IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 8 December 2022, accepted 26 December 2022, date of publication 27 December 2022,
date of current version 30 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3232749

APPLIED RESEARCH

Hardware and RTOS Design of a Flight Controller for Professional Applications

RAMÓN RICO, JAVIER RICO-AZAGRA, AND MONTSERRAT GIL-MARTÍNEZ
Control Engineering Research Group, University of La Rioja, 26004 Logroño, Spain

Corresponding author: Ramón Rico (javier.rico@unirioja.es)

This work was supported in part by the Economic Development Agency for La Rioja (ADER) under Grant 2017-I-IDD-00035, and in part by the University of La Rioja under Grant REGI 22/40.

ABSTRACT Unmanned aerial vehicles (UAV) in the civil sector have recently evolved from being devices for recreational uses to becoming aircraft for professional uses. Professional applications of these devices require the aircraft to ship more and more complex sensors for reasons of safety in the event of failures. However, controlling all these systems is a challenge for flight controllers at the hardware and software levels. In this context, this work proposes a new aircraft real-time flight control system. The flight controller hardware is made up of two systems: a core board houses common sensor and flight devices; a flexible unit, isolated from body vibrations, houses redundant sensors to improve accuracy and reliability. The application functions are driven by the microcontroller running a new real-time operating system (RTOS) to better schedule works on limited computation power. To design a simplified RTOS for embedded systems, a hybrid scheduler (first-come-first-serve scheduling and earliest-deadline-first scheduling) with dynamic priority is proposed. The results obtained show the effectiveness of the system using schedulability tests for uni-processor systems. A set of real data supports the performance of the flight controller.

INDEX TERMS Flight controller, real-time operating system, unmanned aerial vehicles.

I. INTRODUCTION

Multi-rotor type unmanned aerial vehicles (UAVs) have increased in popularity in recent years, given their ability to perform vertical take-offs and landings, their ability to hold a fixed position and their ease of use. From the first devices created in the first decade of the 21st century to the present, there has been an important development in the flight capabilities of this type of aircraft, together with a reduction in costs. This evolution has led to its use in a multitude of applications. However, there is still a very important gap between the two types of users and products. On the one hand, devices intended for the general public have seen their price reduced, which has often been driven by the development of hardware and open software, increasing competitiveness in the sector and leading to price reductions. On the other hand, in the professional sectors, very expensive equipment continues to be used, which makes access difficult for sporadic users of this equipment.

The associate editor coordinating the review of this manuscript and approving it for publication was Yang Tang.

The scientific community has not been oblivious to the rise of these devices. Since the creation of the first UAV, numerous scientific papers have been published regularly. Some propose new strategies to improve the capabilities of UAVs, focusing either on state estimation [1], [2], or attitude control [3], [4], [5], or autonomous navigation and guidance [6], [7], [8], [9]. Others present UAV applications [10], [11], highlighting new fields as in [12].

The success of previous developments in real UAVs depends on implementing new software layers capable of working with the flight controller and with the RTOS (Real-Time Operating System) that manages it. Since the development of new hardware devices is very expensive, off-the-shelf flight controllers such as Pixhawk or Cuav are traditionally used. The problem with them is that they are generic and are not specifically designed for the control of an aircraft. Furthermore, many RTOS are Linux-based, making the software too complex for UAV developers. Under the fact that new developments are usually implemented on existing platforms, few works in the scientific literature focus on the optimization of RTOS for UAVs [13], [14], [15], [16],

[17], [18]. However, the importance of the RTOS in the operation of aircraft is maximum. It ensures that the planned tasks are executed properly, on time, and following the established order of priority. Consequently, the RTOS performance and time optimization can lead to important advantages, such as a higher refresh rate in the control loops and the ability to perform a greater number of tasks on the same hardware.

In this context, this work develops the software and hardware of a flight controller intended for use in professional UAVs. This new development aims to be a cost-effective solution closer to all types of users. The philosophy used in the design of the hardware and software aims to obtain a robust and simple system, in which ease of use and simplicity prevail. The developed hardware uses an ARM M7 microcontroller together with a set of sensors adapted to the control of UAVs. A modular architecture separates part of the sensors from the control system; this feature gives the design versatility and redundancy. The main innovation of the new hardware-software set lies in the development of a hard real-time operating system, easy to implement, flexible, and with a level of optimization that allows it to be implemented in any current embedded system. Three software layers provide the typical functionalities: the upper one executes the flight control, the middle one provides connectivity and the deeper one with a RTOS supports the global system. The new RTOS uses a hybrid Scheduler that combines FCFS (First Come First Serve) scheduling and EDF (Earliest Deadline First) scheduling. This hybridization ensures that critical tasks, such as aircraft stabilization control, are not interrupted, i.e. FCFS subsystem guarantees low latency. This fact gives robustness to the system by prioritizing the integrity of the equipment. The EDF subsystem is in charge of the rest of the tasks, which make up the application layer and are executed according to their priority.

In summary, the main contributions of the present work are: (1) a novel flight controller hardware, competitive with the best in the market, which additionally offers redundancy at the sensor level and reduction of sensor noise by mechanical damping; (2) a new hard real-time operating system for embedded systems that implements a hybrid scheduler to better manage tasks of different priority with limited time and computing power; (3) the entire hardware-software set is conceived for professional aircraft control.

The paper is organized in five sections. The second section shows the hardware architecture that is used to develop the flight controller. The third section presents the software architecture that is used to develop the RTOS. Next, the fourth section corroborates the good work of the new proposal, including results that are obtained in real tests. Finally, the fifth section collects the conclusions.

II. HARDWARE ARCHITECTURE

The hardware design is crucial in a professional flight control system. Figure 1 shows the developed system called URpilot. This device has been fully developed to be used in multi-rotor UAVs keeping in mind previous works [19], [20], [21], [22], [23], [24], although it can be used in other devices.

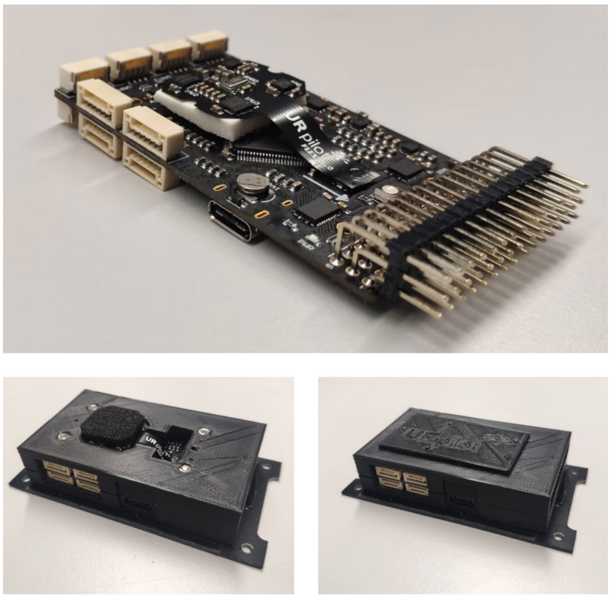


FIGURE 1. Case and anti-vibration system.

The flight controller consists of two functional units interconnected to work together. The main unit, also known as *core*, handles data processing and houses a set of common sensors. The sensor unit, also known as *flex* integrates redundant sensors to improve UAV status estimation. The use of two separated boards provides a mechanical decoupling to the redundant sensor unit, so that the vibrations generated during operation of the UAV do not affect the measurements of the sensors on the flexible board. Additionally, the casing shown in Figure 1 allows the two units to be fixed in their proper position without using ties between them.

Table 1 shows a comparison of several flight controller platforms in terms of the microcontroller unit (MCU), interfaces, built-in sensors, dimensions, and weight features. The proposed flight controller hardware is among the best on the market, with a superior microcontroller which is smaller than on other platforms.

A. CORE BOARD

The main unit or *core* houses the fundamental elements for UAV operation. It can work on its own, despite of the fact that the connection of the *flex* sensor unit provides more reliable status estimations and, consequently, more precise UAV control.

Figure 2 shows a block diagram of the elements built into the main unit. The core of the system is the STM32F767 microcontroller commercialized by STMicroelectronics®. This microcontroller offers the performance of the Cortex-M7 core with the capacities of a digital signal processor (DSP), incorporating a double-precision floating point unit. It is a 32-bit microcontroller with a maximum processing power of 462 DMIPS, which are executed from Flash

TABLE 1. Comparison between flight controller platforms.

Platform	MCU	Interfaces*	Sensors**	Dimensions (mm)	Weight (g)
URpilot	STM32f767	s, i, u, us, c	im, b, m	80×41×10	63.8
Pixhawk 4	STM32f767	s, i, u, us, c	im, b, m	84×44×12	38.8
CUAV v5+	STM32f767	s, i, u, us, c	im, b, m	85.5×42×33	91
Erle-Brain 3	Raspberry Pi	i, u, us, c	im, b, m	95×70×23.8	100

* Interfaces abbreviations: s→ SPI, i→ I2C, U→ UART, us→ USB, c→ CAN.
** Sensors abbreviations: im→ IMU, b→ barometer, m→ magnetometer.

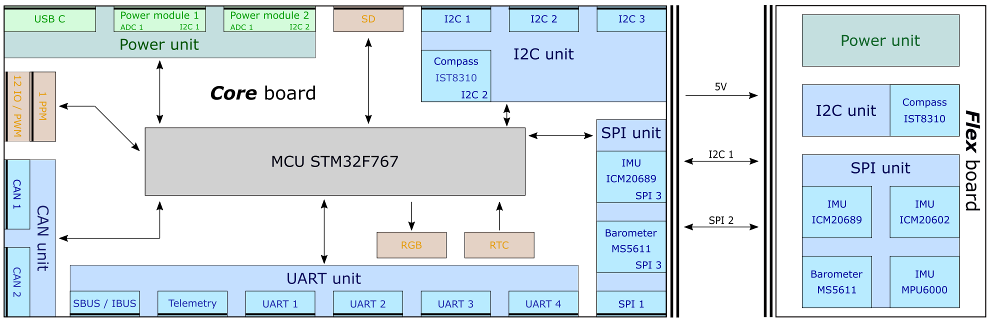


FIGURE 2. Hardware architecture used in the flight controller.

memory with 0-wait states thanks to ST's ART Accelerator. In addition, it has a working frequency of 216 MHz and a large number of peripherals. Finally, it should be noted that the microcontroller has 512 KBytes of SRAM, 128 KBytes of DCM RAM (for critical real-time data), 16 KBytes of instruction TCM RAM (for critical real-time routines) and 4 KBytes of backup SRAM. As further described in Section III-B, this type of memory is essential for the proper functioning of the RTOS. All these characteristics and a contained price make this device ideal for the current application.

The microcontroller is connected to sensors, communication ports and other devices that give the system a series of functionalities and capabilities on a par with the latest of flight controller models available on the market. This configuration is composed of:

- 2 Power module ports 5V.
- 1 IMU ICM20689 with a 3-axis gyroscope and a 3-axis accelerometer.
- 1 Magnetometer IST8310.
- 1 Barometer MS5611.
- 1 SBUS/IBUS port and 1 PPM input.
- 1 Telemetry port.
- 12 PWM isolated outputs, four of which can also be configured as inputs.
- 4 SPI, 3 I2C and 2 Can bus ports.
- 1 Micro-SD slot.

The core board is powered either through a USB Type-C port or two connectors dedicated to the external power modules. This configuration provides energy redundancy to the system and allows the voltage, current and energy of the application to be monitored in real time, as well as the estimation of remaining flight time. Communication ports have over-current protection, and can deliver up to a maximum of 5A to the rest of the auxiliary circuits that can be connected to the system, thus achieving extra protection in the power unit, an aspect of great importance for professional applications where maximum security is required.

B. FLEX BOARD

The flexible sensor unit or *flex* provides sensor redundancy to the system to achieve more precise status estimation and control. It is a flexible PCB whose sensors are arranged on the top face and which connects to the ≪*core*≫ unit. Section IV-E shows how the mechanical flexibility of this unit improves the performance of the sensors on board thanks to the fact that the vibrations are mechanically damped. To achieve mechanical decoupling, the design of the case shown in Figure 1 is proposed. The case has a housing for the sensor unit composed of two shock-absorbing elements at the top and bottom of the PCB forming a sandwich. This system allows the PCB to be sufficiently clamped and the vibrations to be absorbed by the pads that make up the sandwich.

The electronic configuration is as follows:

- 1 IMU ICM20689 with a 3-axis gyroscope and a 3-axis accelerometer.
- 1 IMU ICM20602 with a 3-axis gyroscope and a 3-axis accelerometer.
- 1 IMU MPU6000 with a 3-axis gyroscope and a 3-axis accelerometer.
- 1 Magnetometer IST8310.
- 1 Barometer MS5611.

III. SOFTWARE ARCHITECTURE

An RTOS is a low layer program that runs on a processor that schedules processes and manages peripherals, guaranteeing stability and controllability. Traditional embedded systems employ basic schedulers that handle critical tasks through interrupts while low-level priority tasks are executed on a main loop. However, the evolution of hardware has led to the emergence of new RTOS for embedded systems applied to UAVs [19]. Some issues associated with them are that they are very complex to understand, especially those Linux-based, and they are not designed ad hoc for an aircraft. The challenge is to design a hard real-time operating system that is easy to implement, flexible and with an optimization level that allows it to be implemented in any current embedded system.

The aim is for the scheduler to be as simple as possible, hence a time-sharing RTOS is used. These systems are made simpler by not using the interrupts of the microcontroller. However, they have the weakness that critical tasks are managed in the same way as the rest. A solution to this problem is designing a hybrid scheduler with two distinctive parts: first come first serve (FCFS) scheduling and earliest deadline first (EDF) scheduling. FCFS scheduling handles the highest priority critical tasks, which cannot be interrupted because the system could crash; they include the aircraft control by managing the actuators. As to EDF scheduling, it is the subsystem that handles the rest of the tasks, based on their priority. Within this part of the scheduler there are the different subsystems that make up the application layer of a flight controller, such as the navigation system, data storage, and communications.

A. TASK MODEL

This work defines a task as the minimal function that can be mapped on one core of the microcontroller. The proposed operating system model is designed for a one-core embedded system so that one task can never be interrupted by another, but it is the RTOS itself that decides when they are executed. The software is built according to the following five premises:

1) All tasks run periodically on a single CPU.
2) There are no data dependencies between processes.
3) The execution time for a process is constant.
4) All deadlines are at the ends of their periods.
5) The highest-priority ready process is always selected for execution.

The scheduler considers the software layer as a collection of n tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$. Each task τ_i is characterized by a static priority sp_i, a dynamic priority dp_i, a period T_i, a deadline dl_i equal to T_i, and a worst-case execution time w_i. The parameters sp_i, T_i, and dl_i are defined during the programming stage while dp_i and w_i are defined during run time.

The proposed flight controller operating system employs the hybrid scheduler shown in Figure 3, where tasks are classified as critical and non-critical. The critical tasks are those in the FCFS scheduling; they must be executed very fast, and their latency must be as low as possible. With this type of tasks, the static priority is the maximum that the scheduler supports while a dynamic priority is not necessary in this framework because an FCFS scheduler does not use the dynamic priority promotion system.

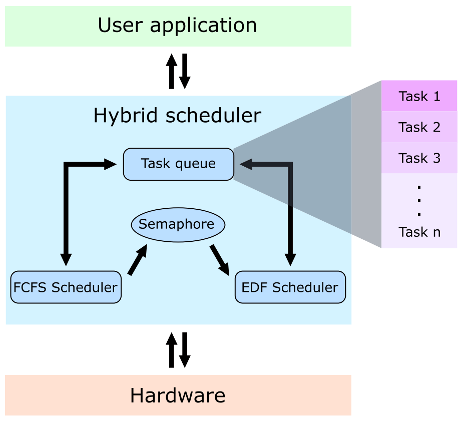


FIGURE 3. Scheduling architecture of the proposed flight controller RTOS.

The non-critical tasks are those that do not have a direct impact on the final application layer or can have deviations in the frequency of execution without jeopardizing software stability. These tasks do not require low latency or a very fast update time. These types of tasks are managed by EDF scheduling with a dynamic priority system where dp_i is calculated in real time, so that a non-critical task can be finally handled by the FCFS scheduler if its dynamic priority reaches the maximum priority value.

Both types of tasks are stored in a single task queue. The scheduler processes the jobs in the order in which they arrive in the ready queue. This linear buffer can change dynamically if the system decides to add or remove tasks from the queue at run time. When a task is added, the first empty space in the queue is searched and the buffer size index is increased. Conversely, when a task is deactivated, it is eliminated from the queue; the algorithm looks for it in the buffer and eliminates it. Subsequently, the tasks are moved one position up so that there are no gaps within the task buffer, and the task queue index is decreased. This feature provides flexibility when programming applications that require system redundancy or need to shut down program blocks.

B. FIRST COME FIRST SERVE SCHEDULING

FCFS scheduling automatically executes queued requests and tasks in order of arrival. The works [25], [26] show that it is the simplest and most intuitive technique for managing a set of tasks stored in a FIFO queue. This methodology does not support the pre-emption of tasks, so priority does not take effect in this type of algorithm. In this case, the tasks to be executed within this framework are those that have the highest priority of the system, called real-time.

This framework is modeled as a pipe where tasks enter; then, each task is checked to see if it must be executed, and the corresponding function is called. The algorithm is made up of a main loop that goes through all the tasks in the queue, discarding the tasks that do not belong to the real-time category.

Once a real-time task is found, the remaining execution time t_r is calculated as

$$t_r = le_i + T_i - t, \tag{1}$$

where le_i is the last time when this task was executed, T_i is the task period, and t is the actual time. If $t_r > 0$, the algorithm discards the task because it does not have to be called yet, and the system returns to the starting point to look for the next task. On the other hand, if $t_r \leq 0$, the task is selected to be executed.

Finally, when a task is selected, it is called, and, immediately afterwards, the statistics of maximum and average consumed time are calculated. The whole process is illustrated in Figure 4 for a system with three critical tasks.

This type of algorithm ensures that tasks can be executed as quickly as possible, but has the disadvantage that it is not possible to eliminate the latency generated when several tasks must be called at the same time. Since the design of the scheduler is oriented towards embedded systems with a single core, it is not possible to make simultaneous calls of several functions. Therefore, when two or more tasks must be called at the same time, there is a delay in the tasks equal to the time it takes for the functions to be executed.

To improve the behavior of this framework, this algorithm and the tasks that comprise it are housed within the ITCM and DTCM memories of the microcontroller. The purpose of the Instruction Tightly Coupled Memory (ITCM) is to provide low-latency memory for instructions that the processor can use without the unpredictability feature of caches; in other words, access to this memory has zero delay clock cycles. Additionally, Data Tightly Coupled Memory (DTCM) is a memory with similar characteristics to the ITCM, whose purpose, however, is to save the data used in critical tasks.

Both memories are directly connected to the processor, whereas traditional memories are wired to the master bus, just like the rest of the microcontroller peripherals. Therefore, these types of memories are perfect for this type of scheduler, where latency is very important and a deterministic execution time is required.

C. EARLIEST DEADLINE FIRST SCHEDULING

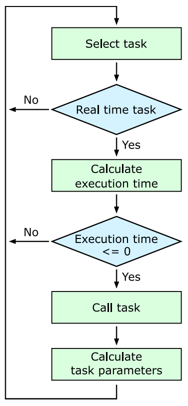
Works [27], [28], [29], [30], [31], [32] show that EDF scheduling can control a multitude of tasks with great results, by assigning static and dynamic priorities to the tasks; which is the reason why this system is chosen to handle non-critical tasks.

Earliest deadline first scheduling is an algorithm that uses a priority queue. This queue is dynamically updated by looking for the closest process to its deadline dl_i, thus ensuring that all the tasks are executed within a specific time for each function. This process is illustrated in Figure 5 for a system with three non-critical tasks.

Before running this algorithm the general semaphore of the scheduler comes into play, which is in charge of blocking the operation of the EDF scheduling framework. This semaphore is important because it allows to meet the requirements of dl_i for critical tasks called by FCFS scheduling. Blocking occurs when there is not enough time to run the algorithm (flowchart of Figure 5), defined by:

$$W > t_r, \tag{2}$$

where W is the time required to execute the EDF algorithm and t_r is the task execution time, which is computed as in (1).



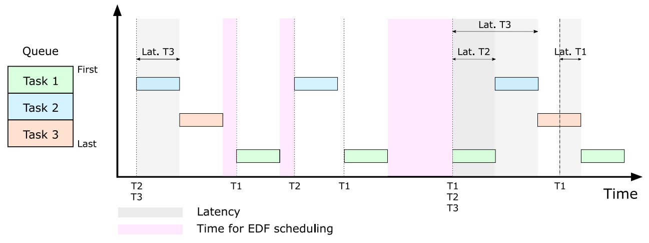


FIGURE 4. FCFS scheduling: flowchart and timing diagram.

When enough time is available to run the algorithm, the system starts looking in the queue for non-real-time tasks. As soon as a task is found, the dynamic priority calculation process begins; the difference between the FCFS and EDF algorithms is that, in the EDF case, the tasks are chosen based on three parameters: available time, priority, and execution time.

First, the dynamic priority of each task is defined by the following equations:

$$dp_i = \begin{cases} 1 + \dfrac{sp_i(t - le_i)}{T_i} & t_r \geq w_i \\ 0 & t_r < w_i \end{cases} \tag{3}$$

This process of calculating dp_i is repeated for all the tasks in the queue until the task with the highest dynamic priority is found. After finding it, the algorithm proceeds by calculating the execution statistics of the task and, finally, the function is called.

This algorithm ensures that all tasks are run within the target deadline dl_i regardless of priority or frequency of execution. As in FCFS scheduling, when two or more tasks must be executed at the same time, there is a latency in the call function. In addition, there is an increase in priority, thus reducing the execution delays in the tasks with respect to other types of algorithms.

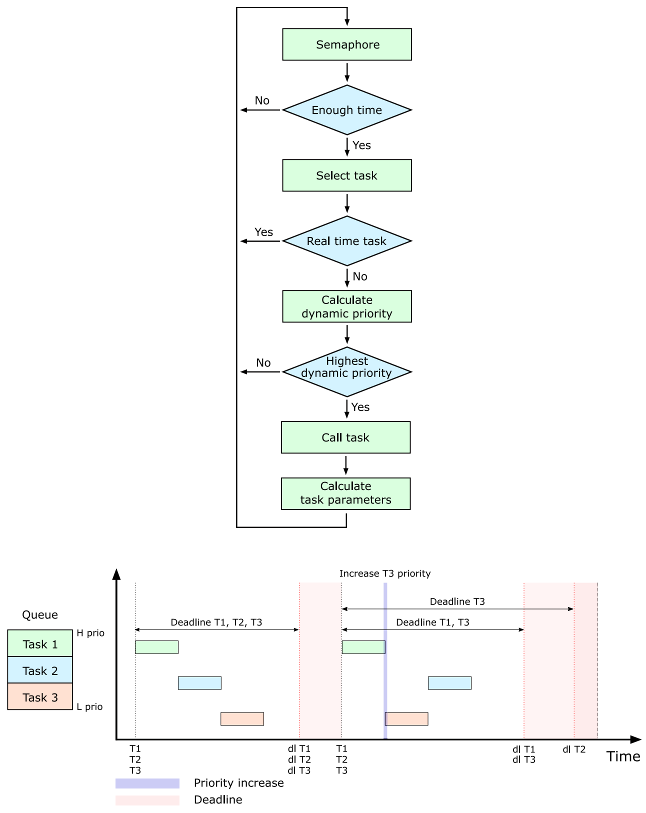


FIGURE 5. EDF scheduling: flowchart and timing diagram.

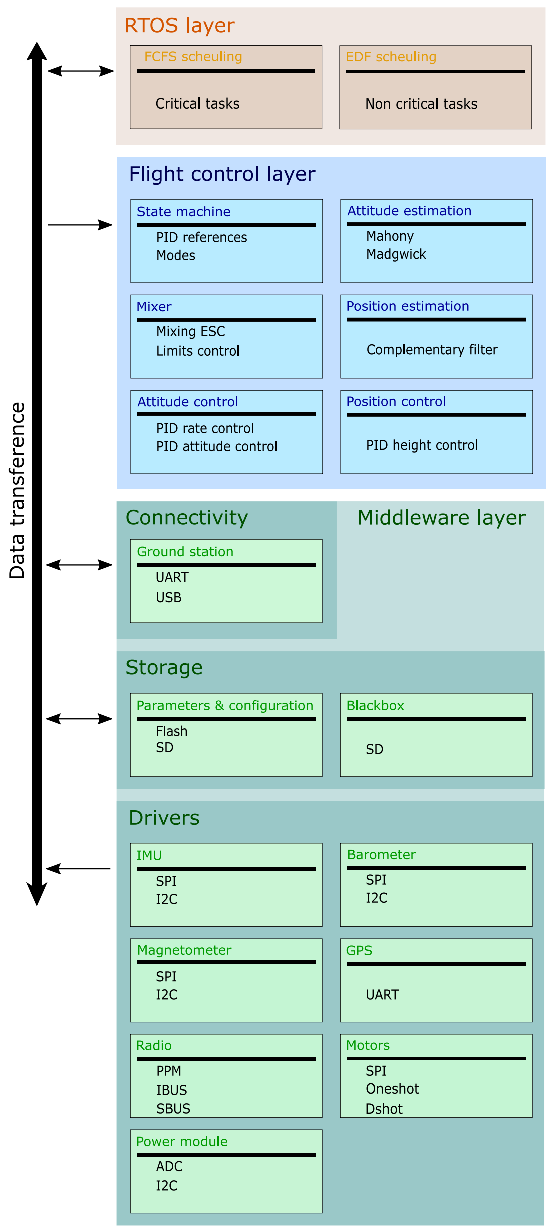


FIGURE 6. Task topology employed in a flight controller.

D. TASK TOPOLOGY

The source code is split into self-contained layers that exchange information to perform their tasks. The flight controller software is composed of three different layers:

- RTOS layer: the part of the software in charge of managing the system tasks.
- Middleware layer: a general layer of software that provides the necessary libraries to manage sensors, actuators and communications.
- Flight control layer: the highest-level layer of the application. In this work, it is the software part in charge of estimating and controlling the attitude and position of the UAV.

Figure 6 shows the different layers of the software and the data interconnections between them. A component in a layer can consume or contribute data asynchronously. Furthermore, this type of information exchange is totally parallel; which means that there are no delays in internal communications between modules.

1) MIDDLEWARE LAYER

This is the general library layer for any robot. It is made up of three different modules: connectivity, storage, and drivers.

The connectivity module manages external bidirectional communications with a ground station either with cable or wireless. This module uses two types of communications: UART and USB. Generally, telemetry modules work with asynchronous serial protocols, while, when connecting the controller to the ground station, the most common method used is USB. Both protocols work in parallel, hoping to find the interlocutor of the ground station to send and receive information or changes in the application configuration. The storage module manages the blackbox and the module configuration. The blackbox collects all the relevant data from the modules that make up the software. These data are stored in an SD card for later analysis. As to the module configuration, it handles the parameters of all the modules. This system uses the flash of the microcontroller or the SD card to store the information. When the software starts, it loads the data stored in RAM for functions to be used.

A large percentage of the middleware layer is responsible for sampling the sensor data to later make an accurate estimation of the attitude and position. Oversampling increase the resolution and signal-to-noise ratio. Then, a oversampling factor of N means a sampling rate of N times the Nyquist rate, which causes the noise power to be reduced by a factor N. This technique implies that there are two tasks for each sensor with different update times and priorities. Figure 7 shows this work methodology whose tasks are called foreground and background.

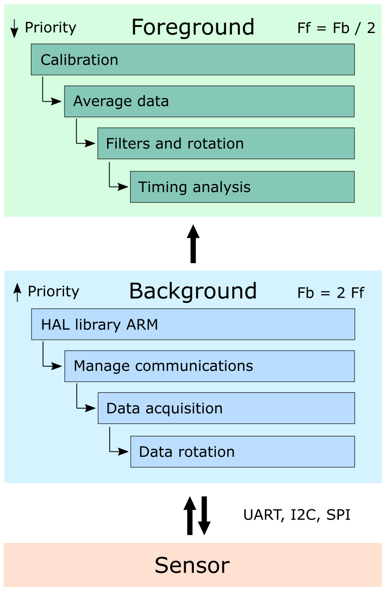


FIGURE 7. **Foreground/Background flowchart.**

The background encompasses high-priority tasks that are responsible for managing communications with the sensor, and collecting and rotating data when necessary. These tasks run at twice the foreground speed to achieve a oversampling factor of two.

The foreground is the framework where lower-priority tasks are encompassed. These tasks are in charge of calibrating, calculating, rotating and filtering the data obtained in the background. In addition, a small analysis of the timing data is done to show the operating status of the sensor in real time.

2) FLIGHT CONTROL LAYER

The flight control layer is a set of algorithms for the estimation, guidance, and control of drones. The core of this layer is the cascade control of the UAV. Figure 8 shows four different chained PID controllers: two for attitude control and two for position control.

The reference generator converts the positions of the sticks of the RC transmitter into the set points that are entered in the position and attitude control blocks. The flight mode causes the activation or deactivation of the aircraft position control, enabling the attitude control references to be changed, which can come from the reference generator or the higher-level position control.

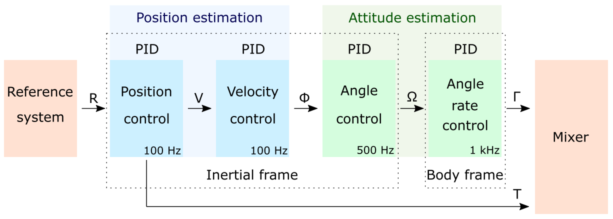


FIGURE 8. **Flight control diagram.**

Each control block contains a PID controller with parallel structure and a proportional feedforward. The control action $u(t)$ is obtained based on the reference $r(t)$ and the tracking error $e(t) = r(t) - y(t)$ according to

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de_d(t)}{dt} + K_{ff} r(t). \quad (4)$$

The use of the derived term $e_d(t)$ corresponds to the tracking error after the application of a low pass filter. This filter is two-pole and allows to select the cut-off frequency in Hertz.

Finally, there is a mixer block. This block transforms the output of the three loops of the angular velocity control, as well as the output of the altitude control loop, into the motor command signals. Depending on the type of drone configured, this block is capable of adapting the inputs to the number of rotors available in the aircraft.

IV. EVALUATION

This section evaluates the proposed RTOS as part of the URPilot flight controller whose hardware and software were defined in previous sections. The results are based both on the operating system efficiency and actual aircraft operation using the proposed hardware.

The hardware-software proposed as flight controller and specially its RTOS have been designed seeking greater flexibility, simplicity, and optimization than other software packages on the market, although a complete analysis is necessary to check its behavior.

A. SCHEDULING PARAMETRIZATION

One of the major issues is set the period of each task so that the scheduler is able to meet its timing constraints. One solution is End-To-End Design proposed by [15]. This period selection methodology abandons the heuristic method of trial and error and focuses on the numerical resolution of the problem, minimizing the cost of the schedulability test and without exceeding latency limits. The procedure relies on two parameters: the retention time E and the freshness time F.

The reaction time is defined as the time it takes for a sample of sensor data to traverse the program and produce a change of the actuator. If the task path from sensor to actuator is framed into two pipes, the worst-case reaction time associated to the path is

$$E^{WC}_{\tau_p \to \pi_p | \tau_c \to \pi_c} = \begin{cases} T^C + C^P & T^C < T^P \\ T^P + C^P & otherwise; \end{cases} \quad (5)$$

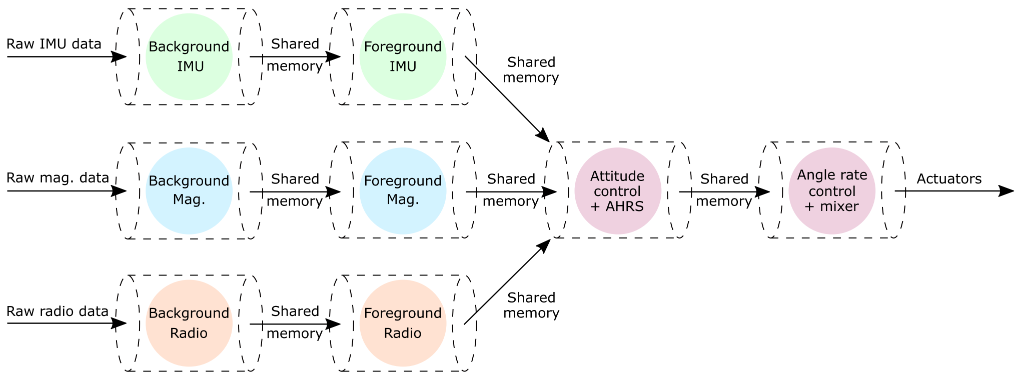


FIGURE 9. URpilot data flow.

where T and C denote the period and the budget time, respectively, of a task associated to a pipe, and subscripts C and P refer to the consumer and producer tasks in the two-pipe chain. The notation $\tau \to \pi$ designates the association of task τ to pipe π; can be in the side of consumer C or producer P. If the chain has more than two pipes, the reaction time (5) is increased by

$$\uparrow E^{WC} = \begin{cases} T^C & T^C < T^P \\ T^P - C^P + C^C & otherwise. \end{cases} \quad (6)$$

On the other hand, the freshness time is the time it takes for an actuator change sample to be detected by a sensor input, and its worst case for a two-pipe chain is

$$F^{WC}_{\tau_p \to \pi_p | \tau_c \to \pi_c} = \begin{cases} 2\,T^P & T^C < T^P \\ T^P + C^C & otherwise. \end{cases} \quad (7)$$

Figure 9 shows a case where the firmware has been simplified into three data paths that start from the IMU, magnetometer and radio, respectively, to the actuator outputs. The tasks labeled in the diagram are associated to pipes and their periods and worst-case execution times are going to be determined (the execution time is closely related to the budget time). The rest of the system tasks will not be taken into account in the analysis because they do not contribute to flight stability, and their periods will be adjusted as multiples of the periods calculated in the analysis. Table 2 collects the whole set of tasks. Certain upper limits are established for the worst case of the reaction and freshness times as in [15]. Thus, $E^{WC} <= 10$ and $F^{WC} <= 20$ for the IMU path, and $E^{WC} <= 25$ ms and $F^{WC} <= 45$ ms for both the magnetometer and radio paths. Using the time limits and formulation (5)-(7), the periods of tasks in Table 2 are defined.

Figure 10 compares several reaction and freshness times: "Constraint" refers to their upper limits, "Predicted" refers to the values obtained by (5)-(7), and "Observed" refers to the current values obtained for the URpilot flight controller. All this verifies the method for the scheduler parametrization and shows a good performance of the proposed hardware-software.

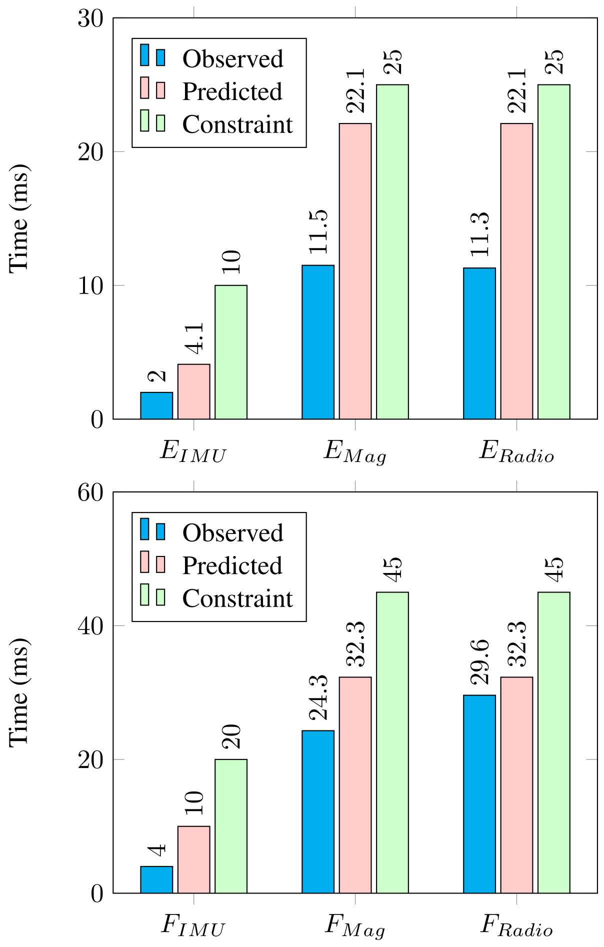


FIGURE 10. URpilot reaction time (up) and freshness times (down).

B. SCHEDULABILITY TEST

Previous analysis show that the latency obtained is within the established limits, but did not give any information on whether the deadlines would actually be met. Works in [33], [34], [35], and [36] propose the use of rate monotonic scheduling (RMS). This algorithm performs an execution modeling of all tasks in the system and determines the amount

TABLE 2. Tasks that make up the flight controller.

Task	Task name	Priority	Period (ms)	WCET (μs)	σ
τ_1	Scheduler analysis	Low	100	10	5.02
τ_2	Stack	Low	100	7	4.95
τ_3	ADC	Low	1 000	18	4.61
τ_4	Background power module	Low	20	8	4.88
τ_5	Foreground power module	Low	100	17	2.01
τ_6	Background barometer	High	10	50	0.66
τ_7	Foreground barometer	Medium-high	20	50	1.13
τ_8	Background magnetometer	High	10	350	0.66
τ_9	Foreground magnetometer	Medium-high	20	37	4.89
τ_{10}	Calibration magnetometer	Low	50	41	3.44
τ_{11}	Background IMU	Real time	1	57	0.08
τ_{12}	Foreground IMU	Real time	2	37	0.55
τ_{13}	Calibration IMU	Low	50	22	3.48
τ_{14}	GPS	Medium	20	355	1.43
τ_{15}	Background radio	Medium	10	21	1.60
τ_{16}	Foreground radio	Medium	20	53	1.85
τ_{17}	Navigation	Medium	20	47	1.97
τ_{18}	Angle rate control	Real time	2	83	0.08
τ_{19}	Angle control	High	2	71	0.55
τ_{20}	Position control	High	10	11	0.76
τ_{21}	State machine	Low	20	11	4.83

of time it takes to comply with the guarantees for the set of processes in question.

Each task is assumed to have the following characteristics:

1) Each task is a periodic task of period T, which represents the frequency with which it executes.
2) An execution time C, which is the CPU time required during the T period.
3) A utilization U, which is the ratio C/T.
4) Only tasks consume time. The scheduler itself is treated as if it executes instantaneously.

Liu and Layland [37] introduced static and dynamic priority scheduling algorithms. Their static priority algorithm was called rate-monotonic scheduling (RMS), and their dynamic priority algorithm was known as earliest deadline first (EDF). The proposed RTOS is a hybrid system that requires the analysis of both solutions.

In the first place, the analysis of critical tasks is presented; let us remember that these tasks are managed by the FCFS scheduler of the RTOS layer of the flight controller and, specifically, they are the tasks of Table 2 that have a real-time priority. Liu and Layland [37] proved that for a set of n periodic tasks with unique periods, any such task set is schedulable using rate monotonic scheduling if:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \tag{8}$$

where $\frac{C_i}{T_i}$ is the percentage of time that task i will occupy the processor at each arrival, in the worst case. Thus, the test states that the sum of these processor utilization percentages, for all n tasks, must not exceed the utilization bound.

When the number of processes tends towards infinity, the utilization bound is defined as:

$$\lim_{n\to\infty} n(\sqrt[n]{2} - 1) = \ln 2 \approx 0.69. \tag{9}$$

Therefore, a rough estimate is that RMS can meet all the deadlines if total CPU utilization, U, is less than 70%.

The EDF scheduler of our flight controller manages the tasks in Table 2 that do not have real-time priority. Applying the left part of Equation (8) to them results in a utilization bound

$$U_{FCFS} = 0.17 \approx 18\%, \tag{10}$$

which is a very good result, since it only contemplates the part of the critical tasks executed by the FCFS framework.

For the EDF framework, Liu and Layland [37] showed that any task set is schedulable under this policy if processor utilization by all tasks does not exceed 100%. A necessary test for independent tasks using earliest deadline first scheduling (EDF) is:

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{11}$$

Applying the left part of Equation (11) to the rest of the tasks involved in the EDF scheduler results in a utilization bound:

$$U_{EDF} = 0.1 = 10\% \tag{12}$$

Both tests result in the great performance of the proposed RTOS and the hardware used for flight control. In total the hybrid scheduler (FCFS + EDF) has an utilization factor of 28% compared to 70% for a pure FCFS system or 100% for a pure EDF system.

Figure 11 shows a comparison of this factor with respect to the best known firmwares on the market for flight control. The proposed system (URpilot) shows a lower utilization factor than the two Ardupilot options; the simplified Ardupilot option is the most similar to the one proposed in this paper due to the number of tasks that are included in the

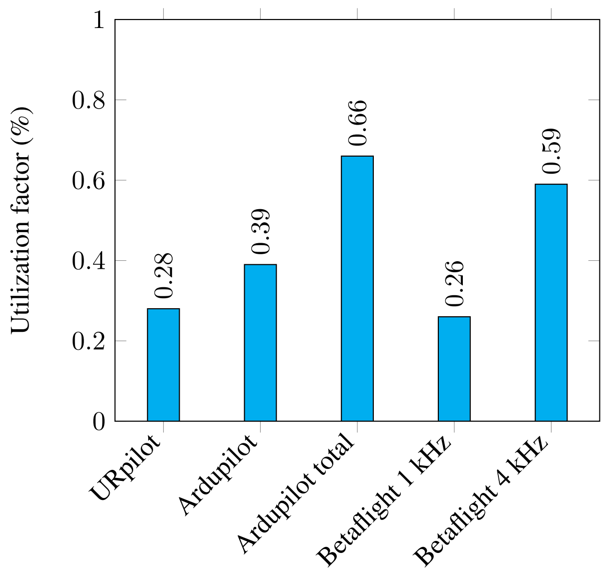


FIGURE 11. Comparison of utilization factors between different firmwares.

firmware. In contrast, Betaflight shows two very different results depending on the running frequency of the PID controllers; the 1KHz version achieves slightly better results than URPilot. Taking into account that Betaflight is a firmware oriented to racing drones and whose functionalities are limited, it can be said that the proposed RTOS has the properties of a light and fast software but that it also has the characteristics and functionalities of professional use software.

C. CPU UTILIZATION

Understanding the processor load in an embedded system is important. It is one more mechanism that allows to know the deadlines of the system and if they are being complied with correctly. Incorrect use of the CPU can cause a system failure, which in the case of an aircraft is usually lethal. For this reason, it is very important that the system executes the tasks loosely, bearing in mind that the final application may require a greater workload. In this case, a 40% utilization target has been defined, with a 30% margin for future functions required in the final application. When it is close to 65%-75%, it is advisable to replace the processor with a higher one or to limit the amount of system tasks as this can cause a serious failure in flight.

CPU utilization is represented by

$$U_{CPU} = 100 - I, \tag{13}$$

where I is the percentage of time that the CPU is idle and is expressed as

$$I = \frac{T_{nl}}{T} 100. \tag{14}$$

T_{nl} represents the time that the processor is without load, while T is the time of the last calculation period.

Applying the previous formulation to the proposed system, we obtain $T_{nl} = 84\,807\mu$s, $T = 100\,000\mu$s, $I = 84.807\%$, and, finally, $U_{CPU} = 15.193\%$. This result shows a great performance, considering that the software is designed for professional applications where external sensors or actuators may be added, causing the need for greater number of tasks to be executed, and therefore, forcing an increase in CPU utilization.

D. PERIOD DEVIATION TEST

The deviation of the period task of a processor is an aspect to take into account when assessing the effectiveness of a scheduler.

Standard deviation of the period is defined as

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{n}(T_i - T)}, \tag{15}$$

where N is the number of samples, T_i is the period measured in a sample and T is the target period.

Applying the previous equation to a set of 1 000 samples, the results of Table 2 are obtained. Very small deviations are observed as task priority increases, resulting in deviations of less than 1% for high priority tasks where the execution times of the tasks are more important.

On the other hand, a comparison of the deviations with traditional schedulers is proposed, thus showing the superiority of the proposed system. Figure 12 details a comparison of standard deviations between the proposed system and a pure EDF scheduler; the tasks are grouped by level of priority (L: low, MH: medium-high, H: high, RT: real-time), which gives σ_L, σ_{MH}, σ_H and σ_{RT}. The results show the superiority of the proposed hybrid scheduler. This improvement is important when executing higher-priority tasks where the latency must be as low as possible to guarantee the stability of the final application.

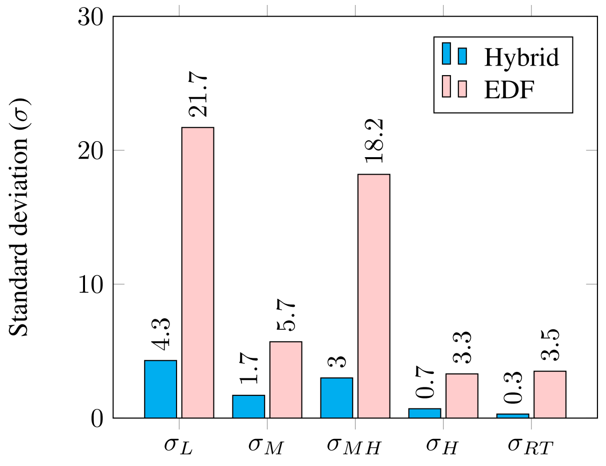


FIGURE 12. Comparison of standard deviation of task execution frequency between hybrid and EDF schedulers.

E. HARDWARE TEST

Hardware is an aspect that also has to be analyzed in the results of the proposed system. Flex board proposal

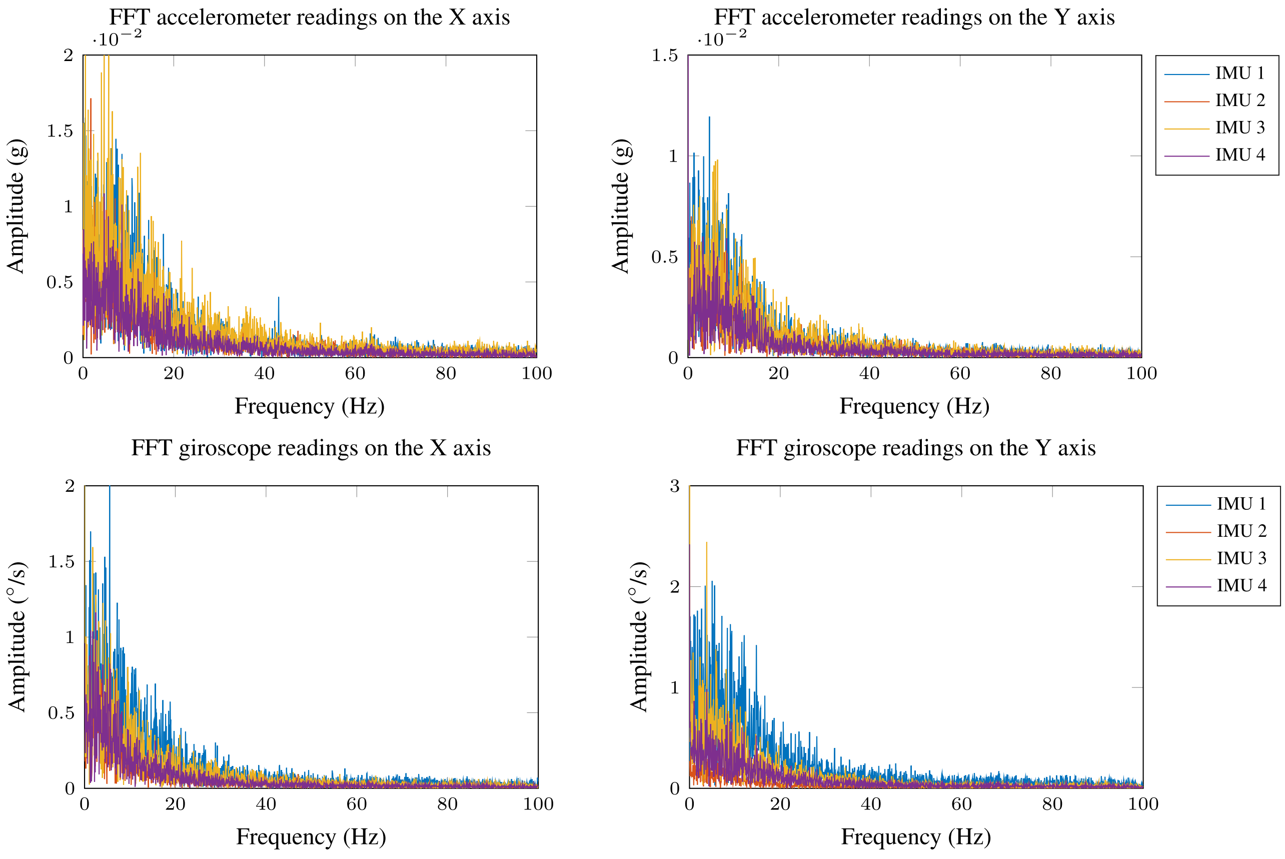


FIGURE 13. FFT analysis of system IMUs.

in Section II-B provides more accurate readings with less interference than core board sensors. To demonstrate this statement, an experiment is carried out with the drone in stationary flight, which records 100 000 samples thrown by its different IMUs. Figure 13 compares the Fast Fourier Transform (FFT) of IMU accelerometer and gyroscope measurements. It proves that the sensors located on the flexible board (IMU 2, IMU 3, IMU 4) are less affected by the mechanical vibrations of the aircraft. On the contrary, the IMU embedded in the core board (IMU 1) provides noisier signals, causing a worse estimation of the aircraft attitude.

The best way to merge information from multiple IMUs is not a trivial task [38], [39]. Neither is fault-tolerant data fusion [40], which allows the aircraft to continue in flight despite an IMU suffering a critical failure. All this would require a specific study outside of the present work. Nevertheless, a simple fault-tolerant data fusion system based on approaches [40], [41] is described below. Then a real experiment will prove its effectiveness, as well as the usefulness of the featured hardware.

The current fault-tolerant arrangement contains two functional units: the error detector and the recovery system; the former warns of errors and the latter restores a proper operating state. Error detection obeys to the Duplication-Comparison method [41], which is one of the most common routines when there are redundant sensors. The algorithm works in parallel with the measurements of several IMUs to compare the results and to make decisions under the existence of inconsistencies. Here there are three IMUs (i= 1, 2 and 3) and the measurement of each IMU i consists of a vector $\boldsymbol{x}_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,6}\}$ with accelerations and velocities in the three axes of rotation from 3D accelerometers and 3D gyroscopes. A raw data fusion will be performed using the Weighted Average Voting system [40] to generate a fault-tolerant virtual IMU. The voter output merges redundant sensor data and certain internal parameters triggers the error-detection and the recovery system routines. The voting algorithm comprises the following three steps.

Firstly, the measurements of each (i,j)-pair of IMUs are compared by calculating the distance vector $\boldsymbol{d}_{ij}$. Three distance vectors, $\boldsymbol{d}_{12} = |\boldsymbol{x}_1 - \boldsymbol{x}_2|$, $\boldsymbol{d}_{13} = |\boldsymbol{x}_1 - \boldsymbol{x}_3|$ and $\boldsymbol{d}_{23} = |\boldsymbol{x}_2 - \boldsymbol{x}_3|$, are calculated. A distance vector contains six components that represent the distances between the homologous components of $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and evidence possible discrepancies between sensors. Thus, a component of $\boldsymbol{d}_{ij}$ that reaches a high magnitude means a mismatch between the information from sensor i and sensor j.

Secondly, a weight coefficient must be associated to each sensor of each IMU; the result for IMU i will be $\boldsymbol{\omega}_i = \{\omega_{i,1}, \omega_{i,2} \ldots, \omega_{i,6}\}$, where $\omega_{i,k}$ will weigh the measurement

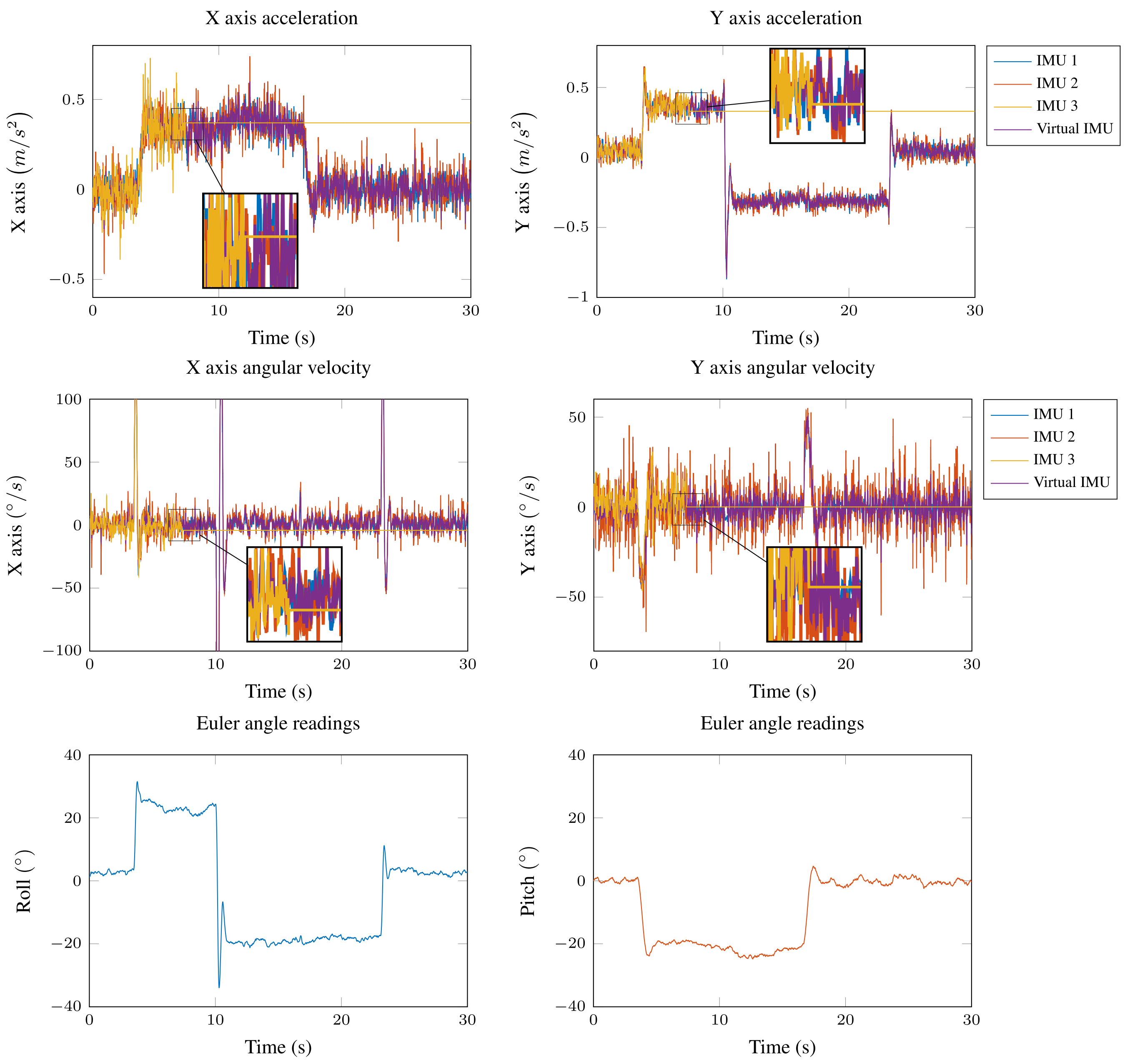


FIGURE 14. **System behavior in the event of IMU failure.**

of sensor k of IMU i according to its consistency with the sensor k of redundant IMUs, i.e. weight coefficients will determine the contribution of each sensor in the voting system that yields the output of the virtual IMU. To find appropriated weights $\omega_{i,k}$, agreement indicators $s_{ij,k}$ are previously computed taking into account distances $d_{ij,k}$ as follows

$$s_{ij,k} = \begin{cases} 1 & \text{if } d_{ij,k} \leq a_k \\ \left(\dfrac{n_k}{n_k-1}\right)\left(1-\dfrac{d_{ij,k}}{n_k a_k}\right) & \text{if } a_k < d_{ij,k} < n_k a_k \\ 0 & \text{if } d_{ij,k} \geq n_k a_k \end{cases} \quad (16)$$

where a_k is a fixed threshold below which measurements are considered eligible by the voting systems, and n_k is another positive tuneable thresholding parameter such that $n_k a_k$ is a second threshold that allows a partial contribution to be configured. Distances above $n_k a_k$ warns of serious discrepancies between sensors that suggest the failure of one of them.

Considering the agreement indicators among IMUs for sensor k, the weight coefficient of sensor k of IMU i is calculated by means of

$$\omega_{i,k} = 0.5 \sum_{j=1, j\neq i}^{3} s_{ij,k} \quad (17)$$

Finally, in the third step, the voting system combines the measurements of each IMU. The voter output of sensor k is calculated as follows:

$$\bar{x}_{k=1,\ldots,6} = \frac{\sum_{i=1}^{3} x_{i,k}\omega_{i,k}}{\sum_{i=1}^{3} \omega_{i,k}} \tag{18}$$

Then, the output of the virtual IMU produces $\bar{\boldsymbol{x}} = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_6\}$.

Beyond being a method of merging redundant information, the Weighted Average Voting system warns of sensor failure. Thus, an error detection algorithm watches when the coefficients $s_{ij,k}$ take null values during 100 consecutive time instants. If this occurs for both $s_{ij,k}$ of sensor k of IMU i, that sensor is faulty. In such a case, the recovery system removes the affected sensor from the estimation system, which starts operating with the two remaining sensors.

Figure 14 demonstrates the sensor fault-tolerance of the featured hardware. A loss of communication with IMU 3 is simulated via software: constant measurements are received from time $t = 8$s. The error detection and recovery systems discard the six measurements of the faulty IMU and allow an adequate estimation of the Roll and Pitch angles. The virtual IMU turns out to be immune to sensor failure.

V. CONCLUSION

The proposed flight controller was designed in response to the latest needs that have emerged in the professional drone market. In recent years, professional applications that use UAVs require aircrafts to be safer to protect the equipment they carry on board. The proposed hardware provides the necessary security for any application. On the one hand, the input and output power supply is protected against failures and, on the other hand, the system has redundancy in the sensor unit, which provides better measurements for the estimation of the orientation, opens the way to fault-tolerant data fusion, and also behaves better against mechanical vibrations generated in flight.

There are numerous RTOS on the market, but it is very difficult to make a selection in terms of flexibility, speed and ease of use. Therefore, this paper proposes the use of a specific RTOS for this type of applications, seeking to increase the task execution speed, minimizing latency, and facilitating the programmer use. This is a hybrid scheduler in which critical real-time tasks are controlled by an FCFS framework. Non-critical tasks, by contrast, are driven by the EDF framework with dynamic priority that causes functions to be executed at the right time. The simplicity of this system is in the use of a single task queue shared by the two schedulers and in which a non-critical task can evolve into a real-time task.

The results demonstrate good system performance compared to other systems on the market in terms of CPU utilization and different schedulability tests proposed in the literature. These results open the way for future work on aircraft estimation and control, seeking to improve the performance of the current traditional control PID loops.

REFERENCES

[1] S. O. H. Madgwick, A. J. L. Harrison, and A. Vaidyanathan, "Estimation of IMU and MARG orientation using a gradient descent algorithm," in *Proc. IEEE Int. Conf. Rehabil. Robot.*, Zurich, Switzerland, Jun./Jul. 2011, pp. 1–7.

[2] P. Narkhede, S. Poddar, R. Walambe, G. Ghinea, and K. Kotecha, "Cascaded complementary filter architecture for sensor fusion in attitude estimation," *Sensors*, vol. 21, no. 6, p. 1937, Mar. 2021.

[3] N. A. Chaturvedi, A. K. Sanyal, and N. H. McClamroch, "Rigid-body attitude control," *IEEE Control Syst. Mag.*, vol. 31, no. 3, pp. 30–51, Jun. 2011.

[4] D. Brescianini and R. D'Andrea, "Tilt-prioritized quadrocopter attitude control," *IEEE Trans. Control Syst. Technol.*, vol. 28, no. 2, pp. 376–387, Mar. 2020.

[5] J. Rico-Azagra, M. Gil-Martínez, R. Rico, S. Nájera, and C. Elvira, "Benchmark de control de la orientación de un multirrotor en una estructura de rotación con tres grados de libertad," *Revista Iberoamericana de Automática E Informática Ind.*, vol. 18, no. 3, pp. 265–276, Jul. 2021.

[6] A. Vetrella, G. Fasano, D. Accardo, and A. Moccia, "Differential GNSS and vision-based tracking to improve navigation performance in cooperative multi-UAV systems," *Sensors*, vol. 16, no. 12, p. 2164, Dec. 2016.

[7] J.-J. Xiong, N.-H. Guo, Y.-X. Hong, and E.-H. Zheng, "Improved position and attitude tracking control for a quadrotor UAV," in *Proc. Chin. Autom. Congr. (CAC)*, Nov. 2019, pp. 4197–4201.

[8] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A system for autonomous flight using onboard computer vision," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011, pp. 2992–2997.

[9] I. Dryanovski, R. G. Valenti, and J. Xiao, "An open-source navigation system for micro aerial vehicles," *Auto. Robots*, vol. 34, no. 3, pp. 177–188, Apr. 2013, doi: 10.1007/s10514-012-9318-8.

[10] A. Caruso, S. Chessa, S. Escolar, J. Barba, and J. C. Lopez, "Collection of data with drones in precision agriculture: Analytical model and LoRa case study," *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16692–16704, Nov. 2021.

[11] L. Yang, B. Li, W. Li, H. Brand, B. Jiang, and J. Xiao, "Concrete defects inspection and 3D mapping using CityFlyer quadrotor robot," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 4, pp. 991–1002, Jul. 2020.

[12] O. S. Oubbati, M. Atiquzzaman, T. A. Ahanger, and A. Ibrahim, "Softwarization of UAV networks: A survey of applications and future trends," *IEEE Access*, vol. 8, pp. 98073–98125, 2020.

[13] M. Zhang, M. Timmerman, L. Perneel, and T. Goedeme, "Which is the best real-time operating system for drones? Evaluation of the real-time characteristics of NuttX and ChibiOS," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*, Jun. 2021, pp. 582–590.

[14] A. Czerniejewski, K. Dantu, and L. Ziarek, "JUAV: A real-time Java UAV autopilot," in *Proc. 2nd IEEE Int. Conf. Robotic Comput. (IRC)*, Jan. 2018, pp. 258–261.

[15] Z. Cheng, R. West, and C. Einstein, "End-to-end analysis and design of a drone flight controller," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 1, pp. 2404–2415, Nov. 2018.

[16] D. A. Ponomarev, T. O. Kuzmina, and A. D. Stotckaia, "Real-time control system for a tracked robot," in *Proc. IEEE Conf. Russian Young Researchers Electr. Electron. Eng. (EIConRus)*, Jan. 2020, pp. 814–818.

[17] B. Sababha, H. C. Yang, and O. Rawashdeh, "An RTOS-based run-time reconfigurable avionics system for UAVs," in *Proc. AIAA Infotech@Aerospace*, Apr. 2010, p. 3414.

[18] R. Hujja, R. Sumiharto, and S. Wibowo, "Realtime operating system implementation on AVR XMEGA for unmanned aerial vehicle autopilot," *Int. J. Adv. Res. Sci., Eng. Technol.*, vol. 5, pp. 5762–5768, May 2018.

[19] V. Kangunde, R. Jamisola, and E. Theophilus, "A review on drones controlled in real-time," *Int. J. Dyn. Control*, vol. 9, pp. 1–15, Dec. 2021.

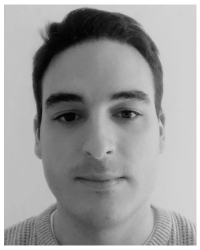
[20] E. Ebeid, M. Skriver, K. H. Terkildsen, K. Jensen, and U. P. Schultz, "A survey of open-source UAV flight controllers and flight simulators," *Microprocessors Microsyst.*, vol. 61, pp. 11–20, Sep. 2018.

[21] D. Stojcsics and A. Molnár, "AirGuardian—UAV hardware and software system for small size UAVs," *Int. J. Adv. Robotic Syst.*, vol. 9, no. 5, p. 174, Nov. 2012.

[22] H. Lim, J. Park, D. Lee, and H. J. Kim, "Build your own quadrotor: Open-source projects on unmanned aerial vehicles," *IEEE Robot. Autom. Mag.*, vol. 19, no. 3, pp. 33–45, Sep. 2012.

[23] C. C. Zhih, S. K. V. Ragavan, and M. Shanmugavel, "Development of a simple, low-cost autopilot system for multi-rotor UAVs," in *Proc. IEEE Recent Adv. Intell. Comput. Syst. (RAICS)*, Dec. 2015, pp. 285–289.

[24] N. Zimmerman, "Flight control and hardware design of multi-rotor systems," M.S. thesis, Sep. 2016. [Online]. Available: http://epublications.marquette.edu/theses_open/370
[25] G. D. Bibu and G. C. Nwankwo, "Comparative analysis between first-come-first-serve (FCFS) and shortest-job-first (SJF) scheduling algorithms," *Int. J. Comput. Sci. Mobile Comput.*, vol. 8, pp. 176–181, May 2019.
[26] A. P. U. Siahaan, "Comparison analysis of cpu scheduling: FCFS, SJF and round Robin," *Int. J. Eng. Develop. Res.*, vol. 4, pp. 124–132, Jul. 2016.
[27] M. Ryu and S. Hong, "Deterministic and statistical deadline guarantees for a mixed set of periodic and aperiodic tasks," in *Real-Time and Embedded Computing Systems and Applications*, J. Chen and S. Hong, Eds. Berlin, Germany: Springer, 2004, pp. 72–87.
[28] R. M. Pathan, "Unifying fixed- and dynamic-priority scheduling based on priority promotion and an improved ready queue management technique," in *Proc. 21st IEEE Real-Time Embedded Technol. Appl. Symp.*, Apr. 2015, pp. 209–220.
[29] M. Wang, Z. Du, Z. Liu, and S. Hao, "The dynamic priority based scheduling algorithm for hard real-time heterogeneous CMP application," *J. Algorithms Comput. Technol.*, vol. 2, no. 3, pp. 409–427, Sep. 2008.
[30] C. Dietrich and D. Lohmann, "OSEK-V: Application-specific RTOS instantiation in hardware," in *Proc. 18th ACM SIGPLAN/SIGBED Conf. Lang., Compil., Tools Embedded Syst.*, Jun. 2017, pp. 111–120.
[31] Z. Dong, C. Liu, S. Bateni, Z. Kong, L. He, L. Zhang, R. Prakash, and Y. Zhang, "A general analysis framework for soft real-time tasks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1222–1237, Jun. 2019.
[32] S. Manolache, P. Eles, and Z. Peng, *Real-Time Applications With Stochastic Task Execution Times: Analysis and Optimisation*. Cham, Switzerland: Springer, 2007.
[33] S. Manolache, P. Eles, and Z. Peng, "Schedulability analysis of applications with stochastic task execution times," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 4, pp. 706–735, Nov. 2004.
[34] C. J. Fidge, "Real-time schedulability tests for preemptive multitasking," *Real-Time Syst.*, vol. 14, pp. 61–93, Jan. 2004.
[35] P. K. Suri and S. Mittal, "Design of stochastic simulator for analyzing the impact of scalability on CPU scheduling algorithms," *Int. J. Comput. Appl.*, vol. 49, no. 17, pp. 4–9, Jul. 2012.
[36] A. Atlas and A. Bestavros, "Statistical rate monotonic scheduling," in *Proc. 19th IEEE Real-Time Syst. Symp.*, Dec. 1998, pp. 123–132.
[37] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, Jan. 1973.
[38] A. Larey, E. Aknin, and I. Klein, "Multiple inertial measurement units—An empirical study," *IEEE Access*, vol. 8, pp. 75656–75665, 2020.
[39] U. N. Patel and I. A. Faruque, "Multi-IMU based alternate navigation frameworks: Performance comparison for UAS," *IEEE Access*, vol. 10, pp. 17565–17577, 2022.
[40] H. Hamadi, B. Lussier, I. Fantoni, and C. Francis, "Data fusion fault tolerant strategy for a quadrotor UAV under sensors and software faults," *ISA Trans.*, vol. 129, pp. 520–539, Oct. 2022.
[41] K. Bader, B. Lussier, and W. Schön, "A fault tolerant architecture for data fusion: A real application of Kalman filters for mobile robot localization," *Robot. Auton. Syst.*, vol. 88, pp. 11–23, Feb. 2017.



RAMÓN RICO received the master's degree in industrial engineering from the University of La Rioja, Spain, in 2016. He is currently an Automation Engineer at Nidec Press & Automation (ARISA) and a Researcher Collaborator with the Control Engineering Group, University of La Rioja. He is also researching and participating in publications on unmanned aerial systems. He is a member of the Spanish Association of Automatic Control (CEA).

JAVIER RICO-AZAGRA received the master's and Ph.D. degrees in industrial engineering from the University of La Rioja, Spain, in 2009 and 2017, respectively. He is currently an Assistant Professor of systems engineering and automation at the University of La Rioja. He has published several articles in reputed journals in the field of robust control theory (quantitative feedback theory). His current research and publications are focused on unmanned aerial systems. He is a member of the Spanish Association of Automatic Control (CEA).

MONTSERRAT GIL-MARTÍNEZ received the master's degree in electrical and control engineering from the University of Cantabria, Spain, in 1995, and the Ph.D. degree in industrial engineering from the Public University of Navarra, Spain, in 2001. She is currently an Associate Professor of systems engineering and automation and the Leader of the Control Engineering Research Group, University of La Rioja, Spain. She has published several articles in reputed journals in the fields of robust control theory (quantitative feedback theory) and in control applied to waste-water treatments. She is also participating in research projects and publications on unmanned aerial systems. She is also a member of the Spanish Association of Automatic Control (CEA).