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

5 Matlab Robot Functions .............................................. 39
  5.1 Coordinate Systems ............................................. 39
  5.2 Point to point motion functions ................................. 41
    5.2.1 Motion in the Cartesian Coordinate System ............... 41
    5.2.2 Motion in the Joint Coordinate System .................. 43
  5.3 Continuous Path motion functions ............................. 46
  5.4 Speed and acceleration/deceleration functions .............. 55
  5.5 Arm mode functions ............................................. 61
  5.6 Mark functions .................................................. 62
  5.7 I/O functions ................................................... 63
  5.8 Pendant output message functions ............................. 67
  5.9 Palletizing functions .......................................... 69
  5.10 Sampling Mode .................................................. 71
    5.10.1 How it works .............................................. 72

6 Vision based applications .......................................... 75
  6.1 HD cam, image acquisition and processing ..................... 75
    6.1.1 HD camera .................................................. 75
    6.1.2 Image acquisition and processing ........................ 77
    6.1.3 Object position detection ................................. 78
  6.2 Vacuum Gripping System ........................................ 78
  6.3 Vibrating surface ............................................... 79
  6.4 Vacuum ejector and motor drive circuit ....................... 81
    6.4.1 Schematic diagram ........................................ 81
    6.4.2 P1 and P2 ................................................. 81
  6.5 Keys pick and place application .............................. 83
    6.5.1 Aim of the application ................................... 83
    6.5.2 Keys detection ............................................ 84
    6.5.3 Application execution .................................... 88
    6.5.4 Conclusion ................................................ 89
  6.6 Coloured discs pick and place application ................... 94
    6.6.1 Aim of the application ................................... 94
    6.6.2 Discs detection ............................................ 94
    6.6.3 Application execution .................................... 100
    6.6.4 Conclusion ................................................ 101

7 Graphical User Interface ........................................... 107

8 Simulink virtual robot .............................................. 113
9 Conclusion and future work 115

A Code 117

Bibliography 150

References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 150
List of Figures

2.1 The Hirata AR-300, one of the first model of a SCARA. ............... 6
2.2 SCARA Sankyo SR8408. ............................................. 7
2.3 Example of the four Kinematic Parameters $\theta_i, d_i, a_i, \alpha_i$. With those four parameters, the coordinates can be translated from $O_i$ to $O_{i-1}$. 8
2.4 SCARA frames placement. ............................................. 9
2.5 SCARA top view scheme. ............................................. 11
3.1 Base of the robot. .................................................... 14
3.2 General assembly. .................................................... 15
3.3 Showing the Robot with the covers in place. ......................... 16
3.4 Robot partially uncovered. ......................................... 16
3.5 $\Theta_1$ and $\Theta_2$ motors. ....................................... 19
3.6 $\Theta_2$ motor. ........................................................ 19
3.7 $\Theta_2$ motor and connectors. ..................................... 20
3.8 Rotor of $\Theta_2$ motor. ............................................. 20
3.9 Encoder electronic board. .......................................... 21
3.10 Harmonic Drive. .................................................... 21
3.11 Harmonic Drive functioning. ..................................... 22
3.12 Roll motor. ........................................................ 22
3.13 Roll motor belts. ................................................... 22
3.14 Z motor and electromagnetic clutch. ............................. 23
3.15 Z axis motion unit and break unit. ................................ 23
4.1 Hardware configuration. ............................................. 26
4.2 Programming, and program execution in Buzz2. ..................... 27
4.3 Programming, and program execution with Matlab. ............... 28
4.4 Interpreter functioning ............................................. 31
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>RS232 cable</td>
<td>32</td>
</tr>
<tr>
<td>4.6</td>
<td>Pseudo code explaining the synchronization protocol between Matlab and the Interpreter from the Interpreter side.</td>
<td>35</td>
</tr>
<tr>
<td>4.7</td>
<td>Function <code>serial_out</code> flowchart.</td>
<td>37</td>
</tr>
<tr>
<td>5.1</td>
<td>Cartesian Coordinate System.</td>
<td>40</td>
</tr>
<tr>
<td>5.2</td>
<td>Joint Coordinate System.</td>
<td>40</td>
</tr>
<tr>
<td>5.3</td>
<td>Arc motion example</td>
<td>49</td>
</tr>
<tr>
<td>5.4</td>
<td>Circular motion example</td>
<td>50</td>
</tr>
<tr>
<td>5.5</td>
<td>Possible starting points for circular motion in X-Y plane.</td>
<td>51</td>
</tr>
<tr>
<td>5.6</td>
<td>Circular motion example by using <code>xycir</code>.</td>
<td>52</td>
</tr>
<tr>
<td>5.7</td>
<td>Possible starting points for circular motion in X-Z plane.</td>
<td>53</td>
</tr>
<tr>
<td>5.8</td>
<td>Circular motion example by using <code>xzcir</code>.</td>
<td>53</td>
</tr>
<tr>
<td>5.9</td>
<td>Possible starting points for a circle in Y-Z plane.</td>
<td>54</td>
</tr>
<tr>
<td>5.10</td>
<td>Circular motion example by using <code>yzcir</code>.</td>
<td>54</td>
</tr>
<tr>
<td>5.11</td>
<td>PTP motion speed profile.</td>
<td>57</td>
</tr>
<tr>
<td>5.12</td>
<td>The three areas of the workspace.</td>
<td>61</td>
</tr>
<tr>
<td>5.13</td>
<td>Examples of pallet definition.</td>
<td>70</td>
</tr>
<tr>
<td>5.14</td>
<td>Trajectory</td>
<td>73</td>
</tr>
<tr>
<td>5.15</td>
<td>$\Theta_1$ and $\Theta_2$ trends during the motion.</td>
<td>74</td>
</tr>
<tr>
<td>6.1</td>
<td>The Creative Live! Cam Chat HD.</td>
<td>75</td>
</tr>
<tr>
<td>6.2</td>
<td>Camera position (upper view).</td>
<td>76</td>
</tr>
<tr>
<td>6.3</td>
<td>Camera position (side view).</td>
<td>76</td>
</tr>
<tr>
<td>6.4</td>
<td>Camera position (close views).</td>
<td>76</td>
</tr>
<tr>
<td>6.5</td>
<td>Air pressure regulator and Vacuum Ejector</td>
<td>79</td>
</tr>
<tr>
<td>6.6</td>
<td>Hose second arm connection</td>
<td>80</td>
</tr>
<tr>
<td>6.7</td>
<td>Vacuum cup (end effector)</td>
<td>80</td>
</tr>
<tr>
<td>6.8</td>
<td>Vibrating surface.</td>
<td>81</td>
</tr>
<tr>
<td>6.9</td>
<td>Vacuum ejector and motor drive circuit schematic diagram.</td>
<td>82</td>
</tr>
<tr>
<td>6.10</td>
<td>Relays K1, K2, and connectors.</td>
<td>82</td>
</tr>
<tr>
<td>6.11</td>
<td>EX. I/O-2 connector</td>
<td>82</td>
</tr>
<tr>
<td>6.12</td>
<td>Random key placement</td>
<td>84</td>
</tr>
<tr>
<td>6.13</td>
<td>Ordered key placement</td>
<td>84</td>
</tr>
<tr>
<td>6.14</td>
<td>Function keys_detection flowchart</td>
<td>85</td>
</tr>
<tr>
<td>6.15</td>
<td>BW image obtained by using <code>opt_threshold</code>_detection_threshold.</td>
<td>86</td>
</tr>
<tr>
<td>6.16</td>
<td>Image obtained by using the function <code>image_processing</code>.</td>
<td>86</td>
</tr>
</tbody>
</table>
6.17 Translated 'Centroid' positions for a generic keys configuration. ... 88
6.18 Function keys_pick_and_place flowchart. ......................... 90
6.19 Nine key configuration before vibrating. ......................... 90
6.20 Nine key configuration after vibrating. ......................... 91
6.21 First key pick up action. ..................................... 91
6.22 First key place down action. ................................. 91
6.23 Key configuration after nine keys picked up. .................. 92
6.24 Placement of four keys before vibrating. ....................... 92
6.25 Ninth key pick up action. ................................. 92
6.26 Ninth key place down action. ................................. 93
6.27 The execution is completed. ................................ 93
6.28 Random coloured discs placement and containers. ............ 94
6.29 True colour image. ..................................... 95
6.30 Red channel. ........................................... 96
6.31 Green channel. ........................................... 96
6.32 Blue channel. ........................................... 96
6.33 Red channel BW conversion. .................................. 97
6.34 Green channel BW conversion. .................................. 97
6.35 Blue channel BW conversion. .................................. 97
6.36 Image after OR operation between channels. .................. 98
6.37 Processed image. ....................................... 98
6.38 Translated 'Centroid' positions for a generic discs configuration. ... 99
6.39 Random disc placement before vibrating. ....................... 101
6.40 Random disc configuration after vibrating. .................... 102
6.41 Red disc pick up operation. .................................. 102
6.42 Red disc place down operation. ................................ 102
6.43 Green disc pick up operation. ................................ 103
6.44 Green disc place down operation. ................................ 103
6.45 Blue disc pick up operation. .................................. 103
6.46 Blue disc place down operation. ................................ 104
6.47 White disc pick up operation. .................................. 104
6.48 White disc place down operation. ................................ 104
6.49 The execution is completed. .................................. 105
7.1 GUI G1 ................................................................. 108
7.2 GUI G2 ................................................................. 109
7.3 GUI G3 ................................................................. 110
8.1 The robot virtual model ........................................... 113
List of Tables

2.1 Link and joint parameters ........................................... 9
3.1 Main operative features of Sankyo SR8408. ..................... 13
3.2 Item list for Figure 3.2. ........................................... 14
3.3 Item list for Figure 3.3. ........................................... 15
3.4 Item list for Figure 3.4. ........................................... 17
3.5 Θ₁ and Θ₂ motor specifications .................................. 17
6.1 Item list for Figure 3.4. ........................................... 79
6.2 Item list for Figure 6.9. ........................................... 81
6.3 Relays switch commands ........................................... 82
6.4 Properties range of values defining a key pattern .............. 87
6.5 Properties range of values defining a disc pattern .......... 99
7.1 Item list for Figure 7.2. ........................................... 111
7.2 Item list for Figure 7.3. ........................................... 112
Listings

4.1 Matlab program: six items are moved from position P1 to a position P2. ................................................. 28
4.2 Matlab code for point to point motion. ......................................................... 29
4.3 Matlab program that configures communication settings of the connection and reads, and sends data through a RS232 port. ...... 32
4.4 SSL/E program that configures communication settings of the connection and for reads, and sends data through a RS232 port. ..... 33
5.1 Trajectory sampling mode example. ................................................................. 72
5.2 Angles sampling mode example. ................................................................. 73
6.1 Matlab application function start.cam. ......................................................... 77
6.2 Function keys_detection piece of code. .................................................. 87
6.3 Function discs&colours_detection piece of code. ............................. 95
A.1 Function serial_out1................................................................. 117
A.2 Function position ................................................................. 119
A.3 Function keys_detection ................................................................. 119
A.4 Script keys_pick_and_place ................................................................. 125
A.5 Function discs&colours_detection .................................................. 128
A.6 Script coloured_discs_pick_and_place ........................................ 133
A.7 Interpreter (SSL/E program) ................................................................. 136
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 overall 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 interpreter 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;
- maintain 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 hardare limitations in terms of I/O 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.](image)

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.](image)

### 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* \((\alpha_i)\). In detail:

- \(a_i\) is the common normal distance between the joint axes, measured from the axis of joint \(i\) to axis of joint \(i + 1\);

- \(\alpha_i\) 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* \((\Theta_i)\).
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 $a_i$;
- $\Theta_i$ 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 $\theta_i, d_i, a_i, \alpha_i$. With those four parameters, the coordinates can be translated from $O_i$ to $O_{i-1}$.](image)

Once all the frames are positioned, the parameters $a_k$, $\alpha_k$, $d_k$ and $\Theta_k$ 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 $4 \times 4$ 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:

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

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.

<table>
<thead>
<tr>
<th>Axis</th>
<th>$\Theta$</th>
<th>d</th>
<th>a</th>
<th>$\alpha$</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\Theta_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
<td>$180^\circ$</td>
<td>$0^\circ$</td>
</tr>
<tr>
<td>2</td>
<td>$\Theta_2$</td>
<td>0</td>
<td>$l_3$</td>
<td>$0^\circ$</td>
<td>$0^\circ$</td>
</tr>
<tr>
<td>3</td>
<td>$0^\circ$</td>
<td>$d_3$</td>
<td>0</td>
<td>$0^\circ$</td>
<td>$d_{max}$</td>
</tr>
<tr>
<td>4</td>
<td>$\Theta_4$</td>
<td>$l_4$</td>
<td>0</td>
<td>$0^\circ$</td>
<td>$90^\circ$</td>
</tr>
</tbody>
</table>
The matrix relating tool position to the base frame is $T_{\text{tool}}^{\text{base}}$.

$$T_{\text{base}} = T_0^1 T_1^2 T_2^3 T_3^4$$  \hspace{1cm} (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:

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

where

$$\Theta_{1-2-4} = \Theta_1 - \Theta_2 - \Theta_4$$  \hspace{1cm} (2.4)$$

$$\Theta_{1-2} = \Theta_1 - \Theta_2.$$  \hspace{1cm} (2.5)$$

**Inverse Kinematics**

Often the matrix $T_{\text{tool}}^{\text{base}}$ 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.

Given the numerical matrix:

\[
T_{\text{tool}} = \begin{bmatrix}
R_{11} & R_{12} & R_{13} & p_x \\
R_{21} & R_{22} & R_{23} & p_y \\
R_{31} & R_{32} & R_{33} & p_z \\
0 & 0 & 0 & 1
\end{bmatrix},
\]

the value of \( d_3 \) can be easily found:

\[
d_3 = l_1 - l_4 - p_z.
\]

To find \( \Theta_2 \):

\[
p_x^2 + p_y^2 = l_2^2 + l_3^2 - 2l_2l_3 \cos (180^\circ - \Theta_2) = l_2^2 + l_3^2 + 2l_2l_3 \cos \Theta_2
\]

Because \( p_x \) and \( p_y \) are known, it is obtained:

\[
\Theta_2 = \pm \arccos \left( \frac{p_x^2 + p_y^2 - l_2^2 - l_3^2}{2l_2l_3} \right).
\]

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 $\Theta_1$:

$$p_x = l_2 C_1 + l_3 C_{1-2} = (l_2 + l_3 C_2)C_1 + l_3 S_2 S_1$$  \hspace{1cm} (2.10)$$

$$p_y = l_2 S_1 + l_3 S_{1-2} = -l_3 S_2 C_1 + (l_2 + l_3 C_2)S_1$$  \hspace{1cm} (2.11)$$

$$\begin{bmatrix} C_1 \\ S_1 \end{bmatrix} = \begin{bmatrix} l_1 + l_2 C_2 & l_2 S_2 \\ -l_2 S_2 & l_1 + l_2 C_2 \end{bmatrix}^{-1} \begin{bmatrix} p_x \\ p_y \end{bmatrix}$$  \hspace{1cm} (2.12)$$

$$\Theta_1 = \arctan 2( l_2 S_2 p_x + (l_1 + l_2 C_2) p_y , (l_1 + l_2 C_2) p_x - l_2 S_2 p_y )$$  \hspace{1cm} (2.13)$$

If it is necessary $\Theta_4$ can be found:

$$R_{21} = S_{1-2-4}$$  \hspace{1cm} (2.14)$$

$$R_{11} = C_{1-2-4}$$  \hspace{1cm} (2.15)$$

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

$$= \Theta_1 - \Theta_2 - \arctan 2(R_{21}, R_{11}).$$  \hspace{1cm} (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>
<thead>
<tr>
<th>Table 3.1: Main operative features of Sankyo SR8408.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arm length</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Operative area</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Maximum speed</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Repeatability</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Load-carrying capacity</td>
</tr>
<tr>
<td>Max couple</td>
</tr>
<tr>
<td>Weight</td>
</tr>
</tbody>
</table>

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.
CHAPTER 3. THE SANKYO SR8408

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 $\Theta_1$ and $\Theta_2$. 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>
<thead>
<tr>
<th>Item</th>
<th>Parts name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>$\Theta_1$ motor</td>
</tr>
<tr>
<td>B1</td>
<td>$\Theta_2$ motor</td>
</tr>
<tr>
<td>C1</td>
<td>Z axis motor</td>
</tr>
<tr>
<td>D1</td>
<td>Roll axis motor</td>
</tr>
<tr>
<td>E1</td>
<td>$\Theta_1$ harmonic drive (inside)</td>
</tr>
<tr>
<td>F1</td>
<td>$\Theta_2$ harmonic drive (inside)</td>
</tr>
<tr>
<td>G1</td>
<td>Roll axis reduction gear unit</td>
</tr>
<tr>
<td>H1</td>
<td>Z axis shaft</td>
</tr>
<tr>
<td>I1</td>
<td>Z axis brake unit</td>
</tr>
<tr>
<td>J1</td>
<td>Connector panel</td>
</tr>
<tr>
<td>K1</td>
<td>Flexible cable hose connection and serial port</td>
</tr>
<tr>
<td>L1</td>
<td>$\Theta_1$ arm</td>
</tr>
<tr>
<td>M1</td>
<td>$\Theta_2$ arm</td>
</tr>
<tr>
<td>N1</td>
<td>Z axis pulley</td>
</tr>
<tr>
<td>O1</td>
<td>Z axis belt</td>
</tr>
</tbody>
</table>
Figure 3.2: General assembly.
Figure 3.3: Showing the Robot with the covers in place.

Figure 3.4: Robot partially uncovered.
3.2 Motors and motion transmission

Rotational joints motion

Both $\Theta_1$ and $\Theta_2$ motors are AC servo motors. See table 3.5 for their specifications.

<table>
<thead>
<tr>
<th>Item</th>
<th>Parts name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2</td>
<td>$\Theta_1$ motor</td>
</tr>
<tr>
<td>B2</td>
<td>$\Theta_1$ harmonic drive (inside)</td>
</tr>
<tr>
<td>C2</td>
<td>$\Theta_2$ motor</td>
</tr>
<tr>
<td>D2</td>
<td>$\Theta_2$ harmonic drive (inside)</td>
</tr>
<tr>
<td>E2</td>
<td>Z axis brake unit</td>
</tr>
<tr>
<td>F2</td>
<td>Roll axis motor</td>
</tr>
<tr>
<td>G2</td>
<td>Z axis shaft</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Parts name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A3</td>
<td>$\Theta_1$ motor</td>
</tr>
<tr>
<td>B3</td>
<td>$\Theta_1$ harmonic drive</td>
</tr>
<tr>
<td>C3</td>
<td>Roll axis belt 1</td>
</tr>
<tr>
<td>D3</td>
<td>$\Theta_2$ motor</td>
</tr>
<tr>
<td>E3</td>
<td>$\Theta_2$ harmonic drive</td>
</tr>
<tr>
<td>F3</td>
<td>Roll axis motor</td>
</tr>
<tr>
<td>G3</td>
<td>Z axis brake unit</td>
</tr>
<tr>
<td>H3</td>
<td>Z axis shaft</td>
</tr>
<tr>
<td>I3</td>
<td>Z axis pulley</td>
</tr>
<tr>
<td>J3</td>
<td>Z axis belt</td>
</tr>
<tr>
<td>K3</td>
<td>Roll axis belt 2 (inside)</td>
</tr>
<tr>
<td>L3</td>
<td>Roll axis pulley</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Motor</th>
<th>Power (W)</th>
<th>Speed (rpm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Theta_1$</td>
<td>366</td>
<td>4000</td>
</tr>
<tr>
<td>$\Theta_2$</td>
<td>267</td>
<td>4000</td>
</tr>
</tbody>
</table>

The two motors are shown in Figure 3.5. In Figures 3.6, 3.7, and 3.8, $\Theta_2$ 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

$\Theta_1$ and $\Theta_2$ 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 $\Theta_1$ 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.
3.2. MOTORS AND MOTION TRANSMISSION

(a) $\Theta_1$ motor. (b) $\Theta_2$ motor.

Figure 3.5: $\Theta_1$ and $\Theta_2$ motors.

Figure 3.6: $\Theta_2$ motor.
CHAPTER 3. THE SANKYO SR8408

Figure 3.7: Θ₂ motor and connectors.

Figure 3.8: Rotor of Θ₂ motor.
Figure 3.9: Encoder electronic board.

Figure 3.10: Harmonic Drive.

(a) Disassembled Harmonic Drive.  
(b) Assembled Harmonic Drive.
Figure 3.11: Harmonic Drive functioning.

Figure 3.12: Roll motor.

Figure 3.13: Roll motor belts.
3.2. MOTORS AND MOTION TRANSMISSION

(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
CHAPTER 4. HW CONFIGURATION AND INTERPRETER

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.

```
1  speed(4);  % Sets the speed (4% of the maximum speed)
2  PIECE_POS=[280,280,10,112];  % Cartesian Position of a piece
3    % [X (mm), Y (mm), Z (mm), rotation
4    % along the Z-axis (deg)]
5  RELEASE_POS=[280,280,10,112];  % Cartesian Position of the
6    % release position [X (mm),
7    % Y (mm), Z (mm), rotation
8    % along the Z-axis (deg)]
```
4.2. INTERPRETER

move(PIECE_POS); % Moves to PIECE_POS
i=1; % Iteration variable
% Cycle for picking up 6 pieces in the workspace (It has
% been considered all the pieces in the same position)
while(i<7)
  move(PIECE_POS); % Moves to PIECE_POS
  out(937,1); % Activates vacuum device in order
              % to pick up a piece by using a sucker
  smove(3,60); % Moves only the third axis (z-axis) straight
              % down in order to reach the piece
  smove(3,-60); % Moves only the third axis (z-axis)
               % straight up
  move(RELEASE_POS); % Moves to RELEASE_POS
  out(937,0); % Deactivates vacuum device, and releases
              % the piece
  pause(0.2); % Delay for allowing piece release (s)
  i=i+1; % Iteration variable updating
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.

P1=[280,280,10,112]; % Cartesian Position of a piece
  % [X (mm), Y (mm), Z (mm), rotation
  % along the Z-axis (deg)]
P2=[-280,280,10,112]; % Cartesian Position of a piece
  % [X (mm), Y (mm), Z (mm), rotation
  % along the Z-axis (deg)]
speed(10); % Sets the speed (10% of the maximum speed)
i=1; % Iteration variable
while (i < 11)

    move(P1); % Moves to P1
    pause(1); % Waits 1 second

    move(P2); % Moves to P2
    pause(1); % Waits 1 second

    i = i + 1; % Iteration variable updating
end

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

function [] = move( A )

    x = serial_out1(1000,A);

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](image-url)
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].

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);
% Connects the RS232 port object to the device
fopen(s);
% Converts the integer variable A into
% a string
A=num2str(A)
```
4.3. SERIAL COMMUNICATION AND SYNCHRONIZATION

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

```plaintext
INT I; // Integer variable
REAL A=25.5; // Real variable
STRING S1="ABCD"; // String
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
```
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 Matlab 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.
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 \texttt{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 message 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, \texttt{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 \texttt{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 \texttt{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 \texttt{startup} function

The Matlab auxiliary function \texttt{startup} is executed at Matlab startup. It includes commands that initialize important state variables.

4.4.2 The \texttt{state\_keeper} function

The Matlab auxiliary function \texttt{state\_keeper} is an important function that includes important persistent variables shared by some functions. Persistent variables are local to the function \texttt{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 \cite{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 \texttt{state	extunderscore keeper} with a specific string as input parameter. The function \texttt{state	extunderscore keeper} compares the input string with two strings defined inside it. Depending on the string comparing result, \texttt{state	extunderscore keeper} allows updating or retrieval of these variables, or denies these operations.

4.4.3 The functions \texttt{serial	extunderscore 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: \texttt{serial	extunderscore out1}, \texttt{serial	extunderscore out2}, \texttt{serial	extunderscore out3}, \texttt{serial	extunderscore out4}, \texttt{serial	extunderscore out5}. The structure of these five functions is pretty much the same. The \texttt{serial	extunderscore out1} flowchart and source code is shown in Figure 4.7 and Listing A.1 in appendix A. For \textit{Sample mode} refer to chapter 5.10.

![Figure 4.7: Function serial_out1 flowchart.](image-url)
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 described 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 \( P_1 = [x, y, z, s] \) 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: \( P_1 = [280, 280, 10, 112] \)
Joint Coordinate System

In Matlab, a vector of four numbers $P1 = [t_1, t_2, z, s]$ 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: $P1 = [90, 15, 35, 220]$
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 \texttt{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

\textbf{MOVE}

Moves the robot to a position specified in the Cartesian coordinate system.

\textbf{Syntax:} \texttt{move(P)}

\textbf{Input:}

- \texttt{P:} matrix of \(N\times4\) 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.

\textbf{Return value:} none. \textbf{Example}

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

\textbf{MOVED}

Moves the robot to a position in the Cartesian coordinate system and not yet declared.

\textbf{Syntax:} \texttt{moved(x,y,z,s)}

\textbf{Input:}

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

Return value: none.

Example

1. \texttt{move(280,280,10,112);} % Moves to the point
2. \texttt{(280,280,10,112)}

\section*{RMOVE}

Moves the robot to a position specified relative to the current position in the Cartesian coordinate system.

Syntax: \texttt{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

1. \texttt{P1=[280,280,10,112];} % Defines position P1
2. \texttt{move(P1);} % Moves to P1
3. \texttt{rmove(20,10,30,10);} % Moves to (300,290,40,122)
4. \texttt{P1=[280,280,10,112];} % Defines position P1
5. \texttt{move(P1);} % Moves to P1
6. \texttt{rmove(-20,-10,30,10);} % Moves to (260,270,40,122)

\section*{SMOVE}

Moves the robot by changing only one of the three Cartesian coordinates or the Roll/S-axis position.

Syntax: \texttt{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**

```plaintext
1. P1=[280,280,10,112];  % Defines position P1
2. move(P1);             % Moves to P1
3. smove(1,30);          % Moves to (300,280,10,112)
4. smove(3,50);          % Moves to (300,280,50,112)
5. 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**

```plaintext
1. P1=[280,280,10,112];  % Defines position P1
2. move(P1);             % Moves to P1
3. srmove(1,30);         % Moves to (310,280,10,112)
4. srmove(3,50);         % Moves to (300,280,60,112)
5. 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**:

```plaintext
```
- \( P \): matrix of \( N \times 4 \) 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

```matlab
1 P1=[90,30,10,112];  % Defines position P1
2 jmove(P1);         % Moves to P1
3
4 P=[90,30,10,112;  % Defines a matrix P of 4 positions
5     110,50,10,112;
6     90,-45,60,200;
7     70,10,10,-200];
8 jmove(P);         % Moves to the positions defined in P
9     % in row order
```

**JMOVED**

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

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

```matlab
1 jmoved(90,30,10,112);  % Moves to the point
2     % (90,30,10,112)
```

**RJMOVE**

Moves the robot to a position specified relative to the current position in the joint coordinate system.

Syntax: \( \text{rjmove}(x,y,z,s) \)

Input:

- \( t1 \): 1\(^{st} \) arm angle variation (deg);
- \( t2 \): 2\(^{nd} \) arm angle variation (deg);
5.2. **POINT TO POINT MOTION FUNCTIONS**

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

**Return value:** none.

**Example**

```plaintext
1 P1=[90,30,10,112];  % Defines position P1
2 jmove(P1);          % Moves to P1
3 rjmove(20,10,30,10); % Moves to (110,40,40,122)
4 P1=[90,30,10,112];  % Defines position P1
5 jmove(P1);          % Moves to P1
6 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**

```plaintext
1 P1=[90,30,10,112];  % Defines position P1
2 jmove(P1);          % Moves to P1
3 sjmove(1,20);       % Moves to (20,30,10,112)
4 sjmove(3,50);       % Moves to (20,50,10,112)
5 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:**
CHAPTER 5. MATLAB ROBOT FUNCTIONS

- 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

```matlab
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 coordinate 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

```matlab
P1=[280,280,10,112]; % Defines position P1
lmove(P1); % Moves to P1
P=[280,280,10,112; 300,280,10,112; 400,0,60,200; 0,300,10,112]; % Defines a matrix P of 4 positions
lmove(P); % Moves to the positions defined in P in row order
```
5.3. CONTINUOUS PATH MOTION FUNCTIONS

**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**

```plaintext
1 lmove(280,280,10,112); % Moves to the point
2 % (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**

```plaintext
1 P1=[280,280,10,112]; % Defines position P1
2 lmove(P1); % Moves to P1
3 rlmove(20,10,30,10); % Moves to (300,290,40,122)
5 P1=[280,280,10,112]; % Defines position P1
7 lmove(P1); % Moves to P1
8 rlmove(-20,-10,30,10); % Moves to (260,270,40,122)
```
CHAPTER 5. MATLAB ROBOT FUNCTIONS

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

```matlab
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

```matlab
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)
```
5.3. **CONTINUOUS PATH MOTION FUNCTIONS**

**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**

```plaintext
1. P1 = [??, ??, ??, ??];  % Generic position inside the
  % workspace
2. P2 = [??, ??, ??, ??];  % Generic position inside the
  % workspace
3. P3 = [??, ??, ??, ??];  % Generic position inside the
  % workspace
4. P4 = [??, ??, ??, ??];  % Generic position inside the
  % workspace
5. P5 = [??, ??, ??, ??];  % Generic position inside the
  % workspace
6. P6 = [??, ??, ??, ??];  % Generic position inside the
  % workspace
7. lmove(P1);  % Moves to P1
8. lmove(P2);  % Moves to P2
9. archmove(P3, P4);  % Performs an arc motion that
  % interpolates P2, P3, P4
10. 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

```
1 P1 = [??, ??, ??, ??]; % Generic position inside the
2 % workspace
3 P2 = [??, ??, ??, ??]; % Generic position inside the
4 % workspace
5 P3 = [??, ??, ??, ??]; % Generic position inside the
6 % workspace
7
8 lmove(P1); % Moves to P1
9 lmove(P2); % Moves to P2
10 cmove(P3, P4, P2); % Performs a circular motion
```

Figure 5.4: Circular motion example
5.3. CONTINUOUS PATH MOTION FUNCTIONS

**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**

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

![Figure 5.5: Possible starting points for circular motion in X-Y plane.](image)

**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 \textit{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).

\textbf{Return value}: none.

\textbf{Example}

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

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

```matlab
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

```plaintext
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

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

**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 `autoalcl` example.

Set with 0 makes the motion through multiple points very smooth. De-
5.4. SPEED AND ACCELERATION/DECELERATION FUNCTIONS

Figure 5.11: PTP motion speed profile.

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.

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
5.4. SPEED AND ACCELERATION/DECELERATION FUNCTIONS

```
1  speed(30)  % Sets 30% as maximum PTP
2               % motion speed
3
4  P1=[300,280,10,112];  % Defines position P1
5  P2=[-300,200,15,140];  % Defines position P2
6
7  move(P1);  % Moves to P1 in automatic
8               % acceleration & deceleration
9               % since autoacl has not be
10              % disabled.
11
12  acct(2);  % It is ignored, since autoacl
13         % has not be disabled
14
15  dacct(2);  % It is ignored, since autoacl
16              % has not be disabled
17
18  autoacl(0);  % Disables the automatic acceleration
19               % and deceleration settings
20
21  acct(2);  % Set 2 s as accelerating time
22
23  dacct(2.6);  % Set 2.6 s as decelerating time
24
25  move(P1);  % Moves to P1 (PTP motion).
26               % The acceleration a deceleration
27               % time are set to 2 s and 2.6 s
28               % respectively
29
30  autoacl(1);  % Enables the automatic acceleration
31               % and deceleration settings
32
33  weight(2);  % Weight is 2 kg
34
35  move(P1);  % Moves to P1 in automatic
36               % 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**

```
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
```
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**

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**

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

![Figure 5.12: The three areas of the workspace.](image)

**LEFT**

See comments about `right` above.

**Syntax:** `left()`
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

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

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.
5.7. I/O Functions

Syntax: \texttt{jmark()}

Input: none.

Return value: a position, i.e. a vector of four numbers.

Example

\begin{verbatim}
1 P1=[300,280,10,112];  % Defines position P1
2 move(P1);             % Moves to P1
3 srmotion(1,-100);     % Moves only the first axis
4 srmotion(2,-10);      % Moves only the second axis
5 P_current = jmark();  % Calculates the current
6 % position and stores it
7
8 \end{verbatim}

5.7 I/O functions

IN

Checks the status of any digital input (DI) port.

Syntax: \texttt{in(a)}

Input:

- \texttt{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

\begin{verbatim}
1 P1=[300,280,10,112];  % Defines position P1
2 P2=[-300,200,15,140]; % Defines position P2
3 port_status = in(922);  % Checks the status of
4 % the port 922
5
6 if(port_status==1)  % If port 922 is ON
7   move(P1);        % it moves to P1, if
8 else                % port 922 is OFF it
9   move(P2);        % moves to P2
10 \end{verbatim}
CHAPTER 5. MATLAB ROBOT FUNCTIONS

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

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

% Moves to P1
move(P1);

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

i=1; % Iteration variable

while(i<11) % This cycle is for blinking the output
  out(18,1); % port with a period of 1 s
  pause(1);
  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.
5.7. I/O FUNCTIONS

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

```
1 P1=[300,280,10,112]; % Defines position P1
2 wintime(5); % Sets the waiting time
3  % to 5 s
4 win(10,1); % Waits for state 1 of port 10. If this
5  % doesn't happen within 5 seconds,
6  % time-out error is caused and the
7  % program stops.
8 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

```
1 P1=[300,280,10,112]; % Defines position P1
```
CHAPTER 5. MATLAB ROBOT FUNCTIONS

2 \texttt{P2=[-300,200,15,140];} \quad \text{\% Defines position P2}
3 \texttt{wintime(5);} \quad \text{\% Sets the waiting time to 5 s}
5
6 \texttt{x = tri(10,1);} \quad \text{\% Waits for state 1 of port 10.}
8 \text{\% If this happen within 5 seconds}
9 \text{\% x=1, else x=0}
10 \texttt{if(x==1)}
11 \texttt{move(P1);} \texttt{end}
12 \texttt{else}
13 \texttt{move(P2);} \texttt{end}

**BLINK**

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

**Syntax:** \texttt{blink(a, b, c)}

**Input:**

- \texttt{a}: number that identifies an output port (17-20 I/O-1, 21-24 I/O-2, 937-952 EX. I/O-2);

- \texttt{b}: time while the port is on (a positive real number);

- \texttt{c}: time while the port is off (a positive real number).

If “\texttt{c}” is not specified, the time while the port is off becomes the same as “\texttt{b}”.

**Return value:** none

**Example**

1 \texttt{blink(19,1,3);} \quad \text{\% Blinks port 19: time while the}
2 \texttt{\text{\% port is on=1 s, time while}}
3 \texttt{\text{\% the port 19 is off=3 s}}
4
5 \texttt{blink(20,2);} \quad \text{\% Blinks port 19: time while the}
6 \texttt{\text{\% port is on=2 s, time while}}
7 \texttt{\text{\% the port 19 is off=2 s}}
5.8. **PENDANT OUTPUT MESSAGE FUNCTIONS**

**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**

```plaintext
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

```matlab
1 x='Hello World'; % Defines string x
2 y='11111'; % Defines string y
3 z='UCC'; % Defines string z
4 locate(0,0); % Locates the position on the Pendant display
5 opeout(x); % Outputs the string x on Pendant display
6 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:
Linear pallet  \texttt{setplt(1,p0,p1,p0,p0,6,1,1)}

\begin{center}
\begin{tikzpicture}
\draw (0,0) -- (1,0) node[above] {p0} -- (2,0) node[above] {p1};
\draw[->] (0,0) -- (1,0.5) node[above] {A pallet element (to be accessed)};
\draw (1,0) -- (2,0) node[above] {2} -- (3,0) node[above] {3} -- (4,0) node[above] {4} -- (5,0) node[above] {5} -- (6,0) node[above] {6} -- (7,0) node[above] {7} -- (8,0) node[above] {8} -- (9,0) node[above] {9} -- (10,0) node[above] {10} -- (11,0) node[above] {11} -- (12,0) node[above] {12} -- (13,0) node[above] {13} -- (14,0) node[above] {14} -- (15,0) node[above] {15} -- (16,0) node[above] {16} -- (17,0) node[above] {17} -- (18,0) node[above] {18} -- (19,0) node[above] {19} -- (20,0) node[above] {20} -- (21,0) node[above] {21} -- (22,0) node[above] {22} -- (23,0) node[above] {23} -- (24,0) node[above] {24} -- (25,0) node[above] {25} -- (26,0) node[above] {26} -- (27,0) node[above] {27} -- (28,0) node[above] {28} -- (29,0) node[above] {29} -- (30,0) node[above] {30};
\end{tikzpicture}
\end{center}

\texttt{setplt(1,p0,p1,p0,p0,3,1,1)}

\begin{center}
P1 \ldots P0
3-2-1
\end{center}

Plane pallet  \texttt{setplt(2,p0,p1,p0,p0,6,5,1)}

\begin{center}
\begin{tikzpicture}
\draw (0,0) -- (1,1) node[above] {p0} -- (2,2) node[above] {p1};
\draw (0,0) -- (1,1) node[above] {25} -- (2,2) node[above] {26} -- (3,3) node[above] {27} -- (4,4) node[above] {28} -- (5,5) node[above] {29} -- (6,6) node[above] {30};
\draw (1,1) -- (2,2) node[above] {19} -- (3,3) node[above] {20} -- (4,4) node[above] {21} -- (5,5) node[above] {22} -- (6,6) node[above] {23} -- (7,7) node[above] {24};
\draw (2,2) -- (3,3) node[above] {13} -- (4,4) node[above] {14} -- (5,5) node[above] {15} -- (6,6) node[above] {16} -- (7,7) node[above] {17} -- (8,8) node[above] {18};
\draw (3,3) -- (4,4) node[above] {7} -- (5,5) node[above] {8} -- (6,6) node[above] {9} -- (7,7) node[above] {10} -- (8,8) node[above] {11} -- (9,9) node[above] {12};
\end{tikzpicture}
\end{center}

3-D pallet  \texttt{setplt(4,p0,p1,p2,p3,3,3,2)}

\begin{center}
\begin{tikzpicture}
\draw (0,0) -- (1,1) node[above] {p0} -- (2,2) node[above] {p1};
\draw (0,0) -- (1,1) node[above] {9} -- (2,2) node[above] {6} -- (3,3) node[above] {3};
\draw (1,1) -- (2,2) node[above] {8} -- (3,3) node[above] {5} -- (4,4) node[above] {2};
\draw (2,2) -- (3,3) node[above] {7} -- (4,4) node[above] {4} -- (5,5) node[above] {1};
\draw (3,3) -- (4,4) node[above] {18} -- (5,5) node[above] {15} -- (6,6) node[above] {12};
\draw (4,4) -- (5,5) node[above] {17} -- (6,6) node[above] {14} -- (7,7) node[above] {11};
\draw (5,5) -- (6,6) node[above] {16} -- (7,7) node[above] {13} -- (8,8) node[above] {10};
\end{tikzpicture}
\end{center}

Figure 5.13: Examples of pallet definition.
- n: pallet number (positive integer);

- a: point number (positive integer).

Return value: none.

Example

```plaintext
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 $\Theta_1$ and $\Theta_2$ 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_keeper`). 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.

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

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

---

```
cpspeed(50); % Sets the speed of linear motion
% and circular motion
P1=[313;300;90;112];
P2=[0;231;3;112];
P3=[-313;300;90;112];
move(P1); % Moves to P1 (PTP motion)
```

---

Figure 5.14: Trajectory.

Listing 5.2: Angles sampling mode example.
CHAPTER 5. MATLAB ROBOT FUNCTIONS

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

![Figure 5.15: Θ₁ and Θ₂ trends during the motion.](image)

Figure 5.15: $\Theta_1$ and $\Theta_2$ 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.
CHAPTER 6. VISION BASED APPLICATIONS

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`.

```matlab
function [o] = start_cam( a )
    persistent vid;

    if(strcmp(a,'open')) % The inner expression is true if
        % the input parameter is the string
        % 'open'

        pause on;

        % The video input object is stored in the variable vid
        vid = videoinput('winvideo',1);

        % The next statement sets to 1 the number of frames that
        % are captured each time 'trigger' is executed
        set(vid, 'FramesPerTrigger', 1);

        % The next statement sets 'TriggerRepeat' to inf, that
        % allows to use 'trigger? infinite times
        set(vid, 'TriggerRepeat', Inf);

        % Sets the object to
        % manual triggering

        triggerconfig(vid, 'manual'); % Starts the video capture
    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.](image)

<table>
<thead>
<tr>
<th>Item</th>
<th>Parts name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pressured air hose</td>
</tr>
<tr>
<td>2</td>
<td>Pressure regulator</td>
</tr>
<tr>
<td>3</td>
<td>Vacuum Ejector</td>
</tr>
<tr>
<td>4</td>
<td>Pressure port (input)</td>
</tr>
<tr>
<td>5</td>
<td>Solenoid valve</td>
</tr>
<tr>
<td>6</td>
<td>Suction filter</td>
</tr>
<tr>
<td>7</td>
<td>Vacuum port (output)</td>
</tr>
</tbody>
</table>

### 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
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.
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>
<thead>
<tr>
<th>Parts</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>Switch inside the robot controller. It can be opened or closed with Matlab robot function out.</td>
</tr>
<tr>
<td>K1</td>
<td>Relay for the vacuum ejector solenoid valve.</td>
</tr>
<tr>
<td>Ev1</td>
<td>Vacuum ejector solenoid valve.</td>
</tr>
<tr>
<td>P2</td>
<td>Switch inside the robot controller. It can be opened or closed with Matlab robot function out.</td>
</tr>
<tr>
<td>K2</td>
<td>Relay for the DC motor of the vibrating surface.</td>
</tr>
<tr>
<td>M1</td>
<td>Vibrating surface DC motor.</td>
</tr>
</tbody>
</table>

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.
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>
<thead>
<tr>
<th>Relays</th>
<th>ON</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1 (Valve)</td>
<td><code>out(937,1)</code></td>
<td><code>out(937,0)</code></td>
</tr>
<tr>
<td>K2 (Motor)</td>
<td><code>out(938,1)</code></td>
<td><code>out(938,0)</code></td>
</tr>
</tbody>
</table>
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.
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.
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:
- ‘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];

- ‘BoundingBox’: 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.

```
1 M1=image_processing(M1)
2 [L n] = bwlabel(M1);
3 stats = regionprops(L,'Area', 'Centroid', 'BoundingBox',
4 'Orientation', 'MajorAxisLength',
5 '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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Area'</td>
<td>2600 - 3100</td>
</tr>
<tr>
<td>'Perimeter'</td>
<td>490 - 670</td>
</tr>
<tr>
<td>'MajorAxisLength'</td>
<td>106 - 115</td>
</tr>
<tr>
<td>'MinorAxisLength'</td>
<td>40 - 43</td>
</tr>
</tbody>
</table>
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.](image)

### 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.
6.5. KEYS PICK AND PLACE APPLICATION

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.

![Random coloured discs placement and containers.](image)

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 $M_1$ (see Figure 6.30), $M_2$ (see Figure 6.31), and $M_3$ (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.

```matlab
M1 = M(:,:,1);
M2 = M(:,:,2);
M3 = M(:,:,3);
M1 = im2bw(M1, opt_thr);
M2 = im2bw(M2, opt_thr);
M3 = im2bw(M3, opt_thr);
M = M1 | M2 | M3;
M = image_processing(M);
[L n] = bwlabel(M);
stats = regionprops(L, 'Area', 'Centroid', 'Orientation', 'MajorAxisLength', 'MinorAxisLength', 'Perimeter');
```

Figure 6.29: True colour image.
CHAPTER 6. VISION BASED APPLICATIONS

Figure 6.30: Red channel.

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 bw\_label and region\_props 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>
<thead>
<tr>
<th>Property</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Area'</td>
<td>1950 - 2200</td>
</tr>
<tr>
<td>'Perimeter'</td>
<td>430 - 470</td>
</tr>
<tr>
<td>'MajorAxisLength'</td>
<td>51 - 56</td>
</tr>
<tr>
<td>'MinorAxisLength'</td>
<td>49 - 54</td>
</tr>
</tbody>
</table>

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.](image)

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 affect 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.
CHAPTER 6. VISION BASED APPLICATIONS

Figure 6.40: Random disc configuration after vibrating.

Figure 6.41: Red disc pick up operation.

Figure 6.42: Red disc place down operation.
6.6. COLOURED DISCS PICK AND PLACE APPLICATION

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, especially 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.

<table>
<thead>
<tr>
<th>Part</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Coordinates input textboxes The user can input the Cartesian coordinates of the target point.</td>
</tr>
<tr>
<td>2</td>
<td>Move button When it is pressed, the robot moves to the target position specified in the coordinates input text boxes (PTP motion).</td>
</tr>
<tr>
<td>3</td>
<td>Arm mode buttons Pressing one of these buttons, the user can select the arm mode (left or right).</td>
</tr>
<tr>
<td>4</td>
<td>Current position text boxes Show the current position.</td>
</tr>
<tr>
<td>5</td>
<td>Point creation button Pressing this button, the current position is output to the command window.</td>
</tr>
<tr>
<td>6</td>
<td>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.</td>
</tr>
<tr>
<td>7</td>
<td>Z step motion Allows Z axis PTP motion. The step of the motion can be input in a 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.</td>
</tr>
<tr>
<td>8</td>
<td>Acceleration 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.</td>
</tr>
<tr>
<td>9</td>
<td>Decceleration time text box 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.</td>
</tr>
<tr>
<td>10</td>
<td>Speed control slide bar For activating and disactivating the vacuum gripping system</td>
</tr>
<tr>
<td>11</td>
<td>Vacuum ON and OFF buttons Allows T4 PTP motion. The step of the motion can be input in a text box.</td>
</tr>
<tr>
<td>12</td>
<td>Back button. Allows the user to return to G1.</td>
</tr>
</tbody>
</table>
Table 7.2: Item list for Figure 7.3.

<table>
<thead>
<tr>
<th>Part</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Coordinates input text boxes</td>
</tr>
<tr>
<td>2</td>
<td>Move button</td>
</tr>
<tr>
<td>3</td>
<td>Current position text boxes</td>
</tr>
<tr>
<td>4</td>
<td>Point creation button</td>
</tr>
<tr>
<td>5</td>
<td>Speed control slide bar</td>
</tr>
<tr>
<td>6</td>
<td>Step motion control</td>
</tr>
<tr>
<td>7</td>
<td>Z step motion</td>
</tr>
<tr>
<td>8</td>
<td>Acceleration time text box</td>
</tr>
<tr>
<td>9</td>
<td>Decceleration time text box</td>
</tr>
<tr>
<td>10</td>
<td>Vacuum ON and OFF buttons</td>
</tr>
</tbody>
</table>
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.

![Figure 8.1: The robot virtual model.](image)
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 development 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 \texttt{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

```matlab
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(state_keeper('retrieve',2) == 0)
    if(state_keeper('retrieve',3) == 0)
        state_keeper('store',3,1);
        disp('Please be sure that "Start" on the pedant has been selected,'
          'then type the prog() statement to continue with programming,'
          'otherwise the next statement will be useless. '
        )
        state_keeper('store',1,1);
    else
        if(state_keeper('retrieve',4)==0)
```
\[n,m\] = size(A);  \% Size of the second input parameter

\% The next statement converts the number 1111
\% into a string and sends it to the robot
\% controller through the serial port \texttt{s} (and cable).
\% The code 1111 if for notifying the controller,
\% which is polling its serial port, about the
\% transmission
fprintf(s,num2str(1111));

\% The next statement converts the Code into a string
\% and sends it to the robot controller through the
\% serial port \texttt{s} (and cable)
fprintf(s,num2str(Code));

\% The next statement converts the number \(n\) into a
\% string and sends it to the robot controller through
\% the serial port \texttt{s} (and cable)
fprintf(s,num2str(n));

i=1;  \% iteration variable

\% The next cycle sends the items of \texttt{A} to the robot
\% controller through the serial port \texttt{s} (and cable)
while(i \leq n)
  fprintf(s,num2str(A(i,1)));  
  fprintf(s,num2str(A(i,2)));  
  fprintf(s,num2str(A(i,3)));  
  fprintf(s,num2str(A(i,4)));  
  i = i + 1;
end

\% The next statement performs a polling operation
\% of the serial port until it receives a feedback
\% message from the controller
x=fscanf(s)

\% If the inner expression is TRUE, an error has occurred
if(str2num(x) == 7777)
  state_keeper('store',4,0);
  disp('  ')  
  disp('A problem occurred. There are two possible reasons:')
  disp('  ')  
  disp('1)A robot ERROR has occurred. ')  
  disp('2)'Start" on the pedant has not been selected when you ...
  typed prog() or the next' )  
  disp('statement in Matlab.')  
  disp('  ')  
  disp('SOLUTION:')  
  disp('Be sure that "Start" on the pedant has been selected, then ...
  type the prog()')  
  disp('statement to continue with programming, otherwise the next ...
  statements ')  
  disp('will be useless.')  

In the case 1) obviously fix your code!!!

% The next statement disconnects serial 
% port object from device
fclose(s);

% The next statement removes the 
% serial port object from memory
delete(s);
state_keeper('store',1,1);
return;
end

% If the sample mode has been selected (see section 5.10), 
% the function sstorage is called in order 
% to store the position data coming from the controller
if(state_keeper('retrieve',8) == 1 |
    state_keeper('retrieve',8) == 2)
    sstorage(str2num(x));
end
end
end
end

Listing A.2: Function position

function [ o ] = position( a )
% This function receives as an input parameter the centroid 
% of an object in an image, and it outputs the position of 
% the centroid of the same object in the workspace.

% Pixel to mm conversion
a = a / 96 * 10;

% X coordinate
px = 5.5141 * a(1,1) + 0.2346 * a(1,2) - 270.5216;

% Y coordinate
py= 0.1974 * a(1,1) - 5.4673 * a(1,2) + 463.5217;

o=[px,py];
end

Listing A.3: Function keys_detection.

function [o] = keys_detection()
% 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 colour 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 statement 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] = bwlable(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', 
'EquivalentDiameter', '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
% The next four if statements detect if the key bow is
% pointing towards the top of the image (in this case
% '1' is stored in 'OBJ&PROP(j,3)', or towards the bottom of
% the image (in this case '2' is stored in 'OBJ&PROP(j,3)'

if(stats(i).Orientation ≥ 0 & stats(i).Orientation < 45 )
  if(stats(i).Centroid(1) < stats(i).BoundingBox(1) + ...
    stats(i).BoundingBox(3)/2)
    OBJ&PROP(j,3) = 1;
  else
    OBJ&PROP(j,3) = 2;
  end
end

if(stats(i).Orientation ≥ 45 & stats(i).Orientation < 90 )
  if(stats(i).Centroid(2) > stats(i).BoundingBox(2) + ...
    stats(i).BoundingBox(4)/2)
    OBJ&PROP(j,3) = 1;
  else
    OBJ&PROP(j,3) = 2;
  end
end

if(stats(i).Orientation < 0 & stats(i).Orientation ≥ - 45 )
  if(stats(i).Centroid(1) < stats(i).BoundingBox(1) + ...
    stats(i).BoundingBox(3)/2)
    OBJ&PROP(j,3) = 1;
  else
    OBJ&PROP(j,3) = 2;
  end
end

if(stats(i).Orientation ≤ -45 & stats(i).Orientation ≥ - 90 )
  if(stats(i).Centroid(2) < stats(i).BoundingBox(2) + ...
    stats(i).BoundingBox(4)/2)
    OBJ&PROP(j,3) = 1;
  else
    OBJ&PROP(j,3) = 2;
  end
end

% The next part of code is for detecting the centroid of each
% key and storing its image coordinates in 'OBJ&PROP(j,4)' % (x coordinates) and 'OBJ&PROP(j,5)' (y coordinates)

if(stats(i).Orientation ≤ 0)
  if(OBJ&PROP(j,3) == 1)
    OBJ&PROP(j,4) = round(stats(i).Centroid(2)+
      8*sind(stats(i).Orientation));
    OBJ&PROP(j,5) = round(stats(i).Centroid(1)-
      8*cosd(stats(i).Orientation));
  end
APPENDIX A. CODE

121  M1(round(stats(i).Centroid(2) +
122      8*sin(stats(i).Orientation)),
123      round(stats(i).Centroid(1) -
124      8*cos(stats(i).Orientation))) = 0;
125  else
126      OBJ&PROP(j,4)=round(stats(i).Centroid(2) +
127          8*sin(-stats(i).Orientation));
128      OBJ&PROP(j,5)=round(stats(i).Centroid(1) +
129          8*cos(-stats(i).Orientation));
130      M1(round(stats(i).Centroid(2) +
131          8*sin(-stats(i).Orientation)),
132          round(stats(i).Centroid(1) +
133          8*cos(-stats(i).Orientation))) = 0;
134  end
135
136  end
137
138  if(stats(i).Orientation > 0)
139      if(OBJ&PROP(j,3) == 1)
140          OBJ&PROP(j,4) = round(stats(i).Centroid(2) +
141            8*sin(stats(i).Orientation));
142          OBJ&PROP(j,5) = round(stats(i).Centroid(1) -
143            8*cos(stats(i).Orientation));
144          M1(round(stats(i).Centroid(2) +
145            8*sin(stats(i).Orientation)),
146            round(stats(i).Centroid(1)-8*cos(stats(i).Orientation))) = 0;
147      else
148          OBJ&PROP(j,4) = round(stats(i).Centroid(2) -
149            8*sin(stats(i).Orientation));
150          OBJ&PROP(j,5) = round(stats(i).Centroid(1) +
151            8*cos(stats(i).Orientation));
152          M1(round(stats(i).Centroid(2) -
153            8*sin(stats(i).Orientation)),
154            round(stats(i).Centroid(1)+8*cos(stats(i).Orientation))) = 0;
155      end
156  end
157
158  if(OBJ&PROP(j,3) == 1)
159      OBJ&PROP(j,4) = round(stats(i).Centroid(2) +
160          8*sin(stats(i).Orientation));
161      OBJ&PROP(j,5) = round(stats(i).Centroid(1) +
162          8*cos(stats(i).Orientation));
163      M1(round(stats(i).Centroid(2) +
164          8*sin(stats(i).Orientation)),
165          round(stats(i).Centroid(1) +
166          8*cos(stats(i).Orientation))) = 0;
167  else
168      OBJ&PROP(j,4) = round(stats(i).Centroid(2) -
169          8*sin(stats(i).Orientation));
170      OBJ&PROP(j,5) = round(stats(i).Centroid(1) +
171          8*cos(stats(i).Orientation));
172      M1(round(stats(i).Centroid(2) -
173          8*sin(stats(i).Orientation)),
174          round(stats(i).Centroid(1)+8*cos(stats(i).Orientation))) = 0;
175  end
176  end
177
178  i=i+1;
179  j=j+1;
180  end
181
182  i=i+1;
183  end
184
185  if(OBJ&PROP(1,1)==0 & OBJ&PROP(1,2)==0 & OBJ&PROP(1,3)==0 & OBJ&PROP(1,4)==0 & OBJ&PROP(1,5)==0)
186      o=[-1];
187  else
188      o=[OBJ&PROP(:,2), OBJ&PROP(:,3), OBJ&PROP(:,4), OBJ&PROP(:,5)];
189  end
190  end
191  end
192
193  "If no keys have been detected, '-1' is outputted, otherwise OBJ&PROP"
194  "is outputted"
function [o] = opt_threshold_detection(vid)
% This local function detects the optimum BW threshold.
% Input: video object
% Output: optimum BW threshold

opt_thr = 0.8; % Optimum BW threshold
opt_thr.#obj = 0; % Number of objects that match key shape
% by using opt_thr

current.#obj = 0; % Number of objects that match key shape
% by using a lower threshold than 'opt_thr'

l=l; % Iteration variable

while(l ≤ 6)

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

M1 = im2bw(M1,0.8 - 0.1*l); % Converts the coulored image
% into a black and white
% image by using '0.8-0.1*l'
% 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.
[L n] = bwlabel(image_processing(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

while(i ≤ n)
% The next 'if' statement detects which object matches
% 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)

current.#obj = current.#obj + 1;
end

i = i + 1;
end

% If the number of objects that matches key shape by using 'opt_thr-0.1'
% as BW threshold is greater than with 'opt_thr', 'opt_thr' becomes
% 'opt_thr-0.1'
if(current.#obj > opt_thr.#obj)
    opt_thr.#obj = current.#obj;
    opt_thr = opt_thr - 0.1;
end

current.#obj = 0;
end

o = opt_thr;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [o] = image_processing(M1)

% This local function processes an input image in order
% to reduce noise, clear white border and make clearer
% the shapes of possible keys.
% 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(M1, 4); % Suppresses light structures
M1= bwareaopen(M1,20); % Removes small objects from
% binary image
Listing A.4: Script keys_pick_and_place.

% Starting from a random placement of nine keys, the robot has to
% pick them up and place them down one by one, with the major
% axes aligned in the same manner. The robot has to perform
% this operation autonomously by using a digital camera and a
% vacuum gripping system.
right();     % Selects the right arm mode
speed(50);   % Sets the speed [50% of the maximum speed]
start_cam(); % Opens the video input device and configures
             % image acquisition
A = [-340.9334 135.6999];   % First keys placement position
j = 0;                    % Keys placement position iteration variable
if after four vibration actions, and image acquisition
% and processing, no keys have been detected, the software
% detects that the operation has been completed and stops the
% cycle using a break command.
while(j < 4)
    % The next statement defines the image acquisition position
    image_acq_pos = [125.0769, 184.8158, 6.0472, 119.4322];
    moved(image_acq_pos); % Moves to image_acq_pos
    l = 0;                 % Iteration variable
    % The next cycle stops after four iterations with no detected
    % keys or when at least one key is detected
    while(m == 1 & l < 4)
        out(938,1);   % Switches on the DC motor in order to vibrate
                        % the surface where the keys are randomly
                        % placed
pause(0.8);    % Pause of 0.8 sec during which the DC motor
              % is working
out(938,0);    % Switches on the DC motor in order to block
              % vibration
M = keys_detection();    % The function keys_detection is
                        % called in order to acquiring an
                        % image and detect possible keys. The
                        % x-y coordinates and orientation of
                        % each key are stored in M.

[n,m] = size(M);    % Returns size of M dimensions
l = l + 1;    % Iteration variable updating
end
if(l == 4)
    break;
end
i = 1;    % Iteration variable
while(i ≤ n)
    pos = cent_position([p(1,4),p(1,3)]);
    moved(pos(1,1),pos(1,2),20,0);    % Moves above the new key
                                        % centroid position.
    out(937,1);    % Switches on the vacuum gripping system
    smove(3,87.7);    % Lowers the Z-axis in order to grab
                      % the key
    srmove(3,-30);    % Lifts the Z-axis (the key is attached to
                      % the (vacuum cup)
    % The next block of code calculates the rotation angle needed
% to align the key major axis along the specified direction.
% Remind: - M(1,1) is the 'Orientation'
% - M(1,2) is 1 if the key bow points to the top of the
% - image, is 2 if it points to the bottom of the image

if(M(i,1) ≥ 0)
    if(M(i,2) == 1)
        s = M(i,1);
    else
        s = M(i,1) + 180;
    end
else
    if(M(i,2) == 1)
        s = M(i,1);
    else
        s = M(i,1) + 180;
    end
end

if(j == 3) % Updates the Y coordinate of the key placement
   % position
   A(1,2) = A(1,2) + 30;
   j = 0;
end

moved(A(1,1) - j*70,A(1,2),20,s); % Moves above the new
% placement position after
% updating X coordinate
% of the key position

smove(3,67.8137); % Lowers the Z-axis

out(937,0); % Switches off the vacuum gripping system in order
% to release the key

pause(0.2); % Pause of 0.2 sec in order to allow key
% release

srmove(3,-20); % Lifts Z-axis

i = i + 1; % Iteration variable updating

j = j + 1; % Updating of keys placement position iteration
% variable

end
Listing A.5: Function discs&colours_detection.

```matlab
function [ o ] = discs&colours_detection( )
% This main function detects discs in an image
% Output: - '-1' if no discs are detected;
% - Nx3 matrix if N discs are detected, the structure of the
% ith row is: ['Centroid' horizontal (x) coord, 'Centroid'
% vertical (y) coord, disc colour]

vid=cam('retrieve');

opt.thr = opt.threshold_detection(vid)

trigger(vid); % Acquires an image
M = getdata(vid); % Stores the image in the matrix M

% The next three statements split the colour image M to its 3 RGB
% (red, green, blue) channels
M1 = M(:,:,1);
M2 = M(:,:,2);
M3 = M(:,:,3);

% The next three statements perform a colour to BW conversion
% by using by using opt.thr as threshold
M1=im2bw(M1,opt.thr);
M2=im2bw(M2,opt.thr);
M3=im2bw(M3,opt.thr);

% The next statement performs a OR operation between three images
M=M1 | M2 | M3;

M = image-processing(M) % Call to 'image-processing' local
% function that returns a processed
% image

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

% 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','Perimeter','MajorAxisLength',
                    'MinorAxisLength');
```
i = 1; % Iteration variable
j = 1; % Position pointer in 'OBJ&PROP'

% In the next cycle, if an object matches disc shape, it is stored in
% 'OBJ&PROP' with its orientation and coordinates of its centroid
OBJ&PROP=[0,0,0,0];

while(i <= n)
    if(stats(i).Area > 1950 & stats(i).Area < 2200 &
        stats(i).Perimeter > 430 & stats(i).Perimeter < 470 &
        stats(i).MajorAxisLength > 51 & stats(i).MajorAxisLength < 56 &
        stats(i).MajorAxisLength > 49 & stats(i).MajorAxisLength < 54)
        OBJ&PROP(j,1)=i;
        OBJ&PROP(j,2)=stats(i).Centroid(1);
        OBJ&PROP(j,3)=stats(i).Centroid(2);

        % The local function 'colour_detection' is called
        % in order perform a double colour detection.
        % Only if the two colour detections outputs a colour (1 is for
        % white, 2 is for red, 3 is for blue, and 4 is for green) and the
        % colour is the same, the colour information is stored in
        % OBJ&PROP(j,4) and the position pointer 'j' is updated.
        % Otherwise, if the functions output two different colours
        % or 5 (that means a problem with the colour detection has
        % occured): the position pointer 'j' is not updated i.e.
        % the disc is not detected
        first_clour_detection = colour_detection(12);
        second_clour_detection = colour_detection(-12);
        if(first_clour_detection == second_clour_detection &&
            first_clour_detection ≠ 5)
            OBJ&PROP(j,4) = first_clour_detection;
            j=j+1;
        end
    end

    i=i+1;
end

if(OBJ&PROP(1,1)==0 & OBJ&PROP(1,2)==0 & OBJ&PROP(1,3)==0 OBJ&PROP(1,4)==0)
o=[-1];
else
    o=[OBJ&PROP(:,2), OBJ&PROP(:,3), OBJ&PROP(:,4)];
end
function [o] = opt_threshold_detection(vid)
% This local function detects the optimum BW threshold.
% Input: video object
% Output: optimum BW threshold

opt_thr = 0.8; % Optimum BW threshold

opt_thr_obj = 0; % Number of objects that match disc shape by using optimum opt_thr

current_obj = 0; % Number of objects that match disc shape by using a lower threshold than 'opt_thr'

l=1; % Iteration variable

while(l <= 6)
    trigger(vid); % Acquires an image
    M = getdata(vid); % Stores the image in the matrix M

    M1 = M(:,:,1);
    M2 = M(:,:,2);
    M3 = M(:,:,3);

    M1=im2bw(M1,0.8 - 0.1*l);
    M2=im2bw(M2,0.8 - 0.1*l);
    M3=im2bw(M3,0.8 - 0.1*l);

    M=M1 | M2 | M3;

    M = image_processing(M) % Call to 'image_processing' local % function that returns a processed % image
The next statements return a matrix $L$, of the same size as $M$, containing labels for the connected objects.

$$[L \ n] = \text{bwlabel}(\text{image\_processing}(M));$$

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

$$\text{stats} = \text{regionprops}(L, \text{'Area'}, \text{'Centroid'}, \text{'Perimeter'}, \text{'EquivDiameter'});$$

$i = 1; \quad \% \text{Iteration variable}$

while($i \leq n$)

\% The next 'if' statement detects which object matches disc shape

\% $\text{stats}(i).\text{Area}$ > 1950 \& $\text{stats}(i).\text{Area}$ < 2200 \&
\% $\text{stats}(i).\text{Perimeter}$ > 430 \& $\text{stats}(i).\text{Perimeter}$ < 470
\% $\text{stats}(i).\text{MajorAxisLength}$ > 51 \& $\text{stats}(i).\text{MajorAxisLength}$ < 56
\% $\text{stats}(i).\text{MajorAxisLength}$ > 49 \& $\text{stats}(i).\text{MajorAxisLength}$ < 54)

current$\_#\_\text{obj} = \text{current$\_#\_\text{obj}} + 1;$

$i = i + 1;$

end

\% If number of objects that matches disc shape by using 'opt\_thr-0.1'
\% as BW threshold is greater than with 'opt\_thr', 'opt\_thr' becomes
\% 'opt\_thr-0.1'

\% $\text{current$\_#\_\text{obj}} > \text{opt\_thr}$

\% $\text{current$\_#\_\text{obj}} = \text{opt\_thr}$

$\text{current$\_#\_\text{obj}} = 0;$

end

$\text{current$\_#\_\text{obj}} = \text{opt\_thr};$

end

function $[o] = \text{image\_processing}(M)$

\% 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

% 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
if(M1(round(stats(i).Centroid(2)) - t,
    round(stats(i).Centroid(1)) - t) > 200
   & M2(round(stats(i).Centroid(2)) - t,
    round(stats(i).Centroid(1)) - t) < 150
   & M3(round(stats(i).Centroid(2)) - t,
    round(stats(i).Centroid(1)) - t) < 150)
o = 2;
else
if(M1(round(stats(i).Centroid(2)) - t,
    round(stats(i).Centroid(1)) - t) < 120
   & M2(round(stats(i).Centroid(2)) - t,
    round(stats(i).Centroid(1)) - t) < 120
   & M3(round(stats(i).Centroid(2)) - t,
    round(stats(i).Centroid(1)) - t) > 150)
o = 3;
else
if(M1(round(stats(i).Centroid(2)) - t,
    round(stats(i).Centroid(1)) - t) < 130
   & M2(round(stats(i).Centroid(2)) - t,
    round(stats(i).Centroid(1)) - t) > 150
   & M3(round(stats(i).Centroid(2)) - t,
    round(stats(i).Centroid(1)) - t) < 130)
o=4;
else
o=5;
end
end
end
end
end

Listing A.6: Script coloured_discs_pick_and_place.

% 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 4 small containers depending on the colour. The colours are: white, red, green, blue.
right(); % Selects the right arm mode
speed(50); % Sets the speed [50% of the maximum speed]
start_cam(); % Opens the video input device and configures image acquisition
while(1≠0)
    l = 0; % Iteration variable
    m = 1; % Variables for storing the number of rows of
            % matrix M
    image_acq.pos = [125.0769, 184.8158, 6.0472, 119.4322];
    moved(image_acq.pos); % Moves to image_acq_pos
    while(m == 1 & l < 4)
        out(938,1); % Switches on the DC motor in order to vibrate
                    % the surface where the discs are randomly
                    % placed
        pause(0.8); % Pause of 0.8 sec during which the DC motor
                    % is working
        out(938,0); % Switches on the DC motor in order to block
                    % vibration
        % The function discs_detection is called. It
        % acquires an image and detects possible discs.
        % The x-y orientation of each disc are stored in M.
        M = discs&colours_detection();
        [n,m] = size(M); % Returns size of M dimensions
        l = l + 1; % Iteration variable updating
    end
    if(l == 4)
        break;
    end
    i=1; % Iteration variable
    while(i ≤ n)
        % The next statement retrieves row i of matrix M:
        % 'Centroid' horizontal (x) coord, 'Centroid'
        % vertical (y) coord, colour]
        p=M(i,:);
        % The function cent_position is called in order to
        % detect the centroid in the workspace.
% The input parameter are the 'Centroid'
% coordinates
pos = position([p(1,2), p(1,3)]);

moved(pos(1,1), pos(1,2), 20, 0); % Moves above the new % disc centroid % position.

out(937,1); % Switches on the vacuum gripping system

smove(3, 84.8); % Lowers the Z-axis in order to grab % the disc

smove(3, 30); % Lifts the Z-axis (the disc is attached to % the (vacuum cup)

% The next 'switch' construction moves the robot over % the placement position depending on the disc colour, % and releases the disc

white_release_pos = [-374.92, 129.82, 32.00, 111.99];
red_release_pos = [-384.99, 214.99, 31.99, 112.00];
blue_release_pos = [-384.99, 214.99, 31.99, 111.99];
green_release_pos = [-399.99, 375.00, 31.99, 111.99];

switch(M(i, 4))
    case(1)
        moved(white_release_pos);
        out(937, 0); % Switches off the vacuum gripping system % in order to release the disc
        pause(0.2); % Pause of 0.2 sec in order to allow % disc release
    case(2)
        moved(red_release_pos);
        out(937, 0);
        pause(0.2);
    case(3)
        moved(blue_release_pos);
        out(937, 0);
        pause(0.2);
    case(4)
        moved(green_release_pos);
        out(937, 0);
        pause(0.2);
end

end
i = i + 1;
end
Listing A.7: Interpreter (SSL/E program).

```plaintext
// 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

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);
```
ELSE{
    JMARK(P1);
    RSOUT(1, POSGET(P1, 1));
    RSOUT(1, POSGET(P1, 2));
    RSOUT(1, POSGET(P1, 3));
    RSOUT(1, POSGET(P1, 4));
    I=STATM(1);
}
}

// Interpreter

PROG INTERPRETER()

M=1;   // For enabling PROG SAMPLE()

RSCLOSE(1); // Closes the serial port in order to delete any previous
data stored in the port buffer avoiding the
port buffer saturation */

/* The next statement opens the serial port COM1 setting 115200 bps
as baud rate and 512 bytes as BUFFER length. Thus, the settings
of the port are "9600 B8 PE S2 L128 CRLF"

RSOPEN(1,"115200 L512"); /*

/* The next statement sends the Feedback Execution Error Code 7777 through
the serial cable. Since Matlab is waiting for a feedback from the interpreter,
this is a necessary feedback code sent when the interpreter is restarted
after an error occurrence. It alerts Matlab about the occurred error */

RSOUT(1,7777);

/* The software starts a polling operation of the serial port,
waiting for the Matlab Communication Initialization Code 0000
(Matlab sends 0000 in order to establish the communication) */
RSIN(1,K);

WHILE(K!=0000){
    RSIN(1,K);
}

START:

/* The next block of code is for sending a Feedback Execution Confirmation Code.
Depending on the function, the Code can be: 1, 0, 8888, or four numbers*/
IF((1000 ≤ MAT_FUN_CODE) && (MAT_FUN_CODE ≤ 1020) && VIS==0)
RSOUT(1,1);

IF((2000 ≤ MAT_FUN_CODE) && (MAT_FUN_CODE ≤ 3020)){
  IF(MAT_FUN_CODE==2003 || MAT_FUN_CODE==2004 || MAT_FUN_CODE==3014){
    RSOUT(1,POSGET(P,1));
    RSOUT(1,POSGET(P,2));
    RSOUT(1,POSGET(P,3));
    RSOUT(1,POSGET(P,4));
  }

  IF(MAT_FUN_CODE==2018 || MAT_FUN_CODE==2019){
    IF(RET==1){
      RSOUT(1,POSGET(P,1));
      RSOUT(1,POSGET(P,2));
      RSOUT(1,POSGET(P,3));
      RSOUT(1,POSGET(P,4));
    }
    ELSE
      RSOUT(1,8888);
  }

  IF(MAT_FUN_CODE!=2003 && MAT_FUN_CODE!=2004 && MAT_FUN_CODE!=2018
    && MAT_FUN_CODE!=2019 && MAT_FUN_CODE!=3014)
    RSOUT(1,1);
}

IF(MAT_FUN_CODE==4000 || MAT_FUN_CODE==4003 || MAT_FUN_CODE==4004)
RSOUT(1,1);

IF(MAT_FUN_CODE==4001 || MAT_FUN_CODE==4002 && VIS==0)
RSOUT(1,1);

/* The next block of code is for polling the serial port waiting the
  New Matlab Function Communication Code '1111' */
RSIN(1,COMIN);
WHILE(COMIN!=1111)
  RSIN(1,COMIN);
RSIN(1,MAT_FUN_CODE); /* Reads the Matlab Function Code

/* Matlab Robot Functions with identification code between 1000 and 1007:
archmove, cmove, jmove, jmoved, lmove, lmoved, move, moved, rjmove,
rlmove, rmove, sjmove, slmove, smove, srjmove, srlmove, srmove,
xycir, xzcir, yzcir */

IF((1000≤MAT_FUN_CODE) && (MAT_FUN_CODE≤1020)){
    RSIN(1,PAR); // reads the # of parameters of the statement from the port
    I=1;
    WHILE(I≤PAR){
        RSIN(1,X); // Reads a parameter from the port
        RSIN(1,Y);
        RSIN(1,Z);
        RSIN(1,S);
        POS[I]=X,Y,Z,S; // Parameters storing operation
        I=I+1;
    }
    /* Functions execution */
    IF(MAT_FUN_CODE==1000){
        IF(VIS==0){
            IF(PAR==1){
                MOVE(POS[1]);
                CYCLE START;
            }
            IF(PAR==2){
                MOVE(POS[1],POS[2]);
                CYCLE START;
            }
            IF(PAR==3){
                MOVE(POS[1],POS[2],POS[3]);
                CYCLE START;
            }
            IF(PAR==4){
                MOVE(POS[1],POS[2],POS[3],POS[4]);
                CYCLE START;
            }
            IF(PAR==5){
                MOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
                CYCLE START;
            }
            IF(PAR==6){
                MOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6]);
                CYCLE START;
            }
            IF(PAR==7){
                }
APPENDIX A. CODE

MOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6], POS[7]);
CYCLE START;

IF(PAR==8){
    MOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6], POS[7],POS[8]);
    CYCLE START;
}

ELSE {
    IF(PAR==1){
        QMOVE(POS[1]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }
    IF(PAR==2){
        QMOVE(POS[1],POS[2]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }
    IF(PAR==3){
        QMOVE(POS[1],POS[2],POS[3]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }
    IF(PAR==4){
        QMOVE(POS[1],POS[2],POS[3],POS[4]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }
    IF(PAR==5){
        QMOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }
    IF(PAR==6){
        QMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }
    IF(PAR==7){
        QMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }
}
IF(PAR==8) {
    QMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7],POS[8]);
    SAMPLE();
    RSOUT(1,7777);
    CYCLE START;
}

IF(MAT_FUN_CODE==1001) {
   IF(VIS==0) {
   IF(PAR==1) {
      LMOVE(POS[1]);
      CYCLE START;
   }
   }
   IF(PAR==2) {
      LMOVE(POS[1],POS[2]);
      CYCLE START;
   }
   }
   IF(PAR==3) {
      LMOVE(POS[1],POS[2],POS[3]);
      CYCLE START;
   }
   }
   IF(PAR==4) {
      LMOVE(POS[1],POS[2],POS[3],POS[4]);
      CYCLE START;
   }
   }
   IF(PAR==5) {
      LMOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
      CYCLE START;
   }
   }
   IF(PAR==6) {
      LMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6]);
      CYCLE START;
   }
   }
   IF(PAR==7) {
      LMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7]);
      CYCLE START;
   }
   }
   IF(PAR==8) {
      LMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7],POS[8]);
      CYCLE START;
   }
   }
APPENDIX A. CODE

}  

ELSE {

    IF(PAR==1) {
        QLMOVE(POS[1]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }

    IF(PAR==2) {
        QLMOVE(POS[1],POS[2]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }

    IF(PAR==3) {
        QLMOVE(POS[1],POS[2],POS[3]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }

    IF(PAR==4) {
        QLMOVE(POS[1],POS[2],POS[3],POS[4]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }

    IF(PAR==5) {
        QLMOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }

    IF(PAR==6) {
        QLMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }

    IF(PAR==7) {
        QLMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }

    IF(PAR==8) {
        QLMOVE(POS[1],POS[2],POS[3],POS[4],POS[5],POS[6],POS[7],POS[8]);
        SAMPLE();
        RSOUT(1,7777);
        CYCLE START;
    }
IF(MAT_FUN_CODE==1002) {
}

IF(VIS==0) {

IF(PAR==1) {
    JMOVE(POS[1]);
    CYCLE START;
}

IF(PAR==2) {
    JMOVE(POS[1],POS[2]);
    CYCLE START;
}

IF(PAR==3) {
    JMOVE(POS[1],POS[2],POS[3]);
    CYCLE START;
}

IF(PAR==4) {
    JMOVE(POS[1],POS[2],POS[3],POS[4]);
    CYCLE START;
}

IF(PAR==5) {
    JMOVE(POS[1],POS[2],POS[3],POS[4],POS[5]);
    CYCLE START;
}

IF(PAR==6) {
    JMOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6]);
    CYCLE START;
}

IF(PAR==7) {
    JMOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6], POS[7]);
    CYCLE START;
}

IF(PAR==8) {
    JMOVE(POS[1],POS[2],POS[3],POS[4],POS[5], POS[6], POS[7],POS[8]);
    CYCLE START;
}

} }
APPENDIX A. CODE

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

ELSE{
QMOVE(POS[1],POS[2]);
SAMPLE();
CIRCULAR(0);
RSOUT(1,7777);
CYCLE START;
}

IF(MAT_FUN_CODE==1004){
VAR=POSGET(POS[4],1);
CIRCULAR(VAR);
IF(VIS==0){
MOVE(POS[1],POS[2],POS[3]);
CIRCULAR(0);
CYCLE START;
}
QMOVE(POS[1],POS[2],POS[3]);
SAMPLE();
CIRCULAR(0);
RSOUT(1,7777);
CYCLE START;
}

ELSE{
QMOVE(POS[1],POS[2],POS[3]);
SAMPLE();
CIRCULAR(0);
RSOUT(1,7777);
CYCLE START;
}

IF(MAT_FUN_CODE==1005){
VAR=POSGET(POS[3],1);
CIRCULAR(VAR,1);
IF(VIS==0){
MOVE(POS[1],POS[2],POS[3]);
CIRCULAR(0);
CYCLE START;
}
ELSE{
QMOVE(POS[1],POS[2]);
SAMPLE();
CIRCULAR(0);
RSOUT(1,7777);
CYCLE START;
}

ELSE{
QMOVE(POS[1],POS[2],POS[3]);
SAMPLE();
CIRCULAR(0);
RSOUT(1,7777);
CYCLE START;
}

ELSE{
QMOVE(POS[1],POS[2],POS[3]);
SAMPLE();
CIRCULAR(0); 
RSOUT(1,7777); 
CYCLE START; 

/* Matlab Robot Functions with identification code between 2000 and 2030: 
acct, autoacl, cpacct, cpdacct, cpspeed, dacct, in, jmark, 
left, mark, opeclr, opeout, right, sample, 
speed, weight, wintime*/ 

IF((2000 ≤ MAT_FUN_CODE) && (MAT_FUN_CODE ≤ 2030))

IF(MAT_FUN_CODE!=2022) // reads the parameter from the port 
RSIN(1,PAR); 
ELSE 
RSIN(1,S1); 

/* MAT_FUN_CODEtions execution */ 

IF(MAT_FUN_CODE==2000)

IF(MAT_FUN_CODE==2002)

IF(MAT_FUN_CODE==2003)

IF(MAT_FUN_CODE==2004)

IF(MAT_FUN_CODE==2005)
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=ACUTOACL(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;
}

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);
  }
}
I=1;
WHILE(I\leq PAR) {
    RSIN(1, ARRAY_2[I]);
    I=I+1;
}

IF(MAT_FUN_CODE==3006) {
    RET=OUT(ARRAY_1[1], ARRAY_1[2]);
    CYCLE START;
}

IF(MAT_FUN_CODE==3007) {
    WIN_TIME(ARRAY_1[1]);
    RET=TRI(ARRAY_1[2], ARRAY_1[3]);
    CYCLE START;
}

IF(MAT_FUN_CODE==3008) {
    WIN_TIME(ARRAY_1[1]);
    RET=WIN(ARRAY_1[2], ARRAY_1[3]);
    CYCLE START;
}

IF(MAT_FUN_CODE==3009) {
    IF(PAR==2) {
        RET=BLINK(ARRAY_1[1], ARRAY_1[2]);
        CYCLE START;
    }
    IF(PAR==3) {
        RET=BLINK(ARRAY_1[1], ARRAY_1[2], ARRAY_1[3]);
        CYCLE START;
    }
}

IF(MAT_FUN_CODE==3010) {
    IF(PAR==1) {
        RET=BLINKEND(ARRAY_1[1]);
        CYCLE START;
    }
    IF(PAR==2) {
        RET=BLINKEND(ARRAY_1[1], ARRAY_1[2]);
        CYCLE START;
    }
    IF(PAR==3) {
        RET=BLINKEND(ARRAY_1[1], ARRAY_1[2], ARRAY_1[3]);
        CYCLE START;
    }
    IF(PAR==4) {
        RET=BLINKEND(ARRAY_1[1], ARRAY_1[2], ARRAY_1[3], ARRAY_1[4]);
        CYCLE START;
    }
}

}
IF (MAT_FUN_CODE==3013) {
    RET=LOCATE(ARRAY_1[1],ARRAY_1[2]);
    CYCLE START;
}

IF (MAT_FUN_CODE==3014) {
    PLTNS(ARRAY_2[1],ARRAY_2[2],P);
    RSOUT(1,POSGET(P,1));
    RSOUT(1,POSGET(P,2));
    RSOUT(1,POSGET(P,3));
    RSOUT(1,POSGET(P,4));
    CYCLE START;
}

/* Matlab Robot Functions with identification code 4000: setplt */

IF (MAT_FUN_CODE==4000) {
    RSIN(1,PAR);
    I=1;
    WHILE(I\leq4) {
        RSIN(1,X);
        RSIN(1,Y);
        RSIN(1,Z);
        RSIN(1,S);
        POS[I]=X,Y,Z,S;
        I=I+1;
    }
    RSIN(1,ARRAY_2[1]);
    RSIN(1,ARRAY_2[2]);
    RSIN(1,ARRAY_2[3]);
    RSOUT(1,1);
}

/* Function execution */

SETPLTNS(PAR,POS[1],POS[2],POS[3],POS[4],ARRAY_2[1],ARRAY_2[2],ARRAY_2[3]);
CYCLE START;

RSCLOSE(1);
END
Bibliography


