Design and Implementation of a Custom ESP32-Based Quadcopter Flight Controller

Brij Modi

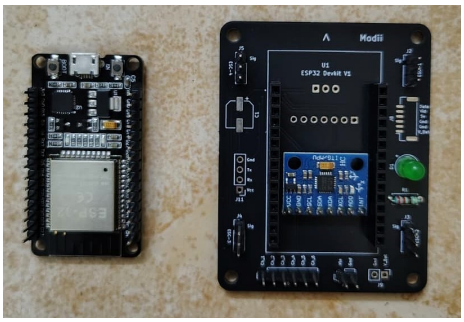
October 2025

Abstract

This report summarizes the design and implementation of a fully custom-built quadcopter with a self-developed flight controller using an ESP32 microcontroller and an MPU6050 IMU. The main goal is to develop a low-cost, modular, and customizable flight control system that allows total hardware and software flexibility, as opposed to conventional commercial controllers. The system includes real-time attitude estimation by using a complementary filter and stabilization using a dual-loop PID control architecture over roll, pitch, and yaw axes. A web-based Wi-Fi interface for wireless PID tuning without firmware reprogramming allows easy calibration. A quadcopter built using an F450 frame with A2212-1400KV motors, SimonK 30A ESCs, and an FS-i4X transmitter showed stable flights with more than 90% stability in manual flight tests. Other optional functions like altitude hold, GPS navigation support, and FPV camera are integrated into the PCB design for potential autonomous functions and broader application scenarios.

1 Introduction

Unmanned Aerial Vehicles (UAVs) have revolutionized various activities such as aerial mapping, surveillance applications, precision agriculture, and scientific exploration. Among the several types of UAVs, quadcopters are renowned for mechanical simplicity, vertical takeoff and landing capabilities, and high maneuverability. Because of this, they are preferred platforms not only in research but also in practical applications.



ESP32-based flight controller PCB.

While commercial flight controllers, such as the KK and Naze32, are very powerful and feature-rich, their closed-source firmware or limited customization usually restricts new control algorithm or sensor configuration experimentation. This limits the educational and research efforts that need deeper insight into embedded control design and firmware customization.

The following work will fill this gap by detailing the design and implementation of a completely custom ESP32-based quadcopter flight controller. It is fitted with an MPU6050 IMU for real-time attitude sensing, a dual-loop PID control structure for precise stabilization, and a web-based interface for wireless tuning. While stable flights can be realized in this project, it focuses more on the end-to-end development from circuit design, PCB fabrication to control firmware implementation, thus offering grounds for open, modular UAV research platforms. Optional enhancement features included in the new PCB version of the flight controller are barometer-based altitude hold, GPS navigation support, and an FPV camera interface. These extend basic stabilization functionalities towards making the platform applicable to semi-autonomous and research-oriented UAV development.

2 System Overview

2.1 Overall System Description

The custom-built quadcopter consists of integrated hardware and software subsystems working together to maintain stable flight and respond to pilot commands. The core modules include:

- **Sensing:** MPU6050 IMU provides acceleration and angular rate data.
- **Computation:** ESP32 microcontroller executes sensor fusion and PID control.
- **Actuation:** Four brushless DC motors driven by ESCs generate thrust and torque.
- **Communication:** Wi-Fi and RC receiver enable wireless control and tuning.

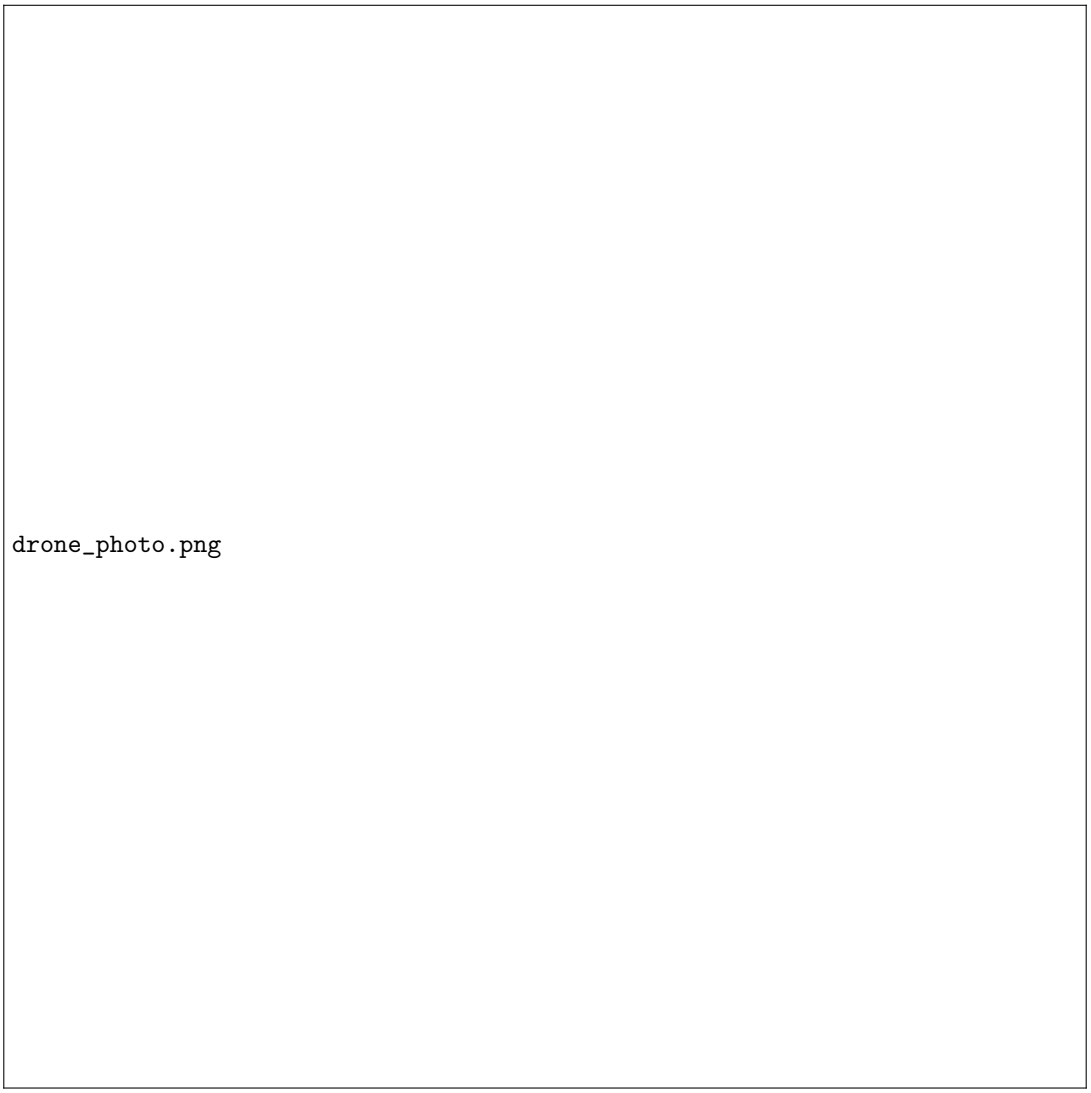


Figure 1: Custom-built quadcopter with ESP32-based flight controller.

2.2 Functional Architecture

The flight controller is a closed-loop system constantly processing sensor data, computing corrective actions, and adjusting motor speeds. Figure 2 depicts the functional architecture with the ESP32 acting as the central processing node.

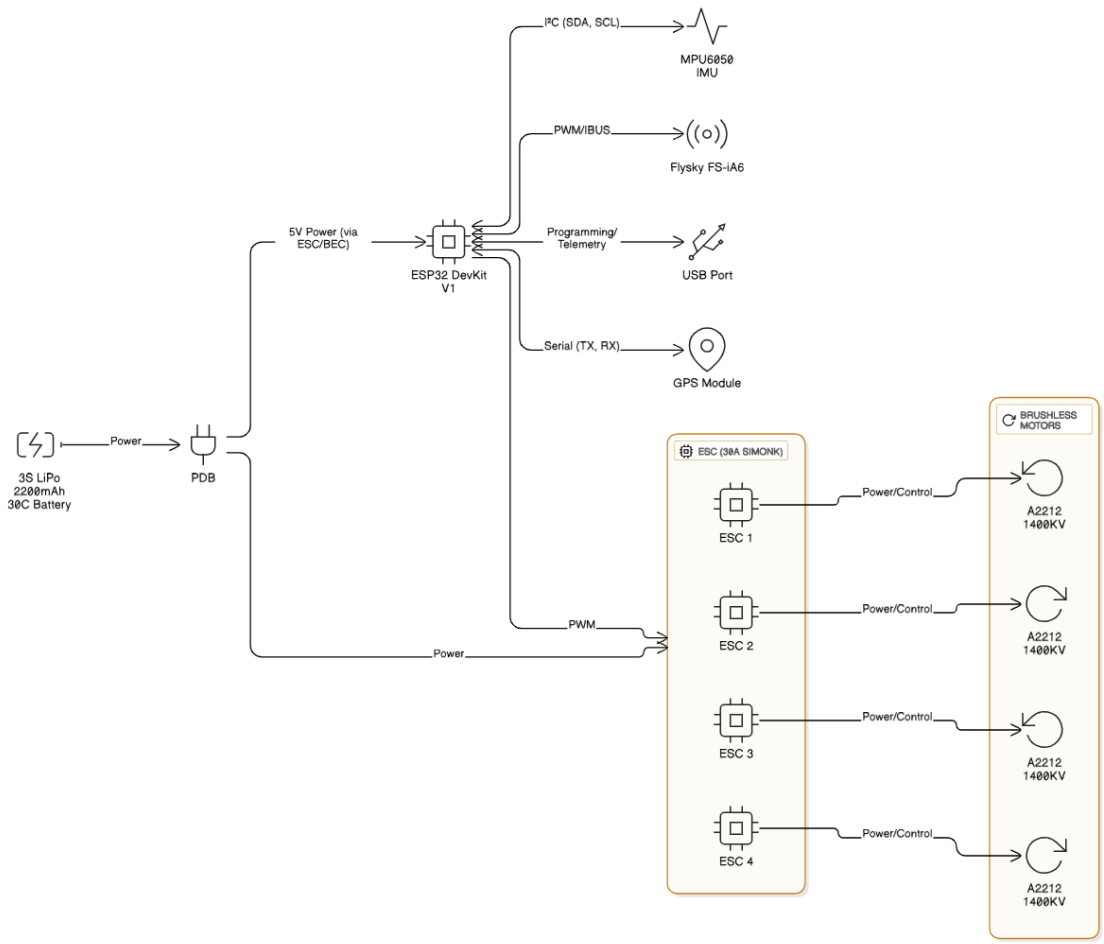


Figure 2: Functional block diagram of the flight control system.

2.3 Sensor Fusion and Communication

Accurate attitude estimation is achieved using a complementary filter that fuses accelerometer and gyroscope data from the MPU6050. Communication between modules is handled through:

- I^2C for IMU communication,
- PWM / i-BUS for RC inputs,
- PWM outputs for ESCs,
- Wi-Fi (HTTP) for telemetry and PID tuning.

2.4 Flight Control Data Flow

The flight controller firmware acts as a continuous closed-loop system that translates sensor data and inputs from the pilot to corresponding precise outputs in motor control. Figure 3 shows the complete flow from initialization to the continuation of the loop. Every element has a specific job in this real-time control architecture to provide stable and agile flight.

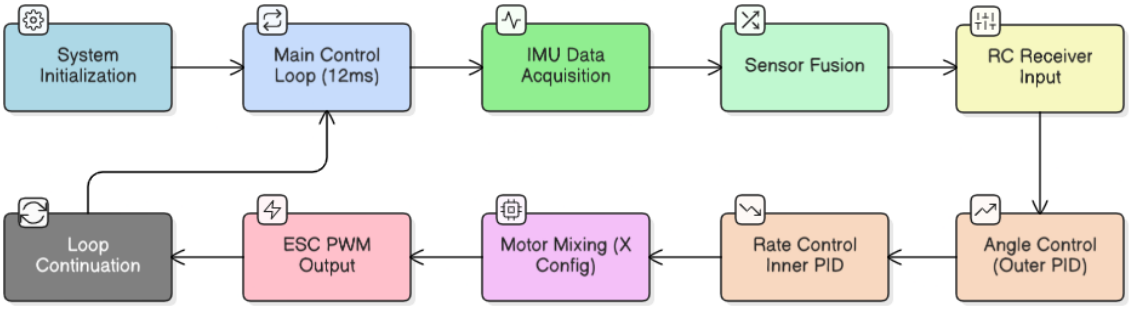


Figure 3: Data flow diagram of the ESP32-based flight control loop.

The process can be divided into ten main stages:

1. **System Initialization:** Initializes the IMU, receiver interrupts, and ESC interfaces. All sensors are calibrated, and timing for the 12 ms control loop is configured.
2. **Main Control Loop (12 ms):** This loop runs continuously at a frequency of approximately 83 Hz, maintaining real-time flight control. Each iteration performs sensing, estimation, control computation, and actuation.
3. **IMU Data Acquisition:** Reads accelerometer and gyroscope data via the I^2C interface and computes initial roll and pitch angles.
4. **Sensor Fusion:** Combines gyro and accelerometer data using a complementary filter to obtain stable, drift-free filtered roll and pitch angles.
5. **RC Receiver Input:** The receiver provides six PWM channels representing pilot inputs—throttle, roll, pitch, yaw, and auxiliary controls. These PWM signals are converted into normalized control values in the firmware to derive the desired attitude and throttle commands.
6. **Angle Control (Outer PID):** This calculates the error between the desired and measured angles; the output of the PID is the target angular rate for roll and pitch correction.
7. **Rate Control (Inner PID):** It controls the angular velocity about each axis based on the rate errors to achieve control signals for roll, pitch, and yaw stabilization.
8. **Motor Mixing (X-Config):** The control signals from the PID loops are combined according to the quadcopter's X-frame geometry. It determines how much each of the four motors contributes to provide roll, pitch, yaw, and throttle commands to ensure well-balanced thrust and torque.
9. **ESC PWM Output:** Converts motor commands into 1000–2000 µs PWM signals and sends them to the ESCs for motor speed control.
10. **Loop Continuation:** Repeats the entire process every 12 ms for continuous feedback and real-time stabilization.

From a control viewpoint, this is a hierarchical feedback structure where the outer loop controls attitude (angle), and the inner one is responsible for angular rate control. This dual PID approach improves stability and responsiveness. Moreover, by fusing sensor data and accurately mixing motor power, the flight performance is dependable, equal or superior to that of commercial implementations.

3 Hardware Design

3.1 Circuit Design and Integration

The ESP32 serves as the central processor, interfacing with the MPU6050 via I
textsuperscript2C and the RC receiver via PWM/i-BUS. Four PWM outputs are used to drive the ESCs, which in turn drive the brushless DC motors. A 5V regulator ensures a stable voltage supply to the logic circuits.

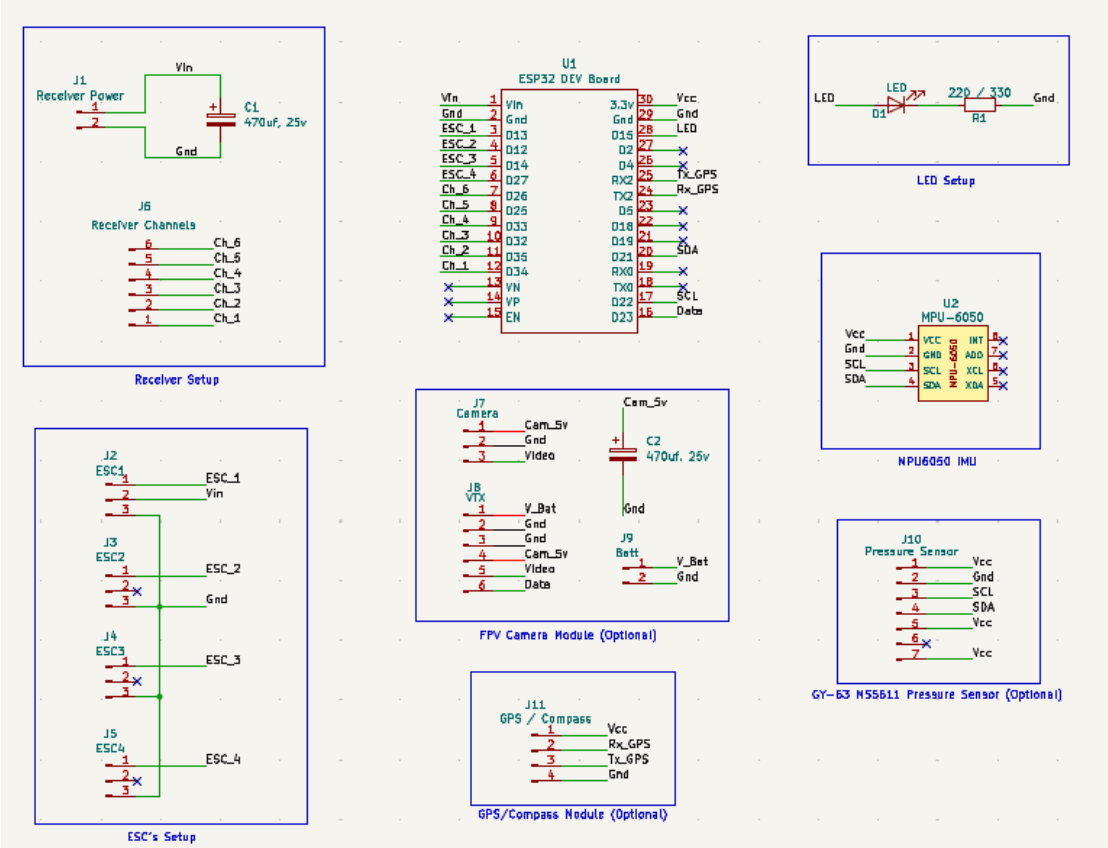


Figure 4: Circuit schematic of the custom ESP32-based flight controller.

3.2 PCB Design and Layout

The PCB was designed in KiCad with a compact two-layer layout, taking care of signal isolation. Ground planes and decoupling capacitors help with electromagnetic interference.

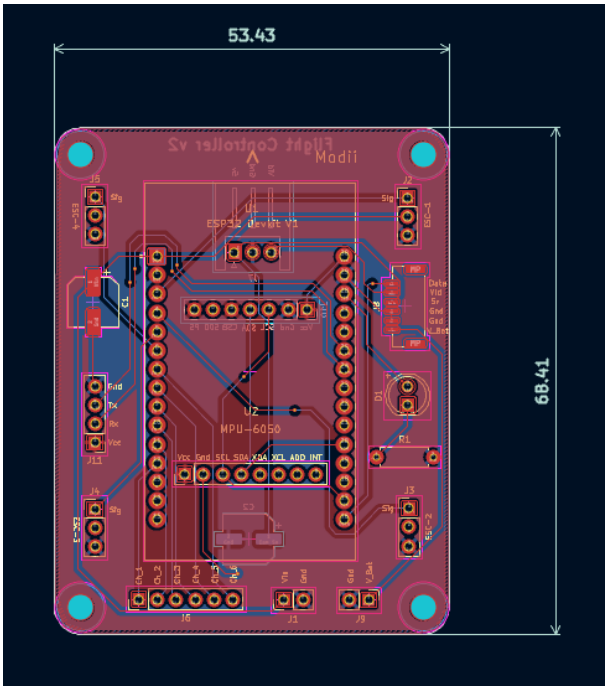


Figure 5: PCB layout of the ESP32-based flight controller.

3.3 Component Selection and Justification

Table 1: Major hardware components used in the quadcopter.

Component	Specification	Function
ESP32 Dev Module	Dual-core MCU, Wi-Fi/BLE	Main flight controller
MPU6050 IMU	3-axis accelerometer + gyroscope	Attitude sensing
Barometer (Optional)	GY-63 MS5611	Enables altitude hold functionality
GPS Module (Optional)	NEO-6M	Provides position data for autonomous navigation
A2212 BLDC Motor	1400 KV	Propulsion
ESC	30A, SimonK firmware	Motor control
FlySky FS-i4X Receiver	4-channel, 2.4 GHz	Pilot input
Li-Po Battery	11.1V, 2200 mAh, 30C	Power source
F450 Frame	450 mm diagonal, plastic/fiber	Drone structure

3.4 Mechanical and Frame Design

The ideal balance of strength and weight is achieved in the F450 frame. The span between the motor mounts in this frame is 450 mm, where the diagonal distance from motor to motor is 45 cm. The approximate weight of the drone was 1.25 kg. Its design guarantees a low center of gravity, even weight distribution, and an optimal thrust-to-weight ratio of about 2:1 for stable flights. Besides, the flight controller was installed at the geometric center of the frame to minimize the rotational inertia and improve attitude stability. The MPU6050 IMU was installed on vibration-isolating foam to damp high-frequency motor and propeller vibrations, thus enhancing the accuracy of angle estimation. Due to the symmetric X-configured F450 frame, the roll and pitch responses were uniform, which reduced complexity in the calculation of motor mixing and improved overall flight performance.

3.5 Power Distribution and Safety

Power is supplied by a 3S Li-Po battery through a PDB. ESCs are powered directly with 11.1V; the ESP32 and IMU each receive their power from a 5V regulator. Electrical noise is reduced by adding capacitors, while proper grounding minimizes interference between digital and power circuits.

4 Software Architecture

4.1 Firmware Structure

The firmware is modular, with separate tasks for sensing, control, and communication. Each control loop (executed every 12 ms) performs the following:

1. Acquire IMU data.
2. Compute attitude using a complementary filter.
3. Read pilot inputs from RC receiver.
4. Run nested PID control loops.
5. Update ESC PWM signals.
6. Send real-time telemetry via Wi-Fi.

4.2 Complementary Filter

$$\theta_{\text{roll,est}}(k) = 0.991[\theta_{\text{roll,est}}(k-1) + \omega_{\text{gyro,roll}}t] + 0.009\theta_{\text{acc,roll}} \tag{1}$$

$$\theta_{\text{pitch,est}}(k) = 0.991[\theta_{\text{pitch,est}}(k-1) + \omega_{\text{gyro,pitch}}t] + 0.009\theta_{\text{acc,pitch}} \tag{2}$$

with $t = 0.012$ s. This filter minimizes gyro drift while preserving responsiveness.

4.3 Control Flow (Dual PID Loops and Yaw)

Roll and pitch stabilization are achieved using a dual PID loop—an outer loop for angle control and an inner loop for angular rate control. Yaw uses a single rate PID.

Outer Loop – Angle PID

$$e_{angle}[k] = \text{Desired\_Angle} - \text{Measured\_Angle} \tag{3}$$

$$u_{angle}[k] = K_{p_a}e_{angle}[k] + K_{i_a}\sum e_{angle} + K_{d_a}\frac{\Delta e_{angle}}{t} \tag{4}$$

Inner Loop – Rate PID

$$e_{rate}[k] = \text{Desired\_Rate} - \text{Measured\_Rate} \tag{5}$$

$$u_{rate}[k] = K_{p_r}e_{rate}[k] + K_{i_r}\sum e_{rate} + K_{d_r}\frac{\Delta e_{rate}}{t} \tag{6}$$

4.4 PID Control Algorithm

The general PID control law:

$$u[k] = K_p e[k] + K_i \sum e[k] + K_d \frac{de}{dt} \tag{7}$$

5 Testing and Results

5.1 Flight Test Procedure

Flight tests were carried out in open-ground conditions under low wind. Tests included hover stability, roll/pitch step responses, and yaw hold performance. PID gains were adjusted wirelessly via Wi-Fi interface.

5.2 Performance Evaluation

- Hover achieved at 40–50% throttle.
- Roll/pitch error maintained within $\pm 2^\circ$.
- Battery endurance: 10–15 minutes.
- 90-95% overall flight stability observed.

5.3 PID Tuning Results

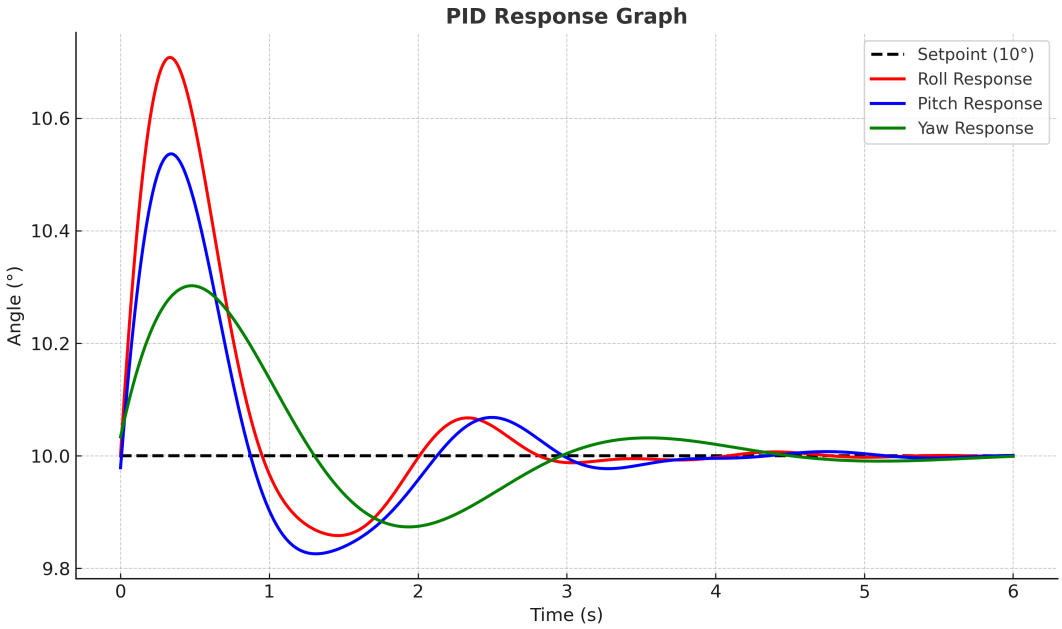


Figure 6: PID tuning response.

5.4 PID Tuning Parameters

During the flight testing phase, PID gains were tuned iteratively through the wireless web interface to achieve smooth and stable flight behavior. Table 2 lists the final optimized gains for roll, pitch, and yaw axes. Identical gains were used for roll and pitch due to the symmetric quadcopter geometry.

Table 2: Final tuned PID gains providing stable flight performance.

Control Loop	**Axis**	$\mathbf{K_P}$	$\mathbf{K_I}$	$\mathbf{K_D}$
Angle (Outer Loop)	Roll	3.6	0.01	0.03
Angle (Outer Loop)	Pitch	3.6	0.01	0.03
Rate (Inner Loop)	Roll	1.25	0.02	0.0086
Rate (Inner Loop)	Pitch	1.25	0.02	0.0086
Rate (Inner Loop)	Yaw	3.8	3.00	0.02

These parameters offered balanced responsiveness and damping, minimizing overshoot and ensuring reliable hover and attitude stability across test flights.

5.5 Observations and Challenges

- Sensitivity to IMU calibration: Small changes in either temperature or mounting orientation affected initial bias values, so multiple test flights were preceded by recalibrations.
- Initial PID tuning resulted in overshoot and oscillations, which were reduced through iterative tunings using the wireless interface.
- ESC signal consistency: Several of the ESCs showed some variation in response time; this made the motor outputs uneven until proper min–max calibration was performed.
- Optimizations in loop timing were required to maintain a constant 12 ms control cycle for reliable performance of the PID.

6 Applications and Future Scope

- **Autonomous Navigation Experiments:** Provides the baseline for testing GPS-based waypoint navigation and altitude hold modes together with autonomous take-off/landing routines.
- **Aerial Surveillance and Monitoring:** Can be adapted for lightweight surveillance applications by integrating an FPV camera or low-latency video transmission system.
- **Precision Agriculture:** Capable of carrying small sensors or cameras that could monitor crop health, soil patterns, or the performance of field mapping tasks.
- **Prototype Development for Custom UAVs:** Offers a flexible flight-control core to assist in building specialized drones, whether they are inspection drones, mapping platforms, or indoor research UAVs.

7 Conclusion

This custom ESP32-based flight controller prototype demonstrates that a low-cost microcontroller, supported by efficient sensor fusion and a twin-loop PID architecture, is capable of reliable quadcopter stabilization, comparable to most commercial systems. It managed to run 10–15 test flights with constant stability of 90–95%, proving the accuracy of the complementary filter, the efficiency of the nested control loops, and the strength of the general hardware–software integration.

The project encompasses all aspects of the design cycle: from circuit development and perfboard prototyping to PCB layout, firmware implementation, and flight testing, thereby showcasing end-to-end competency in embedded control systems. Adding optional features such as altitude hold, GPS navigation support, and FPV camera connectivity extends the functionality of the controller even more in its PCB version.

In general, this work showcases a complete, modular, and extensible flight-control platform that will be an excellent starting point for advanced UAV research, including autonomous navigation experiments, and also for any further enhancements, like computer-vision-based control or multi-sensor fusion.

8 References

- Espressif Systems, *ESP32 Technical Reference Manual*, 2023.
- InvenSense Inc., *MPU6050 Datasheet*.
- Pratik Phadte, *Flight Controller Development Tutorials*, 2024. Educational video series on IMU handling, PID control, and multirotor stabilization.
- Joop Brokking, *DIY Arduino-Based Flight Controller Series*, 2015. Technical tutorials covering IMU filtering, PID tuning, and motor–mixing algorithms for quadcopters.
- Åström, K.J., Hägglund, T., *PID Controllers: Theory, Design, and Tuning*, 2006.