Internship at Brahmastra Aerospace & Defence

Submitted in fulfillment for Summer Industrial Internship credits

Bachelor of Technology

In

Electronics and Instrumentation Engineering

By

J. Ranjith Sharan
22BEI0106

ESP32 based Drone

October 2024

DECLARATION

I, J. Ranjith Sharan (22BEI0106), hereby declare that the report I am submitting as a fulfillment of my summer industrial internship, titled *"ESP32-Based Drone,"* is a genuine record of the work I undertook, along with the insights and observations made throughout the internship program conducted from June 2024 to July 2024.

I have prepared this document with respect to professional standards and ethics, ensuring originality and accuracy in all its parts. To the best of my knowledge, this report is my own work and has not been reproduced, copied, or derived from any other source.

[illegible]

Place: VIT Vellore   Signature of Candidate

Date: 6th November 2024

ACKNOWLEDGEMENT

I extend my gratitude to the VIT management, the School of Electrical Engineering (SELECT), Dr. Mathew M. Noel, Dean of SELECT, Dr. Rajini K., Head of the Department of Electronics and Instrumentation, and my proctor, Dr. B. Jaganatha Pandian, SELECT, for their motivation and support throughout my internship.

It is with great pleasure and heartfelt gratitude that I express my sincere thanks to my guide, Mr. K. Dinesh Hari Haran, Drone engineer at Brahmastra Aerospace & Defence, Chennai. His guidance, motivation, and continuous encouragement throughout my internship enabled me to successfully complete my off-campus summer internship at the organization.

I sincerely thank Mr. Subash P Kuppusamy, Founder & CEO of Brahmastra Aerospace & Defence, for placing his trust in me and providing me with this valuable internship opportunity.

[illegible]

Place: VIT Vellore

Signature of Candidate

Date: 6th November 2024

brahmàstra
AEROSPACE & DEFENCE

CIN:U29100TN2021PTC145913
GST: 33AAKCB1104F1ZD
Contact: 9003900296

22BEI0106/BRHM/IN/2024

7/16/2024 4:20:05 PM/Chennai, India.

CERTIFICATE / LOR

We are glad to inform you that **J. Ranjith Sharan** from **Vellore Institute of Technology** has successfully completed the internship program at Brahmastra Aerospace & Defence Private Limited from **2024-06-05** to **2024-07-12**

During the internship, the candidate was exposed to various activities and discussions in the internship division, the candidate worked as an Electronics Research & Development intern and the candidate successfully completed the project entitled **ESP32 based drone** at Brahmastra Aerospace & Defence Private Limited.

We found the candidate to be extremely inquisitive and hard-working. The candidate was very interested in learning the functions of our core division and willing to put their best efforts to understand the subject and willing to work towards creative thinking. The candidate worked with various software and met the objectives of the project.

The candidate's association with us was very fruitful and we wish him all the best in their future endeavors.

For Brahmastra Aerospace and Defence Pvt. Ltd.
[illegible]
Director.

Subash P Kuppusamy,

Founder & CEO

"Established a collaborative partnership with"

Skill India SYDNEY

brahmastraspace
No.3, Balakrishnan street,
Nanmangalam, Chennai-129.

INDEX

OBJECTIVE

The primary objective of this internship was to design and develop a Wi-Fi-controlled drone prototype using the ESP32 microcontroller, emphasizing precise motor control, power management, and real-time remote operability. This project aimed to strengthen the practical understanding of embedded systems by integrating essential hardware components, including 716 micro-coreless motors, MOSFETs, and custom circuits for stable power delivery and motor control. Leveraging the ESP32's built-in Wi-Fi capabilities, the project included the creation of a web-based interface, allowing users to remotely adjust motor speeds and dynamically control the drone's movements. This setup required a solid understanding of PWM (Pulse Width Modulation) for motor speed regulation, as well as protective circuitry with flyback diodes and capacitors to prevent voltage spikes and ensure efficient operation.

Beyond technical assembly, the internship provided opportunities to develop problem-solving skills, focusing on real-world challenges such as maintaining a stable power supply, synchronizing motor speeds, and ensuring reliable Wi-Fi connectivity. Through iterative testing and firmware adjustments, the project established a responsive control system, enabling basic hovering capability. This hands-on experience offered insights into UAV design, IoT-enabled control systems, and embedded hardware. Future recommendations include motor upgrades and interface optimization to enhance control and mobility, laying a solid foundation for further exploration in IoT-driven drone applications and autonomous flight systems.

INTRODUCTION

Drones have revolutionized numerous industries, from agriculture to surveillance, due to their versatility and adaptability in diverse applications. The demand for compact, affordable, and customizable drones has sparked interest in building UAVs with open-source microcontrollers like the ESP32. Leveraging the ESP32's built-in Wi-Fi capabilities, this project aimed to design a drone controlled through a mobile interface, enabling real-time communication and control. The ESP32 microcontroller serves as the primary processing unit, interpreting sensor data and managing flight dynamics through custom firmware. Integrating components such as 716 micro coreless motors, custom Electronic Speed Controllers (ESCs), and essential sensors like the MPU6050 accelerometer and magnetometer allows for stable flight and responsive control.

The project required a multidisciplinary approach, involving motor control, sensor fusion, and wireless communication. Developing a drone presents challenges, including maintaining synchronized motor speeds, ensuring adequate power supply, and achieving optimal thrust-to-weight ratio. The inclusion of custom ESCs tailored for the coreless motors allowed for precise control over the drone's movement. The MPU6050 sensor provided real-time orientation data to stabilize the drone, which, combined with mobile-based Wi-Fi control, demonstrated the feasibility of an IoT-enabled UAV. The project's outcomes highlight the ESP32's potential for remote and autonomous UAV applications, showcasing a low-cost approach to building functional drones with IoT integration for a wide range of uses.

OVERVIEW OF INDUSTRY

Brahmastra Aerospace & Defence:

brahmàstra AEROSPACE & DEFENCE

ब्रह्मास्त्रं विजयते
| VIMANIK SHIKSHAN |
| VIMANIK NIRMAN |

Brahmastra Research & Development Center (BRDC)

Brahmastra Aerospace & Defence, recently renamed as Brahmastra Research & Development Center (BRDC), is a cutting-edge organization focused on strengthening India's defense sector through innovative research and technology in aerospace, UAVs, and defense systems. Established with a commitment to driving India's self-reliance in defense, BRDC has cultivated partnerships with prominent institutions, including the Indian Space Research Organisation (ISRO), the Ministry of Defence (MOD), and the MSME sector, to advance defense R&D and training in UAV operations. BRDC is also involved with the US Embassy's STEAM Mission in India, collaborating on initiatives that blend science, technology, engineering, arts, and mathematics to encourage innovation in defense applications. This network of collaborations allows BRDC to push the boundaries of UAV and defense technology, positioning India as a global leader in defense innovation.

One of BRDC's core missions is to develop state-of-the-art UAVs and AI-driven defense solutions tailored to India's unique defense needs. The organization emphasizes indigenous technology development, producing UAVs capable of performing high-stakes operations like surveillance, reconnaissance, and disaster response. Its focus on UAV development extends beyond military applications, as BRDC also explores commercial and industrial applications of drone technology, thereby expanding India's UAV manufacturing and innovation ecosystem. By fostering an environment of collaboration and technical expertise, BRDC aims to provide India with locally developed defense solutions, reducing dependency on foreign imports and enhancing national security.

BRDC's Skilltech Contributions significantly impact workforce development in India's defense sector. Through advanced training programs in UAV operations, aerospace engineering, and Industry 4.0 technologies, BRDC empowers India's technical workforce to meet the evolving demands of modern defense industries. BRDC collaborates closely with the Ministry of Defence and MSME to deliver specialized UAV training, helping individuals excel in defense roles by equipping them with the knowledge and skills essential for operating advanced UAV systems. Additionally, BRDC provides consulting services for Bollywood movies to ensure accurate depictions of aerospace and defense-related themes. This unique contribution not only boosts the industry's authenticity in media but also raises awareness of aerospace technology among the general public.

BRDC's vision for the future is aligned with India's goal of becoming a self-reliant defense power. The organization is dedicated to advancing India's capabilities in aerospace and defense technology, focusing on sustainable practices, quality production, and a skilled workforce. By leveraging its partnerships and training programs, BRDC fosters the next generation of defense professionals who will continue to propel India toward technological sovereignty.

The Drone Industry:

The drone, or UAV (Unmanned Aerial Vehicle), industry has undergone rapid growth over the last decade, expanding from defense and surveillance into commercial sectors such as agriculture, logistics, and environmental monitoring. Drones are integral in modern defense for remote surveillance and reconnaissance missions, enabling safe, efficient operations. In civilian applications, UAVs have proven indispensable in fields such as precision agriculture, infrastructure inspection, and disaster response. Their versatility, cost-effectiveness, and ability to access hazardous or remote areas make them invaluable assets across sectors.

Technological advancements, such as miniaturized sensors, improved battery life, and enhanced wireless communication, have been instrumental in UAV evolution. The ESP32 microcontroller, for instance, has become popular in both DIY and commercial drone applications due to its real-time communication capabilities. Sensors like accelerometers, gyroscopes, and magnetometers enable drones to maintain stability, detect orientation, and adjust to environmental conditions, adding reliability to their operation. As these technologies become more accessible, commercial applications continue to grow, showing the versatility of drones in addressing a variety of societal and industrial needs.

METHODOLOGY

1. Research and Preparations

The initial phase involved a comprehensive review of existing literature, tutorials, and technical documentation to gather foundational knowledge on Wi-Fi-controlled drones using ESP32. Relevant microcontroller guides, and community tutorials provided insights into motor control, PWM (Pulse Width Modulation) signals, Wi-Fi connectivity, and electronic speed controllers (ESCs). I also learned about serial communication protocols required to communicate with the sensors which I had to interface with the ESP32. This research not only informed the project's design but also highlighted best practices for implementing reliable wireless communication and ensuring stable motor operation.

Through this exploratory phase, I familiarized myself with the ESP32's capabilities, including its GPIO configurations, Wi-Fi functions, and PWM support for motor speed control. I explored multiple sources that demonstrated ways to handle real-time control using web interfaces and the essential techniques for synchronizing motors in a multi-motor setup. By consolidating this information, I laid the groundwork for the project's architecture and feature requirements, ensuring both technical feasibility and optimal performance.

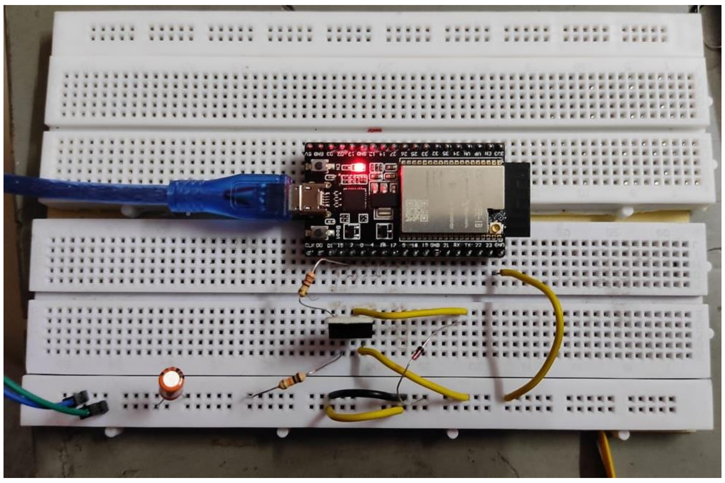
2. Hardware Components and Configuration

The hardware setup involved multiple components, each carefully chosen to ensure efficiency, compatibility, and stability in drone operation. The following is a breakdown of each component and its purpose:

- **ESP32-S2-WROVER**: This microcontroller served as the main processing unit, chosen for its built-in Wi-Fi functionality and sufficient GPIO pins for motor control. It managed real-time control commands and PWM signals, which regulated motor speeds.
- **MPU6050:** This is a 6-axis motion tracking device that combines a 3-axis gyroscope and a 3-axis accelerometer. It provides real-time data on angular velocity and linear acceleration, allowing for precise motion detection and stabilization in applications like drones.
- **HMC5883:** This is a 3-axis digital compass (magnetometer) that measures the magnetic field's strength and direction. It provides orientation information relative to the Earth's magnetic field, which is essential for navigation and maintaining heading stability in drone flight.
- **716 Micro Coreless Motors**: Four lightweight motors were selected for their high-speed operation and compatibility with a 1s LiPo battery, providing the necessary thrust for hover capabilities.
- **MOSFET (IRFZ44N)**: A MOSFET was used to control each motor, functioning as a high-speed switch regulated by the ESP32's PWM signals. The IRFZ44N

MOSFET was chosen for its ability to handle the current requirements of the 716 micro-coreless motors without overheating.

- **1s LiPo Battery**: Two lithium polymer batteries provided power to both the ESP32 and the motors separately. The 1s (3.7V) configuration was lightweight yet sufficient for running the microcontroller and driving the motors at appropriate speeds.
- **10uF Electrolytic Capacitor**: Placed across the battery terminals, this capacitor smoothed out voltage fluctuations, protecting the ESP32 and motors from potential dips during operation.
- **1N5819 Flyback Diode**: Installed across each motor, this diode protected the circuit by allowing current generated by the motors to dissipate safely, preventing back-EMF from damaging the MOSFETs.
- **Resistors**: A 10kΩ resistor was used as a pull-down for the MOSFET gate, and a 220Ω resistor was used for gate current limiting, ensuring stable operation without drawing excessive current from the ESP32.



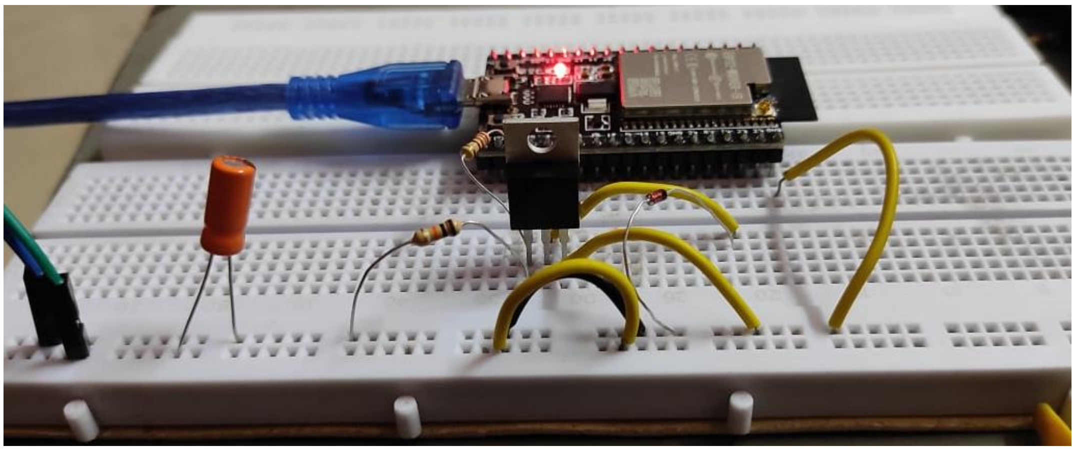


Figure 1 - Motor PWM testing in ESP32

3. Circuit Design and Connections:

a) Overall Circuit Connections Assembly:

MPU6050 (Gyroscope and Accelerometer):

- VCC → 3.3V (ESP32)
- GND → GND (ESP32)
- SDA → GPIO 21 (ESP32)
- SCL → GPIO 22 (ESP32)

HMC5883 (Magnetometer):

- VCC → 3.3V (ESP32)
- GND → GND (ESP32)
- SDA → GPIO 21 (ESP32)
- SCL → GPIO 22 (ESP32)

1s LiPo Battery: Provides power to both the ESP32 and the motors.

10µF Electrolytic Capacitor: Installed across the battery terminals to smooth out voltage fluctuations.

b) ESC Connections:

- **MOSFET Control Circuit**:
  - **Gate (G)**: Connected to a GPIO pin on the ESP32 (GPIO 12, 26, 27, 32) through a 220Ω current-limiting resistor. A 10kΩ pull-down resistor connected the gate to ground, ensuring the MOSFET remains off when not actively driven.
  - **Source (S)**: Connected directly to the ground.
  - **Drain (D)**: Connected to one terminal of the motor, allowing controlled current flow from the battery to the motor.
- **Motor Connections**:
  - Each motor was connected to a MOSFET, with one motor terminal attached to the MOSFET's drain and the other to the positive terminal of the 1s LiPo battery.
  - The negative terminal of the battery was connected to the ground, creating a complete circuit when the MOSFET was activated.
- **Flyback Diode**:

  The cathode of the 1N5819 diode was connected to the positive motor terminal, while the anode was connected to the negative battery terminal. This orientation prevented back-EMF, protecting both the motor and the MOSFET during sudden changes in motor speed.

- **Capacitor Placement**:

  The 10uF capacitor was installed between the positive and negative terminals of the LiPo battery to stabilize voltage levels, ensuring smooth operation of both the ESP32 and motors.

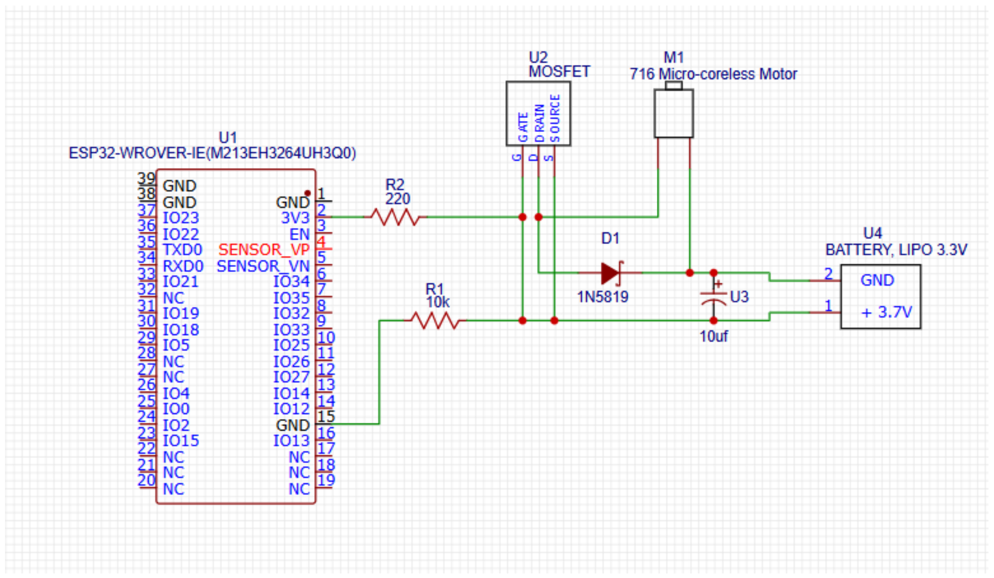
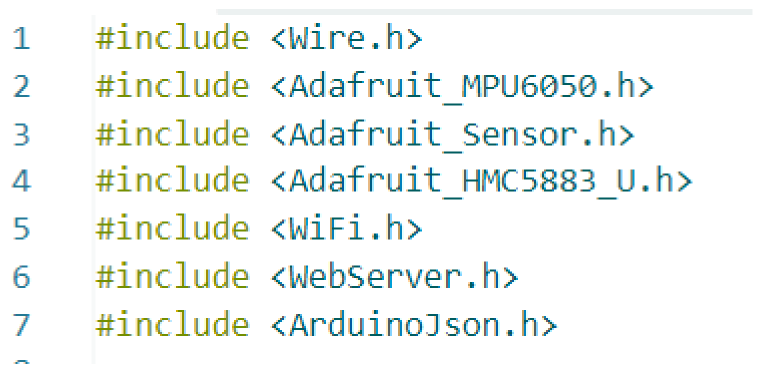


Figure 2 - Schematic of ESC for a single motor

4. Software Code Overview:

a) Library Inclusions

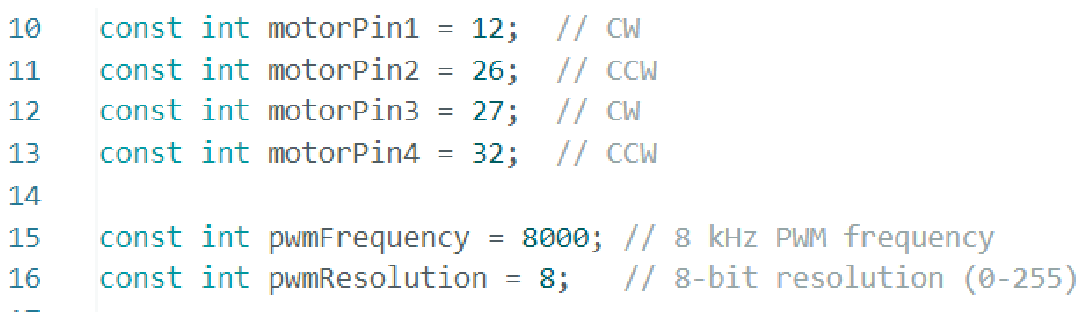


```
#include <Wire.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>
#include <WiFi.h>
#include <WebServer.h>
#include <ArduinoJson.h>
```

- **Wire.h**: This library is essential for I2C communication, which is used to interact with the MPU6050 and HMC5883 sensors.
- **Adafruit_MPU6050.h & Adafruit_Sensor.h**: These libraries provide an interface to the MPU6050 accelerometer and gyroscope sensor, enabling reading of motion data.

- **Adafruit_HMC5883_U.h**: This library interfaces with the HMC5883 digital compass, which helps in determining the heading of the drone.
- **WiFi.h**: This library is necessary for establishing WiFi connections.
- **WebServer.h**: This library allows the ESP32 to handle incoming HTTP requests.
- **ArduinoJson.h**: Though not used in the code provided, this library is typically included for JSON serialization and deserialization, useful for data exchange.

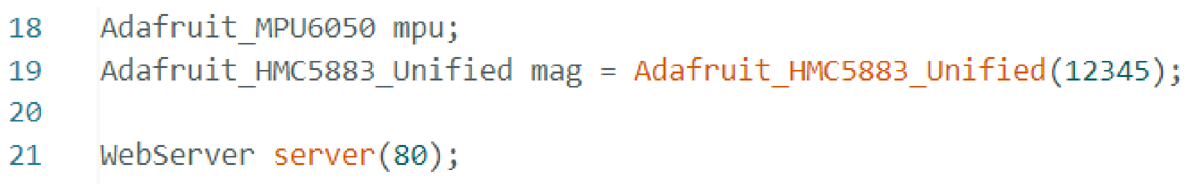
b) Pin Definitions and Constants



```
const int motorPin1 = 12;  // CW
const int motorPin2 = 26;  // CCW
const int motorPin3 = 27;  // CW
const int motorPin4 = 32;  // CCW

const int pwmFrequency = 8000; // 8 kHz PWM frequency
const int pwmResolution = 8;   // 8-bit resolution (0-255)
```

- **Motor Pins**: Defines the GPIO pins connected to the motor driver. Each motor is assigned a specific pin for controlling its speed and direction (CW = clockwise, CCW = counter-clockwise).
- **PWM Settings**: The PWM frequency is set to 8000 Hz with an 8-bit resolution, allowing for motor speed control from 0 to 255.

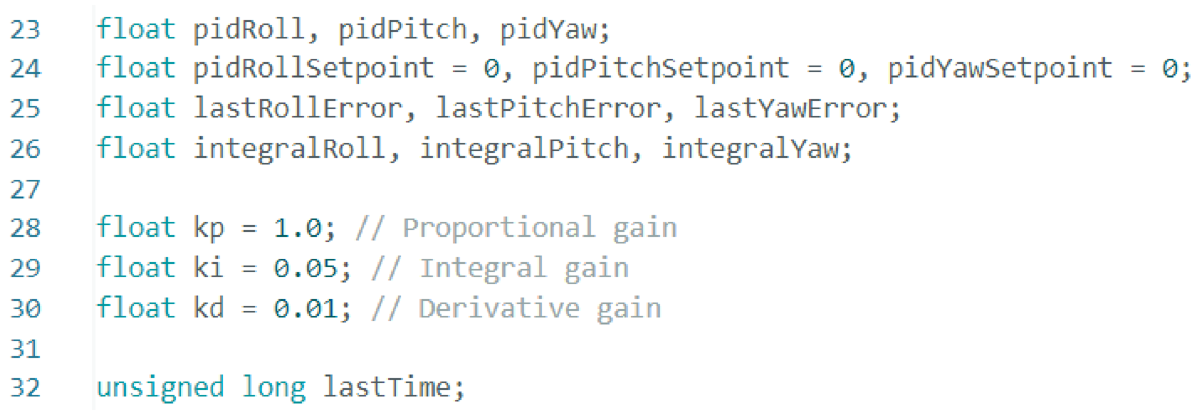
c) Object Instantiation



```
Adafruit_MPU6050 mpu;
Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);

WebServer server(80);
```

- **MPU6050 Instance**: An object to interact with the MPU6050 sensor.
- **HMC5883 Instance**: An object to interface with the HMC5883 compass, initialized with a unique identifier (12345).
- **Web Server**: Creates an instance of the web server listening on port 80 for HTTP requests.

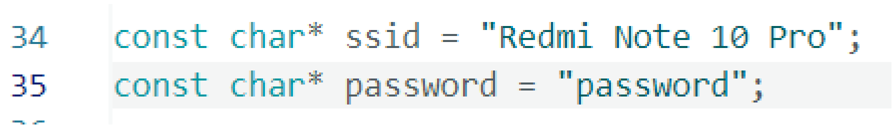
d) PID Control Variables



```
float pidRoll, pidPitch, pidYaw;
float pidRollSetpoint = 0, pidPitchSetpoint = 0, pidYawSetpoint = 0;
float lastRollError, lastPitchError, lastYawError;
float integralRoll, integralPitch, integralYaw;

float kp = 1.0; // Proportional gain
float ki = 0.05; // Integral gain
float kd = 0.01; // Derivative gain

unsigned long lastTime;
```

- **PID Control Variables**: These variables are used for the PID control loop, which helps maintain the drone's stability by adjusting motor speeds based on the roll, pitch, and yaw errors.
- **Setpoints**: Target values for roll, pitch, and yaw, initially set to zero, representing a level flight state.
- **PID Coefficients**: The proportional (kp), integral (ki), and derivative (kd) gains define the response characteristics of the PID controller.
- **Time Tracking**: lastTime is used to measure elapsed time for the PID calculations.

e) WiFi Configuration

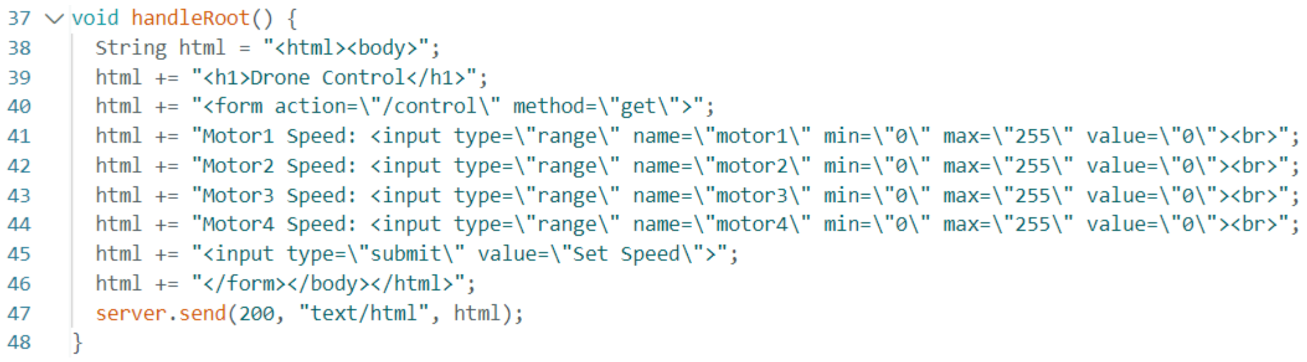


```
const char* ssid = "Redmi Note 10 Pro";
const char* password = "password";
```

- **WiFi Credentials**: The SSID and password for connecting the ESP32 to a WiFi network. Make sure to use secure passwords in real-world applications.

f) Web Server Handlers

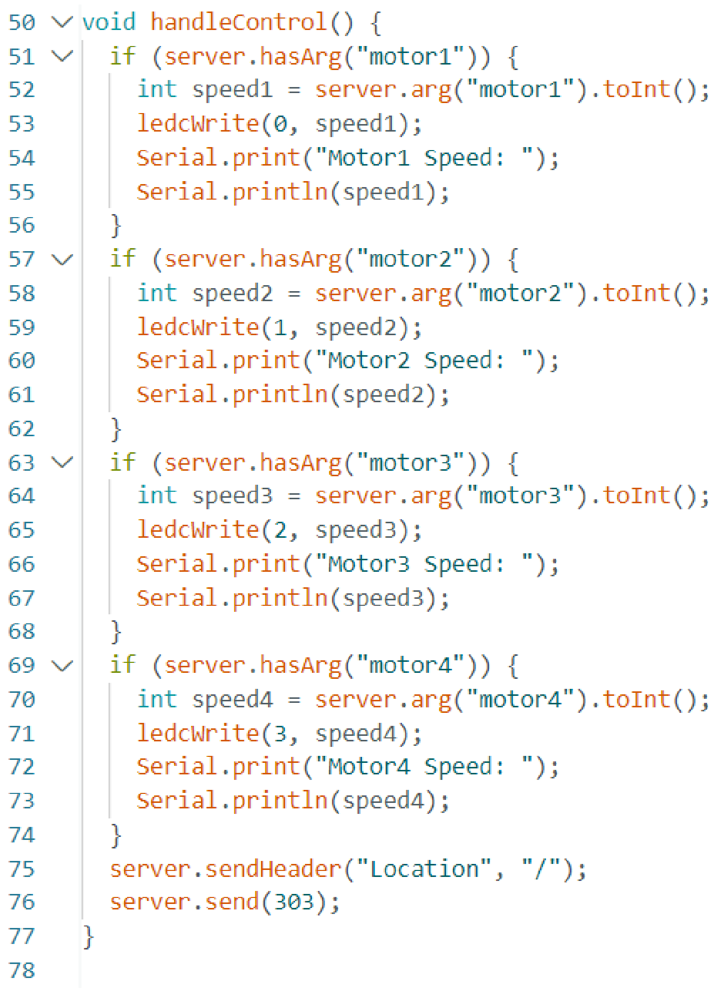
i) Root Handler



```
void handleRoot() {
  String html = "<html><body>";
  html += "<h1>Drone Control</h1>";
  html += "<form action=\"/control\" method=\"get\">";
  html += "Motor1 Speed: <input type=\"range\" name=\"motor1\" min=\"0\" max=\"255\" value=\"0\"><br>";
  html += "Motor2 Speed: <input type=\"range\" name=\"motor2\" min=\"0\" max=\"255\" value=\"0\"><br>";
  html += "Motor3 Speed: <input type=\"range\" name=\"motor3\" min=\"0\" max=\"255\" value=\"0\"><br>";
  html += "Motor4 Speed: <input type=\"range\" name=\"motor4\" min=\"0\" max=\"255\" value=\"0\"><br>";
  html += "<input type=\"submit\" value=\"Set Speed\">";
  html += "</form></body></html>";
  server.send(200, "text/html", html);
}
```

- **HTML Generation**: Constructs an HTML page with a form for controlling motor speeds using sliders. Each slider corresponds to a motor and allows users to input speed values from 0 to 255.
- **Form Submission**: The form sends a GET request to the /control endpoint when submitted, containing the motor speed values as query parameters.

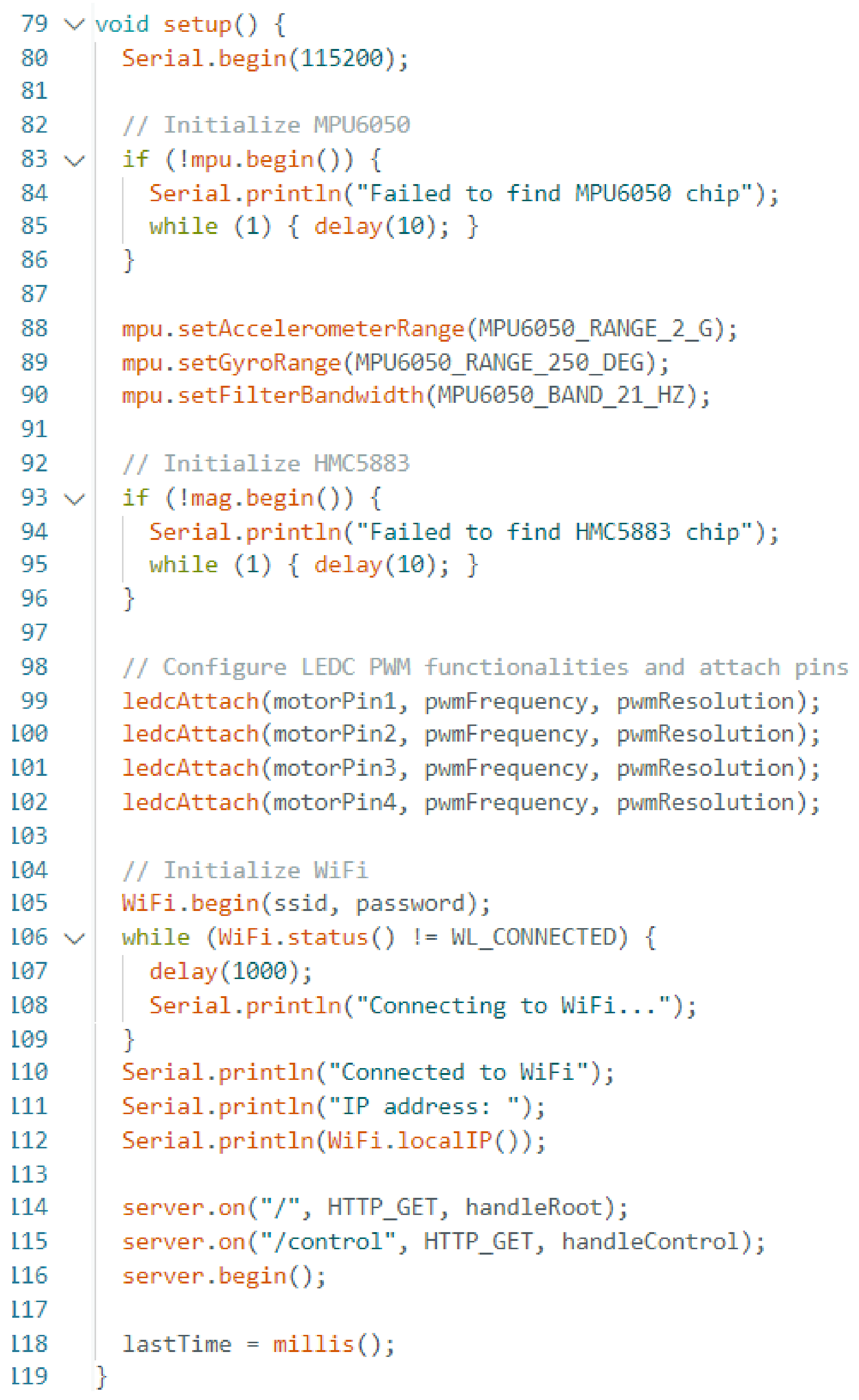
ii) Control Handler



```
void handleControl() {
  if (server.hasArg("motor1")) {
    int speed1 = server.arg("motor1").toInt();
    ledcWrite(0, speed1);
    Serial.print("Motor1 Speed: ");
    Serial.println(speed1);
  }
  if (server.hasArg("motor2")) {
    int speed2 = server.arg("motor2").toInt();
    ledcWrite(1, speed2);
    Serial.print("Motor2 Speed: ");
    Serial.println(speed2);
  }
  if (server.hasArg("motor3")) {
    int speed3 = server.arg("motor3").toInt();
    ledcWrite(2, speed3);
    Serial.print("Motor3 Speed: ");
    Serial.println(speed3);
  }
  if (server.hasArg("motor4")) {
    int speed4 = server.arg("motor4").toInt();
    ledcWrite(3, speed4);
    Serial.print("Motor4 Speed: ");
    Serial.println(speed4);
  }
  server.sendHeader("Location", "/");
  server.send(303);
}
```

- **Parameter Handling**: Checks if motor speed arguments are present in the request. For each motor, it retrieves the speed value and writes it to the corresponding PWM channel using ledcWrite().
- **Serial Output**: Prints the speed values to the Serial Monitor for debugging purposes.
- **Redirect**: After processing, it sends a 303 redirect to the root page, allowing users to see the updated control interface.

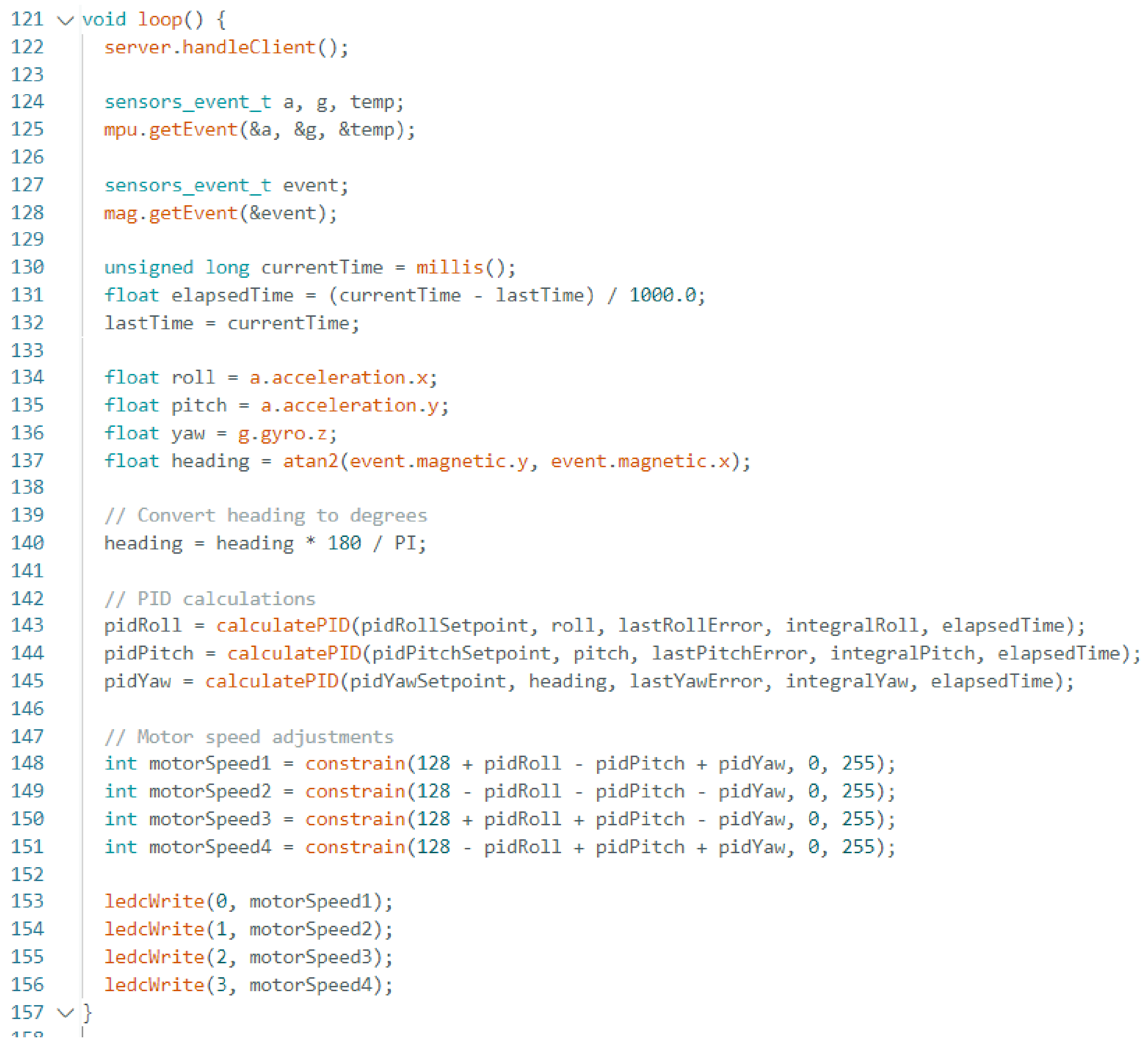
g) Setup Function



```
void setup() {
  Serial.begin(115200);

  // Initialize MPU6050
  if (!mpu.begin()) {
    Serial.println("Failed to find MPU6050 chip");
    while (1) { delay(10); }
  }

  mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
  mpu.setGyroRange(MPU6050_RANGE_250_DEG);
  mpu.setFilterBandwidth(MPU6050_BAND_21_HZ);

  // Initialize HMC5883
  if (!mag.begin()) {
    Serial.println("Failed to find HMC5883 chip");
    while (1) { delay(10); }
  }

  // Configure LEDC PWM functionalities and attach pins
  ledcAttach(motorPin1, pwmFrequency, pwmResolution);
  ledcAttach(motorPin2, pwmFrequency, pwmResolution);
  ledcAttach(motorPin3, pwmFrequency, pwmResolution);
  ledcAttach(motorPin4, pwmFrequency, pwmResolution);

  // Initialize WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  server.on("/", HTTP_GET, handleRoot);
  server.on("/control", HTTP_GET, handleControl);
  server.begin();

  lastTime = millis();
}
```

- **Serial Communication**: Initializes serial communication at 115200 baud for debugging.
- **Sensor Initialization**: Checks for the presence of the MPU6050 and HMC5883 sensors and configures their settings. If initialization fails, the program enters an infinite loop.
- **PWM Configuration**: The ledcAttach() function configures the PWM channels for the motors.
- **WiFi Connection**: Initiates a connection to the specified WiFi network and waits until connected, printing the assigned IP address.
- **Server Start**: Sets up the server to handle incoming requests.

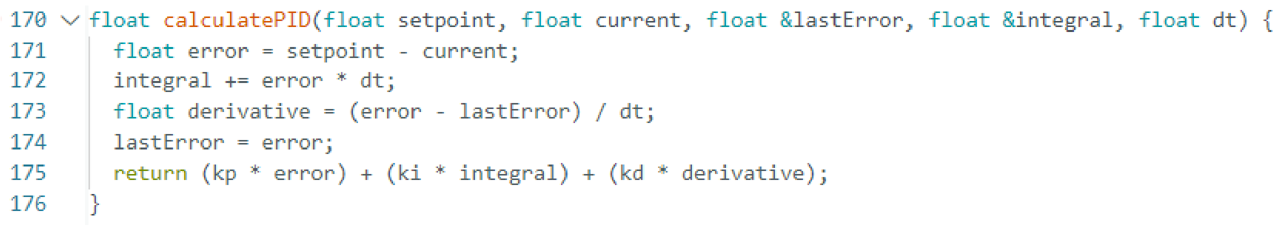
h) Sensor Data Acquisition and PID Control



```
void loop() {
  server.handleClient();

  sensors_event_t a, g, temp;
  mpu.getEvent(&a, &g, &temp);

  sensors_event_t event;
  mag.getEvent(&event);

  unsigned long currentTime = millis();
  float elapsedTime = (currentTime - lastTime) / 1000.0;
  lastTime = currentTime;

  float roll = a.acceleration.x;
  float pitch = a.acceleration.y;
  float yaw = g.gyro.z;
  float heading = atan2(event.magnetic.y, event.magnetic.x);

  // Convert heading to degrees
  heading = heading * 180 / PI;

  // PID calculations
  pidRoll = calculatePID(pidRollSetpoint, roll, lastRollError, integralRoll, elapsedTime);
  pidPitch = calculatePID(pidPitchSetpoint, pitch, lastPitchError, integralPitch, elapsedTime);
  pidYaw = calculatePID(pidYawSetpoint, heading, lastYawError, integralYaw, elapsedTime);

  // Motor speed adjustments
  int motorSpeed1 = constrain(128 + pidRoll - pidPitch + pidYaw, 0, 255);
  int motorSpeed2 = constrain(128 - pidRoll - pidPitch - pidYaw, 0, 255);
  int motorSpeed3 = constrain(128 + pidRoll + pidPitch - pidYaw, 0, 255);
  int motorSpeed4 = constrain(128 - pidRoll + pidPitch + pidYaw, 0, 255);

  ledcWrite(0, motorSpeed1);
  ledcWrite(1, motorSpeed2);
  ledcWrite(2, motorSpeed3);
  ledcWrite(3, motorSpeed4);
}
```

- **Request Handling**: Continuously processes incoming client requests. Each time a request is received, the appropriate handler function is called based on the URL path.
- **Sensor Data Retrieval**: Obtains acceleration and gyro data from the MPU6050. This data is essential for determining the current orientation of the drone.
- **PID Calculations**: Uses the retrieved sensor values to calculate errors for roll, pitch, and yaw. The PID control function is called to compute adjustments based on the setpoints.

i) PID Calculation Function



```
float calculatePID(float setpoint, float current, float &lastError, float &integral, float dt) {
  float error = setpoint - current;
  integral += error * dt;
  float derivative = (error - lastError) / dt;
  lastError = error;
  return (kp * error) + (ki * integral) + (kd * derivative);
}
```

- **PID Control Logic**: This function calculates the control output based on the setpoint, current value, and previous errors. It updates the integral and derivative components, providing feedback to maintain stability.

5. Testing and Calibration

Several rounds of testing were conducted to ensure reliable performance and address any issues with motor synchronization or Wi-Fi control:

- **Initial Motor Testing**:

  Test programs were developed to verify that each motor responded to PWM signals. Fixed PWM values were applied to each motor to check if it ran at expected speeds, validating the MOSFET connections and flyback diode placement.

- **Wi-Fi and Control Testing**:

  Once individual motor functionality was confirmed, the final code was tested for Wi-Fi control. Adjustments were made to the PWM duty cycles and interface layout to ensure consistent response and ease of control.

- **Calibration**:

  Minor discrepancies in motor speeds were observed due to differences in PWM responses. Calibration was performed by fine-tuning the duty cycles, achieving balanced thrust across all motors for stable hover.

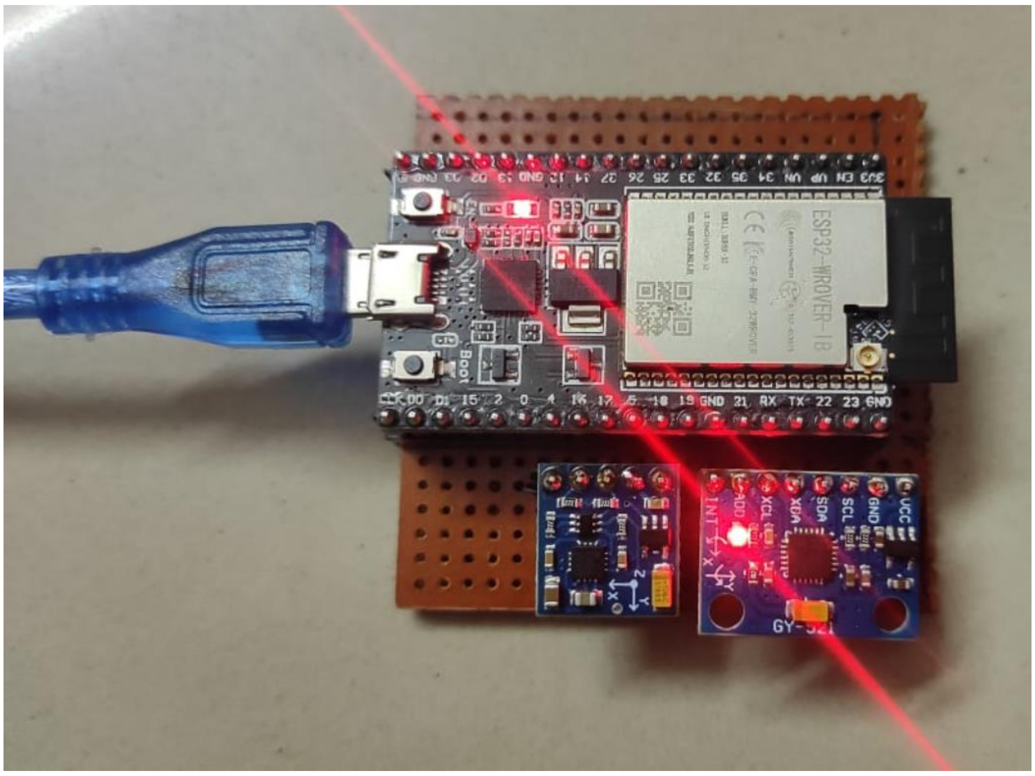


Figure 3 - Sensors integration and testing with ESP32

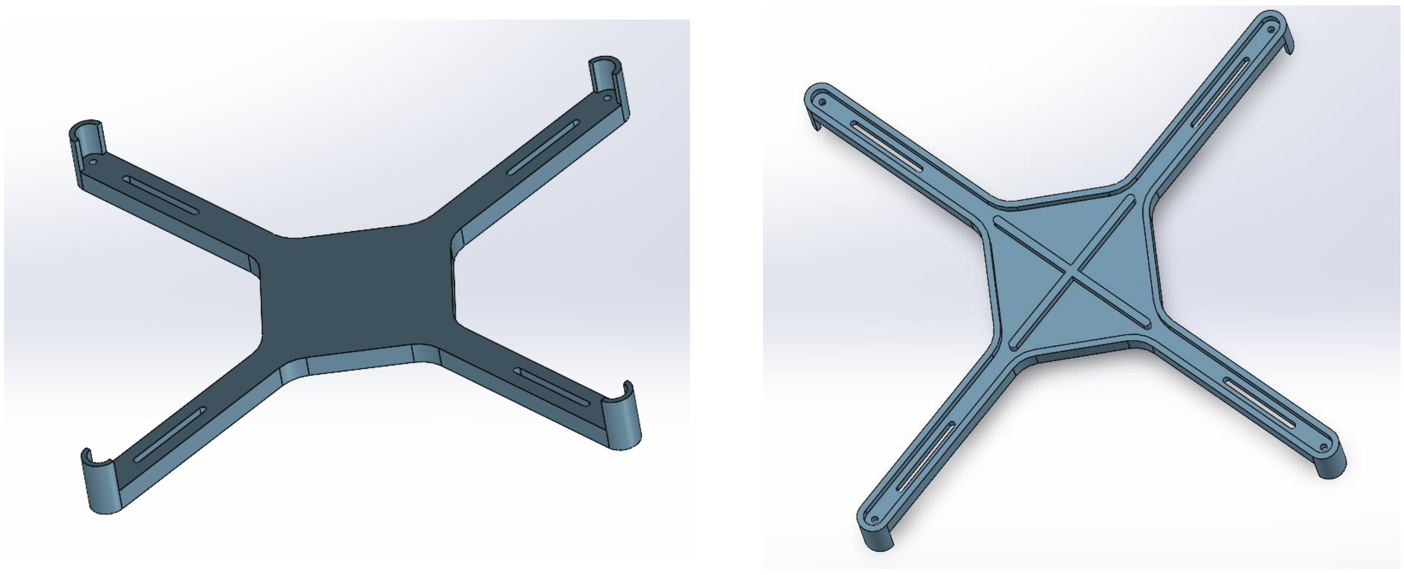


Figure 4 - 3D CAD model of the Drone frame

6. Challenges and Optimizations

Several challenges were encountered, and optimizations were implemented to improve performance:

- **Wi-Fi Stability**: Initial connectivity issues were resolved by ensuring close proximity to the router and adjusting ESP32 settings for stable operation.
- **Power Fluctuations**: Using separate power connections for the ESP32 and motors reduced power dips, stabilizing motor speed during operation.
- **Thrust-weight ratio:** Due to the thrust-weight ratio being less than 1 due to being overweight it was difficult for the drone to liftoff. Slight weight reduction measures were made to attain a stable hovering above ground.
- **Motor Synchronization**: Calibration was essential to maintain consistent motor speeds. Fine adjustments to PWM values ensured all motors operated in sync, achieving hover stability.

These optimizations contributed to a functional prototype capable of remote-controlled operation, meeting the project’s technical requirements.

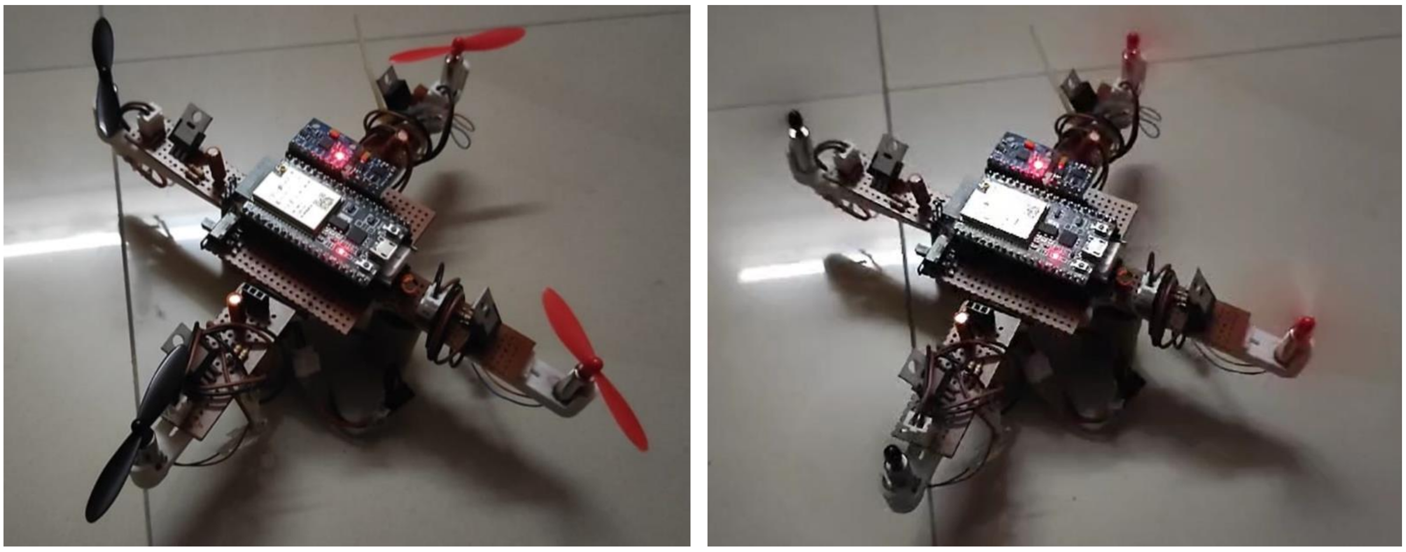


Figure 5 – Final assembly of ESP32 Drone

INFERENCE

The ESP32-based drone project successfully achieved core functionalities, including remote-controlled flight, real-time sensor integration, and stable hovering. The drone’s design utilized the ESP32 microcontroller for efficient processing and Wi-Fi communication, enabling control through a web interface. Custom Electronic Speed Controllers (ESCs) for the 716 micro-coreless motors, along with MPU6050 accelerometer and magnetometer sensors, allowed the drone to adapt to its environment and maintain stability. The project demonstrated the feasibility of developing a cost-effective, mobile-controlled drone with basic flight stability and control, showcasing the ESP32's suitability for IoT-driven UAV applications.

Challenges and Solutions:

Several challenges surfaced during the project, particularly around motor synchronization, power management, and thrust-to-weight ratio. Through iterative testing and troubleshooting, I addressed issues in motor calibration and power distribution, ensuring that the drone could hover steadily. Adjustments to the custom ESCs helped synchronize motor responses, while firmware modifications optimized PWM (Pulse Width Modulation) signals, enhancing motor stability. Tackling these challenges provided valuable experience in problem-solving and reinforced my understanding of electronics, motor control, and sensor integration.

Recommendations for Future Improvements:

1. **Thrust-to-Weight Ratio:**

   The drone’s current thrust-to-weight ratio limited its ability to ascend beyond ground level. Future iterations could improve this by using more powerful motors like 8520 micro-coreless motors or lightweight, high-capacity batteries to increase thrust without sacrificing battery life. This adjustment would enhance lift capability, allowing for extended flight and increased altitude.

2. **Advanced Motor Control:**

   Implementing advanced control algorithms like PID (Proportional-Integral-Derivative) control could significantly improve motor synchronization and stability. This approach would enable the drone to adapt to fluctuations in motor speed, maintaining balance even during sudden changes in orientation or altitude. Optimizing ESC firmware and PWM signals would further support smoother, more responsive flight control.

3. **Additional Sensors:**

   Integrating a gyroscope and barometer could enhance the drone’s stability and provide altitude control. A gyroscope would enable more precise orientation data, while a barometer would support altitude stabilization, especially useful for applications requiring consistent height, such as surveying or mapping. Together, these sensors would increase the drone's adaptability in various environments and expand its functional range.

4. **User Interface Enhancements:**

   While the current web interface provided basic functionality, a dedicated mobile app or controller interface could improve control precision and ease of use. A more intuitive interface could offer real-time sensor data, allow finer motor control adjustments, and perform specific manoeuvres. Additionally, integrating Bluetooth as a backup to Wi-Fi would ensure stable connectivity, minimizing disruptions during flight.

5. **Battery Efficiency and Modularity:**

   Enhancing battery life and efficiency would allow for longer flight durations. Future designs could explore lightweight, high-energy-density batteries like lithium-polymer (Li-Po) or lithium-sulfur (Li-S) to balance power needs with agility. Implementing a modular battery system for quick swapping would also enable continuous operation for extended missions, adding versatility to the drone’s applications.

Skills and Knowledge Gained:

This project provided an in-depth, hands-on understanding of UAV development, from hardware integration to software programming. I learned to design and implement custom ESCs, synchronize motors for stable flight, and calibrate sensors for motion tracking. Working with the ESP32 microcontroller expanded my skills in firmware programming and real-time control while troubleshooting issues in motor synchronization and power distribution taught me valuable problem-solving skills. Additionally, learning about control algorithms, power management, and communication protocols deepened my technical knowledge in drone engineering and IoT applications.

Conclusion:

The ESP32-based drone project successfully demonstrated a cost-effective approach to UAV development, laying a foundation for IoT-based drone control and remote operation. Recommendations for future enhancements, such as improving thrust, refining control algorithms, adding sensors, and optimizing battery life, offer a pathway for advancing the drone’s performance. This project has not only showcased the ESP32’s potential in UAV applications but also equipped me with valuable skills in electronics, programming, and UAV technology, positioning me to contribute effectively to the evolving field of aerospace engineering.

REFERENCES

- Espressif Systems. (n.d.). *Getting Started with ESP-Drone*. Retrieved from https://docs.espressif.com/projects/espressif-esp-drone/en/latest/gettingstarted.html
- Espressif Systems. (n.d.). *ESP-Drone Hardware Overview*. Retrieved from https://docs.espressif.com/projects/espressif-esp-drone/en/latest/hardware.html
- Espressif Systems. (n.d.). *ESP-Drone Repository*. GitHub. Retrieved from https://github.com/espressif/esp-drone
- Random Nerd Tutorials. (n.d.). *Using PWM with ESP32 (Arduino IDE)*. Retrieved from https://randomnerdtutorials.com/esp32-pwm-arduino-ide/
- Random Nerd Tutorials. (n.d.). *Creating a Web Server on ESP32 (Arduino IDE)*. Retrieved from https://randomnerdtutorials.com/esp32-web-server-arduino-ide/
- Circuit Digest. (n.d.). *Interfacing Magnetometer and Gyroscope with ESP32*. Retrieved from https://circuitdigest.com/microcontroller-projects/diy-smartwatch-using-esp32-part3-interfacing-magnetometer-and-gyroscope-with-esp32
- Random Nerd Tutorials. (n.d.). *ESP32 Pinout Reference and GPIOs*. Retrieved from https://randomnerdtutorials.com/esp32-pinout-reference-gpios/s
- GitHub. (n.d.). *ESP32 AnalogWrite Library*. Retrieved from https://github.com/Dlloydev/ESP32-ESP32S2-AnalogWrite
- Random Nerd Tutorials. (n.d.). *ESP32 I2C Communication (Arduino IDE)*. Retrieved from https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/
- YouTube. (n.d.). *Introduction to Drone Control with ESP32*. Retrieved from https://m.youtube.com/watch?v=N_XneaFmOmU