ANKARA YILDIRIM BEYAZIT UNIVERSITY

Department of Aeronautical and Aerospace Engineering

AEE486 - Fundamentals of Robotics

Project Title: Tracking Control of 2-DOF Manipulator

Instructor: Prof. Dr. Arif ANKARALI

Group Members:

Batuhan SAHIN - 20110151003

Beray ÖZCAN 21110111026

Muhammed Berati KILIÇ - 20110111028

Oguzhan ÇAMLICA - 20110111021

May 2025

Contents

1 Abstract

1.1 Brief Summary of the Project

This project describes the design, modelling, simulation, manufacturing and implementation of a robot arm with two degrees of freedom. The robot arm project consists of 2 rotary joints and is intended to draw a rectangle by means of a pen attached to the end effector. The motion of the robot was modelled by kinematic calculations and mathematical expressions were created using Denavit-Hartenberg (D-H) parameters. In the project, the design of the robot arm was done through SolidWorks software, modelling was done through Matlab-Simulink and production was done through 3-D printer.

1.2 Main Objectives and Outcomes

The main aim of the project is to be able to make kinematic calculations of n degree of freedom systems in robotic systems and to be able to apply this in reality. In this project, a robot arm with 2 degrees of freedom was designed and modelled and implemented in reality with Arduino. The aim is to calculate the error rate by comparing the simulation results obtained in Matlab environment with the actual project outputs. In this context, the steps are as follows:

- To design a robot arm with 2 rotary joints
- To make kinematic calculations
- To establish a mathematical model with Denavit-Hartenberg parameters
- To perform simulations of the manipulator in Matlab environment
- To produce the robot arm and implement it with Arduino
- To compare the outputs

2 Introduction

2.1 Definition of Industrial Robots and Importance

According to the definition determined by ISO 8373 standards, industrial robot is defined as an automatically controlled, multi-purpose manipulator with three or more programmable axes that can be used in industrial areas such as assembly, transport, production. Industrial robots have accelerated many jobs in industry, reduced labour costs and increased production. [1]
Robotic systems are systems that work with the combination of many different engineering fields. Kinematic analyses are used to calculate the position and orientation of the robot; dynamic analyses examine the force and moment relationships that occur during the movement of the robot. Trajectory planning is expressed as the path followed by the end part of the robot called end effector throughout the operation of the robot. Control ensures that the robot performs its movements correctly.

3 System Design

3.1 Mechanical Structure

The project consists of 2 rotary joints. Therefore, it is referred to as 2 degrees of freedom. The degree of freedom is the number of axes that can move independently. The robot arm project consists of 2 arms called links. The links are connected to each other by joints that provide movement called joints.
The first joint of the system is positioned at the base point and the second joint is mounted at the end of the first link. Each arm part (2 pieces) is produced from 3D Printer and connected to each other.

- First Link Length:15 cm
- Second Link Length:10 cm

Figure 1 and 2 shows the CAD model of the robotic arm created using SolidWorks software.

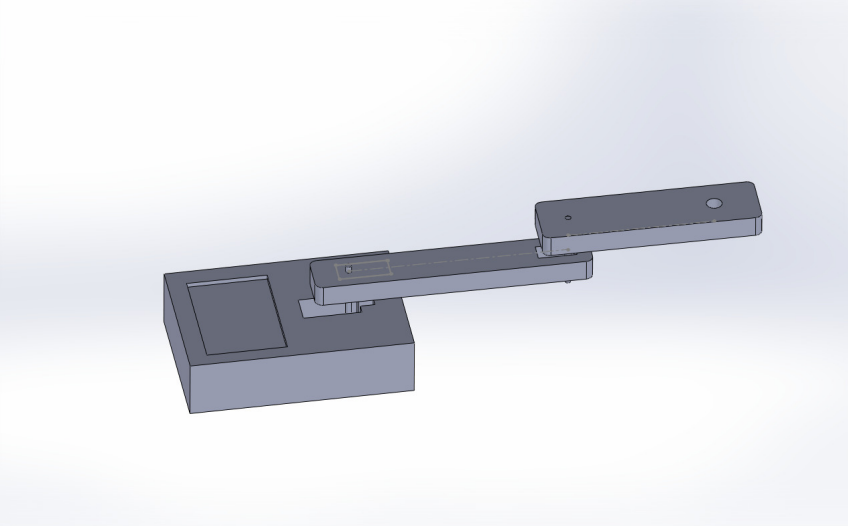

Figure 1: Robot Arm CAD view

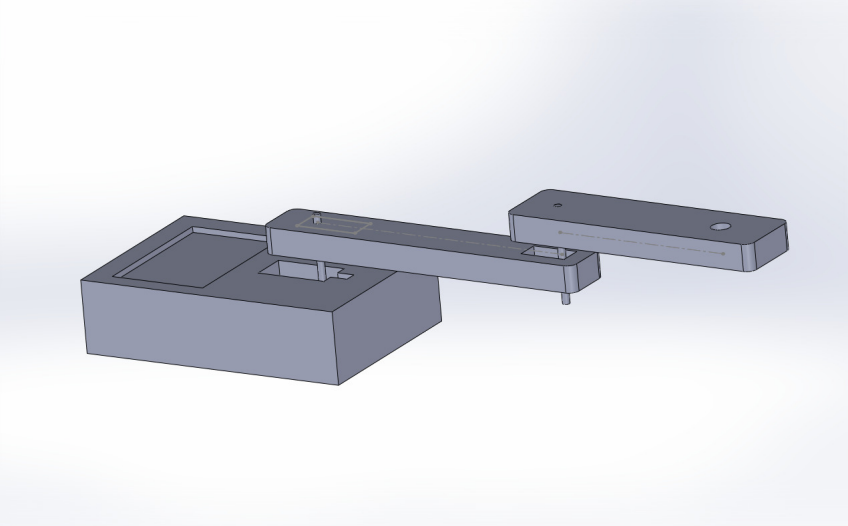

Figure 2: Robot Arm CAD view

3.2 Electronic Components

In this project, an Arduino Uno was selected as the microcontroller. Two servo motors were used to actuate the two joints of the robotic arm. For Joint 1, the DSSERVO DS3225 servo motor was chosen, while the Kingmax KM1703MD servo motor was selected for Joint 2. The specifications of both servo motors are presented in Table 1.

Additionally, the torque requirements for each joint were calculated based on the dynamic torque analysis explained in the following sections.

Table 1: Servo Motor Specifications

Specification	**Kingmax CLS2875H**	**DS3225**
Operating Voltage	4.8 – 6.0 V	4.8 - 6.8 V
Stall Torque (5V)	6.1 kg·cm	21 kg ·cm
Stall Torque (6V)	7.6 kg·cm	25 kg ·cm
Speed (5V)	0.11 s / 60 degree	0.16 s / 60 degree
Speed (6V)	0.09 s / 60 degree	0.14 s / 60 degree
Gear Type	Metal	Metal
Weight	28 g	67 g
Dimensions (mm)	35 * 15 * 29	40.5 * 20 * 40

Torque Requirement Analysis

The aim of this section is to compute the maximum torque requirements for each joint (servo) in a two-link planar robotic arm in order to appropriately size and select the servo motors. The torques are calculated based on predefined trajectories using a dynamic model that includes inertia and coriolis effects. [4]

Formulation

The joint torques required to drive the two-link planar robotic arm are computed using the standard dynamic model of a serial manipulator. [3] The general form of the dynamic equation is:

$$\tau = M(\theta)\ddot{\theta} + C(\theta, \dot{\theta})\dot{\theta} + G(\theta) \tag{1}$$

where:

- $\tau = [\tau_1, \tau_2]^T$: Joint torques [Nm]
- $\theta = [\theta_1, \theta_2]^T$: Joint angles [rad]
- $\dot{\theta} = [\dot{\theta}_1, \dot{\theta}_2]^T$: Joint angular velocities [rad/s]
- $\ddot{\theta} = [\ddot{\theta}_1, \ddot{\theta}_2]^T$: Joint angular accelerations [rad/s^2]

Inertia Matrix $M(\theta)$

$$M(\theta) = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \tag{2}$$

$$
\begin{aligned}
m_{11} &= I_1 + I_2 + m_1 L_{c1}^2 + m_2(L_1^2 + L_{c2}^2 + 2L_1 L_{c2}\cos(\theta_2)) \\
m_{12} &= m_{21} = I_2 + m_2(L_{c2}^2 + L_1 L_{c2}\cos(\theta_2)) \\
m_{22} &= I_2 + m_2 L_{c2}^2
\end{aligned}
$$

Coriolis and Centrifugal Matrix $C(\theta, \dot{\theta})$

$$
C(\theta, \dot{\theta}) = \begin{bmatrix} -h\dot{\theta}_2 & -h(\dot{\theta}_1 + \dot{\theta}_2) \\ h\dot{\theta}_1 & 0 \end{bmatrix} \tag{3}
$$

where:

$$
h = m_2 L_1 L_{c2} \sin(\theta_2)
$$

Gravity Vector $G(\theta)$

$$
G(\theta) = \begin{bmatrix} (m_1 L_{c1} + m_2 L_1) g \cos(\theta_1) + m_2 L_{c2} g \cos(\theta_1 + \theta_2) \\ m_2 L_{c2} g \cos(\theta_1 + \theta_2) \end{bmatrix} \tag{4}
$$

Given that the system operates strictly in the XY plane, the gravitational force acting along the Z-axis does not influence the dynamics.

Peak Torque Calculation For a given trajectory (joint angles, velocities, and accelerations as functions of time), the instantaneous torques $\tau_1(t)$ and $\tau_2(t)$ are computed at each time step using the dynamic equation. The peak torque requirements are then identified as:

$$
\begin{aligned}
\tau_{1,\text{peak}} &= \max_t |\tau_1(t)| \\
\tau_{2,\text{peak}} &= \max_t |\tau_2(t)|
\end{aligned}
$$

These peak torques are critical in selecting appropriately sized servo motors for each joint to ensure reliable performance under maximum load conditions. The function used in this section is shown in the Appendix 9.4. The torque results obtained from the system model closely match the experimental results, as illustrated in Figure 3. At the end of the simulation, the maximum torque was approximately 0.57 kg·cm for Servo-1 and 0.53 kg·cm for Servo-2. The selected servo motors are sufficient to handle these torque requirements.

Parameters Used The following constant values are used for the robot arm during the simulation:

Parameter	Value	Description
L_1	0.15 m	Length of Link-1
L_2	0.10 m	Length of Link-2
L_{c1}	0.075 m	Center of mass of Link-1 Assembly
L_{c2}	0.03 m	Center of mass of Link-2 Assembly
m_1	0.086 kg	Mass of Link-1 Assembly
m_2	0.106 kg	Mass of Link-2 Assembly
I_{1zz}	0.00114957 kg·m^2	Inertia of Link-1 Assembly
I_{2zz}	0.0066028664 kg·m^2	Inertia of Link-2 Assembly
g	9.81 m/s^2	Gravitational acceleration

Table 2: Physical constants used in torque estimation

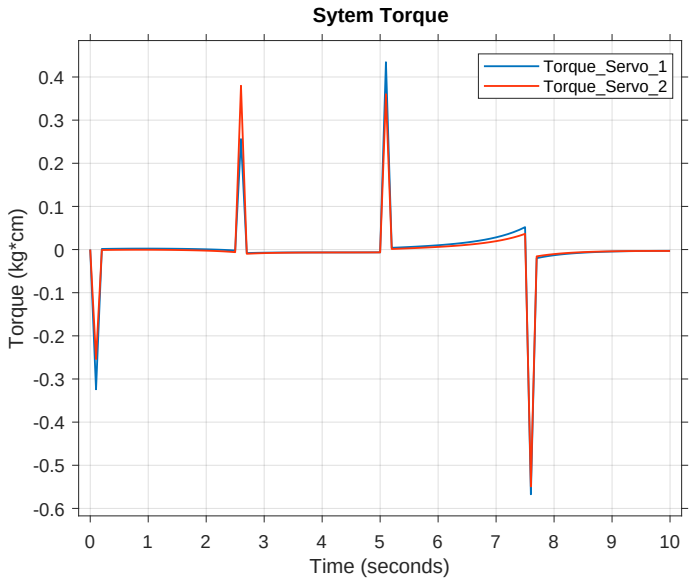

Figure 3: Sytem Torque for two servo motors.

As shown in Figure 3, the torque peaks occur at the corner regions of the trajectory, which is expected due to the abrupt changes in motion direction.

3.3 Overall Architecture

After the mechanical parts were either 3D-printed or procured, the integration of both mechanical and electronic components was carried out. The electronic units, consisting of the Arduino Uno microcontroller and servo motors, were integrated according to the wiring diagram shown in Figure 4.

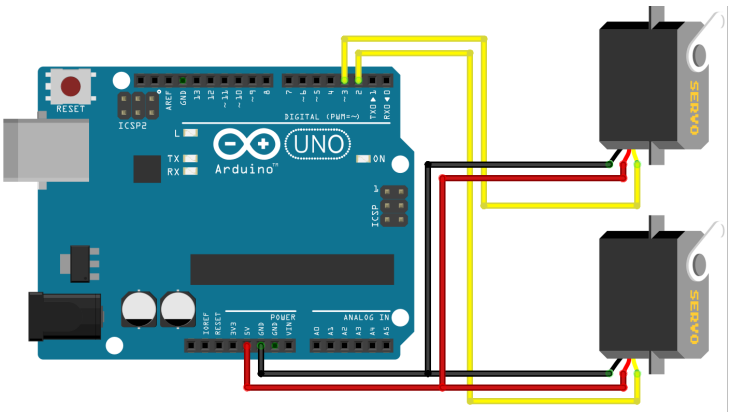

Figure 4: Electronic System Integration

4 Trajectory Planning

4.1 Trajectory Shape

The end effector of the robotic arm follows a square shaped closed trajectory in the counterclockwise direction. The square has a side length of 0.1 meters and is centered around the initial position. Each edge is traversed at a constant speed and within equal time intervals, resulting in a periodic motion. Moreover, the system can be adapted to draw various other shapes by modifying the trajectory code; even letters or custom paths can be rendered if desired. Main square trajectory code and Several code examples for circle, triangle and "AYBU" letter sequence are given in Appendix 9.1. In Figure 5, the square-shape trajectory is shown.

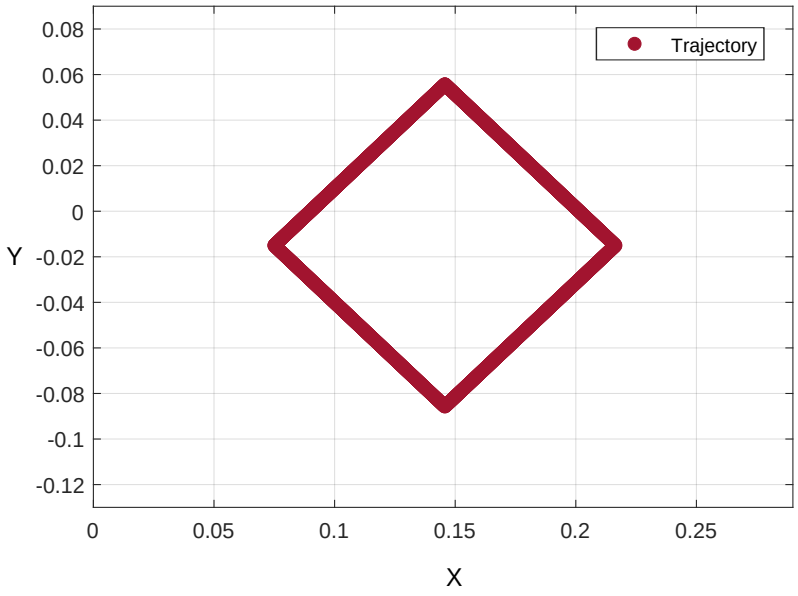

Figure 5: Square Shaped Trajectory

4.2 Mathematical Representation

The trajectory function generates a square path for the robotic arm's end-effector. After the initial transition phase, the end-effector begins tracing a square with a side length of $L = 0.1$ m, starting from the bottom-left corner ($x_0 = 0.175$, $y_0 = -0.05$). The total duration of one complete loop can be adjusted depending on the desired speed or precision of the motion.

The square is drawn in a counterclockwise direction, with the following piecewise-defined segments:

- **Bottom to top (left edge)**: $x = x_0, \quad y = y_0 + L \cdot \frac{t}{T/4}$
- **Left to right (top edge)**: $x = x_0 + L \cdot \frac{t-T/4}{T/4}, \quad y = y_0 + L$
- **Top to bottom (right edge)**: $x = x_0 + L, \quad y = y_0 + L - L \cdot \frac{t-T/2}{T/4}$
- **Right to left (bottom edge)**: $x = x_0 + L - L \cdot \frac{t-3T/4}{T/4}, \quad y = y_0$

Each segment is followed with constant velocity, resulting in a smooth and periodic square trajectory.

5 Robot Arm Kinematics

5.1 Joint Variables

The project includes two rotary joints. Therefore, there are two joint variables in the system. These variables are denoted as θ_1 and θ_2. These angles are used to determine the position of the end-effector of the robot.

- θ_1: It is the rotation angle of the first joint. This angle defines the orientation of the robot arm relative to the base.
- θ_2: It is the rotation angle of the second joint. This angle represents the orientation of the second arm, which is attached to the end of the first link.

The values of these angles are calculated using inverse kinematic analysis. In order for the robot to reach a desired point, the corresponding joint angles must be computed. The Figure 6 illustrates the axis assignment of the robotic arm based on each joint. This configuration is based on the DenavitHartenberg parameters and shows how coordinate frames are positioned for each link.

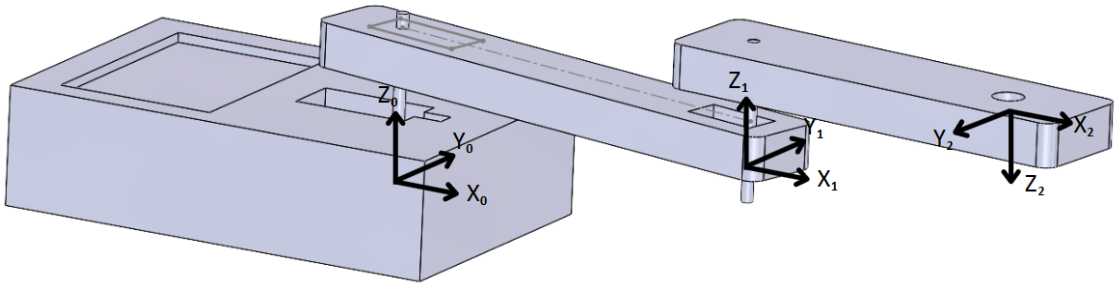

Figure 6: Axis Placement at Each Joint of the Robotic Arm

5.2 Derivation of Transformation Matrices Between Adjacent Joints

Homogeneous transformation matrix is a 4*4 matrix that calculates the position and orientation of an object in space. This matrix consists of rotation matrix, position vector, perspective transformation, scaling components. [2]

$$T = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{p}_{3\times1} \\ \mathbf{f}_{1\times3} & 1 \end{bmatrix} = \begin{bmatrix} \text{rotation matrix} & \text{position vector} \\ \text{perspective transformation} & \text{scaling} \end{bmatrix}$$

The denavit-hartenberg method uses 4 parameters to define the position and orientation between two consecutive links. d, theta, a and alpha parameters.

- θ_i: Joint angle between the x_{i-1} and x_i axes, measured about the z_{i-1} axis (using the right-hand rule).
- d_i: Link offset; the distance from the origin of the $(i-1)$ frame to the intersection of the z_{i-1} axis with the x_i axis, measured along the z_{i-1} axis.
- a_i: Link length; the distance from the intersection of the z_{i-1} axis with the x_i axis to the origin of the ith frame, measured along the x_i axis.
- α_i: Twist angle between the z_{i-1} and z_i axes, measured about the x_i axis (using the right-hand rule). [2]

$$T_z(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_z(\theta_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_x(a_i) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R_x(\alpha_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The relationship between two consecutive links is determined by multiplying these matrices.

$$^{i-1}A_i = T_z(d_i) \cdot R_z(\theta_i) \cdot T_x(a_i) \cdot R_x(\alpha_i)$$

$$
{}^{i-1}A_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i \sin\theta_i & \sin\alpha_i \sin\theta_i & a_i \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cos\theta_i & -\sin\alpha_i \cos\theta_i & a_i \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

5.3 Denavit-Hartenberg Parameters and Coordinate Frames

As previously introduced, the Denavit - Hartenberg (DH) parameters are used to model the manipulators kinematics. In this section, the DH table is constructed based on the assigned coordinate frames at each joint.

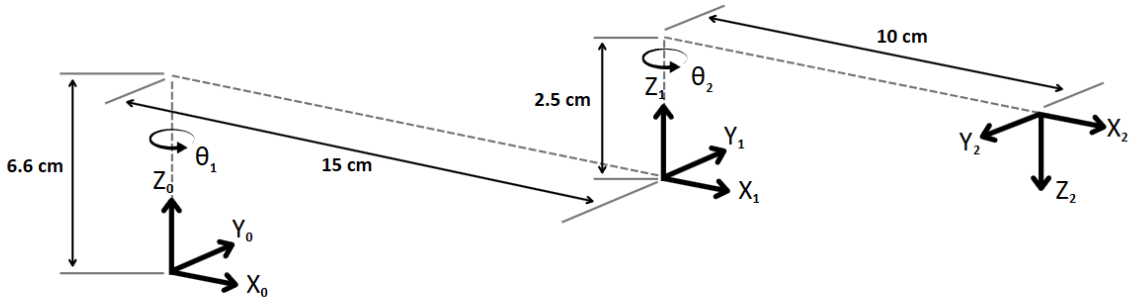

Figure 7: Coordinate Frame Assignment for Each Link Based on the DH Parameters

Figure 7 illustrates the spatial placement of the coordinate frames based on the listed parameters. The axes x_i, y_i, and z_i are assigned to each link according to the DH parameterization rules, capturing both the rotational and translational relationships between the links. This visual representation ensures that each transformation matrix aligns with the physical configuration of the robotic arm.

Joint	θ_i	α_i	a_i	d_i
1	θ_1	0°	15 cm	6.6 cm
2	θ_2	180°	10 cm	2.5 cm

Table 3: Denavit-Hartenberg Parameters

Table 5.3 lists the DH parameters corresponding to each joint of the manipulator, including the joint angle θ_i, link twist α_i, link length a_i, and link offset d_i. These parameters define the relative displacement and orientation between adjacent coordinate frames.

5.4 Calculation of Transformation Matrices

Using the Denavit - Hartenberg method, the ${}^0A_1$ matrix between the base and the first joint, and the ${}^1A_2$ matrix between the first and second joints were calculated through the transformation matrix formula between two links.

By multiplying these two matrices, the position and orientation of the end effector were determined with respect to the reference coordinate frame.

$$
{}^0A_1 = \begin{bmatrix} 1 & 0 & 0 & 0.15 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.066 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^1A_2 = \begin{bmatrix} 1 & 0 & 0 & 0.10 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0.025 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
T = {}^0A_1 \cdot {}^1A_2 = \begin{bmatrix} 1 & 0 & 0 & 0.25 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0.091 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

5.5 Forward and Inverse Kinematics

Figure 8a shows the top view (from the z-axis) of a 2-DOF planar robotic arm. In the figure, l_1 and l_2 represent the lengths of the first and second links, respectively. The angles θ_1 and θ_2 are the joint variables, indicating the rotational positions at each joint. The point (x, y) corresponds to the position of the end effector in the Cartesian x-y plane.

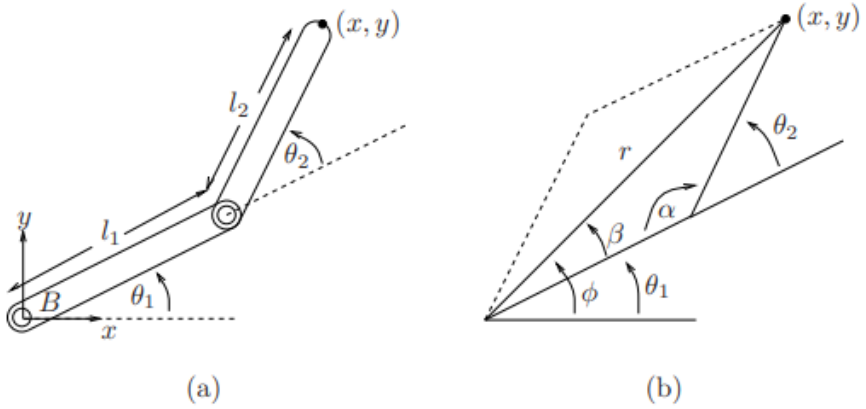

Figure 8: 2-DoF Planar Robotic Arm [5]

Firstly to obtain forward kinematic equations, the position of end effector can be represented [5]:

$$
x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)
$$

$$
y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)
$$

These two equations represent the forward kinematics of the planar manipulator and MATLAB function used in project given in Appendix 9.3 . Given the joint angles θ_1 and θ_2, the position of the end effector in the Cartesian plane can be precisely determined. In contrast, inverse kinematics refers to the process of calculating the required joint angles θ_1 and θ_2 for a desired end-effector position (x, y). Unlike forward kinematics, the inverse problem may have multiple solutions or, in some cases, no feasible solution depending on the reachability and constraints of the robotic arm.

Figure 8b shows a geometric construction used to solve the inverse kinematics of a 2-DOF planar robotic arm. A common approach is to express the problem in polar coordinates (r, ϕ), as illustrated in Figure Xb. Here, the distance r from the base to the end effector is computed as $r = \sqrt{x^2 + y^2}$, and the angle θ_2 can be obtained using the law of cosines [5]:

$$\theta_2 = \pi \pm \alpha, \quad \alpha = \cos^{-1}\left(\frac{l_1^2 + l_2^2 - r^2}{2l_1 l_2}\right)$$

Once θ_2 is found, the value of θ_1 can be computed using the angle $\phi = \tan^{-1}(y/x)$ and another auxiliary angle β [5]:

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) \pm \beta, \quad \beta = \cos^{-1}\left(\frac{r^2 + l_1^2 - l_2^2}{2rl_1}\right)$$

The sign of β must be chosen consistently with that of α, to ensure the selected configuration corresponds to the intended posture of the manipulator.

Inverse kinematics problems often yield multiple solutions due to the manipulator's geometry, and each applicable joint configuration must be separately evaluated. Inverse kinematic MATLAB function given in Appendix 9.2

6 Hardware Implementation

6.1 System Setup

The Figure 9 presents the block diagram of the robotic arm. The trajectory is generated as a function of time, converted into joint angles through inverse kinematics, and then actuated via servo motors. Using the forward kinematics function block, the output trajectory is visualized.

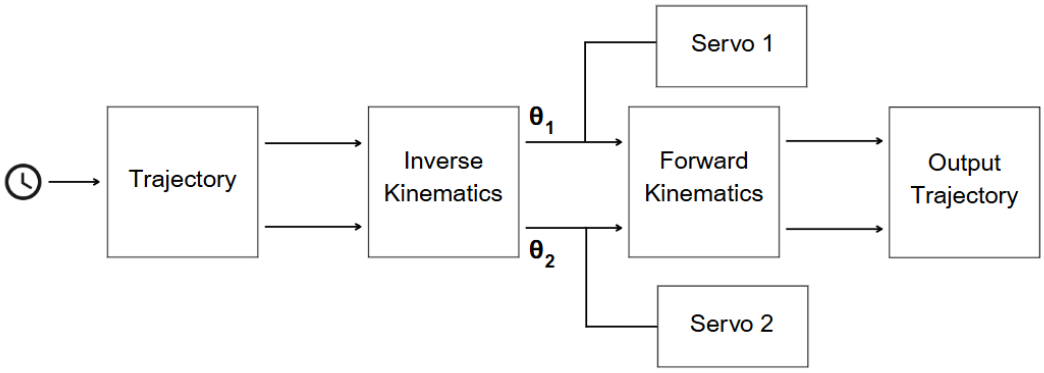

Figure 9: System Block Diagram

7 Experimental Results

Distribution of positions in the XY plane derived solely from the forward kinematics functions output angles, without accounting for the servos real-time responses shown in Figure 9.3. As can be seen, the computed point cloud largely preserves the defined square shape, exhibiting consistency along both corner transitions and edge segments. The end effectors dynamic behaviors and any potential deviations will be addressed in the following section, with the corresponding data presented in a separate graph.

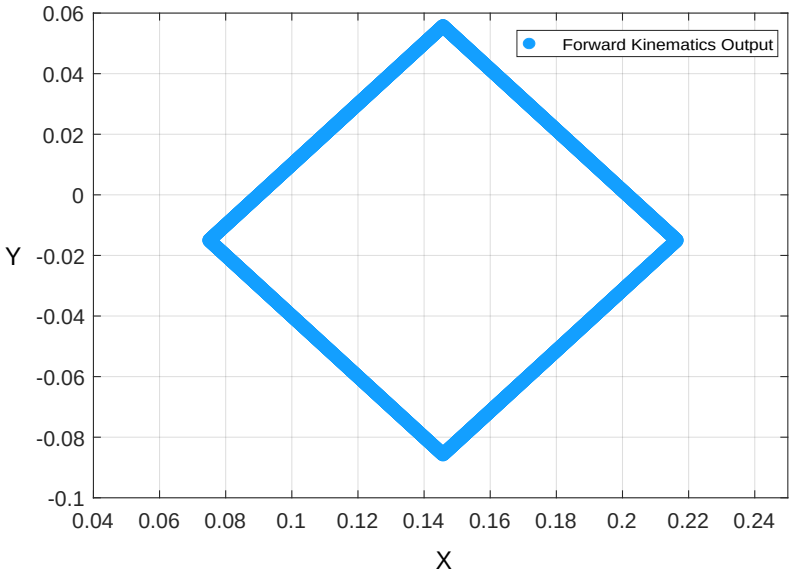

Figure 10: Output of Forward Kinematics

Figure 11 illustrates the actual path traced by the end effector on the work surface in the XY plane, represented as discrete recorded points and lines between them. As shown, the executed trajectory deviates from the ideal square; there are noticeable misalignments at the corners and irregular connections between successive points. Slight overshoots are observed along the edges. In the following section, these deviations will be analyzed and discussed in detail.

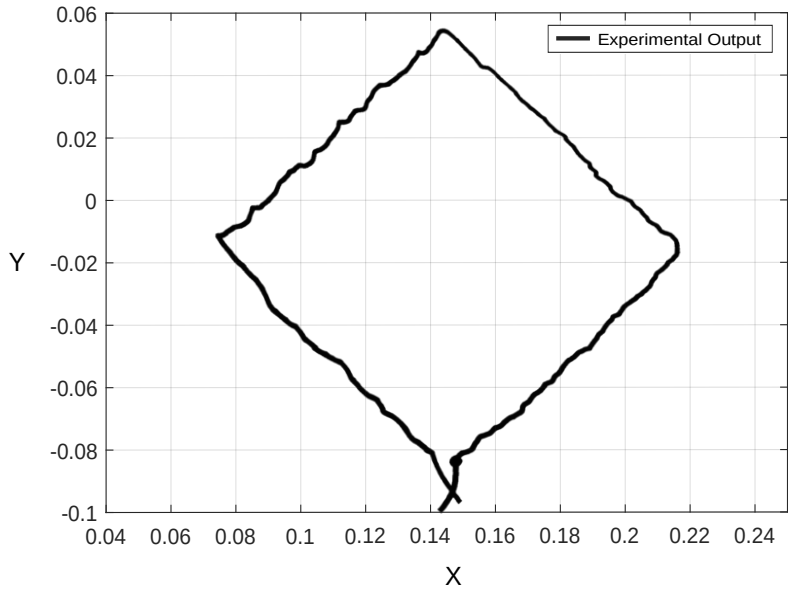

Figure 11: Experimental Output

7.1 Accuracy and Performance

Traces of the ideal (numerical) square trajectory and the actual (experimental) path executed step-by-step by the end effector illustrated in Figure 12. The blue line denotes perfect adherence to the planned trajectory, while the black points and lines reveal pronounced deviations at corner and edge segments and irregularities between successive connection points. These discrepancies stem from the servo motors limited angular resolution, geometric imperfections caused by backlash in the drive mechanisms and assembly tolerances, and the combined influence of other error sources such as controlsignal quantization, timing inconsistencies, and structural compliance. The results highlight the need for a quantitative assessment of the differences between the ideal model and actual hardware performance, followed by the implementation of suitable calibration and corrective measures.

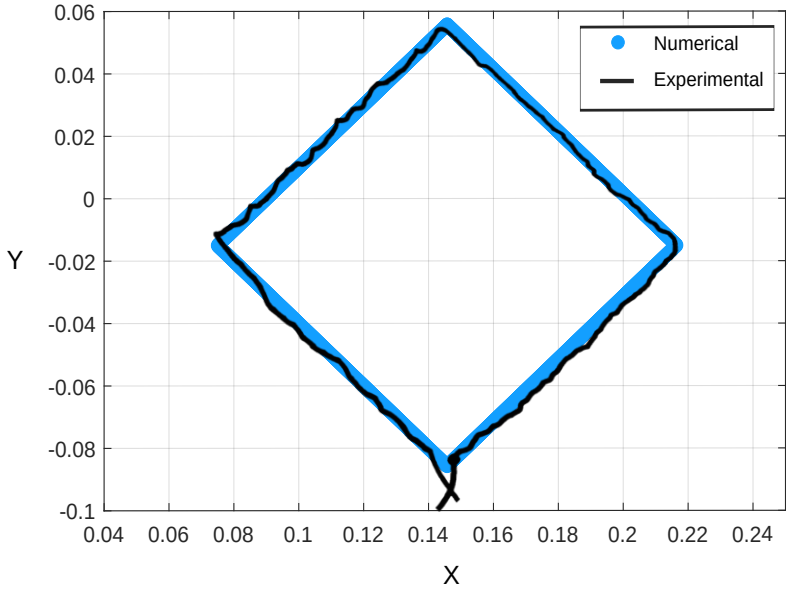

Figure 12: Comparison of Outputs

8 Conclusion and Future Work

8.1 Summary of Achievements

In this project, a 2-DOF robotic arm was successfully designed and implemented. The system is capable of following predefined trajectories and can draw simple shape such as square using a pen mounted on the end-effector. The kinematic model was developed using DenavitHartenberg parameters, and the motion of the arm is controlled based on a time-dependent trajectory function.

8.2 Improvements and Next Steps

In the future, the system can be enhanced with a feedback control mechanism to improve positional accuracy. By upgrading to a 3-DOF design, the arm would be able to lift the pen and move to different positions without drawing, allowing for cleaner and more flexible

transitions between shapes. Using higher-precision servo motors or stepper motors at the joints could also improve movement accuracy and repeatability. Additionally, integrating a spring-damper mechanism at the pen mount could help reduce vibrations during motion, improving line quality. Basic path optimization techniques, such as smoothing corners or using splines, may also be implemented to enhance the overall motion efficiency and precision.

References

[1] G. Gürgüze and . Türkolu, Kullanm Alanlarna Göre Robot Sistemlerinin Snflandrlmas, *Frat Üniversitesi Mühendislik Bilimleri Dergisi*, vol. 31, no. 1, pp. 53–66, 2019.

[2] R. C. Gonzalez, *Robotics: Control, Sensing, Vision, and Intelligence*, New York, NY: McGraw-Hill, 1989.

[3] Spong, M. W., Hutchinson, S., & Vidyasagar, M. (2006). *Robot Modeling and Control*. Wiley.

[4] Craig, J. J. (2017). *Introduction to Robotics: Mechanics and Control* (4th ed.). Pearson.

[5] Richard M. Murray, Zexiang Li, and S. Shankar Sastry, *A Mathematical Introduction to Robotic Manipulation*, CRC Press, 1994.

9 Appendix

9.1 Trajectory Functions

```
%%%%%%%%%%  DIAMOND SQUARE %%%%%%%%%%%%%%%%
function [x, y] = fcn_diamond(u)
%#codegen
% u   : simulation time (s)
% x,y : end-effector position (m)
%
% Moves from starting point to the top vertex of a diamond,
% then traces the diamond clockwise: top  right  bottom  left
    top.
% Edge length L = 0.1 m; diamond bottom-left x-bound = 0.075
   m.

  % --- Parameters ---
  x_start = 0.15;     % initial x-position (m)
  y_start = 0.10;     % initial y-position (m)
  y0      = -0.065;   % reference y for diamond bottom-left (m
    )
  moveT   = 0.0;      % transition duration (s)
  L       = 0.1;      % edge length of square before rotation
    (m)
  T       = 10.0;     % total circuit time (s)

  % --- Diamond radius and center ---
  r  = L / sqrt(2);         % distance from center to each
    vertex
  cx = 0.075 + r;           % center x-coordinate so leftmost
    x = 0.075
  cy = y0 + L/2;            % center y-coordinate

  % --- Coordinates of the top vertex of the diamond ---
  x_d0 = cx;
  y_d0 = cy + r;

  % --- Linear transition (duration = moveT) from start  top
    vertex ---
  if u < moveT
    alpha = u / moveT;
    x = x_start + (x_d0 - x_start) * alpha;
    y = y_start + (y_d0 - y_start) * alpha;
    return
  end

  % --- Circuit time parameter and quarter duration ---
  t       = mod(u - moveT, T);
  quarter = T / 4;
```

```
  % --- Move clockwise around the diamond in four segments
     ---
  if t < quarter
    % Top  Right
    s = t / quarter;
    x = cx + r * s;
    y = (cy + r) - r * s;

  elseif t < 2*quarter
    % Right  Bottom
    s = (t - quarter) / quarter;
    x = (cx + r) - r * s;
    y = cy - r * s;

  elseif t < 3*quarter
    % Bottom  Left
    s = (t - 2*quarter) / quarter;
    x = cx - r * s;
    y = (cy - r) + r * s;

  else
    % Left  Top
    s = (t - 3*quarter) / quarter;
    x = (cx - r) + r * s;
    y = cy + r * s;
  end
end

%
% %%%%%%%%%%%%% CIRCLE  %%%%%%%%%%%%%
% function [x, y] = fcn_circle(u)
% %#codegen
% % u   : simulation time (s)
% % x,y : end-effector position (m)
% %
% % Linear transition from the starting point to the top of
   the circle,
% % followed by a full clockwise circuit.
%
%   % --- Parameters ---
%   x_start = 0.175;    % initial x-position (m)
%   y_start = 0.10;     % initial y-position (m)
%   moveT   = 0.0;      % transition duration (s)
%   T       = 5.0;      % full circle duration (s)
%
%   % --- Circle boundaries ---
%   y_min = -0.05;      % bottom y-boundary
%   y_max = +0.05;      % top y-boundary
%   x_min = 0.075;      % left x-boundary
```

```
%
%   % --- Center and radius ---
%   r  = (y_max - y_min)/2;      % radius = 0.05 m
%   cy = (y_max + y_min)/2;      % vertical center = 0
%   cx = x_min + r;              % horizontal center so that
   cx - r = 0.075
%
%   % --- Top of circle (initial target) ---
%   x_c0 = cx;
%   y_c0 = cy + r;               % = +0.05
%
%   % --- Linear transition ---
%   if u < moveT
%     alpha = u / moveT;
%     x = x_start + (x_c0 - x_start) * alpha;
%     y = y_start + (y_c0 - y_start) * alpha;
%     return
%   end
%
%   % --- Time parameter and angle calculation ---
%   t     = mod(u - moveT, T);
%   theta = 2*pi * t / T;        % from 0 to 2
%
%   % --- Clockwise circular motion ---
%   x = cx + r * sin(theta);     % sin starts at top when
   theta=0
%   y = cy + r * cos(theta);     % cos gives maximum at theta
   =0
% end

% %%%%%%%%%%%% TRIANGLE   %%%%%%%%%%
% function [x, y] = fcn_triangle(u)
% %#codegen
% % u   : simulation time (s)
% % x,y : end-effector position (m)
% %
% % Linear transition from the start point to the top vertex
   of the triangle,
% % followed by a full clockwise tour.
%
%   % --- Parameters ---
%   x_start = 0.175;    % initial x-position (m)
%   y_start = 0.10;     % initial y-position (m)
%   moveT   = 0.0;      % transition duration (s)
%   T       = 6.0;      % full triangle circuit time (s)
%
%   % --- Triangle boundaries ---
%   y_min = -0.05;
%   y_max = +0.05;
%   x_min = 0.075;
```

```
%
%   % --- Circumscribed circle center and radius ---
%   cy = (y_max + y_min)/2;  % = 0
%   r  = (y_max - y_min)/2;  % = 0.05
%   cx = x_min + r;          % so that cx - r = 0.075
%
%   % --- Triangle vertices in clockwise order (start at top)
    ---
%   V1 = [ cx,            cy + r       ];  % top vertex
%   V2 = [ cx + r*sqrt(3)/2, cy - r/2  ];  % bottom-right
   vertex
%   V3 = [ cx - r*sqrt(3)/2, cy - r/2  ];  % bottom-left
   vertex
%
%   % --- Linear transition to top vertex ---
%   if u < moveT
%     alpha = u / moveT;
%     x = x_start + (V1(1) - x_start) * alpha;
%     y = y_start + (V1(2) - y_start) * alpha;
%     return
%   end
%
%   % --- Motion parameter ---
%   t     = mod(u - moveT, T);
%   third = T / 3;
%
%   % --- Clockwise segments: toprightlefttop ---
%   if t < third
%     % Top  bottom-right
%     s = t / third;
%     x = V1(1) + (V2(1) - V1(1)) * s;
%     y = V1(2) + (V2(2) - V1(2)) * s;
%
%   elseif t < 2*third
%     % Bottom-right  bottom-left
%     s = (t - third) / third;
%     x = V2(1) + (V3(1) - V2(1)) * s;
%     y = V2(2) + (V3(2) - V2(2)) * s;
%
%   else
%     % Bottom-left  top
%     s = (t - 2*third) / third;
%     x = V3(1) + (V1(1) - V3(1)) * s;
%     y = V3(2) + (V1(2) - V3(2)) * s;
%   end
% end
%
% %%%%%%%%%% AYBU SEQUENCE %%%%%%%%%%
%% For Writing AYBU directly, uncomment all letter sections
   too
```

```
% function [x, y] = fcn_sequence(u)
% %#codegen
% % u   : simulation time (s)
% % x,y : end-effector position (m)
% %
% % Letter durations (s):
% %   A: 6, Y: 6, B: 8, U: 6
% % Sequentially traces AYBU without pause.
%
%   % --- Individual letter durations ---
%   durA = 6.0;
%   durY = 6.0;
%   durB = 8.0;
%   durU = 6.0;
%   Ttot = durA + durY + durB + durU;
%
%   % --- Local time within the full sequence ---
%   t = mod(u, Ttot);
%
%   % --- Select letter based on elapsed time ---
%   if t < durA
%     tloc = t;
%     [x, y] = fcn_A(tloc);
%
%   elseif t < durA + durY
%     tloc = t - durA;
%     [x, y] = fcn_Y(tloc);
%
%   elseif t < durA + durY + durB
%     tloc = t - (durA + durY);
%     [x, y] = fcn_draw(tloc);
%
%   else
%     tloc = t - (durA + durY + durB);
%     [x, y] = fcn_U(tloc);
%   end
% end
%
% %%%% A %%%%%%
% function [x, y] = fcn_A(u)
% %#codegen
% % u   : simulation time (s)
% % x,y : end-effector position (m)
% %
% % 'A' letter: starts at the top vertex, then draws the left
    leg,
% % the right leg, and finally the middle crossbar, over
   total time T.
% % Operates in x  [0.075, 0.115] m.
%
```

```
%   % --- Parameters ---
%   x_start = 0.115;   % initial x-position (m)
%   y_start = 0.10;    % initial y-position (m)
%   moveT   = 0.0;     % transition duration (s)
%   T       = 6.0;     % total drawing time (s)
%
%   % --- Drawing region (4 cm wide) ---
%   x_min = 0.075;
%   x_max = 0.115;
%   y_min = -0.05;
%   y_max = +0.05;
%   slotW = x_max - x_min;  % =0.04
%   cy    = (y_max + y_min)/2;
%
%   % --- Corner and crossbar endpoints ---
%   x0  = x_min + slotW/2;
%   VA1 = [x0 - 0.012, y_min];  % bottom-left
%   VA2 = [x0,           y_max]; % top vertex
%   VA3 = [x0 + 0.012, y_min];  % bottom-right
%   VA4 = [x0 - 0.006, cy];     % left midpoint
%   VA5 = [x0 + 0.006, cy];     % right midpoint
%
%   % --- Initial move to VA1 ---
%   if u < moveT
%     alpha = u / moveT;
%     x = x_start + (VA1(1) - x_start)*alpha;
%     y = y_start + (VA1(2) - y_start)*alpha;
%     return
%   end
%
%   % --- Time and segment duration ---
%   t   = mod(u - moveT, T);
%   dur = T/3;
%
%   % --- Determine segment and interpolate ---
%   if t < dur
%     s = t/dur; A = VA1; B = VA2;  % left leg
%   elseif t < 2*dur
%     s = (t-dur)/dur; A = VA3; B = VA2;  % right leg
%   else
%     s = (t-2*dur)/dur; A = VA4; B = VA5; % middle bar
%   end
%
%   x = (1-s)*A(1) + s*B(1);
%   y = (1-s)*A(2) + s*B(2);
% end
%
% %%%% Y %%%%%%%%%%
% function [x, y] = fcn_Y(u)
% %#codegen
```

```
% % u   : simulation time (s)
% % x,y : end-effector position (m)
% %
% % 'Y' letter: draws left arm  junction  right arm  stem,
% % over total time T, within x  [0.11, 0.15] m.
%
%   % --- Parameters ---
%   x_start = 0.15;    % initial x-position (m)
%   y_start = 0.10;    % initial y-position (m)
%   moveT   = 0.0;     % transition duration (s)
%   T       = 6.0;     % total drawing time (s)
%
%   % --- Drawing region (4 cm wide) ---
%   x_min = 0.11;
%   x_max = 0.15;
%   y_min = -0.05;
%   y_max = +0.05;
%   slotW = x_max - x_min;  % =0.04
%   cy    = (y_max + y_min)/2;
%
%   % --- Arm and stem endpoints ---
%   x0 = x_min + slotW/2;
%   V1 = [x0 - 0.012, y_max]; % left arm top
%   V2 = [x0,           cy];  % junction
%   V3 = [x0 + 0.012, y_max]; % right arm top
%   V4 = [x0,          y_min]; % stem bottom
%
%   % --- Initial move to V1 ---
%   if u < moveT
%     alpha = u / moveT;
%     x = x_start + (V1(1) - x_start)*alpha;
%     y = y_start + (V1(2) - y_start)*alpha;
%     return
%   end
%
%   % --- Time and segment duration ---
%   t   = mod(u - moveT, T);
%   dur = T/3;
%
%   % --- Segment selection and interpolate ---
%   if t < dur
%     s = t/dur; A = V1; B = V2;
%   elseif t < 2*dur
%     s = (t-dur)/dur; A = V3; B = V2;
%   else
%     s = (t-2*dur)/dur; A = V2; B = V4;
%   end
%
%   x = (1-s)*A(1) + s*B(1);
%   y = (1-s)*A(2) + s*B(2);
```

```
% end
%
% %%%%% B %%%%%%%%5
% function [x, y] = fcn_draw(u)
% %#codegen
% % u   : simulation time (s)
% % x,y : end-effector position (m)
% %
% % 'B'-like shape:
% % 1) vertical stroke at x=0.145, from y=-0.05 to y=+0.05 (
   T1 s)
% % 2) upper semicircle, starting from the top downwards (T2
   s)
% % 3) lower semicircle, starting from the top downwards (T2
   s)
% % Then loop repeats.
%
%   % --- Timing ---
%   T1 = 2.0; % vertical stroke duration (s)
%   T2 = 3.0; % each semicircle duration (s)
%   T  = T1 + 2*T2;
%   t  = mod(u, T);
%
%   % --- Parameters ---
%   x_line   = 0.145;    % x-position for stroke and
   semicircles
%   y_top    = +0.05;
%   y_bottom = -0.05;
%   yc_up    = +0.025;   % upper semicircle center y
%   yc_down  = -0.025;   % lower semicircle center y
%   r        = 0.02;     % radius
%
%   if t < T1
%     % Vertical stroke
%     s = t/T1;
%     x = x_line;
%     y = y_bottom + s*(y_top - y_bottom);
%
%   elseif t < T1 + T2
%     % Upper semicircle (top to bottom)
%     t2    = t - T1;
%     theta = pi/2 - pi*(t2/T2);
%     x     = x_line + r*cos(theta);
%     y     = yc_up  + r*sin(theta);
%
%   else
%     % Lower semicircle (top to bottom)
%     t3    = t - (T1 + T2);
%     theta = pi/2 - pi*(t3/T2);
%     x     = x_line + r*cos(theta);
```

```
%     y     = yc_down + r*sin(theta);
%   end
% end
%
% %%%%%%%%%% U %%%%%%%%%%%%%%%%%
% function [x, y] = fcn_U(u)
% %#codegen
% % u   : simulation time (s)
% % x,y : end-effector position (m)
% %
% % 'U' letter:
% % 1) transition to top-left corner
% % 2) left vertical stroke
% % 3) bottom semicircle
% % 4) right vertical stroke
% % Total drawing time = 6 s, equal-duration segments.
%
%   % --- Constants ---
%   moveT  = 0.0;
%   Tshape = 6.0;
%   dur    = Tshape/3;
%
%   % --- Drawing region (4 cm wide) ---
%   x_min = 0.165; x_max = 0.215;
%   y_min = -0.05; y_max = +0.05;
%   slotW = x_max - x_min;    % =0.04 m
%   x0    = (x_min + x_max)/2;% =0.195 m
%   dx    = slotW/4;          % =0.01 m
%   VA1   = [x0-dx, y_max];    % top-left
%   VA2   = [x0-dx, y_min];    % bottom-left
%   VA3   = [x0+dx, y_min];    % bottom-right
%   VA4   = [x0+dx, y_max];    % top-right
%
%   if u < moveT
%     alpha = u/moveT;
%     x = x_min + (VA1(1)-x_min)*alpha;
%     y = y_max + (VA1(2)-y_max)*alpha;
%     return
%   end
%
%   t = mod(u - moveT, Tshape);
%   if t < dur
%     s = t/dur;
%     x = VA1(1) + s*(VA2(1)-VA1(1));
%     y = VA1(2) + s*(VA2(2)-VA1(2));
%   elseif t < 2*dur
%     s     = (t-dur)/dur;
%     theta = pi - pi*s;
%     xc    = x0; yc = y_min; r = dx;
%     x     = xc + r*cos(theta);
```

```
%      y      = yc - r*sin(theta);
%    else
%      s = (t-2*dur)/dur;
%      x = VA3(1) + s*(VA4(1)-VA3(1));
%      y = VA3(2) + s*(VA4(2)-VA3(2));
%    end
% end
```

9.2 Inverse Kinematic Function

```
function [theta1,theta2] = InverseKinematics(x,y)

l1 = 0.15;
l2 = 0.1;

theta2 = acos((x^2 + y^2 -l1^2 - l2^2)/(2*l1*l2));

s_Theta2 = sin(theta2);
c_Theta2 = cos(theta2);

theta1 = atan2(y,x) - atan2(l2*s_Theta2,(l1+l2*c_Theta2));
```

9.3 Forward Kinematic Function

```
function [xa, ya] = Forward_Kinematics(theta1, theta2)
    l1 = 0.15;
    l2 = 0.1;
    xa = l1*cos(theta1) + l2*cos(theta1 + theta2);
    ya = l1*sin(theta1) + l2*sin(theta1 + theta2);
```

9.4 System Torque Calculation Function

```
function [tau1_peak, tau2_peak, tau1_all, tau2_all] =
   computeMotorPeakTorques(theta1, theta2, dtheta1, dtheta2,
   ddtheta1, ddtheta2)
% ======================================================
% Calculates PEAK (maximum) torques for Servo-1 and Servo-2
   over a time series
% Also returns all torque values over time
% ======================================================
% Inputs (as vectors): arrays of size [1xN]
%   theta1, theta2          [rad]      Joint angles
%   dtheta1, dtheta2        [rad/s]    Angular velocities
%   ddtheta1, ddtheta2      [rad/s^2]  Angular accelerations
%
% Outputs:
%   tau1_peak               [Nm]  Maximum torque required by
   Servo-1
```

```
%   tau2_peak               [Nm]  Maximum torque required by
   Servo-2
%   tau1_all                [1xN]  All torque values for Servo
   -1
%   tau2_all                [1xN]  All torque values for Servo
   -2
% ================================================
% ------------------------------
% Robot arm parameters
% ------------------------------
L1 = 0.15;                % [m] Length of Link-1
L2 = 0.10;                % [m] Length of Link-2
Lc1 = 0.075;              % [m] Center of mass of Link-1
Lc2 = 0.05;               % [m] Center of mass of Link-2

m1 = 0.086;               % [kg]  Mass of Link-1 (excluding
   motor-1)
m2 = 0.106;               % [kg]  Mass of Link-2 + motor-2 +
   payload

I1 = 0.00114957;          % [kg m^2] Inertia of Link-1 about
   its joint
I2 = 0.0066028664;        % [kg m^2] Inertia of Link-2 + motor
   -2 + payload

N = length(theta1);
tau_all = zeros(2, N);  % Array to store torques at each time
    step

for i = 1:N
    % Trigonometric terms
    c2 = cos(theta2(i));
    s2 = sin(theta2(i));

    % Inertia matrix
    m11 = I1 + I2 + m1*Lc1^2 + m2*(L1^2 + Lc2^2 + 2*L1*Lc2*c2
       );
    m12 = I2 + m2*(Lc2^2 + L1*Lc2*c2);
    m21 = m12;
    m22 = I2 + m2*Lc2^2;
    M = [m11, m12; m21, m22];

    % Coriolis and centrifugal matrix
    h = -m2 * L1 * Lc2 * s2;
    C = [h*dtheta2(i), h*(dtheta1(i) + dtheta2(i));
         -h*dtheta1(i), 0];

    % Total torque at this step
    dtheta = [dtheta1(i); dtheta2(i)];
    ddtheta = [ddtheta1(i); ddtheta2(i)];
```

```
    tau_all(:, i) = M * ddtheta + C * dtheta ;
end

% Extract torque values
tau1_all = tau_all(1, :);
tau2_all = tau_all(2, :);

% Calculate peak torques
tau1_peak = max(abs(tau1_all));
tau2_peak = max(abs(tau2_all));

end
```

9.5 Robotic Arm Simulink Block Diagram

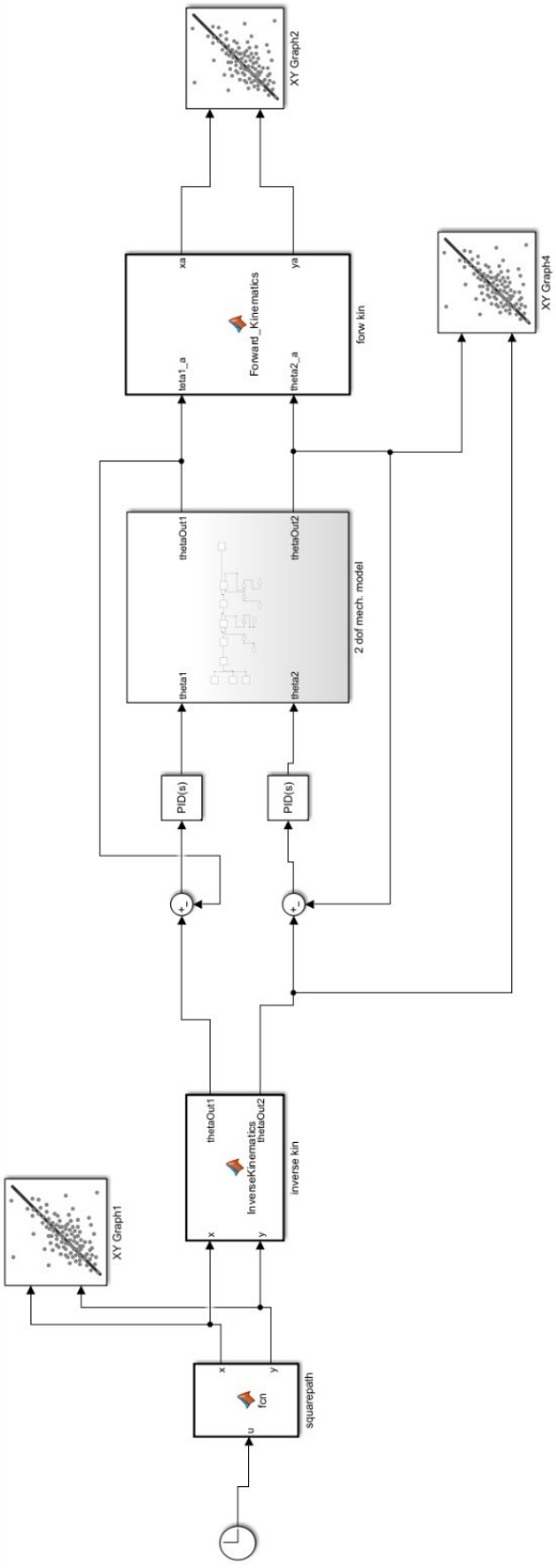

Figure 13: Simulink Block Diagram with PID Controller

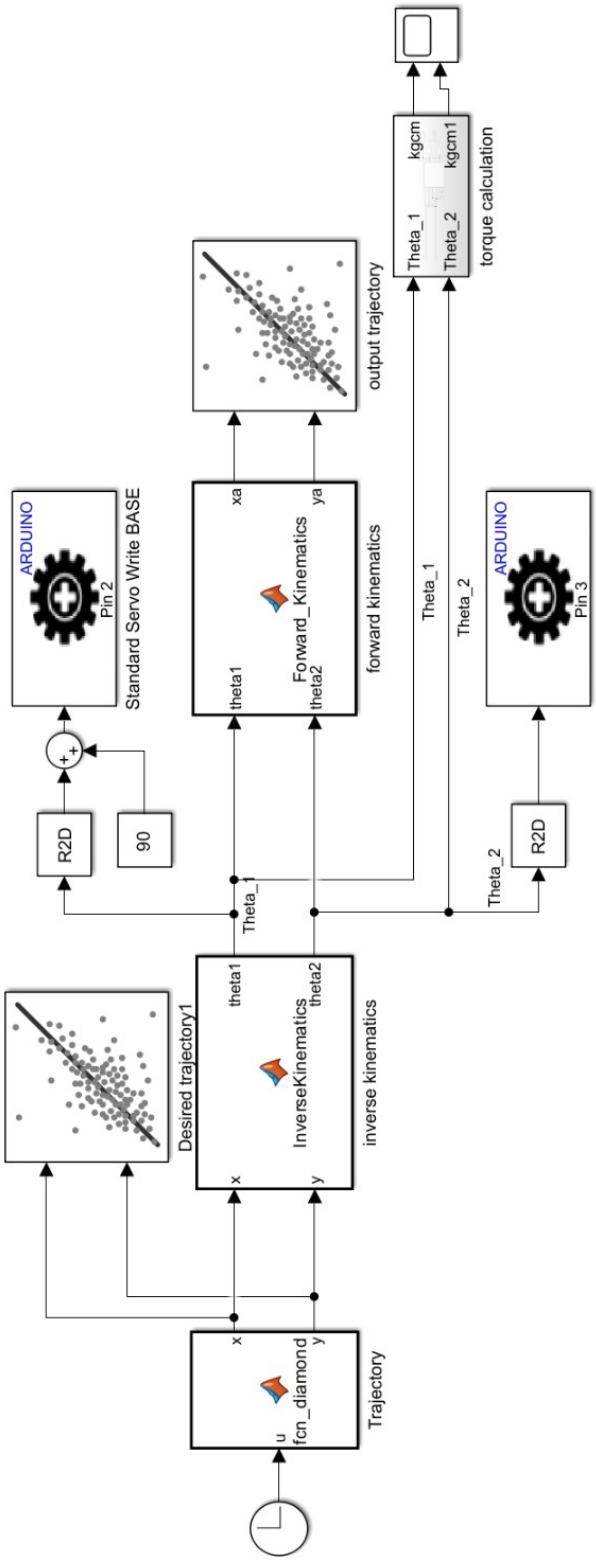

Figure 14: Simulink Block Diagram for Arduino Implementation

9.6 Technical Drawing of the Robotic Arm