SWINBURNE UNIVERSITY OF TECHNOLOGY SARAWAK CAMPUS

RME 40003 ROBOT SYSTEM DESIGN

Hand-Eye Coordination Project
Semester 1, 2021

Report

Group Members

Chuah Ding Ken
Su Bang Xiang
Liew Sheng Kai
Koo Suit May

Table of Contents

Lists of Tables and Figures 3
Introduction 5
Objectives 5
Job Distribution 6
Project Plan and Milestone 6
Image Processing 7
Deriving Inverse Kinematics Solutions 11
    Theoretical Derivation 11
    Deriving Forward and Inverse Kinematics using MATLAB 15
    Simulation of Robot Motion using Robotics Toolbox in MATLAB 17
Creating 3D Model for the SCARA Robot in MATLAB 23
    STL File 23
    Opening STL file using SolidWorks 24
Robot Programming 28
    Adding smart bar 28
    Setting Final Location 30
        Coding 31
    Moving object 31
    Initial Location 33
Conclusion 35
Appendix 36
    Appendix A: Source Code for Forward and Inverse Kinematics Solutions 36
    Appendix B: Source Code for Simulation of Robot Motion with Data from Inverse Kinematics and Image Processing 37
    Appendix C: Source Code for the Image Processing Algorithm 44
    Appendix D: Source Code for Sending Coordinates Location Data from Image Processing Program to Robot Studio RAPID Program 45
    Appendix E: Source Code for the RAPID Program from ABB Robot Studio 47
    Appendix F: Source Code for sending coordination from MATLAB 52
    Appendix G: Dimension of the SCARA Robot model 54

Lists of Tables and Figures

Table 1: Job Distribution among Group Members 6
Table 2: DH-Parameters for each Link 12
Table 3: List of Centroid Coordinates of the coloured-bars (Image2.jpg) 17
Table 4: Joint Angles required for picking the coloured-bars (Image2.jpg) 17
Table 5: Parts name and their respective link name 23

Figure 1: Top View of the Image Captured 7
Figure 2: Code Snippet used for black and white distinguishing 7
Figure 3: Pre-processed Image (Brighter Setting) 7
Figure 4: Pre-processed Image (Improve Contrast) 8
Figure 5: Converted Image (RGB to HSV) 8
Figure 6: Code Snippet (Separation of Red Colour) 8
Figure 7: Colour Wheel Diagram 9
Figure 8: Separation of Different Coloured-Bars (Black and White) 9
Figure 9: From left to right: Green Colour detection, Blue Colour detection, Yellow Colour detection, Black Colour detection, ... 9
Figure 10: Tabulated result of Area, Centroid, and Orientation shown in MATLAB 10
Figure 11: Results shown at the command window (Centroid Coordinates and Orientation angle) ... 10
Figure 12: Robot Viewed in ABB Robot Studio 11
Figure 13: Assignment of Coordinate Frames to each joint 11
Figure 14: Code Snippet for Defining the A matrix function 15
Figure 15: Code Snippet for Defining the D-H Parameters 15
Figure 16: Code Snippet of the Results of the A Matrix obtained 15
Figure 17: Code Snippet for Calculating Orientation Angle 16
Figure 18: Inverse Kinematics Solutions coded in MATLAB 16
Figure 19: Code Snippet used to obtain and display the x and y centroid coordinates of the bars' location 18
Figure 20: Results for the centroid coordinates shown in command window 18
Figure 21: Setting up Parameters required to plot the bar 18
Figure 22: Code Snippet for plotting out all the bars with their respective centroid coordinates as the centre of plot 19
Figure 23: Plotting of bars in MATLAB 19
Figure 24: Inverse Kinematics Solutions for 4 different coloured bars 19
Figure 25: Code Snippet for Robot Motion Simulation in MATLAB 20
Figure 26: Position of Robot and its link 1, 2, and 3 when red-coloured bar is to be picked 20
Figure 27: Position of Robot and its link 1, 2, and 3 when green-coloured bar is to be picked 21
Figure 28: Position of Robot and its link 1, 2, and 3 when blue-coloured bar is to be picked 21
Figure 29: Position of Robot and its link 1, 2, and 3 when -coloured bar is to be picked 22
Figure 30: Selection of SCARA CAD Models 23
Figure 31: Choosing IRB 910 SC 3/0.65 model 23
Figure 32: Link 0 in STL format 24
Figure 33: Link 1 in STL format 24
Figure 34: Link 2 in STL format 24
Figure 35: Link 3 in STL format 25
Figure 36: Link 4 in STL format 25
Figure 37: Original Robot Plot (Stick Figure) 26

Figure 38: Final Robot Plot with actual 3D Model........27
Figure 39: Adding SmartBar........28
Figure 40: setting SmartBar........28
Figure 41: Collision Set........29
Figure 42: Connecting simulation logic........29
Figure 43: final location of bars placed. 30
Figure 44: gap between bars........30
Figure 45: objects location........31
Figure 46: initial location of bars........33
Figure 47: “Image2.jpg” from provided image and coordinations........33
Figure 48: initial location of bars comparing to "image2.jpg" 34
Figure 49: Dimension of the SCARA robot model in the manual........54

Introduction

This project is to understand how to use the MATLAB and robot studio to design and perform a visually guided pick and place operation. There are 4 tasks in this project which are Image Processing, Robot Programming, Inverse Kinematics & Simulation and 3D Modelling. The image processing will be involve using image processing toolbox in MATLAB and reading the sample image posted on Canvas in order to differentiate the colour, shape and objects' region properties. For the robot programming, it involve using MATLAB and RobotStudio to virtually control the robot to pick and place the part into a designated shape. Besides, the inverse kinematics & simulation part involve calculation of the kinematics and develop the mathematical model of the robot in RobotStudio. For the 3D modelling, it involve in applying the 3D model from the STL file into the MATLAB plot which showing the 910SC robot model inside the MATLAB plot by using the Robotics Toolbox.

Objectives

The main objectives of this project are:

- To simulate the robot motion in MATLAB by implementing the inverse kinematics solutions.
- To perform image processing to the image to distinguish bars of different colours.
- To use MATLAB to communicate with Robot Studio to perform pick and place task based on the data obtained from the image processing algorithm.

Job Distribution

The required tasks for this project are distributed evenly among team members. Each of the members has their own responsibilities as shown below:

Table 1: Job Distribution among Group Members

Chuah Ding Ken	**Su Bang Xiang**	**Liew Sheng Kai**	**Koo Suit May**
• Derive forward and kinematics theoretically and then coded into MATLAB. • Simulate the robot motion using Robotics Toolbox	• Extract STL files of each part of the SCARA robot into MATLAB. • Combine the STL part files together and use robot.plot3d to show the SCARA robot model in MATLAB	• Perform image processing to allow MATLAB to distinguish different colours of the bars present in the image. • Extract the centroid location from the identified coloured-bars.	• Write RAPID program to simulate the picking and placing of the bars.

Project Plan and Milestone

Week 8	• Start Inverse Kinematics Derivation (Assignment of Coordinate Frame) • Start Image Processing Task.
Week 9	• Deriving of Forward and Inverse Kinematics theoretically. • Continuing Image Processing Task. • Start Robot Programming. • Searching for STL files from the ABB official website
Week 10	• Finalising Image Processing Task. • Continue Robot Programming. • Code the Inverse Kinematics Solution derived from theoretical calculation into MATLAB
Week 11	• Extracting STL files into MATLAB • Communicating between MATLAB and Robot Studio • Simulate the robot motion in MATLAB • Create a 3D model of the robot plot in MATLAB.
Week 12	• Documentation • Finalising and Recording of the videos of robot simulation

Image Processing

For the following screenshots, the image “Image2.jpg” is used. First the image is imported and read into MATLAB, “Image2.jpg” is shown as below:

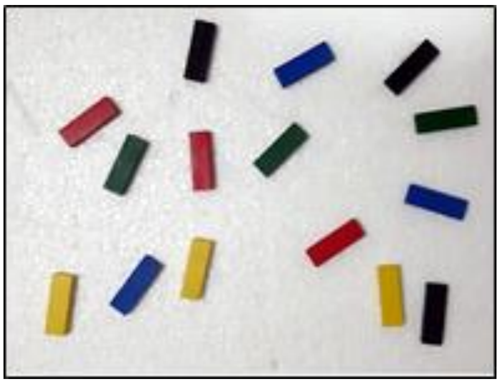
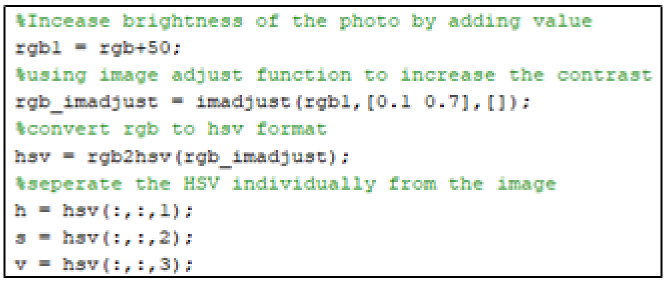


Figure 1: Top View of the Image Captured

To let the program able to separate the colour black and green, some image processing is required. The code used is shown as below:



```
%Incease brightness of the photo by adding value
rgb1 = rgb+50;
%using image adjust function to increase the contrast
rgb_imadjust = imadjust(rgb1,[0.1 0.7],[]);
%convert rgb to hsv format
hsv = rgb2hsv(rgb_imadjust);
%seperate the HSV individually from the image
h = hsv(:,:,1);
s = hsv(:,:,2);
v = hsv(:,:,3);
```

Figure 2: Code Snippet used for black and white distinguishing

After adding a value of 50 to original image, the brightness are increase:

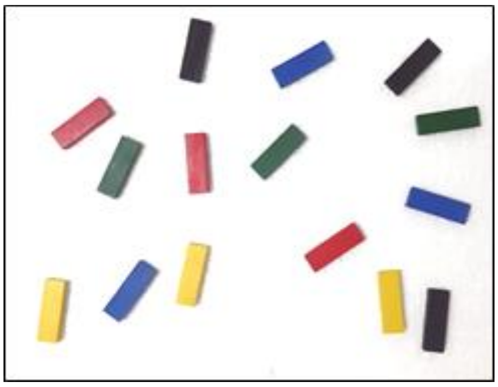


Figure 3: Pre-processed Image (Brighter Setting)

Then, contrast is increased using imagjust function, which make each colour more stand out:

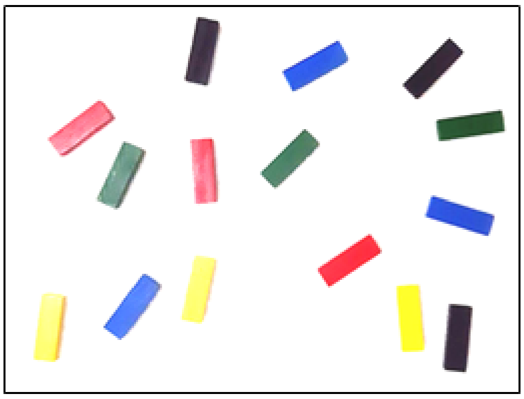


Figure 4: Pre-processed Image (Improve Contrast)

Lastly, to help with the identification of colours for the program, RGB format is converted to HSV format:

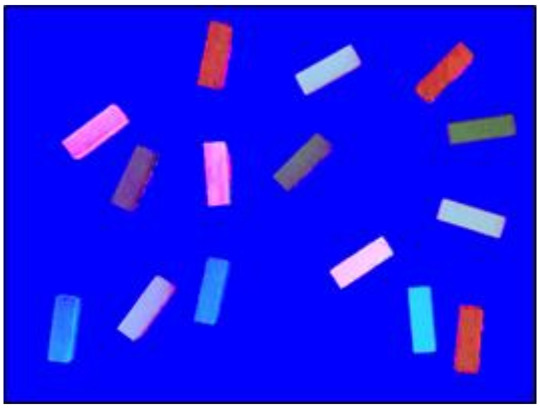
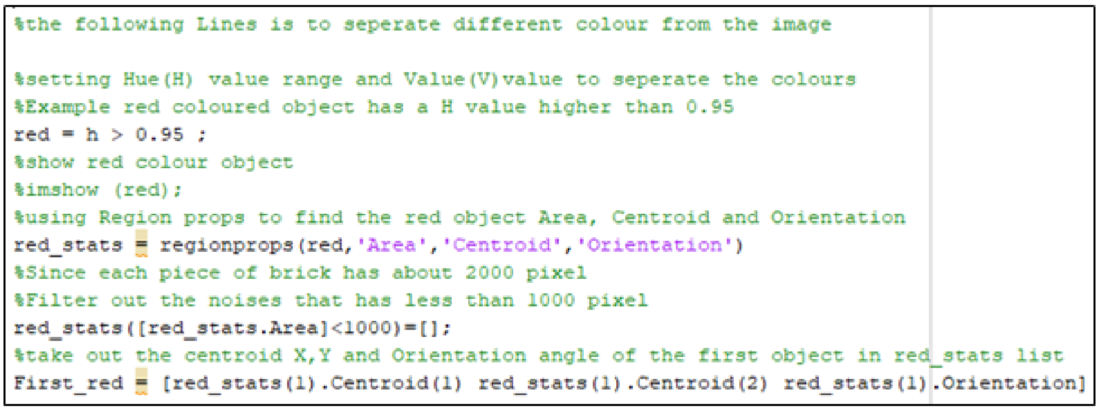


Figure 5: Converted Image (RGB to HSV)

After the images processing, to separate each coloured object(Red, Green, Blue, Green, Black) from the image. The following code is used to separated red coloured object from the image, the other colour also used the same formation of code:



```
%the following Lines is to seperate different colour from the image

%setting Hue(H) value range and Value(V)value to seperate the colours
%Example red coloured object has a H value higher than 0.95
red = h > 0.95 ;
%show red colour object
%imshow (red);
%using Region props to find the red object Area, Centroid and Orientation
red_stats = regionprops(red,'Area','Centroid','Orientation')
%Since each piece of brick has about 2000 pixel
%Filter out the noises that has less than 1000 pixel
red_stats([red_stats.Area]<1000)=[];
%take out the centroid X,Y and Orientation angle of the first object in red_stats list
First_red = [red_stats(1).Centroid(1) red_stats(1).Centroid(2) red_stats(1).Orientation]
```

Figure 6: Code Snippet (Separation of Red Colour)

By specifying the range of Hue(H) value and Value(V) value from HSV, the colour can be singled out. For the case of MATLAB, hue value goes from 0 to 1 which corresponding to the colour position on a colour wheel as shown as below:

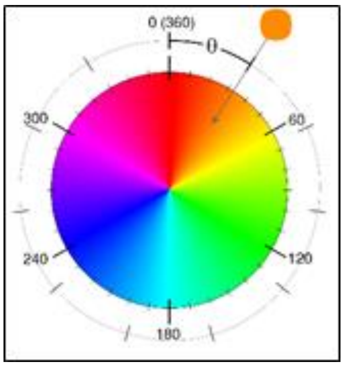


Figure 7: Colour Wheel Diagram

After singled out the red coloured object or pixels, the image outcome is shown as below:

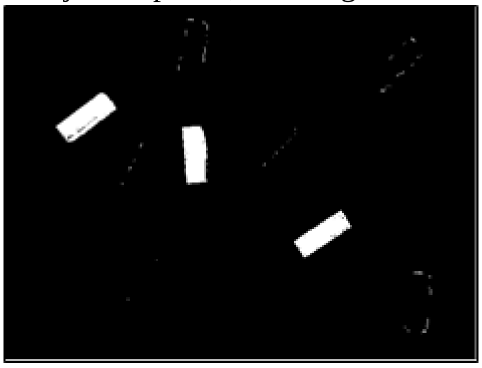


Figure 8: Separation of Different Coloured-Bars (Black and White)

Other four colours are shown below:

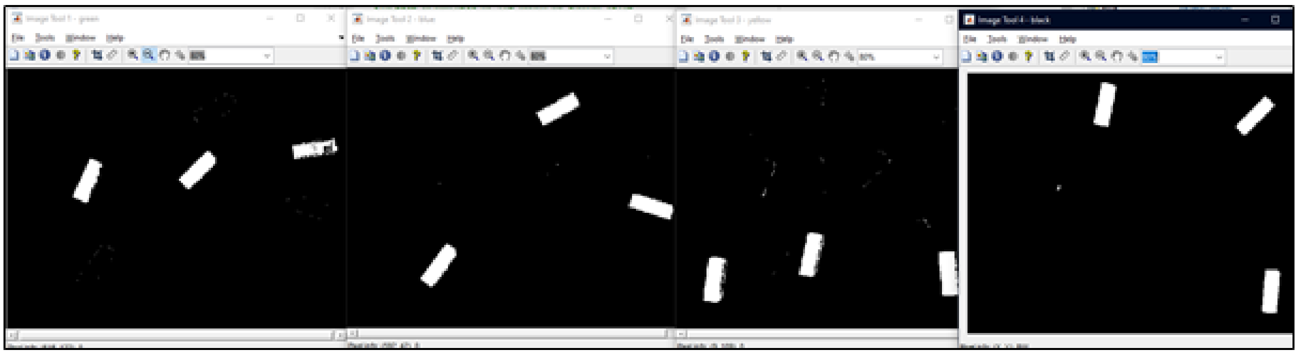
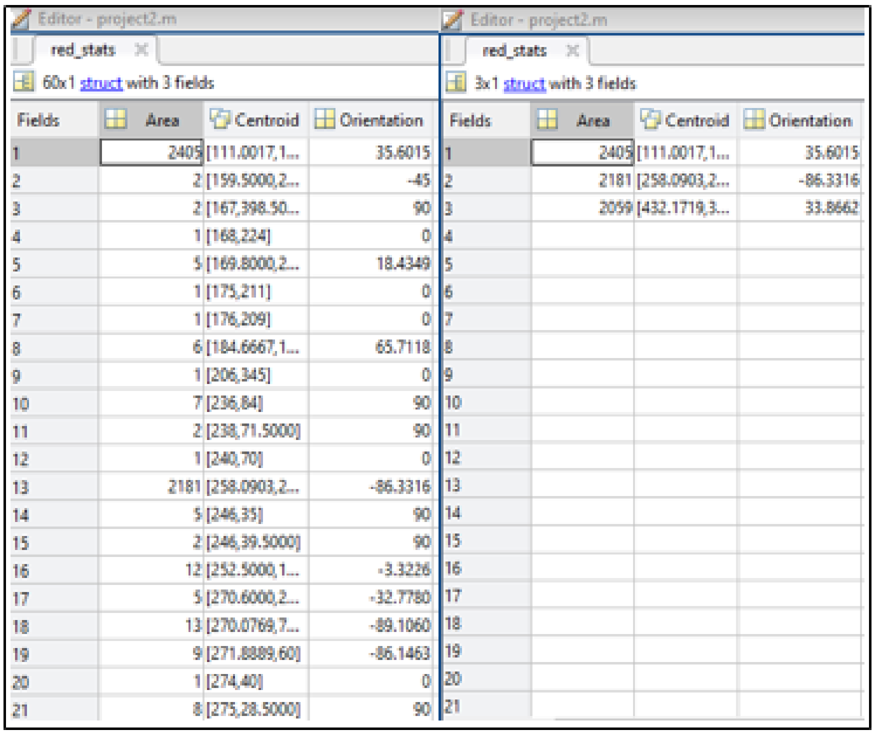


Figure 9: From left to right: Green Colour detection, Blue Colour detection, Yellow Colour detection, Black Colour detection,

Now by using regionprops function, the area, centroid and orientation of the red coloured object can be found out. But There will be some noises and unnecessary object will be added to the list as well. This can be solved by filtering out the noises that has an area under a 1000 pixels which is half of the area of the brick object (around 2000 pixels). The list of object before and after filtering is shown as below.



Editor - project2.m

red_stats

60x1 struct with 3 fields

Fields	Area	Centroid	Orientation
1	2405	[111.0017,1...	35.6015
2	2	[159.5000,2...	-45
3	2	[167,398.50...	90
4	1	[168,224]	0
5	5	[169.8000,2...	18.4349
6	1	[175,211]	0
7	1	[176,209]	0
8	6	[184.6667,1...	65.7118
9	1	[206,345]	0
10	7	[236,84]	90
11	2	[238,71.5000]	90
12	1	[240,70]	0
13	2181	[258.0903,2...	-86.3316
14	5	[246,35]	90
15	2	[246,39.5000]	90
16	12	[252.5000,1...	-3.3226
17	5	[270.6000,2...	-32.7780
18	13	[270.0769,7...	-89.1060
19	9	[271.8889,60]	-86.1463
20	1	[274,40]	0
21	8	[275,28.5000]	90

Editor - project2.m

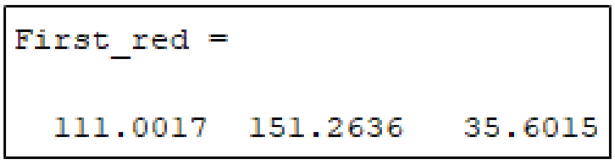
red_stats

3x1 struct with 3 fields

Fields	Area	Centroid	Orientation
1	2405	[111.0017,1...	35.6015
2	2181	[258.0903,2...	-86.3316
3	2059	[432.1719,3...	33.8662
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			

Figure 10: Tabulated result of Area, Centroid, and Orientation shown in MATLAB

Another line is used to extract the data from the first object in object list, the Frist_red array will take the X and Y of the centroid as well as the orientation angle of the object. The other colors identification are done the same way.



```
First_red =

   111.0017   151.2636    35.6015
```

Figure 11: Results shown at the command window (Centroid Coordinates and Orientation angle)

Deriving Inverse Kinematics Solutions

Theoretical Derivation

Figure below shows the robot model:

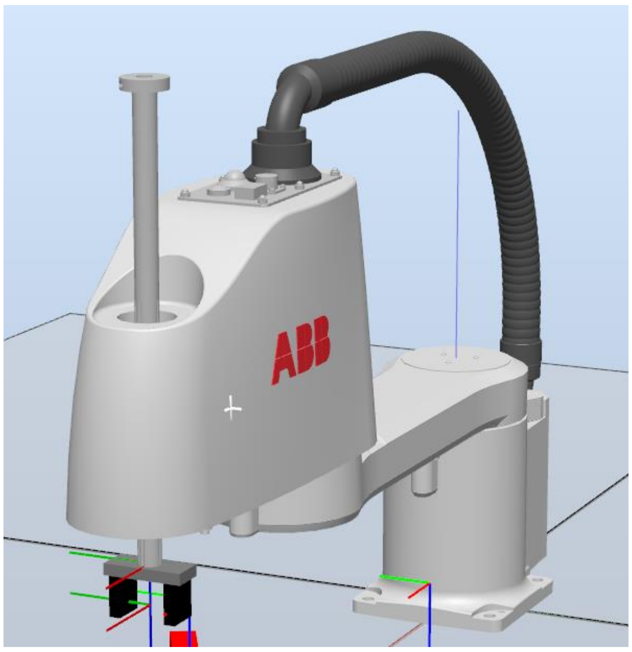


Figure 12: Robot Viewed in ABB Robot Studio

To derive the inverse kinematics of the robot, coordinate frames need to be assigned to each joint first. It will be started from the base of the robot. Figure 13 below shows each link for the robot:

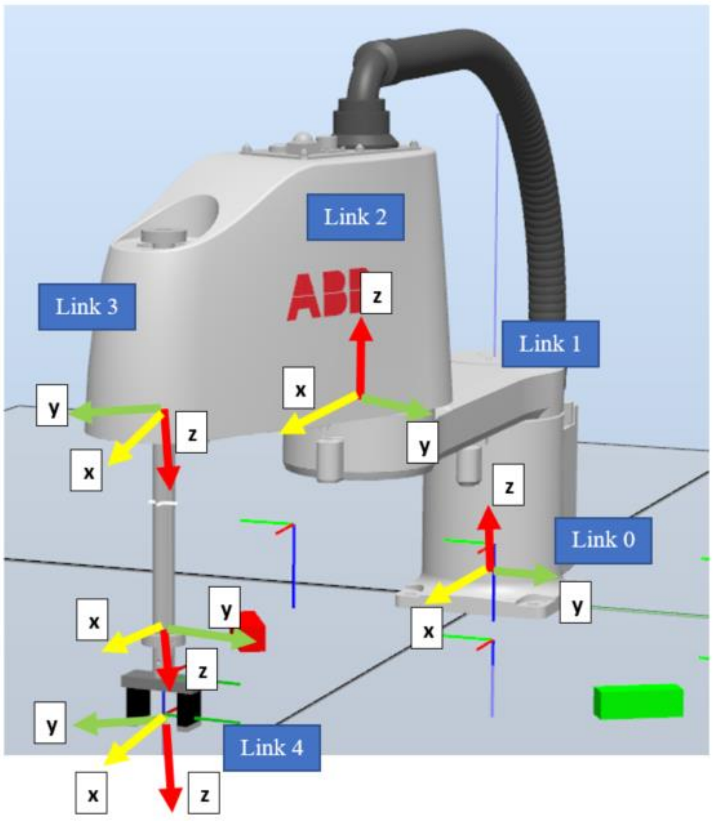


Figure 13: Assignment of Coordinate Frames to each joint

The D-H Parameters for each link is:

Table 2: DH-Parameters for each Link

Link	$\boldsymbol{\theta}$	$\boldsymbol{d}$	$\boldsymbol{l}$	$\boldsymbol{\alpha}$
#1	θ_1	0.1916	0.4	0
#2	θ_2	0	0.25	180
#3	0	d_3	0	0
#4	θ_4	0.05	0	0

The forward transform is the multiplication of 4 individual matrix:
$= A^1A^2A^3A^4$

$$= \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0.4\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0.4\sin\theta_1 \\ 0 & 0 & 1 & 0.1916 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 & 0.25\cos\theta_2 \\ \sin\theta_2 & -\cos\theta_2 & 0 & 0.25\sin\theta_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & 0 \\ \sin\theta_4 & \cos\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0.05 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta_1+\theta_2) & \sin(\theta_1+\theta_2) & 0 & 0.4\cos(\theta_1)+0.25\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2) & -\cos(\theta_1+\theta_2) & 0 & 0.4\sin(\theta_1)+0.25\sin(\theta_1+\theta_2) \\ 0 & 0 & -1 & 0.1916 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & 0 \\ \sin\theta_4 & \cos\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0.05 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta_1+\theta_2) & \sin(\theta_1+\theta_2) & 0 & 0.4\cos(\theta_1)+0.25\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2) & -\cos(\theta_1+\theta_2) & 0 & 0.4\sin(\theta_1)+0.25\sin(\theta_1+\theta_2) \\ 0 & 0 & -1 & -d+0.1916 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & 0 \\ \sin\theta_4 & \cos\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0.05 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\theta_1+\theta_2-\theta_4) & \sin(\theta_1+\theta_2-\theta_4) & 0 & 0.4\cos(\theta_1)+0.25\cos(\theta_1+\theta_2) \\ \sin(\theta_1+\theta_2-\theta_4) & -\cos(\theta_1+\theta_2-\theta_4) & 0 & 0.4\sin(\theta_1)+0.25\sin(\theta_1+\theta_2) \\ 0 & 0 & -1 & -d+0.1416 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From the forward kinematics solutions:
$x = 0.4\cos(\theta_1) + 0.25\cos(\theta_1+\theta_2)$ ... (1)
$y = 0.4\sin(\theta_1) + 0.25\sin(\theta_1+\theta_2)$ ... (2)
$z = -d + 0.1416 \rightarrow d = 0.1416 - z$ ... (3)

Squaring both equations and sum it up:

$$x^2 + y^2 = (0.16\cos^2\theta_1 + 0.2\cos\theta_1\cos(\theta_1+\theta_2) + 0.125\cos^2(\theta_1+\theta_2)) + (0.16\sin^2\theta_1 + 0.2\sin\theta_1\sin(\theta_1+\theta_2) + 0.125\sin^2(\theta_1+\theta_2))$$

$$= 0.16(\cos^2\theta_1 + \sin^2\theta_1) + 0.2(\cos\theta_1\cos(\theta_1+\theta_2) + \sin\theta_1\sin(\theta_1+\theta_2)) + 0.125(\cos^2(\theta_1+\theta_2) + \sin^2(\theta_1+\theta_2))$$

$$x^2 + y^2 = 0.16 + 0.125 + 0.2\cos(\theta_2)$$
$$x^2 + y^2 = 0.285 + 0.2\cos(\theta_2)$$
$$0.2\cos(\theta_2) = x^2 + y^2 - 0.285$$

$$\theta_2 = \cos^{-1}\left(\frac{x^2+y^2-0.285}{0.2}\right)$$

$$x = 0.4\cos(\theta_1) + 0.25\cos\theta_1\cos\theta_2 - 0.25\sin\theta_1\sin\theta_2$$
$$= \cos\theta_1(0.4 + 0.25\cos\theta_2) - (0.25\sin\theta_2)\sin\theta_1$$

$$y = 0.4\sin\theta_1 + 0.25\sin\theta_1\cos\theta_2 + 0.25\cos\theta_1\sin\theta_2$$
$$= 0.25\sin\theta_2\cos\theta_1 + \sin\theta_1(0.4 + 0.25\cos\theta_2)$$

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.4 + 0.25\cos\theta_2 & -0.25\sin\theta_2 \\ 0.25\sin\theta_2 & 0.4 + 0.25\cos\theta_2 \end{bmatrix} \begin{bmatrix} \cos\theta_1 \\ \sin\theta_1 \end{bmatrix}$$

$$\Delta\sin\theta_1 = \begin{vmatrix} 0.4 + 0.25\cos\theta_2 & x \\ 0.25\sin\theta_2 & y \end{vmatrix} = y(0.4 + 0.25\cos\theta_2) - 0.25x\sin\theta_2$$

$$\Delta\cos\theta_1 = \begin{vmatrix} x & -0.25\sin\theta_2 \\ y & 0.4 + 0.25\cos\theta_2 \end{vmatrix} = x(0.4 + 0.25\cos\theta_2) + 0.25y\sin\theta_2$$

$$\Delta = \begin{vmatrix} 0.4 + 0.25\cos\theta_2 & -0.25\sin\theta_2 \\ 0.25\sin\theta_2 & 0.4 + 0.25\cos\theta_2 \end{vmatrix} = (0.16 + 0.2\cos\theta_2 + 0.125\cos^2\theta_2) + 0.125\sin^2\theta_2$$
$$= 0.285 + 0.2\cos\theta_2$$

$$\sin(\theta_1) = \frac{y(0.4 + 0.25\cos\theta_2) - 0.25x\sin\theta_2}{0.285 + 0.2\cos\theta_2}$$

$$\cos(\theta_1) = \frac{x(0.4 + 0.25\cos\theta_2) + 0.25y\sin\theta_2}{0.285 + 0.2\cos\theta_2}$$

$$\tan(\theta_1) = \frac{\dfrac{y(0.4 + 0.25\cos\theta_2) - 0.25x\sin\theta_2}{0.285 + 0.2\cos\theta_2}}{\dfrac{x(0.4 + 0.25\cos\theta_2) + 0.25y\sin\theta_2}{0.285 + 0.2\cos\theta_2}} = \frac{y(0.4 + 0.25\cos\theta_2) - 0.25x\sin\theta_2}{x(0.4 + 0.25\cos\theta_2) + 0.25y\sin\theta_2}$$

$$\theta_1 = \tan^{-1}\left(\frac{y(0.4 + 0.25\cos\theta_2) - 0.25x\sin\theta_2}{x(0.4 + 0.25\cos\theta_2) + 0.25y\sin\theta_2}\right)$$

Therefore the joint angles and the height distance are:

$$\theta_1 = \tan^{-1}\left(\frac{y(0.4 + 0.25\cos\theta_2) - 0.25x\sin\theta_2}{x(0.4 + 0.25\cos\theta_2) + 0.25y\sin\theta_2}\right)$$

$$\theta_2 = \cos^{-1}\left(\frac{x^2 + y^2 - 0.285}{0.2}\right)$$

$$d = 0.1416 - z$$

In order for the robot gripper to grip the bars at same coordinates but rotating at different angle along the axis of bar centroid, the orientation angle of roll (the area where the tool is attached to)

$$Pitch\ angle, \theta = -\sin^{-1}(x_z) = -\sin^{-1}(0) = 0°$$

$$Yaw\ angle, \psi = \sin^{-1}\left(\frac{y_z}{\cos\theta}\right) = \sin^{-1}\left(\frac{0}{\cos 0}\right) = 0°$$

$$\therefore \phi = \sin^{-1}\left(\frac{x_y}{\cos\theta}\right) = \sin^{-1}\left(\frac{\sin(\theta_1 + \theta_2 - \theta_4)}{\cos\theta}\right) = \sin^{-1}\left(\frac{\sin(\theta_1 + \theta_2 - \theta_4)}{1}\right)$$

$$\therefore Roll\ angle, \phi = \theta_1 + \theta_2 - \theta_4$$

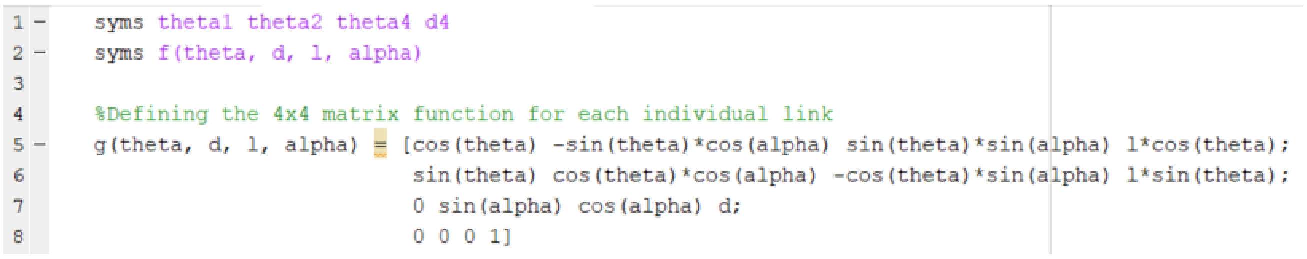
$$\therefore Pitch\ angle, \theta = 0°$$

$$\therefore Yaw\ angle, \psi = 0°$$

Deriving Forward and Inverse Kinematics using MATLAB

The previous section shows the forward and inverse kinematics solutions derived from theoretical calculations. In this section, same thing will be carried out, but MATLAB will be used to code the forward and inverse kinematics solution will be coded into MATLAB.

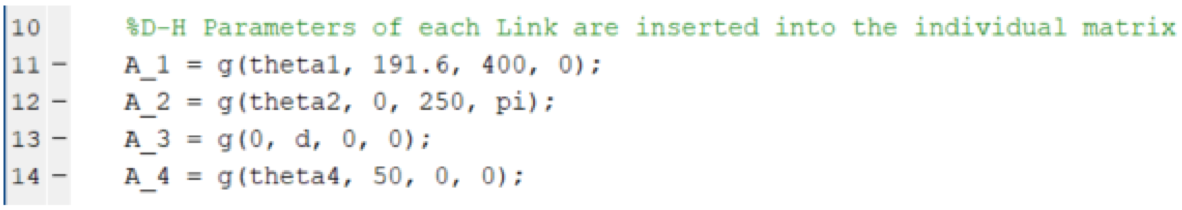
For the forward kinematic solutions, initially the A matrix function will be defined as Figure 14 below:



```
syms theta1 theta2 theta4 d4
syms f(theta, d, l, alpha)

%Defining the 4x4 matrix function for each individual link
g(theta, d, l, alpha) = [cos(theta) -sin(theta)*cos(alpha) sin(theta)*sin(alpha) l*cos(theta);
                         sin(theta) cos(theta)*cos(alpha) -cos(theta)*sin(alpha) l*sin(theta);
                         0 sin(alpha) cos(alpha) d;
                         0 0 0 1]
```

Figure 14: Code Snippet for Defining the A matrix function

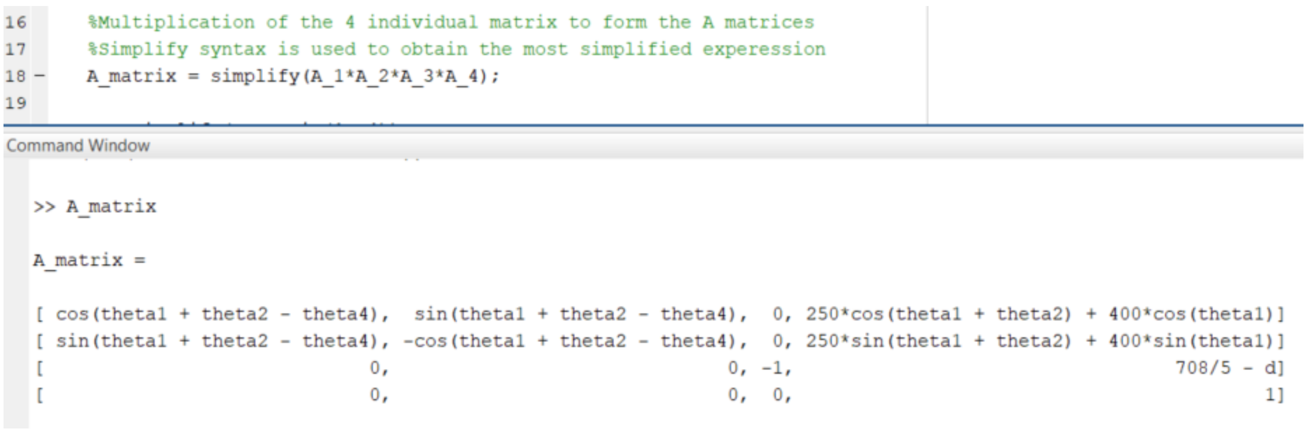
The D-H Parameters obtained from the Theoretical Calculation section will be coded into MATLAB, as shown in Figure 15 below:



```
%D-H Parameters of each Link are inserted into the individual matrix
A_1 = g(theta1, 191.6, 400, 0);
A_2 = g(theta2, 0, 250, pi);
A_3 = g(0, d, 0, 0);
A_4 = g(theta4, 50, 0, 0);
```

Figure 15: Code Snippet for Defining the D-H Parameters

The resultant A matrix is calculated with the results shown at the command window:



```
%Multiplication of the 4 individual matrix to form the A matrices
%Simplify syntax is used to obtain the most simplified experession
A_matrix = simplify(A_1*A_2*A_3*A_4);
```

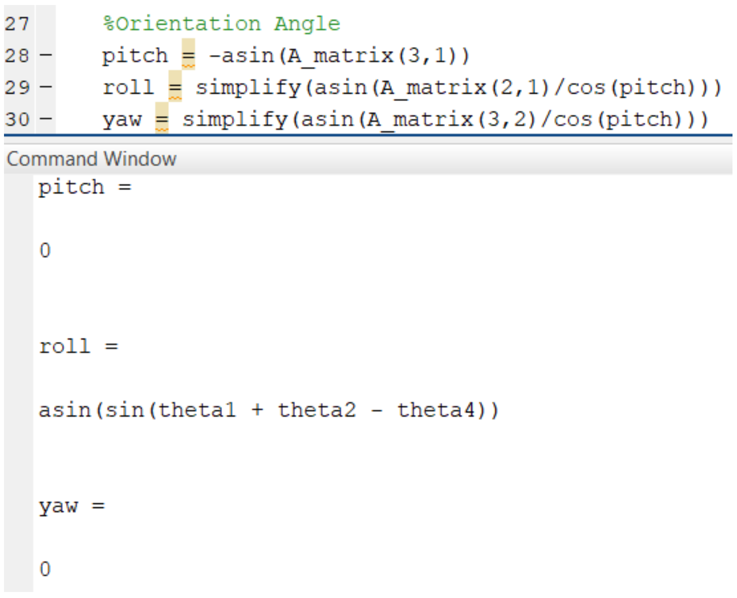
Command Window

```
>> A_matrix

A_matrix =

[ cos(theta1 + theta2 - theta4),  sin(theta1 + theta2 - theta4),  0, 250*cos(theta1 + theta2) + 400*cos(theta1)]
[ sin(theta1 + theta2 - theta4), -cos(theta1 + theta2 - theta4),  0, 250*sin(theta1 + theta2) + 400*sin(theta1)]
[                             0,                              0, -1,                                   708/5 - d]
[                             0,                              0,  0,                                           1]
```

Figure 16: Code Snippet of the Results of the A Matrix obtained

Based on the results in Figure 5 above, the resultant solution has the same answer as the matrix derived theoretically. The function simplify() is used to expand the trigonometric expression by implementing the trigonometric identities into the expression and further simplify it. Therefore, the forward kinematics solution is validated.

Apart from that, the orientation angle is also calculated:



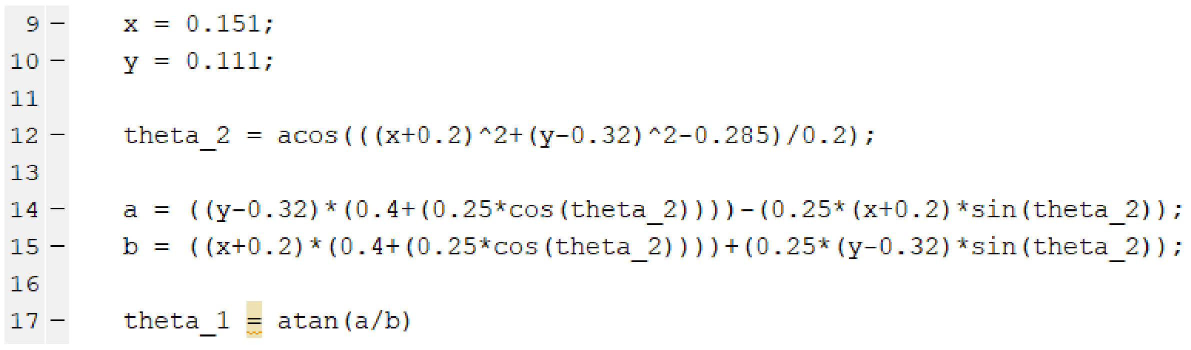
```
%Orientation Angle
pitch = -asin(A_matrix(3,1))
roll = simplify(asin(A_matrix(2,1)/cos(pitch)))
yaw = simplify(asin(A_matrix(3,2)/cos(pitch)))
```

Command Window

```
pitch =

0


roll =

asin(sin(theta1 + theta2 - theta4))


yaw =

0
```

Figure 17: Code Snippet for Calculating Orientation Angle

Other than that, after obtaining the inverse kinematics solution from theoretical calculation, it is coded into MATLAB for calculating the joint angles required for different location of the bars.



```
x = 0.151;
y = 0.111;

theta_2 = acos(((x+0.2)^2+(y-0.32)^2-0.285)/0.2);

a = ((y-0.32)*(0.4+(0.25*cos(theta_2))))-(0.25*(x+0.2)*sin(theta_2));
b = ((x+0.2)*(0.4+(0.25*cos(theta_2))))+(0.25*(y-0.32)*sin(theta_2));

theta_1 = atan(a/b)
```

Figure 18: Inverse Kinematics Solutions coded in MATLAB

Simulation of Robot Motion using Robotics Toolbox in MATLAB

For this section, the centroid location of the bars obtained from the image processing will be used so that the bars can be plotted in MATLAB together with the MATLAB. Then, by using the inverse kinematics solution derived, the robot will move its arm until the end effector is above the bar by calculating the θ_1 and θ_2 values in MATLAB and then coded using the robot.plot() function. Based on the information obtained from the image processing code, the centroid locations of each coloured bars are listed below:

Table 3: List of Centroid Coordinates of the coloured-bars (Image2.jpg)

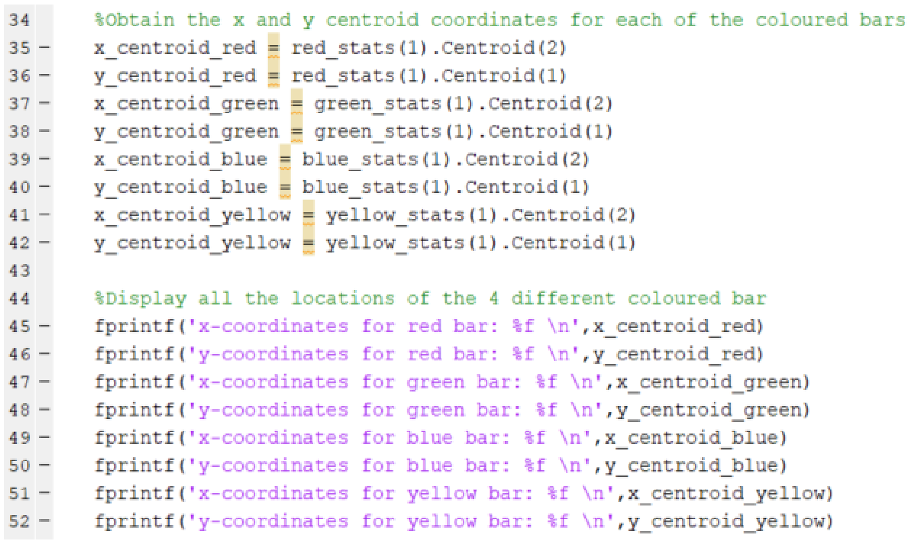
Colour of the bar	**Centroid Coordinates**	
Red	x-coordinates	0.151
	y-coordinates	0.111
Green	x-coordinates	0.208
	y-coordinates	0.156
Blue	x-coordinates	0.365
	y-coordinates	0.172
Yellow	x-coordinates	0.390
	y-coordinates	0.071

By input their x and y centroid coordinates into the inverse kinematics code, the θ_1 and θ_2 for each bars are listed as below:

Table 4: Joint Angles required for picking the coloured-bars (Image2.jpg)

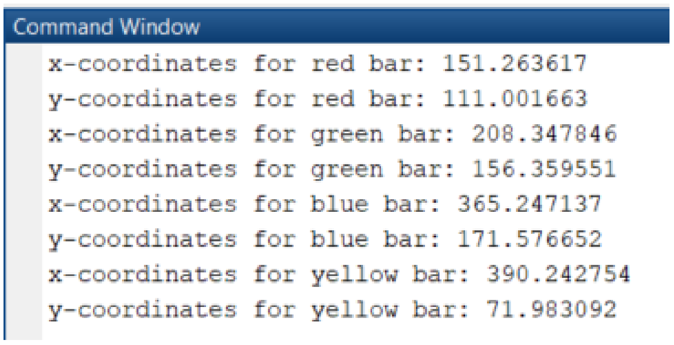
Colour of the bar	**Joint Angles**
Red	$\theta_1 = -1.2115\, rad$ $\theta_2 = 2.2026\, rad$
Green	$\theta_1 = -1.0437\, rad$ $\theta_2 = 2.0468\, rad$
Blue	$\theta_1 = -0.7281\, rad$ $\theta_2 = 1.2863\, rad$
Yellow	$\theta_1 = -0.7365\, rad$ $\theta_2 = 0.895\, rad$

The x and y centroid coordinates of the bars obtained from the image processing is obtained and displayed in MATLAB:



```
%Obtain the x and y centroid coordinates for each of the coloured bars
x_centroid_red = red_stats(1).Centroid(2)
y_centroid_red = red_stats(1).Centroid(1)
x_centroid_green = green_stats(1).Centroid(2)
y_centroid_green = green_stats(1).Centroid(1)
x_centroid_blue = blue_stats(1).Centroid(2)
y_centroid_blue = blue_stats(1).Centroid(1)
x_centroid_yellow = yellow_stats(1).Centroid(2)
y_centroid_yellow = yellow_stats(1).Centroid(1)

%Display all the locations of the 4 different coloured bar
fprintf('x-coordinates for red bar: %f \n',x_centroid_red)
fprintf('y-coordinates for red bar: %f \n',y_centroid_red)
fprintf('x-coordinates for green bar: %f \n',x_centroid_green)
fprintf('y-coordinates for green bar: %f \n',y_centroid_green)
fprintf('x-coordinates for blue bar: %f \n',x_centroid_blue)
fprintf('y-coordinates for blue bar: %f \n',y_centroid_blue)
fprintf('x-coordinates for yellow bar: %f \n',x_centroid_yellow)
fprintf('y-coordinates for yellow bar: %f \n',y_centroid_yellow)
```

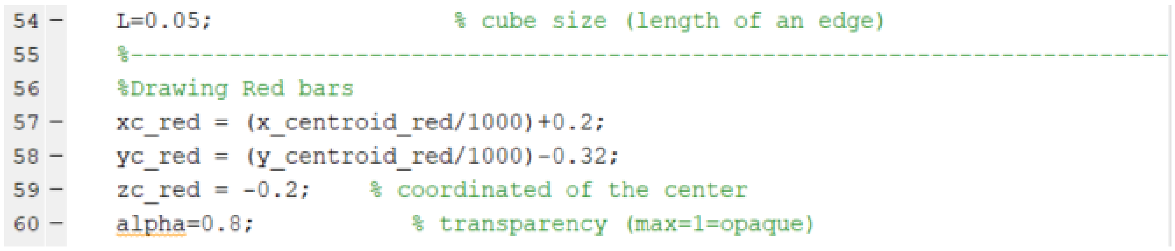
Figure 19: Code Snippet used to obtain and display the x and y centroid coordinates of the bars' location



```
x-coordinates for red bar: 151.263617
y-coordinates for red bar: 111.001663
x-coordinates for green bar: 208.347846
y-coordinates for green bar: 156.359551
x-coordinates for blue bar: 365.247137
y-coordinates for blue bar: 171.576652
x-coordinates for yellow bar: 390.242754
y-coordinates for yellow bar: 71.983092
```

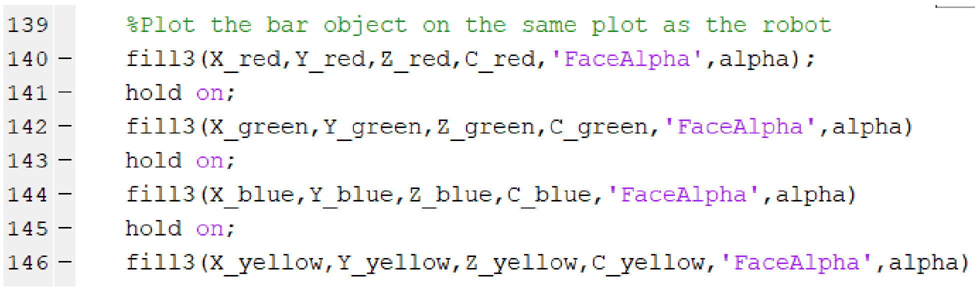
Figure 20: Results for the centroid coordinates shown in command window

After obtaining the centroid coordinates, the bars will be plotted on the 3d plot one by one. The length of the bar is set at 0.05, so that the bars won't be overlapping with each other when plotted in MATLAB. The xyz centre coordinates are defined, with z coordinates placed at -0.2. So that the pointing of the end effector on the bar can be seen clearly when the prismatic link extends downwards in the simulation.



```
L=0.05;                   % cube size (length of an edge)
%-----------------------------------------------------------------------
%Drawing Red bars
xc_red = (x_centroid_red/1000)+0.2;
yc_red = (y_centroid_red/1000)-0.32;
zc_red = -0.2;      % coordinated of the center
alpha=0.8;               % transparency (max=1=opaque)
```

Figure 21: Setting up Parameters required to plot the bar



```matlab
%Plot the bar object on the same plot as the robot
fill3(X_red,Y_red,Z_red,C_red,'FaceAlpha',alpha);
hold on;
fill3(X_green,Y_green,Z_green,C_green,'FaceAlpha',alpha)
hold on;
fill3(X_blue,Y_blue,Z_blue,C_blue,'FaceAlpha',alpha)
hold on;
fill3(X_yellow,Y_yellow,Z_yellow,C_yellow,'FaceAlpha',alpha)
```

Figure 22: Code Snippet for plotting out all the bars with their respective centroid coordinates as the centre of plot

In Figure 22, the fill3() function is used for 3D plotting. The hold on function is used so that the 1st plotted bar will not be overwrote by the 2nd bar when the 2nd bar is plotted on the 3D plot. Figure 12, below shows the resultant 3d plot with the bars plotted out.

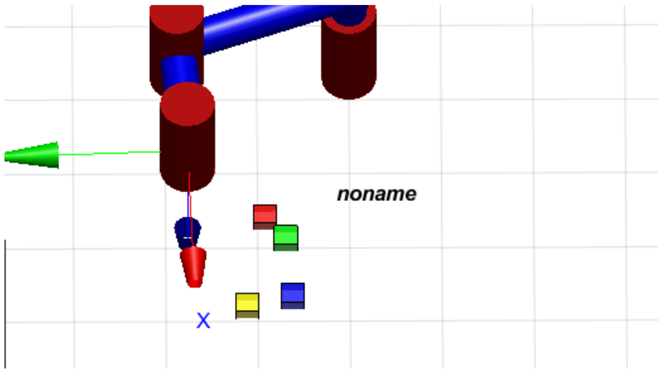
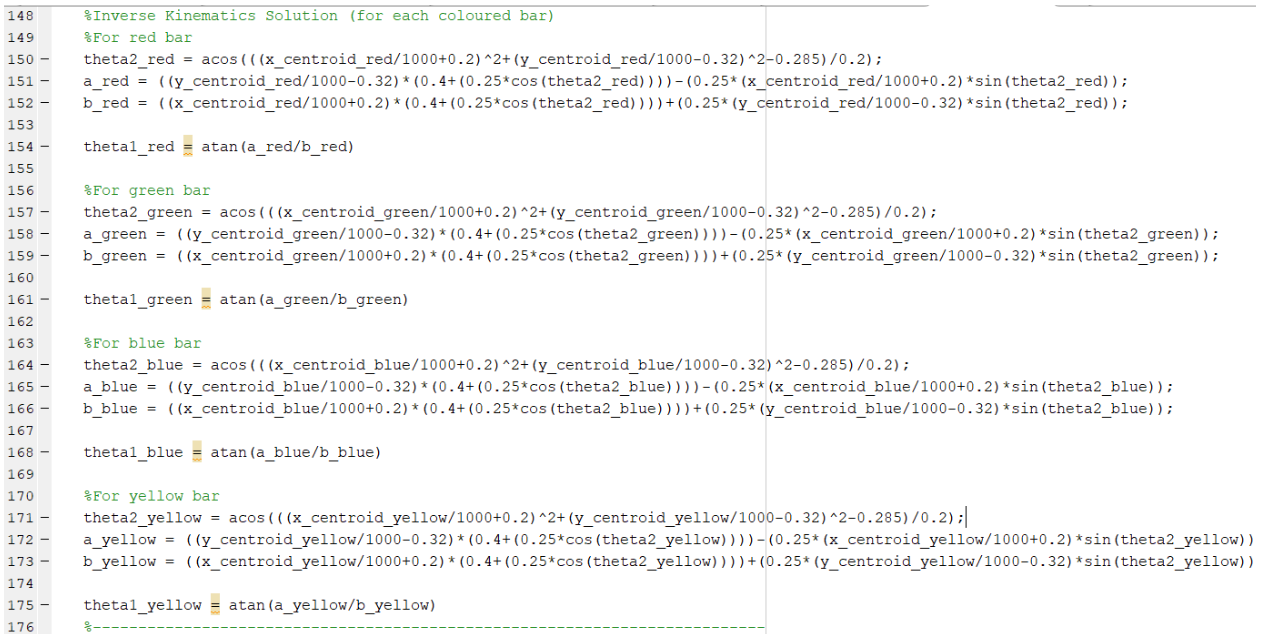


Figure 23: Plotting of bars in MATLAB

From the Figure 23 shown above, the 4 different coloured bars are plotted on the MATLAB. In order to verify whether the inverse kinematics solution is validated, the joint angles for Link 1 and 2 to pick up different colour objects will be coded into MATLAB.

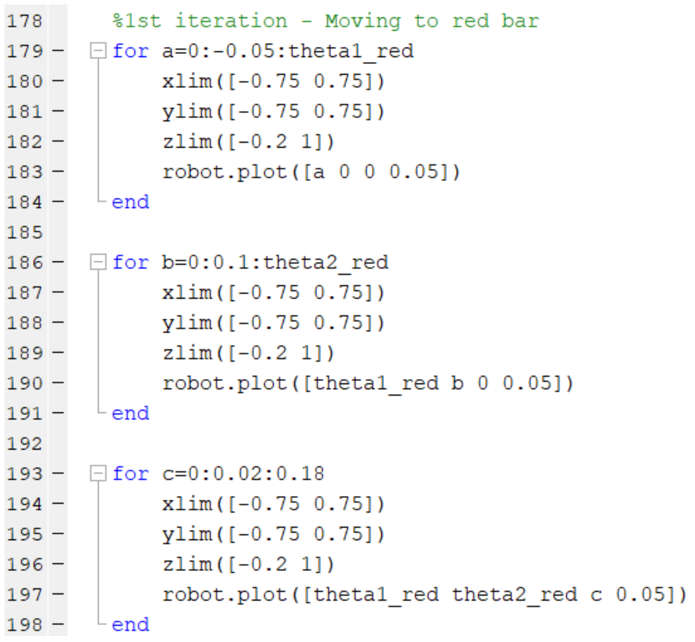


```matlab
%Inverse Kinematics Solution (for each coloured bar)
%For red bar
theta2_red = acos(((x_centroid_red/1000+0.2)^2+(y_centroid_red/1000-0.32)^2-0.285)/0.2);
a_red = ((y_centroid_red/1000-0.32)*(0.4+(0.25*cos(theta2_red))))-(0.25*(x_centroid_red/1000+0.2)*sin(theta2_red));
b_red = ((x_centroid_red/1000+0.2)*(0.4+(0.25*cos(theta2_red))))+(0.25*(y_centroid_red/1000-0.32)*sin(theta2_red));

theta1_red = atan(a_red/b_red)

%For green bar
theta2_green = acos(((x_centroid_green/1000+0.2)^2+(y_centroid_green/1000-0.32)^2-0.285)/0.2);
a_green = ((y_centroid_green/1000-0.32)*(0.4+(0.25*cos(theta2_green))))-(0.25*(x_centroid_green/1000+0.2)*sin(theta2_green));
b_green = ((x_centroid_green/1000+0.2)*(0.4+(0.25*cos(theta2_green))))+(0.25*(y_centroid_green/1000-0.32)*sin(theta2_green));

theta1_green = atan(a_green/b_green)

%For blue bar
theta2_blue = acos(((x_centroid_blue/1000+0.2)^2+(y_centroid_blue/1000-0.32)^2-0.285)/0.2);
a_blue = ((y_centroid_blue/1000-0.32)*(0.4+(0.25*cos(theta2_blue))))-(0.25*(x_centroid_blue/1000+0.2)*sin(theta2_blue));
b_blue = ((x_centroid_blue/1000+0.2)*(0.4+(0.25*cos(theta2_blue))))+(0.25*(y_centroid_blue/1000-0.32)*sin(theta2_blue));

theta1_blue = atan(a_blue/b_blue)

%For yellow bar
theta2_yellow = acos(((x_centroid_yellow/1000+0.2)^2+(y_centroid_yellow/1000-0.32)^2-0.285)/0.2);
a_yellow = ((y_centroid_yellow/1000-0.32)*(0.4+(0.25*cos(theta2_yellow))))-(0.25*(x_centroid_yellow/1000+0.2)*sin(theta2_yellow))
b_yellow = ((x_centroid_yellow/1000+0.2)*(0.4+(0.25*cos(theta2_yellow))))+(0.25*(y_centroid_yellow/1000-0.32)*sin(theta2_yellow))

theta1_yellow = atan(a_yellow/b_yellow)
%-----------------------------------------------------------------------
```

Figure 24: Inverse Kinematics Solutions for 4 different coloured bars

In Figure 24 above, the value *theta1_red*, *theta1_green*, *theta1_blue*, *theta1_yellow*, *theta2_red*, *theta2_green*, *theta2_blue*, and *theta1_yellow* are the joint angles θ_1 and θ_2 required for the robot joint to rotate.

After that, it will be used to code into the robot motion algorithm, which is shown below:



```
%1st iteration - Moving to red bar
for a=0:-0.05:theta1_red
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([a 0 0 0.05])
end

for b=0:0.1:theta2_red
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([theta1_red b 0 0.05])
end

for c=0:0.02:0.18
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([theta1_red theta2_red c 0.05])
end
```

Figure 25: Code Snippet for Robot Motion Simulation in MATLAB

In Figure 25 above, there 3 for-loops in which each of the loop is responsible for rotating each individual for link 1, 2, and the prismatic link 3. It can be seen that in order for the robot to move to the coordinates right above the red-coloured bar, *theta1_red* and *theta2_red* are used so that the robot will rotate each individual link until it reaches that angle. Hence in this scenario, 1st for-loop will rotate link 1 until it reaches the *theta1_red* calculated by the MATLAB program, followed by the 2nd for-loop which rotates link 2 until it reaches the *theta2_red*. Finally, the 3rd for-loop will extend the prismatic link downwards.

The results for the position of the robot arm for each colour bars are shown below:

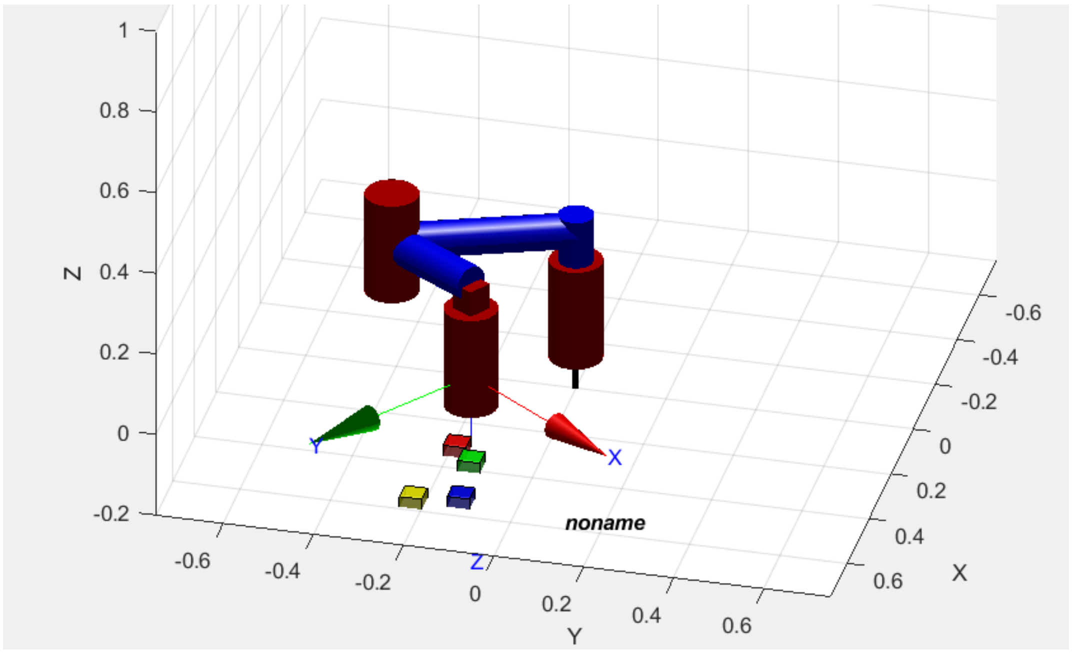


Figure 26: Position of Robot and its link 1, 2, and 3 when red-coloured bar is to be picked

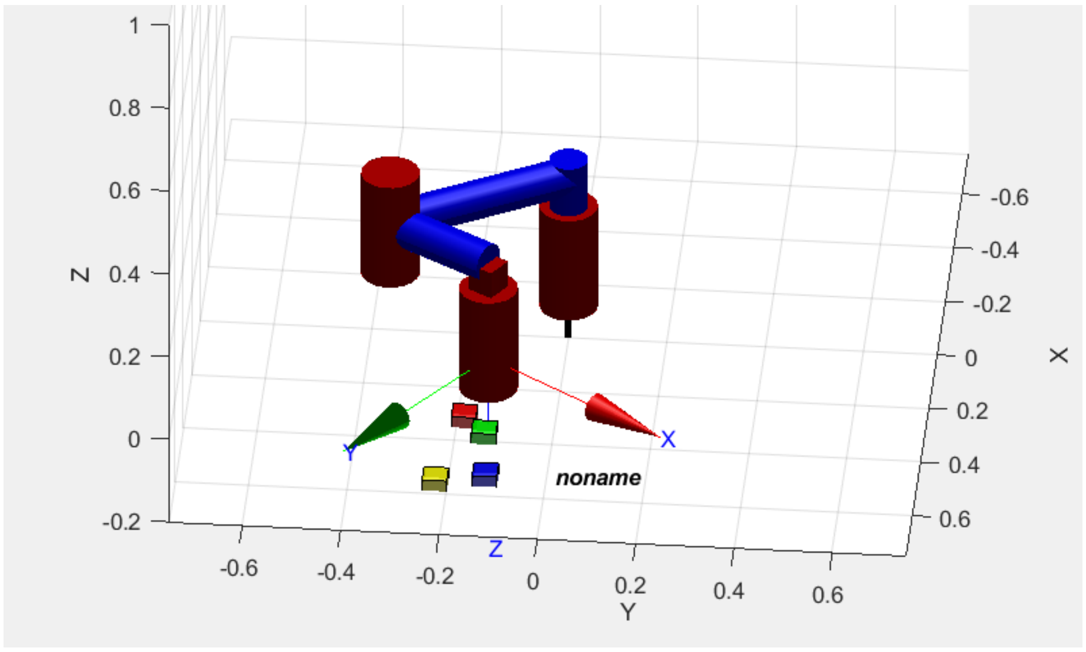


Figure 27: Position of Robot and its link 1, 2, and 3 when green-coloured bar is to be picked

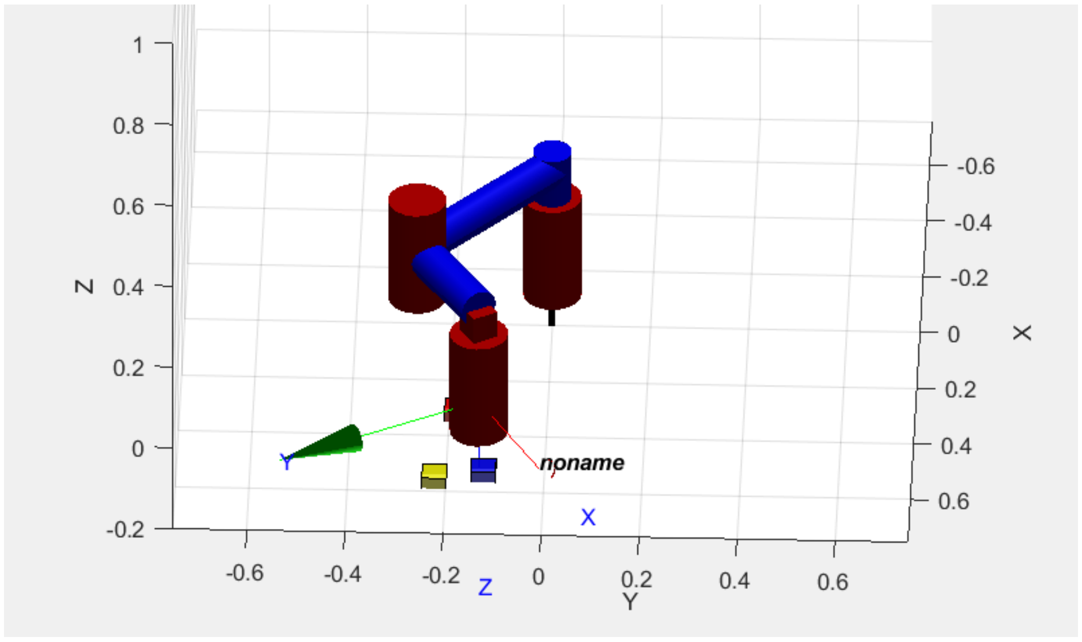


Figure 28: Position of Robot and its link 1, 2, and 3 when blue-coloured bar is to be picked

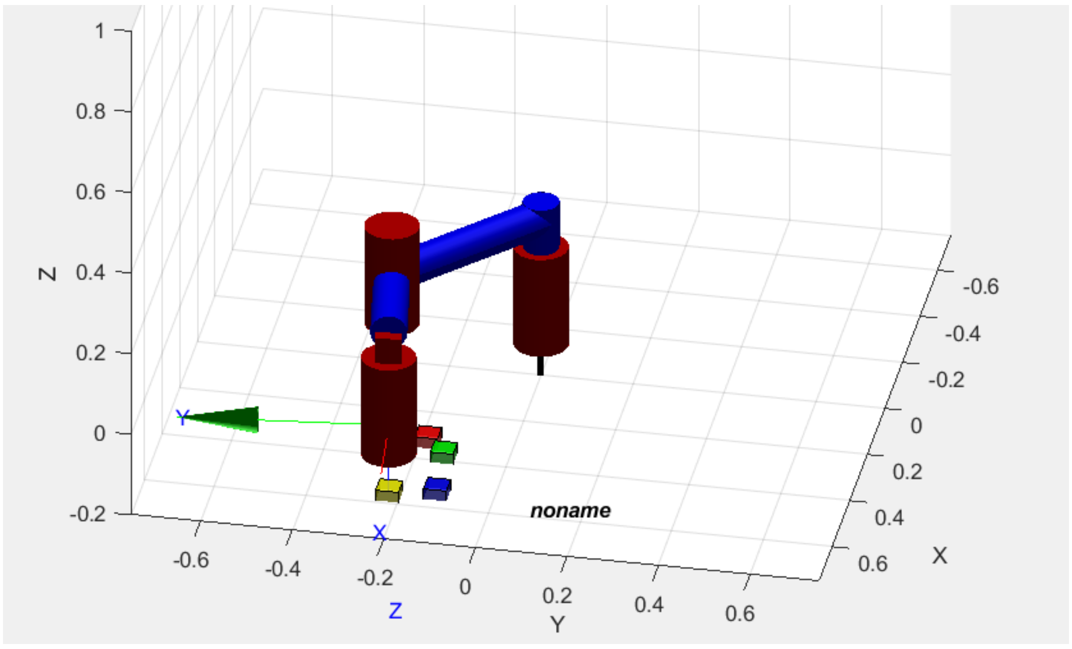
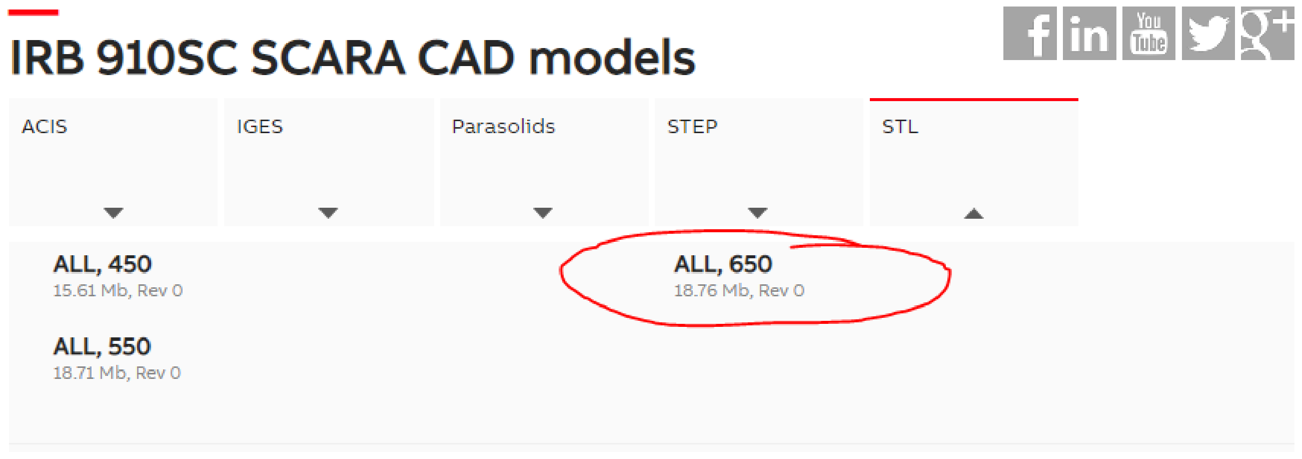


Figure 29: Position of Robot and its link 1, 2, and 3 when -coloured bar is to be picked

Based on the results obtained, the end effector pointer is able to point to the correct colour of the bar. There are minor errors in which the pointing is off-centre and instead it points to the near edge of the bar, such as the pointing of red-coloured bar in Figure 26. In overall, the inverse kinematics derived from the theoretical calculation is mostly validated when simulation of the robot motion is carried out, with the exception of the minor errors occurred.

Creating 3D Model for the SCARA Robot in MATLAB

STL File



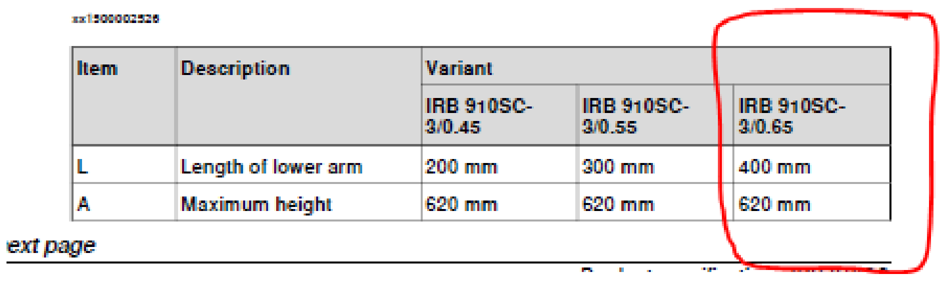
IRB 910SC SCARA CAD models

ALL, 450
15.61 Mb, Rev 0

ALL, 550
18.71 Mb, Rev 0

ALL, 650
18.76 Mb, Rev 0

Figure 30: Selection of SCARA CAD Models



xx1500002526

Item	Description	Variant		
		IRB 910SC-3/0.45	IRB 910SC-3/0.55	IRB 910SC-3/0.65
L	Length of lower arm	200 mm	300 mm	400 mm
A	Maximum height	620 mm	620 mm	620 mm

ext page

Figure 31: Choosing IRB 910 SC 3/0.65 model

The STL file is downloaded from the ABB website. ALL,650 is downloaded which has 400mm for the length of lower arm.

Download Link: https://new.abb.com/products/robotics/es/robots-industriales/irb-910sc/irb-910sc-cad-models

After download the file, there is a lot of part inside the file. From the file, choose the part for Base, Shaft, Arm 1, Arm 2 and the hand. After that, rename the STL file to link 0 to link 4 where:

Table 5: Parts name and their respective link name

Parts	**Link**
Base	link 0
Arm 1	link 1
Arm 2	link 2
Shaft	link 3
Hand (get it from Dr. Hudy)	link 4

After that, place the file into the correct path where inside MATLAB>> Add Ons>>Toolboxes>>Robotics Toolbox for MATLAB>> code>> data>> meshes.

Opening STL file using SolidWorks

Base, link 0

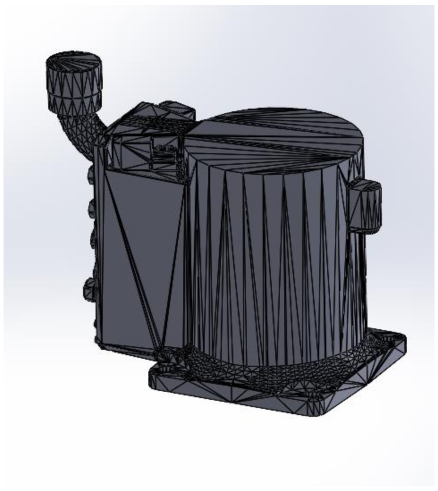


Figure 32: Link 0 in STL format

Arm 1, link 1

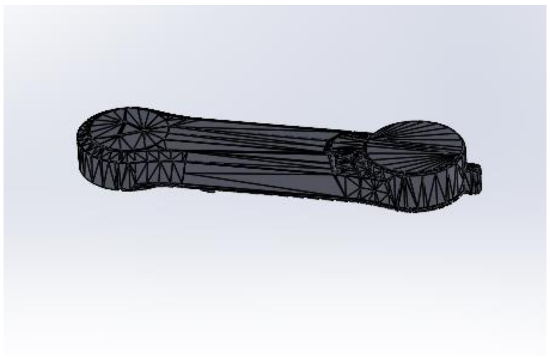


Figure 33: Link 1 in STL format

Arm 2, link 2

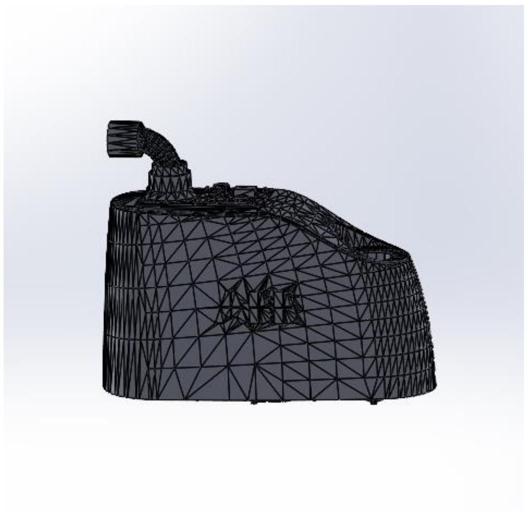


Figure 34: Link 2 in STL format

Shaft, link 3

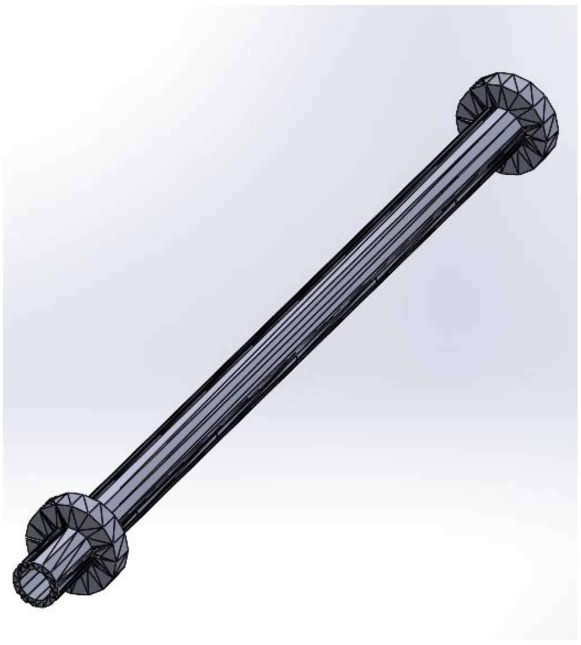


Figure 35: Link 3 in STL format

Hand, link 4

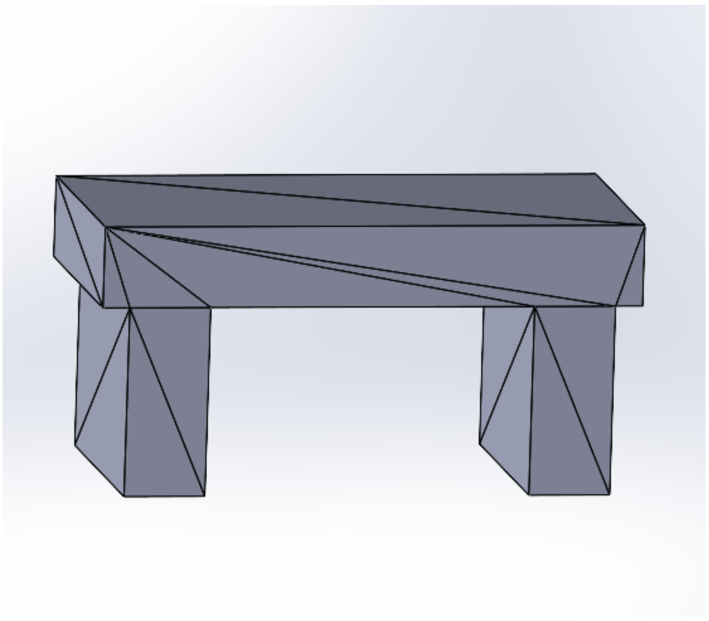


Figure 36: Link 4 in STL format

MATLAB Code using Plot:

```
L1 = Link([0 0.1916 0.4 0])
L2 = Link([0 0 0.25 0])
L3 = Link([0 0.18 0 pi 1])
L4 = Link([0 0.05 0 0])

L3.qlim = [0 0.18]

robot = SerialLink ([L1 L2 L3 L4])
robot.plot ([0 0 0 0]);
```

Result:

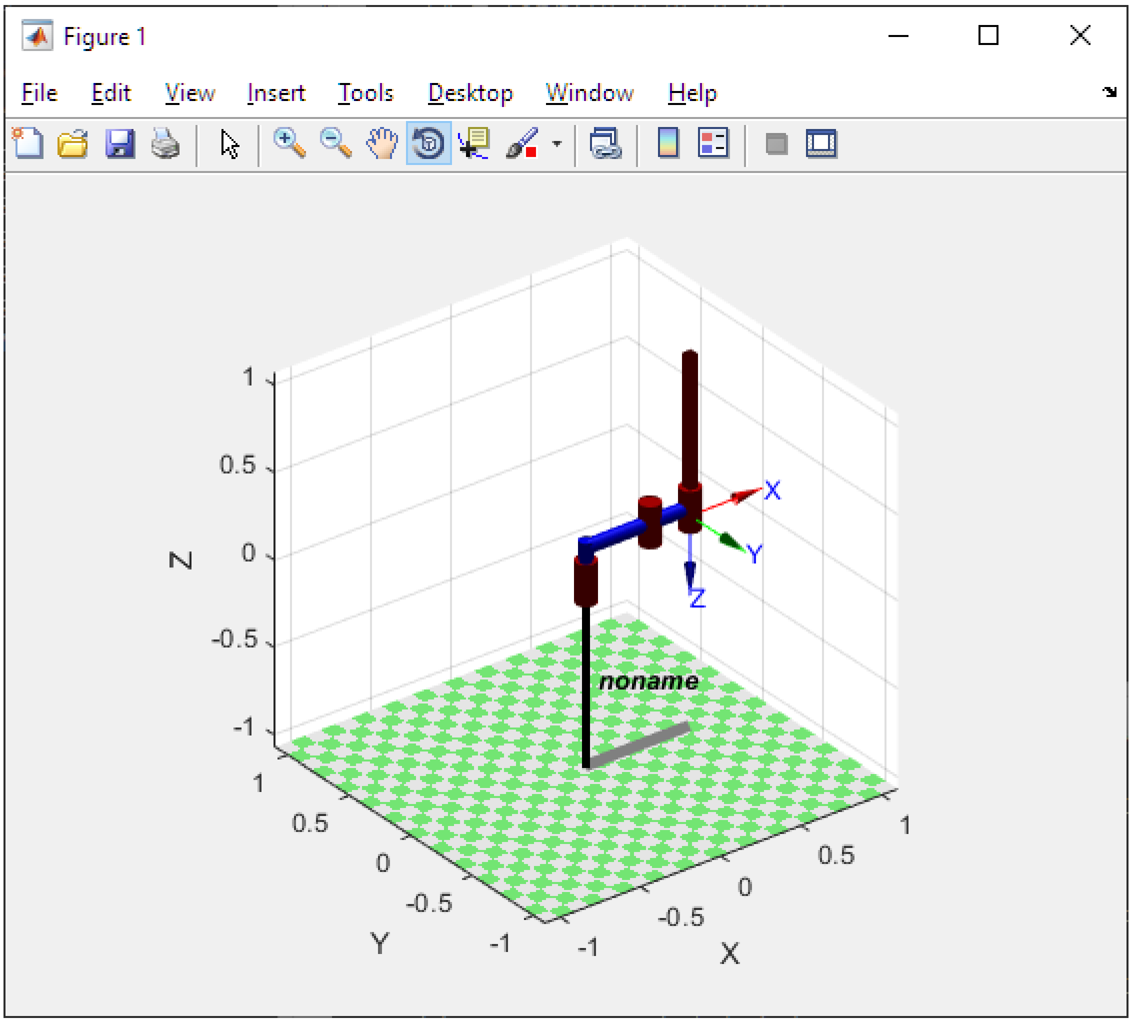


Figure 37: Original Robot Plot (Stick Figure)

MATLAB Code using Plot3d:

```
L1 = Link([0 0.1916 0.4 0])
L2 = Link([0 0 0.25 0])
L3 = Link([0 0.18 0 pi 1])
L4 = Link([0 0.05 0 0])

L3.qlim = [0 0.18]

robot = SerialLink ([L1 L2 L3 L4])

!We don't want to change anything from the link, so just put [0 0 0 0].Locate
the STL file in 'workspace' and set the scale of XYZ.
robot.plot3d ([0 0 0 0],'workspace',[-1 1 -1 1 0 1]);
```

Result:

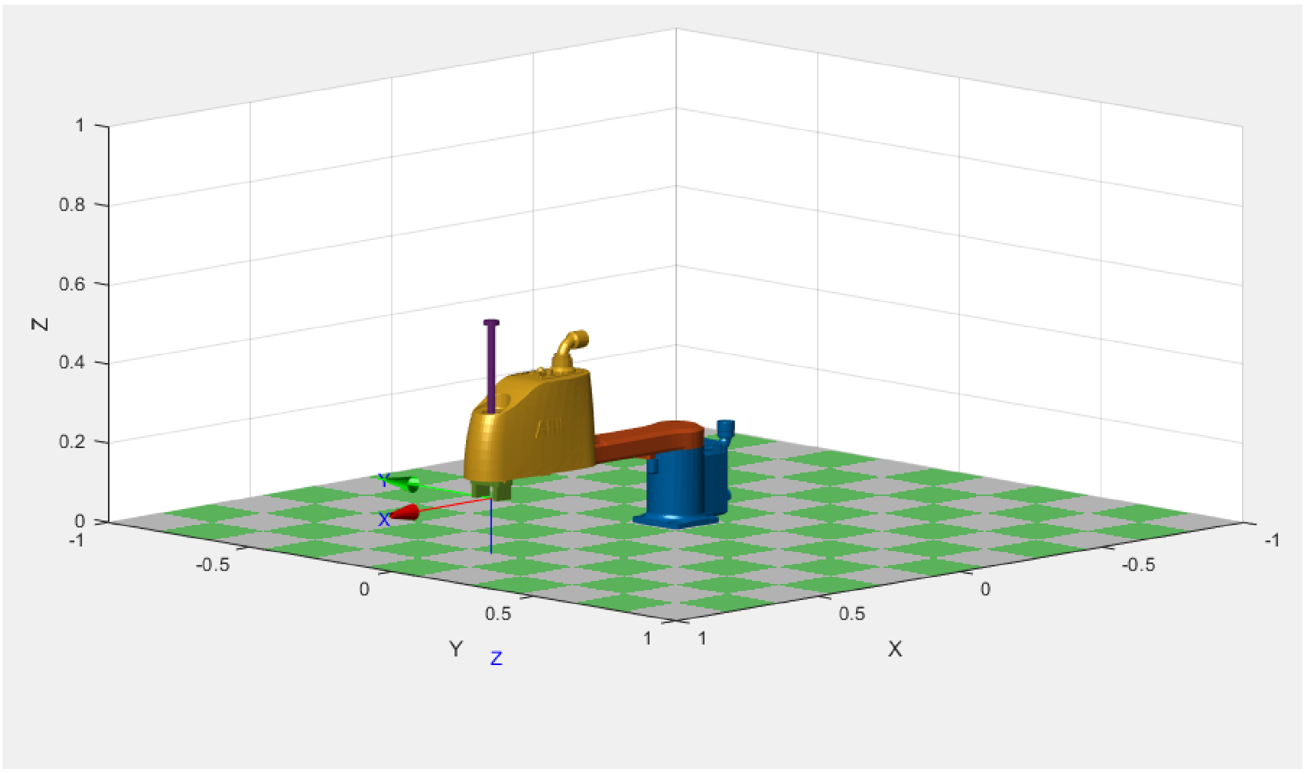


Figure 38: Final Robot Plot with actual 3D Model

The orange arm will be attached to L1, the yellow arm will be attached to L2, the purple shaft will be attached to L3 and the green hand will be attached to L4.

Robot Programming

Adding smart bar

A SmartBar can be imported through Import Library >> Solution library>> SmartBar as shown in figure below.

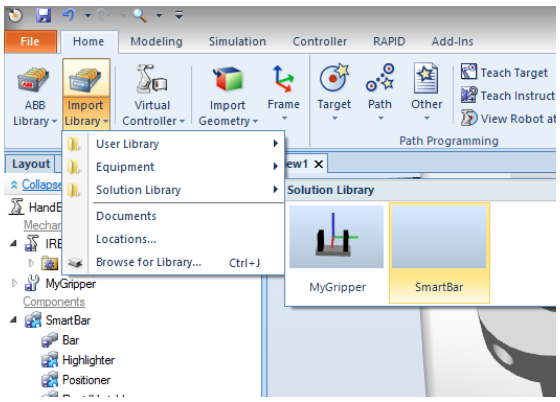
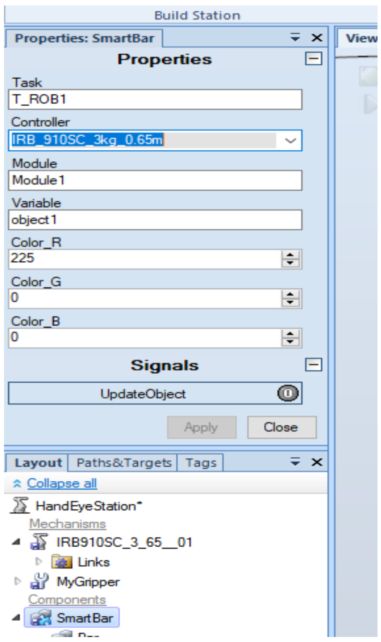


Figure 39: Adding SmartBar

SmartBar can be set up by double clicking on SmartBar component. At SmartBar Properties as shown below.



Task: T_ROB1
Controller: IRB_910SC_3kg_0.65m
Module: Module1
Variable: object1
Color_R: 225
Color_G: 0
Color_B: 0
UpdateObject

Figure 40: setting SmartBar

Setting for each SmartBar:
Controller: IRB_910SC_3kg_0.65m
Module: Module1 (Module to be use in RAPID)

Red Bar:
Variable = object1 (variable name to be use in RAPID)
Color_R = 255

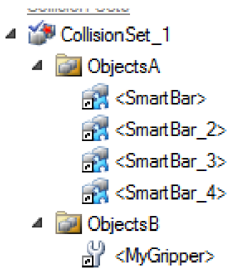
Yellow Bar:
Variable = object2
Color_R = 255

Color_G = 255

Blue Bar:
Variable = object3
Color_B = 255

Green Bar:
Variable = object4
Color_G = 255

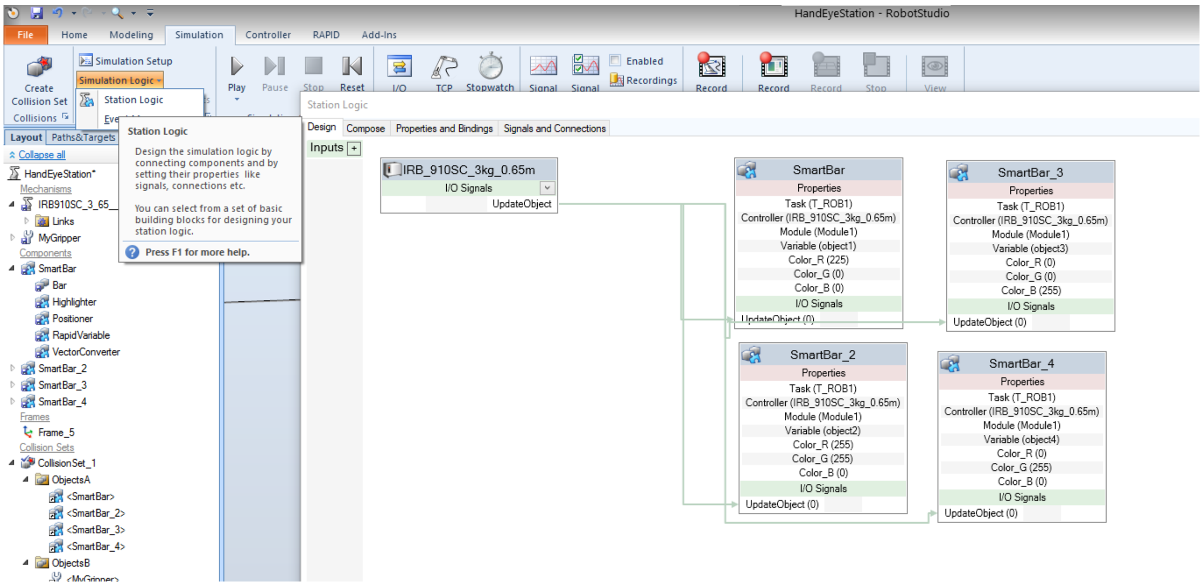
The Bars need to be added to collision Set >> objectA and gripper to collision Set >> objectB, so that the bars can be grab by the gripper.



CollisionSet_1
- ObjectsA
  - <SmartBar>
  - <SmartBar_2>
  - <SmartBar_3>
  - <SmartBar_4>
- ObjectsB
  - <MyGripper>

Figure 41: Collision Set

Lastly, each bar needs to be connected in simulation logic as shown in figure below.



Station Logic

Design the simulation logic by connecting components and by setting their properties like signals, connections etc.

You can select from a set of basic building blocks for designing your station logic.

Press F1 for more help.

IRB_910SC_3kg_0.65m
I/O Signals
UpdateObject

SmartBar
Properties
Task (T_ROB1)
Controller (IRB_910SC_3kg_0.65m)
Module (Module1)
Variable (object1)
Color_R (225)
Color_G (0)
Color_B (0)
I/O Signals
UpdateObject (0)

SmartBar_2
Properties
Task (T_ROB1)
Controller (IRB_910SC_3kg_0.65m)
Module (Module1)
Variable (object2)
Color_R (255)
Color_G (255)
Color_B (0)
I/O Signals
UpdateObject (0)

SmartBar_3
Properties
Task (T_ROB1)
Controller (IRB_910SC_3kg_0.65m)
Module (Module1)
Variable (object3)
Color_R (0)
Color_G (0)
Color_B (255)
I/O Signals
UpdateObject (0)

SmartBar_4
Properties
Task (T_ROB1)
Controller (IRB_910SC_3kg_0.65m)
Module (Module1)
Variable (object4)
Color_R (0)
Color_G (255)
Color_B (0)
I/O Signals
UpdateObject (0)

Figure 42: Connecting simulation logic.

Setting Final Location

The following screenshot shows the final location of 4 bars is placed.

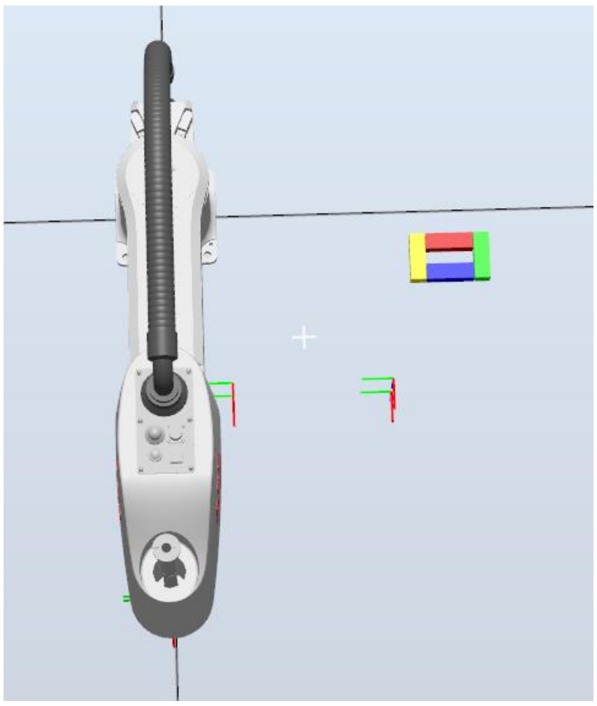


Figure 43: final location of bars placed.

The location of each bar from the origin is:

Red:	translation (x = 55, y = 451),	orientation (z = 90°)
Yellow:	translation (x = 80, y = 400),	orientation (z = 0°)
Blue:	translation (x = 105, y = 451),	orientation (z = 90°)
Green:	translation (x = 80, y = 502),	orientation (z = 0°)

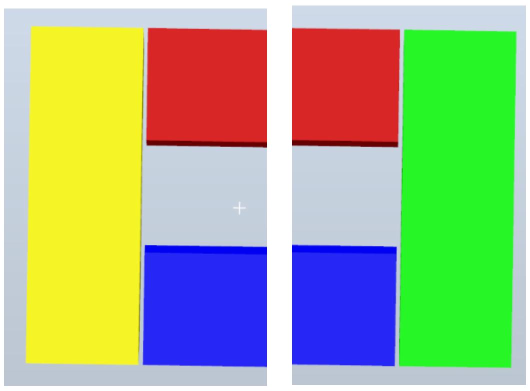


Figure 44: gap between bars

The red and blue bars are design to be placed at 1mm apart from yellow and green bar.

Coding

```
    ! Destination Target
    VAR robtarget DestiantionTargetLift1:=[[55,451,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget DestiantionTarget1:=[[55,451,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    VAR robtarget DestiantionTargetLift2:=[[80,400,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget DestiantionTarget2:=[[80,400,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    VAR robtarget DestiantionTargetLift3:=[[105,451,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget DestiantionTarget3:=[[105,451,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    VAR robtarget DestiantionTargetLift4:=[[80,502,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget DestiantionTarget4:=[[80,502,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Orientation for green bar (object4)

```
        DestiantionTargetLift4.rot:=OrientZYX(0,180,0);
        DestiantionTarget4.rot:=OrientZYX(0,180,0);
```

Orientation for yellow bar (object2)

```
        DestiantionTargetLift2.rot:=OrientZYX(0,180,0);
        DestiantionTarget2.rot:=OrientZYX(0,180,0);
```

Orientation for red bar (object1)

```
        DestiantionTargetLift1.rot:=OrientZYX(90,180,0);
        DestiantionTarget1.rot:=OrientZYX(90,180,0);
```

Orientation for yellow bar (object3)

```
        DestiantionTargetLift3.rot:=OrientZYX(90,180,0);
        DestiantionTarget3.rot:=OrientZYX(90,180,0);
```

Moving object

The figure below shows end product of how the object being placed.

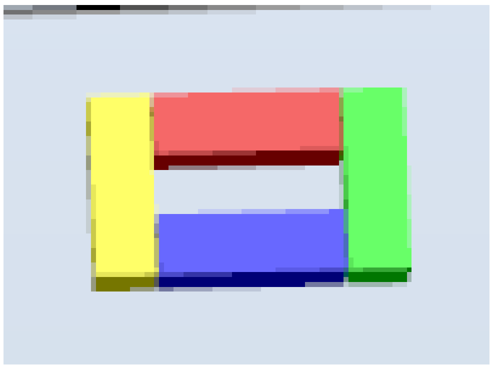


Figure 45: objects location

As the figure above shows the yellow bar and green bar are place at the ends of red and blue bar. There is a problem for the gripper to release the tallow and green bar, the gripper will collide with red and blue bar. Therefore, the yellow and green bar will have to be place at first before moving red and blue bar.

The main code will be run as show below.

```
PROC main()
        socketTest;
        RepositionObject;

        Path_20;
        Path_40;
        Path_10;
        Path_30;
    ENDPROC
```

Path_20; !moving object2 (yellow bar)
Path_40; !moving object4 (green bar)
Path_10; !moving object1 (red bar)
Path_30; !moving object3 (blue bar)

The movement of each path will be set as shown below.

```
        MoveL BendPose,v300,fine,MyGripper\WObj:=wobj0;
        MoveL PickupTargetLift1,v300,fine,MyGripper\WObj:=wobj0;
        MoveL PickupTarget1,v300,fine,MyGripper\WObj:=wobj0;
        WaitTime 0.5;
                Set GripSet;
        WaitTime 0.5;
        MoveL PickupTargetLift1,v300,fine,MyGripper\WObj:=wobj0;
        MoveL BendPose,v300,fine,MyGripper\WObj:=wobj0;
        MoveL DestiantionTargetLift1,v300,fine,MyGripper\WObj:=wobj0;
        MoveL DestiantionTarget1,v300,fine,MyGripper\WObj:=wobj0;
        WaitTime 0.5;
                Reset GripSet;
        WaitTime 0.5;
        MoveL DestiantionTargetLift1,v300,fine,MyGripper\WObj:=wobj0;
```

First, the robot gripper will be initialize to BendPose, then move to where the initial location of the object. Wait for 0.5 second then set the gripper to grip the object and wait for 0.5second to make sure the stability of the object. Liftup object move to BendPose to create a middle point where the robot arm can move in straight line to any location. Move to destination point and let go of the gripper. This robot movement is repeated for all 4 bars.

Initial Location

The initial coordination is shown in figure below. Based on “Image2.jpg” from one of the provided images.

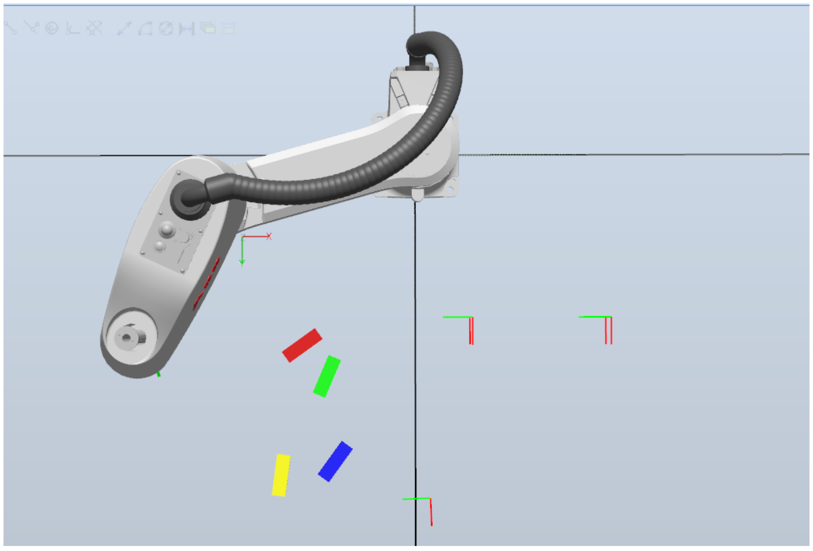


Figure 46: initial location of bars

From image processing the coordinates are define as shown below.

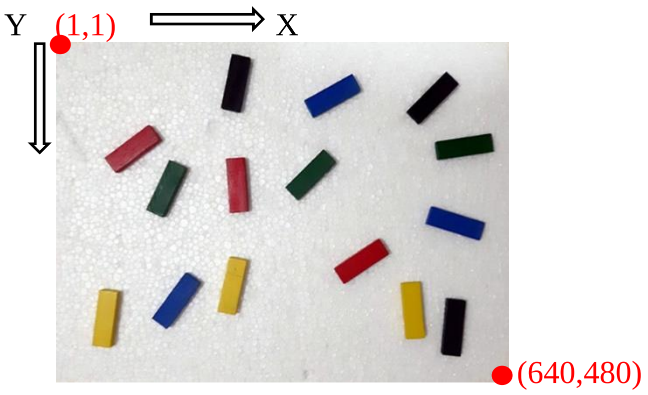


Figure 47: “Image2.jpg” from provided image and coordinations

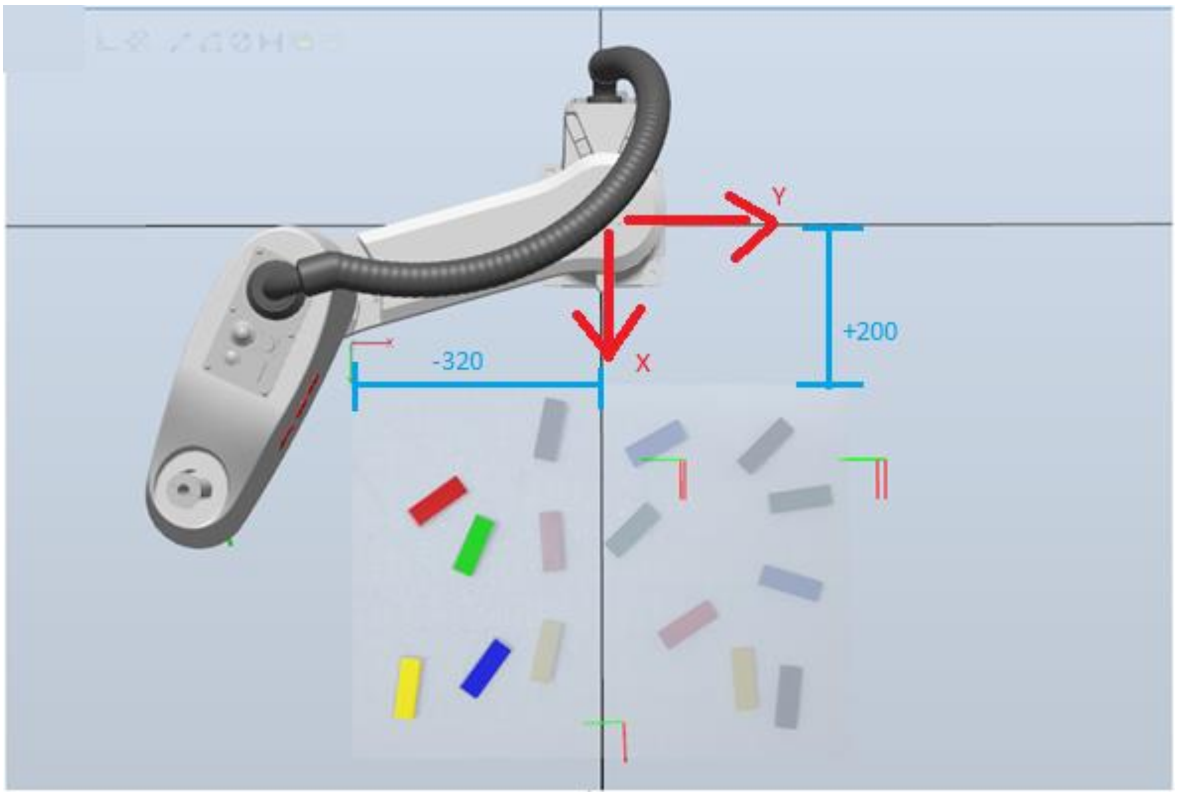


Figure 48: initial location of bars comparing to "image2.jpg"

When the image and the robot station is put together as shown above. The origin of image and robot are different. Therefore, the following code are added in MATLAB to match the origin in robot studio.

```
X1 = First_red(2) +200;
Y1 = First_red(1) -320 ;
R1 = First_red(3) +90 ;
```

The representation of variables in MATLAB is as First_red = [X, Y, R]. However, the X direction in robot studio is equal to Y coordinate in MATLAB. Hens, X1 = First_red(2) with addition of 200mm away from the robot. Same go to Y direction in robot studio is the equal to X coordinate from image processing. Hens, Y1 = First_red(1) with 320mm move to left.

The code for sending coordinates from MATLAB to ROBOT STUDIO is in Appendix E. Code for Robot Studio to receive coordinates is in Appendix D in socketTest().

Note: Run ‘project2.m’ file before running ‘sendCoordinate.m’ file.

Conclusion

In conclusion, the robotics toolbox is able to apply the calculated inverse kinematic and using plot3d to plot the 3d model inside the MATLAB graph. During this project, the team encounter several challenges and difficulties. For example, the importing and extracting of STL files for each robot link part took 2-3 weeks due to the pathname folder being extracted incorrectly. Besides, the MATLAB can communicate with Robot Studio to virtually control the robot to pick and place the bar according to the code. The process will need to have the image processing code running in MATLAB and run the RAPID code in Robot Studio in order to obtain the image processing’s colour bar coordinate. After that, the Robot Studio will allow the MATLAB to control the pick and place process using the built code in MATLAB. It is a very useful tools where the team can gain experiences to operate the robot virtually, even without operating the robot physically.

Appendix

Appendix A: Source Code for Forward and Inverse Kinematics Solutions

```
syms theta1 theta2 theta4 d4
syms f(theta, d, l, alpha)

%Defining the 4x4 matrix function for each individual link
g(theta, d, l, alpha) = [cos(theta) -sin(theta)*cos(alpha)
sin(theta)*sin(alpha) l*cos(theta);
                         sin(theta) cos(theta)*cos(alpha) -
cos(theta)*sin(alpha) l*sin(theta);
                         0 sin(alpha) cos(alpha) d;
                         0 0 0 1]

%D-H Parameters of each Link are inserted into the individual
matrix
A_1 = g(theta1, 191.6, 400, 0);
A_2 = g(theta2, 0, 250, pi);
A_3 = g(0, d, 0, 0);
A_4 = g(theta4, 50, 0, 0);

%Multiplication of the 4 individual matrix to form the A
matrices
A_matrix = A_1*A_2*A_3*A_4;

x = simplify(A_matrix(1, 4))
y = simplify(A_matrix(2, 4))
z = simplify(A_matrix(3, 4))

%Expand the trigonometric functions
expand(x)
expand(y)
```

Appendix B: Source Code for Simulation of Robot Motion with Data from Inverse Kinematics and Image Processing

```
rgb = imread('Image2.jpg');
rgb1 = rgb+50;
rgb_imadjust = imadjust(rgb1,[0.1 0.7],[]);
hsv = rgb2hsv(rgb_imadjust);
%h = round(hsv(:,:,1)*360);
h = hsv(:,:,1);
s = hsv(:,:,2);
v = hsv(:,:,3);

red = h > 0.95 ;
%imshow (red);
red_stats = regionprops(red,'Area','Centroid','Orientation')
red_stats([red_stats.Area]<1000)=[];
First_red = [red_stats(1).Centroid(1) red_stats(1).Centroid(2)
red_stats(1).Orientation]

green = h > 0.35 & h < 0.55   ;
%imshow (green);
green_stats =
regionprops(green,'Area','Centroid','Orientation')
green_stats([green_stats.Area]<1000)=[];
First_green = [green_stats(1).Centroid(1)
green_stats(1).Centroid(2) green_stats(1).Orientation]

blue = h > 0.6 & h < 0.65 & v > 0.7  ;
%imshow (blue);
blue_stats = regionprops(blue,'Area','Centroid','Orientation')
blue_stats([blue_stats.Area]<1000)=[];
First_blue = [blue_stats(1).Centroid(1)
blue_stats(1).Centroid(2) blue_stats(1).Orientation]

yellow = h > 0.1 & h < 0.2  ;
%imshow (yellow);
yellow_stats =
regionprops(yellow,'Area','Centroid','Orientation')
yellow_stats([yellow_stats.Area]<1000)=[];
First_yellow = [yellow_stats(1).Centroid(1)
yellow_stats(1).Centroid(2) yellow_stats(1).Orientation]

%Obtain the x and y centroid coordinates for each of the
coloured bars
x_centroid_red = red_stats(1).Centroid(2)
y_centroid_red = red_stats(1).Centroid(1)
x_centroid_green = green_stats(1).Centroid(2)
y_centroid_green = green_stats(1).Centroid(1)
```

```
x_centroid_blue = blue_stats(1).Centroid(2)
y_centroid_blue = blue_stats(1).Centroid(1)
x_centroid_yellow = yellow_stats(1).Centroid(2)
y_centroid_yellow = yellow_stats(1).Centroid(1)

%Display all the locations of the 4 different coloured bar
fprintf('x-coordinates for red bar: %f \n',x_centroid_red)
fprintf('y-coordinates for red bar: %f \n',y_centroid_red)
fprintf('x-coordinates for green bar: %f \n',x_centroid_green)
fprintf('y-coordinates for green bar: %f \n',y_centroid_green)
fprintf('x-coordinates for blue bar: %f \n',x_centroid_blue)
fprintf('y-coordinates for blue bar: %f \n',y_centroid_blue)
fprintf('x-coordinates for yellow bar: %f
\n',x_centroid_yellow)
fprintf('y-coordinates for yellow bar: %f
\n',y_centroid_yellow)

L=0.05;                 % cube size (length of an edge)
%---------------------------------------------------------------
-------------
%Drawing Red bars
xc_red = (x_centroid_red/1000)+0.2;
yc_red = (y_centroid_red/1000)-0.32;
zc_red = -0.2;     % coordinated of the center
alpha=0.8;            % transparency (max=1=opaque)

X_red = [0 0 0 0 0 1; 1 0 1 1 1 1; 1 0 1 1 1 1; 0 0 0 0 0 1];
Y_red = [0 0 0 0 1 0; 0 1 0 0 1 1; 0 1 1 1 1 1; 0 0 1 1 1 0];
Z_red = [0 0 1 0 0 0; 0 0 1 0 0 0; 1 1 1 0 1 1; 1 1 1 0 1 1];

C_red='red';                  % red

X_red = L*(X_red-1) + xc_red;
Y_red = L*(Y_red) + yc_red;
Z_red = L*(Z_red-0.5) + zc_red;
%---------------------------------------------------------------
-------------


%---------------------------------------------------------------
-------------
%Drawing Green bars
xc_green = x_centroid_green/1000+0.2;
yc_green = y_centroid_green/1000-0.32;
zc_green = -0.2;     % coordinated of the center
alpha=0.8;            % transparency (max=1=opaque)
```

```
X_green = [0 0 0 0 0 1; 1 0 1 1 1 1; 1 0 1 1 1 1; 0 0 0 0 0
1];
Y_green = [0 0 0 0 1 0; 0 1 0 0 1 1; 0 1 1 1 1 1; 0 0 1 1 1
0];
Z_green = [0 0 1 0 0 0; 0 0 1 0 0 0; 1 1 1 0 1 1; 1 1 1 0 1
1];

C_green='green';                    % green

X_green = L*(X_green-1) + xc_green;
Y_green = L*(Y_green) + yc_green;
Z_green = L*(Z_green-0.5) + zc_green;
%---------------------------------------------------------------
-------------


%---------------------------------------------------------------
-------------
%Drawing Blue bars
xc_blue = x_centroid_blue/1000+0.2;
yc_blue = y_centroid_blue/1000-0.32;
zc_blue = -0.2;     % coordinated of the center
alpha=0.8;            % transparency (max=1=opaque)

X_blue = [0 0 0 0 0 1; 1 0 1 1 1 1; 1 0 1 1 1 1; 0 0 0 0 0 1];
Y_blue = [0 0 0 0 1 0; 0 1 0 0 1 1; 0 1 1 1 1 1; 0 0 1 1 1 0];
Z_blue = [0 0 1 0 0 0; 0 0 1 0 0 0; 1 1 1 0 1 1; 1 1 1 0 1 1];

C_blue='blue';                      % blue

X_blue = L*(X_blue-1) + xc_blue;
Y_blue = L*(Y_blue) + yc_blue;
Z_blue = L*(Z_blue-0.5) + zc_blue;
%---------------------------------------------------------------
-------------


%---------------------------------------------------------------
-------------
%Drawing Yellow bars
xc_yellow = x_centroid_yellow/1000+0.2;
yc_yellow = y_centroid_yellow/1000-0.32;
zc_yellow = -0.2;     % coordinated of the center
alpha=0.8;            % transparency (max=1=opaque)

X_yellow = [0 0 0 0 0 1; 1 0 1 1 1 1; 1 0 1 1 1 1; 0 0 0 0 0
1];
```

```
Y_yellow = [0 0 0 0 1 0; 0 1 0 0 1 1; 0 1 1 1 1 1; 0 0 1 1 1
0];
Z_yellow = [0 0 1 0 0 0; 0 0 1 0 0 0; 1 1 1 0 1 1; 1 1 1 0 1
1];

C_yellow='yellow';                 % yellow

X_yellow = L*(X_yellow-1) + xc_yellow;
Y_yellow = L*(Y_yellow) + yc_yellow;
Z_yellow = L*(Z_yellow-0.5) + zc_yellow;
%------------------------------------------------------------
-------------

L1=Link([0 0.1916 0.4 0]);
L2=Link([0 0 0.25 pi]);
L3=Link([0 0.18 0 0 1]);
L4=Link([0 0.05 0 0]);

L3.qlim = [0 0.18]
zlim([0 1])
robot=SerialLink([L1 L2 L3 L4])

%Plot the bar object on the same plot as the robot
fill3(X_red,Y_red,Z_red,C_red,'FaceAlpha',alpha);
hold on;
fill3(X_green,Y_green,Z_green,C_green,'FaceAlpha',alpha)
hold on;
fill3(X_blue,Y_blue,Z_blue,C_blue,'FaceAlpha',alpha)
hold on;
fill3(X_yellow,Y_yellow,Z_yellow,C_yellow,'FaceAlpha',alpha)

%Inverse Kinematics Solution (for each coloured bar)
%For red bar
theta2_red =
acos(((x_centroid_red/1000+0.2)^2+(y_centroid_red/1000-
0.32)^2-0.285)/0.2);

a_red = ((y_centroid_red/1000-
0.32)*(0.4+(0.25*cos(theta2_red))))-
(0.25*(x_centroid_red/1000+0.2)*sin(theta2_red));
b_red =
((x_centroid_red/1000+0.2)*(0.4+(0.25*cos(theta2_red))))+(0.25
*(y_centroid_red/1000-0.32)*sin(theta2_red));

theta1_red = atan(a_red/b_red)

%For green bar
```

```
theta2_green = 
acos((((x_centroid_green/1000+0.2)^2+(y_centroid_green/1000-
0.32)^2-0.285)/0.2);

a_green = ((y_centroid_green/1000-
0.32)*(0.4+(0.25*cos(theta2_green))))-
(0.25*(x_centroid_green/1000+0.2)*sin(theta2_green));
b_green = 
((x_centroid_green/1000+0.2)*(0.4+(0.25*cos(theta2_green))))+(
0.25*(y_centroid_green/1000-0.32)*sin(theta2_green));

theta1_green = atan(a_green/b_green)

%For blue bar
theta2_blue = 
acos((((x_centroid_blue/1000+0.2)^2+(y_centroid_blue/1000-
0.32)^2-0.285)/0.2);

a_blue = ((y_centroid_blue/1000-
0.32)*(0.4+(0.25*cos(theta2_blue))))-
(0.25*(x_centroid_blue/1000+0.2)*sin(theta2_blue));
b_blue = 
((x_centroid_blue/1000+0.2)*(0.4+(0.25*cos(theta2_blue))))+(0.
25*(y_centroid_blue/1000-0.32)*sin(theta2_blue));

theta1_blue = atan(a_blue/b_blue)

%For yellow bar
theta2_yellow = 
acos((((x_centroid_yellow/1000+0.2)^2+(y_centroid_yellow/1000-
0.32)^2-0.285)/0.2);

a_yellow = ((y_centroid_yellow/1000-
0.32)*(0.4+(0.25*cos(theta2_yellow))))-
(0.25*(x_centroid_yellow/1000+0.2)*sin(theta2_yellow));
b_yellow = 
((x_centroid_yellow/1000+0.2)*(0.4+(0.25*cos(theta2_yellow))))
+(0.25*(y_centroid_yellow/1000-0.32)*sin(theta2_yellow));

theta1_yellow = atan(a_yellow/b_yellow)
%-------------------------------------------------------------
-------------
for a1=0:0.001:0.05
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([0 0 0 a1])
end
```

```
%1st iteration - Moving to red bar
for a=0:-0.05:theta1_red
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([a 0 0 0.05])
end

for b=0:0.1:theta2_red
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([theta1_red b 0 0.05])
end

for c=0:0.02:0.18
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([theta1_red theta2_red c 0.05])
end
%--------------------------------------------------------------
-------------
%2nd iteration - Moving to green bar
for a=0:-0.05:theta1_green
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([a 0 0 0.05])
end

for b=0:0.1:theta2_green
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([theta1_green b 0 0.05])
end

for c=0:0.02:0.18
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([theta1_green theta2_green c 0.05])
end
%--------------------------------------------------------------
-------------
%3rd iteration - Moving to blue bar
```

```
for a=0:-0.05:theta1_blue
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([a 0 0 0.05])
end

for b=0:0.1:theta2_blue
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([theta1_blue b 0 0.05])
end

for c=0:0.02:0.18
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([theta1_blue theta2_blue c 0.05])
end
%---------------------------------------------------------------
-------------
%4th iteration - Moving to yellow bar
for a=0:-0.05:theta1_yellow
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([a 0 0 0.05])
end

for b=0:0.1:theta2_yellow
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([theta1_yellow b 0 0.05])
end

for c=0:0.02:0.18
    xlim([-0.75 0.75])
    ylim([-0.75 0.75])
    zlim([-0.2 1])
    robot.plot([theta1_yellow theta2_yellow c 0.05])
end
%---------------------------------------------------------------
-------------
```

Appendix C: Source Code for the Image Processing Algorithm

```
rgb = imread('Image2.jpg');
rgb1 = rgb+50;
rgb_imadjust = imadjust(rgb1,[0.1 0.7],[]);
hsv = rgb2hsv(rgb_imadjust);
%h = round(hsv(:,:,1)*360);
h = hsv(:,:,1);
s = hsv(:,:,2);
v = hsv(:,:,3);

red = h > 0.95 ;
%imshow (red);
red_stats = regionprops(red,'Area','Centroid','Orientation')
red_stats([red_stats.Area]<1000)=[];
First_red = [red_stats(1).Centroid(1) red_stats(1).Centroid(2)
red_stats(1).Orientation]

green = h > 0.35 & h < 0.55   ;
%imshow (green);
green_stats =
regionprops(green,'Area','Centroid','Orientation')
green_stats([green_stats.Area]<1000)=[];
First_green = [green_stats(1).Centroid(1)
green_stats(1).Centroid(2) green_stats(1).Orientation]

blue = h > 0.6 & h < 0.65 & v > 0.7  ;
%imshow (blue);
blue_stats = regionprops(blue,'Area','Centroid','Orientation')
blue_stats([blue_stats.Area]<1000)=[];
First_blue = [blue_stats(1).Centroid(1)
blue_stats(1).Centroid(2) blue_stats(1).Orientation]

yellow = h > 0.1 & h < 0.2  ;
%imshow (yellow);
yellow_stats =
regionprops(yellow,'Area','Centroid','Orientation')
yellow_stats([yellow_stats.Area]<1000)=[];
First_yellow = [yellow_stats(1).Centroid(1)
yellow_stats(1).Centroid(2) yellow_stats(1).Orientation]

black = h > 0.5 & v > 0.15 & v < 0.55 ;
%imshow (black);
black_stats =
regionprops(black,'Area','Centroid','Orientation')
black_stats([black_stats.Area]<1000)=[];
First_black = [black_stats(1).Centroid(1)
black_stats(1).Centroid(2) black_stats(1).Orientation]
```

Appendix D: Source Code for Sending Coordinates Location Data from Image Processing Program to Robot Studio RAPID Program

```
X1 = First_red(2) +200;
Y1 = First_red(1) -320 ;
R1 = First_red(3) +90 ;

Xred = num2str(X1);
Yred = num2str(Y1);
Rred = num2str(R1);

X2 = First_yellow(2) +200;
Y2 = First_yellow(1) -320;
R2 = First_yellow(3) +90;

Xyellow = num2str(X2);
Yyellow = num2str(Y2);
Ryellow = num2str(R2);

X3 = First_blue(2) +200;
Y3 = First_blue(1) -320;
R3 = First_blue(3) +90;

Xblue = num2str(X3);
Yblue = num2str(Y3);
Rblue = num2str(R3);

X4 = First_green(2) +200;
Y4 = First_green(1) -320;
R4 = First_green(3) +90;

Xgreen = num2str(X4);
Ygreen = num2str(Y4);
Rgreen = num2str(R4);

con=pnet('tcpconnect','127.0.0.1',1025);
pnet(con,'setreadtimeout',10);
pnet(con,'printf',Xred);
pnet(con,'readline')
pnet(con,'printf',Yred);
pnet(con,'readline')
pnet(con,'printf',Rred);
pnet(con,'readline')
pnet(con,'printf',Xyellow);
pnet(con,'readline')
pnet(con,'printf',Yyellow);
pnet(con,'readline')
pnet(con,'printf',Ryellow);
pnet(con,'readline')
pnet(con,'printf',Xblue);
pnet(con,'readline')
pnet(con,'printf',Yblue);
pnet(con,'readline')
pnet(con,'printf',Rblue);
pnet(con,'readline')
pnet(con,'printf',Xgreen);
pnet(con,'readline')
pnet(con,'printf',Ygreen);
pnet(con,'readline')
pnet(con,'printf',Rgreen);
pnet(con,'readline')
```

Appendix E: Source Code for the RAPID Program from ABB Robot Studio

```
MODULE Module1
    VAR socketdev server_socket;
    VAR socketdev client_socket;
    VAR num XA;
    VAR num YA;
    VAR num RA;

    VAR num XB;
    VAR num YB;
    VAR num RB;

    VAR num XC;
    VAR num YC;
    VAR num RC;

    VAR num XD;
    VAR num YD;
    VAR num RD;

    VAR string colorString;
    VAR robtarget BendPose:=[[500.941680868,0,100],[0,0.999768036,0.02153772,0],[-
1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    ! pick up target
    VAR robtarget PickupTargetLift1:=[[100,451,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget
PickupTarget1:=[[275,451,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    VAR robtarget PickupTargetLift2:=[[275,451,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget
PickupTarget2:=[[275,451,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    VAR robtarget PickupTargetLift3:=[[275,451,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget
PickupTarget3:=[[275,451,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    VAR robtarget PickupTargetLift4:=[[275,451,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget
PickupTarget4:=[[275,451,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

    ! Destination Target
    VAR robtarget DestiantionTargetLift1:=[[55,451,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

```
    VAR robtarget
DestiantionTarget1:=[[55,451,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
;

    VAR robtarget DestiantionTargetLift2:=[[80,400,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget
DestiantionTarget2:=[[80,400,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
;

    VAR robtarget DestiantionTargetLift3:=[[105,451,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget
DestiantionTarget3:=[[105,451,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]
];

    VAR robtarget DestiantionTargetLift4:=[[80,502,100],[0,1,-
0.00000003,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    VAR robtarget
DestiantionTarget4:=[[80,502,25],[0,1,0,0],[0,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]]
;

    VAR pose object1;
    VAR pose object2;
    VAR pose object3;
    VAR pose object4;

    PROC main()
        socketTest;
        RepositionObject;

        Path_20;
        Path_40;
        Path_10;
        Path_30;
        !Add your code here
    ENDPROC
            PROC Path_10()
        PickupTargetLift1.trans:=([XA,YA,118]);
        PickupTargetLift1.rot:=OrientZYX(RA,180,0);
        PickupTarget1.trans:=([XA,YA,25]);
        PickupTarget1.rot:=OrientZYX(RA,180,0);
        DestiantionTargetLift1.rot:=OrientZYX(90,180,0);
        DestiantionTarget1.rot:=OrientZYX(90,180,0);
        MoveL BendPose,v300,fine,MyGripper\WObj:=wobj0;
        MoveL PickupTargetLift1,v300,fine,MyGripper\WObj:=wobj0;
        MoveL PickupTarget1,v300,fine,MyGripper\WObj:=wobj0;
        WaitTime 0.5;
                Set GripSet;
        WaitTime 0.5;
    MoveL PickupTargetLift1,v300,fine,MyGripper\WObj:=wobj0;
    MoveL BendPose,v300,fine,MyGripper\WObj:=wobj0;
    MoveL DestiantionTargetLift1,v300,fine,MyGripper\WObj:=wobj0;
    MoveL DestiantionTarget1,v300,fine,MyGripper\WObj:=wobj0;
    WaitTime 0.5;
            Reset GripSet;
    WaitTime 0.5;
    MoveL DestiantionTargetLift1,v300,fine,MyGripper\WObj:=wobj0;
        ENDPROC

PROC Path_20()
    PickupTargetLift2.trans:=([XB,YB,100]);
    PickupTargetLift2.rot:=OrientZYX(RB,180,0);
    PickupTarget2.trans:=([XB,YB,25]);
    PickupTarget2.rot:=OrientZYX(RB,180,0);
    DestiantionTargetLift2.rot:=OrientZYX(0,180,0);
    DestiantionTarget2.rot:=OrientZYX(0,180,0);
    MoveL BendPose,v300,fine,MyGripper\WObj:=wobj0;
    MoveL PickupTargetLift2,v300,fine,MyGripper\WObj:=wobj0;
    MoveL PickupTarget2,v300,fine,MyGripper\WObj:=wobj0;
    WaitTime 0.5;
            Set GripSet;
    WaitTime 0.5;
    MoveL PickupTargetLift2,v300,fine,MyGripper\WObj:=wobj0;
    MoveL BendPose,v300,fine,MyGripper\WObj:=wobj0;
    MoveL DestiantionTargetLift2,v300,fine,MyGripper\WObj:=wobj0;
    MoveL DestiantionTarget2,v300,fine,MyGripper\WObj:=wobj0;
    WaitTime 0.5;
            Reset GripSet;
    WaitTime 0.5;
    MoveL DestiantionTargetLift2,v300,fine,MyGripper\WObj:=wobj0;
ENDPROC

PROC Path_30()
    PickupTargetLift3.trans:=([XC,YC,100]);
    PickupTargetLift3.rot:=OrientZYX(RC,180,0);
    PickupTarget3.trans:=([XC,YC,25]);
    PickupTarget3.rot:=OrientZYX(RC,180,0);
    DestiantionTargetLift3.rot:=OrientZYX(90,180,0);
    DestiantionTarget3.rot:=OrientZYX(90,180,0);
    MoveL BendPose,v300,fine,MyGripper\WObj:=wobj0;
    MoveL PickupTargetLift3,v300,fine,MyGripper\WObj:=wobj0;
    MoveL PickupTarget3,v300,fine,MyGripper\WObj:=wobj0;
    WaitTime 0.5;
            Set GripSet;
    WaitTime 0.5;
    MoveL PickupTargetLift3,v300,fine,MyGripper\WObj:=wobj0;
    MoveL BendPose,v300,fine,MyGripper\WObj:=wobj0;
    MoveL DestiantionTargetLift3,v300,fine,MyGripper\WObj:=wobj0;
    MoveL DestiantionTarget3,v300,fine,MyGripper\WObj:=wobj0;
    WaitTime 0.5;
        Reset GripSet;
    WaitTime 0.5;
    MoveL DestiantionTargetLift3,v300,fine,MyGripper\WObj:=wobj0;
ENDPROC

PROC Path_40()
    PickupTargetLift4.trans:=([XD,YD,100]);
    PickupTargetLift4.rot:=OrientZYX(RD,180,0);
    PickupTarget4.trans:=([XD,YD,25]);
    PickupTarget4.rot:=OrientZYX(RD,180,0);
    DestiantionTargetLift4.rot:=OrientZYX(0,180,0);
    DestiantionTarget4.rot:=OrientZYX(0,180,0);
    MoveL BendPose,v300,fine,MyGripper\WObj:=wobj0;
    MoveL PickupTargetLift4,v300,fine,MyGripper\WObj:=wobj0;
    MoveL PickupTarget4,v300,fine,MyGripper\WObj:=wobj0;
    WaitTime 0.5;
        Set GripSet;
    WaitTime 0.5;
    MoveL PickupTargetLift4,v300,fine,MyGripper\WObj:=wobj0;
    MoveL BendPose,v300,fine,MyGripper\WObj:=wobj0;
    MoveL DestiantionTargetLift4,v300,fine,MyGripper\WObj:=wobj0;
    MoveL DestiantionTarget4,v300,fine,MyGripper\WObj:=wobj0;
    WaitTime 0.5;
        Reset GripSet;
    WaitTime 0.5;
    MoveL DestiantionTargetLift4,v300,fine,MyGripper\WObj:=wobj0;
ENDPROC

PROC RepositionObject()
    object1.trans:=([XA,YA,0]);
    object1.rot:=OrientZYX(RA,0,0);
    object2.trans:=([XB,YB,0]);
    object2.rot:=OrientZYX(RB,0,0);
    object3.trans:=([XC,YC,0]);
    object3.rot:=OrientZYX(RC,0,0);
    object4.trans:=([XD,YD,0]);
    object4.rot:=OrientZYX(RD,0,0);
    PulseDO UpdateObject;
ENDPROC

PROC socketTest()
    VAR string receive_string;
    VAR bool status;

    SocketClose server_socket;
    SocketClose client_socket;
    SocketCreate server_socket;
    SocketBind server_socket, "127.0.0.1", 1025;
    SocketListen server_socket;
    SocketAccept server_socket, client_socket, \Time:=100;

    !receive coordination for object 1
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, XA);
    SocketSend client_socket \Str := "Received X1: "+receive_string + "\0A";
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, YA);
    SocketSend client_socket \Str := "Received Y1: "+receive_string + "\0A";
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, RA);
    SocketSend client_socket \Str := "Received R1: "+receive_string + "\0A";

    !receive coordination for object 2
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, XB);
    SocketSend client_socket \Str := "Received X2: "+receive_string + "\0A";
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, YB);
    SocketSend client_socket \Str := "Received Y2: "+receive_string + "\0A";
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, RB);
    SocketSend client_socket \Str := "Received R2: "+receive_string + "\0A";

    !receive coordination for object 3
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, XC);
    SocketSend client_socket \Str := "Received X3: "+receive_string + "\0A";
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, YC);
    SocketSend client_socket \Str := "Received Y3: "+receive_string + "\0A";
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, RC);
    SocketSend client_socket \Str := "Received R3: "+receive_string + "\0A";

    !receive coordination for object 4
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, XD);
    SocketSend client_socket \Str := "Received X4: "+receive_string + "\0A";
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, YD);
    SocketSend client_socket \Str := "Received Y4: "+receive_string + "\0A";
    SocketReceive client_socket \Str := receive_string;
    status:=StrToVal(receive_string, RD);
    SocketSend client_socket \Str := "Received R4: "+receive_string + "\0A";

    SocketClose server_socket;
    SocketClose client_socket;

ENDPROC
```

```
ENDMODULE
```

Appendix F: Source Code for sending coordination from MATLAB

```
X1 = First_red(2) +200;
Y1 = First_red(1) -320 ;
R1 = First_red(3) +90 ;
Xred = num2str(X1);
Yred = num2str(Y1);
Rred = num2str(R1);

X2 = First_yellow(2) +200;
Y2 = First_yellow(1) -320;
R2 = First_yellow(3) +90;
Xyellow = num2str(X2);
Yyellow = num2str(Y2);
Ryellow = num2str(R2);

X3 = First_blue(2) +200;
Y3 = First_blue(1) -320;
R3 = First_blue(3) +90;
Xblue = num2str(X3);
Yblue = num2str(Y3);
Rblue = num2str(R3);

X4 = First_green(2) +200;
Y4 = First_green(1) -320;
R4 = First_green(3) +90;
Xgreen = num2str(X4);
Ygreen = num2str(Y4);
Rgreen = num2str(R4);

con=pnet('tcpconnect','127.0.0.1',1025);
pnet(con,'setreadtimeout',10);
pnet(con,'printf',Xred);
pnet(con,'readline')
pnet(con,'printf',Yred);
pnet(con,'readline')
pnet(con,'printf',Rred);
pnet(con,'readline')
pnet(con,'printf',Xyellow);
pnet(con,'readline')
pnet(con,'printf',Yyellow);
pnet(con,'readline')
pnet(con,'printf',Ryellow);
pnet(con,'readline')
pnet(con,'printf',Xblue);
pnet(con,'readline')
pnet(con,'printf',Yblue);
pnet(con,'readline')
pnet(con,'printf',Rblue);
pnet(con,'readline')
pnet(con,'printf',Xgreen);
pnet(con,'readline')
pnet(con,'printf',Ygreen);
pnet(con,'readline')
pnet(con,'printf',Rgreen);
pnet(con,'readline')
```

Appendix G: Dimension of the SCARA Robot model

Dimensions

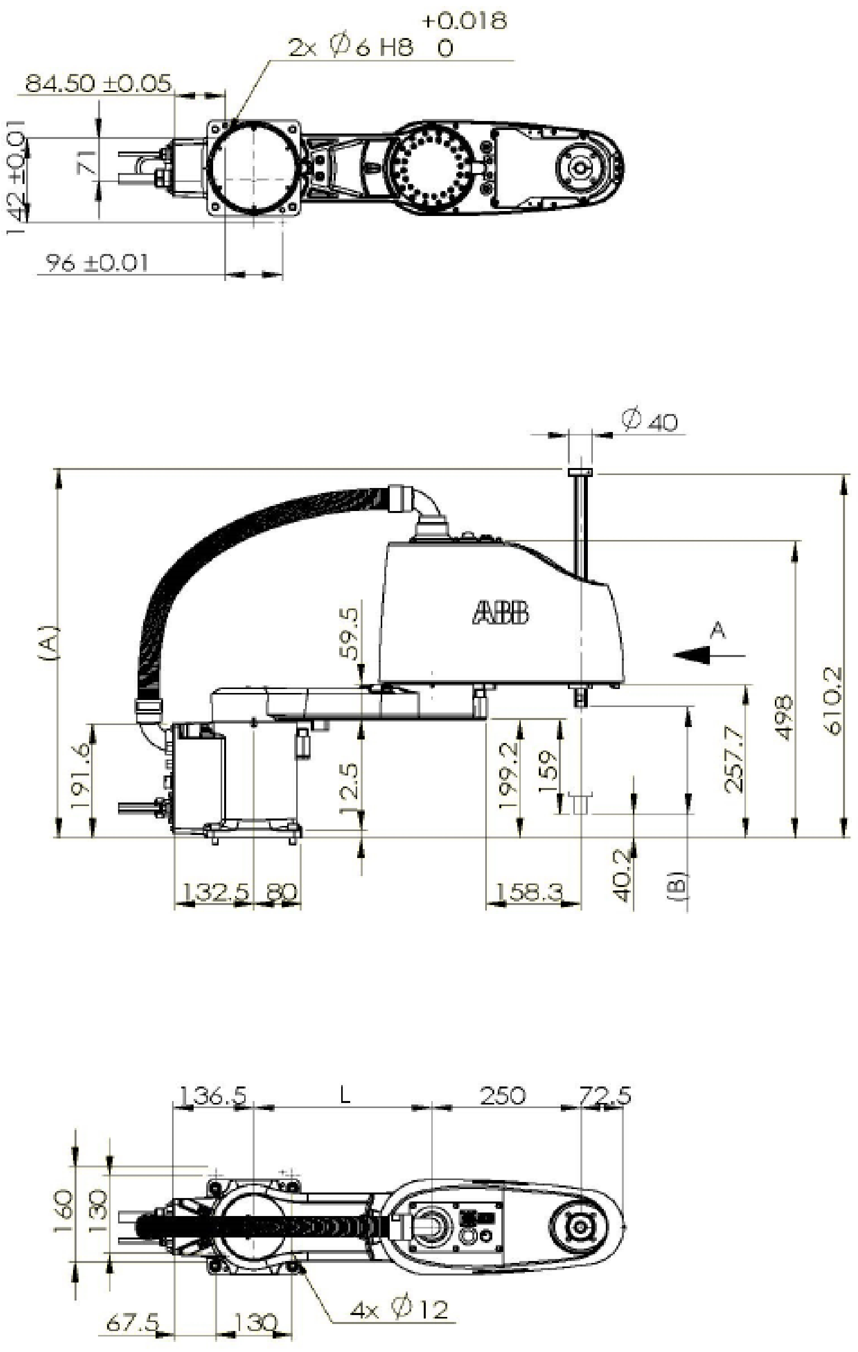


Figure 49: Dimension of the SCARA robot model in the manual