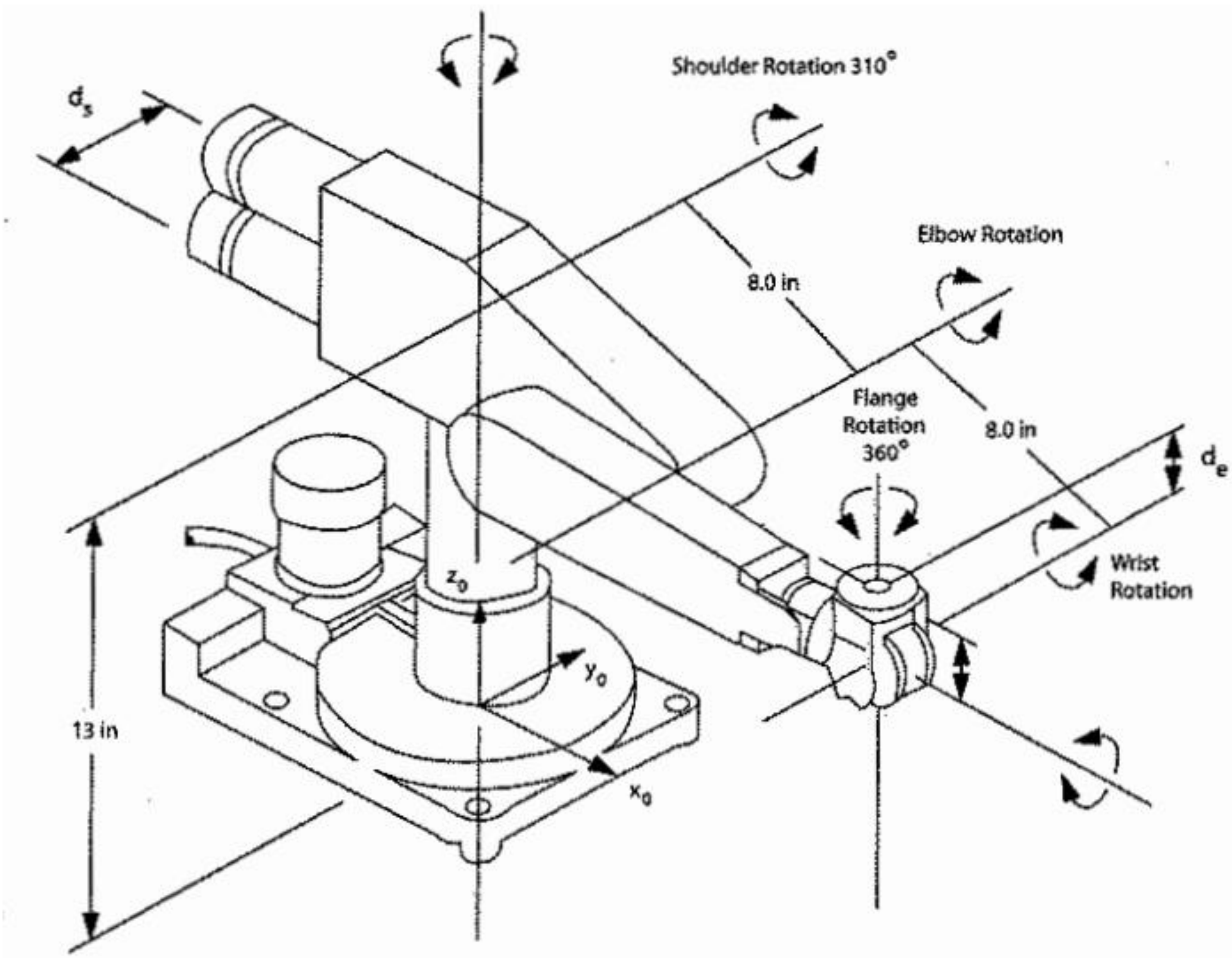
3-10) The forward kinematic equation using DH coordinate for the image below is as followed:



Link	θ	α (Degree)	r	d
1	Θ_1	90°	0	13
2	Θ_2	0	8	d_2
3	Θ_3	90°	8	0
4	Θ_4	-90°	0	d_4
5	Θ_5	90°	0	0
6	Θ_6	0	0	d_6

$$R_1 = \begin{bmatrix} c1 & 0 & s1 & 0 \\ s1 & 0 & -c1 & 0 \\ 0 & 1 & 0 & 13 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_2 = \begin{bmatrix} c2 & -s2 & 0 & 8c2 \\ s2 & c2 & 0 & 8s2 \\ 0 & 0 & 1 & d2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_3 = \begin{bmatrix} c3 & 0 & s3 & 8c3 \\ s3 & 0 & -c3 & 8s3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_4 = \begin{bmatrix} c4 & 0 & -s4 & 0 \\ s4 & 0 & c4 & 0 \\ 0 & -1 & 0 & d4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_5 = \begin{bmatrix} c5 & 0 & s5 & 0 \\ s5 & 0 & -c5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_6 = \begin{bmatrix} c6 & -s6 & 0 & 0 \\ s6 & c6 & 0 & 0 \\ 0 & 0 & 1 & d6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

R11 = c1[c_{23} (c4c5c6 – s4s6) – s4s6s23] + s1[c4s6 + s4c5c6]

R12 = c1[-c23(c4c5s6 + s4c6) + s5s6s23] + s1[c4c6 – s4c5s6]

R13 = c1[c4s5c23 + c5s23] – s1s4s5

Dx = d2s1 + d4c1s23 + d6[c1(c4s5c23 + c5s23) + s1s4 + s5] +8c1[c23 + c2]

R21 = -c1[c4s6 + s4c5c6] + s1[c23(c4c5c6 + s4s6) – s5c6s23]

R22 = c1[s4c5s6 – c4c6] + s1[-c23(c4c5s6 + s4c6) + s5s6s23]

Dy = -dc1 + d4ss23 + d6[s1(c4s5c23 + s5c6c23) – c1s4s5] – c1s4s5] + 8s1[c23 + c2]

R31 = s23(c4c5c6 – s4s6) + s5c6c23

R32 = -s23(c4c5s6 + s4c6) – s5s6c23

R33 = -c5c23 + c4s5s23

Dz = 13- d4c23 + d6[-c5c23 + c4s5s23] + 8[s23 + s2]

3-14)

d = $\begin{bmatrix} dx \\ dy \\ dz \end{bmatrix}$ ➔ The inverse position kinematic is ➔ $\begin{bmatrix} dz \\ dy \\ dx \end{bmatrix} = \begin{bmatrix} d3 \\ d2 \\ d1 \end{bmatrix}$

Question 7)

Write a program to compute the rotation matrix R for RPY (Roll, Pitch, Yaw Euler angles

```
clc; clear; close all;
% Testing the functions
my_rotx(pi/4)
my_roty(-pi/4)
my_rotz(pi/2)
my_transl(2,3,4)
my_transl_2("x",pi/4,2,3,4)

% compute the fundamental rotation matrix
function mat_x = my_rotx(theta)
      mat_x = [1        0            0;
               0    cos(theta)    sin(theta);
               0    sin(theta)     cos(theta)];
end
function mat_y = my_roty(theta)
       mat_y = [cos(theta)     0    sin(theta);
                     0         1       0;
                 sin(-theta)   0     cos(theta)];
```

```
end
function mat_z  = my_rotz(theta)
        mat_z = [cos(theta) sin(-theta) 0;
                 sin(theta) cos(theta) 0;
                 0 0 1];
end


% Compute the homogenous transformation matrix

function htm = my_transl(x,y,z)
         htm = [1 0 0 x;
                0 1 0 y;
                0 0  1 z;
                0 0 0 1];
end

function htm = my_transl_2(axis,theta,x,y,z)
     t = [x;y;z];
     if(axis =="x")
         htm = [my_rotx(theta) t;
                0 0 0 1];
     elseif(axis =="y")
         htm = [my_roty(theta) t;
                0 0 0 1];
     elseif(axis == "z")
         htm = [my_rotz(theta) t;
                0 0 0 1];
     else
         disp("Incorrect Input Value")
     end
end
```

Result

ans =

1.0000	0	0
0	0.7071	0.7071
0	0.7071	0.7071

ans =

0.7071	0	-0.7071

```
         0    1.0000         0
    0.7071         0    0.7071


ans =

    0.0000   -1.0000         0
    1.0000    0.0000         0
         0         0    1.0000


ans =

     1     0     0     2
     0     1     0     3
     0     0     1     4
     0     0     0     1


ans =

    1.0000         0         0    2.0000
         0    0.7071    0.7071    3.0000
         0    0.7071    0.7071    4.0000
         0         0         0    1.0000
```

Python code for the question 7, including results of conversion from Rotation Matrix to Euler Angles and vice versa as well as Euler Angles to Quaternion equation and vice versa.

```
print("Calculate the rotation matrix from the Euler Angles")

import numpy as np

def rot_mat(teta1, teta2, teta3, order='xyz'):
    c1 = np.cos(teta1 * np.pi / 180)
    s1 = np.sin(teta1 * np.pi / 180)
    c2 = np.cos(teta2 * np.pi / 180)
    s2 = np.sin(teta2 * np.pi / 180)
    c3 = np.cos(teta3 * np.pi / 180)
    s3 = np.sin(teta3 * np.pi / 180)

    if order == 'xzx':
        matrix=np.array([[c2, -c3*s2, s2*s3],
                         [c1*s2, c1*c2*c3-s1*s3, -c3*s1-c1*c2*s3],
                         [s1*s2, c1*s3+c2*c3*s1, c1*c3-c2*s1*s3]])
    elif order=='xyx':
        matrix=np.array([[c2, s2*s3, c3*s2],
                         [s1*s2, c1*c3-c2*s1*s3, -c1*s3-c2*c3*s1],
                         [-c1*s2, c3*s1+c1*c2*s3, c1*c2*c3-s1*s3]])
    elif order=='yxy':
        matrix=np.array([[c1*c3-c2*s1*s3, s1*s2, c1*s3+c2*c3*s1],
                         [s2*s3, c2, -c3*s2],
                         [-c3*s1-c1*c2*s3, c1*s2, c1*c2*c3-s1*s3]])
    elif order=='yzy':
        matrix=np.array([[c1*c2*c3-s1*s3, -c1*s2, c3*s1+c1*c2*s3],
```

```
                [c3*s2, c2, s2*s3],
                [-c1*s3-c2*c3*s1, s1*s2, c1*c3-c2*s1*s3]])
elif order=='zyz':
    matrix=np.array([[c1*c2*c3-s1*s3, -c3*s1-c1*c2*s3, c1*s2],
                [c1*s3+c2*c3*s1, c1*c3-c2*s1*s3, s1*s2],
                [-c3*s2, s2*s3, c2]])
elif order=='zxz':
    matrix=np.array([[c1*c3-c2*s1*s3, -c1*s3-c2*c3*s1, s1*s2],
                [c3*s1+c1*c2*s3, c1*c2*c3-s1*s3, -c1*s2],
                [s2*s3, c3*s2, c2]])
elif order=='xyz':
    matrix=np.array([[c2*c3, -c2*s3, s2],
                [c1*s3+c3*s1*s2, c1*c3-s1*s2*s3, -c2*s1],
                [s1*s3-c1*c3*s2, c3*s1+c1*s2*s3, c1*c2]])
elif order=='xzy':
    matrix=np.array([[c2*c3, -s2, c2*s3],
                [s1*s3+c1*c3*s2, c1*c2, c1*s2*s3-c3*s1],
                [c3*s1*s2-c1*s3, c2*s1, c1*c3+s1*s2*s3]])
elif order=='yxz':
    matrix=np.array([[c1*c3+s1*s2*s3, c3*s1*s2-c1*s3, c2*s1],
                [c2*s3, c2*c3, -s2],
                [c1*s2*s3-c3*s1, c1*c3*s2+s1*s3, c1*c2]])
elif order=='yzx':
    matrix=np.array([[c1*c2, s1*s3-c1*c3*s2, c3*s1+c1*s2*s3],
                [s2, c2*c3, -c2*s3],
                [-c2*s1, c1*s3+c3*s1*s2, c1*c3-s1*s2*s3]])
elif order=='zyx':
    matrix=np.array([[c1*c2, c1*s2*s3-c3*s1, s1*s3+c1*c3*s2],
                [c2*s1, c1*c3+s1*s2*s3, c3*s1*s2-c1*s3],
```

```
                 [-s2, c2*s3, c2*c3]])
    elif order=='zxy':
        matrix=np.array([[c1*c3-s1*s2*s3, -c2*s1, c1*s3+c3*s1*s2],
                 [c3*s1+c1*s2*s3, c1*c2, s1*s3-c1*c3*s2],
                 [-c2*s3, s2, c2*c3]])

    return matrix
print("Calculate Euler Angles from the Rotation Matrix")
def rot_angl(matrix, order):

    r11, r12, r13 = matrix[0]
    r21, r22, r23 = matrix[1]
    r31, r32, r33 = matrix[2]

    if order == 'xzx':
        theta1 = np.arctan(r31 / r21)
        theta2 = np.arctan(r21 / (r11 * np.cos(theta1)))
        theta3 = np.arctan(-r13 / r12)

    elif order == 'xyx':
        theta1 = np.arctan(-r21 / r31)
        theta2 = np.arctan(-r31 / (r11 *np.cos(theta1)))
        theta3 = np.arctan(r12 / r13)

    elif order == 'yxy':
        theta1 = np.arctan(r12 / r32)
        theta2 = np.arctan(r32 / (r22 *np.cos(theta1)))
        theta3 = np.arctan(-r21 / r23)
```

```
elif order == 'yzy':
    theta1 = np.arctan(-r32 / r12)
    theta2 = np.arctan(-r12 / (r22 *np.cos(theta1)))
    theta3 = np.arctan(r23 / r21)

elif order == 'zyz':
    theta1 = np.arctan(r23 / r13)
    theta2 = np.arctan(r13 / (r33 *np.cos(theta1)))
    theta3 = np.arctan(-r32 / r31)

elif order == 'zxz':
    theta1 = np.arctan(-r13 / r23)
    theta2 = np.arctan(-r23 / (r33 *np.cos(theta1)))
    theta3 = np.arctan(r31 / r32)

elif order == 'xzy':
    theta1 = np.arctan(r32 / r22)
    theta2 = np.arctan(-r12 * np.cos(theta1) / r22)
    theta3 = np.arctan(r13 / r11)

elif order == 'xyz':
    theta1 = np.arctan(-r23 / r33)
    theta2 = np.arctan(r13 * np.cos(theta1) / r33)
    theta3 = np.arctan(-r12 / r11)

elif order == 'yxz':
    theta1 = np.arctan(r13 / r33)
    theta2 = np.arctan(-r23 * np.cos(theta1) / r33)
    theta3 = np.arctan(r21 / r22)
```

```
    elif order == 'yzx':
        theta1 = np.arctan(-r31 / r11)
        theta2 = np.arctan(r21 * np.cos(theta1) / r11)
        theta3 = np.arctan(-r23 / r22)

    elif order == 'zyx':
        theta1 = np.arctan(r21 / r11)
        theta2 = np.arctan(-r31 * np.cos(theta1) / r11)
        theta3 = np.arctan(r32 / r33)

    elif order == 'zxy':
        theta1 = np.arctan(-r12 / r22)
        theta2 = np.arctan(r32 * np.cos(theta1) / r22)
        theta3 = np.arctan(-r31 / r33)

    theta1 = theta1 * 180 / np.pi
    theta2 = theta2 * 180 / np.pi
    theta3 = theta3 * 180 / np.pi

    return (theta1, theta2, theta3)
print("Calculate the rotation matrix from the Euler Angles")
rotation_matrix = rot_mat(45,60, 32)
print(rotation_matrix)
print("\n")
print("Calculate Euler Angles from the Rotation Matrix")
rotation_angles = rot_angl(rotation_mat, 'yzx')
print(rotation_angles)
```

Results Results (As can be seen in the code I retrieved the same values that I assigned to the Euler to rotation matrix from the conversion from rotation matrix to Euler Angles)

```
Calculate the rotation matrix from the Euler Angles
[[ 0.42402405 -0.26495963  0.8660254 ]
 [ 0.89403078  0.27515261 -0.35355339]
 [-0.14461177  0.92416851  0.35355339]]


Calculate Euler Angles from the Rotation Matrix
(45.0, 59.99999999999999, 90.0)

convert Euler Angles to Quaternions

import numpy as np

def get_quat_from_euler(roll, pitch, yaw):

  qx = np.sin(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) - np.cos(roll/2) * np.sin(pitch/2) * np.sin(yaw/2)
  qy = np.cos(roll/2) * np.sin(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.cos(pitch/2) * np.sin(yaw/2)
  qz = np.cos(roll/2) * np.cos(pitch/2) * np.sin(yaw/2) - np.sin(roll/2) * np.sin(pitch/2) * np.cos(yaw/2)
  qw = np.cos(roll/2) * np.cos(pitch/2) * np.cos(yaw/2) + np.sin(roll/2) * np.sin(pitch/2) * np.sin(yaw/2)

  return [qx, qy, qz, qw]

import math

def euler_from_quat(x, y, z, w):

        t0 = +2.0 * (w * x + y * z)
        t1 = +1.0 - 2.0 * (x * x + y * y)
        roll_x = math.atan2(t0, t1)

        t2 = +2.0 * (w * y - z * x)
        t2 = +1.0 if t2 > +1.0 else t2
        t2 = -1.0 if t2 < -1.0 else t2
        pitch_y = math.asin(t2)

        t3 = +2.0 * (w * z + x * y)
        t4 = +1.0 - 2.0 * (y * y + z * z)
        yaw_z = math.atan2(t3, t4)

        return roll_x, pitch_y, yaw_z
print("Convert Euler Angles to Quaternion ")

Euler_to_Quaternion = get_quat_from_euler(0, 1, 1.68)

print(Euler_to_Quaternion)
```

print("\n")

print("Convert Quaternions to Euler Angles")

Quaternion_to_Euler_Angles = euler_from_quat(-0.3570009288599434, 0.3199987247772525, 0.653485816918067, 0.5857537366684029)

print(Quaternion_to_Euler_Angles)

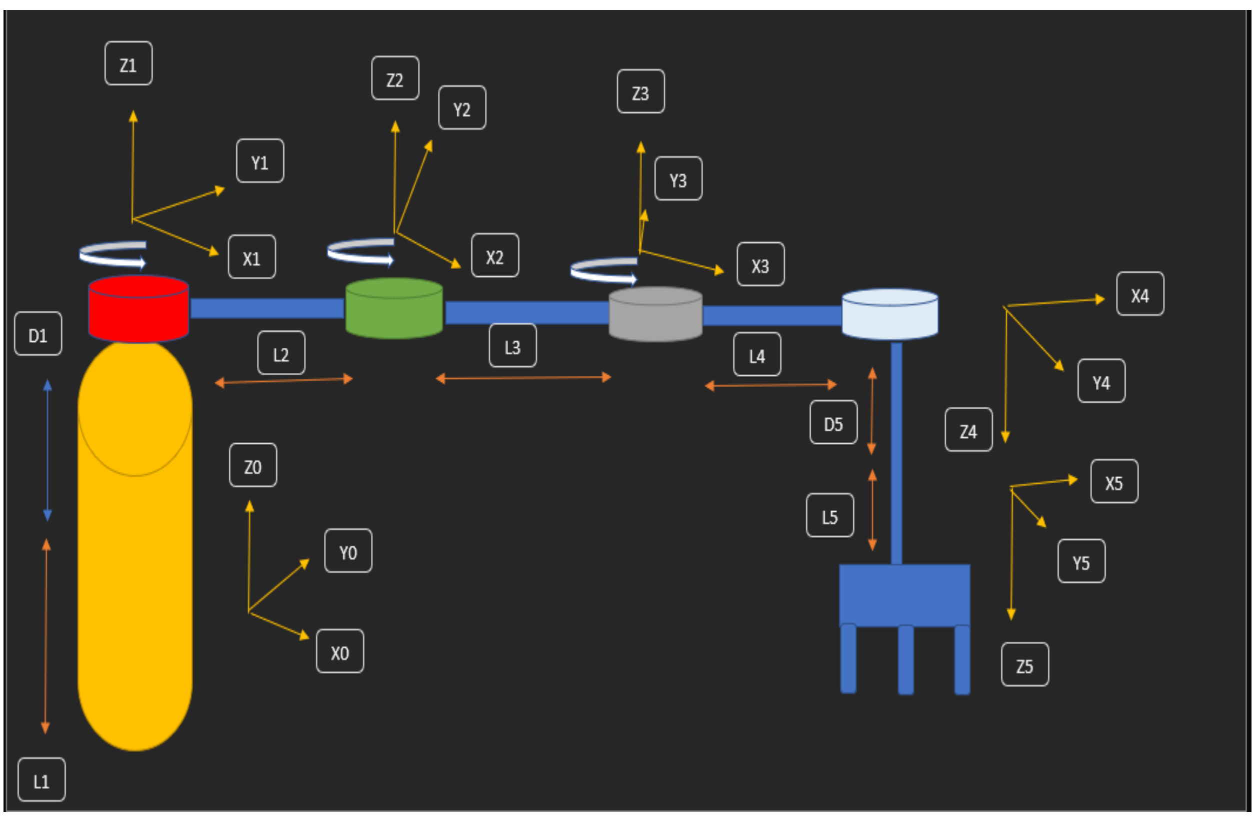
Results (As can be seen in the code I retrieved the same values that I assigned to the Euler to Quaternion equation from the conversion from Quaternion to Euler Angles)

```
Convert Euler Angles to Quaternion
[-0.3570009288599434, 0.3199987247772525, 0.653485816918067, 0.58575373666
84029]


Convert Quaternions to Euler Angles
(0.0, 1.0000000000000002, 1.6800000000000002)
```

Project

Schematic Diagram of my 5 DOF Robot Arm



The analysis of the forward kinematic for 5 DOF Robotic Arm

In this project, I used the Denavit-Hartenberg (DH) approach to derive the robotic arm kinematics. DH approach helped me to find the position and orientation of end effector with respect to the robot base. The table below shows the twenty parameters involved in this design.

Link	θ	α (Degree)	r	d
1	Θ_1	0	0	L1+d1
2	Θ_2	0	L2	0
3	Θ_3	0	L3	0
4	Θ_4	-90	0	L4
5	Θ_5	0	0	L5+d5
6				

The MATLAB code for Forward Kinematic and Inverse Kinematic

```
% The forward kinematic of 5 DOF Robotic Arm
syms q1 q2 q3 q4 q5;
Pi = sym(pi);
q1 = input('Joint angle 1:');
q2 = input('Joint angle 2:');
q3 = input('Joint angle 3:');
q4 = input('Joint angle 4:');
q5 = input('Joint angle 5:');

q1=q1*pi/180;
disp(q1)
q2=q2*pi/180;
disp(q2)
q3=q3*pi/180;
disp(q3)
q4=q4+90;
q4=q4*pi/180;
disp(q4)
q5=q5*pi/180;
disp(q5)

syms q1 q2 q3 q4 q5 d1 a2 a3 a4 d5;
d1=1;
a2=10.5;
a3=14.7;
a4=7.6;
d5=11;
t1 = [cosd(q1) 0 sind(q1) 0; sind(q1) 0 -cosd(q1) 0; 0 1 0 d1; 0 0 0 1];
disp(t1)
```

```
t2 = [cosd(q2) -sind(q2) 0 a2*cosd(q2); sind(q2) cosd(q2) 0 a2*sind(q2); 0 0 1 0; 0 0
0 1];
disp(t2)
t3 = [cosd(q3) -sind(q3) 0 a3*cosd(q3); sind(q3) cosd(q3) 0 a3*sind(q3); 0 0 1 0; 0 0
0 1];
disp(t3)
t4 = [cosd(q4) 0 sind(q4) 0; sind(q4) 0 -cosd(q4) 0; 0 1 0 0; 0 0 0 1];
disp(t4)
t5 = [cosd(q5) -sind(q5) 0 0; sind(q5) cosd(q5) 0 0; 0 0 1 a4+d5; 0 0 0 1];
disp(t5)
t = t1*t2*t3*t4*t5;
disp(t)
x=t(1,4)
y=t(2,4)
z=t(3,4)

% Inverse Kinematics of a 5 DOF Robotic Arm

d1 = 1;
a2 = 10.5;
a3 = 14.7;
d4 = 7.6;
d5 = 11;
x = input('Input x location:');
y = input('Input y location:');
z = input('Input z location:');
q1=atan2(y,x);
q1=real(q1);
q3=acos((x^2+y^2+((z-d1)^2)-(a2+a3)^2-(d4+d5)^2)/(2*(a2+a3)*(d4+d5)));
q3=real(q3);
q2=atan2(z-d1,sqrt(x^2+y^2))-atan2(sin(q3)*(d4+d5),(a2+a3)+cos(q3)*(d4+d5));
q2=real(q2);
q1=q1*180/pi;
q2=q2*180/pi;
q3=q3*180/pi;
disp(['q1(in degrees)=' num2str(q1)]);
disp(['q2(in degrees)=' num2str(q2)]);
disp(['q3(in degrees)=' num2str(q3)]);
```

Results for Forward and Inverse Kinematic

```
>> dof5_forward_kinematics
Joint angle 1:90
Joint angle 2:90
Joint angle 3:45
Joint angle 4:45
Joint angle 5:60
    1.5708

    1.5708

    0.7854

    2.3562
```

```
    1.0472

[cos((pi*q1)/180), 0,  sin((pi*q1)/180), 0]
[sin((pi*q1)/180), 0, -cos((pi*q1)/180), 0]
[                0, 1,                 0, 1]
[                0, 0,                 0, 1]

[cos((pi*q2)/180), -sin((pi*q2)/180), 0, (21*cos((pi*q2)/180))/2]
[sin((pi*q2)/180),  cos((pi*q2)/180), 0, (21*sin((pi*q2)/180))/2]
[                0,                 0, 1,                        0]
[                0,                 0, 0,                        1]

[cos((pi*q3)/180), -sin((pi*q3)/180), 0, (147*cos((pi*q3)/180))/10]
[sin((pi*q3)/180),  cos((pi*q3)/180), 0, (147*sin((pi*q3)/180))/10]
[                0,                 0, 1,                          0]
[                0,                 0, 0,                          1]

[cos((pi*q4)/180), 0,  sin((pi*q4)/180), 0]
[sin((pi*q4)/180), 0, -cos((pi*q4)/180), 0]
[                0, 1,                 0, 0]
[                0, 0,                 0, 1]

[cos((pi*q5)/180), -sin((pi*q5)/180), 0,    0]
[sin((pi*q5)/180),  cos((pi*q5)/180), 0,    0]
[                0,                 0, 1, 93/5]
[                0,                 0, 0,    1]

[  sin((pi*q1)/180)*sin((pi*q5)/180) - 
cos((pi*q5)/180)*(cos((pi*q4)/180)*(cos((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180
) - cos((pi*q1)/180)*cos((pi*q2)/180)*cos((pi*q3)/180)) + 
sin((pi*q4)/180)*(cos((pi*q1)/180)*cos((pi*q2)/180)*sin((pi*q3)/180) + 
cos((pi*q1)/180)*cos((pi*q3)/180)*sin((pi*q2)/180))), 
cos((pi*q5)/180)*sin((pi*q1)/180) + 
sin((pi*q5)/180)*(cos((pi*q4)/180)*(cos((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180
) - cos((pi*q1)/180)*cos((pi*q2)/180)*cos((pi*q3)/180)) + 
sin((pi*q4)/180)*(cos((pi*q1)/180)*cos((pi*q2)/180)*sin((pi*q3)/180) + 
cos((pi*q1)/180)*cos((pi*q3)/180)*sin((pi*q2)/180))), 
cos((pi*q4)/180)*(cos((pi*q1)/180)*cos((pi*q2)/180)*sin((pi*q3)/180) + 
cos((pi*q1)/180)*cos((pi*q3)/180)*sin((pi*q2)/180)) - 
sin((pi*q4)/180)*(cos((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180) - 
cos((pi*q1)/180)*cos((pi*q2)/180)*cos((pi*q3)/180)), 
(93*cos((pi*q4)/180)*(cos((pi*q1)/180)*cos((pi*q2)/180)*sin((pi*q3)/180) + 
cos((pi*q1)/180)*cos((pi*q3)/180)*sin((pi*q2)/180)))/5 - 
(93*sin((pi*q4)/180)*(cos((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180) - 
cos((pi*q1)/180)*cos((pi*q2)/180)*cos((pi*q3)/180)))/5 + 
(21*cos((pi*q1)/180)*cos((pi*q2)/180))/2 - 
(147*cos((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180))/10 + 
(147*cos((pi*q1)/180)*cos((pi*q2)/180)*cos((pi*q3)/180))/10]
[- 
cos((pi*q5)/180)*(cos((pi*q4)/180)*(sin((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180
) - cos((pi*q2)/180)*cos((pi*q3)/180)*sin((pi*q1)/180)) + 
sin((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q1)/180)*sin((pi*q3)/180) + 
cos((pi*q3)/180)*sin((pi*q1)/180)*sin((pi*q2)/180))) - 
cos((pi*q1)/180)*sin((pi*q5)/180), 
sin((pi*q5)/180)*(cos((pi*q4)/180)*(sin((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180
```

```
) - cos((pi*q2)/180)*cos((pi*q3)/180)*sin((pi*q1)/180)) +
sin((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q1)/180)*sin((pi*q3)/180) +
cos((pi*q3)/180)*sin((pi*q1)/180)*sin((pi*q2)/180))) -
cos((pi*q1)/180)*cos((pi*q5)/180),
cos((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q1)/180)*sin((pi*q3)/180) +
cos((pi*q3)/180)*sin((pi*q1)/180)*sin((pi*q2)/180)) -
sin((pi*q4)/180)*(sin((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180) -
cos((pi*q2)/180)*cos((pi*q3)/180)*sin((pi*q1)/180)),
(93*cos((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q1)/180)*sin((pi*q3)/180) +
cos((pi*q3)/180)*sin((pi*q1)/180)*sin((pi*q2)/180)))/5 -
(93*sin((pi*q4)/180)*(sin((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180) -
cos((pi*q2)/180)*cos((pi*q3)/180)*sin((pi*q1)/180)))/5 +
(21*cos((pi*q2)/180)*sin((pi*q1)/180))/2 -
(147*sin((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180))/10 +
(147*cos((pi*q2)/180)*cos((pi*q3)/180)*sin((pi*q1)/180))/10]
[
cos((pi*q5)/180)*(cos((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q3)/180) +
cos((pi*q3)/180)*sin((pi*q2)/180)) +
sin((pi*q4)/180)*(cos((pi*q2)/180)*cos((pi*q3)/180) -
sin((pi*q2)/180)*sin((pi*q3)/180))),
-sin((pi*q5)/180)*(cos((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q3)/180) +
cos((pi*q3)/180)*sin((pi*q2)/180)) +
sin((pi*q4)/180)*(cos((pi*q2)/180)*cos((pi*q3)/180) -
sin((pi*q2)/180)*sin((pi*q3)/180))),
sin((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q3)/180) +
cos((pi*q3)/180)*sin((pi*q2)/180)) -
cos((pi*q4)/180)*(cos((pi*q2)/180)*cos((pi*q3)/180) -
sin((pi*q2)/180)*sin((pi*q3)/180)),
(21*sin((pi*q2)/180))/2 - (93*cos((pi*q4)/180)*(cos((pi*q2)/180)*cos((pi*q3)/180) -
sin((pi*q2)/180)*sin((pi*q3)/180)))/5 +
(93*sin((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q3)/180) +
cos((pi*q3)/180)*sin((pi*q2)/180)))/5 + (147*cos((pi*q2)/180)*sin((pi*q3)/180))/10 +
(147*cos((pi*q3)/180)*sin((pi*q2)/180))/10 + 1]
[
0,
0,
0,
1]


x =

(93*cos((pi*q4)/180)*(cos((pi*q1)/180)*cos((pi*q2)/180)*sin((pi*q3)/180) +
cos((pi*q1)/180)*cos((pi*q3)/180)*sin((pi*q2)/180)))/5 -
(93*sin((pi*q4)/180)*(cos((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180) -
cos((pi*q1)/180)*cos((pi*q2)/180)*cos((pi*q3)/180)))/5 +
(21*cos((pi*q1)/180)*cos((pi*q2)/180))/2 -
(147*cos((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180))/10 +
(147*cos((pi*q1)/180)*cos((pi*q2)/180)*cos((pi*q3)/180))/10


y =

(93*cos((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q1)/180)*sin((pi*q3)/180) +
cos((pi*q3)/180)*sin((pi*q1)/180)*sin((pi*q2)/180)))/5 -
```

```
(93*sin((pi*q4)/180)*(sin((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180) -
cos((pi*q2)/180)*cos((pi*q3)/180)*sin((pi*q1)/180)))/5 +
(21*cos((pi*q2)/180)*sin((pi*q1)/180))/2 -
(147*sin((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180))/10 +
(147*cos((pi*q2)/180)*cos((pi*q3)/180)*sin((pi*q1)/180))/10


z =

(21*sin((pi*q2)/180))/2 - (93*cos((pi*q4)/180)*(cos((pi*q2)/180)*cos((pi*q3)/180) -
sin((pi*q2)/180)*sin((pi*q3)/180)))/5 +
(93*sin((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q3)/180) +
cos((pi*q3)/180)*sin((pi*q2)/180)))/5 + (147*cos((pi*q2)/180)*sin((pi*q3)/180))/10 +
(147*cos((pi*q3)/180)*sin((pi*q2)/180))/10 + 1

Input x location:35
Input y location:45
Input z location:12
q1(in degrees)=52.125
q2(in degrees)=10.9212
q3(in degrees)=0
```