Technical Documentation

Firmware Development and Control System Design for a Multirotor Drone

TEAM 20

Abhinav Singh	Mechanical Engineering
Mohan Krishna	Mechanical Engineering

January 10, 2026

Contents

1 Overview

This report presents the design, development, and evaluation of a firmware-based control system for a multirotor drone, with the primary objective of achieving stable and predictable flight under manual control. The work focuses on bridging theoretical control concepts with practical firmware implementation, while accounting for real-world constraints such as sensor noise, actuator nonlinearities, and hardware limitations.

The scope of the work includes experimental characterization of the propulsion system, dynamic modeling of the drone, analysis of open-loop stability, and the design of a cascaded control architecture. A phased firmware development strategy was adopted, beginning with sensor-level validation and progressing through arming logic, sensor integration, bias compensation, joystick command mapping, and closed-loop control implementation.

Given the constraints of limited hardware capability, development time, and safety considerations, manual PID tuning was selected as the primary method for controller refinement. Tuning was carried out iteratively using a test rig and small-amplitude control inputs, allowing systematic observation of system behavior while minimizing operational risk. Sensor fusion and filtering techniques were implemented to mitigate noise, drift, and sensitivity issues inherent in low-cost onboard sensors.

Throughout the development process, emphasis was placed on user-centric design, ensuring intuitive control response, pilot-in-the-loop operation, and clear visual feedback during flight. The resulting firmware represents a practical and robust solution for manual flight stabilization, while also providing a foundation for future enhancements such as advanced system identification, adaptive control, and autonomous operation.

2 Dynamic Modelling of the Multirotor Drone

2.1 Physical Parameters and Experimental Data

The quadrotor platform used in this work has the following experimentally measured parameters:

Total Mass

$$m = 162\,\mathrm{g} = 0.162\,\mathrm{kg}$$

Propeller Dimensions

- Length: 64 mm
- Width: 15 mm
- Height: 8.5 mm

Battery Voltage During Tests

$$V \approx 4.2\,\mathrm{V}$$

Experimental thrust tests were conducted for a single motor–propeller pair. The thrust–throttle relationship revealed a dead-band region.

- Dead-band lower limit: 16–18% throttle
- Effective thrust generation: $\approx$ 20% throttle
- Thrust saturation: $\approx$ 88% throttle

Maximum thrust generated by one motor:

$$T_{\text{max}} \approx 75.5\,\text{g} = 0.74\,\text{N}$$

Total thrust available:

$$T_{\text{total,max}} \approx 4 \times 0.74 = 2.96\,\text{N}$$

Thrust required to hover:

$$mg = 0.162 \times 9.81 \approx 1.59\,\text{N}$$

This confirms that the system possesses a sufficient thrust-to-weight ratio for stable flight.

2.2 Open-Loop Equations of Motion

2.2.1 Translational Dynamics

The vertical motion of the quadrotor is governed by Newton's second law:

$$m\ddot{z} = T - mg \tag{1}$$

where:

- z is the vertical position (m),
- $\ddot{z}$ is the vertical acceleration (m/s^2),
- $T = \sum_{i=1}^{4} T_i$ is the total thrust generated by the four motors (N),
- m is the total mass of the drone (kg),
- g is the gravitational acceleration (9.81 m/s^2).

Linearizing the system about the hover condition ($T \approx mg$), the equation becomes:

$$m\ddot{z} = \Delta T \tag{2}$$

Taking the Laplace transform of the linearized equation yields:

$$\frac{Z(s)}{\Delta T(s)} = \frac{1}{ms^2} \tag{3}$$

This transfer function represents a double integrator, which is unstable in open loop.

2.2.2 Rotational Dynamics

For small angular motions about the body-fixed axes, the rotational dynamics are described by Euler's equations.

Roll and Pitch Dynamics

$$I_x\ddot{\phi} = \tau_\phi, \qquad I_y\ddot{\theta} = \tau_\theta \tag{4}$$

Yaw Dynamics

$$I_z\dot{r} = \tau_\psi \tag{5}$$

where:

- ϕ and θ are the roll and pitch angles (rad),
- r is the yaw rate (rad/s),
- I_x, I_y, and I_z are the moments of inertia about the respective body axes (kg·m^2),
- τ_ϕ, τ_θ, and τ_ψ are the control torques (N·m).

From these equations, the open-loop characteristics of the rotational dynamics are:

- Roll and pitch behave as double integrators,
- Yaw behaves as a single integrator.

All rotational modes are either unstable or marginally stable in open loop and therefore require feedback control for stabilization.

2.3 Thrust Dead-Band and Its Effect on Dynamics

Experimental thrust measurements indicate the presence of a dead-band region in thrust generation, characterized by:

$$\delta_{db} \approx 0.18 \quad \text{(18\% throttle)} \tag{6}$$

Below this threshold, increases in throttle input produce negligible thrust. Above the dead-band region, thrust increases approximately quadratically with motor speed:

$$T \approx k_f \omega^2 \tag{7}$$

The existence of this dead-band leads to several undesirable flight characteristics:

- delayed lift-off,
- poor low-throttle control authority,
- steady-state drift even when zero control input is commanded.

To mitigate these effects, control inputs must be offset by a bias term to shift the operating point beyond the dead-band, thereby improving control effectiveness near hover.

2.4 Open-Loop Stability Analysis

From the linearized equations derived in the previous sections, the characteristic equations of the multirotor dynamics can be expressed as follows.

Altitude Dynamics

$$s^2 = 0 \tag{8}$$

Roll and Pitch Dynamics

$$s^2 = 0 \tag{9}$$

Yaw Dynamics

$$s = 0 \tag{10}$$

The altitude, roll, and pitch dynamics exhibit double poles at the origin, while the yaw dynamics exhibit a single pole at the origin. This indicates that the multirotor system is open-loop unstable or marginally stable. As a result, feedback control is essential to achieve stabilization.
To illustrate this behavior, an open-loop time-domain simulation was performed by applying a small initial angular disturbance, demonstrating the absence of inherent restoring dynamics in the system.

MATLAB Code: Open-Loop Pitch Response to Initial Disturbance

```
% Moment of inertia about pitch axis (estimated)
Iy = 2.5e-3;   % kg m^2

% Time span
tspan = [0 2];

% Small initial angular disturbance (linear region)
theta0 = 5 * pi/180;   % 5 degrees
omega0 = 0;

x0 = [theta0; omega0];

% Open-loop pitch dynamics (no control torque)
f = @(t,x) [
      x(2);
      0
];

[t,x] = ode45(f, tspan, x0);

figure;
plot(t, x(:,1)*180/pi, 'LineWidth', 1.5);
```

```
xlabel('Time␣(s)');
ylabel('Pitch␣angle␣(deg)');
title('Open-loop␣Pitch␣Response␣to␣Initial␣Disturbance');
grid on;
```

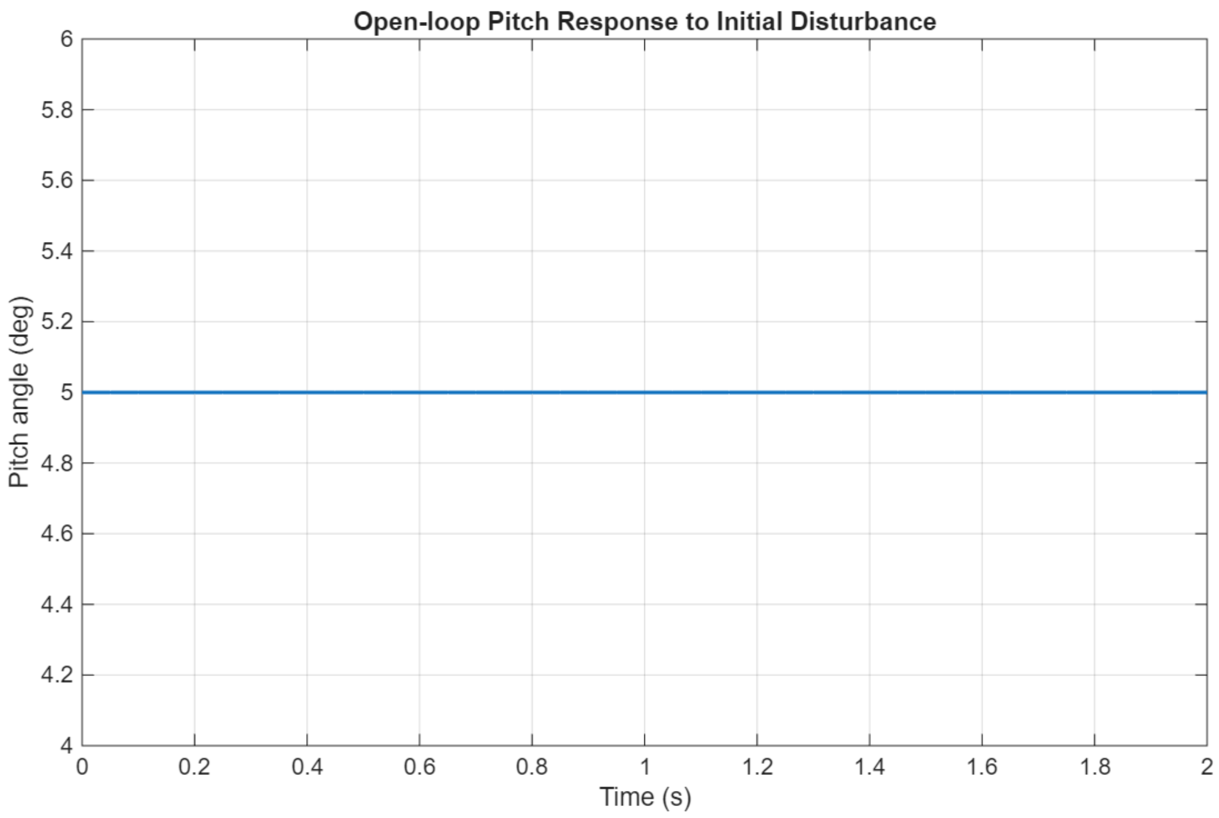

Figure 1: Enter Caption

3 Control Architecture (Cascaded Structure)

The implemented control system follows a **cascaded architecture:**
Inner loop: attitude control (roll, pitch, yaw)
Outer loop: altitude control through throttle modulation

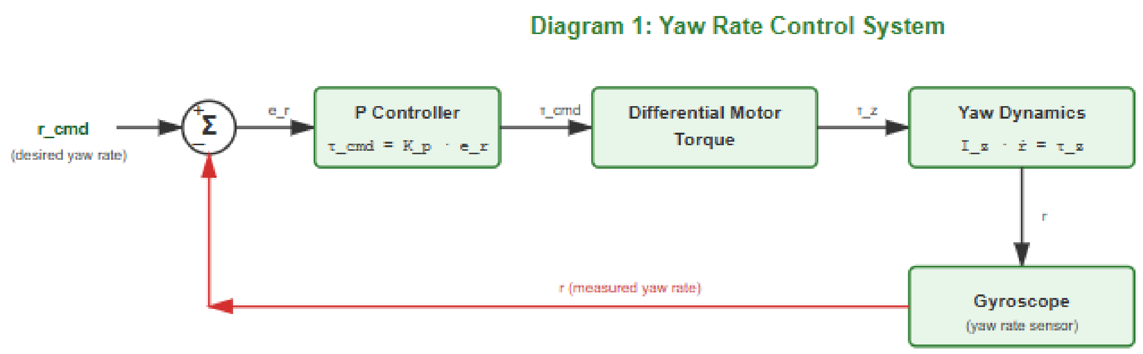

Figure 2: Yaw rate Block diagram

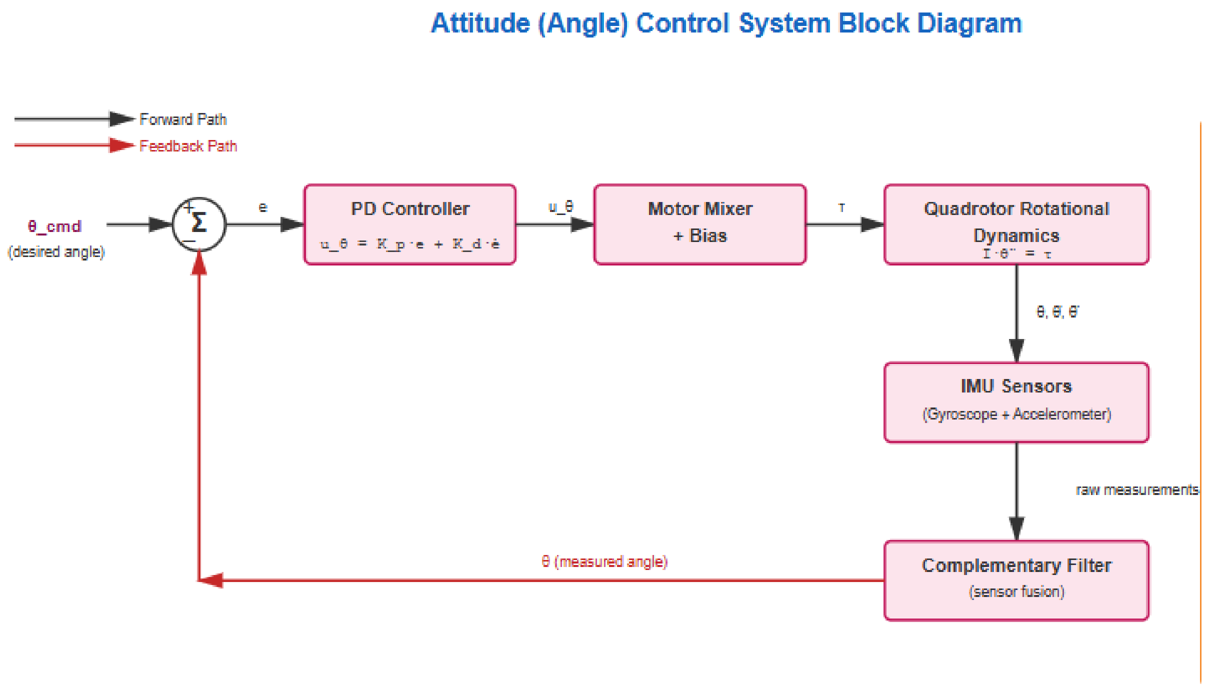

Figure 3: Attitude Block diagram

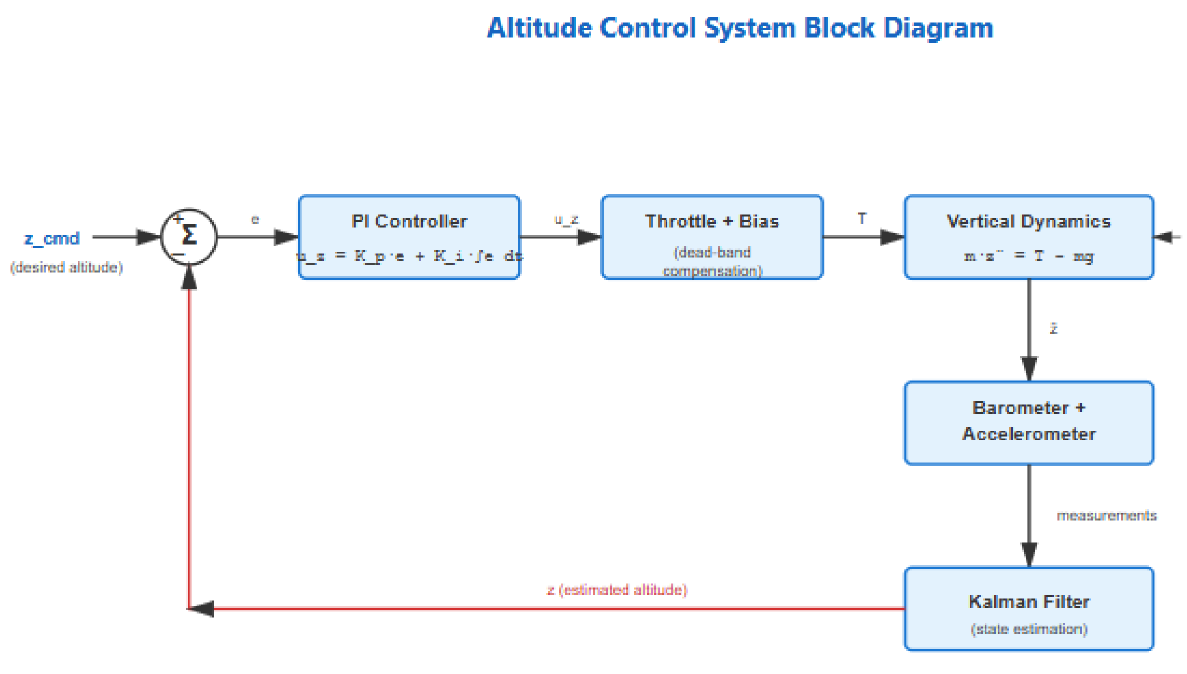

Figure 4: Altitude Block diagram

Although implemented in a simplified form, this structure is consistent with standard multirotor control design practices.

3.1 Dead-Band Compensation (Bias Modeling)

To overcome the thrust dead-band identified in the experimental analysis, a constant bias term is introduced in the control input:

$$u = u_{\text{control}} + u_b \tag{11}$$

where u_b corresponds to the minimum throttle required to exit the dead-band region (approximately 20% throttle).

The introduction of this bias shifts the equilibrium operating point of the system beyond the dead-band without altering the underlying system dynamics. As a result, control effectiveness near hover is improved, and steady-state drift caused by insufficient thrust authority at low throttle levels is reduced.

3.2 PID Controller Design and Stability

Since the open-loop dynamics of the multirotor system are unstable or marginally stable, PID-based feedback control is employed to achieve stabilization.

The linearized plant models used for controller design are as follows.

Pitch and Roll Dynamics

$$G_\theta(s) = \frac{1}{I_y s^2} \tag{12}$$

Altitude Dynamics

$$G_z(s) = \frac{1}{ms^2} \tag{13}$$

Yaw Rate Dynamics

$$G_r(s) = \frac{1}{I_z s} \tag{14}$$

3.2.1 Pitch and Roll Control Using PD Controller

A proportional-derivative (PD) controller is employed for roll and pitch stabilization:

$$C(s) = K_p + K_d s \tag{15}$$

The closed-loop characteristic equation becomes:

$$I_y s^2 + K_d s + K_p = 0 \tag{16}$$

Comparing this with the standard second-order characteristic equation:

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = 0 \tag{17}$$

yields the relationships:

$$\omega_n^2 = \frac{K_p}{I_y}, \qquad 2\zeta\omega_n = \frac{K_d}{I_y} \tag{18}$$

3.2.2 Altitude Control Using PI Controller

The altitude dynamics are given by:

$$m\ddot{z} = u \tag{19}$$

A proportional-integral (PI) controller is used for altitude control:

$$C(s) = K_p + \frac{K_i}{s} \tag{20}$$

The resulting closed-loop characteristic equation is:

$$ms^2 + K_p s + K_i = 0 \tag{21}$$

3.2.3 Yaw Control

Yaw dynamics are first order:

$$I_z \dot{r} = \tau_\psi \tag{22}$$

Due to the first-order nature of the yaw dynamics, proportional rate feedback is sufficient to achieve yaw stabilization.

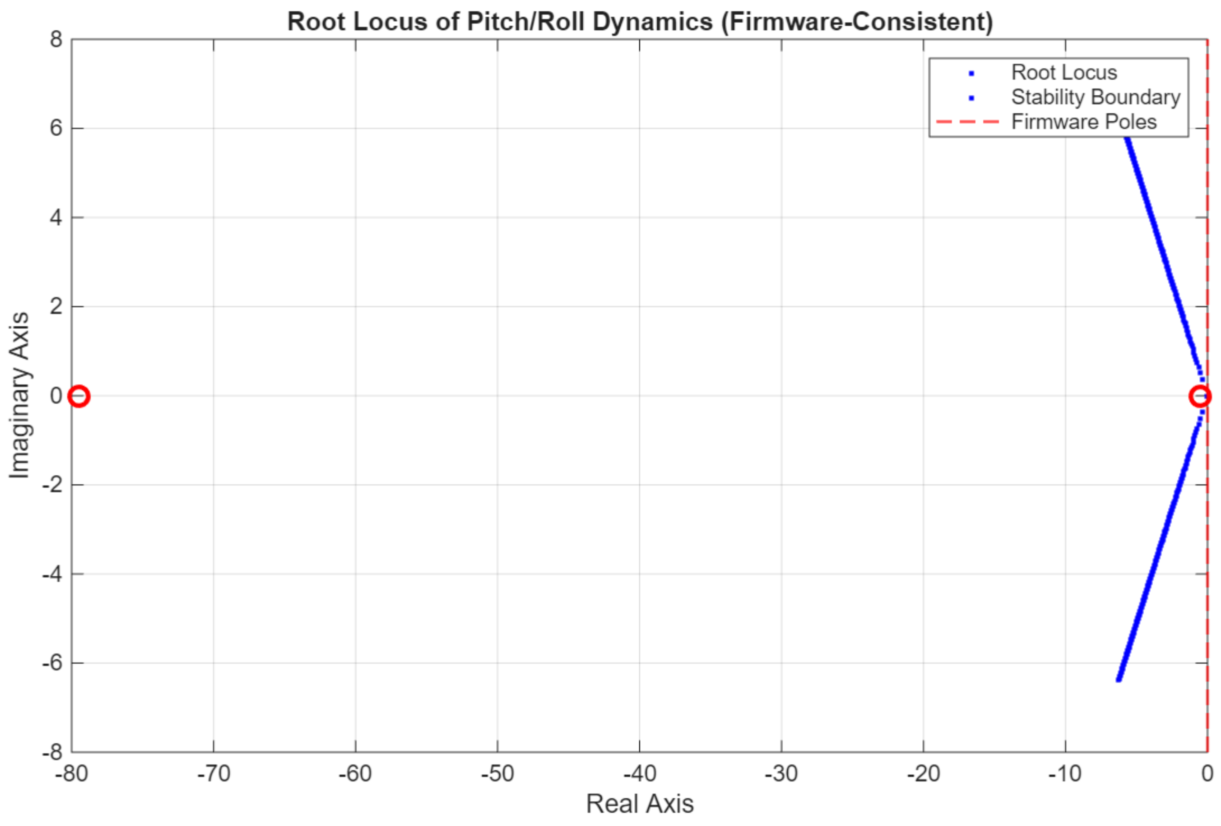

Figure 5: Root locus Plot

4 Firmware Development Strategy

4.1 User Objectives and Development Considerations

The firmware development strategy was guided by clearly defined user objectives, with an emphasis on practical operation, safety, and development efficiency. The key user objectives were as follows:

- Achieving stable and predictable flight behavior under manual control.
- Ensuring intuitive and consistent control response for operators with varying levels of experience.
- Maintaining pilot-in-the-loop operation, preserving direct user control authority at all times.
- Supporting ease of tuning, testing, and incremental development.
- Providing clear visual orientation feedback during flight using colored LEDs to help identify the drone's heading and orientation.
- Enabling user-controlled activation of a front-facing white LED to improve visibility in low-light conditions and enhance situational awareness when required.
- Implementing fail-safe mechanisms to ensure safe system behavior during communication loss, sensor faults, or abnormal operating conditions.

- Minimizing operational and handling risks during development and testing.

To meet these objectives, the firmware was designed with a focus on simplicity, reliability, and robustness, while accounting for hardware and computational constraints. Control logic and parameters were structured to allow straightforward debugging and progressive refinement without compromising safety.
Additionally, Wi-Fi–based firmware flashing and parameter update mechanisms were explored to reduce dependence on wired connections. This approach was intended to enable faster PID tuning and firmware iteration, particularly when the drone is mounted on a test rig, thereby improving development efficiency and reducing risks associated with repeated physical handling.

4.2 Phased Firmware Implementation

4.2.1 Sensor-Level Firmware Validation

As an initial step in firmware development, independent test codes were implemented for the IMU and barometer sensors to verify correct hardware interfacing and data integrity. This step focused on validating communication protocols, sensor initialization, and real-time data acquisition without involving control logic or motor actuation.
Each sensor was tested in isolation to confirm expected output ranges, stability of readings, and responsiveness to physical motion or environmental changes. This approach enabled early identification of wiring issues, configuration errors, and sensor noise characteristics. Validating sensors independently ensured that subsequent control behavior could be attributed to firmware logic rather than hardware or interface faults.

4.2.2 Arming and Disarming Logic

The arming mechanism requires a deliberate user action before motor actuation is permitted, while disarming immediately disables motor outputs and returns the system to a safe state. This logic prevents accidental motor startup during power-up, firmware flashing, or sensor testing.
Incorporating arming and disarming as a dedicated firmware layer established a clear operational boundary between safe idle states and active flight states, forming the foundation for all subsequent control and tuning activities.

4.2.3 Sensor Integration and State Estimation

After implementing arming and disarming logic, sensor integration was carried out to enable closed-loop control of the drone. Sensor data is essential for estimating the current and previous states of the drone, which form the basis for all control decisions.
The onboard sensors continuously provide measurements related to motion and orientation. These measurements are processed by the flight controller to estimate the drone's state at each control cycle. By comparing the estimated state with the commanded inputs from the user, the firmware computes appropriate control outputs for the next time step.

This sequential estimation and control process allows the drone to respond smoothly to user inputs while correcting deviations caused by disturbances or inherent system dynamics. Reliable sensor integration therefore serves as the foundation for stable control.

4.2.4 Bias Removal Principle

Experimental observation of multirotor propulsion systems shows the presence of a thrust dead-band, in which small throttle commands do not produce effective thrust. In this region, the motor–propeller system is unable to generate sufficient lift, and thrust output remains negligible despite variations in input command.
Once the command exceeds the dead-band threshold, thrust generation becomes effective and increases rapidly with motor speed, following the nonlinear characteristics of the propulsion system. The existence of this dead-band adversely affects flight behavior by causing delayed lift-off, poor low-throttle control authority, and steady-state drift even when zero control input is commanded.
To overcome these effects, a bias is introduced in the control input to shift the operating point beyond the dead-band. By ensuring that control commands are applied around an effective thrust-producing region, bias removal improves responsiveness near hover, enhances low-throttle stability, and enables more reliable and predictable control of the drone.

4.2.5 Joystick Command Mapping and Control Response

Joystick inputs from the transmitter are mapped directly to the drone's control axes, namely roll, pitch, yaw, and throttle. This mapping converts user commands into corresponding control signals for the flight controller, ensuring that the drone responds in the intended direction and manner.
Input sensitivity was adjusted to provide a balance between smooth control and adequate responsiveness. Low sensitivity near the center position enables stable flight, while higher sensitivity at larger stick deflections allows effective maneuvering. Latency between joystick input and drone response was kept minimal to ensure timely control action.

4.2.6 Common Techniques for Drone Control System Development

Several established techniques are used in the development and tuning of drone flight control systems. The choice of technique depends on system complexity, hardware constraints, and development objectives.
Manual PID Tuning: PID gains are adjusted iteratively based on observed flight behavior such as oscillations, sluggish response, or instability. This method relies on operator experience and real-time testing and is widely used during early firmware bring-up and rapid prototyping.
Autotuning-Based PID Estimation: The firmware excites the system with controlled inputs and automatically adjusts PID gains by analyzing the system response. This approach reduces tuning time and provides consistent baseline gains, particularly for standardized airframes.
Model-Based PID Derivation: PID parameters are derived from the mathematical model of the drone using its equations of motion. This method requires accurate knowledge of system dynamics

and is commonly used in simulation-driven or research-oriented development.
Experimental Data–Driven System Identification: Control gains are obtained by collecting real flight data and identifying system parameters using experimental methods. The identified model is then used to compute controller gains, capturing real-world effects such as delays, nonlinearities, and sensor noise.

5 Final Firmware Selection and Justification

In view of limited development time, hardware constraints, and safety considerations, manual PID tuning was selected as the primary control tuning approach. This method enabled direct observation of the drone's response and allowed rapid, controlled adjustments without requiring extensive system identification or data processing.
The tuning process was carried out iteratively using small control inputs and a test rig to validate stability and response, ensuring safe refinement of control gains before full free-flight testing.

5.1 Manual PID Tuning and Gain Refinement

Manual PID tuning was performed iteratively, beginning with a common baseline set of gains and progressively refining them based on observed flight behavior. All tuning was carried out using small control inputs, initial validation on a test rig, and controlled testing to ensure safety and repeatability.

5.1.1 Initial Baseline Gain Selection

At the start of tuning, identical and conservative PID values were applied across roll, pitch, yaw, and altitude to establish a stable reference point and simplify early observations. These baseline values enabled verification of control loop functionality and sensor feedback before axis-specific refinement.

5.1.2 Roll and Pitch Gain Tuning

Roll and pitch were tuned together under the assumption of a symmetric airframe. A proportional gain of 0.10 provided a fast and well-defined response without inducing high-frequency oscillations. A derivative gain of 0.01 was sufficient to reduce overshoot and improve damping.
Integral gain for roll and pitch was set to zero, as no significant steady-state drift was observed. Introducing integral action resulted in slow oscillations due to sensor noise.
Final gains:

$$\text{Roll: } P = 0.10,\ I = 0.00,\ D = 0.01$$

$$\text{Pitch: } P = 0.10,\ I = 0.00,\ D = 0.01$$

5.1.3 Yaw Axis Gain Tuning

Yaw dynamics were observed to be slower and inherently more damped. A proportional gain of 0.05 provided stable yaw response, and no derivative or integral action was required.

Final gains:

$$\text{Yaw: } P = 0.05,\ I = 0.00,\ D = 0.00$$

5.1.4 Altitude Control Gain Tuning

Altitude control was tuned independently. A proportional gain of 0.60 provided adequate vertical response, while an integral gain of 0.20 compensated for steady-state altitude drift caused by thrust nonlinearity.
Final gains:

$$\text{Altitude: } P = 0.60,\ I = 0.20,\ D = 0.00$$

6 Sensor Fusion, Noise Filtering, and Sensitivity Management

Low cost onboard sensors used in small multirotor platforms are inherently noisy and susceptible to drift, vibration, and environmental disturbances. To ensure stable flight, raw sensor data must be processed through appropriate filtering and fusion techniques before being used by the control system.
In this project, sensor fusion is implemented for both attitude and altitude estimation, along with additional filtering mechanisms to reduce sensitivity to noise and prevent control instability.

6.1 Attitude Estimation Using Complementary Filtering

The onboard Inertial Measurement Unit (LSM6DS3) provides measurements from:

- A gyroscope, which measures angular velocity
- An accelerometer, which measures linear acceleration including gravity

Each sensor exhibits complementary characteristics:
Gyroscope data is smooth and responsive but suffers from long-term drift, while accelerometer data provides an absolute gravity reference but is noisy and sensitive to vibration.
To exploit the strengths of both sensors, a complementary filter is implemented in `imu.cpp`.

6.1.1 Accelerometer-Based Angle Estimation

Roll and pitch angles are computed using the gravity vector measured by the accelerometer:

```
float rollAcc  = atan2(ay, az) * 57.2958f;
float pitchAcc = atan2(-ax, sqrt(ay * ay + az * az)) * 57.2958f;
```

6.1.2 Complementary Filter Implementation

The final attitude estimate is obtained using a weighted combination of gyroscope integration and accelerometer correction:

```
roll  = 0.98f * (roll  + gx * dt) + 0.02f * rollAcc;
pitch = 0.98f * (pitch + gy * dt) + 0.02f * pitchAcc;
yaw  += gz * dt;
```

Here:

- Gyroscope integration dominates short-term dynamics (98%)
- Accelerometer correction dominates long-term stability (2%)
- `dt` is the control loop time
- Yaw is estimated purely from gyroscope integration, as no magnetometer is used

This filtering approach suppresses high-frequency vibration noise, reduces gyroscopic drift, and provides smooth real-time attitude estimates suitable for control.

6.2 Gyroscope Bias Estimation and Removal

Gyroscope sensors exhibit constant offset (bias), which causes angular drift even when the drone is stationary. To mitigate this effect, a bias calibration step is performed during initialization in `imu.cpp`.

6.2.1 Bias Estimation

During startup, yaw-rate measurements are averaged while the drone is stationary:

```
for (int i = 0; i < 2000; i++) {
    sumZ += read16(OUTX_L_G + 4);
    delay(2);
}
gyroBiasZ = (sumZ / 2000.0f) * GYRO_SCALE;
```

This assumes zero true rotation during calibration.

6.2.2 Bias Compensation During Runtime

The estimated bias is subtracted from the measured yaw rate:

```
float gz = read16(OUTX_L_G + 4) * GYRO_SCALE - gyroBiasZ;
```

This significantly improves yaw stability and prevents continuous rotation during hover.

6.3 Vertical Acceleration Processing

The accelerometer measures specific force, which includes gravity. To estimate vertical motion, the gravity component must be removed:

```
accZ_ms2 = (az - 1.0f) * G_TO_MS2;
```

This produces gravity-compensated vertical acceleration, which is later used for altitude estimation.

6.4 Altitude Estimation Using Kalman Filtering

Altitude estimation is performed using a Kalman filter implemented in `kalman.cpp`. The filter fuses:

- Barometer altitude measurements (absolute but noisy and slow)
- Vertical acceleration measurements (fast but prone to drift)

6.4.1 Prediction Step

```
X += V * dt + 0.5f * accelZ * dt * dt;
V += accelZ * dt;
```

6.4.2 Measurement Update

```
float K = P / (P + R);
X += K * (measuredAlt - X);
P *= (1.0f - K);
```

Where:

- X is the estimated altitude
- V is the estimated vertical velocity
- P is the estimation uncertainty
- Q and R tune sensitivity to acceleration and barometer noise

This fusion yields a smooth, low-noise altitude estimate suitable for closed-loop control.

6.5 Sensitivity Reduction in Control Inputs

Even after sensor filtering, small fluctuations can cause unwanted actuator activity. To reduce sensitivity, command deadbands are applied:

```
if (abs(rollCmd) < 2 && abs(pitchCmd) < 2)
```

This ensures that small joystick noise or ADC quantization does not trigger altitude corrections. Without this deadband, altitude control would continuously react to minor lateral command noise.

6.6 Integral Windup Protection

Integral action accumulates error over time. If left unchecked, this accumulation can grow excessively during actuator saturation, leading to instability when control authority is restored. This phenomenon is known as integral windup.

6.6.1 Windup Protection Implementation

```
p.i += e * dt;
p.i = constrain(p.i, -40, 40);
```

By limiting the integral term:

- Sudden overshoot is prevented
- Recovery after disturbances is smoother
- Controller stability is preserved during saturation

6.7 Output Saturation for Noise Robustness

Final control outputs are constrained before being sent to the motors:

```
rollPID  = constrain(rollPID,  -25, 25);
pitchPID = constrain(pitchPID, -25, 25);
yawPID   = constrain(yawPID,   -30, 30);
```

This prevents excessive motor commands due to sensor spikes, reduces loss of control during sudden disturbances, and protects the mechanical structure from excessive stress.

6.8 Overview of Sensor Fusion and Filtering Strategy

The sensor fusion and filtering strategy implemented in this system consists of the following key components:

- Complementary filtering for attitude estimation, combining gyroscope and accelerometer measurements to achieve both short-term responsiveness and long-term stability.
- Gyroscope bias calibration for drift removal, significantly improving yaw stability during hover.
- Kalman filtering for altitude estimation by fusing barometer measurements with vertical acceleration.
- Command deadbands and output saturation to reduce sensitivity to sensor noise, joystick jitter, and sudden disturbances.
- Integral windup protection to prevent excessive accumulation of integral error and maintain controller stability during actuator saturation.

Together, these mechanisms ensure reliable state estimation and robust control performance despite noisy sensors, mechanical vibrations, and environmental disturbances, enabling stable and predictable flight behavior.

6.9 Manual Tuning Implementation on the Drone

In the initial tuning phase (Case 1), higher gains were applied:

$$\text{Roll/Pitch: } P = 0.18,\ I = 0,\ D = 0.25$$

$$\text{Yaw: } P = 0.05,\ I = 0,\ D = 0.02$$

$$\text{Altitude: } P = 1.2,\ I = 0.05,\ D = 0.3$$

This configuration resulted in rightward tilt and clockwise yaw, indicating excessive control authority.
In the second phase (Case 2), gains were reduced:

$$\text{Roll/Pitch: } P = 0.14,\ I = 0.03,\ D = 0.32$$

$$\text{Yaw: } P = 0.08,\ I = 0.01,\ D = 0.04$$

$$\text{Altitude: } P = 1.1,\ I = 0.05,\ D = 0.3$$

This reduced tilt bias but introduced oscillations due to excessive derivative and integral action.
In the final phase (Case 3), conservative gains were applied:

$$\text{Roll/Pitch: } P = 0.10,\ I = 0.00,\ D = 0.01$$

$$\text{Yaw: } P = 0.05,\ I = 0.00,\ D = 0.00$$

$$\text{Altitude: } P = 0.60,\ I = 0.20,\ D = 0.00$$

This configuration resulted in minimal drift, reduced oscillations, and stable altitude acquisition. Although perfectly stable flight was not achieved due to sensor noise, thrust nonlinearity, and mechanical asymmetries, the final gains provided a reliable and robust flight response.
Drafted experimental data was recorded for reference during tuning.
Link for Draft of Readings.
The video shows the drone responding to small control inputs while variations in roll, pitch, yaw, and altitude were visually monitored. The test rig restricted translational motion while allowing rotational movement, making it easier to identify oscillations, drift, and damping effects. Gain values were adjusted incrementally based on the observed response, enabling stable tuning without the risks associated with free flight. The video provided below is included for reference to illustrate the tuning process and observed behavior.
Manual Tuning Implemnetation on drone with help of a Testing Jig

7 Manual PID Gain Validation with Theoretical PID Gains

Theoretical PID gains derived from linearized system models and root locus analysis were used as an initial reference for the controller design. Although these gains did not result in fully stable flight when applied directly to the physical system, they provided an essential framework for guiding the tuning process. Final controller gains were therefore obtained through manual validation in order to account for real-world effects not captured by the theoretical model.
The key observations from this validation process are summarized as follows:

- Theoretical gains provided correct orders of magnitude, enabling the identification of appropriate gain ranges and preventing the selection of unstable or non-physical initial values.
- These gains established boundary conditions for manual tuning, ensuring that the controller parameters remained within regions predicted to be stable by linear analysis.
- Direct implementation of theoretical gains resulted in high sensitivity to sensor noise and structural asymmetries, which manifested as oscillations and control imbalance.
- Practical deviations from the theoretical model arose due to unmodeled dynamics, including uneven mass distribution, motor–propeller mismatch, thrust dead-band effects, actuator saturation, and frame-induced vibrations.
- Sensor imperfections and state estimation noise further amplified control effort when higher theoretical gain values were used.

Consequently, manual tuning was necessary to reduce sensitivity, improve robustness, and achieve stable flight performance across a range of operating conditions. In conclusion, theoretical PID design served as a critical starting point and validation reference, while experimental refinement ensured reliable and robust controller performance on the physical multirotor platform.

8 Validation and Final Gain Selection

Validation and Final Gain Selection The final set of PID gains was validated using small amplitude control inputs, test rig verification, and hover based step I/P testing. Prior to bias removal, the system exhibited constant drift and unstable behavior, particularly near low throttle and hover conditions, which limited the effectiveness of the tuned gains. After applying bias removal, the operating point was shifted outside the thrust dead-band, resulting in a noticeable improvement in control authority and stability. The tuned system then demonstrated immediate response, quick settling, and stable behavior across all control axes without sustained oscillations or drift. These gains, in conjunction with bias removal, were therefore selected as the final configuration for manual flight operation due to their improved responsiveness, stability, and overall robustness.

9 Control Commands and Navigation Lighting

9.1 Control Command Interface

The drone is operated using a standard transmitter interface, where user inputs are mapped directly to the primary control axes: roll, pitch, yaw, and throttle. These commands represent the pilot's intent and are processed by the onboard firmware to generate appropriate motor control signals.
Roll and pitch commands control lateral and longitudinal motion of the drone, enabling directional movement and attitude adjustment. Yaw commands control the rotational motion about the vertical axis, allowing the drone to change heading. Throttle commands regulate overall thrust and are used to control altitude and vertical motion. Command scaling and sensitivity were tuned to ensure smooth response near the neutral stick position while still allowing sufficient authority for maneuvering.
To prevent unintended behavior due to small input noise or joystick jitter, deadbands were applied around the neutral positions of the control inputs. This ensures that minor fluctuations do not result in unnecessary control action, contributing to stable hover and improved user handling.

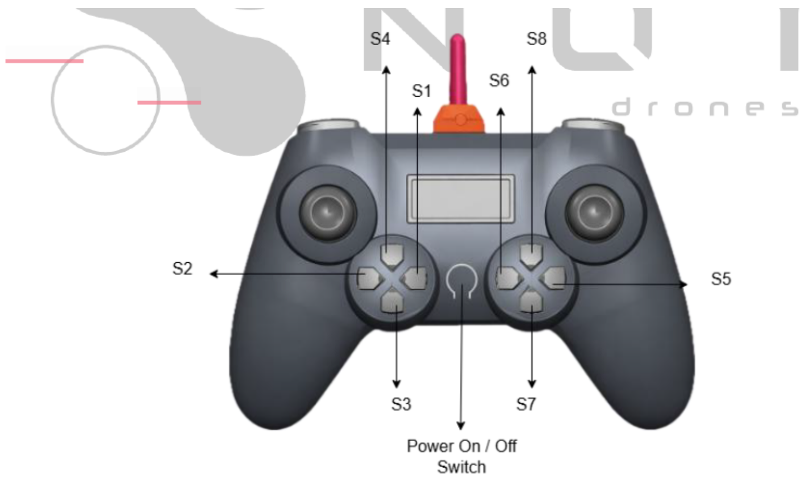

Figure 6: Tx controller

Here S3-To enable front LEDs.

9.2 Navigation Lights and Visual Feedback

Navigation lights were implemented to provide clear visual feedback regarding the drone's orientation and operational state. Colored LEDs were used to help the operator easily identify the front and rear of the drone during flight, which is particularly important for manual control and orientation awareness.
In addition to orientation indicators, a front-facing white LED was integrated as a navigation and visibility aid. This light can be enabled or disabled by the user through a dedicated control command, allowing improved visibility in low-light conditions or during specific maneuvers.
The use of navigation lights enhances flight safety, reduces operator confusion, and supports intuitive piloting, especially during hover, orientation changes, and low-visibility operation.

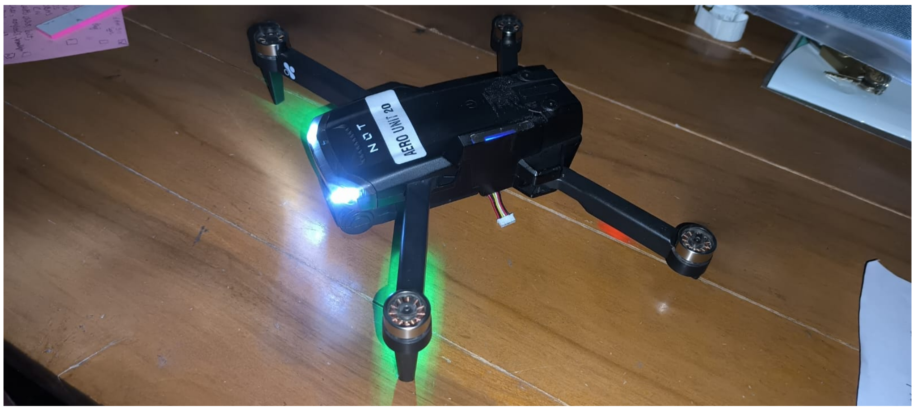

Figure 7: Navigation LEDs enabled

10 Limitations and Known Issues

The current system has the following limitations, mainly due to hardware constraints and practical testing conditions:

- Uneven placement of hardware components on the drone, which can cause slight imbalance during flight.
- Lack of a proper test bench for extended ground testing, limiting safe and repeatable experiments.
- Noise in sensor readings, which affects measurement reliability and control smoothness.
- Limited ability to fully remove drift from sensors, especially during long flight durations.
- Nonlinear motor and propeller behavior at low throttle levels, making fine control more difficult.
- Changes in battery voltage during operation, which influence motor performance and control response.
- Dependence on manual tuning, requiring re-adjustment when system conditions change.
- Sensitivity to environmental factors such as airflow and vibrations during testing.

11 Challenges Faced During Development

- **Slow firmware flashing:** Firmware updates required repeated USB flashing, which was time-consuming and required physical access to the hardware. Wireless (Wi-Fi–based) flashing would have significantly reduced iteration time.

- **Sensor noise and vibrations:** Low-cost sensors were sensitive to vibrations and noise, affecting stability and requiring additional filtering and tuning.
- **Hardware inconsistencies:** Variations in motor thrust, propeller balance, and weight distribution caused asymmetric behavior that was not predicted during design.
- **Limited debugging capability:** Lack of real-time telemetry and onboard logging made it difficult to diagnose issues during flight tests.

12 Future Scope and Planned Improvements

Future work can focus on improving the accuracy of plant modeling and controller tuning through the following approaches:

- Exploring experimental system identification methods by collecting flight data and estimating the drone's dynamic behavior from real measurements.
- Using model-based approaches to derive PID gains from the mathematical representation of the drone dynamics.
- Implementing autotuning techniques to automatically adjust PID gains based on the system response.
- Reducing dependence on manual tuning by validating controller gains through simulation before flight testing.
- Improving robustness of the control system under changing hardware configurations and environmental conditions.

These improvements can lead to more reliable, repeatable, and stable flight performance in future iterations of the system.

13 Conclusion

This work demonstrates the successful development of a firmware-based control system capable of achieving stable and predictable flight for a multirotor drone under manual control. Through a combination of experimental thrust characterization, dynamic modeling, cascaded control architecture design, and systematic firmware implementation, the key challenges associated with small multirotor stabilization were addressed.
Manual PID tuning, supported by test-rig-based validation and controlled hover testing, proved effective in achieving a reliable balance between responsiveness and stability. The introduction of thrust bias compensation significantly improved low-throttle control authority, while sensor fusion and filtering techniques enhanced state estimation robustness in the presence of noise and disturbances.

Although certain limitations remain, including sensor noise, hardware asymmetries, and reliance on manual tuning, the final firmware configuration provides consistent and safe flight behavior suitable for practical operation. The results highlight the importance of iterative testing, bias compensation, and sensitivity management in real-world drone control systems.

Overall, this project establishes a solid foundation for future work in advanced controller design, system identification, and autonomous flight capabilities. The insights gained from this development process are directly applicable to further research and refinement of embedded flight control systems for multirotor platforms.

14 References

References

[1] MathWorks, *PID Tuner App — MATLAB Control System Toolbox Documentation*. Available at: https://www.mathworks.com/help/control/ref/pidtuner-app.html

[2] PX4 Development Team, *Multicopter PID Tuning Guide*. Available at: https://docs.px4.io/main/en/config_mc/pid_tuning_guide_multicopter

Acknowledgements

We would like to thank NOT Drones for providing the drone kits that made this work possible. Having access to the hardware allowed us to directly implement, test, and validate the firmware and control strategies discussed in this report.We are also grateful for the mentorship and continuous support provided by the team of NOT Drones throughout the hackathon. Their guidance, technical suggestions, and timely feedback were extremely helpful during development and testing.