University of Groningen

Robotics for IEM

Assignment 3

Author:
Jonathan Chandra
(s5161533)

Lecturer:
Dr. Bahar Haghighat

October 26, 2023

Question 1

This question involves trajectory planning using MATLAB to move the end effector of a low-cost robotic manipulator AX-18A Smart Robotic Arm from a start position to its final position. In this solution, two (2) trajectory were planned, that is 1) Plan a trajectory to move the end effector from the starting pose to the final pose in the **joint space** and 2) Plan a trajectory for the end effector that is straight in the **Cartesian space**.

1.1 Trajectory planning in the joint space

A MATLAB function file called `traj_plan.m` is used to bring the end effector from its starting pose to its final pose. Given the starting pose and final pose with:

$$\begin{aligned} \text{pose\_start} &= \begin{bmatrix} 1 & 0 & 0 & -4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 54 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ \text{pose\_stop} &= \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 21 \\ 0 & 0 & -1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{1}$$

, a trajectory in joint space can be generated using `jtraj` function of Peter Corke's Robotic Toolbox. The function `jtraj` takes the input of initial and desired joint angle value with the number of steps between the intial and desired joint angle value. Given the pose_start and pose_stop in (1) with a Homogenous Transformation matrix form, the function `ikine` was used to generate the initial and desired joint angle.

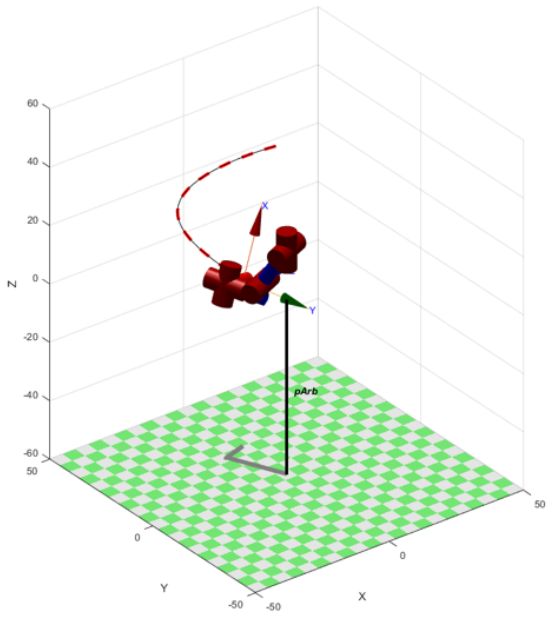

Figure 1: End effector trajectory (dashed line) using `jtraj` with the robot arm model

Figure 1 depicts the robot arm model with its end effector trajectory generated using `jtraj` function. The trajectory was generated using `fkine` function, since `jtraj` outputs the joint angle value from its pose_start to pose_stop configuration. Figure 2 in the next page shows the trajectories of the joint variables. It is to be noted that `jtraj` uses a quintic (5th order) polynomial with default zero boundary conditions for velocity and acceleration. This ensures a smooth and continuous change in the joint angles, creating a smooth motion for the robot.

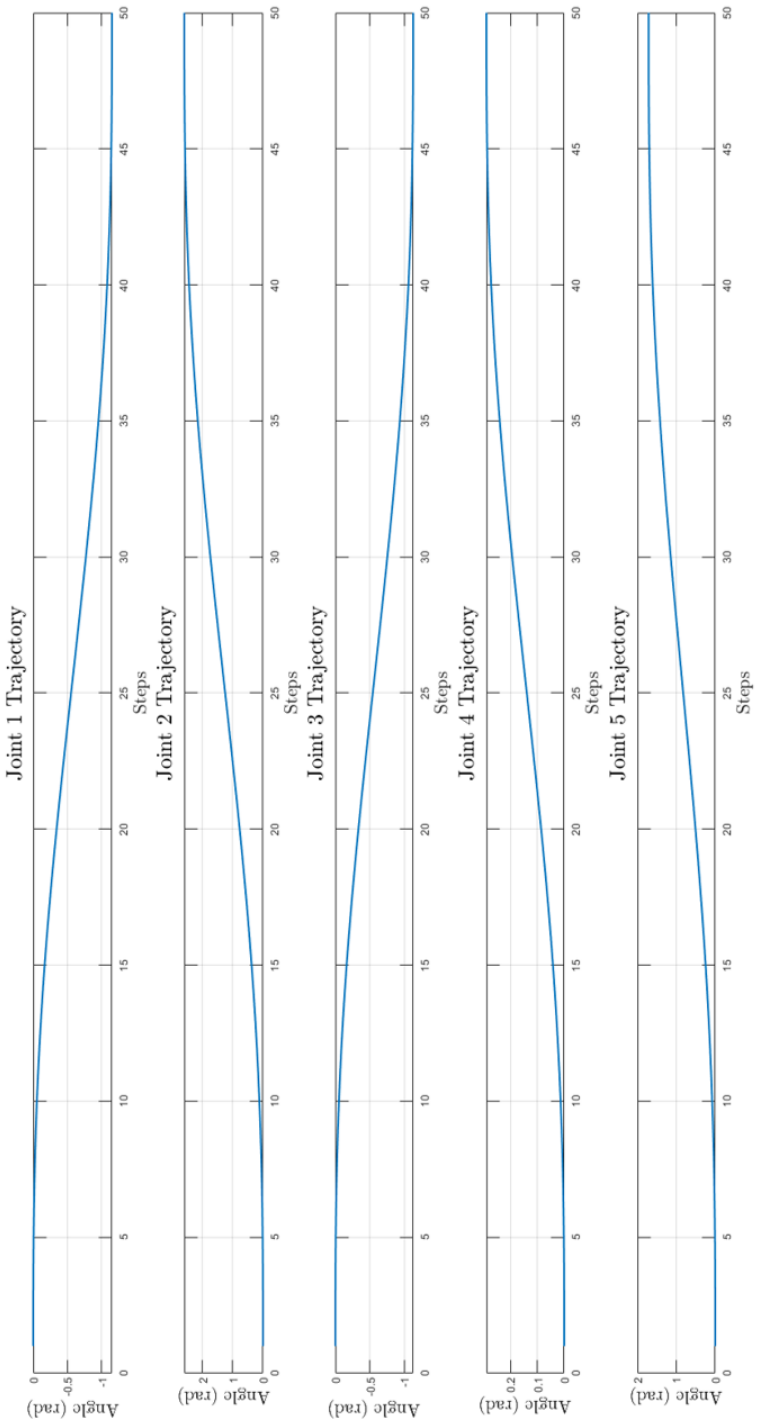

Figure 2: End effector joint trajectory

1.2 Straight trajectory planning in the Cartesian space

Recall the initial and desired pose from (1). Creating a straight trajectory between these two pose can be achieved using Cartesian space trajectory. The function `ctraj` was used to derive a straight trajectory between these two points, irrespective of the joint movements required to achieve that path [1]. The joint movements will be determined by the robot's inverse kinematics to ensure the end effector follows the straight-line path.

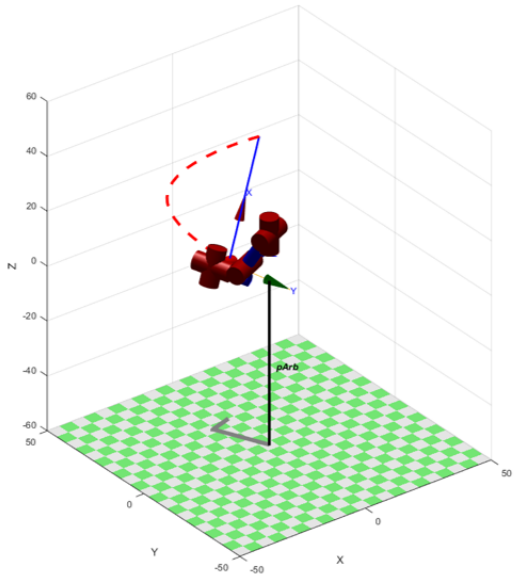

Figure 3: End effector trajectory (blue line) using `ctraj`

Using `ctraj` function was straight forward. It takes the initial and final pose homogeneous transformation matrices with the desired number of steps between those two pose. This function will output an array (n number of step) of homogeneous transformation matrices representing the interpolated poses between the initial and final poses.

Figure 3 depicts the robot arm model with its end effector trajectory that is straight in Cartesian space (blue line). Figure 4 shows the trajectories of the joint variables. Through this plot, it can be seen that while the trajectory generated is a straight line, the joint trajectory varies, irrespective of the joint movements required to achieve that path.

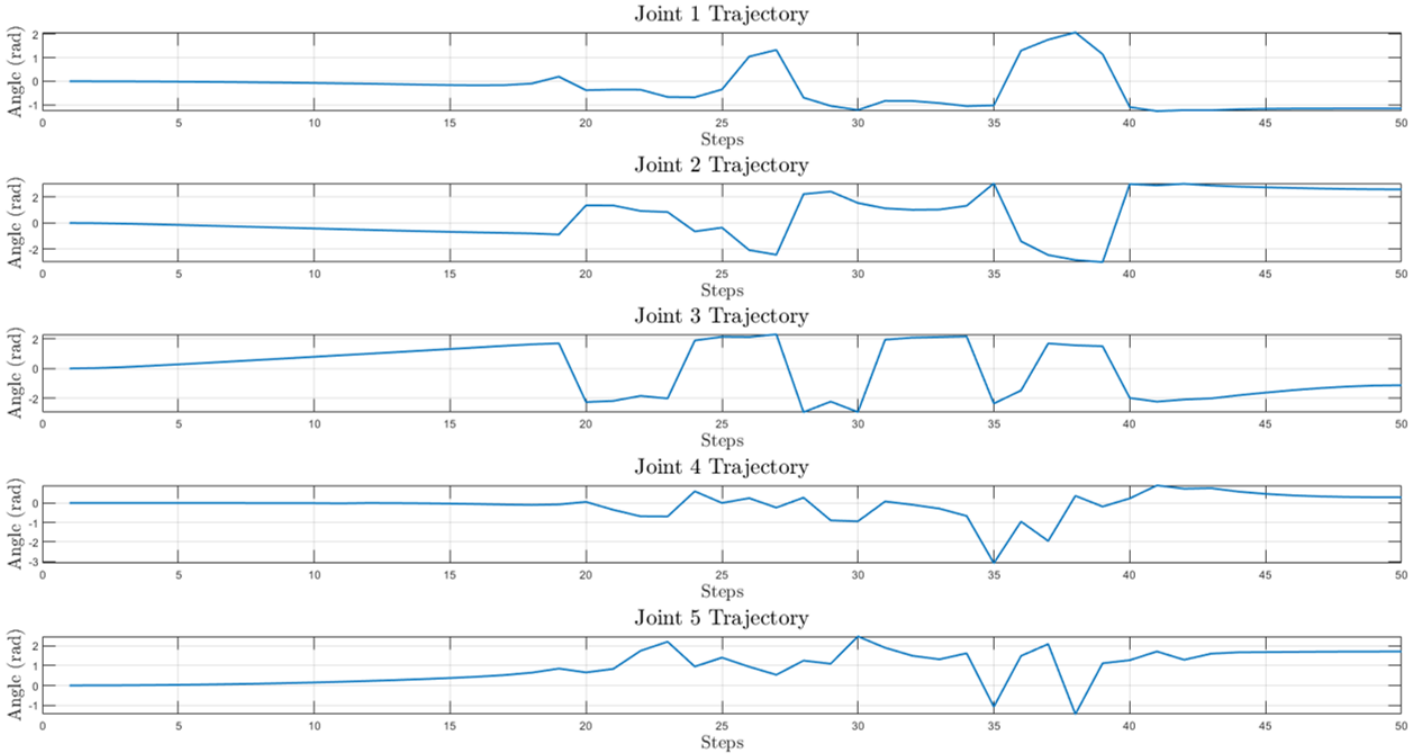

Figure 4: End effector joint trajectory

Question 2

This question involves developing a resolved-rate motion controller that leads to an approximation of a straight line motion between two points.

2.1 Joint coordinate computation

Resolved-rate motion control algorithm can be used to generate a straight line motion of an end effector (EF) manipulator [1]. The motion control scheme is implemented in discrete time form as:

$$\begin{aligned} \dot{q}\langle k\rangle &= J(q\langle k\rangle)^{-1}\nu \\ q\langle k+1\rangle &\leftarrow q\langle k\rangle + \delta_t \dot{q}\langle k\rangle \end{aligned} \tag{2}$$

where $J(q\langle k\rangle)^{-1}$ is a pseudo-inverse of the Jacobian $J(\langle q\rangle)$, δ_t is the sample interval. Now, assume a proportional gain $K > 0$ is introduced with the term $\delta_t \dot{q}\langle k\rangle$. Then (2) becomes

$$q\langle k+1\rangle \leftarrow q\langle k\rangle + K\delta_t \dot{q}\langle k\rangle \tag{3}$$

Introducing K will adjust the velocity (ν) based on how far the current state (in this case, EF position) is from the desired state. The aim of introducing such gain should improve the response of the system, possibly making it faster or more accurate based on a proper gain.

Computing $J(q\langle k\rangle)^{-1}$ is straight forward by using `pinv(J)` in MATLAB, while J can be computed directly using `jacob0(q)`. ν is computed by

$$\nu = \begin{bmatrix} \frac{p_{\text{end}} - p_{\text{current}}}{\delta_t} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

where p_{end} denotes the translation matrix inside the homogeneous transformation matrix of the desired pose that can be computed using `fkine`, while p_{current} denotes the translation matrix of current pose. ν is updated at each time step, from 1 to n steps. Three additional zero is present in (4) since `jacob0` computes all the Jacobian (linear and angular velocity). Angular velocity is ignored since we are only computing (4) which is a linear velocity, therefore three additional zero is placed there to have a compatible matrix size with the Jacobian Matrix.

The listing code to compute the joint coordinate is available in Listing B, Line 81-116 is a function to calculate the trajectory which outputs the

variable Qn_1, Qn_2, Qn_3 that is the joint coordinate, ct_1, ct_2, ct_3 is the cartesian trajectory. Listing B is made to compare three scenario of three different gain K values that will be explained in the next subsection.

2.2 Trajectory result

As explained in the previous subsection, the Cartesian trajectory points is saved in the variable ct_1, ct_2, ct_3.

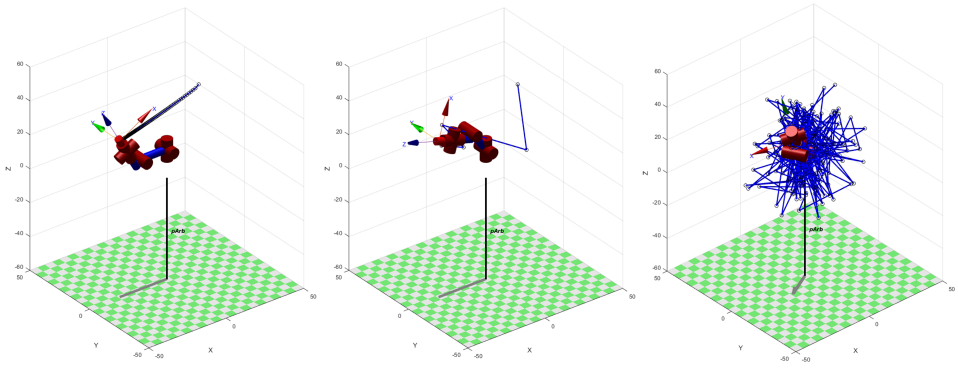

Figure 5: End effector trajectory (blue line) using different values of K (Left[K=0.03], Middle[K=1], Right[K=10])

Figure 5 shows the trajectory of the end effector with different value of K. Gain K was chosen based on a trial and error method. It can be seen on the left image of Figure 5 an approximate of a straight line trajectory was able to be generated using a relatively small value of K = 0.03. Moving to the remaining image of Figure 5 shows trajectories with a higher value of K that might cause the joints to overshoot when they are moving to the desired state.

It is confirmed in Figure 6 in the next page that using a relatively high gain K causes the joint states to overshoot when the resolved-rate motion controller is moving the end effector to its desired state. It is to be noted also that when running the m-file in Listing B, a noticeable time of 20 seconds pause happens after the the plot of the end effector ends.

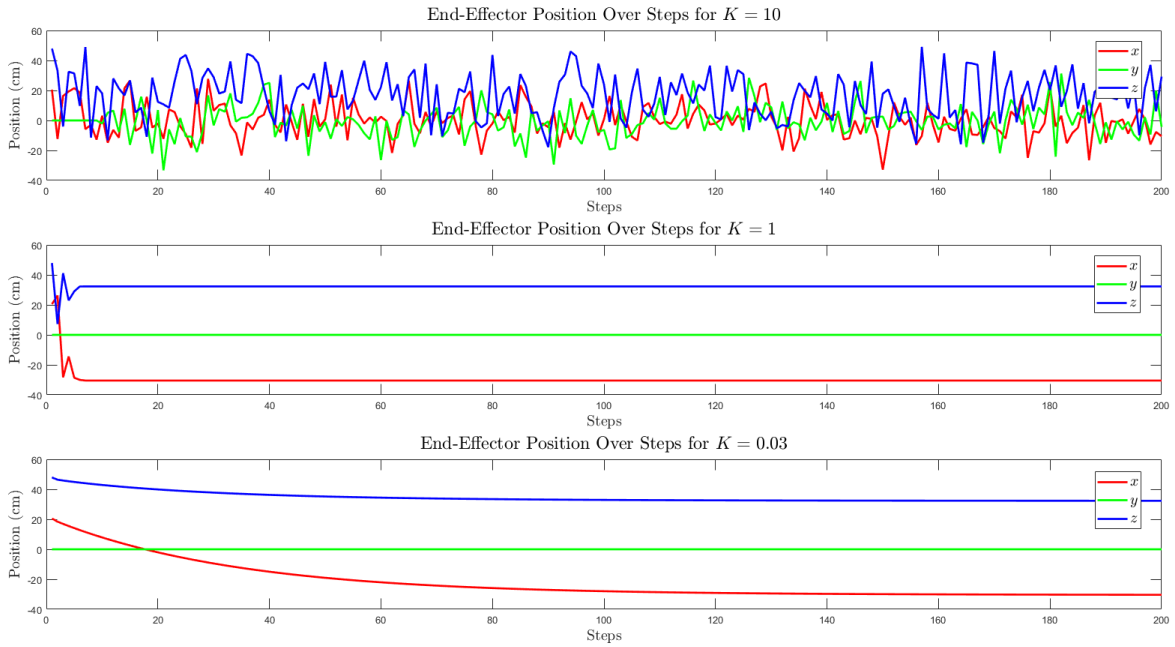

Figure 6: Position of the end-effector in x, y, z over the steps

References

[1] Corke, P. (2017). Robotics, Vision and Control: Fundamental Algorithms in MATLAB, 2nd Edition. Switzerland: Springer.

Appendix

A. Listing for `traj_plan.m`

```
clear all
close all
clc

L1 = 17; % note: all lengths are given in cm.
L2 = 17;
L3 = 7;
L4 = 4;
L5 = 4;
L6 = 9;

L(1) = Link('revolute', 'd',L1,'a',0,'alpha',0, '
   modified','offset', pi);
L(2) = Link('revolute', 'd',0,'a',0,'alpha',-pi/2, '
   modified','offset', -pi/2);
L(3) = Link('revolute', 'd',0,'a',L2,'alpha',0, '
   modified','offset', pi/2);
L(4) = Link('revolute', 'd',L3+L5,'a',L4,'alpha',pi
   /2, 'modified','offset', pi/2);
L(5) = Link('revolute', 'd',0,'a',0,'alpha',pi/2, '
   modified','offset', 0);

pArb = SerialLink(L, 'name', 'pArb')
pArb.plotopt={'workspace', [-50 50 -50 50 -60 60]};
pArb.tool = [0 -1 0 0; 0 0 1 L6; -1 0 0 0; 0 0 0 1];

pose_start = [1, 0, 0, -4;
              0, 1, 0, 0;
              0, 0, 1, 54;     % This corresponds to
                  pArb.fkine([0 0 0 0 0])
              0, 0, 0, 1];
```

```
pose_stop =  [-1, 0, 0, 0;
               0, 1, 0, 21;
               0, 0, -1, -3;
               0, 0, 0, 1];   % This corresponds to
                  pArb.fkine([pi/2 -pi/2 -pi/2 0 0])


%% Your answer
%%%%%% Write your code below this line. DO NOT
   change other parts. Some hints are given, but you
    are allowed to change them %%%%%%
%%%% This is for the first sub-question

% Plan trajectories in the joint space using
   pose_start and pose_stop
q_start = pArb.ikine(pose_start, [0 0 0 0 0], 'mask'
   , [1 1 1 0 0 0]);
q_stop = pArb.ikine(pose_stop, [0 0 0 0 0], 'mask',
   [1 1 1 0 0 0]);
num_steps = 50;
[q_traj, qd_traj, qdd_traj] = jtraj(q_start, q_stop,
    num_steps); % p114

num_points = size(q_traj, 1);
cartesian_traj = zeros(num_points, 3);

for k = 1:num_points
    T = pArb.fkine(q_traj(k, :));
    cartesian_traj(k, :) = T.t(1:3);
end

% Plot the robot arm trajectories in 3D
figure(1);
plot3(cartesian_traj(:, 1), cartesian_traj(:, 2),
   cartesian_traj(:, 3), 'r--', 'LineWidth', 3);
hold on;

% Plot the robot arm movement using the robotics
   toolbox
pArb.plot(q_traj, 'trail', {'k-', 'LineWidth', 0.1})
```

```
   ; % trail -> visualize movement

% Plot the trajectories of the joint variables
figure(2);
num_joints = size(q_traj, 2);
for i = 1:num_joints
    subplot(num_joints, 1, i);
    plot(q_traj(:, i), 'LineWidth', 1.5);
    hold on;
    title(['Joint ', num2str(i), ' Trajectory'], '
       Interpreter', 'latex', 'FontSize', 18);
    xlabel('Steps', 'Interpreter', 'latex', '
       FontSize', 15);
    ylabel('Angle (rad)', 'Interpreter', 'latex', '
       FontSize', 15);
    grid on;
end

%%%%% Write your code above this line. DO NOT change
    other parts. %%%%%%


%%%%%% Write your code below this line. DO NOT
   change other parts. %%%%%%
%%%% This is for the second sub-question

% Plan trajectories in the Cartesian space using
   pose_start and pose_stop
T = ctraj(pose_start, pose_stop, num_steps);
q_traj = zeros(num_steps, pArb.n);

% IK to find joint angles for each Cartesian pose
for i = 1:num_steps
    q_traj(i, :) = pArb.ikine(T(:,:,i), 'mask', [1 1
        1 0 0 0]);
end
X = squeeze(T(1,4,:));
Y = squeeze(T(2,4,:));
Z = squeeze(T(3,4,:));
```

```
% Plot the robot arm trajectories in 3D;
figure(1);
plot3(X, Y, Z, 'b', 'LineWidth', 2);
hold on;

% Plot the robot arm movement using the robotics
   toolbox
pArb.plot(q_traj);

% Plot the trajectories of the joint variables
num_joints = size(q_traj, 2);
figure(3);
for i = 1:num_joints
    subplot(num_joints, 1, i);
    plot(q_traj(:, i), 'LineWidth', 1.5);
    title(['Joint ', num2str(i), ' Trajectory'], '
       Interpreter', 'latex', 'FontSize', 18);
    xlabel('Steps', 'Interpreter', 'latex', '
       FontSize', 15);
    ylabel('Angle (rad)', 'Interpreter', 'latex', '
       FontSize', 15);
    grid on;
end


%%%%%% Write your code above this line. DO NOT
   change other parts. %%%%%%
```

0.1 B. Listing for resolved_rate.m

```
%% Create a robot arm
L1 = 17; % note: all lengths are given in cm.
L2 = 17;
L3 = 7;
L4 = 4;
L5 = 4;
L6 = 9;

L(1) = Link([ 0, L1, 0, 0, 0, 0], 'modified');
L(2) = Link([ 0, 0, 0, pi/2, 0, pi/2], 'modified');
L(3) = Link([ 0, 0, L2, 0, 0, pi/2], 'modified');
L(4) = Link([ 0, L3+L5, L4, pi/2, 0,pi/2], 'modified
   ');
L(5) = Link([ 0, 0, 0, pi/2, 0, 0], 'modified');

pArb = SerialLink(L, 'name', 'pArb');
pArb.plotopt={'workspace', [-50 50 -50 50 -60 60]};
pArb.tool = [0 -1 0 0; 0 0 1 9; -1 0 0 0; 0 0 0 1];

%% define q0 and pose_stop
q0 = [0 -pi/7 -pi/7 0 0];
pose_stop = pArb.fkine([0 pi/2 -pi/3 pi/6 0]);

%% Your answer
%%%%%% Write your code below this line. DO NOT
   change other parts. %%%%%%
% Instruction
% The motion will take n steps with the step size dt
   . At each time step, from 1 to n, compute the
% joint coordinates that the robot must reach using
   the resolved rate motion control algorithm.
   Notice
% that v has to change at each time instant such
   that it helps the robot move towards the goal
   point

dt = 0.05;                          % time step
n = 10/dt;                          % total steps
```

```
Qn = zeros(5,n);                % a matrix to store
   all the generated joint angles
q_null = q0';                   % Initial Joint
   angle var
pend = pose_stop.t;             % the goal point

% Qn_1, Qn_2, Qn_3 is the joint coordinate.
% ct_1, ct_2, ct_3 is the cartesian trajectory.
[ct_1, Qn_1] = calculate_trajectory(10, Qn, q_null,
   pend, n, dt, pArb, 1); % For K=10, Figure 1
[ct_2, Qn_2] = calculate_trajectory(1, Qn, q_null,
   pend, n, dt, pArb, 2); % For K=1, Figure 2
[ct_3, Qn_3] = calculate_trajectory(0.03, Qn, q_null
   , pend, n, dt, pArb, 3); % For K=0.03, Figure 3

figure(4);
subplot(3,1,1);
plot(ct_1(:,1), 'r-', 'LineWidth', 2); hold on;
plot(ct_1(:,2), 'g-', 'LineWidth', 2); hold on;
plot(ct_1(:,3), 'b-', 'LineWidth', 2);
legend('$x$','$y$','$z$', 'Interpreter','latex', '
   FontSize', 15);
title('End-Effector Position Over Time for $K=10$',
   'Interpreter','latex', 'FontSize', 18);
xlabel('Steps', 'Interpreter','latex', 'FontSize',
   15);
ylabel('Position (cm)', 'Interpreter','latex', '
   FontSize', 15);

subplot(3,1,2);
plot(ct_2(:,1), 'r-', 'LineWidth', 2); hold on;
plot(ct_2(:,2), 'g-', 'LineWidth', 2); hold on;
plot(ct_2(:,3), 'b-', 'LineWidth', 2);
legend('$x$','$y$','$z$', 'Interpreter','latex', '
   FontSize', 15);
title('End-Effector Position Over Time for $K=1$', '
   Interpreter','latex', 'FontSize', 18);
xlabel('Steps', 'Interpreter','latex', 'FontSize',
   15);
ylabel('Position (cm)', 'Interpreter','latex', '
```

```
   FontSize', 15);

subplot(3,1,3);
plot(ct_3(:,1), 'r-', 'LineWidth', 2); hold on;
plot(ct_3(:,2), 'g-', 'LineWidth', 2); hold on;
plot(ct_3(:,3), 'b-', 'LineWidth', 2);
legend('$x$','$y$','$z$', 'Interpreter','latex', '
   FontSize', 15);
title('End-Effector Position Over Time for $K=0.03$'
   , 'Interpreter','latex', 'FontSize', 18);
xlabel('Steps', 'Interpreter','latex', 'FontSize',
   15);
ylabel('Position (cm)', 'Interpreter','latex', '
   FontSize', 15);



% This part is just to make sure the start and end
   position is the same as
% the desired start and end position
pose_start = pArb.fkine(q0);
desired_pos = [(pose_start.t)' pend']
pos_final_k1 = [ct_1(1,:) ct_1(end,:)]
pos_final_k2 = [ct_2(1,:) ct_2(end,:)]
pos_final_k3 = [ct_3(1,:) ct_3(end,:)]

%  trajectory calculation function
function [cartesian_traj,q_traj] =
   calculate_trajectory(K, Qn, q, pend, n, dt, pArb,
    fig_num)
    % Initial variables
    cartesian_traj = zeros(n, 3);
    Qn(:, 1) = q;

    % Trajectory
    for i = 2:n
        J = pArb.jacob0(q');
        pcurrent = pArb.fkine(q');
        pcurrent = pcurrent.t;
```

```
        v = [((pend - pcurrent) / dt);
             0;
             0;
             0];

        qdot = pinv(J) * v;
        q = q + K * dt * qdot;
        Qn(:, i) = q;
    end

    q_traj = Qn';
    for k = 1:n
        T = pArb.fkine(q_traj(k, :));
        cartesian_traj(k, :) = T.t(1:3);
    end

    % Plot the trajectory
    figure(fig_num);
    plot3(cartesian_traj(:, 1), cartesian_traj(:, 2)
       , cartesian_traj(:, 3), 'b-','LineWidth', 2);
    hold on

    % Plot the robot arm movement using the robotics
        toolbox
    pArb.plot(q_traj, 'trail', {'k--o', 'LineWidth',
        0.5});
    set(gcf,'HandleVisibility','off');
end

%%%%%% Write your code above this line. DO NOT
   change other parts. %%%%%%
```

C. Listing for `jacob.m`

```
function J0v = jacob(q)
%INPUT: joint position of the robot.
%           - q : 5x1 [5x1 rad]
%OUTPUT:    - J0v [3x5]: Jacobian containing the map
    between end effectors
%linear velocities and joint velocities
% For this question, DO NOT use the robotics toolbox
   , you should derive answers yourself!
q1 = q(1);
q2 = q(2);
q3 = q(3);
q4 = q(4);
q5 = q(5);
c1 = cos(q1);
c2 = cos(q2);
c3 = cos(q3);
c4 = cos(q4);
c5 = cos(q5);
s1 = sin(q1);
s2 = sin(q2);
s3 = sin(q3);
s4 = sin(q4);
s5 = sin(q5);
c23 = cos(q2+q3);
s23 = sin(q2+q3);
f11 = 9*c1*c4*s5 - 9*s1*c23*s4*s5 + 9*s1*s23*c5 + s1
   *(4*c23 + 11*s23 + 17*s2);
f12 = 9*s1*c4*s5 + 9*c1*c23*s4*s5 - 9*c1*s23*c5 - c1
   *(4*c23 + 11*s23 + 17*s2);
f13 = 0;
f21 = -9*c1*s23*s4*s5 - 9*c1*c23*c5 - c1*(-4*s23 +
   11*c23 +17*c2);
f22 = -9*s1*s23*s4*s5 - 9*s1*c23*c5 - s1*(-4*s23 +
   11*c23 +17*c2);
f23 = -9*c23*s4*s5 - 9*s23*c5 -17*s2 - 11*s23 - 4*
   c23;
f31 = -9*c1*s23*s4*s5 - 9*c1*c23*c5 - c1*(-4*s23 +
   11*c23 + 17*s2);
```

```
f32 = -9*s1*s23*s4*s5 - 9*s1*c23*c5 - s1*(-4*s23 +
   11*c23 + 17*s2);
f33 = -9*c23*s4*s5 - 9*s23*c5 - 11*s23 - 4*c23;
f41 = -9*s1*s4*s5 + 9*c1*c23*c4*s5;
f42 = 9*c1*s4*s5 + 9*s1*c23*c4*s5;
f43 = -9*s23*c4*s5;
f51 = 9*s1*c4*c5 + 9*c1*c23*s4*c5 + 9*c1*s23*s5;
f52 = -9*c1*c4*c5 + 9*s1*c23*s4*c5 + 9*s1*s23*s5;
f53 = -9*s23*s4*c5 - 9*c23*s5;

J0v = [f11 f21 f31 f41 f51;
    f12 f22 f32 f42 f52;
    f13 f23 f33 f43 f53];

end
```