

# **Department of Electrical & Computer Engineering**

**Pak-Austria Fachhochschule: Institute of Applied Sciences and  
Technology, Haripur, Pakistan**



## **ECE-261L Introduction to Embedded Systems Lab**

### **Project Report**

#### **Submitted by:**

Huzaifa Tahir

Reg # B23F0178RO022

Hussain Khan

Reg # B23F0032RO033

#### **Submitted to:**

Engr. Rafi Ullah Khan

# Title – Modeling and Analysis of a 5 DOF Robotic Arm Manipulator

## Objective

- To use technical skills gained from the Embedded Systems course in implementing a 5-DoF Robot Arm.

## Introduction

This project develops a kinematic model for a 5-DOF (Degrees of Freedom) industrial robot that controls its hand position. In Robotics terms, we call the hand position as end effector position.

An easier method to control the end effector position is by changing the servo position using `servo.write()` function. This can essentially move the end-effector to any random position.

The goal of this project is to control the end effector coordinate position in a precise manner. By coordinate position, we mean the xyz points that it can reach given its configurations (link lengths, etc.).

We achieve this goal by developing a mathematical model of the whole robot. This is done by assigning individual xyz coordinate frames where each servo motor is located, and then analytically estimating where the end effector coordinate frame might be on the reference frame. The reference frame is where the first joint is located **in case of our project**. This whole process is called **Forward Kinematics**.

To make the end effector reach a specific xyz coordinate position in the robot's workspace we perform **Inverse Kinematics**, where we use the equations of Forward Kinematics to come up with the required joint angles for that specific xyz coordinate place. Workspace is the set of all positions that the robot's end effector is able to reach.

This project was a bright opportunity for us to achieve what we had learned about Robot Mechanics and implementing them using the techniques we learned in Embedded Systems course.

## Components

The project is based on the following components.

1. ESP32
2. Robot Frame
3. Servo Motors

The ESP32 microcontroller was used for better processing efficiency and memory and basic AD002 servo motors were used. So we were able to achieve the tasks using basic `<Servo.h>` library, and advanced servo motor control techniques were not required for this project.

## Structure and Mathematical Modeling

We used the frame of Adeept 5 DoF Robot Arm on which all the servo motors are properly attached.

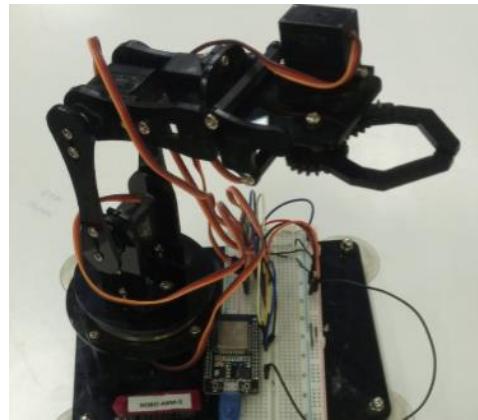


Figure 1

The following diagram shows how we abstract and model the robot structure that helps us in calculations. For ease, only the first 3 Revolute Joints (or servo motors) are modeled using Inverse Kinematics and the remaining 2 Revolute Joints are taken as Wrist and Grip. So the overall structure is indeed 5 DoF.

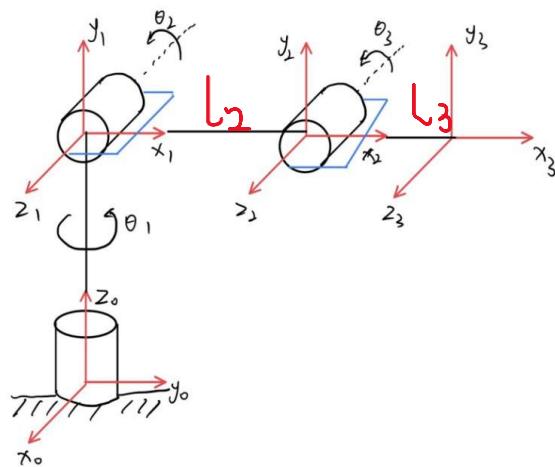


Figure 2

After obtaining this structure, we fill in the following table.

{i}	$\alpha_{i-1}$	$a_{i-1}$	$d_i$	$\theta_i$
1.	0	0	$l_1$	$\theta_1$
2.	$90^\circ$	0	0	$\theta_2$
3.	0	$l_2$	0	$\theta_3$
4.	0	$l_3$	0	0

This is called the DH table, and {i} corresponds to the frame number; the frame number is also a kind of joint number. Using forward kinematics techniques, we reach the following equation that relates the servo motor position and end effector position.

```
T04 =
[cos(q2 + q3)*cos(q1), -sin(q2 + q3)*cos(q1), sin(q1), cos(q1)*(13*cos(q2 + q3) + 12*cos(q2))]
[cos(q2 + q3)*sin(q1), -sin(q2 + q3)*sin(q1), -cos(q1), sin(q1)*(13*cos(q2 + q3) + 12*cos(q2))]
[sin(q2 + q3), cos(q2 + q3), 0, 11 + 13*sin(q2 + q3) + 12*sin(q2)]
[0, 0, 0, 1]
```

Figure 3

From this matrix equation in Figure 3 we get to know that the xyz position has the following equations,

$$x = l_2 c_1 c_2 + l_3 c_1 c_{23}$$

$$y = l_2 s_1 c_2 + l_3 s_1 c_{23}$$

$$z = l_2 s_2 + l_1 s_{23}$$

Where 'c' and 's' correspond to cosine and sine angles.

We solve these transcendental equations and derive equations for the thetas.

$$\theta_1 = \text{atan2}(y, x)$$

$$r = \sqrt{x^2 + y^2}$$

$$\theta_2 = \text{atan2}\left(\frac{z}{r}\right) - \text{atan2}\left(\frac{l_3 s_3}{l_2 + l_3 c_3}\right)$$

$$\theta_3 = \text{atan2}(s_3, c_3)$$

## Methadology

After doing all this, we proceed towards coding.

The code is focused more on pick and place operations using the coordinate positions (xyz) of the object to be picked and the placement destination (xyz).

The robot first moves to the XYZ position of the object using inverse kinematics. Then it releases the gripper to release the object at the destination XYZ position, using inverse kinematics.

Instead of immediately reaching the robot to the desired xyz coordinates, interpolation techniques have been incorporated to make the robot's movement smooth and stable-looking.

So the overall working is,

1. The robot approaches the object position and grabs the object using the wrist and gripper
2. Robot moves to the top position
3. Robot releases the object at the destination position using the wrist and gripper.

## Code Flow Diagram

Figure 4 briefly demonstrates how the code performs inverse kinematics for our robot.

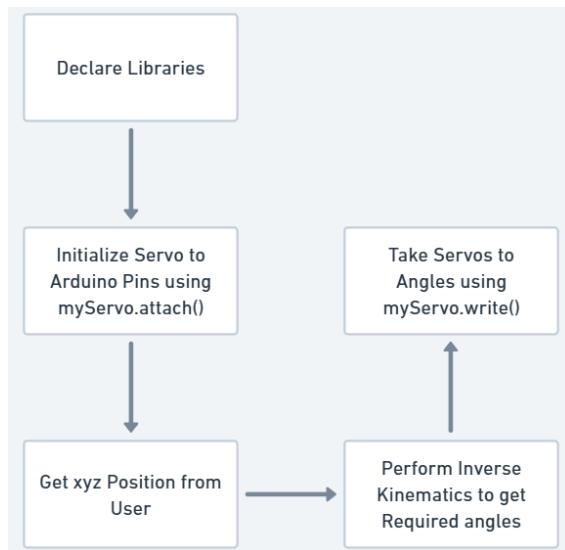


Figure 4

Figure 5 demonstrates the working of the project.

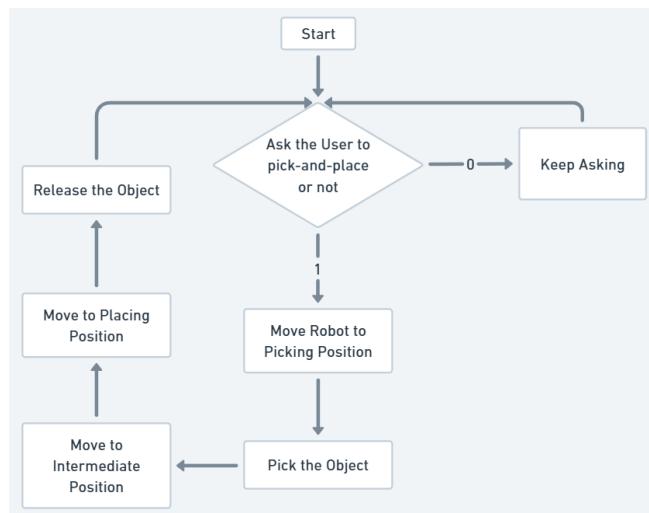


Figure 5

## Code

### Library Initialization

```
#include <ESP32Servo.h>
#include <math.h>
const float pi = 3.14159265;
```

### Initializing Revolute Joints

```
Servo F0;
Servo F1;
Servo F2;
Servo F3;
Servo F4;
Servo Wrist;
```

### Joint and Wrist Pins Assignments

```
int P0 = 23;
int P1 = 22;
int P2 = 21;
int P3 = 19;
int wr = 18;
```

### Number of Links, Lengths, and Joints

```
const int n = 4;
float L1 = 0; float L2 = 10; float L3 = 20;
float q1; float q2; float q3;
```

### Initialize Robot Parameters

```
// Link Twist
float alpha[n] = {0, pi/2, 0, 0};
```

```

// Link Length
float a[n] = {0, 0, L2, L3};
// Offset Distance
float d[n] = {0, L1, 0, 0};
// Joint Angles
float theta[n] = {q1, q2, q3, 0};

```

## Joint Angles

```

struct IKResult
{
    long double q1;
    long double q2;
    long double q3;
};

```

## Inverse Kinematics Function

```

IKResult ik_hw(float x, float y, float z, float L1, float L2, float L3)
{
    IKResult result;

    // radial distance
    long double r = sqrt((long double)x*x + (long double)y*y);

    // compute cos(q3)
    long double cos3 = (r*r + (z - L1)*(z - L1) - L2*L2 - L3*L3) / (2.0L * L2 * L3);

    // clamp to avoid domain error
    if (cos3 > 1.0L) cos3 = 1.0L;
    if (cos3 < -1.0L) cos3 = -1.0L;

    // compute sin(q3) safely
    long double sin3 = sqrtl(1.0L - cos3*cos3);

    // joint 3
    long double q3 = atan2l(sin3, cos3);

    // joint 1
    long double q1 = atan2l(y, x);

    // helper angle
    long double psi = atan2l(z - L1, r);

    // joint 2
    long double q2 = psi - atan2l(L3 * sin3, L2 + L3 * cos3);

    // Keep inside 0-180 range
    if (q1 < 0) q1 += 360;           // bring to 0-360
    if (q1 > 180) q1 = 180;         // servo limit
}

```

```

    if (q2 < 0) q2 = 0;
    if (q2 > 180) q2 = 180;

    if (q3 < 0) q3 = 0;
    if (q3 > 180) q3 = 180;

    // assign
    result.q1 = q1;
    result.q2 = q2;
    result.q3 = q3;

    return result;
}

```

## rad2deg() Function

```

long double rad2deg(float angle)
{
    return angle * (180 / pi);
}

```

## Initial Configurations

```

float x = 0;
float y = 0;
float z = 0;
IKResult target;

```

## The Setup Function

```

void setup()
{
    // Attach Servos
    F0.attach(P0);
    F1.attach(P1);
    F2.attach(P2);
    F3.attach(P3);
    Wrist.attach(wr);
    // F4.attach(P4);
    // Serial Setup
    Serial.begin(115200);
    delay(1000);
}

```

```
char decide = false;
```

## The Loop Function

```

void loop()
{
    Serial.println("Pick / Place Decide ('1' for Yes & '0' for No):");

```

```

// Wait for user input
while (Serial.available() == 0) { delay(10); }

// Read 3 floats from a single line
decide = Serial.read();

if(decide)
{
    Serial.print("Picking : ");
    pick();
    delay(2000);
    move_top();
    delay(2000);
    Serial.println("Placing : ");
    place();
}
}

```

### This Function Closes the Jaw or Gripper

```

void close_jaw()
{
    F3.write(70);
}

```

### This Function Opens the Jaw or Gripper

```

void open_jaw()
{
    F3.write(10);
}

```

### Pick the Object

```

void pick()
{
    x = -10; y = -10; z = -10;
    target = ik_hw(x, y, z, L1, L2, L3);
    q1 = round(rad2deg(target.q1));
    q2 = round(rad2deg(target.q2));
    q3 = round(rad2deg(target.q3));

    // Move the Robot
    move_robot(q1, q2, q3);

    // Move Wrist :
    move_wrist(90);
    // Open Jaw :
    open_jaw();
    // Wait for 2 seconds
}

```

```

delay(2000);
// Close Jaw :
close_jaw();
// Wait for 2 seconds
delay(2000);
}

```

## Place the Object

```

void place()
{
    x = 0; y = 20; z = 10;
    target = ik_hw(x, y, z, L1, L2, L3);
    q1 = round(rad2deg(target.q1));
    q2 = round(rad2deg(target.q2));
    q3 = round(rad2deg(target.q3));

    // Move the Robot
    move_robot(q1, q2, q3);

    // Move Wrist :
    move_wrist(90);
    // Open Jaw :
    close_jaw();
    // Wait for 2 seconds
    delay(2000);
    // Close Jaw :
    open_jaw();
    // Wait for 2 seconds
    delay(2000);
}

```

## Move to Top

```

void move_top()
{
    x = 0; y = 30; z = 0;
    target = ik_hw(x, y, z, L1, L2, L3);
    q1 = round(rad2deg(target.q1));
    q2 = round(rad2deg(target.q2));
    q3 = round(rad2deg(target.q3));
    move_wrist(0);
    move_robot(q1, q2, q3);
}

```

## Function to Move the Wrist to Desired Orientation

```

void move_wrist(float ang)
{
    Wrist.write(ang);
}

```

## Move the Robot using Interpolation Techniques

```
void move_robot(float q1_target, float q2_target, float q3_target)
{
    int steps = 20;

    float q1_start = F0.read();
    float q2_start = F1.read();
    float q3_start = F2.read();

    float dq1 = (q1_target - q1_start) / steps;
    float dq2 = (q2_target - q2_start) / steps;
    float dq3 = (q3_target - q3_start) / steps;

    for(int i = 0; i < steps; i++)
    {
        q1_start += dq1;
        q2_start += dq2;
        q3_start += dq3;

        F0.write(q1_start);
        F1.write(q2_start);
        F2.write(q3_start);
    }
}
```

## MATLAB Simulation

MATLAB simulations are performed to verify whether our equations work or not. We mainly focused on the pick-and-place operation, a task that has applications. Figure 6 represents the object picking position, and Figure 8 represents the object placing position. Figure 7 is the intermediate position. We can see the robot is performing tasks as it is intended to.

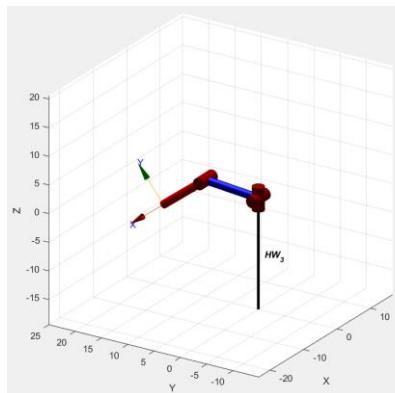


Figure 6

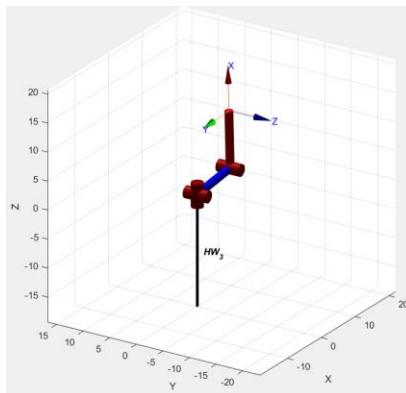


Figure 7

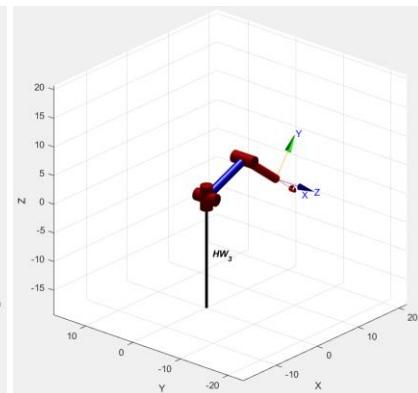


Figure 8

## Practical Results

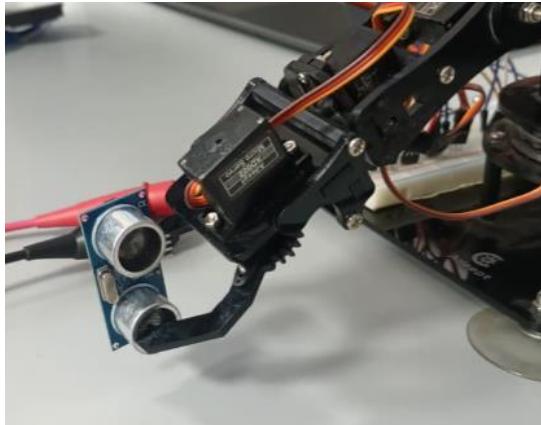


Figure 9



Figure 10

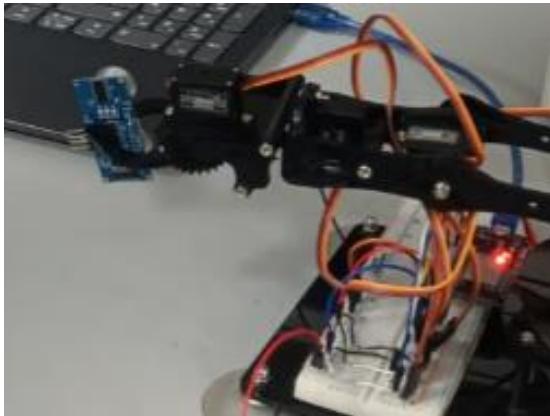


Figure 11

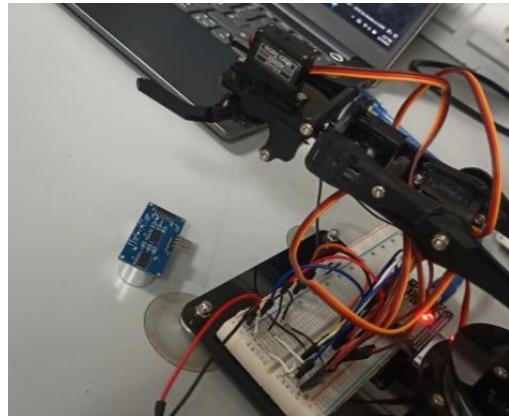


Figure 12

## Conclusion

From this Lab Semester Project of Embedded Systems, we got the opportunity to implement both what we learned in the Introduction to Embedded Systems and in the Robot Mechanics course. The project introduced us to various challenges and improved our hands-on practice with practical equipment. It made us realize how theoretical and real-world implementations can vastly differ in achieving desired results. The project was accomplished, and the developed model and equations operate for the designed 5DoF robot.