ME5250 Robot Mechanics and Control

Project 2

Submitted by: Haard Shah

Introduction

For my project, I will be analysing the forward kinematics of a 6-DOF robot i.e. the Universal Robot's UR-5e. I will be using published article as a guide for the link lengths and Denavit-Hartenberg (DH) parameters. I will determine the analytical forward kinematics by assigning coordinate frames to each link, calculating the individual homogeneous transformation matrices (T), and multiplying them in the proper order to obtain the overall transformation matrix that relates the end-effector frame to the base frame. Next, I will specify a trajectory in task space, here I am using the Ellipse trajectory, by defining a series of end-effector poses. To solve the inverse kinematics and determine the joint angles for each pose, I will use the Newton Raphson Method. Finally, I will create a simple "stick figure" 3D plot animation in MATLAB, showing the robot following the path.

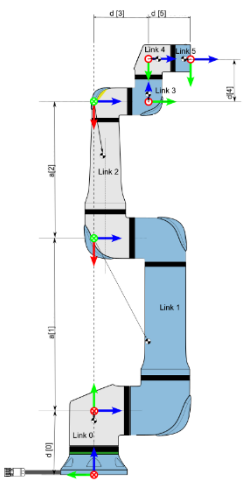


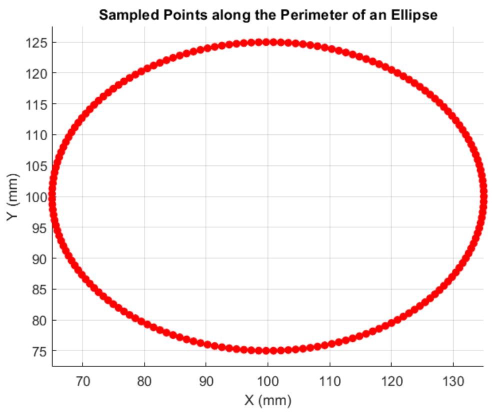
Fig. UR5e Robot with End-effector (Source: Universal Robots)

UR5e								
Kinematics	theta [rad]	a [m]	d [m]	alpha [rad]	**Dynamics**	Mass [kg]	Center of Mass [m]	Inertia Matrix
Joint 1	0	0	0.1625	π/2	Link 1	3.761	[0, -0.02561, 0.00193]	0
Joint 2	0	-0.425	0	0	Link 2	8.058	[0.2125, 0, 0.11336]	0
Joint 3	0	-0.3922	0	0	Link 3	2.846	[0.15, 0.0, 0.0265]	0
Joint 4	0	0	0.1333	π/2	Link 4	1.37	[0, -0.0018, 0.01634]	0
Joint 5	0	0	0.0997	-π/2	Link 5	1.3	[0, 0.0018,0.01634]	0
Joint 6	0	0	0.0996	0	Link 6	0.365	[0, 0, -0.001159]	[0, 0, 0, 0, 0, 0, 0, 0, 0.0002]

Fig. DH parameters of UR5e Robot (Source: UR5e)

To execute this project, we will follow the following steps systematically:

1) Formulate Forward Kinematics using provided D-H parameters
2) Select a trajectory and Sample Points (Our case: Ellipse)
3) Solve Inverse Kinematics for Sampled Points to get Joint angles
4) Calculate Joint Angles for Visualization
5) Visualize of the Robot following the Trajectory



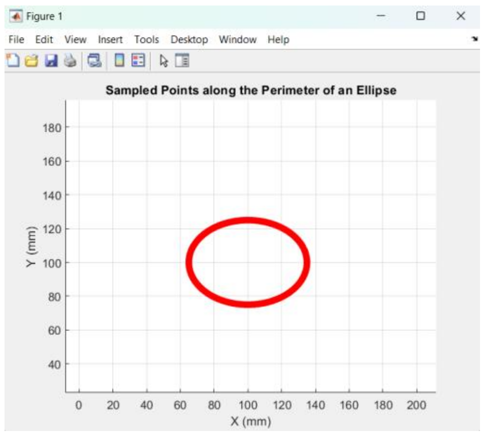
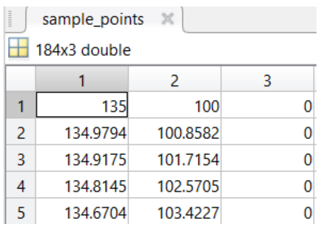


Figure 1
File Edit View Insert Tools Desktop Window Help
Sampled Points along the Perimeter of an Ellipse
Y (mm)
X (mm)
180
160
140
120
100
80
60
40
0
20
40
60
80
100
120
140
160
180
200

Fig. Sampled data points from the ellipse trajectory on a plot.

There are total of 184 points in the Ellipse formed for the trajectory and each having their individual (x, y, z) co-ordinates.



sample_points

184x3 double

	1	2	3
1	135	100	0
2	134.9794	100.8582	0
3	134.9175	101.7154	0
4	134.8145	102.5705	0
5	134.6704	103.4227	0

Fig. MATLAB saved list of point about trajectory – sampled points.

Inverse Kinematics:

$$\begin{aligned} \text{posx} &= d_5\cos(\theta_1)\sin(\theta_1+\theta_2+\theta_3) + d_4\sin(\theta_1) \\ &- d_6\cos(\theta_1)\cos(\theta_2+\theta_3+\theta_4) + a_2\cos(\theta_1)\cos(\theta_2) \\ &+ d_6\cos(\theta_5)\sin(\theta_1) + a_3\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) \\ &+ a_3\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) \end{aligned}$$

$$\begin{aligned} posy &= d_5\sin(\theta_1)\sin(\theta_2+\theta_3+\theta_4) - d_4\cos(\theta_1) \\ &- d_6\cos(\theta_1)\cos(\theta_5) + a_2\cos(\theta_2)\sin(\theta_1) \\ &a_3\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) - a_3\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) \end{aligned}$$

$$\begin{aligned} posz &= d_1 - d_6\sin(\theta_2+\theta_3+\theta_4)\sin(\theta_5) \\ &- a_3\sin(\theta_2+\theta_3) + a_2\sin(\theta_2) \\ &a_2\sin(\theta_2) - d_5\cos(\theta_2+\theta_3+\theta_4) \end{aligned}$$

- Taking partial derivatives of the position x to obtain components of the co-ordinate Jacobian Matrix

$$\frac{\partial p_x}{\partial \theta_1} = -d_5 \sin(\theta_1)\sin(\theta_2+\theta_3+\theta_4) + d_4\cos(\theta_1) + d_6\sin(\theta_1)\cos(\theta_2+\theta_3+\theta_4) - a_2\sin(\theta_1)\cos(\theta_2) + d_6\cos(\theta_5)\cos(\theta_1) - a_3\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) + a_3\sin(\theta_1)\sin(\theta_2)\sin(\theta_3)$$

$$\frac{\partial p_x}{\partial \theta_2} = d_5\cos(\theta_1)\cos(\theta_2+\theta_3+\theta_4) + d_6\cos(\theta_1)\sin(\theta_2+\theta_3+\theta_4) - a_2\sin(\theta_2)\cos(\theta_1) - a_3\cos(\theta_1)\sin(\theta_2)\cos(\theta_3) - a_3\cos(\theta_1)\cos(\theta_2)\sin(\theta_3)$$

$$\frac{\partial p_x}{\partial \theta_3} = d_5\cos(\theta_1)\cos(\theta_2+\theta_3+\theta_4) + d_6\cos(\theta_1)\sin(\theta_2+\theta_3+\theta_4) - a_3\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) - a_3\sin(\theta_2)\cos(\theta_1)\cos(\theta_3)$$

$$\frac{\partial p_x}{\partial \theta_4} = -d_5\cos(\theta_1)\cos(\theta_2+\theta_3+\theta_4) + d_6\cos(\theta_1)\cos(\theta_2+\theta_3+\theta_4)$$

$$\frac{\partial p_x}{\partial \theta_5} = -d_6\sin(\theta_1)\sin(\theta_5)$$

$$\frac{\partial p_x}{\partial \theta_6} = 0$$

- Taking partial derivatives of the position y to obtain components of the co-ordinate Jacobian Matrix

$$\frac{\partial p_y}{\partial \theta_1} = d_5\cos(\theta_1)\sin(\theta_2+\theta_3+\theta_4) + d_4\sin(\theta_1) - d_6\cos(\theta_1)\cos(\theta_2+\theta_3+\theta_4) - d_6\sin(\theta_1)\cos(\theta_5) + a_2\cos(\theta_2)\cos(\theta_1) - a_3\cos(\theta_1)\sin(\theta_2)\sin(\theta_3) + a_3\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)$$

$$\frac{\partial p_y}{\partial \theta_2} = d_5\sin(\theta_1)\cos(\theta_2+\theta_3+\theta_4) + d_6\sin(\theta_1)\sin(\theta_2+\theta_3+\theta_4) - a_2\sin(\theta_2)\sin(\theta_1) - a_3\sin(\theta_1)\sin(\theta_2)\cos(\theta_3) - a_3\sin(\theta_1)\cos(\theta_2)\sin(\theta_3)$$

$$\frac{\partial p_y}{\partial \theta_3} = d_5\sin(\theta_1)\cos(\theta_2+\theta_3+\theta_4) + d_6\sin(\theta_1)\sin(\theta_2+\theta_3+\theta_4) - a_3\cos(\theta_2)\sin(\theta_1)\sin(\theta_3) - a_3\sin(\theta_2)\sin(\theta_1)\cos(\theta_3)$$

$$\frac{\partial p_y}{\partial \theta_4} = d_5\sin(\theta_1)\cos(\theta_2+\theta_3+\theta_4) + d_6\sin(\theta_1)\sin(\theta_2+\theta_3+\theta_4)$$

$$\frac{\partial p_y}{\partial \theta_5} = d_6\cos(\theta_1)\sin(\theta_5)$$

$$\frac{\partial p_y}{\partial \theta_6} = 0$$

- Taking partial derivatives of the position z to obtain components of the co-ordinate Jacobian Matrix

$$\frac{\partial p_z}{\partial \theta_1} = 0$$

$$\frac{\partial p_z}{\partial \theta_2} = -d_6\sin(\theta_5)\cos(\theta_2+\theta_3+\theta_4) + a_3\cos(\theta_1+\theta_2) + a_2\cos(\theta_2) + d_5\sin(\theta_2+\theta_3+\theta_4)$$

$$\frac{\partial p_z}{\partial \theta_3} = -d_6\sin(\theta_5)\cos(\theta_2+\theta_3+\theta_4) + a_3\cos(\theta_2+\theta_3) + d_5\sin(\theta_2+\theta_3+\theta_4)$$

$$\frac{\partial p_z}{\partial \theta_4} = -d_6\sin(\theta_5)\cos(\theta_2+\theta_3+\theta_4) + d_5\sin(\theta_2+\theta_3+\theta_4)$$

$$\frac{\partial p_z}{\partial \theta_5} = -d_6\sin(\theta_2+\theta_3+\theta_4)\cos(\theta_5)$$

$$\frac{\partial p_z}{\partial \theta_6} = 0$$

Inconsistencies in the trajectory:

During this project, two persistent errors were identified. Firstly, there is inconsistency observed in the axis of the manipulator as it moves along the plane. This variability in axis orientation results in deviations from the intended path, affecting the accuracy of the manipulator's trajectory tracking. Secondly, anomalies have been noted in the initial guesses utilized for the Newton-Raphson computation. These initial guesses, intended to align with the desired ellipse trajectory, exhibited deviations that hindered the achievement of precise motion control.

- I have hereby attached a video submission which showcases how the UR5e Robot would track the Ellipse trajectory with its End-effector.

Link: https://drive.google.com/file/d/1RQrwv48EqXHK9hTAsnFB4-m1nwjCU6JM/view?usp=sharing

Please sign in through the Gmail - ID to access the video submission as Google Drive has restriction on sharing.

Acknowledgements:

I would like to acknowledge the following resources which have been extremely helpful to me throughout this project:

- ChatGPT v3.5
- Stack overflow
- Lynch & Park – Modern Robotics
- Paper citation: Villalobos, J.; Sanchez, I.Y.; Martell, F. Singularity Analysis and Complete Methods to Compute the Inverse Kinematics for a 6-DOF UR/TM-Type Robot. *Robotics* **2022**, *11*, 137. https://doi.org/10.3390/robotics11060137
- Universal Robots
- https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/
- https://youtu.be/rA9tm0gTln8?feature=shared

Appendix:

```
%Project2: ME5250
%Submitted by: Haard Shah
%Code1: Sampling points from the perimeter of the desired Ellipse trajectory

% Defining the Semi - Major & Minor Axis for the Ellipse (units: in millimeters (mm))
a = 35; % Semi-major axis
b = 25;  % Semi-minor axis

% Defining the interval (units: in mm)
interval = 1.2; % 1.2 mm interval

% Compute the number of points along the perimeter
theta = linspace(0, 2*pi, ceil(2*pi*a/interval));

% Create and Initialize an array to store the sampled points from trajectory
sample_points = [];

% Sample points along the perimeter of the ellipse
sample_points = [sample_points; a * cos(theta)', b * sin(theta)'];

% Shift the ellipse to desired position
center = [50, 50]; % Center of the ellipse
sample_points = sample_points + center;
sample_points = sample_points +50;

% Display the sampled points
scatter(sample_points(:,1), sample_points(:,2), 'filled', 'MarkerFaceColor', 'red');
axis equal;
xlabel('X (mm)');
ylabel('Y (mm)');
title('Sampled Points along the Perimeter of an Ellipse');
grid on;

% To consider Z-axiz is 0 so it is in a 2-D plane
sample_points(:,3) = 0;



%Code2: Calculating the Inverse Kinematics based on sampled trajectory

% guess_in: Initial guess for joint angles
%This is for the computation to begin as without this Newton Raphson cannot
%proceed without inital guess.
guess_in = [15 20 25 10 15 12]; % (units: in degrees)
guess_in = guess_in*0.0175; % (Conversion degrees to radians = pi/180 = 0.01745 ~ 0.0175)

syms a b c real;

%Desired Transformation Matrix
Desired_tr = [
    -0.5 -0.866 0 a;
    0.866 -0.5 0 b;
    0 0 1 c;
    0 0 0 1];

max_iterations = 300;
threshold = 1; %Upper limit of convergence

[nrows, ncols] = size(sample_points);
d_list_angles = zeros(nrows,6);
for i = 1:nrows
    row = sample_points(i,:);
    Desired_tr(1:3,4) = row;
    [out_joint, output_error]= newton_inverse_kinematics(Desired_tr, guess_in, max_iterations, threshold);
    out_joint = out_joint/0.0175;
    d_list_angles(i,:) = out_joint;
    out_joint = out_joint*0.01744;
    guess_in = out_joint;

end

function [jointangles, error] = newton_inverse_kinematics(Desired_tr, guess_in, max_iterations, threshold)

    joint_angles = guess_in;
    error = [inf inf inf];
    iteration = 0;

    while (abs(error(1)) > threshold || abs(error(2)) > threshold || abs(error(3)) > threshold) && iteration < max_iterations
        % Compute the forward kinematics for the current joint angles
        T_current = forward_kinematics(joint_angles);
        jointangles = double(joint_angles);

        % Compute the error between desired and current end-effector positions

        error = Desired_tr(1:3, 4) - T_current(1:3, 4);

        % Compute the Jacobian matrix
        J = compute_jacobian(joint_angles);

        % Compute the update for joint angles using Newton's method
        delta_theta = pinv(J) * (error);

        % Update joint angles
        joint_angles = joint_angles + delta_theta';
        iteration = iteration + 1;
    end
    disp('Loop is completed');
    if iteration >= max_iterations
        disp('Could not converge within the specified threshold.');
    end
end

function T = forward_kinematics(joint_angles)

   syms h1 h2 h3 h4 h5 h6 real;
   d1 = 162.5;
   a2 = 425;
   a3 = 392.2;
   d4 = 133.3;
   d5 = 99.7;
   d6 = 99.6;


   e11 = cos(h1)*cos(h2+h3+h4)*cos(h5)*cos(h6) + cos(h6)*sin(h1)*sin(h5) - cos(h1)*sin(h2+h3+h4)*sin(h6);
   e12 = -cos(h1)*cos(h2+h3+h4)*cos(h5)*cos(h6) - sin(h1)*sin(h5)*sin(h6) - cos(h1)*cos(h6)*sin(h2+h3+h4);
   e13 = -cos(h1)*cos(h2+h3+h4)*sin(h5) + cos(h5)*sin(h1);
   e21 = cos(h2+h3+h4)*cos(h5)*cos(h6)*sin(h1) - cos(h1)*cos(h6)*sin(h5) - sin(h1)*sin(h2+h3+h4)*sin(h6);
   e22 = -cos(h2+h3+h4) + cos(h1)*sin(h5)*sin(h6) - cos(h6)*sin(h1)*sin(h2+h3+h4);
   e23 = -cos(h2+h3+h4)*sin(h1)*sin(h5) - cos(h1)*cos(h5);
   e31 = cos(h5)*cos(h6)*sin(h2+h3+h4) + cos(h2+h3+h4)*sin(h6);
   e32 = -cos(h5)*sin(h2+h3+h4)*sin(h6) + cos(h2+h3+h4)*cos(h6);
   e33 = -sin(h2+h3+h4)*sin(h5);

   posx = d5*cos(h1)*sin(h2+h3+h4) + d4*sin(h1) - d6*cos(h1)*cos(h2+h3+h4) + a2*cos(h1)*cos(h2) + d6*cos(h5)*sin(h1) + a3*cos(h1)*cos(h2)*cos(h3) -
a3*cos(h1)*sin(h2)*sin(h3);
   posy = d5*sin(h1)*sin(h2+h3+h4) - d4*cos(h1) - d6*sin(h1)*cos(h2+h3+h4) - d6*cos(h1)*cos(h5) + a2*cos(h2)*sin(h1) + a3*cos(h2)*cos(h3)*sin(h1) -
a3*sin(h1)*sin(h2)*sin(h3);
   posz = d1 - d6*sin(h2+h3+h4)*sin(h5) + a3*sin(h2+h3) + a2*sin(h2) - d5*cos(h2+h3+h4);

   T = [
       e11 e12 e13 posx;
       e21 e22 e23 posy;
       e31 e32 e33 posz;
       0 0 0 1];

   h1_val = joint_angles(1);
   h2_val = joint_angles(2);
   h3_val = joint_angles(3);
   h4_val = joint_angles(4);
   h5_val = joint_angles(5);
   h6_val = joint_angles(6);

   T_sub = subs(T, h1, h1_val);
   T_sub = subs(T_sub, h2, h2_val);
   T_sub = subs(T_sub, h3, h3_val);
   T_sub = subs(T_sub, h4, h4_val);
   T_sub = subs(T_sub, h5, h5_val);
   T_sub = subs(T_sub, h6, h6_val);
   T_numeric = double(T_sub);
   T = T_numeric;

end

function J = compute_jacobian(joint_angles)

    syms h1 h2 h3 h4 h5 h6 real;
    d1 = 162.5;
    d4 = 133.3;
    d5 = 99.7;
    d6 = 99.6;
    a2 = 425;
    a3 = 392.2;

    dposx_1 = -d5*sin(h1)*sin(h2+h3+h4) + d4*cos(h1) + d6*sin(h1)*cos(h2+h3+h4) - a2*sin(h1)*cos(h2) + d6*cos(h5)*cos(h1) - a3*sin(h1)*cos(h2)*cos(h3) +
a3*sin(h1)*sin(h2)*sin(h3);
    dposy_1 = d5*cos(h1)*sin(h2+h3+h4) + d4*sin(h1) - d6*cos(h1)*cos(h2+h3+h4) + d6*sin(h1)*cos(h5) + a2*cos(h2)*cos(h1) + a3*cos(h2)*cos(h3)*cos(h1) -
a3*cos(h1)*sin(h2)*sin(h3);
    dposz_1 = 0;

    dposx_2 = d5*cos(h1)*cos(h2+h3+h4) + d6*cos(h1)*sin(h2+h3+h4) - a2*cos(h1)*sin(h2) - a3*cos(h1)*sin(h2)*cos(h3) - a3*cos(h1)*cos(h2)*sin(h3);
    dposy_2 = d5*sin(h1)*cos(h2+h3+h4) + d6*sin(h1)*sin(h2+h3+h4) - a2*sin(h2)*sin(h1) - a3*sin(h2)*cos(h3)*sin(h1) - a3*sin(h1)*cos(h2)*sin(h3);
    dposz_2 = -d6*cos(h2+h3+h4)*sin(h5) + a3*cos(h2+h3) + a2*cos(h2) + d5*sin(h2+h3+h4);

    dposx_3 = d5*cos(h1)*cos(h2+h3+h4) + d6*cos(h1)*sin(h2+h3+h4) - a3*cos(h1)*cos(h2)*sin(h3) - a3*cos(h1)*sin(h2)*cos(h3);
    dposy_3 = d5*sin(h1)*cos(h2+h3+h4) + d6*sin(h1)*sin(h2+h3+h4) - a3*cos(h2)*sin(h3)*sin(h1) - a3*sin(h1)*sin(h2)*cos(h3);
    dposz_3 = -d6*cos(h2+h3+h4)*sin(h5) + a3*cos(h2+h3) + d5*sin(h2+h3+h4);

    dposx_4 = d5*cos(h1)*cos(h2+h3+h4) + d6*cos(h1)*sin(h2+h3+h4);
    dposy_4 = d5*sin(h1)*cos(h2+h3+h4) + d6*sin(h1)*sin(h2+h3+h4);
    dposz_4 = -d6*cos(h2+h3+h4)*sin(h5) + d5*sin(h2+h3+h4);

    dposx_5 = -d6*sin(h5)*sin(h1);
    dposy_5 = d6*cos(h1)*sin(h5);
    dposz_5 = -d6*sin(h2+h3+h4)*cos(h5);

    dposx_6 = 0;
    dposy_6 = 0;
    dposz_6 = 0;

    J = [
        dposx_1 dposx_2 dposx_3 dposx_4 dposx_5 dposx_6;
        dposy_1 dposy_2 dposy_3 dposy_4 dposy_5 dposy_6;
        dposz_1 dposz_2 dposz_3 dposz_4 dposz_5 dposz_6];

    h1_val = joint_angles(1);
    h2_val = joint_angles(2);
    h3_val = joint_angles(3);
    h4_val = joint_angles(4);
    h5_val = joint_angles(5);
    h6_val = joint_angles(6);

    J_sub = subs(J, h1, h1_val);
    J_sub = subs(J_sub, h2, h2_val);
    J_sub = subs(J_sub, h3, h3_val);
    J_sub = subs(J_sub, h4, h4_val);
    J_sub = subs(J_sub, h5, h5_val);
    J_sub = subs(J_sub, h6, h6_val);
    J_numeric = double(J_sub);
    J = J_numeric;

end


%Code3: Visualization of Robot trajectory

joint_angles = d_list_angles * 0.01745; % Convert to radians

% Initialize the UR5 Denavit-Hartenberg (DH) Parameters
d1 = 162.5;
a2 = 425;
a3 = 392.2;
d4 = 133.3;
d5 = 99.7;
d6 = 99.6;

dhparams = [
    0, pi/2, L1, 0;
    L2, 0, 0, 0;
    L3, 0, 0, 0;
    0, pi/2, L4, 0;
    0, -pi/2, L5, 0;
    0, 0, L6, 0;
];

% Define the link lengths
links = [L1 L2 L3 L4 L5 L6];

% Create the figure and set the initial configuration
figure;
ax = gca; % Get current axis
hold on;
plotUR5(joint_angles(1, :), dhparams, links, ax); % Display the initial configuration
axis equal; % Keep axis scaling consistent
%axis([-500 500 -500 500 -200 800])
view(3); % 3D view
title('UR5e Manipulator Robot Animation');
grid on;

% Set the animation speed (frames per second)
framesPerSecond = 1;
dt = 1 / framesPerSecond;

% Animate the robot motion using the joint angle list
for i = 2:size(joint_angles, 1) % Start from the second set of joint angles
    cla(ax); % Clear the plot to avoid overlapping
    plotUR5(joint_angles(i, :), dhparams, links, ax); % Show new configuration
    drawnow; % Refresh the plot
    pause(dt); % Control frame rate
end

% Function to plot the UR5 robot given the joint angles and DH parameters
function plotUR5(q, dhparams,links, ax)
    T = eye(4); % Initialize transformation matrix

    % Initialize end-effector position
    end_effector = zeros(3, size(q, 2));

    % Define colors for each link
link_colors = {'r', 'g', 'b', 'k', 'm', 'y'};

% Plot each link with its individual color
for i = 1:length(q)
    T = T * DHTransform(dhparams(i, :), q(i)); % Update transformation matrix

    % Plot link with individual color
    if i > 1
        plot3([T(1, 4) end_effector(1, i-1)], [T(2, 4) end_effector(2, i-1)], [T(3, 4) end_effector(3, i-1)], link_colors{i}, 'LineWidth', 3);
    end
    end_effector(:, i) = T(1:3, 4);
    hold on;
end

% Plot end-effector
plot3(T(1, 4), T(2, 4), T(3, 4), 'ro', 'MarkerSize', 5, 'MarkerFaceColor', 'r');

end

% Function to compute the Denavit-Hartenberg transformation matrix
function T = DHTransform(params, q)
    a = params(1);
    alpha = params(2);
    d = params(3);
    theta = params(4);

    T = [
        cos(q), -sin(q)*cos(alpha), sin(q)*sin(alpha), a*cos(q);
        sin(q), cos(q)*cos(alpha), -cos(q)*sin(alpha), a*sin(q);
        0, sin(alpha), cos(alpha), d;
        0, 0, 0, 1
    ];
end
```