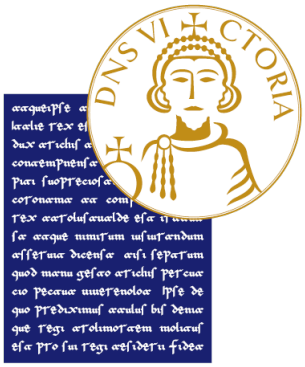
UNIVERSITÀ DEGLI STUDI DEL SANNIO

Dipartimento di Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica



A data-driven test bench for fault diagnosis of UAV propellers

Relatore
Prof. Francesco Picariello

Correlatore
Prof. Ioan Tudosa

Candidato
Antonio Pedicini

Matr. 863002919

Anno Accademico 2023/2024

A data-driven test bench for fault diagnosis of UAV propellers

Antonio Pedicini
Student

Prof. Francesco Picariello
Advisor

Prof. Ioan Tudosa
Co-Advisor

Università degli studi del Sannio

31 October 2024

Dedication

To my high-school mathematics and physics professor,
Prof. Cataudo.

Abstract

Our society has entered the fifth industrial revolution as now, more than ever before, companies implement decision-making processes that are based on the so-called Big Data. Industrial plants gradually integrate sensors that collect data during the whole production process giving the opportunity to be business competitive in every step. Among the jobs of the third sector human operators are being replaced by the more efficient autonomous systems, that do not require any human intervention to complete the task assigned to them. In this scenario, autonomous drones driven by artificial intelligence will be deployed to accomplish tasks such as transportation, package delivery, first-aid in rescue missions, and many more, as some of these jobs already exploit the benefits of using such systems. At the same time, given the increased complexity of these systems, these systems have to be designed in order that their missions cannot threaten or harm uninvolved and involved humans. For this reason, several methods that study Fault Tolerant Control (FTC) have been proposed in the literature and employed successfully. In this study we take on the first step in every FTC method: Fault Detection & Diagnosis (FDD). In particular, a data-driven testbench is proposed, that, contrary to the majority of laboratory stands previously proposed in the literature developed by the researchers community in this field, does not require testing UAV's components separately, but is able to test the system as a whole, exploiting the power of statistical learning, also known as Machine Learning, to classify between several kinds of faults on UAV propellers. Having first introduced the general theory behind the FDD method, some case studies will be presented to describe the state of the art on these methods and their many variants. Then, the proposed testbench will be described in its hardware, firmware, and software components. Finally, the experimental results will be discussed so that they can be compared to other studies' results and some directions for the development of future works will be given.

Acknowledgements

I want to thank Prof. Picariello and Prof. Tudosa.

This whole project is Prof. Picariello's brilliant idea, and without his constant support and valuable insights, this work would have never come to an end.

Prof. Tudosa, made me appreciate the wonderful world of electronics, and, if I chose to work on this project, it is mostly because of his guidance.

I have to thank each and every one of the researchers who worked in the laboratory where test flights were carried out for the patience they demonstrated despite the strong noise produced by the UAV under test.

I'd like to acknowledge my father for his assistance and creativity in the construction of the equipment used during the calibration procedure.

Finally, I want to thank each family member and friend who showed his/her support in the time period that preceded the final dissertation.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Scope of the Work

This work describes a solution for the detection and diagnosis of faults generated in the propulsion system of a lightweight UAV, specifically in the context of propeller faults. The proposed solution is based on machine learning techniques, and, therefore, can be classified as a data-driven method, according to the surveys that review the main methods used for fault detection of machinery systems. Several systems have been developed to prevent fault conditions by means of machine learning algorithms, exploiting the characteristic trait of industries in the era of the fourth industrial revolution: the datafication of virtually every step of production processes. A Fault Detection (FD) algorithm should be able to detect and identify faults and is considered the first step of Fault Tolerant Control (FTC) algorithms. This procedure is also common in the context of Predictive Maintenance, which, as opposed to preventive or corrective maintenance, is performed following a prediction derived from repeated analysis or known characteristics of the component of the system. The main advantage over non-predictive maintenance is the opportunity to be able to analyze trends of data collected and thus build an evolution model of the system based on experience acquired over time. A physical system over time can be subject to failures of various kinds, and, therefore, it is necessary to introduce a fault detection system. The failure is both an event and a state that causes the entity that it affects to not work at all. Depending on the technology involved, failures can be classified as mechanical or electrical. Failure can be caused also by an incorrect design of the control algorithm that governs a given system. The events that cause the system to deviate from its expected behaviour, but that do not evolve into breakdowns, are called faults. Faults can be permanent, non-permanent or transient. In the latter scenario, faults appear only in conjunction with certain events or environmental conditions. Moreover, it is necessary to specify the nature of the failure, which can be split into systematic or non-systematic failure. In the case of Unmanned Aerial Vehicles (UAVs) faults usually originate from actuator or sensor malfunctioning. The appearance of fault within these systems is almost unavoidable. This is primarily attributable to the fact that UAVs embed a variety of subsystems, sensors, actuators and components that are susceptible to failures. UAVs can be classified according to weight, mechanical functioning, and other similar metrics. The most popular characterization distinguishes these devices as rotary wing, fixed-wing and flapping wing. The latter is certainly the most unusual, while the second kind of UAV is mostly used in military applications. Rotary Wing UAVs, instead, are used in several circumstances, especially for recreational purposes. Each kind of UAV is subject to different restrictions, and each has its advantages and disadvantages in terms of performance. Because these systems are widely used even in populated areas, they have to conform to some kind of safety system. Nowadays, it is often required to adopt a fail-safe subsystem, that monitors the evolution of the system instant by instant, and whenever a fault is detected some safety procedures are executed, allowing, in the worst case, to shut down the system gracefully. The most popular method used to detect a fault in a UAV remains the model-based approach. This approach benefits from all the knowledge that has been acquired in the aviation industry. Nevertheless, with the advent of the fifth industrial revolution, and the exponential increase in the application of Artificial Intelligence (AI) algorithms, researchers are focusing now on leveraging the powerful capabilities of machine learning algorithms to avoid the use of complex mathematical models, that are inappropriate when the model is too complex or not accurate enough due to oversimplifications made in the modelling

phase. Usually, each component of the system is tested for faults separately, and the result is thus independent of the other components of the UAV. By far, the most delicate components of all in a UAV system are its rotors. These are often made with special materials that are analysed in depth with computational software that studies their effect on the efficiency of the aircraft. Given that UAVs are often employed in dangerous environments and situations, numerous events could affect the correct functioning of the motor-propeller pair. Several testbenches have been proposed to assess the performances of UAV propellers. These testbenches consist of laboratory stands that, by means of a test stand equipped with force and speed sensors, are able to measure the thrust and rotational speed generated by the propulsion system. None of these systems has ever been applied with the specific aim of fault detection. Moreover, none of these systems can account for the interaction of the faulty component with the other subsystems, because the motor-rotor pair is tested separately from the rest of the UAV. The proposed solution in this work is one example of what a full-system testbench could look like. The remainder of the chapter will introduce the historical context that demands the design and application of fault detection systems to ensure the reliability of UAVs, which are considered to be extensively used in the near future.

1.2 The intersection between AI and UAVs

Unmanned Aerial Vehicles (UAVs) and Artificial Intelligence (AI) modern technologies were conceived in the early and mid-20th century, respectively. It is plain to see how UAVs and various industries have largely benefited from AI infusion. As reported in Obiuto et al. [2024], in the early 2000s, AI's role in UAV development was limited to enhancing their efficiency and performance and improving navigation and control. Still, in the mid-2010s, as the first application of Machine Learning (ML) techniques and Neural Networks (NN) into UAV systems took place, the possibility to analyze sensor data, recognize patterns, and thus make important decisions in real-time without constant human intervention, became a reality. In the same timeframe, UAVs started integrating sophisticated payloads, such as thermal cameras and advanced sensors. Since 2018 the cost to train image classification algorithms decreased by 63.3% while training times have impressively improved by 94.4% Zhang et al. [2022]. Computer vision technology allows UAVs to perceive and interpret their surroundings, and thus detect and avoid obstacles in unpredictable environments. Moreover, UAVs can now benefit from predictive maintenance based on machine learning, enhancing their reliability, preventing failures, and ensuring that drones do not have the potential to threaten involved or uninvolved people. AI is now helping to shift the level of industrial automation, as the fifth industrial revolution era just started. Industry has traditionally dominated AI research producing 2023 51 notable machine learning models Maslej et al. [2024]. Now that new complementary technologies have started to arise, industries and public organizations are starting to adopt autonomous aerial systems (AAS) more often in their business tasks. Some examples are the role that UAVs will have in Urban Air Mobility (UAM) and the 6th generation of wireless communication networks. The first has been foreseen for a while now. The European Union Aviation Safety Agency (EASA) points out eas that soon there is the potential for travelling in densely populated areas from one place to another using remotely piloted aircraft which have no pilot on board. These vehicles are often called 'airtaxis', and a concept of these aircraft is shown in 1.1.

Figure 1.1: A concept of an airtaxi

While attractive, as of today, this vision is mystified by the fact that UAVs are constrained in terms of battery endurance, flight autonomy and limited flight time to perform persistent missions. To overcome these limitations, urban development and mobility planning are needed. Vertiports,

platforms much like helipads, under construction in many regions of the globe, could evolve into something akin to bus stations or subway platforms, making air travel within a city more accessible to the general public. These platforms could offer services such as recharging and, more importantly, maintenance proceduresver. In the case of 6G JIANG et al. [2022], which is supposed to be an all-covering network, providing ubiquitous connections for space, air, ground and underwater, UAVs could provide airborne wireless coverage flexibility, serving as relays to connect isolated nodes. Moreover, these systems could be used for data offloading means, and their mobility can be exploited to move closer to the target user to achieve better channel gain and avoid obstacles. This scenario undergoes the same limitations as in the case of UAM. Indeed, it is essential to provide enough lifetime and low power consumption, as these systems should be able to provide energy for communication and propulsion simultaneously. Furthermore, UAV-to-UAV networks are known to present some challenges in the form of the kind of new protocols that should be adopted. Indeed, in general, continuous end-to-end connectivity cannot be guaranteed as transmission delays could extend over TCP threshold limits and several packets could be dropped. Despite the challenges, several protocols have been proposed such as proactive protocols Hentati and Fourati [2020]. To enhance the lifetime of UAVs new fueling solutions, such as hydrogen fuel cells and wireless power transfer (WPT) have been already developed, and UAV applications continue to expand Mohsan et al. [2022].

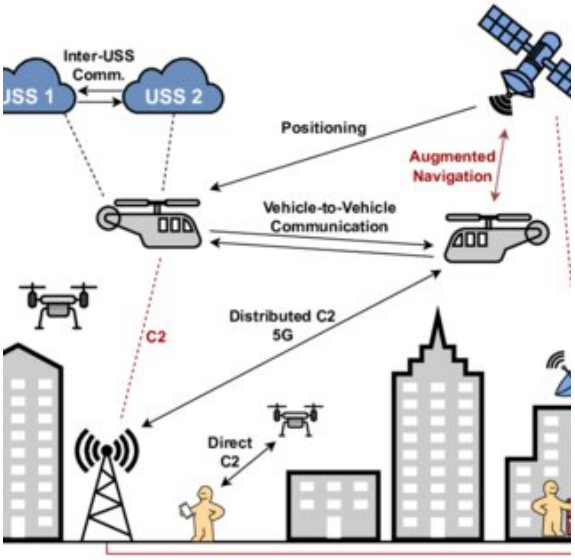


Figure 1.2: Urban aircraft system communication and data links, credits:Lukic and Durak [2023]

Making these aircraft systems reliable is essential. It is easy to see how potential injuries could arise by using UAVs in the contexts that have been introduced. Indeed, the development of such technologies has generally been limited by the general public acceptance and awareness. In 2019, EASA produced an international set of regulations to help standardize the legal restrictions on the use of UAV systems . In particular, this brought new criteria of classification for UAV devices, now labelled between the Open category for low-risk operations, Specific Category and Certified categories for more dangerous operations that require skilful personnel. In general, for leisure purposes, UAVs are restricted to flying beneath 120m, limiting collision with manned aircraft systems, which occupy the airspace above 150m. All the UAVs that fall under the Open category classification, have a MTOM less or equal to 25kg. A common requirement for these systems is a program for regular maintenance to counteract the degradation of motors and other components. To this end, the solution provided in this work could be adopted for this class of vehicles, also previously known as small or lightweight UAVs. In particular, the system is based on data-driven Fault Detection and Diagnosis (FDD).

1.3 Fault Detection and Diagnosis methods

The field and applications of Fault Detection & Diagnosis (FDD) methods have been studied over the last forty years, and progress has been made thanks to the development of numerous techniques that range from analytical models to statistical models, and that, more recently, take advantage of powerful deep neural networks, to solve the general problem of a procedure, which embraces three steps: Fault Detection, Isolation and Identification. These are sometimes mixed together but more in general executed as separate steps, that yield an outcome that states whether and when a fault happened, at what level and of what kind. In general, Fault detection & diagnosis methods assume the system fault distinguishability, which allows the fault to be detected and to be discriminated against. Few textbooks are available that cover FDD data-driven methods. The reason for this can be given in terms of the fundamental properties that a system should guarantee at all times: safety, reliability, availability and dependability. Model-based approaches in FDD methods continue to be the more reliable choice. The reliability of a system is essential in the aeronautical field, where avoiding false positives is a priority. False positives could trigger Fault Tolerant Control (FTC) safety procedures that are best avoided when unnecessary. Nevertheless, what drives researchers to test the potential of the so-called soft-computing and statistical methods is the massive impact that this technology already has on our society, as discussed. Within public, scientific, military, and recreational spheres, autonomous systems based on artificial intelligence (AI) accomplish actions that humans are often no longer active participants in. For the most part, these kinds of systems rely only upon algorithms that process data and make decisions based on that data, extracting the most relevant pieces of information. The automotive industry is a clear example of this phenomenon. In the future, self-driven cars are expected to take over manually driven vehicles. In the past, international projects began with the aim of doing research on fault detection for aircraft such as NASA's System-Wide Safety (SWS) Project, DARPA's Assured Autonomy and the ReMAP (Real-time Condition-based Maintenance for Adaptive Aircraft Maintenance Planning) project, funded under the Horizon2020 program. The latter focused on FD methods involving data-driven probabilistic algorithms for aircrafts. One of the things that were addressed during this project was the possibility of providing real-time fault identification systems that could rely primarily on the sensors installed on board and enable to detection of faults mid-flight using embedded systems, reducing, at the same time, the bandwidth that would be required for the aircraft to communicate to a tower station its health status, where prognostics procedures would be conducted and the result of this analysis would be returned to the embedded system. Nothing better explains why scholarly articles are focusing on developing real-time FDD data-driven methods. Some of these case studies are discussed in Chapter II. These projects also solved in part one of the key limitations in any ML application: the absence or incompleteness of datasets. As an example, NASA promoted a collection of data sets that have been created to be donated to universities, agencies and companies that can be used for the development of prognostic algorithms. These datasets consist of time series that acquired measurements from a nominal state to a failed state of various systems. Aircrafts benefit from an ever-increasing number of sensors that could be used to generate datasets of acquired measurements during flight. One peril in this scenario is the possibility of the phenomenon of the curse of dimensionality, which occurs when the dimensions of the dataset exceed a certain extent. This could be addressed by using techniques such as sensor data fusion. In the case of UAV systems, several steps forward in this direction have been made. One example is the BASiC dataset (Ahmad and Akram [2024])which comprises 70 autonomous flight data, spanning over several hours, in which data of pre-failure and post-failure measurements are available. These datasets comprise different kinds of sensor failure types, covering the global positioning system (GPS), accelerometer, gyroscope, barometer and others. At the heart of UAV failures, sensor faults are one of the most well-known causes, and their consequences have been considered in detail. The remainder of this section introduces relevant terms within the realm of fault detection.

The general problem of fault detection

Avoiding false positives in the aeronautical field is essential, as false positives trigger safety procedures that must be avoided when unnecessary. The procedures are controlled by a Fault Tolerant Control (FTC) system. The first step in every FTC task is to detect and diagnose the fault.

A fault in a dynamic system is a deviation from the nominal situation from the system structure or the system parameters. Structural changes can be seen as the loss of a sensor, the lock in place

of a sensor, or the loss in effectiveness of an actuator. The result of these faults causes deviations from the expected behaviour of the system, which is usually predicted using a model. Faults are different entities concerning disturbances or model uncertainties. Indeed, faults are different in the sense that they can occur, while model uncertainties and disturbances are considered to be always present. Moreover, disturbances can be compensated using filtering, for example, but faults cannot be compensated using fixed controllers.

The general task of Fault Diagnostics is composed of three steps: 1. Fault detection 2. Fault Isolation 3. Fault Identification and Estimation The first step states whether or not a fault happened. This step also determines the time instant at which the system is subject to some faults. The second step finds which component is faulty. Generally, three classes of faults are distinguished: actuators faults, process faults, and sensor faults. The third step consists of assessing the severity of the fault and the other components that it affects. Therefore, the object of fault diagnosis systems is to determine the location and occurrence of faults based on accessible data and knowledge about the behaviour of the diagnosed process, for example, by using mathematical models. This is the case of the so-called model-based FDD methods.

Model-based methods

Model-based methods were introduced to replace the inefficient hardware redundancy methods, which were too expensive and could not be used where space limitations were imposed. Model-based methods therefore use analytical models to evaluate residuals. There are many variants of these methods, the main of which are State Observers and Parameter Estimation. State Observers include the Luenberger type of estimator, the famous Kalman filter, and others. They are all based on the fact that the residuals are generated using an estimate of system outputs. The applications of observers in fault detection and identification systems differ from the general application of these concepts as only the output vector is estimated and not the state vector as in general happens. The parameter Estimation task consists of detecting a fault in a system by measuring the input and the output and then estimating the parameters of the model of the system. Although these methods achieve their task successfully, for a large class of systems they cannot simply be applied to the fact that the models that they are based on have to approximate the real system that is being modeled. Therefore, whenever the system is too complex to be modeled, or approximations made in the modeling phase are not acceptable other alternatives are explored. One such example is the so-called Soft-Computing approaches, which include Neural Networks and Fuzzy Systems.

Neural Networks

Artificial Neural Networks (ANN) are processing systems that consist of a number of interconnected components called neurons. Each neuron can be considered as a mathematical function. The network topology defines the capacity of a neural network to learn from its input data. In principle, there are infinite forms of network topologies, and larger and more complex networks are capable of identifying more subtle patterns. One of the main reasons for ANN usage is that such structures can represent non-linear systems without the need for complex analytical models as in the case of model-based approaches. To do this, the network is structured in a number of layers: the input layer, the hidden layers, and the output layers. The input layer receives unprocessed signals. The number of input nodes is given by the number of significant characteristics that describe the input data. The input data is used together with a bias. The bias enables the network to fit the data better by shifting the activation threshold. Indeed, the neurons are activated according to some mathematical function. These functions are usually non-linear, and when the input exceeds a determined threshold the neuron is said to be activated. The number of outcomes is given by the number of outcomes yielded by the classification algorithm. The size of the hidden layer can generally vary. The training procedure of an ANN consists of determining the value of some weight coefficient values between the neighbouring processing units. One popular choice for this task is the Back-Propagation algorithm. This iterative algorithm is based on the minimization of the sum squared errors using the gradient descent method. The main drawback of such structures is the fact that they are considered to be 'grey boxes' meaning that, although the general mechanism of the learning process they undergo is known, there is no evidence of the kind of relationships/links that the network establishes between inputs and outputs during its execution. Therefore, to overcome these disadvantages, new techniques such as Explainable AI, have been developed to explain how these networks work.

Fuzzy Systems

Knowledge-based approaches are another alternative to model-based approaches. They are usually implemented as Fuzzy systems. Fuzzy systems are the antipodes of neural networks. The latter used essentially quantitative information, fuzzy systems rely on rules that are represented in a form that is close to that used by humans. These rules are also called heuristic rules which perform a mapping between data abstraction and solution abstraction. The rules often lack mathematical rigour, and therefore, fuzzy systems are seldom used in practice. Nevertheless, they reflect the operator's experience, rather than a theory of how the system under diagnosis works.

Statistical Learning

Statistical methods are part of the so-called data-driven methods. In particular, these methods rely on the application of statistical analysis of the raw data that are collected within the datasets, to reveal the underlying relationships between the input and the output. The term statistical learning is nowadays replaced by machine learning. The starting point of any learning algorithm is a set of data. Datasets can be represented as a set of input-output pairs, where the inputs are vectors of dimension equal to the number of features, whereas the outputs are usually scalar numbers. The objective is to learn the underlying relationships between the inputs and the outputs, in terms of a function f(x), whose dependent variable is the input vector, or sample. The output of the function is the corresponding output value, also known as the class label. Machine learning can be implemented through Supervised learning, Semi-supervised Learning, Unsupervised Learning and Reinforcement Learning.

Supervised Learning

Supervised Learning refers to the type of learning where all the available data have been labelled with class labels. Therefore, the classification algorithm is presented with a subset of the whole dataset that includes the input and output values. The input variable is called the independent variable (or features) and the output variable is called the dependent variable (or label).

The algorithm, once the training ends, should be able to classify never-before-seen data samples where the class label is missing. It can be easily understood that the actual representation of each pattern in the computer affects the classification algorithm and determines, also, the kind of algorithm that can be used.

Semi-Supervised Learning

Semi-supervised learning is a variant of supervised learning where the algorithm is trained on data samples where the class label is missing in some cases.

Unsupervised Learning

Unsupervised Learning aims to discover the underlying model within a given dataset without any knowledge of class labels. One such example is the clustering approach, and one of its implementations is the K-means clustering which is described as an iterative process that minimizes the sum of distances between the data points and their cluster centroids, chosen at random at the start of the algorithm. The algorithm groups data into clusters by using a mathematical distance measure from the centre of the cluster.

Reinforcement Learning

Reinforcement learning attempts to learn sequences of action by trial and error which maximize its objective function. Optimal actions are not learned from class labels but from what is commonly known as a reward. The use of reinforcement learning in fault detection is still not very common.

1.4 Lightweight UAV architecture

If the approach on which the solution proposed in Chapter III had been based on an analytical model, a thorough description of a Unmanned Autonomous Vehicle (UAV) system would be required. As the test and methods performed in this work belong to the realm of data-driven

methodologies, which can be considered the direct opposite of mathematical models, the amount of know-how needed on UAVs is limited to the electronic system commonly found on board for off-the-shelf commercial devices and the motor-propeller subsystem, that will be used to acquire measurements which evaluation and analysis will reveal the occurrence of faults.

When used for civilian applications, lightweight UAVs are classified as those that have an MTOW which ranges between 5-50kg and a maximum range of 70km. The safe and efficient operation of a UAV requires additional elements, i.e. a ground-based controller (components that control the unmanned aircraft) and a related communications link (between the UAV and controller). The UAV, controller and system of communication form together an unmanned aircraft system (UAS).

Whatever the case, lightweight UAV architecture usually comprises the following components:

- electric propulsion system (battery, ESC, BLDC motors, propellers)
- a GPS module
- an IMU module
- a controller (and its relative software)
- a mean for remote connection

The Global Positioning System (GPS) module is used to acquire the position of the UAV for navigation purposes. When the GPS of a UAV fails, no positioning information and no navigation are available.

The Inertial Measurement Unit (IMU) is composed of a three-axis accelerometer, three-axis gyroscope and three-axis magnetometer, and is used for stabilizing the device while moving through the air. If the accelerometer fails while measuring the orientation of the UAV for stabilization during the flight, the result is flight destabilization.

The controller usually consists of a microcontroller, that runs control algorithms on a given chip. The software that is run generally on a lightweight UAV can vary. Nowadays coding of embedded software programs for UAVs is manual and ad-hoc. For example, there are no standard autopilots. One of the most used is ArduPilot, which uses a cyclic execution model that can only run on one computing core. The autopilot software architecture is usually composed of two layers:

1. the navigation loop layer
2. the attitude regulation layer

The first layer computes the desired trajectory of the device and transforms this objective into an attitude forwarded to the attitude regulation layer. The attitude of the system can be expressed with a triple: roll, pitch and yaw. Respectively, these quantities describe the movement of the multirotor in the three degrees of freedom (DOF).

The desired trajectory is communicated via Pulse With Modulation (PWM) signals, which are yielded either by a radio controller (in the case of ground station communication) or by an autopilot. Autopilots allow the UAV to achieve either two levels of autonomy. In the case of fully autonomous UAVs, these devices can carry out delegated tasks without human interaction, and all the decisions are made on board based on sensor observations and environmental changes.

In the case of semi-autonomous UAVs, a human operator is needed to provide a set of waypoints that the device has to follow during its mission (task).

A fully autonomous UAV has to rely on several sensors that allow for object detection and collision avoidance. These instruments comprise imaging and non-imaging payloads. The main limitation to instrument deployment for the lightweight class of UAVs is weight. Most classes, indeed, are limited to carrying low-weight payloads. Limited payload size restricts users to simple sensors with few moving parts.

Non-imaging sensors are usually triggered remotely or set to record data at regular time intervals during flights. An example of this class of sensors is the Light Detection and Ranging(LiDAR). A LiDAR is classified as a non-imaging sensor because it primarily focuses on measuring distances to generate spatial data rather than capturing images. Its strength lies in creating detailed 3D representations of the environment, which is widely used in applications like topographic mapping, forestry, urban planning, and autonomous vehicles.

Imaging sensors, instead, consist of RGB or thermal cameras.

These sensors are used in a variety of applications, that range from military applications, wildlife monitoring, precision agriculture, and other leisure activities.

The electric propulsion systems will be now explained in depth, in order to grasp the dynamic forces that are essential in the functioning of the proposed solution in this work.

Electric Propulsion System

The electric propulsion system (EPS) in a lightweight UAV converts electric energy into an aerodynamic force, called thrust. Its function is in some sense opposed to that found in wind turbines which capture the kinetic energy of the wind and convert it into mechanical energy, which is then converted into electrical energy by a generator.

The EPS system is usually based on an electric battery, that transfers power to each of the Electronic Speed Controller (ESC) modules, which in turn control the motor speed according to the percentage duty cycle of the PWM control signals received either from a onboard radio frequency receiver or an auto-pilot system. The power generated by each ESC unit is transferred to each Brush Less Direct Current (BLDC) motor. This unit converts the electrical power to a torque (M) on its output shaft at a specific angular velocity. Opposed to conventional DC motors, a BLDC motor is composed of a stationary armature and magnetic components that move and thus generate a permanent magnetic field. BLDC motors are one of the key components that fueled the rapid growth of the drone industry. Its advantages over conventional direct current motors (that are built with brushes) are:

1. high power-to-weight ratio
2. durability
3. reduced noise
4. reduced maintenance cost

Finally, the last element involved in the electric propulsion system is the propeller. This is by far the most significant component that affects the performance of the overall system. The propeller converts its rotational speed into an aerodynamic thrust. Its functioning is based on two fundamental laws: Newton's third law of dynamics and the Bernoulli Principle.

The propeller moves a volume of air defined by the diameter of the propeller and the theoretical distance the propeller would advance in one second, referred to as pitch.

Propellers can be built using different materials, each with its advantages and disadvantages. Usually, carbon fibre is considered the most efficient material for propellers.

Another factor to consider is the number of blades of the rotor. The most efficient configuration is the two-bladed propeller, whereas the three or four-bladed propellers are less efficient due to the fact that multiple blades disturb the air that the trailing blade is entering.

In a recent study, it was found that the smaller the number of rotors, the better the efficiency of the vehicle. On the other hand, the achievable dynamics, and thus the manoeuvrability of the aircraft can be enhanced by a large number of propellers. Moreover, a study conducted on hexacopters, UAVs that have six rotors, found that this configuration, when one of the propellers is damaged, can develop up to 66% per cent of the maximum thrust when all propellers are healthy, a result that is unattainable with other configurations, such as quadcopters or octocopters.

The aerodynamical forces and moments on the rotor's blade can be determined by splitting the rotor blade into multiple sections each with its chord. Each section is associated with lift and drag forces. Usually, three levels of abstraction can be used to describe these forces. In the first case, the thrust and torque are calculated by multiplying some thrust and torque coefficients by the rotational speed of the rotor. This is a simplified method that can be used in experimental test for easiness of use.

This method uses dimensionless coefficients for thrust (C_T) and torque (C_Q) to represent the aerodynamic performance of the propeller:

$$T = C_T \cdot \rho n^2 D^4 \quad [\mathrm{N}],$$

$$Q = C_Q \cdot \rho n^2 D^5 \quad [\mathrm{Nm}],$$

where T is the thrust, Q is the torque, ρ is the air density, n^2 is the is the rotational speed, and D is the propeller's diameter. The method encapsulates the complex details of the blade's aerodynamics into empirical coefficients that represent the propeller's overall performance. These coefficients are obtained from experimental data or theoretical models and allow for quick estimates of thrust and torque. Since C_T and C_Q are dimensionless, they allow for easy scaling of the propeller's performance for different sizes and speeds. This makes the method well-suited for preliminary design and performance scaling.

The second level of abstraction uses momentum theory. In this case, the force (thrust) is expressed as the change in momentum of an object with a change in time, just as in Newton's second law of dynamic.

Thrust definition using momentum change is expressed by:

$$F = \frac{(mV)_2 - (mV)_1}{t_2 - t_1} \quad [\text{N}]$$

This equation expresses thrust as the rate of change of momentum. It calculates the net change in momentum of a mass of air being accelerated by the propeller or engine over a time period. Here, $(mV)_2$ and $(mV)_1$ are the momentum at the exit and the entry of the control volume, respectively. For a moving fluid, the important parameter is the mass flow rate. Mass flow rate is the amount of mass moving through a given plane over some amount of time. Its dimensions are mass/time (kg/sec), and it is equal to the density (r) times the velocity (V) times the area (A). Mass Flow Rate is calculated as:

$$\dot{m} = \rho \times V \times A \quad [\text{kg/s}]$$

where $\dot{m}$ is the mass flow rate (mass per unit time). ρ is the air density, V is the velocity of the flow, and A is the cross-sectional area through which the air passes.

This equation is used to calculate the amount of air being accelerated by the propeller per unit time. Thus, the general Thrust Equation in momentum theory is expressed by the following:

$$F = \dot{m}_e v_e - \dot{m}_0 v_0 \quad [\text{N}]$$

This equation calculates the net force (thrust) generated by the acceleration of the air mass. $\dot{m}_e v_e$ represents the momentum at the exit (engine exhaust or propeller wake), and $\dot{m}_0 v_0$ represents the momentum at the entry (incoming air). When air density and velocity are constant across the propeller disk, this can simplify the mass times the change in velocity, yielding a force that accelerates the air. There is an additional effect which we must account for if the exit pressure (p) is different from the free stream pressure. The fluid pressure is related to the momentum of the gas molecules and acts perpendicular to any boundary which we impose. If there is a net change of pressure in the flow there is an additional change in momentum. Across the exit area we may encounter an additional force term equal to the exit area (A_e) times the exit pressure minus the free stream pressure. The most general thrust equation is then given by:

$$F = \dot{m}_e v_e - \dot{m}_0 v_0 + (p_e - p_0) A_e \quad [\text{N}]$$

The third method includes the momentum theory but goes much more into detail about the aerodynamical forces that are generated on each element of the blade of the rotor. It is referred to as Blade Element Momentum Theory (BEMT). BEMT is a more precise method and is typically used for designing the optimal propeller for a specific model of UAV.

The figure shows the decomposition of forces that act on the surface of a propeller's blade 1.4.

The blade of each propeller is an airfoil, which is a shaped surface that produces lift and drag when moved through the air. When an airfoil is located in a fluid, the first point that hits the wind is called the leading edge, whereas the last point is referred to as the trailing edge. The straight line that connects this to points is called a chord. At the leading edge, the incoming flow is split into two streams: one over the top surface of the wing and the other along the bottom surface. The flow over the wingtop surface will experience an increase in velocity and a corresponding decrease in pressure. This difference will produce a net force on the wing called lift. The rotation of the blade, thus, generates a thrust directed towards the vertical axis of rotation, that, according the Newton's third law of dynamics, allows the drone to overcome gravity during takeoff.

Each motor-propeller generates a force couple that has to be balanced by another opposed motor-propeller pair, explaining why, usually, the number of rotors is an even number.

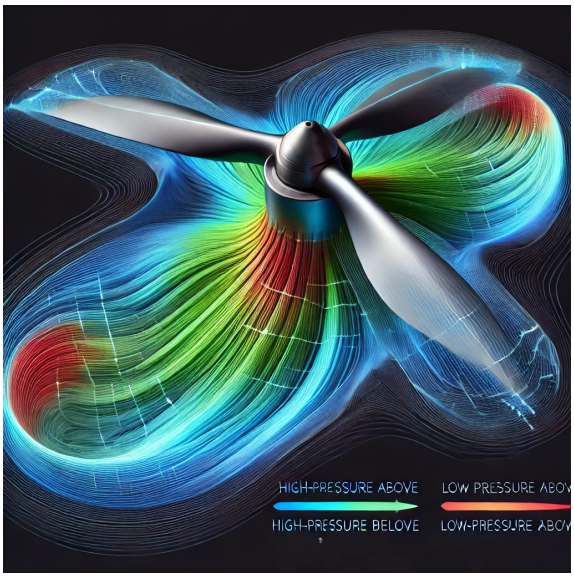


Figure 1.3: The upper and lower difference of pressure generated by the propeller rotation in the air. Credits: This image was created using "DALL-E, an AI image generation tool by OpenAI."

The blade of the propeller can be divided into infinite sections Zhu et al. [2021]. These are continuous elements, and, therefore, summing the aerodynamical forces on each, gives the aerodynamical force of the whole blade.

As shown in Figure 1.4, W, the relative air velocity, can be calculated according to the following equation:

$$W = (V_{\text{up}} + V_i) + \Omega R \quad [\text{m/s}],$$

where V_{up} is the speed at which the UAV takes off vertically, V_i is the induced velocity, Ω is the rotor angular velocity, and R is the propeller radius. The lift, C_l, and drag coefficients, C_d, are needed to calculate the lift and drag generated by blades according to blade element momentum theory. They are defined as

$$C_l = \frac{Y}{1/2\rho W^2 S}$$
$$C_d = \frac{X}{1/2\rho W^2 S}$$

where ρ is the air density, S is the effective area of the blade element, Y is the airfoil lift, and X is the airfoil drag. These quantities are dimensionless and they account for the aerodynamic properties of the blade shape (airfoil).

The net force on the blade element along the rotation axis, $\mathrm{d}T$, the net force on the blade element along the vertical rotation axis, $\mathrm{d}Q$, and the lift force, T, and drag force, Q, generated by a single blade can be calculated as

$$\mathrm{d}T = \cos(\varepsilon)\mathrm{d}Y - \sin(\varepsilon)\mathrm{d}X \quad [\text{N}],$$
$$\mathrm{d}Q = \sin(\varepsilon)\mathrm{d}Y - \cos(\varepsilon)\mathrm{d}X \quad [\text{N}],$$
$$T = \int_{R_0}^{R} \mathrm{d}T \quad [\text{N}],$$
$$Q = \int_{R_0}^{R} \mathrm{d}Q \quad [\text{N}].$$

By default, the rotor has a symmetric mass distribution with its centre located on the axis of rotation. The centrifugal forces induced by rotation are balanced in this very point, but when at least one propeller is damaged (fault scenario) then a resultant force causes vibrations on the airframe, which can be measured by means of Micro Electrical Mechanical Systems (MEMS) to detect a given propeller fault. Moreover, the thrust developed by each moto-propeller pair is drastically affected.

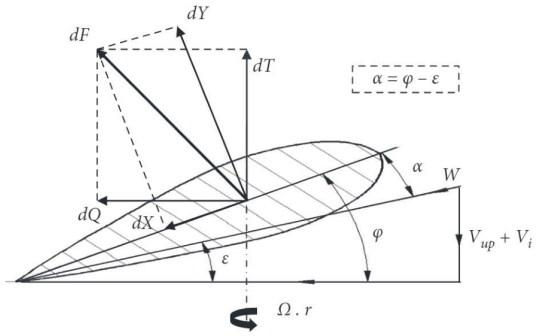


Figure 1.4: Blade Element Moment Theory can explain the aerodynamic forces that affect each portion of a propeller's blade. Credits:Zhu et al. [2021]

Common failure of the rotors can be caused by prolonged use or the entering of particles into the housing of the motor.

Regular maintenance, therefore, is required to counteract the degradation of the motor-propeller pairs.

A load cell transforms a force into an electrical signal. This indirect conversion procedure is divided into two steps. In the first place, the applied force deforms a strain gauge with a suitable mechanic structure. A strain gauge is a small electric device that changes its resistance when strained. The essential property of a strain gauge is its gauge factor (GF)which measures its sensitivity to strain The change in resistance of the strain gauge is then converted into electric signals through the Wheatstone Bridge. The output voltage from the bridge circuit is typically millivolts, which needs increment by a device amplifier. The load cell used in this work, shown in figure 1.5, is sold with an Hx711 24-bit Analogue to Digital converter (ADC) that is equipped with a Programmable Logic Array (PGA) which allows a change of the gain that multiplies the voltage output of the load cell. Load Cells are often used to measure thrust (axial) forces in the context of rocket propulsion systems or for underwater vehicles.

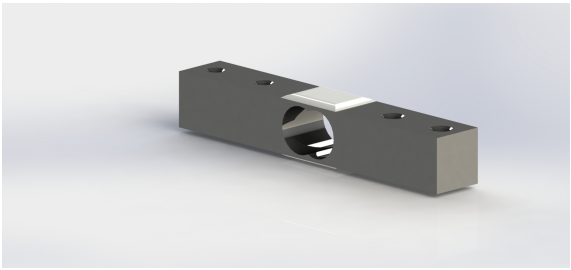


Figure 1.5: Load Cell SolidWorks model

An example of a propeller testbench that uses a load cell to measure the thrust generated by a single rotor can be found in Vorotović et al. [2023] The thrust developed by the rotor can be measured using a load cell that is properly calibrated and positioned to measure the axial force that is generated in the axis of ratio of the propeller.

Experiments performed in the pressure-neutral acoustic wind tunnel facility at the University of Bristol (Jamaluddin et al.), showed that with various modifications to the propeller's blade tip geometry, in the form of anhedral, sweep or tapering to a blade tip, the propeller noise can be reduced in static hover conditions while giving the same thrust output. Each of the tip modifications performs differently depending on the percentage of the blade span which is modified near the tip. Such propeller's blade modifications have been used in Iannace et al. [2019] in the context of fault detection. It is possible, therefore, to measure the propeller's aerodynamic and acoustical properties with different methods. Each can be used in the context of fault detection and diagnosis. Both the load cell-based and the acoustic-based propeller testbench is constrained by the fact that they test the component individually, without the interconnections with the UAV's subsystems that could be affected by a potential fault. The proposed testbench, shown in Figure 1.6 in this work can overcome this limitation, testing the whole UAV system through the use of

several sensors and load cells.

Figure 1.6: The proposed testbench

1.5 Summary & Chapters Organization

The first chapter introduced the general concepts that are needed to grasp the meaning and the value of the results that will be derived shortly. Next, Chapter 2 presents different case studies that do and do not implement Fault Detection strategies in real-time. Chapter 3 introduces the proposed testbench and its architecture, in terms of hardware, firmware and software components. Chapter 4, presents the reader with the results of the calibration procedure, the dataset generation and the assessment of the performances of the classifiers trained. Finally, some directions on how to improve the systems as it is at the moment will be given.

Chapter 2

Data-driven UAV propellers Fault Diagnostics methodologies: State of the art

Instead of the traditional approach to dealing with commonly used techniques when it comes to FD methods, which usually are split into model-based and data-driven, this section gives a discussion on the state-of-the-art strategies that are applied for the step of Dataset Generation; after all, datasets are the basis of every Machine Learning Algorithm. Three strategies are addressed. In the first case the generated datasets, used to train and test classification algorithms before validating their performances with proper test flights, are the result of the collected measurements retrieved from the onboard sensors that can be commonly found on commercial devices. On-flight strategies do not require any laboratory test bench, as the measurements are acquired in real-time thanks to small added sensors or the IMU module that can be commonly found on board commercial devices. This solution has the advantage of being able to run in parallel to the control algorithm in the embedded microprocessor are real-time on-flight. The second strategy consists of generating datasets by collecting the measurements of acoustical acquisition instruments in anechoic acoustic chambers. Anechoic chambers guarantee the cancellation of reflecting echoes from the surface of the walls of the laboratories where the tests are taken. Thus, the methods used in this context rely mainly on the acquisition of acoustic signals and their analysis to extract frequency domain features that will be used to train the classification algorithms. In turn, the success of these algorithms requires the complete knowledge of the embedded microprocessor and suffers from its computational power limitations. The third strategy consists of testbenches designed to test the performances of the motor-propeller pair of UAV systems. The third strategy still has no popular application in the context of data-driven Fault Diagnosis for UAVs. It is the testbench approach, that consists of taking measurements by means of laboratory stands, either propeller's testbenches or full-system testbenches. The distinction between propellers' testbenches and full-system testbenches is that while the first category test separately the faulty component from the rest of the system architecture, the latter is able to test the system as a whole. The main drawback of full-system testbenches is that the solutions that have been proposed more recently, are specific for certain kinds of UAV, and cannot, therefore, be easily adapted to be used for every configuration of UAV system. At the same time, propellers' testbenches are limited in the fact that they do not enable them to reproduce real flight scenarios. Although to the knowledge of the author of this work, there is no example of a testbench devoted to FD methods which test the full systems of a UAV, some indication of the future development for such system designs is given in [Putsep et al, 2021]
.

2.1 On flight strategies

The Fault Detection phase has to do the task assigned as quickly as possible for the fault tolerant control algorithm to take action against possible faults. Therefore, there has been extensive research on real-time FD methods. This solution has been around for the past 20 years. In this

section, we explore in more detail two case studies of the data-driven real-time FD methods category. Real-time FD means, in this scenario, exploiting the possibility of using AI to classify faults of multirotor actuators in a small, energy-saving embedded system, as those that are usually found on commercial devices. Therefore, these methods usually do not require any expensive external measurement instrument, but, instead, rely only on the sensors that are commonly found on board the multicopter, such as the IMU unit, or small non-invasive sensors, such as MEMS accelerometers, that do not affect the flight of the device under test. The first study comes from researchers from the Department of Information Engineering of the University of Marche. The second article was the result of a collaboration between experienced researchers in the field of fault detection for UAVs. The co-author of this review, Youmin Zhang, wrote several articles and reviews on methods and the state of the art for fault detection and diagnosis methods. The other authors all come from a Polish university. Polish researchers through the years contributed extensively to the research in the field of Fault Diagnosis, starting forty years ago with Cempel, followed by the precious contributions of Korbicz to soft-computing approaches above all. As of right now, several articles have been written by the same authors in this review, and they continue to push forward new solutions that take advantage of artificial intelligence.

2.1.1 Case Study: Real-time propeller fault detection for multirotor drones based on vibration data analysis

This article by Baldini et al. [2023] presented an FD method to deal with propeller faults. The authors claim that because this method does not involve any external or additional sensor, but only data coming from the on-board IMU, the developed solution is more feasible than other solutions, in real-time applications. Interestingly, the real flight data come from a hexacopter, so the article has an added value as it can be used later for comparison with this work's results. Moreover this solution, the authors claim, can be implemented in off-the-shelf devices, without the need for additional payload or computational power, powered by artificial intelligence. The UAV used in this article is a DJI F550 hexacopter, with self-tightening propellers made of nylon, which are known to be durable and very flexible, but are less efficient than carbon-fibre propellers. The flight controller is installed on a commercial anti-vibration platform with four rubber dampeners that the authors claim do not affect much the measurements of the IMU module. The authors injected faults in the actuators by chipping off the tips of the blade of the propeller. A total of 18 manual flights were taken over a delimited area, six of which were in nominal conditions and the other 6 + 6 flights where the faulty propeller changed position at every flight, with 5% and 10% fault injected respectively. For the healthy test flights, the authors used a propeller with two regular blades whose length is equal to 11cm.

For the 5% fault, the authors used a propeller with a regular blade and a chipped blade whose remaining length is 10.45 cm.

For the 10% fault, the authors used a propeller with a regular blade and a chipped blade whose remaining length is 9.9 cm.

The authors employ two approaches for AI-based fault detection, one of which is the MATLAB Diagnostic Feature Designer (DFD) app, part of the Predictive maintenance toolbox. With the use of this software, the authors are able to extract 261 features in the time and frequency domain, such as mean values, Damping factors, Kutosis, skewness, standard deviation and more.

By selecting the top 15 features they report that the best result achieved by using Fault detection was obtained with a particular kind of classification algorithm known as 1 vs 1 Lagrangian Support Vector Machine. A binary tree, for comparison, reached a maximum accuracy of 81.63The authors, though, note that running a DFD-based FD detection algorithm on board is essentially impossible due to time and space complexity reasons. Therefore, there is no result that can be used to indicate the real value of such a solution. On the contrary, a solution based on FIR filtering to extract relevant features was developed and has been deployed on board, obtaining up to 93.37%, 97.19% and 98.21% accuracy using a classification tree with 16 input features, an ordinal LSVM with 61 input features and a one versus all LSVM with 114 input features, respectively. The algorithms tested in real scenarios are consistent, achieving at best 100%, 89.6% and 88.8% accuracy using the classification tree, LSVM, and the one versus all LSVM, respectively.

2.1.2 Case Study: Actuator fault detection and isolation system for multirotor unmanned aerial vehicles

In this case study, Puchalski et al. [2022] the authors implement a network model on a microcontroller, thus enabling the real-time FD. The developed solution uses the measurements of four accelerometers, digitally processed to extract frequency domain features. The accelerometers are able to detect the vibrations of the system due to the imbalanced conditions that arise when one of the propellers has been compromised by wear, cuts or other such means.

The drone that has been used as a testbed for this analysis is an octarotor but with 4 dual coaxial propulsion units arranged in an X shape. Each propulsion unit is equipped with a small MMA8452 accelerometer sensor that measures axial and radial accelerations in the rotor's plane of rotation. Therefore, four sensors are connected to an independent data acquisition (DAQ) system. The authors report that the acquisition speed reached 400 samples-pre-second (SPS) and 12-bit resolution within the 2g acceleration range.

The authors led different test flights with different fault combinations, where pairs of opposite or adjacent propellers were damaged. During the tests, different flight paths were followed, and for every scenario over 300 seconds of measurements were collected. The ANN was trained off-board and was later deployed in the small embedded system on board in a .h5 file format, which is commonly used in aerospace, physics, engineering, and academic research fields to store data in a hierarchical format. In particular, this file can contain the trained model's weight values, architecture and training configuration. To pre-process measurement data, fixed-length vectors were populated and multiplied by a Hanning window function; after that, the FFT for each data row was calculated and the values corresponding to specific frequency ranges were selected. The vectors selected like so were combined into tensors corresponding to the frequencies of the eight sensor axes. Only data coming from the axes parallel to the rotor's plane were used due to the small impact of the vertical axis measurements. The authors concluded, that by training in such a way a classical feed-forward neural network with one input layer, one hidden layer and one output layer, using a categorical cross-entropy loss function, and adjusting the network model's parameters, such as the loss function, input's layer activation function and others, good performances can be achieved, and faults can be promptly be detected.

As for the real-time test procedure, the data coming from the accelerometers was analyzed exactly in the same manner as in the offline data pre-processing in MATLAB. The processing times of data ranged within several milliseconds and were affected by the length of the measurement window, the type of window function and the number of neurons in the hidden layer of the ANN. The authors claim that they were able to yield a test accuracy that reached 98.08% (204 out of 208 correct classifications) with the total absence of false alarms.

2.2 Anechoic chambers

By default, the rotor on a UAV has a symmetrical mass distribution. When at least one of the propellers is damaged or incomplete, then the imbalance results in some force that induces a vibration of the airframe. Moreover, given that the noise of the propellers is predominantly given by the tip movement in the air when in fault conditions acoustic signal measurements can be used to test for propellers' health status. Vibroacoustical diagnostics of machinery were introduced more than forty years ago by eminent figures such as Cempel. It can be noted that it is a current trend to apply the theory developed over the years to UAV propellers, given the strong correlation between faults and the relative change in the noise produced by the device. At first, a recent case study will be presented, which is the result of a collaboration between two Italian universities, one of which is the University of Marche, that hosts a semi-anechoic chamber within its facility. The study that will then follow, comes from researchers of the German Aerospace Center in Humburg. The authors of the article took test flights and relative measurements in both anechoic and echoic chambers to test the differences and practicability of FD in both environments. An anechoic chamber is a room developed to stop the echoes of either sound or any electromagnetic source that could interfere with the experiments based on acoustic signal measurements.

2.2.1 Case Study: Lightweight UAV Propeller Fault Detection Through Audio Signals Measurements

In this study by Bruschi et al. [2024], the authors developed a method based on the classification of audio signals emitted by the UAV to detect two different types of faults on a drone propeller. The audio signals are acquired in a certified semi-anechoic chamber. The Machine learning architecture that is developed is intended to be an example of what could potentially be used in the future for adoption on board a commercial drone.

The drone used is a custom quadcopter fixed on a metal tripod. The brushless direct current motors are driven by ESC drivers supplied by a bench power supply, providing a 12 V voltage to the power connector. The rotation speed reaches a maximum of 12000RPM with an increase of rotor rotations per minute, when the voltage is increased by 1V without any load, of 1000 RPM/V.

The propellers used in this study are plastic propellers, installed on each motor. They are composed of two blades, with a total length of approximately 26 cm.

Four professional acquisition systems acquire the operating sound of the drone. They are positioned around the quadcopter at the vertices of an ideal square of side 1m. The microphones were calibrated using an external calibrator and a digital volume indicator. Each signal acquired lasts 116s and is saved in a .wav file format with a sampling frequency of 48kHz. A .wav audio file format is a lossless audio format that does not alter in any way the original audio recorded, for example by means of compression. Three different conditions of the propellers are used for testing. In one case the propeller used is completely healthy; then, the tip of the propeller used is, in one scenario, chipped off at 5mm from the edge, and in another scenario at 20mm from the edge, yielding a similar result as that achieved in the case study 2.1.1. The audio signals were acquired for 116s by the four acquisition systems. A decimation process with a factor of three was applied, therefore the sampling frequency used was equal to 16 kHz. Each test with three different propeller health statuses is repeated three times, yielding a large dataset, although due to some problems that arose during the calibration process, one microphone's signals were excluded from the datasets. In total, the audio signals are divided into windows of 4s composed of 64000 samples. This division yields a dataset of 64000 x 29 x 3 x 3 samples for each test setup. A two-dimensional Convolutional Neural Network was used taking as input the Mel-scaled spectrograms that were the results of the feature extraction phase, where each temporal window was preprocessed in such a way. Mel-spectrograms perform standard spectrograms from audio signals, and they are filtered by exploiting Mel-filterbanks. The authors note that the spectrograms belonging to the healthy propeller and those of the first fault class are quite similar. On the other hand, those belonging to the second class are quite different.

The CNN was trained with a standard back-propagation algorithm, and by considering categorical cross entropy as a loss function. Therefore, the results can be directly compared with the achievements of the study presented in section 2.1.1.

The ML model was converted using the TensorFlow Lite framework, allowing to conversion of the network model into a C++ library that can be deployed on the microcontroller on board, which is an Arduino Nano 33 BLE Sense. The quantised version of the model needs 41.9 kB of RAM and 56.5 kB of flash memory and can be executed in approximately 486 ms.

The trained model achieved in general an accuracy that ranged between 91.20% and 90.41%. Interestingly, the samples belonging to the faulty propeller cut from 5mm concerning the edge of the clade, achieved in the test an overall accuracy lower compared with the fully healthy conditioned propellers. This, the authors claim, can be explained by the fact that the damage injected is too mild and not significant.

2.2.2 Case Study: Development of an Acoustic Fault Diagnosis System for UAV Propeller Blades

This article by Steinhoff et al. [2024] presents an acoustic signal analysis FD method that was performed within both an anechoic and an echoic chamber. Also, tests were carried out in a standard application center laboratory, with concrete walls and a large window front. The operating noise of the device under test, a Holybor X500 UAV quadcopter, was recorded thanks to an acoustic camera that was placed above the drone. The distance from the UAV was set at 0.28 m, positioned at the centre of the drone Z axis. The UAV is attached to a mounting plate, that holds it in place during the tests. Cleverly, the UAV in this study was not operated manually, but it was controlled via WiFi by a laptop that adjusted the thrust generated by the propellers, ensuring the

reproducibility of the tests that were taken. Another laptop acquired the audio signals before a CNN and a linear classifier was trained and tested.

The rotational speed of the propellers was set to 4500 RPM. As in the previous study, three classes were designed for testing. The first with all propellers working correctly, the second with one propeller with a cut tip of 0.5 mm length (2% of the blade length) and the third with a tip of 1 cm length (4% of the blade length)

The input data was converted from the output data recorded by the acoustic camera to a .h5 file format and split into files of 1s each before a beamforming algorithm was applied. This yields two image sources that will be fed to a CNN and an LC, respectively. The first is represented by the application of a functional beamformer that produces acoustic images of resolution 50 x 50 x 1, normalized in the range of 0 to 1 for the CNN model. A grid-search algorithm was used to fine-tune the hyperparameters of the CNN model. A total of 108 possible combinations of hyper-parameters were used. To account for statistical variance in the training step, each parameter combination is repeated three times. Therefore, a total of 324 CNNs were tested. The average validation accuracy of the three repetitions is calculated. The best resulting model achieved a validation accuracy of 99.86%. The second is represented by a modification of the original algorithm, the rotating beamformer, which eliminates the rotary movement of the source and enables a true acoustic map that resembles the source as it was stationary. For the calculation of rotating beamforming images, an algorithm based on the fluctuation of brightness patterns was applied to predict the true angular velocity of the propeller. The LC algorithm classifies these images on the basis of just one feature, thus requiring little computational time and system complexity, and, more importantly, no machine learning at all.

2.3 Testbenches

Normally, UAV's components testbenches are suitable tools that can be used to test these subsystems in a standalone fashion, in distinct conditions. Usually, commercial testbenches that can be found on the market do not include a GUI that can be used by the operator to have a glance at the signals being acquired and do not offer a pre-defined test that can be repeated to assure the reproducibility of the test procedure.

2.3.1 Case Study: Test-bench Development for the Efficiency Analysis of UAV Motor-Propeller Sets

In Hernandez et al. [2021] proposed a solution to test the efficiency of the propulsion unit, that comes with a graphical user interface, and is easily adaptable to most of the propellers that can be found on the market. To measure the thrust force that the pair motor-propeller produces along the axial axis of the bench test, the authors built a mobile base that allows the displacement of the pair on this axis to simulate real conditions, as the motor-propeller motion produces a thrust that is perpendicular to the plain in which it rotates, so the load cell used to measure this force has to be positioned in order to be able to measure this variable. Therefore, the motor and the propeller are mounted perpendicularly to the testbench, and the mobile base can have a linear movement when the propeller generates the push, and, consequently, the load cell is flexed. This solution, though, was not intended for FD methods.
A similar solution can be found in the study Bondyra et al. [2018] where, although the datasets consist of measurements coming from MEMS sensor modules installed on board, a propeller test stand is used to measure the thrust force that the propellers, in different health statuses can develop. The analysis of the impaired rotors leads to the conclusion that while the ESC ensures that the angular velocity of the propeller remains the same, the loss of generated thrust can be observed even in the case of slight damage. The power efficiency, moreover, decreases significantly. Measurements were acquired by 6-axis force/torque sensors, that revealed that even slightly damaged rotors cause a significant and observable increase of structure vibrations, which can be analyzed and processed to identify and locate the fault.

2.3.2 Case Study: A new facility for UAV testing in climate-controlled environments

In Scanavino et al. [2019] the authors explore in detail an important factor that is inevitable for testing UAVs within closed environments, which is the fact that outdoor conditions and disturbances cannot be easily reproduced. Environmental conditions play a crucial role and have a great influence on aircraft. Indeed, thrust is a steadily decreasing function of altitude and temperature, a well-known problem in the aviation industry. Moreover, temperature can have a great impact on battery performance degradation. Together with the complexity of performing the same manoeuvres to test the device, another factor that affects the reproducibility of the tests taken is the variability, within laboratories, of climatic conditions, which can be considered as a bias. Eurac Research funded a project named DronEx to study the environmental parameters that have a consequence on drone flights. In this article, the authors present the results that were yielded following some test flights taken into a research infrastructure in Bolzano, Italy. In this facility, initially thought for medical research, simulation of the Earth's most extreme climatic conditions is possible; two climatic chambers are available for use: the first, the Large Cube, is suitable for industrial tests. The latter, the Small Cube, can be used for replicating different atmospheres. In a first scenario, a motor-propeller pair testbench was prepared to test the effects on the propulsion system of the drone, within the Large Cube. The propellers that were used for this case study are T-Motor 15 x 5 carbon fibre propellers with T-Motor T60A ESC. The rotor assembly was mounted on a dedicated test bench platform, named RCBenchmark 1520, available on the market. In another scenario, the authors test also the overall UAV, mounted on a similar testbench, focusing on the complete UAV performance. In both use cases, the testbench is designed to reduce as much as possible the vibrations, and the sensor suite includes a six-axis load cell, a speed sensor and a power meter to measure thrust, motor speed and power consumption, respectively, while a dedicated power supply, installed into the control room, provides with the necessary electric power to the motors. Moreover, a temperature sensor was installed to measure the temperature of the motor and the electronic speed controller. A total of four simulated atmospheres are used in these two scenarios. At first, the temperature changes constantly in the range -40°C to +60°C, with a 20° step. In the second case, the pressure changes at constant temperature and humidity (from sea level up to 4000m, with a 500m step, and in the opposite direction to go back finally to sea level). In the third case, the combined effect of temperature increase and humidity changes are used. Finally, the combined effect of temperature and pressure changes is used.

The tests were carried out by setting the temperature and waiting for stationary conditions before taking measurements for 60 sec by the data acquisition system. The procedure was repeated by varying the temperature by means of a PWM signal, that also controlled the motor. The tests confirmed the assumption that thrust and motor speed are, indeed, decreasing functions of the temperature. Using curve fitting techniques, it was shown that while between temperature and thrust, there exists a form of linear relation, that of motor speed-temperature, which is some kind of second-order relationship. The authors note that thrust measurements are affected by the thermal effect on the load cell, motor speed reduction and turbulence inside the climatic chamber. The behaviour of thrust concerning the temperature can be explained by the fact that when temperature decreases, the air density changes due to temperature reduction at constant pressure. According to Momentum Theory, the propeller thrust in hover conditions is directly proportional to the air density as well as the squared-induced velocity of the propellers. Therefore, thrust changes are correlated with air density variations, thus with temperature.

2.3.3 Case Study: Conceptual Test Bench for Small Class Unmanned Autonomous Vehicle Performance Estimation

As for full-system testbenches Pütsep et al. [2021] presented a conceptual test bench for small-class UAVs to test their performances and a review of the state of the art of popular testbenches for drones. They suggest that three kinds of UAV Test Benches are used for testing performances, namely motor testbenches, VTOL test benches, and Ground fixed flight control test benches. The first kind tests the motors' thrust force, the motor speed response time, and the power efficiency (in terms of N/W), by varying the speed of the motors by means of the electronic speed controller. VTOL testbenches allow us to test the UAV as a whole, and implement a system of levers, together with force sensors to measure the unit under test and its relative thrust force and power consumption. More importantly, in the context of UAV performance testing, the Figure of Merit (FoM)

is evaluated which is a quantity used to summarize the performance of the propeller-moto-ESC subsystem. Moreover, this quantity is influenced by the wear of the propellers, the degradation of the motor and faults related to the electronic components of the drone. Ground Fixed testbenches are used for fixed flight control to test the device under various circumstances within closed environments. The authors define these systems as mechanical constructions that can hold the UAV, allowing the drone to perform hovering and altitude flight in the indoor environment. From this point of view, the proposed solution that will be described in the next chapter can be seen as a practical combination of VTOL and Ground Fixed testbenches.

Chapter 3

Proposed testbench for small UAVs

The last chapter introduced the main ideas behind the strategies that could be applied to testbench approaches and presented some case studies of Fault Diagnosis data-driven methods for the propellers of a UAV system based on these systems. In literature, many examples of such systems exist, although their existence is devoted to performance or educational purposes. Nevertheless, to this day, there is no evidence of popular ground-based testbenches thought for data-driven Fault Detection of UAV systems, nor propellers-only testbenches.

Figure 3.1: The testbench with a UAV mounted on its support structure

The testbench proposed in this work is for the most part almost identical to the one used in Daponte et al. [2017]. The authors used a testbench composed of a testbed, a power supply system for the test bed and the drone, and a metallic support frame, that holds the structure together. The authors claim that such a testbench can be used before any flight mission, similar to what is done for spaceflight missions, to test lightweight UAVs. Indeed, as previously stated, in the public domain

the first vertiports are being constructed in the UAE emirates, which are designed for missions that require the use of drones BVLOS (Beyond Visual Line of Sight). In this context, VTOLs can land on these platforms, where according to the task being performed, refuelling/recharging of the batteries can take place, and more importantly maintenance procedures can be used to check for the UAV and its subsystems' health status and performances. Therefore, this solution can be used for these means and, more in general, to test a large class of small multirotor UAVs.

3.1 Hardware

The testbench can measure the thrust force developed by the motor-propeller pairs induced on four load cells and can measure various other parameters that are acquired thanks to the monitoring board, which has been completely redesigned. It is now surrounded by a ring structure that houses the monitoring board and three rectangular brackets that are fixed to three load cells, respectively; each load cell has a maximum payload of 10 kg, and they are positioned parallel to the horizontal plane of the test bed. A fourth load cell whose maximum is the payload of 40 kg, can be accommodated underneath the ring housing utilizing a small spring that measures approximately 6 cm in length. The load cells are made of alloy and they have been designed for this particular test bed. Their weight is equal to 84.15g, the longer side of their rectangular shape is 80mm long, while the smaller side is equal to 12.70mm. They are fixed to the monitoring board by means of metallic brackets, that are in turn attached to the metallic frame support using metallic chains, to keep the load cells stretched. Each load cell produces an analogue signal that needs to be converted into a digital signal. Therefore, four Avia Semiconductor Hx711 24-bit analogue-to-digital (ADC) converters are installed on the monitoring board. This unit is specifically designed for weighing scales and offers two channels, namely A and B. These channels can be programmed with different gains, corresponding to different full-scale differential input voltages. These gains are needed in order to cope with the small output signal from the sensor. The four load cells are each connected to channel A of one of the Hx711 ADCs, respectively. The gain for channel A is set to 128, which in turn gives a +/- 20mV range, while for channel B the gain is fixed by the manufacturer and equal to 32, which translates into a +/- 80mV voltage range. Both voltage ranges are based on the assumption that a 5V supply powers the unit. The 24-bit digital values that are produced by the load cells have to be converted into values whose measurement unit is that of kilogram-per-force [kpf]. Because there was no evidence of the existence of any calibration diagrams for these sensors, a calibration procedure took place by using a metallic basket weight support, that was attached to the holes that are used to keep together the metallic support of the testbench and the metallic brackets that house the load cells, by means of metallic chains. A set of two different kinds of weight plates were used to calibrate each load cell. The set included 4 half-kilogram Corenght cast iron weight plates whose internal diameter is equal to 28mm, and 3 one-kilogram Domyos cast iron weight plates, also with a 28mm internal diameter. Because the product specifications don't include any information about the weight values tolerances, an external Ghibertini electronic balance was used to yield a measure of the weight of each plate with a resolution of 1 microgram.

Indeed, the weight of each plate is nowhere near the nominal value indicated on its surface, as the diagram below testifies. Of course, to yield more appropriate results the procedure should be repeated many many times, but nevertheless, it is useful to explain why the fourth load cell calibration was so difficult, compared to the three load cells in the horizontal plane, as the latter have a lower full-scale. The calibration procedure, thus, consisted of placing the weights on the basket support frame, carefully avoiding any oscillation that could affect the measurements, and acquiring 50 measurements each time a weight plate was added. The known weight ranged from 0g to 5035.6g, as the bench pressing machine used to keep the monitoring board in place, could not bear any payload that exceeded these values. This procedure was repeated in 10 iterations, by first increasing the load up to the maximum of approximately 5kg, and then by decreasing the load back to 0g. The result is 20 datasets for each load cell, where each dataset consists of 550 samples. Thus, for each load cell, the available data in total is equal to 11,000 samples, divided into ascending and descending datasets depending on the step of the calibration procedure that produced them.

Channel B of only one out of the four ADCs that are available is also used to measure the voltage and current drained by the multirotor while it is operated on the testbench. These quantities are provided by the output of an external battery connected to the monitoring board and linked to

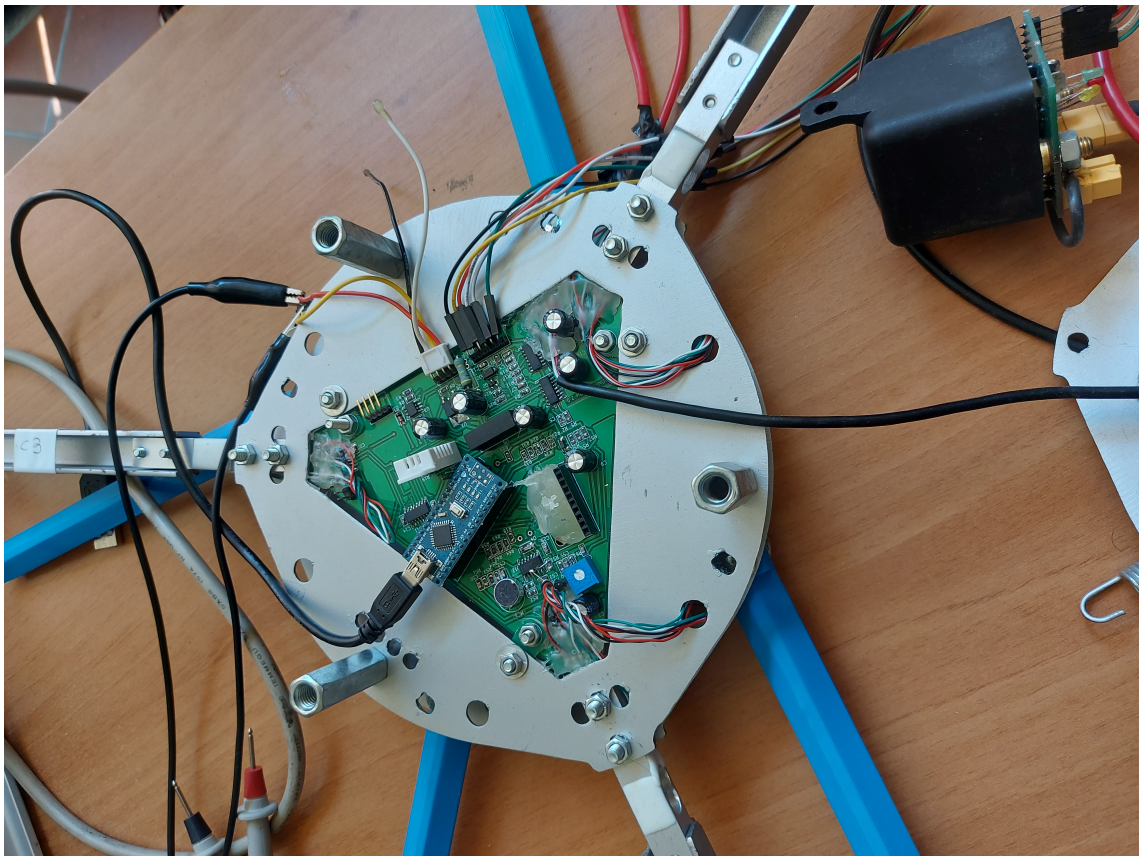


Figure 3.2: The monitoring board

the previously mentioned ADC. The external battery that powers the drone and is connected to the testbench power supply, has been calibrated thanks to an external Fluke 5550a calibrator. The current was calibrated in the range 0A to 11A, which is the maximum range that the external calibrator can allow to be set up. First, the calibrator output is set to go from 0A to 11A, with a 0.5A step, and then from 11A to 0A. This procedure was repeated 10 times yielding a collection of ascending and descending curves, similar, to what has been done for the load cells. This 10 iterations procedure has been reproduced for the voltage outputs, this time in the range 0V to 17V, with step 0.5V. To test the measure of the current drained, a short circuit was crafted. A short circuit is used to simulate a load and allow current to flow through the circuit. The calibrator sent known currents into the battery, and these could be finally measured. The calibration would have not been possible using a regular resistance load, as the voltage on this element would have increased with the increase of the current, and the external instrument would have entered a safety mode, preventing current output flow. The short-circuit was crafted by molding together two wires as shown in the picture. The application software that was used for all of the calibration procedures is further explained in section 3.3.

The monitoring board also features a GY-181 Inertial Measurement Unit (IMU) module. This device contains a three-axis ADXL345 accelerometer, a three-axis HMC5883L magnetometer, and a three-axis L3G4200 gyroscope. These devices communicate using the Inter-integrated Circuit (I2C) protocol. It has been shown plenty of times, even in the brief description of some case studies given in this work, the importance of measuring the strong vibrations that result from unbalanced rotor-propeller pairs. Therefore, these sensors are essential for fault detection of UAV propellers. The accelerometer is used to measure the acceleration of the testbed in the three degrees of freedom (DOF) and theoretically can measure up to +/- 16 g, where g is the gravitational acceleration constant with a resolution equal to 3.9 milli-g/LSB, but, during the experiments that took place in the laboratory, the output was set into the +/- 2g range. The gyroscope measures the angular rate in terms of degrees-per-second (dps). Its output is set at +/- 250 full-scale range. The HMC5883L was not used during experiments.

As noted by [Scanavino et al, 2019], temperature plays a key role in terms of its direct rela-

tionship with thrust. Therefore, concluding the sensor suite list of devices, is a temperature and pressure sensor, the BMP085.

The testbed also includes the UAV used for the test flights. The innovative characteristic of the kind of solution that the testbench provides is that no knowledge about the small UAV under test is needed, especially when it comes to the microcontroller that is installed on board, nor is the multirotor's DJI Naza IMU unit sensor, which measurements are never used in the developed fault detection and diagnosis method. The ESC included in the electric propulsion system of the UAV is controlled via PWM signals, generated by a Futaba T6J radio controller at a given frequency during the test flights, used to increase systematically the thrust generated by the propellers.

3.2 Firmware

Turning the attention now to the firmware of the microprocessor. This software provides the possibility of programming each device included in the sensor suite and finally to produce a string that contains the measurements of each unit. At the start of this project, has it happens often in the context of programming, no exhaustive documentation was available that explained the inner workings of the firmware used during the project developed by [Daponte et al, 2017]; thus the firmware that controlled the legacy monitoring board was refactored. Refactoring, as is intended by Agile Software Development Approaches, means making the existing code of a program cleaner and more understandable for the reader. In the case of the old firmware, it lacked a systematic approach in the organization of modules, as it was written in what could be termed as a monolithic approach. It lacked the documentation of functions, and the description of how variables are used, and what their use is intended for. Furthermore, it implemented functions obsolete or no longer needed.

In software development practises, especially in the context of firmware development for embedded systems, a module is defined as code that encapsulates a given functionality, and has a defined interface that can be used to build levels of abstractions. This means that when a task has to be accomplished, the programmers call a given module using its interface, by passing input arguments, without any knowledge required as to what instructions are executed by the module itself. The user of the module only has access to the interface of the module, and, more importantly, a module should never depend on the code of its user.

In the case of the DronesBench existing firmware, it was first modularized completely. Then libraries to program specific modules in use were added or replaced by more efficient ones. The most sensible of these changes regards the function that reads the data coming from the ADC. It was found that this function was not working as expected, because the 24-bit digital values that were acquired using firmware instruction were not processed correctly.

3.2.1 Programming the ADCs

The datasheet of the product manufacturer specifies clearly how the values produced by the module should be interpreted and how the unit should be controlled using its serial interface, by applying a clock pulse to retrieve each time one bit, starting from the most significant bit (MSB), until all 24 bits are shifted out. The pulse that follows the twenty-fourth selects the channel for the next acquisition. Between each acquisition, no time delays are needed to be added, given that the channel, either A or B, remains unchanged. Whenever this is not the case, then a time delay that can take up to 400ms has to be used. This time delays were not used correctly in the old firmware. Moreover, they processed data in a wrong way. The output 24 bits of data is in 2's complement format, as stated in the datasheet. The same source of information states that when the input differential signal goes out of the 24-bit range, the output data will be saturated until the input signal comes back to the input range. In the current version of the firmware, the function that reads the output of each ADC makes sure that these values are interpreted correctly, and that a sufficient time delay is forced. While initially there was one unique function that read data from both channel A and channel B, in the new firmware these tasks are separated into distinct functions. The function that reads channel A initially sets the clock line to a low level to put the HX711 in normal mode. It waits for each load cell to be ready for data retrieval and then a series of pulses on the clock line (equal for each of the four HX711) commands to transfer the data. As previously stated, each pulse shifts out one bit, starting from the MSB, until all 24 bits are shifted out. Particular attention is paid at the bit that represents the sign of the value returned. The

26th pulse switches to channel B to take the UAV's current & voltage measurements soon after. Finally, a time delay is forced of 100ms. During test procedures, it was found that this value is sufficient to switch properly between channels. The function that reads channel B implements the same functionality but uses 25th pulses to switch back to channel A.

The function that reads channel A is reported in Appendix C, section C.1.

3.2.2 Programming the IMU module

The previous version of the firmware used a completely different GY-181 IMU module, therefore all the libraries that were used before had to be replaced. In particular, starting from the acceleration sensor, the ADXL345, the library selected was found in the extension manager tool into the Arduino Programming Environment, named ADXL345_WE.h. When using libraries that control the sensors, one has always to refer to the .h file where the interfaces of the functions included in the library are listed and optionally can explore the .cpp file that contains the actual instruction that will be compiled into low-level instructions. By exploring different libraries, the one that was chosen was found to be the most straightforward to use. It is used in the function that sets up all the sensors provided by the IMU module, and later in the loop that is run continuously on the microprocessor installed on the monitoring board. The accelerometer is set so to measure the acceleration of the monitoring board in the range of +/- 2g. The values that the sensors yield are expressed in terms of milli-g. The multiplication factor that is stated by the datasheet is applied in the MATLAB application software presented in the next section.

The gyroscope is programmed, within the same IMU set-up function, to return values in the range of +/- 245 dps, and the low-level programming is delegated to a third-party library found on the library manager, named L3G4200D.h. The magnetometer was not used, although the values of the sensor are acquired and sent through the serial interface to the laptop that runs the applications software that stores measurements into .csv files.

The function that initialized the IMU module can be found in Appendix C, section C.2.

3.2.3 Main Loop

In the Arduino programming environment, two functions are essential to run a program (named sketch): the setup() and loop() functions. In a way, they replace the role of the main() function used in the C programming language. The setup function includes all the instructions that have to be executed only once when the microcontroller is turned on. The loop function includes all the functions that are repeated over and over again, as long as the microcontroller is turned on. These functions are both reported in Appendix C, sections C.4 and C.5. The libraries used can be found in section C.3.

In the setup function, the first task is to set up the defined pins as either input or output sources. Four pins are set up as input sources, that will be used to read the bits returned by the four ADC converters, respectively. One pin is set as the source, that is used to pulse the ADC to retrieve bits iteratively. Then the ADCs are set to 'normal mode' by setting the clock line to low, which is shared by every ADC. Each of them when output data is not ready for retrieval, sets the digital output pin DOUT high. The next step, consists of initializing each sensor included in the IMU module, by calling the library functions that set the full-scale ranges for these devices. Finally, the first raw values are read from each ADC, to make sure that channel A is correctly set and ready for the first measurement acquisition.

In the loop function, the first task is that of transmitting on the serial interface, at a rate of 115200 baud (symbols per second), the measurements of the magnetometer, gyroscope, and accelerometer. This data is transmitted as a unique string of characters separated by white space characters, along with the other measurements formatted in the same way. The temperature and pressure measurements are also transferred, as well as the load cells, the voltage, and current measurements returned by the ADCs via the functions that have been defined in the firmware.

The sketch is transferred to the microcontroller by a program called bootloader, which will load the program into the Application Program Section of the Flash Memory of the device. The firmware uses predominantly signed long data types, that are represented in the microprocessor as 32-bit values, where the LSB represents the sign that the digital value conveys.

3.3 LabVIEW Virtual Instrument

Because these procedures consisted of several iterations of the same sequence of steps repeated over and over again, and to reduce the percentage of errors due to distractions of the operator who supervised the calibration of these devices, for this work one LabVIEW (Laboratory Virtual Instrument Engineering Workbench) Virtual Instrument was required to control the external calibrator during the calibration procedure. The VI (Virtual Instrument) is the basic building block of programs written in the Graphic Language. By Utilizing this VI, together with the MATLAB GUI app, the calibration procedure steps for the current and voltage measurements have been fully automated.

3.3.1 Block Diagram

The block diagram contains the graphical source code of a LabVIEW program. The concept of the block diagram is to separate the graphical source code from the user interface in a logical and simple manner. Front panel objects appear as terminals on the block diagram. Terminals on the block diagram reflect the changes made to their corresponding front panel objects and vice versa.

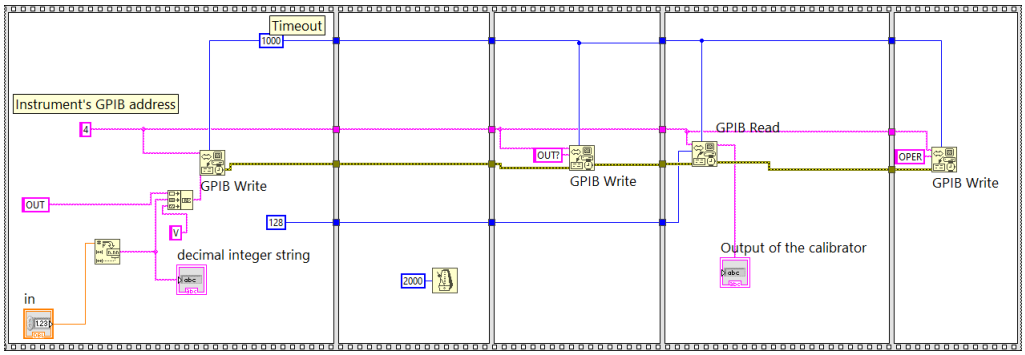


Figure 3.3: LABVIEW VI BLOCK DIAGRAM

The block diagram of the VI used during the calibration procedure was quite simple in its design. Basically, a flat sequence structure is used in order to force a sequential execution of the steps required. The first frame in the sequence sets the address of the calibrator and asks the instrument to set the desired Voltage (or Current) by using the command string that can be found in the product's manufacturer programmer manual. The function that is used to send this command to the instrument is a simple GPIB Write function of the subpalette Instruments I/O GPIB, a package that can be added to the LabVIEW programming environment using the NI Package Manager. The next frame is set to wait for a settling time that is required for the instrument's output to reach regime conditions. After that time elapses, the GPIB Read function is used to check that the correct output has been set correctly on the instrument.

3.3.2 Front Panel

This window is the user interface for the VI. The front panel has controls and indicators, which are the interactive input and output terminals, respectively, of the VI. Controls and indicators placed on the front panel are automatically placed on the block diagram.

In the case of the VI used during the calibration procedure, a simple indicator is used to show the output Voltage or Current that is set on the instrument by the GPIB Write function. A control can be used to manually force a desired input Voltage or Current. This numeric value has to be expressed in scientific notation.

3.3.3 VI terminals

As one would do after defining a function in a normal programming environment, the input and output of the VI are defined by connecting the indicator to the left side of the VI icon, and the input control to the right side of the VI Icon. The VI Icon is an important documentation tool because it controls the appearance of the VI when it is used by other users. The user of this VI is the MATLAB GUI APP that will be described next.

3.4 MATLAB GUI Apps

The MATLAB programming environment provides a framework, called App Designer, to create GUI applications that exploit the MATLAB editor and a wide range of UI's interactive components. This framework enables also to packaging and distribution of apps in the form of installer files directly from the App Designer environment, or by creating a standalone app. The App building components can be summarized into: 1. common UI components (buttons, LEDs, etc.) 2. axes, used to plot functions 3. containers, such as panels 4. user defined components These apps are based on callback functions, and the paradigm used is that of object-oriented programming. All callback functions that APP Designer creates have two default input arguments: app and event. The app object is used to access UI components as well as other variables stored as properties. There are both public and private properties, and they are used to store data that needs to be accessed by multiple callbacks/functions. The private properties can be accessed only by other callbacks/functions of the same app. Public properties are accessible both inside and outside the app. The event object contains specific information about the nature of the user's interaction with the UI component. The event possesses different properties, depending on the specific callback.

Every App designed through App Designer has a startup function, called startupFcn, which can be modified by the user to execute as many statements are needed to initialize property values, for example. This function specifies also the input arguments for the app being designed. This is useful when for a specific task multiple apps are created to address the overall problem with a divide-et-impera approach, where the general task is divided into sub-tasks and each app solves one part of the problem.

For the calibration procedure, while the monitoring board lies on a table, either free or fixed in a given position by a mechanical bench press in the case of load cells calibration, the data stream that the microcontroller produces is transferred to a laptop that runs a GUI app that is set to acquire either voltage, current or load cell signals measurements. The starting app is the same for every procedure.

3.4.1 Calibration Procedure

For the calibration procedure, while the monitoring board lies on a table, either free or fixed in a given position by a mechanical bench press in the case of load cells calibration, the data stream that the microcontroller produces is transferred to a laptop that runs a GUI app that is set to acquire either voltage, current or load cell signals measurements. The starting app is the same for every procedure, except for small differences in the appearance of the GUI, or for the fact that while voltage and current calibrator use the LabVIEW VI created ad-hoc, in the case of the load cells calibration the procedure is predominantly manual. The GUI for the Voltage calibration

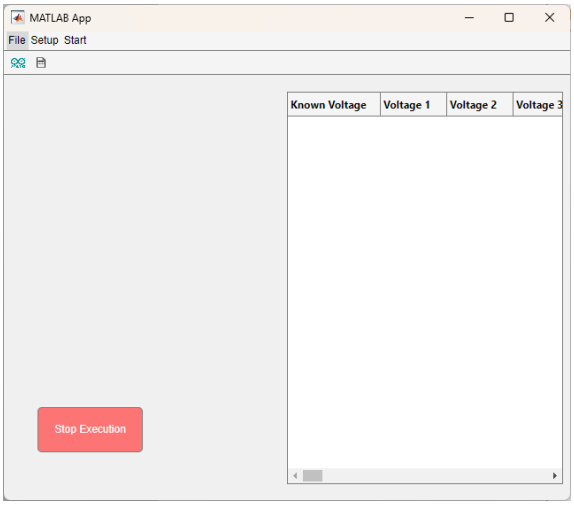


Figure 3.4: Voltage Calibration GUI

consists of a window that displays a menu bar and a toolbar. In the menu bar, the most relevant option is the start menu. Its purpose is to set whether the measurements that will be acquired are

coming from an ascending or descending procedure. In the first case, the voltage is brought from 0V up to the maximum voltage, by using the VI created for this purpose. In the second case, the voltage goes down from the minimum value of 0V. The attention made to these different setups is that the calibration procedure not only has to generate a dataset that will be used to solve a curve fitting problem, but it is also crucial to prove the absence or the presence of hysteresis, in the measurements acquired.

The tool menu has three push buttons. The first opens another GUI app that enables to set the serial communication baud rate and the serial port used. This should agree with the values that are specified in the Arduino sketch that is loaded by the bootloader on the microcontroller. The second button is used to save in a .csv file format the tables that are fulfilled by the 50 acquisitions transferred from the monitoring board to the laptop at each step of an ascending or descending procedure.

The GUI displays a table that can be checked during the execution of the calibration procedure.

The startup function of this app essentially declares the initial values set for the baud rate and the serial port used and initializes the table by giving a name for each of the columns. Each column is the i-th measurement being acquired for a certain step k, where i ranges from 1 to 50, and k ranges between 1 and 34, as the voltage goes from 0v to 17V, with 0.5V step. Once opened, the user selects from the app the Arduino icon in the toolbox to initialize the serial connection with the monitoring board. The app that is delegated with this role is used across all the MATLAB GUI apps created during this work, and it essentially consists of a GUI app that takes as input parameters the values for baud rate and serial port used by the startup function and displays a window where the user can modify such parameters. Once the user chooses such settings, a serial port object is created. This object has the property that it can be used to retrieve strings of serial data from the serial port. The function that is used for this purpose is the readline() function. The next step consists of choosing either an ascending or descending procedure. After this a loading screen appears, while from within the MATLAB environment, a Virtual Instrument object is created to call the dedicated VI that controls the external calibrator. The iteration goes on until the maximum (or minimum) voltage is reached. After this, the user is prompted with an alert that displays a message communicating that the iteration concluded successfully. Then, the user selects the File icon from the Tool menu. A dedicated app, that is used across all the software applications that were designed for this work, is delegated with the task of saving the tables that have been fulfilled. A window is displayed that can be used to select the file path where the user will save the calibration .csv files, under names that are suggested should differ only for the iteration number, to simplify the data processing that will take place with other programming environments. Finally, the user can begin the ascending procedure that is essentially identical to the ascending procedure, the only difference being that the voltage goes from 17V to 0V, with 0.5 steps as before.

While for the current, this routine is the same, the calibration procedure for the load cell is quite different, as the GUI that the user has to use changes to a window where more menu options are available. One such example is the settings menu, that can be used to select the load cell used in the calibration operation. A popup menu is displayed, so that the user can select from a drop-down menu one of the load cells. The toolbar, instead, remains the same as before. The GUI, this time, shows a Tab Group. Each tab is filled with a table that will store temporarily the measurements from a given iteration for a certain load cell. The start menu has the same functionality as before and allows to start of either an ascending or descending iteration, where the weight added onto the basket weight support is going to range from 0g up to approximately 5 kg (in an ascending iteration). The start menu opens another GUI APP that displays the current weight that is loaded on the basket support and shows messages that communicate to the user the actions to be performed. This procedure is manual and cannot be automated as in the case of the current/voltage calibration. The user, thus, is prompted with messages that tell him/her to load (or remove) a certain weight plate from the basket support. Then, the user, once sure that no oscillations are affecting the measurements acquired, can press a button that transfers the relevant measurements to the laptop through the serial interface.

The serial communication transfers data only in one direction, from the monitoring board to the user's laptop, and the format of strings of ASCII characters transferred is known. Therefore, the functions used by each of the apps that are used in calibration routines, are read in each iteration line from the serial port, tokenizing the string to extract only the relevant measurements that are needed for the task (either voltage, current or load cell signal measurements).

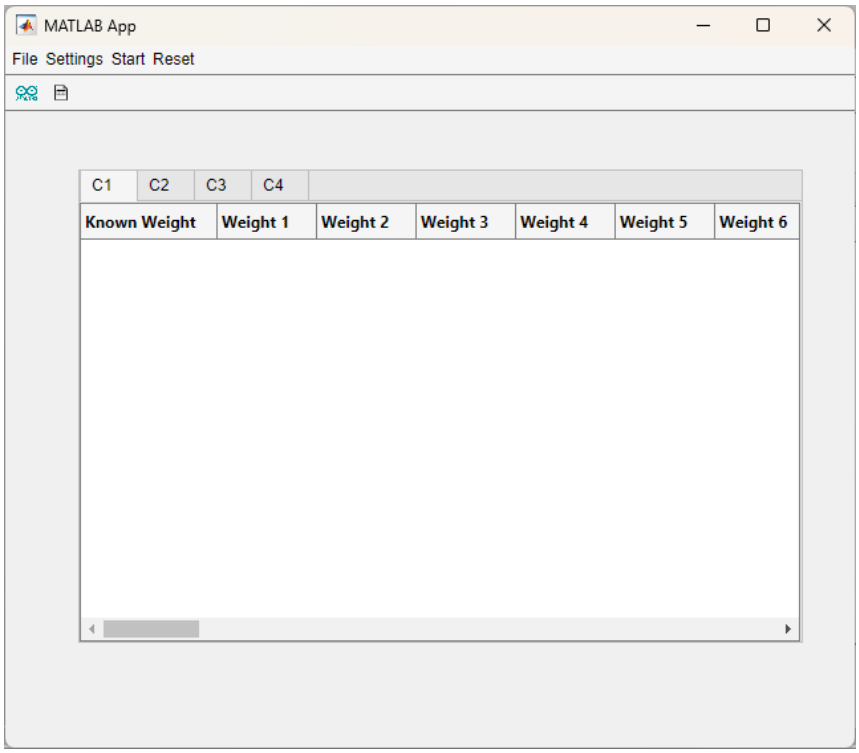


Figure 3.5: Load Cell Calibration GUI

3.4.2 Test Flights

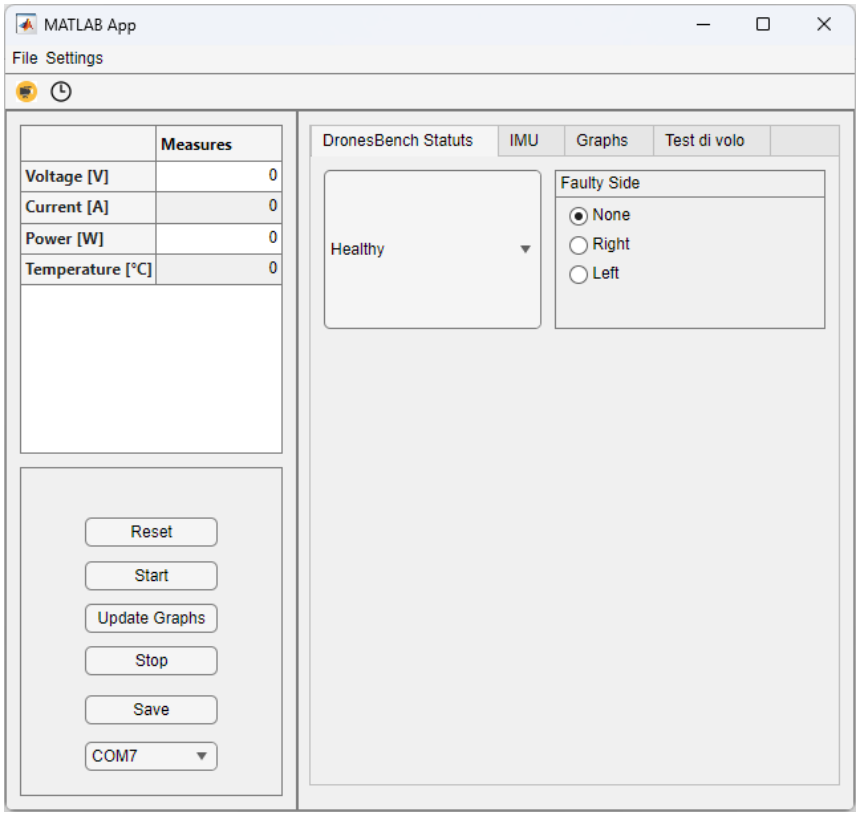


Figure 3.6: Test Flights GUI

As for the test flights, the GUI app that was used was completely different, as was the setup of

the monitoring board and the other equipment. First, the testbench was brought to the centre of a large laboratory room, where it was connected to a power supply. Then, the monitoring board is mounted in the centre of the test bench. The battery is connected to the UAV and the hexacopter is controlled via the Futaba T6J radio-controller. The user will be aided by a GUI app that displays a window where several UI components appear. The menu bar displays a menu option that can be selected to open another app whose only purpose is to select the serial port used to connect the laptop to the microcontroller installed on the monitoring board.

The toolbox displays push buttons that were used to record videos of the experiments conducted via the IP camera Tool that is provided in MATLAB. This tool can be used to control remotely IP cameras that can begin to record videos, which are transferred to the laptop while the test flights take place. This feature worked fine, but it was used only for trivial purposes and could be exploited for future work in the context of image recognition or acoustic signal processing.

The main portion of the GUI is split into two panes. In the left pane, a Table displays live measurements such as Voltage, Power, Current, and Temperature. In the bottom part of the left pane, a group of push buttons is displayed. These buttons are used to:

1. clear the tables filled with measurement acquisitions and the graphs that plot the signals while they are acquired.
2. Start a new acquisition before the UAV's motors are turned on
3. Update the Graphs that plot live signal measurements
4. Stop the acquisition when the UAV goes back to an idle status
5. Save the tables that contain the measurements acquired
6. Select the serial port to be used to retrieve data streams from the monitoring board

The right pane of the GUI shows a Tab Group. In the first tab, a drop-down menu can be used to select which fault case the given dataset will refer to, as an extra column will be added to encode the status of the extra-propeller (either healthy, one tip cut, or both-tips cut). To the right of the drop-down menu, a group of radio buttons show options that the user will select according to where the faulty propeller is positioned (either on the left side of the UAV or the right side).

The second tab shows two graphs that plot the live acquisitions from the gyroscope and accelerometer sensors. Similarly, in the third tab, a graph is shown on which 4 signals overlapped one on the other are plotted; these signals are the measurements coming from the values returned by the ADCs for the voltage signal produced by each of the the load cells, respectively.

Finally, the fourth tab displays a table that along with the measurements being acquired will show for each of them a time stamp, so that the user can improve reproducibility of the test by taking test flights of equal time length.

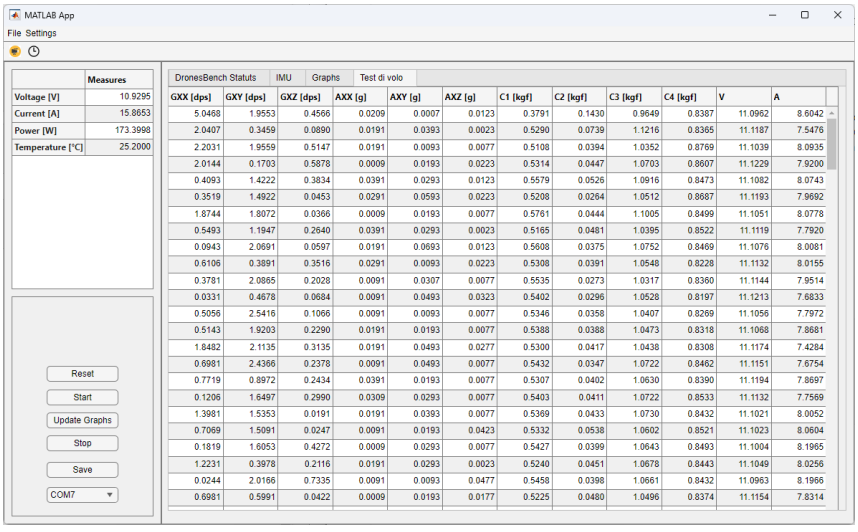


Figure 3.7: Table filled with measurements during the test flights

As for the code of this app, several functions play major roles in the acquisition process.

As for the code of this app, several functions play major roles in the acquisition process.

The startup function is entitled to initiate the serial connection with the microcontroller on the monitoring board. The serial baud rate and the serial port have pre-defined values, but as explained they can be modified by the user via the GUI. The startupFcn then initializes the content of the tables by naming the columns with appropriate names to state to which sensor a given measurement belongs in the fourth tab of the Tab Group in the right pane. The app.counter is set to 1. This property is used to fulfil correctly the table of measurements during the test flights.

The user, once the app opens, can select the appropriate settings for the serial communication, and begin an acquisition by pressing the start button. This will activate a callback which will in turn call a function named start_reading(). This function will start to acquire a stream of data from the serial port. The data stream is acquired in a while loop whose condition is based on a flag variable that is set to 1 when the acquisition begins but can be controlled at any time from the GUI as MATLAB automatically runs the software with multiple threads. Therefore, the user can stop the acquisition routine whenever he considers it to be concluded. The while loop will acquire the data stream from the serial port and tokenize the string of data read. From within the loop, several functions will be called that will process only the portion of the data that they need to solve the task assigned to them. Therefore, although multiple threads can run in parallel, no data race conditions are ever reached. The loop ends by updating the graphs with the newest data that are available. This process goes on until the flag variable is set to -1. The user will press the file icon in the toolbox to store the tables in .csv file format. The program automatically encodes whether the dataset generated belongs to the healthy or any of the two fault classes.

Figure 3.8: The graphs are updated every time a new measurement is acquired

Chapter 4

Experimental Results

This chapter discusses the experimental results from the calibration procedures and the test flights that took place to assess the possibility of using the proposed testbench-based solution in FDD methods. Again, there are no such examples in literature which are designed for Fault Diagnostics purposes. Therefore, the results will be compared against the achievements of the case studies that were analysed in Chapter 2, but that rely on different approaches which are not testbench-based.

4.1 Calibration

The data pre-processing stage is similar for each sensor's datasets. After they are loaded in the Python programming environment, they are separated into two collections: one stores the datasets that contain measurements from the ascending iterations, and the other collects descending iteration datasets.

To address the problem of outliers, two functions are defined. Their code is available in Appendix B, sections B.1 and B.2.

The *outliers()* function is used to detect the presence of outliers by means of the quantiles expedient. Quantiles are values that partition the finite set of observations into subsets of nearly equal sizes. Because the program deals with values which represent measurements, that constitute theoretically an infinite population, and the quantiles have to be calculated over a finite sample of size N=50, the Pandas library uses a linear interpolation method by default to estimate the values of the quantiles using a method adopted by the majority of statistical software packages. Therefore, a real-valued index k is computed such that, if k is an integer, then the k-th element of the N values is the quantile estimate; otherwise, the estimate, h, is computed by a linear interpolation between the values indexed at $\lceil k \rceil$ and $\lfloor k \rfloor$. The equation, thus, assumes the form of a simple linear interpolation:

$$h = x\lfloor k \rfloor + (k - \lfloor k \rfloor)(x\lceil k \rceil - x\lfloor k \rfloor)$$

where $x\lfloor k \rfloor$ is the $\lfloor k \rfloor$ element in the sample of size N.

In the *outliers()* function the 0.25 and 0.75 quantiles are calculated to compute the interquartile range which is equal to the difference between the two estimated values. Finally, the upper and lower bounds are calculated. A value detected within a feature column in the dataset that exceeds these bounds is an outlier.

The *replace()* function receives a list of indices of rows that contain outliers; this list is returned by the outliers function after scanning the dataset. The replace() function, then, replaces the spurious values with the mean value of the column to which the outlier belongs. Each dataset, therefore, is processed by these two functions, selecting a different column each time, and replacing the values that exceed the upper or lower bound defined by the quantiles method. Finally, the datasets are stored in .csv files to be analysed in the MATLAB programming environment where the Curve Fitter Toolbox will be used to solve the problem of generating the calibration diagram for each sensor.

In MATLAB, the datasets are investigated first to assess their statistical distribution. This is not a trivial step; as a matter of fact, the statistical distribution of each feature has to show a Gaussian distribution in order to evaluate Type A uncertainty which is a fundamental step in enquiring the presence of the hysteresis phenomena. If that is not the case, the best practice is

to repeat the dataset generation process. Fortunately, the voltage measurements show a Gaussian distribution, as demonstrated by 4.1, which was created by selecting a column and passing it to a *histogram()* function.

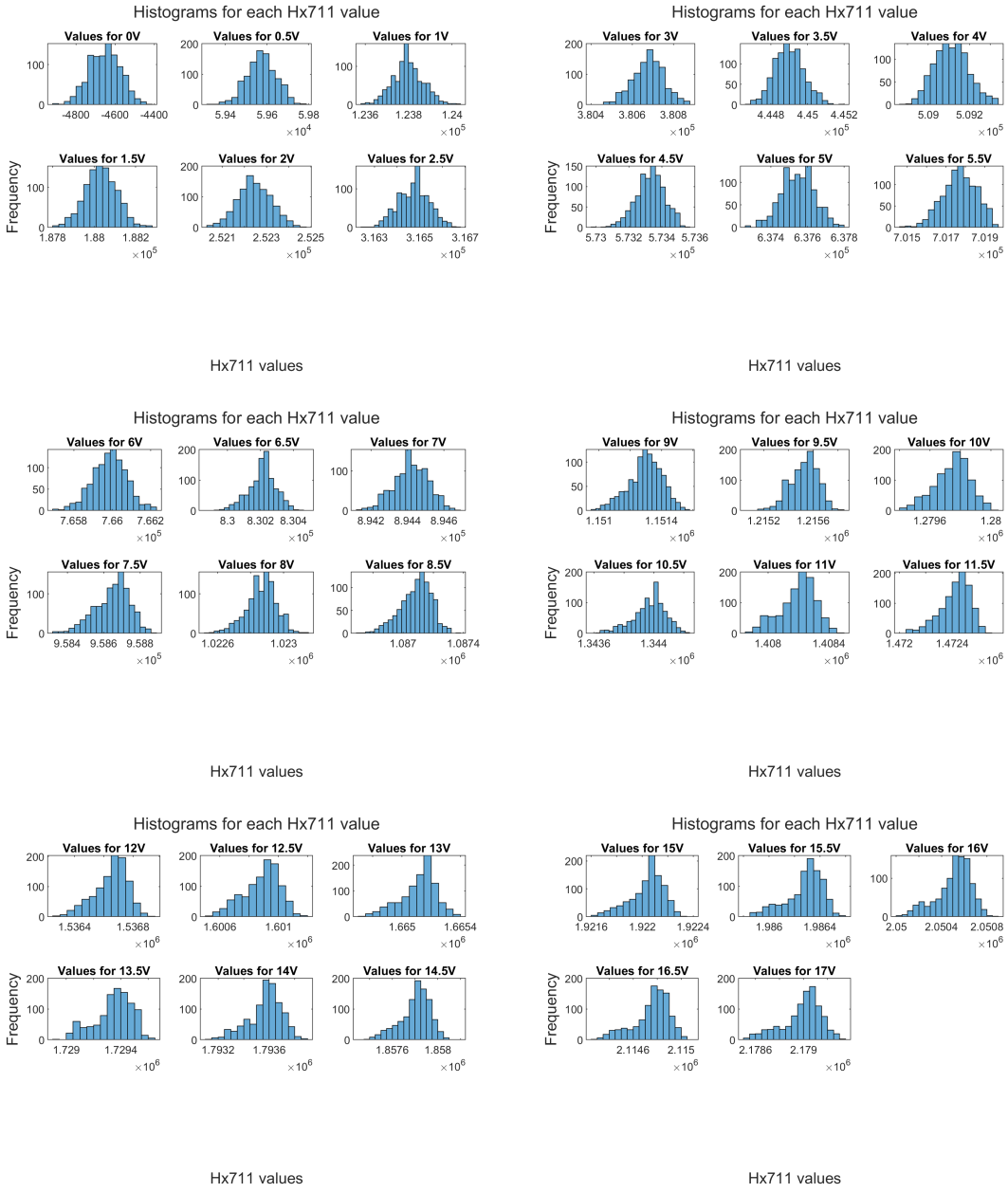


Figure 4.1: The combined Gaussian plots

In order to verify the presence of hysteresis, a total of 6 steps is performed.

1. For each of the ten ascending and descending iterations, the expected value returned by the ADC is computed by taking the mean value over each measurement in the ascending and descending datasets, respectively.

2. The absolute difference between the mean values for each known input quantity (measurement) along ascending and descending datasets, gives the deviation of the mean values among ascending and descending measurements. The maximum deviation is selected.

3. The Type A uncertainty is evaluated for each of the measurements among every ascending dataset, and the maximum value is stored for each feature

Load Cell 1	Load Cell 2	Load Cell 3	Load Cell 4	Voltage	Current
$(19.55 \pm 0.89)\, g$	$(9.87 \pm 0.49)\, g$	$(8.75 \pm 0.48)\, g$	$(10.15 \pm 0.47)\, g$	$(0.32 \pm 0.23)\, mV$	$(0.10 \pm 0.01)\, A$

Table 4.1: Hysteresis Values for each calibrated quantity

4. The Type A uncertainty is evaluated for each of the measurements among every descending dataset, and the maximum value is stored for each feature

5. Finally, if the maximum deviation is bigger than the square root of the sum of the squares of the maximum uncertainties then we can conclude the instrument is affected by hysteresis.

The live script code used to perform this task in MATLAB is reported in Appendix A, section A.1.

As shown in the code, the mean is computed over the collection of ascending and descending datasets. Then Type A uncertainty is computed, and the test for hysteresis is run. The code shows an excerpt from the Live Script created for the Voltage Calibration, but it's the same for every Calibration Procedure.

The results inform that each sensor, unfortunately, is affected by hysteresis. Thus, this constitutes a further component whenever the uncertainty over the measurements acquired is to be evaluated.

The live script, then, uses the mean values to plot the curve for each ascending and descending iteration.

Finally, the Curve Fitting problem is addressed with the aid of the Curve Fitting Toolbox from MATLAB which provides the means to perform regression analysis using linear and non-linear models. This tool is used to fit the Hx711 integer data values to the known voltage/current/weight input values used during the calibration procedure so that, in the future, we can transform the raw values yielded by the ADCs into physical quantities. A first-order polynomial model was used to relate the known inputs (Y values) to the relative mean values yielded by the converter (X values) over all the iterations (both ascending and descending steps). The coefficients and goodness for each of the three transformers for the three kinds of sensors to be calibrated are shown in the table 4.2.

The hysteresis value for each sensor is reported in 4.1.

The calibration curves are shown in figure 4.2.

Sensor	Linear Regression Coefficients (95% Confidence Bound)	RMSE	R^2
Load Cell C1	P1 = 8.89×10^{-3}, P2 = -361.6	62.19 g	0.99
Load Cell C2	P1 = 4.89×10^{-3}, P2 = 1089	0.72 g	1.00
Load Cell C3	P1 = 4.72×10^{-3}, P2 = 259.8	1.32 g	1.00
Load Cell C4	P1 = 4.69×10^{-3}, P2 = -1108	0.82 g	1.00
Voltage	P1 = 7.78×10^{-6}, P2 = 0.036	0.01 V	1.00
Current	P1 = 3.23×10^{-5}, P2 = 0.10	0.02 A	1.00

Table 4.2: Curve Fitter Transformers

4.2 Dataset Generation & Outliers Removal

During experimental test flights, as has been done in the case studies that were presented in Chapter II, one propeller, not initially mounted on the device, is compromised intentionally. The difference with the case studies presented is that the blades' tips are not chipped but simply cut to an extent that is sufficient to produce alterations in the behaviour of the hexacopter. In one scenario, all the propellers are working fine; then, one blade of an extra propeller is cut. The latter, replaces in turn one propeller on the drone, in four different configurations by varying the propeller that is replaced, as shown in the figure. Then, the extra propeller is cut also on the other blade, to produce another scenario where both blades are cut. As before, the extra propeller, in turn, replaces one of the healthy propellers in four different arrangements, so that a fault affecting a propeller is detected in any configuration, whether it is a left-side propeller or right-side propeller. Two propellers, as can be noted, were never used to mount the faulty propeller, as their motors were not brand new, therefore their degraded performances could yield false positives for this

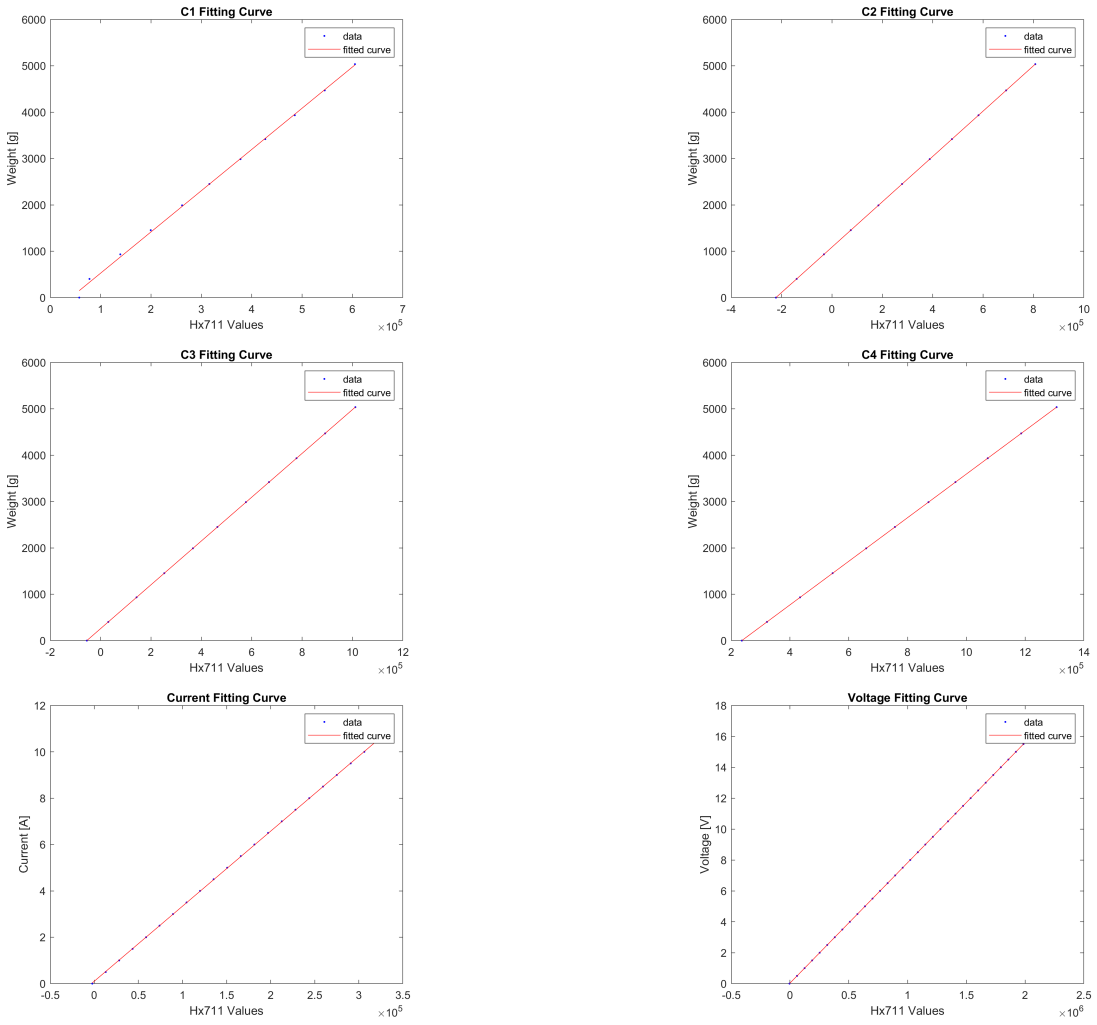


Figure 4.2: The calibration curves yielded through the Curve Fitting Tollbox

reason, they were kept always as in the scenario where all the propellers are in healthy conditions. The images show how to propellers were cut. Test flights were conducted across two days. On the first day, 60 flight tests took place in an indoor environment with all propellers working correctly.

The temperature remained constant during the whole time that was needed to take the measurements. Flights lasted about one minute each, and each consisted of approximately 300 samples. Such a drastic low performance in terms of sampling frequency is caused by the fact that acquiring the measurements produced by the four ADCs required particular attention. Because one of the ADC is used both on channels A and B, changing between these channels required up to 400ms, as specified on page 3 of the datasheet. Therefore, an acquisition cycle can take up to 500ms, with this restriction. In practice, it was noted that this imposed time delay could be lowered down to 200ms. Therefore, the sampling rate achieved is still low, but acceptable for the fault detection method, given that this solution is not thought for real-time strategies for assessing the presence of faults. Future works could improve this inefficiency, by replacing on the monitoring board the ADCs, or by other means. Similarly, the next day, 100 test flights were taken for the first fault class, where one propeller's blade installed on the device was compromised voluntarily. For this case scenario, a total of 25 test flights took place for four different configurations of fault propellers, as summarized in the table below. Finally, the same day, another 100 test flights were taken for the second fault class, where one propeller has two damaged blades, divided again into 25 subsets where four configurations are arranged. The temperature was measured to be in both the first and second fault scenarios equal to the one registered the previous day, when healthy conditioned propeller flights took place.

During the test flights, the total thrust developed by the drone was increased little by little until a maximum was reached, and then it was slowly brought back to an idle state. Because the

Figure 4.3: The weight basket that holds the payload for each load cell during the calibration procedure

Figure 4.4: From left to right: healthy propeller, the right tip of the blade cut, the left tip of the blade cut

UAV was operated by not sufficiently qualified operator, undoubtedly, uncertainties due to the operator's skill have to be taken into account Therefore, the reproducibility of the test taken is not guaranteed. Future works could implement an autonomous system that controls the thrust generated by the motor-propeller pairs, without any human intervention. A grand total of 260 datasets was finally generated, yielding almost 70,000 measurements divided into 'healthy', 'one-cut' and 'two-cut' datasets. The data acquired by the microprocessor on board, an Arduino Nano, were transferred to a laptop via USB that ran a MATLAB App designed ad-hoc for this specific project. The software, along with the other apps, will be described in section 3.3.

Each compensation curve yielded by the calibration procedure was used in the MATLAB GUI app to convert in real time the raw voltage, current, and weight numerical values into physical quantities.

The graphs plotting the values of the data acquired during the test flights immediately revealed the presence of outliers, which in this case, contrasting with what was done in the analysis of calibration datasets, have to be completely removed. Indeed, they cannot be simply substituted with an average value over one or more acquisitions as this would result in changing the shape of

Table 4.3: Labeling

Y1	Y2	Meaning
0	X	All the propellers are healthy
1	1	One of the left side propellers' blade has been cut
1	2	One of the right side propellers' blade has been cut
2	1	One of the left side propellers' blade has been cut twice
2	2	One of the right side propellers' blades has been cut twice

the signals that have been acquired.

Furthermore, the raw values that are stored within the datasets have to be pre-processed as they cannot be used directly in machine learning. Datasets need to be converted into timetables in MATLAB, by adding a time dependency for each measurement. This ensures that the newly created structures will become available to be used in the Diagnostic Feature Explorer app, part of the Predictive Maintenance Toolbox, which can extract time, frequency and time-frequency features from the raw data of which datasets are composed.

Finally, the feature table that arose from analysing the raw data will be fed into the Classification Learner Toolbox from MATLAB app and, in turn, the two fault classes will be targeted by multiple classification algorithms which will attempt to predict the values for these labels. Each classifier's performance will be judged on several metrics common in machine learning, such as classification accuracy, precision and confusion matrices.

Similarly as before, to analyse the datasets the Python Pandas library was chosen for the task of exploratory analysis and data cleaning, as it gives great flexibility and easiness of use when dealing with huge datasets that are stored in comma-separated files. In the first place, the datasets' columns were renamed after the measurements they referred to. In the second place, the datasets were processed to detect the outliers using the same criteria of quartiles and interquartile distance values that were operated before the calibration analysis. This time, though, the replacing function that deals with outlier values is substituted with a remove function that simply drops the rows that contain outlier values instead of replacing the erroneous value with an average over one acquisition or more.

To account for the fact that the monitoring board streams non-zero values even when the UAV is not operated using the remote controller (especially this affects the values for the load cells) over a thousand samples were acquired when the hexacopter is at rest on the testbench, and yet another thousand samples were acquired with the testbench by itself, without the UAV mounted on it. The absolute difference between the two cases is computed and, finally, the values in each dataset are subtracted by the values obtained when the UAV is dismounted from the test bench

Finally, the time column is converted into the index of each dataset so that each measurement is intended as a function of the time variable (expressed in seconds).

The same procedure is repeated for the dataset of test flights with broken propellers.

4.3 Labelling

Class labels are part of every dataset to encode the condition of the propellers during the test flight into numerical values assigned to two classes: Yl and Y2. The table 4.3 summarises what the values for each class label stand for:

4.4 Feature Extraction

Features should include as much class-discriminatory information as possible. Furthermore, the raw values by themselves cannot be used in the context of Statistical Learning, as they are suited only to be used to train Soft-Computing Approaches, such as neural networks, as shown by some of the case studies in Chapter 2. Thus, once all the datasets have been pre-processed they are loaded into the MATLAB workspace. To be easily digested by the Diagnosis Feature Explorer (DFE) app that will follow in this analysis, the list of all the datasets is converted into one MATLAB table whose columns represent measurements and rows represent datasets. In this sense, each element is a timetable structure of the values for a given measurement during a given acquisition. In total the

table size is 260x14. Then, the 260x14 table is then loaded into the app which will help to extract the time and frequency features that will be used to train the classifiers. For each measurement, a plot is generated with the Signal Trace function which plots the signals with the same fault code for the same measurement.

Table 4.4: The 260x14 input table structure that is going to be analysed in the DFE toolbox in MATLAB

gyroX	gyroY	gyroZ	...	Y1	Y2
317x1 timetable	317x1 timetable	317x1 timetable	...	0	0
289x1 timetable	289x1 timetable	289x1 timetable	...	0	0
...	...	...	...	...	...
314x1 timetable	314x1 timetable	314x1 timetable	...	2	2
321x1 timetable	321x1 timetable	321x1 timetable	...	2	2

Once a signal trace is generated for each measurement, the time-domain features are extracted from the raw data. These are the mean, Peak Value, std deviation, Crest factor, Skewness, Waveform Cross Entropy, Root Mean Square entropy Estimator and Kurtosis Factor.

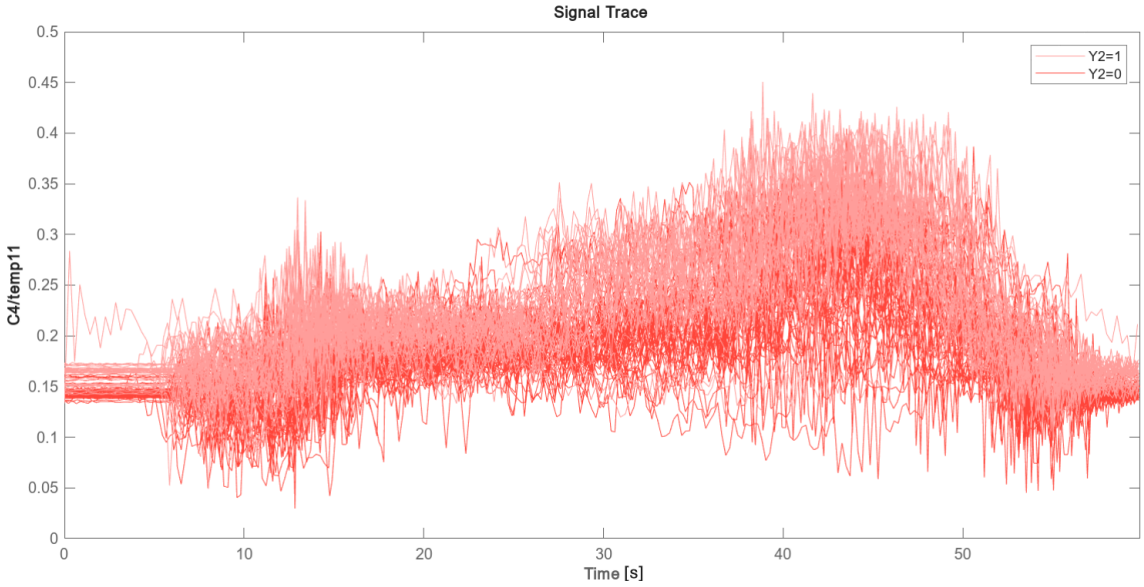


Figure 4.5: Signal Trace generated by DFE for Load Cell C4

The formulas of the signal characteristics computed are shown below:

Serial Number	Feature Name	Formula
1	Root mean square (RMS)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i)^2}$
2	Standard deviation (STD)	$\sqrt{\frac{1}{\mathrm{n}}\sum_{i=1}^{n}(x_\mathrm{i}-\bar{x})^2}$
3	Peak	$\max(x)$
4	Root mean square entropy estimator (RMSEE)	$\frac{1}{n}\sum_{i=1}^{n}(-RMS(i)*\log(RMS(i)))$
5	Waveform entropy (WFE)	$\frac{1}{M}\sum_{i=1}^{M}(W_{t-i}*\log(W_{t-i}))$
6	Kurtosis	$\frac{1}{n*(STD)^4}\sum_{i=1}^{n}(x_i-\bar{x})^4$
7	Skewness	$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i-\bar{x}}{STD}\right)^3$
8	Crest factor (CRF)	$\frac{\mathrm{Peak}}{\mathrm{RMS}}$

To compute frequency-domain features, first, every measurement is inspected to calculate its power spectrum. Then, frequency features are extracted as before. Only the first 3 peaks are considered for each signal, as the others showed no relevant information, or were not computable.

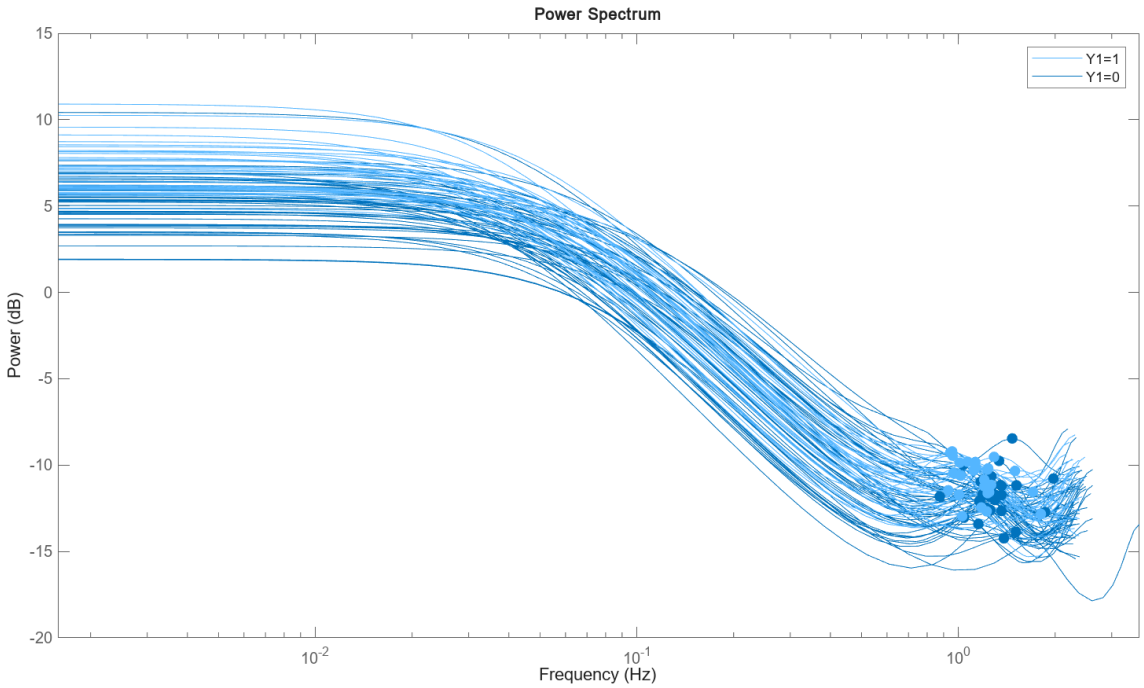


Figure 4.6: Power Spectrum for the X-axis measurements acquired from the gryroscope

4.5 Training

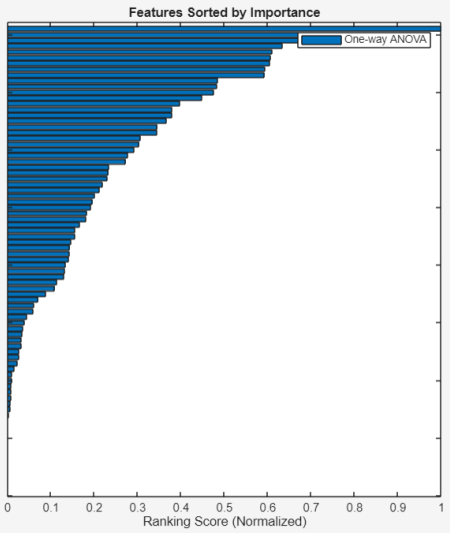
The Diagnosis Feature Explorer allows also to ranking and exporting of the features to develop application-specific algorithms for fault and anomaly detection. The following is the ranking of the most significant features over all the time and frequency features generated before, according to the one-way Analysis of Variance (ANOVA) criteria, for class label Yl.

The one-way ANOVA method can tell whether an independent variable, in this case, represented by the class labels, is influenced by any dependent variable (the features). The table summarizes the quantities evaluated by the one-way ANOVA method. The procedure is also known as F-test.

Source of Variation	Sum of Squares	Degrees of Freedom	Mean Squares (MS)	F
Within	$\mathrm{SSW} = \sum_{j=1}^{k}\sum_{j=1}^{i}\left(X - \bar{X}_j\right)^2$	$df_w = \mathrm{k} - 1$	$\mathrm{MSW} = \frac{\mathrm{SSW}}{df_w}$	$F = \frac{\mathrm{MSB}}{\mathrm{MSW}}$
Between	$\mathrm{SSB} = \sum_{j=1}^{k}\left(\bar{X}_j - \bar{X}\right)^2$	$df_b = \mathrm{n} - \mathrm{k}$	$\mathrm{MSB} = \frac{\mathrm{SSB}}{df_b}$	
Total	$\mathrm{SST} = \sum_{j=1}^{n}\left(\bar{X}_j - \bar{X}\right)^2$	$df_t = \mathrm{n} - 1$		

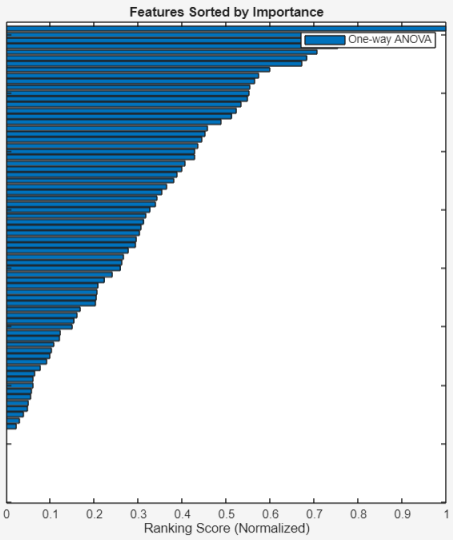
The degrees of freedom within (df_w) are equal to the number of features minus one. The between degrees of freedom (df_b) are equal to the total number of observations within the dataset minus the value of df_w computed above. The sum of squares total (SST) tells how much the data are spread around the Grand total mean (computed by summing all the observations and dividing by their size). The sum of squares within (SSW) is a sum of the variances computed over the dependent variable columns. The Mean Square Within (MSW) is computed by dividing the SSW by (df_w); the Mean Square Between (MSB) is computed by dividing the (SSB) by (df_b). Finally, the dependent variables are ordered in descending order according to the F-test score, computed by dividing (MSB) by (MSW).

The classification Learner Toolbox is set so that the dataset is split into two portions: one that will be used to train the classifiers and another that will be used to test the classifiers predictions. This ranking procedure is performed separately for class labels Y1 and Y2, yielding different results. For each ranking, only the top 15 features are exported to the Classification Learner Toolbox. The feature ranking is shown in figure 4.7 and 4.8. For this analysis, multiple classifiers have been trained and tested. The metric used to judge the prediction quality for each classification algorithm is accuracy which is calculated as:



Feature	One-way ANOVA
gyroX_sigstats/Mean	164.2797
adxlY_sigstats/RMS	122.4477
adxlX_sigstats/RMS	112.5729
adxlY_sigstats/Mean	104.0233
Current_sigstats/Mean	100.2392
adxlX_sigstats/Std	99.7197
Current_sigstats/RMS	99.2511
Voltage_sigstats/Mean	97.4789
Voltage_sigstats/RMS	97.3639
C3_sigstats/Std	79.5984
adxlX_sigstats/Mean	79.2208
gyroX_sigstats/RMS	78.0513
adxlZ_sigstats/Mean	73.5987
C4_sigstats/Mean	65.3163
adxlY_sigstats/Std	62.3227
C4_sigstats/RMS	62.3172
gyroX_ps_spec/BandP...	60.1020
gyroZ_ps_spec/PeakA...	56.7396
C2_sigstats/Std	56.6078
gyroX_ps_spec/PeakA...	50.4137
Voltage_sigstats/Std	49.8574
gyroZ_sigstats/Mean	47.7997

Figure 4.7: Feature Ranking for Class Label Y1



Feature	One-way ANOVA
C2_sigstats/PeakValue	99.5338
gyroX_sigstats/Mean	91.7508
gyroX_sigstats/RMS	78.6871
C4_sigstats/Mean	74.9815
gyroX_ps_spec/BandP...	70.3008
C4_sigstats/RMS	68.0409
Current_sigstats/RMS	66.8177
gyroX_ps_spec/PeakA...	59.5551
Current_sigstats/Mean	57.1232
C2_sigstats/Mean	56.2336
gyroX_ps_spec/PeakFr...	55.1163
Voltage_sigstats/Mean	54.8539
Voltage_sigstats/RMS	54.6184
C2_sigstats/RMS	53.1310
C1_sigstats/PeakValue	52.0605
C3_sigstats/Std	51.0181
C2_sigstats/Std	48.5694
gyroY_ps_spec/BandP...	45.5126
gyroX_sigstats/Std	44.9595
adxlX_sigstats/RMS	44.1629
C1_sigstats/SINAD	43.2680
gyroY_sigstats/RMS	42.6131

Figure 4.8: Feature Ranking for Class Label Y2

$$Accuracy = \frac{\#Correct\ Classifications}{\#Total\ Classifications}$$

4.6 Assessments of the performance of the fault classifiers

The assessment of the results yielded by the most accurate classifiers are reported as confusion matrices, for both validation and test. In the case of the label Y2, as expected, the load cell's signal features are among the most significant, which encodes which side one of the propellers is faulty. Indeed, when one of the propellers was broken, maximizing the thrust generated by the UAV resulted in a tilt of the drone towards the opposite direction to the side where the fault was.

The assessment of the results yielded by the most accurate classifiers is reported as confusion matrices, shown in Figures 4.9 and 4.10 for class label Y1, and Figures 4.11 and 4.12 for class label

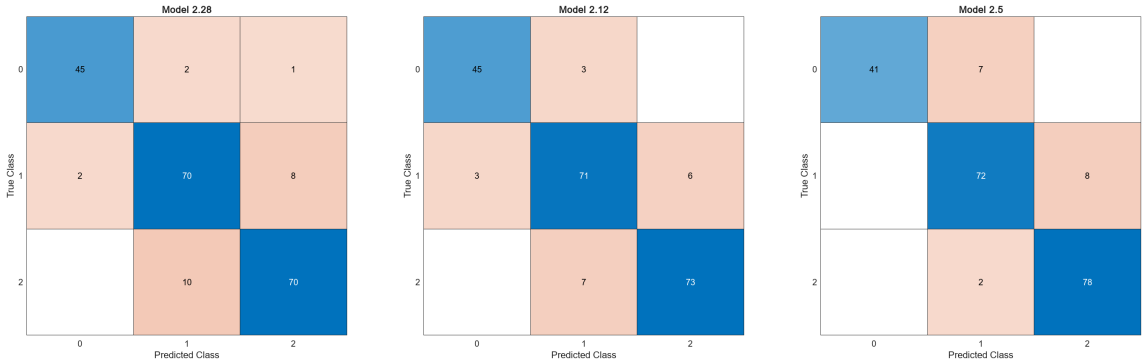


Figure 4.9: Y1 Model Validation Accuracy from left to right: ANN, SVM, QD

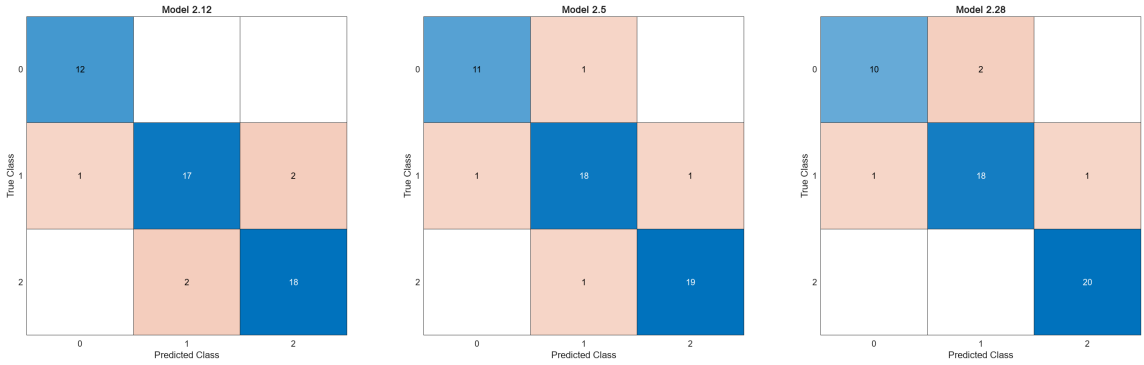


Figure 4.10: Y1 Model Test Accuracy from left to right: ANN, SVM, QD

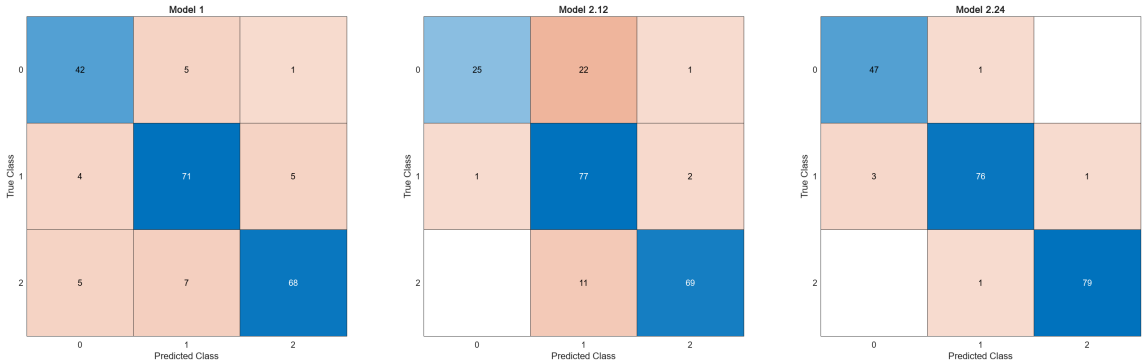


Figure 4.11: Y2 Model Validation Accuracy from left to right: ANN, SVM, Subspace Discriminant Ensemble

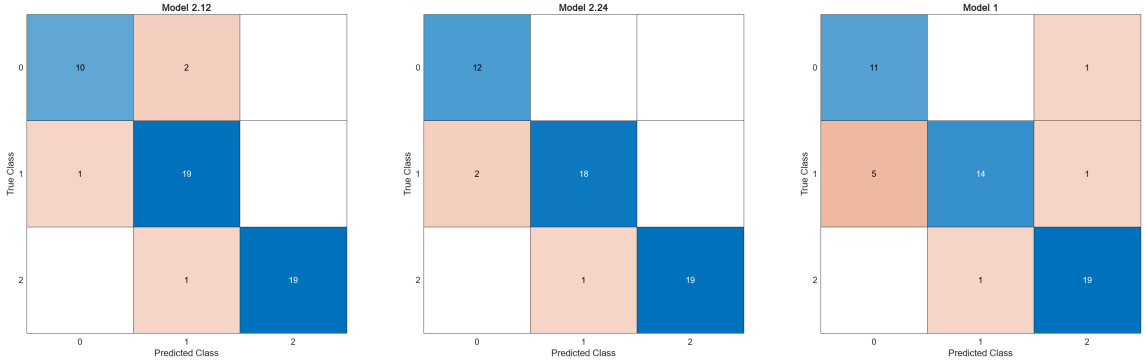


Figure 4.12: Y2 Model Test Accuracy from left to right: ANN, SVM, Subspace Discriminant Ensemble

Y2, respectively, for both validation and test. In the case of the label Y2, as could be expected, the Load Cells signals features are among the most significant, which encodes on which side one

of the propellers is faulty. Indeed, when one of the propellers was broken, maximizing the thrust generated by the UAV resulted in a tilt of the drone towards the opposite direction to the side where the fault was. For this class label, the most accurate model trained resulted in a Subspace Discriminant Ensemble, with a validation accuracy of 97.1%, and test accuracy of 94.2%.

In the case of class label Y1, which encodes whether or not a fault occurred and its severity, the most accurate model trained resulted in a Quadratic Discriminant, with a validation accuracy of 91.8%, and a test accuracy of 92.3%. The results of the most accurate models are summarized in the table below :

Model	Accuracy Validation	Accuracy Test
Quadratic Discriminant	91.8%	92.3%
KNN	89.9%	92.3%
SVM	88.9%	90.4%
NN	90.9%	92.3%

Table 4.5: Y1 class label Classifiers Performances

Model	Accuracy Validation	Accuracy Test
Subspace Discriminant Ensemble	97.1%	94.2%
Tree	87.0%	84.6%
Ensemble (Trees)	89.4%	88.5%
NN	82.7%	90.4%

Table 4.6: Y2 class label Classifiers Performances

Comparison with real-time methods and acoustical methods

The best results achieved during the test flights will be now compared with the performances achieved in the case studies presented in Chapter II, which are based on other approaches such as real-time or acoustic measurements. The accuracy are referred to the test of each model, and not to its validation accuracy. The table 4.7 below summarizes the comparison

MEASUREMENT METHOD	DETECTION ACCURACY	GLOBAL ACCURACY
ON-FLIGHT (1)	100%	97.2%
ON-FLIGHT (2)	99.5%	98%
ACOUSTIC SIGNALS (1)	91.5%	81%
ACOUSTIC SIGNALS (2)	99.7%	97.25%
TESTBENCH (PROPOSED SOLUTION)	92.3%	94.2%

Table 4.7: Comparing the performance against different measuring approaches

The first column refers to the fault detection accuracy achieved by the most accurate model. This determines the ability to decide whether or not a given fault happened. The second third column determines the global accuracy, which includes the ability to assess the severity of the fault. In the case of the proposed solution, the fault detection accuracy was interpreted as the ability to classify between the values assumed by the class label Y1, while the accuracy achieved in the case of the class label Y2 was chosen as the global accuracy (which includes the healthy case too).

Chapter 5

Conclusions & Future Works

The problem of data-driven fault detection for UAV propellers can be addressed with different methods that rely either on online real-time measurements, acoustic measurements, or as in the case of the proposed solution, on testbenches that can test the UAV as a whole. One of the main drawbacks of real-time methods is the fact that to implement the classification algorithm the inner working, i.e. the embedded software, of the microcontroller on board has to be known. In the case of the testbench solution provided in this work, only the firmware of the monitoring board has to be known. Therefore, UAVs can change concerning the number of rotors, model type and their microcontroller and sensors are never used, and their specifications are not of interest. In the case of acoustic measurements conducted in anechoic chambers, the case studied has reported good results, in terms of accuracy, but they neglect external distances (in particular, the wind). The proposed solution consists of a testbench for the whole UAV system, that has no constraints in terms of configuration or equipment. The only requirement is that the Mean take-off mass (MTOM) is less or equal to 25kg. That is, the UAV tested has to be a lightweight UAV. The test bench includes the support frame and the monitoring board. The monitoring board comprises an IMU module, a barometric sensor, for ADC coupled with the analogue outputs of four load cells. The load cells measure the thrust generated by the propellers of the UAV under test. The voltage and current drained are acquired via one of the ADCs mounted on the monitoring board. Each load cell has been calibrated, and so have the voltage and current readings. The experimental result of the calibration procedure has shown the presence of hysteresis, which can only be precise but not compensated. The test flights consisted of iteration of the same procedure in different working conditions. First, the propellers of the UAV under test are used in healthy conditions. Then, one faulty propeller, with one blade cut, is mounted in four different arrangements, allowing the classification algorithm to be trained to identify on which side of the UAV there is a faulty propeller. Finally, test flights are carried with a propeller with both tips cut, positioned in four different arrangements as before. The datasets are generated by means of a MATLAB GUI APP, that acquires the data using the serial interface, which is the medium through which the microcontroller on the monitoring board transfers each measurement. At the end of each test flight, the measurements are stored in .csv files, that are later processed to remove the outliers that arise from the load cells measurements. Finally, the datasets are imported in the Diagnostic Feature Explorer MATLAB tool, and the raw data are used to extract the time and frequency domain features, which are then ranked according to the one-way ANOVA criteria. The resulting top fifteen features from this step, are used within the Classification Learner Toolbox from MATLAB, to train and validate the accuracy of several classification algorithms. The experimental results showed that the classification algorithm that performed best with respect to class label Y1, which indicates whether a fault happened or not, and on which side of the UAV under test, was a Quadratic Discriminant, which achieved a 92.3% test accuracy. As for class label Y2, the best results were achieved with the Subspace Discriminant Ensemble. With a 94.2% test accuracy, this classification algorithm was able to correctly classify between the three working conditions, hence between different propeller health conditions. The main limitation of the testbench proposed can be identified in the sampling rate imposed by the firmware on board. This is due to the limitation imposed by the fact that one ADC is used to convert one of the load cell measurements and subsequentially current and voltage values. This requires to use of a time delay in the firmware, which amounts to 200ms. Therefore, in future works, it is recommended to solve this issue, either by using a fifth ADC, or by other means.

Another limitation is the fact that only faults affecting one propeller at a time can be identified. Therefore, the case where two or more propellers are faulty at the same time is not contemplated. Finally, for future works, it is suggested to:

- modify the MATLAB GUI APP to provide the possibility of detecting in real time the occurrence of faults;
- test the fault detection and diagnosis method with other UAVs (quadrotors, octocopters, etc);
- test the fault detection and diagnosis method against other components faults;
- test the fault detection and diagnosis method for multiple faults that occur contemporaneously;
- generate more datasets before training classification algorithms to improve performances.

Appendix A

MATLAB Code

Listings

Listing A.1: Hysteresis Test Procedure for Voltage Measurements

```
%%statistic mean over 10 iteration (500 samples in total)
ari_mean1 = (mean(ascending_databases_collection));
%%std. dev
std_dev = (std(ascending_databases_collection));
u_a1 = (std_dev/sqrt(500));
max_u1 = max(u_a1) %maximum ascending Type A uncertainty
k = 3; %coverage factor
max_u1_ext = max_u1 * k;

%%same procedure for descending dbs
ari_mean2 = (mean(descending_databases_collection));
std_dev = (std(descending_databases_collection));
u_a2 = (std_dev/sqrt(500));
max_u2 = max(u_a2) %maximum descending Type A uncertainty
max_u2_ext = max_u2 * k;

%maximum deviation between ascending and descending mean values
abs_diff = (abs(ari_mean1 - ari_mean2));
max_diff = max(abs_diff)
sqrt((max_u1_ext)^2 + (max_u2_ext)^2)

if(max_diff > sqrt((max_u1)^2 + (max_u2)^2))
    disp("Voltage is affected by hysteresis");
else
    disp("Voltage is not affected by the hysteresis phenomena");
end
```

Appendix B

Python Code

Listings

Listing B.1: Outliers Detection Function

```
def outliers(df, ft, dictionary):
    # Calculate the first and third quartiles
    Q1 = df[ft].quantile(0.25)
    Q3 = df[ft].quantile(0.75)
    IQR = Q3 - Q1

    # Calculate the bounds for outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Identify the indices of the rows containing outliers
    outlier_indices = df.index[(df[ft] < lower_bound) | (df[ft] > upper_bound)]

    # Append outlier values to the dictionary for that column
    dictionary[ft] += list(df.loc[outlier_indices, ft])

    # Return the list of outlier indices
    return list(outlier_indices)
```

Listing B.2: Outliers Removal Function

```
    def remove(df, ls, col):
    ls = sorted(set(ls))
    valid_indices = [index for index in ls if index in df.index]
    df = df.drop(valid_indices)
    return df
```

Appendix C

Firmware Program Code

Listings

Listing C.1: Hx711 Read Function

```
/*
  This function reads data from channel A of each HX711, and sets
  channel B for the next data transfer.
  Initially, the clock line is set to low level to put the HX711
  in normal mode.
  Then, a series of pulses on the clock line
  (equal for each of the four HX711)
  comands to transfer the data.
  According to the number of pulses sent on the clock line,
  we select the gain and the channel used
  for the next data transmittion.
  Please refer to the HX711 datasheet for more information.
*/
void HX711ReadA(void) {
  mock3 = 0;
  mock4 = 0;
  mock5 = 0;
  mock6 = 0;
  LData1 = 0;
  LData2 = 0;
  LData3 = 0;
  LData4 = 0;
  //set Hx711 in Normal Mode
  digitalWrite(CLK1, LOW);

  // waits for each load cell to be ready for data retrieval
  while (digitalRead(DOUT1) != LOW
  && (digitalRead(DOUT2) != LOW
  && digitalRead(DOUT3) != LOW
  && (digitalRead(DOUT4) != LOW))) //wait until Data Line goes LOW
    ;

  // read a measurement from the four load cells, channel A,
  //gain 128, +-20mV range.
  for (int i = 0; i < 24; i++) {
    // each pulse shifts out one bit, starting at the MSB,
    //until all 24 bits are shifted out
    clk();
    LData1 = LData1 << 1;  // shift
    LData2 = LData2 << 1;  // shift
    LData3 = LData3 << 1;  // shift
    LData4 = LData4 << 1;  // shift

    if (digitalRead(DOUT1) == HIGH) LData1++;  //c1
    if (digitalRead(DOUT2) == HIGH) LData2++;  //c2
```

```
    if (digitalRead(DOUT3) == HIGH) LData3++;  //c3
    if (digitalRead(DOUT4) == HIGH) LData4++;  //c4
  }
  clk();
  clk();  //26th pulse to change to channel B to take
  //the Drone's current & voltage measurements soon after

  digitalWrite(CLK1, LOW);

  //-------------------------------------------------------
  //VERY IMPORTANT: note that changing gain/channel
  //takes up to 400 ms (page 3 of HX711 datasheet)
  delay(100);
  //-------------------------------------------------------

  mock3 = LData1;
  mock4 = LData2;
  mock5 = LData3;
  mock6 = LData4;

  mock3 = mock3 >> 23;  // extract sign bit
  mock4 = mock4 >> 23;
  mock5 = mock5 >> 23;
  mock6 = mock6 >> 23;




  if (mock3 == 1) {
    LData1 |= 0xFF000000;
  } else LData1 |= 0x00000000;

  if (mock4 == 1) {
    LData2 |= 0xFF000000;
  } else LData2 |= 0x00000000;

  if (mock5 == 1) {
    LData3 |= 0xFF000000;
  } else LData3 |= 0x00000000;

  if (mock6 == 1) {
    LData4 |= 0xFF000000;
  } else LData4 |= 0x00000000;

}
```

Listing C.2: IMU setup function

```
  /*
  This function sets up the IMU module used on the monitoring board.
  The IMU is composed of a three-axis gyroscope, a three-axis accelerometer,
  a three-axis magnetometer (ignored) and a temperature and pressure sensor.
  To setup each sensor, library functions were used to hide low-level
  programming of the module itself.
  Please refer to each of the sensors libraries to get more informations.
*/
void imu_setup() {
  gyro.init();
  gyro.enableDefault(); //full scale = +/- 245 dps (degrees per second)
```

```
  adxl.init();
  adxl.setDataRate(ADXL345_DATA_RATE_12_5);
  adxl.setRange(ADXL345_RANGE_2G);

  // verify connection
  // data seems to be best when full scale is 2000
  //gyro.setFullScale(2000);
  bmp.begin();

  mag.begin();

  am2302.begin();  //temperature & humidity sensor
}
```

Listing C.3: declared variables and libraries used

```
   #include <AM2302-Sensor.h>
#include <Wire.h>
#include <Arduino.h>
#include <I2Cdev.h>
#include <Adafruit_HMC5883_U.h>
#include "Adafruit_BMP085_U.h"


//Adafruit_BMP085 bmp;
L3G gyro;
ADXL345_WE adxl;
Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);
//MMC5883MA MMC5883(Wire);
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);

#define SDA A4
#define CLK A5
#define Vin A6
#define Vin1 A7

#define CLK1 13

/*
  a list of the pins used
  to read data that the HX711
  converts from each of the four Load Cells
*/
#define DOUT1 9   //C1
#define DOUT2 11  //C2
#define DOUT3 10  //C3
// june 2024 : the fourth load cell is added into the firmware
#define DOUT4 8  //C4

int16_t ax, ay, az;
int16_t avx, avy, avz;
float gx = 0, gy = 0, gz = 0, gt = 0;
long LData1 = 0, LData2 = 0, LData3 = 0, LData4 = 0;
long LData5 = 0, LData6 = 0;  //voltage and drone's
//current read from one of the Hx711 ADC converters

//sign bits0

long mock1, mock2;
```

```
long mock3, mock4, mock5, mock6;

sensors_event_t event;                  // imu event data structure
AM2302::AM2302_Sensor am2302{ 5 };  //Temperature & Humidity Sensor
```

Listing C.4: setup function

```
    void setup() {

  Wire.begin();
  Serial.begin(115200);
  analogReference(EXTERNAL);
  pinMode(DOUT1, INPUT);
  pinMode(DOUT2, INPUT);
  pinMode(DOUT3, INPUT);
  pinMode(DOUT4, INPUT);

  pinMode(LedPwr, OUTPUT);

  pinMode(CLK1, OUTPUT);

  digitalWrite(PwrOn, HIGH);
  digitalWrite(CLK1, LOW);  //set Hx711 in normal mode

  // setup accelerometer, gyroscope, magnetometer & barometric sensor
  imu_setup();
  //read to set channel A and discard the raw values
  HX711ReadA();
  HX711ReadB();
}
```

Listing C.5: loop function

```
    void loop() {

  mag.getEvent(&event);
  Serial.print(event.magnetic.x);
  Serial.print("--");
  Serial.print(event.magnetic.y);
  Serial.print("--");
  Serial.print(event.magnetic.z);
  Serial.print("--");
  gyro.read();
  Serial.print(gyro.g.x);
  Serial.print("--");

  Serial.print(gyro.g.y);
  Serial.print("--");

  Serial.print(gyro.g.z);
  Serial.print("--");

  xyzFloat raw = adxl.getRawValues();
  xyzFloat g = adxl.getGValues();

  Serial.print(g.x);
  Serial.print("--");
  Serial.print(g.y);
  Serial.print("--");
  Serial.print(g.z);
```

```
  Serial.print("--");


  auto status = am2302.read();

  Serial.print(am2302.get_Temperature());
  Serial.print("--");

  Serial.print(am2302.get_Humidity());
  Serial.print("--");


  HX711ReadA();

  Serial.print(LData1);
  Serial.print("--");

  Serial.print(LData2);
  Serial.print("--");

  Serial.print(LData3);
  Serial.print("--");

  Serial.print(LData4);
  Serial.print("--");


  HX711ReadB();
  Serial.print(LData5);
  Serial.print("--");
  Serial.println(LData6);

}void loop() {

  mag.getEvent(&event);
  Serial.print(event.magnetic.x);
  Serial.print("--");
  Serial.print(event.magnetic.y);
  Serial.print("--");
  Serial.print(event.magnetic.z);
  Serial.print("--");
  gyro.read();
  Serial.print(gyro.g.x);
  Serial.print("--");

  Serial.print(gyro.g.y);
  Serial.print("--");

  Serial.print(gyro.g.z);
  Serial.print("--");

  xyzFloat raw = adxl.getRawValues();
  xyzFloat g = adxl.getGValues();

  Serial.print(g.x);
  Serial.print("--");
  Serial.print(g.y);
  Serial.print("--");
```

```
  Serial.print(g.z);
  Serial.print("--");


  auto status = am2302.read();

  Serial.print(am2302.get_Temperature());
  Serial.print("--");

  Serial.print(am2302.get_Humidity());
  Serial.print("--");


  HX711ReadA();

  Serial.print(LData1);
  Serial.print("--");

  Serial.print(LData2);
  Serial.print("--");

  Serial.print(LData3);
  Serial.print("--");

  Serial.print(LData4);
  Serial.print("--");


  HX711ReadB();
  Serial.print(LData5);
  Serial.print("--");
  Serial.println(LData6);

}
```

Bibliography

URL https://www.easa.europa.eu/en/domains/drones-air-mobility/operating-drone/.

URL https://www.openvia.io/what-are-vertiports-and-how-do-they-work/.

Muhammad Waqas Ahmad and Muhammad Usman Akram. Uav sensor failures dataset: Biomisa arducopter sensory critique (basic). *Data in Brief*, 52:110069, 2024. ISSN 2352-3409. doi: https://doi.org/10.1016/j.dib.2024.110069. URL https://www.sciencedirect.com/science/article/pii/S2352340924000428.

Alessandro Baldini, Riccardo Felicetti, Francesco Ferracuti, Alessandro Freddi, Sabrina Iarlori, and Andrea Monteriù. Real-time propeller fault detection for multirotor drones based on vibration data analysis. *Engineering Applications of Artificial Intelligence*, 123:106343, 04 2023. doi: 10.1016/j.engappai.2023.106343.

Adam Bondyra, Przemysław Gasior, Stanislaw Gardecki, and Andrzej Kasiński. Development of the sensory network for the vibration-based fault detection and isolation in the multirotor uav propulsion system. pages 102–109, 01 2018. doi: 10.5220/0006846801020109.

Valeria Bruschi, Stefania Cecchi, Gianluca Ciattaglia, Grazia Iadarola, Giacomo Peruzzi, Alessandro Pozzebon, and Susanna Spinsante. Lightweight uav propeller fault detection through audio signals measurements. pages 1–6, 05 2024. doi: 10.1109/I2MTC60896.2024.10560801.

P. Daponte, Luca De Vito, Francesco Lamonaca, Francesco Picariello, Maria Riccio, Sergio Rapuano, Luca Pompetti, and Mauro Pompetti. Dronesbench: An innovative bench to test drones. *IEEE Instrumentation & Measurement Magazine*, 20:8–15, 12 2017. doi: 10.1109/MIM.2017.8121945.

Aicha Idriss Hentati and Lamia Chaari Fourati. Comprehensive survey of uavs communication networks. *Computer Standards & Interfaces*, 72:103451, 2020. ISSN 0920-5489. doi: https://doi.org/10.1016/j.csi.2020.103451. URL https://www.sciencedirect.com/science/article/pii/S0920548919303411.

Juan Hernandez, Johanna Nandar, David Changoluisa, Patricio Cruz, and Esteban Valencia Torres. Test-bench development for the efficiency analysis of uav motor-propeller sets. pages 1–6, 10 2021. doi: 10.1109/ETCM53643.2021.9590717.

Gino Iannace, Giuseppe Ciaburro, and Amelia Trematerra. Fault diagnosis for uav blades using artificial neural network. *Robotics*, 8(3), 2019. doi: 10.3390/robotics8030059. URL https://www.mdpi.com/2218-6581/8/3/59.

Nur Syafiqah Jamaluddin, Alper Celik, Kabilan Baskaran, Djamel Rezgui, and Mahdi Azarpeyvand. *Aeroacoustic Performance of Propellers in Turbulent Flow*. doi: 10.2514/6.2021-2188. URL https://arc.aiaa.org/doi/abs/10.2514/6.2021-2188.

Xu JIANG, Min SHENG, Nan ZHAO, Chengwen XING, Weidang LU, and Xianbin WANG. Green uav communications for 6g: A survey. *Chinese Journal of Aeronautics*, 35(9): 19–34, 2022. ISSN 1000-9361. doi: https://doi.org/10.1016/j.cja.2021.04.025. URL https://www.sciencedirect.com/science/article/pii/S1000936121001801.

Bojan Lukic and Umut Durak. *Technology Requirements for the Development of Aircraft for Urban Air Mobility - Status Quo*, pages 1–10. 10 2023. ISBN 978-3-932182-92-1. doi: 10.25967/590061.

Nestor Maslej, Loredana Fattorini, Raymond Perrault, Vanessa Parli, Anka Reuel, Erik Brynjolfsson, John Etchemendy, Katrina Ligett, Terah Lyons, James Manyika, Juan Carlos Niebles, Yoav Shoham, Russell Wald, and Jack Clark. Artificial intelligence index report 2024, 2024. URL https://arxiv.org/abs/2405.19522.

Syed Agha Hassnain Mohsan, Muhammad Asghar Khan, Fazal Noor, Insaf Ullah, and Mohammed H. Alsharif. Towards the unmanned aerial vehicles (uavs): A comprehensive review. *Drones*, 6(6), 2022. ISSN 2504-446X. doi: 10.3390/drones6060147. URL https://www.mdpi.com/2504-446X/6/6/147.

Nwankwo Constance Obiuto, Igberaese clinton festus ikhuoria, Oladiran Kayode Olajiga, and Riliwan Adekola Adebayo. Reviewing the role of ai in drone technology and applications. *Computer Science & IT Research Journal*, 2024. URL https://api.semanticscholar.org/CorpusID:269076285.

Radoslaw Puchalski, Adam Bondyra, Wojciech Giernacki, and Youmin Zhang. Actuator fault detection and isolation system for multirotor unmanned aerial vehicles. In *2022 26th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 364–369, 2022. doi: 10.1109/MMAR55195.2022.9874283.

Kristjan Pütsep, Anton Rassõlkin, and Toomas Vaimann. Conceptual test bench for small class unmanned autonomous vehicle performance estimation. 04 2021. doi: 10.1109/PEMC48073.2021.9432509.

Matteo Scanavino, Andrea Vilardi, and Giorgio Guglieri. A new facility for uav testing in climate-controlled environments. pages 1436–1444, 06 2019. doi: 10.1109/ICUAS.2019.8798066.

Leon Steinhoff, Ann-Kathrin Koschlik, Emy Arts, Maria Soria Gomez, Florian Raddatz, and Veit Dominik Kunz. Development of an acoustic fault diagnosis system for uav propeller blades. *CEAS Aeronautical Journal*, pages 1 – 13, Juli 2024. URL https://elib.dlr.de/205447/. Published online: 12 July 2024; Electronic ISSN: 1869-5590; Print ISSN: 1869-5582.

Goran Vorotović, Jela Burazer, Aleksandar Bengin, Časlav Mitrović, Miloš Januzović, Nebojsa Petrovic, and Djordje Novkovic. A case study of a methodological approach to the verification of uav propeller performance. 69:199–207, 05 2023. doi: 10.5545/sv-jme.2022.432.

Daniel Zhang, Nestor Maslej, Erik Brynjolfsson, John Etchemendy, Terah Lyons, James Manyika, Helen Ngo, Juan Carlos Niebles, Michael Sellitto, Ellie Sakhaee, Yoav Shoham, Jack Clark, and Raymond Perrault. The ai index 2022 annual report, 2022. URL https://arxiv.org/abs/2205.03468.

Hang Zhu, Zihao Jiang, Hang Zhao, Siyu Pei, Hongze Li, and Yubin Lan. Aerodynamic performance of propellers for multirotor unmanned aerial vehicles: Measurement, analysis, and experiment. *Shock and Vibration*, 2021:1–11, 09 2021. doi: 10.1155/2021/9538647.