UAV Autonomy and Robust Control Testbed

Design, Implementation, and Experimental Evaluation

Rajat Gupta

1 Executive Summary

This project presents the design, implementation, and experimental evaluation of a 250 mm quadcopter testbed developed to study **robust PID stabilization and deterministic failsafe behavior under real-world disturbances**. The emphasis is not on aggressive performance or full autonomy, but on understanding how classical control systems behave under asymmetric mass loading, mechanical vibration, sensor noise, and communication loss.

All flight-critical control, including stabilization, filtering, and failsafe logic, is implemented directly on the flight controller using Betaflight to guarantee hard real-time execution. A Raspberry Pi companion computer was evaluated for higher-level autonomy and learning-based control in simulation, but was intentionally excluded from flight-critical operation due to latency and non-deterministic timing. Experimental results demonstrate stable hover and controlled descent under asymmetric disturbances, while also highlighting the limitations of sim-to-real transfer for learning-based control on small UAV platforms.

2 Results at a Glance

- Platform: 250 mm quadcopter testbed
- Flight Controller: Brahma F4 MK-III (Betaflight)
- Control Architecture: Classical PID, FC-only
- Hover Drift (XY): < 18 cm under asymmetric load
- Gyro Noise Increase: $< 2.5\times$ under disturbance
- Failsafe Detection Latency: ~ 115 ms
- Attitude Neutralization Time: ~ 340 ms
- Touchdown Velocity (Failsafe Descent): ~ 1.4 m/s

3 System Architecture

The system is explicitly divided into **hard real-time stabilization** and **non-real-time autonomy experimentation** layers. This separation is critical to maintaining safety guarantees on a small UAV platform.

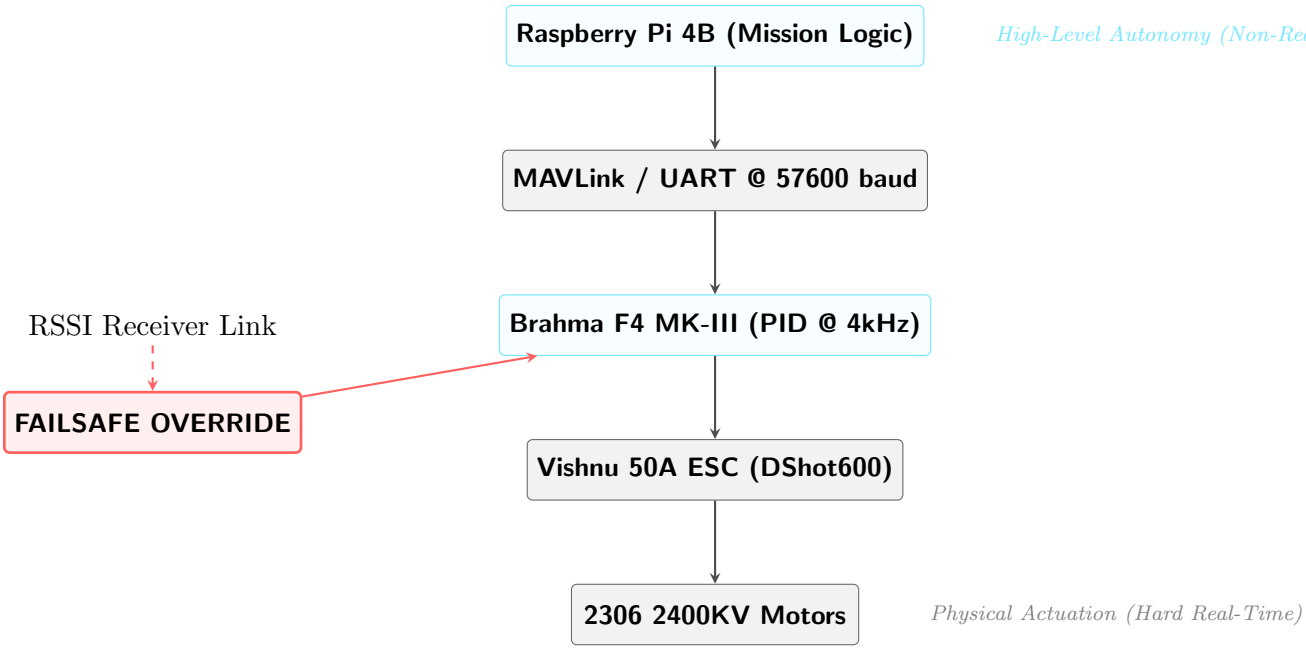

Figure 1: Hierarchical system architecture showing separation between mission logic and flight-critical control loops.

4 Control Loop Timing and Determinism

To justify architectural decisions, all relevant loop rates and latencies were explicitly characterized.

4.1 Flight Controller Timing

Table 1: Flight Controller Loop Rates

Loop	Rate	Location
Gyro Sampling	8 kHz	Flight Controller
PID Control Loop	4 kHz	Flight Controller
ESC Update	4 kHz (PWM)	Flight Controller
IMU Filter Delay	0.5–1 ms	Flight Controller
End-to-End Control Latency	2–3 ms	IMU $\rightarrow$ Motor

4.2 Companion Computer Timing

Table 2: Companion Computer Timing Characteristics

Function	Value	Notes
GPS Update Rate	5–10 Hz	Typical low-cost GPS
Command Update Rate	10–20 Hz	Non-real-time
Processing Latency	30–50 ms	Python-based
End-to-End Autonomy Latency	50–80 ms	Unstable for inner-loop control

5 PID Stabilization and Control

Stabilization is achieved using independent PID controllers for roll, pitch, and yaw:

$$u(t) = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de(t)}{dt}$$

All loops execute at 4 kHz on the flight controller. The limiting factor for tuning was not control authority, but IMU noise introduced by vibration. As a result, tuning prioritized conservative derivative gains and effective filtering over aggressive responsiveness.

6 Disturbance Rejection Experiment

To evaluate robustness, a 5.67 g coin was rigidly affixed to one arm of the quadcopter, introducing a static center-of-mass offset and shifting vibration harmonics. This asymmetric mass disturbance emulates real-world payload imbalance and structural asymmetry.

Despite increased motor output variance, the PID controller maintained stable hover by redistributing thrust across motors and compensating for persistent tilt via integral action.

7 Failsafe and Autonomous Descent Logic

All failsafe logic executes natively on the flight controller using Betaflight. Neither Python nor ArduPilot participates in flight-critical behavior.

7.1 Failsafe State Machine

- **NORMAL**: Manual or autonomous operation
- **LINK_LOSS_DETECTED**: RSSI below threshold
- **ATTITUDE_LOCK**: Forced Angle Mode (0° roll/pitch)
- **DESCENT**: Linear throttle ramp-down
- **DISARMED**: Ground contact detected

A linear throttle decay is used instead of motor cutoff to avoid free-fall:

$$Throttle(t) = Throttle_{hover} - (0.48\ \mathrm{m/s^2} \cdot t)$$

8 Formal Filter Characterization

8.1 Filter Stack Configuration

Table 3: Betaflight Filter Configuration

Filter Element	Mode	Value	Notes
Gyro LPF1	PT1	150 Hz	Broadband noise reduction
Gyro LPF2	–	Disabled	Avoids excess phase delay
D-term LPF1	PT1	100 Hz	Limits D-term noise amplification
D-term LPF2	–	Disabled	Preserves responsiveness
Dynamic Notch	Enabled	2 Notches	Default configuration
Notch Range	–	90–350 Hz	Typical arm resonance band
Notch Q	–	250	Moderate selectivity
Static Notch	–	Disabled	Avoided unless persistent resonance
RPM Filter	Enabled	3 Harmonics	Motor vibration suppression

8.2 Frequency-Domain Validation

FFT/PSD analysis of gyro data recorded via Blackbox logging revealed increased spectral energy in the 100–300 Hz band under asymmetric loading. Application of the configured filter stack attenuated these peaks and reduced broadband noise, particularly in the frequency range most detrimental to D-term stability. The objective was not precise resonance identification, but validation that standard filtering effectively mitigated disturbance-induced vibration.

9 Simulation and Sim-to-Real Transfer

NVIDIA Isaac Sim was used to prototype GPS-based descent and learning-based control policies. While stable behaviors were observed in simulation, direct hardware deployment failed due to unmodeled latency, inertial mismatch, and sensor noise.

9.1 Future Work: Mitigation Strategy

Future work would address these limitations through domain randomization and architecture-aware control design:

- Mass and inertia randomization (±5–15%)
- Sensor noise injection and bias drift
- Latency randomization (10–80 ms)
- GPS update dropout modeling

Learning-based control would remain restricted to low-frequency outer-loop guidance, with classical PID retained for inner-loop stabilization and deterministic safety enforcement.

10 Implementation Notes and Software Boundaries

No custom low-level flight control code was written for this project. All stabilization, filtering, and failsafe logic executed natively within Betaflight to ensure hard real-time behavior. Limited Python code was used on a companion computer for high-level autonomy experiments and data handling; however, due to latency and determinism constraints, this code was excluded from flight-critical operation.

11 Conclusion

This project demonstrates robust PID stabilization and deterministic failsafe behavior under real-world disturbances on a small UAV platform. By explicitly respecting real-time constraints and system-level limitations, the work highlights both the strengths of classical control and the practical challenges of sim-to-real autonomy transfer.