Six DOF Robotic Manipulator

EE-339

Design Laboratory

Rishi Tirpude : 220102081

Yash Murmadkar : 220102103

Under the supervision of

Prof. Indrani Kar

EEE Department

Indian Institute of Technology, Guwahati

Abstract :

This project presents the design and development of a Six Degrees of Freedom (DOF) Robotic Arm controlled wirelessly using the ESP32 microcontroller. The robotic arm consists of six servo motors, each responsible for controlling a specific joint, allowing for complex and precise movements. A custom-built web interface hosted by the ESP32 provides six sliders, each corresponding to one servo. Through this intuitive interface, users can manipulate the position of each joint in real-time over a Wi-Fi connection, without the need for an external server or internet access. The ESP32 acts as both the web server and the control unit, receiving HTTP requests based on slider movements and translating them into angular positions for each motor. This setup enables tasks such as pick-and-place operations, demonstrating a compact, cost-effective, and interactive solution for learning, prototyping, and automation applications.

Contents

I. Introduction:

As part of our Design Laboratory course, we developed a **6 Degrees of Freedom (DOF) robotic arm** capable of performing **basic pick and place operations**. The goal was to create a functional system that demonstrates core control principles while giving us practical experience with real-world robotic applications.

We began prototyping with an Arduino board but soon faced limitations, particularly the absence of wireless connectivity. To address this, we transitioned to the **ESP32 board**, which offers built-in Wi-Fi support. This upgrade enabled us to control the robotic arm wirelessly, allowing for more flexible and convenient operation.

To add a degree of automation, we implemented a **"Save and Reset" feature**, allowing the system to store and repeat servo motor positions from a motion sequence. This functionality introduces repeatability into the process, which is essential in industrial automation tasks.

Robotic arms are widely used in **modern industries** for tasks such as *assembly, packaging, welding, painting*, and even precise operations like *surgery*. Through this project, we explored both theoretical and practical aspects of control systems, gaining valuable experience in electronics, programming, and system integration.

II. Theory:

A **6 DOF robotic arm** provides movement in three translational and three rotational axes, enabling precise positioning in 3D space. Each joint is actuated using a **servo motor**, controlled via **PWM signals** to achieve the desired angular positions.

To build the structure, we **3D printed the arm's body**, allowing for custom design, lightweight construction, and ease of assembly. This also gave us the flexibility to modify parts as needed during the prototyping phase.

The control system runs on an **ESP32 microcontroller**, which interprets user commands sent over Wi-Fi and maps them to corresponding servo positions. The system operates in an open-loop manner, without feedback sensors, focusing on basic control principles.

We also implemented a **Save and Reset feature** to store and replay servo positions, enabling simple automation. This mimics how real-world robotic arms are programmed to perform repetitive tasks with consistency.

ESP32 WIFI Module

The **ESP32** is a low-cost, low-power microcontroller with integrated **Wi-Fi and Bluetooth** capabilities, making it ideal for wireless embedded applications. Its built-in Wi-Fi module complies with the **IEEE 802.11 b/g/n** standards and supports both **Station (STA)** and **Access Point (AP)** modes.

In our project, the ESP32 was configured to receive user commands over Wi-Fi, allowing wireless control of the robotic arm. This eliminated the need for a physical connection between the controller and the arm, enabling more flexible and remote operation. The Wi-Fi module communicates via standard TCP/IP protocols, and the ESP32's dual-core processor ensures smooth handling of both communication and motor control tasks simultaneously.

ESP32 PWM Signal

The **ESP32 Dev Module** provides advanced PWM capabilities through its **LEDC (LED Control)** peripheral, which allows the generation of high-resolution PWM signals on almost any GPIO pin. Unlike traditional microcontrollers with a fixed number of PWM-capable pins, the ESP32 supports **up to 16 independent PWM channels**, giving developers flexibility in signal generation.

In our robotic arm, we used PWM signals to control the servo motors, where the width of the pulse determines the angular position of the servo shaft. The ESP32 can output PWM on nearly all GPIOs, with commonly used PWM-capable pins being **GPIO 2, 4, 5, 12–19, 21–23, 25–27, 32, and 33**. Each channel allows configuration of frequency and duty cycle, enabling precise motor control.

Using the ESP32's dual-core architecture, we ensured that PWM generation and Wi-Fi communication could run concurrently without performance drops, making it ideal for real-time control applications like our robotic arm.

Servo Motors

Servo motors are widely used in robotics for precise angular control. They consist of a small DC motor, a gear system, a position sensor (usually a potentiometer), and a control circuit. The desired angle is achieved by sending a **PWM (Pulse Width Modulated)** signal to the servo, where the **pulse width** determines the target position—typically, a 1 ms pulse moves the shaft to 0°, 1.5 ms to 90°, and 2 ms to 180°.

In our robotic arm, servo motors were used at each joint to control the orientation and position of the links. They are ideal for such applications due to their **compact size, built-in control circuitry, and ease of interfacing** with microcontrollers like the ESP32. Since each motor holds its position based on feedback from the internal sensor, it simplifies motion control in systems that do not require high torque or continuous rotation.

The ability to send specific position commands makes servo motors perfect for applications requiring repeatability, such as pick-and-place operations in our project.

Degree Of Freedom (DOF)

In robotics, **Degrees of Freedom (DOF)** refer to the number of independent movements a robot or robotic joint can perform. Each DOF represents a unique way the robot can move—either through linear translation (along X, Y, or Z axes) or rotational motion (roll, pitch, or yaw). A robotic arm with 6 DOF, like the one in our project, can position its end-effector at any point in 3D space with any orientation, offering full spatial maneuverability.

Higher degrees of freedom provide more flexibility and precision, allowing the arm to perform complex tasks such as reaching around obstacles or orienting tools at specific angles. However, increased DOF also adds complexity to the control system, requiring more sophisticated coordination of joint movements.

Forward Kinematics

Forward Kinematics is the method used in robotics to calculate the position and orientation of the end-effector of a robotic arm, given the angles or displacements of its joints. It involves applying a series of geometric transformations—such as rotations and translations—based on the link lengths and joint angles, to determine the final spatial location of the robotic arm's tip.

In this project, Forward Kinematics has been used to verify the position of the end-effector after assigning specific angles to the joints of the robotic arm. By inputting angles to the base, shoulder, and elbow joints, we can compute the expected (x, y, z) coordinates of the end-effector. This implementation helps validate whether the arm reaches the intended position, especially when working alongside Inverse Kinematics for motion planning. It also provides a reference for repeating automated tasks where fixed joint configurations are used.

Inverse Kinematics

Inverse Kinematics (IK) is a mathematical method used in robotics to determine the joint parameters required to move the end-effector of a robot to a desired position in space. Unlike forward kinematics, where the joint angles are known and the resulting position is calculated, IK works in reverse—it starts with a target position and computes the joint angles that will achieve it.

In our project, we implemented a basic **3-DOF inverse kinematics system** on the base, shoulder, and elbow joints of the robotic arm. When the user enters the coordinates of a target point, the system calculates the necessary joint angles using trigonometric equations derived from the geometry of the arm. This allows the end-effector to move toward the specified location in 2D space, enabling more intuitive and precise control.

III. Development Setup and Components:

Arduino IDE and ESP32 Programming

The **Arduino IDE** is an easy-to-use platform for writing and uploading code to microcontrollers. To program the **ESP32**, users must install the ESP32 board package by adding its URL in the Preferences and selecting it via the `Boards Manager`. After choosing "ESP32 Dev Module" and the correct COM port, the board is ready to be programmed like any Arduino. Additional configurations like flash frequency and upload speed can be set through the `Tools` menu.

Hardware Components

The following hardware components were used in the development of the 6 DOF robotic arm:

- **ESP32 Dev Module** - microcontroller with built-in Wi-Fi and Bluetooth module.

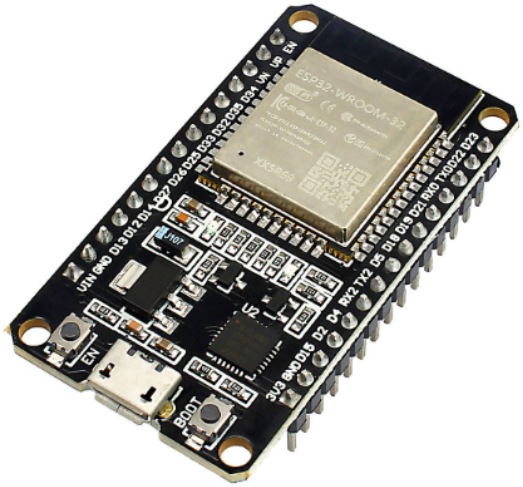


Figure: ESP32 Dev Module (Front View)

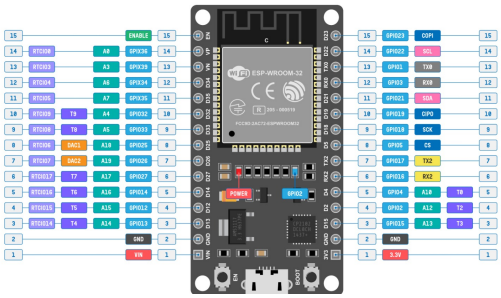


Figure: ESP32 Dev Module (Pin Diagram)

- **SG90 Servo Motors** - for Gripper and Wrist actuation.

Figure: SG90 Servo Motor

- **MG995 Servo Motors** – used for the base, shoulder, and elbow joints as these handle most of the arm's load. MG995 servos are preferred here due to their high torque capabilities.

Figure: MG995 Servo Motor

- **3D Printed Parts** -for the arm structure and joints (Not all parts shown here).

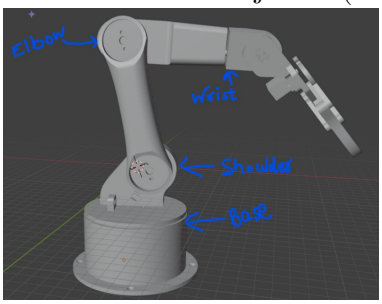


Figure: 3D Printed Robotic Arm Parts

- **Jumper Wires and Connectors** -for circuit connections.

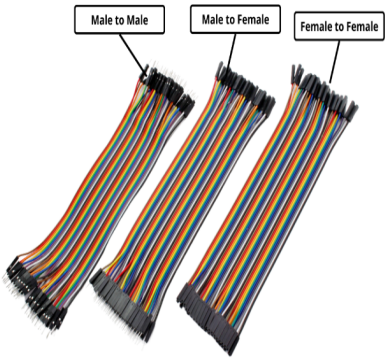


Figure: Jumper Wires

- **Breadboard** (for initial prototyping and testing)

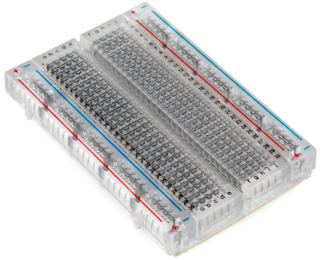


Figure: Breadboard

- **Micro USB Cable** (for programming and power)

Figure: Micro USB Cable

- **PLA material** - for 3D printing.

Figure: Material used for 3D printing

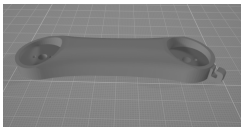
IV. Hardware Implementation:

3D Printed Arm Structure

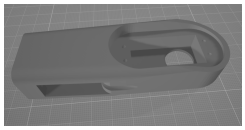
To construct the physical body of the robotic arm, we designed and fabricated custom parts using 3D printing (we took some inspiration from the internet about the design of the robotic arm). This allowed us to precisely tailor the dimensions and geometries of each joint and link to suit the servo motors and overall movement requirements.

The components were modeled using Blender software and printed using PLA filament due to its ease of use, adequate strength, and cost-effectiveness. We ensured appropriate infill density (around 40–60%) to provide a balance between structural integrity and material usage. Additionally, support structures were used for overhanging parts, and care was taken to orient the parts to minimize warping and improve layer adhesion.

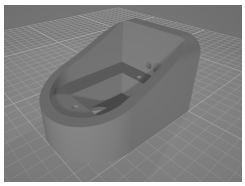
This approach gave us the flexibility to iterate on designs quickly and assemble a lightweight yet durable robotic structure.



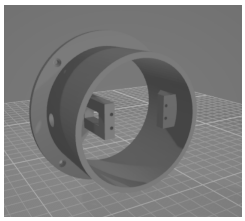
Arm part-1



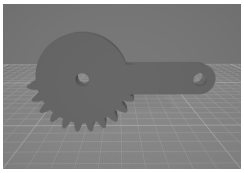
Arm part-2



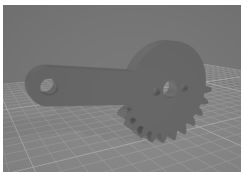
Arm part-3



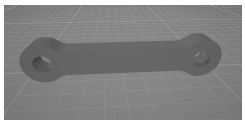
Base



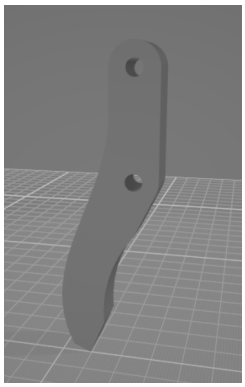
Gear-1



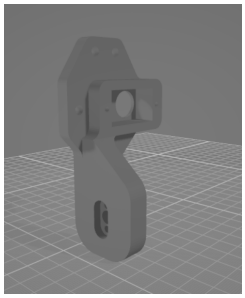
Gear-2



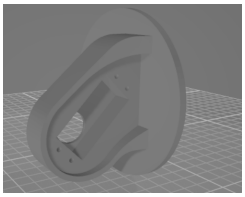
Grip-Link



Gripper



Gripper-Base



Waist

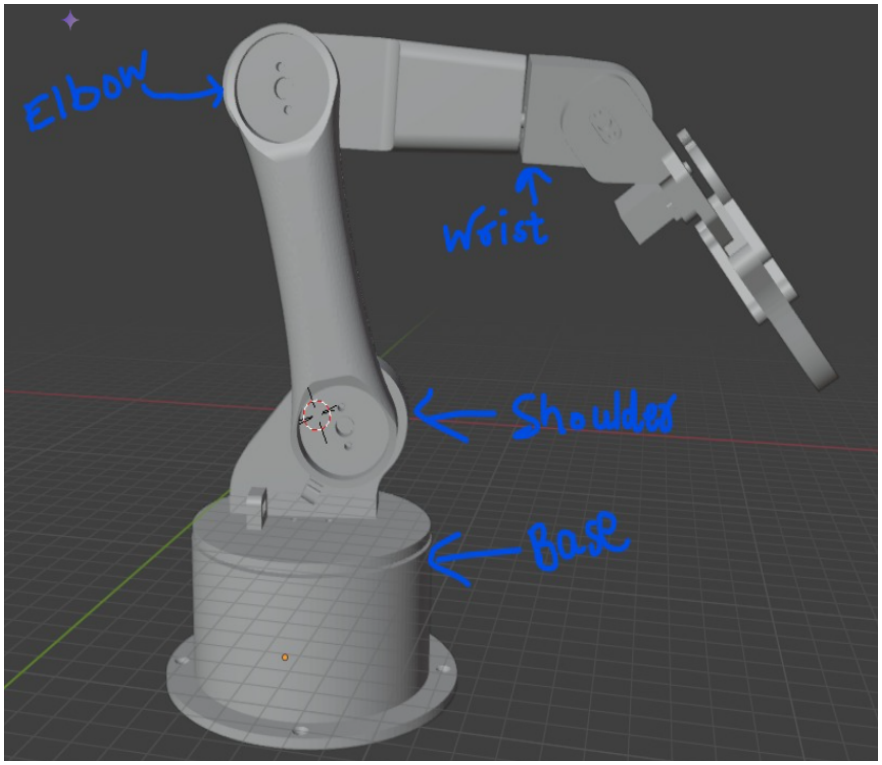


Figure: Assembled Robotic Arm

Circuit Diagram

The circuit diagram of the robotic involves connections between the ESP32 development board and the servo motors, along with the necessary power supply of 5V of voltage and approximately 5Amp of currevt and signal routing. Each of the servo motors is connected to PWM-capable GPIO pins of the ESP32, and powered through an external power source to ensure sufficient current supply. The signal wires of the servos are interfaced with the

ESP32 GPIOs, while the ground of all components is commonly connected to maintain proper reference.

Additionally, care was taken to avoid current surges by including appropriate decoupling capacitors and ensuring wire gauge selection suited for power distribution. The Wi-Fi module on the ESP32 allows wireless commands to be sent to the system, based on which the control logic directs the servo positions.

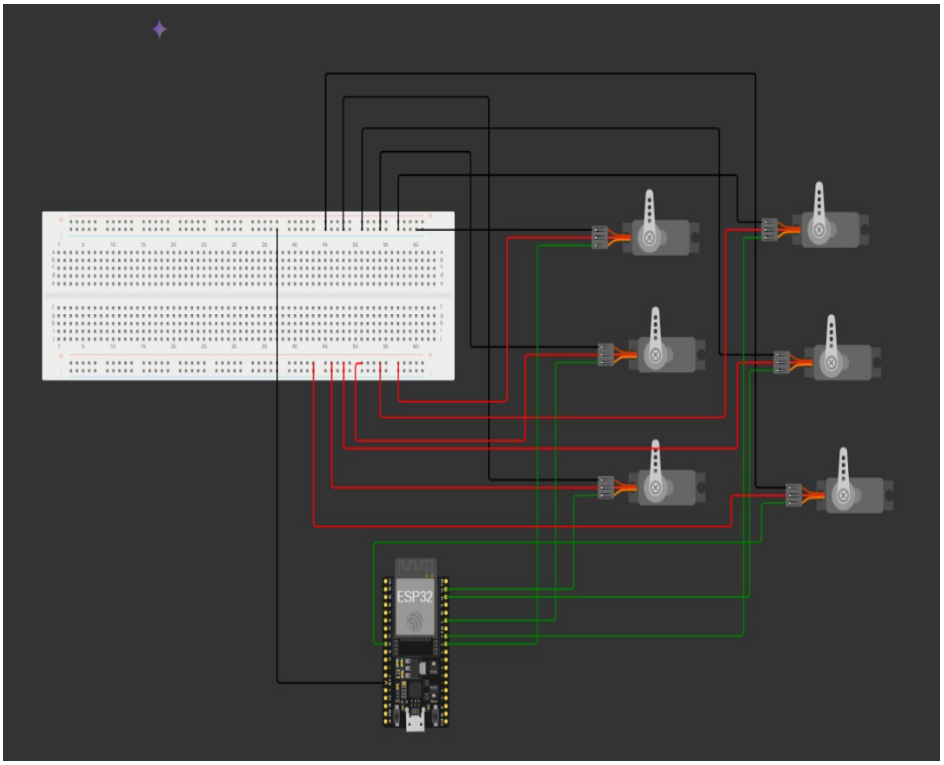


Figure: Circuit Diagram of the Robotic Arm System

v. Software Implementation:

Adrunio IDE

To program the ESP32 microcontroller, we used the Arduino IDE due to its ease of use and support for external boards like the ESP32. The ESP32 board was added through the Board Manager by including the following URL in the preferences:

The main functionality was developed using the `WiFi.h` library, which allows the ESP32 to connect to a Wi-Fi network and communicate wirelessly. This enabled remote control of the robotic arm for operations like positioning and movement commands. Additionally, the `Servo.h` library was used to control the servo motors responsible for the robotic arm's movement.

Our implementation focused on 3 Degrees of Freedom (DOF)—base, shoulder, and elbow. Apart from basic pick-and-place functionality, we also integrated a **"Save and Repeat"** feature that allows the system to record a sequence of servo positions and repeat them on command. This introduces a basic level of automation, closely mimicking repetitive industrial tasks.

Required Libraries:

- `WiFi.h` – Enables Wi-Fi connectivity on the ESP32.
- `Servo.h` – Provides easy control of servo motor angles.

This software framework allowed us to control the arm wirelessly, implement automation, and explore real-world robotic control strategies in an academic setting.

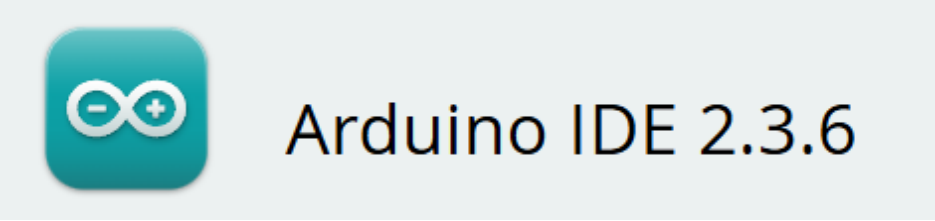


Figure: Arduino IDE used for programming ESP32

Blender

To design the mechanical structure of the robotic arm, we used **Blender**—a powerful open-source 3D modeling software. Blender allows for precise modeling of mechanical parts with accurate dimensions, making it ideal for prototyping and functional 3D prints.

In our project, the joints, base, and links of the robotic arm were modeled using Blender. These models were exported in STL format, which is compatible with most slicing software used for 3D printing. Blender's ability to visualize motion and joint rotation also helped in validating the mechanical feasibility of our design before physical printing.

We took inspiration from open-source robotic arm models available online and customized the structure based on our project's mechanical requirements. Additionally, we received valuable input from our peers in the **Design Department**, who guided us in modeling precise and robust parts.

For the actual printing process, we used a 3D printer with the help of the **Robotics Club** members. They also advised us on the selection of suitable materials (such as PLA or PETG) for better strength and durability of the printed components.

Figure: Blender - Used for Designing the 3D parts.

Code - Complete Breakdown

Library Inclusions and Servo Initialization

—

```
#include <WiFi.h>
#include <ESP32Servo.h>
#include <math.h>

// Servo setup
Servo servo1, servo2, servo3, servo4, servo5, servo6;
const int servoPin1 = 18, servoPin2 = 19, servoPin3 = 21, servoPin4 = 22, servoPin5 = 23, servoPin6 = 25;
int def1 = 90, def2 = 45, def3 = 135, def4 = 90, def5 = 60, def6 = 120;
```

Figure: Working Model of our Project.

The code begins by including essential libraries:

- `WiFi.h`: Enables Wi-Fi connectivity for the ESP32.
- `ESP32Servo.h`: Allows PWM control over servos using the ESP32's GPIO pins.
- `math.h`: Enables use of mathematical functions needed for inverse kinematics.

It then defines six `Servo` objects corresponding to six joints of the robotic arm. Each servo is mapped to a specific GPIO pin (e.g., `servoPin1 = 18`), and default angles are defined to bring the arm to a neutral position upon reset.

Wi-Fi Access Point Configuration

```
WiFiServer server(80);
const char* ssid = "ESP32_Servo_AP";
const char* password = "12345678";

// Mode
String mode = "manual";
```

Figure: Working Model of our Project.

The ESP32 is configured to act as a Wi-Fi access point (AP) using the SSID `ESP32_Servo_AP` and password `12345678`. A `WiFiServer` object is instantiated on port 80 to allow communication over HTTP. The mode of operation is set to `manual` by default, but can later be switched to `auto` (for inverse kinematics).

Teach and Repeat Mode and Inverse Kinematics Constants

```
// Teach and repeat
std::vector<std::vector<int>> recordedAngles;
bool isRecording = false, isRepeating = false;
unsigned long lastStepTime = 0;
const unsigned long repeatInterval = 1000;
int repeatIndex = 0;

// IK link lengths and gripper length
const float L1 = 100; // shoulder to elbow
const float L2 = 100; // elbow to wrist
const float L3 = 100; // wrist to gripper tip
const float SHOULDER_OFFSET_DEG = 20;
```

Figure: Working Model of our Project.

The robot arm supports a 'teach and repeat' mode. A vector named `recordedAngles` is used to store sequences of joint angles. Two Boolean flags, `isRecording` and `isRepeating`, track whether the arm is recording positions or replaying them. `repeatInterval` controls the time delay between each step during replay.

Setup Function

```
void setup() {
  Serial.begin(115200);
  WiFi.softAP(ssid, password);
  server.begin();
  Serial.println(WiFi.softAPIP());

  servo1.attach(servoPin1); servo2.attach(servoPin2); servo3.attach(servoPin3);
  servo4.attach(servoPin4); servo5.attach(servoPin5); servo6.attach(servoPin6);

  resetServos();
}
```

Figure: Working Model of our Project.

The `setup()` function initializes:

- Serial communication at 115200 baud for debugging.
- Wi-Fi access point using the configured SSID and password.
- Starts the HTTP server.
- Attaches each servo object to its respective GPIO pin.
- Calls `resetServos()` to bring the robotic arm to its default position.

Link lengths for each arm segment are defined as `L1`, `L2`, and `L3`. An offset angle `SHOULDER_OFFSET_DEG` accounts for physical misalignment or calibration errors in the shoulder joint.

```
client.println("<script>");
client.println("function send(){");
client.println("let url = '/?servo1=' + s1.value + '&servo2=' + s2.value + '&servo3=' + s3.value + '&servo4=' + s4.value + '&servo5=' + s5.value + '&servo6=' + s6.value;");
client.println("location.href = url; }");
client.println("function reset(){ location.href='/reset'; }");
client.println("function startRec(){ location.href='/recordStart'; }");
client.println("function stopRec(){ location.href='/recordStop'; }");
client.println("function repeat(){ location.href='/repeat'; }");
client.println("function ik(){");
client.println("let x = document.getElementById('x').value;");
client.println("let y = document.getElementById('y').value;");
client.println("let z = document.getElementById('z').value;");
client.println("location.href = '/?x=' + x + '&y=' + y + '&z=' + z;");
client.println("}</script>");
```

JavaScript-Based Web Control Interface

The HTML page served by the ESP32 includes embedded JavaScript code that enables real-time control of the robotic arm from a browser-based interface.

- **sendData(servo, angle)**: This function is triggered whenever a user adjusts a servo slider. It sends an HTTP GET request with parameters `servo` and `angle` to the ESP32 (e.g., `/?servo=2&angle=90`).
- **sendPosition()**: Used in inverse kinematics mode. It captures the (X, Y, Z) coordinates from input fields on the web page and sends them as parameters to the ESP32 (e.g., `/?x=50&y=20&z=30`).
- **XMLHttpRequest API**: This API facilitates asynchronous communication with the ESP32 server without refreshing the page. It ensures smooth, real-time updates as the user interacts with the interface.

Web Interface and HTML Logic

The ESP32 board hosts a web interface using HTML and JavaScript, accessible over a Wi-Fi network. The page provides two modes: **Manual Mode** and **Auto Mode**.

- **Mode Switching:** Buttons allow the user to switch between manual and auto control modes using URL parameters such as `/?mode=manual`.
- **Manual Mode:** Six range sliders (one for each servo motor) allow real-time control. Adjusting a slider triggers the `send()` JavaScript function, which sends the updated angles via a URL query string.
- **Auto Mode:** The user can input desired Cartesian coordinates (X, Y, Z), and clicking the "Go" button invokes the `ik()` function to request inverse kinematics computation.
- **Control Buttons:** The interface includes buttons for resetting the arm, recording servo states, and executing repeat actions. These buttons trigger predefined URLs like

`/reset`, `/recordStart`, etc.

JavaScript functions such as `send()` and `ik()` dynamically construct URLs and redirect the browser to communicate with the ESP32 server. This enables intuitive and wireless control of the robotic arm through a minimal yet functional interface. The complete source code for the ESP32-based 6-DOF robotic arm, including the web interface, servo control logic, inverse kinematics computations, and teach-and-repeat functionality, is available on my GitHub repository. You can evaluate the whole code by visiting: https://github.com/RishiTirpude19.

VI. Working and Results

After assembling the mechanical structure and successfully uploading our program to the ESP32 using the Arduino IDE, the robotic arm was tested for basic pick and place operations. The system was wirelessly controlled via a web interface, which consisted of sliders for adjusting the position of each joint. These sliders performed well during testing and provided smooth manual control over the robotic arm's motion.

However, during multiple trials, we observed certain limitations and areas for improvement. One significant issue was with the **gripper mechanism**. Due to low friction between the gripper surface and the object being picked, there were instances where the object slipped out, especially when the joints came to a sudden halt. This was likely caused by the abrupt transfer of force through the arm's body, making it difficult for the gripper to retain the object securely. This could be improved in the future by adding rubber padding or a more adaptive gripping surface to increase friction and grip reliability.

We also implemented a basic version of **Inverse Kinematics (IK)** for 3 degrees of freedom—specifically for the base, shoulder, and elbow joints. Users could input a coordinate point in 3D space, and the robotic arm would attempt to reach it. While the implementation worked, it lacked precision and exhibited minor errors in final positioning, especially at certain angles or distances. With more time, we would have fine-tuned the kinematic equations and constraints to enhance the accuracy of this functionality. Nevertheless, the current IK system still managed to demonstrate the concept effectively and offered a good degree of movement control.

Another notable feature was the **Save and Repeat** function. This allowed users to record a sequence of servo positions and replay them later. During testing, this functionality worked reliably and added a layer of automation to our system. The robotic arm was able to repeat previously executed motion sequences with consistent performance, mimicking basic task repetition as seen in industrial environments.

Overall, the project met its objectives for the Control Systems course and provided practical insights into the challenges of robotic motion, control systems, and real-world mechanical limitations. The collaborative efforts in design, coding, and testing significantly contributed to the successful demonstration of a functional 6 DOF robotic arm with wireless control and partial automation.

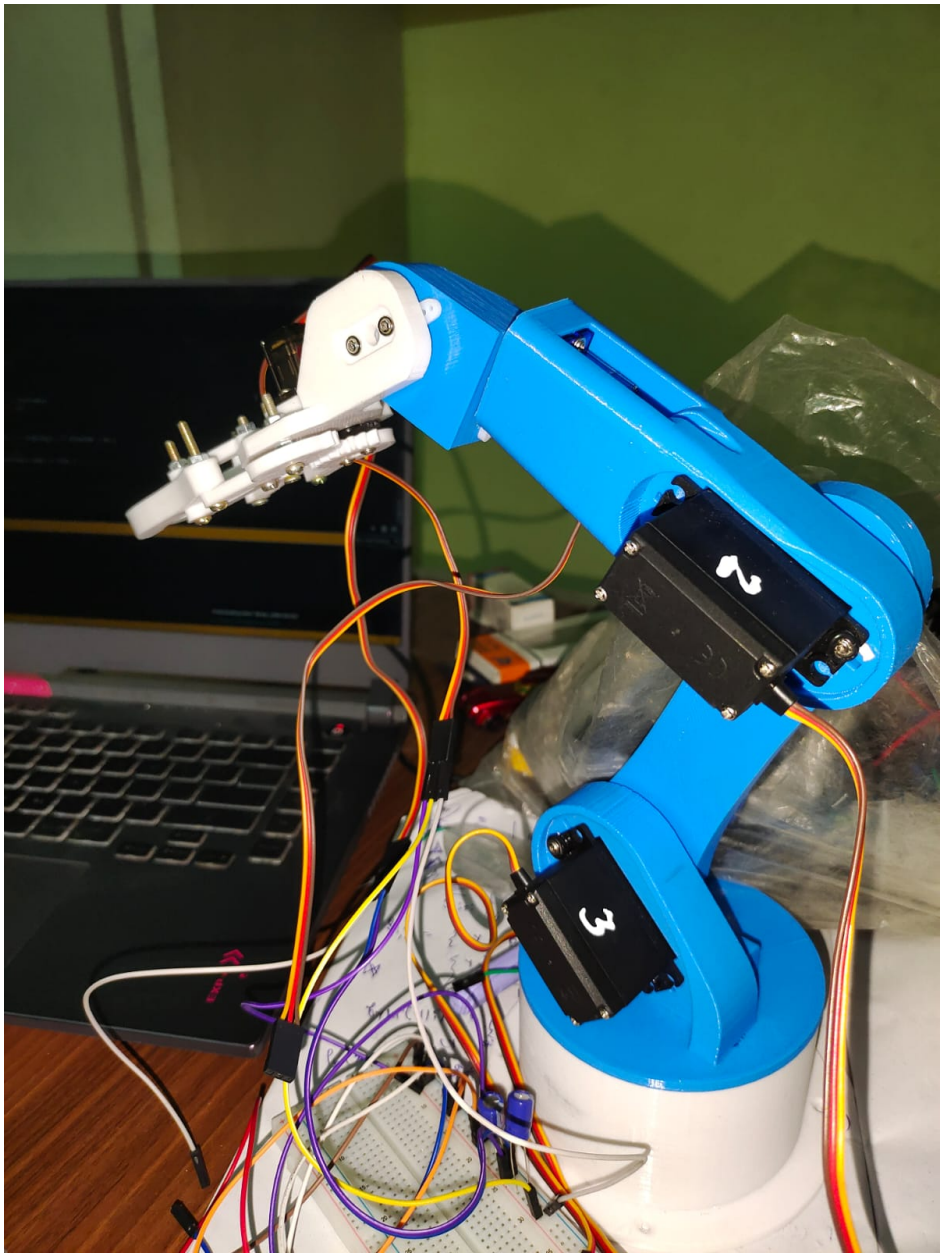


Figure: Working Model of our Project.

VII. Difficulties Faced

Intial use of Arduino

Initially, we began prototyping the robotic arm using the **Arduino Uno** board, as it is a beginner-friendly and widely used microcontroller. However, we soon encountered major limitations related to wireless communication. The Arduino Uno does not have any built-in

wireless modules such as Wi-Fi or Bluetooth, which made it difficult to implement a seamless and responsive remote control interface.

To overcome this, we attempted to integrate an external **Bluetooth module (HC-05)** with the Arduino Uno. While it did allow basic communication between the robotic arm and a Bluetooth-enabled device, the connection proved to be unstable. Frequent disconnections and lag in command response made it unreliable for real-time control of the robotic arm, especially for delicate pick and place operations.

As a result of these limitations, we decided to switch to the **ESP32 Dev Module**, which comes with an inbuilt Wi-Fi module. This change allowed us to create a stable wireless communication link between the web-based slider interface and the robotic arm. With Wi-Fi control, the commands were transmitted more reliably and with lower latency, significantly improving the overall user experience and functionality of the system.

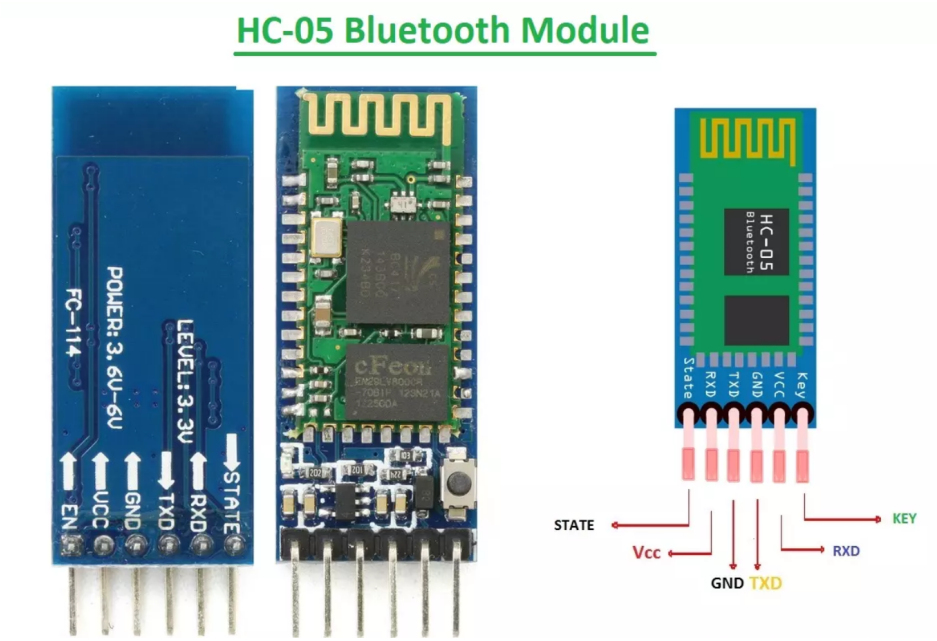


Figure: HC-05 Module.

Setting Tolerances for 3D Printed Parts

One of the major challenges we faced during the fabrication phase of the robotic arm was determining and setting the correct **tolerances for 3D printed parts**. Tolerance refers to

the permissible limit of variation in a physical dimension, and in the context of 3D printing, even a small mismatch can lead to parts not fitting together as intended.

Many components, especially joint mounts, servo brackets, and linkages, had to be assembled precisely. Initially, we encountered issues where holes were too tight for the servo shafts or joints had too much play, affecting the stability and motion of the robotic arm. Since 3D printers can vary slightly in their precision and materials like PLA or PETG expand or shrink slightly after printing, a lot of **trial and error** went into finding the right fit.

This tuning process consumed a significant amount of time, as it involved repeatedly modifying the models, reprinting parts, and testing their fit. With support from our peers in the Design Department and advice from the Robotics Club, we gradually learned to estimate and apply effective clearance values in the range of **0.2mm to 0.5mm**, depending on the type of joint or fit. These insights helped us improve the final assembly quality and structural integrity of the robotic arm.

Power Supply Challenges

During the initial testing of the robotic arm, we faced significant issues due to an **insufficient power supply**, particularly when all servo motors were operated simultaneously. The servo motors, especially the MG995s used for the base, shoulder, and elbow joints, require a considerable amount of current to function reliably under load. Our initial setup, which relied on USB or standard battery packs, was unable to deliver the required current, leading to problems such as servo jittering, random resets of the ESP32 board, and inconsistent arm movements.

To address this, we utilized the **regulated power supply available in the Embedded Systems Laboratory**, capable of providing a stable **5V and 5A output**. This significantly improved the performance and stability of the system, ensuring all motors received sufficient power even during high-load operations like lifting or rotating objects. With a stable power source, the robotic arm functioned smoothly, and the control logic became more predictable and reliable during testing and demonstrations.

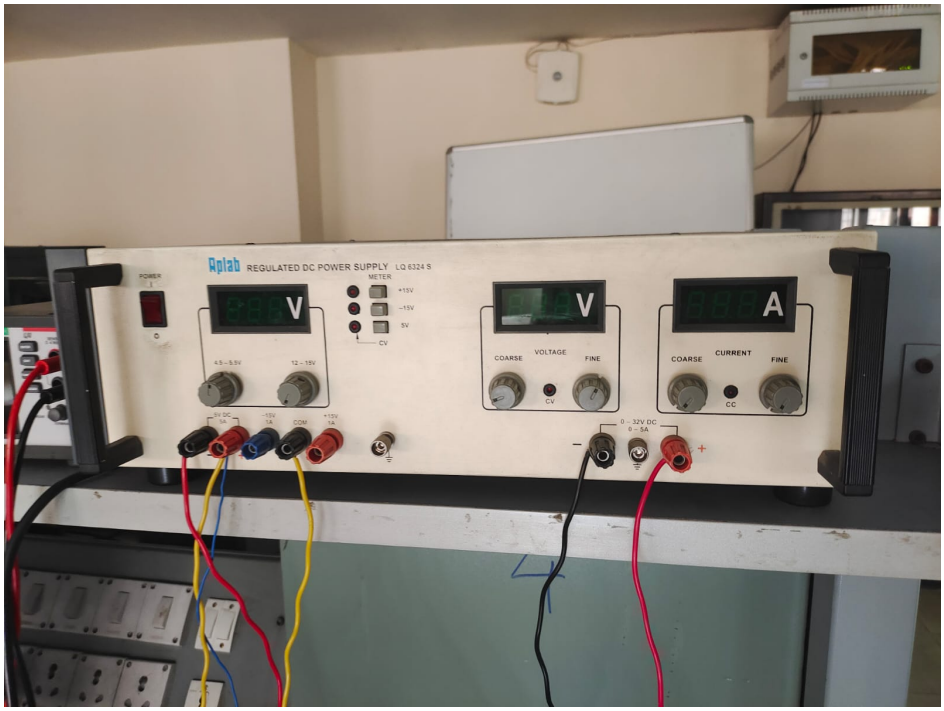


Figure: Power Supply.

VIII. Future Goals

While the current version of our 6 DOF robotic arm performs basic pick and place operations effectively, several improvements can be made to enhance its functionality, precision, and reliability.

One of the immediate goals is to ensure the **smooth and consistent movement of all joints**. Some joints currently experience slight lag or jerks due to either mechanical tolerance issues or fluctuations in power delivery. These can be minimized through better calibration, improved 3D printed part designs, and potentially upgrading to higher precision servo motors or incorporating closed-loop control mechanisms using encoders.

Additionally, our implementation of **inverse kinematics (IK)** is currently limited to 3 degrees of freedom—covering the base, shoulder, and elbow joints. While functional, it lacks the precision required for tasks that demand high accuracy. In the future, we aim to refine this 3 DOF IK logic by incorporating better mathematical models and smoothing algorithms to reduce positional error. With more in-depth study and understanding, we plan to extend inverse kinematics to cover all 6 degrees of freedom, enabling more versatile and complex movements.

To bring this robotic arm closer to industrial standards, several structural and functional upgrades can be explored:

- Replacing servo motors with **stepper motors or brushless DC motors** for higher torque and smoother motion.
- Implementing **feedback systems** using sensors or encoders for precise control.
- Using more **durable materials** such as aluminum or carbon fiber for structural parts to withstand prolonged use and heavy loads.
- Integrating with **computer vision systems** or **machine learning models** for intelligent automation.
- Adding **force feedback and torque sensing** capabilities to handle fragile or variable-weight objects safely.

By addressing these goals, we hope to evolve our prototype into a more robust and intelligent robotic system, suitable for real-world industrial or research applications.

IX. References

- **ESP32 Documentation:** https://docs.espressif.com/projects/esp-idf/en/latest/esp32/
- **Arduino IDE Setup for ESP32:** https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/
- **Servo Motor Basics:** https://www.electronics-tutorials.ws/blog/servo-motors.html
- **Inverse Kinematics Explained:** https://motion.cs.illinois.edu/RoboticSystems/InverseKinematics.html
- **Blender for 3D Printing:** https://www.blendernation.com/2020/01/27/using-blender-for-3d-printing/
- **MG995 Servo Datasheet:** https://components101.com/motors/mg995-servo-motor
- **Web interface for controlling servos:** https://randomnerdtutorials.com/esp32-servo-motor-web-server-arduino-ide/
- **Youtube videos:** DIY TechRush