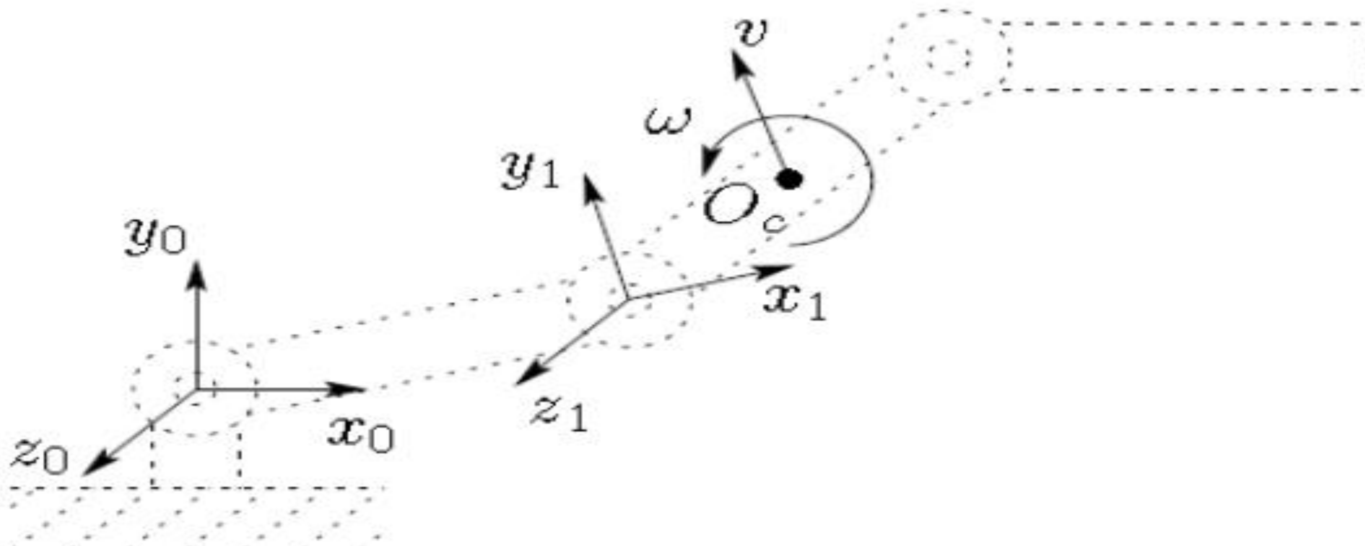
Q2)

**2)** For the following 3-link planar 3R manipulator, compute the vector $O_c$ and derive the manipulator geometric Jacobian matrix.




TE = R(q1) $T_x$(a1) R(q2) $T_x$(a2) R(q3) $T_x$(a3)

$T_E =$

$$\begin{pmatrix} \cos(q1+q2+q3) & -\sin(q1+q2+q3) & a2\cos(q1+q2)+a1\cos(q1)+a3\cos(q1+q2+q3) \\ \sin(q1+q2+q3) & \cos(q1+q2+q3) & a2\sin(q1+q2)+a1\sin(q1)+a3\sin(q1+q2+q3) \\ 0 & 0 & 1 \end{pmatrix}$$

➔

**X = a2 cos(q1+q2) + a1 cos(q1) + a3cos(q1+q2+q3)**

**Y = a2sin(q1+q2) + a1 sin(q1) + a3 sin(q1+q2+q3)**

**One should take the derivative and arrange them into a matrix format:**

$$\begin{pmatrix} \dot{X} \\ \dot{Y} \end{pmatrix} =$$

$$\begin{pmatrix} -a2\,sin(q1+q2) - a1 sinq1 - a3sin(q1+q2+q3) & -a2sin(q1+q2) - a3\sin(q1+q2+q3) & -a3\,sin(q1+q2+q3) \\ a2\,cos(q1+q2) + a1\,cos(q1) + a3\,cos(q1+q2+q3) & a2\,cos(q1+q2) + a3\,cos(q1+q2+q3) & a3\,cos(q1+q2+q3) \end{pmatrix}$$
$$\begin{pmatrix} \dot{q}1 \\ \dot{q}2 \\ \dot{q}3 \end{pmatrix}$$

$$R = \begin{pmatrix} cos(q1+q2+q3) & -sin(q1+q2+q3) \\ sin(q1+q2+q3) & cos(q1+q2+q3) \end{pmatrix} \Rightarrow R = \begin{pmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{pmatrix}$$

$$\theta = q1 + q2 + q3$$

**Derivative of θ** $\Rightarrow \dot{\theta} = \dot{q}1 + \dot{q}2 + \dot{q}3 \Rightarrow \dot{\theta}\ (1\ 1\ 1)\begin{pmatrix} \dot{q}1 \\ \dot{q}2 \\ \dot{q}3 \end{pmatrix}$

**Combining these two derivatives, give us:**

Spatial Velocity

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} =$$

$$\begin{pmatrix} -a2\ sin(q1+q2) - a1sinq1 - a3sin(q1+q2+q3) & -a2sin(q1+q2) - a3sin\ (q1+q2+q3) & -a3\ sin(q1+q2+q3) \\ a2\ cos(q1+q2) + a1\ cos(q1) + a3\ cos(q1+q2+q3) & a2\ cos(q1+q2) + a3\ cos(q1+q2+q3) & a3\ cos(q1+q2+q3) \\ 1 & 1 & 1 \end{pmatrix}$$

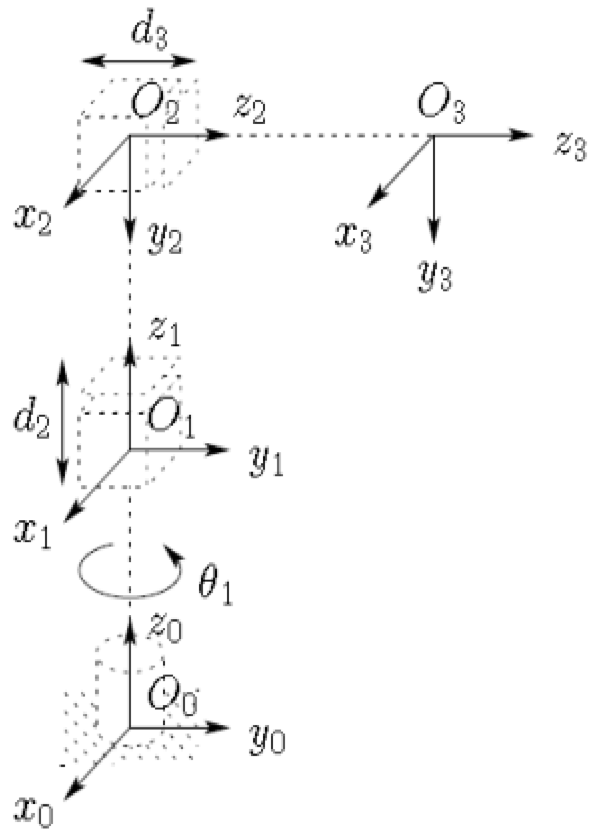
$$\begin{pmatrix} \dot{q}1 \\ \dot{q}2 \\ \dot{q}3 \end{pmatrix}$$

Joint Velocity

V or QC vector

Q3)

**3)** For the cylindrical manipulator derive the 6x3 Jacobian matrix.
Show that there are no singular configurations for this arm. Thus the only singularities for the cylindrical manipulator must come from the wrist.



$$T_{03} = \begin{bmatrix} c1 & 0 & -s1 & -s1d3 \\ s1 & 0 & c1 & c1d3 \\ 0 & -1 & 0 & d1+d2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$J = \begin{matrix} \hat{z}_0 \times {}^0t3 & \hat{z}_1 & \hat{z}_2 \\ \hat{Z}_0 & 0 & 0 \end{matrix} \quad \rightarrow$$

$$J = \begin{bmatrix} -c1d3 & 0 & -s1 \\ -s1d3 & 0 & c1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

J is singular in $R^3$ since the robot has only 3 Degree of Freedom (DOF) and the position singularities occur when determinant $\begin{bmatrix} -c1d3 & 0 & -s1 \\ -s1d3 & 0 & c1 \\ 0 & 1 & 0 \end{bmatrix} = (c_1^2 + s_1^2)\ d3 = 0$ , therefore position singularities occurs when $d3 = 0$

Q4)

**4)** Complete the derivation of the Jacobian matrix for Stanford Arm (following Example 4.7 in the textbook).

Considering the Stanford Manipulator and its DH coordinate frames. The third joint is prismatic and $0_3$, $0_4$, $0_5$ are equal because of the spherical wrist and the frame assignment. The columns of Jacobian have the form of:

$J_i =$ $Z_{i-1} \times (O_6 - O_{i-1})$ $I = 1, 2$

$Z_{i-1}$

$J_3 =$ $Z_2$

$0$

$J_i = \begin{matrix} Z_{i-1} \times (O_6 - O) \\ Z_{i-1} \end{matrix}$

T matrices are the results of the product of the A matrices. The first one is $O_j$, which is driven from the first three entries of the last column $T_j^0 = A_1 \ldots A_j$ with $O_0 = (0,0,0) = O_1$. The vector of $Z_j = R_j^0 k$ where $R_j^0$ is the rotational part of $T_j^0$. Therefore, we calculate the Jacobian for $T_j^0$ for the Stanford Manipulator, which is shown below:

$$O_6 = \begin{bmatrix} c1s2d3 - s1d2 + d6(c1c2c4s5 + c1c5s2 - s1s4s5) \\ s1s2d3 - c1d2 + d6(c1s4s5 + c2c4s1s5 + c5s1s2) \\ c2d3 + d6(c2c5 - c4s2s5) \end{bmatrix}$$

$$O_3 = \begin{bmatrix} c1s2d3 - s1d2 \\ s1s2d3 + c1d2 \\ c2d3 \end{bmatrix}$$

$Z_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ $Z_1 = \begin{bmatrix} -s1 \\ c1 \\ 0 \end{bmatrix}$ $Z2 = \begin{bmatrix} c1s2 \\ s1s2 \\ c2 \end{bmatrix}$ $Z3 = \begin{bmatrix} c1s2 \\ s1s2 \\ c2 \end{bmatrix}$ $z4 = \begin{bmatrix} -c1s2s4 - s1c4 \\ -s1c2s4 + c1c4 \\ s2s4 \end{bmatrix}$ $z5 = \begin{bmatrix} c1c2c4s5 - s1s4s5 + c1s2c5 \\ s1c2c4c5 + c1s4s5 + s1s2c5 \\ -s2c4s5 + c2c5 \end{bmatrix}$

I used MATLAB to generate the Jacobian matrix for Stanford Arm:

```
% Calculations for example 4.7
A0 = [1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
syms s1 s2 c1 c2
A1 = [c1 0 -s1 0;s1 0 c1 0;0 -1 0 0;0 0 0 1];
syms d2
A2 = [c2 0 s2 0;s2 0 -c2 0;0 1 0 d2;0 0 0 1];
syms d3
A3 = [1 0 0 0;0 1 0 0;0 0 1 d3;0 0 0 1];
syms c4 s4
A4 = [c4 0 -s4 0;s4 0 c4 0;0 -1 0 0;0 0 0 1];
syms c5 s5
A5 = [c5 0 s5 0;s5 0 -c5 0;0 -1 0 0;0 0 0 1];
syms c6 s6 d6
A6 = [c6 -s6 0 0 ;s6 c6 0 0;0 0 1 d6;0 0 0 1];
% z_i occurs in the third column of the rotation matrix component of T_0^i
 z0 = [0 0 1];
 T01 = A1;
 z1 = A1(1:3,3);
 T02 = A1*A2;
 z2 = T02(1:3,3);
 T03 = A1*A2*A3;
 z3 = T03(1:3,3);
 T04 = A1*A2*A3*A4;
 z4 = T04(1:3,3);
 T05 = A1*A2*A3*A4*A5;
```

```
z5 = T05(1:3,3);
T06 = A1*A2*A3*A4*A5*A6;
z6 = T06(1:3,3);

% the origins of frames 1 through 6
o0 = T01(1:3,4);
disp(o0)
o1 = T01(1:3,4);
disp(o1)
o2 = T02(1:3,4);
disp(o2)
o3 = T03(1:3,4);
disp(o3)
o4 = T04(1:3,4);
disp(o4)
o5 = T05(1:3,4);
disp(o5)
o6 = T06(1:3,4);
disp(o6)


% Jacobian matrices

J1 = [cross(z0,(o6-o0));z0];
disp(J1)
J2 = [cross(z1,(o6-o1));z1];
disp(J2)
%Joint 3 is prismatic whereas the others are revolute.
J3 = [cross(z2,(o6-o2));z2];
disp(J3)
J4 = [cross(z3,(o6-o3));z3];
disp(J4)
J5 = [cross(z4,(o6-o4));z4];
disp(J5)
J6 = [cross(z5,(o6-o5));z5];
disp(J6)
% finally, this is the result for jacobian matrix for Stanford Arm
J = [J1 J2 J3 J4 J5 J6];
```

## Results

**A1 =**

**[ c1, 0, -s1, 0]**
**[ s1, 0, c1, 0]**
**[ 0, -1, 0, 0]**
**[ 0, 0, 0, 1]**

**A2 =**

**[ c2, 0, s2, 0]**
**[ s2, 0, -c2, 0]**
**[ 0, 1, 0, d2]**
**[ 0, 0, 0, 1]**

**A3 =**

**[ 1, 0, 0, 0]**
**[ 0, 1, 0, 0]**
**[ 0, 0, 1, d3]**
**[ 0, 0, 0, 1]**

**A4 =**
**[ c4, 0, -s4, 0]**
**[ s4, 0, c4, 0]**
**[ 0, -1, 0, 0]**
**[ 0, 0, 0, 1]**

**A5 =**
**[ c5, 0, s5, 0]**
**[ s5, 0, -c5, 0]**

**[ 0, -1, 0, 0]**
**[ 0, 0, 0, 1]**

**A6 =**
**[ c6, -s6, 0, 0]**
**[ s6, c6, 0, 0]**
**[ 0, 0, 1, d6]**
**[ 0, 0, 0, 1]**

**T01 =**
**[ c1, 0, -s1, 0]**
**[ s1, 0, c1, 0]**
**[ 0, -1, 0, 0]**
**[ 0, 0, 0, 1]**

**T02 =**
**[ c1*c2, -s1, c1*s2, -d2*s1]**
**[ c2*s1, c1, s1*s2, c1*d2]**
**[ -s2, 0, c2, 0]**
**[ 0, 0, 0, 1]**

**T03 =**
**[ c1*c2, -s1, c1*s2, c1*d3*s2 - d2*s1]**
**[ c2*s1, c1, s1*s2, c1*d2 + d3*s1*s2]**
**[ -s2, 0, c2, c2*d3]**
**[ 0, 0, 0, 1]**

**T04 =**
**[ c1*c2*c4 - s1*s4, -c1*s2, - c4*s1 - c1*c2*s4, c1*d3*s2 - d2*s1]**
**[ c1*s4 + c2*c4*s1, -s1*s2, c1*c4 - c2*s1*s4, c1*d2 + d3*s1*s2]**
**[ -c4*s2, -c2, s2*s4, c2*d3]**
**[ 0, 0, 0, 1]**

**T05 =**

**[- c5*(s1*s4 - c1*c2*c4) - c1*s2*s5, c4*s1 + c1*c2*s4, c1*c5*s2 - s5*(s1*s4 - c1*c2*c4), c1*d3*s2 - d2*s1]**

**[ c5*(c1*s4 + c2*c4*s1) - s1*s2*s5, c2*s1*s4 - c1*c4, s5*(c1*s4 + c2*c4*s1) + c5*s1*s2, c1*d2 + d3*s1*s2]**

**[- c2*s5 - c4*c5*s2, -s2*s4, c2*c5 - c4*s2*s5, c2*d3]**

**[0,0,0,1]**

**T06 =**

**[s6*(c4*s1 + c1*c2*s4) - c6*(c5*(s1*s4 - c1*c2*c4) + c1*s2*s5),   s6*(c5*(s1*s4 - c1*c2*c4) + c1*s2*s5) + c6*(c4*s1 + c1*c2*s4), c1*c5*s2 - s5*(s1*s4 - c1*c2*c4), c1*d3*s2 - d6*(s5*(s1*s4 - c1*c2*c4) - c1*c5*s2) - d2*s1]**

**[c6*(c5*(c1*s4 + c2*c4*s1) - s1*s2*s5) - s6*(c1*c4 - c2*s1*s4), - s6*(c5*(c1*s4 + c2*c4*s1) - s1*s2*s5) - c6*(c1*c4 - c2*s1*s4), s5*(c1*s4 + c2*c4*s1) + c5*s1*s2, c1*d2 + d6*(s5*(c1*s4 + c2*c4*s1) + c5*s1*s2) + d3*s1*s2]**

**[ - c6*(c2*s5 + c4*c5*s2) - s2*s4*s6, s6*(c2*s5 + c4*c5*s2) - c6*s2*s4, c2*c5 - c4*s2*s5, c2*d3 + d6*(c2*c5 - c4*s2*s5)]**

**[ 0,0,0,1]**

**J1 =**

**- c1*d2 - d6*(s5*(c1*s4 + c2*c4*s1) + c5*s1*s2) - d3*s1*s2**

**c1*d3*s2 - d6*(s5*(s1*s4 - c1*c2*c4) - c1*c5*s2) - d2*s1**

**0**

**0**

**0**

**1**

**J2 =**

**c1*(c2*d3 + d6*(c2*c5 - c4*s2*s5))**

**s1*(c2*d3 + d6*(c2*c5 - c4*s2*s5))**

**c1*(d2*s1 + d6*(s5*(s1*s4 - c1*c2*c4) - c1*c5*s2) - c1*d3*s2) - s1*(c1*d2 + d6*(s5*(c1*s4 + c2*c4*s1) + c5*s1*s2) + d3*s1*s2)**

**-s1**

**c1**

**0**

**J3 =**

s1*s2*(c2*d3 + d6*(c2*c5 - c4*s2*s5)) - c2*(d6*(s5*(c1*s4 + c2*c4*s1) + c5*s1*s2) + d3*s1*s2)
- c2*(d6*(s5*(s1*s4 - c1*c2*c4) - c1*c5*s2) - c1*d3*s2) - c1*s2*(c2*d3 + d6*(c2*c5 - c4*s2*s5))
c1*s2*(d6*(s5*(c1*s4 + c2*c4*s1) + c5*s1*s2) + d3*s1*s2) + s1*s2*(d6*(s5*(s1*s4 - c1*c2*c4) - c1*c5*s2) - c1*d3*s2)
c1*s2
s1*s2
c2

J4 =

d6*s1*s2*(c2*c5 - c4*s2*s5) - c2*d6*(s5*(c1*s4 + c2*c4*s1) + c5*s1*s2)
- c2*d6*(s5*(s1*s4 - c1*c2*c4) - c1*c5*s2) - c1*d6*s2*(c2*c5 - c4*s2*s5)
c1*d6*s2*(s5*(c1*s4 + c2*c4*s1) + c5*s1*s2) + d6*s1*s2*(s5*(s1*s4 - c1*c2*c4) - c1*c5*s2)
c1*s2
s1*s2
c2

J5 =

d6*(c1*c4 - c2*s1*s4)*(c2*c5 - c4*s2*s5) - d6*s2*s4*(s5*(c1*s4 + c2*c4*s1) + c5*s1*s2)
d6*(c4*s1 + c1*c2*s4)*(c2*c5 - c4*s2*s5) - d6*s2*s4*(s5*(s1*s4 - c1*c2*c4) - c1*c5*s2)
d6*(c1*c4 - c2*s1*s4)*(s5*(s1*s4 - c1*c2*c4) - c1*c5*s2) - d6*(s5*(c1*s4 + c2*c4*s1) + c5*s1*s2)*(c4*s1 + c1*c2*s4)
- c4*s1 - c1*c2*s4
c1*c4 - c2*s1*s4
s2*s4

J6 =

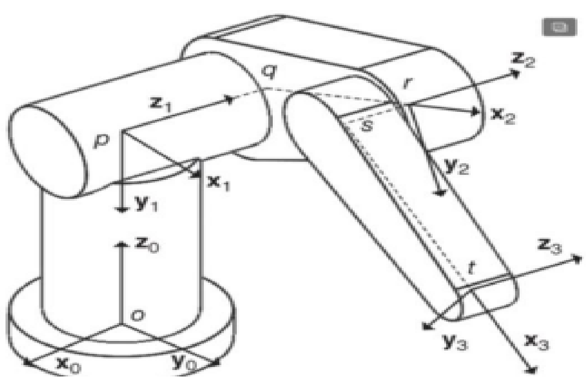
0
0
0
c1*c5*s2 - s5*(s1*s4 - c1*c2*c4)
s5*(c1*s4 + c2*c4*s1) + c5*s1*s2
c2*c5 - c4*s2*s5

# Jacobian matrix is

J = [J1 J2 J3 J4 J5 J6];

Q8)

**8)** For the following spherical RRR robotic manipulator, define the coordinate systems, list the D-H parameters table, using the developed T matrix, show the transformation matrices $T_{i-1}^{i}$ and $T_0^3$, find the forward kinematics for both manipulators, and develop the inverse kinematics. For the purpose of verification, assume all the lengths are 50 and all the rotational angles are 30 degree. (Write a Matlab program for this problem).



**MATLAB code for DH parameters, forward and inverse kinematics**

```
% Creating kinematic model of RRR robot

%            theta      d    a    alpha
clear L
L(1) = Link([pi/3      0     50     0    ]);
L(2) = Link([pi/3      0     50   -pi/2  ]);
L(3) = Link([pi/3      0     50    pi/2  ]);
L(4) = Link([pi/3      0     50   -pi/2  ]);
%###########################################################
%Pose 0; At MASTERING position;
%###########################################################
q0 =[0   -pi/2   0   0];
R=SerialLink(L, 'name', 'RRRrobot');
%###########################################################

% 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:');


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)


syms q1 q2 q3 q4 q5 d1 a2 a3 a4 d5;
d1=10;
```

```
a1=50;
a2=50;
a3=50;
a4=50;
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)

t = t1*t2*t3*t4;
disp(t)
x=t(1,4);
y=t(2,4);
z=t(3,4);

% Inverse Kinematics of a 5 DOF Robotic Arm

d1 = 10;
a1=50;
a2 = 50;
a3 = 50;
d4 = 50;
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:

```
L =
Revolute(std):  theta=q1   d=0        a=50       alpha=0          offset=0
Revolute(std):  theta=q2   d=0        a=50       alpha=-1.571     offset=0
Revolute(std):  theta=q3   d=0        a=50       alpha=1.571      offset=0
Revolute(std):  theta=q4   d=0        a=50       alpha=-1.571     offset=0
>> R
```

# #DH Parameter Table

R =

Dexterrobot:: 4 axis, RRRR, stdDH, slowRNE

```
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|          0|         50|          0|          0|
|  2|         q2|          0|         50|    -1.5708|          0|
|  3|         q3|          0|         50|     1.5708|          0|
|  4|         q4|          0|         50|    -1.5708|          0|
+---+-----------+-----------+-----------+-----------+-----------+
```

# Calculating the forward kinematics
>> R.fkine(q0)

ans =

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 50 |
| -1 | 0 | 0 | -150 |
| 0 | -1 | 0 | 0 |
| 0 | 0 | 0 | 1 |

# Calculating the Inverse Kinematics

Joint angle 1:30
Joint angle 2:30
Joint angle 3:30
Joint angle 4:30
0.5236

0.5236

0.5236

2.0944

```
[cos((pi*q1)/180), 0,  sin((pi*q1)/180),  0]
[sin((pi*q1)/180), 0, -cos((pi*q1)/180),  0]
[               0, 1,                 0, 10]
[               0, 0,                 0,  1]
```

```
[cos((pi*q2)/180), -sin((pi*q2)/180), 0, 50*cos((pi*q2)/180)]
[sin((pi*q2)/180),  cos((pi*q2)/180), 0, 50*sin((pi*q2)/180)]
[                0,                 0, 1,                   0]
[                0,                 0, 0,                   1]

[cos((pi*q3)/180), -sin((pi*q3)/180), 0, 50*cos((pi*q3)/180)]
[sin((pi*q3)/180),  cos((pi*q3)/180), 0, 50*sin((pi*q3)/180)]
[                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*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)),  sin((pi*q1)/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)),
50*cos((pi*q1)/180)*cos((pi*q2)/180) - 50*cos((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180) +
50*cos((pi*q1)/180)*cos((pi*q2)/180)*cos((pi*q3)/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*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)),
50*cos((pi*q2)/180)*sin((pi*q1)/180) - 50*sin((pi*q1)/180)*sin((pi*q2)/180)*sin((pi*q3)/180) +
50*cos((pi*q2)/180)*cos((pi*q3)/180)*sin((pi*q1)/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)),                 0,
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)),                                50*sin((pi*q2)/180) + 50*cos((pi*q2)/180)*sin((pi*q3)/180) +
50*cos((pi*q3)/180)*sin((pi*q2)/180) + 10]
```

[ 0,
0,
0, 1]

Input x location:10
Input y location:10
Input z location:10

**q1(in degrees)=45**
**q2(in degrees)=-1.0975e-14**
**q3(in degrees)=180**

## Project Second part Inverse kinematic

```
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:');


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)
```

```
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)]);
```

## Result:

>> **dof5_forward_kinematics**

**Joint angle 1:90**
**Joint angle 2:45**
**Joint angle 3:30**
**Joint angle 4:45**

**Joint angle 5:90**

```
   1.5708

   0.7854

   0.5236

   2.3562

   1.5708

[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]

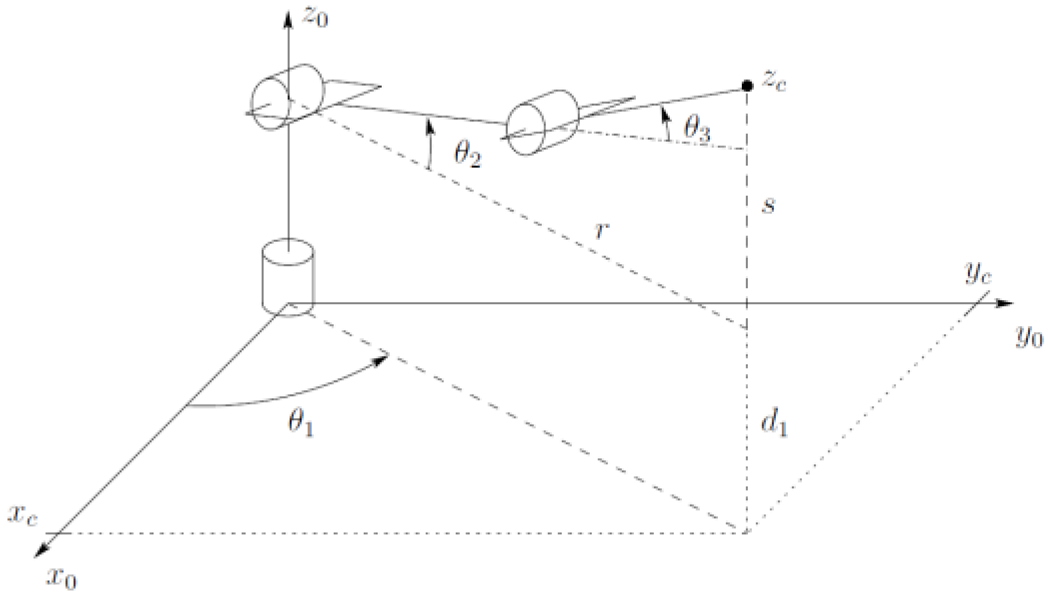
**[**
**cos((pi*q5)/180)*(cos((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q3)/180) + cos((pi*q3)/180)*sin((pi*q2)/180)) + sin((pi*q4)/180)*(cos((pi*q2)/180)*cos((pi*q3)/180) - sin((pi*q2)/180)*sin((pi*q3)/180))),**
**-sin((pi*q5)/180)*(cos((pi*q4)/180)*(cos((pi*q2)/180)*sin((pi*q3)/180) + cos((pi*q3)/180)*sin((pi*q2)/180)) + 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]**

**Input x location:10**
**Input y location:12**
**Input z location:9**

**q1(in degrees)=50.1944**
**q2(in degrees)=-20.4239**
**q3(in degrees)=135.8824**

## Q9)

**9)** Write a general Matlab program to calculate the Jacobian matrix for any serial robot. Using the following robot to verify your results. Also, find the singularities for the robot. [**Bonus** +10%]




**Answer:**

```
% Calculations for serial robot
A0 = [1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
syms s1 s2 c1 c2
A1 = [c1 0 -s1 0;s1 0 c1 0;0 -1 0 0;0 0 0 1];
syms d2
A2 = [c2 0 s2 0;s2 0 -c2 0;0 1 0 d2;0 0 0 1];
syms d3
A3 = [1 0 0 0;0 1 0 0;0 0 1 d3;0 0 0 1];
```

```
% z_i occurs in the third column of the rotation matrix component of T_0^i
 z0 = [0 0 1];
 T01 = A1;
 z1 = A1(1:3,3);
 T02 = A1*A2;
 z2 = T02(1:3,3);
 T03 = A1*A2*A3;
 z3 = T03(1:3,3);



 % the origins of frames 1 through 6

o0 = T01(1:3,4);
 disp(o0)
 o1 = T01(1:3,4);
 disp(o1)
```

**Results**

```
-d2*s1
 c1*d2
   0

c1*d3*s2 - d2*s1
c1*d2 + d3*s1*s2
       c2*d3

[- c1*d2 - d3*s1*s2, c1*d3*s2 - d2*s1, 0]
[                 0,                0, 1]

                         c1*c2*d3
                         c2*d3*s1
c1*(d2*s1 - c1*d3*s2) - s1*(c1*d2 + d3*s1*s2)
```

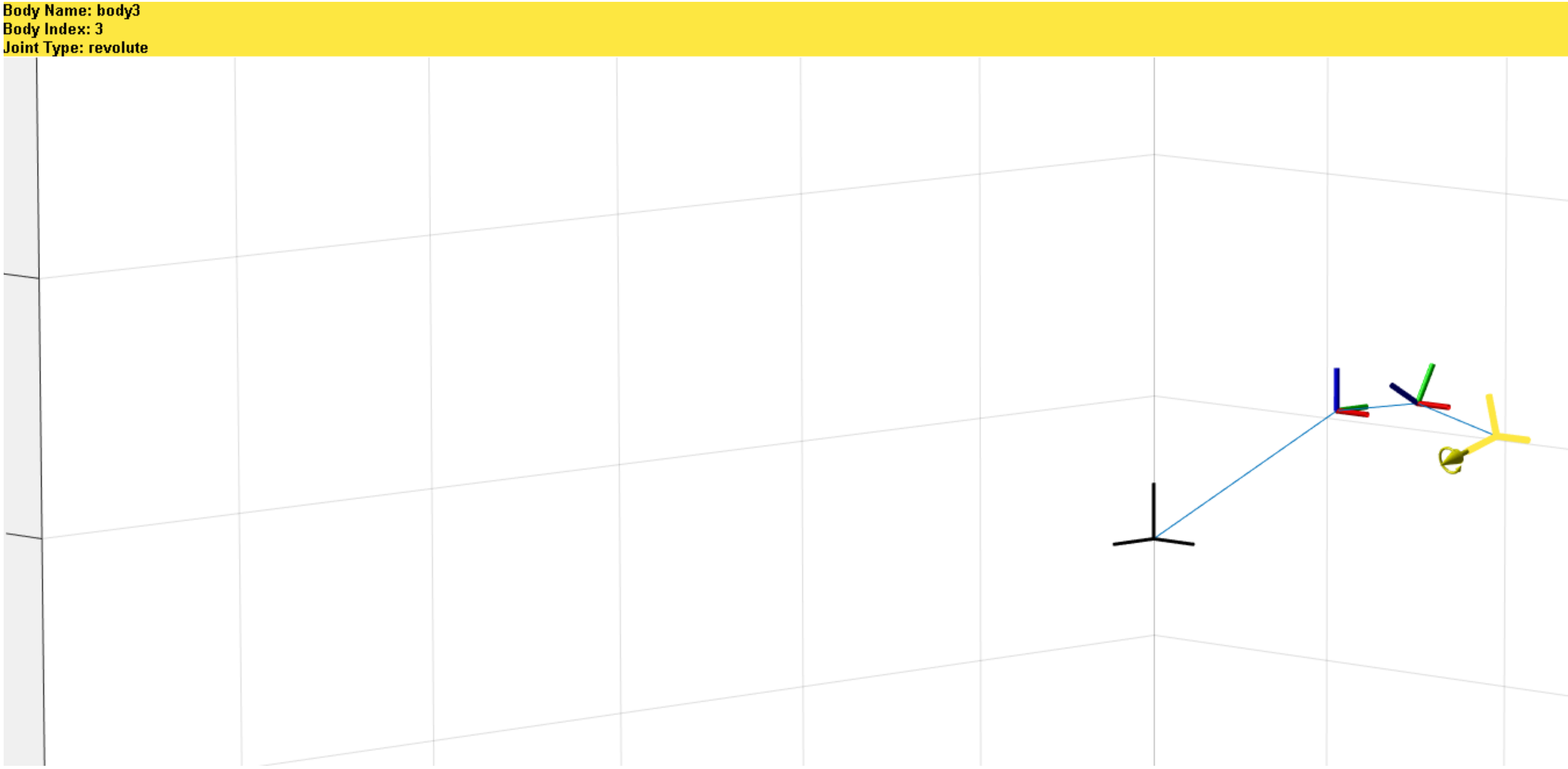
**-s1**
**c1**
**0**

**0**
**0**
**0**
**c1*s2**
**s1*s2**
**c2**

**Simulation of the serial robot**

```
robot = robotics.RigidBodyTree('DataFormat', 'column');
% add first body (fixed, with an offset from ICS)
body1 = robotics.RigidBody('body1');
body1.Joint = robotics.Joint('joint1', 'revolute');
T = trvec2tform([-0.5, 0 0.2])*eul2tform([pi/2 0 0]);
body1.Joint.setFixedTransform(T);
robot.addBody(body1, robot.BaseName);
% add second body (revolute joint)
body2 = robotics.RigidBody('body2');
body2.Joint = robotics.Joint('joint2', 'revolute');
body2.Joint.setFixedTransform([0.3 pi/3 0.1 0], 'mdh');
robot.addBody(body2, 'body1');
% add third body (revolute joint)
body3 = robotics.RigidBody('body3');
body3.Joint = robotics.Joint('joint3', 'revolute');
body3.Joint.setFixedTransform([0.3 pi/4 0.1 0], 'mdh');
robot.addBody(body3, 'body2');
```

**robot.show**




Singular configuration is associated with Jacobian matrix determinant becoming zero and therefore there is no inverse exits meaning that the robot has lost a DOF. This can be observed during the Jacobian inversion computation, when it produces lager joint velocities or delta q which is not acceptable. This is not the case for this problem and no singularities can be identified.

# Project 2

In the second part of the project, we will discuss the calculation of inverse kinematics of 5DOF robotics arm as well as other functions such as Jacobian Matrix and simulation of the robotic arm in space.

Inverse kinematics means the application of kinematic equations to define the motion of a robot to reach a desired position, examples associated with inverse kinematics are automated fruit picking, a robotic arm used for welding or painting which require precise motion from an initial position to a desired position.

In the next part of the project, we will look at the Jacobian matrix. Jacobian is a matrix which provides the relation between joint velocities ($\dot{q}$) & end-effector velocities ( $\dot{x}$) of a robot manipulator.

By figuring out the velocities of the joints of the robot, one will be able to estimate the velocity of the end effector.

The relation between joint velocities and end-effector velocities can be shown by below formula:

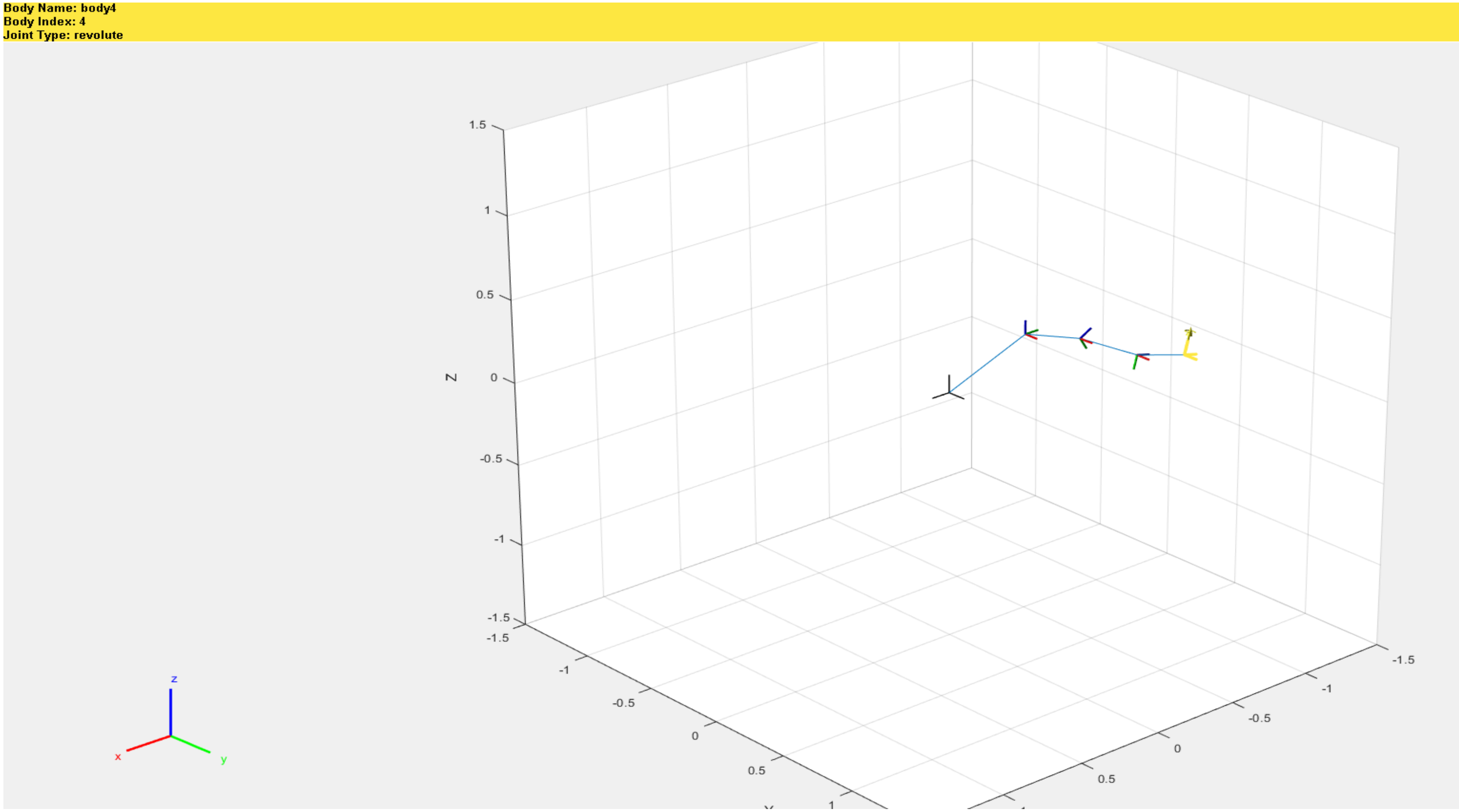
$$\dot{X} = J\dot{q}$$

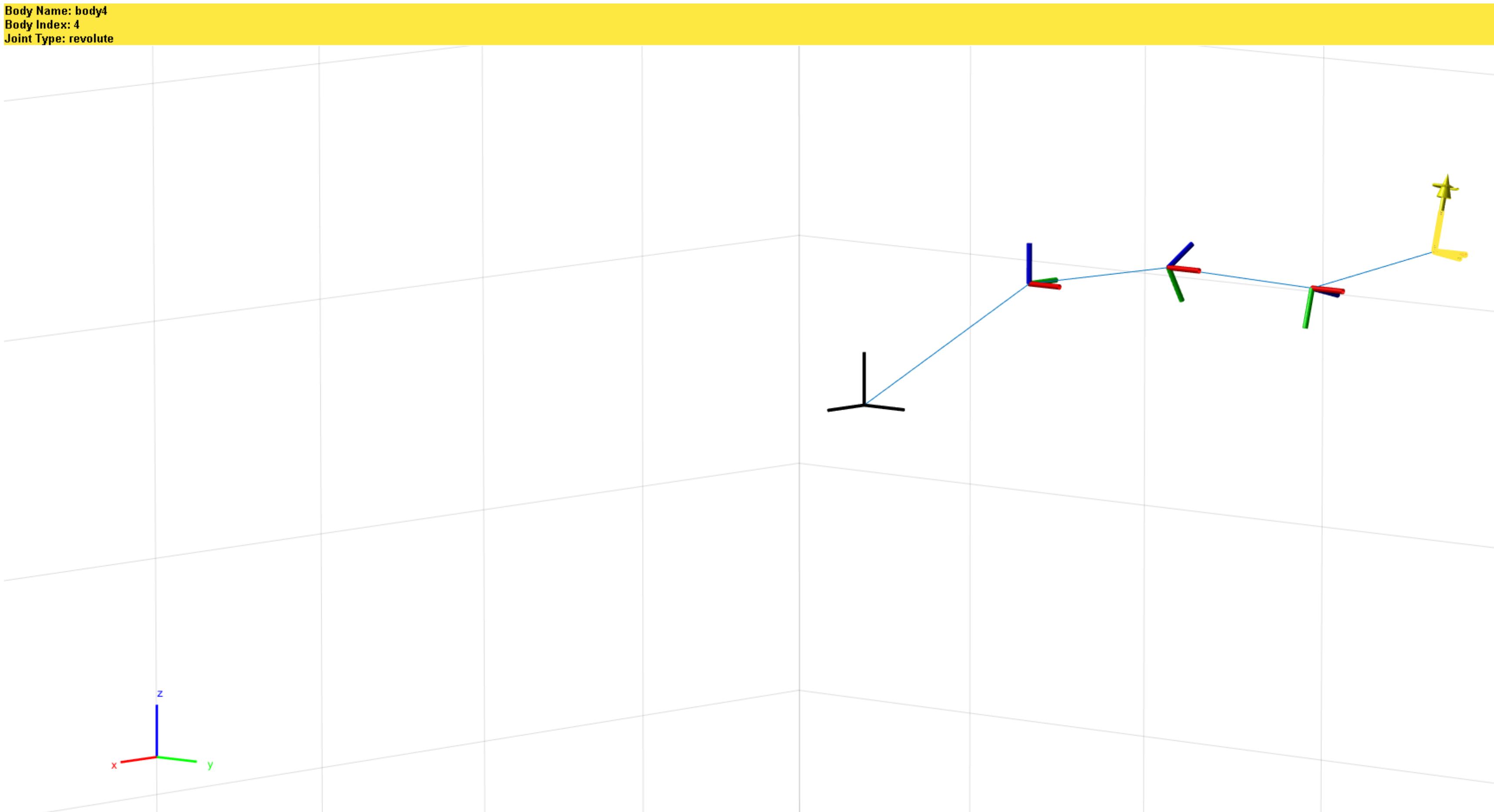
- q is the column matrix representing the joint velocities. The size of this matrix is nx1. 'n' is the number of joints of the robot.
- x is the column matrix related to the end-effector velocities. The size of this matrix is mx1. 'm' is 3 for a planar robot and 6 for a spatial robot.
- J is the Jacobian matrix representing a function of the current pose. The size of Jacobian matrix is mxn.

The expanded matrix form for a spatial robot, it shown below:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix}_{6\times1} = \begin{bmatrix} J_{11} & J_{12} & . & . & . & J_{1n} \\ J_{21} & J_{22} & . & . & . & J_{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ . & \cdots & . & . & . & . \\ J_{61} & \cdots & . & . & . & J_{6n} \end{bmatrix}_{6\times n} * \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ . \\ \vdots \\ \dot{q}_n \end{bmatrix}_{n\times1} \quad \text{-----------} (*)$$

Simulate the inverse kinematics results for the given end-point motion


Body Name: body4
Body Index: 4
Joint Type: revolute
Z
X
x
y
z


Body Name: body4
Body Index: 4
Joint Type: revolute
z
x
y

# Inverse Kinematics Calculations

Joint angle 1:90
Joint angle 2:45
Joint angle 3:60
Joint angle 4:45
Joint angle 5:-90

```
    1.5708

    0.7854

    1.0472

    2.3562

   -1.5708
```

```
[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]

Input x location:10
Input y location:8
Input z location:7

q1(in degrees)=38.6598

q2(in degrees)=-21.5679

q3(in degrees)=146.4206

## MATLAB Code

```
robot = robotics.RigidBodyTree('DataFormat', 'column'); % robot configuration as column vector
% add first body (fixed, with an offset from ICS)
body1 = robotics.RigidBody('body1');
body1.Joint = robotics.Joint('joint1', 'revolute');
T = trvec2tform([-0.5, 0 0.2])*eul2tform([pi/2 0 0]);
body1.Joint.setFixedTransform(T);
robot.addBody(body1, robot.BaseName);
% add second body (revolute joint)
body2 = robotics.RigidBody('body2');
body2.Joint = robotics.Joint('joint2', 'revolute');
body2.Joint.setFixedTransform([0.3 -pi/3 0.1 0], 'mdh');
robot.addBody(body2, 'body1');
% add third body (revolute joint)
body3 = robotics.RigidBody('body3');
body3.Joint = robotics.Joint('joint3', 'revolute');
body3.Joint.setFixedTransform([0.3 -pi/4 0.1 0], 'mdh');
robot.addBody(body3, 'body2');
% add fourth body (revolute joint)
body4 = robotics.RigidBody('body4');
body4.Joint = robotics.Joint('joint4', 'revolute');
body4.Joint.setFixedTransform([0.3 pi/2 0.1 0], 'mdh');
robot.addBody(body4, 'body3');


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)]);
```

## Jacobian matrix for Dexter robot 5 DOF

```
% Calculations for Dexter robot 5 DOF
A0 = [1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1];
syms s1 s2 c1 c2
A1 = [c1 0 -s1 0;s1 0 c1 0;0 -1 0 0;0 0 0 1];
syms d2
A2 = [c2 0 s2 0;s2 0 -c2 0;0 1 0 d2;0 0 0 1];
syms d3
A3 = [1 0 0 0;0 1 0 0;0 0 1 d3;0 0 0 1];
syms c4 s4
A4 = [c4 0 -s4 0;s4 0 c4 0;0 -1 0 0;0 0 0 1];
syms c5 s5
A5 = [c5 0 s5 0;s5 0 -c5 0;0 -1 0 0;0 0 0 1];

% z_i occurs in the third column of the rotation matrix component of T_0^i
 z0 = [0 0 1];
```

```
T01 = A1;
z1 = A1(1:3,3);
T02 = A1*A2;
z2 = T02(1:3,3);
T03 = A1*A2*A3;
z3 = T03(1:3,3);
T04 = A1*A2*A3*A4;
z4 = T04(1:3,3);
T05 = A1*A2*A3*A4*A5;
z5 = T05(1:3,3);

% the origins of frames 1 through 6
o0 = T01(1:3,4);
disp(o0)
o1 = T01(1:3,4);
disp(o1)
o2 = T02(1:3,4);
disp(o2)
o3 = T03(1:3,4);
disp(o3)
o4 = T04(1:3,4);
disp(o4)
o5 = T05(1:3,4);
disp(o5)



% Jacobian matrices

J1 = [cross(z0,(o5-o0));z0];
disp(J1)
J2 = [cross(z1,(o5-o1));z1];
disp(J2)
J3 = [cross(z2,(o5-o2));z2];
disp(J3)
J4 = [cross(z3,(o5-o3));z3];
disp(J4)
J5 = [cross(z4,(o5-o4));z4];
disp(J5)
```

**# Jacobian matrix is**

```
J = [J1 J2 J3 J4 J5];
```

Singular configuration is associated with Jacobian matrix determinant becoming zero and therefore there is no inverse exits meaning that the robot has lost a DOF. This can be observed during the Jacobian inversion computation, when it produces lager joint velocities or delta q which is not acceptable. This is not the case for this problem and no singularities can be identified.