MCTE 4352

ROBOTICS

SEM 1, 2023/2024

MINI PROJECT

SECTION 2

NO.	**NAME**	**MATRIC NO**
1.	NUR BIDAYATUL HIDAYAH BINTI ABDUL KADER	2013180
2.	NUR AZSHIMADATHUL ASYQIN BINTI DARWIS	1910828

DATE OF SUBMISSION	**TOTAL MARK**

INTRODUCTION

This mini project is dedicated to the exploration of a 6-DOF (Degrees of Freedom) robotic arm, a sophisticated mechanism renowned for its versatility and precision in executing complex tasks. Our endeavour encompasses the derivation of fundamental representations crucial for understanding and controlling the robotic arm's movements.

The first facet of our investigation involves the derivation of the position representation of the 6-DOF robotic arm. This entails formulating kinematic equations that map joint angles to the precise spatial configuration of the end effector in three-dimensional space. The goal is to provide a clear and concise mathematical model that accurately describes the arm's spatial orientation.

Moving beyond positional representation, our project delves into the derivation of linear and angular velocity representations. Understanding the dynamic aspects of the robotic arm is essential for real-time control and dynamic task execution. By establishing mathematical relationships between joint velocities and end-effector motion, we aim to articulate expressions governing the arm's speed and direction.

To bridge theoretical insights with practical implementation, we leverage the trial version of RoboDK software. This simulation platform facilitates the modelling and visualization of robotic systems, offering a means to validate our derived representations. Integrating our position, linear, and angular velocity models into the software, we simulate the movement of the 6-DOF robotic arm as it executes a specific task. This simulation serves as a critical step in ensuring the viability and effectiveness of our theoretical derivations in a real-world context.

1. POSITION REPRESENTATION

The position representation in a 6-DOF (Degrees of Freedom) robotic arm is a crucial aspect that defines the spatial location of the arm's end effector in a three-dimensional coordinate system. It involves establishing a mathematical model that describes the relationship between the joint angles of the robotic arm and the resulting position of its end effector in the workspace. This representation is essential for tasks such as trajectory planning, control, and simulation, allowing us to precisely understand and manipulate the arm's movements.

1.1. FORWARD KINEMATIC

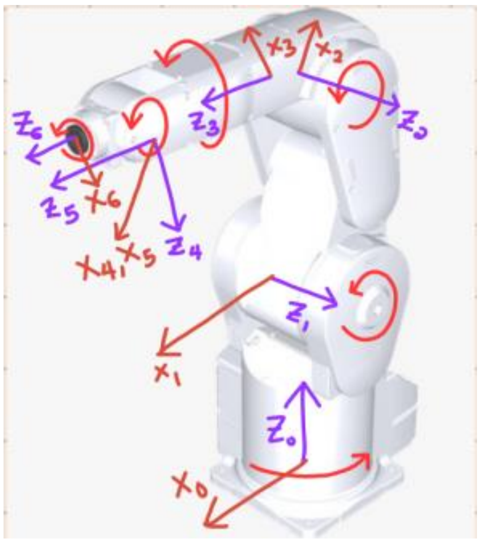


Figure 1 MELFA RV-7FR Frame Assignment

Denavit-Hartenberg (DH) representation was used to find the end effector of the robot. The frame assignment of the robot shown in figure 1 while the related parameters for DH table shown in Table 1.

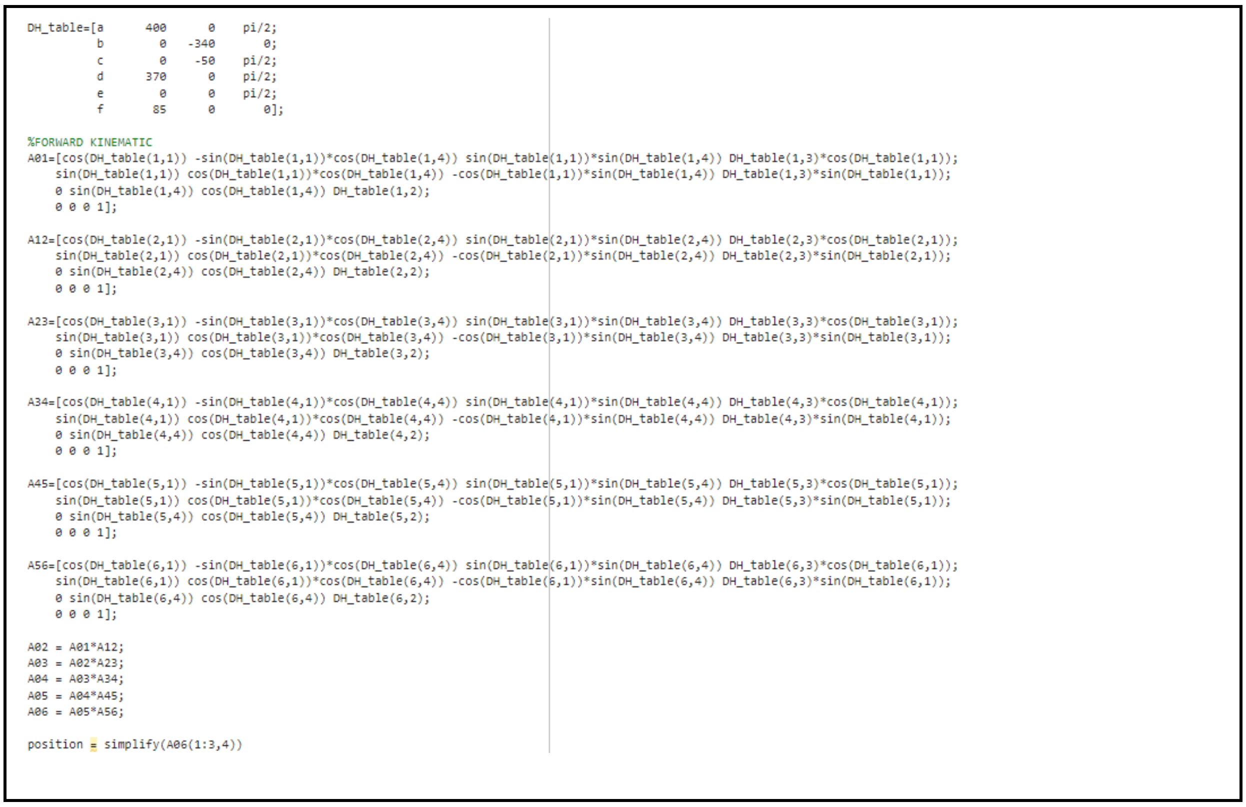
Table 1 DH-Representation

	α	a	θ_i	d
0-1	90°	0	θ_1	400
1-2	0	-340	θ_2	0
2-3	0	-50	θ_3	0
3-4	-90°	0	θ_4	370
4-5	90°	0	θ_5	0
5-6	0	0	θ_6	85

Forward Kinematics Matrices:

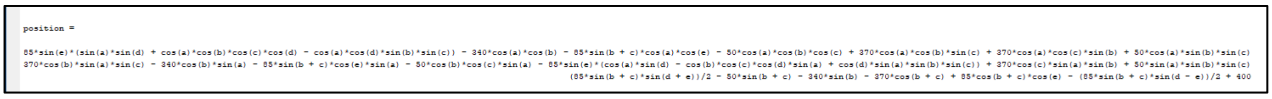
From calculation on MATLAB, we can get A01, A02, A03, A04, A05 and A06

MATLAB CODE:



```
DH_table=[a      400      0    pi/2;
          b        0   -340      0;
          c        0    -50    pi/2;
          d      370      0    pi/2;
          e        0      0    pi/2;
          f       85      0      0];

%FORWARD KINEMATIC
A01=[cos(DH_table(1,1)) -sin(DH_table(1,1))*cos(DH_table(1,4)) sin(DH_table(1,1))*sin(DH_table(1,4)) DH_table(1,3)*cos(DH_table(1,1));
    sin(DH_table(1,1)) cos(DH_table(1,1))*cos(DH_table(1,4)) -cos(DH_table(1,1))*sin(DH_table(1,4)) DH_table(1,3)*sin(DH_table(1,1));
    0 sin(DH_table(1,4)) cos(DH_table(1,4)) DH_table(1,2);
    0 0 0 1];

A12=[cos(DH_table(2,1)) -sin(DH_table(2,1))*cos(DH_table(2,4)) sin(DH_table(2,1))*sin(DH_table(2,4)) DH_table(2,3)*cos(DH_table(2,1));
    sin(DH_table(2,1)) cos(DH_table(2,1))*cos(DH_table(2,4)) -cos(DH_table(2,1))*sin(DH_table(2,4)) DH_table(2,3)*sin(DH_table(2,1));
    0 sin(DH_table(2,4)) cos(DH_table(2,4)) DH_table(2,2);
    0 0 0 1];

A23=[cos(DH_table(3,1)) -sin(DH_table(3,1))*cos(DH_table(3,4)) sin(DH_table(3,1))*sin(DH_table(3,4)) DH_table(3,3)*cos(DH_table(3,1));
    sin(DH_table(3,1)) cos(DH_table(3,1))*cos(DH_table(3,4)) -cos(DH_table(3,1))*sin(DH_table(3,4)) DH_table(3,3)*sin(DH_table(3,1));
    0 sin(DH_table(3,4)) cos(DH_table(3,4)) DH_table(3,2);
    0 0 0 1];

A34=[cos(DH_table(4,1)) -sin(DH_table(4,1))*cos(DH_table(4,4)) sin(DH_table(4,1))*sin(DH_table(4,4)) DH_table(4,3)*cos(DH_table(4,1));
    sin(DH_table(4,1)) cos(DH_table(4,1))*cos(DH_table(4,4)) -cos(DH_table(4,1))*sin(DH_table(4,4)) DH_table(4,3)*sin(DH_table(4,1));
    0 sin(DH_table(4,4)) cos(DH_table(4,4)) DH_table(4,2);
    0 0 0 1];

A45=[cos(DH_table(5,1)) -sin(DH_table(5,1))*cos(DH_table(5,4)) sin(DH_table(5,1))*sin(DH_table(5,4)) DH_table(5,3)*cos(DH_table(5,1));
    sin(DH_table(5,1)) cos(DH_table(5,1))*cos(DH_table(5,4)) -cos(DH_table(5,1))*sin(DH_table(5,4)) DH_table(5,3)*sin(DH_table(5,1));
    0 sin(DH_table(5,4)) cos(DH_table(5,4)) DH_table(5,2);
    0 0 0 1];

A56=[cos(DH_table(6,1)) -sin(DH_table(6,1))*cos(DH_table(6,4)) sin(DH_table(6,1))*sin(DH_table(6,4)) DH_table(6,3)*cos(DH_table(6,1));
    sin(DH_table(6,1)) cos(DH_table(6,1))*cos(DH_table(6,4)) -cos(DH_table(6,1))*sin(DH_table(6,4)) DH_table(6,3)*sin(DH_table(6,1));
    0 sin(DH_table(6,4)) cos(DH_table(6,4)) DH_table(6,2);
    0 0 0 1];

A02 = A01*A12;
A03 = A02*A23;
A04 = A03*A34;
A05 = A04*A45;
A06 = A05*A56;

position = simplify(A06(1:3,4))
```

From A06, we get the position of end effector



```
position =

85*sin(e)*(sin(a)*sin(d) + cos(a)*cos(b)*cos(c)*cos(d) - cos(a)*cos(d)*sin(b)*sin(c)) - 340*cos(a)*cos(b) - 85*sin(b + c)*cos(a)*cos(e) - 50*cos(a)*cos(b)*cos(c) + 370*cos(a)*cos(b)*sin(c) + 370*cos(a)*cos(c)*sin(b) + 50*cos(a)*sin(b)*sin(c)
370*cos(b)*sin(a)*sin(c) - 340*cos(b)*sin(a) - 85*sin(b + c)*cos(e)*sin(a) - 50*cos(b)*cos(c)*sin(a) - 85*sin(e)*(cos(a)*sin(d) - cos(b)*cos(c)*cos(d)*sin(a) + cos(d)*sin(a)*sin(b)*sin(c)) + 370*cos(c)*sin(a)*sin(b) + 50*sin(a)*sin(b)*sin(c)
                                                    (85*sin(b + c)*sin(d + e))/2 - 50*sin(b + c) - 340*sin(b) - 370*cos(b + c) + 85*cos(b + c)*cos(e) - (85*sin(b + c)*sin(d - e))/2 + 400
```

Px =

85*sin(e)*(sin(a)*sin(d) + cos(a)*cos(b)*cos(c)*cos(d) - cos(a)*cos(d)*sin(b)*sin(c)) - 340*cos(a)*cos(b) - 85*sin(b + c)*cos(a)*cos(e) - 50*cos(a)*cos(b)*cos(c) + 370*cos(a)*cos(b)*sin(c) + 370*cos(a)*cos(c)*sin(b) + 50*cos(a)*sin(b)*sin(c)

Py =

370*cos(b)*sin(a)*sin(c) - 340*cos(b)*sin(a) - 85*sin(b + c)*cos(e)*sin(a) - 50*cos(b)*cos(c)*sin(a) - 85*sin(e)*(cos(a)*sin(d) - cos(b)*cos(c)*cos(d)*sin(a) + cos(d)*sin(a)*sin(b)*sin(c)) + 370*cos(c)*sin(a)*sin(b) + 50*sin(a)*sin(b)*sin(c)

Pz =

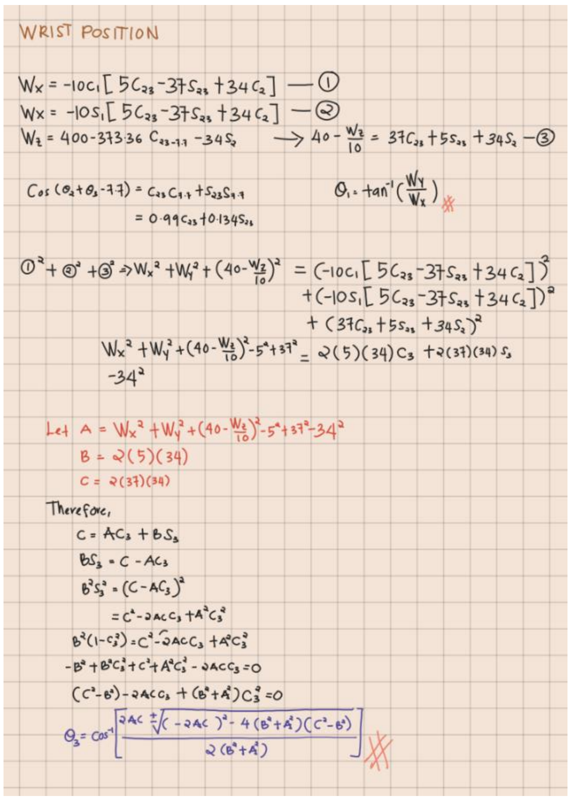
(85*sin (b + c) *sin (d + e))/2 - 50*sin (b + c) - 340*sin(b) - 370*cos (b + c) + 85*cos (b + c) *cos(e) - (85*sin (b + c) *sin (d - e))/2 + 400

1.2. INVERSE KINEMATIC

A mathematical method used in robotics called inverse kinematics (IK) is used to calculate the joint configurations (angles or displacements) of a robotic manipulator that are required to achieve the desired end-effector position and orientation. Inverse kinematics addresses the inverse problem of identifying the joint values needed to obtain a given end-effector position. This contrasts with forward kinematics, which determines the position and orientation of the end effector based on known joint values.

For a 6-degree-of-freedom (6-DOF) robotic arm, inverse kinematics involves figuring out the joint angles required to precisely position and orient the end effector in a desired location. Having six joints, a 6-DOF robot may move in six different directions: three translational and three rotational or all are rotational. The inverse kinematics problem for such a system might be challenging due to the nonlinearity of the equations involved. Since it has 6 thetas need to be identified, we can get them from the position of wrist postion for theta 1, 2 and 3. while the remaining theta get from, the rotation matrix of wrist, R36.

WRIST POSITION: -for theta1, theta2, and theta3



WRIST POSITION

$$W_x = -10c_1[5C_{23} - 37S_{23} + 34C_2] \quad — \textcircled{1}$$

$$W_x = -10S_1[5C_{23} - 37S_{23} + 34C_2] \quad — \textcircled{2}$$

$$W_z = 400 - 373.36\,C_{23-7.7} - 34S_2 \longrightarrow 40 - \frac{W_z}{10} = 37C_{23} + 5S_{23} + 34S_2 \quad — \textcircled{3}$$

$$\cos(\theta_2 + \theta_3 - 7.7) = C_{23}C_{7.7} + S_{23}S_{7.7}$$
$$= 0.99C_{23} + 0.134S_{23}$$

$$\theta_1 = \tan^{-1}\left(\frac{W_y}{W_x}\right) \#$$

$$\textcircled{1}^2 + \textcircled{2}^2 + \textcircled{3}^2 \Rightarrow W_x^2 + W_y^2 + \left(40 - \frac{W_z}{10}\right)^2 = (-10c_1[5C_{23} - 37S_{23} + 34C_2])^2$$
$$+ (-10S_1[5C_{23} - 37S_{23} + 34C_2])^2$$
$$+ (37C_{23} + 5S_{23} + 34S_2)^2$$

$$W_x^2 + W_y^2 + \left(40 - \frac{W_z}{10}\right)^2 - 5^2 + 37^2 = 2(5)(34)C_3 + 2(37)(34)S_3$$
$$-34^2$$

Let $A = W_x^2 + W_y^2 + \left(40 - \frac{W_z}{10}\right)^2 - 5^2 + 37^2 - 34^2$

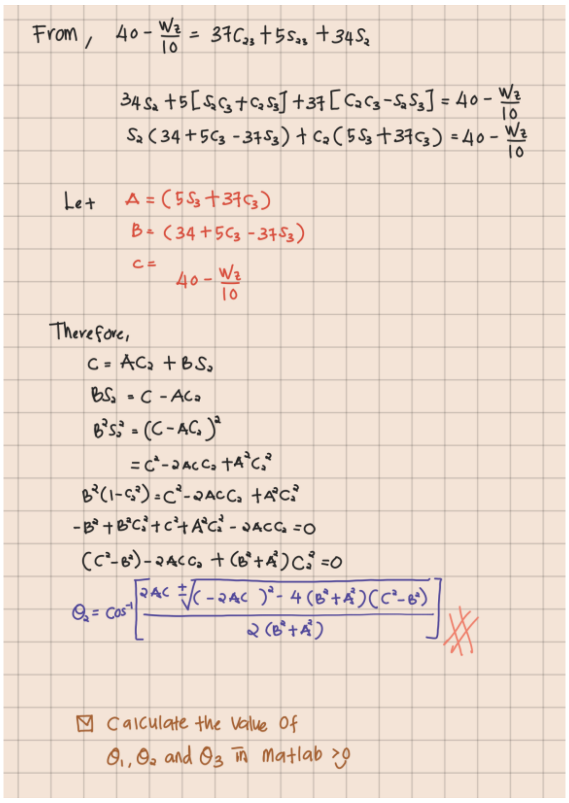
$B = 2(5)(34)$

$C = 2(37)(34)$

Therefore,

$$C = AC_3 + BS_3$$
$$BS_3 = C - AC_3$$
$$B^2S_3^2 = (C - AC_3)^2$$
$$= C^2 - 2ACC_3 + A^2C_3^2$$
$$B^2(1 - C_3^2) = C^2 - 2ACC_3 + A^2C_3^2$$
$$-B^2 + B^2C_3^2 + C^2 + A^2C_3^2 - 2ACC_3 = 0$$
$$(C^2 - B^2) - 2ACC_3 + (B^2 + A^2)C_3^2 = 0$$

$$\theta_3 = \cos^{-1}\left[\frac{2AC \pm \sqrt{(-2AC)^2 - 4(B^2 + A^2)(C^2 - B^2)}}{2(B^2 + A^2)}\right] \#$$



From, $40 - \frac{W_z}{10} = 37C_{23} + 5S_{23} + 34S_2$

$$34S_2 + 5[S_2C_3 + C_2S_3] + 37[C_2C_3 - S_2S_3] = 40 - \frac{W_z}{10}$$

$$S_2(34 + 5C_3 - 37S_3) + C_2(5S_3 + 37C_3) = 40 - \frac{W_z}{10}$$

Let $A = (5S_3 + 37C_3)$

$B = (34 + 5C_3 - 37S_3)$

$C = 40 - \frac{W_z}{10}$

Therefore,

$$C = AC_2 + BS_2$$

$$BS_2 = C - AC_2$$

$$B^2S_2^2 = (C - AC_2)^2$$

$$= C^2 - 2ACC_2 + A^2C_2^2$$

$$B^2(1 - C_2^2) = C^2 - 2ACC_2 + A^2C_2^2$$

$$-B^2 + B^2C_2^2 + C^2 + A^2C_2^2 - 2ACC_2 = 0$$

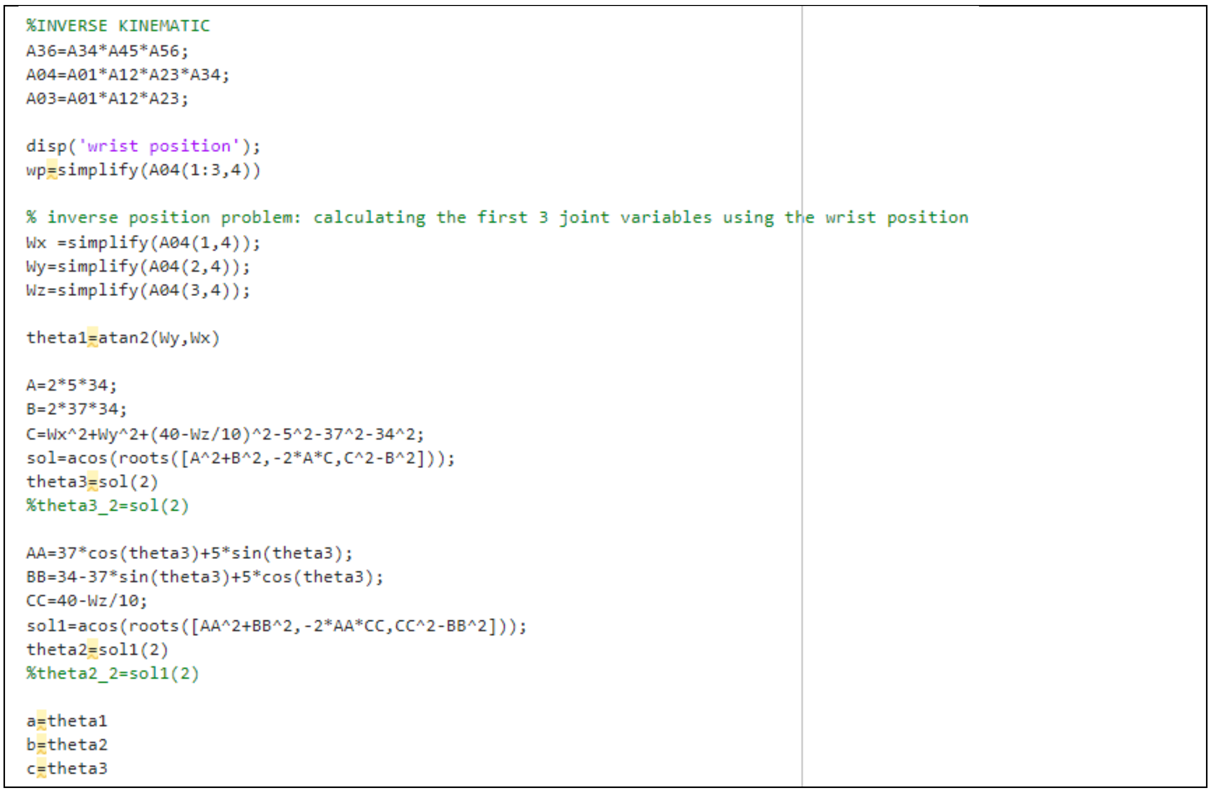
$$(C^2 - B^2) - 2ACC_2 + (B^2 + A^2)C_2^2 = 0$$

$$\theta_2 = \cos^{-1}\left[\frac{2AC \pm \sqrt{(-2AC)^2 - 4(B^2 + A^2)(C^2 - B^2)}}{2(B^2 + A^2)}\right] \#$$

☑ Calculate the Value Of

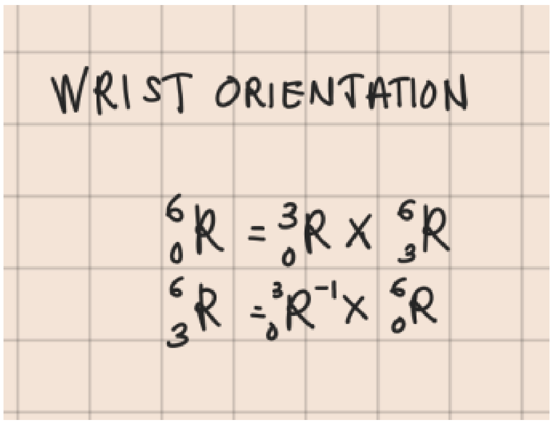
θ_1, θ_2 and θ_3 in Matlab >o

MATLAB CODE:



```
%INVERSE KINEMATIC
A36=A34*A45*A56;
A04=A01*A12*A23*A34;
A03=A01*A12*A23;

disp('wrist position');
wp=simplify(A04(1:3,4))

% inverse position problem: calculating the first 3 joint variables using the wrist position
Wx =simplify(A04(1,4));
Wy=simplify(A04(2,4));
Wz=simplify(A04(3,4));

theta1=atan2(Wy,Wx)

A=2*5*34;
B=2*37*34;
C=Wx^2+Wy^2+(40-Wz/10)^2-5^2-37^2-34^2;
sol=acos(roots([A^2+B^2,-2*A*C,C^2-B^2]));
theta3=sol(2)
%theta3_2=sol(2)

AA=37*cos(theta3)+5*sin(theta3);
BB=34-37*sin(theta3)+5*cos(theta3);
CC=40-Wz/10;
sol1=acos(roots([AA^2+BB^2,-2*AA*CC,CC^2-BB^2]));
theta2=sol1(2)
%theta2_2=sol1(2)

a=theta1
b=theta2
c=theta3
```

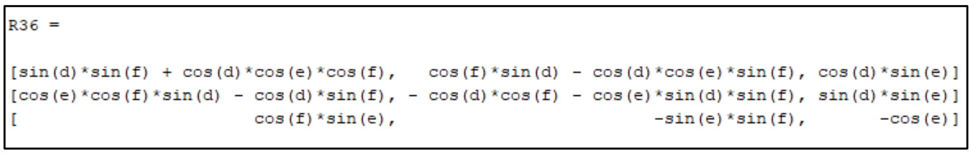
WRIST ORIENTATION: -for theta4, theta5, and theta6



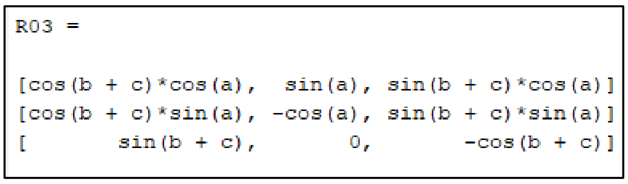
WRIST ORIENTATION

$$ {}^{6}_{0}R = {}^{3}_{0}R \times {}^{6}_{3}R $$

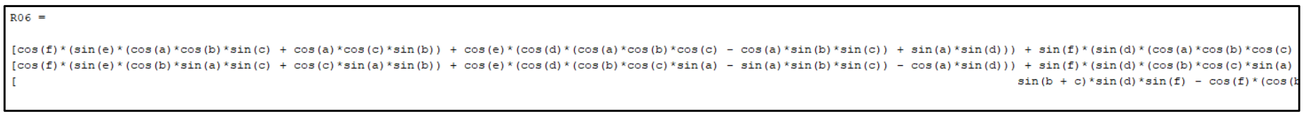
$$ {}^{6}_{3}R = {}^{3}_{0}R^{-1} \times {}^{6}_{0}R $$



```
R36 =

[sin(d)*sin(f) + cos(d)*cos(e)*cos(f),   cos(f)*sin(d) - cos(d)*cos(e)*sin(f), cos(d)*sin(e)]
[cos(e)*cos(f)*sin(d) - cos(d)*sin(f), - cos(d)*cos(f) - cos(e)*sin(d)*sin(f), sin(d)*sin(e)]
[                      cos(f)*sin(e),                          -sin(e)*sin(f),       -cos(e)]
```

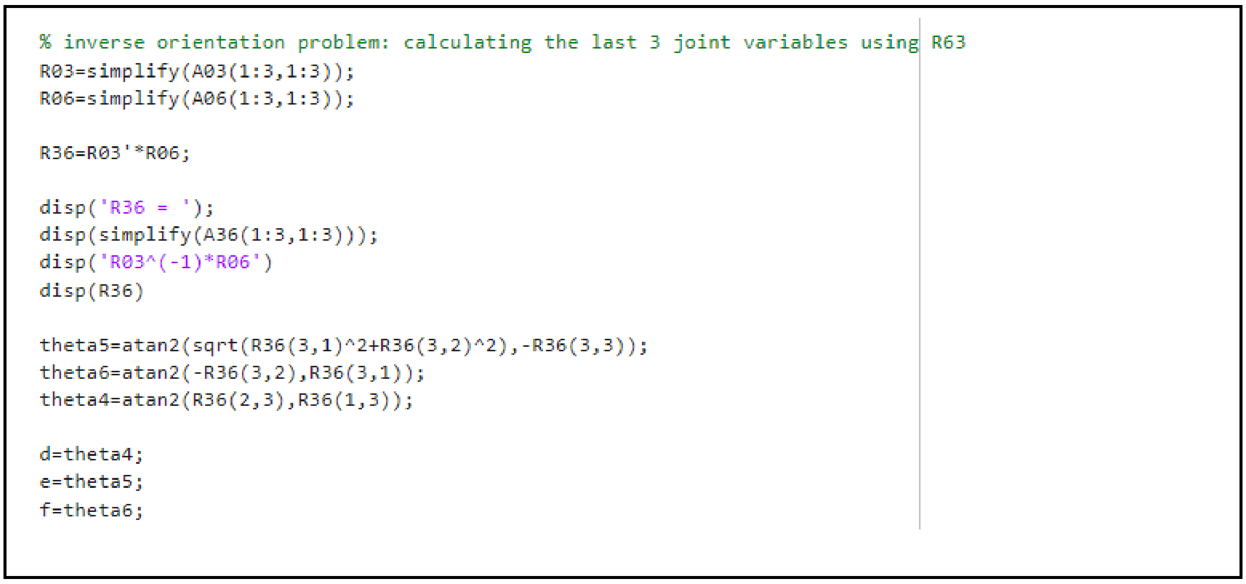


```
R03 =

[cos(b + c)*cos(a),  sin(a), sin(b + c)*cos(a)]
[cos(b + c)*sin(a), -cos(a), sin(b + c)*sin(a)]
[       sin(b + c),       0,       -cos(b + c)]
```

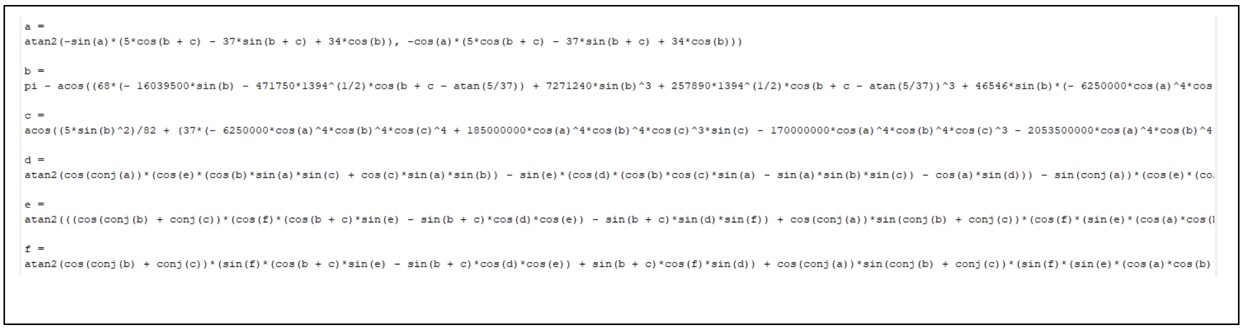


```
R06 =

[cos(f)*(sin(e)*(cos(a)*cos(b)*sin(c) + cos(a)*cos(c)*sin(b)) + cos(e)*(cos(d)*(cos(a)*cos(b)*cos(c) - cos(a)*sin(b)*sin(c)) + sin(a)*sin(d))) + sin(f)*(sin(d)*(cos(a)*cos(b)*cos(c)
[cos(f)*(sin(e)*(cos(b)*sin(a)*sin(c) + cos(c)*sin(a)*sin(b)) + cos(e)*(cos(d)*(cos(b)*cos(c)*sin(a) - sin(a)*sin(b)*sin(c)) - cos(a)*sin(d))) + sin(f)*(sin(d)*(cos(b)*cos(c)*sin(a)
[                                                                                                                                         sin(b + c)*sin(d)*sin(f) - cos(f)*(cos(b
```

MATLAB CODE:



```
% inverse orientation problem: calculating the last 3 joint variables using R63
R03=simplify(A03(1:3,1:3));
R06=simplify(A06(1:3,1:3));

R36=R03'*R06;

disp('R36 = ');
disp(simplify(A36(1:3,1:3)));
disp('R03^(-1)*R06')
disp(R36)

theta5=atan2(sqrt(R36(3,1)^2+R36(3,2)^2),-R36(3,3));
theta6=atan2(-R36(3,2),R36(3,1));
theta4=atan2(R36(2,3),R36(1,3));

d=theta4;
e=theta5;
f=theta6;
```

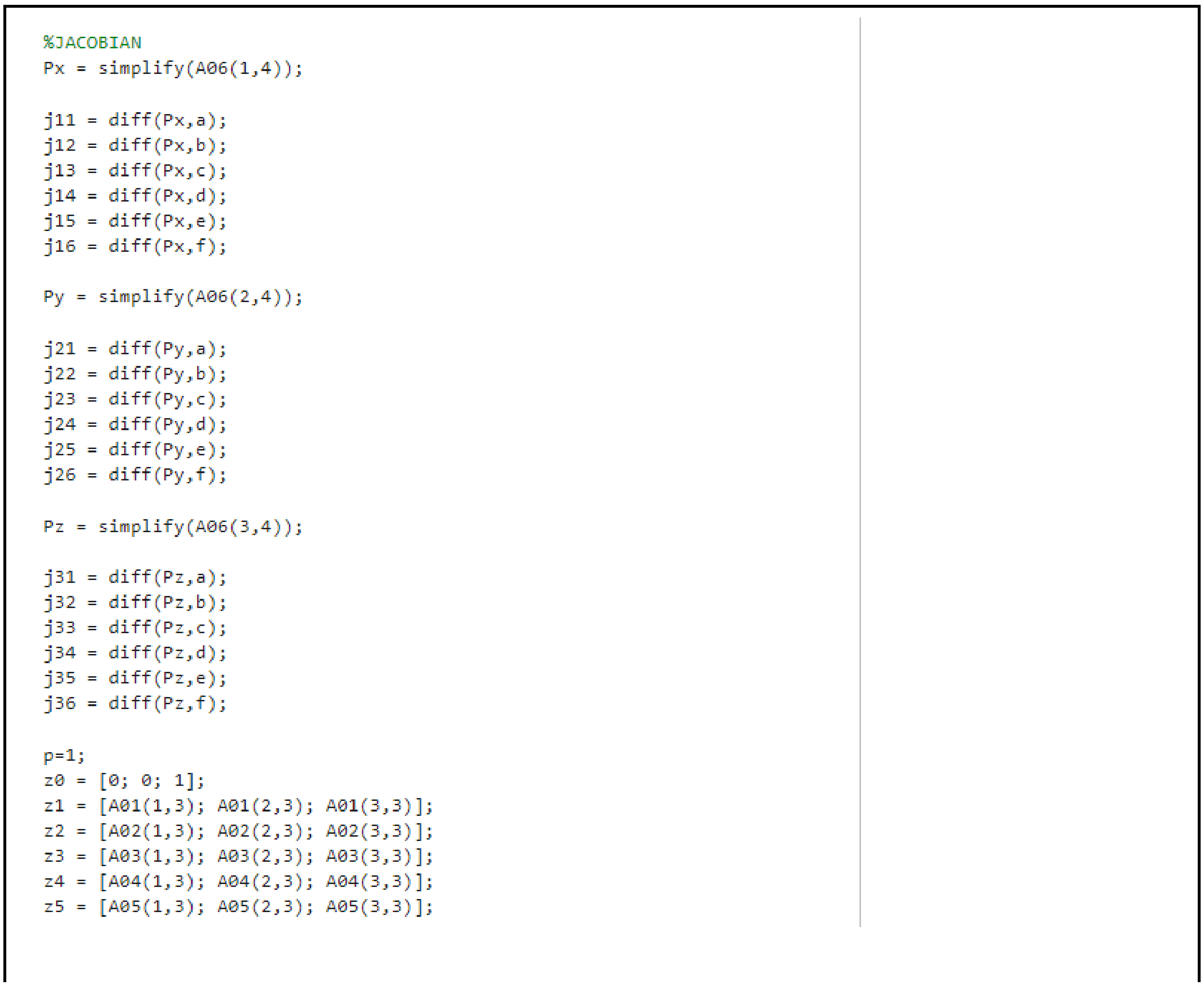


```
a =
atan2(-sin(a)*(5*cos(b + c) - 37*sin(b + c) + 34*cos(b)), -cos(a)*(5*cos(b + c) - 37*sin(b + c) + 34*cos(b)))

b =
pi - acos((68*(- 16039500*sin(b) - 471750*1394^(1/2)*cos(b + c - atan(5/37)) + 7271240*sin(b)^3 + 257890*1394^(1/2)*cos(b + c - atan(5/37))^3 + 46546*sin(b)*(- 6250000*cos(a)^4*cos

c =
acos((5*sin(b)^2)/82 + (37*(- 6250000*cos(a)^4*cos(b)^4*cos(c)^4 + 185000000*cos(a)^4*cos(b)^4*cos(c)^3*sin(c) - 170000000*cos(a)^4*cos(b)^4*cos(c)^3 - 2053500000*cos(a)^4*cos(b)^4

d =
atan2(cos(conj(a))*(cos(e)*(cos(b)*sin(a)*sin(c) + cos(c)*sin(a)*sin(b)) - sin(e)*(cos(d)*(cos(b)*cos(c)*sin(a) - sin(a)*sin(b)*sin(c)) - cos(a)*sin(d))) - sin(conj(a))*(cos(e)*(co

e =
atan2(((cos(conj(b) + conj(c))*(cos(f)*(cos(b + c)*sin(e) - sin(b + c)*cos(d)*cos(e)) - sin(b + c)*sin(d)*sin(f)) + cos(conj(a))*sin(conj(b) + conj(c))*(cos(f)*(sin(e)*(cos(a)*cos(

f =
atan2(cos(conj(b) + conj(c))*(sin(f)*(cos(b + c)*sin(e) - sin(b + c)*cos(d)*cos(e)) + sin(b + c)*cos(f)*sin(d)) + cos(conj(a))*sin(conj(b) + conj(c))*(sin(f)*(sin(e)*(cos(a)*cos(b)
```

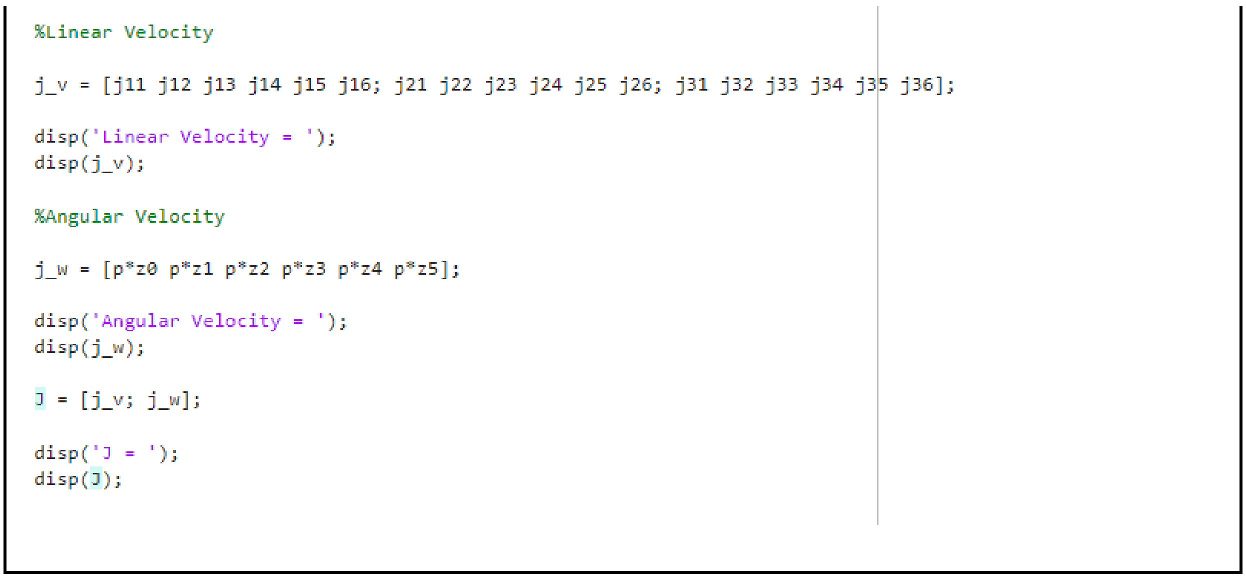
2. LINEAR AND ANGULAR VELOCITY

The Jacobian matrix will be used to determine the robot's linear and angular velocity representation. Joint and end-effector velocities are plotted out and related to each other using the Jacobian. A 6-degree-of-freedom (6-DOF) robotic system's translational motion is correlated with its linear velocity, while its rotational motion is correlated with its angular velocity.

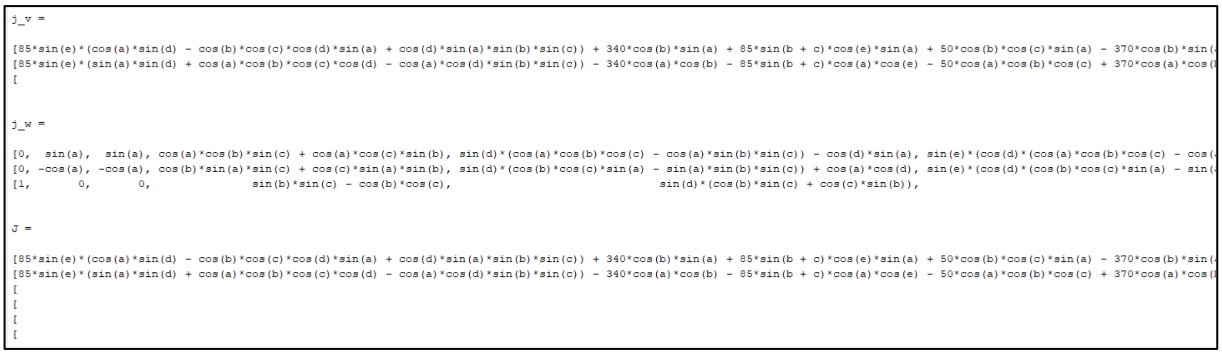
MATLAB CODE:



```
%JACOBIAN
Px = simplify(A06(1,4));

j11 = diff(Px,a);
j12 = diff(Px,b);
j13 = diff(Px,c);
j14 = diff(Px,d);
j15 = diff(Px,e);
j16 = diff(Px,f);

Py = simplify(A06(2,4));

j21 = diff(Py,a);
j22 = diff(Py,b);
j23 = diff(Py,c);
j24 = diff(Py,d);
j25 = diff(Py,e);
j26 = diff(Py,f);

Pz = simplify(A06(3,4));

j31 = diff(Pz,a);
j32 = diff(Pz,b);
j33 = diff(Pz,c);
j34 = diff(Pz,d);
j35 = diff(Pz,e);
j36 = diff(Pz,f);

p=1;
z0 = [0; 0; 1];
z1 = [A01(1,3); A01(2,3); A01(3,3)];
z2 = [A02(1,3); A02(2,3); A02(3,3)];
z3 = [A03(1,3); A03(2,3); A03(3,3)];
z4 = [A04(1,3); A04(2,3); A04(3,3)];
z5 = [A05(1,3); A05(2,3); A05(3,3)];
```



```
%Linear Velocity

j_v = [j11 j12 j13 j14 j15 j16; j21 j22 j23 j24 j25 j26; j31 j32 j33 j34 j35 j36];

disp('Linear Velocity = ');
disp(j_v);

%Angular Velocity

j_w = [p*z0 p*z1 p*z2 p*z3 p*z4 p*z5];

disp('Angular Velocity = ');
disp(j_w);

J = [j_v; j_w];

disp('J = ');
disp(J);
```



```
j_v =

[85*sin(e)*(cos(a)*sin(d) - cos(b)*cos(c)*cos(d)*sin(a) + cos(d)*sin(a)*sin(b)*sin(c)) + 340*cos(b)*sin(a) + 85*sin(b + c)*cos(e)*sin(a) + 50*cos(b)*cos(c)*sin(a) - 370*cos(b)*sin(
[85*sin(e)*(sin(a)*sin(d) + cos(a)*cos(b)*cos(c)*cos(d) - cos(a)*cos(d)*sin(b)*sin(c)) - 340*cos(a)*cos(b) - 85*sin(b + c)*cos(a)*cos(e) - 50*cos(a)*cos(b)*cos(c) + 370*cos(a)*cos(
[


j_w =

[0,  sin(a),  sin(a), cos(a)*cos(b)*sin(c) + cos(a)*cos(c)*sin(b), sin(d)*(cos(a)*cos(b)*cos(c) - cos(a)*sin(b)*sin(c)) - cos(d)*sin(a), sin(e)*(cos(d)*(cos(a)*cos(b)*cos(c) - cos(
[0, -cos(a), -cos(a), cos(b)*sin(a)*sin(c) + cos(c)*sin(a)*sin(b), sin(d)*(cos(b)*cos(c)*sin(a) - sin(a)*sin(b)*sin(c)) + cos(a)*cos(d), sin(e)*(cos(d)*(cos(b)*cos(c)*sin(a) - sin(
[1,       0,       0,                sin(b)*sin(c) - cos(b)*cos(c),                               sin(d)*(cos(b)*sin(c) + cos(c)*sin(b)),


J =

[85*sin(e)*(cos(a)*sin(d) - cos(b)*cos(c)*cos(d)*sin(a) + cos(d)*sin(a)*sin(b)*sin(c)) + 340*cos(b)*sin(a) + 85*sin(b + c)*cos(e)*sin(a) + 50*cos(b)*cos(c)*sin(a) - 370*cos(b)*sin(
[85*sin(e)*(sin(a)*sin(d) + cos(a)*cos(b)*cos(c)*cos(d) - cos(a)*cos(d)*sin(b)*sin(c)) - 340*cos(a)*cos(b) - 85*sin(b + c)*cos(a)*cos(e) - 50*cos(a)*cos(b)*cos(c) + 370*cos(a)*cos(
[
[
[
[
```

3. SIMULATION USING ROBODK SOFTWARE

The simulation is to write at least 3 letters on a board. It was conducted in RoboDk using python.

3.1. Points

Font: Times New Roman

Height: 200 mm

ABC

The letters are drawn in Adobe Illustrator then save on a Scalable Vector Graphics (SVG) file. Then, the svg file is converted to points in x,y coordinate using path to point online converter. The coordinate was saved in Commo-seperated values (CSV) file to be imported into the python script.

3.2. Python

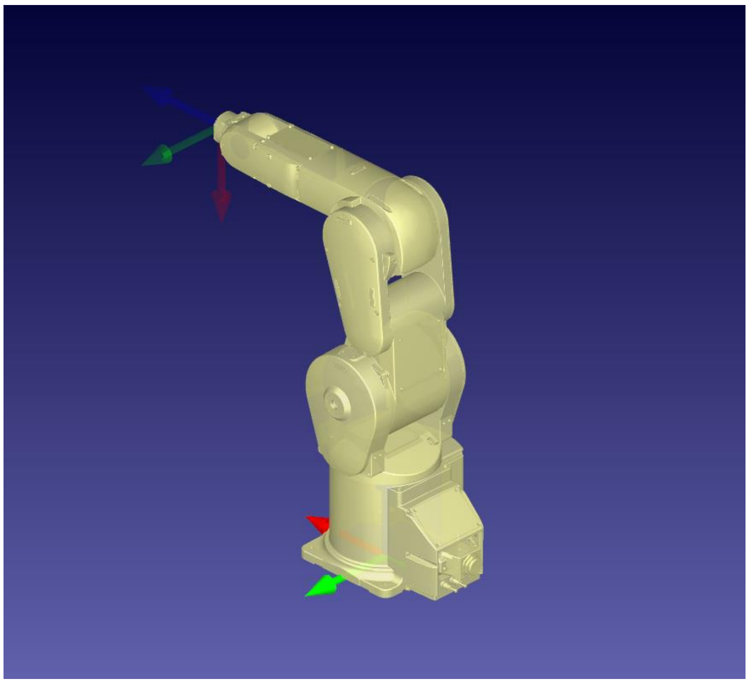
The python code is based on RoboDK points to program example in RoboDK Library with some adjustments and improvements.

```python
#---------------------------------------------------
#--------------- PROGRAM START ---------------------
from robodk.robolink import * # API to communicate with RoboDK
from robodk.robomath import * # basic matrix operations
from robodk.robodialogs import *
from robodk.robofileio import *

RDK = Robolink()

# Generate the points curve path
#POINTS = MakePoints(P_START, P_END, NUM_POINTS)

# Ask the user to pick a file

# Input file name
#input_csv = getFileName(path_file) + ".csv"
input_csv = "C:\\Users\\nurbi\\Desktop\\abcPoint.csv"

# Output list to store the transformed data
output_data = []

# Read data from the input CSV file
with open(input_csv, 'r') as file:
    lines = file.readlines()

# Remove the header if it exists
header = lines.pop(0) if lines and ',' in lines[0] else None

# Process each line
output_data.append([500, float(lines[0].strip().split(',')[0])* -0.353 +600,
                    float(lines[0].strip().split(',')[1])* -0.353 +700])
for line in lines:
    if "#" in line:
        output_data.append([600, output_data[-1][1], output_data[-1][2]])
        continue

    # Assuming X is the first column and Y is the second column
    x, y = map(float, line.strip().split(','))

    # Convert X and Y to mm (multiply by 25.4)
    x_mm = -x * 0.353 +500
    y_mm = -y * 0.353 +700

    # Append a new row to the output data list
    output_data.append([600, x_mm, y_mm])
```

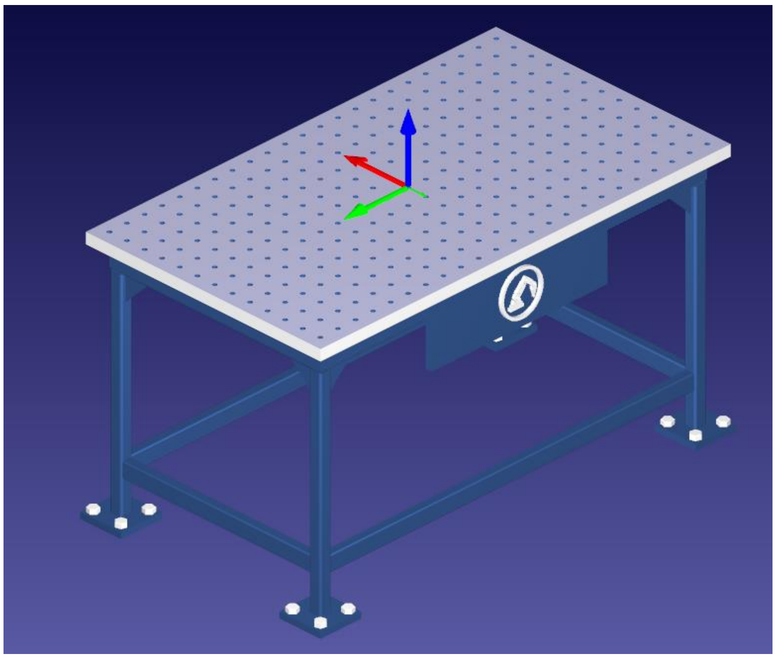
```python
POINTS = output_data

# Initialize the RoboDK API
RDK = Robolink()

# turn off auto rendering (faster)
RDK.Render(False)

# Promt the user to select a robot (if only one robot is available
# it will select that robot automatically)
robot = RDK.ItemUserPick('Select a robot', ITEM_TYPE_ROBOT)

# Turn rendering ON before starting the simulation
RDK.Render(True)

# Abort if the user hits Cancel
if not robot.Valid():
    quit()

# Retrieve the robot reference frame
reference = robot.Parent()

# Use the robot base frame as the active reference
robot.setPoseFrame(reference)

# get the current orientation of the robot (with respect to the active
# reference frame and tool frame)
pose_ref = robot.Pose()
print(Pose_2_TxyzRxyz(pose_ref))

# a pose can also be defined as xyzwpr / xyzABC
#pose_ref = TxyzRxyz_2_Pose([100,200,300,0,0,pi])
#-------------------------------------------------------------
# Option 1: Move the robot using the Python script
# We can automatically force the "Create robot program" action usinga RUNMODE state
# RDK.setRunMode(RUNMODE_MAKE_ROBOTPROG)
# Iterate through all the points
for i in range(len(output_data)):
    # update the reference target with the desired XYZ coordinates
    pose_i = pose_ref
    pose_i.setPos(POINTS[i])
    # Move the robot to that target:
    robot.MoveJ(pose_i)
# Done, stop program execution
```

3.3. Robot and tools

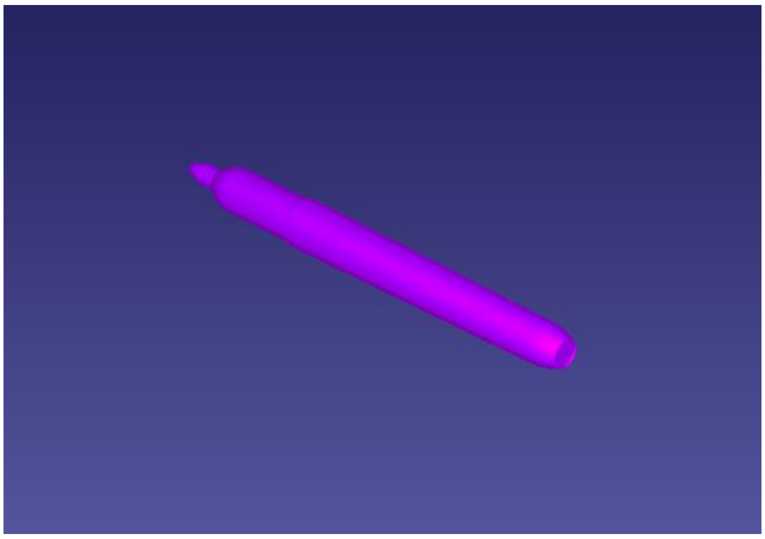
3.3.1. Mitsubishi RV-7FR



3.3.2. Station Table



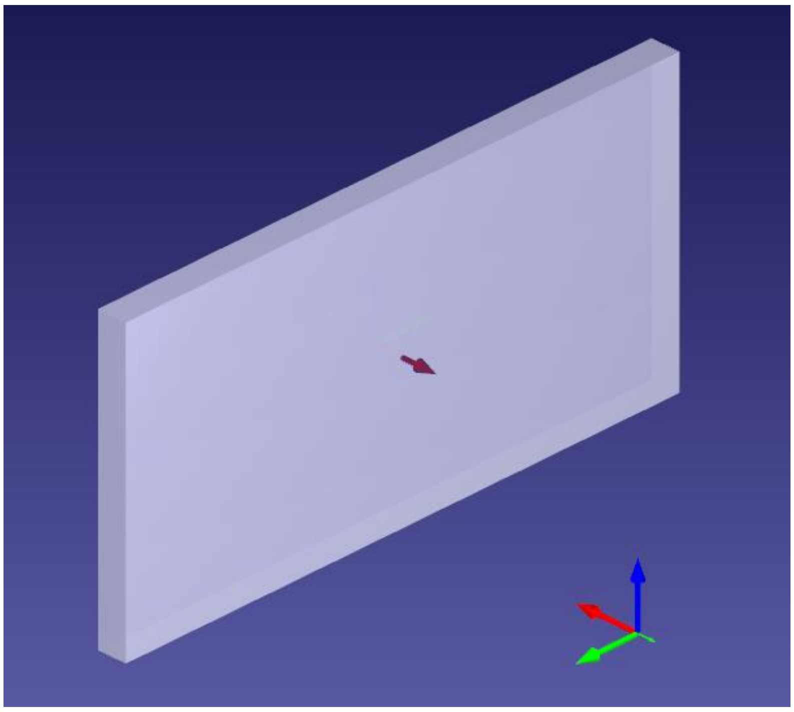
3.3.3. End-effector (Marker)



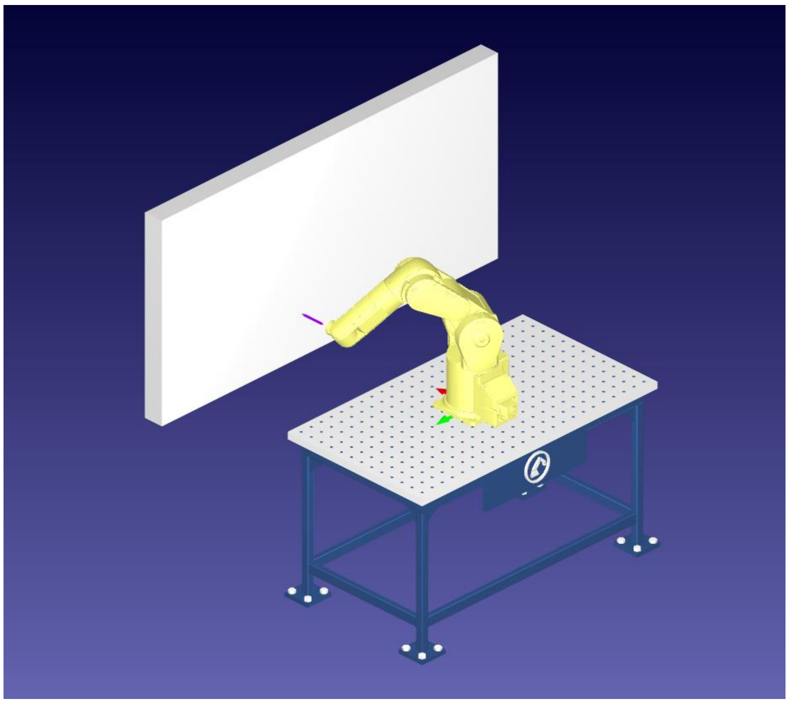
3.3.4. Board

Width: 2 m

Height: 1 m

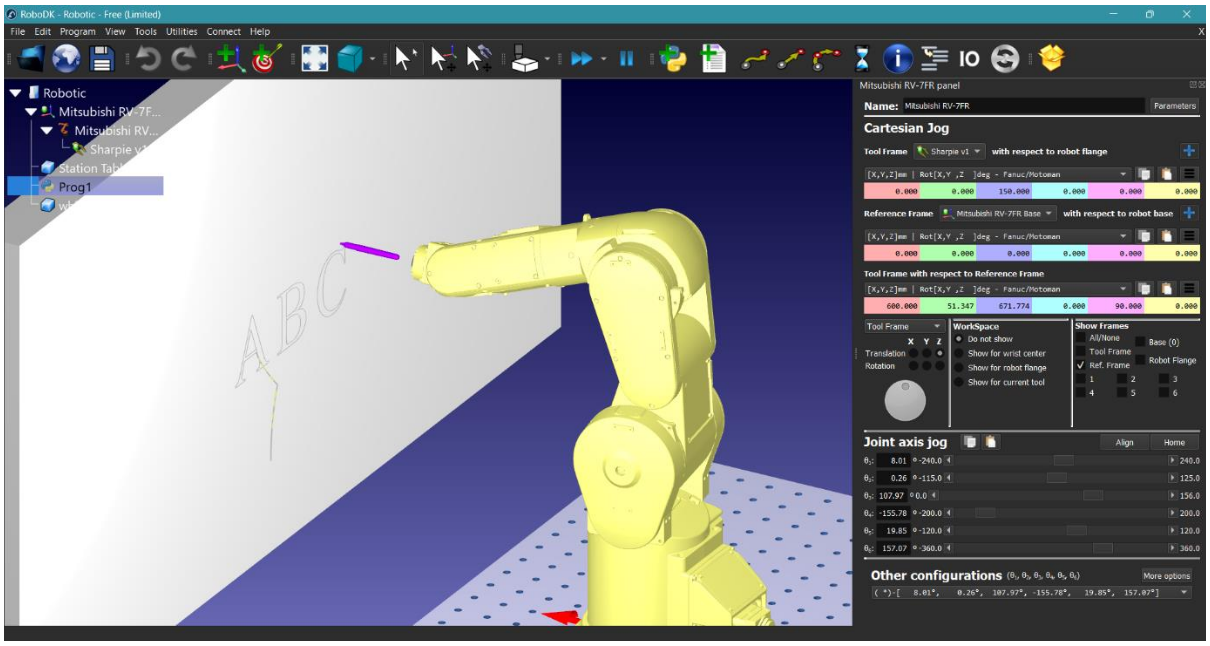


3.3.5. Full Assembly



3.4. Simulation

Simulation video link:


RoboDK - Robotic - Free (Limited)
File Edit Program View Tools Utilities Connect Help
Robotic
Mitsubishi RV-7F...
Mitsubishi RV...
Sharpie v1
Station Table
Prog1
ABC

Mitsubishi RV-7FR panel

Name: Mitsubishi RV-7FR

Cartesian Jog

Tool Frame: Sharpie v1 with respect to robot flange

```
[X,Y,Z]mm | Rot[X,Y ,Z  ]deg - Fanuc/Motoman
0.000   0.000   150.000   0.000   0.000   0.000
```

Reference Frame: Mitsubishi RV-7FR Base with respect to robot base

```
[X,Y,Z]mm | Rot[X,Y ,Z  ]deg - Fanuc/Motoman
0.000   0.000   0.000   0.000   0.000   0.000
```

Tool Frame with respect to Reference Frame

```
[X,Y,Z]mm | Rot[X,Y ,Z  ]deg - Fanuc/Motoman
600.000   51.347   671.774   0.000   90.000   0.000
```

Tool Frame — Translation X Y Z, Rotation

WorkSpace: ● Do not show; Show for wrist center; Show for robot flange; Show for current tool

Show Frames: All/None; Base (0); Tool Frame; Robot Flange; ✓ Ref. Frame; 1; 2; 3; 4; 5; 6

Joint axis jog

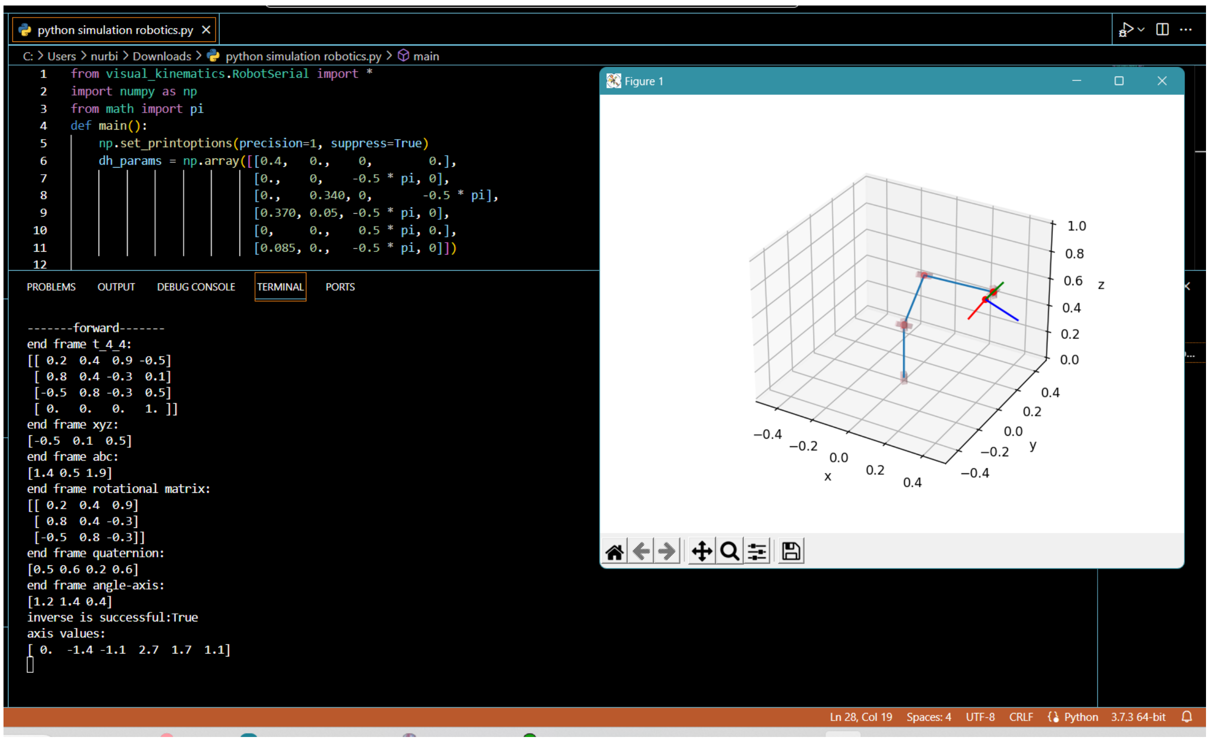
	Value	Min	Max
θ_1:	8.01	-240.0	240.0
θ_2:	0.26	-115.0	125.0
θ_3:	107.97	0.0	156.0
θ_4:	-155.78	-200.0	200.0
θ_5:	19.85	-120.0	120.0
θ_6:	157.07	-360.0	360.0

Other configurations (θ_1, θ_2, θ_3, θ_4, θ_5, θ_6)

```
( *)-[   8.01°,    0.26°,  107.97°, -155.78°,   19.85°,  157.07°]
```

3.5. Limitations

However, we also try to do numerical analysis using python kinematic solver but did not manage to finish it by the end of the project.



```python
from visual_kinematics.RobotSerial import *
import numpy as np
from math import pi
def main():
    np.set_printoptions(precision=1, suppress=True)
    dh_params = np.array([[0.4,   0.,    0,         0.],
                          [0.,    0,    -0.5 * pi, 0],
                          [0.,    0.340, 0,        -0.5 * pi],
                          [0.370, 0.05, -0.5 * pi, 0],
                          [0,     0.,    0.5 * pi, 0.],
                          [0.085, 0.,   -0.5 * pi, 0]])

```

```
-------forward-------
end frame t_4_4:
[[ 0.2  0.4  0.9 -0.5]
 [ 0.8  0.4 -0.3  0.1]
 [-0.5  0.8 -0.3  0.5]
 [ 0.   0.   0.   1. ]]
end frame xyz:
[-0.5  0.1  0.5]
end frame abc:
[1.4 0.5 1.9]
end frame rotational matrix:
[[ 0.2  0.4  0.9]
 [ 0.8  0.4 -0.3]
 [-0.5  0.8 -0.3]]
end frame quaternion:
[0.5 0.6 0.2 0.6]
end frame angle-axis:
[1.2 1.4 0.4]
inverse is successful:True
axis values:
[ 0.  -1.4 -1.1  2.7  1.7  1.1]
```

CONCLUSION

To summarize, the project entailed designing and building a 6-degree-of-freedom (6-DOF) SCARA robot that was assigned to write three letters, particularly "ABC," on a board using a Times New Roman font with height of 0.2 m. The project was divided into many phases. The first involved forward kinematics computations using the Denavit-Hartenberg (DH) representation, followed by inverse kinematics calculations. The robot was precisely modelled and calculated using MATLAB software.

The working environment of the robot was carefully considered when creating its path to guarantee accurate and practical movement. The robot's path planning simulation was made easier using Python coding, which enabled the robot to successfully generate the letters "ABC" in both Python and A4 paper space. The project also used Jacobian analysis and differential motion to examine how minor modifications to the robot's setup affect the orientation and position of the end effector.

A thorough approach to the design and study of the 6-DOF SCARA robot was given by the project overall, addressing topics like kinematics, trajectory planning, differential motion, Jacobian analysis, and dynamic considerations. Through the successful integration of these components, the project's goals were accomplished, leading to the development of a flexible and effective robotic system that can perform precise writing jobs.

PRESENTATION VIDEO

Youtube Link: https://youtu.be/DrzBRtHIL2w