Project 1:

Joint Variables:
• Joint Position A = [-106.12, -162.47, 135.49, 205.71, 15.19, 0.01]
• Joint Position B = [-45.48, -183.35,120.62, 238.42, -69.27, 0.01]

Part A:

Do you encounter any problems? Do you have any initial hypotheses why some motions are
possible, and some are not? Write about what could have impacted what you observed.

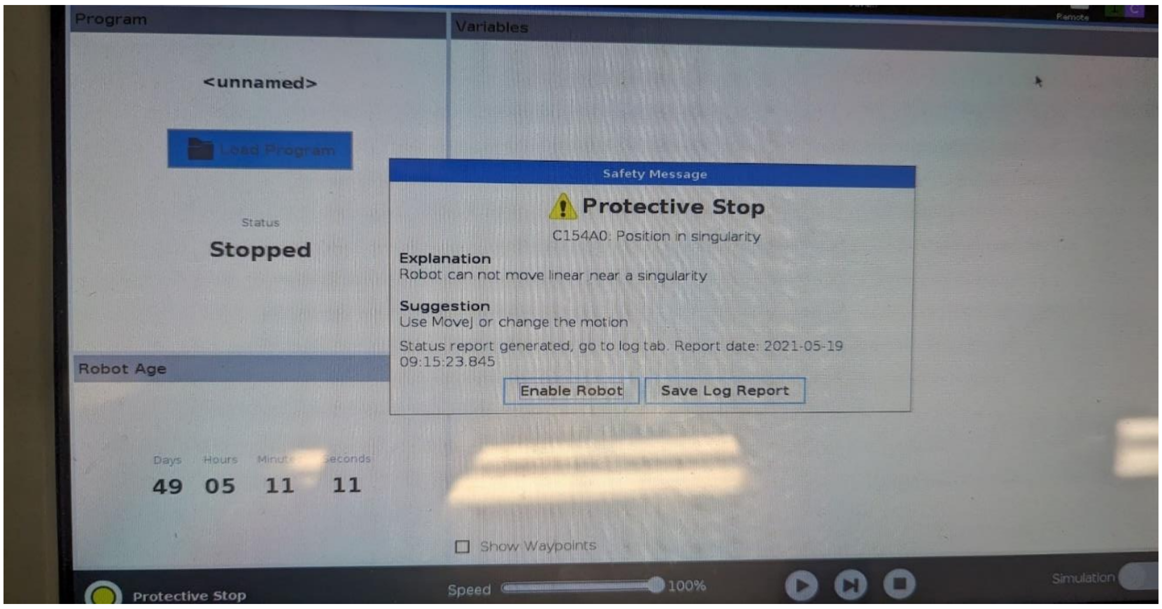
1. **Singularity**: Singularities are configurations of the robot where the Jacobian loses rank and the robot loses one or more degrees of freedom. If jointPosA or jointPosB are singular configurations, the robot might not be able to reach them, or it might not be able to move smoothly between them.
2. **Joint limits**: Every robot has limits on the angles each joint can reach. If jointPosA or jointPosB are outside these limits, the robot won't be able to reach them.
3. **Obstacles**: If there are obstacles in the robot's workspace, it might not be able to move in a straight line between two points. This could be the case even if both points are within the robot's reach.
4. **Speed and acceleration limits**: Robots also have limits on the speed and acceleration of their joints. If the movement between two points requires exceeding these limits, the robot might not be able to perform the movement.

I encountered warning signs that say a singularity is about to occur due to the robots movement.

```
% % TCP Host and Port settings
% host = '127.0.0.1'; % THIS IP ADDRESS MUST BE USED FOR THE
VIRTUAL BOX VM
% host = '192.168.230.128'; % THIS IP ADDRESS MUST BE USED FOR THE
VMWARE
clear all;
host = '192.168.0.100'; % THIS IP ADDRESS MUST BE USED FOR THE
REAL ROBOT
port = 30003;
% Calling the constructor of rtde to setup tcp connection
rtde = rtde(host,port);

% Setting home
home = [-588.53, -133.30, 371.91, 2.2214, -2.2214, 0.00];

jointPosA = deg2rad([106.12, -162.47, 135.49, 205.71, 15.19,
0.01]);
jointPosB = deg2rad([45.48, -183.35,120.62, 238.42, -69.27,
0.01]);
```

```
disp("Part A: ")
rtde.movel(home);pause(4)
disp("Move second")
% rtde.movel(jointPosA, "joint");pause(4);
disp("Move third")
rtde.movel(jointPosB, "joint");pause(4);
rtde.drawPath(tcp_pose);

rtde.close;
```

Figure 1: Code for movement



Safety Message

Protective Stop

C154A0: Position in singularity

Explanation
Robot can not move linear near a singularity

Suggestion
Use MoveJ or change the motion

Status report generated, go to log tab. Report date: 2021-05-19 09:15:23.845

Figure 2: Error from movement

Part B:

Part 1:

Joint Position A = [-106.12, -162.47, 135.49, 205.71, 15.19, 0.01]

We are interested in the all the joints, which has a position in degrees. We convert these joints to radians to get:

- $\theta_1 = -106.12 * \frac{\pi}{180} = -1.853$
- $\theta_2 = -162.47 * \frac{\pi}{180} = -2.837$
- $\theta_3 = 135.49 * \frac{\pi}{180} = 2.3658$
- $\theta_4 = 205.71 * \frac{\pi}{180} = 3.5919$
- $\theta_5 = 15.19 * \frac{\pi}{180} = 0.2652$
- $\theta_6 = 0.01 * \frac{\pi}{180} = 0.000175$

The DH parameters for the joints are:

$\theta_1 = -1.853$	$\theta_2 = 2.837$	$\theta_3 = 2.3658$	$\theta_4 = 3.5919$	$\theta_5 = 0.2652$	$\theta_6 = 0.00017$
a_1 = 0	a_2 = -0.425	a_3 = -0.3922	a_4 = 0	a_5 = 0	a_6 = 0
$d_1 = 0.1625$	d_2 = 0	d_3 = 0	d_4 = 0.1333	d_5 = 0.0997	d_6 = 0.0996
$\alpha_1 = \frac{\pi}{2}$	α_2 = 0	α_3 = 0	$\alpha_4 = \frac{\pi}{2}$	$\alpha_5 = -\frac{\pi}{2}$	α_6 = 0

The general form of the transformation matrix using Denavit-Hartenberg parameters is:

$$(T_i)_{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i * \cos\alpha_i & \sin\theta_i * \sin\alpha_i & a_i * \cos\theta_i \\ \sin\theta_i & \cos\theta_i * \cos\alpha_i & -\cos\theta_i * \sin\alpha_i & a_i * \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Substituting the parameters for all the joints into this formula, we get:
After substitution and simplification of this, we get:

$$T_{0-1} = \begin{bmatrix} -0.2777 & 0 & -0.9607 & 0 \\ -0.9607 & 0 & 0.2777 & 0 \\ 0 & 1 & 0 & 0.1625 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{1-2} = \begin{bmatrix} -0.9536 & 0.3012 & 0 & 0.4053 \\ -0.3012 & -0.9536 & 0 & 0.1280 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{2-3} = \begin{bmatrix} -0.7131 & -0.7010 & 0 & 0.2797 \\ 0.7010 & -0.7131 & 0 & -0.2749 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{3-4} = \begin{bmatrix} -0.9010 & 0 & -0.4338 & 0 \\ -0.4338 & 0 & 0.9010 & 0 \\ 0 & 1 & 0 & 0.1333 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{4-5} = \begin{bmatrix} 0.9651 & 0 & -0.2620 & 0 \\ 0.2620 & 0 & 0.9651 & 0 \\ 0 & -1 & 0 & 0.0997 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{5-6} = \begin{bmatrix} 1 & -0.0002 & 0 & 0 \\ 0.0002 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.0996 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

T_{Total} represents the transformation from the base frame to the end effector. The position of the end effector in the base frame can be found in the top-right column of this matrix. The orientation of the end effector can be found in the top-left 3x3 submatrix.

$$T_{0-6} = \begin{bmatrix} 0.0162 & 0.0062 & -0.9999 & -0.2439 \\ 0.9996 & 0.0211 & 0.0163 & -0.0170 \\ 0.0212 & -0.9998 & -0.0058 & 0.05675 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

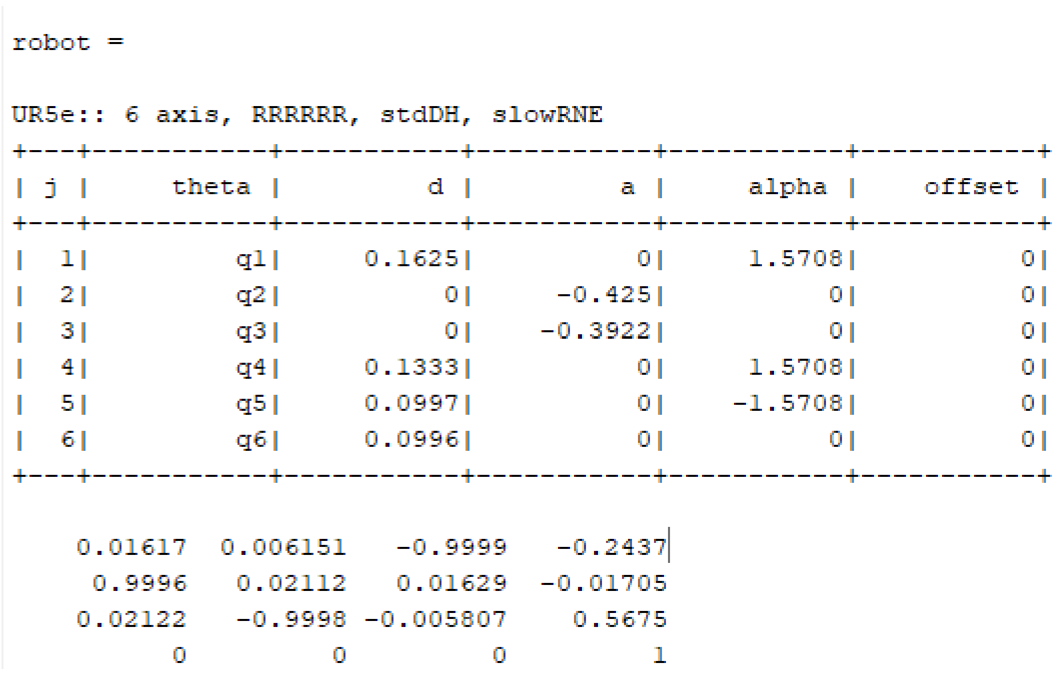
The position of the end effector in the base frame can be found in the top-right column of the transformation matrix and the orientation of the end effector can be found in the top-left 3x3 submatrix of the transformation matrix.

$$Position = \begin{bmatrix} -0.2437 \\ -0.0170 \\ 0.5675 \end{bmatrix}$$

$$Orientation = \begin{bmatrix} 0.0162 & 0.0062 & -0.9999 \\ 0.9996 & 0.0211 & 0.0163 \\ 0.0212 & -0.9998 & -0.0058 \end{bmatrix}$$

```
%% Part 2 =============================================================
% Import the Robotics Toolbox
clear all
startup_rvc;

% Define the DH parameters
% [ theta    d    a    alpha]
L(1) = Link([ 0    0.1625  0   pi/2], 'standard');
L(2) = Link([ 0    0      -0.425  0], 'standard');
L(3) = Link([ 0    0      -0.3922 0], 'standard');
L(4) = Link([ 0    0.1333  0   pi/2], 'standard');
L(5) = Link([ 0    0.0997  0   -pi/2], 'standard');
L(6) = Link([ 0    0.0996  0   0], 'standard');

% Create the robot
robot = SerialLink(L, 'name', 'UR5e');

% Display the robot model
robot.display();
Robot.plot([theta])
```

Figure 3: Code for robot model



```
robot =

UR5e:: 6 axis, RRRRRR, stdDH, slowRNE
+---+-----------+-----------+-----------+-----------+-----------+
| j |     theta |         d |         a |     alpha |    offset |
+---+-----------+-----------+-----------+-----------+-----------+
|  1|         q1|     0.1625|          0|     1.5708|          0|
|  2|         q2|          0|     -0.425|          0|          0|
|  3|         q3|          0|    -0.3922|          0|          0|
|  4|         q4|     0.1333|          0|     1.5708|          0|
|  5|         q5|     0.0997|          0|    -1.5708|          0|
|  6|         q6|     0.0996|          0|          0|          0|
+---+-----------+-----------+-----------+-----------+-----------+

    0.01617  0.006151   -0.9999   -0.2437
     0.9996   0.02112   0.01629  -0.01705
    0.02122   -0.9998 -0.005807    0.5675
          0         0         0         1
```

Figure 4: FKINE results for position A

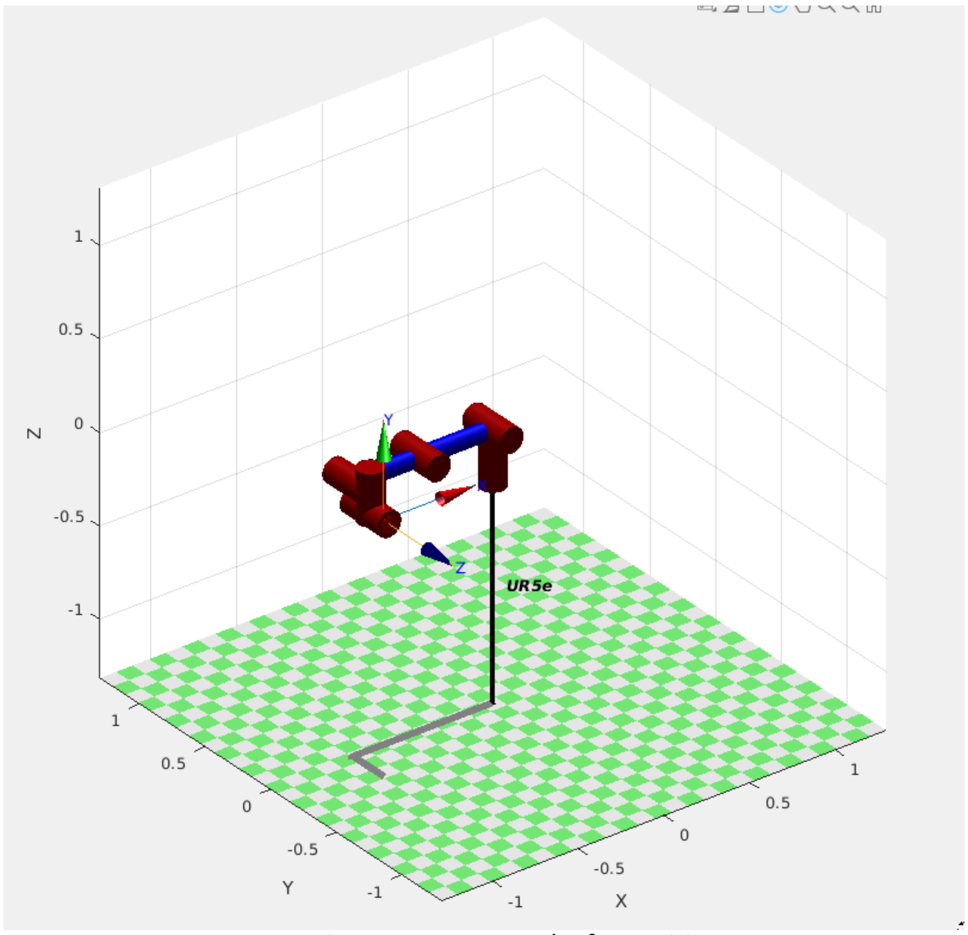
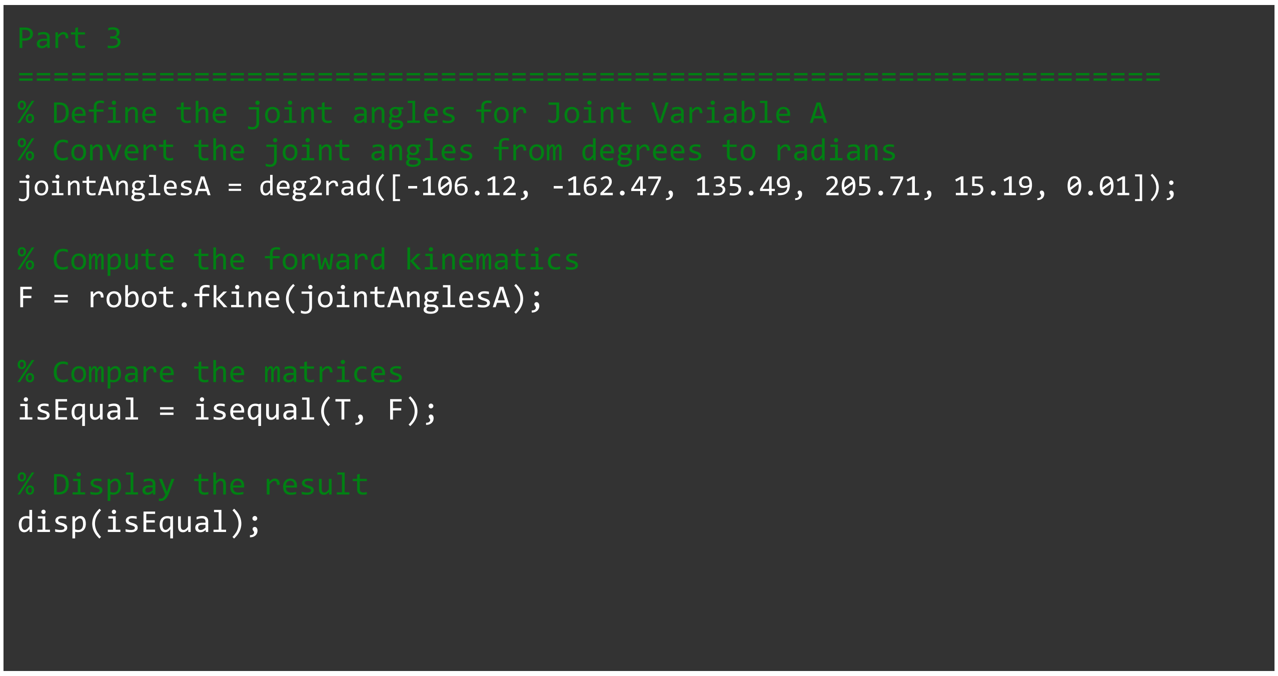
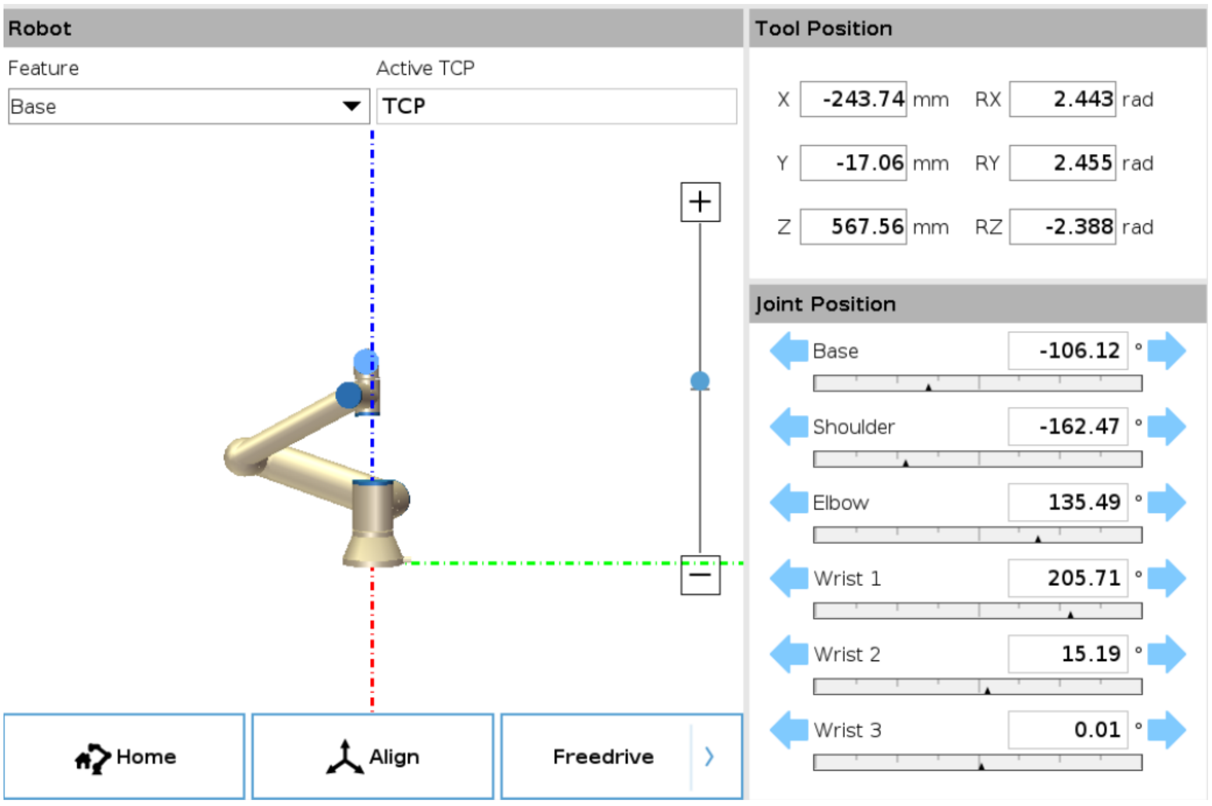


Figure 5: FKINE results for position A



```
Part 3
==================================================================
% Define the joint angles for Joint Variable A
% Convert the joint angles from degrees to radians
jointAnglesA = deg2rad([-106.12, -162.47, 135.49, 205.71, 15.19, 0.01]);

% Compute the forward kinematics
F = robot.fkine(jointAnglesA);

% Compare the matrices
isEqual = isequal(T, F);

% Display the result
disp(isEqual);
```

Figure 6: Mathematically checking movement of A



Feature: Base
Active TCP: TCP

Tool Position

X	-243.74 mm	RX	2.443 rad
Y	-17.06 mm	RY	2.455 rad
Z	567.56 mm	RZ	-2.388 rad

Joint Position

Base	-106.12 °
Shoulder	-162.47 °
Elbow	135.49 °
Wrist 1	205.71 °
Wrist 2	15.19 °
Wrist 3	0.01 °

Figure 7: Position A movement

Part C:

```
%% Fixed Code with a Via point.
clear all;

host = '127.0.0.1'; % THIS IP ADDRESS MUST BE USED FOR THE VIRTUAL BOX VM
% host = '192.168.230.128'; % THIS IP ADDRESS MUST BE USED FOR THE VMWARE
% host = '192.168.0.100'; % THIS IP ADDRESS MUST BE USED FOR THE REAL ROBOT
port = 30003;

% Calling the constructor of rtde to setup tcp connection
rtde = rtde(host,port);

% Setting home
home = [-588.53, -133.30, 371.91, 2.2214, -2.2214, 0.00];

jointPosA = deg2rad([106.12, -162.47, 135.49, 205.71, 15.19, 0.01]);
jointPosB = deg2rad([45.48, -183.35,120.62, 238.42, -69.27, 0.01]);

% Define via point as halfway between jointPosA and jointPosB
viaPoint = deg2rad([-85,-155,82,150,15,0.01]);
% Move to home position

[pose2,a2,b2,c2,d2] = rtde.movel(jointPosA, "joint");
[pose3,a3,b3,c3,d4] = rtde.movel(viaPoint, "joint");
[pose4,a4,b4,c4,d4] = rtde.movel(jointPosB, "joint");
[pose5,a5,b5,c5,d5] = rtde.movel(viaPoint, "joint");
[pose6,a6,b6,c6,d6] =  rtde.movel(jointPosA, "joint");
% pose7 = rtde.movej(home);

tcp_poses = [pose2;pose3;pose4;pose5;pose6];
rtde.drawPath(tcp_poses);
rtde.close;
```

Part D:

The root cause preventing the robot from completing a specified motion in this case is likely a singularity. A singularity is a configuration of the robot where the joints align in such a way that the robot loses one degree of freedom and cannot move in certain directions. This can happen, for example, when the robot's arm is fully extended or when two joint axes align.

In the context of robotic arms, singularities can occur in several situations:

- Wrist singularity: This occurs when the wrist center aligns with the desired end-effector position. In this case, the robot cannot maintain the end-effector orientation while moving the end-effector position.
- Shoulder singularity: This occurs when the shoulder joint is fully extended or fully retracted. In this case, the robot loses a degree of freedom in moving the end-effector position.

- Elbow singularity: This occurs when the elbow joint is fully extended or fully retracted. Similar to the shoulder singularity, the robot loses a degree of freedom in moving the end-effector position.

Mitigation methods for singularities include:

- Redesigning the task: If possible, the task can be redesigned so that the robot does not need to move into a singular configuration.
- Using redundancy: If the robot has more degrees of freedom than necessary for the task (i.e., it is redundant), these extra degrees of freedom can be used to avoid singularities.
- Using software solutions: Some robot control software can detect when the robot is approaching a singularity and automatically adjust the robot's motion to avoid it.
- Using via points: As you did in this task, you can define via points that the robot should move through to avoid singularities. The via points should be chosen so that they are not in singular configurations.

Remember that singularities are not inherently bad and can sometimes be used to the robot's advantage. However, they do need to be managed carefully to prevent unexpected robot behaviour.

<u>Part E:</u>

Obtain the DH Matrix:

Kinematics	Theta [rad]	A[m]	D[m]	Alpha [rad]
Joint 1	θ_1	L_1	d_1	0
Joint 2	θ_2	L_2	0	0
Joint 3	0	0	$-d_3 - d_4$	0

Calculate the Jacobian:

```
% Define the symbolic variables
clear all
syms theta1 theta2 d3 L1 L2 d1 d4 real

% DH parameters
theta = [theta1, theta2, 0]; % theta values in radians
a = [L1, L2, 0]; % a values in meters
d = [d1, 0, (d3-d4)]; % d values in meters
alpha = [0, 0, 0]; % alpha values in radians

% Initialize transformation matrix
T = eye(4);

for i = 1:3
    % Calculate the individual transformation matrix
    A = [cos(theta(i)), -sin(theta(i))*cos(alpha(i)), sin(theta(i))*sin(alpha(i)),
a(i)*cos(theta(i));
         sin(theta(i)), cos(theta(i))*cos(alpha(i)), -cos(theta(i))*sin(alpha(i)),
a(i)*sin(theta(i));
         0, sin(alpha(i)), cos(alpha(i)), d(i);
         0, 0, 0, 1];
    disp("A" + i)
    disp(A)
    % Multiply the overall transformation matrix by the individual transformation
matrix
    T = T * A;
end
disp("Transformation: ")
T = simplify(T);
disp(T)
% Assuming you have the transformation matrix T

% Extract the position vector from the transformation matrix
p = T(1:3, 4);


% Calculate the Jacobian matrix
J = simplify([diff(p, theta1), diff(p, theta2), diff(p, d3)])
```

Forward Kinematics between joints:

$$T_{0-1} = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & L_1\cos\theta_1 \\ \sin\theta_1 & \cos\theta_1 & 0 & L_1\sin\theta_1 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{1-2} = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & L_2\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & L_2\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{2-3} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{0-0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now we can obtain ${}^0T_2$ and ${}^0T_3$ through matrix multiplication.

$${}^0T_2 = {}^0T_1 \cdot {}^1T_2$$

$${}^0T_3 = {}^0T \cdot {}^1T_2 \cdot {}^2T_3$$

This gives us:

$${}^0T_2 = \begin{matrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & L_2\cos(\theta_1+\theta_2) + L_1\cos(\theta_1) \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & L_2\sin(\theta_1+\theta_2) + L_1\sin(\theta_1) \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$${}^0T_3 = \begin{matrix} \cos(\theta_1+\theta_2) & -\sin(\theta_1+\theta_2) & 0 & L_2\cos(\theta_1+\theta_2) + L_1\cos(\theta_1) \\ \sin(\theta_1+\theta_2) & \cos(\theta_1+\theta_2) & 0 & L_2\sin(\theta_1+\theta_2) + L_1\sin(\theta_1) \\ 0 & 0 & 1 & d_1 - d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$T = \begin{bmatrix} \cos\theta_1 + \theta_2 & -\sin\theta_1 + \theta_2 & 0 & L_2\cos(\theta_1+\theta_2) + L_1\cos(\theta_1) \\ \sin\theta_1 + \theta_2 & \cos(\theta_1+\theta_2) & 0 & L_2\sin(\theta_1+\theta_2) + L_2\sin(\theta_1) \\ 0 & 0 & 1 & d_3 + d_3 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From the abovce matrices, we can get the first 3 values of the 3rd column to form 0_{Z^i} and the values of the 4th column for 0_{O^i}.

For a revolute joint, the equation is:

$$J_i = \begin{matrix} {}^0Z_{(i-1)} \times ({}^0O_n - {}^0O_{(i-1)}) \\ {}^0Z_{(i-1)} \end{matrix}$$

And for a prismatic joint, the equation is:

$$J_i = \begin{matrix} {}^0Z_{(i-1)} \\ 0 \end{matrix}$$

$$Jacobian = \begin{bmatrix} -L_2\sin(\theta_1+\theta_2) + L_2\sin(\theta_1) & -L_2\sin\theta_1 + \theta_2 & 0 \\ \sin\theta_1 + \theta_2 & L_2\cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Determine singularities:

To find the singularities, I need to find the conditions under which the determinant of the Jacobian matrix becomes zero. When this is set to 0, it shows that the robot will have a singularity whenever θ2 is π or 0.

Inverse kinematics:

The inverse kinematic solution to this scara robot is:

$$x = \alpha_1 cos(\theta_1) + \alpha_2 cos(\theta_1 + \theta_2)$$

$$y = \alpha_1 sin(\theta_1) + \alpha_2 sin(\theta_1 + \theta_2)$$

$$z = d1 - d3 - d4$$