**Milestone 3 Report**

Mohannad Noaman 52-22111

Ziad sherif 52-3407

Ahmed Gobran 52-8677

Mazen Motaz 52-12369

Mohamed Farid 52-5020

Kirolos George 52-3525

**forward Kinematics Code :**

function T = forward\_kinematics\_func()

% Forward kinematics of a 4-DOF manipulator with symbolic joint angles

% Define symbolic variables for joint angles

syms q1 q2 q3 q4 real

%syms L1 L2 L3 L4 real

%{

% Define constants for joints angles

q1 = deg2rad(90); % Joint 1 angle (degrees to radians)

q2 = deg2rad(45); % Joint 2 angle (degrees to radians)

q3 = deg2rad(60); % Joint 3 angle (degrees to radians)

q4 = deg2rad(90); % Joint 4 angle (degrees to radians)

%

% Define constants for link lengths

L1 = 33.88; % Length of link 1

L2 = 115; % Length of link 2

L3 = 119; % Length of link 3

L4 = 68; % Length of link 4

%{

L1 = 43.55;

L2 = 115;

L3 = 120;

L4 = 68;

%}

% Define the homogeneous transformation matrices symbolically

A = [cos(q1 + pi/2), 0, -sin(q1 + pi/2), 0;

sin(q1 + pi/2), 0, cos(q1 + pi/2), 0;

0, -1, 0, L1;

0, 0, 0, 1];

B = [cos(q2 - pi/2), -sin(q2 - pi/2), 0, -L2 \* cos(q2 - pi/2);

sin(q2 - pi/2), cos(q2 - pi/2), 0, -L2 \* sin(q2 - pi/2);

0, 0, 1, 0;

0, 0, 0, 1];

C = [cos(q3), -sin(q3), 0, -L3 \* cos(q3);

sin(q3), cos(q3), 0, -L3 \* sin(q3);

0, 0, 1, 0;

0, 0, 0, 1];

D = [cos(q4), -sin(q4), 0, -L4 \* cos(q4);

sin(q4), cos(q4), 0, -L4 \* sin(q4);

0, 0, 1, 0;

0, 0, 0, 1];

% Calculate the overall symbolic transformation matrix

T = simplify(A \* B \* C \* D); % Simplify to get a cleaner symbolic expression

%T = A \* B \* C \* D;

disp(T)

% Extract the position of the end effector (fourth column, first three rows)

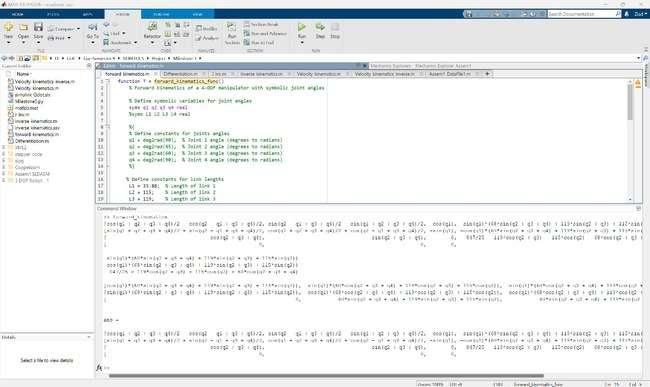
end\_effector\_position = T(1:3, 4);

disp(end\_effector\_position);

j = jacobian(end\_effector\_position,[q1,q2,q3,q4]);

    disp(j);

end



**Jacobian matrix code :**

Jacobian matrix

function J\_Matrix = calculate\_partial\_derivatives()

% Calculate the partial derivatives of the end-effector position with respect to each joint angle

% Get the transformation matrix T from the forward kinematics function

T = forward\_kinematics();

% Extract the fourth column elements of the end-effector position

f1 = T(1, 4); %X.E.E

f2 = T(2, 4); %Y.E.E

f3 = T(3, 4); %Z.E.E

%disp(f1);

%disp(f2);

%disp(f3)

% Define symbolic variables for joint angles to use for differentiation

syms q1 q2 q3 q4 real

% Calculate the partial derivatives with respect to each joint angle

partial\_q1\_f1 = diff(f1, q1);

partial\_q2\_f1 = diff(f1, q2);

partial\_q3\_f1 = diff(f1, q3);

partial\_q4\_f1 = diff(f1, q4);

partial\_q1\_f2 = diff(f2, q1);

partial\_q2\_f2 = diff(f2, q2);

partial\_q3\_f2 = diff(f2, q3);

partial\_q4\_f2 = diff(f2, q4);

partial\_q1\_f3 = diff(f3, q1);

partial\_q2\_f3 = diff(f3, q2);

partial\_q3\_f3 = diff(f3, q3);

partial\_q4\_f3 = diff(f3, q4);

% Store results in a matrix for easy interpretation

J\_Matrix = [partial\_q1\_f1, partial\_q2\_f1, partial\_q3\_f1, partial\_q4\_f1;

partial\_q1\_f2, partial\_q2\_f2, partial\_q3\_f2, partial\_q4\_f2;

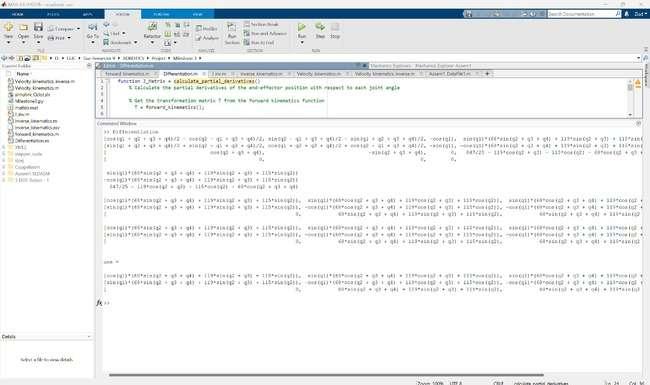
partial\_q1\_f3, partial\_q2\_f3, partial\_q3\_f3, partial\_q4\_f3];

disp(J\_Matrix);

% Display results

% disp('Partial derivatives of the end-effector first row, first column with respect to q1, q2, q3, and q4:'); % disp(partial\_derivatives);

End



**Jacobian inverse code :**

function J\_pseudo\_inv = inverse\_jacobian\_matrix(q)

% Get the Jacobian matrix by calling Differentiation

J\_Matrix = Differentiation();

%disp(Diff);

% Calculate the inverse of the Jacobian matrix

J\_transpose = J\_Matrix';

% Pseudo inverse since the robot is over actuated DOF < DOJ

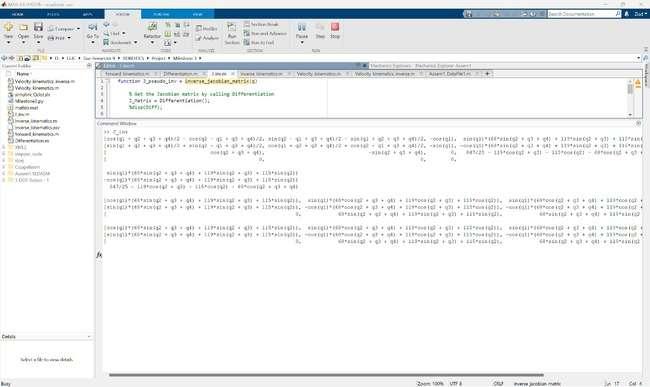
J\_pseudo\_inv = simplify((J\_transpose)\*(inv(J\_Matrix\*J\_transpose)));

% Display the inverse of the Jacobian matrix

disp('Pseudo inverse of the Jacobian matrix (J\_inv):');

disp(J\_pseudo\_inv);

end



**inverse kinematics Code :**

function q = inverse\_kinematics\_func(q0, X)

J\_pseudo\_inv = J\_inv();

T = forward\_kinematics();

end\_effector\_position = T(1:3, 4);

f1 = T(1, 4); %X.E.E

f2 = T(2, 4); %Y.E.E

f3 = T(3, 4); %Z.E.E

% Define symbolic variables

syms L1 L2 L3 L4 q1 q2 q3 q4

% Define constants for joints angles

%q1\_num = deg2rad(90); % Joint 1 angle (degrees to radians)

%q2\_num = deg2rad(45); % Joint 2 angle (degrees to radians)

%q3\_num = deg2rad(60); % Joint 3 angle (degrees to radians)

%q4\_num = deg2rad(90); % Joint 4 angle (degrees to radians)

% End-effector position matrix in matrix form

x\_desired = [178.6628; 0; 49.051];

%disp(x\_desired);

% Define F(Qn)

W = [f1; f2; f3];

%disp(W);

F\_Qn = W - x\_desired;

% Define the joint variables as a vector

% q\_new = [q1; q2; q3; q4];

q\_old = [1.6; 0.8; 1.1; 1.6];

% Define parameters:

i = 0;

error = 11;

err\_toler = 1e-4; % Convergence tolerance

max\_iters = 100; % Maximum number of iteration

while(error > err\_toler)

q1\_num = q\_old(1,1);

q2\_num = q\_old(2,1);

q3\_num = q\_old(3,1);

q4\_num = q\_old(4,1);

W\_sub = subs(W,[q1,q2,q3,q4],[q1\_num,q2\_num,q3\_num,q4\_num]);

F\_Qn = W\_sub - x\_desired;

p\_inv\_sub = vpa(subs(J\_pseudo\_inv,[q1,q2,q3,q4],[q1\_num,q2\_num,q3\_num,q4\_num]))

q\_new = vpa(q\_old - (p\_inv\_sub \* F\_Qn));

disp('Qnew =');

disp (q\_new);

q\_old = q\_new;

i=i+1;

if (i>100)

disp('Warning: Maximum iterations reached without convergence.');

random\_matrix = pi \*rand(4, 1);

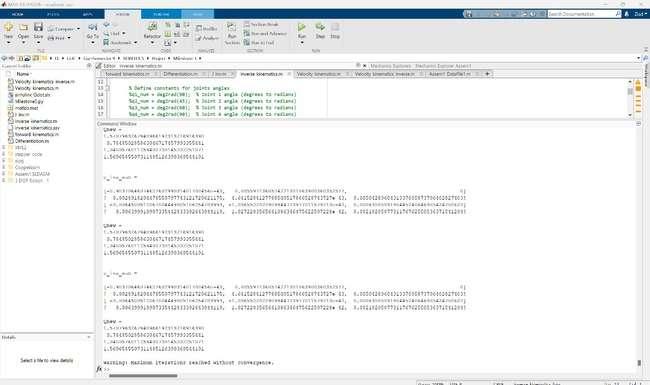
q\_old = random\_matrix;

break;

 end

    end

end



**velocity kinematics Code :**

function V = Velocity\_kinematics

coder.extrinsic('syms', 'subs', 'vpa', 'simplify');

syms p1 p2 p3 p4 q1\_dot q2\_dot q3\_dot q4\_dot real

syms t q1 q2 q3 q4

J\_matrix = Differentiation();

p1 = 0.1 \* t;

p2 = 0.3 \* t;

p3 = 0.4 \* t;

p4 = 0.2 \* t;

q1\_dot = 0.1;

q2\_dot = 0.3;

q3\_dot = 0.4;

q4\_dot = 0.2;

q\_dot = [q1\_dot;

q2\_dot;

q3\_dot;

q4\_dot];

tn = 0;

while (tn < 10)

q1\_subs = subs(p1,t,tn);

q2\_subs = subs(p2,t,tn);

q3\_subs = subs(p3,t,tn);

q4\_subs = subs(p4,t,tn);

J\_matrix\_sub = vpa(subs(J\_matrix,[q1,q2,q3,q4],[q1\_subs,q2\_subs,q3\_subs,q4\_subs]));

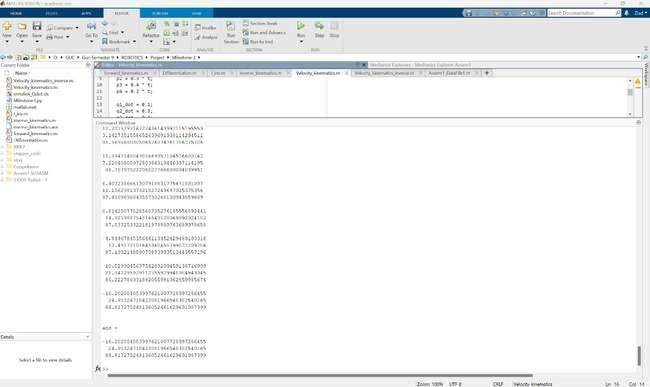
V = simplify(J\_matrix\_sub \* q\_dot);

disp(V);

tn = tn + 0.1;

end

end



**velocity inverse Code :**

function Q\_DOT = Velocity\_kinematics\_inverse

coder.extrinsic('syms', 'subs', 'vpa', 'simplify');

syms X\_dot Y\_dot Z\_dot real

syms p1 p2 p3 p4 q1\_dot q2\_dot q3\_dot q4\_dot real

syms t q1 q2 q3 q4

J\_pseudo\_inv = J\_inv();

V = Velocity\_kinematics();

J\_matrix = Differentiation();

p1 = 0.1 \* t;

p2 = 0.3 \* t;

p3 = 0.4 \* t;

p4 = 0.2 \* t;

X\_dot = V(1);

Y\_dot = V(2);

Z\_dot = V(3);

Q\_DOT = [q1\_dot;

q2\_dot;

q3\_dot;

q4\_dot];

tn = 0;

while (tn < 10)

q1\_subs = subs(p1,t,tn);

q2\_subs = subs(p2,t,tn);

q3\_subs = subs(p3,t,tn);

q4\_subs = subs(p4,t,tn);

J\_mat\_sub = vpa(subs(J\_matrix,[q1,q2,q3,q4],[q1\_subs,q2\_subs,q3\_subs,q4\_subs]));

J\_inv\_sub = pinv(J\_mat\_sub);

%J\_inv\_sub = vpa(subs(J\_pseudo\_inv,[q1,q2,q3,q4],[q1\_subs,q2\_subs,q3\_subs,q4\_subs]));

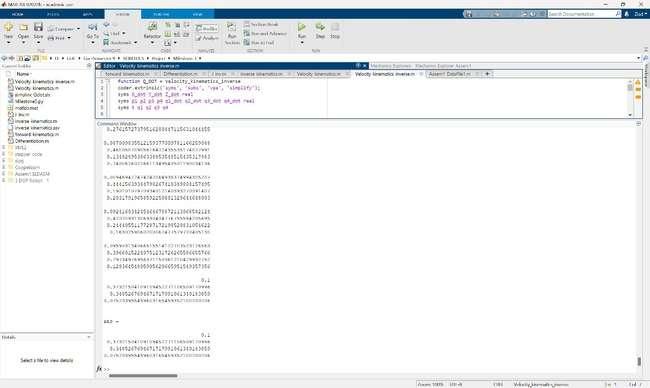
Q\_DOT = simplify(J\_inv\_sub \* V);

disp(Q\_DOT);

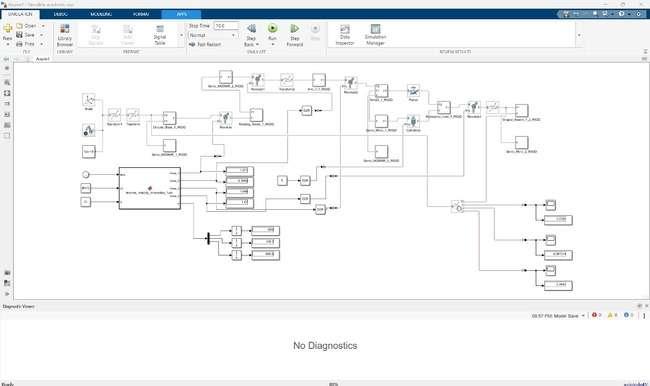
tn = tn + 0.1;

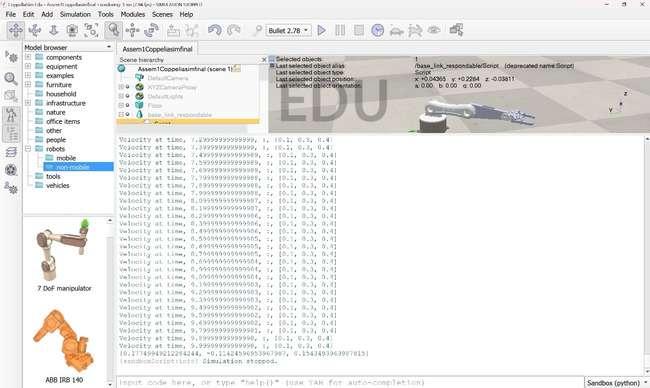
end

end

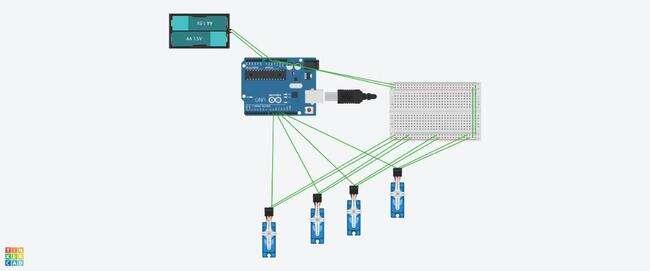


**Simulink simscape Diagram :**

****

**Copliasim results :**

**Circuit diagram :**

****

**Hardware :**

****

**Trajectory 1 Code:**

function Task\_Space\_1 = task\_traj(X0, Xf, Tf, Ts)

% Define initial and final positions, total time, and sampling time

X0 = [180, 0, 45]; % Initial position

Xf = [0, 25, 2]; % Final position

Tf = 5; % Total time (seconds)

Ts = 0.1; % Sampling time (seconds)

% Time vector

timeSteps = 0:Ts:Tf;

% Number of steps

nSteps = length(timeSteps);

% Initialize trajectory data

trajectory = zeros(nSteps, 3);

joint\_angles = zeros(nSteps, 4);

% Generate trajectory

for i = 1:nSteps

t = timeSteps(i) / Tf;

trajectory(i, :) = (1 - t) \* X0 + t \* Xf;

end

disp('Trajectory Points (Xf for each time step):');

disp(trajectory);

% Initial guess for joint angles

q0 = deg2rad([90; 45; 60; 90]);

% Calculate joint angles using inverse kinematics

for i = 1:nSteps

current\_Xf = trajectory(i, :)';

joint\_angles(i, :) = inverse\_kinematics(q0, current\_Xf);

end

disp('Joint Angles (q) for each time step:');

disp(rad2deg(joint\_angles)); % Display in degrees

% Plot trajectory in 2D

figure(1);

plot(timeSteps, trajectory(:, 1), 'r', ...

timeSteps, trajectory(:, 2), 'g', ...

timeSteps, trajectory(:, 3), 'b');

xlabel('Time (s)');

ylabel('Position');

title('End-Effector Trajectory (X, Y, Z)');

legend('X', 'Y', 'Z');

grid on;

% Plot joint angles

figure(2);

plot(timeSteps, rad2deg(joint\_angles));

xlabel('Time (s)');

ylabel('Joint Angles (degrees)');

title('Joint Angles Trajectory');

legend('q1', 'q2', 'q3', 'q4');

grid on;

% 3D trajectory plot and individual components

figure(3);

% 3D plot of trajectory

subplot(2, 2, 1);

plot3(trajectory(:, 1), trajectory(:, 2), trajectory(:, 3), '-o');

grid on;

xlabel('X (mm)');

ylabel('Y (mm)');

zlabel('Z (mm)');

title('3D End-Effector Trajectory');

view(3);

% X component

subplot(2, 2, 2);

plot(timeSteps, trajectory(:, 1), 'r');

xlabel('Time (s)');

ylabel('X Position (mm)');

title('X Trajectory');

grid on;

% Y component

subplot(2, 2, 3);

plot(timeSteps, trajectory(:, 2), 'g');

xlabel('Time (s)');

ylabel('Y Position (mm)');

title('Y Trajectory');

grid on;

% Z component

subplot(2, 2, 4);

plot(timeSteps, trajectory(:, 3), 'b');

xlabel('Time (s)');

ylabel('Z Position (mm)');

title('Z Trajectory');

grid on;

end

function q = inverse\_kinematics(q0, x)

% Robot parameters (lengths of links, offsets, etc.)

L1 = 100; % Link 1 length

L2 = 100; % Link 2 length

L3 = 50; % Link 3 length

% Initialize joint angles with the guess

q = q0;

% Extract the desired end-effector position

x\_des = x(1);

y\_des = x(2);

z\_des = x(3);

% Inverse kinematics equations (example for a planar 4-DOF arm)

% Calculate the base joint (q1) using atan2

q(1) = atan2(y\_des, x\_des); % Base joint

r = sqrt(x\_des^2 + y\_des^2); % Planar distance

% Calculate D for the elbow joint

D = (r^2 + (z\_des - L1)^2 - L2^2 - L3^2) / (2 \* L2 \* L3);

% Ensure D is in the range [-1, 1] for valid atan2 computation

if abs(D) > 1

D = sign(D); % Clamp D to 1 or -1

end

% Elbow joint (q3)

q(3) = atan2(sqrt(1 - D^2), D);

% Shoulder joint (q2)

q(2) = atan2(z\_des - L1, r) - atan2(L3 \* sin(q(3)), L2 + L3 \* cos(q(3)));

% Wrist joint (q4) - assuming it's aligned with the end-effector frame

q(4) = 0;

% Ensure joint angles are real (handle numerical errors)

q = real(q);

end

**Trajectory 2 code:**

function Task\_Space\_1 = task\_traj(X0, Xf, Tf, Ts)

% Define initial and final positions, total time, and sampling time

X0 = [0, 25, 2]; % Initial position

Xf = [40, 20, 35]; % Final position

Tf = 5; % Total time (seconds)

Ts = 0.1; % Sampling time (seconds)

% Time vector

timeSteps = 0:Ts:Tf;

% Number of steps

nSteps = length(timeSteps);

% Initialize trajectory data

trajectory = zeros(nSteps, 3);

joint\_angles = zeros(nSteps, 4);

% Generate trajectory

for i = 1:nSteps

t = timeSteps(i) / Tf;

trajectory(i, :) = (1 - t) \* X0 + t \* Xf;

end

disp('Trajectory Points (Xf for each time step):');

disp(trajectory);

% Initial guess for joint angles

q0 = deg2rad([90; 45; 60; 90]);

% Calculate joint angles using inverse kinematics

for i = 1:nSteps

current\_Xf = trajectory(i, :)';

joint\_angles(i, :) = inverse\_kinematics(q0, current\_Xf);

end

disp('Joint Angles (q) for each time step:');

disp(rad2deg(joint\_angles)); % Display in degrees

% Plot trajectory in 2D

figure(1);

plot(timeSteps, trajectory(:, 1), 'r', ...

timeSteps, trajectory(:, 2), 'g', ...

timeSteps, trajectory(:, 3), 'b');

xlabel('Time (s)');

ylabel('Position');

title('End-Effector Trajectory (X, Y, Z)');

legend('X', 'Y', 'Z');

grid on;

% Plot joint angles

figure(2);

plot(timeSteps, rad2deg(joint\_angles));

xlabel('Time (s)');

ylabel('Joint Angles (degrees)');

title('Joint Angles Trajectory');

legend('q1', 'q2', 'q3', 'q4');

grid on;

% 3D trajectory plot and individual components

figure(3);

% 3D plot of trajectory

subplot(2, 2, 1);

plot3(trajectory(:, 1), trajectory(:, 2), trajectory(:, 3), '-o');

grid on;

xlabel('X (mm)');

ylabel('Y (mm)');

zlabel('Z (mm)');

title('3D End-Effector Trajectory');

view(3);

% X component

subplot(2, 2, 2);

plot(timeSteps, trajectory(:, 1), 'r');

xlabel('Time (s)');

ylabel('X Position (mm)');

title('X Trajectory');

grid on;

% Y component

subplot(2, 2, 3);

plot(timeSteps, trajectory(:, 2), 'g');

xlabel('Time (s)');

ylabel('Y Position (mm)');

title('Y Trajectory');

grid on;

% Z component

subplot(2, 2, 4);

plot(timeSteps, trajectory(:, 3), 'b');

xlabel('Time (s)');

ylabel('Z Position (mm)');

title('Z Trajectory');

grid on;

end

function q = inverse\_kinematics(q0, x)

% Robot parameters (lengths of links, offsets, etc.)

L1 = 100; % Link 1 length

L2 = 100; % Link 2 length

L3 = 50; % Link 3 length

% Initialize joint angles with the guess

q = q0;

% Extract the desired end-effector position

x\_des = x(1);

y\_des = x(2);

z\_des = x(3);

% Inverse kinematics equations (example for a planar 4-DOF arm)

% Calculate the base joint (q1) using atan2

q(1) = atan2(y\_des, x\_des); % Base joint

r = sqrt(x\_des^2 + y\_des^2); % Planar distance

% Calculate D for the elbow joint

D = (r^2 + (z\_des - L1)^2 - L2^2 - L3^2) / (2 \* L2 \* L3);

% Ensure D is in the range [-1, 1] for valid atan2 computation

if abs(D) > 1

D = sign(D); % Clamp D to 1 or -1

end

% Elbow joint (q3)

q(3) = atan2(sqrt(1 - D^2), D);

% Shoulder joint (q2)

q(2) = atan2(z\_des - L1, r) - atan2(L3 \* sin(q(3)), L2 + L3 \* cos(q(3)));

% Wrist joint (q4) - assuming it's aligned with the end-effector frame

q(4) = 0;

% Ensure joint angles are real (handle numerical errors)

q = real(q);

end