback Execution
                                            // Communication Code to Matlab
        }                                   // in order to confirm the correct
                                            // execution of the function


        Y = SERIAL_READ()                // Polling of the serial port until
                                         // the New Matlab Robot Function
        WHILE ( Y !=  #1111 ) {          // Code '1111' is read, i.e. a new
                                         // Matlab Robot Function has been
            Y = SERIAL_READ()            // executed
        }

        RECEIVES  DATA OF                // Block of code for reading the
        THE FUNCTION                     // identification  code of the
                                         // received Matlab statement,
                                         // and for reading and storing
                                         // its  argument(s)

        FUNCTION
        EXECUTION

        JMP   LOOP                       // Jumps to LOOP
```

Figure 4.6: Pseudo code explaining the synchronization protocol between Matlab and the Interpreter from the Interpreter side.

Feedback Execution Error Code

During the robot control operations, some types of error can occur. For instance, a very frequent error is the "out of workspace" error. It occurs when the user tries to move the robot in a position out of the workspace. When an error occurs, the controller stops the program execution, switches on a LED on the pendant and outputs a message on the pendant screen. The Matlab auxiliary function `serial_out1` continues to poll the serial port waiting for feedback from the Interpreter. However the controller has stopped the execution and does not send any massage to Matlab. A manual intervention is necessary. The user has to press the Error Reset Switch on the pendant and then the Start touch key, in order to restart the Interpreter. As has been shown in the pseudocode in Figure 4.6, the first thing that the Interpreter carries out is the output of the Feedback Execution Error Code '7777'. Therefore, `serial_out1` detects that an error has occurred and, after closing the serial port object, it outputs an error message to the command window, asking the user to type the function `prog` in order to reopen and configure the serial port.
It is clear that the Feedback Execution Error Code '7777' is always outputted when the Interpreter is started. In order to allow Matlab to ignores this code when a normal start of the Interpreter is done by the user (i.e. an error condition has not occurred), the Interpreter has to be started before the Matlab robot function `prog` is executed. Indeed, in this case the serial port object is not open and configured, and thus the Feedback Execution Error Code '7777' is not taken into account.

4.4 Auxiliary Matlab Functions

4.4.1 The `startup` function

The Matlab auxiliary function `startup` is executed at Matlab startup. It includes commands that initialize important state variables.

4.4.2 The `state_keeper` function

The Matlab auxiliary function `state_keeper` is an important function that includes important `persistent` variables shared by some functions. Persistent variables are local to the function `state_keeper` itself; yet their values are retained in memory between calls to the function. `persistent` variables are similar to global variables because the MATLAB software creates permanent storage for both. They differ from global variables in that persistent variables are known only to the function in which they are declared. This prevents persistent variables from being changed directly by other functions, or from the MATLAB command line [8]. Actually, few robot functions can change and access these variables but they can't do it directly.

Indeed they have to call the function `state_keeper` with a specific string as input parameter. The function `state_keeper` compares the input string with two strings defined inside it. Depending on the string comparing result, `state_keeper` allows updating or retrieval of these variables, or denies these operations.

4.4.3 The functions serial_out

As has been shown in section 4.3.2, inside each Matlab robot function, a Matlab auxiliary function is called. It has to send the Function Code and the argument(s) of the Matlab Robot Function itself to the Interpreter. Depending on the type (scalar or matrix) and number of argument(s) that have to be sent, five different functions are used: `serial_out1`, `serial_out2`, `serial_out3`, `serial_out4`, `serial_out5`. The structure of these five functions is pretty much the same. The `serial_out1` flowchart and source code is shown in Figure 4.7 and Listing A.1 in appendix A. For *Sample mode* refer to chapter 5.10.

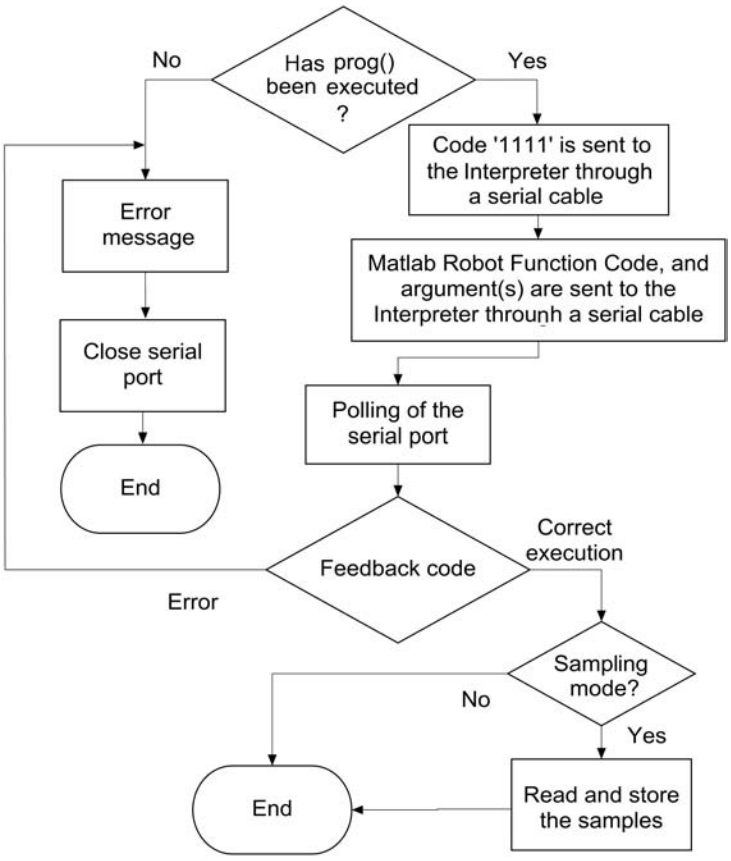


Figure 4.7: Function `serial_out1` flowchart.

Chapter 5

Matlab Robot Functions

The SSL/E language is provided with many functions for managing and converting data types, e.g. for converting number to string, for converting an integer to a real number. It is also provided with mathematical functions, and *cycle* and *if than else* constructions. All these kinds of *basic* functions have not been transposed to Matlab, since Matlab has a more complete and wider set of commands.
Seventy MATLAB robot functions have been developed. Most of them are present in SSL/E [29]. A few new statements, not provided by SSL/E, allow new functionality.

5.1 Coordinate Systems

The robot end effector position in the X-Y-Z space can be decribed by two coordinate systems: the *Cartesian Coordinate System* and the *Joint Coordinate System*. See Figure 5.2 and Figure 5.2.

Cartesian Coordinate System

In Matlab, a vector of four numbers `P1=[x,y,z,s]` is used in order to define a position in the Cartesian Coordinate System, where:

- 1^{st} element: X position (mm);
- 2^{nd} element: Y position (mm);
- 3^{rd} element: Z-axis position (mm);
- 4^{th} element: Roll/S-axis position (deg).

Example: `P1=[280,280,10,112]`

Joint Coordinate System

In Matlab, a vector of four numbers `P1=[t1,t2,z,s]` is used in order to define a position in the Joint Coordinate System, where:

- 1^{st} element: Angle of the 1st arm (deg);

- 2^{nd} element: Angle of the 2nd arm (deg);

- 3^{rd} element: Z-axis position (mm);

- 4^{th} element: Roll/S-axis position (deg).

Example: `P1=[90,15,35,220]`

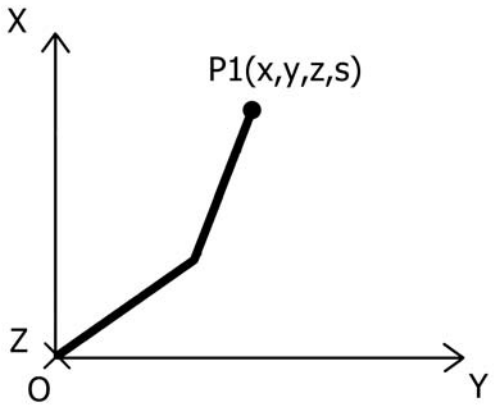


Figure 5.1: Cartesian Coordinate System.

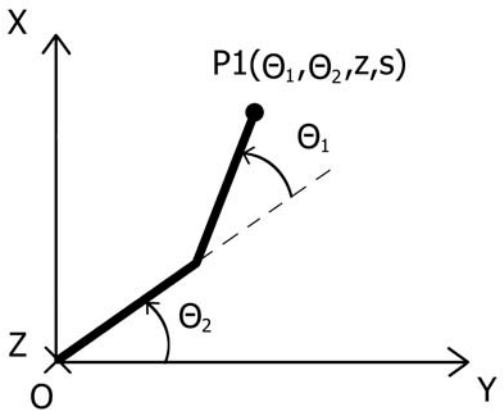


Figure 5.2: Joint Coordinate System.

5.2 Point to point motion functions

In *Point to Point Motion* (PTP motion), a target point is specified for the Manipulator. Neither motion trajectory nor actual motion speed on the way can be set. The motion trajectory and actual motion speed depend on the conditions of the Manipulator type. In general, this motion mode realizes the fastest speed to move to the target point. Thus, the Manipulator speed is specified indirectly with a percentage of the maximum speed of the Manipulator (see the function `speed`).
In *Point to Point Motion*, only the target point is specified. Indeed, the trajectory cannot be selected by the programmer. It is automatically chosen by the controller in order to optimize the motion.

5.2.1 Motion in the Cartesian Coordinate System

MOVE

Moves the robot to a position specified in the Cartesian coordinate system.
Syntax: `move(P)`
Input:

- P: matrix of Nx4 elements, where N is between 1 and 8, i.e. up to 8 positions can be passed to the functions. The positions are reached in row order.

Return value: none. Example

```
1 P1=[280,280,10,112];    % Defines position P1
2 move(P1);               % Moves to P1
3
4 P=[280,280,10,112;      % Defines a matrix P of 4 positions
5 300,280,10,112;
6 400,0,60,200;
7 0,300,10,112];
8 move(P);                % Moves to the positions defined in P
9                         % in row order
```

MOVED

Moves the robot to a position in the Cartesian coordinate system and not yet declared.
Syntax: `moved(x,y,z,s)`
Input:

- x: X position (mm);

- y: Y position (mm);

- z: Z-axis position (mm);

- s: Roll/S-axis position (deg).

Return value: none.
Example

```
move(280,280,10,112);       % Moves to the point
                            % (280,280,10,112)
```

RMOVE

Moves the robot to a position specified relative to the current position in the Cartesian coordinate system.
Syntax: `rmove(x,y,z,s)`
Input:

- x: X position variation (mm);

- y: Y position variation (mm);

- z: Z-axis position variation (mm);

- s: Roll/S-axis position variation (deg).

Return value: none.
Example

```
P1=[280,280,10,112];     % Defines position P1
move(P1);                % Moves to P1

rmove(20,10,30,10);      % Moves to (300,290,40,122)

P1=[280,280,10,112];     % Defines position P1
move(P1);                % Moves to P1

rmove(-20,-10,30,10);    % Moves to (260,270,40,122)
```

SMOVE

Moves the robot by changing only one of the three Cartesian coordinates or the Roll/S-axis position.
Syntax: `smove(n,p)`
Input:

- n: one of the three coordinates or the Roll/S-axis position $(1-4)$;

- p: value of the selected coordinate (mm) or of the Roll/S-axis position (deg).

Return value: none.
Example

```
P1=[280,280,10,112];    % Defines position P1
move(P1);               % Moves to P1

smove(1,300);    % Moves to (300,280,10,112)

smove(3,50);     % Moves to (300,280,50,112)

smove(4,100);    % Moves to (300,280,60,100)
```

SRMOVE

Moves the robot by changing only one of the three Cartesian coordinates or the Roll/S-axis position, relative to the current position in the Cartesian coordinate system.
Syntax: `srmove(n,p)`
Input:

- n: one of the three coordinates or the Roll/S-axis position $(1-4)$;

- p: variation of the selected coordinate (mm) or of the Roll/S-axis position (deg);

Return value: none.
Example

```
P1=[280,280,10,112];    % Defines position P1
move(P1);               % Moves to P1

srmove(1,30);      % Moves to (310,280,10,112)

srmove(3,50);      % Moves to (300,280,60,112)

srmove(4,100);     % Moves to (300,280,60,212)
```

5.2.2 Motion in the Joint Coordinate System

JMOVE

Moves the robot to a position specified in the joint coordinate system.
Syntax: `jmove(P)`
Input:

- P: matrix of Nx4 elements, where N is between 1 and 8, i.e. up to 8 positions can be passed to the functions. The positions are reached in row order.

Return value: none.
Example

```
P1=[90,30,10,112];    % Defines position P1
jmove(P1);            % Moves to P1

P=[90,30,10,112;      % Defines a matrix P of 4 positions
110,50,10,112;
90,-45,60,200;
70,10,10,-200];
jmove(P);             % Moves to the positions defined in P
                      % in row order
```

JMOVED

Moves the robot to a position specified in the joint coordinate system and not yet declared. Syntax: `jmoved(t1,t2,z,s)`
Input:

- t1: angle of the 1^{st} arm (deg);
- t2: angle of the 2^{nd} arm (deg);
- z: Z-axis position (mm);
- s: Roll/S-axis position (deg).

Return value: none.
Example

```
jmoved(90,30,10,112);      % Moves to the point
                           % (90,30,10,112)
```

RJMOVE

Moves the robot to a position specified relative to the current position in the joint coordinate system.
Syntax: `rjmove(x,y,z,s)`
Input:

- t1: 1^{st} arm angle variation (deg);
- t2: 2^{nd} arm angle variation (deg);

- z: Z-axis position variation (mm);

- s: Roll/S-axis position variation (deg).

Return value: none.
Example

```
P1=[90,30,10,112];      % Defines position P1
jmove(P1);              % Moves to P1

rjmove(20,10,30,10);      % Moves to (110,40,40,122)

P1=[90,30,10,112];        % Defines position P1
jmove(P1);                % Moves to P1

rjmove(-90,-40,20,30);    % Moves to (0,-10,30,142)
```

SJMOVE

Moves the robot by changing only one of the three joint coordinates or the Roll/S-axis position.
Syntax: `sjmove(n,p)`
Input:

- n: one of the three coordinates or the Roll/S-axis position $(1-4)$;

- p: value of the selected coordinates (deg or mm) of the Roll/S-axis position (deg).

Return value: none.
Example

```
P1=[90,30,10,112];    % Defines position P1
jmove(P1);            % Moves to P1

sjmove(1,20);      % Moves to (20,30,10,112)

sjmove(3,50);      % Moves to (20,50,10,112)

sjmove(4,100);     % Moves to (20,50,10,100)
```

SRJMOVE

Moves the robot by changing only one of the three joint coordinates or the Roll/S-axis position, relative to the current position in the joint coordinate system.
Syntax: `srjmove(n,p)`
Input:

- n: one of the three coordinates or the Roll/S-axis position $(1-4)$;

- p: variation of the selected coordinate (deg or mm) or of the Roll/S-axis position (deg).

Return value: none.
Example

```
P1=[90,30,10,112];   % Defines position P1
jmove(P1);           % Moves to P1

srjmove(1,20);     % Moves to (110,30,10,112)

srjmove(3,50);     % Moves to (110,30,60,112)

srjmove(4,100);    % Moves to (110,30,60,212)
```

5.3 Continuous Path motion functions

In *Continuous Path motion* (CP motion), not only the target point but also the motion trajectory and motion speed on the path of the Manipulator tip are specified (see the function `cpspeed`). This motion is also called *Interpolated motion* and can be performed only in the Cartesian coodinate system.

LMOVE

Moves the robot to some positions following a straight line as trajectory.
Syntax: `lmove(P)`
Input:

- P: matrix of Nx4 elements, where N is between 1 and 8, i.e. up to 8 positions can be passed to the functions. The positions are reached in row order.

Return value: none.
Example

```
P1=[280,280,10,112];    % Defines position P1
lmove(P1);              % Moves to P1

P=[280,280,10,112;      % Defines a matrix P of 4 positions
300,280,10,112;
400,0,60,200;
0,300,10,112];
lmove(P);               % Moves to the positions defined in P
                        % in row order
```

LMOVED

Moves the Manipulator to a position not yet declared, following a straight line as trajectory.
Syntax: `lmoved(x,y,z,s)`
Input:

- x: X position (mm);
- y: Y position (mm);
- z: Z-axis position (mm);
- s: Roll/S-axis position (deg).

Return value: none.
Example

```
lmove(280,280,10,112);      % Moves to the point
                            % (280,280,10,112)
```

RLMOVE

Moves the robot to a position specified relative to the current position following a straight line as trajectory.
Syntax: `rlmove(x,y,z,s)`
Input:

- x: X position variation (mm);
- y: Y position variation (mm);
- z: Z-axis position variation (mm);
- s: Roll/S-axis position variation (deg).

Return value: none.
Example

```
P1=[280,280,10,112];      % Defines position P1
lmove(P1);                % Moves to P1

rlmove(20,10,30,10);      % Moves to (300,290,40,122)

P1=[280,280,10,112];      % Defines position P1
lmove(P1);                % Moves to P1

rlmove(-20,-10,30,10);    % Moves to (260,270,40,122)
```

SLMOVE

Moves the robot by changing only one of the three Cartesian coordinates or the Roll/S-axis position. The trajectory of the motion is a straight line.
Syntax: `slmove(n,p)`
Input:

- n: one of the three coordinates or the Roll/S-axis position $(1-4)$;
- p: value of the selected coordinate (mm) or of the Roll/S-axis position (deg).

Return value: none.
Example

```
P1=[280,280,10,112];     % Defines position P1
lmove(P1);               % Moves to P1

slmove(1,300);     % Moves to (300,280,10,112)

slmove(3,50);      % Moves to (300,280,50,112)

slmove(4,100);     % Moves to (300,280,60,100)
```

SRLMOVE

Moves the robot by changing only one of the three Cartesian coordinates or the Roll/S-axis position, relative to the current position in the Cartesian coordinate system. The trajectory of the motion is a straight line.
Syntax: `srlmove(n,p)`
Input:

- n: one of the three coordinates or the Roll/S-axis position $(1-4)$;
- p: variation of the selected coordinate (mm) or of the Roll/S-axis position (deg).

Return value: none.
Example

```
P1=[280,280,10,112];       % Defines position P1
lmove(P1);                 % Moves to P1

srlmove(1,30);      % Moves to (310,280,10,112)

srlmove(3,50);      % Moves to (300,280,60,112)

srlmove(4,100);     % Moves to (300,280,60,212)
```

ARCHMOVE

Moves the robot in a 3D arc motion by interpolating three points.
Syntax: `archmove(PA,PB)`
Input:

- PA: intermediate point;

- PB: end point.

Return value: none.
Example

```
P1 = [??, ??, ??, ??];       % Generic position inside the
                             % workspace
P2 = [??, ??, ??, ??];       % Generic position inside the
                             % workspace
P3 = [??, ??, ??, ??];       % Generic position inside the
                             % workspace
P4 = [??, ??, ??, ??];       % Generic position inside the
                             % workspace
P5 = [??, ??, ??, ??];       % Generic position inside the
                             % workspace
P6 = [??, ??, ??, ??];       % Generic position inside the
                             % workspace

lmove(P1);            % Moves to P1
lmove(P2);            % Moves to P2

archmove(P3,P4);         % Performs an arc motion that
                         % interpolates P2, P3, P4

archmove(P5,P6);         % Performs an arc motion that
                         % interpolates P4, P5, P6
```

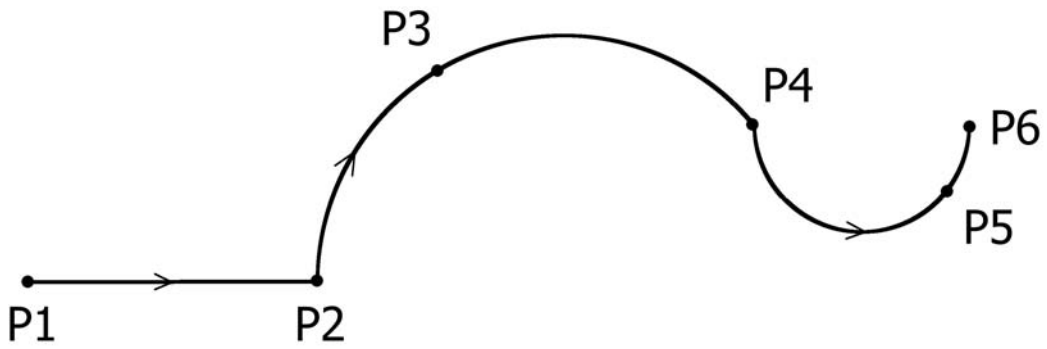


Figure 5.3: Arc motion example

CMOVE

Moves the robot in a 3D circular motion by interpolating three points.
Syntax: `cmove(P1,P2,P3)`
Input:

- P1: first intermediate point;
- P2: second intermediate point;
- P3: end point **(The value of the X- and Y- axes must be the same as those of the starting point)**.

Return value: none.
Example

```
P1 = [??, ??, ??, ??];      % Generic position inside the
                            % workspace
P2 = [??, ??, ??, ??];      % Generic position inside the
                            % workspace
P3 = [??, ??, ??, ??];      % Generic position inside the
                            % workspace

lmove(P1);          % Moves to P1

lmove(P2);          % Moves to P2

cmove(P3, P4, P2);          % Performs a circular motion
```

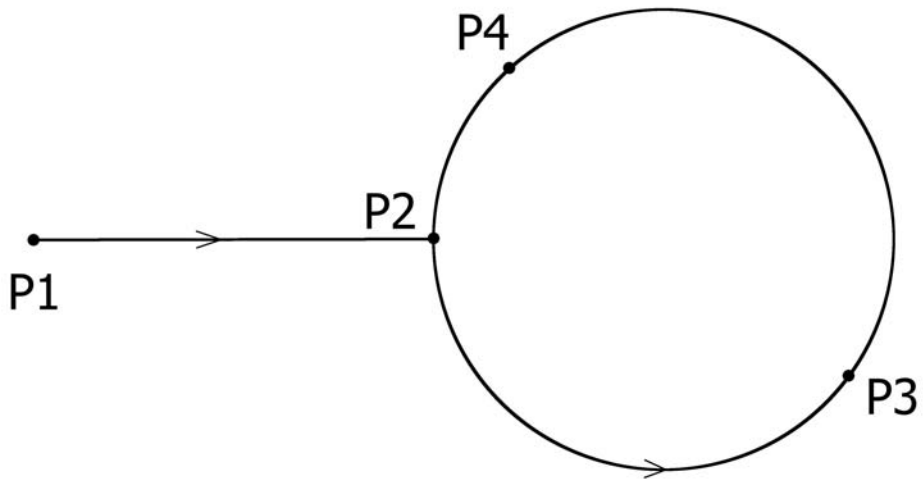


Figure 5.4: Circular motion example

XYCIR

Moves the robot in a 2D circular motion in the X-Y plane by setting the centre and the radius.
Syntax: `xycir(r,P,a,b)`
Input:

- r: radius of the circle;
- P: centre of the circle;
- a: parameter for selecting the starting point $(1-4)$, see Figure 5.5;
- b: parameter for selecting wise (1 clockwise , 2 counterclockwise).

Return value: none.
Example

```
P_centre = [??, ??, ??, ??];      % Generic position inside the
                                  % workspace

r=100;                            % Sets a radius of 100 mm

xycir(r, P_centre, 2, 1);         % Moves the robot in a
                                  % circular clockwise motion
                                  % starting from the point 2
```

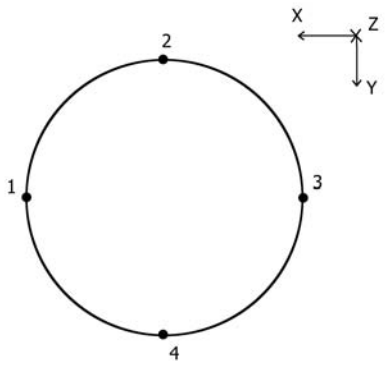


Figure 5.5: Possible starting points for circular motion in X-Y plane.

XZCIR

Moves the robot in a 2D circular motion in the X-Z plane by setting the centre and the radius.
Syntax: `xzcir(r,P,a,b)`
Input:

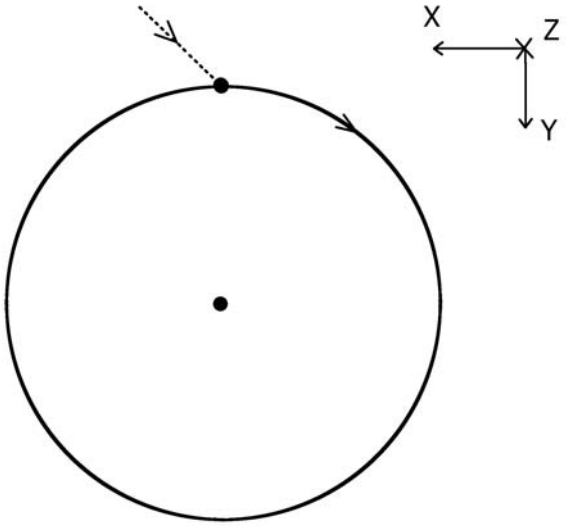


Figure 5.6: Circular motion example by using `xycir`.

- r: radius of the circle;

- P: centre of the circle;

- a: parameter for selecting the starting point $(1-4)$, see Figure 5.7;

- b: parameter for selecting wise (1 clockwise , 2 counterclockwise).

Return value: none.
Example

```
1 P_centre = [??, ??, ??, ??];      % Generic position inside the
2                                   % workspace
3
4 r=100;                            % Sets a radius of 100 mm
5
6 xzcir(r, P_centre, 4, 2);         % Moves the robot in a
7                                   % circular counterclockwise
8                                   % motion starting from the
9                                   % point 4
```

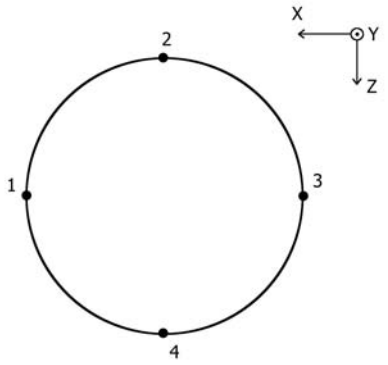


Figure 5.7: Possible starting points for circular motion in X-Z plane.

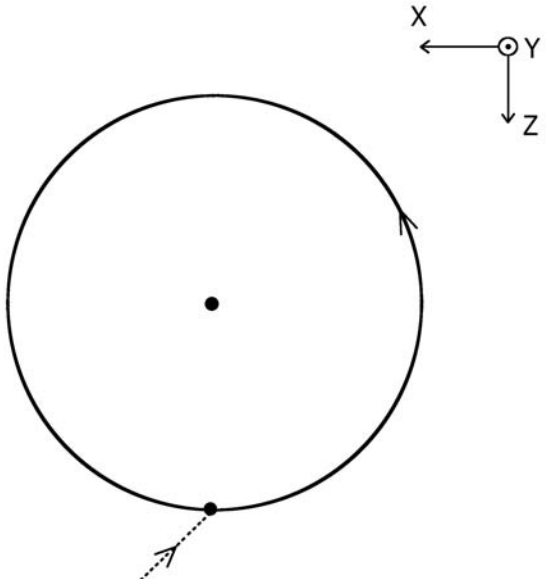


Figure 5.8: Circular motion example by using `xzcir`.

YZCIR

Moves the robot in a 2D circular motion in the Y-Z plane by setting the centre and the radius.
Syntax: `yzcir(r,P,a,b)`
Input:

- r: radius of the circle;
- P: centre of the circle;
- a: parameter for selecting the starting point $(1-4)$, see Figure 5.9;
- b: parameter for selecting wise (1 clockwise , 2 counterclockwise).

Return value: none.

Example

```
P_centre = [??, ??, ??, ??];      % Generic position inside the
                                  % workspace

r=100;                            % Sets a radius of 100 mm

yzcir(r, P_centre, 4, 2);         % Moves the robot in a
                                  % circular clockwise motion
                                  % starting from the point 4
```

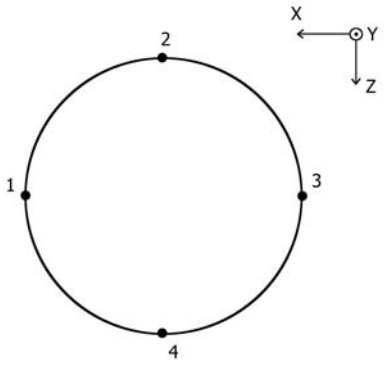


Figure 5.9: Possible starting points for a circle in Y-Z plane.

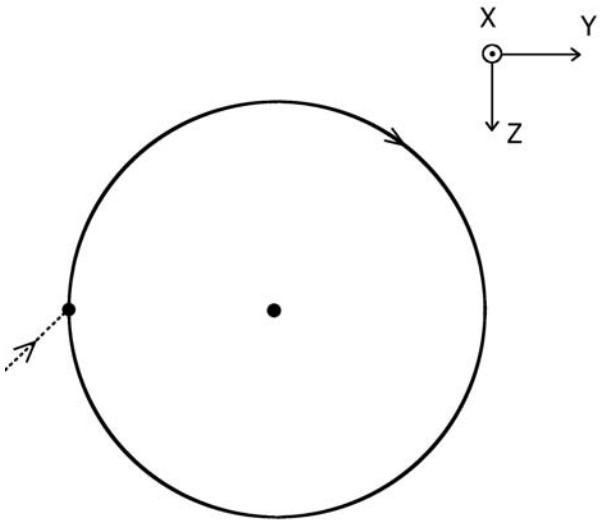


Figure 5.10: Circular motion example by using `yzcir`.

5.4 Speed and acceleration/deceleration functions

SPEED

Sets the PTP motion maximum speed for all axes. The value is a percentage of the speed limit due to the mechanical structure of the robot. When the speed is not specified in the program, the default speed is 10%.
Syntax: `speed(a)`
Input:

- a: real number from 1 to 100 [%].

Return value: previous set value.
Example

```
P1=[280,280,10,112];    % Defines position P1
P2=[-300,200,15,140];   % Defines position P2

speed(18);              % Sets 18% as maximum PTP motion speed

move(P1);               % Moves to P1

speed(40);              % Sets 40% as maximum motion speed

move(P2);               % Moves to P2
```

CPSPEED

Sets the CP motion speed.
Syntax: `cpspeed(a)`
Input:

- a: positive real number (mm/s).

Return value: previous set value.
Example

```
speed(30);              % Sets 30% as maximum PTP
                        % motion speed

P1=[300,280,10,112];          % Defines position P1
P2=[-300,200,15,140];         % Defines position P2

move(P1);               % Moves to P1 (PTP motion)

cpspeed(150);           % Sets 150 mm/s as speed
                        % for CP motion
```

```
11
12 lmove(P2);              % Linear motion to P2 (CP motion)
13
14 P3=[313,300,90,112;];          % Defines position P3
15 P4=[0,231,3,112;];             % Defines position P4
16 P5=[-313,300,90,112;];         % Defines position P5
17
18 cpseed(100);            % Sets 100 mm/s as speed
19                         % for CP motion
20
21 move(P3);               % Moves to P3 (PTP motion)
22
23 cmove(130,P4,P5,P3);           % Performs a circular motion
24                                 % (CP motion)
```

ACCT

The function `acct` sets the accelerating time in PTP motion.
When the robot performs PTP motion, it cannot immediately reach the desired speed set by the function `speed`. Therefore, after starting the motion, the robot increases its speed gradually up to the value set by `speed` and continues with a constant speed. Then it decreases the speed gradually before it finally reaches its commanded position. The period while the Robot is accelerating is called *Accelerating Time*. See Figure 5.11.

By default, this function is disabled. Type `autoalcl(1)` for enabling it and `autoalcl(0)` for disabling it again. Therefore, when the command `autoalcl(1)` is used, the command `acct` becomes invalid and is just ignored.

When `autoalcl` is made invalid and `dacct` is not specified, the default value is 1 second.

Syntax: `acct(a)`
Input:

- a: real non-negative number (s).

Return value: previous set value.
Example

See `autoacl` example.

Set with 0 makes the motion through multiple points very smooth. De-

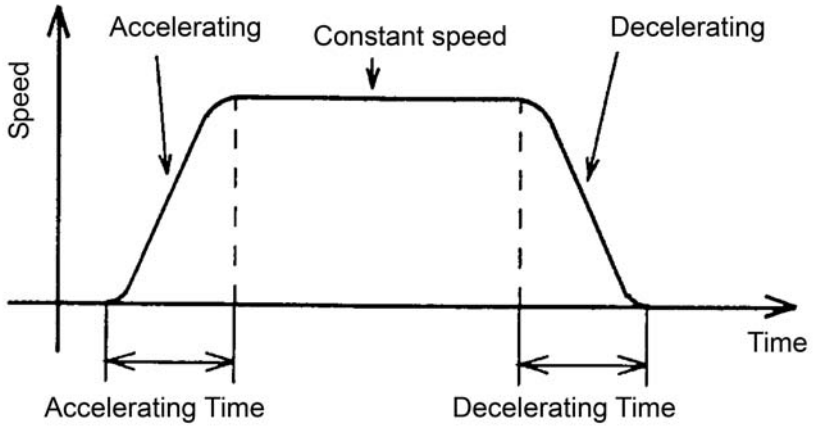


Figure 5.11: PTP motion speed profile.

pending on the setup conditions, this setting could damage the robot. Therefore, try setting at 0 after becoming familiar with the robot operations and programming.

DACCT

The function `dacct` sets the decelerating time in PTP motion.
When the robot performs PTP motion, it cannot immediately reach the desired speed set by the function `speed`. Thus, after starting motion, the robot increases its speed gradually up to the value set by `speed` and continues with a constant speed. Then it decreases the speed gradually before it finally reaches its commanded position. The period while the Robot is decelerating is called *Decelerating Time*. See Figure 5.11.

By default, this function is disabled. Type `autoalcl(1)` for enabling it and `autoalcl(0)` for disabling it again. Therefore, when the command `autoalcl(1)` is used, the command `dacct` becomes invalid and is just ignored.

When `autoalcl` is made invalid and `acct` is not specified, the default value is 1 second.

Syntax: `acct(a)`
Input:

- a: real not negative number (s).

Return value: previous set value.
Example

See `autoacl` example.

Set with 0 makes the motion through multiple points very smooth. Depending on the setup conditions, this setting could damage the robot. Therefore, try setting at 0 after becoming familiar with the robot operations and programming.

WEIGHT

Specifies the payload required for automatic acceleration/deceleration calculation.
Syntax: `weight(a)`
Input:

- a: real number, from 0 to the maximum payload weight [kg].

Return value: none.
Example

See `autoacl` example.

AUTOACL

Enables or disables the automatic optimum acceleration and deceleration settings for PTP motion. The automatic acceleration and deceleration settings depend on the payload, which the robot handles, specified by the function `weight`. When the automatic acceleration and deceleration settings are disabled, acceleration and deceleration time can be set by using `acct` and `dacct`.
Syntax: `autoacl(a)`
Input:

- a: 1 (Enables the automatic acceleration and deceleration settings);
  0 (Disables the automatic acceleration and deceleration settings).

Return value: none.
Example

```
speed(30)               % Sets 30% as maximum PTP
                        % motion speed

P1=[300,280,10,112];          % Defines position P1
P2=[-300,200,15,140];         % Defines position P2

move(P1);               % Moves to P1 in automatic
                        % acceleration & deceleration
                        % since autoacl has not be
                        % disabled.

acct(2);                % It is ignored, since autoacl
                        % has not be disabled

dacct(2);               % It is ignored, since autoacl
                        % has not be disabled

autoacl(0);             % Disables the automatic acceleration
                        % and deceleration settings

acct(2);                % Set 2 s as accelerating time

dacct(2.6);             % Set 2.6 s as decelerating time

move(P1);               % Moves to P1 (PTP motion).
                        % The acceleration a deceleration
                        % time are set to 2 s and 2.6 s
                        % respectively

autoacl(1);             % Enables the automatic acceleration
                        % and deceleration settings

weight(2);              % Weight is 2 kg

move(P1);               % Moves to P1 in automatic
                        % acceleration & deceleration
```

CPACCT

Sets the accelerating time for CP motion. If acceleration time is not set in the program, the default acceleration time is set at 0.1 seconds.
Syntax: `cpacct(a)`
Input:

- a: real not negative number. (s).

Return value: previous set value.
Example

```
P1=[300,280,10,112];          % Defines position P1
P2=[-300,200,15,140];         % Defines position P2

cpspeed(100);                 % Sets 100 mm/s as CP
```

```
5                                   % motion speed
6
7  cpacc(1);                        % Sets 1 s as accelerating
8                                   % time for CP motion
9
10 cpdacc(1.5);                     % Sets 1.5 s as decelerating
11                                  % time for CP motion
12
13 lmove(P1);                       % Moves to P1 (CP motion)
```

Set with 0 makes the motion through multiple points very smooth. Depending on the setup conditions, this setting could damage the robot. Therefore, try setting at 0 after becoming familiar with the robot operations and programming.

CPDACCT

Sets the decelerating time for CP motion. If decelerating time is not set in the program, the default acceleration time is set at 0.1 seconds.
Syntax: `cpdacct(a)`
Input:

- a: real not negative number (s).

Return value: previous set value.
Example

```
1  P1=[300,280,10,112];             % Defines position P1
2  P2=[-300,200,15,140];            % Defines position P2
3
4  cpspeed(100);                    % Sets 100 mm/s as CP
5                                   % motion speed
6
7  cpacc(1);                        % Sets 1 s as accelerating
8                                   % time for CP motion
9
10 cpdacc(1.5);                     % Sets 1.5 s as decelerating
11                                  % time for CP motion
12
13 lmove(P1);                       % Moves to P1 (CP motion)
```

Set with 0 makes the motion through multiple points very smooth. Depending on the setup conditions, this setting could damage the robot. Therefore, try setting at 0 after becoming familiar with the robot operations and programming.

5.5 Arm mode functions

RIGHT

Selects the right *arm mode*.
The workspace is divided in three areas: one can be reached in left arm mode only, one in right arm mode only, and one in both arm modes. See Figure 5.12.
If the arm mode is not set in the program, the right arm mode is set by default.

Syntax: `right()`
Input: none.
Return value: none.
Example

```
1  P1=[300,280,10,112];          % Defines position P1
2  P2=[-300,200,15,140];         % Defines position P2
3
4  right();                      % Selects the right
5                                % arm mode
6
7  move(P1);                     % Moves to P1 (PTP motion)
8                                % in right arm mode
9
10 lmove(P2);                    % Moves to P2 (CP motion)
11                               % in right arm mode
```

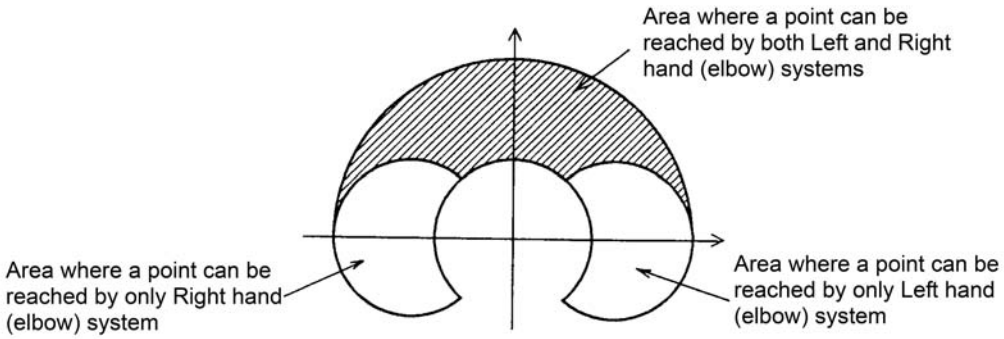


Figure 5.12: The three areas of the workspace.

LEFT

See comments about `right` above.
Syntax: `left()`

Input: none
Return value: none
Example

```
P1=[300,280,10,112];          % Defines position P1
P2=[-300,200,15,140];         % Defines position P2

left();                       % Selects the left
                              % arm mode

move(P1);                     % Moves to P1 (PTP motion)
                              % in left arm mode

lmove(P2);                    % Moves to P2 (CP motion)
                              % in left arm mode
```

5.6 Mark functions

MARK

Calculates the robot current position in the Cartesian coordinate system. Due to the fact that the values are calculated by reverse-conversion of quantized position pulses (not command pulses), a small error can be introduced between the position specified in the program and the values calculated with this function.
Syntax: `mark()`
Input: none.
Return value: a position, i.e. a vector of four numbers.
Example

```
P1=[300,280,10,112];          % Defines position P1

move(P1);                     % Moves to P1

srmotion(1,-100);             % Moves only the first axis
srmotion(2,-10);              % Moves only the second axis

P_current = mark();           % Calculates the current
                              % position and stores it
                              % in P_current
```

JMARK

Calculates the robot current position in the joint coordinate system. As with the function `mark()`, a small error can be introduced between the position specified in the program and the values calculated with this function.

Syntax: `jmark()`
Input: none.
Return value: a position, i.e. a vector of four numbers.
Example

```
P1=[300,280,10,112];          % Defines position P1

move(P1);                     % Moves tp P1

srmotion(1,-100);             % Moves only the first axis
srmotion(2,-10);              % Moves only the second axis

P_current = jmark();          % Calculates the current
                              % position and stores it
                              % in P_current
```

5.7 I/O functions

IN

Checks the status of any digital input (DI) port.
Syntax: `in(a)`
Input:

- a: number that identifies an input port (1-8 I/O-1, 9-16 I/O-2, 921-936 EX. I/O-1).

Return value: 1 (Port is ON);
0 (Port is OFF).

Example

```
P1=[300,280,10,112];          % Defines position P1
P2=[-300,200,15,140];         % Defines position P2

port_status = in(922);        % Checks the status of
                              % the port 922

if(port_status==1)            % If port 922 is ON
                              % it moves to P1, if
   move(P1);                  % port 922 is OFF it
                              % moves to P2

else

   move(P2);

end
```

OUT

Turns ON or OFF a digital output port.

Syntax: `out(a, b)`

Input:

- a: number that identifies an output port (17-20 I/O-1, 21-24 I/O-2, 937-952 EX. I/O-2);
- b: 1 (ON);
  0 (OFF).

Return value: none.

Example

```
P1=[300,280,10,112];          % Defines position P1

move(P1);                     % Moves to P1

out(939,1);                   % Turns ON the output
                              % port

i=1;                          % Iteration variable

while(i<11)                   % This cycle is for
                              % blinking the output
   out(18,1);                 % port with a period
   pause(1);                  % of 1 s
   out(18,0);
   pause(1);
   i=i+1;

end
```

WINTIME

Sets the input wait time for functions `win(a, b)` and `tri(a, b)` .

Syntax: `wintime(a, b)`

Input:

- a: real not negative number (s).

Return value: previous set value.

Example

See `win` and `tri` examples.

WIN

Waits for a specified digital input port of the Controller to turn on or off. If the conditions are not met within the time specified by the function `wintime`, a time-out error is caused and the program stops.
Syntax: `win(a, b)`
Input:

- a: number that identifies an input port (1-8 I/O-1, 9-16 I/O-2, 921-936 EX. I/O-1);
- b: 1 (ON)
  0 (OFF)

Return value: none. Example

```
P1=[300,280,10,112];          % Defines position P1

wintime(5);                   % Sets the waiting time
                              % to 5 s

win(10,1);           % Waits for state 1 of port 10. If this
                     % doesn't happen within 5 seconds,
                     % time-out  error is caused and the
                     % program stops.

move(P1);            % Moves to P1
```

TRI

Waits for a specified digital input port of the Controller to turn on or off, and notifies whether or not it turns on or off within the time specified by the function `wintime`.
Syntax: `tri(a, b)`
Input:

- a: number that identifies an input port (1-8 I/O-1, 9-16 I/O-2, 921-936 EX. I/O-1).

Return value: 1: it met on/off condition within time specified by `wintime`;
0: it did not meet on/off condition within time specified by `wintime`.

Example

```
P1=[300,280,10,112];          % Defines position P1
```

```
P2=[-300,200,15,140];         % Defines position P2

wintime(5);          % Sets the waiting time to 5 s


x = tri(10,1);       % Waits for state 1 of port 10.

% If this happen within 5 seconds
% x=1, else x=0

if(x==1)

   move(P1);

else

   move(P2);

end
```

BLINK

Blinks a specified digital output port.
The Controller can control up to 16 ports simultaneously.

Syntax: `blink(a, b, c)`
Input:

- a: number that identifies an output port (17-20 I/O-1, 21-24 I/O-2, 937-952 EX. I/O-2);
- b: time while the port is on (a positive real number);
- c: time while the port is off (a positive real number).

If "c" is not specified, the time while the port is off becomes the same as "b".

Return value: none
Example

```
blink(19,1,3);            % Blinks port 19: time while the
                          % port is on=1 s, time while
                          % the port 19 is off=3 s

blink(20,2);              % Blinks port 19: time while the
                          % port is on=2 s, time while
                          % the port 19 is off=2 s
```

BLINKED

Invalidates up to 4 functions `blink` turning off the specified ports.

Syntax: `blinked(a [, b, c, d])`

The “b”, “c”, “d” are optional.

Input:

- a: number that identifies an output port (17-20 I/O-1, 21-24 I/O-2, 937-952 EX. I/O-2);
- b: number that identifies an output port (17-20 I/O-1, 21-24 I/O-2, 937-952 EX. I/O-2);
- c: number that identifies an output port (17-20 I/O-1, 21-24 I/O-2, 937-952 EX. I/O-2);
- d: number that identifies an output port (17-20 I/O-1, 21-24 I/O-2, 937-952 EX. I/O-2).

Return value: none.
Example

```
1  blink(19,1,3);              % Blinks port 19: time while the
2                              % port is on=1 s, time while
3                              % the port 19 is off=3 s
4
5  blink(20,2);                % Blinks port 19: time while the
6                              % port is on=2 s, time while
7                              % the port 19 is off=2 s
8
9  blinkend(19,20);            % Invalidates the two previous
10                             % blink commands  turning off
11                             % ports 19 and 20
```

5.8 Pendant output message functions

LOCATE

Locates the position to display a message defined by `opeout` on the Pendant display.
Syntax: `locate(a, b)`
Input:

- a: integer in the range of $1-4$ to specify the line to display the message on the pendant screen;

- b: integer in the range of $0-16$ to specify the column to display the message on the pendant screen.

Return value: none.
Example

See `opeout` example.

OPCLR

Clears the characters on the Pendant display outputted by `opeout`.
Syntax: `opeout()`
Input: none.
Return value: none.
Example

See `opeout` example.

OPEOUT

Outputs a message on Pendant display according to the line and column set by `locate`.
Syntax: `opeout(a)`
Input:

- a: string (max 16 characters);

Return value: none.
Example

```
x='Hello World';      % Defines string x
y='11111';            % Defines string y
z='UCC';              % Defines string z

locate(0,0);  % Locates the position on the Pendant display

opeout(x);    % Outputs the string x on Pendant display

opeclr();     % Clears the characters on the Pendant display

```

```
11 locate(1,0);  % Locates the position on the Pendant display
12
13 opeout(y);    % Outputs the string y on Pendant display
14
15 opeclr();     % Clears the characters on the Pendant display
16
17 locate(1,3);  % Locates the position on the Pendant display
18
19 opeout(z);    % Outputs the string z on Pendant display
20
21 opeclr();     % Clears the characters on the Pendant display
```

5.9 Palletizing functions

SETPLT

Defines a pallet configuration.
Syntax: `setplt(1,p0,p1,p2,p3,i,j,k)`

Input:

- n: pallet number (1 through 10 are valid);
- p0: position variable;
- p1: position variable;
- p2: position variable;
- p3: position variable;
- i: number of element between p0 and p1;
- j: number of element between p0 and p2;
- k: number of element between p0 and p3.

Return value: none
Example

See Figure 5.13.

PLT

Calculates the position corresponding to a point number on an user-defined pallet.
Syntax: `plt(n,a)`
Input:

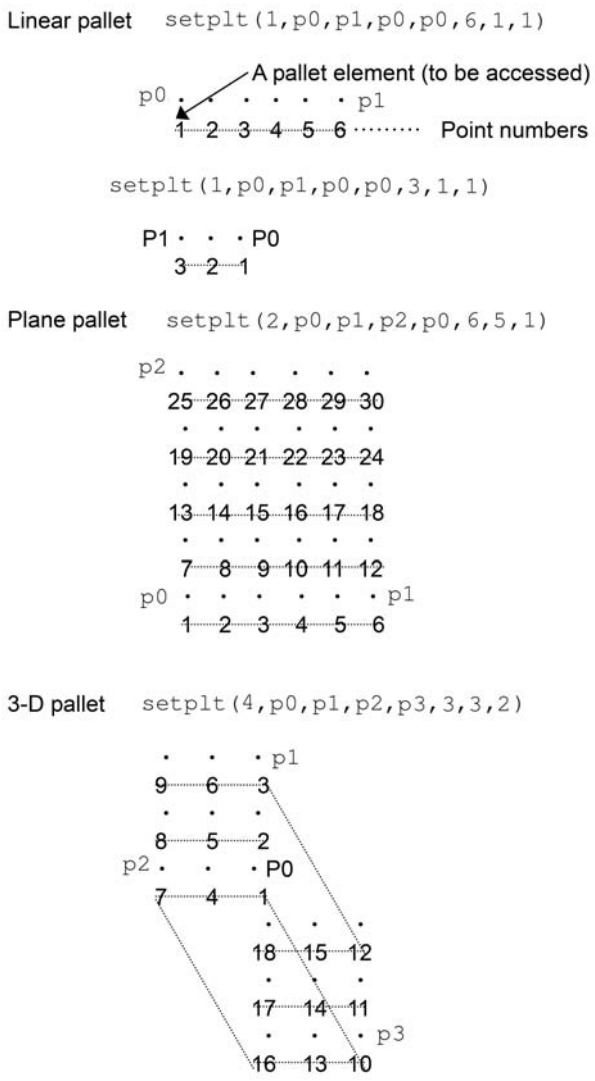


Figure 5.13: Examples of pallet definition.

- n: pallet number (positive integer);

- a: point number (positive integer).

Return value: none.
Example

```
right();

moved(260,260,10,112);

% Palletizing

p0=[300,300,50,112];
p1=[100,300,50,112];
p2=[300,450,50,112];

speed(12);

setplt(1,p0,p1,p2,p0,3,3,1);   % Defines a pallet
                               % configuration.
i=1;

% The next cycle moves the robot over all
% the points defined by setplt function

while(i≤9)

   p=plt(1,i);
   p(1,3)=10;
   move(p);
   speed(25);
   move(plt(1,i));
   speed(25);
   move(p);
   speed(12);
   i=i+1;

end
```

5.10 Sampling Mode

A new important functionality has been developed in this project and it is called *Trajectory/Angles Sampling Mode*. This option allows the trajectory of the end effector or the values of Θ_1 and Θ_2 during the motion to be sampled. The samples are then available for analysis and processing. They can also be interpolated and plotted. An example of the Trajectory Sampling Mode is shown in Listing 5.1 and Figure 5.14. An example of the Angles Sampling Mode is shown in Listing 5.2 and Figure 5.15.

The function `sample` activates the trajectory sampling mode, or the angles sampling mode depending on the input string value: `trajectory` or `angles`. The function `visual` allows data interpolation and plotting, depending on the selected sampling mode.

5.10.1 How it works

The application involves six Matlab functions: three Matlab Robot Functions (`sample`, `visual`, and `samplesstorage`), and three auxiliary Matlab functions (`serial_out1`, `serial_out5`, and `state_keper`). The three Matlab robot functions use the auxiliary Matlab function `state_keeper` (see section 4.4.2) in order to update and retrieve some variables that keep information about the state of the application, and the time information associated with each sample. The function `serial_out5` is called when `sample` is executed and it sends a communication code, depending on the selected sampling mode, to the interpreter through the serial cable. Once the controller receives this code, it sets a variable. Once this variable is set, during whatever motion function execution, i.e. while the robot is moving, a sampling operation is carried out. This is possible due to the fact that the robot controller can perform some basic operations while the robot is moving (turning on or off an output pin, reading the state of an input pin, and sampling the current position).
Therefore, when a Matlab robot function is executed, `serial_out1` sends the Function Code of the function and the target position. Then it waits for the Feedback Execution Confirmation Code. If the sampling mode is activated, it has also to read samples that the controller sends back to Matlab, and store them in a matrix calling the function `samplesstorage`. This function allows both the storing and retrieving of samples.
The time information associated with each sample is not the true sampling instant, but the instant when the sample is received from Matlab. Furthermore, the interval between two samples is not exactly the same every time, due to the fact that the system is not a real-time system. The sampling period lies between 0.2 to 0.27 ms.

Listing 5.1: Trajectory sampling mode example.

```
1  cpspeed(50);    % Sets the speed of linear motion
2                  % and circular motion
3
4  P1=[313,300,90,112];
5  P2=[0,231,3,112];
6  P3=[-313,300,90,112];
7
8  move(P1);   % Moves to P1 (PTP motion)
9
10 sample('trajectory');   % Activates the Trajectory
```

```
11                          % Sampling Mode
12
13 archmove(P2,P3);    % Arc motion
14
15 lmove(P1);  % Linear motion
16
17 visual()    % Plots the trajectory
```

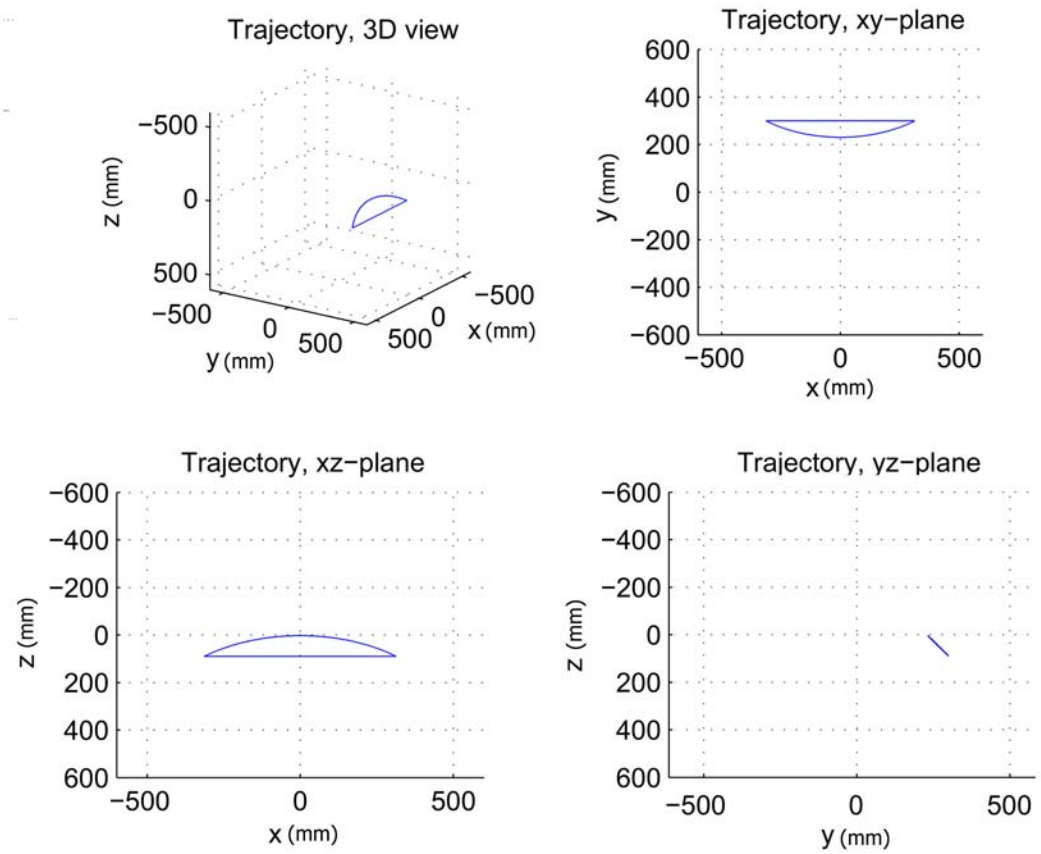


Figure 5.14: Trajectory.

Listing 5.2: Angles sampling mode example.

```
1 cpspeed(50);    % Sets the speed of linear motion
2                 % and circular motion
3
4 P1=[313,300,90,112;];
5 P2=[0,231,3,112;];
6 P3=[-313,300,90,112;];
7
8 move(P1);    % Moves to P1 (PTP motion)
9
```

```
10 sample('angles');    % Activates the Angle
11                      % Sampling Mode
12
13 archmove(P2,P3);    % Arc motion
14
15 lmove(P1);  % Linear motion
16
17 visual()    % Plots Theta1 and Theta2 trends
```

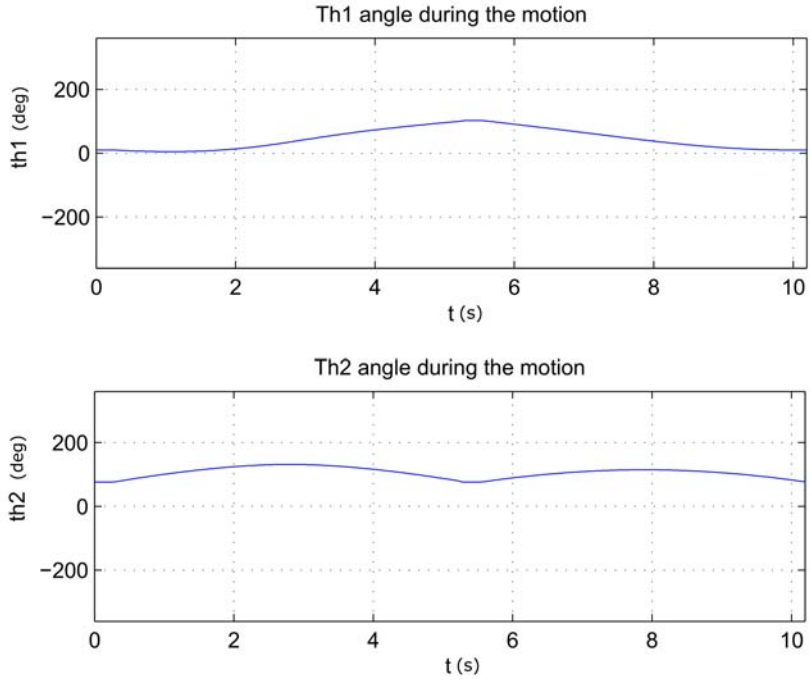


Figure 5.15: Θ_1 and Θ_2 trends during the motion.

Chapter 6

Vision based applications

Without sensory feedback, an industrial robot can not intelligently interact with its environment. The most valuable sense that can be provided to a robot, to establish information about the environment and feedback direction control, is vision [17]. Computer vision and pattern recognition techniques are widely used for industrial applications and especially for robot vision. In many fields of industry, indeed, there is the need to automate the pick-and-place process of picking up objects, possibly performing some tasks, and then placing down them on a different location [18].
In this project, two applications have been developed, using an HD camera combined with the Matlab *Image Acquisition Toolbox* and *Image Processing Toolbox*, in order to widen the robot functionality. The first part of this chapter is an overview of the software and hardware tools. In the second part, the two applications are described.

6.1 HD cam, image acquisition and processing

6.1.1 HD camera

An HD 720p camera, the Creative Live! Cam Chat HD (see Figure 6.1), is used in order to acquire images of the items in the workspace. It is fixed at the end of the second arm of the robot (see Figures 6.2, 6.3, and 6.4) and connected to the PC, where Matlab is installed, through a USB cable.

Figure 6.1: The Creative Live! Cam Chat HD.

Figure 6.2: Camera position (upper view).

Figure 6.3: Camera position (side view).

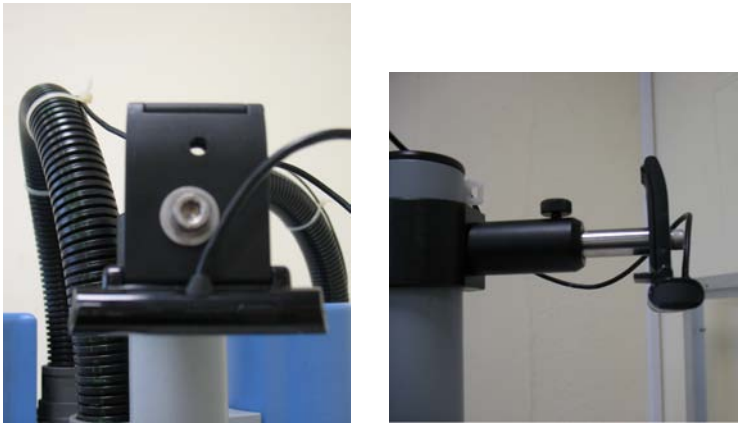


Figure 6.4: Camera position (close views).

6.1.2 Image acquisition and processing

Image Acquisition Toolbox

The Matlab Image Acquisition Toolbox allows an easy acquisition of images and videos directly into Matlab and Simulink [27]. The Matlab native commands for opening the camera, for configuring the images acquisition, and outputting the video input object are written inside the Matlab application function `start_cam`. See Listing 6.1.
The Matlab commands for image capturing are included in other Matlab Application Function that will be shown later.

Image Processing Toolbox

The Matlab Image Processing Toolbox offers a massive set of algorithms, functions, and apps for image processing, analysis, visualization, and algorithm development [25]. The two developed applications exploit some of these functions, especially in order to recognise the target object, its color and orientation, and to reduce noise. A colour image in Matlab is stored as an m x n x 3 matrix where each element is the RGB (Red, Green, Blue) value of that particular pixel (therefore it's a 3D matrix) [8]. It can be considered as three 2D matrices for red, green and blue intensities. The intensity of each pixel lies between 0 and 255. Alternatively, a BW image is stored as a 2D matrix where each pixel is 0 (black) or 1 (white).

Listing 6.1: Matlab application function `start_cam`.

```
1  function [o] = start_cam( a )
2
3  persistent vid;
4
5  if(strcmp(a,'open'))      % The inner expression is true if
6                            % the input parameter is the string
7                            % 'open'
8
9     pause on;
10
11    % The video input object is stored in the variable vid
12
13    vid =videoinput('winvideo',1);
14
15    % The next statement sets to 1 the number of frames that
16    % are captured each time 'trigger' is executed
17
18    set(vid, 'FramesPerTrigger', 1);
19
20    % The next statement sets 'TriggerRepeat' to inf, that
21    % allows to use 'trigger? infinite times
22
23    set(vid, 'TriggerRepeat', Inf);    % Sets the object to
24                                       % manual  triggering
25
26    triggerconfig(vid, 'manual');    % Starts the video capture
```

```
27
28     start(vid);                          % Starts the video capture
29
30  else
31
32        if(strcmp(a,'retrieve'))     % The inner expression is true
33                                     % if the input parameter is the
34                                     % string  'retrieve'
35
36           o=vid      % If the function start_cam has been
37                      % called passing  1 as input parameter,
38                      % the object 'vid' is outputs
39
40        end
41  end
42
43  end
```

6.1.3 Object position detection

In order to refer to the centroid of an object in an image, to the main reference frame of the manipulator, an heuristic approach has been used. The system does not have to determine the Z coordinate of the object because it is considered known. Three objects have been placed in the robot workspace inside the camera field of view. In an acquired image, one object is in the centre, one at the top right corner, and one at the bottom left corner. This positioning has been chosen in order to cover all the camera field of view. Already knowing the position of the object centroids in the image (it will be shown how it is possible in section 6.5.2), and already knowing the position of the object centroids in the workspace (the end effector of the manipulator has been moved over the positioned objects and the robot position has been sampled), the constant parameters of the relationship between the position of the object in the image and the position of the object in the manipulator reference frame has been calculated. These parameters have been used by the Matlab application function `position` (see Listing A.2 in appendix A) which receives the centroid position of an object in the image as input parameter, and it outputs the centroid position of the object in the manipulator workspace. The precision of this camera calibration has been proved good for the purposes of the two pick and place applications that have been developed.

6.2 Vacuum Gripping System

In order to pick up some objects inside the workspace, the SCARA robot has been provided with a *Vacuum Gripping System*. The high speed in picking and releasing items, combined with its reliability, and very low cost make this option the best solution for the project purposes. Considering Figure 6.5 and table 6.1, a pressure

regulator provides 0.35 MPa to a *Vacuum Ejector* [35]. These are fixed on a rail placed at the base of the workspace structure. The Vacuum Ejector, exploits the Venturi effect for creating vacuum. It is provided with two ports: a pressure port (input), and a vacuum port (output). A normally closed *solenoid valve* allows the device activation (24 V has to be provided) and deactivation (0 V has to be provided) for picking up and releasing the piece respectively. A *suction filter* ensures that dust or particulates do not damage the device. A hose connects the vacuum port of the vacuum ejector, to a connector in the rear panel of the robot base. Vacuum can reach the connector panel on the second arm of the robot by way a pipe inside the manipulator structure. Another hose brings vacuum to the hollow Z-axis metal extension. See Figure 6.6. A *vacuum cup* acts as an end effector (see Figure 6.7).

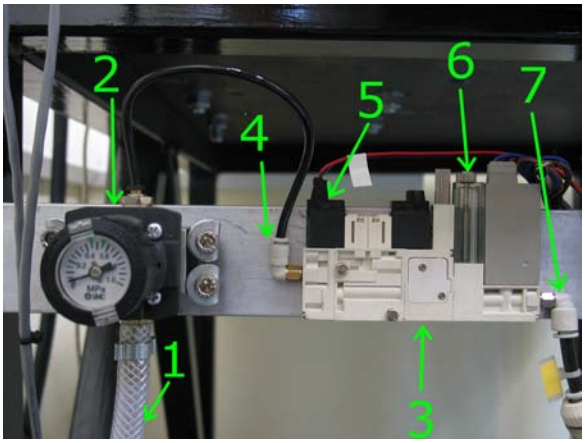


Figure 6.5: Air pressure regulator and Vacuum Ejector .

Table 6.1: Item list for Figure 3.4.

Item	Parts name
1	Pressured air hose
2	Pressure regulator
3	Vacuum Enjector
4	Pressure port (input)
5	Solenoid valve
6	Suction filter
7	Vacuum port (output)

6.3 Vibrating surface

In a pick and place application, using a digital camera for recognising the presence and position of an object in the workspace, two problems that can occur are the

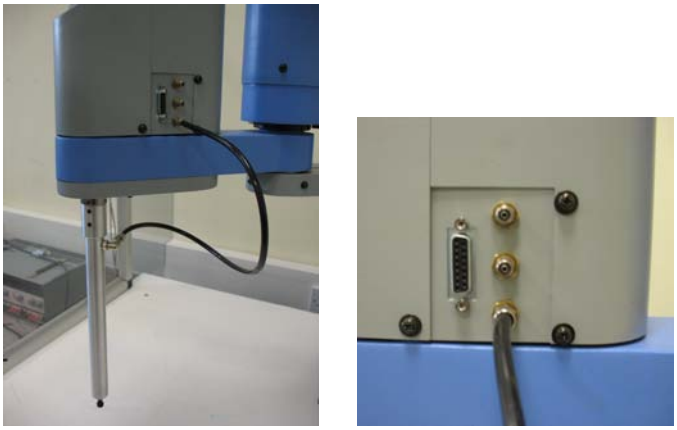


Figure 6.6: Hose second arm connection.

Figure 6.7: Vaccum cup (end effector).

overlap between objects, and touching objects. Indeed if this happens, the system cannot recognise the shape of the target object. In order to fix this, a vibrating surface has been developed. It's a black metal sheet (50 x 30 cm) with a hinge (see Figure 6.8). The vibration action is carried out by a DC motor with an asymmetric load that can be switched on or off using the specific Matlab robot function `out`.

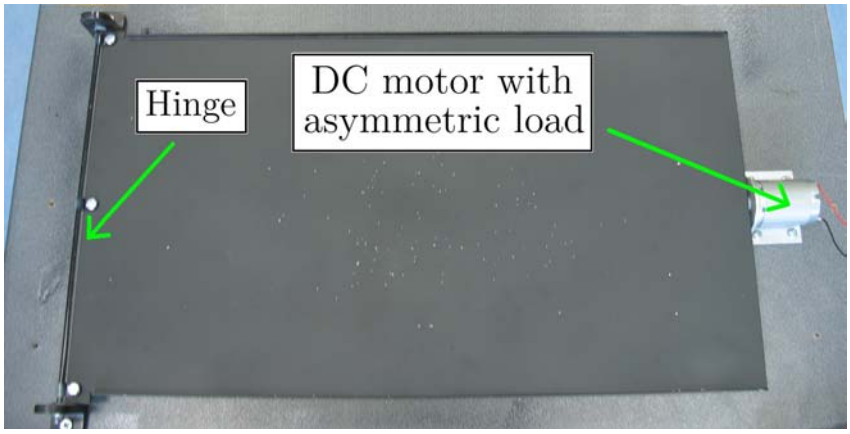


Figure 6.8: Vibrating surface.

6.4 Vacuum ejector and motor drive circuit

6.4.1 Schematic diagram

In order to switch on and off the vacuum ejector solenoid valve and the DC motor of the vibrating surface, a simple drive circuit is necessary. Two relays have been used so that the robot controller is insulated from the two devices that have to be driven. The schematic diagram of the circuit is shown in Figure 6.9. See Table 6.2 for parts description. The relays and connectors are shown in Figure 6.10.

Table 6.2: Item list for Figure 6.9.

Parts	Function
P1	Switch inside the robot controller. It can be opened or closed with Matlab robot function `out`.
K1	Relay for the vacuum ejector solenoid valve.
EV1	Vacuum ejector solenoid valve.
P2	Switch inside the robot controller. It can be opened or closed with Matlab robot function `out`.
K2	Relay for the DC motor of the vibrating surface.
M1	Vibrating surface DC motor.

6.4.2 P1 and P2

The push-button action represented by P1 and P2 in the Figure 6.9, is carried out by two BJTs inside the EX. I/O-2 module of the robot controller. See Figure 6.11. Only the pin couples 1-20 and 2-21 of the HDCB-37P connector are involved.

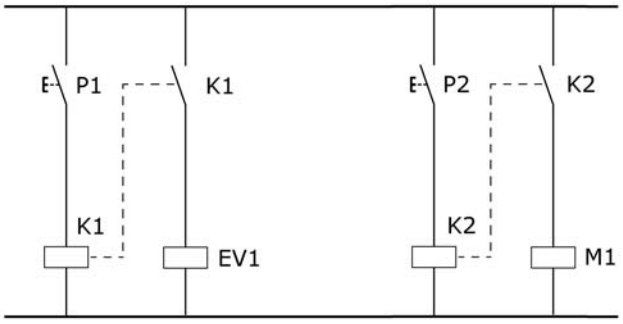


Figure 6.9: Vacuum ejector and motor drive circuit schematic diagram.

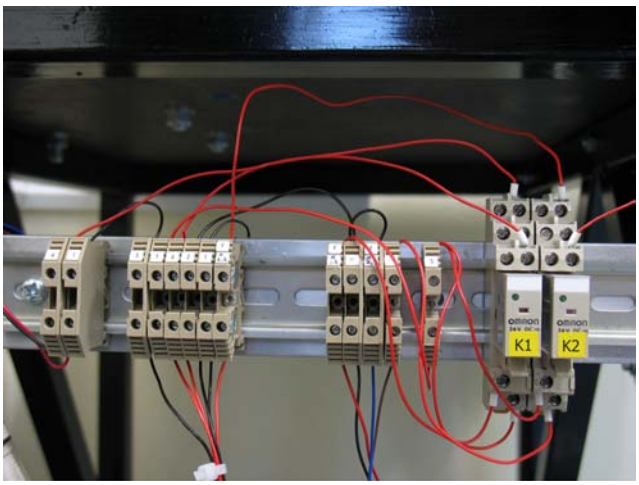


Figure 6.10: Relays K1, K2, and connectors.

The Matlab robot function `out` is used in order to switch on and off the BJTs, i.e. the relays. See Table 6.3.

Table 6.3: Relays switch commands.

Relays	ON	OFF
K1 (Valve)	`out(937,1)`	`out(937,0)`
K2 (Motor)	`out(938,1)`	`out(938,0)`

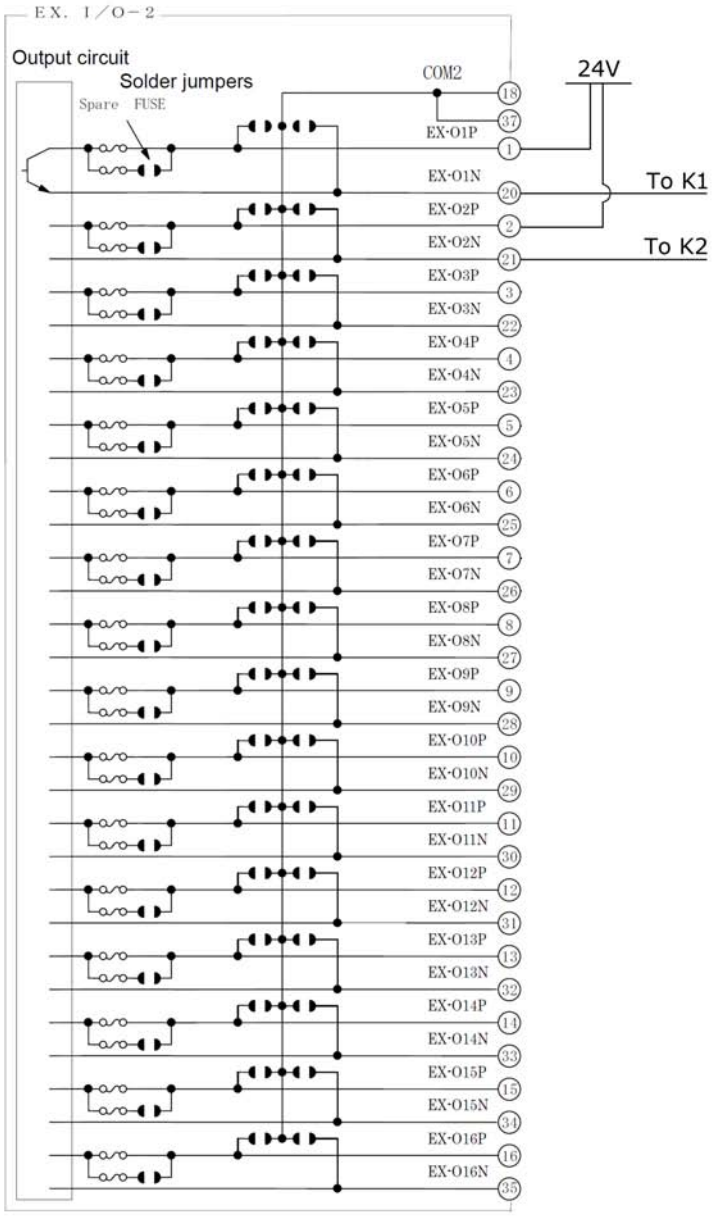


Figure 6.11: EX. I/O-2 connector.

6.5 Keys pick and place application

6.5.1 Aim of the application

Starting from a random placement of nine keys (see Figure 6.12), the robot has to pick them up and place them down one by one, with the longest axes aligned in the same manner (see Figure 6.13). The robot has to perform this operation autonomously by using a digital camera and a vacuum gripping system.

Figure 6.12: Random key placement.

Figure 6.13: Ordered key placement.

6.5.2 Keys detection

The Matlab application function developed for keys detection is `keys_detection`. See the flowchart in Figure 6.14 . The complete source code is shown in Listing A.3 in appendix A. The function `keys_detection` exploits two local functions defined in the same m-file: `opt_threshold_detection` and `image_processing`. The first one is for detecting the optimum BW threshold used in colour to BW conversion. Indeed, the light condition can frequently change, and BW threshold has to change consequently in order to provide the best image conversion. If the threshold is not correct, a key could not be recognised. The function `image_processing` reduces noise, clears white border, and makes the shape of possible keys clearer in an image, by using Matlab Image Processing Toolbox functions.

The BW image obtained by using the `opt_threshold_detection` threshold for a generic keys configuration, is shown in Figure 6.15, and the same image is shown in Figure 6.16 after `image_processing` action. It can be seen that the keys in the Figure 6.16 are very well defined white spots on a black background. The white spot on the left hand side of the image is the DC motor.

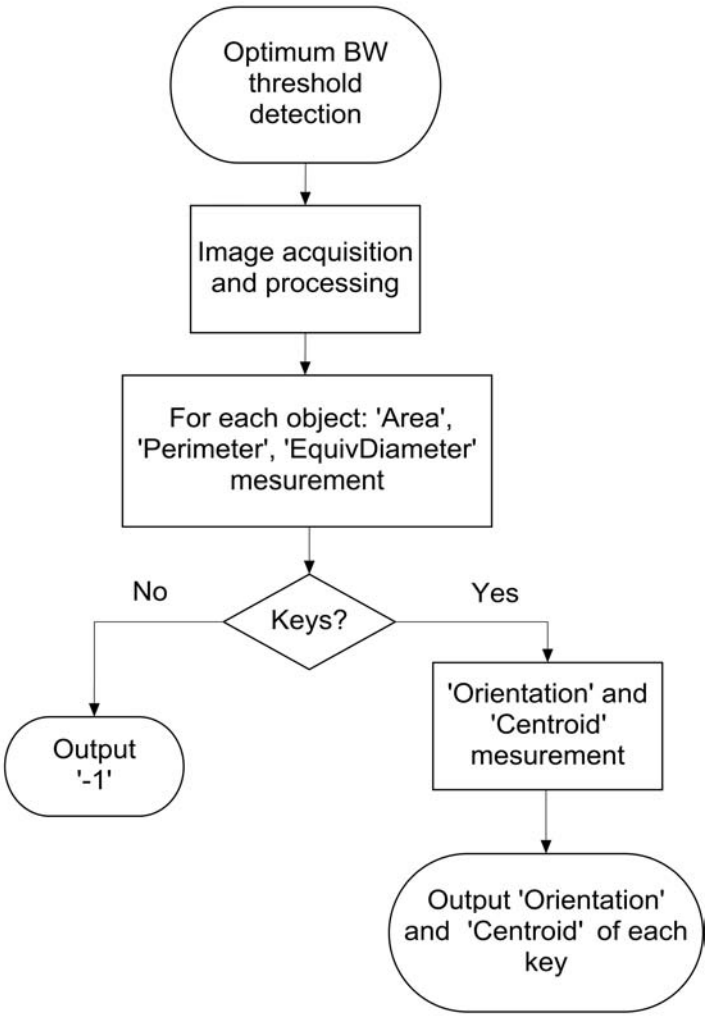


Figure 6.14: Function `keys_detection` flowchart.

The keys detection is directly carried out in `keys_detection`, starting from the image given by `image_processing` (see Figure 6.16 again). Two Matlab Image Processing Toolbox functions are used: `bwlabel` and `regionprops` [25]. Considering the `key_detection` piece of code of Listing 6.2, the function `bwlabel` returns a matrix `L`, of the same size as the image given by `image_processing`, containing labels for the connected objects in M1 (BW image). n is the number of objects. The function `regionprops` measures a set of properties for each labeled region in the label matrix L and stores them in the matrix `stats`. Twenty two different properties can be measured. Since the measuring operation takes time, only six of them have been measured in this application:

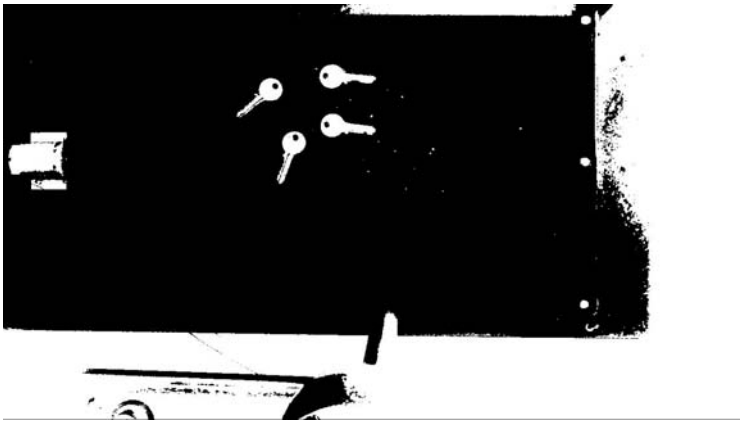


Figure 6.15: BW image obtained by using `opt_threshold_detection` threshold.

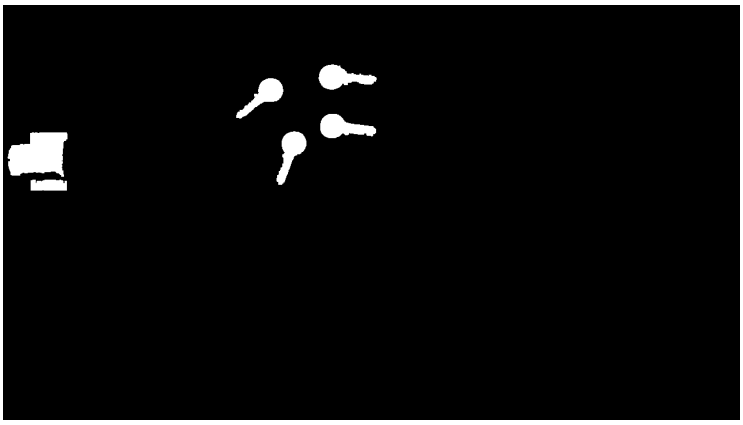


Figure 6.16: Image obtained by using the function `image_processing`.

- `'Area'` (Scalar): the actual number of pixels in the region [25];

- `'Centroid'`: 1-by-Q vector that specifies the center of mass of the region. Note that the first element of Centroid is the horizontal coordinate (or x-coordinate) of the center of mass, and the second element is the vertical coordinate (or y-coordinate). All other elements of Centroid are in order of dimension [25];

- `'BoundixBox'`: the smallest rectangle containing the region, a 1-by-Q2 vector, where Q is the number of image dimensions, and BoundingBox is `[ul_corner ... width]`, where: `ul_corner` is in the form `[x y z ...]` and specifies the upper-left corner of the bounding box, `width` is in the form `[x_width y_width ...]` and specifies the width of the bounding box along each dimension [25];

- `'Orientation'` (Scalar): the angle (in degrees ranging from -90 to 90 degrees)

between the x-axis and the major axis of the ellipse that has the same second-moments as the region [25];

- `'MajorAxisLength'` (Scalar): specifies the length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region [25];

- `'MinorAxisLength'` (Scalar): Scalar; the length (in pixels) of the minor axis of the ellipse that has the same normalized second central moments as the region [25];

- `'Perimeter'` (Scalar): the distance around the boundary of the region. The function `regionprops` computes the perimeter by calculating the distance between each adjoining pair of pixels around the border of the region [25].

Listing 6.2: Function `keys_detection` piece of code.

```
M1=image_processing(M1)
[L n] = bwlabel(M1);

stats = regionprops(L,'Area', 'Centroid', 'BoundingBox',
'Orientation', 'MajorAxisLength',
'MinorAxisLength', 'Perimeter');
```

A key pattern has been determined by setting a range of values for some of these properties, see Table 6.4. These ranges have been obtained after many experimental tests and the resulting findings obtained are significant and necessary for the following three reasons:

- the nine keys are similar, but not completely equal;

- the light condition changes, thus BW images obtained by coloured images that have been acquired in different moments, can have small differences even if the keys configuration is the same;

- prospective effects.

Table 6.4: Properties range of values defining a key pattern.

Property	Range of values
`'Area'`	2600 - 3100
`'Perimeter'`	490 - 670
`'MajorAxisLength'`	106 - 115
`'MinorAxisLength'`	40 - 43

For each object stored in `stats`, an `if` statement detects if all these four properties are verified. Only if this happens, and that means the object is a key, the `'Centroid'` and `'Orientation'` of the key are determined. Then, the `'BoundingBox'` property is used to detect if the key bow points towards the top or the bottom of the image. After that, the `'Centroid'` is translated along key major axis by a few millimetres, in order to allow a better grab by the vacuum cup. The translated `'Centroid'` positions for a generic keys configuration are shown in Figure 6.17. Finally, the `'Orientation'`, the key bow orientation information, and the translated `'Centroid'` coordinates of each key are stored in a matrix and outputted.

Figure 6.17: Translated `'Centroid'` positions for a generic keys configuration.

6.5.3 Application execution

The application is executed by the Matlab script `keys_pick_and_place`, see the flowchart in Figure 6.18. The source code is shown in Listing A.4 in appendix A. First of all, this script moves the robot to `image_acq_pos`, i.e. the position from where images are acquired. Then, the program vibrates for 0.8 seconds the surface where keys are placed on. This is for separating overlapped keys and touching keys. After that, an image is acquired and processed by using `keys_detection` (see previous section). If at a minimum one disc has been detected, a pick and place cycle is carried out in order to move all the detected keys. The pick and place operation in this cycle is made up by many steps:

1) function `position` (see section 6.1.3) is called in order to detect the key centroid Cartesian coordinates in X-Y plane;

2) the robot moves above the key centroid;

3) the gripping vacuum system is switched on;

4) the robot lowers the Z-axis in order to grab the key by using the vacuum cup;

5) the key is grabbed by the vacuum cup;

6) the robot lifts up the Z-axis;

7) the rotation, needed to align the key major axis along the specified direction, is calculated;

8) the key place position is calculated;

9) the rotation of the key is carried out while the robot moves above the placement position;

10) the robot lowers the Z-axis in order to release the key;

11) the gripping vacuum system is switched off;

12) the key is released (a pause of 0.2 seconds is necessary in order to complete the release operation).

After the execution of a pick and place cycle, a new image acquisition and processing is performed. Indeed, some keys can still be on the vibrating surface. That happens because keys can remain or become overlapped or there are touching keys even after the vibration action. Furthermore, sometimes the light reflection and the light condition can effect the image capture of a key in a specific position. Anyway, if after four vibration actions, and image acquisition and processing, no keys have been detected, the software detects that the execution is completed. The four times vibration repeat has been decided after many experimental tests.
An example of the application execution is shown in Figures 6.19, 6.20, 6.21, 6.22, 6.23, 6.24, 6.25, 6.26, 6.27.

6.5.4 Conclusion

It turns out a very reliable application, even if the light condition changes before, or during the execution. Precision results have been very good, considering the difference in the profiles of the keys and the loss of precision due to perspective. Spending more time in testing and setting operations, a very good result would be certain. The speed of the robot has not been increased over the 50% of the maximum speed due to the vibrations of the structure on which the robot is fixed that could damage the manipulator. The time the application takes, is not constant because of the random keys configurations before and after vibrating actions. Anyway, normally the execution takes roughly 39 seconds.

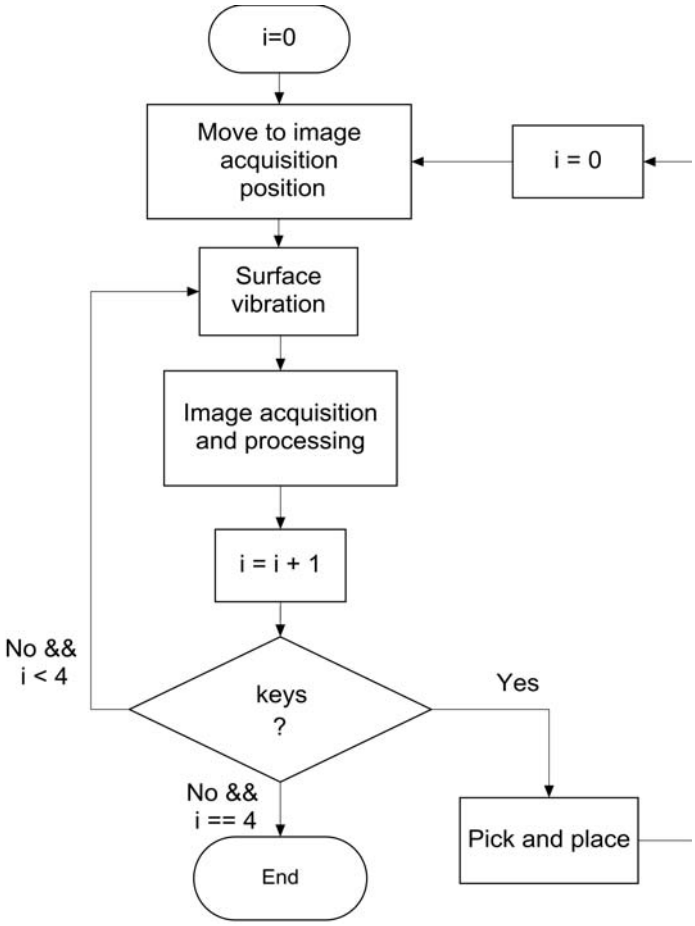


Figure 6.18: Function `keys_pick_and_place` flowchart.

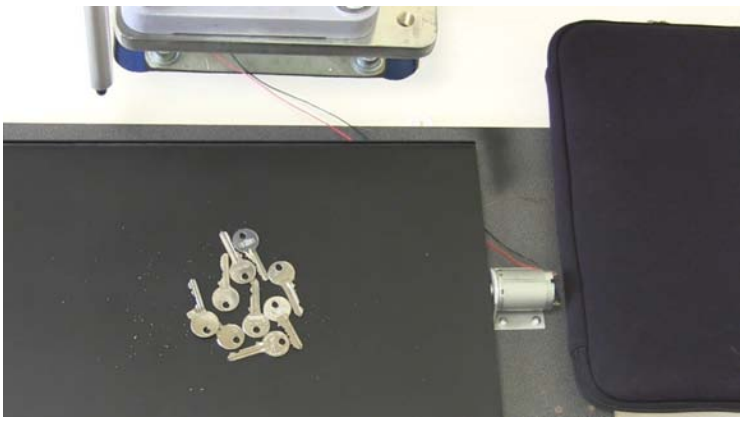


Figure 6.19: Nine key configuration before vibrating.

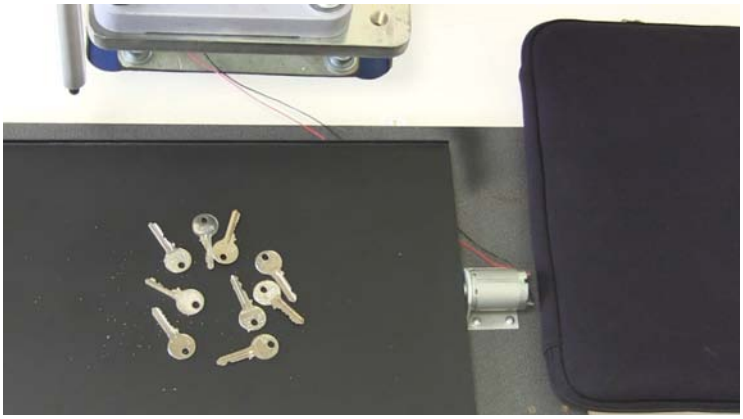


Figure 6.20: Nine key configuration after vibrating.

Figure 6.21: First key pick up action.

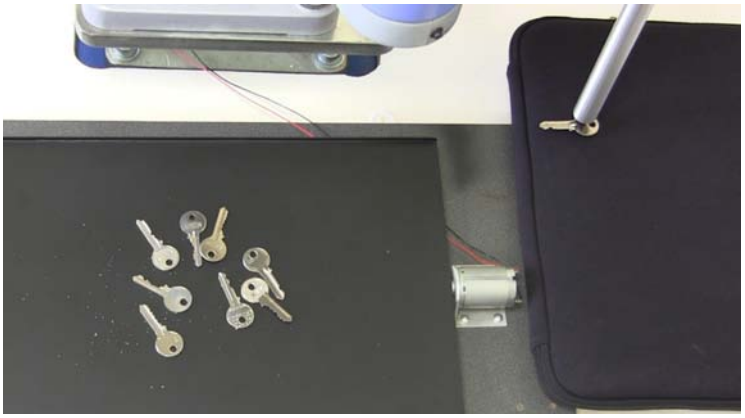


Figure 6.22: First key place down action.

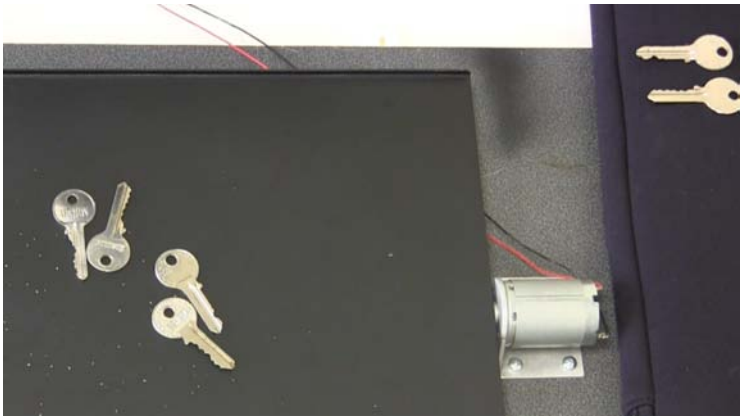


Figure 6.23: Key configuration after nine keys picked up.

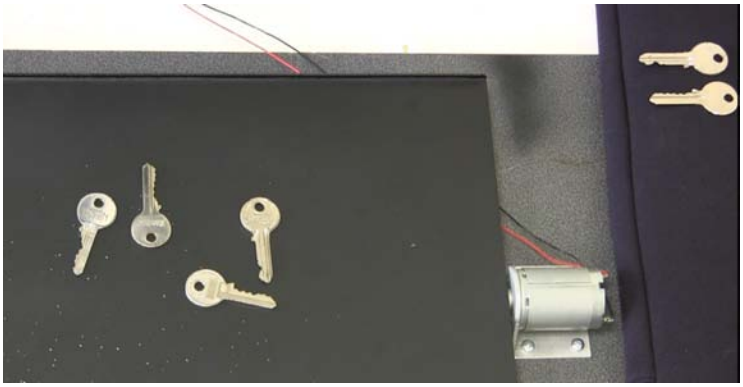


Figure 6.24: Placement of four keys before vibrating.

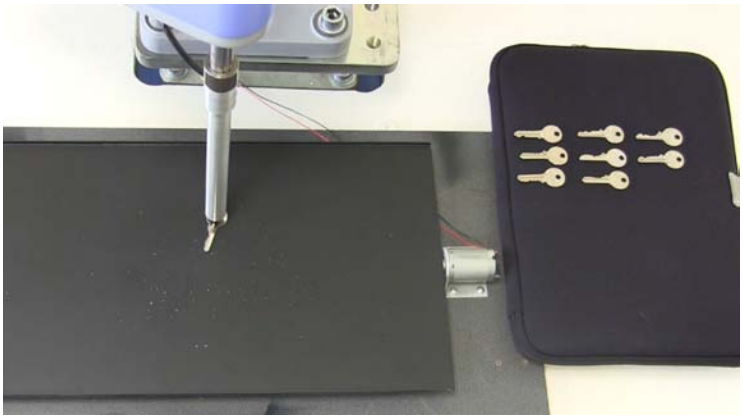


Figure 6.25: Ninth key pick up action.

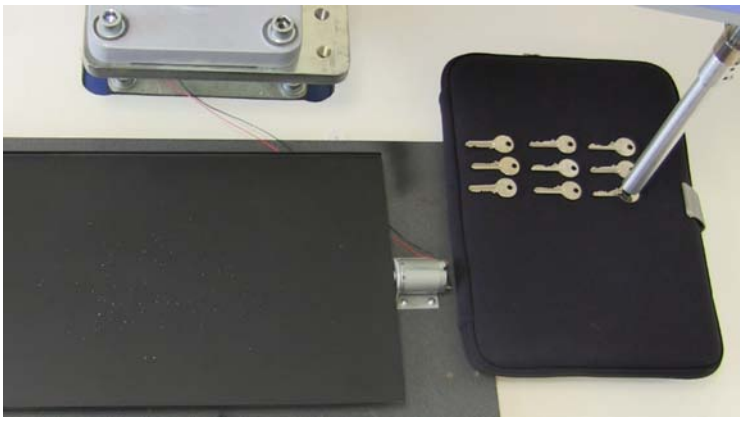


Figure 6.26: Ninth key place down action.

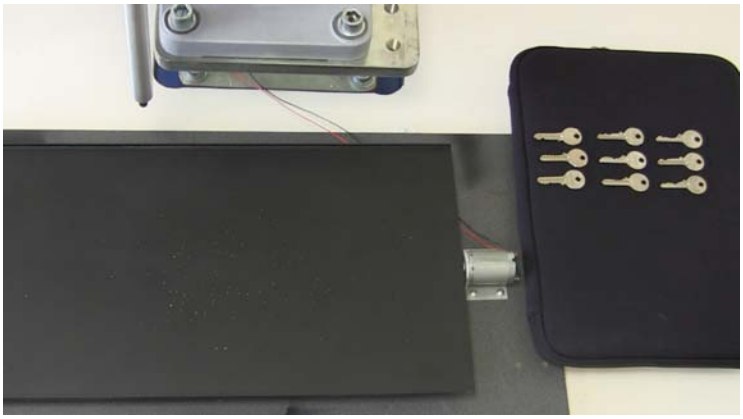


Figure 6.27: The execution is completed.

6.6 Coloured discs pick and place application

6.6.1 Aim of the application

Starting from a random placement of about thirty coloured discs (the exact number is not important), the robot has to pick them up and place them down one by one, in four small containers depending on the colour. The colours are: white, red, green, and blue. See Figures 6.28.

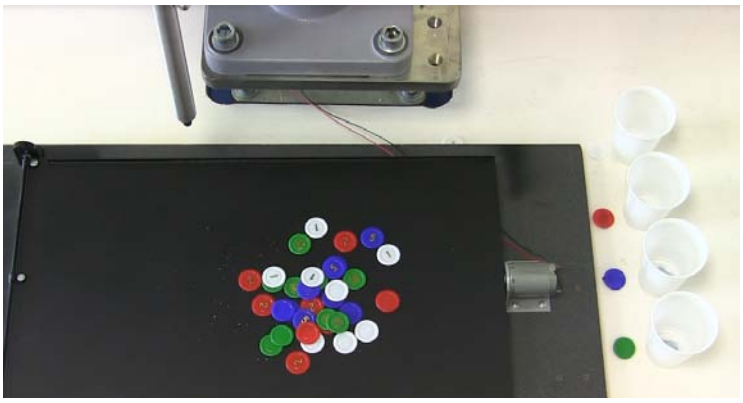


Figure 6.28: Random coloured discs placement and containers.

6.6.2 Discs detection

The Matlab application function developed for detection of discs and colours detection is `discs&colours_detection`. The complete source code is shown in Listing A.5 in appendix A. The structure of this function is quite similar to that of the function `keys_detection` seen in section 6.5.2. It exploits three local functions defined in the same m-file: `opt_threshold_detection`, `image_processing`, and `colour_detection`. The first one is for detecting the optimum BW threshold used in colour to BW conversion. The function `image_processing` reduces noise, clears white border, and makes the shapes of possible keys clearer in an image, by using Matlab Image Processing Toolbox functions [25]. The function `colour_detection` is for detecting the colour of a disc.

Different colours means a different contrast with the black background of the vibrating surface. For instance, if the acquired colour image were converted directly into a BW image, probably some blue or red discs would not be detected.
Considering the `discs&colours_detection` piece of code of Listing 6.3, the acquired true colour image `M` (see Figure 6.29) is split into RGB (Red, Green, and Blue)

channels `M1` (see Figure 6.30), `M2` (see Figure 6.31), and `M3` (see Figure 6.32). After that, each channel is converted to BW image by using the optimum threshold given by `opt_threshold_detection` (see Figures 6.33, 6.34, and 6.35). These three BW images are then combined by using an OR operation obtaining the complete BW image (see Figure 6.36). The final image is provided by `image_processing` and it is shown in Figure 6.37.

Listing 6.3: Function `discs&colours_detection` piece of code.

```
1
2  M1 = M(:,:,1);
3  M2 = M(:,:,2);
4  M3 = M(:,:,3);
5
6  M1=im2bw(M1,opt_thr);
7  M2=im2bw(M2,opt_thr);
8  M3=im2bw(M3,opt_thr);
9
10 M=M1 | M2 | M3;
11
12 M = image_processing(M)
13
14 [L n] = bwlabel(M1);
15
16 stats = regionprops(L,'Area', 'Centroid',
17 'Orientation', 'MajorAxisLength',
18 'MinorAxisLength', 'Perimeter');
```

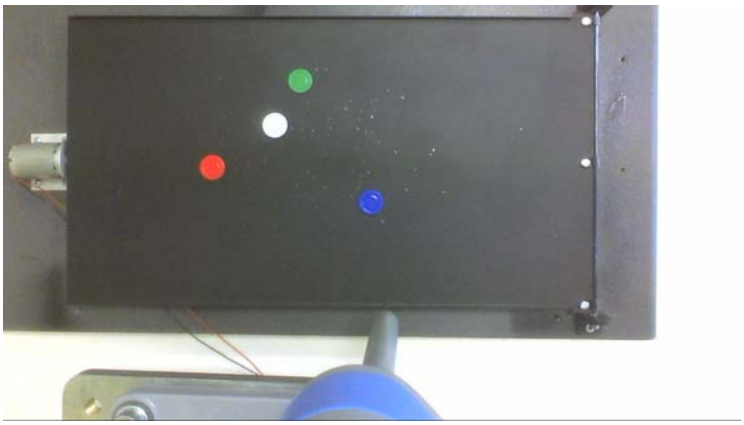


Figure 6.29: True colour image.

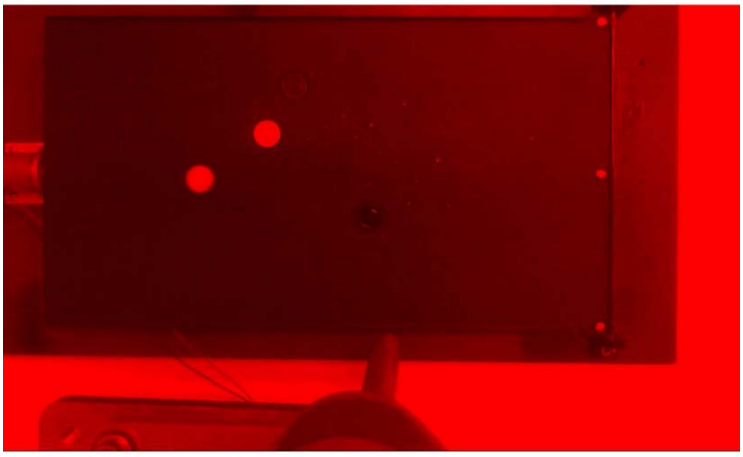


Figure 6.30: Red chanel.

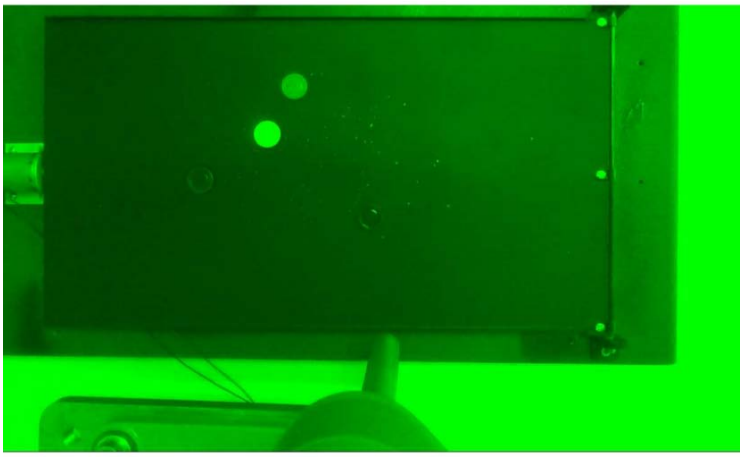


Figure 6.31: Green channel.

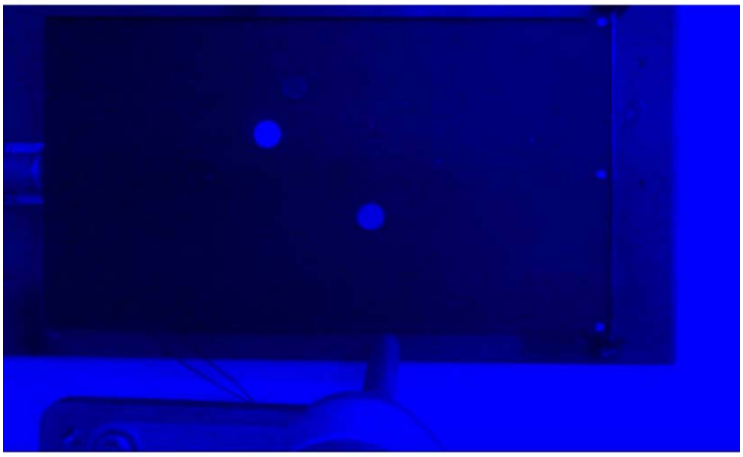


Figure 6.32: Blue channel.

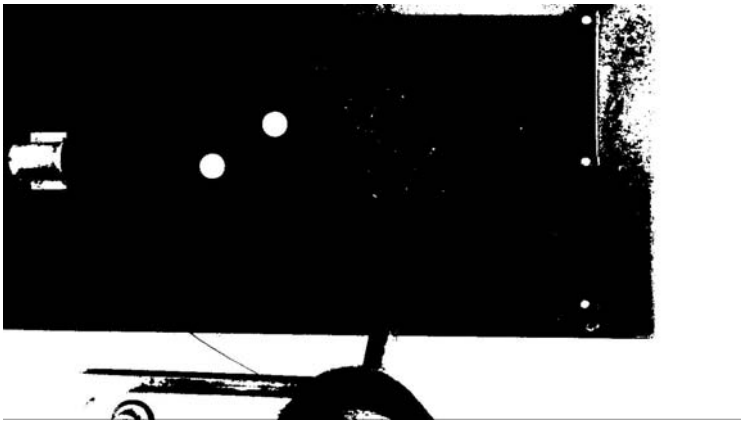


Figure 6.33: Red channel BW conversion.

Figure 6.34: Green channel BW conversion.

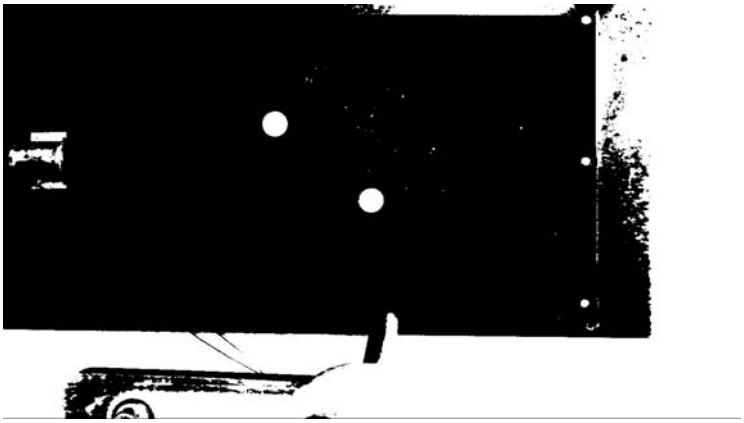


Figure 6.35: Blue channel BW conversion.

Figure 6.36: Image after OR operation between channels.

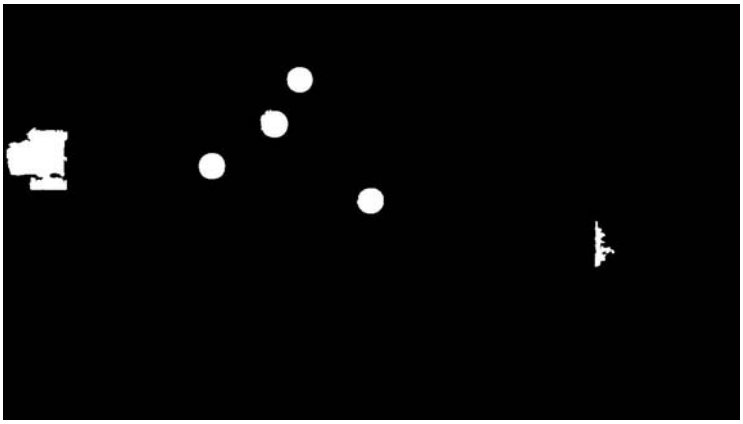


Figure 6.37: Processed image.

The discs detection is directly carried out in `discs&colours_detection`, starting from the image given by `image_processing` (see Figure 6.37 again). As in `keys_detection`, the two Matlab Image Processing Toolbox functions `bwlabel` and `regionprops` are used.
A disc pattern has been determined by setting a range of values for the properties `'Area'`, `'Perimeter'`, `'MajorAxisLength'`, and `'MinorAxisLength'`. See Table 6.5.

Table 6.5: Properties range of values defining a disc pattern.

Property	Range of values
`'Area'`	1950 - 2200
`'Perimeter'`	430 - 470
`'MajorAxisLength'`	51 - 56
`'MinorAxisLength'`	49 - 54

For each object stored in `stats`, an `if` statement detects if all these four properties are verified. Only if this happens, and that means the object is a disc, the `'Centroid'` and colour of the disc are determined. The `'Centroid'` positions for a generic discs configuration is shown in Figure 6.38.

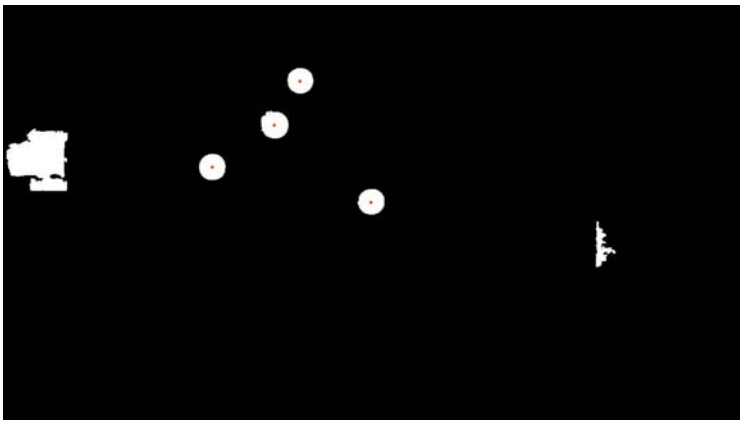


Figure 6.38: Translated `'Centroid'` positions for a generic discs configuration.

Colour detection

In RGB (Red, Green, Blue) channels `M1`, `M2`, `M3`, the pixel colour value is an integer between 0 and 255. For example, considering the pixel (100,100) of the colour image `M`, if `M1(100,100)== 250`, `M2(100,100)== 10`, and `M3(100,100)== 10` the pixel (x,y) in `M` is almost red. It would be true red if `M1(100,100)== 255`, `M2(100,100)== 0`,

and `M3(100,100)== 0`. The function `discs&colours_detection` calls a local function `colour_detection` in order to recognise the colour of each disc. This function uses four `if` statements. Each statement compares the colour value of a tested pixel `(x,y)` with three thresholds, one for each channel. After many experimental tests, it has been deduced that a pixel `(x,y)` in the true colour image (see Figure 6.29) is:

- **white**: if `M1(x,y)> 180` & `M2(x,y)> 200` & `M3(x,y)> 200`;
- **red**: if `M1(x,y)> 200` & `M2(x,y)< 150` & `M3(x,y)< 150`;
- **blue**: if `M1(x,y)< 120` & `M2(x,y)< 120` & `M3(x,y)> 150`;
- **green**: if `M1(x,y)< 130` & `M2(x,y)> 150` & `M3(x,y)< 130`;

Where the tested pixel `(x,y)` is not the centroid but another one. This is necessary due to the fact that a gold number is printed on the centre of one side of each disc. This function is called twice in order to perform a double check. If the output colour detection of the two calls are different, or one of them detects that the colour does not lie in any of the previous ranges, the disc is not taken into account.

6.6.3 Application execution

The application is executed by the Matlab script `coloured_discs_pick_and_place` The source code is shown in Listing A.6 in appendix A. First of all, this script moves the robot to `image_acq_pos`, i.e. the position images are acquired from. Then, the program for 0.8 seconds vibrates the surface where discs are placed on. This is for separating overlapped discs and touching discs. After that, an image is acquired and processed by using `discs&colours_detection` (see previous subsection). If, at a minimum, one disc has been detected, a pick and place cycle is carried out in order to move all the detected discs. The pick and place operation in this cycle is made up by many steps:

1) function `position` (see section) is called in order to detect the disc centroid Cartesian coordinates in X-Y plane;
2) the robot moves above the disc centroid;
3) the gripping vacuum system is switched on;
4) the robot lowers the Z-axis in order to grab the disc by using the vacuum cup;
5) the disc is grabbed by the vacuum cup;
6) the robot lifts up the Z-axis;

7) the robot moves above the place position that depends on the detected colour of the disc;

8) the robot lowers the Z-axis in order to release the disc;

9) the gripping vacuum system is switched off;

10) the disc is released (a pause of 0.2 seconds is necessary in order to complete the release operation);

After the execution of a pick and place cycle, a new image acquisition and processing is performed. Indeed, some discs can still be on the vibrating surface. That happens because discs can remain or become overlapped and touching, even after the vibration action. Furthermore, the reflection and condition of the light can effect the image capture of a disc in a specific position. If after four vibration actions, and image acquisition and processing, no discs have been detected, the software detects that the execution is completed. The four times vibration repeat has been decided after many experimental tests. An example of the application execution is shown in Figures 6.39, 6.40, 6.41, 6.42, 6.43, 6.44, 6.45, 6.46, 6.47, 6.48, 6.49.

6.6.4 Conclusion

It turns out a very reliable application, even in changing light conditions, before or during the execution. The speed of the robot has not been increased over the 50% of the maximum speed due to the vibrations of the structure on which the robot is fixed that could damage the manipulator. The time the application takes, is not constant because of the random discs configurations before and after vibrating actions. Anyway, normally the execution with 34 discs takes roughly 1 minute and 25 seconds.

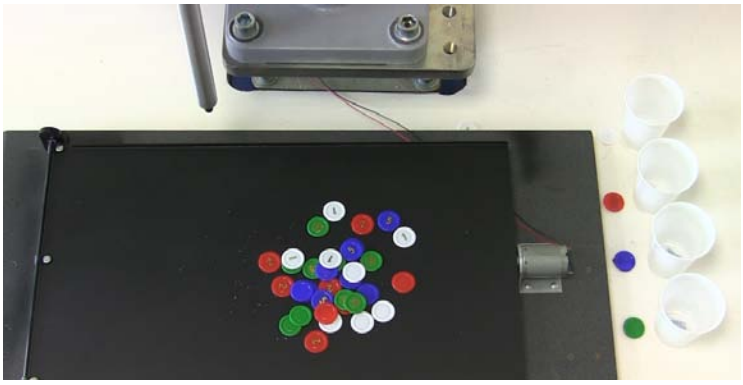


Figure 6.39: Random disc placement before vibrating.

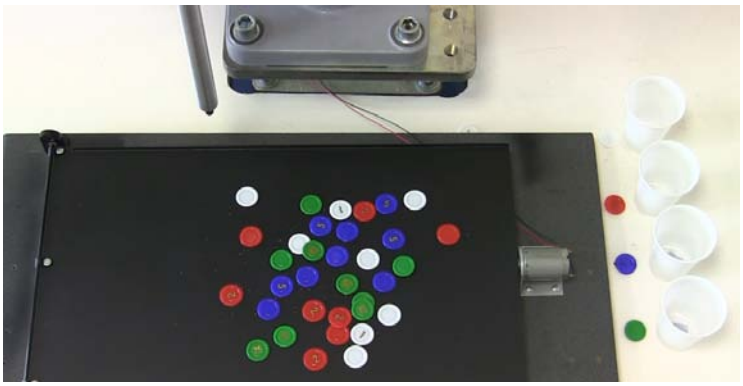


Figure 6.40: Random disc configuration after vibrating.

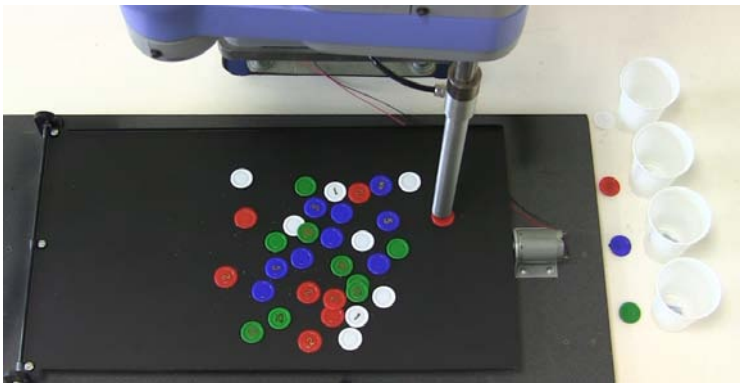


Figure 6.41: Red disc pick up operation.

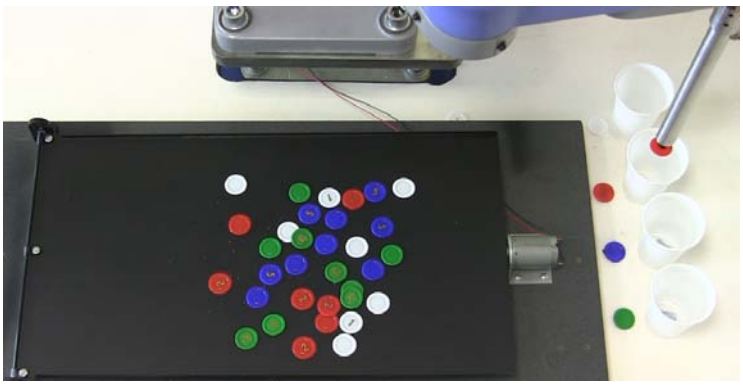


Figure 6.42: Red disc place down operation.

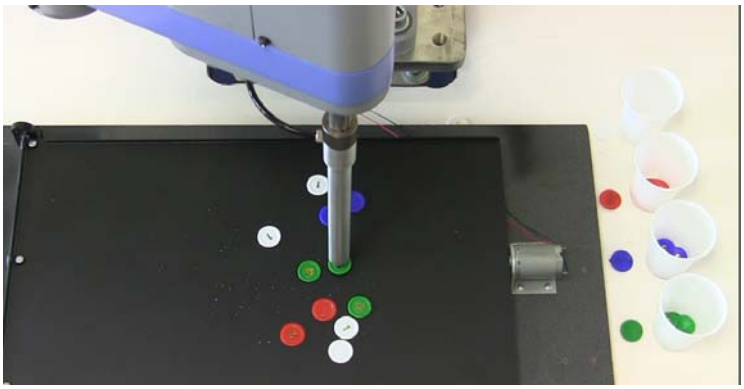


Figure 6.43: Green disc pick up operation.

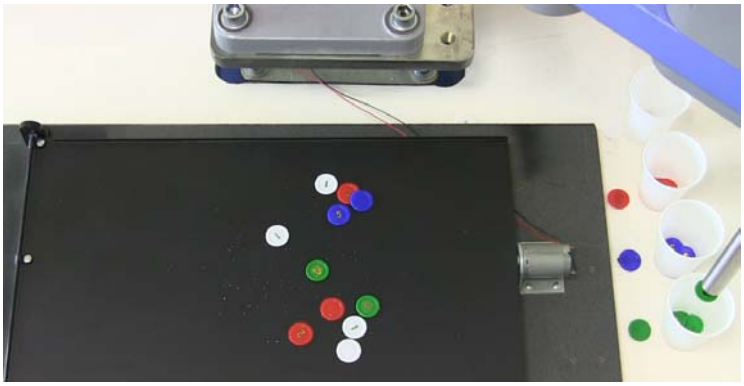


Figure 6.44: Green disc place down operation.

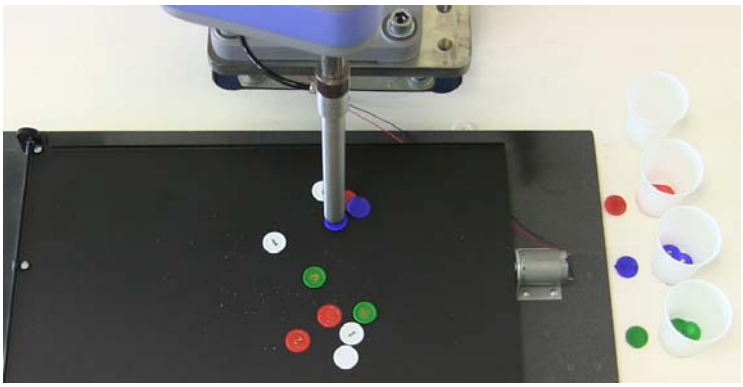


Figure 6.45: Blue disc pick up operation.

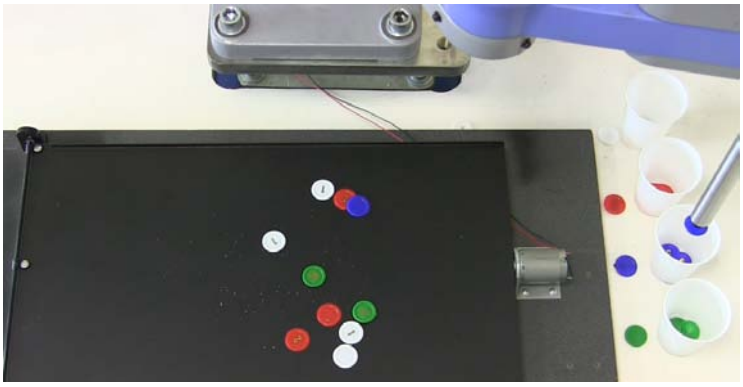


Figure 6.46: Blue disc place down operation.

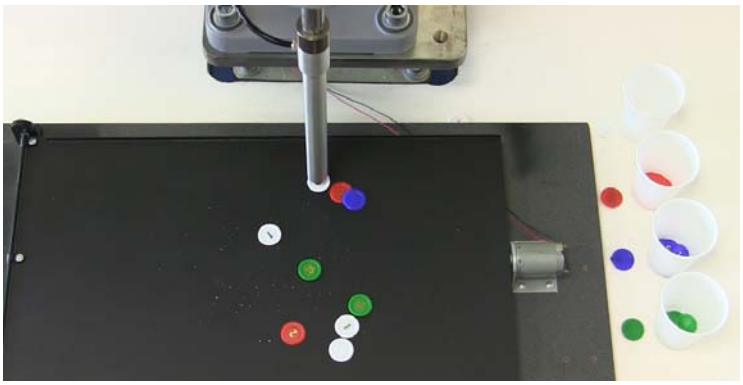


Figure 6.47: White disc pick up operation.

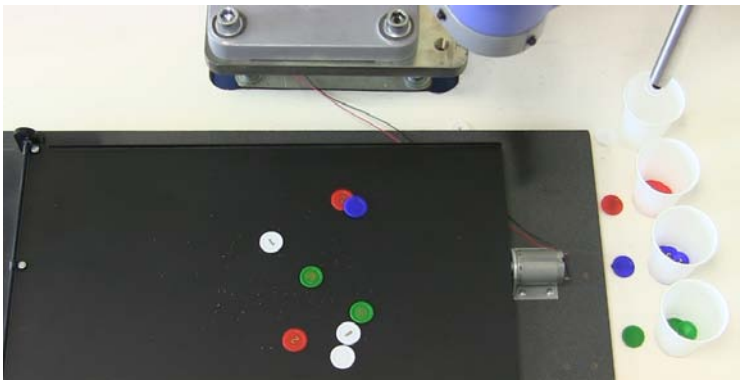


Figure 6.48: White disc place down operation.

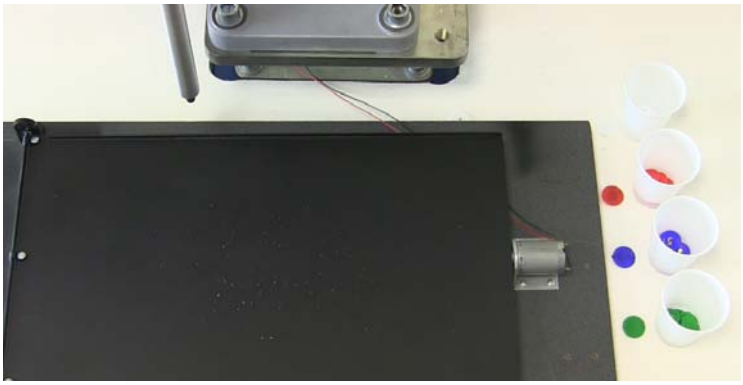


Figure 6.49: The execution is completed.

Chapter 7

Graphical User Interface

In order to allow a simple and user-friendly control of the robot, three GUIs (Graphical User Interfaces) [8] have been developed:

1) G1: GUI for selecting G1 or G2 (see Figure 7.1);

2) G2: GUI for controlling the robot in the Cartesian Coordinate System (see Figure 7.2);

3) G3: GUI for controlling the robot in the joint Coordinate System (see Figure 7.3);

Although the control options are limited, these tools are extremely useful, expecially for non expert users. GUIs have been designed with a Matlab tool called GUIDE. This software, after panel compiling, creates two types of files: m-files and fig-files. The m-files contain MATLAB commands to initialise the GUI and the GUI callbacks. The callbacks are the routines that execute when a user interacts with a GUI component: pressing a screen button, clicking a mouse button, selecting a menu item, typing a string or a numeric value, or passing the cursor over a component. Code is added to the callbacks to perform the functions that are required. The fig-files contain a full description of GUI layout and GUI components such as push buttons, axes, panels, menus, etc. G1 is launched by typing `GUI` in the command window and executing it.

The G2 parts are listed in Table 7.1. The G3 parts are listed in Table 7.2.

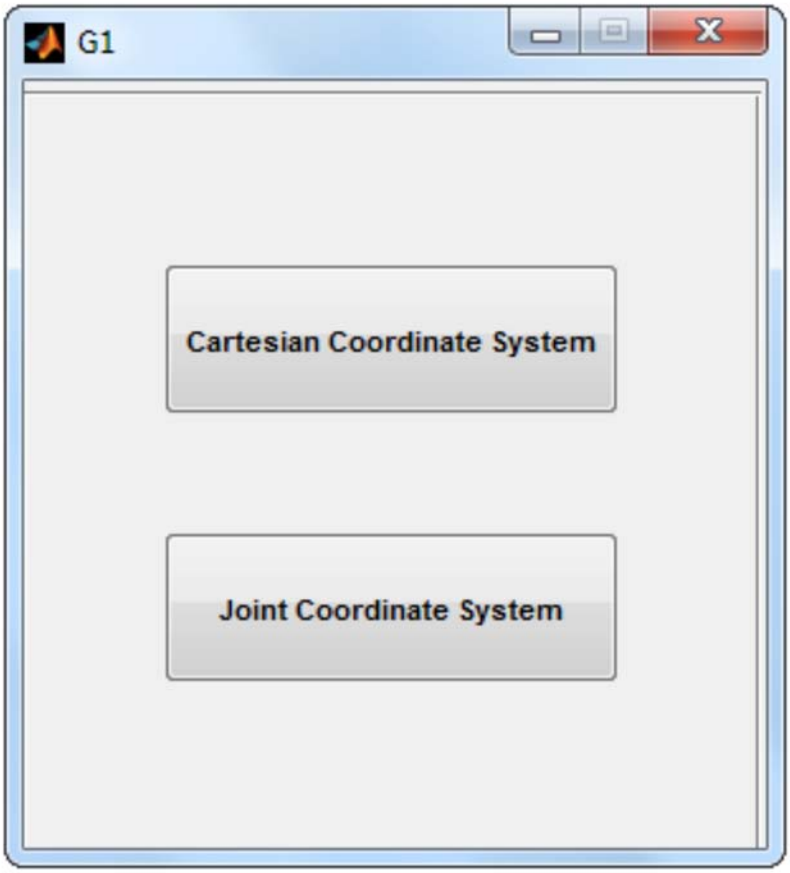


Figure 7.1: GUI G1.

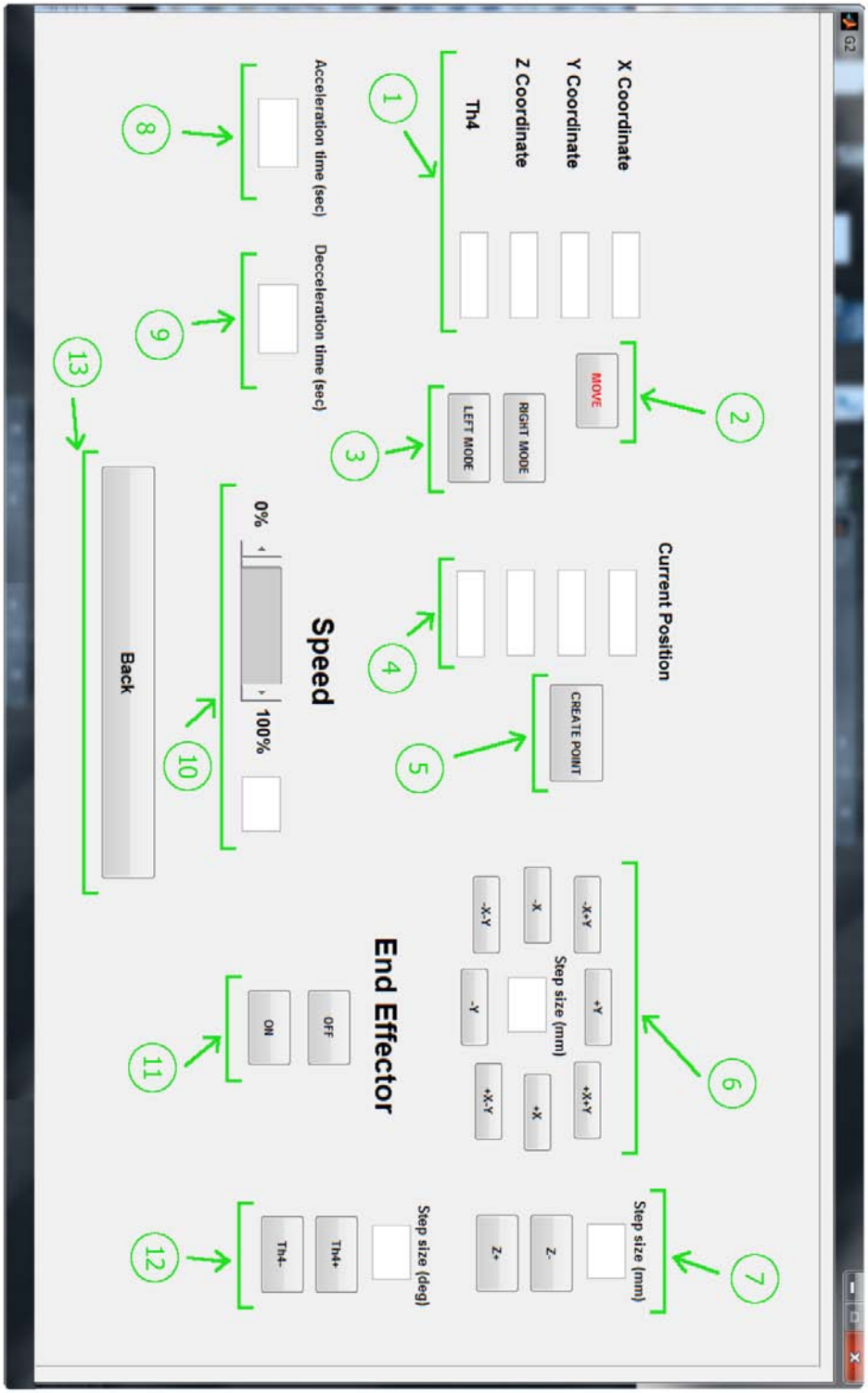


Figure 7.2: GUI G2.

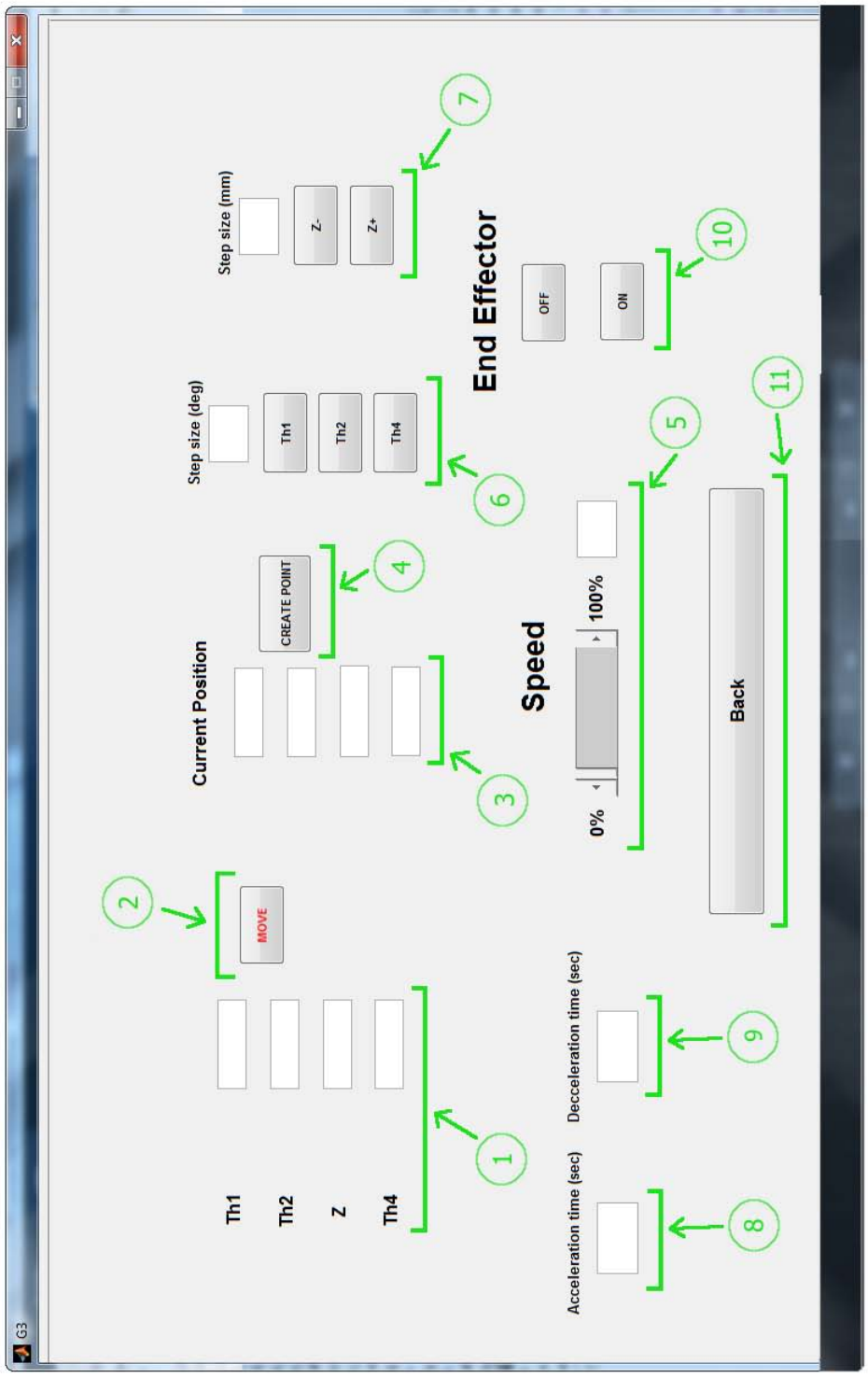


Figure 7.3: GUI G3.

Table 7.1: Item list for Figure 7.2.

	Part	Function
1	Coordinates input textboxes	The user can input the Cartesian coordinates of the target point.
2	Move button	When it is pressed, the robot moves to the target position specified in the coordinates input text boxes (PTP motion).
3	Arm mode buttons	Pressing one of these buttons, the user can select the arm mode (left or right).
4	Current position text boxes	Show the current position.
5	Point creation button	Pressing this button, the current position is output to the command window.
6	X-Y step motion control	Allows PTP motion in X-Y plane along 8 directions. The step of the motion can be input in a text box.
7	Z step motion	Allows Z axis PTP motion. The step of the motion can be input in a text box.
8	Acceleration time text box	The user can input the acceleration time. If this box is left empty, then a default value of 0.1 second is considered.
9	Decceleration time text box	The user can input the decceleration time. If this box is left empty, then a default value of 0.1 second is considered.
10	Speed control slide bar	The user can set the speed of the PTP motion in terms of percentage. I.e. 100% speed would refer to the maximum speed attained by the robot. The default setting is 10% of the maximum speed.
11	Vacuum ON and OFF buttons	For activating and disactivating the vacuum gripping system
11	Th4 step motion.	Allows Th4 PTP motion. The step of the motion can be input in a text box.
12	Back button.	Allows the user to return to G1.

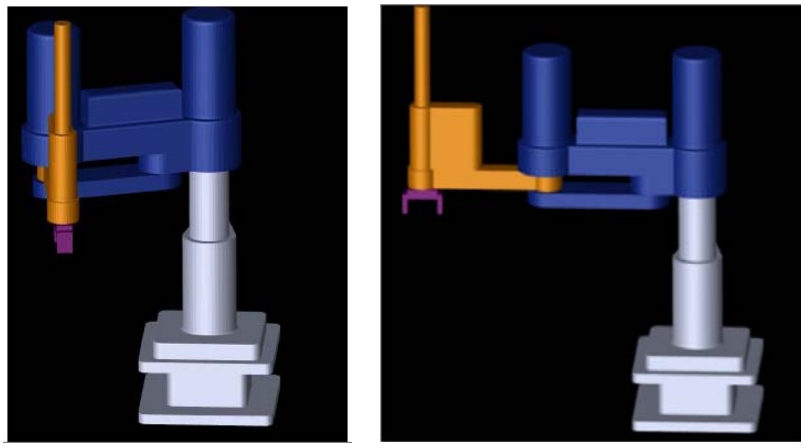
Table 7.2: Item list for Figure 7.3.

	Part	Function
1	Coordinates input text boxes	The user can input the joints coordinates of the target point.
2	Move button	When it is pressed, the robot moves to the target position specified in the coordinates input text boxes (PTP motion).
3	Current position text boxes	Show the current position.
4	Point creation button	Pressing this button the current position is output to the command window.
5	Speed control slide bar	The user can set the speed of the PTP motion in terms of percentage. I.e. 100% speed would refer to the maximum speed attained by the robot. The default setting is 10% of the maximum speed.
6	Step motion control	Allows Th1, Th2, Th3 PTP motion. The step of the motion can be input in a text box.
7	Z step motion	Allows Z axis PTP motion. The step of the motion can be input in a text box.
8	Acceleration time text box	The user can input the acceleration time. If this box is left empty, then a default value of 0.1 second is considered.
9	Decceleration time text box	The user can input the decceleration time. If this box is left empty, then a default value of 0.1 second is considered.
10	Vacuum ON and OFF buttons	For activating and disactivating the vacuum gripping system
11	Back button.	Allows the user to return to G1.

Chapter 8

Simulink virtual robot

One of the tasks of the project is the development of a Simulink virtual model of the robot. A UCC student, Milind Sudhir Rokade, has worked on that topic. The parts of the robot structure have been designed in Solidworks (a CAD software). This 3D model has then been exported to MATLAB/Simulink, using SimMechanics software. SimMechanics provides a multibody simulation environment for 3D mechanical systems, such as robots, pendulums, vehicle suspensions, and aircraft landing gear. The imported model carries properties like mass, inertia, joint, constraint and 3D geometry [26]. SimMechanics generates a 3D animation which lets the user visualize the system dynamics. The robot virtual model is shown in figure 8.1 (a) and (b).
A big effort has been made in order to integrate the control of the actual robot, with the control of the virtual robot. A folder, provided with the more significant Matlab robot functions, has been developed. This allows the control of both the physical and virtual robots, in a manner so that the motion of the virtual robot closely matches that of the physical Sankyo system.



(a) (b)

Figure 8.1: The robot virtual model.

Chapter 9

Conclusion and future work

Matlab is not designed for real-time applications. Moreover, the communication between the PC where Matlab is installed and the robot controller introduces a delay. For example, the reaction time of the system to an external event is about 40 ms rather than 4 ms of the original configuration (i.e. without Matlab control). These two facts show that the developed system is more suitable for prototyping than for industrial applications.
The flexibility of the robot has been increased. Indeed, it can communicate easy with other devices due to the intermediate action of a PC where Matlab is installed. Furthermore, new functions have been added to the robot such as the Trajectory/Angles Sampling Mode. Two GUIs (Graphical User Interfaces) allow the robot control by non expert users.
The user safety conditions are not effected by the Matlab control because the robot controller carries out the same supervision operations which it would perform with the usual direct control.
With Matlab, the robot programming results more comfortable. For instance, all the available functions are listed on the left side of the Matlab command window. Clicking on the function name, a brief description of the function is shown. The devolopment of two vision-based applications has been proved quite easy due to the functions provided by the Matlab Image Acquisition Toolbox, and the Matlab Image Processing Toolbox. These two applications can be a base for further developments.
From the collaboration with Milind Sudhir Rokade, the student who has developed a Simulink virtual model of the robot, it turns out that the integration between the actual robot control and the virtual robot control in Matlab, can open very interesting scenarios in terms of motion comparison and investigation of inertia problems.
Future work will also focus on the other tasks of the projects such as:

- design a gripper (pneumatic-based) so that the robot can pick up a part;

- construct a conveyor-based work cell to demonstrate the operation of a SCARA-based work-cell;
- design a software/hardware based system so that the robot can be controlled remotely (via the Internet);
- use two robots to operate cooperatively on a task.

Moreover, a micro camera may be mounted near the end effector. The video stream may be visualized on a GUI in order to have a better control and supervision of the end effector operations. Some microcontroller boards could be used in order to improve the efficiency and the supervision of the system.
On September 20 the company ODG Technologies carried out some tests on the SCARA robot in the UCC Mechatronic Laboratory by using the software developed in this project. This demonstrates that prototyping developed in this University-based project can be interesting for private entities. This kind of collaboration is probably one of the keys to innovation.
Two final folders have been created, both include all the Matlab functions developed in this project. In one of them the code is visible to the user. This will allow further developments. In the other folder, the developed Matlab robot functions and the Matlab auxiliary functions are hidden in p-files. A p-file is obtained from an m-file by using the Matlab `pcode` command, and its content is very hard to understand. This folder is intended for users who are not involved in the research project.

Appendix A

Code

Listing A.1: Function `serial_out1`

```
function [y] = serial_out1( Code, A)

% This function sends data to the robot controller on which
% the interpreter is running, and waits for a feedback
% message from the controller.
% PARAMETERS:
% Code:
% Identification code of the function in which serial_out1
% has been called.
% A: Data that has to be sent to the robot
% controller.
%
% The function state_keeper is called many times
% in order to verify if the transmission is allowed, and to
% update the state of the program.

s = ser('retrieve');    % Serial port object

% If the inner expression of the next statement is TRUE, it means
% "Start" on the pedant has not be selected and an error message
% is outputted

if(state_keeper('retrieve',2) == 0)

   if(state_keeper('retrieve',3) == 0)

      state_keeper('store',3,1);
      disp(' ')
      disp('Please be sure that "Start" on the pedant has been selected,')
      disp('then type the prog() statement to continue with programming,')
      disp('otherwise the next statement will be useless.')
      disp(' ')
      state_keeper('store',1,1);

   end

   else

      if(state_keeper('retrieve',4)≠0)

```

```
41          [n,m] = size(A);  % Size of the second input parameter
42
43          % The next statement converts the number 1111
44          % into a string and sends it to the robot
45          % controller through the serial port s (and cable).
46          % The code 1111 if for notifying the controller,
47          % which is polling its serial port, about the
48          % transmission
49
50          fprintf(s,num2str(1111));
51
52          % The next statement converts the Code into a string
53          % and sends it to the robot controller through the
54          % serial port s (and cable)
55
56          fprintf(s,num2str(Code));
57
58          % The next statement converts the number n into a
59          % string and sends it to the robot controller through
60          % the serial port s (and cable)
61
62          fprintf(s,num2str(n));
63
64          i=1;      % iteration variable
65
66          % The next cycle sends the items of A to the robot
67          % controller through the serial port s (and cable)
68
69          while(i ≤ n)
70             fprintf(s,num2str(A(i,1)));
71             fprintf(s,num2str(A(i,2)));
72             fprintf(s,num2str(A(i,3)));
73             fprintf(s,num2str(A(i,4)));
74             i = i + 1;
75          end
76
77          % The next statement performs a polling operation
78          % of  the serial port until it receives a feedback
79          % message from the controller
80
81          x=fscanf(s)
82
83          % If the inner expression is TRUE, an error has occurred
84
85          if(str2num(x) == 7777)
86
87             state_keeper('store',4,0);
88             disp(' ')
89             disp('A problem occurred. There are two possible reasons:')
90             disp(' ')
91             disp('1)A robot ERROR has occurred. ')
92             disp('2)"Start" on the pedant has not been selected when you ...
                   typed prog() or the next' )
93             disp('statement in Matlab.')
94             disp(' ')
95             disp('SOLUTION:')
96             disp('Be sure that "Start" on the pedant has been selected, then ...
                   type the prog()')
97             disp('statement to continue with programming, otherwise the next ...
                   statements ')
98             disp('will be useless.')
```

```
99              disp('In the case 1) obviously fix your code!!!')
100             disp(' ')
101
102             % The next statement disconnects serial
103             % port object from device
104
105             fclose(s);
106
107             % The next statement removes the
108             % serial port object from memory
109
110             delete(s);
111             state_keeper('store',1,1);
112             return;
113
114          end
115
116          % If the sample mode has been selected (see section 5.10),
117          % the function sstorage is called in order
118          % to store the position data coming from the controller
119
120          if(state_keeper('retrieve',8) == 1 |
121             state_keeper('retrieve',8) == 2)
122             sstorage(str2num(x));
123          end
124
125       end
126
127    end
128
129 end
```

Listing A.2: Function `position`

```
1  function [ o ] = position( a )
2  % This function receives as an input parameter the centroid
3  % of an object in an image, and it outputs the position of
4  % the centroid of the same object in the workspace.
5
6
7  a = a / 96 * 10; % Pixel to mm conversion
8
9  % X coordinate
10
11 px = 5.5141 * a(1,1) + 0.2346 * a(1,2) - 270.5216;
12
13 % Y coordinate
14
15 py= 0.1974 * a(1,1) - 5.4673 * a(1,2) + 463.5217;
16
17 o=[px,py];
18
19 end
```

Listing A.3: Function `keys_detection`.

```
1  function [o] = keys_detection( )
2  % This main function detects keys in an image
```

```
% Output: - '-1' if no keys are detected;
%         - Nx4 matrix if N keys are detected, the structure of the
%           ith row is: ['Orientation', key bow pointing,
%           'Centroid' horizontal (x) coord, 'Centroid'
%           vertical (y) coord]



vid = cam('retrieve');   % Retrieves the video input
                         % object

% The 'opt_threshold_detection' local
% function is called, in order to detect the optimum BW threshold

opt_thr = opt_threshold_detection(vid)

M1 = getsnapshot(vid);   % Takes a picture and stores it in
                         % a matrix variable

M1 = im2bw(M1,opt_thr);        % Converts the coulour image
                               % into  a black and white
                               % image by using 'opt_thr'
                               % as threshold

M1 = image_processing(M1)   % The 'image_processing' local function
                            % is called. It returns a processed image

% The next statements returns a matrix L, of the same size as M1,
% containing labels for the connected objects. n is the number of
% connected objects.

[L n] = bwlabel(M1);

% The next statement measures a set of properties for each
% labeled region in the label matrix L and stores them
% in the matrix 'stats'

stats = regionprops(L,'Area', 'Centroid', 'BoundingBox', 'Orientation',
            'EquivDiameter', 'Perimeter');

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


% In the next cycle, if an object matches key shape, it is stored in
% 'OBJ&PROP' with its orientation and coordinates of its centroid

OBJ&PROP=[0,0,0,0,0];

while(i ≤ n)

   % The next if statement detects if an onject matches the key shape

   if(stats(i).Area > 2600 & stats(i).Area < 3100 &
      stats(i).Perimeter > 490 &  stats(i).Perimeter < 670
       & stats(i).MajorAxisLength > 106  &  stats(i).MajorAxisLength < 115
          & stats(i).MajorAxisLength > 40  &  stats(i).MajorAxisLength < 43)


       OBJ&PROP(j,1)=i;
       OBJ&PROP(j,2)=stats(i).Orientation;  % Stores in
```

```
64                                          % 'OBJ&PROP(j,2)'
65                                          % the orientation
66                                          % of the key (deg)
67
68
69          % The next four if statements detect if the key bow is
70          % pointing towards the top of the image (in this case
71          % '1' is stored in 'OBJ&PROP(j,3)', or towards the bottom of
72          % the image (in this case '2' is stored in 'OBJ&PROP(j,3)'
73
74          if(stats(i).Orientation ≥ 0 & stats(i).Orientation < 45 )
75             if(stats(i).Centroid(1) < stats(i).BoundingBox(1) + ...
                   stats(i).BoundingBox(3)/2)
76                 OBJ&PROP(j,3) = 1;
77             else
78                 OBJ&PROP(j,3) = 2;
79             end
80          end
81
82          if(stats(i).Orientation ≥ 45 & stats(i).Orientation < 90 )
83             if(stats(i).Centroid(2) > stats(i).BoundingBox(2) + ...
                   stats(i).BoundingBox(4)/2)
84                 OBJ&PROP(j,3) = 1;
85             else
86                 OBJ&PROP(j,3) = 2;
87             end
88          end
89
90          if(stats(i).Orientation < 0 & stats(i).Orientation >- 45 )
91             if(stats(i).Centroid(1) < stats(i).BoundingBox(1) + ...
                   stats(i).BoundingBox(3)/2)
92                 OBJ&PROP(j,3) = 1;
93             else
94                 OBJ&PROP(j,3) = 2;
95             end
96          end
97
98          if(stats(i).Orientation ≤ -45 & stats(i).Orientation ≥ - 90 )
99
100            if(stats(i).Centroid(2) < stats(i).BoundingBox(2) + ...
                   stats(i).BoundingBox(4)/2)
101               OBJ&PROP(j,3) = 1;
102            else
103               OBJ&PROP(j,3) = 2;
104            end
105         end
106
107
108
109         % The next part of code is for detecting the centroid of each
110         % key and storing its image coordinates in 'OBJ&PROP(j,4)'
111         % (x coordinates) and 'OBJ&PROP(j,5)' (y coordinates)
112
113         if(stats(i).Orientation ≤ 0)
114
115           if(OBJ&PROP(j,3) == 1)
116
117              OBJ&PROP(j,4) = round(stats(i).Centroid(2)+
118                                  8*sind(stats(i).Orientation));
119              OBJ&PROP(j,5) = round(stats(i).Centroid(1)-
120                                  8*cosd(stats(i).Orientation));
```

```
121             M1(round(stats(i).Centroid(2)+
122                                 8*sind(stats(i).Orientation)),
123             round(stats(i).Centroid(1)-
124                                 8*cosd(stats(i).Orientation))) = 0;
125           else
126             OBJ&PROP(j,4)=round(stats(i).Centroid(2)+
127                                 8*sind(-stats(i).Orientation));
128             OBJ&PROP(j,5)=round(stats(i).Centroid(1)+
129                                 8*cosd(-stats(i).Orientation));
130             M1(round(stats(i).Centroid(2)+
131                                 8*sind(-stats(i).Orientation));
132             round(stats(i).Centroid(1)+
133                                 8*cosd(-stats(i).Orientation))) = 0;
134           end
135
136         end
137
138         if(stats(i).Orientation > 0)
139
140            if(OBJ&PROP(j,3) == 1)
141               OBJ&PROP(j,4) = round(stats(i).Centroid(2)+
142                                 8*sind(stats(i).Orientation));
143               OBJ&PROP(j,5) = round(stats(i).Centroid(1)-
144                                 8*cosd(stats(i).Orientation));
145               M1(round(stats(i).Centroid(2)+
146                                 8*sind(stats(i).Orientation)),
147               round(stats(i).Centroid(1)-8*cosd(stats(i).Orientation))) = 0;
148            else
149               OBJ&PROP(j,4) = round(stats(i).Centroid(2)-
150                                 8*sind(stats(i).Orientation));
151               OBJ&PROP(j,5) = round(stats(i).Centroid(1)+
152                                 8*cosd(stats(i).Orientation));
153               M1(round(stats(i).Centroid(2)-
154                                 8*sind(stats(i).Orientation)),
155               round(stats(i).Centroid(1)+
156                                 8*cosd(stats(i).Orientation))) = 0;
157            end
158
159         end
160
161         j=j+1;
162
163     end
164
165 i=i+1;
166
167 end
168
169 % If no keys have been detected, '-1' is outputted, otherwise OBJ&PROP
170 % is outputted
171
172 if(OBJ&PROP(1,1)==0 & OBJ&PROP(1,2)==0 & OBJ&PROP(1,3)==0
173      & OBJ&PROP(1,4)==0 & OBJ&PROP(1,5)==0)
174
175     o=[-1];
176 else
177     o=[OBJ&PROP(:,2), OBJ&PROP(:,3), OBJ&PROP(:,4), OBJ&PROP(:,5)];
178 end
179
180 end
181
```

```
182
183 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
184 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
185
186
187
188 function [o] = opt_threshold_detection(vid)
189
190 % This local function detects the optimum BW threshold.
191 % Input: video object
192 % Output: optimum BW threshold
193
194 opt_thr = 0.8;    % Optimum BW threshold
195
196 opt_thr_#_obj = 0;      % Number of objects that match key shape
197                         % by using opt_thr
198
199 current_#_obj = 0;      % Number of objects that match key shape
200                         % by using a lower threshold than 'opt_thr'
201
202 l=1;    % Iteration variable
203
204
205
206 while(l ≤ 6)
207
208    M1 = getsnapshot(vid);   % Takes a picture and store it in
209                             % a matrix variable
210
211    M1 = im2bw(M1,0.8 - 0.1*l);        % Converts the coulored image
212                                       % into  a black and white
213                                       % image by using '0.8-0.1*l'
214                                       % as threshold
215
216
217    M1 = image_processing(M1)    % The 'image_processing' local
218                                 % function is called. It returns a
219                                 % processed image
220
221
222    % The next statements returns a matrix L, of the same size as M1,
223    % containing labels for the connected objects.
224
225    [L n] = bwlabel(image_processing(M1));
226
227
228    % The next statement measures a set of properties for each
229    % labeled region in the label matrix L and stores them
230    % in the matrix 'stats'
231
232    stats = regionprops(L,'Area', 'Centroid', 'BoundingBox',
233                    'Orientation', 'EquivDiameter', 'Perimeter');
234
235    i = 1;   % iteration variable
236
237    while(i ≤ n)
238
239       % The next 'if' statemet detects which object matches
240       % key shape
241
242       if(stats(i).Area > 2600 & stats(i).Area < 3100 &
```

```
243           stats(i).Perimeter > 490 &  stats(i).Perimeter < 670
244              & stats(i).MajorAxisLength > 106  &  stats(i).MajorAxisLength < 115
245                 & stats(i).MajorAxisLength > 40  &  stats(i).MajorAxisLength < 43)
246
247
248
249           current_#_obj = current_#_obj + 1;
250
251        end
252
253        i = i + 1;
254
255     end
256
257     % If the number of objects that matches key shape by using 'opt_thr-0.1'
258     % as BW threshold is greater than with 'opt_thr', 'opt_thr' becomes
259     % 'opt_thr-0.1'
260
261     if(current_#_obj > opt_thr_#_obj)
262
263        opt_thr_#_obj = current_#_obj;
264
265
266        opt_thr = opt_thr - 0.1;
267
268     end
269
270     current_#_obj = 0;
271
272 end
273
274 o = opt_thr;
275
276 end
277
278
279 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
280 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
281
282 function [o] = image_processing(M1)
283
284 % This local function processes an input image in order
285 % to reduce noise, clear white border and make clearer
286 % the shapes of possible keys.
287 % Input: image
288 % Output: image
289
290
291 % The image is dilated using linear structuring elements,
292 % that can be created with the strel function
293
294 se90 = strel('line', 3, 90);
295 se0 = strel('line', 3, 0);
296
297 M1 = imdilate(M1, [se90 se0]);    % Dilates the image
298
299 M1 = imclearborder(M1, 4);    % Suppresses light structures
300                               % connected to image border
301
302 M1= bwareaopen(M1,20);      % Removes small objects from
303                             % binary image
```

```
304
305 M1 = imfill(M1,'holes');    % Fills image regions and holes
306
307 M1 = bwareaopen(M1,800);    % Removes small objects from
308                              % binary image
309
310 imwrite(M1, 'M1.tif');   % Writes the image to Tagged Image
311                          % File Format
312
313 BW = imread('M1.tif');   % Reads the image from a Tagged Image
314                          % File Format
315 o = M1
316
317 end
```

Listing A.4: Script keys_pick_and_place.

```
1  % Starting from a random placement of nine keys, the robot has to
2  % pick them up and place them down one by one, with the major
3  % axes aligned in the same manner. The robot has to perform
4  % this operation autonomously by using a digital camera and a
5  % vacuum gripping system.
6
7  right();   % Selects the right arm mode
8
9  speed(50);   % Sets the speed [50% of the maximum speed]
10
11 start_cam();  % Opens the video input device and configures
12               % image acquisition
13
14 A = [-340.9334  135.6999];    % First keys placement position
15
16 j = 0;     % Keys placement position iteration variable
17
18 % If after four vibration actions, and image acquisition
19 % and processing, no keys have been detected, the software
20 % detects that the operation has been completed and stops the
21 % cycle using a break command.
22
23
24 while(1 ≠ 0)
25
26
27    % The next statement defines the image acquisition position
28
29    image_acq_pos = [125.0769, 184.8158, 6.0472, 119.4322];
30
31    moved(image_acq_pos);  % Moves to image_acq_pos
32
33    l = 0;  % Iteration variable
34    m = 1;  % Variables for storing the number of rows of matrix M
35
36    % The next cycle stops after four iterations with no detected
37    % keys or when at least one key is detected
38
39    while(m == 1 & l < 4)
40
41       out(938,1);   % Switches on the DC motor in order to vibrate
42                     % the surface where the keys are randomly
43                     % placed
```

```
44
45        pause(0.8);    % Pause of 0.8 sec during which the DC motor
46                       % is working
47
48        out(938,0);    % Switches on the DC motor in order to block
49                       % vibration
50
51        M = keys_detection();  % The function keys_detection is
52                               % called in order to acquiring  an
53                               % image and detect possible keys. The
54                               % x-y coordinates and orientation of
55                               % each key are stored in M.
56
57        [n,m] = size(M);       % Returns size of M dimensions
58
59        l = l + 1;     % Iteration variable updating
60    end
61
62    % If l == 4, that means four iterations have been executed with
63    % no keys detected, the cycle is interrupted by using a break
64    % command
65
66    if(l == 4)
67
68        break;
69
70    end
71
72
73    i = 1;  % Iteration variable
74
75    % The next cycle is for picking up and placing down
76    % keys
77
78    while(i ≤ n)
79
80    % The next statement retrieves row i of matrix M:
81    % ['Orientation', key bow pointing information, Image
82    % 'Centroid' horizontal (x) coord, Image 'Centroid'
83    % vertical (y) coord]
84
85    p = M(i,:);
86
87    % The next statement is a call to cent_position function for
88    % detecting the centroid in the workspace. The input parameter
89    % are the 'Centroid' image coordinates
90
91    pos = cent_position([p(1,4),p(1,3)]);
92
93    moved(pos(1,1),pos(1,2),20,0);   % Moves above the new key
94                                     % centroid position.
95
96    out(937,1);   % Switches on the vacuum gripping system
97
98    smove(3,87.7);   % Lowers the Z-axis in order to grab
99                     % the key
100
101   srmove(3,-30);   % Lifts the Z-axis (the key is attached to
102                    % the (vacuum cup)
103
104   % The next block of code calculates the rotation angle needed
```

```
105     % to align the key major axis along the specified direction.
106     % Remind: - M(1,1) is the 'Orientation'
107     %         - M(1,2) is 1 if the key bow points to the top of the
108     %         - image, is 2 if it points to the bottom of the image
109
110     if(M(i,1) ≥ 0)
111
112         if(M(i,2) == 1)
113
114            s = M(i,1);
115
116         else
117
118            s = M(i,1) + 180;
119
120         end
121
122     else
123         if(M(i,2) == 1)
124
125             s = M(i,1);
126
127         else
128            s = M(i,1) + 180;
129
130         end
131
132     end
133
134
135     if(j == 3)   % Updates the Y coordinate of the key placement
136                  % position
137
138         A(1,2) = A(1,2) + 30;
139
140         j = 0;
141     end
142
143
144     moved(A(1,1) - j*70,A(1,2),20,s);  % Moves above the new
145                                        % placement position after
146                                        % updating X coordinate
147                                        % of the key position
148
149     smove(3,67.8137);   % Lowers the Z-axis
150
151     out(937,0);   % Switches off the vcuum gripping system in order
152                   % to release the key
153
154     pause(0.2);   % Pause of 0.2 sec in order to allow key
155                   % release
156
157     srmove(3,-20);   % Lifts Z-axis
158
159     i = i + 1;  % Iteration variable updating
160
161     j = j + 1;  % Updating of keys placement position iteration
162                 % variable
163
164 end
165
```

```
166
167 end
168
169 close_cam();   % Closes the input video object
```

Listing A.5: Function `discs&colours_detection`.

```
1  function [ o ] = discs&colours_detection( )
2  % This main function detects discs in an image
3  % Output: - '-1' if no discs are detected;
4  %         - Nx3 matrix if N discs are detected, the structure of the
5  %           ith row is: ['Centroid' horizontal (x) coord, 'Centroid'
6  %           vertical (y) coord, disc colour]
7
8
9
10
11 vid=cam( 'retrieve' );
12
13 opt_thr = opt_threshold_detection(vid)
14
15 trigger(vid);   % Acquires an image
16
17 M = getdata(vid);   % Stores the image in the matrix M
18
19
20 % The next three statements split the colour image M to its 3 RGB
21 % (red, green, blue) channels
22
23 M1 = M(:,:,1);
24 M2 = M(:,:,2);
25 M3 = M(:,:,3);
26
27 % The next three statements perform a colour to BW conversion
28 % by using by using opt_thr as threshold
29
30 M1=im2bw(M1,opt_thr);
31 M2=im2bw(M2,opt_thr);
32 M3=im2bw(M3,opt_thr);
33
34 % The next statement performs a OR operation between three images
35
36 M=M1 | M2 | M3;
37
38 M = image_processing(M)     % Call to 'image_processing' local
39                             % function that returns a processed
40                             % image
41
42
43 % The next statements returns a matrix L, of the same size as M,
44 % containing labels for the connected objects.
45
46 [L n] = bwlabel(M);
47
48 % The next statement measures a set of properties for each
49 % labeled region in the label matrix L and stores them
50 % in the matrix 'stats'
51
52 stats = regionprops(L,'Area','Centroid','Perimeter','MajorAxisLength',
53             'MinorAxisLength');
```

```
54
55
56 i = 1;   % Iteration variable
57 j = 1;   % Position pointer in 'OBJ&PROP'
58
59
60 % In the next cycle, if an object matches disc shape, it is stored in
61 % 'OBJ&PROP' with its orientation and coordinates of its centroid
62
63 OBJ&PROP=[0,0,0,0];
64
65 while(i ≤ n)
66
67    if(stats(i).Area > 1950 & stats(i).Area < 2200 &
68       stats(i).Perimeter > 430 &  stats(i).Perimeter < 470
69        & stats(i).MajorAxisLength > 51  &  stats(i).MajorAxisLength < 56
70           & stats(i).MajorAxisLength > 49  &  stats(i).MajorAxisLength < 54)
71
72       OBJ&PROP(j,1)=i;
73       OBJ&PROP(j,2)=stats(i).Centroid(1);
74       OBJ&PROP(j,3)=stats(i).Centroid(2);
75
76       % The local function 'colour_detection' is called
77       % in order perform a double colour detection.
78       % Only if the two colour detections outputs a colour (1 is for
79       % white, 2 is for red, 3 is for blue, and 4 is for green) and the
80       % colour is the same, the colour information is stored in
81       % OBJ&PROP(j,4) and the position pointer 'j' is updated.
82       % Otherwise, if the functions output two different colours
83       % or 5 (that means a problem with the colour detection has
84       % occured): the position pointer 'j' is not updated i.e.
85       % the disc is not detected
86
87       first_clour_detection = colour_detection(12);
88       second_clour_detection = colour_detection(-12);
89
90       if(first_clour_detection == second_clour_detection &&
91           first_clour_detection ≠ 5)
92
93          OBJ&PROP(j,4) = first_clour_detection;
94
95          j=j+1;
96
97       end
98
99   end
100
101   i=i+1;
102
103 end
104
105
106 if(OBJ&PROP(1,1)==0 & OBJ&PROP(1,2)==0 & OBJ&PROP(1,3)==0 OBJ&PROP(1,4)==0)
107
108    o=[-1];
109
110 else
111
112    o=[OBJ&PROP(:,2), OBJ&PROP(:,3), OBJ&PROP(:,4)];
113
114 end
```

```
115
116
117
118
119 end
120
121
122 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
123 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
124
125
126 function [o] = opt_threshold_detection(vid)
127 % This local function detects the optimum BW threshold.
128 % Input: video object
129 % Output: optimum BW threshold
130
131
132
133 opt_thr = 0.8;    % Optimum BW threshold
134
135 opt_thr_#_obj = 0;      % Number of objects that match disc shape
136                         % by using optimum opt_thr
137
138 current_#_obj = 0;      % Number of objects that match disc shape
139                         % by using a lower threshold than 'opt_thr'
140
141 l=1;    % Iteration variable
142
143
144
145
146 while(l ≤ 6)
147
148    trigger(vid);   % Acquires an image
149
150    M = getdata(vid);   % Stores the image in the matrix M
151
152
153    % The next three statements split the colour image M to its 3 RGB
154    % (red, green, blue) channels
155
156    M1 = M(:,:,1);
157    M2 = M(:,:,2);
158    M3 = M(:,:,3);
159
160    % The next three statements perform a colour to BW conversion
161    % by using by using '0.8-0.1*l' as threshold
162
163    M1=im2bw(M1,0.8 - 0.1*l);
164    M2=im2bw(M2,0.8 - 0.1*l);
165    M3=im2bw(M3,0.8 - 0.1*l);
166
167    % The next statement performs a OR operation between three images
168
169    M=M1 | M2 | M3;
170
171    M = image_processing(M)    % Call to 'image_processing' local
172                               % function that returns a processed
173                               % image
174
175
```

```
176     % The next statements returns a matrix L, of the same size as M,
177     % containing labels for the connected objects.
178
179     [L n] = bwlabel(image_processing(M));
180
181
182     % The next statement measures a set of properties for each
183     % labeled region in the label matrix L and stores them
184     % in the matrix 'stats'
185
186     stats = regionprops(L,'Area','Centroid','Perimeter','EquivDiameter');
187
188
189     i = 1;   % Iteration variable
190
191     while(i≤n)
192
193        % The next 'if' statemet detects which object matches
194        % disc shape
195
196        if(stats(i).Area > 1950 & stats(i).Area < 2200 &
197           stats(i).Perimeter > 430 &  stats(i).Perimeter < 470
198              & stats(i).MajorAxisLength > 51  &  stats(i).MajorAxisLength < 56
199                 & stats(i).MajorAxisLength > 49  &  stats(i).MajorAxisLength < 54)
200
201        current_#_obj = current_#_obj + 1;
202
203        end
204
205        i=i+1;
206
207     end
208
209     % If number of objects that matches disc shape by using 'opt_thr-0.1'
210     % as BW threshold is greater than with 'opt_thr', 'opt_thr' becomes
211     % 'opt_thr-0.1'
212
213     if(current_#_obj > opt_thr_#_obj)
214
215        opt_thr_#_obj = current_#_obj;
216
217        opt_thr = opt_thr - 0.1;
218
219     end
220
221     current_#_obj = 0;
222
223  end
224
225  o = opt_thr;
226
227  end
228
229
230
231  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
232  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
233
234
235  function [o] = image_processing(M)
236  % This local function processes an input image in order
```

```
% to reduce noise, clear white border and make clearer
% the shapes of possible discs.
% Input: image
% Output: image


% The image is dilated using linear structuring elements,
% that can be created with the strel function

se90 = strel('line', 3, 90);
se0 = strel('line', 3, 0);

M1 = imdilate(M1, [se90 se0]);    % Dilates the image

M1 = imclearborder(M, 4);     % Suppresses light structures
                               % connected to image border

M1= bwareaopen(M1,20);       % Removes small objects from
                              % binary image

M1 = imfill(M,'holes');     % Fills image regions and holes

M1 = bwareaopen(M,800);     % Removes small objects from
                            % binary image

imwrite(M1, 'M1.tif');    % Writes the image to Tagged Image
                          % File Format

BW = imread('M1.tif');    % Reads the image from a Tagged Image
                          % File Format
o = M1

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [o] = colours_detection(t)
% This local function, by checking the colour value of the same
% pixel for  each channel, detects the colour (red, green,
% blue, or white) of the pixel (and thus of the disc) in  the
% true colour image. The checked pixel is not the centroid but
% another pixel of the disc
% This is necessary due to the fact that a gold number is printed
% on the centre of one side of the circle.
% Input: translation (pixels)
% Output: 1 white, 2 red, 3 blue, 4 green, 5 no detected colour


if(M1(round(stats(i).Centroid(2)) - t,
         round(stats(i).Centroid(1)) - t) > 180
            & M2(round(stats(i).Centroid(2)) -t,
               round(stats(i).Centroid(1)) - t) > 200
                  & M3(round(stats(i).Centroid(2)) - t,
                     round(stats(i).Centroid(1)) - t) > 200)

   o = 1;

else
```

```
298
299     if(M1(round(stats(i).Centroid(2)) - t,
300        round(stats(i).Centroid(1)) - t) > 200
301           & M2(round(stats(i).Centroid(2)) - t,
302               round(stats(i).Centroid(1)) - t) < 150
303                  & M3(round(stats(i).Centroid(2)) - t,
304                      round(stats(i).Centroid(1)) - t) < 150)
305
306        o = 2;
307
308     else
309
310        if(M1(round(stats(i).Centroid(2)) - t,
311           round(stats(i).Centroid(1)) - t) < 120
312              & M2(round(stats(i).Centroid(2)) - t,
313                 round(stats(i).Centroid(1)) - t) < 120
314                    & M3(round(stats(i).Centroid(2)) - t,
315                       round(stats(i).Centroid(1)) - t) > 150)
316
317           o = 3;
318
319         else
320
321              if(M1(round(stats(i).Centroid(2)) - t,
322                 round(stats(i).Centroid(1)) - t) < 130
323                   & M2(round(stats(i).Centroid(2)) - t,
324                      round(stats(i).Centroid(1)) - t) > 150
325                         & M3(round(stats(i).Centroid(2)) - t,
326                            round(stats(i).Centroid(1)) - t) < 130)
327
328                 o=4;
329
330              else
331
332                 o=5;
333
334              end
335
336           end
337
338        end
339
340  end
341
342  end
```

Listing A.6: Script `coloured_discs_pick_and_place`.

```
1  % Starting from a random placement of about thirty coloured
2  % discs (the exact number is not important), the robot has
3  % to pick them up and place them down one by one, in 4 small
4  % containers depending on the colour. The colours are: white,
5  % red, green, blue.
6
7  right();   % Selects the right arm mode
8
9  speed(50);   % Sets the speed [50% of the maximum speed]
10
11 start_cam();  % Opens the video input device and configures
12               % image acquisition
```

```
13
14
15
16 while(1≠0)
17
18    l = 0;  % Iteration variable
19    m = 1;  % Variables for storing the number of rows of
20            % matrix M
21
22    image_acq_pos = [125.0769, 184.8158, 6.0472, 119.4322];
23
24    moved(image_acq_pos);  % Moves to image_acq_pos
25
26
27    while(m == 1 & l < 4)
28
29       out(938,1);   % Switches on the DC motor in order to vibrate
30                     % the surface where the discs are randomly
31                     % placed
32
33       pause(0.8);   % Pause of 0.8 sec during which the DC motor
34                     % is working
35
36       out(938,0);   % Switches on the DC motor in order to block
37                     % vibration
38
39
40       % The function discs_detection is called. It
41       % acquires an image and detects possible discs.
42       % The x-y orientation  of each disc are stored in M.
43
44       M = discs&colours_detection();
45
46       [n,m] = size(M);       % Returns size of M dimensions
47
48       l = l + 1;     % Iteration variable updating
49    end
50
51    % If l == 4, that means four iterations have been executed
52    % with no detected discs, the cycle is interrupted by using
53    % a break command
54
55    if(l == 4)
56
57       break;
58
59    end
60
61
62    i=1;  % Iteration variable
63
64    while(i ≤ n)
65
66       % The next statement retrieves row i of matrix M:
67       % 'Centroid' horizontal (x) coord, 'Centroid'
68       % vertical (y) coord, colour]
69
70       p=M(i,:);
71
72       % The function cent_position is called in order to
73       % detect the centroid in the workspace.
```

```
74        % The input parameter are the 'Centroid'
75        % coordinates
76
77        pos=position([p(1,2),p(1,3)]);
78
79        moved(pos(1,1),pos(1,2),20,0);  % Moves above the new
80                                        % disc centroid
81                                        % position.
82
83        out(937,1);  % Switches on the vacuum gripping system
84
85        smove(3,84.8);  % Lowers the Z-axis in order to grab
86                        % the disc
87
88        smove(3,30);  % Lifts the Z-axis (the disc is attached to
89                      % the (vacuum cup)
90
91        % The next 'switch' construction moves the robot over
92        % the placement position depending on the disc colour,
93        % and releases the disc
94
95
96        white_release_pos = [-374.92, 129.82,  32.00, 111.99];
97        red_release_pos = [-384.99, 214.99, 31.99, 112.00];
98        blue_release_pos = [-384.99, 214.99, 31.99, 111.99];
99        green_release_pos = [-399.99, 375.00, 31.99, 111.99];
100
101
102       switch(M(i,4))
103          case(1)
104
105             moved(white_release_pos);
106
107             out(937,0); % Switches off the vcuum gripping system
108                         % in order to release the disc
109
110             pause(0.2);  % Pause of 0.2 sec in order to allow
111                          % disc release
112
113          case(2)
114             moved(red_release_pos);
115             out(937,0);
116             pause(0.2);
117          case(3)
118             moved(blue_release_pos);
119             out(937,0);
120             pause(0.2);
121          case(4)
122             moved(green_release_pos);
123             out(937,0);
124             pause(0.2);
125          end
126       end
127
128    i=i+1;
129
130 end
131
132
133 end
134
```

```
close_cam();
```

Listing A.7: Interpreter (SSL/E program).

```
// Variables declaration and initialization

POSITION POS[8];               // Position array of 8 elements
POSITION P;                    // Position variable
POSITION P1;                   // Position variable

REAL X;     // Variable for storing a coordinate read from the serial port
REAL Y;     // Variable for storing a coordinate read from the serial port
REAL Z;     // Variable for storing a coordinate read from the serial port
REAL S;     // Variable for storing a coordinate read from the serial port

REAL ARRAY_1[13];         // Real array (13 elements)
INT ARRAY_2[5];           // Real array (5 elements)


INT MAT_FUN_CODE=0;  // Variable for storing Matlab Robot Function
                     // Identification Code

REAL PAR;
INT COMIN;
STRING S1;
INT I=0;          // Iteration variable
INT M=0;          // It's for avoiding PROG SAMPLE execution before
                  // it is called in PROG


INTERPRETER()

INT VIS=0;      // If this variable is 0 the trajectory sampling mode
                // is not active, if it is 1 the trajectory  sampling
                // mode is active.
REAL VAR;
INT K=7777;
REAL RET=7777;
INT MAT_FUN_CODE_CODE=0;



////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////

/* Routine for sampling position during the motion */

PROG SAMPLE()

IF(M!=0){
   I=1;
   WHILE(I!=0){
      IF(VIS==1){
         TIMEST(0);
         MARK(P1);
         RSOUT(1,POSGET(P1,1));
         RSOUT(1,POSGET(P1,2));
         RSOUT(1,POSGET(P1,3));
         RSOUT(1,POSGET(P1,4));
         I=timerd(1);
```

```
57        }
58
59        ELSE{
60           JMARK(P1);
61           RSOUT(1,POSGET(P1,1));
62           RSOUT(1,POSGET(P1,2));
63           RSOUT(1,POSGET(P1,3));
64           RSOUT(1,POSGET(P1,4));
65           I=STATM(1);
66        }
67     }
68  }
69
70  END
71
72
73  // Interpreter
74
75  PROG INTERPRETER()
76
77  M=1;              // For enabling PROG SAMPLE()
78
79
80  RSCLOSE(1);       /* Closes the serial port in order to delate any previous
81                    data stored in the port buffer avoiding the
82                    port buffer saturation */
83
84
85  /* The next statement  opens the serial port COM1 setting 115200 bps
86  as baud rate and 512 bytes as BUFFER length. Thus, the settings
87  of the port are "9600 B8 PE S2 L128 CRLF"
88
89  RSOPEN(1,"115200 L512");   /*
90
91
92  /* The next statement sends the Feedback Execution Error Code 7777 through
93  the serial cable. Since Matlab is waiting for a feedback from the interpreter,
94  this is a necessary feedback  code sent when the interpreter is restarted
95  after  an error occurrance. It alerts Matlab about the occurred error */
96
97  RSOUT(1,7777);
98
99  /* The software  starts a polling operation of the serial port,
100 waiting for the Matlab Communication Intialization Code 0000
101 (Matlab sends 0000 in order to establish the communication) */
102
103 RSIN(1,K);
104
105 WHILE(K!=0000){
106    RSIN(1,K);
107 }
108
109
110
111 ////////////////////////////////////////////////////////////////
112
113
114 START:
115
116 /* The next block of code is for sending a Feedback Execution Confirmation Code.
117 Depending on the function, the Code can be: 1, 0, 8888, or four numbers*/
```

```
118
119
120 IF((1000≤MAT_FUN_CODE) && (MAT_FUN_CODE≤1020) && VIS==0)
121    RSOUT(1,1);
122
123 IF((2000≤MAT_FUN_CODE) && (MAT_FUN_CODE≤3020)){
124
125    IF(MAT_FUN_CODE==2003 || MAT_FUN_CODE==2004 || MAT_FUN_CODE==3014){
126       RSOUT(1,POSGET(P,1));
127       RSOUT(1,POSGET(P,2));
128       RSOUT(1,POSGET(P,3));
129       RSOUT(1,POSGET(P,4));
130    }
131
132 IF(MAT_FUN_CODE==2018 || MAT_FUN_CODE==2019){
133
134    IF(RET==1){
135       RSOUT(1,POSGET(P,1));
136       RSOUT(1,POSGET(P,2));
137       RSOUT(1,POSGET(P,3));
138       RSOUT(1,POSGET(P,4));
139    }
140
141    ELSE
142
143       RSOUT(1,8888);
144
145 }
146
147 IF(MAT_FUN_CODE!=2003 && MAT_FUN_CODE!=2004 && MAT_FUN_CODE!=2018
148               && MAT_FUN_CODE!=2019 && MAT_FUN_CODE!=3014)
149
150    RSOUT(1,1);
151 }
152
153 IF(MAT_FUN_CODE==4000 || MAT_FUN_CODE==4003 || MAT_FUN_CODE==4004)
154    RSOUT(1,1);
155
156 IF(MAT_FUN_CODE==4001 || MAT_FUN_CODE==4002 && VIS==0)
157    RSOUT(1,1);
158
159 /* The next block of code is for polling the serial port waiting the
160 New Matlab Function Communication Code '1111' *\
161
162 RSIN(1,COMIN);
163
164 WHILE(COMIN!=1111)
165    RSIN(1,COMIN);
166
167
168 RSIN(1,MAT_FUN_CODE);      /* Reads the Matlab Function Code
169
170
171
172
173
174
175
176 /* Matlab Robot Functions with identification code between 1000 and 1007:
177
178 archmove, cmove, jmove, jmoved, lmove, lmoved, move, moved, rjmove,
```

```
179 rlmove, rmove, sjmove, slmove, smove, srjmove, srlmove, srmove,
180 xyrcir, xzrcir, yzrcir */
181
182
183
184 IF((1000≤MAT_FUN_CODE) && (MAT_FUN_CODE≤1020)){
185
186    RSIN(1,PAR); // reads the # of parameters of the statement from the port
187
188 I=1;
189 WHILE(I≤PAR){
190
191    RSIN(1,X);              // Reads a parameter from the port
192    RSIN(1,Y);
193    RSIN(1,Z);
194    RSIN(1,S);
195    POS[I]=X,Y,Z,S;         // Parameters storing operation
196    I=I+1;
197
198 }
199
200
201 /* Functions execution */
202
203
204 IF(MAT_FUN_CODE==1000){
205
206    IF(VIS==0){
207
208       IF(PAR==1){
209          MOVE(POS[1]);
210             CYCLE START;
211       }
212
213       IF(PAR==2){
214          MOVE(POS[1],POS[2]);
215             CYCLE START;
216       }
217
218       IF(PAR==3){
219
220          MOVE(POS[1],POS[2],POS[3]);
221          CYCLE START;
222       }
223
224       IF(PAR==4){
225          MOVE(POS[1],POS[2],POS[3],POS[4]);
226          CYCLE START;
227       }
228
229       IF(PAR==5){
230          MOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
231          CYCLE START;
232       }
233
234       IF(PAR==6){
235          MOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6]);
236          CYCLE START;
237       }
238
239       IF(PAR==7){
```

```
240          MOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6], POS[7]);
241          CYCLE START;
242       }
243
244       IF(PAR==8){
245          MOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6], POS[7],POS[8]);
246          CYCLE START;
247       }
248
249    }
250
251
252    ELSE {
253
254       IF(PAR==1){
255          QMOVE(POS[1]);
256          SAMPLE();
257          RSOUT(1,7777);
258          CYCLE START;
259       }
260
261       IF(PAR==2){
262          QMOVE(POS[1],POS[2]);
263          SAMPLE();
264          RSOUT(1,7777);
265          CYCLE START;
266       }
267
268       IF(PAR==3){
269          QMOVE(POS[1],POS[2],POS[3]);
270          SAMPLE();
271          RSOUT(1,7777);
272          CYCLE START;
273       }
274
275       IF(PAR==4){
276          QMOVE(POS[1],POS[2],POS[3],POS[4]);
277          SAMPLE();
278          RSOUT(1,7777);
279          CYCLE START;
280       }
281
282       IF(PAR==5){
283          QMOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
284          SAMPLE();
285          RSOUT(1,7777);
286          CYCLE START;
287       }
288
289       IF(PAR==6){
290          QMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6]);
291          SAMPLE();
292          RSOUT(1,7777);
293          CYCLE START;
294       }
295
296       IF(PAR==7){
297          QMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7]);
298          SAMPLE();
299          RSOUT(1,7777);
300          CYCLE START;
```

```
301         }
302
303         IF(PAR==8){
304            QMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7],POS[8]);
305            SAMPLE();
306            RSOUT(1,7777);
307            CYCLE START;
308         }
309
310      }
311
312   }
313
314
315
316
317
318   IF(MAT_FUN_CODE==1001){
319
320      IF(VIS==0){
321
322         IF(PAR==1){
323            LMOVE(POS[1]);
324            CYCLE START;
325         }
326
327         IF(PAR==2){
328            LMOVE(POS[1],POS[2]);
329            CYCLE START;
330         }
331
332         IF(PAR==3){
333            LMOVE(POS[1],POS[2],POS[3]);
334            CYCLE START;
335         }
336
337         IF(PAR==4){
338            LMOVE(POS[1],POS[2],POS[3],POS[4]);
339            CYCLE START;
340         }
341
342         IF(PAR==5){
343            LMOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
344            CYCLE START;
345         }
346
347         IF(PAR==6){
348            LMOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6]);
349            CYCLE START;
350         }
351
352         IF(PAR==7){
353            LMOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6], POS[7]);
354            CYCLE START;
355         }
356
357         IF(PAR==8){
358            LMOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6], POS[7],POS[8]);
359            CYCLE START;
360         }
361
```

```
362     }
363
364
365     ELSE {
366
367        IF(PAR==1){
368           QLMOVE(POS[1]);
369           SAMPLE();
370           RSOUT(1,7777);
371           CYCLE START;
372        }
373
374        IF(PAR==2){
375           QLMOVE(POS[1],POS[2]);
376           SAMPLE();
377           RSOUT(1,7777);
378           CYCLE START;
379        }
380
381        IF(PAR==3){
382           QLMOVE(POS[1],POS[2],POS[3]);
383           SAMPLE();
384           RSOUT(1,7777);
385           CYCLE START;
386        }
387
388        IF(PAR==4){
389           QLMOVE(POS[1],POS[2],POS[3],POS[4]);
390           SAMPLE();
391           RSOUT(1,7777);
392           CYCLE START;
393        }
394
395        IF(PAR==5){
396           QLMOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
397           SAMPLE();
398           RSOUT(1,7777);
399           CYCLE START;
400        }
401
402        IF(PAR==6){
403           QLMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6]);
404           SAMPLE();
405           RSOUT(1,7777);
406           CYCLE START;
407        }
408
409        IF(PAR==7){
410           QLMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7]);
411           SAMPLE();
412           RSOUT(1,7777);
413           CYCLE START;
414        }
415
416        IF(PAR==8){
417           QLMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7],POS[8]);
418           SAMPLE();
419           RSOUT(1,7777);
420           CYCLE START;
421        }
422
```

```
423     }
424
425 }
426
427
428
429 IF(MAT_FUN_CODE==1002){
430
431     IF(VIS==0){
432
433         IF(PAR==1){
434             JMOVE(POS[1]);
435             CYCLE START;
436         }
437
438         IF(PAR==2){
439             JMOVE(POS[1],POS[2]);
440             CYCLE START;
441         }
442
443         IF(PAR==3){
444             JMOVE(POS[1],POS[2],POS[3]);
445             CYCLE START;
446         }
447
448         IF(PAR==4){
449             JMOVE(POS[1],POS[2],POS[3],POS[4]);
450             CYCLE START;
451         }
452
453         IF(PAR==5){
454             JMOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
455             CYCLE START;
456         }
457
458         IF(PAR==6){
459             JMOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6]);
460             CYCLE START;
461         }
462
463         IF(PAR==7){
464             JMOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6], POS[7]);
465             CYCLE START;
466         }
467
468         IF(PAR==8){
469             JMOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6], POS[7],POS[8]);
470             CYCLE START;
471         }
472
473     }
474
475
476     ELSE {
477
478         IF(PAR==1){
479             QJMOVE(POS[1]);
480             SAMPLE();
481             RSOUT(1,7777);
482             CYCLE START;
483         }
```

```
484
485         IF(PAR==2){
486            QJMOVE(POS[1],POS[2]);
487            SAMPLE();
488            RSOUT(1,7777);
489            CYCLE START;
490         }
491
492         IF(PAR==3){
493            QJMOVE(POS[1],POS[2],POS[3]);
494            SAMPLE();
495            RSOUT(1,7777);
496            CYCLE START;
497         }
498
499         IF(PAR==4){
500            QJMOVE(POS[1],POS[2],POS[3],POS[4]);
501            SAMPLE();
502            RSOUT(1,7777);
503            CYCLE START;
504         }
505
506         IF(PAR==5){
507            QJMOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
508            SAMPLE();
509            RSOUT(1,7777);
510            CYCLE START;
511         }
512
513         IF(PAR==6){
514            QJMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6]);
515            SAMPLE();
516            RSOUT(1,7777);
517            CYCLE START;
518         }
519
520         IF(PAR==7){
521            QJMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7]);
522            SAMPLE();
523            RSOUT(1,7777);
524            CYCLE START;
525         }
526
527         IF(PAR==8){
528            QJMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7],POS[8]);
529            SAMPLE();
530            RSOUT(1,7777);
531            CYCLE START;
532         }
533
534      }
535
536   }
537
538
539
540   IF(MAT_FUN_CODE==1003){
541      VAR=POSGET(POS[3],1);
542      CIRCULAR(VAR);
543      IF(VIS==0){
544         MOVE(POS[1],POS[2]);
```

```
545         CIRCULAR(0);
546         CYCLE START;
547      }
548   }
549   ELSE{
550      QMOVE(POS[1],POS[2]);
551      SAMPLE();
552      CIRCULAR(0);
553      RSOUT(1,7777);
554      CYCLE START;
555   }
556
557
558   IF(MAT_FUN_CODE==1004){
559      VAR=POSGET(POS[4],1);
560      CIRCULAR(VAR);
561      IF(VIS==0){
562         MOVE(POS[1],POS[2],POS[3]);
563         CIRCULAR(0);
564         CYCLE START;
565      }
566   }
567   ELSE{
568      QMOVE(POS[1],POS[2],POS[3]);
569      SAMPLE();
570      CIRCULAR(0);
571      RSOUT(1,7777);
572      CYCLE START;
573   }
574
575
576   IF(MAT_FUN_CODE==1005){
577      VAR=POSGET(POS[3],1);
578      CIRCULAR(VAR,1);
579      IF(VIS==0){
580         MOVE(POS[1],POS[2]);
581         CIRCULAR(0);
582         CYCLE START;
583      }
584   }
585   ELSE{
586      QMOVE(POS[1],POS[2]);
587      SAMPLE();
588      CIRCULAR(0);
589      RSOUT(1,7777);
590      CYCLE START;
591   }
592
593
594   IF(MAT_FUN_CODE==1006){
595      VAR=POSGET(POS[4],1);
596      CIRCULAR(VAR,1);
597      IF(VIS==0){
598         MOVE(POS[1],POS[2],POS[3]);
599         CIRCULAR(0);
600         CYCLE START;
601      }
602   }
603   ELSE{
604      QMOVE(POS[1],POS[2],POS[3]);
605      SAMPLE();
```

```
606     CIRCULAR(0);
607     RSOUT(1,7777);
608     CYCLE START;
609  }
610
611
612
613  /* Matlab Robot Functions with identification code between 2000 and 2030:
614
615  acct, autoacl, cpacct, cpdacct, cpspeed, dacct, in, jmark,
616  left, mark, opeclr, opeout, right, sample,
617  speed, weight, wintime*/
618
619
620  IF((2000≤MAT_FUN_CODE) && (MAT_FUN_CODE≤2030)){
621
622  IF(MAT_FUN_CODE!=2022)     // reads the parameter from the port
623     RSIN(1,PAR);
624  ELSE
625     RSIN(1,S1);
626
627  /* MAT_FUN_CODEtions execution */
628
629  IF(MAT_FUN_CODE==2000){
630     RET=SPEED(PAR);
631     CYCLE START;
632  }
633
634
635  IF(MAT_FUN_CODE==2002){
636     RET=CPSPEED(PAR);
637     CYCLE START;
638  }
639
640
641  IF(MAT_FUN_CODE==2003){
642     MARK(P);
643     RSOUT(1,POSGET(P,1));
644     RSOUT(1,POSGET(P,2));
645     RSOUT(1,POSGET(P,3));
646     RSOUT(1,POSGET(P,4));
647     CYCLE START;
648  }
649
650
651  IF(MAT_FUN_CODE==2004){
652     JMARK(P);
653     RSOUT(1,POSGET(P,1));
654     RSOUT(1,POSGET(P,2));
655     RSOUT(1,POSGET(P,3));
656     RSOUT(1,POSGET(P,4));
657     CYCLE START;
658  }
659
660
661  IF(MAT_FUN_CODE==2005){
662     RET=LEFT();
663     CYCLE START;
664  }
665
666
```

```
IF(MAT_FUN_CODE==2006){
   RET=RIGHT();
   CYCLE START;
}

IF(MAT_FUN_CODE==2007){
   RET=ACCT(PAR);
   CYCLE START;
}

IF(MAT_FUN_CODE==2008){
   RET=DACCT(PAR);
   CYCLE START;
}

IF(MAT_FUN_CODE==2009){
   RET=AUTOACL(PAR);
   CYCLE START;
}


IF(MAT_FUN_CODE==2012){
   RET=CPACCT(PAR);
   CYCLE START;
}

IF(MAT_FUN_CODE==2013){
   RET=CPDACCT(PAR);
   CYCLE START;
}

IF(MAT_FUN_CODE==2015){
   RET=WEIGHT(PAR);
   CYCLE START;
}


/* Matlab Robot Functions with identification code between 3000 and 3014:

acct, autoacl, cpacct, cpdacct, cpspeed, dacct, delay, forder, in, jmark,
left, mark, opeclr, opeout, right, sample,
speed, weight, wintime */



IF((3000≤MAT_FUN_CODE) && (MAT_FUN_CODE≤3014)){

   IF(MAT_FUN_CODE!=3014){
      RSIN(1,PAR);
      I=1;
      WHILE(I≤PAR){

         RSIN(1,ARRAY_1[I]);
         I=I+1;

      }
}

ELSE {

   RSIN(1,PAR);
```

```
728     I=1;
729     WHILE(I≤PAR){
730
731        RSIN(1,ARRAY_2[I]);
732        I=I+1;
733
734     }
735  }
736
737
738
739  IF(MAT_FUN_CODE==3006){
740     RET=OUT(ARRAY_1[1],ARRAY_1[2]);
741     CYCLE START;
742  }
743
744
745  IF(MAT_FUN_CODE==3007){
746     WINTIME(ARRAY_1[1]);
747     RET=TRI(ARRAY_1[2],ARRAY_1[3]);
748     CYCLE START;
749  }
750
751  IF(MAT_FUN_CODE==3008){
752     WINTIME(ARRAY_1[1]);
753     RET=WIN(ARRAY_1[2],ARRAY_1[3]);
754     CYCLE START;
755  }
756
757  IF(MAT_FUN_CODE==3009){
758     IF(PAR==2){
759        RET=BLINK(ARRAY_1[1],ARRAY_1[2]);
760        CYCLE START;
761     }
762
763     IF(PAR==3){
764        RET=BLINK(ARRAY_1[1],ARRAY_1[2],ARRAY_1[3]);
765        CYCLE START;
766     }
767  }
768
769  IF(MAT_FUN_CODE==3010){
770     IF(PAR==1){
771        RET=BLINKEND(ARRAY_1[1]);
772        CYCLE START;
773     }
774     IF(PAR==2){
775        RET=BLINKEND(ARRAY_1[1],ARRAY_1[2]);
776        CYCLE START;
777     }
778     IF(PAR==3){
779        RET=BLINKEND(ARRAY_1[1],ARRAY_1[2],ARRAY_1[3]);
780        CYCLE START;
781     }
782     IF(PAR==4){
783        RET=BLINKEND(ARRAY_1[1],ARRAY_1[2],ARRAY_1[3],ARRAY_1[4]);
784        CYCLE START;
785     }
786
787  }
788
```

```
789
790 IF(MAT_FUN_CODE==3013){
791    RET=LOCATE(ARRAY_1[1],ARRAY_1[2]);
792    CYCLE START;
793 }
794
795 IF(MAT_FUN_CODE==3014){
796    PLTNS(ARRAY_2[1],ARRAY_2[2],P);
797    RSOUT(1,POSGET(P,1));
798    RSOUT(1,POSGET(P,2));
799    RSOUT(1,POSGET(P,3));
800    RSOUT(1,POSGET(P,4));
801    CYCLE START;
802 }
803
804
805
806
807
808 /* Matlab Robot Functions with identification code 4000: setplt */
809
810 IF(MAT_FUN_CODE==4000){
811
812    RSIN(1,PAR);
813    I=1;
814    WHILE(I≤4){
815
816       RSIN(1,X);
817       RSIN(1,Y);
818       RSIN(1,Z);
819       RSIN(1,S);
820       POS[I]=X,Y,Z,S;
821       I=I+1;
822    }
823
824    RSIN(1,ARRAY_2[1]);
825    RSIN(1,ARRAY_2[2]);
826    RSIN(1,ARRAY_2[3]);
827    RSOUT(1,1);
828
829 /* Function execution */
830
831 SETPLTNS(PAR,POS[1],POS[2],POS[3],POS[4],ARRAY_2[1],ARRAY_2[2],ARRAY_2[3]);
832    CYCLE START;
833 }
834
835
836
837 RSCLOSE(1);
838
839 END
```

Bibliography

[1] R. H. Bishop, *The Mechatronics Handbook*, CRC Press Inc, 2002.

[2] *Handbook of Industrial Robotics*, Second Edition. Edited by Shimon Y. Nof Copyright © 1999 John Wiley Sons, Inc.

[3] Hiroshi Makino, Akitaka Kato, and Yasunori Yamazaki. Research and Commercialization of SCARA Robot − The Case of Industry-University Joint Research and Development. *Int. J. of Automation Technology*, Vol.1 No.1, 2007.

[4] Jacques Denavit, Richard S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices, *Trans ASME J. Appl. Mech*, nº 23, 1955, pp. 215-221.

[5] Jacques Denavit, Richard S.Hartenberg. *Kinematic synthesis of linkages*, New York, McGraw-Hill, 1964.

[6] Richard M. Murray, Zexiang Li, S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*, CRC Press, 1994.

[7] H Kazerooni. Instrumented Harmonic Drives for Robotic Compliant Maneuvers. *Proceedings of the 1991 IEEE International Conference on Robotics and Automation* Sacramento, California, April 1991.

[8] MATLAB and Simulink for Technical Computing. The MathWorks Inc., USA. [Online]: http://www.mathworks.com/.

[9] P.I. Corke. MATLAB toolboxes: robotics and vision for students and teachers. *IEEE Robotics and Automation Magazine, Volume 14(4)*, 2007, pp. 16-17

[10] Francesco Chinello, Stefano Scheggi, Fabio Morbidi, Domenico Prattichizzo. KCT: a MATLAB toolbox for motion control of KUKA robot manipulators. *Robotics and Automation (ICRA), 2010 IEEE International Conference*, 2010, pp. 4603 - 4608

[11] W.E. Dixon, D. Moses, I.D. Walker, and D.M. Dawson. A Simulink- Based Robotic Toolkit for Simulation and Control of the PUMA 560 Robot Manipulator. In *Proc. IEEE/RSJ Int. Conf. Intel. Robots Syst*, pages 2202-2207, 2001.

[12] Sean G. McSweeney and William M. D. Wright, Software Interface and Vision System for a Scara Robot, *Proceedings of the 24th International Manufacturing Conference*, IMC 24, 2007.

[13] J. Norberto Pires. Using Matlab to interface Industrial Robotic Automation Equipment. *IEEE Robotics and Automation Magazines*, September 2000.

[14] Ford, W. What is an open architecture robot controller?, *Proceedings of the 1994 IEEE International Symposium on Intelligent Control, Piscataway*, NJ, USA: IEEE Press, Columbus, USA, pp. 27-32.

[15] Nikolaos P. Papanikolopoulos, Pradeep K. Khosla, Takeo Kanade. Visual Tracking of a Moving Target by a Camera Mounted on a Robot: A Combination of Control and Vision. *IEEE Transections on Robotics and Automation*, VOL. 9, NO. 1. Frebruary 1993.

[16] Ahmed Sh. Khusheef, Ganesh Kothapalli and Majid Tolouei-Rad. An Approach for Integration of Industrial Robot with Vision System and Simulation Software, *World Academy of Science, Engineering and Technology*, Vol: 5, 2011-10-24.

[17] Nello Zucch, *Machine Vision*, The fairmont Press Inc, 1987.

[18] P. Piccinini, A. Prati, R. Cucchiara, Real-time object detection and localization with sift-based clustering, in *Proc. of Image and Vision Computing*, 2012, pp. 1-15.

[19] Stormy Attaway. *MATLAB: a practical introduction to programming and problem solving* - 2nd ed., Elsevier Inc, 2012.

[20] Peter Corke. Robotics,Vision and Control, Fundamental Algorithms in MATLAB. Springer, 2011.

[21] Fu, K. S., Gonzales, R. C. and Lee, C. S. G. *Robotics Control, Sensing, Vision and Intelligence*, Industrial Enginering Series, McGraw-Hill, New York, 1987.

[22] R. Kavanagh. *Mechatronics and Robotics Course Notes*, UCC (Cork), 2013.

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

[24] *Instrument Control Toolbox User's Guide*, MathWorks, 2014, `http://www.mathworks.cn/help/pdf_doc/instrument/instrument.pdf`.

[25] *Image Processig Toolbox User's Guide*, MathWorks, 2014, `http://www.mathworks.com/help/pdf_doc/images/images_tb.pdf`.

[26] MathWorks Inc, *SimMechanics*, 2014. `http://www.mathworks.co.uk/products/simmechanics/`.

[27] *Image Acquisition Toolbox User's Guide*, MathWorks, 2014, `http://www.mathworks.com/help/pdf_doc/imaq/imaq_ug.pdf`.

[28] *SC3000 Series Robot Systems, C3150 Controller Hardware Manual*, NIDEC SANKYO CORPORATION, 2005.

[29] *SC3000 Robot System, SSL/E Language Reference Manual Rev. 2*, NIDEC SANKYO CORPORATION, 2005.

[30] *SC3000 Robot System, System Instructions Rev. 3*, NIDEC SANKYO CORPORATION, 2005.

[31] *SC3000 Series Robot Systems, Pendant Operation Manual Rev. 3*, NIDEC SANKYO CORPORATION, 2005.

[32] *SC3000 Series Robot Systems, Installation Manual Rev. 3*, NIDEC SANKYO CORPORATION, 2005.

[33] *Buzz2 User's Guide*, NIDEC SANKYO CORPORATION, 2005.

[34] *SC3000 Series Robot Systems, SCARA Series Hardware Manual*, NIDEC SANKYO CORPORATION, 2005.

[35] *Space Saving Vacuum Ejector Series ZQ Datasheet*, SMC Corporation of America.

[36] Milind Sudhir Rokade. *Modeling, simulation and virtual reality based visualisation of SCARA robot*, UCC (Cork), 2014.