FlyHigh: Design and Implementation of a Quadcopter Flight Controller

Raghavpravin K S, Sushant Gudmewar
Under the guidance of: **Prof. Jhuma Saha**

April 21, 2025

Abstract

This report presents the comprehensive design, implementation, and testing of a quadcopter. The project involves ***hardware selection and integration***, ***flight controller programming with multiple control modes***, ***sensor calibration***, and ***PID tuning***. The quadcopter employs the ***MPU6050*** for inertial measurement and the ***NRF24L01*** module for wireless communication. Throughout development, we encountered key technical challenges, notably ***ESC synchronization***, ***PWM signal inconsistencies***, and ***sensor noise induced by mechanical vibrations***. The control system was refined through iterative algorithm development and tuning, culminating in the implementation of a ***rate mode control strategy*** inspired by the ***YMFC flight control algorithm***.

Contents

1 Introduction

1.1 Project Overview

The **FlyHigh** project focused on building a custom flight controller for a quadcopter, with the long-term aim of designing a single-board PCB integrating all necessary components. The project involved a comprehensive process from initial research and component selection to flight controller algorithm implementation and iterative tuning.

1.2 Objectives

The main objectives of the project included understanding the principles of drone flight control, selecting appropriate hardware components within a budget, developing a flight control algorithm, addressing technical challenges encountered during implementation and tuning controller for stable flight. Ultimately, the goal was to create a foundation for a custom, integrated flight controller.

1.3 Background and Motivation

The project was motivated by an interest in understanding the inner workings of drones and gaining hands-on experience in embedded systems design and control systems. Initial research involved studying the existing drone community, particularly FPV drones.

2 Hardware Architecture

2.1 Component Selection

We followed a structured approach for component selection, considering factors like functionality, budget, and compatibility.

2.1.1 Flight Controller

We opted for the **Aries V3 board with a RISC-V microprocessor running at 100 MHz**. This board was chosen despite common FPV drones using STM32 F4/H7/F7 chips, and its Arduino-like form factor was noted. Our only consideration while choosing the controller was the clock speed as we wanted the control loop to at least run at 250 Hz.

2.1.2 Sensors - MPU6050

For inertial measurements, the **MPU650** IMU (Inertial Measurement Unit), containing both a gyroscope and an accelerometer, was selected as a cost-effective solution. It can measure linear acceleration in 3 directions and angular velocity about the 3 axes, enabling it to be used in the control loop of objects having 6 degrees of freedom, like quadcopters.

2.1.3 Communication - NRF24L01

Wireless communication was to be handled by the **nRF24L01** board operating in the 2.4 GHz spectrum. A module with a power amplifier and a low noise amplifier was paired with the nRF24L01 to achieve a target range of approximately 1000 metres.

2.1.4 Motors and ESCs

Budget constraints led to the choice of **SimonK 30A sensorless Electronic Speed Controller** for brushless motors. The selected motors were AT2212 brushless DC motors with a KV rating of 2200.

2.1.5 Power System

A **3S 1800mAh 40C LiPo battery** was chosen to power the drone, estimated to draw around 15A during hover.

2.1.6 Frame Design

A **5-inch freestyle FPV frame made of carbon fibre, the Mark4**, was selected, influencing the choice of 5-inch three-blade propellers.

2.2 System Integration

The integration phase involved methodically assembling all selected components onto the drone's airframe, ensuring mechanical stability and electrical compatibility. Special attention was paid to the positioning of the IMU to minimize exposure to vibration sources and ensure accurate sensor readings. Soft mounting technique using grommets was employed to dampen mechanical shocks.

Additionally, rigorous validation was conducted at each stage—verifying sensor outputs, ESC signal integrity, and communication between subsystems. This ensured that once airborne, the system could operate cohesively with minimal debugging required in-flight. Integration testing was iteratively performed with subsystem logging enabled, making it easier to trace faults and refine the setup.

2.2.1 Hardware Connectivity

The *MPU6050* inertial measurement unit (IMU) was interfaced with the Aries V3 flight controller via the **I^2C** protocol, ensuring synchronized acquisition of accelerometer and gyroscope data. The *NRF24L01* radio transceiver was connected using the **SPI** interface, allowing low-latency, high-throughput wireless communication. Four ESCs were connected to the PWM output pins of the Aries V3 board, each controlling a brushless DC motor.

2.2.2 Power Distribution

The power architecture was based on an **11.1 V 40C Li-Po battery**, which served as the primary energy source for the entire system. The **Aries V3** flight controller board was directly powered from this battery, and it internally regulated and distributed power to peripheral modules. Both the **IMU** and the **nRF24L01** communication module were supplied via regulated outputs from the Aries board, ensuring stable voltage levels and reducing the risk of noise coupling from the high-current motor lines.

3 Software Architecture

3.1 Flight Controller Firmware

The core of the project was the development and deployment of custom flight controller firmware on the Aries V3 board. Initially, a **Proportional (P) control algorithm** was implemented to regulate the drone's roll, pitch, and yaw axes. We explored two fundamental control modes: **angle mode**, which attempts to stabilize attitude using accelerometer data, and **rate mode**, which regulates angular velocities using gyroscope feedback.

Early trials involved both P and PI controllers; however, these approaches failed to stabilize the system. The primary issue was the poor quality and noise in accelerometer readings, which rendered attitude estimation in angle mode unreliable.

To address these deficiencies, a full **PID controller** was implemented in rate mode, leveraging gyroscope data for closed-loop control. This approach significantly improved stability, allowing the drone to respond smoothly to control inputs while suppressing oscillations and external disturbances. The derivative term proved particularly effective in damping rapid deviations, leading to a more robust and responsive flight behavior.

3.2 Communication Protocol

A bidirectional communication link was established between a custom-built transmitter constructed using an additional Aries V3 board and an *NRF24L01* transceiver and the receiver module onboard the drone. This wireless interface was used to transmit real-time control commands including throttle, roll, pitch, and yaw, with minimal latency and high reliability.

The **NRF24L01** module operated in the 2.4 GHz ISM band and supported a data rate of up to 2 Mbps. To enhance communication robustness, the *Enhanced ShockBurst™* protocol was employed, which provided built-in features such as **auto-retransmission**, **auto-acknowledgment**. These features significantly reduced the risk of data loss during high-frequency transmissions and simplified packet-level error handling. Payloads were encapsulated in structured packets and transmitted at a consistent refresh rate, ensuring smooth control updates.

3.3 Control System Design

3.3.1 Angle Mode Implementation

The control strategy initially attempted was **angle mode**, wherein the roll and pitch angles were computed using `arctangent` functions applied to the accelerometer's x, y, and z components. However, the presence of high-frequency vibrations from the propulsion system led to noisy accelerometer outputs, making the estimated attitude highly unreliable.

Moreover, yaw control in angle mode was inherently limited, as the accelerometer does not provide information about heading. It was concluded that **gyroscopic integration was indispensable for yaw estimation**, necessitating the use of a hybrid or rate-based control even in nominally angle-based modes. These findings motivated the permanent shift to rate mode with advanced filtering and full PID control.

3.3.2 Rate Mode Implementation

The persistent instability in angle mode caused by mechanical vibrations and unreliable accelerometer data, we transitioned to a **rate mode** control strategy. This mode leveraged gyroscopic measurements to regulate angular velocities directly. To estimate orientation drift while maintaining responsiveness, a **complementary filter** was implemented with a high weighting of **99.96% on gyroscope data** and a minimal contribution of **0.04% from the accelerometer**. This allowed for low-pass correction of gyro drift without compromising response speed.

PID control, improved the drone's damping characteristics and enabled precise control across all three axes. With real-time tuning, we further refined the PID gains, significantly enhancing flight stability.

4 Sensor Data Processing

4.1 Accelerometer Data

The accelerometer data was initially used to estimate the roll and pitch angles in angle mode. However, this data was found to be heavily corrupted by mechanical vibrations from the motors and ESCs.

4.2 Gyroscope Data

The gyroscope data provided measurements of the drone's angular velocity in three axes. In the later rate mode implementation, the gyroscope data became the primary source for attitude estimation through the complementary filter.

4.3 Data Fusion Techniques

The team explored various data fusion techniques to mitigate the noise in the accelerometer data, including the **Kalman filter, digital low-pass filter, moving average filter, and complementary filter**. Initially, these filters were applied directly to the roll and pitch data, and later to the raw accelerometer data.

4.4 Kalman Filtering

The Kalman filter was one of the techniques attempted for noise reduction in the sensor data. However, it did not yield satisfactory results in the presence of significant vibrations.

4.5 Moving Average Filter

A moving average filter was also tried to smooth the sensor data, but it was not effective enough to overcome the vibration-induced noise.

4.6 Plots of Data indicating noise

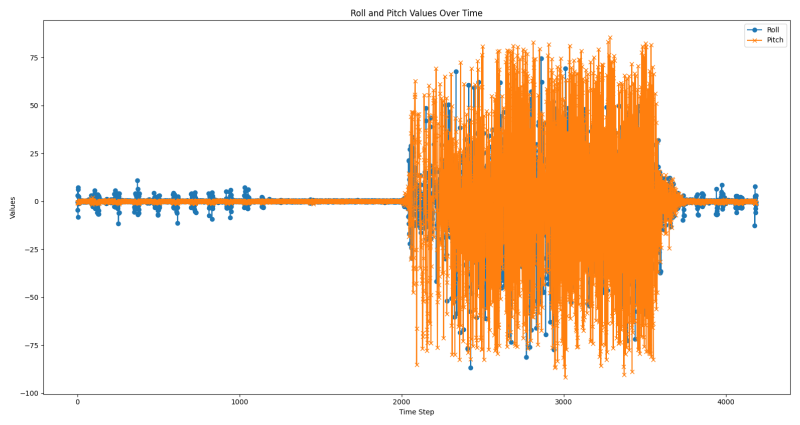

Figure 1: Raw roll and pitch measurements showing significant noise from motor vibrations

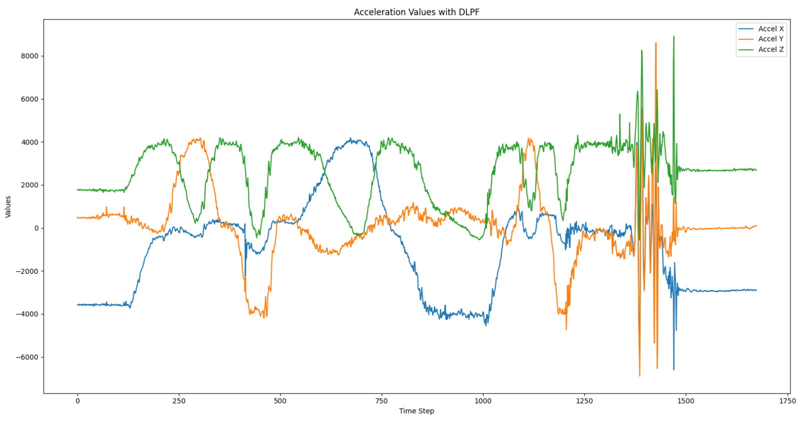

Figure 2: Accelerometer data with digital low-pass filtering applied

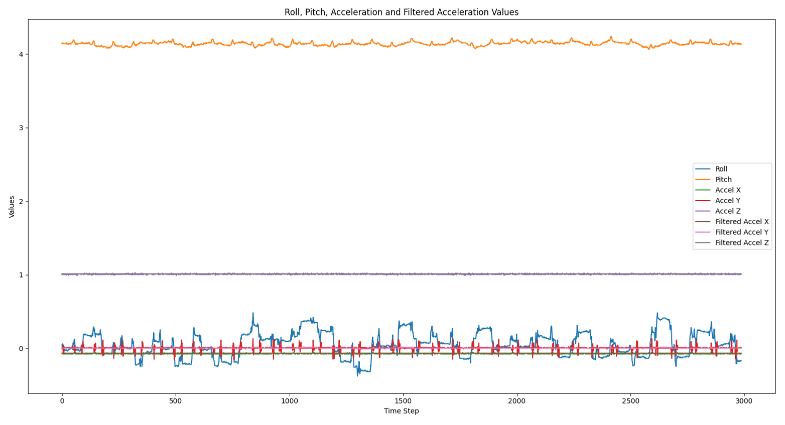

Figure 3: Accelerometer data processed with basic moving average filter

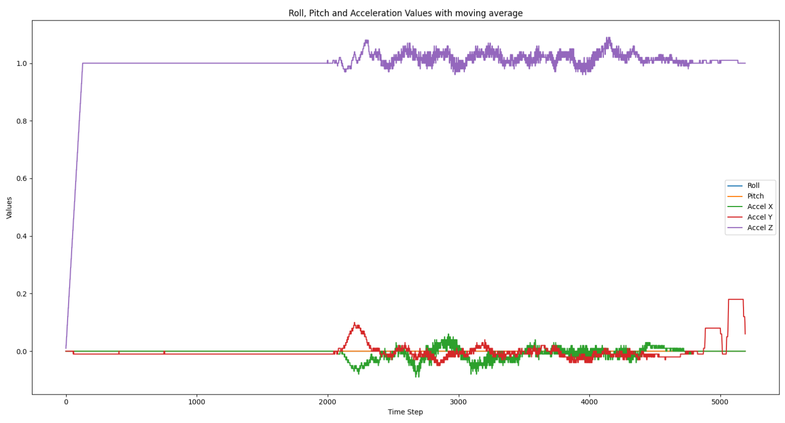

Figure 4: Accelerometer data processed with moving average filter (130 samples)

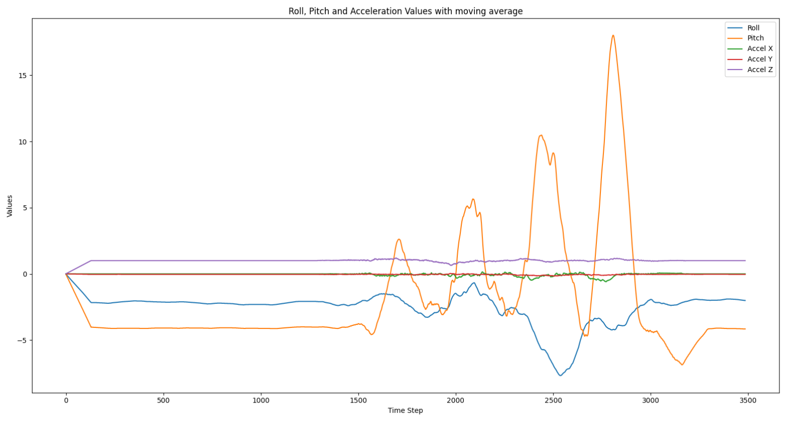

Figure 5: Roll and pitch estimations using accelerometer data with 130-sample moving average filter

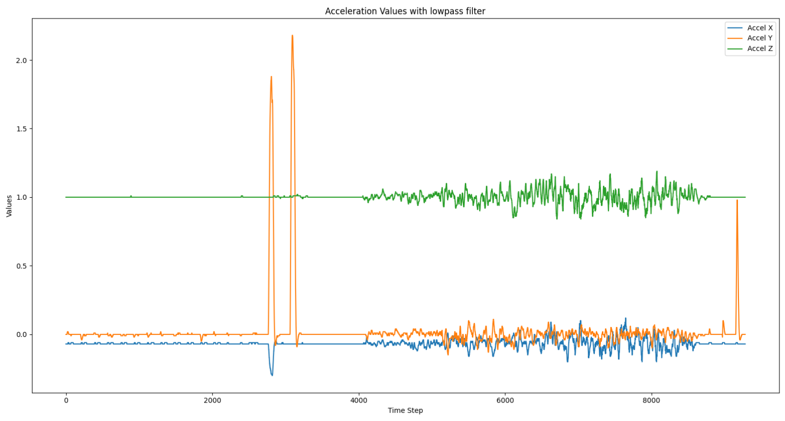

Figure 6: Accelerometer data processed with low-pass filter

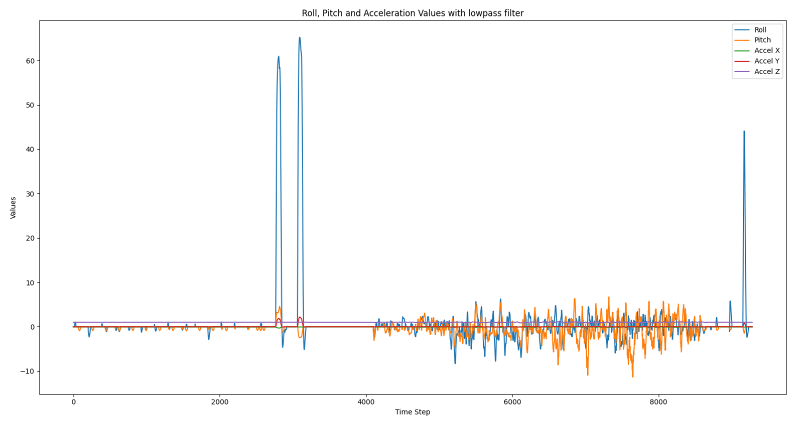

Figure 7: Roll and pitch estimations after applying low-pass filter to accelerometer data

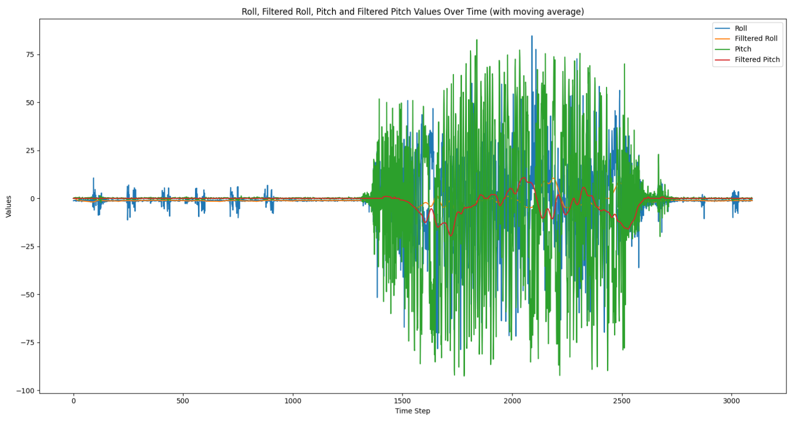

Figure 8: Roll angle filtered with 90-sample moving average

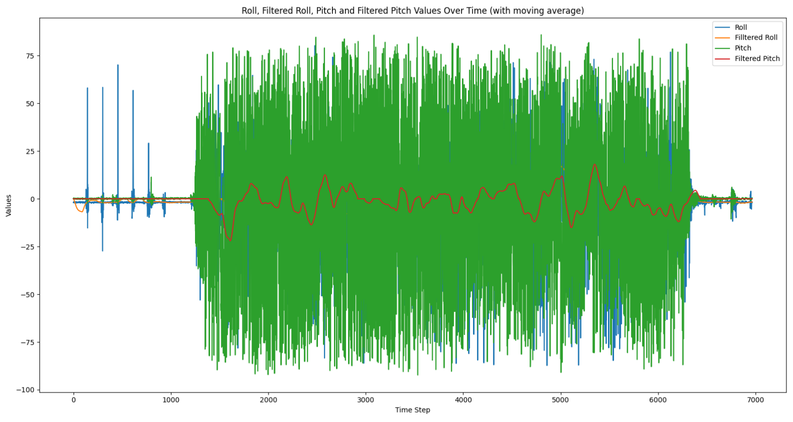

Figure 9: Roll angle with 90-sample moving average and 1.5Hz low-pass filter

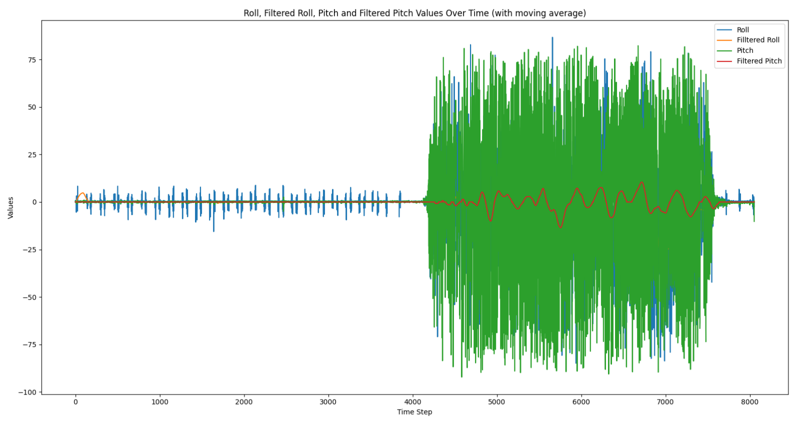

Figure 10: Roll angle with 90-sample moving average, 1Hz low-pass filter, and 330Hz control loop

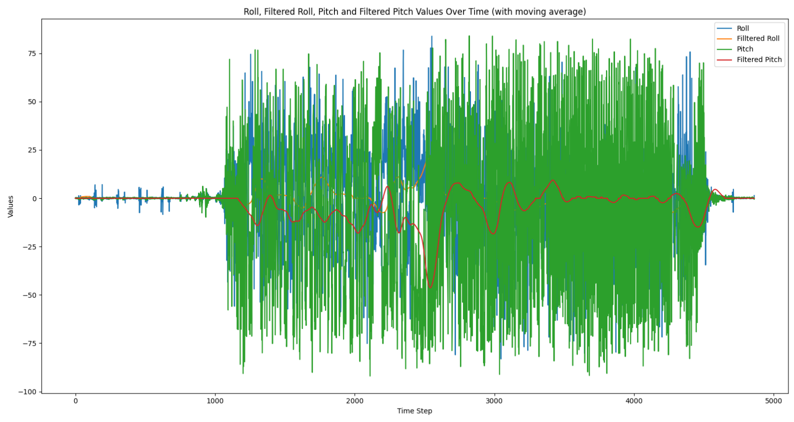

Figure 11: Improved roll angle filtering with 90-sample moving average, 1Hz low-pass filter, and 330Hz control loop

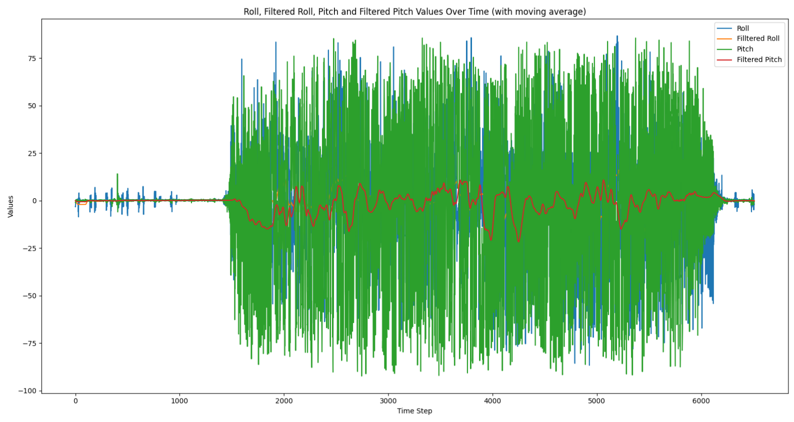

Figure 12: Roll angle filtered with 200-sample moving average, showing excessive lag

5 Control System Implementation

5.1 PID Controller Design

We initially aimed for a PID (Proportional-Integral-Derivative) controller, but the primary focus during the described development was on the **Proportional (P) component**, especially after transitioning to rate mode.

5.2 Tuning Methodology

Tuning the P constants proved to be a significant challenge. We learned that P values are highly dependent on the specific drone hardware. We attempted to tune roll and pitch axes separately by physically restricting motion in other axes. Initial attempts using strings were flawed due to torsional strain in the strings aiding stabilisation. A redesigned landing gear with tubes for more rigid restraint was implemented to improve the tuning process. We used two tables as support for the test bench setup. The rod, which can be seen between, was used to hold the drone in order to tune the coefficients.

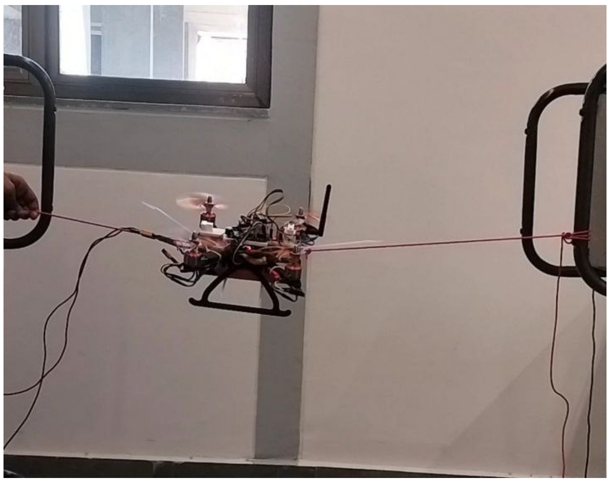

Figure 13: Testbench with two tables and string

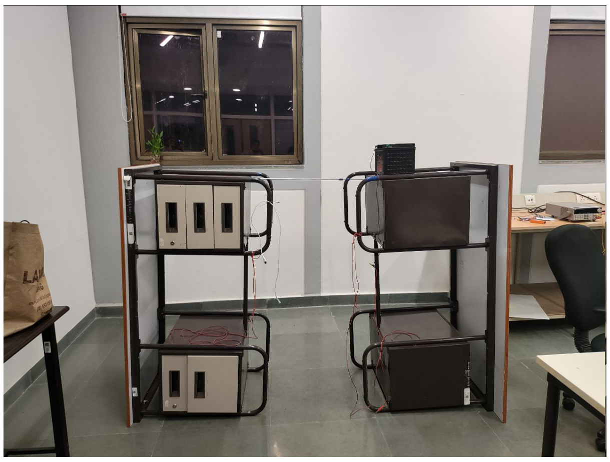

Figure 14: Testbench with two tables and rod

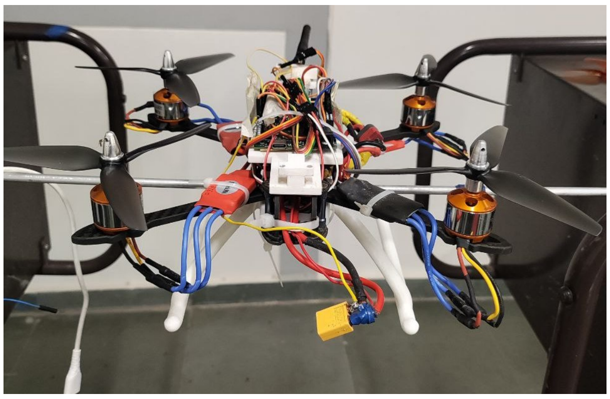

Figure 15: Drone mounted in the test bench

5.2.1 Attitude Control Performance

The initial attitude control based on accelerometer data in angle mode was unsuccessful due to noise. The attitude estimation in rate mode, while more stable in terms of drift, was primarily used to control angular rates rather than directly targeting specific angles.

5.2.2 Rate Control Performance

The shift to rate mode with the highly weighted complementary filter on gyroscope data led to a significant stability, even with vibrations present in accelerometer data.

5.3 Control Mode Evolution

5.3.1 Angle Mode Implementation (Initial Approach)

The initial control strategy relied on pure accelerometer-based angle calculation:

```
float current_roll = atan(Ay_cal/sqrt(Ax_cal*Ax_cal+Az_cal*Az_cal))*(180/3.142);
float current_pitch = -atan(Ax_cal/sqrt(Ay_cal*Ay_cal+Az_cal*Az_cal))*(180/3.142);
roll = current_roll;
pitch = current_pitch;
```

This approach:

- Relied on moving average filtering (100 samples) for stability
- Used high PI gains: Kp_roll = 300.0, Ki_roll = 40
- Calculated error directly: `roll_error = roll_setpoint - roll`
- Had no derivative term (PI controller only)
- Required per-motor scaling factors for balance compensation:

```
Motor1 = rx_data.height * 0.920 - pitch_output + rate_yaw_output - roll_output;
Motor2 = rx_data.height * 1.030 + pitch_output - rate_yaw_output - roll_output;
Motor3 = rx_data.height * 0.98 + pitch_output + rate_yaw_output + roll_output;
Motor4 = rx_data.height * 1.05 - pitch_output - rate_yaw_output+ roll_output;
```

5.3.2 Acro/Rate Mode Implementation (Final Approach)

After encountering significant noise issues, we transitioned to a rate mode strategy with:

- Complementary filter with heavy gyro bias (99.96%):

```
angle_roll = 0.9996 * (angle_roll + Gx_prev * 0.002) +
             0.0004 * (atan2(Ay_conv, sqrt(Ax_conv*Ax_conv + Az_conv*Az_conv)) * 57.324
```

- Gyro smoothing: `Gx_prev = 0.7* Gx_prev + 0.3* Gx_conv;`
- Moderate PID gains: Kp_roll = 1.70, Ki_roll = 0.35, Kd_roll = 0.55
- Full PID implementation with derivative term
- Direct yaw rate control: `error_yaw = Gz_prev - 0.0;`
- Standard motor mixing algorithm:

```
motor_1 = throttle - pid_roll - pid_pitch - pid_yaw; // Front right
motor_2 = throttle + pid_roll - pid_pitch + pid_yaw; // Front left
motor_3 = throttle - pid_roll + pid_pitch - pid_yaw; // Rear right motor
motor_4 = throttle + pid_roll + pid_pitch + pid_yaw; // Rear left motor
```

5.3.3 Key Differences Between Control Modes

- **Angle mode** seeks absolute orientation based on gravity vector, providing self-stabilization for beginners
- **Acro/Rate mode** prioritizes angular rate control with minimal self-leveling, offering more precise control for experienced pilots
- The shift to rate mode was necessitated by accelerometer noise from vibrations
- Rate mode proved significantly more robust in our implementation

5.4 PID Coefficient Optimization

The majority of the tuning effort focused on **optimising the P coefficients** for roll and pitch in rate mode. This was an iterative process spanning several weeks.

For more information on PID control, refer to the whitepaper by Zurich Instruments: Principles of PID Controllers.

6 Testing and Results

6.1 Initial Testing

Initial flight tests in angle mode ended in failure due to the heavily noisy accelerometer data. Subsequent testing in rate mode with borrowed P constants also resulted in a crash.

6.2 Stability Analysis

After extensive tuning with the improved restraint system, the drone achieved near stability in roll and pitch, though a slight steady-state error causing forward drift remained.

6.3 Performance Metrics

The sources do not provide quantitative performance metrics such as flight time, maximum speed, or tracking accuracy. The primary focus was on achieving stable hover and basic control.

6.4 Flight Tests

Multiple flight tests were conducted throughout the project, initially unsuccessful but gradually improving with algorithm refinement and PID tuning.

7 Challenges and Solutions

7.1 Hardware Challenges

- **Inconsistent ESC Startup:** Initially, the four ESCs were not starting simultaneously or consistently. This was resolved by identifying and correcting a **slight error in the PWM frequency output from the Aries V3 board**, tuning the prescaler values to achieve 50 Hz.
- **Ringing in PWM Signal:** Oscilloscope analysis revealed ringing in the PWM output from the Aries V3 board, suspected of affecting the ESCs. While RC filters were attempted, they proved ineffective. The frequency correction ultimately addressed the ESC issues despite the ringing.
- **Heavy Drone Weight:** The overall weight of the drone reached around 850g due to the cheap, bulky ESCs. This impacted the power requirements and flight dynamics.
- **Torsional Strain in string setup:** While we were tuning the drone initially, we forgot the fact that the string would aid in the correction of the drone's attitude. We observed it when the drone became unstable, when the string was left freely.

7.2 Software Challenges

- **Software Library Overheads:** Overheads in the I2C (wire) and PWM libraries of the Aries V3 board limited the achievable loop frequency. This was addressed by **modifying library implementations and directly using system calls in the main C file**, as well as directly using the wire library for MPU6050 communication instead of a dedicated library.
- **Dealing with Vibrations and Noisy Accelerometer Data:** Mechanical vibrations from the cheap motors and ESCs severely corrupted the accelerometer readings, making angle mode control unreliable. Numerous filtering techniques proved largely ineffective. The solution was to **shift to rate mode control with a complementary filter heavily weighted towards the gyroscope**, which was less susceptible to short-term vibrations.

7.3 Control System Optimization

- **Complimentary Filter Tuning:** Finding the appropriate weighting for the complementary filter was crucial. The team learned from the YMFC algorithm to use a **very high weight for the gyroscope (99.96%) and a very low weight for the accelerometer (0.04%)**, effectively operating in rate mode.
- **PID Constant Tuning:** Tuning the P constants, even in rate mode, was challenging. Initial use of P values from the YMFC algorithm led to a crash. The team realised the plant-specific nature of these constants and developed a tuning methodology involving physically restricting the drone's movement. The initial string-based restraint was flawed, necessitating a redesign of the landing gear for more effective isolation of axes. Despite weeks of tuning, a slight steady-state error resulting in forward drift persisted.

7.4 Noise Reduction

Despite trying Kalman, digital low-pass, moving average, and complementary filters, effectively reducing the noise in the accelerometer data proved difficult. The most successful approach to mitigate the impact of noise on flight control was the shift to rate mode with high gyroscope weighting in the complementary filter.

8 Future Improvements

8.1 Advanced Control Algorithms

The team plans to explore more advanced control algorithms beyond the basic P controller to improve stability and performance, and potentially address the steady-state error.

8.2 Additional Sensors

Incorporating additional sensors, such as a barometer or GPS, could enable altitude hold and autonomous navigation capabilities in future iterations.

8.3 Autonomous Capabilities

The ultimate goal is to develop a fully autonomous drone by implementing features like waypoint navigation and obstacle avoidance. The current project serves as a foundational step towards these more advanced capabilities. A key future step is to **design a single PCB integrating the flight controller and other necessary components**.

9 Conclusion

The FlyHigh project successfully navigated numerous technical challenges in the development and tuning of a custom quadcopter flight controller. Significant hurdles related to ESC compatibility, PWM signal integrity, and sensor noise were overcome through methodical investigation and innovative solutions. The shift to rate mode control, inspired by existing algorithms, proved crucial for achieving initial stable flight. The iterative process of PID tuning, while demanding, provided valuable insights into the complexities of real-world control system implementation. The project lays a solid foundation for future work aimed at creating a fully integrated flight controller with advanced capabilities.

A Appendix: Code Repositories

https://www.github.com/RaghavpravinKS/flyhigh

B Appendix: Component Specifications

References to component specifications are made (e.g., PWM frequency for ESCs), but detailed specification sheets are not included in the excerpts, though the MPU6050 and nRF24L01+ specifications are cited in the bibliography.

C Appendix: Testing Data

While we logged sensor data, this data is not included in the provided excerpts.

References

[1] InvenSense Inc., *MPU-6000 and MPU-6050 Product Specification*, Document Number: PS-MPU-6000A-00, Revision: 3.4, Release Date: 08/19/2013.

[2] Nordic Semiconductor, *nRF24L01+ Single Chip 2.4GHz Transceiver Product Specification*, v1.0, September 2008.

[3] Astrom, K.J. and Hagglund, T., *PID Controllers: Theory, Design, and Tuning*, Instrument Society of America, 1995.