Original Article

Revista elektron, Vol. 6, No. 2, pp. 65-76 (2022)
https://doi.org/10.37537/rev.elektron.6.2.162.2022

Design of a flight controller to achieve improved fault tolerance

Diseño de una controladora de vuelo para lograr tolerancia a fallas mejorada

Claudio Pose[¶*§1], Leonardo Garberoglio[†§2], Ezequiel Pecker-Marcosig[*‡§3], Ignacio Mas[¶§4] and Juan Giribet[¶§5]

[¶]*Departamento de Ingeniería - Universidad de San Andrés*
Vito Dumas 284, Buenos Aires, Argentina
[*]*Facultad de Ingeniería, Universidad de Buenos Aires*
Paseo Colón 850, CABA, Argentina
[§]*Consejo Nacional de Investigaciones Científicas y Técnicas*
Godoy Cruz 2290, Ciudad Autónoma de Buenos Aires, Argentina
[†]*Grupo de Estudio de Sistemas de Control, Facultad Regional San Nicolás, Universidad Tecnológica Nacional*
Colón 332, San Nicolás, Argentina
[‡]*Instituto de Ciencias de la Computación (ICC-CONICET)*
Madero 351, Ciudad Autónoma de Buenos Aires, Argentina
[1]cldpose@fi.uba.ar
[2]lgarberoglio@frsn.utn.edu.ar
[3]epecker@fi.uba.ar
[4]imas@udesa.edu.ar
[5]jgiribet@conicet.gov.ar

Recibido: 04/10/22; Aceptado: 24/11/22

***Abstract*—In the last years, multirotor aerial vehicles have gained popularity both as consumer products and in professional applications. Safety is one of the main concerns during operation, and different approaches to fault tolerance have been proposed and continue to be developed. For a control system to be able to handle off-nominal situations, failures must be properly detected and identified; therefore, a fault detection and identification algorithm is required. Also, the control loop has to be accordingly modified to cope with each particular failure in the best way possible. These algorithms usually run on the vehicle's low-level flight computer, imposing on it a large additional computational load. In this work, a fault detection and identification module is used to evaluate its impact in terms of additional processing time on a flight computer based on the Cortex-M3 micro-controller. While a highly optimized version of the algorithm is able to run, it still suggests potential hardware limitations for expanding the system capabilities. The evaluation of the same module on an improved flight computer design based on a Cortex-M7 micro-processor shows a significantly reduced footprint in the overall performance, allowing for the addition of an augmented method for faster failure detection.**

Keywords: Flight computer; Unmanned Aerial Vehicles; Fault Tolerance; Fault Detection and Identification.

***Resumen*— En los últimos años, los vehículos aéreos multirotores han ganado popularidad tanto en productos de consumo como en aplicaciones profesionales. La seguridad es una de las principales preocupaciones durante la operación y diferentes enfoques a la tolerancia a fallas se han propuesto y continúan desarrollándose. Para que un sistema de control maneje situaciones fuera de lo nominal, las fallas deben detectarse e identificarse adecuadamente, por lo tanto, se requiere un algoritmo de detección e identificación de fallas. Además, el lazo de control debe modificarse en consecuencia para hacer frente a cada falla de la mejor manera posible. Estos algoritmos generalmente se ejecutan en la computadora de vuelo de bajo nivel del vehículo, lo que le impone una gran carga computacional adicional. En este trabajo se utiliza un módulo de detección e identificación de fallas para evaluar su impacto en términos de tiempo de procesamiento adicional en una computadora de vuelo basada en el microcontrolador Cortex-M3. Si bien se puede ejecutar una versión altamente optimizada del algoritmo, aún sugiere posibles limitaciones de hardware para expandir las capacidades del sistema. La evaluación del mismo módulo en un diseño de computadora de vuelo mejorado basado en un microprocesador Cortex-M7 muestra una huella significativamente reducida en el rendimiento general, lo que permite agregar un método aumentado para una detección de fallas más rápida.**

Palabras clave: Computadora de Vuelo; Vehículo Aéreo no Tripulado; Tolerancia a Fallas; Detección e Identificación de Fallas.

I. Introduction

During the last few years, small-scaled unmanned aerial vehicles have become very popular. Due to the reduction in production costs, advances in technology and their ease of use, they have been captivating the public for both simple recreational uses and professional industry applications.

Among them, multirotor-type vehicles have been preferred for many of these applications, as their vertical take-off and landing (VTOL) capabilities, along with their ability to hover in place (remain static in the air), offer a simpler learning curve, greater maneuverability, and a higher degree of safety for less experienced users. In commercial fields, many applications have been adopting this solution instead of their manned counterparts, i.e., manned helicopters, as has happened in the movie industry and in aerial inspection for civil structures. One application that promises a significant use of this kind of vehicles is package delivery, in which companies such as Amazon and UPS have been investing and developing for the past few years, and encouraged research in the same field, aiming for integration of this kind of vehicles with public transport to extend their operational range [1].

As the use of aerial vehicles is becoming massive, it is starting to make use of aerial space over densely populated areas, and safety concerns begin to arise. Manned aerial vehicles, such as commercial planes and helicopters, have decades of invested research and development, have been thoroughly tested, and count on international standards to comply. Unlike their manned counterparts, the history of unmanned commercial vehicles is quite recent, with new advances and discoveries that are continually improving their reliability. Furthermore, there are countries that haven't yet issued regulations concerning their use, while the ones that do have, only consider local regulations, lacking a worldwide panorama.

For the aforementioned reasons, fault tolerance has become an important aspect to be accounted for in unmanned vehicles' design. The ability to continue a normal flight in the event of a component failure is not only critical to ensure the integrity of the vehicle and prevent possible damages to third parties, but also to provide a high degree of reliability in the completion of sensitive missions, such as delivering medical equipment, or search and rescue in disaster zones. Some of the solutions for fault tolerance are rather simple: relying, for example, in hardware redundancy [2], which is possible when considering failures in the commonly used sensors, as the added cost is not prohibitive, and the additional weight is negligible.

On the other hand, when considering failures in the actuator set, i.e. the motors and propellers, the solution is not so straightforward. In many occasions, multirotors with a high number of motors are used to increase the payload capacity, and also gain more stability and robustness against perturbations. In these cases, the design can be exploited to achieve fault tolerance against motor failure based on hardware redundancy, when certain conditions are satisfied [3]–[6]. If hardware redundancy is not possible, more complex solutions based on partial loss of control, mechanical design and/or vehicle reconfigurability have been proposed [7], [8].

In cases of failure in one of the motors in a multirotor vehicle, the control strategy will heavily depend on which of the motors is the one presenting the failure, so that maximum performance can be achieved in any situation. In consequence, the failure has to be properly detected and identified in order to choose the optimal control solution, for which a Fault Detection and Identification (FDI) algorithm is required. This kind of algorithms are usually implemented through an observer-based solution [9], and generally are quite demanding in terms of required processing power.

This poses a challenge to unmanned aerial vehicles, in which the on-board computer that runs the algorithms for sensor data acquisition and control generally relies on a low level microcontroller as the main processing unit, with limited resources and processing power. Special attention has to be paid to the complexity of the algorithms used, in order to ensure that they can be executed in a given time window [10], as the low level microcontroller has to prioritize tasks such as sensor data acquisition and attitude control.

The implementation of efficient algorithms in microcontrollers with limited resources is a relevant research topic. Most of the existing autopilot firmware intended for small-scale UAVs rely on Proportional-Integral-Derivative (PID) algorithms to perform low-level control. Some efforts were carried out to implement innovative algorithms for the attitude control in Cortex-M microcontrollers over an existing firmware [11]. However, it's usually preferable to consider simplified versions of more advanced and well-established control techniques [12]. Moreover, efforts have been made to get rid of the idea of periodic execution of control laws [13], [14], leaving room to run more intensive algorithms on the same hardware.

This work focuses on analyzing the additional load imposed by the implementation of a fault tolerant control module in multirotor-type unmanned air vehicles, in flight computers based on low-level microcontrollers of the Cortex-M family. Particularly, the implementation of a classical bank of observers for fault detection and identification in a custom-made flight computer will be shown to consume roughly 12% of a Cortex-M3 microcontroller resources, as it doesn't count with a Floating Point Unit (FPU). This additional load, together with the rest of the data acquisition and control algorithms that concurrently run inside the microcontroller, push its processing capabilities to the limit, with the possibility of control loop overruns. For these reasons, a new flight computer was developed in our Lab, using a Cortex-M7 microcontroller with a single-precision FPU, in order to reduce the processing times. Counting with an FPU, as well as a higher clock speed, will show to reduce the processing time for this kind of algorithms, in order to run together with all the common guidance, navigation and control (GN&C) algorithms required in multirotor systems. Moreover, the Cortex-M7 will be able to run a more complex fault detection and identification algorithm, allowing for faster fault detection. The proposed algorithm includes a model of how a faulty motor behaves, in particular its transient response until it stops working completely.

This manuscript is organized as follows. Section II presents the common characteristics in multirotor flight computers and the design of a custom-made, Cortex-M3 based board. Section III introduces the fault tolerance basics, and Section IV, the case of study with the common fault tolerant techniques for multirotor vehicles. Section V describes the implementation of fault tolerant control in the Cortex-M3 based board and its limitations, while Section VI presents a new board design based on a Cortex-M7 to overcome them. Section VII will show a new method for faster fault detection, taking advantage of the extra resources. Finally, Section VIII includes the concluding remarks and future work.

II. Flight controller in unmanned vehicles

At the core of a UAV is the flight controller (FC) board, a small computer in which, generally, a low-level microcontroller is used as the main processing unit, to perform tasks such as sensor data acquisition and execution of the algorithms needed to stabilize and control the vehicle. Flight computers usually count with a variety of sensors, such as an Inertial Measurement Unit (IMU) composed of a tri-axial accelerometer and gyroscope, a tri-axial magnetometer, a barometer, a Global Positioning System (GPS) receiver, and different kinds of speed and distance sensors, which generally provide data at a fixed rate, ranging from tens of Hz to several thousands of kHz. Moreover, the FC receives commands from a remote controller encoded in a Pulse Width Modulated (PWM) signal of $50\,\mathrm{Hz}$ and produces PWM commands between $200\,\mathrm{Hz}$ and $800\,\mathrm{Hz}$ to set the speed of a fixed number of BLDC motors (brushless, DC) through an Electronic Speed Controller (ESC).

In practice, some of these tasks must be executed as soon as an external event occurs, others with a certain periodicity, and others fall somewhere in the middle. For example, for the attitude control loop that is executed at a fixed frequency, the IMU needs to be read with the same periodicity to estimate the attitude. As IMU units have their own internal clock, this can be done by either letting the IMU perform the readings at a fixed frequency and then inform the microcontroller that a measurement is available through an external interrupt (Interrupt ReQuest - IRQ), or by polling the IMU from the microcontroller at a fixed frequency. Other sensors, such as barometers and magnetometers, do not usually count with an internal clock, but may have a Data ReadY (DRY) or End Of Conversion (EOC) pin to trigger an external interrupt in the microcontroller. In this case, the microcontroller requests a measurement and continues executing other tasks while the sensors perform and store the reading in their internal registers, and let the microcontroller get the fresh measurement as soon as possible using the DRY pin.

On the other hand, peripherals, such as the remote receiver which outputs several consecutive PWM signals (each with a high time ranging from $0.5\,\mathrm{ms}$ to $2\,\mathrm{ms}$) corresponding to attitude/position reference values and/or flight modes, don't have a predictable timing as they depend solely on the transmitter-receiver link and the user. Hence, an interrupt input should be used to accurately measure time between each rising and falling edge, to reconstruct the signal. This issue shows the need for different levels of priority for each task. For example, a delay of tens or hundreds of nanoseconds in reading the IMU measurements from given registers may not be critical, but a similar delay in detecting an edge of a PWM signal will result in a significant error when reconstructing the pulse signal.

For those reasons, it is evident why commercial FCs rely on low-level microcontrollers that are event-oriented, rather than using a general purpose operating system (OS) with no real-time guarantees where deadlines are managed in a best-effort way, which in some cases is unacceptable. In the case of real-time operating systems (RTOS), which do not suffer from this setback, its use may or may not be advantageous, as any operating system will increase the computational load of the microcontroller. In cases where the resources are tight, it may be preferable not to use them. Some off-the-shelf FCs rely on light RTOS, such as PixHawk with NuttX OS, that, due to its emphasis on technical standards compliance and scalability, proves to be useful when porting the software between different microcontrollers.

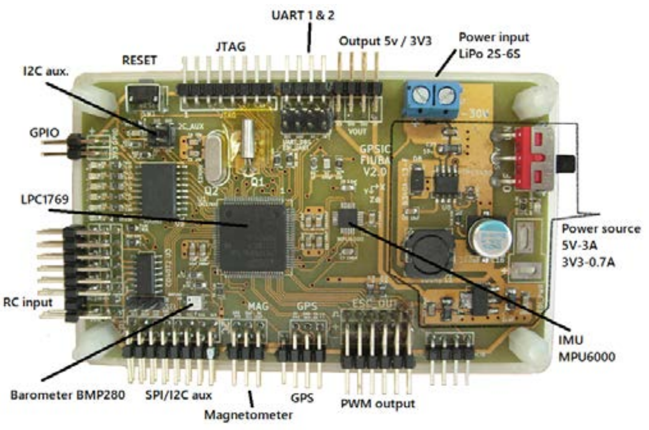

Fig. 1: Flight computer Choriboard v2.

Several models of commercial FCs are available nowadays, and many of them are open hardware and software, so any user is able to modify them at will to adapt them to their needs. However, from an academic perspective, the development of a custom FC presents some advantages, namely adapting board size, power requirements, processing power, and input/output ports for experimental vehicles which may have non-standard actuators and/or sensors, and to promote educational projects on UAV technologies. Moreover, knowing the firmware in detail is an upper hand when modifying key algorithms, and the access to low-level code allows for efficient implementation of new algorithms, without depending on third parties.

A. *Choriboard FC v1*

In this line of thought, a custom FC was developed for academic purposes called Choriboard, the first version of which (Choriboard v1) dates back to 2014 within the context of an undergrad student project. This board was designed on an oversized PCB, with enough space between components to allow for error checking, and several additional on-board integrated circuits that were initially used for testing and for educational purposes. Also, as an academic research project, the components' cost and board manufacturing were important issues, so surface mount chips and through-hole, $2.54\,\mathrm{mm}$ pin headers were chosen for ease of soldering and connection. Once the original design was debugged and errors were corrected, a second version was produced (Choriboard v2), that retained the main features, and optimized the form factor and the number of on-board components. As these two versions are essentially the same, only the Choriboard v2 will be described in detail below.

B. *Choriboard FC v2*

The second version of the Choriboard was designed in a $100\,\mathrm{mm} \times 60\,\mathrm{mm}$ board, which is shown in Fig. 1. The microcontroller chosen for this board was the LPC1769, a 32-bit RISC ARM Cortex-M3 chip that can work at $100\,\mathrm{MHz}$, has 512kB of flash memory and 64kB of data memory. This model doesn't have a FPU, so floating-point operations are emulated by software, which makes them much less efficient considering that it must run the GN&C

algorithm. At the time, the LPC1769 was chosen due to its availability, as well as for its well known libraries that were constantly updated by a wide online community.

The use of Cortex-M microcontrollers as the core of an FC presents several advantages in terms of efficiency in the kind of tasks that these computers execute. For example, the Nested Vectored Interrupt Controller (NVIC) allows for low-latency interrupt processing, and to establish groups and subgroups of priorities for both internal and external IRQs. This is convenient in order to prioritize tasks such as the reading of the RC receiver, where sub-microsecond precision is required to accurately reconstruct the signal, or where sensor data has to be stamped with the exact time of acquisition in order to implement a robust navigation system. Also, a useful feature present in Cortex-M is the Direct Memory Access (DMA), which allows for data transfer between peripherals and RAM memory (and vice-versa), and between memory and memory, without additional burden to the processor. This comes in handy for any sensor reading task, as the processor can continue executing other tasks until all sensor data is transferred, as well as for bidirectional communication channels with a ground station, as no additional processing time is required independently of the amount of data sent and received. Another common feature of these series of microcontrollers is the PWM module, whose operation is based on one or more internal timers and a set of registers where matching conditions are stored. Additionally, they're able to work asynchronously of the main program. This is ideal for motor speed control, as one signal of this kind is required for each rotor present in the vehicle.

One of the most important components in an FC is the Inertial Measurement Unit, generally consisting of a tri-axial accelerometer and gyroscope, which provides the minimum required information to estimate the attitude of the vehicle, and thus achieve a stable flight. The selected component was the MPU6000 from Invensense, an SPI device with characteristics summarized in Table I. It comes with an important feature, called the Digital Motion Processor (DMP), that performs internal calculations using the accelerometer and gyroscope measurements, and outputs a filtered attitude estimation, which otherwise would have to be done using microcontroller resources. The attitude estimation is completed using the information of a digital compass (magnetometer) that provides the orientation with respect to the Earth's magnetic north pole, for which this board uses an HMC5883L from Honeywell connected through an I2C interface.

To estimate position, considering outdoor vehicle operation, the board includes a BMP280 barometer from Bosch (measuring pressure that can be converted to height), and a UART serial channel to connect a GPS receiver that supports both NMEA (ASCII) and SiRF (binary) protocols. However, the serial channel can also be used for other positioning devices, such as indoor positioning systems.

Regarding the radio controller receiver, such devices output a series of PWM signals, one for each channel of the remote controller, generally corresponding to four analogue sticks (for commanding vertical thrust and orientation) and the state of several switches and dials. These PWM signals may be available as a series of consecutive signals in a single output (known as PPM output), or as separate outputs, for which a PWM to PPM encoder is implemented on the board, and allows for both type of signals to be read.

TABLE I: IMU Comparison

PRODUCT Package-Pin	ICM 20602 2x2x0,75mm LGA 16 leads	MPU6000 4x4x0,9mm QFN 24 Leads
	GYROSCOPE	
FSR (dps)	±250/500/1000/2000	±250/500/1000/2000
ZRO @25ºC	±1dps	±20dps
ZRO over Temp	± 0,02dps/ºC	± 0,16dps/ºC
Sens. Tol. @25ºC	±1%	±3%
Sens. Over Temp.	±2%	±2%
Gyro Noise	0.004dps/$\sqrt{Hz}$	0.005dps/$\sqrt{Hz}$
	ACCELEROMETER	
FSR	±2/4/8/16g	±2/4/8/16g
Offset @ 25ºC	±25mg	X,Y : ±50mg Z: ±80mg
Offset Over Temp	X,Y: ±0.5mg/ºC Z: ±1mg/ºC	X,Y: ±0.5mg/ºC Z: ±85mg/ºC
Sens. Tolerance	±1%	±3%
Sens. Over Temp	±1,5%	±2,5%
Accel Noise	100µg/$\sqrt{Hz}$	400µg/$\sqrt{Hz}$

The board also has several additional communication interfaces, such as two UARTs, two I2C and an SPI port for additional sensors, GPIO ports, and digital inputs. A switching power source is also included on board, that takes as input a 7 to 30 V direct voltage, so it connects directly to batteries generally used in unmanned aerial vehicles, and provides 5 V and 3.3 V outputs for the on-board devices and possible future co-processor boards.

III. Fault tolerance

Fault tolerance is the ability of a given system to continue operating in case a failure occurs, which could be of different degrees of severity, and which may or may not degrade the performance of the system. For example, a failure in a redundant sensor may not affect the performance, as there is still a way to obtain the same information from another sensor, but a full power failure in electronic systems may render them unusable. The objective of designing fault tolerant systems is to account for a variety of possible failures, in order make the system robust to its occurrence.

Fault-Tolerant Control Systems (FTCS) are generally divided into two main categories, Active FTCS (AFTCS) and Passive FTCS (PFTCS) [15]. The latter is designed considering only a limited number of failure scenarios, and the same control law operates in nominal conditions and in failure, although with degraded performance. On the other hand, AFTCS are more of a tailored solution, as they usually cover a wider range of possible failures, and each failure is tackled in a different way in order to obtain the maximum possible performance. This implies that, as the behavior of the system will depend on the type of failure that occurs, it is a requirement for the system to know as accurately as possible the kind and magnitude of the failure, for which a Fault Detection and Isolation (FDI) subsystem is required. Also, the control law has to be actively modified to cope with the appearance of a failure, in order to maximize performance in each case. Also, the system may or may not include a reconfiguration mechanism to make fault tolerance

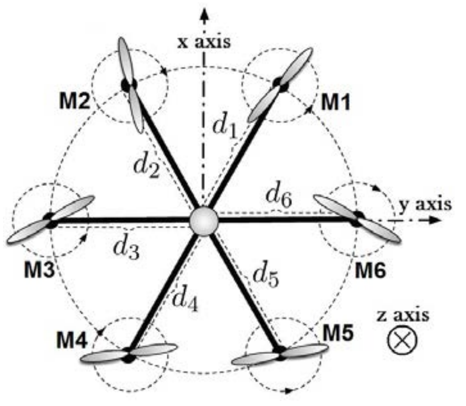

Fig. 2: Top view of a standard hexarotor vehicle distribution.

possible, or to improve its performance in case of failure. This results in a FTCS able to deal with more cases of failure, while still maintaining the best performance in any situation. However, the controller is sensitive to the output of the FDI, as the failure has to be correctly isolated for an optimal behavior.

An important consequence of choosing AFTCS over PFTCS becomes evident at this point. The PFTCS operates in the same way in both the nominal and failure cases, with no modification in the control algorithm. Conversely, the AFTCS needs a way to analyze the system's behavior to detect and identify the cause of failure, and then modify the control law accordingly. Therefore, the latter necessarily requires more processing power to work properly. This requirement generally arises from the need to track the system states, and find deviations from its expected behavior, which is the *detection* part of the FDI subsystem. Once a failure is deemed to have occurred, the capability of the system to discover exactly what kind of failure is producing the unexpected behavior constitutes the *isolation* part of the FDI subsystem.

IV. Case study: Fault Tolerant Control for an Hexarotor

In this work, a multirotor is considered fault tolerant against motor failures if it can keep the minimum maneuverability needed for normal flight, i.e. if it can exert torque over its three axes and ascend/descend. Hence, the analysis on fault tolerance will be carried out for a hexarotor-type vehicle, which is depicted in Fig. 2. Its six rotors are placed at the vertices of a regular hexagon, at the same distance d_i from its geometric center, all pointing upwards. The position of the center of mass is coincident with the geometric center, where a reference frame is defined with the x axis pointing to the nose of the vehicle, the z axis pointing down, and the y axis completing the frame. Unlike sensor failures, that may be solved easily with redundant hardware due to their low weight and affordable cost, the motor group has a significantly higher weight and cost, meaning that the number, size and type of motors has to be carefully selected for a given vehicle, and hardware redundancy may not be an option.

For this kind of vehicles, where the vehicle's dynamics will heavily depend on which motor is failing, AFTCS are preferred as they tackle the fault by modifying the control strategy to get maximum performance. There are several works that implement this kind of fault tolerant systems, such as [8], [16], [17] for hexarotors, focused on multirotor design to achieve fault tolerance, and [5], [18], where the actuator redundancy in standard octorotor vehicles is exploited to achieve fault tolerance even in some case of multiple rotor failures. To implement the FDI, there are two main approaches commonly found in the literature: direct measurement of the state of the motors, and indirect measurement or estimation.

A. Direct Measurement FDI

The direct measurement FDI approach is straightforward: the system should have a way to directly measure some state(s) of a motor that provide(s) enough information to decide if the motor is working as expected, or if it presents a fault [19], [20]. This is usually achieved using additional sensors (other than the strictly necessary for the vehicle to perform a standard flight) for each motor, measuring, for example, their speed or current consumption, and by additional filtering by software to estimate if these parameters are consistent with the commanded speed. The additional sensors produce an increase in cost and weight, and also present scalability issues, as the number of sensors is proportional to the number of rotors.

Another problem of using direct measurements to perform FDI is shown in Figs. 3 and 4. In the first figure, a speed sensor had been installed in one motor in a test bench, and four different experiments were carried out with the motor initially working at 40%, 50%, 60% and 70% of its maximum speed. At $t = 0\,\mathrm{s}$, the power of the motor is turned off and the motor's speed begins to decrease, however, due to its inertia, it continues spinning and, thus, generating a significant force and torque for at least a couple of seconds, causing significant delays in the fault detection. On the other hand, if the propeller is suddenly detached from the motor, the force and torque would instantly disappear, and the lower moment of inertia would make the motor stop spinning much faster.

In the second experiment (Fig. 4), a test bench was built to measure the consumption of a motor as a function of its PWM command signal, shown in the blue continuous line, depicted as reference. Then, a flight was performed with an hexarotor vehicle using six motors of the same kind with varying degrees of wearing due to use, and propellers in different conditions (new, slightly wore, slightly damaged), while measuring the current consumption without any filtering. It is shown that the current consumption of each motor deviates from the test bench curve, and also presents measurement noise. In these cases, to infer if the motor is working properly or not becomes more difficult due to a wide array of possible behaviors.

Both speed and current sensors have to deal with many different situations and behaviors of the variable measured, and be robust enough to guarantee a detection in a reasonable amount of time for the fault tolerant control to be able to correct it. This poses a challenge when designing FDI subsystems by direct measurement, as it is difficult to ensure robustness against possible modeling errors. Additionally,

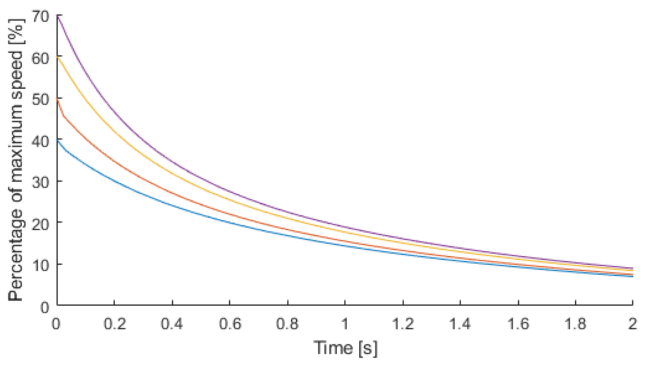

Fig. 3: Speed over time of an hexarotor motor that is initially working at 40%, 50%, 60% and 70% of its maximum speed, for which the power is cut off at $t = 0\,\text{s}$.

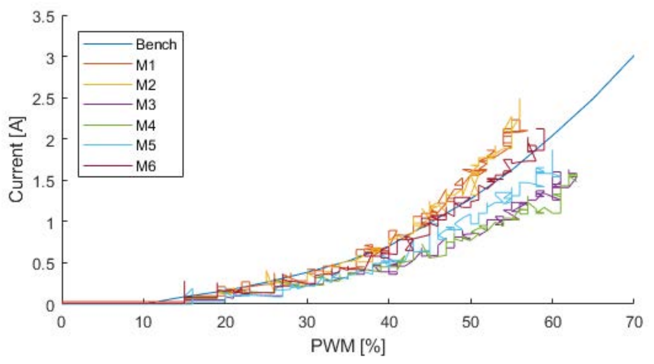

Fig. 4: Measured current with respect to PWM commands for each motor, for a hexarotor in flight, and test bench reference.

for the reasons described above, both sensors (speed and current) are simultaneously needed to obtain a robust fault detection system, as each one has the ability to cover for the shortcomings of the other.

B. Indirect Measurement FDI

The indirect measurement approach, on the other hand, is generally implemented by means of an observer-based detection and isolation algorithm. A standard implementation of this solution is shown in Fig. 5, in a typical attitude control loop of a multirotor, that estimates the attitude using the on-board sensors, and then generates a desired torque $\mathbf{q}$ to follow the reference, which is accomplished by commanding an adequate set of forces $\mathbf{f}$ to the motors. The FDI subsystem uses the information of the commanded motor forces to predict the dynamics of the multirotor; if the difference between the predicted dynamics with respect to the real behavior (called the *residues*) is close to zero, then all the motors are working properly, if a deviation exists, it may be caused by the appearance of a failure.

The simplified vehicle dynamics for the hexarotor shown in Fig. 2, disregarding the gyroscopic and aerodynamics components, among other factors, and linearised around hovering, are shown below:

$$\begin{bmatrix}\ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}J_x^{-1}q_x\\ J_y^{-1}q_y\\ J_z^{-1}q_z\end{bmatrix} \tag{1}$$

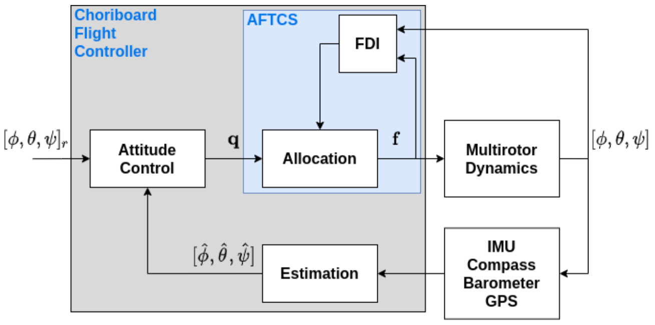

Fig. 5: Block diagram of the typical attitude estimation and control system in a multirotor, where the FDI algorithm detects and identifies failures based on input-output variables.

where $[\phi, \theta, \psi]$ correspond to the roll, pitch, and yaw angles respectively (rotation around the x, y and z axes), and J_i and q_i, $i \in \{x, y, z\}$ to the inertia moment and the torque exerted in the i axis. Then, the state-space representation is as follows:

$$\begin{aligned}\mathbf{x} &= \begin{bmatrix} \phi\;\theta & \dot{\phi}\;\dot{\theta}\;\dot{\psi} \end{bmatrix}^T\\ \mathbf{y} &= \begin{bmatrix} \phi\;\theta & \end{bmatrix}^T\\ \dot{\mathbf{x}} &= \underbrace{\begin{bmatrix}\mathbf{0}^{3\times3} & \mathbf{1}^{3\times3}\\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3}\end{bmatrix}}_{A_c}\mathbf{x} + \underbrace{\begin{bmatrix}\mathbf{0}^{3\times6}\\ I_m^{-1}A\end{bmatrix}}_{B_c}\mathbf{f}\\ \mathbf{y} &= \underbrace{\begin{bmatrix}\mathbf{1}^{3\times3} & \mathbf{0}^{3\times3}\end{bmatrix}}_{C_c}\mathbf{x}\end{aligned} \tag{2}$$

with $\mathbf{q} = [q_x, q_y, q_z]^T = A\cdot\mathbf{f}$, where $\mathbf{f}$ is the vector of forces exerted by the motors, A is the force-torque matrix dependent on the motor disposition, and $I_m = diag(J_x, J_y, J_z)$. The corresponding discrete state-space model, considering a fixed time step T_d, where $f_d = 1/T_d$ is the frequency of execution of the control loop, is as follows:

$$\begin{aligned}\mathbf{x}_k &= \begin{bmatrix} \phi_k\;\theta_k\;\psi_k\;\dot{\phi}_k\;\dot{\theta}_k\;\dot{\psi}_k \end{bmatrix}^T\\ \mathbf{y}_k &= \begin{bmatrix} \phi_k\;\theta_k\;\psi_k \end{bmatrix}^T\\ \mathbf{x}_{k+1} &= A_d\mathbf{x}_k + B_d\mathbf{f}_k\\ \mathbf{y}_k &= C_d\mathbf{x}_k,\end{aligned} \tag{3}$$

where matrices (A_d, B_d, C_d) correspond to the discretization model of system (2).

To implement a classical FDI algorithm, usually a bank of observers is used. Since the vehicle's state vector is observable, it is possible to design an observer to estimate its attitude around nominal state (i.e. hovering) and with all the motors working properly:

$$\begin{aligned}\hat{\mathbf{x}}_{k+1} &= A_d\hat{\mathbf{x}}_k + B_d\mathbf{f}_k + L(\mathbf{y}_k - \hat{\mathbf{y}}_k)\\ \hat{\mathbf{y}}_k &= C_d\hat{\mathbf{x}}_k\\ \mathbf{r}_k &= \mathbf{y}_k - \hat{\mathbf{y}}_k\end{aligned} \tag{4}$$

where the $\hat{i}$ notation corresponds to the estimation of the variable i, and $\mathbf{r}_k$ is the residue, the difference between the real measurement $\mathbf{y}_k$ and the estimated (observed) output $\hat{\mathbf{y}}_k$.

So, if L is designed such as $A_d - LC_d$ is Hurwitz, and the vehicle doesn't present a failure in any rotor, the observer is

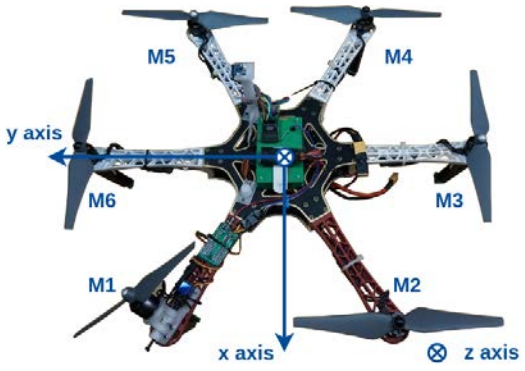

Fig. 6: Reconfigurable fault-tolerant hexarotor.

consistent with the behavior of the system, and the residue tends to zero. If a failure appears, the estimated output deviates from the measured output, which allows to use the norm of the residue as a failure detection subsystem.

In a similar way, considering a total failure may occur in motor i, the following observer can be proposed:

$$\begin{aligned} \hat{\mathbf{x}}_{k+1,i} &= A_d\hat{\mathbf{x}}_{k,i} + B_{d,i}\mathbf{f}_k + L(\mathbf{y}_k - \hat{\mathbf{y}}_{k,i}) \\ \hat{\mathbf{y}}_{k,i} &= C_d\hat{\mathbf{x}}_{k,i} \\ \mathbf{r}_{k,i} &= \mathbf{y}_{k,i} - \hat{\mathbf{y}}_{k,i} \end{aligned} \tag{5}$$

where the only difference with respect to the previous case is replacing B_d with $B_{d,i}$, which corresponds to the matrix B_d with its i-th column replaced by zeros. While the vehicle is in nominal state, the estimated output differs from the measured output, but, when a failure occurs in motor i, this observer becomes consistent and the norm of $\mathbf{r}_{k,i}$ tends to zero, identifying that this motor is the one presenting a failure. By using one of these observers for each possible failure, in this case one for each of the six motors, a total failure in any of them can be isolated. This set of observers is called a bank of observers.

V. Experimental analysis

The bank of observers developed in the previous section is a computationally demanding algorithm that requires a periodic execution in the control loop. This implies an additional load for the microcontroller in the FC.

This FDI algorithm was implemented in the Choriboard v2, which is mounted in a fault tolerant hexarotor shown in Fig. 6. This vehicle corresponds to an optimal reconfigurable hexarotor that was proposed to deal with total motor failures, in order to achieve maximum maneuverability in case of an occurrence of such a fault [21]. The control algorithm is executed at a frequency of 200 Hz, which results in $T_d = 5\,\text{ms}$. This means that every process that involves attitude estimation and control, execution of the FDI, command of the motors, and others, must be executed as a whole in less than 5 ms.

The force-torque matrix A for the vehicle in Fig. 6 is:

$$A = \begin{bmatrix} -\frac{d}{2} & \frac{d}{2} & d & \frac{d}{2} & -\frac{d}{2} & -d \\ \frac{\sqrt{3}d}{2} & \frac{\sqrt{3}d}{2} & 0 & -\frac{\sqrt{3}d}{2} & -\frac{\sqrt{3}d}{2} & 0 \\ k_t & -k_t & k_t & -k_t & k_t & -k_t \\ -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \tag{6}$$

TABLE II: Vehicle technical specs

Variable	Value	Units
d (arm length)	0.275	m
P (weight)	2.2	kg
f_{max} (per motor)	0.98	kg
$[J_x, J_y, J_z]$	[0.047, 0.047, 0.109]	Nm^2
k_t	0.03	-

where d is the distance of each motor to the geometric center, and k_t is a constant of the motors that relates the force exerted to the torque generated in the z axis. The values for these constants, as well as other characteristics of the vehicle are summarized in Table II.

Several experiments were carried out to test the performance and robustness of the bank of observers, performing flights where a fault was injected mid-flight in one of the motors. The residues were recorded in real time, and their values for some of the flights are shown in Fig. 7. In four different flights, the vehicle was driven to a hovering state, remaining in a fixed position, and a fault was injected in $t = 0\,\text{s}$ by cutting the power to motor number 3. It is shown that, for a short time of around 100 ms, the residues show no change, due to the fact that the motor-propeller set continues spinning due to inertia, and is still producing thrust. After that, the detection residue r_0 that corresponds to the observer of the nominal plant begins to increase as the predicted output deviates from the measured output, while the residue r_3 that corresponds to motor 3 tends to zero. When both r_0 is greater than a detection threshold $\tau_d = 1.0$ and r_3 is lower than an isolation threshold $\tau_i = 0.75$, the fault is detected and isolated, and the vehicle reconfigures the control system to cope with it. In the experiments, after the failure is isolated (where r_3 crosses the threshold), the vehicle changes the orientation of the rotors to deal with the occurrence of the failure, and all the observers are no longer consistent with the behavior of the vehicle. This means that the values of the residues after the failure is detected don't have a particular meaning.

From the point of view of processing power, the bank of observers showed to be very demanding. In Fig. 8, the recorded execution time of the whole control routine is shown. This routine comprises sensor reading, filtering and normalization, the computation of the error with respect to the desired attitude of the vehicle, the calculation of the maneuver needed to correct the error, the individual motor forces needed to produce the desired torque, and the execution of the bank of observers. During the first 30 s, the execution of the bank of observers is turned off, which yields an execution time of around 1129 µs (slight differences between executions of the control routine are expected due to interruptions taking place). At $t = 30\,\text{s}$, the observer bank is turned on, and the execution time increases to 1699 µs, a difference of 570 µs. The execution of the bank of observers, without considering interruptions, is performed in around 57000 clock cycles with the microcontroller running at 100 MHz, as the number of floating point operations remains constant. As the Cortex-M3 does not have an FPU, floating point operations are not efficient, and thus it takes a great

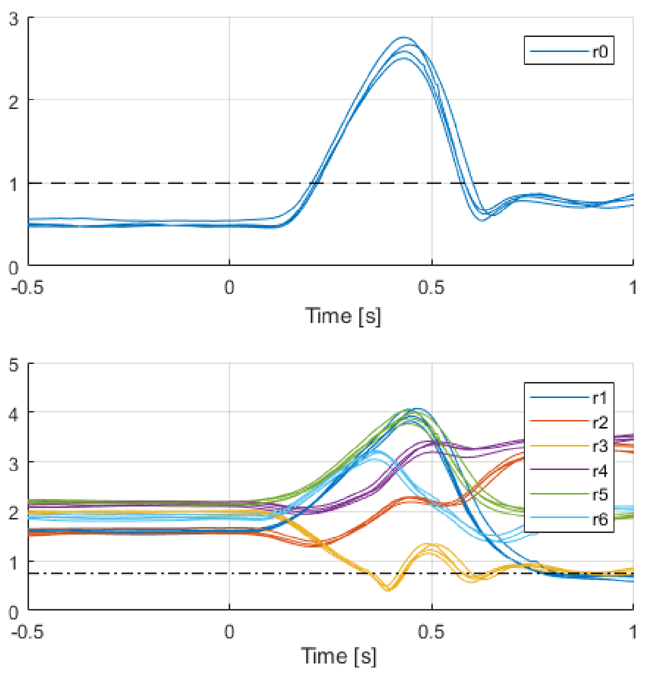

Fig. 7: Residues of the bank of observers, with the detection residue corresponding to the observer of the nominal plant (top), and the isolation residues corresponding to the observers of the faulty plants (bottom) for a hexarotor flight where a total failure is remotely injected in motor 3 in $t = 0\,\mathrm{s}$. Four different flights are shown, with very similar behavior.

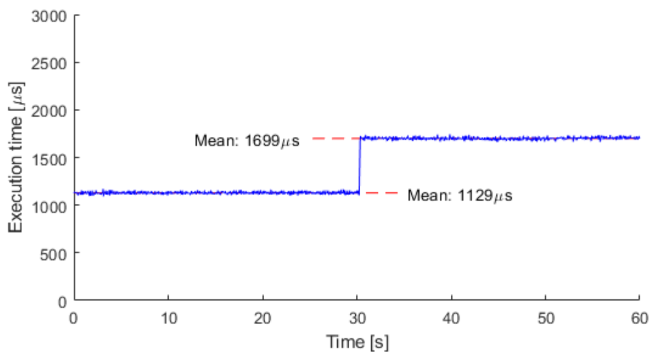

Fig. 8: Execution time of the control algorithm in the Chori-board v2, without executing the FDI subsystem ($t < 30\,\mathrm{s}$), and executing it ($t > 30\,\mathrm{s}$).

toll on the processing time [22].

In Table III, a summary of the main routines' execution time is shown. An important routine to be considered is the IMU data acquisition and processing, where, while the reading of the registers is done in negligible time, the DMP outputs the attitude in quaternion form. As the control algorithm uses the Euler angles, a conversion must take place, consisting on several floating point operations, that results in a considerable execution time. As one IMU measurement is needed for every control loop execution, this routine has to be executed inside the $5\,\mathrm{ms}$ time window. Still, the Cortex-M3 is able to execute the whole set of routines in a reasonable amount of time, guaranteeing the periodicity for the control loop.

TABLE III: Execution time of different algorithms

Routine	**Freq.** [Hz]	**Exec. time** [ms]
Estimation + control	200	1.129
Bank of observers	200	0.570
IMU acquisition	200	0.595
Message assembly	200	0.006
Magnetometer read (IRQ)	10	0.006
RC receiver read (IRQ)	600	0.002

However, as more routines are included, such as the proposed approach in [23] to improve manoeuvrability in case of failures, which adds around $0.3\,\mathrm{ms}$ to the control routine, the processing power of this microcontroller is pushed further towards its limits. One of the most common devices in commercial multirotors, both for professional and recreational applications, is the GPS receiver, that allows for outdoors positioning, and usually outputs data at a frequency between 1 and $10\,\mathrm{Hz}$. While the frequency at which the data is received may be low, the additional processing that this sensor brings is considerable. On one hand, the NMEA protocol is usually used, which provides latitude, longitude, height, speed, and other relevant data in an ASCII string form. This means that a great number of floating point operations are needed to convert them to float-type variables, which in this board results in a processing time of $225\,\mu\mathrm{s}$, not taking into account the time required for the interruption produced by each byte received that is stored. On the other hand, the raw position measurement may not be useful by itself, due to measurement errors and noise, so a sensor fusion algorithm that includes the IMU data is generally implemented by means of a Kalman filter. Moreover, to obtain good accuracy in this process, double-precision floating point operations are required, which are even less efficient in the Cortex-M3.

When implementing all this algorithms together in the Choriboard v2, timing could no longer be guaranteed, as the interruptions taking place while the FTCS was executing would sometimes extend the total execution time well beyond the $5\,\mathrm{ms}$ window, and cause the overall system to be unstable. All these reasons showed that this FC was not suitable to incorporate this kind of FTCS while still running the common algorithms of unmanned aerial systems, and brought forward the need for a new version of the proposed board that incorporates a processor with more capacity and better suited for the applications herein considered.

VI. Improved Flight Controller: Choriboard v3

The third version of the Choriboard was designed with two main objectives:

1) Reduce the size and weight of the board, for it to be used in very small air vehicles.
2) Update the microcontroller and sensors, to improve performance, and to replace obsolete and/or discontinued components.

To increase the processing power of the FC, the microcontroller was replaced with the STM32F722, a chip from the ARM Cortex-M7 family, that operates at $216\,\mathrm{MHz}$ and has an integrated single precision FPU, 512KB of flash

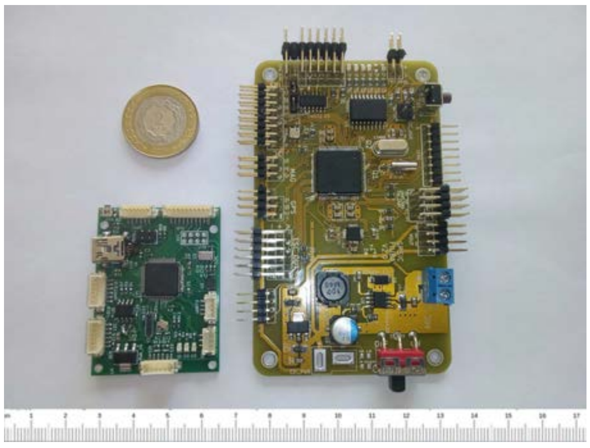

Fig. 9: Flight computers Choriboard v2 (right) and Choriboard v3 (left).

memory and 256kB of RAM; 64kB of the latter are used to store data for critical real-time tasks and another 16KB to execute these tasks. This particular chip was chosen to obtain the best balance between the type and size of the package (maximize the number of pins used, ease of manual soldering due to limited resources), computational power, energy efficiency, manufacturer documentation (Reference Manual, data sheets, specific manuals of their peripherals), availability of low-level libraries and their maintenance, and support communities.

At the time of designing the new version, the MPU6000 IMU had already been discontinued, so another device from Invensense was chosen as replacement, the ICM20602, which presented better specifications in all the relevant fields, as shown in Table I, for a direct comparison with its predecessor. An important missing aspect in the new IMU is that it does not count with the DMP feature, so the processing required to obtain attitude from raw accelerometer and gyroscope data now falls on the microcontroller.

The $2.54\,\mathrm{mm}$ pin headers were replaced by Hirose's DF13 line, surface mount models, and their pinout was designed to match those used in commercial flight controllers, so that off-the-shelf components (GPS, magnetometer, power source, and others) can be connected directly to the Choriboard. The BMP280 barometer, was replaced by the MS5611 from TE Connectivity, an I2C chip with a 24-bit ADC converter that allows to measure height with a sensitivity of up to $10\,\mathrm{cm}$.

One of the most used peripherals in a FC is the Universal Asynchronous Receiver Transmitter (UART), which is why the new version of the Choriboard has five of them. Typically, a GPS receiver, a communication module for telemetry, a high-level computer with the capacity to carry out more demanding tasks and a radio control receiver are connected using this bus.

In this version, an input for PPM-type signals is provided for the RC receiver, and an external PWM to PPM encoder can be added if required. In addition, an input for DSM signals for Spektrum RC receivers that operate through a serial UART channel is included, to directly connect this brand's Satellite micro-receivers, which is a feature also common in commercial flight controllers. All PPM and PWM related signals, including six PWM outputs for motor control are found in a single 10-pin connector, that also provides a $5\,\mathrm{V}$ output for powering the RC receivers.

The switching power supply was removed from the board, and replaced with an input connector that is powered by a $5\,\mathrm{V}$ direct voltage, which also adds analogue inputs for measuring battery voltage and current consumption.

An image of the Choriboard v3, together with the previous version for size comparison, is shown in Fig. 9, and the details of the hardware layout of the new version is shown in Fig. 10.

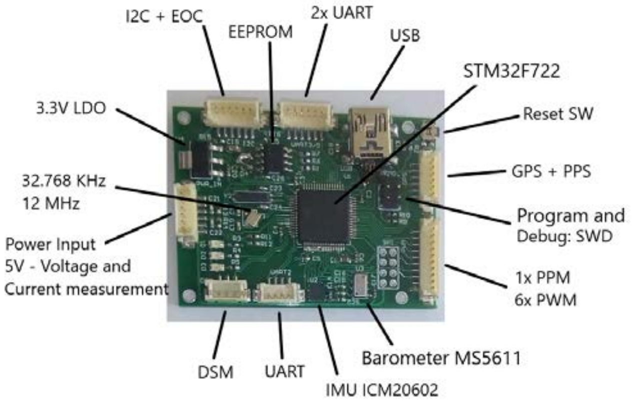

Fig. 10: Choriboard v3 hardware layout.

To compare the performance of the Choriboard v3 in the execution of the control routine, the same code implemented in the previous version was used for the control loop, only changing variables related to the use of specific resources of the Cortex-M7. As this microcontroller counts with a single-precision FPU, this kind of floating point operations are much more efficient, greatly reducing the execution time. A similar experiment to that depicted in Fig. 8 is carried out for the Choriboard v3, and is shown in Fig. 11. In the same way as before, the execution time for the full control loop is shown, for the first $30\,\mathrm{s}$ with the bank of observers turned off, leading to an execution time of $38\,\mu\mathrm{s}$, and for the last $30\,\mathrm{s}$ with the bank of observers turned on and an execution time of $92\,\mu\mathrm{s}$, a $54\,\mu\mathrm{s}$ difference. The bank of observers is executed in 11530 clock cycles in average, almost 7 times less than in the Cortex-M3 due to the use of the FPU, which results in a 14-times reduction in the total execution time when considering that the clock frequency is now more than double ($216\,\mathrm{MHz}$ instead of $100\,\mathrm{MHz}$).

On the other hand, the same routine for processing an NMEA message from a GPS receiver showed to be executed in average in $94.8\,\mu\mathrm{s}$, 2.37 times less when compared to the Cortex-M3. The reduction in the execution time in this case is mainly due to the use of a higher clock frequency (2.16 times), as the conversion of the NMEA sentence to position variables (latitude, longitude, height) requires mainly the execution of double-precision floating point operations. As the FPU in the Cortex-M7 does not support this kind of operations, they have to be emulated by software in the same way that is done in the Cortex-M3, so the efficiency in both cases is quite similar.

Overall, the more efficient implementation of the FTCS and FDI in the Choriboard v3 greatly reduces the total

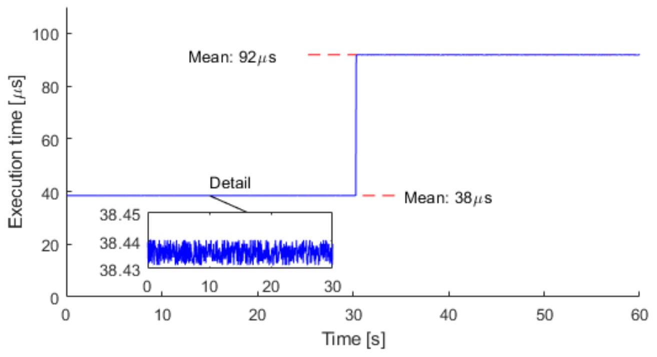

Fig. 11: Execution time of the control algorithm in the Choriboard v3, without executing the FDI subsystem ($t < 30\,\mathrm{s}$), and executing it ($t > 30\,\mathrm{s}$).

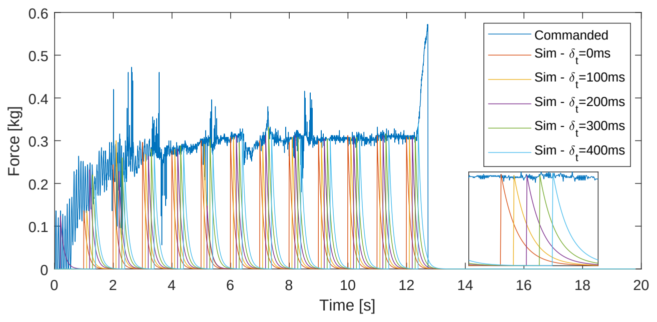

Fig. 12: Motor 3 input force in five of the observers corresponding to that failure. Every $T_c = 1\,\mathrm{s}$, and with a separation of $\delta_t = 100\,\mathrm{ms}$, each observer takes the current force commanded to motor 3 and supposes a failure occurs at that instant, with an exponential decay in the force and torque produced.

time required to execute all the algorithms involved, leaving plenty of room for other common algorithms typically present in commercial flight computers. This fact opens up the possibility to modify the design of the bank of observers to include the effects of the motor inertia that causes a delay in the failure detection. This extension will be studied in more detail in the following section. Also, the increased computing power opens the possibilities to implement FDI algorithms that are more complex in nature, such as those based on sliding mode observers [24], [25], or on machine learning methods [26].

VII. Extended bank of observers

In Section IV-B, it was shown that the bank of observers works properly and allows to detect and isolate a failure in any motor. As stated in Eq. 5, a total failure is considered, with the assumption that the motor abruptly stops. However, as can be noticed in Fig 3, there are some types of failures where the motor remains spinning for a short period of time, producing force and torque due to the inertia. This effect causes a delay in the detection, as there is not an immediate change in the value of the residues.

Then, it is of interest to design a bank of observers that takes into consideration the possibility of this type of failure, one where the motor continues spinning after the failure, which is consistent with a sudden loss of power. That is, the aim is to design observers that are consistent with the exponential decay nature of this failure when it occurs.

However, designing a consistent observer for this behavior would require to meet two conditions. First, a model for the behavior of the motor is needed, which could be obtained by means of identification tests, such as the one represented in Fig. 3. Second, it is required to estimate the exact moment when the failure occurs, which is very difficult to achieve.

One way to approximate the time when a failure is produced is to incorporate a finite set of new observers. A new observer is initialized every δ_t; at that moment, it takes the current value of the force applied to its corresponding motor and considers that it goes to zero following the characterized exponential decay. The proposition here is to place these observers in a circular queue, resetting each one a given time T_c after they are initialized, when it can be considered that the observers are not being consistent with the behavior of the vehicle. This time T_c may be defined from Fig. 3, considering the time for the motor to get to zero speed, or using practical considerations, such as a time limit to detect a failure. Moreover, in order for the circular queue to work properly, the sampling time T_c must be an integer multiple of δ_t.

The selection of T_c and δ_t imposes a trade-off between the precision in the fault detection, and the load imposed on the microcontroller. For a fixed δ_t, a longer T_c allows to evaluate the evolution of the residues over a larger interval of time, while it increases the number of observers that simultaneously run. On the other hand, for a fixed T_c, a smaller δ_t also means an increased number of observers, but enables for a more precise detection of the moment of occurrence of the failure.

In what follows, consider $T_c = 1\,\mathrm{s}$ and $\delta_t = 100\,\mathrm{ms}$, leading to ten new observers per motor. In Fig. 12 it is shown the input force for motor 3, where only the first five observers are shown for a clearer picture. Every time an observer is triggered, the current force commanded to motor 3 is registered, and an exponential decay is simulated from that point on. Fig 13 shows the detection (top) and the new failure residues along with the original total failure residue (bottom). Moreover, Fig. 14 shows in detail what happens around $t_f = 12.33\,\mathrm{s}$ when the failure occurs. The green and cyan residues, that correspond to the observers triggered closer to t_f, decay more quickly than the others. It can be noticed that for any threshold in the range $[0.04, 0.07]$ a detection time between $100\,\mathrm{ms}$ and $200\,\mathrm{ms}$ can be achieved. Therefore, the use of this solution would allow us to detect a failure in less than half the time required by the total failure observer (blue residue) used in Section IV-B.

This new bank of observers is composed of ten additional observers for each of the motors in failure, meaning that it would require a total of sixty new observers. This would be impossible to run in the Choriboard v2, as it was already very limited, however, the Choriboard v3 has plenty of resources to take care of the additional load. As the execution time of the original bank of seven observers (nominal plus six total failures) was about $54\,\mu\mathrm{s}$, the total load when adding the new ones would be of around $516\,\mu\mathrm{s}$.

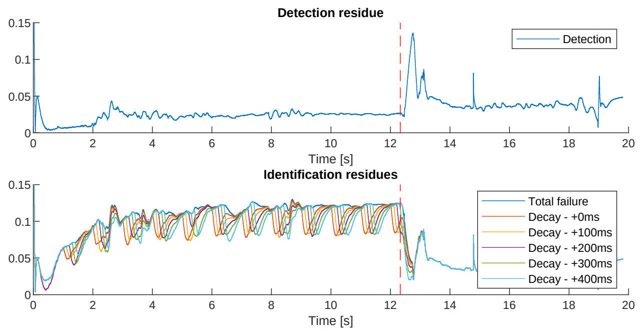

Fig. 13: Detection residue (top) and identification residues of motor 3 in the different observers (bottom), for a vehicle in which a failure occurs at $t = 12.33\,\mathrm{s}$.

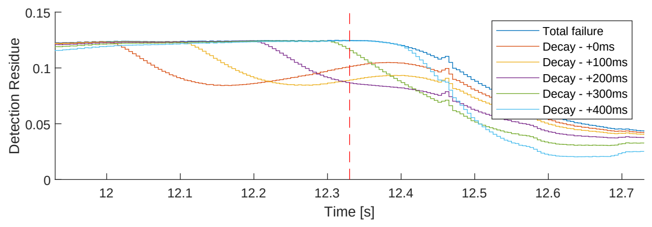

Fig. 14: Detail of the identification residues of motor 3 in the different observers, for a vehicle in which a failure occurs at $t = 12.33\,\mathrm{s}$.

VIII. Concluding Remarks and Future Work

A comparative study was conducted on the implementation of a fault tolerant control system running on two flight computers, one based in a middle-rated Cortex-M3 and a high-rated M7. This study reveals the minimum processing requirements needed to perform a flight control robust against a single rotor failure for a multirotor-type unmanned aerial vehicle.

It was shown that the additional burden imposed by the implementation of an active fault tolerant control algorithm in an existing flight computer running on the Cortex-M3 is possible, but might bring it dangerously close to control overruns. The implementation of the FDI subsystem in the custom-made flight computer consumes an additional 12% of a Cortex-M3 microcontroller resources, as it does not have an FPU. This fact pushes its processing capabilities to the limit, resulting in lack of guarantees for the algorithms to be executed with precise timing. Thus the Choriboard v2 imposes a limitation in the kind and size of algorithms for the FDI.

To overcome these issues, a new version of the flight computer is presented: the Choriboard v3, which features a Cortex-M7 microcontroller with a single-precision FPU. This unit, along with a higher clock speed, reduces the processing times of the AFCTS algorithms, allowing them to run together with all the common navigation and control algorithms present in multirotor systems.

With this upgraded version of the Choriboard, it was feasible to incorporate the knowledge about the motor dynamics in the FDI algorithm, leading to a reduction in the detection time to almost half its previous value. This fact is encouraging to continue exploring on other variations of the bank of observers.

Future development directions include studying different alternatives of the fault tolerance control routines and fault detection algorithms. Innovative AFTCs techniques based on machine-learning principles are usually expensive in terms of resources, and therefore they were prohibitive in the previous versions of the Choriboard flight controller based on a Cortex-M3, since these techniques are prone to starve available processing resources. However, the capabilities provided by the Cortex-M7 in the Choriboard v3 leaves room to consider the implementation of some of these algorithms to perform fault-tolerance control of unmanned aerial vehicles. Furthermore, a supervisory controller, with an overall view of the tasks being run on the microcontroller, could be used to solve the aforementioned trade-off between the number of observers (and consequently their corresponding computing resources) and the time needed to detect a failure.

Acknowledgments

This research was partially supported by PICT-2019-2371 and PICT-2019-0373 projects from Agencia Nacional de Investigaciones Científicas y Tecnológicas, and UBACyT 0421 project from the Universidad de Buenos Aires (UBA), Argentina.

References

[1] H. Huang, A. V. Savkin, and C. Huang, "When drones take public transport: Towards low cost and large range parcel delivery*," in *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, vol. 1, 2019, pp. 1657–1660.

[2] E. D'Amato, M. Mattei, A. Mele, I. Notaro, and V. Scordamaglia, "Fault tolerant low cost IMUS for UAVs," in *2017 IEEE International Workshop on Measurement and Networking (M N)*, 2017, pp. 1–6.

[3] A. Marks, J. F. Whidborne, and I. Yamamoto, "Control allocation for fault tolerant control of a VTOL octorotor," in *Proceedings of 2012 UKACC International Conference on Control*, 2012, pp. 357–362.

[4] M. Saied, B. Lussier, I. Fantoni, C. Francis, H. Shraim, and G. Sanahuja, "Fault diagnosis and fault-tolerant control strategy for rotor failure in an octorotor," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5266–5271.

[5] M. Saied, B. Lussier, I. Fantoni, C. Francis, and H. Shraim, "Fault tolerant control for multiple successive failures in an octorotor: Architecture and experiments," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 40–45.

[6] J. I. Giribet, R. S. Sanchez-Pena, and A. S. Ghersin, "Analysis and design of a tilted rotor hexacopter for fault tolerance," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 4, pp. 1555–1567, 2016.

[7] M. W. Mueller and R. D'Andrea, "Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 45–52.

[8] D. Vey and J. Lunze, "Structural reconfigurability analysis of multirotor UAVs after actuator failures," in *2015 54th IEEE Conference on Decision and Control (CDC)*, 2015, pp. 5097–5104.

[9] ——, "Experimental evaluation of an active fault-tolerant control scheme for multirotor uavs," in *2016 3rd Conference on Control and Fault-Tolerant Systems (SysTol)*, 2016, pp. 125–132.

[10] G. M. Mancuso, E. Bini, and G. Pannocchia, "Optimal priority assignment to control tasks," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 5s, Oct. 2014. [Online]. Available: https://doi.org/10.1145/2660496

[11] W. Koch, R. Mancuso, and A. Bestavros, "Neuroflight: Next generation flight control firmware," *arXiv preprint arXiv:1901.06553*, 2019.

[12] M. Hofer, M. Muehlebach, and R. D'Andrea, "Application of an approximate model predictive control scheme on an unmanned aerial vehicle," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 2952–2957.

[13] E. Bregu, N. Casamassima, D. Cantoni, L. Mottola, and K. Whitehouse, "Reactive control of autonomous drones," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 207–219.

[14] E. Pecker-Marcosig, J. I. Giribet, and R. Castro, "Hybrid adaptive control for UAV data collection: A simulation-based design to trade-off resources between stability and communication," in *2017 Winter Simulation Conference (WSC)*, 2017, pp. 1704–1715.

[15] A. Fekih, "Fault diagnosis and fault tolerant control design for aerospace systems: A bibliographical review," in *2014 American Control Conference*, 2014, pp. 1286–1291.

[16] G. Michieletto, M. Ryll, and A. Franchi, "Control of statically hoverable multi-rotor aerial vehicles and application to rotor-failure robustness for hexarotors," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2747–2752.

[17] J. I. Giribet, C. D. Pose, A. S. Ghersin, and I. Mas, "Experimental validation of a fault tolerant hexacopter with tilted rotors," *International Journal of Electrical and Electronic Engineering and Telecommunications*, vol. 7, no. 2, pp. 58–65, 2018.

[18] D. Vey, K. Schenk, and J. Lunze, "Simultaneous control reconfiguration of systems with non-isolable actuator failures," in *2016 American Control Conference (ACC)*, 2016, pp. 7541–7548.

[19] D. Wolfram, F. Vogel, and D. Stauder, "Condition monitoring for flight performance estimation of small multirotor unmanned aerial vehicles," in *2018 IEEE Aerospace Conference*, 2018, pp. 1–17.

[20] O. Zandi and J. Poshtan, "Fault diagnosis of brushless dc motors using built-in hall sensors," *IEEE Sensors Journal*, vol. 19, no. 18, pp. 8183–8190, 2019.

[21] C. D. Pose, J. I. Giribet, and I. Mas, "Fault tolerance analysis for a class of reconfigurable aerial hexarotor vehicles," *IEEE/ASME Transactions on Mechatronics*, vol. 25, no. 4, pp. 1851–1858, 2020.

[22] H. Smeets, M. Ceriotti, and P. J. Marrón, "Adapting recursive sinusoidal software oscillators for low-power fixed-point processors," *ACM Trans. Embed. Comput. Syst.*, vol. 19, no. 3, May 2020. [Online]. Available: https://doi.org/10.1145/3378559

[23] C. D. Pose, J. I. Giribet, and A. S. Ghersin, "Hexacopter fault tolerant actuator allocation analysis for optimal thrust," in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2017, pp. 663–671.

[24] M. Saied, H. Shraim, C. Francis, I. Fantoni, and B. Lussier, "Actuator fault diagnosis in an octorotor UAV using sliding modes technique: Theory and experimentation," in *2015 European Control Conference (ECC)*, 2015, pp. 1639–1644.

[25] F. Chen, R. Jiang, K. Zhang, B. Jiang, and G. Tao, "Robust Backstepping Sliding-Mode Control and Observer-Based Fault Estimation for a Quadrotor UAV," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 8, pp. 5044–5056, 2016.

[26] C. D. Pose, A. Giusti, and J. I. Giribet, "Actuator fault detection in a hexacopter using machine learning," in *2018 Argentine Conference on Automatic Control (AADECA)*, 2018, pp. 1–6.