SCARA Robot 3-DOF RPR For Cutting 2D Shapes

Khaled Omar*, Omar Montaser*, Kareem Waleed*,
Mohamed Eshall*, Bassem Ismail*, and Yasmina Tamer*
*German University in Cairo (GUC), Egypt
Emails: khaled.alkotob@student.guc.edu.eg, omar.hussien@student.guc.edu.eg,
kareem.alsabaag@student.guc.edu.eg, mohammad.eshall@student.guc.edu.eg,
Bassem.abohageel@student.guc.edu.eg, Yasmina.antar@student.guc.edu.eg

***Abstract*—This paper presents the development and implementation of a 3 Degree-of-Freedom (3DOF) SCARA robot for trajectory tracking and hardware realization. The system modeling was conducted in MATLAB and Simulink using the Simscape toolbox, enabling detailed simulation of the robot's dynamics. Forward position kinematics was derived using the Denavit-Hartenberg (DH) convention. Inverse position ,velocity and acceleration kinematics were also analyzed. The robot's capability to follow desired trajectories—such as half-circle, straight-line, heart, and circular paths—was demonstrated through Python-based control scripts and visualized in Coppeliasim EDU and also in MATLAB codes and visualized in Simscape toolbox. Furthermore, the hardware implementation was realized using stepper motors, a CNC shield, belts, and pulleys. The prismatic joint was constructed using a power screw mechanism coupled with a stepper motor, complementing the two revolute joints. This work provides a comprehensive framework for designing, modeling, simulating, and physically implementing a SCARA robot.**

***Keywords*-scara; Simscape; CoppeliaSim**

I. Introduction

Robotics has become a cornerstone of modern industry, facilitating precision, speed, and efficiency in tasks ranging from manufacturing to healthcare. Among various robotic configurations, the SCARA (Selective Compliance Assembly Robot Arm) robot stands out for its simplicity, accuracy, and versatility in pick-and-place operations, assembly lines, and other applications requiring planar motion. This paper focuses on the development, modeling, simulation, and hardware implementation of a 3 Degree-of-Freedom (3DOF) SCARA robot capable of following desired trajectories with high precision.

The motivation for this project stems from the increasing demand for cost-effective and adaptable robotic solutions in both industrial and educational domains. SCARA robots, known for their compact design and high repeatability, offer an excellent platform for exploring fundamental robotic concepts and their real-world applications. This project aims to bridge the gap between theoretical understanding and practical implementation, offering insights into the complete development cycle of a robotic system.

The primary objectives of this work are as follows:

- To design and model a 3-DOF SCARA robot in MATLAB and Simulink using the Simscape toolbox
- To derive and implement forward and inverse kinematics, as well as velocity and acceleration kinematics, using MATLAB
- To enable the robot to track complex trajectories, including a half-circle, straight line, heart shape, and circular paths, through Python-based control in Coppeliasim EDU and MATLAB code in Simscape
- To visualize the robot's motion in simulation environments such as Simscape and Coppeliasim
- To construct a physical prototype of the SCARA robot using stepper motors, CNC shields, belts,pulleys and power screw

The developed SCARA robot consists of two revolute joints and one prismatic joint, providing three degrees of freedom. The prismatic joint is realized using a power screw mechanism driven by a stepper motor, while the revolute joints are actuated through stepper motors connected via belts and pulleys. The modeling, simulation, and trajectory tracking tasks were carried out using both the Simscape toolbox in MATLAB/Simulink and Python scripts in Coppeliasim EDU, ensuring comprehensive analysis and compatibility between simulation and physical implementation. The system combines theoretical rigor with practical execution, demonstrating the robot's effectiveness in performing precise, smooth, and complex motions across various trajectories.

This paper outlines the complete design and implementation process, providing a detailed account of the challenges encountered and solutions devised during the project

II. Hardware Design and Implementation

In order to build the hardware of the system, certain components are required as well as the circuit diagram of the system is built.

Figure 1: components of the hardware

Figure 2: hardware

A. Hardware Components

The hardware implementation of the SCARA robot involved the integration of several key components, chosen to achieve the desired functionality and reliability. These components are detailed in Table 1.

Table I: Hardware Components Table

Mechanical parts	Electrical parts
3d printed 1245	nema 17 4 pieces
gripper "Cutter"	cables 4 pieces
Smooth rod 10mm 1m 2	limit swtch 4 pieces
Lead screw 0.5	power 10a 1 piece
Lead screw nut 1	power cable 1 piece
Linear bearings 10mm lm10uu 4	uno 1 piece
(51108)Thrust ball bearing 40x60x13mm 2	
(51107)Thrust ball bearing 35x52x12mm 2	uno shield 1 piece
(608) ball bearing 8x22x7mm 5	a4988 4 piece
6807 ball bearing 35*47 1	power plug 1 piece
6806 ball bearing 30*42 1	wires 10 m
Belt 200mm 1	servo mg995 1
Belt 300mm 2	
belt 400mm 2	
Coupler 5*8 1	
pully no teeth 7	
m3*16 5	
m3*12 20	
m3*20 10	
m3*25 5	
m3*30 10	
m4*15 10	
m4*25 8	
m4*20 40	
m4*45 10	
m4*40 10	
m8*45 2	

B. Circuit Diagram

Power supply 12V 6A, Connected to CNC shield attached on Arduino Uno along with gbrl driver for each stepper motor.

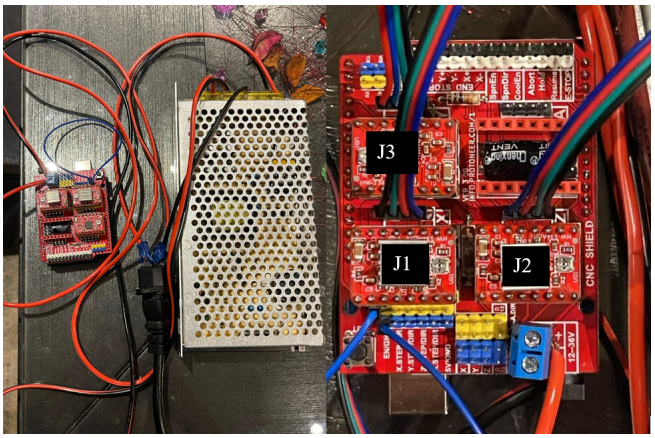

Figure 3: Circuit connections

III. Robot Kinematics

A. Robot's Frame Assignment

The assigned frames of the robot are shown in the figure 4 below.

Table II: DH-Parameters Table

theta	d	a	alpha
q1	l1	0	0
0	q2	l2	0
q3	-l4	l3	0

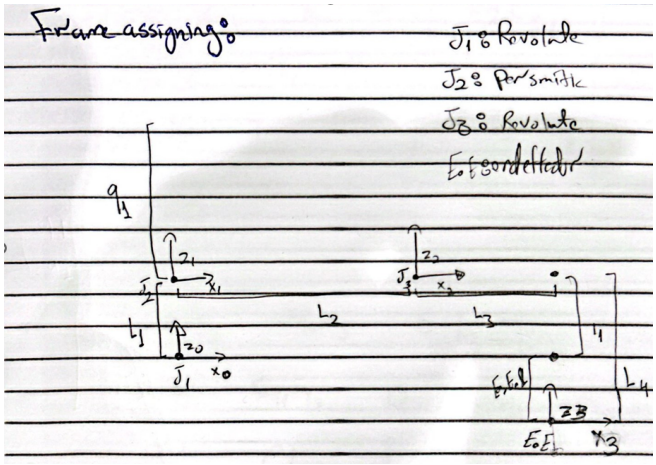

Figure 4: assigned frames

B. DH Convention

The DH convection of the robot is shown in the table. Table II

IV. Simulation Results

For simulating the robotic arm of both Simscape and CoppeliaSim.

For Simscape Figure 17.

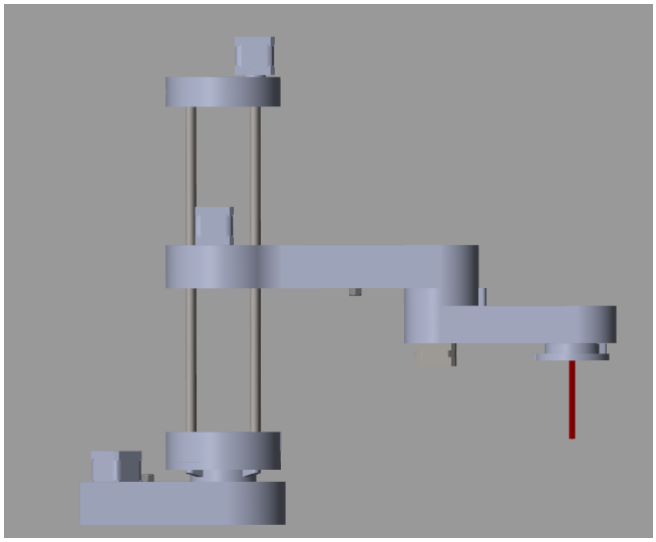

Figure 5: Scara in Simscape enviroment

For CoppeliaSim Figure 17.

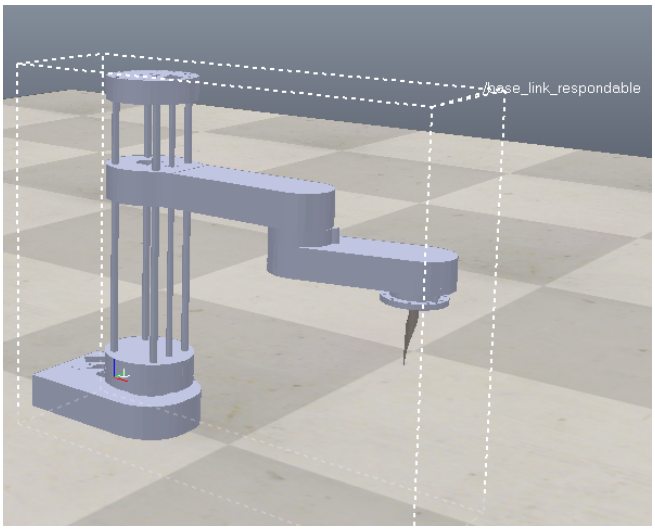

Figure 6: Scara in CoppeliaSim enviroment

A. Simscape Model

For developing the Simscape model, some modifications were made to the model in order to contribute to the end effector. Figure 17.

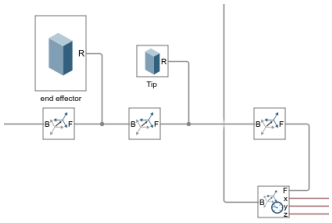

Figure 7: End effector in Simscape

And finally, add a function to control the prismatic joint using mm input instead of angles : :

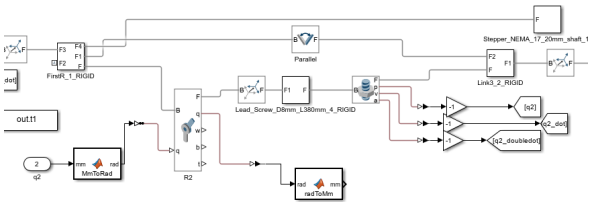

Figure 8: prismatic joint in Simscape

B. Kinematics Model

For the next stage, development of the kinematics model of the robotic arm itself and determining the parameters of the robotic arm.

```
L1 = 0.645;
L2 = 0.228;
```

```
L3 = 0.1365;
L4 = L1 + 0.1-0.031+0.025;%L4 contains L1 with the end effector.
```

Then we develop the forward position kinematics:

```
function X = forward_Position_kinematics(q1, q2, q3)
    % Link lengths
    L1 = 0.645;
    L2 = 0.228;
    L3 = 0.1365;
    L4 = L1 + 0.1 - 0.031 + 0.025;

    % Define the DH parameters for each joint
    DH_table = [q1, L1, 0, 0;      % Revolute joint 1
                0, q2, L2, 0;      % Prismatic joint 2
                q3, -L4, L3, 0];   % Revolute joint 3


    T_total = eye(4);

    for i = 1:size(DH_table, 1)
        theta = DH_table(i, 1);
        d = DH_table(i, 2);
        a = DH_table(i, 3);
        alpha = DH_table(i, 4);

        T_i = transformation_func(theta, d, a, alpha);
        T_total = T_total * T_i;
    end

    Px = T_total(1, 4) * 1000 + 79;  % X position in mm with offset
    Py = T_total(2, 4) * 1000;       % Y position in mm
    Pz = T_total(3, 4) * 1000;       % Z position in mm
    X = [Px; Py; Pz]; % Make sure X is a column vector for compatibility
end
```

Then developing the inverse position kinematics.

```
function q = inverse_kinematics_func(q0, X_desired)
    % Set tolerance and max iterations for the inverse kinematics solver
    tolerance = 1e-6;
    max_iterations = 100;
    q = q0; % Initial guess for joint angles
    for i = 1:max_iterations
        % Compute the current end effector position using forward kinematics
        X_current = forward_Position_kinematics(q(1), q(2), q(3)) ; % Assuming forward kinematics function for 3-joint robot
        error = X_desired' - X_current';
        error = error(:) ; % This should convert the error to a column vector if needed
        % Break if the error is below the specified tolerance
        if norm(error) < tolerance
            break;
        end
        J_inv = inverse_jacobian_matrix(q);  % Assuming inverse_jacobian_matrix returns a 3x3 matrix
         q = q + J_inv * error;
         q(1) = getin_scale(q(1)); % to be between 0 - 2pi
         q(3) = getin_scale(q(3));
    end
end
```

Then both forward and inverse velocity kinematics are added to be used for further analysis.

```
function X_dot = forward_velocity_kinematics(q, q_dot)
    % Obtain the Jacobian matrix for the current joint configuration
    J = numeric_jacobian_func(q(1), q(2), q(3));

    % Multiply the Jacobian matrix with q_dot to obtain end-effector velocity
    X_dot = J * q_dot;
end
function q_dot = inverse_velocity(q, X_dot)
    % Obtain the Jacobian matrix for the current joint configuration

    J_inv = inverse_jacobian_matrix(q);

    % Compute joint velocities (q_dot) by multiplying the Jacobian inverse with the desired end-effector velocity
    q_dot = J_inv * X_dot;
end
```

Finally reaching a full kinematics model:

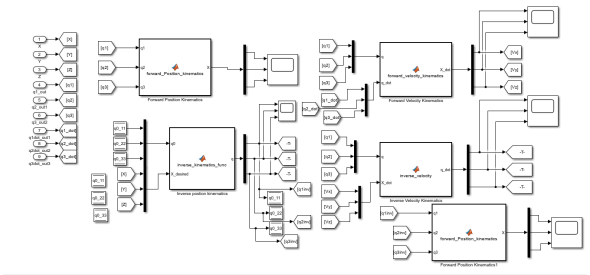

Figure 9: Kinematics in Simscape

Also important to highlight using of the output of the inverse kinematics block to be used again as the initial guess this helps to avoid the singularities and having a similar output but with far angles from the previous position causing failing of any trajectory generated.
All these implemented functions have a python version having the same functionality to be used in CoppeliaSim simulation and analysis.

Having Finally a full representative model ready for further analysis and simulation integrating the robotic arm model with kinematics model.

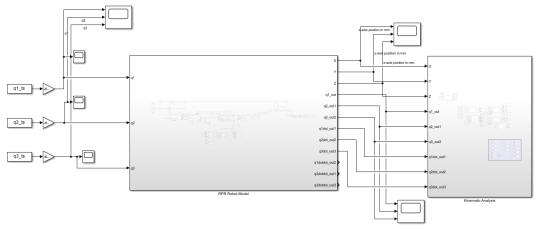

Figure 10: Final Model in Simscape

C. Trajectory generation

At this part 2 main trajectories where created in MATLAB in order to be passed to be passed to the model created previously.

1) Heart trajectory generation:

```
% Parameters
scale = 6; % Scale for a smaller heart (~50 mm wide)
center_x = 200; % Offset X position in mm
center_y = 200; % Offset Y position in mm
z_height = 0; % Cutting plane height in mm
down_height = 0; % Lower Z height in mm for cutting
initial_height = 40; % Initial Z height in mm
num_points = 100; % Number of points for a smooth heart

% Parametric equations for heart shape
t = linspace(0, 2*pi, num_points); % Parameter t
x_traj = scale * 16 * sin(t).^3 + center_x; % X coordinates
y_traj = scale * (13 * cos(t) - 5 * cos(2*t) - 2 * cos(3*t) - cos(4*t)) + center_y; % Y coordinates
z_traj = z_height * ones(1, num_points); % Z height remains constant for the heart

% Initial descent and final ascent points
descent = [center_x, center_y, initial_height;center_x, center_y, initial_height; center_x, center_y, down_height];
ascent = [center_x, center_y, down_height; center_x, center_y, initial_height];

% Combine the trajectory: descent -> heart -> ascent
trajectory = [descent; [x_traj', y_traj', z_traj']; ascent];

% Plot the trajectory in 3D space
figure;
plot3(trajectory(:, 1), trajectory(:, 2), trajectory(:, 3), 'r-', 'LineWidth', 2);
grid on;
xlabel('X (mm)');
ylabel('Y (mm)');
zlabel('Z (mm)');
title('Heart-Shaped Trajectory with Descent and Ascent');
axis equal;

% Initialize joint variables
q1_values = [];
q2_values = [];
q3_values = [];
q0 = [0.1; 0.3; 2]; % Initial guess for inverse kinematics
contiuscycle = false;

% Compute joint variables for the entire trajectory
for i = 1:size(trajectory, 1)
    % Extract desired end-effector position
    Px = trajectory(i, 1) / 1000; % Convert mm to m
    Py = trajectory(i, 2) / 1000; % Convert mm to m
    Pz = trajectory(i, 3) / 1000; % Convert mm to m

    % Inverse kinematics for RPR SCARA robot
    qq = inverse_kinematics_func(q0, [Px * 1000; Py * 1000; Pz * 1000]);
    if contiuscycle
        qtemp = q0;
        while (abs(qq(1) - qtemp(1)) > 0.5 || abs(qq(3) - qtemp(3)) > 0.5)
            q0 = [2 * pi * rand; 0.3; 2 * pi * rand];
            qq = inverse_kinematics_func(q0, [Px * 1000; Py * 1000; Pz * 1000]);
        end
    end

    % Store the joint values
    q1_values = [q1_values; qq(1)];
    q2_values = [q2_values; qq(2)];
    q3_values = [q3_values; qq(3)];

    q0 = qq; % Update the initial guess
    contiuscycle = true;
end

% Define time vector
time = linspace(0, 20, size(trajectory, 1))'; % Simulate for 20 seconds

% Create timeseries for joint variables
q1_ts = timeseries(q1_values, time);
q2_ts = timeseries(q2_values, time);
q3_ts = timeseries(q3_values, time);

% Save to .mat file for Simscape use
save('heart_trajectory_with_z_motion.mat', 'q1_ts', 'q2_ts', 'q3_ts');
```

Trajectory started by mathematically generate the heart it self and visualizing it and having an descent and ascent parts at the beginning and at the end to accommodate for going up and down to cut and after finishing.

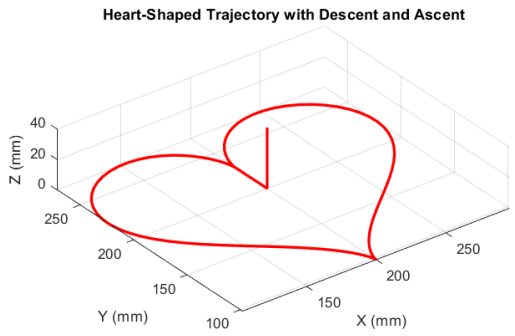

Figure 11: Heart trajectory in MATLAB

Note also a while loop where added to the inverse kinematics loop in order to limit and make sure that output coming is near to the one generated previously to have a continuous trajectory.

For validating the trajectory generated the output time serious where passed to the simscape model created previously having results on scope as follows in 20 seconds simulation.

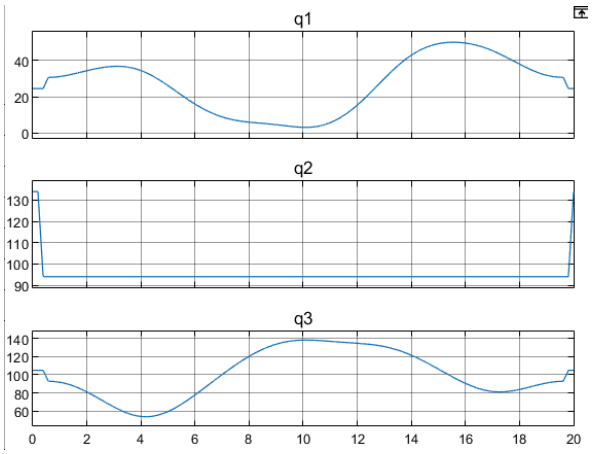

Figure 12: Heart trajectory in Simscape

Having continuous generated angles means that the trajectory could be performed on hardware and simulated as well.

Also for the position of the end effector it can be seen it follows the trajectory created exactly, perfomring the trajectory successfully.

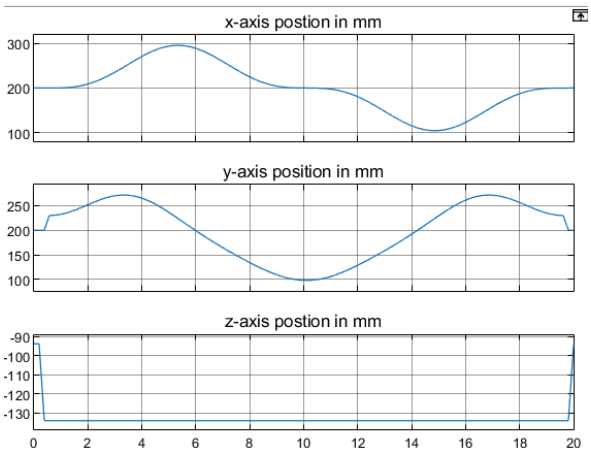

Figure 13: Heart trajectory in Simscape

2) Circle Trajectory:

```
% Parameters
scale = 50; % Radius of the circle
center_x = 250; % Offset X position in mm
center_y = 0; % Offset Y position in mm
z_height = 0; % Cutting plane height in mm
down_height = 0; % Lower Z height in mm for
    cutting
initial_height = 40; % Initial Z height in mm
num_points = 100; % Number of points for a smooth
    circle

% Parametric equations for circle
angle = linspace(0, 2*pi, num_points); %
    Parameter t
x_traj = scale * cos(angle) + center_x; % X
    coordinates
y_traj = scale * sin(angle) + center_y; % Y
    coordinates
z_traj = z_height * ones(1, num_points); % Z
    height remains constant for the circle

% Initial descent and final ascent points
temp = initial_height;
descent = [];
ascent = [];
while(temp>down_height)
    tt= [center_x, center_y, temp] ;
    descent = [descent; tt];
    temp = temp -5;
end
while(temp<initial_height)
   tt= [center_x, center_y, temp] ;
    ascent = [ascent; tt];
    temp = temp +5;
end
%descent = [center_x, center_y, initial_height;
    center_x, center_y, down_height];
%ascent = [center_x, center_y, down_height;
    center_x, center_y, initial_height];

% Combine the trajectory: descent -> circle ->
    ascent
trajectory = [descent; [x_traj', y_traj',
    z_traj']; ascent];

% Plot the trajectory in 3D space
figure;
plot3(trajectory(:, 1), trajectory(:, 2),
    trajectory(:, 3), 'b-', 'LineWidth', 2);
```

```
grid on;
xlabel('X (mm)');
ylabel('Y (mm)');
zlabel('Z (mm)');
title('Circular Trajectory with Descent and Ascent');
axis equal;

% Initialize joint variables
q1_values = [];
q2_values = [];
q3_values = [];
q0 = [0.1; 0.3; 2]; % Initial guess for inverse kinematics
contiuscycle = false;

% Compute joint variables for the entire trajectory
for i = 1:size(trajectory, 1)
    % Extract desired end-effector position
    Px = trajectory(i, 1) / 1000; % Convert mm to m
    Py = trajectory(i, 2) / 1000; % Convert mm to m
    Pz = trajectory(i, 3) / 1000; % Convert mm to m

    % Inverse kinematics for RPR SCARA robot
    qq = inverse_kinematics_func(q0, [Px * 1000; Py * 1000; Pz * 1000]);
    if contiuscycle
        qtemp = q0;
        while (abs(qq(1) - qtemp(1)) > 0.3 || abs(qq(3) - qtemp(3)) > 0.4)
            q0 = [2 * pi * rand; 0.3; 2 * pi * rand];
            qq = inverse_kinematics_func(q0, [Px * 1000; Py * 1000; Pz * 1000]);
        end
    end

    % Store the joint values
    q1_values = [q1_values; qq(1)];
    q2_values = [q2_values; qq(2)];
    q3_values = [q3_values; qq(3)];

    q0 = qq; % Update the initial guess
    contiuscycle = true;
end

% Define time vector
time = linspace(0, 20, size(trajectory, 1))'; % Simulate for 20 seconds

% Create timeseries for joint variables
q1_ts = timeseries(q1_values, time);
q2_ts = timeseries(q2_values, time);
q3_ts = timeseries(q3_values, time);

% Save to .mat file for Simscape use
save('circle_trajectory_with_z_motion.mat', 'q1_ts', 'q2_ts', 'q3_ts');
```

Same as the previously generated trajectory it can be visualized and analyzed the same way.

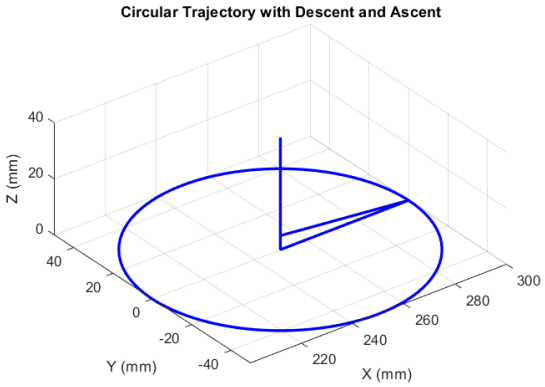

Figure 14: Circle trajectory in MATLAB

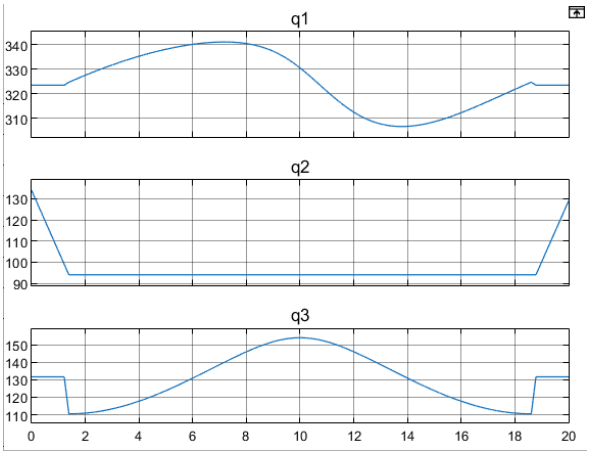

Figure 15: Circle trajectory in Simscape

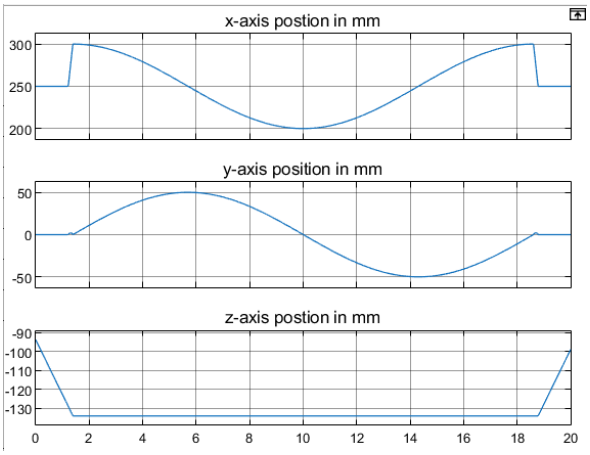

Figure 16: Circle trajectory in Simscape

Having continuous angle trajectory along with position trajectory exactly same as the planned means that trajectory is performed successfully.

V. Conclusions and Future Recommendations

In conclusion, this project successfully developed and implemented a 3-DOF SCARA robot capable of precise trajectory tracking. The integration of modeling, kinematics, and simulation using MATLAB, Simscape, and Coppeliasim provided a robust framework for analyzing and visualizing the robot's motion. The hardware implementation validated the simulation results, demonstrating the system's feasibility and accuracy in real-world scenarios. The results highlight the robot's capability to execute complex trajectories with smooth and reliable motion.

Future work will focus on enhancing the system's capabilities by integrating advanced control algorithms and sensor feedback to improve precision and adaptability. Additionally, further development will include automating calibration processes and incorporating end-effectors for specific tasks, such as assembly or cutting operations. These steps aim to evolve the SCARA robot into a fully functional robotic system, paving the way for advanced applications in industrial and educational environments.

Figure 17: SCARA robotic arm coupling with pick and place robotic arm

VI. References

https://mostelectronic.com/shop/robotic-arms/scara-robot-4-dof-robotic-arm-kit-copy/