Robot System Design

Hand-Eye Coordination Project Report

Table of Contents

1. Introduction

Creating a vision-guided pick-and-place operation that uses an industrial robotic manipulator is the objective of this project. **Image acquisition and processing**

Apply image processing techniques and implement the algorithm in MATLAB to identify the objects properties in the robot workspace and send this information to Robot Studio.

Module interfacing and robot arm programming
Perform a basic vision-guided pick-and-place operation with at least four coloured items, then write a program to direct the robot's mobility based on the data collected and communicate with the image recognition software.

MATLAB Robotics Toolbox animation and inverse kinematics solution
Solve the robot's inverse kinematic equations and code the inverse kinematic solution in MATLAB, then use the MATLAB Robotics Toolbox to simulate the robot's movements.

Image Acquisition and Processing

For Image Acquisition and Processing we used MATLAB. We installed the Image Processing Toolbox to Apply image processing techniques. We first read the image into MATLAB.

From the image we:

1. Applied colour filtering
2. Object identification
3. Obtaining Object orientation and coordinates.

The matlab code was summerised into:

Main.m and a function **findCoordinate.m** The

code in **Main.m** is:

```
close;
clear; clc;

yellowThresholds = [45.493, 100.000, -22.572, 55.141, 30.885, 69.530];
redThresholds = [0.000, 100.000, 30.258, 55.141, -21.106, 69.530];
blueThresholds = [0.000, 31.027, -22.572, 55.141, -63.035, -18.101];
greenThresholds = [0.000, 35.430, -22.572, -5.655, 1.116, 69.530];
```

```
blackThresholds = [0.000, 33.753, -3.285, 10.697, -10.624, 2.024]; %These
are pre-calculated values, that are input in the function to help
%analyse the image by filtering/separating it by colour

x = imread('Image1.jpg');
con=pnet('tcpconnect','127.0.0.1',1025);
%to connect to RobotStudio

%to get the yellow boxes
[temp, temp1] = findCoordinate(x, yellowThresholds);
for i = 1:temp1     temp2 = temp(i).Centroid;
pnet(con,'setreadtimeout',5);
pnet(con,'printf',int2str(temp2(1)));
pnet(con,'readline')
    pnet(con,'printf',int2str(temp2(2)));
pnet(con,'readline')
    pnet(con,'printf',int2str(temp(i).Orientation));
pnet(con,'readline') end

%to get the red boxes
[temp, temp1] = findCoordinate(x, redThresholds);
for i = 1:temp1     temp2 = temp(i).Centroid;
pnet(con,'setreadtimeout',5);
pnet(con,'printf',int2str(temp2(1)));
pnet(con,'readline')
pnet(con,'printf',int2str(temp2(2)));
pnet(con,'readline')
    pnet(con,'printf',int2str(temp(i).Orientation));
pnet(con,'readline') end

%to get the blue boxes
[temp, temp1] = findCoordinate(x, blueThresholds);
for i = 1:temp1     temp2 = temp(i).Centroid;
pnet(con,'setreadtimeout',5);
pnet(con,'printf',int2str(temp2(1)));
pnet(con,'readline')
pnet(con,'printf',int2str(temp2(2)));
pnet(con,'readline')
    pnet(con,'printf',int2str(temp(i).Orientation));
pnet(con,'readline') end

%to get the green boxes
[temp, temp1] = findCoordinate(x, greenThresholds);
for i = 1:temp1     temp2 = temp(i).Centroid;
pnet(con,'setreadtimeout',5);
pnet(con,'printf',int2str(temp2(1)));
pnet(con,'readline')
pnet(con,'printf',int2str(temp2(2)));
pnet(con,'readline')
    pnet(con,'printf',int2str(temp(i).Orientation));
pnet(con,'readline') end

%to get the black boxes
[temp, temp1] = findCoordinate(x, blackThresholds);
for i = 1:temp1     temp2 = temp(i).Centroid;
pnet(con,'setreadtimeout',5);
```

```
pnet(con,'printf',int2str(temp2(1)));
pnet(con,'readline')
pnet(con,'printf',int2str(temp2(2)));
pnet(con,'readline')
    pnet(con,'printf',int2str(temp(i).Orientation));
pnet(con,'readline') end
```

The function code in **findCoordinate.m** is:

```
function [output, objCount] = findCoordinate(x, thresholdValue)
    % Convert RGB image to chosen color space
    I = rgb2lab(x);

    % Define thresholds for channel 1 based on histogram settings
channel1Min = thresholdValue(1);     channel1Max =
thresholdValue(2);

    % Define thresholds for channel 2 based on histogram settings
channel2Min = thresholdValue(3);     channel2Max =
thresholdValue(4);

    % Define thresholds for channel 3 based on histogram settings
channel3Min = thresholdValue(5);     channel3Max =
thresholdValue(6);

    % Create mask based on chosen histogram thresholds     sliderBW =
(I(:,:,1) >= channel1Min ) & (I(:,:,1) <= channel1Max) & ...
        (I(:,:,2) >= channel2Min ) & (I(:,:,2) <= channel2Max) & ...
        (I(:,:,3) >= channel3Min ) & (I(:,:,3) <= channel3Max);
    BW = sliderBW;

    cleanedImg = imfill(BW, 'holes');
cleanedImg = medfilt2(cleanedImg);
cleanedImg = bwareaopen(cleanedImg,50);

    output = regionprops(cleanedImg, 'Orientation', 'Centroid');

    label = bwlabel(cleanedImg);
objCount = max(max(label)); end
```

Module Interfacing and Robot Arm Programming

Using Robotstudio, the ABB IRB 910SC - 3/0.65 robot arm was simulated. The purpose of this simulation is to test a scenario in which coloured bars are littered around a work area and whether the robot is capable of locating them and arranging them to form letters and shapes.

This process can be broken down into 3 steps:

a) Connecting to MATLAB image processing code to obtain coordinates
b) Simulating the bars in Robot studio
c) Picking up the bars and placing them in a new position

a) Connecting to MATLAB

Using Robotstudio app, a server using RAPID was created to accept the MATLAB image processing code as a client. This will allow the coordinates generated from image processing to be transferred to Robotstudio where the bars can be simulated.

```
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;

    SocketReceive client_socket \Str := receive_string; !---------------------------------------!
status:=StrToVal(receive_string, Y1X);                                                          !
SocketSend client_socket \Str := "Received Y1X: "+receive_string + "\0A";                       !
SocketReceive client_socket \Str := receive_string;                                             !
status:=StrToVal(receive_string, Y1Y);                                                          !
    SocketSend client_socket \Str := "Received Y1Y: "+receive_string + "\0A";                   ! ------YELLOW 1--------
SocketReceive client_socket \Str := receive_string;                                             !     status:=StrToVal(receive_string,
Y1R);                                                                                           !
    SocketSend client_socket \Str := "Received Y1R: "+receive_string + "\0A";                   !
```

The snippet of code above shows the process of obtaining the coordinates and orientation of the first yellow bar and storing it in a variable for later use. The rest of the process follows this same principle to obtain the coordinates of all the other bars as well.

b) Simulating the bars

Using the coordinates obtained from part a, the bars were able to be simulated in the Robotstudio program with the following code:

```
PROC RepositionObject()
    yellow1.trans:=([Y1X/1.5+200,Y1Y/1.5-200,0]);
yellow1.rot:=OrientZYX(Y1R,0,0);

    yellow2.trans:=([Y2X/1.5+200,Y2Y/1.5-200,0]);
yellow2.rot:=OrientZYX(Y2R,0,0);

    yellow3.trans:=([Y3X/1.5+200,Y3Y/1.5-200,0]);
yellow3.rot:=OrientZYX(Y3R,0,0);

    red1.trans:=([R1X/1.5+200,R1Y/1.5-200,0]);
red1.rot:=OrientZYX(R1R,0,0);
```

The code snippet above shows the coordinates of the bars being used to place and orient them in robot studio. the rest of this process places the rest of the bars in a similar fashion.

The received coordinates obtained through image processing tend to be spread wide and only appear in one quadrant due to the way the coordinates are generated from the provided image.

The coordinates were shifted away from the origin of the workstation and the space between them reduced in order for the robot arm to reach them.

c) Picking and placing the bars

For this part of the program, two processes were created to simplify the task.

The first being the Locate process which moves and orients the arm to pick up the bar based on the coordinates obtained from part a.

```
PROC locate(num x,num y, num r)
    XYMOVE.trans:=([300,300,100]);      ! moves to  location to avoid passing through out of range area
    MoveL XYMOVE,v1000,fine,MyGripper\WObj:=wobj0;
    WaitTime 0.5;

    XYMOVE.trans:=([x,y,100]);
```

```
Reset GripSet;
MoveL XYMOVE,v1000,fine,MyGripper\WObj:=wobj0;
   ZMOVE.trans:=([x,y,25]);
   ZMOVE.rot:=OrientZYX(r,180,0);
   MoveL ZMOVE,v1000,fine,MyGripper\WObj:=wobj0;          ! grips object
WaitTime 0.5;
Set GripSet;
WaitTime 0.5;


   ZMOVE.trans:=([x,y,100]);
   MoveL ZMOVE,v1000,fine,MyGripper\WObj:=wobj0;          ! lifts object
WaitTime 0.5;
ENDPROC
```

The process above accepts the bars coordinates as input and from there handles the rest of the arms movement by gripping and lifting up the bar.

The second one is the Place function which moves the robot arm after it has lifted one of the bars and moves it to a new location that has already been determined in order to arrange the bars in shapes and letters.

```
PROC place(num x, num y, num r)
   XYMOVE.trans:=([300,300,100]);        ! moves to  location to avoid passing through out of range area


   MoveL XYMOVE,v1000,fine,MyGripper\WObj:=wobj0;
   WaitTime 0.5;


   XYMOVE.trans:=([x,y,100]);                    ! moves to new location
   MoveL XYMOVE,v1000,fine,MyGripper\WObj:=wobj0;


   ZMOVE.rot:=OrientZYX(r,180,0);
   ZMOVE.trans:=([x,y,25]);
   MoveL ZMOVE,v1000,fine,MyGripper\WObj:=wobj0;
   Reset GripSet;
   WaitTime 0.5;                                 ! sets down object and raises back up
   ZMOVE.trans:=([x,y,100]);
   MoveL ZMOVE,v1000,fine,MyGripper\WObj:=wobj0;
```

```
    WaitTime 0.5;
ENDPROC
```

The process above allows us to simply enter the desired coordinates for the object to be placed at and the code takes care of the rest by moving there and placing down the bar in its correct location and orientation.

MATLAB Robotics Toolbox animation and inverse kinematics solution

As the output of the image processing is going to be in a form of coordinate (x, y, z, Φ), inverse kinematics will be used to derive the angle required for the robot to move to the exact object location. To begin with, below are the known parameters according to the physical dimension of the robotic arm as it will be used to calculate the corresponding angle.

l_1	400mm
l_2	250mm
d_1	199.2
d_4	50

Table 3 _ Known Parameters

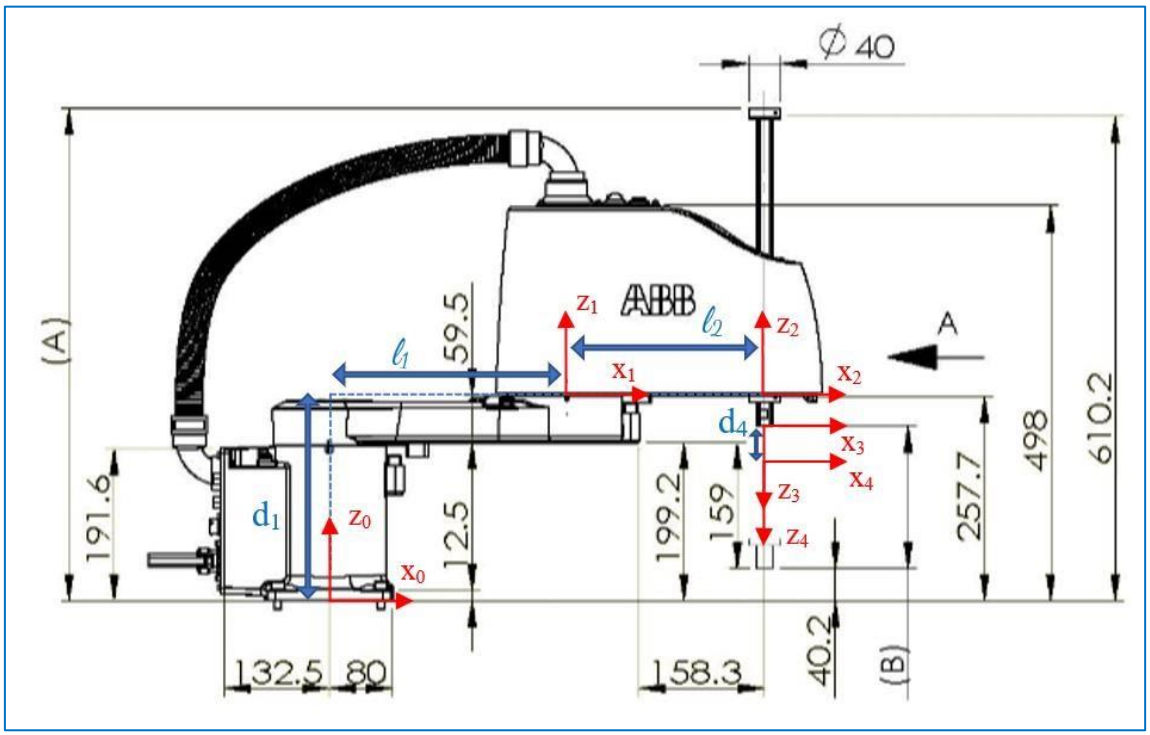

Figure 1 _ Robot Specification

Based on the specification of IRB 910SC – 3/0.65", the D-H parameters are as follows:

Axis Number	**θ**	**d**	**l**	**α**
1	θ_1	d_1	l_1	0
2	θ_2	0	l_2	π
3	0	d_3	0	0
4	θ_4	d_4	0	0

Table 4 – D-H Parameters

A Matrices

$$Rot(z,\theta)=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\qquad Trans(l,0,0)=\begin{bmatrix}1 & 0 & 0 & l\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$Trans(l,0,d)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & d\\ 0 & 0 & 0 & 1\end{bmatrix}\qquad Rot(x,\alpha)=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha & 0\\ 0 & \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$A_n=Rot(z,\theta)Trans(0,0,d)Trans(l,0,0)Rot(x,\alpha)$$

$$A_n=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & d\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & l\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha & 0\\ 0 & \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$A_n=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & 0\\ \sin\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & d\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & l\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha & 0\\ 0 & \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$A_n=\begin{bmatrix}\cos\theta & -\sin\theta & 0 & l\cos\theta\\ \sin\theta & \cos\theta & 0 & l\sin\theta\\ 0 & 0 & 1 & d\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\alpha & -\sin\alpha & 0\\ 0 & \sin\alpha & \cos\alpha & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$A_n=\begin{bmatrix}\cos\theta & -\sin\theta\cos\alpha & \sin\theta\sin\alpha & l\cos\theta\\ \sin\theta & \cos\theta\cos\alpha & -\cos\theta\sin\alpha & l\sin\theta\\ 0 & \sin\alpha & \cos\alpha & d\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$${}^{0}A_{1}=\begin{bmatrix}c_1 & -s_1 & 0 & l_1c_1\\ s_1 & c_1 & 0 & l_1s_1\\ 0 & 0 & 1 & d_1\\ 0 & 0 & 0 & 1\end{bmatrix}\qquad {}^{2}A_{3}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & d_3\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$
{}_1A_2 = \begin{bmatrix} c_2 & s_2 & 0 & l_2\,c_2 \\ s_2 & -c_2 & 0 & l_2\,s_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\qquad
{}_3A_4 = \begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Transform Matrix

$$
{}_R T_H = \begin{bmatrix} c_1 & -s_1 & 0 & l_1\,c_1 \\ s_1 & c_1 & 0 & l_1\,s_1 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} c_2 & s_2 & 0 & l_2\,c_2 \\ s_2 & -c_2 & 0 & l_2\,s_2 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} c_4 & -s_4 & 0 & 0 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{R}T_{H} =
\begin{matrix}
c_1c_2 - s_1s_2 & c_1s_2 + s_1c_2 & 0 & l_2c_1\,c_2 - l_2s_1s_2 + l_1\,c_1 \\
s_1c_2 + c_1s_2 & s_1s_2 - c_1c_2 & 0 & l_2s_1c_2 + l_2c_1s_2 + l_1\,s_1 \\
0 & 0 & -1 & d_1 \\
0 & 0 & 0 & 1
\end{matrix}
\begin{matrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & d_3 \\
0 & 0 & 0 & 1
\end{matrix}
\begin{matrix}
c_4 & -s_4 & 0 & 0 \\
s_4 & c_4 & 0 & 0 \\
0 & 0 & 1 & d_4 \\
0 & 0 & 0 & 1
\end{matrix}
$$

$$
{}^{R}T_{H} =
\begin{matrix}
c_1c_2 - s_1s_2 & c_1s_2 + s_1c_2 & 0 & l_2c_1\,c_2 - l_2s_1s_2 + l_1\,c_1 \\
s_1c_2 + c_1s_2 & s_1s_2 - c_1c_2 & 0 & l_2s_1c_2 + l_2c_1s_2 + l_1\,s_1 \\
0 & 0 & -1 & d_1 - d_3 \\
0 & 0 & 0 & 1
\end{matrix}
\begin{matrix}
c_4 & -s_4 & 0 & 0 \\
s_4 & c_4 & 0 & 0 \\
0 & 0 & 1 & d_4 \\
0 & 0 & 0 & 1
\end{matrix}
$$

$$
{}^{R}T_{H} =
\begin{matrix}
c_4(c_1c_2 - s_1s_2) + s_4(c_1s_2 + s_1c_2) & c_4(c_1s_2 + s_1c_2) - s_4(c_1c_2 - s_1s_2) & 0 & l_2c_1\,c_2 - l_2s_1s_2 + l_1\,c \\
c_4(s_1c_2 + c_1s_2) + s_4(s_1s_2 - c_1c_2) & c_4(s_1s_2 - c_1c_2) - s_4(s_1c_2 + c_1s_2) & 0 & l_2s_1c_2 + l_2c_1s_2 + l_1\,s \\
0 & 0 & -1 & d_1 - d_3 - d_4 \\
0 & 0 & 0 & 1
\end{matrix}
$$

$$
{}^{R}T_{H} =
\begin{matrix}
c_4(c_{12}) + s_4(s_{12}) & c_4(s_{12}) - s_4(c_{12}) & 0 & l_2c_{12} + l_1\,c_1 \\
c_4(s_{12}) - s_4(c_{12}) & c_4(c_{12}) - s_4(s_{12}) & 0 & l_2s_{12} + l_1\,s_1 \\
0 & 0 & -1 & d_1 - d_3 - d_4 \\
0 & 0 & 0 & 1
\end{matrix}
$$

$$P_x = l_2c_{12} + l_1\,c_1 \qquad P_y = l_2s_{12} + l_1\,s_1 \qquad P_z = d_1 - d_3 - d_4$$

$$P_{x2} + P_{y2} = (l_2c_1\,c_2 - l_2s_1s_2 + l_1\,c_1)_2 + (l_2s_1c_2 + l_2c_1s_2 + l_1\,s_1)_2$$

$$P_{x2} + P_{y2} = (c_1(l_2c_2 + l_1) - s_1(l_2s_2))_2 + (c_1(l_2s_2) + s_1(l_2c_2 + l_1))_2$$

$$P_{x2} + P_{y2} = l_{12} + l_{22} + 2l_1l_2C_2$$

$$C_2 = \frac{P_{x2} + P_{y2} - l_{12} - l_{22}}{2l_1l_2}$$

Cramer's rule

$$P_x = c_1(l_2c_2 + l_1) - s_1(l_2s_2)$$

$$P_y = c_1(l_2s_2) + s_1(l_2c_2 + l_1)$$

$$S_1 = \frac{\Delta S_1}{\Delta} \qquad C_1 = \frac{\Delta C_1}{\Delta}$$

Thus,

$$\theta_1 = \tan^{-1}\left(\frac{\Delta S_1}{\Delta C_1}\right) = \tan^{-1}\left(\frac{\Delta S_1}{\Delta C_1}\right)$$

Where:

2

$$\Delta C_1 = [P P_{xy}\, l_2 - c_2 l_2 + s_2 l_1] = P_x(l_2 c_2 + l_1) - P_y(-l_2 s_2)$$

$$\Delta s_1 = \begin{bmatrix} c_1(l_2 c_2 + l_1) & P_x \\ c_1(l_2 s_2) & P_y \end{bmatrix} = P_y(c_1(l_2 c_2 + l_1)) - P_x(c_1(l_2 s_2))$$

General Orientation Transform

$$RPY(\Phi, \theta, \psi) = Rot(z, \Phi) Rot(y, \theta) Rot(x, \psi)$$

$$= \begin{bmatrix} C(\Phi)C(\theta) & C(\Phi)S(\theta)S(\psi) - S(\Phi)C(\psi) & C(\Phi)S(\theta)C(\psi) + S(\Phi)S(\psi) & 0 \\ S(\Phi)C(\theta) & S(\Phi)S(\theta)S(\psi) + C(\Phi)C(\psi) & S(\Phi)S(\theta)C(\psi) - C(\Phi)S(\psi) & 0 \\ -S(\theta) & C(\theta)S(\psi) & C(\theta)C(\psi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R T_H = \begin{bmatrix} x_x & y_x & z_x & P_x \\ x_y & y_y & z_y & P_y \\ x_z & y_z & z_z & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R T_H = \begin{bmatrix} c_4(c_{12}) + s_4(s_{12}) & c_4(s_{12}) - s_4(c_{12}) & 0 & l_2 c_{12} + l_1 c_1 \\ c_4(s_{12}) - s_4(c_{12}) & c_4(c_{12}) - s_4(s_{12}) & 0 & l_2 s_{12} + l_1 s_1 \\ 0 & 0 & -1 & d_1 - d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\theta = -\sin^{-1}(x_z) = 0$$

$$^{-1}\left(\frac{\quad}{\quad} y^z\right) = 0$$

2

$$\psi = \sin \qquad \cos(\theta)$$

$$\Phi = \sin^{-1}\left(\_\_\_\_\_ x({}^{y}\theta)\right) = \sin^{-1}\left(c \_\_\_\_\_ {}^{4}(s^{12}) - 1\ s^{4}(c^{12})\right) = \sin^{-1}(- \sin(\theta_4 - (\theta_1 + \theta_2))) \cos$$

$$= \theta_1 + \theta_2 - \theta_4$$

Thus,

$\theta_4 = \theta_1 + \theta_2 - \Phi$

Where:

$$\theta_1 = \tan{-1}\left(\quad P_x(l_2c_2 + l_1) - P_y(-l_2s_2) \quad \right)$$

$$\overline{P_y(c_1(l_2c_2 + l_1)) - P_x(c_1(l_2s_2))}$$

$$\theta_2 = \cos{-1}\left(\frac{P_2 + P_2 - l_2 - l_2}{2l_1l_2} \right) \quad d_3 = d_1 - d_4 - P_z$$

With all parameters and calculations completed, the simulation can be created using MATLAB with Robotics Toolbox created by Peter Corke. Step by step explanation of the basic implementation of the system (with the assumed input as [400 -100 0 90]) is as follows:

1. Define what is the input of the system.

```
% Input:
inputC = [400 -100 0 90];
px = inputC(1);
py = inputC(2);
pz = inputC(3);
rollAngle = ((inputC(4))*pi)/180;
```

Figure 2 – MATLAB Simulation Step 1

2. Define all known parameters

```
% Robot known parameters according to the robot specification stated in the report (in mm)
l1 = 400;
l2 = 250;
d1 = 257.7;
d4 = 50;
```

Figure 3 – MATLAB Simulation Step 2

3. Define the inverse kinematics of the robot

```
% Inverse Kinematics
theta2 = acos((px^2 + py^2 - l1^2 - l2^2) / (2 * l1 * l2));
dC1 = det([px -(l2*sin(theta2)); py ((l2*cos(theta2))+l1)]);
dS1 = det([(l1+(l2*cos(theta2))) px; (l2*sin(theta2)) py]);
theta1 = atan(dS1/dC1);
theta4 = theta1 + theta2 - rollAngle;

d3 = d1 - d4 - pz;
```

Figure 4 _ MATLAB Simulation Step 3

4. Define the links

```
% Link Definition
L1 = Link([0 d1 l1 0]);
L2 = Link([0 0 l2 pi]);
L3 = Link([0 0 0 0 1]); %prismatic joint
L3.qlim = [0 257.7];
L4 = Link([0 d4 0 0]);
```

Figure 5 _ MATLAB Simulation Step 4

5. Create the robot model

```
% Robot Model
robot = SerialLink([L1 L2 L3 L4]);
```

Figure 6 – MATLAB Simulation Step 5

6. Animate the robot movement

```
% Position Plot
step = 50;
a1 = 0;
a2 = 0;
a3 = 0;
a4 = 0;
cArray = [0 0 0 0];

for a = 1:2
    for i = 1:step
        a1 = a1 + (theta1/step);
        a2 = a2 + (theta2/step);
        a3 = a3 + (d3/step);
        a4 = a4 + (theta4/step);
        cArray = [cArray; a1 a2 a3 a4];
    end
    for i = 1:step
        a1 = a1 - (theta1/step);
        a2 = a2 - (theta2/step);
        a3 = a3 - (d3/step);
        a4 = a4 - (theta4/step);
        cArray = [cArray; a1 a2 a3 a4];
    end
end
```

Figure 7 _ MATLAB Simulation Step 6

For the application of the simulation, the code has been adjusted to animate through a several stets of movements. The full code and demonstration can be found in the last section of the report.

Project Related Links

- **Video demonstration**
  https://drive.google.com/file/d/1eAj7Qm27YhvVyrr_fMhbDLaqw70HjsEc/view?usp =sharing