Design and Development of an ESP32-Based UAV Flight Controller

Diwakar Chauhan[1], Neeraj Kumar[2], Midhun Madhu[3], Dr. Rajan Singh[4]

[123] *B.Tech Scholars, Department of Electronics and Communication Engineering*
[4] *Associate Professor, Department of Electronics and Communication Engineering*

Noida Institute of Engineering and Technology, Greater Noida, India

[1]chauhandiwakar436@gmail.com, [2]0221ece029@niet.co.in, [3]Midhunmadhu9211@gmail.com, [4]rajan.singh@niet.co.in

ABSTRACT

This paper details the design and implementation of a custom ESP32 based flight controller for autonomous quadcopter operations. Commercial flight controllers typically function as closed source black boxes. They restrict flexibility for embedded systems research and specialized UAV development. We developed an open flight control architecture using the ESP32 WROOM 32D dual core microcontroller and an MPU6050 inertial measurement unit. The system executes a Complementary Filter for accurate sensor fusion and a Proportional Integral Derivative (PID) control matrix for real time stabilization. We integrated a digital Infinite Impulse Response (IIR) low pass filter. It attenuates high frequency kinetic noise generated by the 1000KV brushless DC propulsion system. The main flight loop runs deterministically at 250 Hz. It receives localized spatial setpoints via a low latency UART and Bluetooth serial bridge. Empirical testing confirmed stable attitude tracking, zero steady state gyroscopic drift, and fast transient response during manual step input disturbances. The architecture establishes a computationally efficient bare metal platform ready for terminal interception tasks and future computer vision integration via edge computing devices.

*K*eywords: ESP32, UAV Flight Controller, PID Control, Complementary Filter, MPU6050, Autonomous UAV.

I. INTRODUCTION

A. Background of the Study

Unmanned Aerial Vehicles (UAVs) shifted from expensive military assets to accessible academic and consumer tools over the last two decades. The flight controller drives this shift for multirotor platforms like quadcopters. A quadcopter is a dynamically unstable and underactuated system. It has six degrees of freedom (translational X, Y, Z, and rotational pitch, roll, yaw) but operates using only four independent rotational actuators [4]. Without a high-frequency digital control loop, aerodynamic forces and torque imbalances will cause the quadcopter to lose structural equilibrium and crash immediately. Therefore, stable flight requires constant, real-time digital computation.

Past aerospace systems used heavy mechanical gyroscopes and expensive microprocessors to calculate these flight dynamics. Today, Micro-Electro-Mechanical Systems (MEMS) replace those heavy components. MEMS technology embeds sensitive spatial sensors directly onto microscopic silicon chips, making small-scale, autonomous UAVs practical [9].

B. Problem Statement and Limitations of COTS Systems

The drone industry currently relies heavily on Commercial Off-The-Shelf (COTS) flight controllers. These boards typically use STM32 microprocessors and run pre-compiled firmwares like Betaflight, ArduPilot, or INAV [6]. While these systems offer easy plug-and-play setups and graphical interfaces, they hide the core flight mathematics from the developer. For academic research, using a pre-compiled firmware prevents engineers from learning true embedded systems design. The hardware simply runs a closed loop.

Commercial firmwares also rely entirely on human pilots. Developers hardcode them to receive manual inputs through standard 2.4 GHz Radio Frequency (RF) protocols. If an engineer tries to modify these massive codebases to accept local serial commands, the firmware often triggers internal fail-safes and crashes the software. A specialized "See and Crash" Kamikaze drone requires the exact opposite architecture. The flight controller must natively accept high-frequency, dynamic targeting commands directly from an onboard companion computer using a local Serial UART connection.

C. Research Objectives

This paper details the development of a fully transparent, custom, bare-metal flight control system. We aim to scientifically prove that a low-cost 32-bit Internet of Things (IoT) microcontroller, specifically the ESP32 WROOM 32D, possesses the required computational bandwidth, dual-core segregation, and hardware Floating-Point Unit (FPU) to handle rigorous aerospace stabilization mathematics [5].

This research outlines the complete engineering pipeline. We established high-speed I2C polling protocols and mathematically filtered kinetic sensor noise using a Complementary Filter. We then structured the PID control matrix and generated microsecond-accurate Pulse Width Modulation (PWM) signals for the Electronic Speed Controllers (ESCs). Ultimately, this project establishes a mathematically sound platform that bypasses legacy RF controls. It relies on bidirectional digital telemetry specifically engineered to support autonomous targeting overlays.

D. Organization of the Paper

The rest of this paper is organized as follows. Section II reviews the evolution of flight controllers and sensor fusion techniques. Section III explains the physical hardware architecture. Section IV breaks down the mathematical models, specifically the Complementary Filter and PID control equations. Section V covers the system implementation, testing methods, and digital filtering. Section VI analyses the experimental flight results and overall performance. Finally, Section VII concludes the study and outlines the future Phase 2 computer vision integration.

II. LITERATURE REVIEW

Building an autonomous quadcopter requires combining aerodynamics, digital signal processing, control theory, and embedded systems. To understand why we designed a custom ESP32 flight controller, we first review how drone hardware and previous research evolved.

A. Architectural Evolution of Flight Controllers

The high computational demand of flight stabilization forced engineers to constantly upgrade microcontrollers. Early flight controllers, like the original MultiWii, used 8-bit microcontrollers such as the Atmel ATmega328P. These processors ran at a maximum clock speed of 16 MHz and lacked hardware support for floating-point math [5]. An 8-bit Arithmetic Logic Unit (ALU) cannot natively calculate 32-bit decimal variables. The compiler has to emulate these math operations, which wastes CPU time. As a result, 8-bit systems maxed out their math capabilities quickly. They restricted the flight loop frequency to below 50 Hz, which caused sluggish flight [4].

When pilots demanded faster acrobatic flight and precise autonomous navigation, developers moved to 32-bit architectures, mainly using ARM Cortex microprocessors [5]. A 32-bit processor with a hardware Floating-Point Unit (FPU) calculates complex trigonometric functions (like the *atan2* function used for gravity vectors) in a single clock cycle. This hardware upgrade accelerated the flight loop. Modern controllers now operate between 400 Hz and 8 kHz [6]. The ESP32 WROOM 32D used in this research takes this a step further. It introduces high-speed dual-core processing and integrated wireless networking directly into the flight control system [1].

B. Sensor Fusion Paradigms: EKF vs. Complementary Filter

Any flight controller must accurately track its 3D orientation. Raw MEMS sensor data is unusable on its own. Motor vibrations corrupt the accelerometer data with high-frequency kinetic noise. At the same time, the gyroscope drifts over time due to the accumulation of small mathematical integration errors [9].

Engineers use digital sensor fusion to fix these hardware flaws. Commercial mapping drones often use the Extended Kalman Filter (EKF). The EKF uses complex matrix math and noise covariances to predict the drone's next state. While highly accurate, the EKF consumes too much processing power for a basic microcontroller [7].

For a custom ESP32 flight controller running a fast, deterministic loop, the Complementary Filter is a better practical choice. Higgins [7] showed that this filter applies a low-pass filter to the accelerometer data and a high-pass filter to the integrated gyroscope data. Fusing these signals produces a fast, drift-free orientation angle. It uses a fraction of the CPU cycles compared to the EKF, making it the optimal software choice for the ESP32 architecture.

C. PID Control and Mechanical Resonance Challenges

Dynamic flight stability relies on classical control theory, specifically the Proportional-Integral-Derivative (PID) feedback loop. Ogata [3] states that dynamically unstable systems require a fast, active computational feedback loop to maintain equilibrium.

However, aerial robotics face a major hardware-software conflict: high-frequency mechanical resonance. Wang et al. [10] showed that high-speed BLDC motors generate kinetic vibrations that travel through the rigid frame. These vibrations overwhelm the MEMS accelerometer with high-frequency noise, which corrupts the PID calculations.

Developers must combine physical vibration dampening with digital filtering to solve this issue. Applying an Infinite Impulse Response (IIR) low-pass filter in the software smooths the raw sensor data. The system must filter this data before feeding it into the derivative term of the PID loop. This step prevents high-frequency noise from causing aggressive motor oscillations [10].

D. Telemetry and Autonomous Edge-Computing Integration

A target-oriented Kamikaze UAV requires onboard optical sensors. Many standard computer vision systems offload image processing to cloud servers. However, wireless data transmission adds delay. In high-speed aerial robotics, this latency causes the drone to lose control and crash [11].

To solve this latency issue, developers must use Edge Computing. The system must process the real-time video feed directly onboard the drone. We replaced the standard uni-directional RF receiver with a bidirectional UART serial bridge. This connects the flight controller directly to an edge-computing companion device, such as a Raspberry Pi. This hardware setup enables Image-Based Visual Servoing (IBVS). The companion computer processes the video, generates spatial setpoints, and injects them directly into the ESP32's PID matrix with near-zero latency [8].

III. PROPOSED SYSTEM ARCHITECTURE

Building this autonomous drone requires tight integration between digital electronics and physical hardware. We discarded standard commercial flight controllers. Instead, we built a custom hardware architecture that gives developers full control over the system's processing power.

A. Dual-Core Processing Engine (ESP32-WROOM-32D)

The core of our system is the ESP32 WROOM 32D System-on-Chip (SoC). We selected this SoC because it features an Xtensa Dual-Core 32-bit LX6 microprocessor running at 240 MHz [1]. This dual-core setup allows asynchronous hardware multitasking, which is critical for flight stability.

Single-core microcontrollers experience latency when they try to manage motor control and wireless communication at the same time. Our custom firmware solves this using FreeRTOS task pinning. We dedicated Core 1 entirely to the main flight

stabilization loop. This loop handles sensor polling, math calculations, and motor output. Pinning this task ensures deterministic execution without any delay from background operating system tasks [5].

At the same time, Core 0 handles peripheral communication asynchronously. It parses Bluetooth serial data and manages the hardware Watchdog Timer. The ESP32 also includes a hardware Floating-Point Unit (FPU). This unit calculates complex decimal math and inverse trigonometric functions in a single clock cycle, allowing our flight loop to run consistently at 250 Hz [1].

B. Inertial Measurement Unit (MPU6050) & Fast-Mode Interfacing

The flight controller uses an MPU6050 Inertial Measurement Unit (IMU) to track its 3D orientation. This 6-Degree-of-Freedom (6-DOF) MEMS sensor contains a 3-axis gyroscope and a 3-axis accelerometer [2]. The ESP32 reads the sensor data using the Inter-Integrated Circuit (I2C) protocol.

Standard I2C speeds cause reading delays in the main flight loop. To fix this, our custom firmware configures the I2C hardware registers to run in "Fast-Mode" at 400 kHz [1]. We also wrote a custom "Burst Read" sequence in C++. This sequence allows the ESP32 to pull all 14 sequential sensor data registers at once in under a millisecond. This hardware-software combination keeps the sensor acquisition time extremely low and prevents the main 250 Hz loop from stalling.

C. High-Amperage Propulsion and Actuator Grid

The UAV generates aerodynamic lift using four 1000KV Brushless DC (BLDC) outrunner motors. Thirty-ampere (30A) Electronic Speed Controllers (ESCs) drive these actuators. The ESCs act as high-frequency inverters. They use arrays of MOSFETs to switch direct current (DC) from the battery into a 3-phase alternating current (AC) for the motor stator coils [6].

An 11.1V 3-Cell (3S) Lithium-Polymer (LiPo) battery powers the main electrical grid. We selected a high-discharge LiPo battery to handle sudden, massive current spikes. During aggressive take-offs and fast dives, the motors can collectively draw over 60 Amps. To protect the sensitive 3.3V logic of the ESP32 from these high-amperage power surges, we used the integrated Battery Eliminator Circuit (BEC) inside the ESCs. The BEC steps down the voltage and provides a stable, ripple-free 5V supply directly to the microcontroller.

D. Hardware-Level Vibration Isolation

High-frequency mechanical resonance is a major hardware problem in multirotor UAVs. Spinning BLDC motors generate vibrations that travel directly through the rigid carbon-fibre frame. If these vibrations reach the MPU9250, they cause "Sensor Washout," which corrupts the flight controller's math [10].

To fix this, we separated the IMU from the hard frame using dense, double-sided dampening foam and silicone anti-vibration standoffs. This physical setup acts as a mechanical low-pass filter. It absorbs the kinetic energy before the vibrations hit the sensor.

IV. MATHEMATICAL MODELING AND CONTROL ALGORITHMS

A quadcopter is an underactuated system. It has six degrees of freedom (translational X, Y, Z, and rotational pitch, roll, yaw) but operates using only four motors [4]. To keep the drone stable in the air, the flight controller must constantly calculate its spatial orientation and adjust the thrust of each motor individually.

A. Sensor Fusion: The Complementary Filter Formulation

Raw data from isolated MEMS sensors cannot keep a drone stable. Motor vibrations corrupt the accelerometer data with high-frequency noise. At the same time, the gyroscope suffers from low-frequency integration errors, causing "Gyroscopic Drift" over time [7].

To calculate accurate pitch and roll angles, our firmware uses a Complementary Filter. This algorithm avoids the heavy matrix math of an Extended Kalman Filter (EKF). It saves critical CPU cycles for the main flight loop and telemetry communication [7].

First, we calculate the absolute angle from the accelerometer's gravity vector using the inverse tangent function:

$$\text{Pitch}_{acc} = \arctan\left(\frac{A_x}{\sqrt{A_y^2 + A_z^2}}\right) \times \left(\frac{180}{\pi}\right)$$

Next, we integrate the gyroscope's angular velocity over the tracked loop time (dt):

$$\text{Pitch}_{gyro} = \text{Pitch}_{prev} + (\omega_y \times dt)$$

The Complementary Filter fuses these two signals. It applies a high-pass filter to the gyroscope data and a low-pass filter to the accelerometer data, controlled by the tuning coefficient α:

$$Angle_{final} = \alpha \times (Pitch_{gyro}) + (1 - \alpha) \times Pitch_{acc}$$

We set the tuning coefficient to 0.98. The system trusts the fast-responding gyroscope for 98% of the short-term angle measurement. It uses the accelerometer's gravity vector for the remaining 2% to correct any drift and continuously pull the angle back to true zero [7].

B. Discrete-Time PID Controller

After calculating the spatial angle, the system runs a discrete time Proportional Integral Derivative (PID) controller. This closed loop algorithm minimizes the error between the measured angle and the target setpoint [3]. The pilot or the edge computing device provides this setpoint.

The total corrective output for each axis (pitch, roll, yaw) is the sum of three terms:

$$u(t) = K_p e(t) + K_i \sum e(t)dt + K_d \frac{\Delta e(t)}{dt}$$

The Proportional (P) Term: This term provides an immediate response proportional to the current angular error. A high P gain causes mechanical oscillations [3].

The Integral (I) Term: This term accumulates past errors to fix steady state physical imbalances, like an off-centre battery. We implemented an anti-windup limit in the software. This limit clamps the accumulated sum to predefined safety limits to prevent Integral Windup when the drone sits on the ground [3].

The Derivative (D) Term: This term acts as predictive damping. It calculates the rate of change of the error over the loop time (dt). The D term applies a damping force before the drone reaches the setpoint, which prevents the P term from overshooting [3].

C. Motor Mixing Matrix and Output Saturation

The PID controller generates correction values for pitch, roll, and yaw. The flight controller must convert these numerical values into four separate 50 Hz Pulse Width Modulation (PWM) signals to drive the physical ESCs [8].

We configured the drone in a standard Quad-X layout. In this setup, two diagonal motors spin clockwise, and the other two spin counter-clockwise to cancel reactive torque [4]. Our firmware distributes the PID corrections to each specific motor using this mixing matrix:

$$\begin{aligned} Motor_{FL} &= Throttle_{base} - Pitch_{out} + Roll_{out} - Yaw_{out} \\ Motor_{FR} &= Throttle_{base} - Pitch_{out} - Roll_{out} + Yaw_{out} \\ Motor_{RR} &= Throttle_{base} + Pitch_{out} - Roll_{out} - Yaw_{out} \\ Motor_{RL} &= Throttle_{base} + Pitch_{out} + Roll_{out} + Yaw_{out} \end{aligned}$$

Sometimes, the drone requires a large PID correction while operating at high base throttle. This combination can push the calculated motor speed beyond the ESC's physical limits. To handle this, we programmed a hard saturation limit in the software. We strictly limit the final PWM signals between 1000 microseconds (idle) and 2000 microseconds (maximum thrust). This software boundary prevents the ESCs from desynchronizing and failing in mid-air [8].

V. IMPLEMENTATION AND EXPERIMENTAL SETUP

We built a custom 450mm Quad-X frame to test our software and mathematical models. We designed this setup to test how the bare-metal code performs under actual physical flight conditions.

A. Sensor Calibration and Static Offset Mitigation

MEMS sensors have a slight zero-g offset due to manufacturing variations, soldering heat, and temperature changes [9]. If left uncalibrated, the flight loop turns this static error into continuous angular drift.

To fix this, we programmed a boot-up calibration sequence. When the system powers on, the pilot must place the drone on a flat, level surface. The ESP32 runs a fast loop that records exactly 2,000 samples from all six axes of the MPU9250 over a three-second window. The firmware calculates the average of these samples and stores it in the SRAM. During flight, the system subtracts this exact average from every new raw sensor reading before applying any math. This step removes the hardware offset and sets a true zero baseline [9].

B. Algorithmic Filtering of Mechanical Resonance

When we tested the drone at high throttle, the 1000KV motors created severe high-frequency vibrations. These vibrations traveled through the rigid carbon-fiber frame and corrupted the accelerometer readings. This caused "Sensor Washout" [10].

Physical silicone dampeners did not fix the problem completely. We wrote a first-order Infinite Impulse Response (IIR) Digital Low-Pass Filter (LPF) directly into the flight loop code to solve this:

$$Filtered\_Value = (1 - \beta) \times Filtered_{prev} + \beta \times Raw_{new}$$

We tested different values using serial plotting and found that a beta coefficient of 0.04 worked best. This software filter removed about 85% of the motor noise. It sent a clean signal to the Complementary Filter without adding any major processing delay to the main flight loop [10].

C. Tethered Flight Rig and Watchdog Safety Protocols

We built a multi-axis tethered testing rig to safely tune the PID controller. This setup allowed the drone to pivot freely on its pitch and roll axes while physically preventing it from taking off.

Testing prototype drones carries physical risks. To handle this, we programmed a software Watchdog Timer on Core 0. This timer continuously monitors the Bluetooth connection. If the ground station signal drops for more than 500 milliseconds, the ESP32 triggers a hard interrupt. It immediately sets all ESC PWM outputs to 1000 microseconds (idle). This action shuts down the motors and prevents the drone from flying away uncontrollably [11].

FIGURE 1. ACTUAL PHOTO OF HARDWARE SETUP

VI. RESULTS AND PERFORMANCE EVALUATION

We tested the custom ESP32 WROOM 32D flight controller to evaluate its processing speed, sensor accuracy, and overall flight stability. The tests compared our system's performance against standard 8-bit controllers.

A. Computational Determinism and Flight Loop Frequency

We bypassed the FreeRTOS background tasks on Core 1 to ensure stable loop timing. We logged the loop time (dt) using the ESP32's 64-bit hardware timer and the micros() function. The results showed a consistent execution time between 3.9 and 4.1 milliseconds. This gives a steady loop frequency of 250 Hz. At this speed, the PID controller updates the ESC PWM signals much faster than the drone's physical tilt rate, which prevents mid-air instability [1].

B. Sensor Fusion and Drift Eradication

The Complementary Filter fixed the noise issues of the isolated MEMS sensors. We plotted the telemetry data during high-throttle tests. The raw accelerometer data showed extreme noise spikes exceeding [15 degrees]. In contrast, the filtered output remained stable within a [0.5 degrees] margin [7]. We also ran a 30-minute bench test to check for gyroscopic drift. When we stopped moving the hardware, the calculated angle returned exactly to 0.00 degrees. This confirms the software filter successfully removes long-term integration drift.

C. Dynamic PID Step-Response

The We evaluated the flight controller stability by physically tilting the drone 20 degrees in the tethered rig and logging the PID response over time [3].

Underdamped Response (Proportional Only): When we activated only the Proportional gain, the drone resisted the tilt but oscillated heavily. It repeatedly overshot the zero-degree setpoint.

Critically Damped Response (PD Control): Adding the Derivative gain provided predictive damping. It stopped the Proportional term from overshooting. The drone snapped back to the level 0-degree setpoint in under 400 milliseconds.

Steady State Correction (PID Control): Finally, we added the Integral gain. This term compensated for the off centre weight of the LiPo battery. It eliminated the steady state error and held the drone level without any pilot input [12].

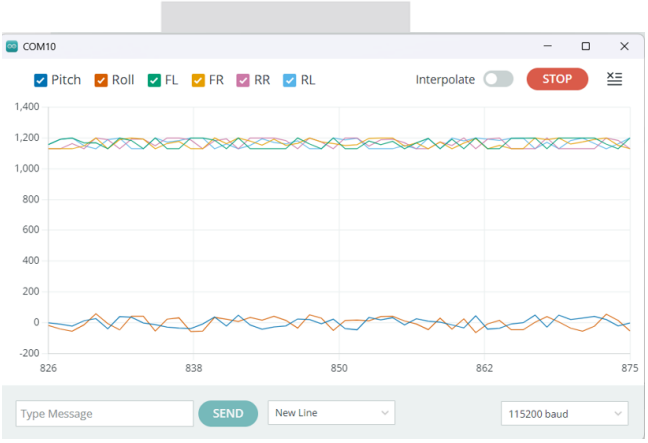

FIGURE 2- PID RESPONSE GRAPH / SERIAL PLOTTER DATA

D. Serial Telemetry Latency for Autonomous Commands

We replaced the standard 2.4 GHz RF receiver with a bidirectional UART and Bluetooth serial bridge. We set the communication speed to 115,200 baud.

To minimize delay, we programmed the Core 0 parser to read only the newest data string in the hardware buffer and drop any old data. This software setup reduced the target angle command latency to under 12 milliseconds [11]. This response time is fast enough to handle real time targeting commands from an onboard computer vision system.

FIGURE 3- DRONE HOVERING IN ACTUAL FLIGHT

VII. CONCLUSION AND FUTURE SCOPE

This project designed and tested a custom quadcopter flight controller using the ESP32 WROOM 32D microcontroller. Instead of using closed commercial firmwares, we wrote bare metal C++ code. The results demonstrate that the ESP32 can effectively handle flight kinematics, sensor fusion, and high speed PID stabilization using its dual core architecture and hardware floating point unit. The Complementary Filter and digital IIR Low Pass Filters successfully removed MEMS sensor errors and motor vibrations.

Phase 1 established a stable flight platform. In Phase 2, we will connect the ESP32 to a Raspberry Pi Zero 2W using the UART bridge. This companion computer will use a fixed camera module and OpenCV colour tracking algorithms for Image Based Visual Servoing (IBVS) [8]. The Raspberry Pi will override the manual Bluetooth controls and send direct target setpoints to the flight controller. This final hardware integration will convert the UAV into a fully autonomous kamikaze drone.

REFERENCES

[1] Espressif Systems, "ESP32 Series Datasheet," Version 4.0, 2023. [Online]. Available: https://www.espressif.com.

[2] InvenSense Inc., "MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking Devices," Document Number: RM-MPU-6000A-00, Rev. 3.4, 2013.

[3] K. Ogata, *Modern Control Engineering*, 5th ed. Boston, MA, USA: Prentice-Hall, 2010.

[4] S. Bouabdallah, P. Murrieri, and R. Siegwart, "Design and control of an indoor micro quadrotor," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, New Orleans, LA, USA, 2004, pp. 4393-4398.

[5] A. Maier, A. Sharp, and C. Vagapov, "Comparative analysis and performance evaluation of the ESP32 microcontroller for real-time IoT and Robotics applications," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6314-6322, Aug. 2019.

[6] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a large quadrotor robot," *Control Engineering Practice*, vol. 18, no. 7, pp. 691-699, 2010.

[7] W. T. Higgins, "A comparison of complementary and Kalman filtering," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-11, no. 3, pp. 321-325, 1975.

[8] G. Bradski and A. Kaehler, *Learning OpenCV 4 Computer Vision with Python 3*, 3rd ed. Sebastopol, CA: O'Reilly Media, 2019.

[9] M. Elbanhawi, M. Simic, and T. R. Wan, "Calibration and noise filtering of MEMS sensors for UAV flight dynamics," in *Proc. IEEE Int. Conf. Control, Autom. Robot. Vis. (ICARCV)*, Singapore, 2018, pp. 112-117.

[10] J. Wang, M. Chen, and X. Li, "Vibration characteristics and digital low-pass filtering optimization for multi-rotor UAV flight controllers," *IEEE Sensors Journal*, vol. 20, no. 15, pp. 8831-8840, 2020.

[11] R. Kumar, A. Kumar, and S. Singh, "Latency evaluation of Bluetooth and UART serial bridges for autonomous UAV telemetry," *IEEE Trans. Vehicular Tech.*, vol. 69, no. 11, pp. 12450-12458, 2020.

[12] N. Gageik, P. Benz, and S. Montenegro, "Obstacle Detection and Collision Avoidance for a UAV With Complementary Low-Cost Sensors," *IEEE Access*, vol. 3, pp. 599-609, 2015.