An Intelligent 4-Axis Robotic Arm with Dual Interface Control

A Project Report Submitted in partial fulfilment for the Award of the Diploma in Electronics (Communication) Engineering

Submitted By:

SUNIL KUMAR (10922232)

MOHD AMIR (10922194)

Under the Guidance of:

Mr. Anuj Kalra

CERTIFICATE

The project report, " An Intelligent 4-Axis Robotic Arm with Dual Interface Control," authored by Sunil Kumar (10922232), Mohd Amir (10922194) has been approved and certified as a commendable study in technological subjects. It fulfils the requirements for the partial fulfilment of the degree for which it is submitted.

It should be noted that by granting this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed, or conclusion drawn within the report. The approval is granted solely for the purpose for which it is submitted.

Mr. Anuj Kalra

Lecturer

……

Mrs. Shivani Neb Head of Department
(Electronics)

…….

DECLARATION

We, Sunil Kumar (10922232), Mohd Amir (10922194) hereby declare that the project work titled " An Intelligent 4-Axis Robotic Arm with Dual Interface Control" submitted to DSEU Pusa campus, represents original work completed by us under the guidance of Mr. Anuj Kalra, Lecturer, DSEU Pusa campus.

This project work is being submitted as part of the requirements for the diploma in Digital Electronics Engineering

The findings presented in this project have not been previously submitted to any other University or Institute in pursuit of any degree or diploma.

SUNIL KUMAR (10922232)

MOHD AMIR (10922194)

ACKNOWLEDGEMENT

It is indeed a great privilege for us to convey our sincere gratitude to our esteemed teacher, Mr. Anuj Kalra, Electronics Engineering, DSEU PUSA CAMPUS (formerly known as Pusa Institute of Technology), for granting us the opportunity to undertake this project. His consistent guidance, valuable suggestions, supervision, and inspiration throughout the course work were indispensable in enabling us to complete the work within the scheduled time.

We would also like to seize this moment to express our gratitude to all the esteemed teachers of this department for their constant inspiration and guidance, leading us along the right path in times of need.

SUNIL KUMAR (10922232)

MOHD AMIR (10922194)

ABSTRACT

The project titled **“An Intelligent 4-Axis Robotic Arm with Dual Interface Control”** presents a cost-effective and scalable solution for industrial automation, tailored to address the limited availability of accessible robotic technologies in India. Designed to operate within a factory-like environment featuring conveyor belts and item-based workflows, this system simulates real-world industrial tasks with precision and flexibility.

The core of the system is a custom-built ESP12F-based console, which integrates an SSD1306 OLED display and three-button interface. The console runs a custom firmware with a bitmap-enhanced UI, enabling an intuitive, visually rich user experience directly on the device. In addition to the hardware interface, the robotic arm also offers a web-based control panel, allowing remote control of each axis and access to preprogrammed motion sequences from any browser-enabled device.

With its **dual interface architecture**, this robotic arm delivers both physical and remote-control capabilities, empowering users to perform complex operations effortlessly. The system serves as a prototype for affordable industrial automation in small-scale manufacturing setups, educational environments, and R&D applications, aiming to inspire the broader adoption of robotics in India’s evolving technological landscape.

Content

Chapter 1: Introduction

The increasing demand for affordable and efficient automation in India has driven the need for innovative solutions that are both technically feasible and cost-effective. While industrial automation has advanced globally, many small and medium enterprises in India still lack access to practical robotic systems due to high costs and complex implementations’

This project, titled **"An Intelligent 4-Axis Robotic Arm with Dual Interface Control,"** aims to bridge this gap by introducing a compact and smart robotic hand with four degrees of freedom. Designed for a simulated factory environment, the arm operates in sync with moving conveyor belts, capable of handling item-based tasks efficiently.

The core feature of the system is its **dual control interface**—a physical console and a web interface. The console is built using the **ESP12F microcontroller**, embedded with an **SSD1306 OLED display** and **three physical buttons**. This setup allows users to interact with the system through a neatly designed graphical UI with bitmap icons, offering an intuitive control experience. Alongside, the **web interface** enables remote access and control of individual axes and predefined movements, making the system highly flexible.

1.1 Background and Motivation

In recent years, student projects have often leaned toward reusing existing online code, pre-built modules, and sensor combinations that offer minimal innovation. While this approach ensures ease of execution, it limits creativity, technical growth, and true engineering application.

Our motivation stemmed from a desire to break this pattern.

Instead of following the common route, we aimed to build something truly original—both in **hardware and software**. We didn't want to just combine sensors and use premade code; we wanted to **design, build, and understand** every part of the system ourselves. From designing our own **custom firmware** to developing a dedicated **microcontroller console** and a **graphical interface**, every aspect of this project was developed from scratch.

The challenges we faced during development—whether in firmware logic, PCB design, or mechanical structure—became our source of motivation. Solving each obstacle gave us the confidence that we were doing something meaningful.

Ironically, now that the project is complete, we're quite certain future students might be tempted to **copy our work**—and in a way, that itself proves that we created something worth replicating.

1.2 Objective of the Project

The objective of this project is to develop a smart robotic arm system equipped with a dual interface architecture that allows both local and remote-control functionalities. The design focuses on offering an intuitive console-based user interface via a 0.96-inch I2C OLED display, along with network connectivity through the ESP8266 module to enable wireless interaction over a backend server.

This dual interface architecture ensures that the system remains fully operational even without internet access by enabling local control through onboard buttons, while also allowing remote commands via HTTP requests when networked. Such flexibility makes the system highly adaptable for educational, industrial, and prototyping environments.

The project aims to demonstrate an embedded solution that combines hardware-level motor control with software-level communication logic, offering precise real-time manipulation of the robotic arm. By integrating local feedback and network-driven control, the system delivers a seamless experience without compromising reliability or responsiveness.

System Overview & Block Diagram

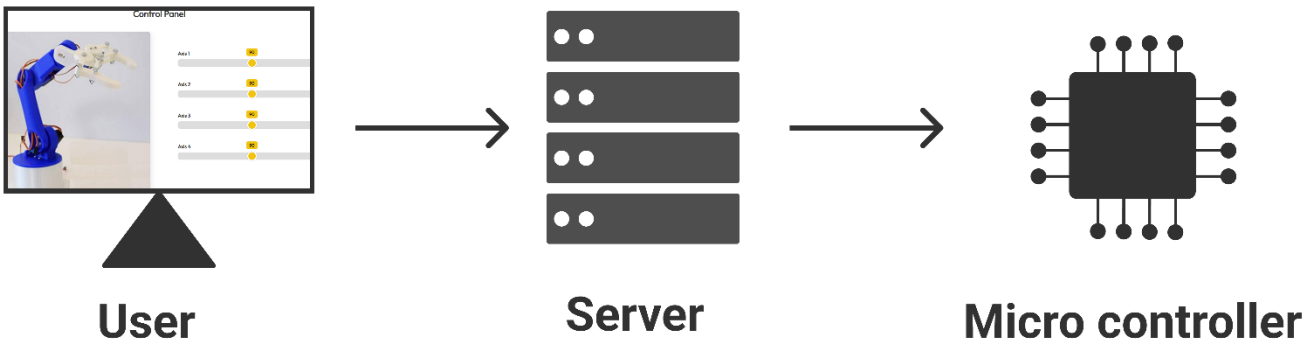
2.1 System Overview

This major project involves the design and development of a factory-themed 4-axis robotic arm aimed at advancing industrial automation in India, where such technology is still emerging. The robotic system is equipped with dual control interfaces: a physical OLED-based console and a web-based control panel. This allows users to either interact locally or remotely, enhancing both accessibility and flexibility.

The heart of the system is a custom-built **ESP12F Console**, a dedicated microcontroller board that integrates an SSD1306 OLED display, three control buttons, and essential GPIOs required for axis control. The console runs a tailor-made firmware that not only manages motion control but also features bitmap image rendering on the OLED, delivering a visually engaging user experience.

For local control, commands are executed directly through button interactions and displayed instantly on the OLED screen. For web-based interaction, the system uses the onboard ESP12F as a standalone server. The web interface sends HTTP GET requests to the ESP module, which interprets and performs the required operations on the robotic arm. This setup supports both real-time control of each axis and the execution of preprogrammed motion sequences.

The entire system simulates a miniaturized smart factory setup, complete with conveyor belts and item handling operations. It serves as a functional demonstration of how embedded automation systems can be implemented in small-scale industries, educational labs, or prototyping environments, especially in developing regions.



2.1 Block Diagram for Web control

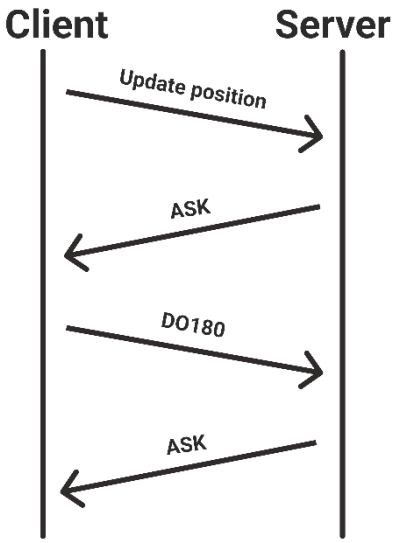
2.2 Web-Based Control System

In the web interface model, the **ESP12F microcontroller acts as both the server and actuator controller**. The user interacts with a web panel hosted on the ESP itself. This panel allows control over each robotic axis using sliders and provides access to predefined motion sequences. When a command is issued, the panel sends a **GET request** to the server (ESP12F), which then translates the instruction into a motor movement.

The communication pipeline is shown in the block diagram Above

2.3 Web-Based Control System

The diagram below represents the interaction between the client (web interface) and the ESP12F server. This **sequence diagram** outlines how the client sends HTTP GET requests to issue commands like position updates or specific actions (e.g., DO180). The server processes these requests in real-time and updates the robotic arm accordingly.



2.2 sequence diagram of web control communication

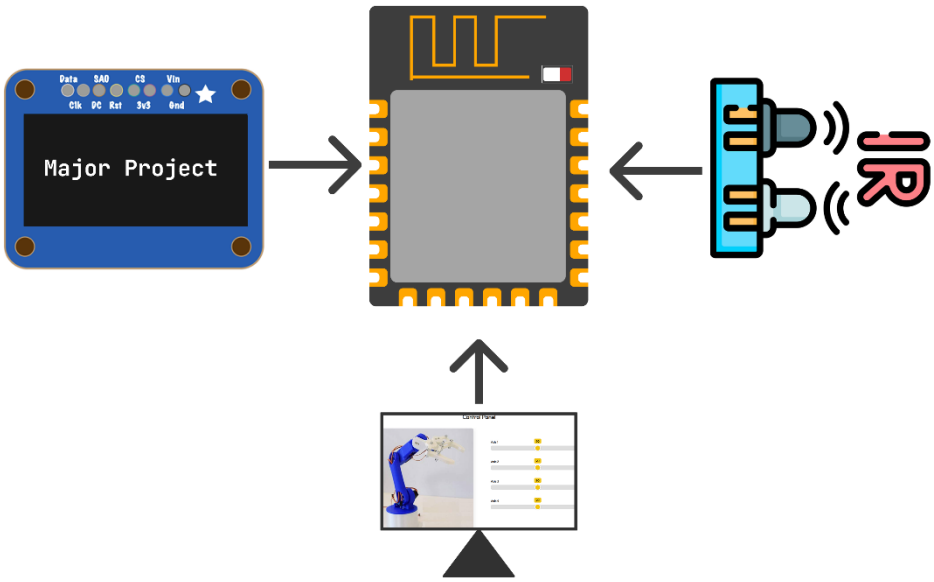
This is a **sequence diagram of web control communication** between a client and a microcontroller-based server (like your ESP12F).

Interaction Flow:

1. **Update Position** – The client sends a command to update the robotic arm's position.
2. **ASK** – The client makes a query, possibly checking the current state or requesting confirmation.
3. **DO180** – A specific command (e.g., rotate the servo or arm by 180°) is sent to the server.
4. **ASK** – Another query is sent, likely to verify if the command was executed.

2.4 – Hardware Architecture

The hardware architecture is designed around the ESP12F microcontroller, which acts as the brain of the system. It receives input from both the local console (equipped with an OLED display and buttons) and a web-based control panel. The ESP12F processes these inputs and controls the robotic arm accordingly. Additionally, IR sensors provide environmental feedback, enhancing interactivity and automation.



2.2 Hardware Architecture

2.5 Software Architecture

The software architecture of our system follows a modular and event-driven approach. It consists of three primary components: the microcontroller firmware, the web-based control interface, and the backend logic responsible for command handling.

The firmware running on the ESP12F handles sensor inputs, display updates, and motor controls. The web interface acts as a front-end dashboard for real-time control, sending user inputs to the backend. The backend processes these commands and communicates with the ESP12F via HTTP requests. This layered architecture ensures smooth communication between the user and the robotic arm, both locally and remotely.

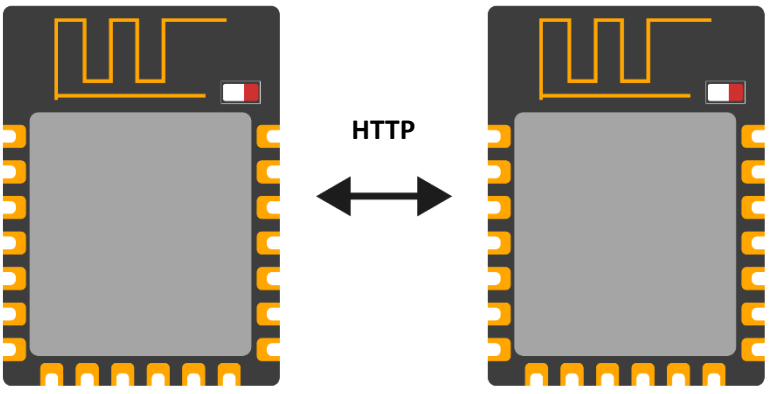
A detailed explanation of each module and its interconnections is provided in the Software Architecture chapter.

2.6 Software Architecture

To enable seamless control of the robotic arm without burdening the main controller, we implemented **ESP-to-ESP communication** using HTTP-based requests. This setup allows one ESP module (designated as the console unit) to function as a dedicated **remote controller with a display interface**, while the second ESP module (the main controller) handles all actuator and system operations.

Why This Approach?

1. **Resource Efficiency**: By offloading the display and user input logic to a separate ESP, the main controller can focus purely on real-time robotic control.
2. **Decoupling Logic**: Separating the UI from the actuator logic ensures modularity and easier debugging.
3. **Ease of Integration**: ESP8266/ESP32 boards natively support HTTP server/client functionalities, making this architecture simple to implement using readily available libraries.



2.3 ESP-ESP- communication

Communication Method

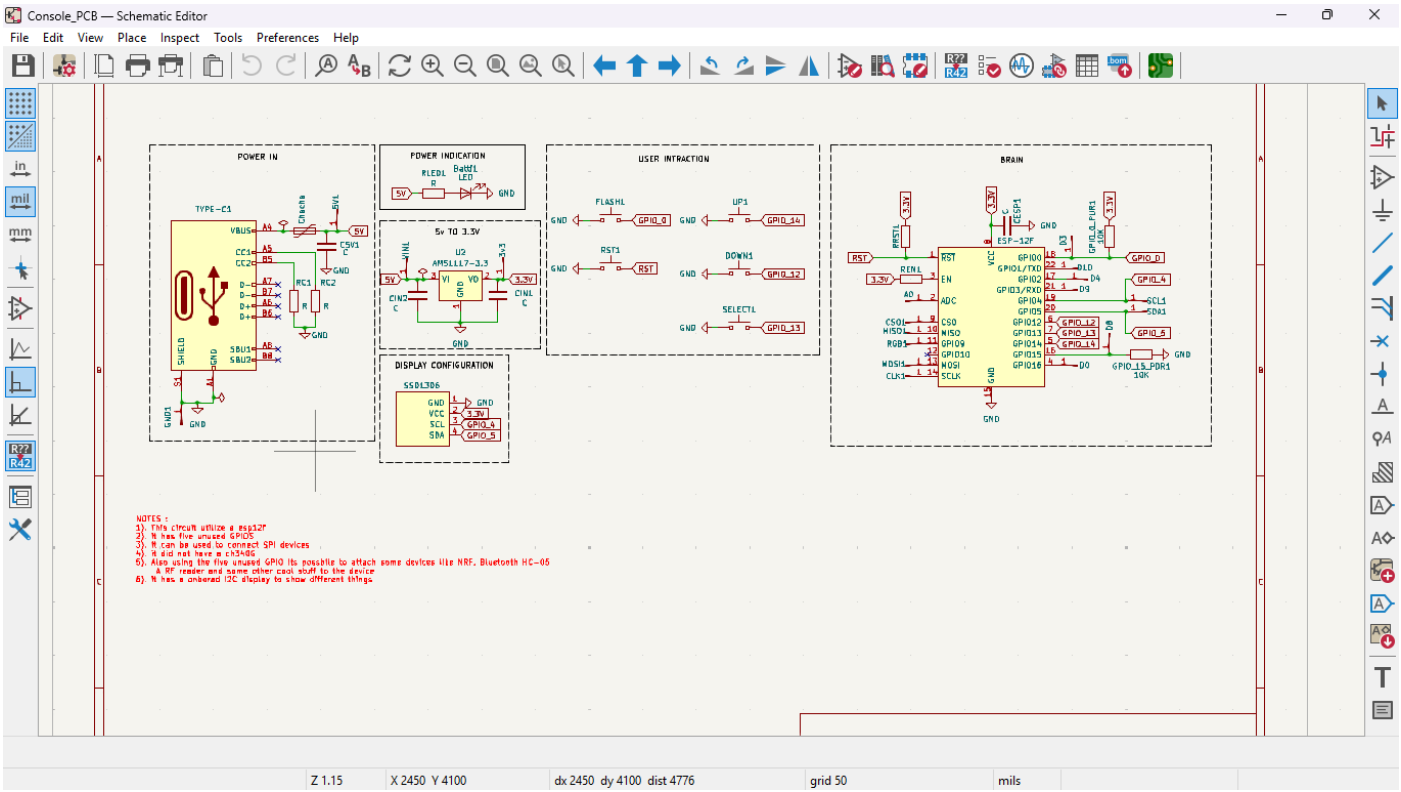
- The console ESP sends HTTP GET requests to specific endpoints hosted by the main ESP's web server.
- These endpoints trigger pre-defined functions on the main ESP, such as adjusting servo angles or initiating a movement routine.
- For example, a command like http://192.168.43.69/Preset?func=Do180

Chapter-3 Hardware Design

3.1 Designing Our Own ESP Console

So here's how the story goes — we searched the *entire market* looking for a board that ticks all the boxes. We needed a microcontroller that had its own display, looked aesthetically cool, came with onboard buttons, supported both I2C and SPI, delivered power through Type-C, and had enough GPIOs to handle our use case.

We couldn't find a single board that had *everything* we wanted.

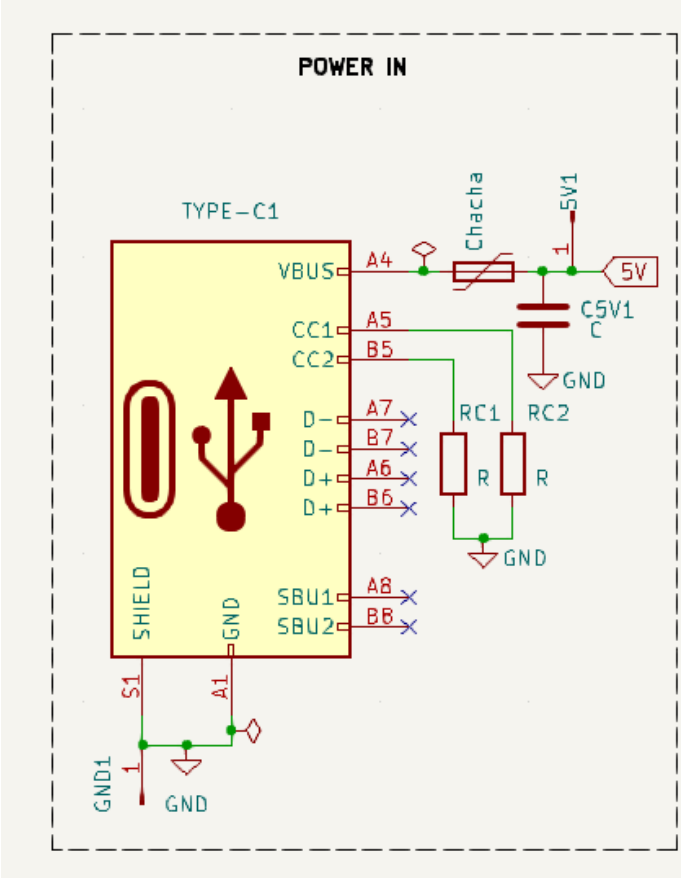


3.1 Schematics

3.2.1 Power In Block – Type-C Power

The Power In Block is the entry point for electrical power in the robotic arm system. It utilizes a USB **Type-C connector** to ensure universal compatibility and reliable power delivery. This block is designed with essential safety and filtering components to protect and stabilize the downstream electronics.

To support **Type-C to Type-C** cable connections, the circuit includes **5.1kΩ pull-down resistors** on the **CC1 and CC2** lines, which signal the host to provide power. A **resettable fuse** acts as overcurrent protection, allowing safe operation and automatic recovery in case of faults. Additionally, **decoupling capacitors** are used to filter noise and provide a smooth voltage supply.



3.1 Power in

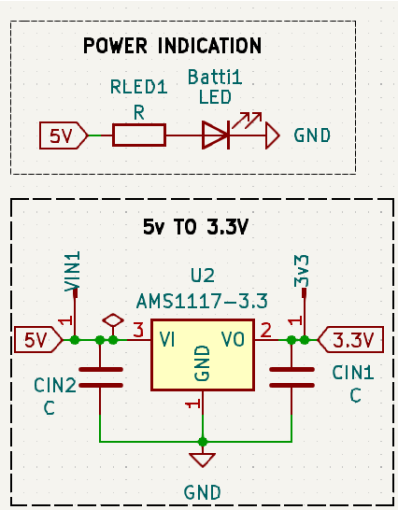
3.2.2 Power Conversion & status LED

To step down the 5V input voltage to 3.3V for the ESP-based console and display, the AMS1117-3.3 voltage regulator is used. This linear regulator provides a stable 3.3V output suitable for the microcontroller and other logic-level components.

Two ceramic capacitors are placed at the input and output of the AMS1117 to ensure voltage stability and suppress ripple. Typically, a 10µF capacitor is used on both sides for optimal filtering.

For power indication, an 0805 SMD LED is connected through a current-limiting resistor. The LED lights up when 5V is present, providing a simple visual confirmation that the board is receiving power. The use of compact 0805 metric components helps maintain a clean and space-efficient layout.

This block ensures that the circuit receives a regulated 3.3V supply and confirms power delivery through a dedicated indicator, improving both safety and user interaction.



3.2 Power Conversion and status LED

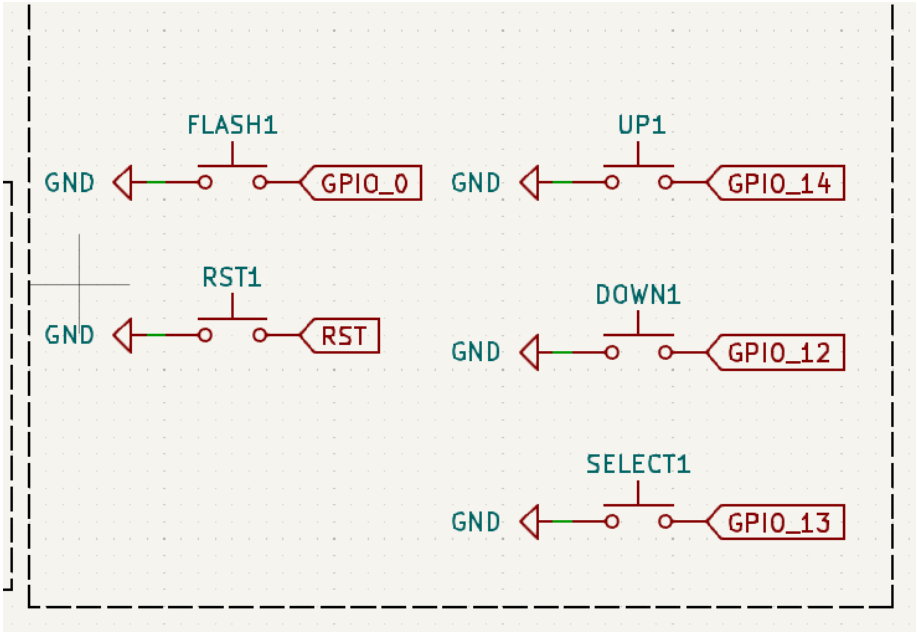
3.2.3 User Interaction Block

The user interaction block is designed to provide basic navigation and control options through tactile push buttons. A total of five buttons are used in this block, out of which three are dedicated for menu interaction and two are used for system-level functions.

The UP, DOWN, and OK buttons are connected to GPIO14, GPIO12, and GPIO13 of the ESP module. These buttons allow the user to navigate the on-screen interface shown on the OLED display. Each button is connected in a pull-down configuration using an external resistor to ensure reliable logic level detection.

In addition to these, there are two more buttons—RESET and BURN. The RESET button is directly connected to the reset pin of the ESP to manually restart the module. The BURN button is connected to GPIO0 and is used during firmware uploading. Pressing this button pulls GPIO0 to ground, enabling programming mode when the module is reset.

This block allows the user to interact with the system intuitively, enabling both firmware flashing and user-level control from a minimal console interface.

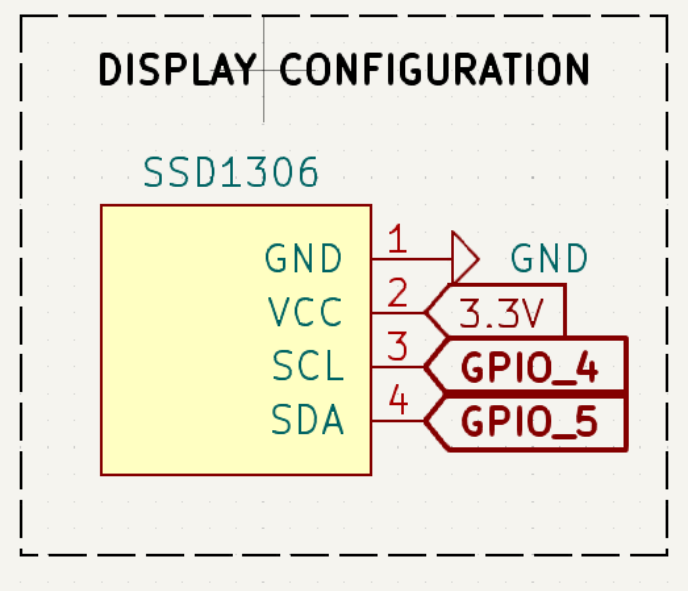


3.3 User Interaction

3.2.4 Display Block

The display block is built around a 0.96-inch I2C OLED screen based on the SSD1306 driver. This display is chosen due to its sharp pixel quality, good contrast, and low power consumption, making it ideal for compact embedded interfaces.

Being an I2C device, the display requires only two signal lines—SCL and SDA—along with power and ground. This greatly simplifies wiring and reduces the number of solder joints, making the assembly process easier and more reliable. In the system, SCL is connected to GPIO4 and SDA is connected to GPIO5 of the ESP module.



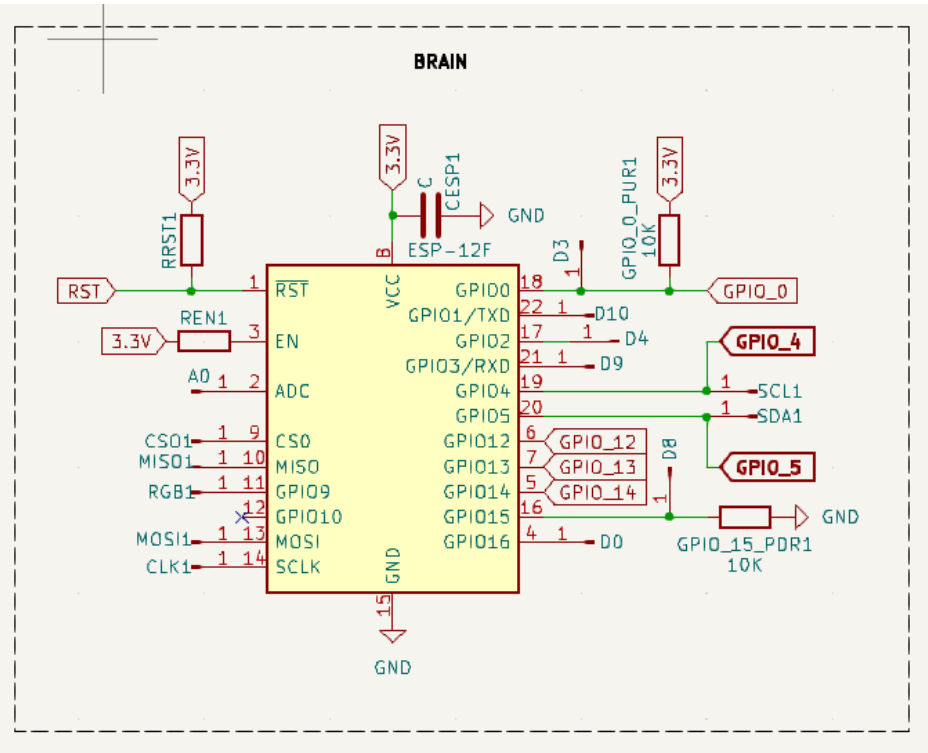
3.3 Display Configuration

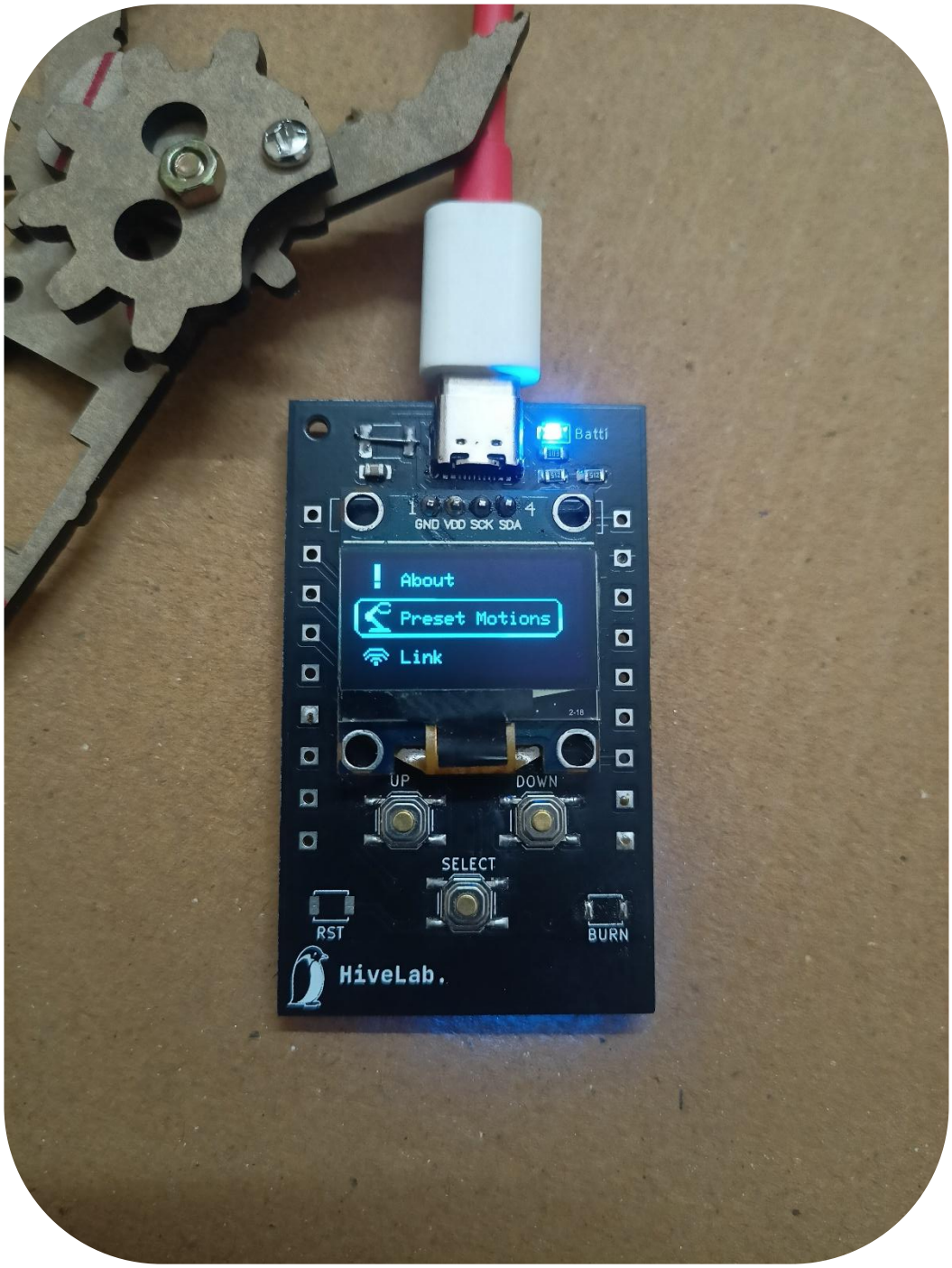
3.2.5 ESP12F Controller Block

At the core of the entire system lies the ESP-12F module, which serves as the main controller. It is responsible for handling user input, controlling the servos, managing the web interface, and displaying output on the OLED screen.

The ESP-12F is a compact Wi-Fi enabled microcontroller based on the ESP8266 chip. It offers sufficient GPIOs, stable wireless connectivity, and a low power footprint, making it suitable for both real-time control and web communication.

3.4 Brain





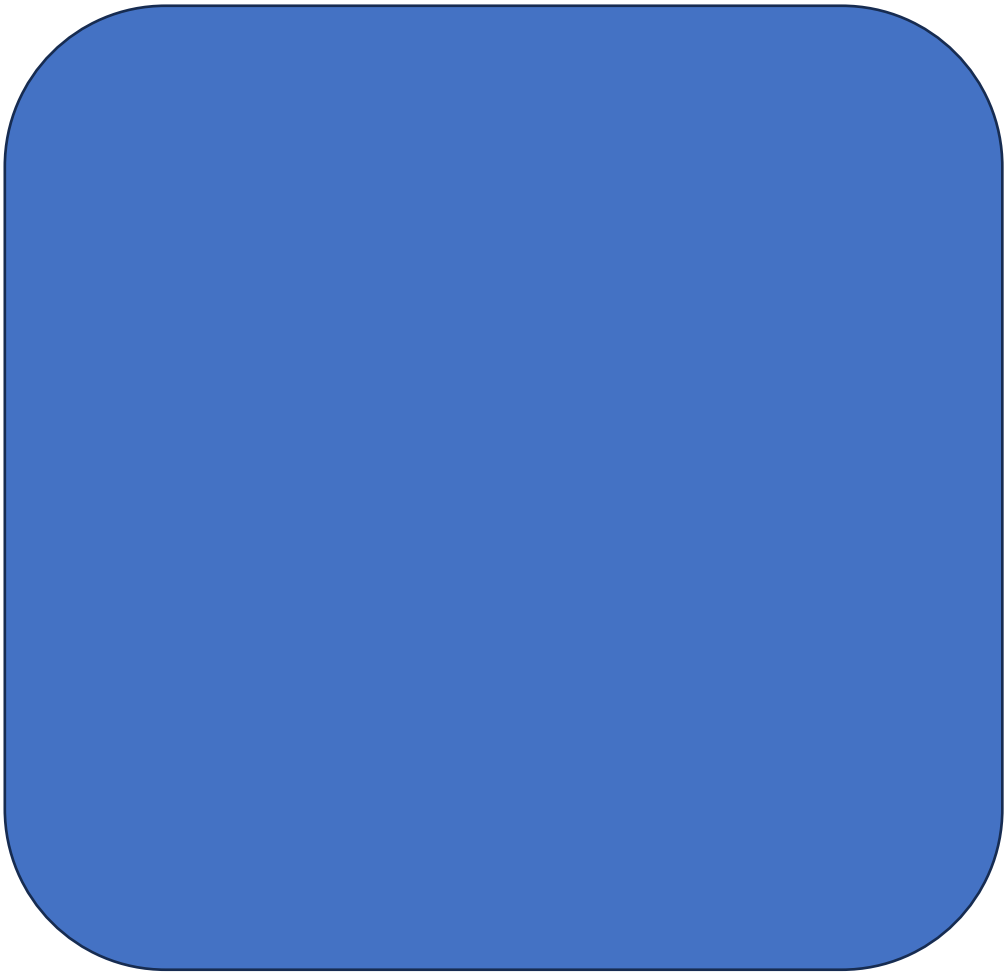
3.5 ESP-12F Console

3.2 Structure Design of Robotic Arm

The physical structure of the robotic arm was initially based on the open-source Merobot project. We had access to the 3D printable files, but due to limited resources and lack of access to a 3D printer, we decided to convert the 3D model into an equivalent 2D design using Fusion 360.

Our first plan was to use recycled PVC pipes to build the frame, keeping sustainability in mind. However, due to fitting issues and time constraints, we opted to get the more complex parts laser-cut from acrylic sheets using a local service provider. To reduce the cutting cost, we only outsourced the intricate components while the simpler parts were manually cut using a rotary tool.

The final structure is a mix of materials. It uses laser-cut acrylic for precision parts, repurposed PVC pipe for structural arms, cardboard as the base platform, a conveyor belt for motion demonstration, and a control panel for user interaction. This hybrid approach helped us balance cost, accessibility, and functionality in the overall design.



3.3 Hardware Peripherals

3.3.1 SSD1306 I2C Display

The SSD1306 is a compact 0.96-inch OLED display module that communicates over the I2C protocol, requiring only two pins for data transfer. It offers a resolution of 128x64 pixels, which is sufficient for rendering sharp text, icons, and animations on a small embedded interface. Its low power consumption and high contrast make it ideal for battery-operated or compact devices like our robotic arm console.

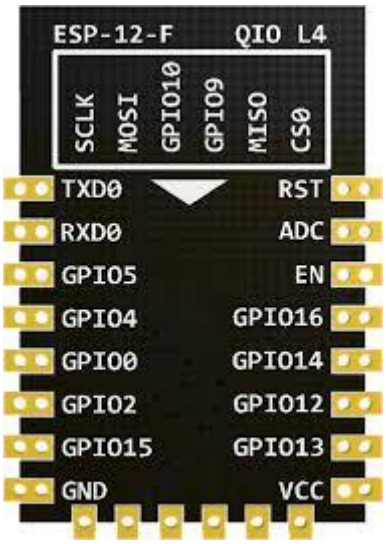
In this project, the SSD1306 display is used to show various interface elements and feedback to the user. It displays simple animations and bitmap images to enhance visual interaction. The interface includes predefined sections such as pre-made motion sequences, connection status checks, a demo mode to

3.7 SSD1306

3.3.2 ESP-12F Wi-Fi Module

The ESP-12F is the core microcontroller used in this project. It is a powerful and compact module based on the ESP8266 chip, known for its built-in Wi-Fi capabilities and sufficient GPIOs for small to medium embedded applications. It supports multiple communication protocols including UART, I2C, and SPI, making it a flexible choice for both control and networking tasks.

In our robotic arm, the ESP-12F acts as the brain of the system. It handles all logic, user input processing, servo control, and Wi-Fi communication for the web interface. The module connects to the OLED display via I2C and also manages input from the UP, DOWN, OK, RESET, and BURN buttons. Its ability to host a local web server allows users to control the robotic arm wirelessly, which adds to the overall convenience and versatility of the system.



3.8 ESP12F

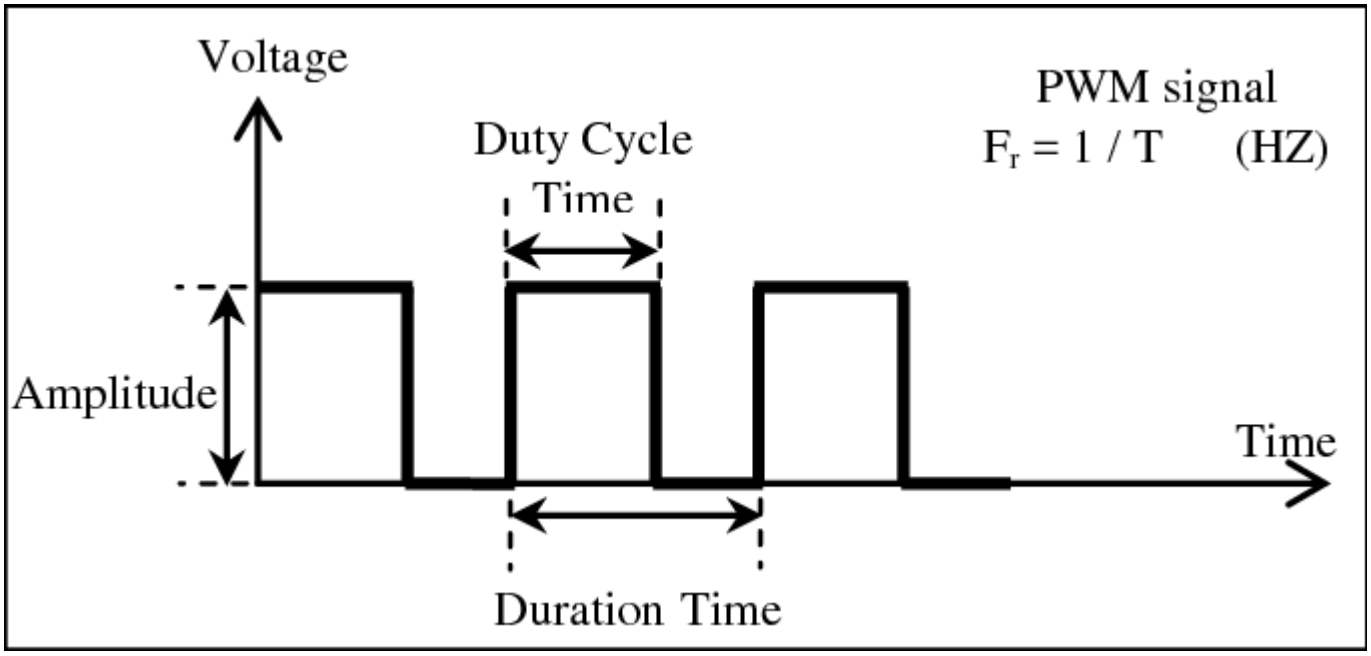
3.3.3 Servo Motors

Servo motors act as the primary actuators in the robotic arm, responsible for the precise movement of each axis. In this project, we have used a combination of SG90 and MG90 servo motors. While SG90s are lightweight and suitable for low-load movements, MG90s feature metallic gears that offer improved strength, rigidity, and durability—especially useful for joints that require better load handling.

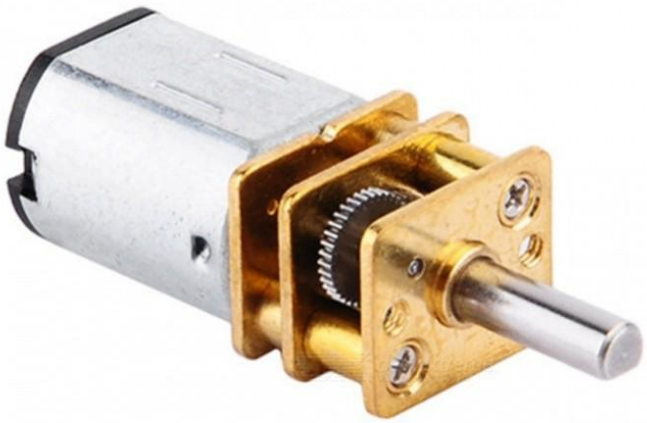
Each servo is controlled using PWM signals generated by the ESP-12F. To achieve different types of motion, the maximum PWM range is defined individually in the code for each motor. Some servos are allowed to move across the full 180-degree range, while others are limited to 150 or even 45 degrees, depending on their mechanical role and placement in the arm. This controlled range ensures smooth and safe operation while performing pre-defined or manual tasks.

3.9 SG90

3.10 MG90



3.10 PWM



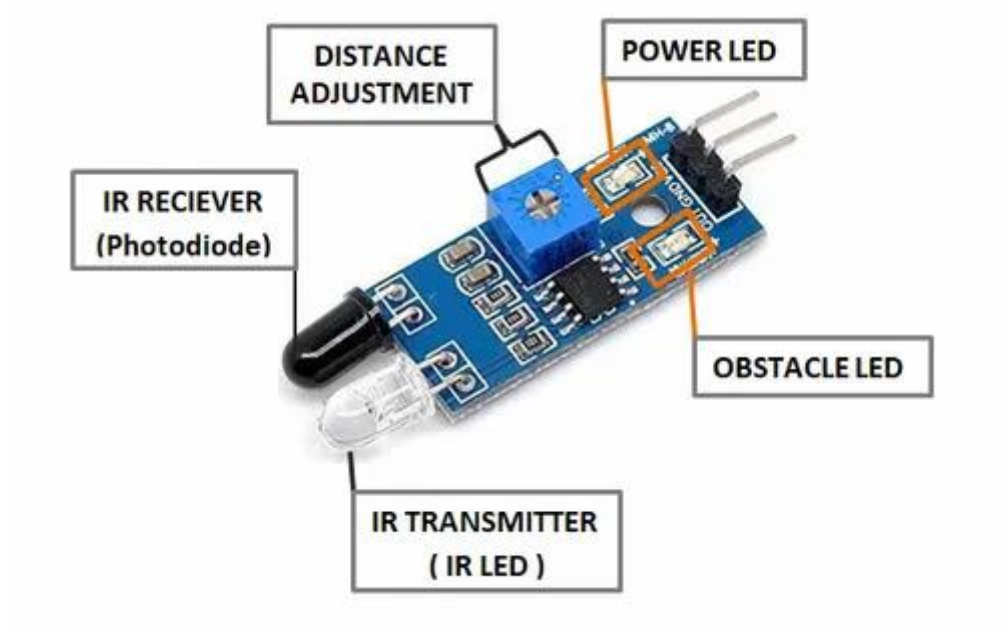
3.11 Gear Motor

3.3.4 Conveyor Belt Mechanism

To simulate object handling, a small conveyor belt setup is included in the system. It uses a basic DC gear motor that provides steady torque and slow RPM, making it ideal for moving lightweight objects in a controlled manner.

The gear motor is powered through a relay module, allowing the ESP-12F to turn it on or off programmatically based on sensor input.

An IR sensor is positioned near the end of the conveyor to detect the presence of objects. It continuously monitors for any obstacle in its line of sight and sends a signal to the ESP when something is detected. Once an object is identified, the ESP triggers the relay to stop the gear motor, halting the conveyor. This adds a basic level of automation and safety to the system and simulates a real-world object detection and sorting scenario.



3.12 IR sensor

3.3.5 Relay Module

The relay module in this project is used to control the gear motor of the conveyor belt. It acts as a switch that can be triggered by the ESP-12F to enable or cut off power to the motor. The relay is driven by a GPIO pin and allows isolation between the control logic and the motor's power supply. This ensures safety and prevents voltage spikes from affecting the microcontroller. The relay setup helps implement a simple logic-based motor stop mechanism based on input from the IR sensor.

3.13 Relay (SPDT)

3.3.6 Power Supply

For powering the entire system, we reused an old regulated power supply (RPS) that was built during a previous minor project. This power supply was tested and provides a stable 5V output suitable for driving both logic-level components and motors. The main power input is taken through a USB Type-C connector with filtering and protection, while a voltage regulator (AMS1117) is used to step down 5V to 3.3V for the ESP-12F and display modules. This reuse of old hardware aligns with our goal of minimizing waste and using available resources efficiently.

Chapter 4: Firmware

The firmware design was one of the most exciting and personally fulfilling parts of the project. Written entirely from scratch, the firmware not only controls the hardware but also brings life to the robotic arm through custom logic, user interfaces, and smart features. This section explores how each part of the system was programmed and integrated.

One of the key highlights was designing the bitmap animations and graphics for the OLED display. These were created manually to match the aesthetic of the console, enhancing the user experience. I also developed a custom navigation menu system to allow interaction using only three physical buttons, enabling smooth access to all features.

The firmware follows a Dual Interface architecture, which allows the robotic arm to be controlled both through a local OLED-based console and over a Wi-Fi-based web interface. In addition, the system supports ESP-to-ESP communication, opening the door for more advanced multi-device coordination in the future.

Preset motion sequences are embedded into the firmware, allowing the arm to perform complex tasks with a single click. From motion logic to safety handling and visual feedback, the firmware plays a central role in unifying the hardware into one functional system.

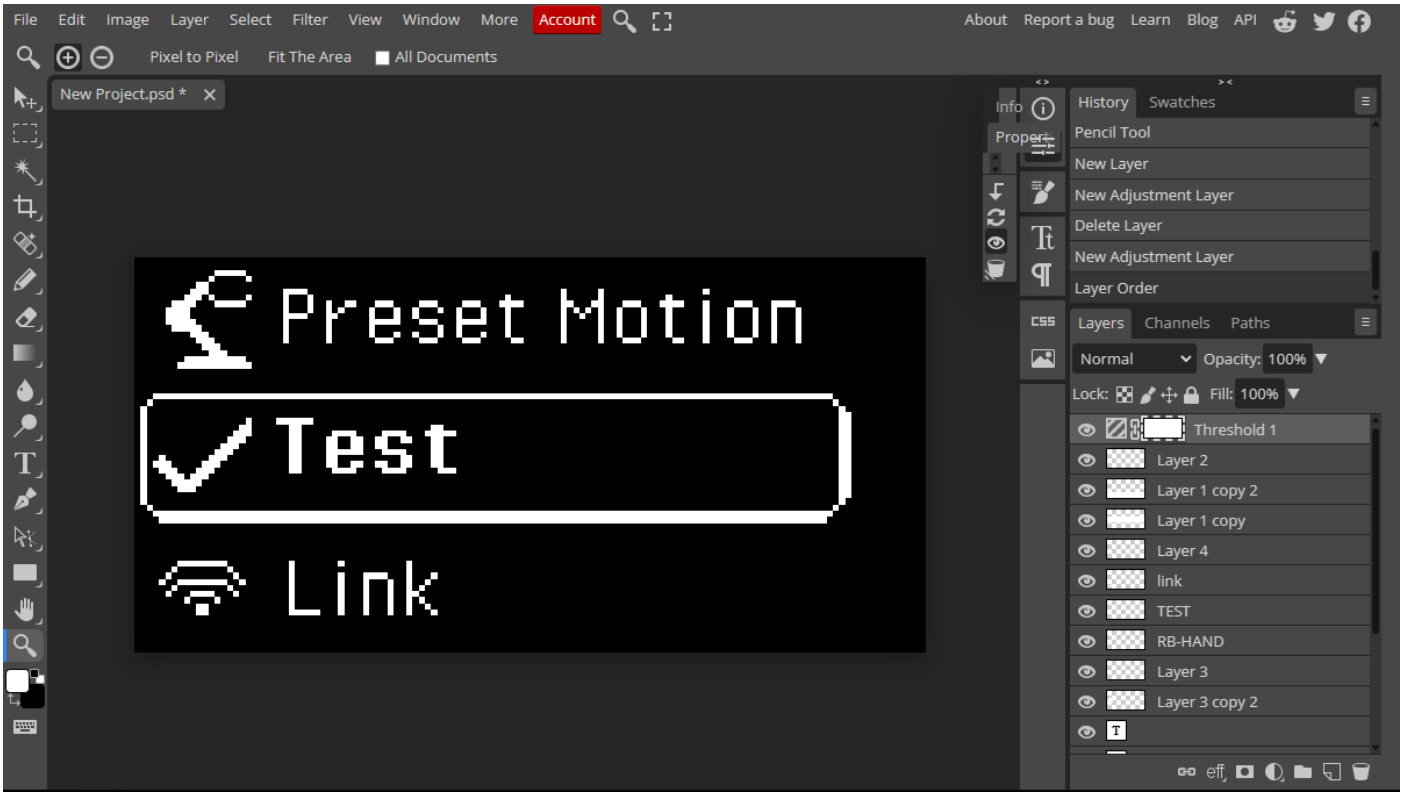
4.1 Interface & Bitmap Design

The first step in developing the user interface was designing custom bitmap images for the OLED display. Since the SSD1306 display has a resolution of 128×64 pixels, I set up a canvas of the same size in Photopea—an online image editing tool compatible with Photoshop workflows. Using basic tools like the pencil and text tool, I manually created pixel-perfect icons and screens tailored for the console layout.

Each screen in the navigation system includes a combination of icons, textual labels, and a highlight box used for selection indication.

The navigation box moves based on user input from the UP, DOWN, and OK buttons. Every graphic element was exported as a monochrome bitmap and converted into byte arrays compatible with the Adafruit SSD1306 library used in the ESP firmware.

This manual approach gave me complete control over the look and feel of the interface, allowing for a consistent and highly optimized UI with minimal memory usage. Each bitmap was mapped to specific screens like the main menu, motion presets, demo mode, about section, and connection check, making the interface both functional and visually informative despite the hardware limitations.



4.1 Bitmap Design

4.2 Navigation System Logic

The navigation system forms the backbone of the console's user interface. It enables users to browse through different operational modes and presets using three physical buttons: UP, DOWN, and OK (A). The logic is built using a menu structure defined by 2D character arrays and controlled through debounced button input.

At the core, two main menu arrays are defined:

- manu_name[4][20] holds the main menu items like Preset Motions, Link, Demo, and About.
- preset_motions_manu[5][20] contains submenu items for predefined motion routines such as High Five, Pick Up, and Dance.

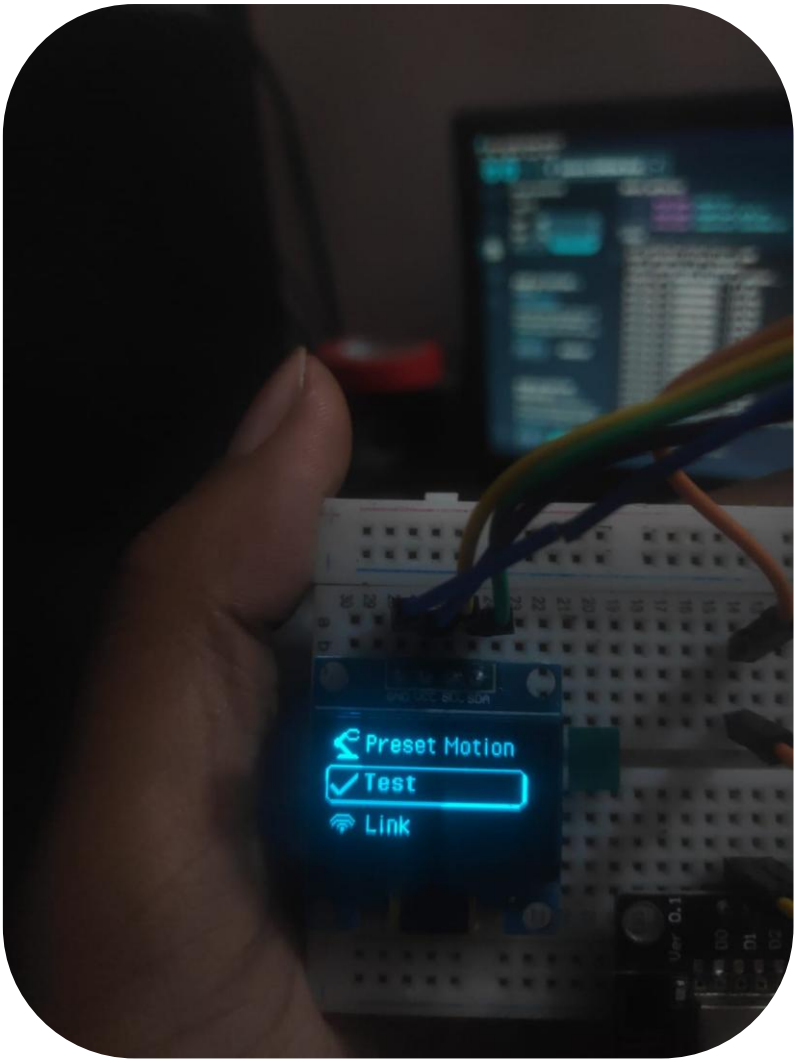
Each screen update is triggered when a button is pressed. To prevent unintended multiple jumps due to bouncing, a software debounce mechanism is implemented using millis() with a threshold defined by debounceDelay.

The menu system is designed with circular navigation. When the user presses UP or DOWN, the pointer (item_selected) moves accordingly and wraps around the list—ensuring smooth, continuous navigation. Three items are shown on the OLED screen at any time: the previous, current, and next menu item, with icons rendered using preloaded bitmap arrays (epd_bitmap_allArray[]).

When the OK button is pressed:

- If the user selects "Preset Motions," it opens a nested submenu with similar scroll and selection behavior.
- The selected motion is then executed by calling the corresponding function (e.g., highfive(), pickup()).
- If "Link" is selected, the system attempts to establish an ESP-to-Web UI connection.
- The "Demo" and "About" options trigger visual output or static screens accordingly.

The navigation logic is lightweight yet effective, enabling responsive control while maintaining low memory usage on the ESP12F. By using custom bitmaps and a menu-driven lookup structure, the system offers both usability and extendibility for future updates.



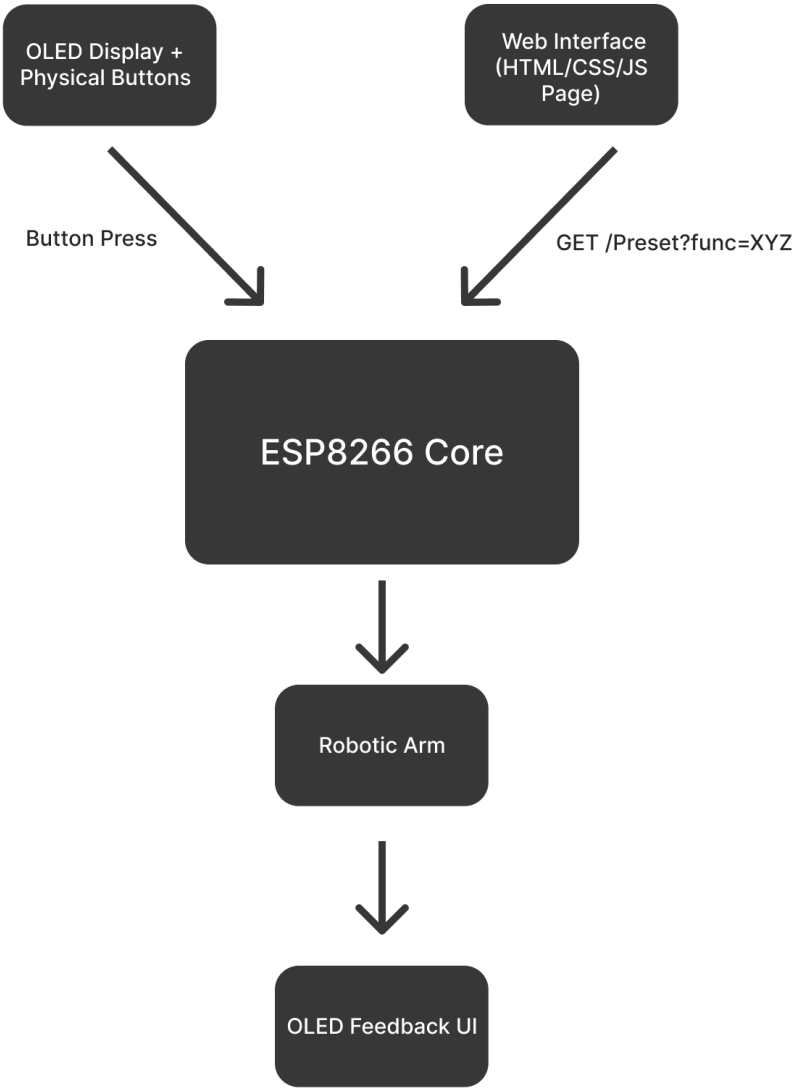
4.2 User Interface

4.3 ESP to ESP Communication

The firmware was designed to support a **Dual Interface Architecture**, allowing the robotic hand to be controlled either locally through the onboard ESP console or remotely via a web interface.

On the local side, the ESP8266 features a dedicated OLED display powered by SSD1306, where a custom menu system built using bitmap icons and animations provides an intuitive user experience. Navigation is handled using physical buttons, and preset functions can be triggered directly through this console.

On the remote side, a web interface—built using HTML, CSS, and JavaScript—hosts a control panel accessible over the local Wi-Fi network. This interface communicates with the ESP via HTTP GET requests.



4.4 Block diagram Dual Interface

4.2 Navigation System Logic

To enhance flexibility and remote control, the system includes ESP-to-ESP communication within the same Wi-Fi network. This setup allows an additional ESP12F console, held by the user, to act as a wireless controller for the robotic arm.

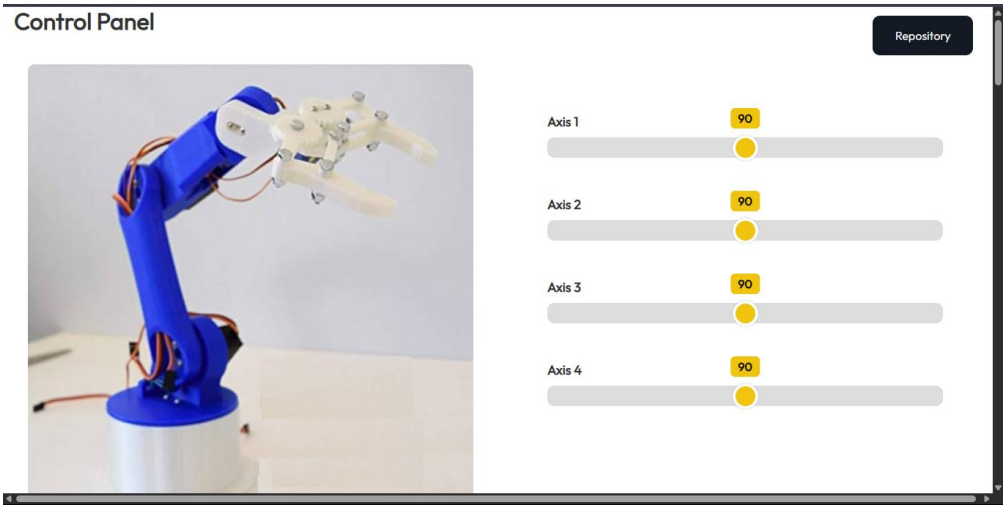
The handheld ESP console communicates with the robotic hand's console through standard HTTP GET requests, the same way the web interface interacts with it. By mirroring the web application's request functions, the handheld unit can trigger preprogrammed motions, control individual axes, and monitor status in real time without needing a browser.

4.4 Web interface

The web interface integrated into the firmware provides a seamless way to interact with the robotic arm using any device connected to the same network. This interface is hosted directly on the ESP12F microcontroller, which acts as a lightweight HTTP server. Once the user connects to the ESP's IP address, the web UI is loaded on the browser, allowing real-time control and monitoring of the robotic system.

The design of the web interface is simple, responsive, and functional. It includes sliders and buttons for controlling each axis of the robotic hand, along with options to execute predefined motion sequences. The interface also displays feedback from the system, such as the current position of the arm or confirmation messages when an action is performed.

The interface communicates with the firmware through HTTP GET requests. Each interaction on the web page, whether it's moving an axis or starting a motion sequence, sends a corresponding GET request to the server running on the ESP. The firmware parses the request and carries out the operation, updating the state of the robotic hand accordingly.



4.5 Web Interface

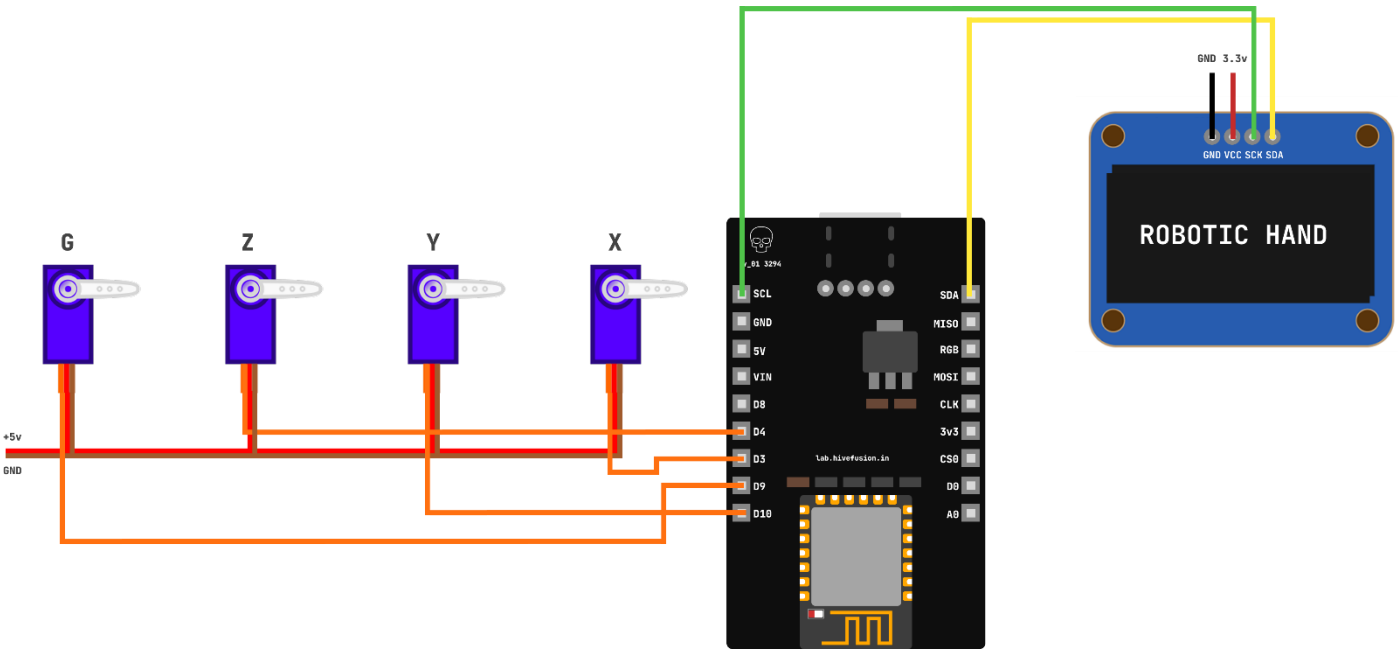
Chapter 5: Methodology

This chapter explains the step-by-step process followed to build, program, and deploy the robotic arm system. It covers hardware connections, firmware uploading procedures, and how the built-in web interface is hosted and accessed.

5.1 Hardware Connections

To establish the complete control system, the ESP12F console is connected to four servo motors corresponding to the robotic arm's axes. Each servo is powered through an external regulated 5V power supply to ensure stable performance. The control pins of the servos are connected to the GPIOs of the ESP12F module. The SSD1306 OLED is connected using the I2C protocol, where SDA and SCL lines are routed to the corresponding pins on the ESP. Three buttons are connected with pull-down resistors to digital input pins, acting as input controls for the on-device OLED interface.

Proper attention is given to the current ratings of the components. Decoupling capacitors and bypass resistors are used wherever necessary to stabilize the power supply and avoid brownout issues during motor movement.



5.1 Connections

5.2 Setting Up the Arduino IDE

To upload firmware to the ESP12F console, the Arduino IDE must be configured correctly with the required board definitions and drivers. Below is the complete step-by-step process to set up the Arduino IDE for ESP8266 development.

Step 1: Install Arduino IDE

Download and install the Arduino IDE from the official website:
https://www.arduino.cc/en/software

Step 2: Add ESP8266 Board to Arduino IDE

1. Open the Arduino IDE.
2. Go to File > Preferences.
3. In the Additional Board Manager URLs field, paste the following link:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

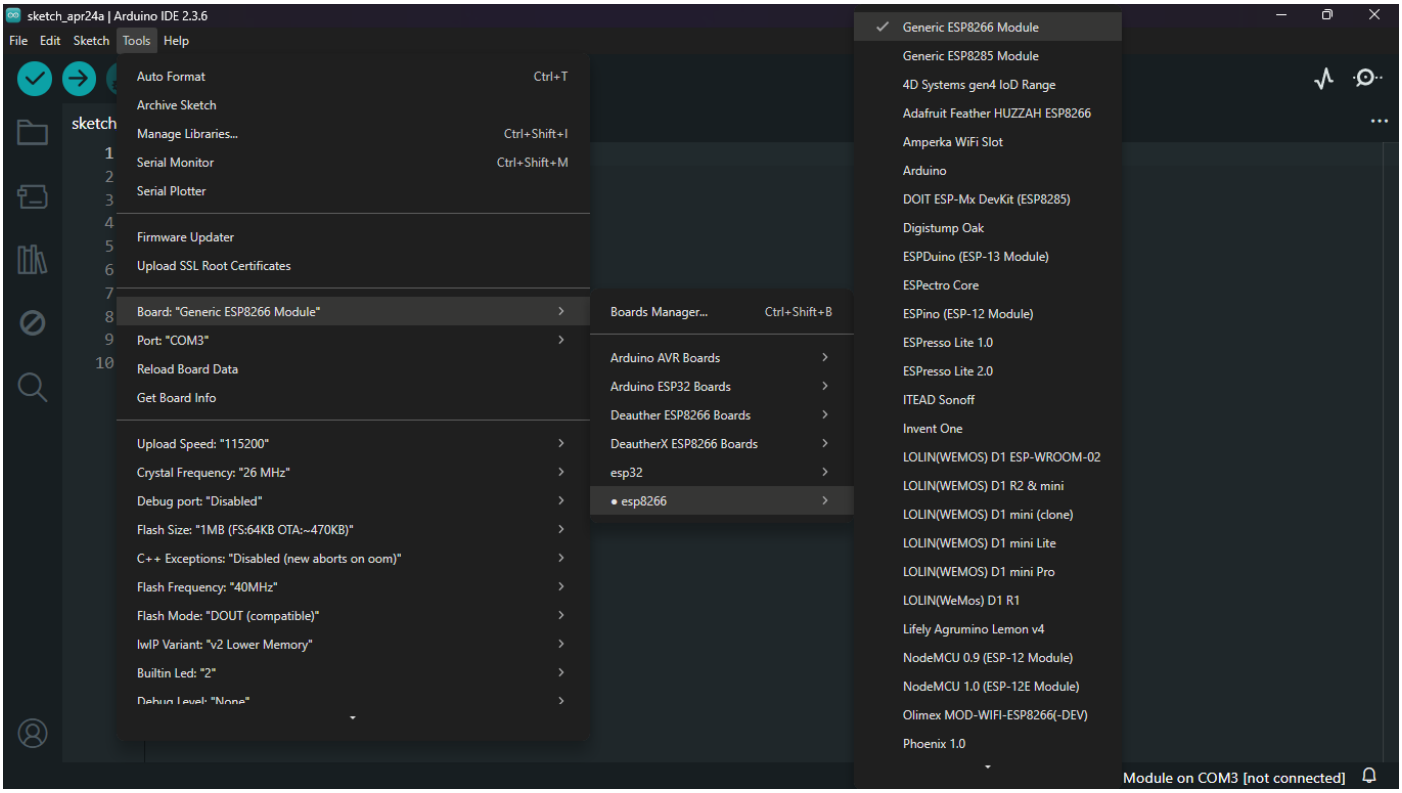
4. Click **OK** to save the preferences.

Step 3: Install ESP8266 Board Package

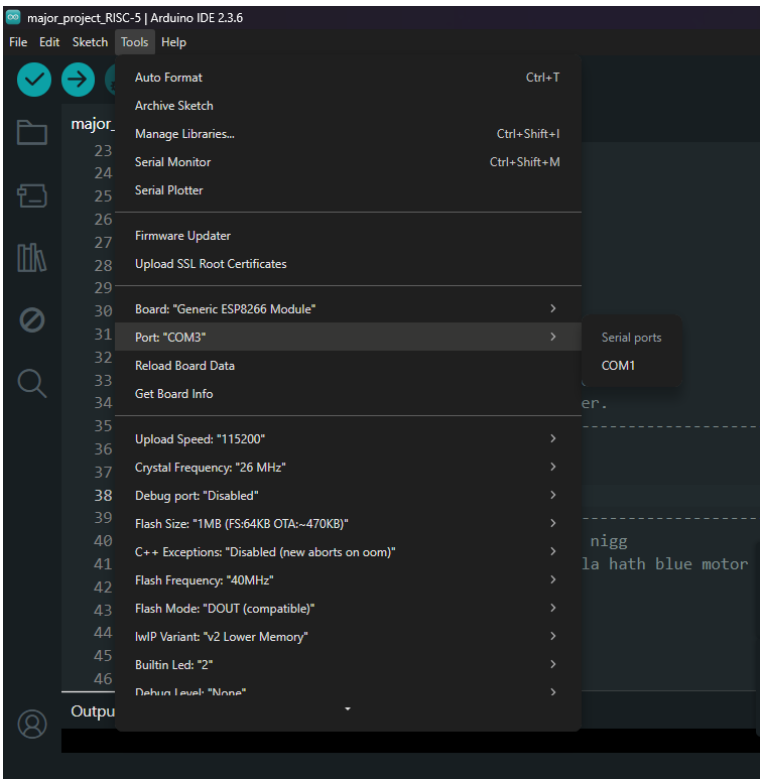
1. Go to Tools > Board > Boards Manager.
2. In the search bar, type esp8266.
3. Install the package named esp8266 by ESP8266 Community.
4. Wait for the installation to complete.
5. After installation, select the board as Generic ESP8266 Module from the Tools > Board menu.

Step 4: Install CH340G USB to Serial Driver

The ESP12F development board usually includes a CH340G USB-to-Serial converter, which requires a driver installation on your PC.



5.2 Board Selection



5.2 COM port

Connect the ESP8266/Console to your computer and upload the following code

```
/*
  ---------------------------------------------------------
  Project Title   : ESP8266 OLED screen based Robotic Arm
  Version         : 1.2
  Author          : Mohd Amir
  Contributor     : Sunil Kumar
  Project Type    : Major Project
  ---------------------------------------------------------

  Description:
  This project implements an interactive menu system using an ESP8266
  microcontroller with a 0.96" I2C OLED display. It features a smooth
  UI experience with bitmap icons, animated startup screen, and
  navigation using active-low push buttons.

  Designed as a versatile interface console, this system can be easily
  adapted for various applications like motion presets, testing routines,
  and device linking. Clean aesthetics and logical flow ensure both
  usability and expandability for embedded solutions.

  Developed with passion and precision to serve as a solid foundation
  for real-world IoT and embedded menu-driven applications.
*/


#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
```

```
#include <Servo.h>

#define D2 5  //Bhai D2 ki pin SCL se connect ker.
#define D1 4  //Bhai D1 ki pin SDL se connect ker.
//<---------------------------------buttons------------------------------------->
#define BUTTON_UP 14    // D5
#define BUTTON_DOWN 12  // D6
#define BUTTON_A 13     // D7
//<-----------------------------Servo Pins------------------------------->
const int pinX = 0;  //D3 age jane wala hath mr nigg
const int pinY = 1;  //D10 upper niche kerne wala hath blue motor
const int pinZ = 2;  //D4 kekdha
const int pinG = 3;  //D9 base

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);
//<----------------------Wifi credential and port-------------------------->
const char* ssid = "Robotic_hand";
const char* password = "12345678";
ESP8266WebServer server(80);

bool awaitingPing = false;
unsigned long pingStartTime = 0;
bool linkConfirmed = false;
bool stopreverse180 = false;
bool IsKekdhanepakadhrakhahai = false;

//Referance positions of robotic Hand

int posX;
int posY;
int posZ;
```

```
int posG;

Servo servoX, servoY, servoZ, servoG;

// 'Outline-select', 128x21px
const unsigned char epd_bitmap_Outline_select[] PROGMEM = {
  0x1f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0x00,
  0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00,
  0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x00,
  0x40, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x00,
  0x20, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00,
  0x1f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x80, 0x00,
  0x0f, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00
};
// 'RoboticHandicon', 16x16px
const unsigned char epd_bitmap_RoboticHandicon[] PROGMEM = {
  0x00, 0x3e, 0x00, 0x41, 0x01, 0x80, 0x03, 0x80, 0x07, 0x41, 0x1e, 0x3e, 0x3c, 0x00, 0x3c, 0x00,
  0x3e, 0x00, 0x1f, 0x00, 0x03, 0x80, 0x01, 0xc0, 0x00, 0xe0, 0x00, 0x70, 0x07, 0xfe, 0x0f, 0xff
};
// 'TestIcon', 16x16px
```

```
const unsigned char epd_bitmap_TestIcon[] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x03, 0x00, 0x07, 0x00, 0x0e, 0x00, 0x1c, 0x00, 0x38,
  0x40, 0x70, 0xe0, 0xe0, 0x71, 0xc0, 0x3b, 0x80, 0x1f, 0x00, 0x0e, 0x00, 0x00, 0x00, 0x00, 0x00
};
// 'Checklink', 16x16px
const unsigned char epd_bitmap_Checklink[] PROGMEM = {
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, 0xf0, 0x10, 0x08, 0x20, 0x04, 0x4f, 0xf2, 0x10, 0x08,
  0x23, 0xc4, 0x04, 0x20, 0x01, 0x80, 0x01, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

const unsigned char epd_bitmap_About[] PROGMEM = {
  0x00, 0x00, 0x01, 0xc0, 0x01, 0xc0, 0x01, 0xc0, 0x01, 0xc0, 0x01, 0xc0, 0x01, 0xc0, 0x01, 0xc0,
  0x01, 0xc0, 0x01, 0xc0, 0x00, 0x00, 0x01, 0xc0, 0x01, 0xc0, 0x01, 0xc0, 0x00, 0x00, 0x00, 0x00
};

// Array of all bitmaps for convenience. (Total bytes used to store images in PROGMEM = 496)
const int epd_bitmap_allArray_LEN = 5;
const unsigned char* epd_bitmap_allArray[5] = {
  epd_bitmap_RoboticHandicon,
  epd_bitmap_Checklink,
  epd_bitmap_TestIcon,
  epd_bitmap_About,
  epd_bitmap_Outline_select
};


void startAnimation() {
  display.setTextSize(1.5);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor((SCREEN_WIDTH - 60) / 2, (SCREEN_HEIGHT + 22) / 2);
  display.println("Starting...");
  display.display();
  delay(1000);
  display.clearDisplay();
```

```
  for (int i = 0; i < 3; i++) {
    display.clearDisplay();
    display.drawBitmap(
      (SCREEN_WIDTH - 16) / 2,  // center 16x16 bitmap
      (SCREEN_HEIGHT - 16) / 2,
      epd_bitmap_allArray[i],
      16, 16,
      SSD1306_WHITE);
    display.display();
    delay(800);
  }

  delay(300);
  display.clearDisplay();
}

void LoadingBar(int timeDelay) {
  int barWidth = 80;
  int barHeight = 10;
  int barX = (SCREEN_WIDTH - barWidth) / 2;
  int barY = 35;

  display.drawRect(barX, barY, barWidth, barHeight, WHITE);
  display.display();


  for (int i = 0; i <= 100; i += 2) {
    int fillWidth = (i * (barWidth - 2)) / 100;
    display.fillRect(barX + 1, barY + 1, fillWidth, barHeight - 2, WHITE);


    //presentage number bhai
    display.fillRect((SCREEN_WIDTH - 20) / 2, barY + 15, 30, 10, BLACK);
    display.setCursor((SCREEN_WIDTH - 20) / 2, barY + 15);
```

```
    display.setTextColor(WHITE);
    display.printf("%d%%", i);


    display.display();
    delay(timeDelay);
  }
}

void showAbout() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(WHITE);


  display.setCursor((128 - 96) / 2, 2);
  display.println("Robotic Arm ");


  display.setCursor((128 - 96) / 2, 20);
  display.println("By Amir & Sunil");


  display.setCursor((128 - 94) / 2, 38);
  display.println("Guided by Anuj sir");

  display.display();

  delay(3000);
}
```

```
void highfive() {
  display.clearDisplay();

  // Show text message
  display.setTextSize(1);
  display.setTextColor(WHITE);
  display.setCursor((SCREEN_WIDTH - 60) / 2, (SCREEN_HEIGHT + 22) / 2 - 20);
  display.println("High Five...");


  LoadingBar(90);
}


void test() {
  display.clearDisplay();
  display.setCursor((SCREEN_WIDTH - 60) / 2, (SCREEN_HEIGHT + 22) / 2 - 20);
  display.println("Testing...");
  LoadingBar(60);
  display.display();
  delay(1000);
}

void pickup() {
  display.clearDisplay();
  display.setCursor((SCREEN_WIDTH - 60) / 2, (SCREEN_HEIGHT + 22) / 2 - 20);
  display.println("picking up...");
  LoadingBar(60);
  display.display();
  delay(1000);
}

void Dance() {
  display.clearDisplay();
```

```
  display.setCursor((SCREEN_WIDTH - 60) / 2, (SCREEN_HEIGHT + 22) / 2 - 20);
  display.println("Danceing...");
  display.display();
  delay(1000);
}

void testServoandConnection() {
  display.clearDisplay();
  display.setCursor((SCREEN_WIDTH - 60) / 2, (SCREEN_HEIGHT + 22) / 2 - 20);
  LoadingBar(60);
  display.display();
  delay(1000);
  moveX(posX, 20);
  moveX(posY, 20);
  moveX(posZ, 20);
  moveX(posG, 20);
  int tempX = posX;
  int tempY = posY;
  int tempZ = posZ;
  int tempG = posG;

  posX = 20;
  posY = 20;
  posZ = 20;
  posG = 20;

  moveX(posX, tempX);
  moveX(posY, tempY);
  moveX(posZ, tempZ);
  moveX(posG, tempG);

  posX = tempX;
  posY = tempY;
  posZ = tempZ;
```

```
  posG = tempG;
}

void checkWebUILink() {
  display.clearDisplay();
  display.setTextSize(1);
  display.setCursor(0, 20);
  display.println("Checking Web UI...");
  display.println("Please click TEST on UI.");
  display.display();

  awaitingPing = true;
  linkConfirmed = false;
  delay(4000);
  pingStartTime = millis();
}

// ek function jo 180 degree motion kerwayga base ka
void Do180() {
  display.clearDisplay();
  display.setCursor((SCREEN_WIDTH - 60) / 2, (SCREEN_HEIGHT + 22) / 2 - 20);
  display.println("Doing 180");
  LoadingBar(20);
  display.display();
  if (stopreverse180) {
    moveG(posG, 0);
    posG = 0;
    stopreverse180 = false;
  } else {
    moveG(posG, 180);
    posG = 180;
    stopreverse180 = true;
  }
}
```

```
//to Grab a object and to reales it it does it autometically
void kekdha() {
  display.clearDisplay();
  display.setCursor((SCREEN_WIDTH - 60) / 2, (SCREEN_HEIGHT + 22) / 2 - 20);
  if (IsKekdhanepakadhrakhahai) display.println("Closing");
  else display.println("Opening");
  LoadingBar(15);
  display.display();

  if (IsKekdhanepakadhrakhahai) {
    for (int i = 145; i >= 0; i--) {
      servoZ.write(i);
      delay(15);
    }
    IsKekdhanepakadhrakhahai = false;
  } else {
    // If not holding, grab
    for (int i = 0; i <= 145; i++) {
      servoZ.write(i);
      delay(15);
    }
    IsKekdhanepakadhrakhahai = true;
  }
}


void moveX(int from, int to) {
  int step = (to > from) ? 1 : -1;
  for (int i = from; i != to + step; i += step) {
    servoX.write(i);
    delay(15);
  }
}
```

```
void moveY(int from, int to) {
  int step = (to > from) ? 1 : -1;
  for (int i = from; i != to + step; i += step) {
    servoY.write(i);
    delay(15);
  }
}

void moveZ(int from, int to) {
  int step = (to > from) ? 1 : -1;
  for (int i = from; i != to + step; i += step) {
    servoZ.write(i);
    delay(15);
  }
}

void moveG(int from, int to) {
  int step = (to > from) ? 1 : -1;
  for (int i = from; i != to + step; i += step) {
    servoG.write(i);
    delay(15);
  }
}



//The demo function
void demo() {
  display.clearDisplay();
  display.setCursor((SCREEN_WIDTH - 60) / 2, (SCREEN_HEIGHT + 22) / 2 - 20);
  display.println("Demo Mode...");
  LoadingBar(10);
  display.display();
  //the initial postion
```

```
moveX(posX, 50);
moveY(posY, 35);
moveZ(posZ, 22);
moveG(posG, 0);
delay(800);

moveX(50, 116);
delay(200);
moveY(33, 45);
delay(200);
moveZ(20, 102);
```

While a substantial segment of the ESP12F console's firmware is included in this chapter to demonstrate core features such as the graphical user interface on the OLED and the control algorithms for the robotic arm's axes, the entirety of the codebase is provided via the QR code below for readers who require a comprehensive view.

5.3 Website Code and Local Hosting

The website developed for controlling the robotic arm is designed to offer a clean, responsive, and user-friendly interface. It allows users to interact with the robotic system using simple controls to move each axis manually or trigger preprogrammed motion sequences. The layout is minimalistic, optimized for quick loading and ease of use across desktops, tablets, and mobile devices.

- Ensure Node.js and npm (or yarn) are Installed: Before you begin, make sure you have Node.js and its package manager (npm is included with Node.js, yarn is an alternative you might have installed) on your system. You can check if they are installed by opening your terminal or command prompt and running the commands node -v and npm -v (or yarn --version). If they are installed, you will see their respective version numbers. If not, you'll need to download and install Node.js from the official website (https://nodejs.org/).

- Create Your Project Directory: Choose a location on your computer where you want to create your website files. Open your terminal or command prompt, navigate to that location using the cd command, and create a new directory for your project. For example, you can use the command mkdir my-website and then cd my-website to enter the newly created directory.

- Initialize Your Node.js Project: Inside your project directory, you need to initialize a Node.js project. This will create a package.json file, which keeps track of your project's dependencies and other metadata. Run the command npm init -y (for default settings) or npm init (to go through the setup prompts). If you are using yarn, run yarn init -y.

- Install the 'express' Package: For easily creating a web server in Node.js, the 'express' framework is very popular and helpful. In your terminal or command prompt, within your project directory, run the command npm install express or yarn add express. This will download and install the express library and add it to your project's dependencies in the package.json file.

- Create Your Main Server File (e.g., server.js): Create a new file in your project directory. You can name it something like server.js (or any other name you prefer with a .js extension). This file will contain the code for your Node.js web server.

- **Write Your Node.js Server Code:** Open the server.js file in a text editor or code editor and paste the following basic code to create a simple web server that serves static files (your HTML, CSS, JavaScript, images, etc.) from a directory named 'public':

```
const express = require('express');
const app = express();
const port = 3000; // You can choose a different port number

app.use(express.static('public'));

app.listen(port, () => {
  console.log(`Server listening at http://localhost:${port}`);
});
```

5.3 Hosting

Create three saperate files index.html , style.css , app,js

Or download Official Repo - https://github.com/mohdamirwebdeveloper/robotic-hand

Index.html ->

<!--

Version = 1.0.1v

Project name : Control penal for Robotic Arm

Project Desctption : Its a control penal in which you can control a 4 axis freedom robotic arm

This project is created by Mohd Amir and Sunil kumar

Its a

class : ECE B

Group - 02

```
-->

<!DOCTYPE html>
<html lang="en">

<head>

  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>ECE-B-Robotic-Hand-Controller-SYS </title>
  <link rel="stylesheet" href="style.css">

  <!-- Fonts-->

  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
    href="https://fonts.googleapis.com/css2?family=Montserrat:ital,wght@0,100..900;1,100..900&family=Outfit:wght@389&family=Roboto:ital,wght@0,100..900;1,100..900&display=swap"
    rel="stylesheet">
  <script src="app.js" defer></script>
</head>

<body>

  <div class="heading-container flex center M-5">

    <div class="heading flex">
      <h1 class="">
        Control Panel
      </h1>
    </div>
    <div class="manu">
```

```
      <div><button type="button" id="repository" class="R-10"><a
            href="https://github.com/mohdamirwebdeveloper/robotic-hand" target="_blank"
            style="color: white; text-decoration: none;">Repository</a></button></div>
    </div>
</div>

<div id="root" class="root flex">
    <div class="section-1 overflow-H ">

      <div class="image-container overflow-H R-10 flex center">

        <img class="R-H-img R-10" src="./img/roboticarm_3d_printied.jpg" alt="Robotic-Hand-Image" srcset="">
      </div>
    </div>

    <!-- Control Penal code -->

    <div class="section-2 flex j-c ">
      <div class="container-section-2">
        <!-- Axis 1 -->
        <div class="slider-wrapper">
          <div class="label-container">
            <span class="slider-label">Axis 1</span>
          </div>
          <div class="slider-container">
            <input type="range" min="0" max="180" value="90" id="axis1" oninput="updateSlider(this, 1)">
            <div class="tooltip" id="tooltip1">90</div>
            <div class="axis-value none">0° - 180°</div>
          </div>
        </div>

        <!-- Axis 2 -->
        <div class="slider-wrapper">
          <div class="label-container">
```

```
        <span class="slider-label">Axis 2</span>
      </div>
      <div class="slider-container">
        <input type="range" min="0" max="180" value="90" id="axis2" oninput="updateSlider(this, 2)">
        <div class="tooltip" id="tooltip2">90</div>
        <div class="axis-value none">0° - 180°</div>
      </div>
    </div>

    <!-- Axis 3 -->
    <div class="slider-wrapper">
      <div class="label-container">
        <span class="slider-label">Axis 3</span>
      </div>
      <div class="slider-container">
        <input type="range" min="0" max="180" value="90" id="axis3" oninput="updateSlider(this, 3)">
        <div class="tooltip" id="tooltip3">90</div>
        <div class="axis-value none">0° - 180°</div>
      </div>
    </div>

    <!-- Axis 4 -->
    <div class="slider-wrapper">
      <div class="label-container">
        <span class="slider-label">Axis 4</span>
      </div>
      <div class="slider-container">
        <input type="range" min="0" max="180" value="90" id="axis4" oninput="updateSlider(this, 4)">
        <div class="tooltip" id="tooltip4">90</div>
        <div class="axis-value none">0° - 180°</div>
      </div>
    </div>
  </div>
```

```
    </div>
  </div>
  <div class="pre-programmed-functions-container flex center">
    <!--<div><button class="btn-style M-10 R-18" type="button">Pick Up</button></div>-->
    <div><button class="btn-style M-10 R-18" type="button" onclick="PresetMotions('Do180')">Do 180</button></div>
    <div><button class="btn-style M-10 R-18" type="button" onclick="testESPLink()">Ping</button></div>
    <div><button class="btn-style M-10 R-18" type="button">High five</button></div>
    <div><button class="btn-style M-10 R-18" type="button" onclick="PresetMotions('Dance')">Dance</button></div>
    <div><button class="btn-style M-10 R-18" type="button" onclick="PresetMotions('Kekdha')">Kekdha</button></div>
    <div><button class="btn-style M-10 R-18" type="button" onclick="PresetMotions('demo')">Demo</button></div>
  </div>

  <section class="section-3">
    <div class="main-container-about-us">
      <div class="section-3-heading-container flex center">
        <h1>About Project</h1>
      </div>

      <section style="max-width: 900px; margin: 0 auto; padding: 40px 20px; font-family: 'Segoe UI', sans-serif;">
        <h2 style="font-size: 2rem; margin-bottom: 20px; text-align: center; color: #333;">🤖 Robotic Arm Control System</h2>

        <p style="font-size: 1.1rem; color: #555; line-height: 1.7; text-align: justify;">
          This project demonstrates a custom-built 4-axis robotic arm, designed and developed from scratch using affordable and recycled materials.
          The system is powered by our own microcontroller with a custom PCB and integrates web-based remote control, allowing real-time motion control from anywhere in the world.
          The robotic arm supports both manual and pre-programmed functions, making it suitable for a variety of educational, research, and automation applications.
          With intuitive frontend controls, built-in display navigation, and secure backend communication using Django, this system showcases an end-to-end IoT + robotics solution.
```

```
    Applications include learning platforms, light automation tasks, basic pick-and-place operations,
    and remote demonstration tools.
  </p>
</section>

<div class="section-3-microcontroller">
  <div class="microcontroller-foto-container flex center R-18">
    <div class="R-18 overflow-H microcontroller-foto-container-2"><img class="microcontroller-foto R-18"
        src="./img/microcontroller.jpeg" alt="" srcset=""></div>
  </div>
  <div>
    <div class="sec3-micro-heading">
      <h2> Our Own Microcontroller Custom PCB Designs</h2>
    </div>
    <div class="microcontroller-features ">
      <p style="font-size: 1.1rem; color: #555; line-height: 1.6;">
        At the heart of our robotic arm lies a <strong>custom-designed microcontroller PCB</strong>,
        engineered specifically for robotics applications. This board is not just a controller —
        it's a compact powerhouse designed to provide precise, reliable, and flexible control over
        every movement of the arm.
      </p>

      <div style="margin-top: 30px;">
        <h3 style="color: #222; margin-bottom: 10px;">Key Features:</h3>
        <ul style="padding-left: 20px; font-size: 1.05rem; color: #444; line-height: 1.8;">
          <li><strong>Built-in OLED Display (I2C):</strong> 0.96" screen to monitor real-time
            data, angles, or system status directly.</li>
          <li><strong>18 Versatile Pins:</strong>
            <ul style="padding-left: 20px; margin-top: 5px;">
              <li>7 GPIOs for input/output control</li>
              <li>I2C and SPI interfaces</li>
              <li>5V and 3.3V regulated output</li>
              <li>Ground pins for flexible circuit connections</li>
              <li>Dedicated SPI pins , MISO,MOSI,CSO,CLK</li>
```

```
            </ul>
          </li>
          <li><strong>3 Built-in Control Buttons:</strong> Easy access for manual control or
            configuration tasks.</li>
          <li><strong>USB Type-C Power Input:</strong> Modern and robust, compatible with common
            USB-C cables.</li>
        </ul>
      </div>

      <p style="font-size: 1.1rem; color: #555; margin-top: 30px;">
        This board not only simplifies the wiring and reduces clutter, but also enhances performance
        and modularity for robotic applications. It acts as the brain of our robotic arm, enabling
        both manual and remote control with ease.
      </p>
    </div>

  </div>

</div>

<div class="sec3-material-used">
  <h2 style="font-size: 2rem; margin-bottom: 20px; color: #333;">Materials Used &
    Design Overview</h2>

  <div style="font-size: 1.1rem; color: #555; line-height: 1.6;">
    <p>
      Our robotic arm is built using accessible and recycled materials, showcasing not just technical
      innovation, but also sustainable design choices.
    </p>

    <ul style="padding-left: 20px; margin-top: 20px; color: #444;">
      <li><strong>Acrylic Sheet:</strong> Used for the structural components of the arm due to its
        strength, durability, and aesthetic finish.</li>
      <li><strong>Recycled PVC Pipe:</strong> Repurposed to serve as the joints and connectors,
```

```
        providing a lightweight yet robust framework.</li>
      <li><strong>Cardboard Platform:</strong> A simple yet effective base structure that supports the
        arm securely during operation.</li>
      <li><strong>Power Supply:</strong> An old mobile charger has been repurposed as a reliable 5V
        power source for the entire system.</li>
    </ul>

    <h3 style="margin-top: 30px; color: #222;">4-Axis Robotic Arm Mechanism</h3>
    <p>
      The robotic arm is designed with <strong>4 degrees of freedom</strong>, powered by <strong>4
        SG90 servo motors</strong>. These lightweight and cost-effective motors allow smooth
      movement across four independent axes, enabling complex actions such as lifting, rotating, and
      tilting.
    </p>
    <p>
      This setup demonstrates precise control and movement, all coordinated through our custom-built
      PCB and a user-friendly web interface.
    </p>
  </div>

  <div class="sec-3-programming MT-20">
    <div class="R-18 overflow-H sec-blockdiagram-container flex center">
      <img class=" R-18" src="./img/bloackdiagram.png" alt="" srcset="">
    </div>
    <h2 style="font-size: 2rem; margin-bottom: 20px; color: #333;">Programming &
      Functionality</h2>

    <p style="font-size: 1.1rem; color: #555; line-height: 1.6;">
      The robotic arm system is powered by a combination of hardware-level programming, modern web
      technologies, and backend integration to offer real-time control and automation.
    </p>

    <div style="margin-top: 30px;">
      <h3 style="color: #222; margin-bottom: 10px;">Microcontroller Programming</h3>
```

```
    <ul style="padding-left: 20px; font-size: 1.05rem; color: #444; line-height: 1.8;">
      <li><strong>Languages Used:</strong> C / C++</li>
      <li><strong>Libraries:</strong> <code>Servo.h</code>, <code>WiFi.h</code>,
        <code>HTTPServer.h</code>, <code>OLED.h</code>
      </li>
      <li>
        The custom microcontroller features a menu system navigated using 3 buttons:
        <strong>Select</strong>, <strong>Up</strong>, and <strong>Down</strong>. Functions
        include:
        <ul style="padding-left: 20px;">
          <li>Robot Calibration / Testing</li>
          <li>Connection Status Check</li>
          <li>Pre-Programmed Motions</li>
        </ul>
      </li>
    </ul>
  </div>

  <div style="margin-top: 30px;">
    <h3 style="color: #222; margin-bottom: 10px;"> Web Interface</h3>
    <ul style="padding-left: 20px; font-size: 1.05rem; color: #444; line-height: 1.8;">
      <li><strong>Frontend:</strong> HTML5, CSS3, JavaScript</li>
      <li>
        Control Panel Features:
        <ul style="padding-left: 20px;">
          <li>Real-time control of 4 motor axes (0°–180°)</li>
          <li>Connectivity Test Button</li>
          <li>Trigger Pre-Defined Robotic Motions</li>
          <li>Global Access via Internet</li>
        </ul>
      </li>
    </ul>
  </div>
```

```
            <div style="margin-top: 30px;">
              <h3 style="color: #222; margin-bottom: 10px;"> Backend Integration</h3>
              <p style="font-size: 1.05rem; color: #444;">
                The backend is built with <strong>Python using Django</strong>. It handles secure API
                endpoints for receiving commands from the web interface. Here's how the interaction works:
              </p>

              <ul style="padding-left: 20px; font-size: 1.05rem; color: #444; line-height: 1.8;">
                <li>User clicks a button on the control panel</li>
                <li>Frontend sends a secure HTTP request with authentication cookies</li>
                <li>Backend verifies the user and forwards the command to the robotic arm</li>
                <li>Robot executes the motion or action in real time</li>
              </ul>
            </div>

            <p style="margin-top: 30px; font-size: 1.1rem; color: #555;">
              This fully integrated system allows users to operate the robotic arm from any location, ensuring
              flexibility, security, and ease of access.
            </p>

          </div>
        </div>
      </div>
    </section>

    <footer
      style="background: #1b2738; padding: 20px 0; text-align: center; font-family: 'Segoe UI', sans-serif; font-size: 0.95rem; color: #ffffff; margin-top: 60px; border-top: 1px solid #e0e0e0;">
      <p>© 2025 Robotic Arm Control System – Final Year Major Project</p>
      <p>Made with ❤️ by <strong>Mohd Amir</strong> & <strong>Sunil Kumar</strong></p>
      <p>Under the guidance of <strong>Anuj Kalra Sir</strong></p>
      <p>Mentor Email: <a href="mailto:ftd10922194@dseu.ac.in"
          style="color: #007bff; text-decoration: none;">anuj.kalra@dseu.ac.in</a></p>
    </footer>
```

```
</body>

</html>
```

Style.css

```
* {
    margin: 0;
    padding: 0;
    font-family: "Outfit", sans-serif;
}

.root {
    width: 100%;
    height: 100vh;
    background-color: white;
}

.flex {
    display: flex;
}

.center {
    align-items: center;
    justify-content: center;
}

.j-c {
    justify-content: center;
}

.a-c {
```

```
    align-items: center;
}

.overflow-H {
    overflow: hidden;
}

.R-5 {
    border-radius: 5px;
}

.R-10 {
    border-radius: 10px;
}



.R-18 {
    border-radius: 18px;
}

.M-5 {
    margin: 5px;
}

.M-10 {
    margin: 10px;
}

.MT-20 {
    margin-top: 20px;
}

.none {
    display: none;
```

```
}

.heading-container {
    width: 100%;
    height: 10vh;
    justify-content: space-between;
}

.heading {
    margin: 10px;

}

.manu {
    margin: 10px;
    margin-right: 2rem;
}

#repository {
    width: 10vw;
    height: 8vh;
    outline: none;
    background-color: #141a23;
    border: none;
    color: white;
    padding: 10px;
    text-align: center;
    font-size: 16px;
    transition: ease-in-out 300ms;
}

#repository:hover {
    background-color: #1b2738;
    cursor: pointer;
```

```
    scale: 1.1;
}

.section-1 {
    width: 50%;
    height: 100vh;
    user-select: none;
}

.section-2 {
    width: 50%;
    height: 100vh;
    margin-top: 5%;
}

.R-H-img {
    width: 90%;
    height: 80%;



}

/* Section - 2 control penal code*/

.container-section-2 {
    width: 80%;
}


h1 {
    color: #333;
    margin-bottom: 40px;
}
```

```
.slider-wrapper {
    margin-bottom: 40px;
    position: relative;
}

.label-container {
    display: flex;
    justify-content: space-between;
    margin-bottom: 10px;
}

.slider-label {
    font-weight: bold;
    font-size: 18px;
    color: #333;
}

.slider-container {
    position: relative;
    height: 40px;
}

input[type="range"] {
    -webkit-appearance: none;
    width: 100%;
    height: 4vh;
    background: #ddd;
    border-radius: 10px;
    outline: none;
}

input[type="range"]::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
```

```
    height: 35px;
    width: 35px;
    border-radius: 50%;
    background: #f1c40f;
    border: 4px solid white;
    cursor: pointer;
    box-shadow: 0 0 2px rgba(0, 0, 0, 0.5);
    position: relative;
    z-index: 2;
}

input[type="range"]::-moz-range-thumb {
    height: 30px;
    width: 30px;
    border-radius: 50%;
    background: #f1c40f;
    border: 4px solid white;
    cursor: pointer;
}

.tooltip {
    position: absolute;
    background: #f1c40f;
    color: black;
    font-weight: bold;
    padding: 4px 10px;
    border-radius: 6px;
    top: -40px;
    transform: translateX(-50%);
    transition: left 0.1s ease;
}

.axis-value {
    position: absolute;
```

```
    right: 0;
    top: -10px;
    font-size: 14px;
    color: #444;
}

.pre-programmed-functions-container {
    width: 100%;
    height: auto;



}

.btn-style {
    width: 10vw;
    height: 8vh;
    outline: none;
    background-color: #f1c40f;
    border: none;
    color: #313131;
    font-weight: bold;
    padding: 10px;
    text-align: center;
    font-size: 16px;
    transition: ease-in-out 200ms;
    box-shadow: 0 8px 24px rgba(0, 0, 0, 0.12);
}

.btn-style:hover {
    cursor: pointer;
    scale: 1.1;
    box-shadow: 0 10px 26px rgba(0, 0, 0, 0.12);
}
```

```
.btn-style:active {
    background-color: #deb200;
}

/* Section 3 About the project */

.microcontroller-foto-container {
    width: 100%;
    height: 70vh;
    overflow: hidden;

}




.microcontroller-foto-container-2 {
    width: 60%;
    height: 100%;
}

.microcontroller-foto-container img {
    width: 100%;
    height: 100%;
    object-fit: cover;
    box-shadow: 2px 2px 3px rgba(131, 130, 130, 0.496);
}



.section-3-heading-container {

    margin: 15px;
}

.section-3-heading-container h1 {
```

```
    font-size: 2.5rem;
    font-weight: bold;

}

.microcontroller-features {
    width: 60%;
    margin: 15px;
    margin-left: 3rem;

}

.microcontroller-features p {
    line-height: 190%;
}

.sec3-micro-heading {
    width: 60%;
    margin: 15px;
    margin-left: 3rem;
    font-size: 24px;
    margin-top: 5%;

}

.sec3-material-used {
    width: 60%;
    margin: 15px;
    margin-left: 3rem;
    font-size: 24px;
    margin-top: 5%;
}

.sec-blockdiagram-container {
```

```
    width: 90vw;
    height: 60vh;
}

.sec-blockdiagram-container img {
    width: 100%;
    height: 100%;
    object-fit: contain;
}

/* Footer css*/

.footer{
    width: 100%;
    height: 20vh;
    background-color: #1b2738;
}

/* optimization  for small screen devices like cell phone my phone*/

@media (max-width: 575.98px) {


    .section-3-heading-container h1 {
        font-size: 32px;
        font-weight: bold;

    }

    .heading {
        width: 50%;
        height: 100%;
        display: flex;
        align-items: center;
```

```
    justify-content: center;
    text-align: center;
    margin: 0;
}

.heading h1 {
    font-size: 18px;
    margin: 0;
}

#repository {
    width: fit-content;
    height: 6vh;
    outline: none;
    background-color: #1b2738;
    border: none;
    color: white;
    padding: 10px;
    text-align: center;
    font-size: 16px;
    transition: ease-in-out 300ms;
    box-shadow: 2px 2px 4px rgba(131, 130, 130, 0.596);
}

.heading-container {
    display: flex;
    align-items: center;
    width: 100%;
    height: 10vh;
    justify-content: space-between;
}

.root {
    display: flex;
```

```
    flex-direction: column;
    width: 100%;
    height: fit-content;

}

.section-1 {
    width: 100%;
    height: fit-content;
    overflow: hidden;
    display: flex;
    justify-content: center;
    align-items: center;
}

input[type="range"]::-webkit-slider-thumb {
    -webkit-appearance: none;
    appearance: none;
    height: 30px;
    width: 30px;
    border-radius: 50%;
    background: #f1c40f;
    border: 4px solid white;
    cursor: pointer;
    box-shadow: 2px 2px 1px rgba(0, 0, 0, 0.2);
    position: relative;
    z-index: 2;
}

.section-2 {
    margin-top: 40px;
    width: 100%;
    height: fit-content;
}
```

```
.btn-style {
    width: fit-content;
    height: 10vh;
    outline: none;
    background-color: #ffd325;
    border: none;
    color: #313131;
    font-weight: bold;
    padding: 15px;

    text-align: center;
    font-size: 16px;
    transition: ease-in-out 200ms;
    box-shadow: 2px 2px 3px rgba(131, 130, 130, 0.496);
    margin: 0;
    margin-right: 6px;
}

.section-3 {
    margin: 0;
    margin-top: 40px;
}

.microcontroller-foto-container-2 {
    width: 90%;

    border-radius: 18px;
}

.microcontroller-foto-container-2 img {
    border-radius: 18px;
}
```

```
.sec3-micro-heading {
    width: 90%;
    margin: 0;
    padding-left: 15px;
}

.microcontroller-features {
    width: 90%;
    margin: 0;
    padding-left: 15px;
}

.sec3-material-used {
    width: 90%;
    margin: 0;
    padding-left: 15px;
    margin-top: 30px;
    line-height: 190%;
}
.main-container-about-us{
    line-height: 190%;
}

.sec3-micro-heading{
    line-height: 190%;
    margin-top: 30px;
    margin-bottom: 20px;
}

.pre-programmed-functions-container{
    overflow: hidden;
    flex-wrap: wrap;
    gap: 10px;
    height: auto;
```

```
        padding-top: 5px;
        padding-bottom: 5px;
    }
}
```

App.js ->

```
const delayMap = {};
const axisValues = { 1: 0, 2: 0, 3: 0, 4: 0 };



const ESP8266_IP = "192.168.43.2";  // <----------IP of esp8266

function updateSlider(slider, axisNumber) {
    const tooltip = document.getElementById("tooltip" + axisNumber);
    tooltip.textContent = slider.value;

    const tooltipX = (slider.value - slider.min) / (slider.max - slider.min) * slider.offsetWidth;
    tooltip.style.left = tooltipX + "px";

    axisValues[axisNumber] = parseInt(slider.value);

    clearTimeout(delayMap[axisNumber]);
    delayMap[axisNumber] = setTimeout(() => {
        console.log(`Sending to ESP: Axis ${axisNumber}: ${slider.value}°`);
        sendAxisData();
    }, 300);
}

function sendAxisData() {

    const url =
`http://${ESP8266_IP}/servoPos?x=${axisValues[1]}&y=${axisValues[2]}&z=${axisValues[3]}&g=${axisValues[4]}`;
    console.log(url);
    fetch(url)
```

```
        .then(response => {
            if (!response.ok) throw new Error("Network response was not ok");
            return response.text();
        })
        .then(data => {
            console.log("ESP Response:", data);
        })
        .catch(error => {
            console.error("Error contacting ESP8266:", error);
        });
}

// Initialize tooltips on page load
document.addEventListener("DOMContentLoaded", () => {
    document.querySelectorAll('input[type="range"]').forEach((slider, index) => {
        updateSlider(slider, index + 1);

        // Attach input event listener
        slider.addEventListener("input", () => {
            updateSlider(slider, index + 1);
        });
    });
});



function testESPLink() {
    const url = `http://${ESP8266_IP}/ping`
    fetch(url)
        .then(response => {
            if (response.ok) {
                alert("Pong");
            } else {
                alert("Not connected");
            }
```

```
    })
    .catch(() => {
      alert("Time out");
    });
}

// function PresetMotions(PresetMotionName){
//     const url = `http://${ESP8266_IP}/Preset?func=${PresetMotionName}`;
//     fetch(url)
//     .then(response =>{
//         if(response.ok){
//             alert(`Function ${PresetMotionName} done successful`);
//         }
//         else{
//             alert("Please check PING");
//         }
//     })
//     .catch(()=>{
//         alert('Some Error Accured!');
//     })
// }

function PresetMotions(PresetMotionName) {
  // If the demo button is pressed
  if (PresetMotionName === "demo") {
    // Update the axisValues to the initial position
    axisValues[1] = 35;
    axisValues[2] = 50;
    axisValues[3] = 22;
    axisValues[4] = 0;

    // Step 1: Move to the initial position first
    const initUrl = `http://${ESP8266_IP}/servoPos?x=35&y=50&z=22&g=0`;
    fetch(initUrl)
```

```
        .then(response => {
            if (!response.ok) throw new Error("Failed to move to initial position");
            return response.text();
        })
        .then(() => {
            console.log("Initial position reached. Now running demo...");

            // Step 2: Call the preset motion after short delay
            setTimeout(() => {
                const url = `http://${ESP8266_IP}/Preset?func=${PresetMotionName}`;
                fetch(url)
                    .then(response => {
                        if (response.ok) {
                            alert(`Function ${PresetMotionName} done successfully`);
                        } else {
                            alert("Please check PING");
                        }
                    })
                    .catch(() => {
                        alert('Some Error Occurred while running preset!');
                    });
            }, 1000); // 1 second delay
        })
        .catch(error => {
            alert('Error during initial positioning: ' + error.message);
        });
    }
    else {
        // For other preset motions, call directly
        const url = `http://${ESP8266_IP}/Preset?func=${PresetMotionName}`;
        fetch(url)
            .then(response => {
                if (response.ok) {
                    alert(`Function ${PresetMotionName} done successfully`);
```

```
            } else {
                alert("Please check PING");
            }
        })
        .catch(() => {
            alert('Some Error Occurred!');
        });
    }
}
```

CHAPTER V: CONCLUSION

The development of the Robotic Arm project has provided a practical and innovative solution toward addressing the growing need for low-cost, customizable automation systems in India. Through the integration of hardware and software, the project successfully demonstrates a compact, four-axis robotic hand capable of being controlled via both a local OLED interface and a web-based dashboard.

The use of the ESP12F microcontroller has allowed for a dual-interface system that enhances flexibility and ease of control. With custom firmware, responsive web design, and local hosting via Node.js, the entire setup functions efficiently without reliance on external servers or cloud services. This self-contained approach ensures both reliability and speed—two essential factors in factory-like environments.

Moreover, the modular design of the firmware and the console allows future improvements and specific reprogramming for other applications, making the system highly adaptable. Overall, this project not only showcases the potential of locally engineered automation solutions but also lays the foundation for further research, development, and scaling in the field of affordable industrial robotics.

5.1 Limitations

While the Robotic Arm project demonstrates effective control and automation using affordable components, there are several limitations that restrict its scalability and precision for industrial-grade applications.

The ESP12F, based on the ESP8266 architecture, offers limited GPIO pins, which constrains the number of peripherals that can be connected simultaneously. This poses challenges when scaling the system to more complex robotic designs or additional sensors.

The use of hobby-grade servo motors introduces further limitations. These servos lack positional feedback, meaning the system cannot confirm whether a movement was executed accurately. The firmware operates under the assumption that commands are always followed correctly, which may result in cumulative positional errors. Additionally, hobby servos have restricted torque, making them incapable of lifting heavier objects—limiting the system to lightweight demonstrations only.

Due to the absence of closed-loop control, the system's precision is relatively low. It requires the robotic arm to start from a manually defined position each time for predictable movements. Any deviation from this reference point results in reduced motion accuracy.

Physically, the compact size of the robotic hand restricts its application to small-scale tasks. Furthermore, while the web interface offers convenience, it also introduces latency issues that can impact real-time responsiveness, particularly over less stable local networks.

Despite these limitations, the project serves as a strong proof of concept and highlights areas for future improvement in hardware design, control algorithms, and communication protocols.

REFERENCES

1. Espressif Systems – *ESP8266 Technical Reference Manual*, https://www.espressif.com/en/products/socs/esp8266
2. SSD1306 OLED Display Datasheet – Adafruit Industries, https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf
3. Arduino IDE Documentation – Arduino.cc, https://www.arduino.cc/en/Guide
4. CH340G USB Driver – WCH Electronics, http://www.wch.cn/download/CH341SER_EXE.html
5. Servo Motor Basics – SparkFun Electronics, https://learn.sparkfun.com/tutorials/hobby-servo-tutorial
6. Node.js Official Documentation – Node.js Foundation, https://nodejs.org/en/docs/
7. ESPAsyncWebServer Library – GitHub Repository, https://github.com/me-no-dev/ESPAsyncWebServer
8. OLED Bitmap Image Converter Tool – https://javl.github.io/image2cpp/
9. W3Schools Responsive Web Design – https://www.w3schools.com/html/html_responsive.asp
10. PlatformIO for ESP Development – PlatformIO IDE, https://platformio.org/