ME5250: Robot Mechanics and Control (Spring 2024)

Project 2

Submitted by:
Rayyan Muhammad Rafikh

About

This project is aimed at the Inverse Kinematics Simulation of a 6-DOF UR5 Cobot (by Universal Robots). Firstly, the forward kinematics of the Cobot is formulated with the help of physical Denavit-Hartenberg parameters provided by published research. The forward kinematics solution gives us the transformations from the base link/joint to successive links/joints. The project then proceeds by generating inverse kinematics solutions for a desired trajectory of the end-effector, which was done through the Newton-Raphson implementation of the Inverse Kinematics Solver. The trajectory (desired) chosen for our Inverse Kinematics problem is a planar square which was sampled at 1mm intervals in the x-y plane. Lastly, the simulation is an animation of the 6 links of the cobot rotating (about the 6 joints) in the x-y-z space, with the end-effector following the desired square shaped trajectory.

Forward Kinematics

The base-to-end-effector transformation matrix was referenced from [1]. The position vector of the end-effector tool tip was noted to be as follows:

$$
\begin{aligned}
p_x =& d_5 c_1 s_{234} + d_4 s_1 - d_6 c_1 c_{234} + a_2 c_1 c_2 + d_6 c_5 s_1 \\
&+ a_3 c_1 c_2 c_3 - a_3 c_1 s_2 s_3 \\
p_y =& d_5 s_1 s_{234} - d_4 c_1 - d_6 s_1 c_{234} - d_6 c_1 c_5 + a_2 c_2 s_1 \\
&+ a_3 c_2 c_3 s_1 - a_3 s_1 s_2 s_3 \\
p_z =& d_1 - d_6 s_{234} s_5 + a_3 s_{23} + a_2 s_2 - d_5 c_{234}
\end{aligned}
$$

where $c_{i..j} = \cos(i + \cdots + j)$, $s_{i..j} = \sin(i + \cdots + j)$, and d_1, a_2, a_3, d_4, d_5, d_6 are the link lengths. The study followed the DH Formulation method to arrive at this result. The individual DH transformation matrices are populated using the DH parameters given by Universal Robots in [2]. The matrices are then post-multiplied in order from the base to the end-effector resulting in T_{06}. The aforementioned position vector is then simply determined as the top three elements of the last column of T_{06}. The transformation matrices, and hence the position vector provided by the paper were verified through my implementation of the forward Kinematics using MATLAB.

Inverse Kinematics

The three elements of the position vector are each partially differentiated with respect to each (six) of the joint angles of the UR5 Cobot. This would give us elements of the coordinate Jacobian, which when stacked together result in the complete coordinate Jacobian required for computation of Inverse Kinematics through the Newton Raphson method.

The following pictures show the determination of the partial derivatives (which are components of our coordinate Jacobian) –

$$P_x = d_5 C_1 S_{234} + d_4 S_1 - d_6 C_1 C_{234} + a_2 C_1 C_2 + d_6 C_5 S_1 + a_3 C_1 C_2 C_3 - a_3 C_1 S_2 S_3$$

$$\frac{\partial P_x}{\partial \theta_1} = -d_5 S_1 S_{234} + d_4 C_1 + d_6 S_1 C_{234} - a_2 S_1 C_2 + d_6 C_5 C_1 - a_3 S_1 C_2 C_3 + a_3 S_1 S_2 S_3$$

$$\frac{\partial P_x}{\partial \theta_2} = d_5 C_1 C_{234} + 0 + d_6 C_1 S_{234} - a_2 C_1 S_2 + 0 - a_3 C_1 S_2 C_3 - a_3 C_1 C_2 S_3$$

$$\frac{\partial P_x}{\partial \theta_3} = d_5 C_1 C_{234} + 0 + d_6 C_1 S_{234} + 0 + 0 - a_3 C_1 C_2 S_3 - a_3 C_1 S_2 C_3$$

$$\frac{\partial P_x}{\partial \theta_4} = d_5 C_1 C_{234} + 0 + d_6 C_1 S_{234} + 0 + 0 + 0 + 0$$

$$\frac{\partial P_x}{\partial \theta_5} = 0 + 0 + 0 + 0 - d_6 S_5 S_1 + 0 + 0$$

$$\frac{\partial P_x}{\partial \theta_6} = 0 + 0 + 0 + 0 + 0 + 0 + 0$$

$$P_y = d_5 S_1 S_{234} - d_4 C_1 - d_6 S_1 C_{234} - d_6 C_1 C_5 + a_2 C_2 S_1 + a_3 C_2 C_3 S_1 - a_3 S_1 S_2 S_3$$

$$\frac{\partial P_y}{\partial \theta_1} = d_5 C_1 S_{234} + d_4 S_1 - d_6 C_1 C_{234} + d_6 S_1 C_5 + a_2 C_2 C_1 + a_3 C_2 C_3 C_1 - a_3 C_1 S_2 S_3$$

$$\frac{\partial P_y}{\partial \theta_2} = d_5 S_1 C_{234} + 0 + d_6 S_1 S_{234} + 0 - a_2 S_2 S_1 - a_3 S_2 C_3 S_1 - a_3 S_1 C_2 S_3$$

$$\frac{\partial P_y}{\partial \theta_3} = d_5 S_1 C_{234} + 0 + d_6 S_1 S_{234} + 0 + 0 - a_3 C_2 S_3 S_1 - a_3 S_1 S_2 C_3$$

$$\frac{\partial P_y}{\partial \theta_4} = d_5 S_1 C_{234} + 0 + d_6 S_1 S_{234} + 0 + 0 + 0 + 0$$

$$\frac{\partial P_y}{\partial \theta_5} = 0 + 0 + 0 + d_6 C_1 S_5 + 0 + 0 + 0$$

$$\frac{\partial P_y}{\partial \theta_6} = 0 + 0 + 0 + 0 + 0 + 0 + 0$$

$$P_z = d_1 - d_6 S_{234} S_5 + a_3 S_{23} + a_2 S_2 - d_5 C_{234}$$

$$\frac{\partial P_z}{\partial \theta_1} = 0 + 0 + 0 + 0 + 0$$

$$\frac{\partial P_z}{\partial \theta_2} = 0 - d_6 C_{234} S_5 + a_3 C_{23} + a_2 C_2 + d_5 S_{234}$$

$$\frac{\partial P_z}{\partial \theta_3} = 0 - d_6 C_{234} S_5 + a_3 C_{23} + 0 + d_5 S_{234}$$

$$\frac{\partial P_z}{\partial \theta_4} = 0 - d_6 C_{234} S_5 + 0 + 0 + d_5 S_{234}$$

$$\frac{\partial P_z}{\partial \theta_5} = 0 - d_6 S_{234} C_5 + 0 + 0 + 0$$

$$\frac{\partial P_z}{\partial \theta_6} = 0 + 0 + 0 + 0 + 0$$

$$J = \begin{bmatrix} \frac{\partial P_x}{\partial \theta_1} & \frac{\partial P_x}{\partial \theta_2} & \frac{\partial P_x}{\partial \theta_3} & \frac{\partial P_x}{\partial \theta_4} & \frac{\partial P_x}{\partial \theta_5} & \frac{\partial P_x}{\partial \theta_6} \\ \frac{\partial P_y}{\partial \theta_1} & \frac{\partial P_y}{\partial \theta_2} & \frac{\partial P_y}{\partial \theta_3} & \frac{\partial P_y}{\partial \theta_4} & \frac{\partial P_y}{\partial \theta_5} & \frac{\partial P_y}{\partial \theta_6} \\ \frac{\partial P_z}{\partial \theta_1} & \frac{\partial P_z}{\partial \theta_2} & \frac{\partial P_z}{\partial \theta_3} & \frac{\partial P_z}{\partial \theta_4} & \frac{\partial P_z}{\partial \theta_5} & \frac{\partial P_z}{\partial \theta_6} \end{bmatrix}$$

The code implemented in MATLAB takes care of all computations through the following steps –

1. Points along the perimeter of a square are sampled and stored in memory as a 3-column vector, with columns corresponding to x, y and z values of the end-effector tooltip when tracing the desired square shaped trajectory.
2. We then define variables to initialize our Newton Inverse Kinematics Solver, consisting of the tolerance (error at which to terminate computation of new joint angles), the initial guess for the 6 joint angles (as a single 1x6 vector).
3. We then enter a loop over the list of sampled points. In this loop, we first define our desired transformation matrix by having the top three elements of the last column of the matrix as our x, y and z values obtained from the sampled point (in the current iteration of the loop). We then iteratively try to converge to an error (resulting from the subtraction of the top three values of the last column from the top three values of the last column of the last three column vectors of our initial guess (which is fixed for the computation of joint angles for each desired point). The joint angles computed at the end of each convergence are appended to a list.
4. The populated list of joint angles is then used to visualize all the links of the cobot (as a whole) in a stick figure form as the end-effector traverse the desired square shape over the sampled points.

Link to UR5 Animation –

https://youtu.be/xci_1yeBswg

The robot, as observed from the video, does not seem to follow the desired square path during the start of its journey. This may be due to the vague initial guess of theta value of 20 degrees for each joint angle, which lands the end-effector at a position much far away from the first sampled point.

References:

[1] Kebria, Parham M., et al. "Kinematic and dynamic modelling of UR5 manipulator." *2016 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 2016.

[2] https://www.universal-robots.com/articles/ur/application-installation/dh-parameters-for-calculations-of-kinematics-and-dynamics/

[3] Stackoverflow

[4] Modern Robotics by Lynch and Park

Appendix

1. Forward Kinematics MATLAB Code

****

```
c_thetai = sym('c_thetai');
s_thetai = sym('s_thetai');
c_alphaiminus1 = sym('c_alphaiminus1');
s_alphaiminus1 = sym('s_alphaiminus1');
a_iminus1 = sym('a_iminus1');
d_i = sym('d_i');

%T_iminus1toi = [
%    c_thetai, -s_thetai, 0, a_iminus1;
%    s_thetai*c_alphaiminus1, c_thetai*c_alphaiminus1, -s_alphaiminus1, -s_alphaiminus1*d_i;
%    s_thetai*s_alphaiminus1, c_thetai*s_alphaiminus1, c_alphaiminus1, c_alphaiminus1*d_i;
%    0, 0, 0, 1]

T_iminus1toi = [
    c_thetai, -s_thetai*c_alphaiminus1, s_thetai*s_alphaiminus1, a_iminus1*c_thetai;
    s_thetai, c_thetai*c_alphaiminus1, -c_thetai*s_alphaiminus1, a_iminus1*s_thetai;
    0, s_alphaiminus1, c_alphaiminus1, d_i;
    0, 0, 0, 1];

a_iminus1 = 0;
c_alphaiminus1 = 0;
s_alphaiminus1 = 1;
c1 = sym('c1');
s1 = sym('s1');
c_thetai = c1;
s_thetai = s1;
d_1 = sym('d_1')
d_i = d_1;

T_01 = [
    c_thetai, -s_thetai*c_alphaiminus1, s_thetai*s_alphaiminus1, a_iminus1*c_thetai;
    s_thetai, c_thetai*c_alphaiminus1, -c_thetai*s_alphaiminus1, a_iminus1*s_thetai;
    0, s_alphaiminus1, c_alphaiminus1, d_i;
    0, 0, 0, 1];

a2 = sym('a2')
a_iminus1 = a2;
c_alphaiminus1 = 1;
s_alphaiminus1 = 0;
c2 = sym('c2');
s2 = sym('s2');
c_thetai = c2;
s_thetai = s2;
d_i = 0;

T_12 = [
```

```
    c_thetai, -s_thetai*c_alphaiminus1, s_thetai*s_alphaiminus1, a_iminus1*c_thetai;
    s_thetai, c_thetai*c_alphaiminus1, -c_thetai*s_alphaiminus1, a_iminus1*s_thetai;
    0, s_alphaiminus1, c_alphaiminus1, d_i;
    0, 0, 0, 1];

a3 = sym('a3')
a_iminus1 = a3;
c_alphaiminus1 = 1;
s_alphaiminus1 = 0;
c3 = sym('c3');
s3 = sym('s3');
c_thetai = c3;
s_thetai = s3;
d_i = 0;

T_23 = [
    c_thetai, -s_thetai*c_alphaiminus1, s_thetai*s_alphaiminus1, a_iminus1*c_thetai;
    s_thetai, c_thetai*c_alphaiminus1, -c_thetai*s_alphaiminus1, a_iminus1*s_thetai;
    0, s_alphaiminus1, c_alphaiminus1, d_i;
    0, 0, 0, 1];

a_iminus1 = 0;
c_alphaiminus1 = 0;
s_alphaiminus1 = 1;
c4 = sym('c4');
s4 = sym('s4');
c_thetai = c4;
s_thetai = s4;
d_4 = sym('d_4')
d_i = d_4;

T_34 = [
    c_thetai, -s_thetai*c_alphaiminus1, s_thetai*s_alphaiminus1, a_iminus1*c_thetai;
    s_thetai, c_thetai*c_alphaiminus1, -c_thetai*s_alphaiminus1, a_iminus1*s_thetai;
    0, s_alphaiminus1, c_alphaiminus1, d_i;
    0, 0, 0, 1];

a_iminus1 = 0;
c_alphaiminus1 = 0;
s_alphaiminus1 = -1;
c5 = sym('c5');
s5 = sym('s5');
c_thetai = c5;
s_thetai = s5;
d_5 = sym('d_5')
d_i = d_5;

T_45 = [
    c_thetai, -s_thetai*c_alphaiminus1, s_thetai*s_alphaiminus1, a_iminus1*c_thetai;
    s_thetai, c_thetai*c_alphaiminus1, -c_thetai*s_alphaiminus1, a_iminus1*s_thetai;
    0, s_alphaiminus1, c_alphaiminus1, d_i;
    0, 0, 0, 1];

a_iminus1 = 0;
c_alphaiminus1 = 1;
s_alphaiminus1 = 0;
c6 = sym('c6');
s6 = sym('s6');
c_thetai = c6;
s_thetai = s6;
d_6 = sym('d_6')
d_i = d_6;
```

```
T_56 = [
    c_thetai, -s_thetai*c_alphaiminus1, s_thetai*s_alphaiminus1, a_iminus1*c_thetai;
    s_thetai, c_thetai*c_alphaiminus1, -c_thetai*s_alphaiminus1, a_iminus1*s_thetai;
    0, s_alphaiminus1, c_alphaiminus1, d_i;
    0, 0, 0, 1];

T_06 = T_01*T_12*T_23*T_34*T_45*T_56;

end_effector_position = T_06(1:3, 4)
```

****

2. Points Sampling MATLAB Code

****

```
square_size = 100; % 100x100 mm square

interval = 1; % 1mm interval

num_points_per_side = floor(square_size / interval);

points = [];

points = [points; [0:interval:square_size]', ones(num_points_per_side + 1, 1) * square_size];

points = [points; ones(num_points_per_side, 1) * square_size, (square_size - interval:-
interval:0)'];

points = [points; (square_size - interval:-interval:0)', zeros(num_points_per_side, 1)];

points = [points; zeros(num_points_per_side, 1), (0:interval:square_size - interval)', ];

points = points + 50;
points(:,3) = 0;
```

****

3. Newton's Inverse Kinematics Solver MATLAB Code

****

```
initial_guess = [20 20 20 20 20 20]; % in degrees
initial_guess = initial_guess*0.0175; % degrees to radians

syms x y z real;

T_desired = [
    -0.5 -0.866 0 x;
    0.866 -0.5 0 y;
    0 0 1 z;
    0 0 0 1];

max_iterations = 200;
```

```
tolerance = 1;

[nrows, ncols] = size(points);
list_desired_joint_angles = zeros(nrows,6);
for i = 1:nrows
    row = points(i,:);
    T_desired(1:3,4) = row;
    [output_joint_angles output_error]= newton_inverse_kinematics(T_desired, initial_guess,
max_iterations, tolerance);
    output_joint_angles = output_joint_angles/0.0175; % convert result from radians to degrees
    list_desired_joint_angles(i,:) = output_joint_angles;
    output_joint_angles = output_joint_angles*0.01744;
    initial_guess = [output_joint_angles];
end

function [jointangles, error] = newton_inverse_kinematics(T_desired, initial_guess, max_iterations,
tolerance)

    joint_angles = initial_guess;
    norm_error = inf;
    error = [inf inf inf];
    iteration = 0;

    while (abs(error(1)) > tolerance || abs(error(2)) > tolerance || abs(error(3)) > tolerance) &&
iteration < max_iterations

        T_current = forward_kinematics(joint_angles);
        jointangles = double(joint_angles);

        %Tdesired = T_desired(1:3, 4);
        %Tcurrent = T_current(1:3, 4);
        error = T_desired(1:3, 4) - T_current(1:3, 4);

        J = compute_jacobian(joint_angles);

        delta_theta = pinv(J) * (error);

        %joint_angles = double(joint_angles)
        %s1 = size(joint_angles);
        %s2 = size(delta_theta');
        joint_angles = joint_angles + delta_theta';
        %updatedjointangles = double(joint_angles)
        iteration = iteration + 1;
    end
    disp('LOOP ENDED ONCE');
    if iteration >= max_iterations
        disp('Inverse kinematics did not converge within the maximum number of iterations.');
    end
end

function T = forward_kinematics(joint_angles)

   syms q1 q2 q3 q4 q5 q6 real;
   d1 = 89.2; %mm
   d4 = 109.3; %mm
   d5 = 94.75; %mm
   d6 = 82.5; %mm
   a2 = 425; %mm
   a3 = 392; %mm
   r11 = cos(q1)*cos(q2+q3+q4)*cos(q5)*cos(q6) + cos(q6)*sin(q1)*sin(q5) -
cos(q1)*sin(q2+q3+q4)*sin(q6);
```

```
   r21 = cos(q2+q3+q4)*cos(q5)*cos(q6)*sin(q1) - cos(q1)*cos(q6)*sin(q5) -
sin(q1)*sin(q2+q3+q4)*sin(q6);
   r31 = cos(q5)*cos(q6)*sin(q2+q3+q4) + cos(q2+q3+q4)*sin(q6);
   r12 = -cos(q1)*cos(q2+q3+q4)*cos(q5)*cos(q6) - sin(q1)*sin(q5)*sin(q6) -
cos(q1)*cos(q6)*sin(q2+q3+q4);
   r22 = -cos(q2+q3+q4) + cos(q1)*sin(q5)*sin(q6) - cos(q6)*sin(q1)*sin(q2+q3+q4);
   r32 = -cos(q5)*sin(q2+q3+q4)*sin(q6) + cos(q2+q3+q4)*cos(q6);
   r13 = -cos(q1)*cos(q2+q3+q4)*sin(q5) + cos(q5)*sin(q1);
   r23 = -cos(q2+q3+q4)*sin(q1)*sin(q5) - cos(q1)*cos(q5);
   r33 = -sin(q2+q3+q4)*sin(q5);
   %p_x = r13*d6 + cos(q1)*(sin(q2+q3+q4)*d5 + cos(q2+q3)*a3 + cos(q2)*a2) + sin(q1)*d4;
   %p_y = r23*d6 + sin(q1)*(sin(q2+q3+q4)*d5 + cos(q2+q3)*a3 + cos(q2)*a2) - cos(q1)*d4;
   %p_z = r33*d6 - cos(q2+q3+q4)*d5 + sin(q2+q3)*a3 + sin(q2)*a2 + d1;
   p_x = d5*cos(q1)*sin(q2+q3+q4) + d4*sin(q1) - d6*cos(q1)*cos(q2+q3+q4) + a2*cos(q1)*cos(q2) +
d6*cos(q5)*sin(q1) + a3*cos(q1)*cos(q2)*cos(q3) - a3*cos(q1)*sin(q2)*sin(q3);
   p_y = d5*sin(q1)*sin(q2+q3+q4) - d4*cos(q1) - d6*sin(q1)*cos(q2+q3+q4) - d6*cos(q1)*cos(q5) +
a2*cos(q2)*sin(q1) + a3*cos(q2)*cos(q3)*sin(q1) - a3*sin(q1)*sin(q2)*sin(q3);
   p_z = d1 - d6*sin(q2+q3+q4)*sin(q5) + a3*sin(q2+q3) + a2*sin(q2) - d5*cos(q2+q3+q4);

   T = [
       r11 r12 r13 p_x;
       r21 r22 r23 p_y;
       r31 r32 r33 p_z;
       0 0 0 1];

   q1_value = joint_angles(1);
   q2_value = joint_angles(2);
   q3_value = joint_angles(3);
   q4_value = joint_angles(4);
   q5_value = joint_angles(5);
   q6_value = joint_angles(6);

   T_substituted = subs(T, q1, q1_value);
   T_substituted = subs(T_substituted, q2, q2_value);
   T_substituted = subs(T_substituted, q3, q3_value);
   T_substituted = subs(T_substituted, q4, q4_value);
   T_substituted = subs(T_substituted, q5, q5_value);
   T_substituted = subs(T_substituted, q6, q6_value);
   T_numeric = double(T_substituted);
   T = T_numeric;
end

function J = compute_jacobian(joint_angles)

    syms q1 q2 q3 q4 q5 q6 real;
    d1 = 89.2; %mm
    d4 = 109.3; %mm
    d5 = 94.75; %mm
    d6 = 82.5; %mm
    a2 = 425; %mm
    a3 = 392; %mm

    dp_x_1 = -d5*sin(q1)*sin(q2+q3+q4) + d4*cos(q1) + d6*sin(q1)*cos(q2+q3+q4) - a2*sin(q1)*cos(q2)
+ d6*cos(q5)*cos(q1) - a3*sin(q1)*cos(q2)*cos(q3) + a3*sin(q1)*sin(q2)*sin(q3);
    dp_x_2 = d5*cos(q1)*cos(q2+q3+q4) + d6*cos(q1)*sin(q2+q3+q4) - a2*cos(q1)*sin(q2) -
a3*cos(q1)*sin(q2)*cos(q3) - a3*cos(q1)*cos(q2)*sin(q3);
    dp_x_3 = d5*cos(q1)*cos(q2+q3+q4) + d6*cos(q1)*sin(q2+q3+q4) - a3*cos(q1)*cos(q2)*sin(q3) -
a3*cos(q1)*sin(q2)*cos(q3);
    dp_x_4 = d5*cos(q1)*cos(q2+q3+q4) + d6*cos(q1)*sin(q2+q3+q4);
    dp_x_5 = -d6*sin(q5)*sin(q1);
    dp_x_6 = 0;
```

```
    dp_y_1 = d5*cos(q1)*sin(q2+q3+q4) + d4*sin(q1) - d6*cos(q1)*cos(q2+q3+q4) + d6*sin(q1)*cos(q5) + a2*cos(q2)*cos(q1) + a3*cos(q2)*cos(q3)*cos(q1) - a3*cos(q1)*sin(q2)*sin(q3);
    dp_y_2 = d5*sin(q1)*cos(q2+q3+q4) + d6*sin(q1)*sin(q2+q3+q4) - a2*sin(q2)*sin(q1) - a3*sin(q2)*cos(q3)*sin(q1) - a3*sin(q1)*cos(q2)*sin(q3);
    dp_y_3 = d5*sin(q1)*cos(q2+q3+q4) + d6*sin(q1)*sin(q2+q3+q4) - a3*cos(q2)*sin(q3)*sin(q1) - a3*sin(q1)*sin(q2)*cos(q3);
    dp_y_4 = d5*sin(q1)*cos(q2+q3+q4) + d6*sin(q1)*sin(q2+q3+q4);
    dp_y_5 = d6*cos(q1)*sin(q5);
    dp_y_6 = 0;

    dp_z_1 = 0;
    dp_z_2 = -d6*cos(q2+q3+q4)*sin(q5) + a3*cos(q2+q3) + a2*cos(q2) + d5*sin(q2+q3+q4);
    dp_z_3 = -d6*cos(q2+q3+q4)*sin(q5) + a3*cos(q2+q3) + d5*sin(q2+q3+q4);;
    dp_z_4 = -d6*cos(q2+q3+q4)*sin(q5) + d5*sin(q2+q3+q4);
    dp_z_5 = -d6*sin(q2+q3+q4)*cos(q5);
    dp_z_6 = 0;

    J = [
        dp_x_1 dp_x_2 dp_x_3 dp_x_4 dp_x_5 dp_x_6;
        dp_y_1 dp_y_2 dp_y_3 dp_y_4 dp_y_5 dp_y_6;
        dp_z_1 dp_z_2 dp_z_3 dp_z_4 dp_z_5 dp_z_6];

    q1_value = joint_angles(1);
    q2_value = joint_angles(2);
    q3_value = joint_angles(3);
    q4_value = joint_angles(4);
    q5_value = joint_angles(5);
    q6_value = joint_angles(6);
    J_substituted = subs(J, q1, q1_value);
    J_substituted = subs(J_substituted, q2, q2_value);
    J_substituted = subs(J_substituted, q3, q3_value);
    J_substituted = subs(J_substituted, q4, q4_value);
    J_substituted = subs(J_substituted, q5, q5_value);
    J_substituted = subs(J_substituted, q6, q6_value);
    J_numeric = double(J_substituted);
    J = J_numeric;

end
```

****

4. Visualization of UR5 Cobot MATLAB Code

****

```
joint_angles = list_desired_joint_angles * 0.01744;

L_1 = 89.2;
L_2 = 425;
L_3 = 392;
L_4 = 109.3;
L_5 = 94.75;
L_6 = 82.5;

dh_params = [
    0, pi/2, L_1, 0;
    L_2, 0, 0, 0;
    L_3, 0, 0, 0;
    0, pi/2, L_4, 0;
```

```
    0, -pi/2, L_5, 0;
    0, 0, L_6, 0;
];

link_lengths = [L_1 L_2 L_3 L_4 L_5 L_6];

% Initialize end effector trajectory
end_effector_trajectory = zeros(size(joint_angles, 1), 3);

figure;
a_x = gca;
hold on;
plotUR5(joint_angles(1, :), dh_params, link_lengths, a_x);
axis equal;
view(3);
title('UR5 IK Simulation');
grid on;

d_t = 1 / 10;

for i = 2:size(joint_angles, 1)
    cla(a_x);
    plotUR5(joint_angles(i, :), dh_params, link_lengths, a_x);
    % Update end effector trajectory
    end_effector_trajectory(i, :) = getEndEffectorPosition(joint_angles(i, :), dh_params);
    % Plot trajectory
    plot3(end_effector_trajectory(1:i, 1), end_effector_trajectory(1:i, 2),
end_effector_trajectory(1:i, 3), 'b--', 'LineWidth', 1);
    drawnow;
    pause(d_t);
end

function plotUR5(q, dh_params, link_lengths, a_x)
    T = eye(4);

    end_effector = zeros(3, size(q, 2));

    for i = 1:length(q)
        T = T * dh_transform(dh_params(i, :), q(i));

        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)], 'g', 'LineWidth', 1);
        end
        end_effector(:, i) = T(1:3, 4);
        hold on;
    end

    plot3(T(1, 4), T(2, 4), T(3, 4), 'ro', 'MarkerSize', 5, 'MarkerFaceColor', 'b');
end

function T = dh_transform(params, q)
    a_iminus1 = params(1);
    alphaiminus1 = params(2);
    d_i = params(3);
    theta = params(4);

    T = [
        cos(q), -sin(q)*cos(alphaiminus1), sin(q)*sin(alphaiminus1), a_iminus1*cos(q);
        sin(q), cos(q)*cos(alphaiminus1), -cos(q)*sin(alphaiminus1), a_iminus1*sin(q);
        0, sin(alphaiminus1), cos(alphaiminus1), d_i;
        0, 0, 0, 1
```

```
    ];
end

function end_effector_position = getEndEffectorPosition(q, dh_params)
    T = eye(4);
    for i = 1:length(q)
        T = T * dh_transform(dh_params(i, :), q(i));
    end
    end_effector_position = T(1:3, 4)';
end
```

****