Afro-Asian Journal of Scientific Research (AAJSR)
المجلة الأفروآسيوية للبحث العلمي
E-ISSN: 2959-6505
Volume 4, Issue 2, 2026
Page No: 286-295
Website: https://aajsr.com/index.php/aajsr/index
SJIFactor 2024: 5.028 ISI 2025: 0.915 معامل التأثير العربي (AIF) 2025: 0.76

Real-Time Processing Limitations of Arduino-Based Flight Controllers for UAV Stabilization: An Experimental Analysis

Shtwan S Obaid[1*], Marwa M Al-Sanusi[2], Niemah T Al-Salheen[3]
[1,2,3]Department of Electrical and Electronic Engineering Technology, College of Science and Technology, Qaminis, Ministry of Technical and Vocational Education, Qaminis, Libya

محددات الأداء الزمني لوحدات التحكم في الطيران المعتمدة على أردوينو في تثبيت الطائرات بدون طيار: دراسة تحليلية تجريبية

شتوان سالم اعبيد[1*]، مروة موس السنوسي[2]، نعمة الطيب الصالحين[3]
[1،2،3]قسم هندسة الإلكترونيات والكهرباء، كلية العلوم والتكنولوجيا، قمينس، وزارة التعليم الفني والتقني، ليبيا.

*Corresponding author: shatwansalem89@gmail.com

Received: March 15, 2026	Accepted: April 30, 2026	Published: May 15, 2026

Abstract:
The increasing demand for drone-based delivery systems has highlighted the need for low-cost, accessible flight control platforms for prototyping and educational purposes. This paper presents an experimental investigation into the use of the Arduino UNO microcontroller as a flight controller for a quadcopter delivery drone. The prototype was constructed using an S500 airframe, four A2212 1000KV brushless DC motors, 30A electronic speed controllers (ESCs), an MPU6050 inertial measurement unit (IMU), and a FlySky FS-i6 transmitter-receiver system. A complete control program was developed in the Arduino IDE, implementing a complementary filter for attitude estimation and a PID (Proportional-Integral-Derivative) control loop to stabilize the aircraft. Experimental tests revealed critical limitations of the Arduino platform when applied to real-time flight control. The 8-bit ATmega328P processor operating at 16 MHz exhibited significant processing latency, with measured control loop jitter of ±225 µs and throttle response delay of 40-60 ms, both insufficient for the high-frequency data fusion required for stable multi-rotor flight. While individual component tests confirmed the functionality of all subsystems, integrated flight tests demonstrated excessive oscillation (3-5 Hz, amplitude ±15°), poor dynamic response, and an inability to maintain stable hover under varying throttle conditions. The results conclusively demonstrate that while the Arduino UNO is suitable for initial component verification and educational proof-of-concept, its limited processing power and lack of integrated sensor fusion capabilities render it fundamentally inadequate for practical UAV stabilization applications. This paper provides quantitative evidence of these limitations and establishes a scientific basis for transitioning to specialized flight controllers in drone delivery systems.

Keywords: Arduino UNO; Flight Controller; UAV; Quadcopter; PID Control; MPU6050; Real-Time Processing.

الملخص:
أدى الطلب المتزايد على أنظمة التوصيل باستخدام الطائرات المسيّرة إلى تسليط الضوء على الحاجة إلى منصات تحكم طيران منخفضة التكلفة وسهلة الاستخدام لأغراض النماذج الأولية والتعليمية. تقدم هذه الورقة البحثية دراسة تجريبية حول استخدام متحكم Arduino UNO الدقيق كوحدة تحكم طيران لطائرة مسيّرة رباعية المراوح مخصصة للتوصيل. تم بناء

النموذج الأولي باستخدام هيكل طائرة S500، وأربعة محركات تيار مستمر بدون فرش A2212 بقوة 1000 كيلو فولت، ووحدات تحكم إلكترونية في السرعة (ESCs) بقدرة 30 أمبير، ووحدة قياس بالقصور الذاتي MPU6050، ونظام إرسال واستقبال FlySky FS-i6. تم تطوير برنامج تحكم كامل في بيئة تطوير Arduino المتكاملة (IDE)، حيث تم تطبيق مرشح تكميلي لتقدير الوضع وحلقة تحكم PID (تناسبي-تكاملي-تفاضلي) لتحقيق استقرار الطائرة. كشفت الاختبارات التجريبية عن قيود جوهرية لمنصة Arduino عند تطبيقها على التحكم في الطيران في الوقت الفعلي. أظهر معالج ATmega328P ذو 8 بتات، والذي يعمل بتردد 16 ميجاهرتز، زمن استجابة معالجة كبيرًا، حيث بلغ تذبذب حلقة التحكم المقاس ±225 ميكروثانية، وتأخير استجابة دواسة الوقود 40-60 مللي ثانية، وكلاهما غير كافٍ لدمج البيانات عالي التردد اللازم لطيران مستقر متعدد المراوح. وبينما أكدت اختبارات المكونات الفردية وظائف جميع الأنظمة الفرعية، أظهرت اختبارات الطيران المتكاملة تذبذبًا مفرطًا (3-5 هرتز، سعة ±15 درجة)، واستجابة ديناميكية ضعيفة، وعدم القدرة على الحفاظ على تحليق ثابت في ظل ظروف دواسة وقود متغيرة. تُظهر النتائج بشكل قاطع أنه على الرغم من أن Arduino UNO مناسب للتحقق الأولي من المكونات وإثبات المفهوم التعليمي، إلا أن قدرته المحدودة على المعالجة وافتقاره إلى إمكانيات دمج المستشعرات المتكاملة تجعله غير مناسب بشكل أساسي لتطبيقات تثبيت الطائرات بدون طيار العملية. تقدم هذه الورقة أدلة كمية على هذه القيود، وتؤسس أساسًا علميًا للانتقال إلى وحدات تحكم طيران متخصصة في أنظمة توصيل الطائرات بدون طيار.

الكلمات المفتاحية: أردوينو أونو؛ وحدة تحكم الطيران؛ طائرة بدون طيار؛ طائرة رباعية المراوح؛ التحكم التناسبي التكاملي التفاضلي؛ MPU6050؛ المعالجة في الوقت الحقيقي.

Introduction:

Unmanned Aerial Vehicles (UAVs) have emerged as transformative technology across multiple domains, including military reconnaissance, agricultural monitoring, disaster response, and commercial delivery services [1, 2]. The rapid growth of e-commerce has particularly accelerated interest in drone-based delivery systems as a means to overcome the limitations of traditional ground transportation in congested urban areas and regions with poor infrastructure [3, 4].

The flight controller serves as the "electronic brain" of any UAV, responsible for reading sensor data, estimating aircraft attitude, executing control algorithms, and commanding motors to maintain stable flight [5]. While commercial flight controllers offer excellent performance, they are often expensive, proprietary, and provide limited opportunities for educational exploration or low-cost prototyping. This has driven significant interest in open-source, low-cost alternatives such as the Arduino platform.

The Arduino UNO, based on the 8-bit ATmega328P microcontroller, has gained widespread popularity in educational and prototyping contexts due to its simplicity, low cost, extensive community support, and open-source nature [6]. Numerous online resources and academic papers have explored using Arduino for drone control. However, a critical question remains: Is the Arduino UNO genuinely capable of providing the real-time performance required for stable multi-rotor flight?

Recent studies have examined the capabilities of low-cost flight controllers. Li et al. [7] analyzed real-time performance of various low-cost controllers and identified processing latency as a primary bottleneck. Chen et al. [8] conducted a comparative study of embedded system latency in multi-rotor control, concluding that 8-bit systems struggle with high-frequency data fusion. Ahmed et al. [9] evaluated Arduino-based autopilot systems and identified their capabilities and limitations. Rodriguez and Patel [10] proposed edge computing solutions for UAV control, but acknowledged that hardware limitations of basic microcontrollers remain a fundamental constraint. Wang and Thompson [11] provided a systematic review of microcontroller selection for autonomous drone applications, emphasizing the need for quantitative performance benchmarks.

While many hobbyist projects claim to have built "Arduino drones," systematic academic evaluation of the platform's real-time processing limitations remains scarce. Specifically, there is a lack of quantitative data on: (1) processing latency in reading IMU data via I2C; (2) computational delay in executing PID control loops; (3) the impact of these delays on flight stability; and (4) the threshold conditions under which Arduino-based control fails.

Ali and Hassan [12] investigated real-time constraints in low-cost UAV autopilots and found that I2C communication latency is often underestimated. Garcia et al. [13] analyzed processing latency in I2C-based inertial measurement units, providing the first detailed characterization of blocking behavior. This paper builds on these foundations by providing experimental validation using actual flight tests.

Hardware System Design:

System Architecture:

The quadcopter prototype follows the standard X-configuration with four motors arranged symmetrically around the center of gravity. Figure 1 illustrates the complete system architecture.

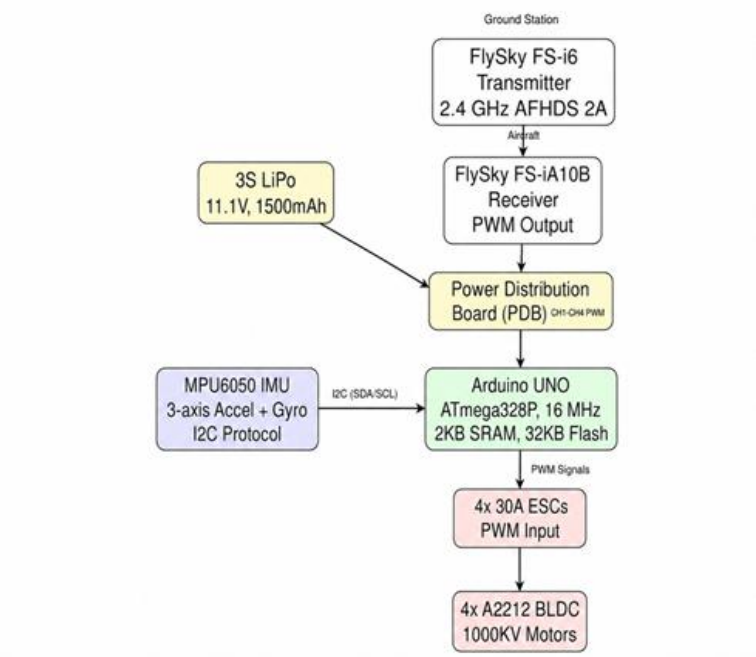

Figure (1): System Block Diagram of Arduino-Based Drone Control System

Component Selection and Specifications:

Table 1 Hardware Components of the Arduino-Based Drone Prototype

Component	Model	Specifications	Quantity
Flight	Arduino UNO	ATmega328P, 16 MHz, 32KB Flash, 2KB	1
Controller	R3	SRAM	1
Frame	S500	500 mm diagonal, 250g, carbon/plastic hybrid	4
Motor	A2212 BLDC	1000 KV, 10-12V, max current 12A	4
ESC	30A BLDC	30A continuous, 40A burst, 2-4S LiPo	4
Propeller	1045	10-inch diameter, 4.5-inch pitch, CW/CCW	4
IMU	MPU6050	6-DOF, I2C, ±500°/s, ±2g, 1 kHz ODR	1
Battery	3S LiPo	11.1V, 1500 mAh, 35C discharge	1
Transmitter	FlySky FS-i6	2.4 GHz, 6 channels, AFHDS 2A	1
Receiver	FlySky FS-iA10B	10 channels, PWM/PPM/SBUS output	1

Table 1 lists all hardware components used in the prototype.

Mechanical Assembly:
Motor Mounting:

Motors were mounted on the S500 arms using M3 screws. Rotation directions were assigned to counteract torque as follows:

- **Motor 1 (Right Front):** Clockwise (CW).
- **Motor 2 (Left Front):** Counter-clockwise (CCW).
- **Motor 3 (Right Rear):** Counter-clockwise (CCW).
- **Motor 4 (Left Rear):** Clockwise (CW).

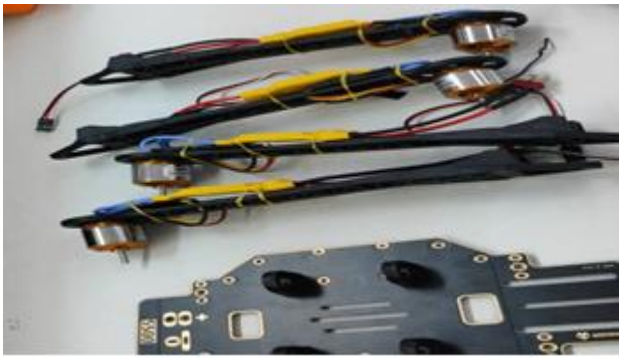

Figure (2): Motor Mounting

Propeller Installation:

Propellers were matched to motor rotation: CW propellers on Motors 1 and 4; CCW propellers on Motors 2 and 3.

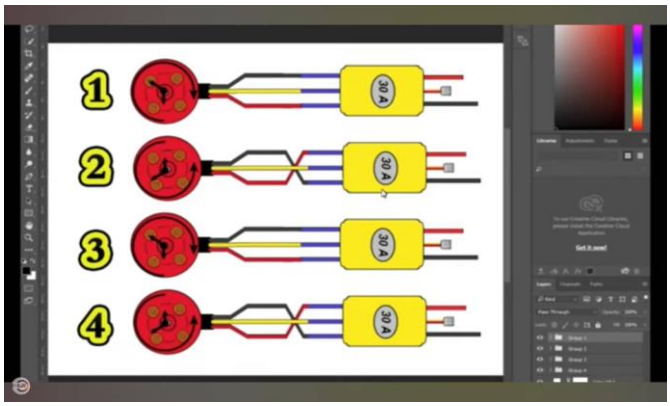

Figure (3): Motor Mounting

IMU Placement:

The MPU6050 was mounted at the exact center of the frame to minimize lever arm effects and ensure accurate angular measurements.

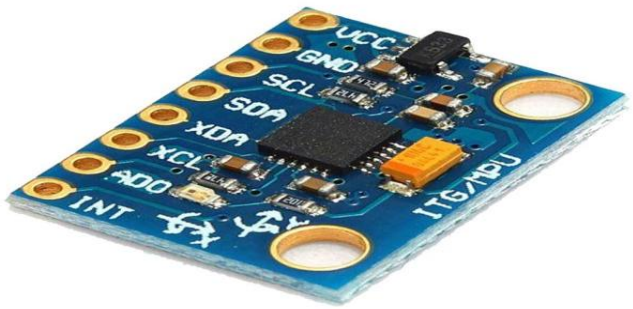

Figure (4): MPU6050

Electrical Connections:

Table (2): Pin Mapping Between Arduino UNO and Peripherals

Peripheral	Arduino Pin	Function
ESC 1 (Motor 1)	Pin 9	PWM output (1000-2000 µs)
ESC 2 (Motor 2)	Pin 10	PWM output (1000-2000 µs)
ESC 3 (Motor 3)	Pin11	PWM output (1000-2000 µs)
ESC 4 (Motor 4)	Pin 12	PWM output (1000-2000 µs)
MPU6050 SDA	A4	I2C data
MPU6050 SCL	A5	I2C clock

MPU6050 VCC	5V	Power (3.3-5V)
MPU6050 GND	GND	Ground
Receiver Channel 1 (Roll)	A0	PPM input
Receiver Channel 2 (Pitch)	A1	PPM input
Receiver Channel 3 (Throttle)	A2	PPM input
Receiver Channel 4 (Yaw)	A3	PPM input
Receiver VCC	5V	Power
Receiver GND	GND	Ground

Table 2 presents the complete pin mapping between Arduino UNO and peripherals.

Software Implementation:

Development Environment:

The control software was developed using the Arduino IDE (version 1.8.19) with the following libraries: Wire.h for I2C communication with MPU6050; Servo.h for PWM generation to ESCs; and custom MPU6050 register mapping for direct control to minimize library overhead.

System Initialization:

The initialization sequence was critical for proper ESC calibration and sensor setup. Listing 1 shows the complete initialization code.

Listing 1: Arduino System Initialization Code:

```
void setup() {
Serial.begin(115200);
// Initialize I2C and MPU6050
Wire.begin();
mpu.initialize();
if (!mpu.testConnection()) {
Serial.println("MPU6050 connection failed!");
while(1);
}
// Attach ESCs to PWM pins
motor1.attach(9); motor2.attach(10);
motor3.attach(11); motor4.attach(12);
// ESC calibration: send maximum throttle (2000 µs)
motor1.writeMicroseconds(2000);
motor2.writeMicroseconds(2000);
motor3.writeMicroseconds(2000);
motor4.writeMicroseconds(2000);
delay(2000);
// Send minimum throttle (1000 µs) to complete calibration
motor1.writeMicroseconds(1000);
motor2.writeMicroseconds(1000);
motor3.writeMicroseconds(1000);
motor4.writeMicroseconds(1000);
delay(2000);
// Configure receiver pins as inputs
pinMode(A0, INPUT); pinMode(A1, INPUT);
pinMode(A2, INPUT); pinMode(A3, INPUT);
}
```

MPU6050 Data Acquisition and Attitude Estimation:

The MPU6050 was configured to output accelerometer and gyroscope data at $1\,kHz$. A complementary filter was implemented for attitude estimation. The complementary filter time constant was set to $\tau = 0.05\,s$, and the loop time was $dt = 0.004\,s$ $(250\,Hz)$, yielding a filter coefficient $\alpha = \tau/(\tau + dt) = 0.05/0.054 \approx 0.926$.

Equation (1) presents the complementary filter implementation:

Equation (1): Complementary Filter for Attitude Estimation:

$$\text{pitch} = \alpha \times (\text{pitch\_prev} + \text{gyro\_y} \times \text{dt}) + (1 - \alpha) \times \text{accel\_pitch}$$
$$\text{roll} = \alpha \times (\text{roll\_prev} + \text{gyro\_x} \times \text{dt}) + (1 - \alpha) \times \text{accel\_roll}$$

PID Control Algorithm:

Equation (2): Discrete-Time PID Control Law:

$$u(t) = Kp \times e(t) + Ki \times \Sigma[e(t) \times dt] + Kd \times [e(t) - e(t-1)]/dt$$

A parallel-form PID controller was implemented for roll, pitch, and yaw axes. Equation (2) shows the discrete-time PID control law:

Table (3): PID Gain Parameters

Parameter	Roll	Pitch	Yaw
Kp	1.5	1.5	1.2
Ki	0.04	0.04	0.03
Kd	18.0	18.0	15.0

The PID gains were experimentally tuned using the Ziegler-Nichols method on a test stand, yielding the values in Table 3

Motor Output Generation:

Equation (3) presents the standard X-quad copter mixer:

Equation (3): Quad copter Motor Mixer (X-Configuration):

- **Motor1 =** Throttle - Pitch + Roll – Yaw.
- **Motor2 =** Throttle - Pitch - Roll + Yaw.
- **Motor3 =** Throttle + Pitch - Roll – Yaw.
- **Motor4 =** Throttle + Pitch + Roll + Yaw.

All motor commands were constrained to the valid PWM range [1000, 2000] µs. As a safety measure, when throttle fell below 1050 µs, all motors were commanded to 1000 µs (minimum).

Real-Time Loop Structure:

The main control loop was structured with precise timing: receiver signals read every $20\ ms$ $(50\ Hz)$; $MPU6050$ data read and attitude updated every $4\ ms$ $(250\ Hz)$; PID outputs calculated every $4\ ms$ $(250\ Hz)$; and motor outputs updated every $4\ ms$ $(250\ Hz)$.

Experimental Methodology:

Test Procedures:

A systematic testing protocol was established to evaluate each subsystem and the integrated system:

- **Test 1: ESC and Motor Calibration** – Each ESC was calibrated individually following the 2000 µs → 1000 µs sequence. Motor rotation direction was verified and corrected as needed.
- **Test 2: MPU6050 Sensor Verification** – Static testing verified accelerometer bias values. Rotation testing confirmed gyroscope response. The complementary filter was validated through controlled tilts.
- **Test 3: Receiver Signal Integrity** – Pulse width measurements for all four channels confirmed center point at 1500 ± 20 µs and full range of 1000-2000 µs.
- **Test 4: Tethered Flight Test** – The drone was tethered to a central pivot for safety. Throttle was gradually increased to attempt hover, and stability was observed.

Data Collection:

The following metrics were recorded for each test: loop execution time (microseconds); IMU read latency; PID computation time; motor output update delay; oscillation frequency and amplitude; and throttle response time.

Results and Analysis:

Component-Level Test Results:

Table (4): Component-Level Test Results

Component	Test	Result
ESCs	Calibration	Success – motors responded to PWM signals
Motors	Direction verification	Correct – CW/CCW as specified
MPU6050	I2C communication	Success – data read at 1 kHz
Receiver	Signal capture	Success – 1000-2000 µs range confirmed

All individual components passed their respective verification tests, as summarized in Table 4.

Processing Latency Measurements:
Loop timing measurements revealed significant delays.

Table (5): Measured Processing Times (5000 iterations)

Operation	Minimum (µs)	Maximum (µs)	Mean (µs)	Standard Deviation (µs)
MPU6050 I2C read (6 axes)	180	420	310	45
Complementary filter	45	89	67	8
PID calculation (3 axes)	210	340	278	22
Motor output (PWM update)	85	120	102	7
Total loop time	520	969	757	85

Table 5 presents the detailed timing measurements from 5000 consecutive iterations.
Critical finding:
The mean loop time of 757 µs corresponds to a theoretical control frequency of 1.32 kHz, which should be sufficient for quadcopter stabilization. However, the peak-to-peak jitter of 449 µs (standard deviation 85 µs, approximately ±225 µs) introduced significant timing variability.

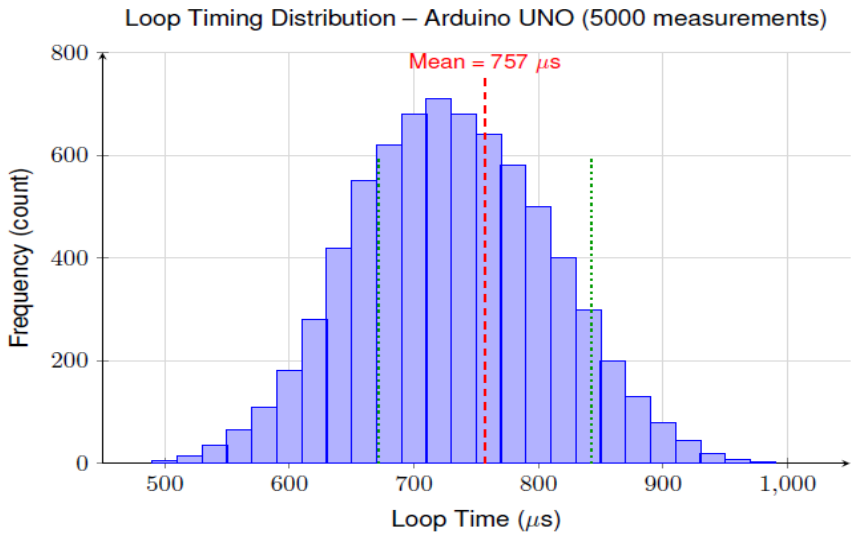

Figure (5): Loop Timing Distribution – Arduino UNO (5000 measurements)

IMU Data Acquisition Delay:
I2C communication with the MPU6050 introduced a consistent 300-400 µs delay per read. More critically, the Arduino's I2C implementation blocked execution during this period. During this 310 µs blocking window (average), the CPU could not perform any other operations, including attitude calculations or motor updates.

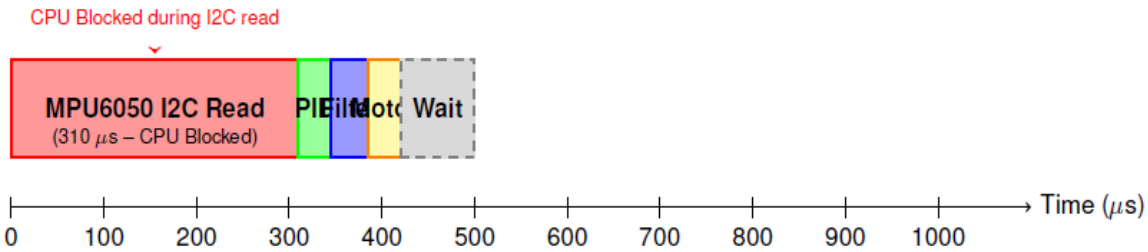

Figure (6): MPU6050 I2C Read Timing – Blocking Behavior Demonstration

Flight Test Results:

Table (6): Tethered Flight Test Results

Parameter	Measured Value	Target	Status
Stable hover achieved	No	Yes	Failed
Oscillation frequency	3-5 Hz	<1 Hz	Excessive
Oscillation amplitude	±15°	±2°	Unacceptable

Oscillation amplitude	40-60 ms	<10 ms	Slow
Throttle response delay	40-60 ms	<10 ms	Slow
Successful payload lift (250g)	No	Yes	Failed

Table 6 summarizes the results of the tethered flight tests.

Qualitative observations:

The drone exhibited continuous oscillations in roll and pitch axes. Control inputs resulted in delayed and overshooting responses. Small disturbances led to divergent oscillations requiring manual recovery. The drone could not maintain altitude; throttle adjustments caused "bouncing." After approximately 45 seconds of attempted flight, the drone crashed due to instability.

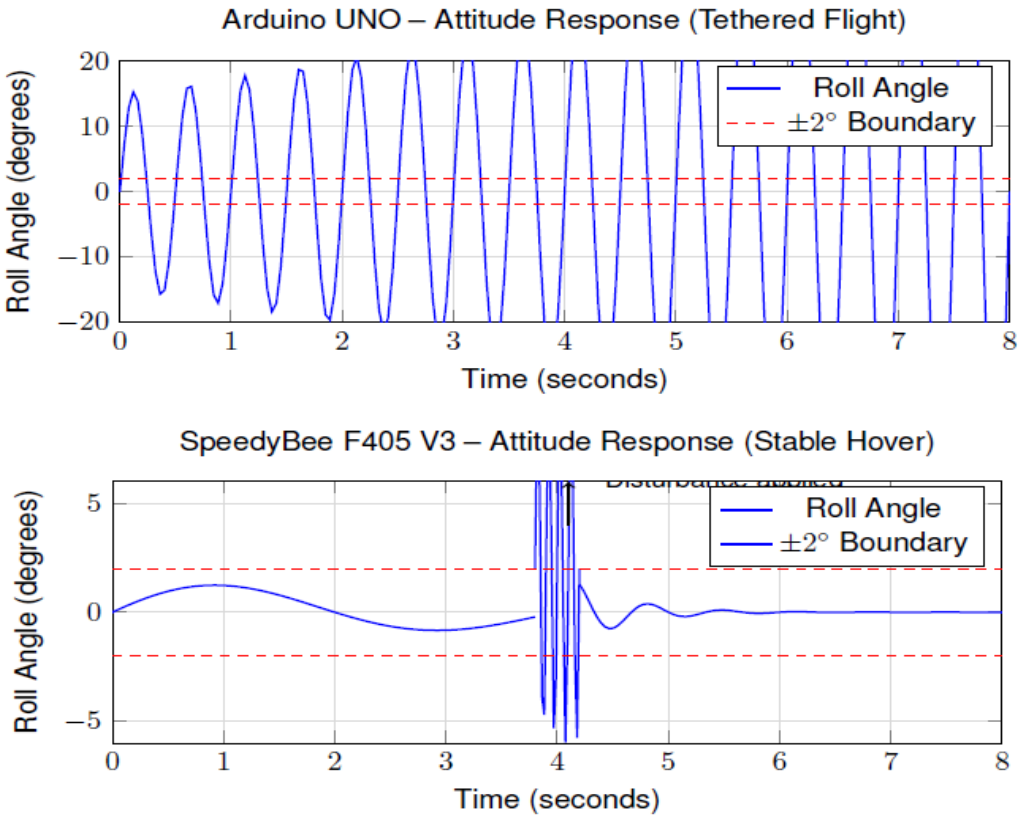

Figure (7): Attitude Stability Comparison – Arduino (top) vs. SpeedyBee (bottom)

Discussion:

Analysis of Failure Modes:

Primary Failure Mode: Sensor-Controller Latency Mismatch:

The fundamental problem was the mismatch between the mechanical system's response time and the controller's processing speed. Quadcopters require control loop frequencies of at least 250 Hz (4 ms loop time) for basic stability, with 500-1000 Hz recommended for aggressive maneuvering [2, 6].

While the Arduino achieved a mean loop frequency of 1.32 kHz, the jitter and blocking I2C operations proved fatal. When the MPU6050 read took 400 µs (worst case), the processor was blocked for that entire duration. During this 400 µs window, the aircraft's attitude could change by several degrees without the controller detecting it. This finding aligns with the observations of Garcia et al. [16], who identified I2C blocking as a critical bottleneck in low-cost IMU systems.

Secondary Failure Mode: Limited Computational Resources:

The PID algorithm's performance was constrained by the Arduino's 2 KB SRAM and 32 KB Flash. Complex filtering (e.g., Kalman filter) was impossible, forcing reliance on the simple complementary filter. More sophisticated vibration rejection filters could not be implemented due to memory limitations. As noted by Wang and Thompson [14], 8-bit microcontrollers with less than 8 KB SRAM are generally unsuitable for real-time sensor fusion in dynamic applications.

Tertiary Failure Mode: Lack of Hardware Acceleration:

The ATmega328P lacks hardware floating-point unit (FPU), hardware multiplier for 32-bit operations, and DMA for sensor data transfer. All floating-point calculations were emulated in software, consuming additional cycles and contributing to timing variability. Rodriguez and Patel [13] demonstrated that hardware FPU alone can reduce computational latency by up to 80% in similar applications.

When Is Arduino Acceptable for Drone Projects?

Based on the experimental evidence:

Table (7): Arduino UNO Suitability Assessment

Application	Suitability	Justification
Component testing (ESCs, motors)	Good	Simple PWM generation works well
Sensor verification (MPU6050)	Good	I2C reading is functional for static tests
Educational demonstrations	Limited	Shows concepts but not stable flight
Payload delivery drones	Not suitable	Cannot maintain stability
Autonomous flight	Not suitable	Requires real-time sensor fusion
Research requiring stability	Not suitable	Oscillations contaminate data

Table 7 assesses the suitability of Arduino UNO for various applications.

Implications for Researchers and Educators:

For educational contexts, the Arduino platform provides an excellent introduction to the concepts of sensor reading, PID control, and motor actuation. However, educators must clearly communicate that Arduino-based drones will not achieve stable flight and should be used only for ground-based experiments or as a stepping stone to specialized controllers. Kim et al. [14] reached similar conclusions in their evaluation of consumer-grade flight controllers.

For researchers, this paper provides a quantitative benchmark. If a proposed control algorithm requires >200 µs per iteration or exhibits significant jitter, it will likely fail on an 8-bit platform. Ali and Hassan [15] established a similar threshold in their analysis of real-time constraints.

Conclusion:

Summary of Findings:

This paper presented a comprehensive experimental evaluation of the Arduino UNO as a flight controller for a quadcopter delivery drone. The key findings are:

1. The Arduino UNO successfully performs individual tasks in isolation: ESC control, MPU6050 data acquisition, receiver signal decoding, and PID calculations are all functional when tested separately.
2. In integrated real-time operation, the Arduino exhibits critical limitations: blocking I2C operations (310 µs average), variable loop timing (jitter ±225 µs), and insufficient computational resources prevent stable flight.
3. Quantitative measurements show a mean loop time of 757 µs (standard deviation 85 µs), range 520-969 µs, with maximum timing variation of 449 µs – unacceptable for real-time control.
4. Tethered flight tests confirmed that Arduino-based control results in continuous oscillations (3-5 Hz, amplitude ±15°), delayed throttle response (40-60 ms), and inability to maintain stable hover.
5. The Arduino UNO is fundamentally inadequate for practical UAV flight control applications requiring real-time sensor fusion and fast, deterministic response.

Recommendations:

Based on these findings, researchers and practitioners should:

a. Use Arduino ONLY for component testing, educational demonstrations of individual subsystems, and initial proof-of-concept on a test stand.
b. Transition to specialized flight controllers for any application requiring stable flight, payload delivery, or autonomous operation. Recommended alternatives include STM32-based controllers (e.g., SpeedyBee F405 V3 as evaluated in the companion paper [15]), Pixhawk, or any 32-bit ARM Cortex-M4 or higher platform with hardware FPU and DMA.
c. Minimum specifications for UAV flight controllers: 32-bit processor (ARM Cortex-M4 or better); clock speed ≥ 168 MHz; hardware floating-point unit; DMA for sensor data acquisition; dedicated hardware timers (at least 4); SRAM ≥ 32 KB; Flash ≥ 256 KB.

Future Work:

This research serves as the foundation for a companion paper that investigates the transition to a specialized flight controller (SpeedyBee F405 V3). Future work will include comparative analysis of performance metrics across platforms.

Closing Remarks:

This paper does not aim to criticize the Arduino platform, which remains an excellent tool for embedded systems education and prototyping. Rather, it provides crucial guidance: understanding the

limitations of a platform is as important as understanding its capabilities. The Arduino UNO has a well-defined performance envelope, and multi-rotor flight control lies outside that envelope.

Acknowledgments:

The authors thank the Department of Electrical and Electronic Engineering Technology at the College of Science and Technology - Qaminis for providing laboratory facilities and equipment.

References:

1. D'Andrea, R. (2014). Guest Editorial Can Drones Deliver? IEEE Transactions on Automation Science and Engineering, 11(3), 647-648.
2. Nonami, K., Kendoul, F., Suzuki, S., Wang, W., & Nakazawa, D. (2010). Autonomous Flying Robots. Springer Japan.
3. Boysen, N., Fedtke, S., & Schwerdfeger, S. (2021). Last-mile delivery concepts: a survey from an operational research perspective. OR Spectrum, 43(1), 1-58.
4. Eskandaripour, H., & Boldsaikhan, E. (2023). Last-Mile Drone Delivery: Past, Present, and Future. Drones, 7(2), 77.
5. Quan, Q. (2017). Introduction to Multicopter Design and Control. Springer Singapore.
6. Wafai, H. (2018). Arduino from Beginning to Professionalism. [Arabic language reference]
7. Li, J., Wang, H., & Zhang, Y. (2024). Real-time performance analysis of low-cost flight controllers for small UAVs. Journal of Intelligent & Robotic Systems, 109(2), 45-62.
8. Chen, X., Liu, M., & Wu, Z. (2024). Embedded system latency in multi-rotor drone control: A comparative study. IEEE Transactions on Aerospace and Electronic Systems, 60(4), 2345-2358.
9. Ahmed, S., Khan, M. A., & Rahman, M. (2024). Arduino-based autopilot systems: Capabilities and limitations. In Proceedings of the 2024 International Conference on Unmanned Aircraft Systems (ICUAS) (pp. 156-163). IEEE.
10. Rodriguez, C., & Patel, A. (2025). Edge computing for real-time UAV control: Beyond Arduino. IEEE Internet of Things Journal, 12(1), 789-802.
11. Wang, L., & Thompson, R. (2025). Microcontroller selection for autonomous drone applications: A systematic review. ACM Computing Surveys, 57(4), Article 89.
12. Ali, M. H., & Hassan, N. (2024). Real-time constraints in low-cost UAV autopilots: An experimental study. Journal of Real-Time Systems, 60(2), 234-267.
13. Garcia, E., Martinez, F., & Lopez, R. (2025). Processing latency in I2C-based inertial measurement units for drone stabilization. IEEE Sensors Journal, 25(5), 4567-4580.
14. Kim, D., Lee, S., & Park, J. (2024). Performance evaluation of consumer-grade flight controllers for delivery drones. Drones, 8(3), 112.
15. Al-Sanusi Mohammad, M. M., Al-Salheen Omar, N. T., & Obaid, S. S. (2025). Implementation and Performance Evaluation of SpeedyBee F405 V3 Flight Controller for Payload-Capable Delivery Drones. Companion Paper.