INTERNATIONAL MULTI-AWARD WINNING INSTITUTION FOR SUSTAINABILITY

**MCTA 3332 & MCTE 4352**

**FUNDAMENTAL OF ROBOTICS**

**SEMESTER 1 2024/2025**

**LECTURER NAME: DR AHMAD IMRAN IBRAHIM**

**SECTION: 1**

**GROUP ASSIGNMENT**

**PROJECT: ROBOTICS**

| NO. | STUDENT'S NAME | MATRIC NO. |
|---|---|---|
| 1. | IBRAHIM BIN NASRUM | 2116467 |
| 2. | SHAREEN ARAWIE BIN HISHAM | 2116943 |
| 3. | AHMAD HAFIZULLAH BIN IBRAHIM | 2216185 |
| 4. | YOUSHA ABDULLAH | 1929821 |
| 5. | MUHAMMAD SAAD ELDIN | G2321795 |

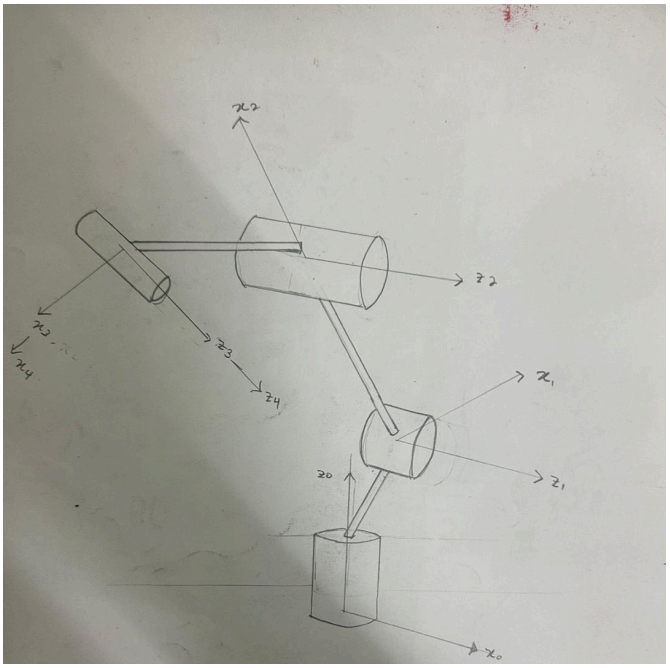
# Table Of Content

## 1.0 PROBLEM STATEMENT

The problem statements for this technical report on the 4 DOF ReBel robotic arm cover several important aspects of robotic kinematics, trajectory planning, and task execution. The first task is to model the end-effector's position using Denavit-Hartenberg (DH) parameters, which involves deriving the transformation matrices for each joint. Following this, the inverse kinematics of the robot is addressed to find the joint angles necessary to position the gripper at a specific location on a 4x4 rack, while respecting geometric constraints. Trajectory planning is then examined by plotting the joint trajectories (θ vs. time) for each joint during a pick-and-place operation. The report also includes a simulation of the entire pick-and-place process, ensuring that workpieces are placed in various rows and columns. The optimization of the robot’s motion is discussed, with the goal of minimizing execution time while taking joint limits and time constraints into account. Additionally, dynamic simulation is considered, which involves simulating the robot's motion and accounting for factors such as velocity and acceleration. We will visualize the robot’s movement in 3D space using MATLAB to display the trajectories of both the joints and the end-effector during the operation. Finally, an analysis of the robot's workspace is performed to identify the best placement of workpieces on the rack, ensuring that the robot can access all target positions without exceeding joint limits.

## 2.0 PROPOSED SOLUTION AND RESULT

### 2.1 Model of end-effector's position transformation matrix




### 2.2 DH- Representation Table

| # | Θ | d | a | α |
|---|---|---|---|---|
| 1 | Θ1 | d1 | 0 | 90° |
| 2 | Θ2 | 0 | a2 | 0 |
| 3 | Θ3 | 0 | a3 | 0 |
| 4 | Θ4 | d4 | 0 | 0 |

d1=252;

a2=237;

a3=244;

d4=0;

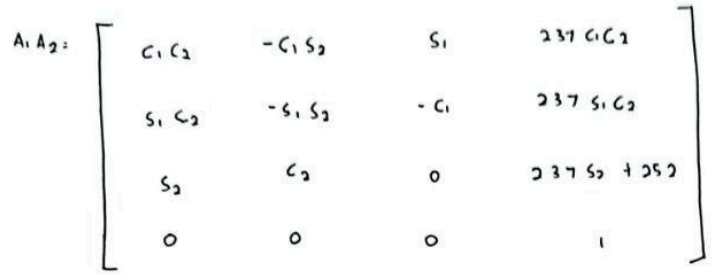
## 2.3 Transformation Matrix

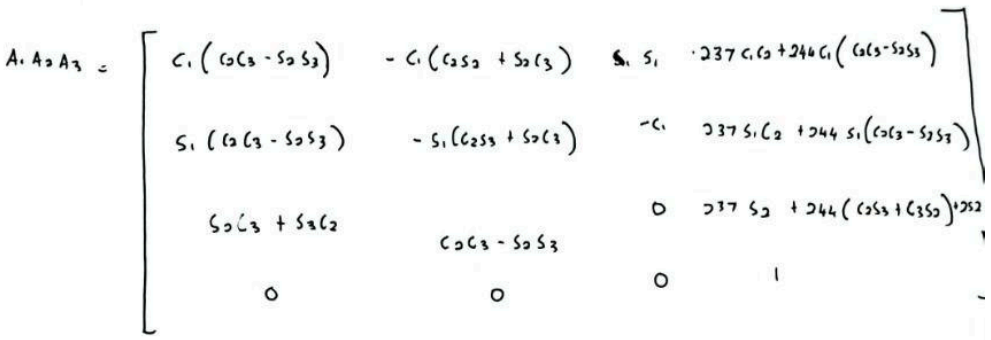
$$A_{n+1} = \begin{bmatrix} C\theta_{n+1} & -S\theta_{n+1}\,C\alpha_{n+1} & S\theta_{n+1}\,S\alpha_{n+1} & a_{n+1}\,C\theta_{n+1} \\ S\theta_{n+1} & C\theta_{n+1}\,C\alpha_{n+1} & -C\theta_{n+1}\,S\alpha_{n+1} & a_{n+1}\,S\theta_{n+1} \\ 0 & S\alpha_{n+1} & C\alpha_{n+1} & d_{n+1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A1 = $\begin{bmatrix} \cos(a) & 0 & \sin(a) & 0 \\ \sin(a) & 0 & -\cos(a) & 0 \\ 0 & 1 & 0 & 252 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

A2 = $\begin{bmatrix} \cos(b) & -\sin(b) & 0 & 237\cos(b) \\ \sin(b) & \cos(b) & 0 & 237\sin(b) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

A3 = $\begin{bmatrix} \cos(c) & -\sin(c) & 0 & 244\cos(c) \\ \sin(c) & \cos(c) & 0 & 244\sin(c) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$





A4 = $\begin{bmatrix} \cos(d) & -\sin(d) & 0 & 0 \\ \sin(d) & \cos(d) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

## 2.4 Transformation Matrix and Forward Kinematics Matlab Code:

```
syms a b c d  % Define symbolic variables for the joint angles

% Define the transformation matrix for each joint

mat1 = [cos(a), 0, sin(a), 0;

        sin(a), 0, -cos(a), 0;

        0, 1, 0, 252;

        0, 0, 0, 1];
```

```
mat2 = [cos(b), -sin(b), 0, 237*cos(b);
        sin(b), cos(b), 0, 237*sin(b);
        0, 0, 1, 0;
        0, 0, 0, 1];
mat3 = [cos(c), -sin(c), 0, 244*cos(c);
        sin(c), cos(c), 0, 244*sin(c);
        0, 0, 1, 0;
        0, 0, 0, 1];
mat4 = [cos(d), -sin(d), 0, 0;
        sin(d), cos(d), 0, 0;
        0, 0, 1, 0;
        0, 0, 0, 1];
% Calculate the combined transformation matrices
One = mat1 * mat2;    % Transformation from base to second link
Two = One * mat3;     % Transformation from base to third link
Three = Two * mat4;   % Transformation from base to end-effector
% Display the transformation matrices
disp('Transformation Matrix from A1 to A2:');
disp(One);
disp('Transformation Matrix from A1 to A3:');
disp(Two);
disp('Transformation Matrix from A1 to A4 (End-effector):');
disp(Three);
```

Below, we see the resultant matrices given.

Command Terminal:

```
Transformation_matrix

Transformation Matrix from A1 to A2:

[cos(a)*cos(b), -cos(a)*sin(b),  sin(a), 237*cos(a)*cos(b)]

[cos(b)*sin(a), -sin(a)*sin(b), -cos(a), 237*cos(b)*sin(a)]

[       sin(b),         cos(b),       0,  237*sin(b) + 252]

[            0,              0,       0,                 1]


      Transformation Matrix from A1 to A3:

      [cos(a)*cos(b)*cos(c) - cos(a)*sin(b)*sin(c), - cos(a)*cos(b)*sin(c) -
      cos(a)*cos(c)*sin(b),             sin(a),      237*cos(a)*cos(b)      +
      244*cos(a)*cos(b)*cos(c) - 244*cos(a)*sin(b)*sin(c)]

      [cos(b)*cos(c)*sin(a) - sin(a)*sin(b)*sin(c), - cos(b)*sin(a)*sin(c) -
      cos(c)*sin(a)*sin(b),        -cos(a),        237*cos(b)*sin(a)        +
      244*cos(b)*cos(c)*sin(a) - 244*sin(a)*sin(b)*sin(c)]

      [                                    cos(b)*sin(c)   +   cos(c)*sin(b),
      cos(b)*cos(c) - sin(b)*sin(c),         0,                  237*sin(b) +
      244*cos(b)*sin(c) + 244*cos(c)*sin(b) + 252]

      [                                                                    0,
      0,                                                                   0,
      1]


Transformation Matrix from A1 to A4 (End-effector):

[cos(d)*(cos(a)*cos(b)*cos(c)         -         cos(a)*sin(b)*sin(c))        -
sin(d)*(cos(a)*cos(b)*sin(c)         +         cos(a)*cos(c)*sin(b)),        -
cos(d)*(cos(a)*cos(b)*sin(c)         +         cos(a)*cos(c)*sin(b))         -
sin(d)*(cos(a)*cos(b)*cos(c) - cos(a)*sin(b)*sin(c)),   sin(a), 142*sin(a) +
237*cos(a)*cos(b) + 244*cos(a)*cos(b)*cos(c) - 244*cos(a)*sin(b)*sin(c)]

[cos(d)*(cos(b)*cos(c)*sin(a)         -         sin(a)*sin(b)*sin(c))        -
sin(d)*(cos(b)*sin(a)*sin(c)         +         cos(c)*sin(a)*sin(b)),        -
cos(d)*(cos(b)*sin(a)*sin(c)         +         cos(c)*sin(a)*sin(b))         -
```

```
sin(d)*(cos(b)*cos(c)*sin(a)      -      sin(a)*sin(b)*sin(c)),      -cos(a),
237*cos(b)*sin(a)     -     142*cos(a)     +     244*cos(b)*cos(c)*sin(a)     -
244*sin(a)*sin(b)*sin(c)]

[                                    cos(d)*(cos(b)*sin(c) + cos(c)*sin(b)) -
sin(d)*(sin(b)*sin(c) - cos(b)*cos(c)),                                      -
cos(d)*(sin(b)*sin(c)    -    cos(b)*cos(c))    -    sin(d)*(cos(b)*sin(c)    +
cos(c)*sin(b)),          0,                                        237*sin(b) +
244*cos(b)*sin(c) + 244*cos(c)*sin(b) + 252]

[
0,
0,                                                                          0,
1]
```

## 2.5 Inverse Kinematics:

```
% 4-DOF Robotic Arm Inverse Kinematics

% Define robot parameters

d1 = 252;     % Base height (mm)

a2 = 237;     % Link 1 length (mm)

a3 = 244;     % Link 2 length (mm)

d4 = 0;     % Gripper offset (mm)

% Target positions (in mm)

WP1 = [-200, 20, 370.5];  % Pickup point 1

WP2 = [-200, 50, 370.5];  % Pickup point 2

Rack2 = [-180, 70, 550];  % Placement point 1

Rack16 = [180, 70, 450];   % Placement point 2

% Desired gripper orientation (in degrees, 0 for horizontal)

desired_orientation = 0;

% Compute joint angles for each target position
```

```
[theta1_WP1, theta2_WP1, theta3_WP1, theta4_WP1] = inverseKinematics(WP1(1), WP1(2), WP1(3), desired_orientation, d1, a2, a3, d4);

[theta1_WP2, theta2_WP2, theta3_WP2, theta4_WP2] = inverseKinematics(WP2(1), WP2(2), WP2(3), desired_orientation, d1, a2, a3, d4);

[theta1_Rack1, theta2_Rack1, theta3_Rack1, theta4_Rack1] = inverseKinematics(Rack1(1), Rack1(2), Rack1(3), desired_orientation, d1, a2, a3, d4);

[theta1_Rack6, theta2_Rack6, theta3_Rack6, theta4_Rack6] = inverseKinematics(Rack6(1), Rack6(2), Rack6(3), desired_orientation, d1, a2, a3, d4);

% Display the results

fprintf('Joint Angles for WP1 (Pickup 1): θ1=%.2f°, θ2=%.2f°, θ3=%.2f°, θ4=%.2f°\n', theta1_WP1, theta2_WP1, theta3_WP1, theta4_WP1);

fprintf('Joint Angles for WP2 (Pickup 2): θ1=%.2f°, θ2=%.2f°, θ3=%.2f°, θ4=%.2f°\n', theta1_WP2, theta2_WP2, theta3_WP2, theta4_WP2);

fprintf('Joint Angles for Rack2 (Place 1): θ1=%.2f°, θ2=%.2f°, θ3=%.2f°, θ4=%.2f°\n', theta1_Rack1, theta2_Rack1, theta3_Rack1, theta4_Rack1);

fprintf('Joint Angles for Rack16 (Place 2): θ1=%.2f°, θ2=%.2f°, θ3=%.2f°, θ4=%.2f°\n', theta1_Rack6, theta2_Rack6, theta3_Rack6, theta4_Rack6);

% Inverse Kinematics Function

function [theta1, theta2, theta3, theta4] = inverseKinematics(x, y, z, desired_orientation, d1, a2, a3, d4)

    % Compute theta1 (Base Joint)

    theta1 = atan2(y, x);

    % Project into the X-Z plane

    r = sqrt(x^2 + y^2);  % Horizontal distance

    s = z - d1;           % Vertical distance

    % Solve for theta3 (Elbow Joint) using the Law of Cosines

    D = (r^2 + s^2 - a2^2 - a3^2) / (2 * a2 * a3);

    if abs(D) > 1
```

```
        error('Target position is out of reach.');
    end
    theta3 = atan2(sqrt(1 - D^2), D); % Ensure positive elbow configuration
    % Solve for theta2 (Shoulder Joint)
    phi = atan2(s, r);  % Angle to the target point
    psi = atan2(a3 * sin(theta3), a2 + a3 * cos(theta3)); % Correction angle
    theta2 = phi - psi;
    % Solve for theta4 (Wrist Orientation)
    theta4 = deg2rad(desired_orientation) - (theta1 + theta2 + theta3);
    % Convert angles to degrees for output
    theta1 = rad2deg(theta1);
    theta2 = rad2deg(theta2);
    theta3 = rad2deg(theta3);
    theta4 = rad2deg(theta4);
end
```

Command Line:

```
>> InverseKinematics

Joint Angles for WP1 (Pickup 1): θ1=174.29°, θ2=-31.97°, θ3=121.99°,
θ4=-264.30°

Joint Angles for WP2 (Pickup 2): θ1=165.96°, θ2=-31.96°, θ3=120.77°,
θ4=-254.77°

Joint Angles for Rack2 (Place 1): θ1=158.75°, θ2=13.87°, θ3=84.84°,
θ4=-257.46°

Joint Angles for Rack16 (Place 2): θ1=21.25°, θ2=-10.38°, θ3=109.81°,
θ4=-120.68°
```

### 2.6 Verifying The Inverse Is True Using Forward Kinematics:

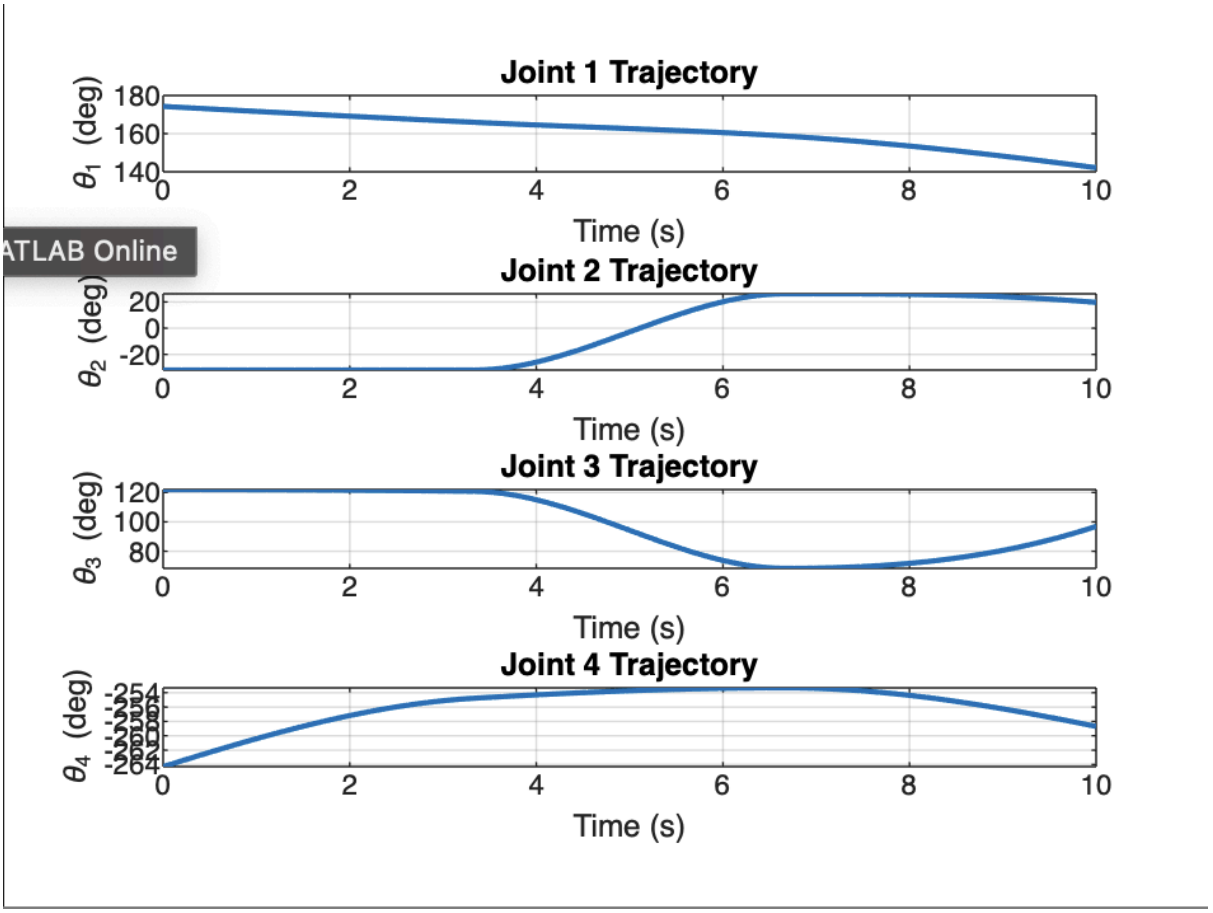
```
%input the thetas value to verify whether the angles that we got in inverse

%kinematics are true or not

function T4_0 = forwardKinematics(theta1, theta2, theta3, theta4)

   % Compute individual transformation matrices

   T1_0 = [

       cosd(theta1), 0, sind(theta1), 0;

       sind(theta1), 0, -cosd(theta1), 0;

       0, 1, 0, 252;

       0, 0, 0, 1

   ];

   T2_1 = [

       cosd(theta2), -sind(theta2), 0, 237*cosd(theta2);

       sind(theta2), cosd(theta2), 0, 237*sind(theta2);

       0, 0, 1, 0;

       0, 0, 0, 1

   ];

   T3_2 = [

       cosd(theta3), -sind(theta3), 0, 244*cosd(theta3);

       sind(theta3), cosd(theta3), 0, 244*sind(theta3);

       0, 0, 1, 0;

       0, 0, 0, 1

   ];

   T4_3 = [
```

```
        cosd(theta4), -sind(theta4), 0, 0;
        sind(theta4), cosd(theta4), 0, 0;
        0, 0, 1, 0;
        0, 0, 0, 1
    ];
    % Compute overall transformation matrix
    T4_0 = T1_0 * T2_1 * T3_2 * T4_3;
end
```

## 2.7 Command Window (Result of verifying using rack 16)

```
>> VerifyInverseTrue(21.25,-10.38,109.81,-120.68)


ans =


    0.8686    0.3378    0.3624  180.0115
    0.3378    0.1314   -0.9320   70.0027
   -0.3624    0.9320         0  450.0010
         0         0         0    1.0000
```

## 2.8 Trajectory Planning:



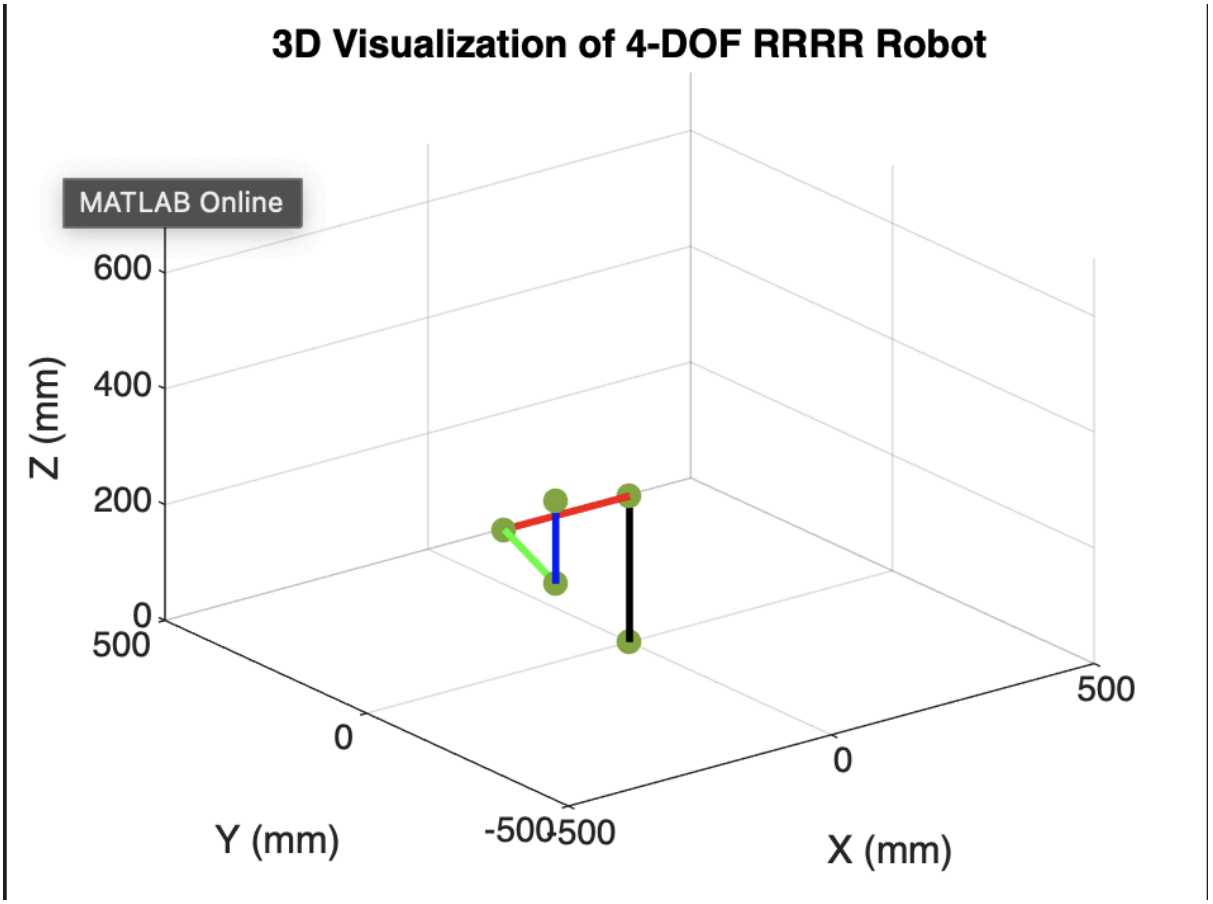

```
% Define robot parameters

d1 = 252;     % Base height (mm)

a2 = 237;     % Link 1 length (mm)

a3 = 244;     % Link 2 length (mm)

d4 = 142;     % Gripper offset (mm)

% Target positions (in mm)

waypoints = [

    -200, 20, 370.5;  % WP1: Pickup point 1

    -200, 50, 370.5;  % WP2: Pickup point 2

    -180, 70, 600;    % Rack1: Placement point 1

    -90,  70, 550     % Rack6: Placement point 2
```

```
];

% Desired gripper orientation (in degrees, 0 for horizontal)

desired_orientation = 0;

% Time for trajectory planning

t_total = 10;         % Total time for the trajectory (seconds)

n_points = 100;       % Number of points in the trajectory

time_steps = linspace(0, t_total, n_points);

% Preallocate storage for joint angles

theta_trajectory = zeros(n_points, 4);

% Compute joint angles for each waypoint

joint_angles = zeros(size(waypoints, 1), 4);

for i = 1:size(waypoints, 1)

     [theta1, theta2, theta3, theta4] = inverseKinematics(waypoints(i, 1),
waypoints(i, 2), waypoints(i, 3), desired_orientation, d1, a2, a3, d4);

   joint_angles(i, :) = [theta1, theta2, theta3, theta4];

end

% Trajectory planning using cubic polynomial interpolation

for j = 1:4  % For each joint

   theta_trajectory(:, j) = interp1(linspace(0, t_total, size(waypoints, 1)),
joint_angles(:, j), time_steps, 'pchip');

end

% Plot the trajectory for each joint

figure;

for j = 1:4

   subplot(4, 1, j);

   plot(time_steps, theta_trajectory(:, j), 'LineWidth', 1.5);
```

```
    title(['Joint ', num2str(j), ' Trajectory']);
    xlabel('Time (s)');
    ylabel(['\theta_', num2str(j), ' (deg)']);
    grid on;
end

% Function for Inverse Kinematics
function [theta1, theta2, theta3, theta4] = inverseKinematics(x, y, z, desired_orientation, d1, a2, a3, d4)
    % Compute theta1 (Base Joint)
    theta1 = atan2(y, x);
    % Project into the X-Z plane
    r = sqrt(x^2 + y^2);  % Horizontal distance
    s = z - d1;           % Vertical distance
    % Solve for theta3 (Elbow Joint) using the Law of Cosines
    D = (r^2 + s^2 - a2^2 - a3^2) / (2 * a2 * a3);
    if abs(D) > 1
        error('Target position is out of reach.');
    end
    theta3 = atan2(sqrt(1 - D^2), D); % Ensure positive elbow configuration
    % Solve for theta2 (Shoulder Joint)
    phi = atan2(s, r);  % Angle to the target point
    psi = atan2(a3 * sin(theta3), a2 + a3 * cos(theta3)); % Correction angle
    theta2 = phi - psi;
    % Solve for theta4 (Wrist Orientation)
    theta4 = deg2rad(desired_orientation) - (theta1 + theta2 + theta3);
```

```
    % Convert angles to degrees for output
    theta1 = rad2deg(theta1);
    theta2 = rad2deg(theta2);
    theta3 = rad2deg(theta3);
    theta4 = rad2deg(theta4);
end
```

### 2.9 3D Visualization of 4-DOF RRRR Robot:




```
% Define robot parameters
% MATLAB Script: 3D Visualization of 4-DOF RRRR Robot Trajectory
% Define robot parameters
d1 = 252;     % Base height (mm)
a2 = 237;     % Link 1 length (mm)
```

```
a3 = 244;     % Link 2 length (mm)

d4 = 142;     % Gripper offset (mm)

% Joint angles for waypoints

angles = [

   174.29, -31.97, 121.99, -264.30;  % WP1 to Rack1

   165.96, -31.96, 120.77, -254.77;  % WP2 to Rack6

   158.75,  26.24,  68.33, -253.32;  % Rack1

   142.13,  19.67,  96.90, -258.70   % Rack6

];

% Convert degrees to radians

angles_rad = deg2rad(angles);

% Number of steps for animation

n_steps = 50;

time_steps = linspace(0, 1, n_steps);

% Generate interpolated trajectories for each joint

trajectories = zeros(n_steps, size(angles, 2));

for j = 1:size(angles, 2)

      trajectories(:,  j)  =  interp1(1:size(angles,  1),  angles_rad(:,  j),
linspace(1, size(angles, 1), n_steps), 'pchip');

end

% 3D visualization setup

figure;

axis equal;

grid on;

view(3);
```

```
xlabel('X (mm)'); ylabel('Y (mm)'); zlabel('Z (mm)');

title('3D Visualization of 4-DOF RRRR Robot');

% Initialize link positions

base = [0; 0; 0];

for step = 1:n_steps

   % Extract joint angles for the current step

   theta1 = trajectories(step, 1);

   theta2 = trajectories(step, 2);

   theta3 = trajectories(step, 3);

   theta4 = trajectories(step, 4);

   % Compute transformation matrices

   T1 = [cos(theta1), -sin(theta1), 0, 0;

         sin(theta1),  cos(theta1), 0, 0;

         0,            0,           1, d1;

         0,            0,           0, 1];

   T2 = [cos(theta2), -sin(theta2), 0, a2*cos(theta2);

         sin(theta2),  cos(theta2), 0, a2*sin(theta2);

         0,            0,           1, 0;

         0,            0,           0, 1];

   T3 = [cos(theta3), -sin(theta3), 0, a3*cos(theta3);

         sin(theta3),  cos(theta3), 0, a3*sin(theta3);

         0,            0,           1, 0;

         0,            0,           0, 1];

   T4 = [cos(theta4), -sin(theta4), 0, 0;

         sin(theta4),  cos(theta4), 0, 0;
```

```
        0,          0,          1, d4;
        0,          0,          0, 1];
    % Compute link positions
    P1 = T1(1:3, 4);
    P2 = T1 * T2; P2 = P2(1:3, 4);
    P3 = T1 * T2 * T3; P3 = P3(1:3, 4);
    P4 = T1 * T2 * T3 * T4; P4 = P4(1:3, 4);
    % Plot the robot
    clf;
    hold on;
       plot3([base(1), P1(1)], [base(2), P1(2)], [base(3), P1(3)], 'k',
'LineWidth', 2);
    plot3([P1(1), P2(1)], [P1(2), P2(2)], [P1(3), P2(3)], 'r', 'LineWidth',
2);
    plot3([P2(1), P3(1)], [P2(2), P3(2)], [P2(3), P3(3)], 'g', 'LineWidth',
2);
    plot3([P3(1), P4(1)], [P3(2), P4(2)], [P3(3), P4(3)], 'b', 'LineWidth',
2);
    scatter3([base(1), P1(1), P2(1), P3(1), P4(1)], [base(2), P1(2), P2(2),
P3(2), P4(2)], [base(3), P1(3), P2(3), P3(3), P4(3)], 50, 'filled');
    % Format the plot
    axis([-500, 500, -500, 500, 0, 700]);
    view(3);
    xlabel('X (mm)'); ylabel('Y (mm)'); zlabel('Z (mm)');
    title('3D Visualization of 4-DOF RRRR Robot');
    grid on;
    drawnow;
```

```
end
```

## 2.10. Cartesian

```
% Plot the first trajectory

figure;

plot3(P(:, 1), P(:, 2), P(:, 3), 'b-', 'LineWidth', 2); % Trajectory 1 in
blue

hold on;

% Plot the second trajectory

plot3(P1(:, 1), P1(:, 2), P1(:, 3), 'r-', 'LineWidth', 2); % Trajectory 2 in
red

% Plot Start and End points

scatter3(P_start(1), P_start(2), P_start(3), 'ro', 'filled'); % Start point
in red

scatter3(P_end(1), P_end(2), P_end(3), 'go', 'filled');        % End point in
green

% Plot rack location points with markers and labels

scatter3(P_1(1), P_1(2), P_1(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_1(1) + 10, P_1(2), P_1(3), '1', 'FontSize', 10, 'Color', 'k');

scatter3(P_2(1), P_2(2), P_2(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_2(1) + 10, P_2(2), P_2(3), '2', 'FontSize', 10, 'Color', 'k');

scatter3(P_3(1), P_3(2), P_3(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_3(1) + 10, P_3(2), P_3(3), '3', 'FontSize', 10, 'Color', 'k');
```

```
scatter3(P_4(1), P_4(2), P_4(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_4(1) + 10, P_4(2), P_4(3), '4', 'FontSize', 10, 'Color', 'k');

scatter3(P_5(1), P_5(2), P_5(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_5(1) + 10, P_5(2), P_5(3), '5', 'FontSize', 10, 'Color', 'k');

scatter3(P_6(1), P_6(2), P_6(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_6(1) + 10, P_6(2), P_6(3), '6', 'FontSize', 10, 'Color', 'k');

scatter3(P_7(1), P_7(2), P_7(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_7(1) + 10, P_7(2), P_7(3), '7', 'FontSize', 10, 'Color', 'k');

scatter3(P_8(1), P_8(2), P_8(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_8(1) + 10, P_8(2), P_8(3), '8', 'FontSize', 10, 'Color', 'k');

scatter3(P_9(1), P_9(2), P_9(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_9(1) + 10, P_9(2), P_9(3), '9', 'FontSize', 10, 'Color', 'k');

scatter3(P_10(1), P_10(2), P_10(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_10(1) + 10, P_10(2), P_10(3), '10', 'FontSize', 10, 'Color', 'k');

scatter3(P_11(1), P_11(2), P_11(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_11(1) + 10, P_11(2), P_11(3), '11', 'FontSize', 10, 'Color', 'k');

scatter3(P_12(1), P_12(2), P_12(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_12(1) + 10, P_12(2), P_12(3), '12', 'FontSize', 10, 'Color', 'k');

scatter3(P_13(1), P_13(2), P_13(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');
```

```
text(P_13(1) + 10, P_13(2), P_13(3), '13', 'FontSize', 10, 'Color', 'k');

scatter3(P_14(1), P_14(2), P_14(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_14(1) + 10, P_14(2), P_14(3), '14', 'FontSize', 10, 'Color', 'k');

scatter3(P_15(1), P_15(2), P_15(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_15(1) + 10, P_15(2), P_15(3), '15', 'FontSize', 10, 'Color', 'k');

scatter3(P_16(1), P_16(2), P_16(3), 150, 's', 'MarkerFaceColor', [1, 0.5, 0],
'MarkerEdgeColor', 'k');

text(P_16(1) + 10, P_16(2), P_16(3), '16', 'FontSize', 10, 'Color', 'k');

% Plot Waypoints WP1 and WP2

scatter3(WP1(1), WP1(2), WP1(3), 150, 'o', 'MarkerFaceColor', [1, .5, 0],
'MarkerEdgeColor', 'k'); % WP1 in yellow

text(WP1(1) + 10, WP1(2), WP1(3), 'WP1', 'FontSize', 10, 'Color', 'k'); % Add
label for WP1

scatter3(WP2(1), WP2(2), WP2(3), 150, 'o', 'MarkerFaceColor', [1, 1, 0],
'MarkerEdgeColor', 'k'); % WP2 in yellow

text(WP2(1) + 10, WP2(2), WP2(3), 'WP2', 'FontSize', 10, 'Color', 'k'); % Add
label for WP2

% Set plot labels and title

xlabel('X (mm)');

ylabel('Y (mm)');

zlabel('Z (mm)');

title('Robot End-Effector Trajectories');

grid on;

% Add legend

legend('Trajectory 1', 'Trajectory 2', 'Start Point', 'End Point', 'Rack
Location Coordinate'); % Updated legend to include trajectories
```

```
hold off;
```

### 2.11 Graph plot of the theta, θ vs t graph for each joint.

Here, we have placed the graph showing the relationship between theta and time for each joint in the robot as it completes the desired task. Task 1 is picking the object from Workplace 1 at the coordinates [-200, 20, 370.5] , and then placing the object at rack 2, with the coordinates [-180, 70, 550]. And task 2 is picking the object from Workplace 2 at the coordinates [-200, 50, 370.5] , and then placing the object at rack 16, with the coordinates [180, 70, 450] respectively. The values shown in these graphs proves that the inverse kinematics values were true, because the initial values and final values are indeed similar when compared to our inverse kinematics solution.

```
>> InverseKinematics
Joint Angles for WP1 (Pickup 1): θ1=174.29°, θ2=-31.97°, θ3=121.99°, θ4=-264.30°
Joint Angles for WP2 (Pickup 2): θ1=165.96°, θ2=-31.96°, θ3=120.77°, θ4=-254.77°
Joint Angles for Rack2 (Place 1): θ1=158.75°, θ2=13.87°, θ3=84.84°, θ4=-257.46°
Joint Angles for Rack16 (Place 2): θ1=21.25°, θ2=-10.38°, θ3=109.81°, θ4=-120.68°
```

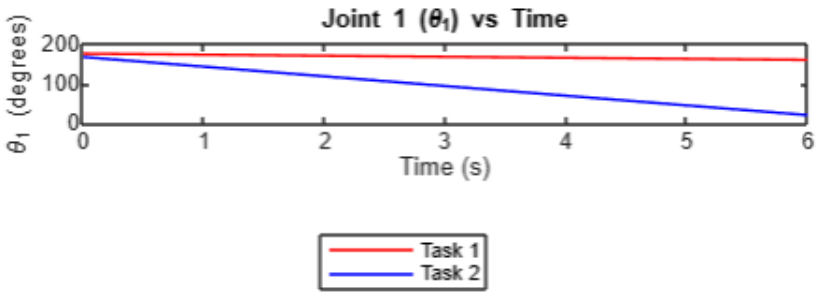



**Figure a** : Shows that initial angle of joint 1 for the first task was 174.29 degrees and the final angle is 158.75 degrees. And we can see that for task 2 , the initial angle was 165.96 degrees and the final angle was 21.25 degrees

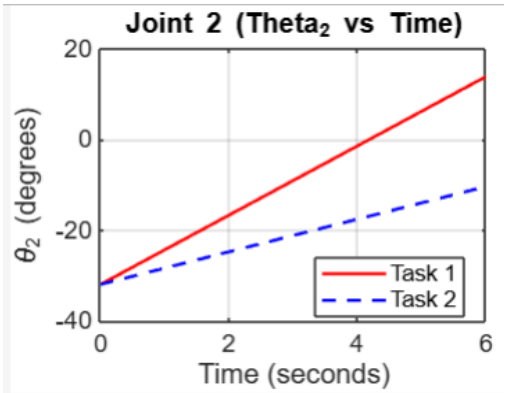



**Figure b** : Shows that initial angle of joint 2 for the first task was negative 31 degrees and the final angle is 13.87 degrees. And we can see that for task 2 , the initial angle was negative 31.96 degrees and the final angle was negative 10.38 degrees.

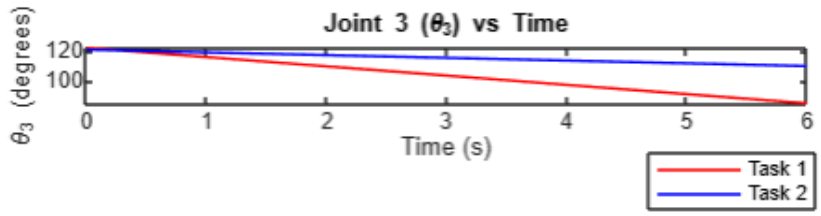



**Figure c** : Shows that initial angle of joint 3 for the first task was 121.99 degrees and the final angle is 84.84 degrees. And we can see that for task 2 , the initial angle was negative 120.77 degrees and the final angle was negative 109.81 degrees.

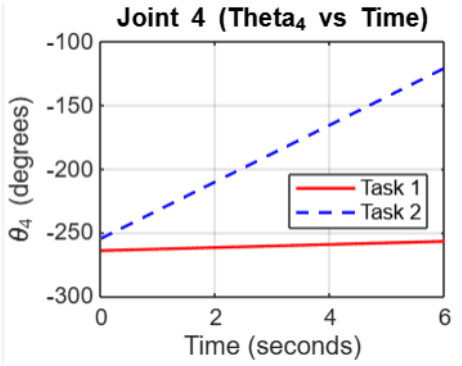



**Figure d** : Shows that the initial angle of joint4 for the first task was negative 264 degrees and the final angle is negative 257.46 degrees. And we can see that for task 2 , the initial angle was negative 254.77 degrees and the final angle was negative 120.68 degrees.

## 3.0 CODING or MATLAB

### 3.1 Matlab code to plot the theta VS time graph for each joint.

```
% Time vector for smooth trajectory
t = linspace(0, 6, 100); % 6 seconds duration
% Define Parameters (Assumed values for robot)
d1 = 252; % Base height (mm)
a2 = 237; % Length of link 2 (mm)
a3 = 244; % Length of link 3 (mm)
d4 = 62; % Assumed length of end-effector (mm)
% Pick and Place Points
WP1 = [-200, 20, 370.5]; % Pick location 1
WP2 = [-200, 50, 370.5]; % Pick location 2
Place1 = [-180, 70, 550]; % Example Place Rack 2
Place2 = [180, 70, 450]; % Example Place Rack 16
% Define Trajectory Points
PickPoints = [WP1; WP2];
```

```
PlacePoints = [Place1; Place2];
% Initialize joint angle storage
Theta1 = zeros(length(t), 4);
Theta2 = zeros(length(t), 4);
Theta3 = zeros(length(t), 4);
Theta4 = zeros(length(t), 4);
% Function to compute inverse kinematics (Assume planar motion for simplicity)
inverseKinematics = @(x, y, z) [
    atan2(y, x), ... % Theta1: Base rotation
    atan2(z - d1, sqrt(x^2 + y^2) - a2), ... % Theta2
    acos((x^2 + y^2 + (z - d1)^2 - a2^2 - a3^2) / (2 * a2 * a3)), ... % Theta3
    0];
% Compute joint trajectories
for i = 1:2 % Loop over pick and place tasks
    % Get pick and place points
    Pick = PickPoints(i, :);
    Place = PlacePoints(i, :);
    % Compute IK for start (pick) and end (place)
    theta_pick = inverseKinematics(Pick(1), Pick(2), Pick(3));
    theta_place = inverseKinematics(Place(1), Place(2), Place(3));
    % Interpolate joint angles
    for j = 1:4
        Theta1(:, i) = linspace(theta_pick(1), theta_place(1), length(t));
        Theta2(:, i) = linspace(theta_pick(2), theta_place(2), length(t));
        Theta3(:, i) = linspace(theta_pick(3), theta_place(3), length(t));
        Theta4(:, i) = linspace(theta_pick(4), theta_place(4), length(t));
    end
end
```

```
% Convert radians to degrees

Theta1 = rad2deg(Theta1);

Theta2 = rad2deg(Theta2);

Theta3 = rad2deg(Theta3);

Theta4 = rad2deg(Theta4);

% Plot Theta vs Time Graphs

figure;

subplot(4, 1, 1);

plot(t, Theta1(:, 1), 'r', t, Theta1(:, 2), 'b');

title('Joint 1 (\theta_1) vs Time');

xlabel('Time (s)');

ylabel('\theta_1 (degrees)');

legend('Task 1', 'Task 2');

subplot(4, 1, 3);

plot(t, Theta3(:, 1), 'r', t, Theta3(:, 2), 'b');

title('Joint 3 (\theta_3) vs Time');

xlabel('Time (s)');

ylabel('\theta_3 (degrees)');

legend('Task 1', 'Task 2');

time = linspace(0, 6, 100);

theta2_task1_initial = -31.97;

theta2_task1_final = 13.87;

theta2_task2_initial = -31.96;

theta2_task2_final = -10.38;

theta2_task1 = linspace(theta2_task1_initial, theta2_task1_final, length(time));

theta2_task2 = linspace(theta2_task2_initial, theta2_task2_final, length(time));

figure;

plot(time, theta2_task1, 'r-', 'LineWidth', 1.5);
```

```
hold on;
plot(time, theta2_task2, 'b--', 'LineWidth', 1.5);
hold off;
xlabel('Time (seconds)');
ylabel('\theta_2 (degrees)');
title('Joint 2 (Theta_2 vs Time)');
legend('Task 1', 'Task 2', 'Location', 'Best');
grid on;
set(gca, 'FontSize', 12);
time = linspace(0, 6, 100);
theta4_task1_initial = -264.3;
theta4_task1_final = -257;
theta4_task2_initial = -254.77;
theta4_task2_final = -120.68;
theta4_task1 = linspace(theta4_task1_initial, theta4_task1_final, length(time));
theta4_task2 = linspace(theta4_task2_initial, theta4_task2_final, length(time));
figure;
plot(time, theta4_task1, 'r-', 'LineWidth', 1.5);
hold on;
plot(time, theta4_task2, 'b--', 'LineWidth', 1.5);
xlabel('Time (seconds)');
ylabel('\theta_4 (degrees)');
title('Joint 4 (Theta_4 vs Time)');
legend('Task 1', 'Task 2', 'Location', 'Best');
grid on;
set(gca, 'FontSize', 12);
```

## 4.0 CONCLUSION

This report provides a thorough analysis and simulation of the 4 DOF ReBel robotic arm, shedding light on its strengths and weaknesses in executing complex pick-and-place tasks. By applying Denavit-Hartenberg parameters to derive the kinematic models and solving the inverse kinematics, we accurately identified the joint configurations necessary for effective task performance. Our trajectory planning and optimization efforts not only facilitated smooth and efficient movements but also reduced execution time while respecting joint and operational limits.

Dynamic simulations and 3D visualizations further confirmed the practicality of the proposed strategies, enhancing our understanding of the robot's performance across different scenarios. The workspace analysis identified the best placements for workpieces, ensuring that the target positions were accessible within the robot’s reach and without hindering the robot's functionality constraints or limitation. These results highlight the ReBel robotic arm's potential for various industrial applications, and the methods used provide a solid foundation for improving robotic system design and implementation.