FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Vibration Analysis for Anomaly Detection in Unmanned Aerial Vehicles
Student:	Tomáš Koranda
Supervisor:	Ing. Lukáš Brchl
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of winter semester 2022/2023

Instructions

This work aims to design and implement algorithms that will detect anomalies of the flying drone based on vibration analysis. This is needed to prevent malfunctions and accidents common in the drone world due to the moving parts (e.g. propellers, motors) that may lead to loose screws or the gradual wear of components. Vibrations can be measured using the IMU unit, which is already inside the drone, or an external device with an accelerometer. The evaluation of the data from the vibration sensor will take place in offline mode without any requirements for real-time processing.

- Research existing solutions.
- Build a suitable test and measuring system (drone parts + hardware).
- Design and implement vibration analysis algorithms to detect flying drone anomalies.
- Gather sample data from several different scenarios (e.g. damaged propellers or engine) and evaluate the implemented algorithms.
- Evaluate the results achieved and propose future extensions.

Electronically approved by Ing. Karel Klouda, Ph.D. on 23 February 2021 in Prague.

Bachelor's thesis

VIBRATION ANALYSIS FOR ANOMALY DETECTION IN UNMANNED AERIAL VEHICLES

Tomáš Koranda

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Ing. Lukáš Brchl
May 12, 2022

Citation of this thesis: Tomáš Koranda. *Vibration Analysis for Anomaly Detection in Unmanned Aerial Vehicles.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Contents

List of Figures

List of Tables

First of all, I would like to thank my supervisor Ing. Lukáš Brchl, for his advice and help with acquiring all the necessary hardware needed for this work. Secondly, I would like to thank Ing. Tomáš Beneš for his support with all the HW setup. I would also like to thank my family and friends for their continuous support. And lastly, big thanks to my partner Rose for her motivation and everlasting support.

Declaration

In Prague on May 12, 2022 . . . .

Abstrakt

Cílem této práce je tvorba algoritmu pro detekci vad v bezpilotních letadlech pomocí vibrační analýzy. Závady na pohonném systému bezpilotních letadel, jako například poškozené vrtule nebo motory, mohou způsobit ztrátu vztlaku, vyšší spotřebu energie a v nejhorším případě vést k havárii letadla. Což může způsobit vážné poškození samotného letadla, a zároveň ohrozit osoby v blízkém dosahu. Vibrace jsou měřeny pomocí 3 osového akcelerometru na vývojové desce Blip. Pro účely sběru dat je navržena a implementována aplikace založená na Zephyr RTOS. Jsou pořízeny tři různé datasety, měřením dvou různých zařízení. Dva různé modely na bázi strojového učení jsou otestovány na naměřených datasetech. Oba modely dosahují více než 95% testovací přesnosti, když jsou trénovány pomocí vzorků ze všech tříd.

Klíčová slova UAV, detekce vad, zpracování signálu, FFT, Zephyr

Abstract

This thesis aims to design an algorithm for the detection of faults on UAVs using vibration analysis. Defects in the propulsion system of an UAV, such as damaged propellers or motors, can cause loss of thrust, higher power consumption, and in the worst case, lead to a crash of the aircraft. Which may cause severe damage to the aircraft itself, and pose a safety hazard to nearby people. Vibrations are measured using the 3 axis MEMS capacitive onboard accelerometer of a Blip development board. A custom Zephyr RTOS based application is designed and implemented for the data collection purposes. Three different datasets are collected using mutiple different devices, as vibration sources. Two different machine learning based models are tested and evaluated on the collected datasets. Both models reach above 95% test accuracy when trained using samples of all fault classes.

Keywords UAV, fault detection, signal processing, FFT, Zephyr

Introduction

UAV, commonly known as a drone, is an aircraft without a human pilot, crew, or passengers on board. UAVs are used in various fields for a wide range of applications. The applications range from hobby and recreational use cases, through commercial applications, such as in agriculture, filmmaking and healthcare, to military applications, such as mapping, surveying and load dropping. Over the past few years, the number of UAVs in operation has steeply risen, especially in the civil segment, and a further increase is expected.

In this work, we are going to focus on multirotor UAVs, a specific subset of UAVs, characterized by having two or more lift generating rotors.

Motivation

The number of active UAVs keeps rising, whether that is caused by more people purchasing drones for recreational purposes or companies trying to employ drones for new applications such as delivery, many of those are UAVs, that operate or will operate in heavily urbanized areas.

These areas are often space-constrained, meaning that the aircraft usually requires the ability of vertical takeoff and landing, which is most often enabled with the use of propeller based propulsion systems.

Defects in the propulsion system of an UAV, such as damaged propellers or motors, can cause loss of thrust, higher power consumption, and in the worst case, lead to a crash of the aircraft. This may cause severe damage to the aircraft itself, and pose a safety hazard to nearby people.

Since propellers are one of the cheapest parts of a multirotor UAV, an early detection of any damage to them, and their subsequent replacement, not only improves the safety of operating such an aircraft, but also prevents, a possibly much more costly, damage to the aircraft.

Objectives

The primary objectives of this thesis are summarized as follows:

- Review existing methods for fault/anomaly detection of UAVs.
- Design and build a suitable measuring test bench for data acquisition.

- Create a dataset representing different scenarios, such as imbalanced propellers, damaged propellers, or damaged engine of an UAV.
- Propose and implement an algorithm using vibration analysis, to detect faults/anomalies in the UAV's behaviour.
- Evaluate proposed methods and discuss possible future extensions.

Thesis outline

The rest of this thesis is organized as follows. In Chapter 1, we introduce the domain of UAVs, and we discuss their history, applications and classification. Additionally, we list the main components of a multirotor UAV and describe their role within the system. Chapter 2 describes the basic working principle of accelerometers and classifies them based on their underlying technology. In Chapter 3, we define the problem, provide a brief overview of prior relevant work on the topics of fault and anomaly detection/identification; and propose and design possible solutions. In Chapter 4 we discuss and analyse the requirements for creating the necessary datasets. Chapter 5 provides the minimum necessary theoretical background regarding signal theory, digital signal processing, and machine learning. We describe the data acquisition process and implementation of algorithms in Chapter 6, followed up by experimental evaluation of implemented algorithms in Chapter 7. The last part of the thesis summarizes and evaluates achieved results, and discusses possible future work.

Chapter 1

Unmanned Aerial Vehicles

UAV, commonly known as a drone, is defined as an aircraft without any human pilot, crew, or passengers on board. An UAV together with all its associated elements, such as ground control station (GCS), transmission systems, data links, and other support, form an unmanned aircraft system (UAS). [1]

1.1 History

The history of UAVs dates back to the beginning of the 20th century. One of their first deployments was during the 2nd World War, when they were used as training targets, or for aerial photography and reconnaissance purposes. During the Vietnam war, UAVs also began to be used in a range of new roles, such as acting as decoys in combat, launching missiles against fixed targets or dropping leaflets. This trend of predominantly military applications continued until the end of the 20th century. [2, 3]

Various factors, such as general advancements in semiconductor manufacturing allowing for smaller onboard electronics, new battery technologies offering better power to weight ratios, or availability of global positioning system (GPS), enabled the manufacturing of smaller and more affordable UAVs. This led to a gradual adoption of drones for other than military applications. Nowadays, drones are easily available to the wide public. They can be purchased as general consumer electronics and are also used for many commercial purposes or by government organisations. Some examples of applications include the following:

- Mapping, inspecting and surveying dangerous areas such as mining and construction sites
- Analysing crop health and conducting topographic surveys to support agriculture
- Patrolling of restricted areas
- Delivering emergency medical supplies or organs for transplantation
- Transferring real-time data from fire and emergency scenes to fire-fighter and police forces
- Responding to natural disasters such as an earthquakes, floods, hurricanes or forest fires

By market capital UAVs used for military applications still hold the biggest market share; however, forecasts predict that should change in the upcoming years, where commercial and consumer purpose drones should hold the lead. [4, 5]

1.2 Classification

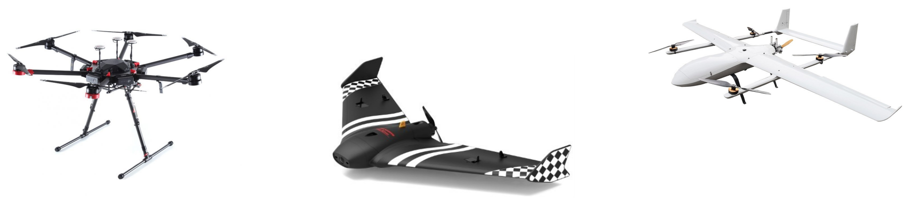
UAVs, like any other aircraft, may be classified based on various parameters, including but not limited to:

- Size and weight
- Range and endurance
- Construction and design
- Propulsion system
- Degree of operational autonomy
- Application purpose (military, civil, ...)

[6, 7]

1.3 Classification Based on Propulsion System

Drones are often associated with quadcopters, however that is only one of many existing construction designs. Some of the most common designs include fixed-wing, multirotor, and hybrid, as shown in Figure 1.1.



(a) Multirotor UAV [8] **(b)** Fixed-wing UAV [9] **(c)** Hybrid UAV [10]

Figure 1.1 Example UAVs

1.3.1 Fixed-wing

Fixed-wing UAVs are similar to a regular aeroplane, both visually and construction-wise. They have a central body with two wings attached to it, and although the wings are fixed, they may contain movable control surfaces, such as flaps, ailerons, and a rudder.

Compared to multirotor UAVs, which use their rotors to generate both lift and thrust forces, fixed-wing UAVs use their rotors (or another type of engine) only to generate thrust (move forward), while the lift is generated by the airfoil-shaped wings, when moved relative to the air. [11, 12]

Unlike multirotor UAVs, which are able to perform vertical take-off and landing, fixed-wing UAVs usually require a significantly bigger area to perform these procedures. However, once airborne, thanks to their aerodynamics, they consume less energy and are relatively energy efficient. This makes them ideal for missions that require the drone to be airborne for longer periods of time or when long distances need to be covered, such as surveillance, mapping and climate monitoring. [11, 13]

1.3.2 Multirotor

Multirotor UAVs consist of a central body and two or more lift generating rotors. Common (conventional) multirotor configurations consist of an even number of rotors arranged symmetrically in one or more parallel planes, with the most common being quadcopter, hexacopter, and octocopter, as shown in Figure 1.2. [14]

The more rotors a multirotor UAVs has, the more thrust it can generate, and thus, the greater the payload it can lift. Additionally, UAVs with more than four rotors also can have a degree of redundancy, allowing them to complete a flight or make a safe descent in the case of one or more rotor failures. However, the trade-off is that a greater number of rotors requires a higher current draw, and therefore hexacopters and octocopters will need to carry a greater weight in batteries to achieve the same flight endurance as a quadcopter of similar size. [15, 16]

By changing the speeds of individual rotors, UAVs are able to achieve 6 degrees of freedom, i.e., it can translate along or rotate around the x, y, z axes. They can perform vertical take-offs and landings, hover mid-flight, and easily manoeuvre up and around objects. This makes them ideal for high-resolution mapping, where a high overlap of captured data is required, or surveying in constrained spaces. However, the improved manoeuvrability comes at the cost of flight endurance. Most multirotor UAVs can fly for about 30 minutes before needing a battery replacement. [15, 17, 18]

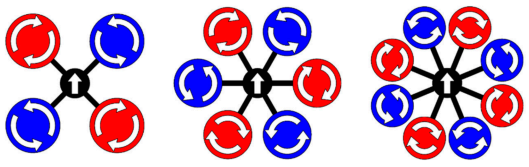


Figure 1.2 Common multirotor configurations [14]

1.3.3 Hybrid

Hybrid UAVs are an attempt to combine the advantages of both fixed-wing and multirotor UAVs into a single aircraft. The designs can vary significantly since it still is an area of development. However, in general, the UAV has a set of rotors allowing for vertical take-offs and landings, or mid-flight hovering, then once airborne, the aircraft transitions to forward fixed-wing-style propulsion. [11, 13]

1.4 Multirotor Construction

The following section will centre around mainly multirotor UAVs, since they are the focus of this work.

1.4.1 Frame

The frame is the main body of a quadcopter. It provides a housing facility to all the other components, such as the flight controller (FC), motors and electronic speed controllers (ESCs) .

It is usually determined by the number of rotors it needs to hold and its weight. The materials used may vary, but in general, they should be as light, rigid and though as possible in order to not add too much additional weight, hold all the components and withstand all the stress forces created during a flight. Carbon fibre composite materials offer an outstanding balance between structural rigidity and weight; however, they are relatively expensive. Another significantly cheaper material often used is aluminium or plastic. Plastic frames are especially popular since they can be easily designed and printed using a 3D printer. [12, 19, 20]

Frames are usually classified by the number of arms and the shape, a set of example frame designs is shown in Figure 1.3.

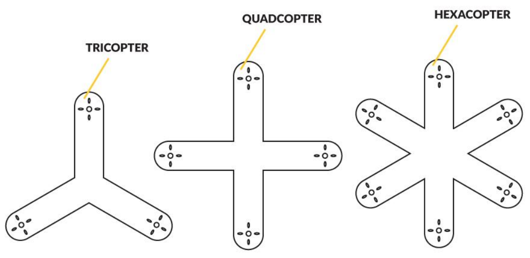


Figure 1.3 Quadcopter frame designs [21]

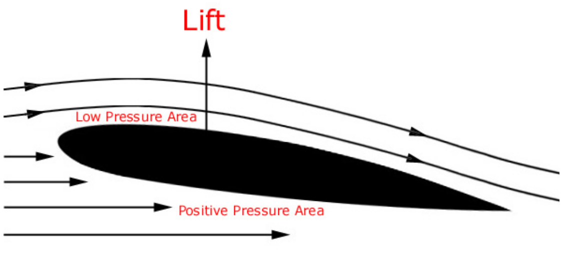
1.4.2 Landing Gears

Landing gears serve mainly as protection or to provide additional space to hold cargo; therefore, smaller drones often do not have any. [19]

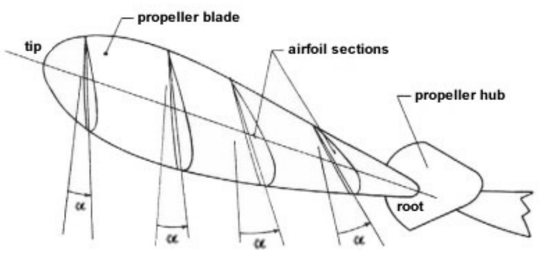
1.4.3 Propellers

The purpose of the propellers is to convert the mechanical energy from the motor into thrust. They are essentially just rotating airfoils, similar in shape to aeroplane wings. [22, 23]

By rotating the propeller, the airfoil shape causes a pressure difference at the top and bottom of the airfoil resulting in the a lift force, as shown in Figure 1.4a. [24]



(a) Airfoil principle [24]



(b) Propeller [25]

Figure 1.4 Airfoil in UAV propellers

The shape and size of a propeller have a major impact on the speed at which the aircraft can fly, the load it can carry, and the speed at which it can manoeuvre. Propellers are usually defined by two numbers: the diameter and the pitch. [20, 26, 27]

The diameter of a propeller refers to the length of the propeller measured from tip to tip. Longer propellers can achieve stronger lift at lower revolutions per minute (RPM) than shorter propellers and, in general, are more efficient. However, due to their larger size and mass, they take longer to speed up and slow down. Shorter propellers, on the other hand, allow the aircraft to change speed quickly and tend to produce better manoeuvring capabilities; however, to produce the same amount of lift as a larger propeller, they need to spin at higher RPM, requiring more energy. This may cause excess strain on the motors and lead to a shorter life span. [20, 26, 27]

Propeller pitch is defined as the distance the propeller would move forward in one rotation if it were moving through a soft solid, the same way a screw would sink into a surface for every rotation of the screwdriver. The pitch of a propeller is largely dependent on its pitch angle. A flat propeller will encounter very little resistance as it slices through the air. However, the lift it generates will also be minimal. On the other hand, a propeller with a steep pitch angle will have to overcome a lot of drag as it rotates, but it will convert more of the wind resistance to generate lift. [20, 26, 27]

Propellers are usually made of either plastic or carbon fibre. Plastic propellers tend to be soft, flexible, and cheap. Their flexibility makes them less likely to crack on impact but may also lead to producing vibrations and generating more noise. On the other hand, carbon fibre propellers are significantly more expensive. Their rigidity makes them less prone to vibrating as they rotate, which means they cut through the air more efficiently and generate lift more consistently. Generally, they result in more stable flights and produce less noise. [26]

1.4.4 Motors

Motors convert electrical energy into mechanical energy, in combination with propellers, they generate lift needed to fly a drone. Since most commercial and hobby UAVs are powered by batteries, they most often use DC motors. There are two types of DC motors used in drones: brushed and brushless motors. Generally, brushed motors are used on smaller, cheaper drones, while brushless motors are used on larger and more expensive aircrafts.

1.4.4.1 Brushed DC Motor

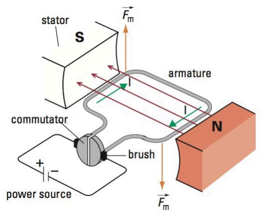
Generally, motors have a stator (stationary part of a motor) and a rotor (rotating part of a motor). The simplest model of a brushed DC motor is shown in Figure 1.5a. The stator is a pair of fixed magnets providing a constant magnetic field. [28–30]

Between these magnets, an armature (simple coil) is placed. The armature is connected to a DC power source through a pair of commutator ring segments. A commutator is simply a ring with gaps in between, creating separated ring segments. The commutator ring sits between two springloaded brushes to ensure contact is always made. [28–30]

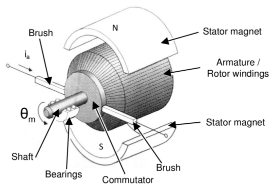
The brushes are connected to a DC power supply, hence creating a circuit. The current flows from the supply, through the brush, the commutator ring segment, the armature loop, back to the other side, i.e., the other commutator ring segment and the other brush, back to the power supply. [28–30]

As the current flows through the coil, the armature becomes an electromagnet; thus the armature will be repulsed by one of the fixed magnets while being attracted to the other one, causing the armature to spin. [28–30]

As the armature rotates, the commutator rotates with it, sliding between the brushes, until the brushes switch contacts between the rings, causing the polarity of the electromagnet switches as well. The switching of polarities keeps the armature spinning. [28–30]



(a) Simplest model of a brushed DC motor [31]



(b) Brushed DC motor diagram [32]

Figure 1.5 Brushed DC motors

This simple model has many flaws, the spinning speed will be irregular, and in the worst case, the armature might get stuck. Adding another pair of commutator ring segments with another coil loop solves this issue, ensuring the armature does not get stuck and making the rotation speed more constant. [28, 29, 33]

Typical brushed DC motors have many loops, the fixed magnets are usually attached to an outer casing, the coils are wrapped around the rotor, and a shaft runs through the middle of the motor, i.e., through the rotor and commutator as shown in 1.5b.

The biggest downside of brushed DC motors is the brushes. Since the brushes rub against the commutator plates, they wear out over time because of friction and eventually will have to be replaced. [34]

1.4.4.2 Brushless DC Motor

In comparison, brushless DC motors do not have any brushes; hence less friction is created, making them more efficient and longer-lasting. [34]

A brushless DC motor also has an outer casing with permanent magnets attached to it; however, in this case, the casing rotates, while the coils within the case are stationary. An example of a brushless DC motor is shown in Figure 1.6. This design is called outrunner since the outer casing rotates. There are also inrunner designs, where the rotor with permanent magnets rotates

Figure 1.6 Brushless DC motor [36]

inside. Although inrunner motors tend to rotate faster, outrunner motors have more torque, which is why they are generally prefered for most applications, including UAVs. [34, 35]

The working principle of a brushless DC motor is similar, to that of a brushed DC motor. It passes an electrical current through coil windings to create electromagnets, while switching their polarity in such manner, that the attraction/repulsion forces created between the magnets of the rotor and stator result in rotation of the rotor. [30, 35]

However, there are a few key differences.

Firstly, the fixed magnets of a brushed DC motor serve as the stator, while the coil windings serve as the rotor. In a brushless DC motor, whether inrunner or outrunner design, the fixed magnets serve as the rotor, while the coil windings serve as the stator. [34, 35]

Secondly, brushed DC motors, use a physical commutator, to mechanically switch the polarities of the coil windings. Thus they require only two wires, and the moment sufficient electrical current is supplied, the brushes making contact with the commutator complete a circuit, causing the rotor to rotate immediately. Brushless DC motors do not have any brushes nor a commutator; instead, the switching has to be handled by an ESC. [34, 35]

To understand why an ESC is required, let us take a look at a simple model of a brushless DC motor, shown in Figure 1.7. It is an inrunner design consisting of a single permanent magnet with two poles serving as the rotor and six coil windings, symmetrically spaced around the rotor.

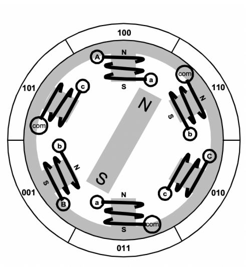


Figure 1.7 Simple model of an outrunner brushless DC motor [37]

Applying an appropriate current to a single-coil winding generates an electromagnetic field that will attract (or repel) the rotor's permanent magnet. Sequentially applying current to one coil after another will cause the rotor to spin. To increase efficiency, two opposite coils can be winded as a single coil, in such a way that each side will generate an opposite pole, thus generating double the force in total, as shown in Figure 1.8. [30]

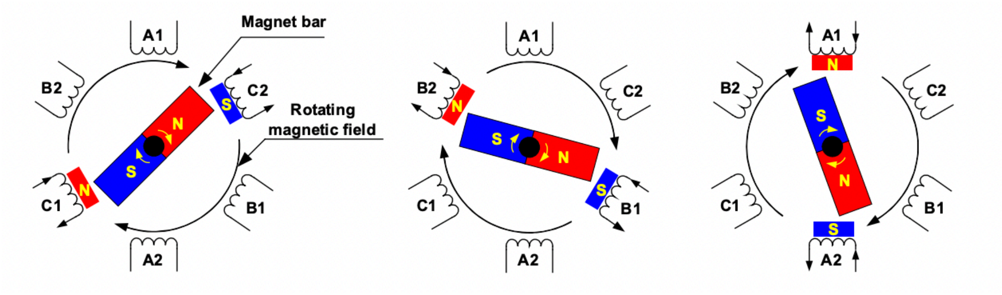


Figure 1.8 Brushless DC motor principle [30]

Since there are three coil windings, we call this design a three-phase configuration. Although brushless motors can be constructed with different numbers of phases, three-phase brushless motors are the most common ones. The three coil windings are connected in either a ”star” or a ”delta” configuration, as shown in Figure 1.9, allowing us to control all three phases using only three wires instead of six. [35]

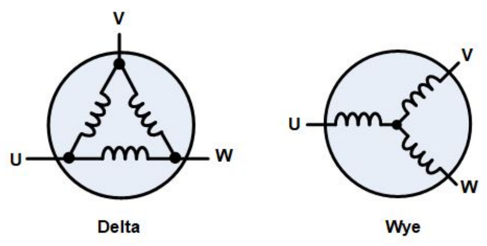


Figure 1.9 Phase configuration [35]

Unlike a brushed motor, whose speed is determined by the supplied voltage, the speed of a brushless motor is controlled by appropriately powering individual phases, which is usually done through the use of an ESC. [35]

1.4.4.3 KV Rating

Brushless DC motors have a ”KV rating”, which refers to the constant velocity of the motor. The KV simply tells us by how many RPM, the motor’s speed will increase, per each volt applied. The KV rating is defined for motors with no load (no propellers), and typically this is just a rough estimation specified by the manufacturer. For example, when powering a 600 KV motor with a 10V power supply, it will rotate at 6000 RPM. [38]

Higher KV motors generally require more current to generate the same amount of torque compared to a lower KV motor. That is why pairing a high KV motor with a too large propeller, may cause the motor to overheat or even burn out, since the motor would attempt to rotate at the same high speeds as it would with a smaller propeller. However, the torque required would be substantially larger, and too much current might get drawn, causing the motor to overheat. [38]

1.4.5 Electronic Speed Controllers

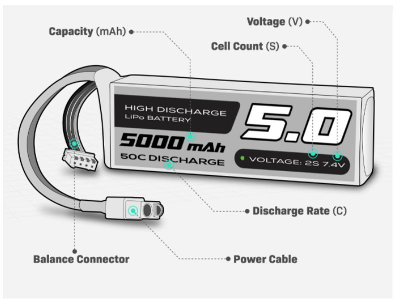
ESCs allow the FC to adjust the speed of individual motors. They come in different shapes and sizes, based on how much power they need to deliver and how many motors they need to control. FPV drones often use 4-in-1 ESCs to reduce weight; however, the most common type are individual ESCs. [39, 40]

To control the speed of a motor, the FC sends a PWM signal, which is essentially just a digital square wave signal. The speed is then determined by the duty cycle (the ratio of time when the amplitude of the signal is high, compared to the time the amplitude is low), of the incoming signal. The frequency of the incoming signal is usually fixed at 50Hz (the period is 20 milliseconds); however, the duty cycles may vary, and usually will have to be calibrated. The calibration's purpose is to match the minimum and maximum duty cycle, of an incoming signal (usually sent by the FC), to the minimum and maximum motor speed.

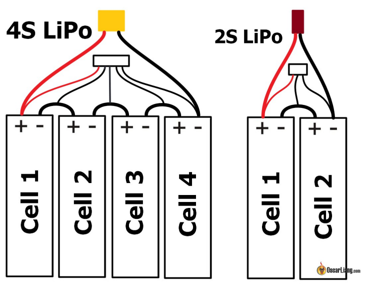
1.4.6 Power source

The most common power source of UAVs, especially commercial and hobby ones, are LiPo batteries. The main reason being their good power-to-weight ratio and large discharge rate compared to other types of batteries, such as nickel metal hydride (NiMH) or lithium-ion (Li-ion) batteries. [12, 20, 41]

When choosing a LiPo battery, several parameters need to be considered, mainly: voltage, cell configuration, capacity, and discharge rate. All of these parameters are usually included on the battery itself, as shown in Figure 1.10a.



(a) LiPo battery parameters [41]



(b) LiPo battery cells[42]

Figure 1.10 LiPo batteries

1.4.6.1 Cell Configuration

LiPo batteries are made out of cells, which are connected and wrapped together in a semi-rigid plastic film, forming a single battery. Each cell has a nominal voltage of 3.7V, i.e., the average voltage a cell outputs when charged. Depending on how the individual cells are connected, the battery capacity or voltage can be increased. By connecting the cells in parallel, the capacity increases; on the other hand, by connecting them in series, the battery voltage increases. [20, 41, 43]

Cell configuration is usually denoted in the format "xSyP", where the "S" and "P" stand for series and parallel, respectively. For example, 2S means that there are 2 cells in series within the

LiPo battery pack, while a 3S2P would mean 4 cells connected in series and 2 cell sets connected in parallel. [20, 41, 43]

Since the cells of a LiPo battery use a dry solid polymer electrolyte, compared to Li-ion battery cells that use an organic liquid electrolyte, they can be produced in a wider variety of shapes and sizes, and generally tend to be lighter and smaller. [44]

Note, you can also tell the number of cells in a LiPo battery by taking the number of wires in the balance connector, minus one. If the balance connector has 3 wires, it is a 2S battery, if it has 7 wires, it is a 6S battery, etc. [41]

1.4.6.2 Voltage

Voltage has a direct influence on how much power a battery can provide to a motor, Having a higher voltage means the battery can provide more power and thus be able to drive a bigger motor. The power can be calculated as $P = V \cdot I$, with V being the voltage and I being the current. When people talk about battery packs, usually they do not refer to the voltage, but most often just to the "S rating", i.e., how many cells it has connected in series since that directly determines the voltage. [20, 41, 43]

The voltage of a battery is not constant, it changes depending on its charge. A fully charged LiPo cell will have a peak voltage of around 4.2V. When being discharged, the voltage slowly decreases. The discharge curve of a LiPo battery is relatively flat until a certain threshold when a drop occurs. In comparison to a NiMH battery, which has a linear discharge curve. [45]

1.4.6.3 Capacity

The capacity of a battery is defined as the amount of electric charge that it can deliver at the rated voltage. Simply, it tells us how much power a battery can hold. The unit of LiPo battery capacity is most often milliamp-hours (mAh). Capacity can be interpreted as how much current can be drawn from the battery for one hour until it is empty. [20, 41, 43]

Larger capacity packs may power an aircraft for longer periods of time but will also get heavier and bigger in size, hence consuming more energy to fly. There is a trade-off between capacity and weight; for optimal performance, there needs to be a balance between size and capacity. [43]

1.4.6.4 Discharge Rate

The discharge rate, often referred to as the "C rating", tells us how fast a battery can discharge safely, i.e., the maximum safe amount of current that can be drawn. The C rating by itself may be a little misleading since it is relative to the battery's capacity. The formula for maximum current draw is as follows: Max Current Draw = Capacity $\cdot$ C Rating. Some batteries come with two C ratings, a continuous rating and a burst rating, allowing to draw extra current for a short period of time. Drawing over the recommended rating can lead to overheating and subsequently damaging the battery. [20, 41, 43]

1.4.6.5 Main Connector

With the exception of 1S batteries, all LiPo batteries come with two sets of connectors: a balance connector and a main connector, as shown in Figure 1.10a. There are quite a few different main connectors used in LiPo batteries, with their main difference being their shape, weight, and current rating. [20, 43]

1.4.6.6 Balance Connector

Since most battery packs comprise of more than one cell, and although the cells might appear to be identical, each cell will be slightly different, i.e., have different peak voltage or capacity or be in different states of charge.

LiPo battery cells should always be in the range of 3V - 4.2V, Any lower than 3V and the battery may permanently degrade. Any higher than 4.2V creates a significant risk of a battery being damaged or even bursting into flames. If all battery cells were to be charged at the same rate, some cells might get overcharged, while others might get undercharged. A balance connector, alongside a balance charger, is designed to overcome this problem, by monitoring each cell's charge state, ensuring that all the cells are charged equally.

1.4.7 Inertial Measurement Unit

The inertial measurement unit (IMU) is basically just a set of sensors, mainly consisting of accelerometers, gyroscopes and magnetometers. Accelerometers measure linear acceleration, gyroscopes angular velocity and magnetometers magnetic field strength, to be specific, earth's magnetic field strength, in order to determine a heading relative to the earth's magnetic north, unless there is some interference present. An individual inertial sensor can only sense along a single axis. That is why to provide measurements along all three axes, three individual inertial sensors must be mounted together into an orthogonal cluster. [46]

Unlike GPS, which is used to determine an absolute position, IMUs are used to determine a relative position with respect to itself. Since all sensors are prone to errors, these errors can accumulate over time, resulting in so-called drift. To prevent this, a periodic calibration is necessary. [47, 48]

The main purpose of a IMU is to help the FC, to estimate the aircraft's state, mainly the roll, pitch and yaw of an aircraft, enabling it to perform desired movements. [49]

1.4.8 Flight Controller

A FC is the "brain" of the aircraft, it basically monitors and controls everything the drone does. Physically, a FC is just a circuit board with various chips and sensors, and similarly to a PC motherboard, it enables the connection and communication of various components. All a FC does can be split into the following three categories:

Sensing The FC is connected to the IMU (often the IMU is part of the FC), and other sensors, such as a barometer or GPS. It reads values from the various sensors to estimate the aircraft's state, i.e., its altitude, position, orientation, and velocity.

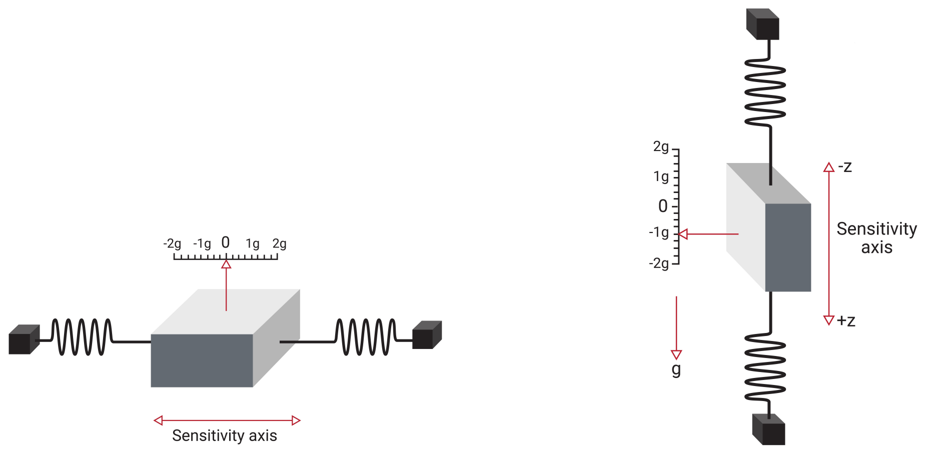
Controlling Since all of the multirotor's movements (translations and rotations), can be achieved by simply adjusting the speed of individual rotors. The FC uses the data from the IMU to estimate its state, to accordingly adjust rotor speeds (through the ESCs) to perform the desired movement.

Communicating Lastly, the FC takes care of transmitting and receiving information to the operator. The receiver/transmitter can be built-in or externally connected. Transmitted information may include information about the state of the aircraft (position, battery levels), received information might include movement commands. [50]

Chapter 2
Accelerometers

An accelerometer is a device enabling the measurement of acceleration, in the direction of the sensitivity axis.

One of the simplest models of an accelerometer, as seen in Figure 2.1a, is a mass-spring system. The mass is known as the proof mass, and the direction in which the mass is allowed to move is known as the sensitivity axis. When the system is subjected to a linear acceleration along the sensitivity axis, the acceleration causes the proof mass to shift to one side, with the amount of displacement being proportional to the acceleration. [51]



(a) Horizontal **(b)** Vertical

Figure 2.1 Simple Accelerometer Model [51]

If we take the system from Figure 2.1a and rotate it by 90 degrees in such a way that the sensitivity axis is aligned with the gravity vector as shown in Figure 2.1b, the proof mass will naturally deflect downwards due to gravity. Hence, the accelerometer measures both the linear acceleration due to motion, as well as the pseudo-acceleration caused by gravity. The acceleration caused by gravity is referred to as a pseudo-acceleration since it does not actually result in a change in velocity or position.

Notice, if the accelerometer is dropped, i.e. submitted to freefall, the springs will not deflect.

Hence the measured acceleration will be zero, although the actual acceleration is non-zero. [51]

2.1 Accelerometer classification

One of the possible ways to classify accelerometers is based on their output:

AC-RESPONSE accelerometers, as their name implies, have an AC coupled output. AC response accelerometers cannot be used to measure static or sustained acceleration such as gravity and constant centrifugal acceleration. Thus, they are suitable only for measuring dynamic events. [52, 53]

DC-RESPONSE accelerometers, on the other hand, are DC coupled and can respond down to zero Hertz. Therefore, they can be used to measure static, as well as dynamic acceleration. [52, 53]

2.1.1 AC-Response Accelerometers

2.1.1.1 Piezoelectric Accelerometers

The most common AC-response accelerometers employ the piezoelectric effect of certain materials, most often, PZT (Lead Zirconate Titanate). A piezoelectric element is placed between the housing and the proof mass, under the influence of acceleration, the material is deformed, displacing a charge, i.e. producing an electrical output. The electrical output is proportional to the force causing the displacement. [52]

Piezoelectric accelerometers are the most popular and widely used for industrial applications. They have very low noise levels and often offer performance superior to capacitive MEMS or piezoresistive accelerometers in all vibration and most shock applications due to their wide frequency response and good sensitivity. [53, 54]

Their most significant downside is that they are AC coupled, so they cannot measure the gravity vector or sustained accelerations and generally cannot measure vibrations below a few hertz. [53, 54]

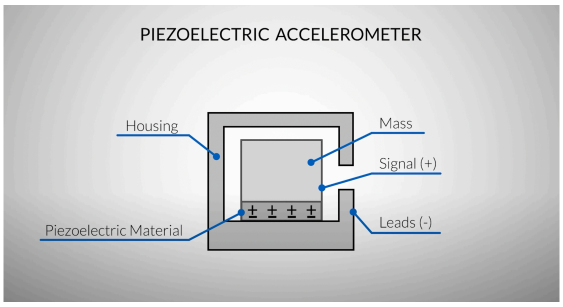


Figure 2.2 Piezoelectric accelerometer[55]

2.1.2 DC-Response Accelerometers

DC-response accelerometers usually use either the capacitive or piezoresistive sensing technology.

2.1.2.1 Piezoresistive Accelerometers

Piezoresistive accelerometers employ the piezoresistive effect, which is the change of electrical resistivity of a semiconductor when mechanical stress is applied. In contrast to the piezoelectric effect, the piezoresistive effect causes a change only in electrical resistance, not in electric potential. The output of most piezoresistive designs is generally sensitive to temperature variation. Therefore, it is necessary to apply temperature compensation to its output internally or externally. [52–54]

For their wide bandwidth capability (from 0 to several thousand Hertz), piezoresistive type accelerometers are suitable for impulse/impact measurements where frequency range and g levels are typically high. They are commonly used for industrial applications like automotive safety testing or weapons testing. [53, 54]

Compared to MEMS accelerometers, piezoresistive accelerometers are much more expensive, so they are generally not used for lower frequency and amplitude testing. [53]

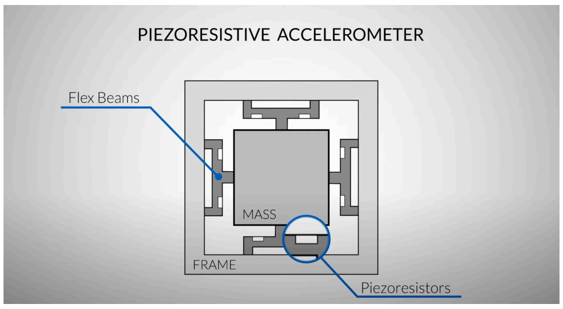


Figure 2.3 Piezoresistive accelerometer[55]

2.1.2.2 Capacitive Accelerometers

Capacitive based accelerometers use the changes in capacitance to determine acceleration.

A capacitor is a device that stores electrical charge. One of the simplest examples of a capacitor is a pair of parallel metallic plates separated by a dielectric medium (insulating material), as shown in Figure 2.4. When a voltage is applied to these plates, an electrical current flows, charging up one plate with a positive charge with respect to the supply voltage and the other plate with an equal and opposite negative charge. The capacitor's ability to store this electrical charge between its plates is proportional to the applied voltage. The more charge a capacitor is able to store, the greater is its capacitance.

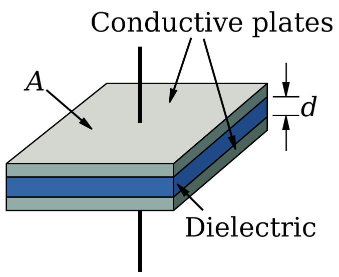


Figure 2.4 Parallel plate capacitor[56]

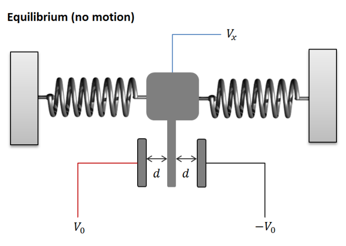
Formally, capacitance C of a capacitor is defined as $C = \frac{Q}{V}$, where Q is the charge stored on the capacitor, and V being voltage across the capacitor. Capacitance can also be determined from the geometry of the conductors and the dielectric properties of the insulator between the conductors. Given:

- the area of overlap of the two plates A
- the separation between the plates d
- and the dielectric material permittivity ε

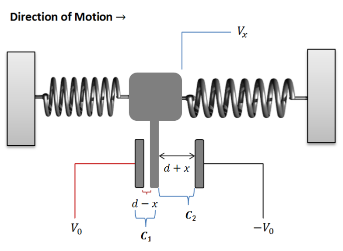
Capacitance can then be calculated as $C = \frac{Q}{V} = \frac{\varepsilon A}{d}$. It is easy to notice that the capacitance increases as the area gets bigger or the distance gets smaller.

Using this property, we can extend our original model of a spring-suspended mass 2.1, by adding a pair of fixed plates and attaching a single plate to the proof mass itself, two capacitors are created, as shown in Figure 2.5a.

When the system is subjected to acceleration, it causes the proof mass to shift to one side. As the proof mass moves, the distance between the fixed plates and the mass attached plate changes, as shown in Figure 2.5b. Hence the capacitance between the mass attached plate and one of the fixed plates increases, while the capacitance of the other capacitor decreases. [56, 57]



(a) Equilibrium state [56]



(b) Displaced state [56]

Figure 2.5 Capacitive accelerometer model

Capacitive type is the most common technology used for accelerometers today. The use of the micro-electro-mechanical systems (MEMS) fabrication technology has allowed capacitive accelerometers to dominate the consumer electronics markets due to their high sensitivity, good temperature performance, low fabrication costs, small size, and ease of integration into printed circuit boards. However, this class of low-price capacitive accelerometers typically suffers from poor signal to noise ratio and limited dynamic range. [52, 53, 58]

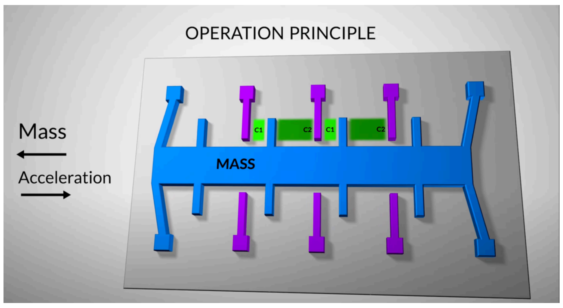


Figure 2.6 MEMS accelerometer model[55]

Chapter 3

Related Work

This chapter provides a brief overview of fault detection and diagnostics methods, and presents relevant work in the area of UAV fault detection and isolation.

3.1 Fault Detection and Identification

Fault detection (FD), sometimes also referred to as Fault detection and isolation (FDI), is a subset of fault detection and diagnostics (FDD), a subfield of control engineering, that deals with monitoring systems, identifying when a fault has occurred, and pinpointing the type of fault and its location.

3.1.1 Fault Diagnosis Methods

There are several ways to categorize FDD methods, one of the simplest and most common ones is to divide these methods into two main categories, model-based methods and model-free (also known as process history-based or data-driven) methods. [59–61] A more detailed overview of FD methods classification is shown in Figure 3.1.

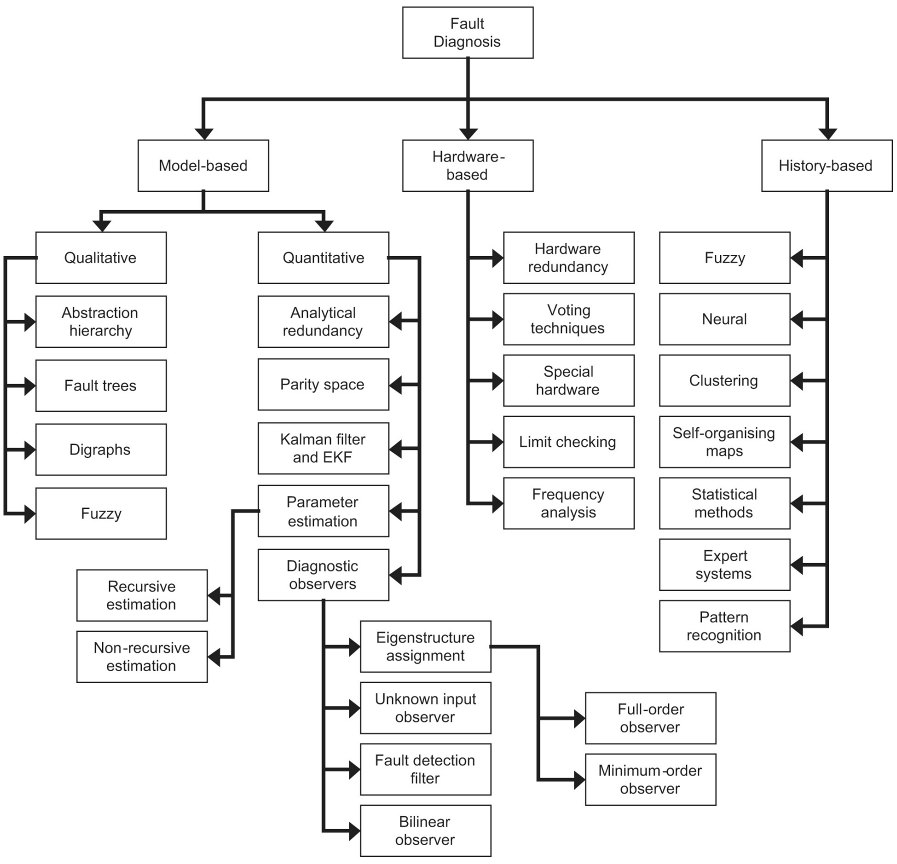


Figure 3.1 Classification of fault diagnosis methods[61]

3.1.2 Model-Based Methods

In model-based methods, some prior knowledge or fundamental understanding of the system is required. This knowledge or understanding is then used to create a suitable mathematical model of the system, based on the system's underlying physics. The constructed model is then used for identifying and evaluating differences, so-called residuals, between the actual operating state of the system (measured by monitoring the system), and expected operating state (output of the model).

Generally, model-based methods are broadly classified as qualitative or quantitative. [61–63]

3.1.2.1 Quantitative

Quantitative model-based methods utilize a model, where the input-output relationship of the system is expressed in terms of mathematical functions. Such methods mainly include state estimation based on filters and observers, and parameter estimation methods. [61, 62]

3.1.2.2 Qualitative

Qualitative models sometimes called knowledge-based methods, unlike quantitative models, where the model is expressed in terms of quantitative mathematical relationships, use the qualitative relationships or knowledge bases to determine the system's state. An example of such methods would be rule-based systems, expert systems or systems based on fuzzy inference. [61, 63].

3.1.3 History-Based Methods

On the other hand, we have process history-based methods, often called data-driven, for which no prior knowledge or understanding of the system is necessary. Instead of creating a model based on a physical understanding of the system, large amounts of collected historical data are used to derive a model, which relates measured inputs to measured outputs. The derived model can then also be used to generate residuals as in model-based methods. [61, 63]

The modelled relationship between measured inputs and outputs can be both statistical or non-statistical. Statistical methods include regression methods and stochastic process modelling. Examples of non-statistical methods (pattern recognition) are support vector machines, decision trees, or neural networks. [62, 63]

Data-driven methods can often be easily implemented if provided with a sufficient amount of historical data, especially compared to model-based approaches. Additionally, in some use-cases, data-driven methods are the only option, mainly when mathematical models of a monitored system are either too imprecise or computationally intensive, or they simply do not exist. [61, 62]

On the other hand, data-driven methods often employing ”black-box” models, such as neural networks, are unable to provide transparent reasoning ability over their predictions (unlike many qualitative based methods). Also, depending on the problem, the required amount of data to derive a model might be too big. The models are often system-specific, i.e., they work only for a system on which they were trained on, and often cannot be used to extrapolate beyond the range of the training data. [62, 63]

3.2 UAV Fault Detection and Identification

Fault detection and identification in UAVs is an area of active research. The majority of existing FDI solutions for multirotor UAVs fall into one or more of the previously mentioned categories, with data-driven methods dominating in recent years.

Jiang et al. identified the rotor's fault of a quadrotor by using airframe vibration signals. Their proposed algorithm uses a three-level wavelet packet decomposition (WPD) to analyze vibration signals. Then, the standard deviations of wavelet packet coefficients are used as feature vectors. Finally, a neural network is trained, which acts as a fault diagnostor to detect and identify rotor faults. A cellphone (iPhone) with an embedded triaxial accelerometer was fixed at the bottom of the quadrotor to collect acceleration data. A set of healthy, distorted and fractured propellers were used during testing. [64]

X. Zhang et al. proposed a FDI method using the onboard acceleration sensors. First, the three-axis accelerometer data of the quadcopter from the flight experiment is regarded as the airframe vibration signal. Second, the wavelet packet decomposition (WPD) method is employed to extract the data features and the standard deviations of the wavelet packet coefficients are employed to compose the feature vector. Finally, an FDI model is established by Long and

Short-Term Memory (LSTM) network that realizes the detection and identification of the propeller blade faults of a quadcopter. The LSTM network has shown to reach significantly higher accuracy compared to a standard backpropagation network. Data were acquired using a Parrot AR Drone for both healthy non-damaged propellers and damaged propellers (of various degrees of impairment). [65]

Bondyra et al. proposed a three-stage algorithm based on signal processing and machine learning to detect the occurrence of rotor fault and determine its scale and type. The method used acceleration data from the onboard IMU. A Falcon V5 quadcopter with different sets of healthy/-damaged rotors was used for data acquisition in a series of test flights. Three different methods of feature extraction were considered: fast Fourier transform (FFT), wavelet packet decomposition (WPD), and measuring the signal power in linearly spaced frequency bands (BP, BandPower). Then, three support vector machines (SVMs) classifiers were used to determine the occurrence, scale and type. [66]

Using onboard sensors is convenient, not only due to the limited space of the UAV airframe but also by not adding any extra weight. However, sensors vary across different UAVs, some might not have sufficient sampling frequency, and transmitting or processing data might be substantially more difficult than if an external device were to be used.

Alternatively, G. Iannace et al. used measurements of the noise emitted by an UAV to detect an unbalanced blade in a UAV propeller. The acoustic measurements were performed in an anechoic chamber. These data were pre-processed using frequency analysis; specifically, various bandpass filters were used to construct the feature vectors. Subsequently, a model based on neural networks was built to detect unbalanced blades in a UAV's propeller. [67]

This kind of approach has the advantage of not needing the sensor to be directly attached to the UAV, however, it is limited to being performed indoors.

Above mentioned studies, all proposed methods relying on supervised learning-based models; and although they generally perform well, they come with several drawbacks, the main one being the necessity of a finely-labelled training dataset. With respect to the above-mentioned drawbacks, K. H. Park et al. proposed an unsupervised learning based FD model utilizing a stacked autoencoder. The autoencoder was trained with data from safe UAV states, and its reconstruction loss was examined to distinguish the safe states and faulty states [68]

The downside of this approach is the limitation of not being able to distinguish between different types of faults or their severity.

Chapter 4

Analysis

The main goal of this work is to propose and implement an algorithm using vibration analysis, for the purpose of detecting faults/anomalies in the UAV 's behaviour. With this goal in mind, first we need to acquire a sufficient amount of sample data, only then can we proceed with proposing and implementing above mentioned algorithms. This chapter discusses the design of suitable test benches for acquiring data, which includes the selection of hardware, mainly the monitored systems (motors, drones) and the monitoring device (accelerometer). And the design and architecture of a data acquisition application required for reading and storing the measured data from the accelerometer.

4.1 Hardware

Since we have not found any suitable and publicly available dataset, we have decided to create our own. In order to create our own dataset we first and foremost need a measuring device capable of capturing vibrations, and secondly, a target device to be monitored, i.e., a device whose state we will try to predict/assess based on measured vibration.

Every UAV already has an onboard accelerometer in their IMU. However, they are most often surrounded by damping tapes, in order to reduce noise (the high and medium frequency vibrations, most often produced by motor induced vibrations), and to prevent possible damage caused by excessive vibrations to the onboard electronic, such as the FC, or IMU. [69–71]

The presence of damping materials, alongside the often low sampling frequency, makes these onboard accelerometers unsuitable for our purposes. Additionally, the onboard sensors may differ greatly, depending on the UAV, and depending on the manufacturer, extracting data from these sensors might be cumbersome or even outright impossible. Mainly for these reasons, we decided to use an external device for capturing the data. An external device should provide us with more accurate and undamped data. Additionally, capturing data from different devices should become easier, since the device needs to be simply detached from one device and attached to another one.

4.1.1 Blip

Blip is a development board based on the Nordic Semiconductor nRF52840 system on a chip (SoC), made by Electronut Labs. [72]

It has an embedded accelerometer sensor, specifically, the LIS2DH12 made by STMicroelectronics. Which is an ultra-low-power high-performance three-axis MEMS linear accelerometer, providing digital inter-integrated circuit (I2C) and serial peripheral interface (SPI) serial interface standard outputs. The LIS2DH12 has user-selectable full scales of ±2g/±4g/±8g/±16g, and is capable of measuring accelerations with output data rates from 1 Hz to 5.3 kHz. [73] It also has provisions for a microSD card slot, which makes it a complete and versatile development board. [72]

This specific device was selected mainly for: its relatively low price (50$), the accelerometer sensor it contains, and the support of Zephyr OS.

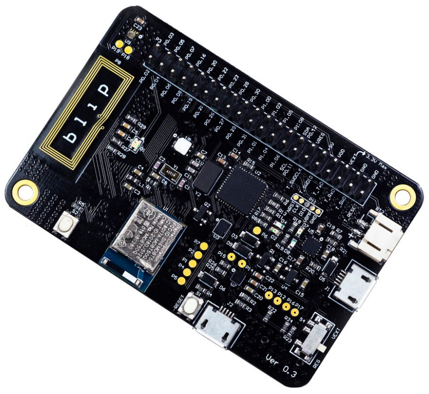


Figure 4.1 Blip development board [72]

4.1.2 Monitored Devices

All the hardware used for the dataset collection is listed below:

Single motor setup composed of the following parts:

- Antigravity MN4006 KV380 brushless motor
- HobbyKing 20A BlueSeries ESC
- Single propeller
- OWON ODP3031 programmable DC power supply

Quadcopter setup based on the Holybro X500 kit, with the following list of parts:

- 4 Holybro 2216 KV880 brushless motors
- 4 Holybro BLHeli S 20A ESCs
- Frame and a set of arms
- Other, for our purposes irrelevant parts

All devices were selected mainly for their availability, but also, as they should reflect common hardware used in the hobby drone community.

(a) Antigravity MN4006 KV380 [36] **(b)** HobbyKing 20A BlueSeries [74]

Figure 4.2 Single motor and ESC setup

Figure 4.3 Holybro x500 kit[75]

4.2 Datasets

With the ultimate goal of fault detection of UAVs in operation (in-flight), being rather complex, we decided to split the dataset acquisition process into three separate phases. With each phase getting increasingly more complex, and gradually building up to the end goal.

4.2.1 First Phase

In the first phase, we will attempt to detect a damaged propeller, in almost perfect conditions, completely devoid of any external variables (wind, other rotors, ...). Data will be acquired from a single, statically mounted motor, with an attached propeller.

This dataset should serve mainly as a proof of concept, to test whether faults are detectable in the first place. Additionally, various pre-processing and feature-extracting transformations can be tested here.

4.2.2 Second Phase

In the second phase, we will add another motor to test whether faults are still detectable when multiple motors are spinning. For this dataset, we will employ a statically mounted quadcopter, with two motors disabled (powering only 2 out of the 4 total motors).

4.2.3 Third Phase

The third and final phase extends upon the second one. The same statically mounted quadcopter will be used, this time with all 4 motors powered and rotating, to acquire data. Additionally, two different sets of propellers will be used in order to test the algorithm's ability to generalize.

4.3 Embedded systems

Embedded systems are most often defined as a combination of hardware and software, designed to perform a specific task or set of tasks. In comparison to general-purpose devices such as PCs or smartphones, that are designed to perform a wide range of non-predefined tasks. Embedded devices are most often based on some microprocessor (MPU), microcontroller (MCU), SoC, or field programmable gate array (FPGA), and often take the form of a single board design.

General-purpose devices typically run some sort of general-purpose operating system (OS). On the other hand, embedded systems are usually designed to perform only some pre-defined set of functions, they typically do not require a full-fledged OSs, and due to their limited resources (memory, computing power, …), they often are outright unable to run such OSs.

Embedded systems designed for simple tasks often do not need any OS, and run a simple program in an infinite loop. Systems that are designed for more complex tasks, such as smart/fitness devices or automotive subsystems will run some embedded OS, such as Embedded Linux, QNX, INTEGRITY, or VxWorks. Lastly, embedded systems running critical applications, such as pacemakers or vehicular control systems, are usually characterized by not only requiring logical correctness ($1 + 1 = 2$), but also temporal correctness (getting the results at the right time). Such requirements are most often satisfied by using a real time operating system (RTOS).

4.3.1 Zephyr

Zephyr is an open-source RTOS. It is relatively new, released in 2017, compared to other RTOSs, such as FreeRTOS, which was released in 2003. It has a broad SoC and development board support and is supported by many hardware architectures, including Intel x86, multiple ARM architectures, RISC-V, and many others.

Although relatively new, Zephyr has gained a lot of traction over the years. It has become one the most popular and well maintained free open-source RTOSs, Compared to other similar projects, it has a lead in terms of contributors and upstream commits, with many companies, such as Intel or Nordic Semiconductor, contributing to the development. [76]

Some of the features of Zephyr include:

- High configurability and modularity (allowing an application to incorporate only the capabilities it needs).
- Mostly static allocation of memory and resources
- POSIX compliance
- Support of both cooperative and preemptive threading
- Support of multiple scheduling algorithms
- Implementation of configurable architecture-specific stack-overflow protection
- Use of devicetree to describe hardware

4.3.1.1 Architecture

A high-level schematic of the Zephyr OS architecture is shown in Figure 4.4. It consists of 3 main layers as follows:

Kernel and Drivers consisting of, but not limited to:

- scheduler
- kernel synchronization objects and primitives
- kernel services
- low-level architecture and board support
- peripheral and hardware drivers

OS Services and Low-Level APIs consisting of platform-specific drivers, I/O APIs, file systems, logging and others

Application Services consisting of high-level APIs, enabling the applications to access standardized data such as JSON or CBOR, and networking protocols, such as HTTP [76]

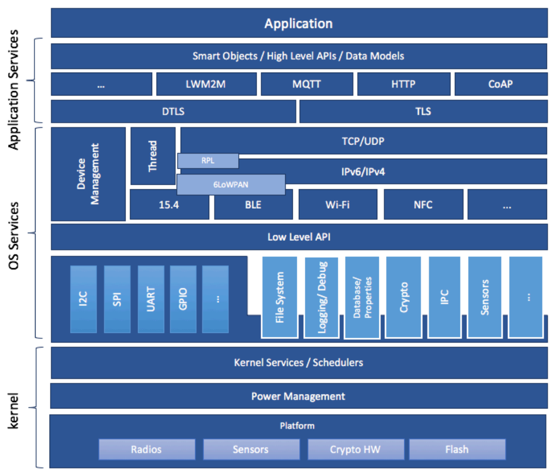


Figure 4.4 Zephyr System Architecture[77]

4.3.1.2 Kconfig

Kconfig was inspired by the configuration system used by the Linux kernel. The main purpose of Kconfig is to enable configuration without having to change any source code. It allows us to select what features should be included in the build of the OS image.

An example would be whether to add networking support, which drivers are needed by the application.

4.3.1.3 Devicetree

Devicetree is a hardware description language, but it can also refer to the hierarchical data structure that describes the hardware, using said language. We describe the hardware in order to allow the operating system's kernel to use and manage those components.

In Zephyr, we use the Devicetree to describe the hardware and its boot-time configuration, this configuration is then used during the build time, generating a macro-based API, through which we can interact with the hardware from the application's source code.

An example of what is devicetree used for would be what peripherals to include on a board, setting boot-time clock frequencies, configuring interrupt lines and so on.

4.4 Data Acquisition Application

The main purpose of this embedded application is to allow us, as easily and as consistently as possible, to acquire large amounts of data from various devices (motors, drones, ...).

4.4.1 Requirements

Reading data from accelerometer
The application has to extract the raw data from the sensors' registers, and interpret them correctly.

Writing or transmitting data
The application has to either permanently store the extracted data onto a medium, such as an SD card, or transmit the extracted data via a suitable interface, such as UART.

Calibrating and programming ESCs
The application has to enable the calibration and programming of multiple ESCs at once.

Communication interface
It also needs to provide a communication interface to allow communication with a client.

Test automization
Last it should allow automatization of the data acquisition process, by being able to sequentially perform multiple tests (measurements).

4.4.2 Architecture

We decided on a modular multi-thread application architecture, with a command-line interface for numerous reasons, with the main reasons being as follows:

- In order to ensure a consistent and reliable environment for data acquisition, that couldn't be achieved by, for example, manually setting up the motor speed using a servo controller.
- For the ease of use, to enable data acquisition of a large amount of data without constant supervision.
- To prepare for possible future extensions, such as sending data over Bluetooth.
- And lastly, to ensure the safety of the user, a command-line interface allows us to control the device and hence the UAV, from the safety of another room while measurements take place.

The application consists of multiple components and services. A brief overview of the application's architecture is shown in Figure 4.5.

The client communicates with the application using a command-line interface, and the tester module encapsulates, most of the functionality, and exposes a command-line API to the client.

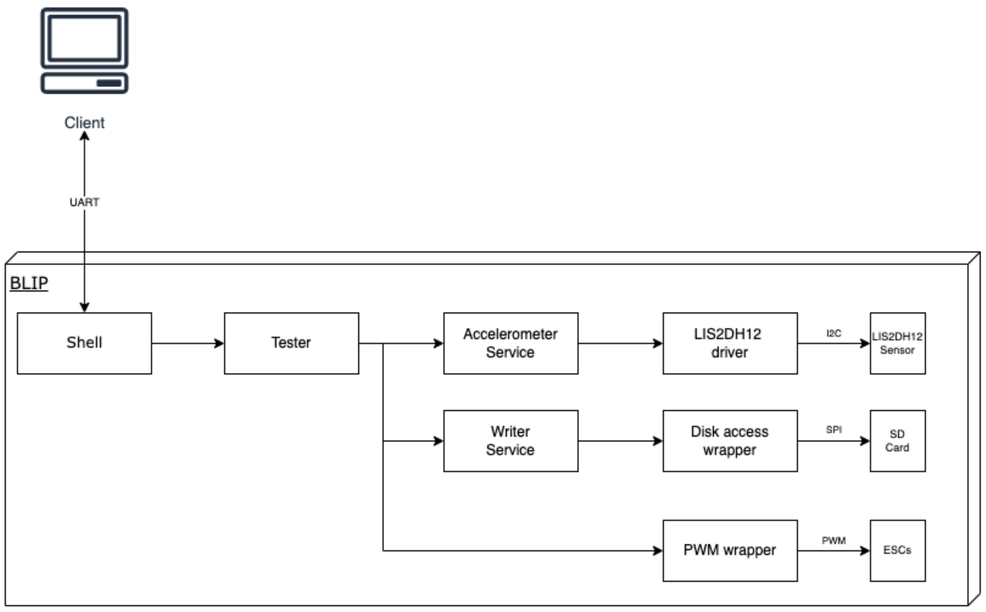


Figure 4.5 High level application architecture

4.4.3 Components

The components mainly include various wrappers around system or hardware-specific calls.

4.4.3.1 LIS2DH12 Driver

Since the Zephyr provided driver wasn't suitable for our purposes, we wrote our own simple custom implementation, in order to easily interact with the LIS2DH12 onboard accelerometer.

Except for extracting and reading data of the sensor, the custom driver should also enables us to configure the sensor's settings, such as the following:

Sampling rate The frequency at which the sensor measures the accelerations.

Sensitivity Determines the range of values (maximum acceleration value) the sensor is able to capture; values out of this range will get clipped.

Resolution In other words, bit depth, i.e., how many bits are used to encode measured data.

FIFO mode The sensor exposes measured data through a cyclical hardware FIFO. The FIFO mode determines whether the sensor should measure the data in the first place (to save energy), and, if so, in what manner it should do so.

Interrupt The sensor is capable of generating HW interrupts. Different interrupt triggers can be configured, such as: when acceleration surpasses a threshold, when a certain number of data samples are ready to be read in the FIFO, when FIFO is full, and data samples get overwritten.

4.4.3.2 Disk Access Module

Mounts the file system and handles common disk operations such as opening, closing and writing to files.

4.4.3.3 PWM Module

Provides API for setting motor speed, through an ESC, using values in the range 0-100, instead of having to specify the pulse-width modulation (PWM) parameters (period and pulse width).

4.4.3.4 Shell

The shell is a built-in feature of Zephyr, and it just needs to be included in the build by specifying a Kconfig option. Subsequently, new commands can be added by using the provided API.

4.4.3.5 Tester

The tester provides a command-line API for initiating tests (series of measurements), by calling the services and other components in a specific order. The tester also ensures all the peripherals and devices are ready, and enables the user to sequentially perform multiple tests in a row.

4.4.4 Services

Since the sensor's FIFO is of limited size (32 values per axis), writing or transmitting the data directly from the FIFO is not possible, even if a small sampling rate were to be used. Thus the data has to be copied to an intermediate data structure for further processing. In other words, it is a classical producer-consumer problem. The producer, in our case, is the accelerometer service, while the consumer is the writer service.

Each service runs in its own separate thread. In order to avoid repeated thread creation and termination, which both can be resource-intensive operations, the application creates each of the service threads only once. All the threads live for the entirety of the application's lifetime.

Propper thread suspension and resumption are achieved by using a combination of mutexes and conditional variables. Both the consumer and producer services employ a similar design, allowing the services to be suspended and resumed from the main thread, for example, by the tester module from within the command line. A brief and simplified design is shown in Figure 1.

Algorithm 1: Producer/Consumer Service

```
service_mutex ← mutex_init();
service_condvar ← condvar_init();
is_service_enabled ← false;

/* Called from another thread                                   */
Function enable_service():
    is_service_enabled ← true;
    condvar_signal(service_condvar);

/* Called from another thread                                   */
Function disable_service():
    is_service_enabled ← false;

/* Thread entry function                                        */
Function service_loop():
    while true do
        mutex_lock(service_mutex);          /* blocking wait, forever */
        if is_service_enabled = false then
            condvar_wait(service_condvar);  /* blocking wait, forever */
        end
        service_functionality();              /* perform some action */
        /* Give up mutex and allow other threads to run             */
        mutex_unlock(service_mutex);
        yield_thread();
    end
```

4.4.4.1 Accelerometer Service

The accelerometer service's main job is to read data from the sensor's FIFO, using the custom driver, into a ring buffer and subsequently, notify the consumer service.

4.4.4.2 Writer Service

The writer service awaits a notification, upon which it reads data from a ring buffer, and subsequently writes them to the disk.

A simplified sequence diagram of a single test is shown in Figure 4.6.

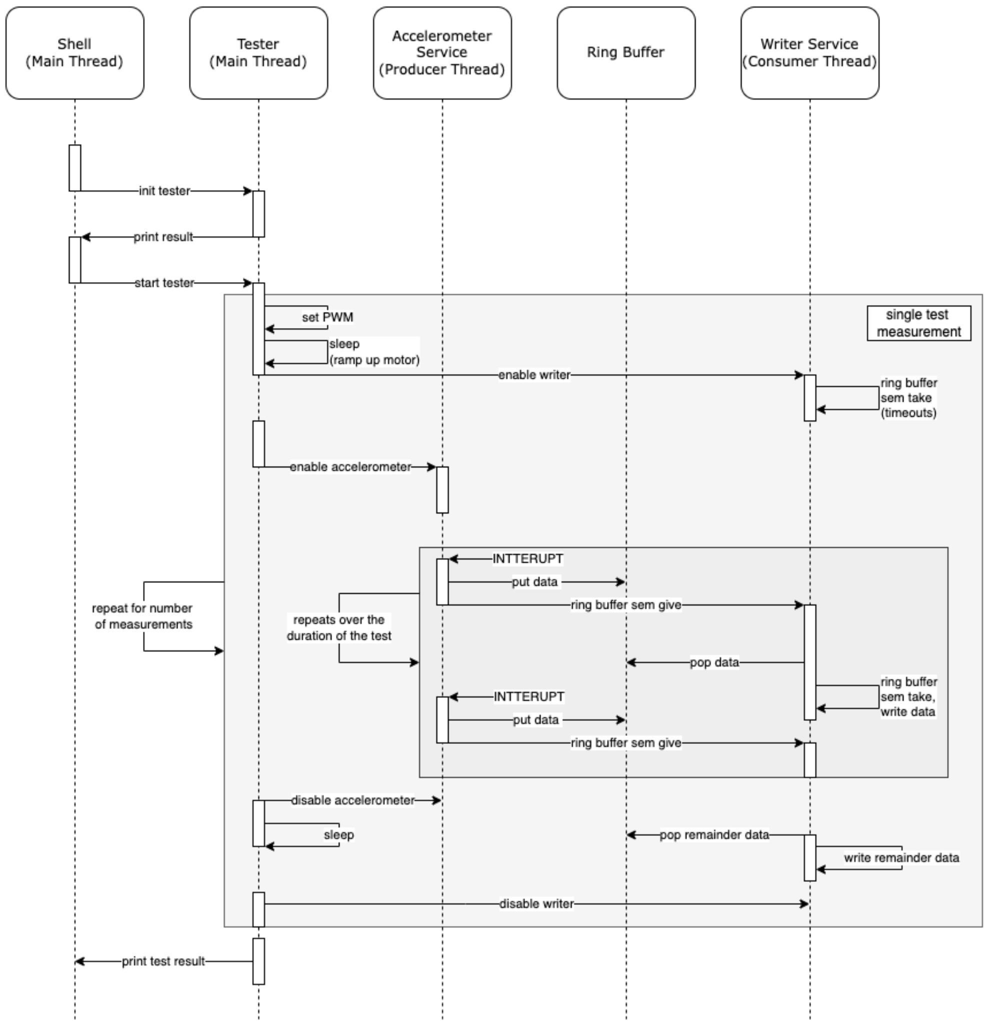


Figure 4.6 Test sequence diagram

4.4.5 Alternative Architectures

We additionally propose 2 alternative architectures, which both build on the original one. They both come with their own set of advantages and disadvantages.

4.4.5.1 Alternative Architectures 1

The first alternative modifies the writer service, instead of writing measured data to an SD card, they write measured data to the command line.

To enable filtering of the measured data from other command-line outputs, a prefix has to be attached to the data before transmitting. The client will then have to filter out the data.

The main advantage of this solution is the removal of the SD card, thus the files can be managed through a client-based script, instead of having to do all operations on-device. On the other hand, a custom client script has to be written to filter the measured data.

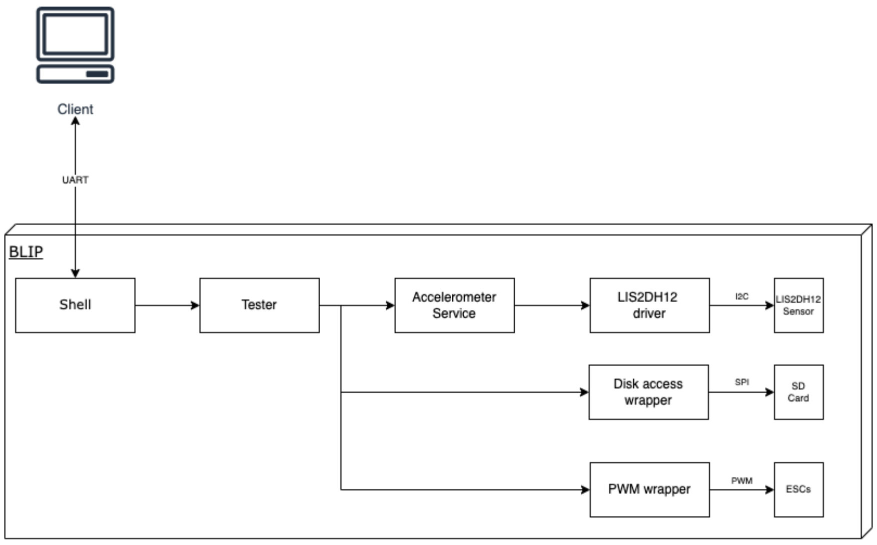


Figure 4.7 Alternative 1

4.4.5.2 Alternative Architectures 2

A second alternative outright removes the writer service. For the duration of a single measurement (ramping up the motor, starting data collection, stoping collection, stoping motor) all data get buffered. Once the data collection ends, all data get written to a file at once.

The main advantage of this solution is reduced complexity, through the removal of the consumer thread, leading to less overhead from context switching. The biggest downside is, however, the limitation on how much data can be buffered before we run out of physical memory.

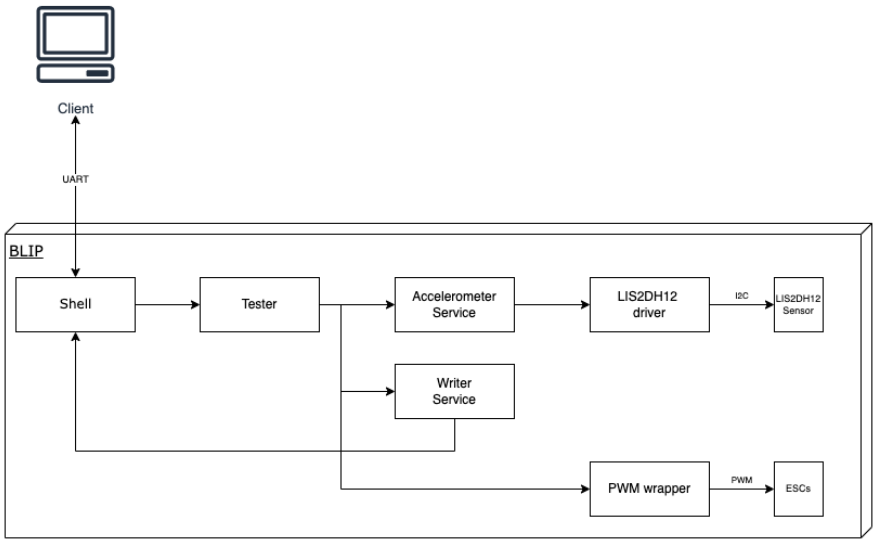


Figure 4.8 Alternative 2

4.5 Vibration Analysis

Vibrations are defined as mechanical oscillations around some equilibrium point. Vibration can be induced by various sources, including rotating machinery, rolling bearing elements, fluid flows, combustion events, structural resonance and many others. The oscillations may be periodic, such as the motion of a pendulum, or random, such as the vibrations of a flying aircraft.

Vibration analysis is then usually defined as the process of measuring vibrations, and using the obtained information to assess the state of the monitored system and its components. The two most common approaches include time domain vibration analysis, frequency domain vibration analysis or time-frequency analysis, which incorporates information from both the time and frequency domain. In the following chapter, we will describe some commonly used techniques and transformations, that are used for extracting information from vibrations.

Chapter 5

Theory

5.1 Signals and Digital Signal Processing

Signal is generally defined as anything that conveys information about a phenomenon. Most often signals are used as a description of how one parameter varies with another parameter. We represent them as functions of one or more independent variables describing the properties of the source. Most often they are functions of time, i.e., $y(t) = f(t)$; however that is only the most common type. Another type of signal commonly encountered are images, which are represented as functions of two spatial variables. [78, 79]

5.1.1 Signal Classification

Signals can be classified with respect to various properties, some ...properties are the following:

5.1.1.1 Causality

Signals that have a clear start/origin, i.e., $y(t) = 0; t < t_0$, are called **causal**, an example of causal signals are various biological signals, that have a start and an end. On the other hand with **non-causal** signals, we are unable to determine the origin.

5.1.1.2 Determinism

Signals can be also classified based on their ”randomnes”.

deterministic signals have deterministic values, i.e., the value of $y(t)$ is determined only by its input.

stochastic signals, are signals with some degree of uncertainity, they are described using probability functions

5.1.1.3 Periodicity

Deterministic signals are either **periodic**, or **aperiodic**. Periodic signals are defined as:

Definition 5.1.1 (Periodic signal)**:**
A signal for which exists a T, such that for all t:

$$y(t+T) = y(t)$$

Definition 5.1.2 (Period)**:**
The smallest T satisfying $y(t+T) = y(t)$ for all t is called the **period** of a signal.

Definition 5.1.3 (Frequency)**:**
Frequency is defined as the inverse of the period expressed in seconds,

$$f = \frac{1}{T}$$

The unit of frequency are hertz (Hz), meaning cycles per second.

Periodic signals can be further classified as **harmonic**, and **non-harmonic**. An example of harmonic signals are sine waves, of varying frequencies, while non-harmonic are more complex functions that can be decomposed into individual harmonics.

5.1.1.4 Continuity

Signals can be classified as either continuous or discrete, in both their amplitude and time (range and domain of a fucntion), continuous time signals, are expressed as functions with a continuous domains, while discrete time functions are often expressed as number sequences.

5.1.2 Analog-to-Digital Conversion

In order to digitally procecess signals they first have to be digitized, mathematically this procecess would be called discretization. The process of analog-to-digital conversion (ADC) ussualy done by a D/C converter and can be divided into two parts, quantization and sampling.

5.1.2.1 Quantization

Quantization is the process of converting the continuous amplitude of a function into a finite range of discrete values. The size of this range is refered to as **resolution**, and since most often it is determined by the number of bits used to represent the values, it is also often called the **bit-depth**.

The difference of the original signal and the quantized signal is called the **quantization noise**, which is used to calculate the signal to quantization ratio (SQNR).

Definition 5.1.4 (SQNR)**:**

$$SQNR = \frac{E(y_{in})}{E(y_{err})} = \frac{E(y_{in})}{E(y_{in} - y_{out})}$$

where E stands for the energy of a signal, and is defined as follows:

Definition 5.1.5 (Energy of continuous signal)**:**

$$E = \int_{t_1}^{t_1} |y(t)|^2 dt$$

Definition 5.1.6 (Energy of discrete signal)**:**

$$E = \sum_{n=1}^{N} |y[n]|^2$$

5.1.2.2 Sampling

Sampling is process of reducing a a continuous time signal to a discrete time signal.

Most often sampling is done in a uniform/equidistant manner, i.e., given a signal $y(t)$ and a sampling period T_s, the sampled discrete time signal will be defined as

$$y[n] = y(nT_s)$$

where $n \in \mathbb{N}$ is the n-th sample.

The sampling frequency (sampling rate) is defined as $f_s = \frac{1}{T_s}$.

An approriate sampling frequency is essential to properly discretize a signal, if the sampling frequency is too low, a so called **aliasing** might occur, where high frequency components of a signal will appear as low frequency ones, thus introducing "fake" frequencies, that are not acctually part of the sampled signal. If an undersampled digitzied signal were to be reconstructed (converetd back to analog), it would not match the original signal. The action of sampling a signal with a sufficiently high sampling frequency is called **proper sampling**. [78, 79]

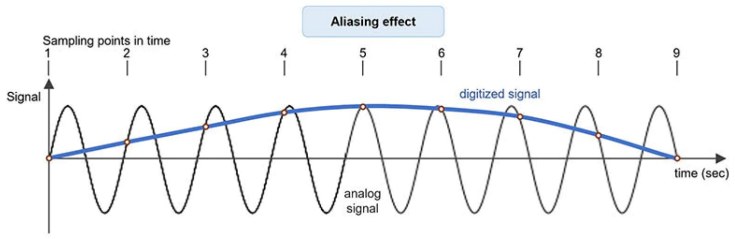


Figure 5.1 Aliasing effect example [80]

5.1.3 The Sampling Theorem

The sampling theorem, also known as Nyquist-Shannon theorem, states the minimum sampling frequency required to avoid aliasing, and may be stated as follows:

Theorem 5.1.7 (sampling theorem)**:**
A continuous signal can be properly sampled, if and only if, it does not contain frequency components above one-half of the sampling rate.

Or in other words, the sampling frequency has to be at least twice the the highest frequency contained within a signal. This minimum sampling frequency, is often refered to as the **nyquist frequency**.

5.1.4 Frequency Analysis

The goal of frequency analysis is to decompose signals into their individual harmonics.

5.1.4.1 Fourier Series

Fourier series allow us to approxiamte an arbitrary periodic function as a trigonometric series, more formally as follows.

Theorem 5.1.8 (Fourier Series)**:**
Given a periodic function f with a period of $2T$, f can be expressed as

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos \frac{k\pi x}{T} + b_k \sin \frac{k\pi x}{T} \right), \quad x \in \mathbb{R},$$

where the Fourier coefficients $a_0, a_k, b_k, for k \in \mathbb{N}$ can be calculated using the following formulas

$$a_k = \frac{1}{T} \int_{-T}^{T} f(x) \cos \frac{k\pi x}{T} \, \mathrm{d}x, \quad k \in \mathbb{N},$$

$$b_k = \frac{1}{T} \int_{-T}^{T} f(x) \sin \frac{k\pi x}{T} \, \mathrm{d}x, \quad k \in \mathbb{N}^*.$$

Additionaly, it can be proven under certaian conditions, that such an approximation converges.

5.1.4.2 Fourier Transform

The term Fourier transform is sometimes used ambiguously; however in general it can be broken into four categories, resulting from the four basic types of signals that can be encountered. A signal can be either continuous or discrete, and it can be either periodic or aperiodic.

Aperiodic and continuous signals: functions such as the Gaussian curve. The Fourier Transform for this type of signal is simply called the Fourier Transform.

Periodic and continuous signals: signals such as sine waves, square waves, and any waveforms that repeat themselves in a periodic pattern infinetly. This version of the Fourier transform is called the Fourier Series.

Aperiodic and discrete signals: signals which are only defined at discrete points between positive and negative infinity, and do not repeat themselves in a periodic fashion. This type of Fourier transform is called the Discrete Time Fourier Transform.

Periodic and discrete signals: lastly, signals which are discrete and repeat themselves in a periodic fashion from negative to positive infinity. This class of Fourier Transform is most often called the Discrete Fourier Transform. [79]

5.1.4.3 Discrete Fourier Transform

Definition 5.1.9 (Discrete Fourier Transform)**:**
Discrete Fourier transform transforms a sequence of complex numbers $\{x_n\} := x_0, x_1, \ldots x_{N-1}$ into a sequence of complex numbers $\{X_k\} := X_0, X_1, \ldots X_{N-1}$, where X_k is defined as follows

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n \cdot [\cos \frac{2k\pi n}{N} - i \cdot \sin \frac{2k\pi n}{N}] \\ &= \sum_{k=0}^{N-1} x_n \cdot e^{\frac{-2k\pi n}{N}}. \end{aligned}$$

Each of the complex number output sequence, corresponds to a frequency, amplitude, and phase on the resulting frequency spectrum.

5.1.4.4 Fast Fourier Transform

Fast Fourier transform (FFT) is an algorithm for fast calculation of the discrete Fourier transform (DFT). Although, there are multiple possible formulations of the FFT algorithm, the most commonly used one is the Cooley-Tukey algorithm, which is a divide-and-conquer type of algorithm. It essentially recursively breaks down the DFT calculation into two calculations of half the size, this allows the complexity of a regular DFT calculation to drop from $O(N)$ to $O(N \log_2 N)$, where N is the data size.

5.2 Machine Learning

Machine learning (ML) is a subset of artificial intelligence. A possible definition of ML, by Mitchell, is as follows:

Definition 5.2.1 (Machine Learning)**:**
A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E. [81]

In other words ML happens when a program, often called a model, learns from an experience of doing some task, that is measured/evaluated using some metric.

The process of the model learning from experience, is most often done by giving it some training data, it performing a task and subsequently being evaluated, is referred to as training a model.

The Task T

The tasks T beign solved by ML vary, but generally will fall into of the following categories:

Classification: assigning one of k categories to a given input.

Regression: predicting a number $x \in \mathbb{R}$ for a given input.

Clustering: dividing given input into clusters of similar examples.

Other: includes problems such as transcription, machine translation, structured prediction, anomaly detection, denoising, density estimation, and others. [82, 83]

The Experience E

Depending on the nature of gained experience E during the learning process, i.e., the type of data used by the model to learn, ML can be mostly classified into the following categories:

Supervised-learning: training data have associated labels (desired outcomes/targets), typical for classification or regression problems.

Unsupervised-learning: training data do not have any annotations/labels, typical for clustering or density estimation problems.

Reinforcement-learning: the learning is driven mainly using the performance metric instead of using data to learn.
For example, if a model were learning to play chess, individual moves would not be labelled as good or bad; instead, the model would be evaluated at the end of the game on how well it performed. [83]

The Performance Metric P

Performance metrics (Error measures) are used to evaluate the abilities of ML models, and are usually specific to the task T being solved by the model.

For classification problems, the simplest and most common performance metric is **accuracy**, which is defined as the proportion of examples for which the model has predicted correct output. We can also calculate the **error rate**, which is the complement to accuracy, i.e., the proportion of examples for which the model has predicted incorrect output. [83]

The choice of an appropriate performance metric is a task in and on of itself. Depending on the problem being solved, the selection might not be straightforward at all. When evaluating, for example, the performance of transcription or diacrtization, the accuracy can be measured at different levels of granularity (character-level, word-level, sentence-level). [83]

In other cases, such as in regression problems, accuracy can not be used, instead, the performance is measured using the **error**, which is simply the difference between a model's prediction and the ground truth. However, once again, depending on the problem being solved, different errors might be appropriate, commonly used regression metrics are mean squared error or root mean squared error, mean squared error will penalize bigger mistakes compared to root mean squared error. [83]

5.2.1 Generalization

ML is often associated with optimization, and although optimization techniques are often employed in ML, they are not the same. In the context of ML, given a training set (inputs and associated targets), the goal of optimization is to minimize the **training error**. That is, given an input and associated targets, match the output of a model to the provided targets as well as possible. [83]

However, that is not the main goal of ML. The main goal of ML is **generalization**, which is defined as the ability of a model to perform well on previously unseen inputs (data samples that differ from the ones used during training). [82, 83]

In other words, the aim is to minimize the **generalization error**, also called the test error. The generalization error is defined as the expected value of the error on new unseen inputs. It is typically estimated by measuring a model's performance on a test set of examples that were collected separately from the training set. [83]

A logical question that arises is how to affect the performance on the test set, if it can not be used during the training. The solution to this problem is to make an assumption about the test and train sets, specifically, we assume that the examples in each dataset are independent of each other, and that the train set and test set are identically distributed, and are drawn from the same probability distribution as each other. [83]

By making this assumption, if we were to sample a test and train dataset, for some fixed set of parameters, the expected test and train error of will equal, since both expectations were formed based on the same data sampling process. However, when training a model, the parameters are not fixed ahead of time, instead, the training set is sampled, and subsequently, the parameters are adjusted to minimize the train error, and only then is the test set sampled. Under this process, the test error will be greater or equal to the training error. [83]

5.2.2 Capacity, Overfitting and Underfitting

With the now established relationship between train and test error, another way to look at minimizing the generalization error is to minimize the train error while keeping the gap between test and train error as small as possible. [83]

These two goals relate to so-called **underfitting** and **overfitting**. Underfitting occurs when the model is not able to sufficiently minimize to train error, while overfitting occurs when the gap between train and test error gets too large. [83]

Whether a model overfits or underfits can be controlled by adjusting the model's **capacity**, which is informally defined as a model's ability to fit a wide variety of functions. [83]

5.2.3 Regularization

Regularization techniques aim to reduce generalization error (often increasing the training error). There are many regularization techniques, often they involve limiting the model's capacity, or somehow penalizing the model during training, to preferer certain models over others, a simple, commonly used technique is L_2 regularization, also known as weight decay, which penalizes models with large weights. [82, 83]

5.2.4 Hyperparameters

The process of training a model, involves finding the best parameters of a model (for example, finding the weights in linear regression). However, most ML models also have parameters that do not change during the training and need to be set before the training itself. These parameters are called **hyperparameters**. [82, 83]

Hyperparameters often alter the model's capacity, that is, they should not be chosen based on the training error, since the model would always preferer such hyperparameters that increase the model's capacity, leading to overfitting. A solution to this problem is to add another set of data, called the validation set (sometimes called the development set). For the same reasons, the test set can not be used to train the model (adjust parameters), and samples can not be used to select hyperparameters. Typically the validation set is created by splitting the train set into two sets, most often with an 80:20 ratio for the train and validation set, respectively. Since the validation set is used to select the hyperparameters, the validation set error will also underestimate the generalization error. [82, 83]

5.2.5 Parametric and Nonparametric Models

Parametric models are characterized by having a fixed and finite number of parameters (usually depending on the number of features), before any data is observed, i.e. the parameter count is fixed with respect to the training set. [82, 83]

On the other hand, there are nonparametric models. Contrary to their name, they do have parameters; however, the number of parameters is not fixed. Often they are determined by the size of the training data set, and therefore, the model size (parameter count) usually grows with the size of the training set. [82, 83]

5.2.6 K-Nearest Neighbors

KNN is one of the simplest nonparametric models, it can be used for regression, as well as classification problems. The main idea of kNN is simple. The training phase consists of simply storing the whole training set (the training set is the parameters), while the prediction is then made, by for a given input, finding the k most similar training examples, and based on those returning a target class or a number, depending on the type of problem. [82, 83]

Hyperparameters

The main hyperparameters affecting the prediction are:

k: the number of k most simmilar training examples to use for prediction;

metric: the function that determinse how similar/close two samples are;

weights: the impact of an example on the prediction results; most common weighting shcemes include:

- **uniform:** all k nearest neighbors are considered equally;
- **inverse** the weight of an exmple is inversely proportional to the distance from the input;
- **softmax** the weight is proportional to softmax of the negative distance from the input.

Formally a metric is defined as follows:

Definition 5.2.2 (metric)**:**
Let $d : X \times X \to \mathbb{R}$,
be called a **metric**, if for all $x, y, z \in X$ the following axioms hold:

i) $d(x, y) = 0 \Leftrightarrow x = y$ (identity)

ii) $d(x, y) = d(y, x)$ (symmetry)

iii) $d(x, y) + d(y, z) \geq d(x, z)$ (triangle inequality)

Commonly used metrics are often based on the L_p norm,

$$\|x\|_p := \sqrt[p]{\sum_{i=1}^{n} |x_i|^p} = \sqrt[p]{|x_1|^p + \cdots + |x_n|^p} \tag{5.1}$$

Sometimes, based on the nature of data, functions that formally are not metrics are used as well, such as the cosine similarity or dynamic time warping (DTW).

Prediction

Regression: given the k nearest neighbors and their respective target values t_i and weights w_i the prediction t is calculated as follows $t = \sum_i \frac{w_i}{\sum_j w_j} \cdot t_i$

Classification: if the weighting scheme is uniform, a simple max-voting approach is used, i.e., the most frequent class of the nearest neighbors is used as the prediction. Otherwise, the prediction is done similarly to regression, i.e., $\boldsymbol{t} = \sum_i \frac{w_i}{\sum_j w_j} \cdot \boldsymbol{t}_i$; however in this case is a vector $\boldsymbol{t} \in \mathbb{R}^k$ representing the distribution of the target classes (ussually simply one-hot encoded vectors). The predicted class will be the one with the maximum probability of the sum.

5.2.7 Estimators and Bias

In statistics, an estimator is a rule for calculating an estimate of a given quantity based on observed data. For example, the sample mean is a commonly used estimator of the population mean.

The bias of an estimator is defined as the difference between the expected value of the estimator and the true value being estimated, formally:

Definition 5.2.3 (Estimator Bias)**:**

$$B(\hat{\theta}) = \mathbb{E}(\hat{\theta}) - \theta \tag{5.2}$$

If the bias of an estimator is equal to zero, we call the estimator unbiased; otherwise, we call the estimator biased.

For example, suppose we want to estimate the mean result of a coin toss. We could estimate the mean by simply sampling from the distribution (by tossing the coin); although each estimation might be wrong most of the time (or all the time), since we are using the distribution itself to make an estimate, the expected value of this estimator will be exactly equal to the expected value of the distribution. Therefore the bias will be zero, and such an estimator will be unbiased.

5.2.8 Gradient Descent

Gradient Descent is a first-order iterative optimization method for finding the local minimum of a function. It is used mainly when the explicit solution does not exist, is too difficult to find or is simply too computationally expensive.

Definition 5.2.4 (Gradient Descent)**:**
Assuming we want to minimzes some error fucntion

$$\arg\min_{\boldsymbol{w}}(\text{Error}(\boldsymbol{w}))$$

We may use gradient descend, which iteratively adjusts the vector $\boldsymbol{w}$ in the following manner

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \nabla_{\boldsymbol{w}} \text{Error}(\boldsymbol{w})$$

where α is called the learning rate, and specifies the step size in each itteration.

Let us define the erorr function as the expected value of some loss function over an entire dataset

$$\nabla_{\boldsymbol{w}} \operatorname{Error}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}} \mathbb{E}(L(y(\boldsymbol{x};\boldsymbol{w}),t))$$

There are different variants of gradient descent, depending on how we calculate/estimate the expectation of the loss function.

Gradient Descent: Uses all training data to compute gradient.

Stochastic Gradient Descent Uses a single sample to estimate the gradient
$\nabla_{\boldsymbol{w}} \operatorname{Error}(\boldsymbol{w}) \approx \nabla_{\boldsymbol{w}} L(y(\boldsymbol{x};\boldsymbol{w}),t)$, for randomly choosen $(\boldsymbol{x},t)$ from the dataset. As we have shown on the example of a coin toss such an estimate although noisy will be unbiased.

Minibatch Stochastic Gradient Descent Is the middleground between regular and stochastic gradient descent. The expectation is estimated using B random independently sampled examples from the dataset.
$\nabla_{\boldsymbol{w}} \operatorname{Error}(\boldsymbol{w}) \approx \frac{1}{B}\sum_{i=1}^{B} \nabla_{\boldsymbol{w}} L(y(\boldsymbol{x}_i;\boldsymbol{w}),t_i)$, for randomly choosen $(\boldsymbol{x}_i,t_i)$ from the dataset.

Theorem 5.2.5 (Gradient Descent Convergence)**:**
Let us perform a stochastic gradient descent, using a sequence of learning rates α_i, If a loss function L is continuous then SGD converges to a local optimum almost surely if:

$$\forall i : \alpha_i > 0 \tag{5.3}$$

$$\sum_i \alpha_i = \infty \tag{5.4}$$

$$\sum_i \alpha_i^2 < \infty \tag{5.5}$$

Note, if the function is convex, the reached optimum will be global.

5.2.9 Multilayer Perceptron

Multilayer perceptron (MLP) also known as FNN can be considered an extension of generalized linear models such as linear regression, binary logistic regression or multiclass logistic regression. However, instead of a single generalized linear model, there are several in parallel, and additionally, they can be nested sequentially.

Neural networks are organized in so called layers, where each node in one layer is connected all the other nodes in the following layer via a directed edge, and every edge has an associated weight.

The first layer is called an input layer and represents input variables, and the last layer is called an output layer. The remaining layers in between are called hidden layers. While the size of the input and output layers are determined by the shape of the input data and output targets respectively, the number and size of hidden layers can be freely selected.

A simple architecture consist of only a single input layer and a single output layer with activation function a is shown in Figure 5.2. The value of each output node is computed as follows:

$$y_i = a\left(\sum_j x_j w_{j,i} + b_i\right)$$

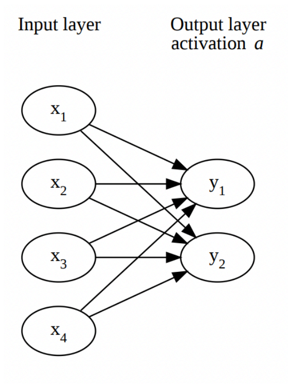


Figure 5.2 Neural network without hidden layer

When extending the model by adding a hidden layer with an activation function f the computation is perfomed analogically, i.e.,

$$y_i = a(\sum_j h_j w_{j,i}^{(y)} + b_i^{(y)})$$

$$h_i = f(\sum_j x_j w_{j,i}^{(h)} + b_i^{(h)})$$

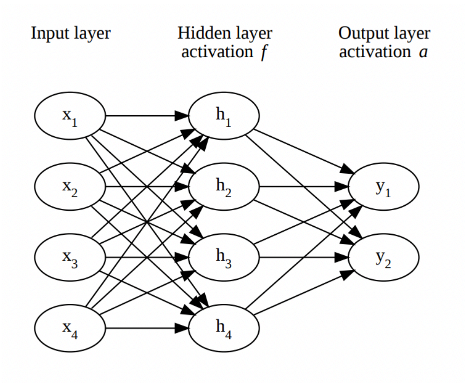


Figure 5.3 Neural network with hidden layer

The addition of additional hidden is then analogous.

5.2.9.1 Activtion Functions

The output layer activation functions are ussually determined by the output distribution we want to generate. In case of regression we use identity, in case of binary distribution we use a sigmoid, and lastly in the case of categorical distribution we use a softmax function.

The selection of the activation functions in the hidden layer are slightly less straightforward, but in general they must be some type of non-linear function. Historically *tanh* was used, however nowdays the most commonly used function is the *ReLu* (rectified linear unit).

5.2.9.2 Training

To train a FNN a gradient descent can be used

Algorithm 2: MPL SGD

```
Input: NN computing function $f(\boldsymbol{x}, \theta)$
Output: updated parameter $\theta$
while Stop criterion is not met do
    sample training examples $(\boldsymbol{x}_i, y_i)$ $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{L(x_i;\theta),y_i}}\theta$ $\theta \leftarrow \theta - \alpha\boldsymbol{g}$
end
```

Chapter 6

Implementation

This chapter describes the process of dataset acquisition, and implementation of the fault detection algorithms.

6.1 Data Acquisition Application

The implemented Zephyr RTOS based application was designed with specifically the Blip development board; however, it should be relatively simple to make adjustments to the application, to run it on other similar boards.

6.1.1 Installation

In order to develop, build and flash a Zephyr application, firstly a development environment has to be created. The process is described in detail by the official Zephyr documentation. It consists of the following steps:

1. Install host dependencies, such as python or cmake, required for creating a development environment.
2. Creating a python virtual environment and installing west via pip. West is an all-around tool used for various tasks, such as for managing the Zephyr workspace, managing the Zephyr repository and its submodules, or for building and flashing the Zephyr applications onto a device.
3. Create a new Zephyr workspace using west and clone the Zephyr repository into it.
4. Install Zephyr dependencies and required submodules.
5. Install a Toolchain or a cross compiler depending on the host platform.

Once the development environment is set up, our application should be placed into the same workspace, next to the Zephyr repository. The resulting file structure should look as shown in Figure 6.1, where ”zephyrproject” is the name of the west created workspace.

The application can then be built and flashed onto the device using west, assuming the host system has an appropriate compiler.

```
zephyrproject/
├── .west/
├── bootloader/
├── modules/
├── tools/
├── zephyr/
└── our_application/
```

Figure 6.1 Zephyr workspace folder structure

6.1.2 Application Implementation

In the Analysis Chapter we proposed in total 3 different possible architectures, all of which we sequentially implemented; however, due to various reasons, only one of them proved to be reliable enough to be used for the final data acquisition. Additionally, although the LIS2DH12 sensor supports various sampling rates, we were unable to implement a reliable way of acquiring data with sampling rates above 400Hz.

6.1.2.1 Architecture 1 Implementation

At first, we implemented our first proposed design, i.e., the producer/consumer design, where measured data are continuously stored onto an SD card. However, for yet to be explained reasons, the action of writing data to an SD card proved to be very problematic. Multiple SD cards were tested from various manufacturers, and with different specifications; however, after many trials and error, we were unable to identify the cause of the problems. The main problems included write errors or mounting errors.

Additionally, the overhead from context switching and delays caused by writing data to the SD card relatively often lead to FIFO overflow of the LIS2DH12 sensor, i.e., the application was unable to read and write data from the sensor fast enough, leading to loss of data.

6.1.2.2 Architecture 2 Implementation

Therefore we implemented our second proposed design, where the data are transmitted to a client; however, it also proved to be insufficient. The communication with the client was done over the universal asynchronous receiver/transmitter (UART) protocol, which is a communication protocol, for asynchronous serial communication, with configurable transmission speeds. Although UART has a simple error checking mechanism, it does not provide error correction. We tested various transmission speeds, but all of them resulted in a significant amount of transmission errors, i.e., corrupted or lost data.

6.1.2.3 Architecture 3 Implementation

Lastly, we further inspected the SD card errors. A simple stress test consisting of a series of continuous writes into single and multiple files were implemented, in order to identify and locate the error causes. While the mounting errors of certain SD cards remain unsolved, we were able to locate the cause of the write errors. By making a small adjustment in the Zephyr's "sdmmc_spi" driver code, we significantly lowered the frequency of write errors, making the writing to an SD card a viable solution for storing data.

We implemented the last proposed design, where the data are buffered for the duration of the measuring, and then all at once written to an SD card. This solution proved to be the most reliable one for our purposes; however, it poses severe limitations on the amount of data that can be measured.

6.2 Test Benches

Two test benches were designed and built, one for each of the devices used for data acquisition.

6.2.0.1 Test Bench 1

The first test bench was designed for the, Antigravity MN4006 KV380 brushless motor. We used an aluminium profile with C-shape railings attached to a wooden board as the basis for our test bench. Subsequently, a carbon fibre arm alongside the attached motor was mounted to the aluminium profile with the help of a 3D printed loop. The Blip development board was then attached to the carbon fibre arm using electrical tape. The entire test bench is shown in Figure 6.2.

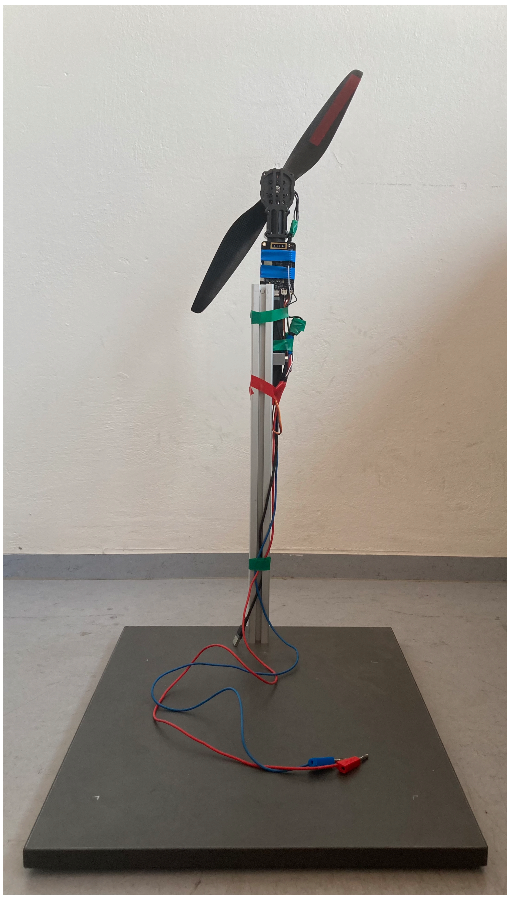


Figure 6.2 Single motor test bench

6.2.0.2 Test Bench 2

The second test bench was designed to accommodate the Holybro X500 quadcopter, we opted for a similar design approach as with the previous test bench. A single wooden board with four threaded inserts was used as the base of the structure. Four 3D printed grips were used to hold the quadcopter's landing gears and attach them to the wooden board. The Blip development board was then once again attached using electrical tape, to the top of the quadcopter's flight controller. The entire test bench is shown in Figure 6.3.

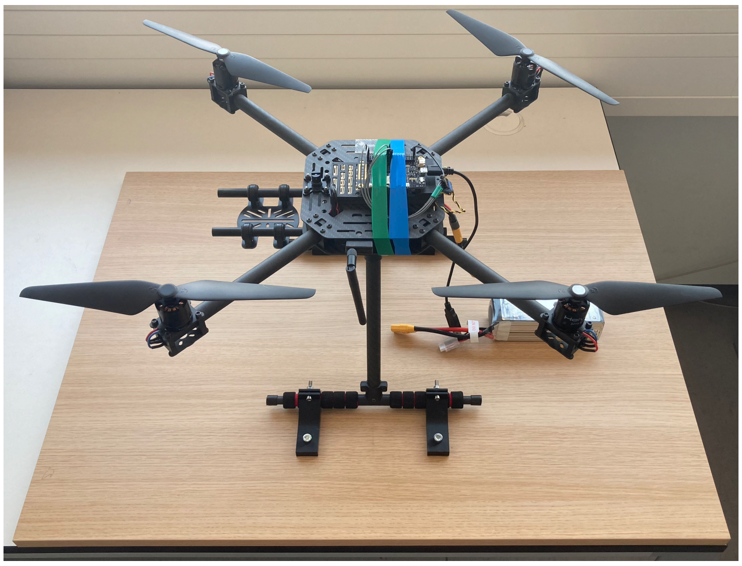


Figure 6.3 Quadcopter test bench

6.3 Data Acquisition

In total, three datasets were created. All of which employed the same methodology (sequence of steps). Firstly the devices needed to be calibrated (and programmed), secondly, the tester module of the Zephyr application had to be configured, and lastly, the individual datasets were acquired.

6.3.1 ESC Calibration and Programming

Usually, ESCs are calibrated using a controller, the calibration's purpose is to mainly match the minimum and maximum position of a controller throttle stick, to the minimum and maximum speed of the motor; and secondly, to ensure all ESCs spin the motor at the same speed, upon receiving the same PWM signal.

In our case, the calibration was done via the Zephyr application, through the command-line interface, mainly because the ESCs were to be controlled by the application, secondly, because the calibration and subsequent setting of the motor speed should be more precise, than if calibrated through an external device.

ESCs can also be programmed, to set the behaviour of an ESC, in our case, we simply resetted the ESCs into factory settings, not changing any setting.

6.3.2 Tester configuration

The configuration is done by setting designated macros with appropriate values. The most important parameters affecting the data acquisition are the following:

SET_COUNT: an integer determining how many sets of measurements will be performed;

MOTOR_SPEEDS: a list of numbers in the range from 0 to 100, representing a single set of measurements, i.e., determining how many measurements should be performed, in what order, and at what percentage of maximum speed

MEASUREMENT_DURATION_SEC: duration of an individual measurement (for how long are data sampled) in seconds

MEASUREMENT_PAUSE_DURATION_SEC: duration from beginning to spin the motor to initiating the measurement in seconds.

The sampling rate was also intended to be configurable; however, at the moment, the application is unable to reliably and continuously perform measurements using high sampling rates, which is why the default sampling rate was fixed at 400Hz, the maximum possible, stable value.

Once the configuration is set, the application needs to be built and flashed onto the device. Using the command-line API, the data acquisition process can be initiated, by the client, without having to physically interact with the device.

6.3.3 Dataset 1

The first dataset was acquired using the first test bench, alongside the Antigravity MN4006 KV380 motor. The motor was powered by an owon ODP3031 programmable DC power supply, set to 20V.

Specific parameters are listed in the following Table 6.1.

Acquisition parameters	
Sampling rate (Hz)	400
KV	380
Voltage	20
Motor speeds (% of maximum speed)	0, 10, 15, 20, 25, 30, 35, 40, 45, 50
Measurement duration (seconds)	8

Table 6.1 Dataset 1 acquisition parameters

In total, 3000 measurements were taken, with each measurement resulting in a single csv file, consisting of 3 columns (one for each axis of the accelerometer). The measurements were acquired using a ”healthy” (reference, non-damaged) propeller and using various sized strips of electrical tape attached to the bottom of the propeller, in order to create a disbalance, simulating a damaged propeller.

Composition		
Rotor state		Measurement per each sampled speed
Healthy		120
Simulated damage	single small tape	60
	double small tape	60
	single big tape	60
Total		300

Table 6.2 Dataset 1 composition

6.3.4 Dataset 2

The second dataset was acquired using the second test bench, alongside the Holybro X500 quadcopter, with two out of total 4 motors disabled, specifically, a pair of diagonally placed motors were not powered. The motors were powered using a 4S 6700mAh LiPo battery. Specific parameters are listed in the following Table 6.3.

Acquisition parameters	
Sampling rate (Hz)	400
KV	880
Voltage	4S (14.8-16.8V)
Motor speeds (% of maximum speed)	0, 10, 15, 20, 25, 30, 35, 40
Measurement duration (seconds)	8

Table 6.3 Dataset 2 acquisition parameters

Similarly to the first dataset, the measurements were taken using a ”healthy” propeller and simulated damage using electrical tape. In total, 3080 measurements were acquired.

Composition		
Rotor state		Measurement per each sampled speed
Healthy		110
Simulated damage	single small tape	100
	double small tape	78
	single big tape	98
Total		385

Table 6.4 Dataset 2 composition

6.3.5 Dataset 3

The last dataset was acquired using the second test bench as well; however, this time all four motors were powered during the measurements. Specific parameters are listed in the following Table 6.5.

Acquisition parameters	
Sampling rate (Hz)	1344
KV	880
Voltage	4S (14.8-16.8V)
Motor dpeeds (% of maximum speed)	10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
Measurement duration (seconds)	4

Table 6.5 Dataset 3 acquisition parameters

In total, 2475 measurements were acquired. In addition to the healthy and simulated damage measurements, measurements with actual mechanical damage were acquired as well.

Composition		
Rotor state		Measurement per each sampled speed
Healthy		45
Simulated damage	single small tape	40
	double small tape	40
	single big tape	40
Real damage	small cuts	40
	medium cuts	20
Total		225

Table 6.6 Dataset 3 composition

The strips of tapes alongside the damaged propeller are shown in Figure 6.5.

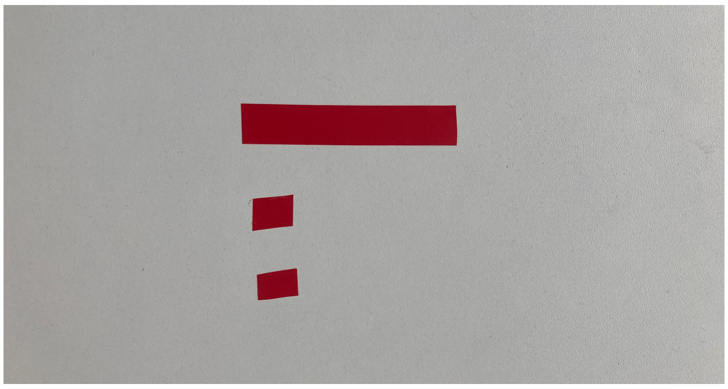
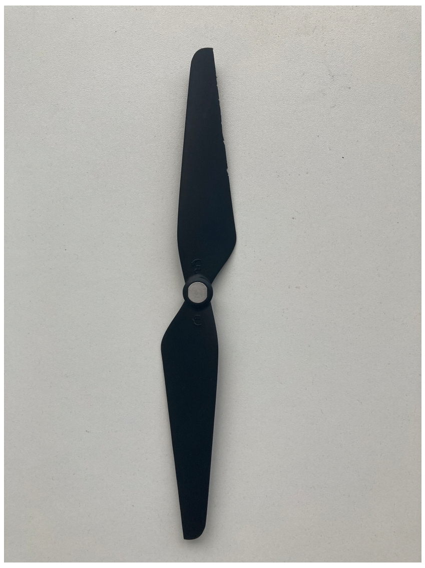


Figure 6.4 Varying sized electrical tape strips (8x1.5cm and 1x1.5cm)


(a) Small mechanical damage

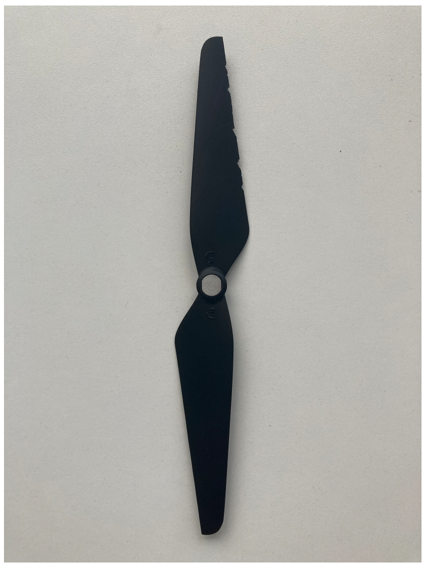

(b) Medium mechanical damage

Figure 6.5 Damaged propellers

6.3.6 Data Exploration

After analyzing the first two datasets, by performing spectral analysis, we made a few observations, namely:

- When calculating the expected RPM of the motor and dividing it by 60, the principal frequencies of signals matched that number (with a small deviation caused by the propeller).
- The frequency of higher motor speeds naturally shifts to the right, and the magnitudes increase as the RPMs increase.
- The main difference in frequency spectrums of healthy and damaged rotors were mainly the magnitudes.
- Damage propellers cause higher signal energy compared to healthy propellers rotating at the same speed.

Additionally, after the second dataset, we also observed non-negligible amplitudes in the high frequencies indicating the occurrence of spectral leakage. It is for that reason that we decided to sample the third dataset using a higher sampling rate, at the cost of acquiring fewer measurements, since the data acquisition application does not work reliably with these high sampling rates. However, even then, it is likely that spectral leakage is occurring, possibly caused by the structural resonance of the quadcopter's frame; however, we did not have the means to verify that.

6.4 Fault Detection Algorithms

Two types of classifiers were used and tested, namely a kNN classifier and a FNN. KNN was selected for its simplicity, as it has few hyperparameters and can serve as a good baseline, while the FNN was selected as it is generally well suited for unstructured data and generally shows good generalization capabilities if provided with sufficient data. Since we encountered many problems during the data acquisition process and did not collect as much data as we expected, data augmentation techniques were employed to prevent overfitting.

6.4.1 Preprocessing Pipeline

A preprocessing pipeline based on the Scikit-learn API was implemented, consisting of multiple steps, with the main ones being the following:

Normalization: scaling all signal amplitudes to a fixed range using min-max normalization

Detrending: removing the mean value from each signals axis

Axis Selection: selecting which axes to drop and which to keep

Augmentation: augmenting the input signal by either adding noise or performing a shift in all axes

Transformation: applying a transformation on each axis of the signal separately

Flattening: reshaping each transformed signal into a one-dimensional vector

Some of the mentioned steps were used as hyperparameters during the training.

6.4.2 Data Splitting

On each dataset 3 train test splits were conducted, specifically:

- train-test split all data equally
- train-test split excluding samples of certain fault types from the training set, meant to test the generalization capabilities on unseen faults
- train-test split excluding samples of certain motor speeds from the training set, meant to test the generalization capabilities on unseen devices (such as motors with higher KV rating)

6.4.2.1 Training

All data were labeled as either faulty or healthy, models were trained to perform binary classification. On both models, some form of grid search was applied to find the best hyperparameters. In the case of the kNN classifier, the Scikit-learn implementation was used, while for the TensorFlow FNN, the grid search was performed manually. Additionally, data augmentation was performed in between each epoch to prevent overfitting caused by the lack of data. Accuracy was used as our main metric for model validation. The tuned hyperparameters are listed in the following Tables 6.7 and 6.8.

Hyperparameters	
Normalization	none, min-max(-1,1)
Axis Selection	[0], [1], [2], [0,1,2]
Transformation	none, FFT, root-mean-square energy (RMSE)
Number of Neighbours	3, 7

Table 6.7 kNN pipeline hyperparameters

Hyperparameters	
Optimizer	Adam
Learning rate	0.001
Architecture	2 hidden layers of 128 neurons with 25% dropout
Epochs	30
Batch size	32
Normalization	none, min-max(-1,1)
Axis selection	[0,1,2]
Transformation	FFT, RMSE
Gausian noise standard deviation	1, 0.025
Gausian noise probability	0.5, 0.75

Table 6.8 FNN pipeline hyperparameters

Chapter 7

Evaluation

This chapter presents the performed experiments and evaluates our algorithms.

K-Nearest Neighbors pipeline performance

The pipeline based on kNN was able to achieve test accuracy of over 94% on all three datasets, when trained on samples of all fault and motor speed classes. On the third dataset, when real damage samples were completely excluded from training, the model still reached about 90% accuracy. However, when excluding data of different motor speeds, the model rarely reached an accuracy of over 60%. In almost all the performed tests, the best model was using RMSE as the transformation. Indicating that prediction based solely on the total energy of a signal can yield good results if a sufficient amount and variety of data are sampled.

Neural Network pipeline performance

The FNN pipeline performed in general in most cases either the same or worse compared to the kNN pipeline. This was caused mainly due to the lack of data, which often led to large overfitting of the model. Using data augmentation, specifically adding random Gaussian noise to each sample at the start of each epoch, improved the accuracy greatly; other types of augmentation, such as shifting, did not yield significant changes in performance. In contrast to the kNN pipeline, best results were achieved using the FFT transformation.

Conclusion

The main goal of this thesis was to design and implement an algorithm for fault detection of UAVs using vibration analysis. In the first two chapters, we did a brief overview on UAVs and accelerometers. In the next chapter, we summarized some of the previous relevant work on the topic of fault detection of UAVs. Next, we designed an embedded application for the purposes of data collection, and designed two test benches. In total, three datasets were collected; however, due to hardware issues, the datasets were all limited in size. Two machine learning based classifiers were tested on all three datasets, both showing good performance when trained on samples of all classes. When exposed to previously unseen faults depending on the scale

There are many opportunities for future improvements. Mainly the data acquisition process is at the moment limited by not only by the maximum sampling rate but also by the maximum duration of a single measurement. Different development boards supporting Zephyr OS might yield better results.

Bibliography

[1] *Unmanned Aircraft Systems: UAS*, ser. ICAO Cir 328-AN/190. Montréal: International Civil Aviation Organization, 2011, 38 pp., ISBN: 978-92-9231-751-5.
[2] "A Brief History of Drones," Imperial War Museums. (), [Online]. Available: https://www.iwm.org.uk/history/a-brief-history-of-drones (visited on 03/17/2022).
[3] J. Alkobi. "The Evolution of Drones: From Military to Hobby & Commercial," Percepto. (), [Online]. Available: https://percepto.co/the-evolution-of-drones-from-military-to-hobby-commercial/ (visited on 05/06/2022).
[4] L. Tania, "Civil and military drones," p. 12,
[5] "European_Drones_Outlook_Study_2016.pdf." (), [Online]. Available: https://www.sesarju.eu/sites/default/files/documents/reports/European_Drones_Outlook_Study_2016.pdf (visited on 05/06/2022).
[6] "Classification of the Unmanned Aerial Systems | GEOG 892: Unmanned Aerial Systems." (), [Online]. Available: https://www.e-education.psu.edu/geog892/node/5 (visited on 03/16/2022).
[7] M. Hassanalian *et al.*, "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences*, vol. 91, pp. 99–131, May 2017, ISSN: 03760421. DOI: 10.1016/j.paerosci.2017.04.003. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0376042116301348 (visited on 04/19/2022).
[8] "2475-dron-dji-matrice-600-pro-zesikma.jpg (950×712)." (), [Online]. Available: https://dronpro.cz/data/images-xl/2475-dron-dji-matrice-600-pro-zesikma.jpg (visited on 05/08/2022).
[9] "Ar_wing_mini.jpg (500×266)." (), [Online]. Available: https://www.buildyourowndrone.co.uk/media/catalog/product/cache/1/image/9df78eab33525d08d6e5fb8d27136e95/a/r/ar_wing_mini.jpg (visited on 05/08/2022).
[10] "5df993be8c1eb.jpg (1000×1000)." (), [Online]. Available: http://www.dronefromchina.com/Uploads/5df993be8c1eb.jpg (visited on 05/08/2022).
[11] G. Staff. "Types of Drones and UAVs," Tyto Robotics. (16:39:14 -0500), [Online]. Available: https://www.tytorobotics.com/blogs/articles/types-of-drones (visited on 04/19/2022).
[12] H. Yang *et al.*, "Multi-rotor drone tutorial: Systems, mechanics, control and state estimation," *Intelligent Service Robotics*, vol. 10, no. 2, pp. 79–93, Apr. 1, 2017, ISSN: 1861-2784. DOI: 10.1007/s11370-017-0224-y. [Online]. Available: https://doi.org/10.1007/s11370-017-0224-y (visited on 04/21/2022).
[13] *UAVs for Land Rights A Guide to Regulatory Practice.*
[14] D. Kotarski *et al.*, "A Modular Multirotor Unmanned Aerial Vehicle Design Approach for Development of an Engineering Education Platform," *Sensors*, vol. 21, no. 8, p. 2737,

Apr. 13, 2021, ISSN: 1424-8220. DOI: 10.3390/s21082737. [Online]. Available: https://www.mdpi.com/1424-8220/21/8/2737 (visited on 04/20/2022).

[15] "Multirotor Drones | Multirotor & Multicopter Manufacturers & Suppliers," Unmanned Systems Technology. (), [Online]. Available: https://www.unmannedsystemstechnology.com/category/supplier-directory/platforms/multirotor-drones/ (visited on 04/20/2022).

[16] R. Niemiec *et al.*, "Multi-rotor Coordinate Transforms for Orthogonal Primary and Redundant Control Modes for Regular Hexacopters and Octocopters," p. 18,

[17] A. Abdul Salam *et al.*, "Nonlinear PID controller design for a 6-DOF UAV quadrotor system," *Engineering Science and Technology, an International Journal*, vol. 22, Mar. 1, 2019. DOI: 10.1016/j.jestch.2019.02.005.

[18] N. Haala *et al.*, "DENSE MULTIPLE STEREO MATCHING OF HIGHLY OVERLAPPING UAV IMAGERY," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XXXIX-B1, pp. 387–392, Jul. 24, 2012. DOI: 10.5194/isprsarchives-XXXIX-B1-387-2012.

[19] "Drones - Unmanned Aerial Vehicles (UAVs), Types, Components, Works," electricalfundablog.com. (), [Online]. Available: https://electricalfundablog.com/drones-unmanned-aerial-vehicles-uavs/ (visited on 04/21/2022).

[20] A. t. A. D. W. D. i. a. q. e. s. e. h. l. a. q. t. h. r. H. c. r. A. F. D. P. V. w. a. Z. h3-3d gimbal *et al.* "Quadcopter Parts: What are they and what do they do?" Quadcopter Academy. (), [Online]. Available: https://www.quadcopteracademy.com/quadcopter-parts-what-are-they-and-what-do-they-do/ (visited on 04/21/2022).

[21] "Step 1: Design your frame | Unmanned Aerial Vehicles Team." (), [Online]. Available: https://sites.bu.edu/uav/first-build/step1/ (visited on 04/21/2022).

[22] I. P. Tracy, "Propeller design and analysis for a small, autonomous UAV," Thesis, Massachusetts Institute of Technology, 2011. [Online]. Available: https://dspace.mit.edu/handle/1721.1/68926 (visited on 04/21/2022).

[23] B. Rutkay *et al.*, "Design and manufacture of propellers for small unmanned aerial vehicles," *Journal of Unmanned Vehicle Systems*, vol. 4, no. 4, pp. 228–245, Dec. 2016, ISSN: 2291-3467. DOI: 10.1139/juvs-2014-0019. [Online]. Available: https://cdnsciencepub.com/doi/full/10.1139/juvs-2014-0019 (visited on 04/21/2022).

[24] "Airfoils and Lift." (), [Online]. Available: http://www.aviation-history.com/theory/airfoil.htm (visited on 04/21/2022).

[25] "Aircraft propellers and how they work." (), [Online]. Available: http://www.pilotfriend.com/training/flight_training/fxd_wing/props.htm (visited on 04/21/2022).

[26] "Types of Drone Propellers: Everything You Need to Know," 3D Insider. (), [Online]. Available: https://3dinsider.com/types-of-drone-propellers/ (visited on 04/21/2022).

[27] "UAV & Drone Propellers | Drone Propeller Manufacturers & Suppliers," Unmanned Systems Technology. (), [Online]. Available: https://www.unmannedsystemstechnology.com/category/supplier-directory/propulsion-power/propellers/ (visited on 04/21/2022).

[28] Jared Owen, director, *How does an Electric Motor work? (DC Motor)*, Jun. 10, 2020. [Online]. Available: https://www.youtube.com/watch?v=CWulQ1ZSE3c (visited on 04/24/2022).

[29] "What are Brushless DC Motors | Renesas." (), [Online]. Available: https://www.renesas.com/us/en/support/engineer-school/brushless-dc-motor-01-overview (visited on 04/24/2022).

[30] P. J. Zhao *et al.*, "Brushless DC Motor Fundamentals Application Note," p. 19, 2014.

[31] "Simple DC motor." (), [Online]. Available: https://electronics.stackexchange.com/questions/452474/self-inductance-in-steady-state-equivalent-model-of-brushed-dc-machine (visited on 04/24/2022).

[32] "Model-of-a-brush-DC-motor.png (850×569)." (), [Online]. Available: https://www.researchgate.net/profile/Jacob-Pedersen-6/publication/288617020/figure/fig5/AS:669369784868869@1536601601338/Model-of-a-brush-DC-motor.png (visited on 04/23/2022).

[33] The Engineering Mindset, director, *How does an Electric Motor work? DC Motor explained*, Apr. 15, 2020. [Online]. Available: https://www.youtube.com/watch?v=GQatiB-JHdI (visited on 04/23/2022).

[34] The Engineering Mindset, director, *Brushless Motor - How they work BLDC ESC PWM*, Feb. 5, 2022. [Online]. Available: https://www.youtube.com/watch?v=yiD5nCfmbV0 (visited on 04/23/2022).

[35] "Brushless Vs Brushed DC Motors: When and Why to Choose One Over the Other | Article | MPS." (), [Online]. Available: https://www.monolithicpower.com/en/brushless-vs-brushed-dc-motors (visited on 04/24/2022).

[36] "Antigravity MN4006 KV380 - 2PCS/SET_Antigravity Type_Motors_Multirotor_T-MOTOR Store-Official Store for T-motor drone motor,ESC,Propeller." (), [Online]. Available: https://store.tmotor.com/goods.php?id=440 (visited on 04/06/2022).

[37] W. Brown, "Brushless DC Motor Control Made Easy," p. 48, 2002.

[38] Oscar. "How to Choose FPV Drone Motors," Oscar Liang. (), [Online]. Available: https://oscarliang.com/quadcopter-motor-propeller/ (visited on 04/23/2022).

[39] "Blog - Everything You Need to Know About Electronic Speed Controllers (ESC)," Hobbyking. (), [Online]. Available: https://hobbyking.com/en_us/blog/cat/planes/post/everything-need-know-electronic-speed-controllers-esc/?___store=en_us (visited on 05/09/2022).

[40] ——, "FPV Drone ESC Buyer's Guide," Oscar Liang. (), [Online]. Available: https://oscarliang.com/choose-esc-racing-drones/ (visited on 05/09/2022).

[41] "Blog - Understanding the Numbers and Letters on a LiPo Battery," Hobbyking. (), [Online]. Available: https://hobbyking.com/en_us/blog/reading-and-understanding-lipo-batteries/?___store=en_us (visited on 04/21/2022).

[42] ——, "Everything About LiPo Battery for Racing Drones," Oscar Liang. (), [Online]. Available: https://oscarliang.com/lipo-battery-guide/ (visited on 04/21/2022).

[43] "A Lipo Battery Guide to Understand Lipo Battery." (), [Online]. Available: https://www.genstattu.com/bw/ (visited on 04/22/2022).

[44] "Lithium ion Battery vs Lithium polymer Battery," WAAREE ESS. (), [Online]. Available: https://waareeess.com/lithium-ion-vs-lithium-polymer-battery (visited on 04/22/2022).

[45] "Blog - RC Batteries: LiPo Vs NiHM - Why LiPos are Better," Hobbyking. (), [Online]. Available: https://hobbyking.com/en_us/blog/rc-batteries-lipo-vs-nimh-differences (visited on 04/22/2022).

[46] "What is an Inertial Measurement Unit?" VectorNav. (), [Online]. Available: https://www.vectornav.com/resources/inertial-navigation-articles/what-is-an-inertial-measurement-unit-imu (visited on 04/30/2022).

[47] "What is IMU? Inertial Measurement Unit Working | Arrow.com," Arrow.com. (), [Online]. Available: https://www.arrow.com/en/research-and-events/articles/imu-principles-and-applications (visited on 05/01/2022).

[48] "The Case of the Misguided Gyro | Analog Devices." (), [Online]. Available: https://www.analog.com/en/analog-dialogue/raqs/raq-issue-139.html (visited on 05/01/2022).

[49] K. P. Valavanis *et al.*, Eds., *Handbook of Unmanned Aerial Vehicles.* Dordrecht: Springer Netherlands, 2015, ISBN: 978-90-481-9706-4 978-90-481-9707-1. DOI: 10.1007/978-90-481-9707-1. [Online]. Available: http://link.springer.com/10.1007/978-90-481-9707-1 (visited on 04/21/2022).

[50] "Flight Controllers explained for everyone," Fusion Engineering. (), [Online]. Available: https://fusion.engineering/flight-controllers-explained-for-everyone/ (visited on 04/30/2022).

[51] "Learn about MEMS accelerometers, gyroscopes, and magnetometers." (), [Online]. Available: `https://www.vectornav.com/resources/inertial-navigation-primer/theory-of-operation/theory-mems` (visited on 04/10/2022).

[52] "Accelerometer Types," Accelerometer Types. (), [Online]. Available: `https://www.te.com/usa-en/industries/sensor-solutions/insights/types-of-accelerometers.html` (visited on 04/12/2022).

[53] S. Hanly. "Accelerometers: Taking the Guesswork out of Accelerometer Selection." (), [Online]. Available: `https://blog.endaq.com/accelerometer-selection` (visited on 04/12/2022).

[54] ——, "Vibration Measurements: Accelerometer Basics." (), [Online]. Available: `https://blog.endaq.com/vibration-measurements-accelerometer-basics` (visited on 04/12/2022).

[55] Transfer Multisort Elektronik, director, *ACCELEROMETER - What it is and how it works*, Jul. 14, 2021. [Online]. Available: `https://www.youtube.com/watch?v=To7JagpPDwY` (visited on 04/15/2022).

[56] "Introduction To Capacitive MEMS Accelerometers and A Case Study On An Elevator," Maker Portal. (), [Online]. Available: `https://makersportal.com/blog/2017/9/25/accelerometer-on-an-elevator` (visited on 04/11/2022).

[57] "Introduction to Capacitive Accelerometers: Measuring Acceleration with Capacitive Sensing - Technical Articles." (), [Online]. Available: `https://www.allaboutcircuits.com/technical-articles/introduction-to-capacitive-accelerometer-measure-acceleration-capacitive-sensing/` (visited on 04/15/2022).

[58] "Capacitive Accelerometer - an overview | ScienceDirect Topics." (), [Online]. Available: `https://www.sciencedirect.com/topics/computer-science/capacitive-accelerometer` (visited on 04/11/2022).

[59] E. Sobhani-Tehrani *et al.*, *Fault Diagnosis of Nonlinear Systems Using a Hybrid Approach*, ser. Lecture Notes in Control and Information Sciences. Boston, MA: Springer US, 2009, vol. 383, ISBN: 978-0-387-92906-4 978-0-387-92907-1. DOI: `10.1007/978-0-387-92907-1`. [Online]. Available: `http://link.springer.com/10.1007/978-0-387-92907-1` (visited on 04/26/2022).

[60] M. J. de la Fuente, "Fault Detection and Isolation: An overview," p. 68,

[61] A. Mouzakitis, "Classification of Fault Diagnosis Methods for Control Systems," *Measurement and Control*, vol. 46, no. 10, pp. 303–308, Dec. 1, 2013, ISSN: 0020-2940. DOI: `10.1177/0020294013510471`. [Online]. Available: `https://doi.org/10.1177/0020294013510471` (visited on 08/24/2021).

[62] S. A. Aleem *et al.*, "Methodologies in power systems fault detection and diagnosis," *Energy Systems*, vol. 6, no. 1, pp. 85–108, Mar. 1, 2015, ISSN: 1868-3975. DOI: `10.1007/s12667-014-0129-1`. [Online]. Available: `https://doi.org/10.1007/s12667-014-0129-1` (visited on 08/24/2021).

[63] S. Katipamula *et al.*, "Methods for Fault Detection, Diagnostics and Prognostics for Building Systems - A Review Part I," *HVAC and R Research*, vol. 11, Apr. 1, 2005. DOI: `10.1080/10789669.2005.10391133`.

[64] Y. Jiang *et al.*, *Fault Detection and Identification for Quadrotor Based on Airframe Vibration Signals: A Data-Driven Method*. Jul. 1, 2015, p. 6361, 6356 pp. DOI: `10.1109/ChiCC.2015.7260639`.

[65] X. Zhang *et al.*, "Fault Detection and Identification Method for Quadcopter Based on Airframe Vibration Signals," *Sensors*, vol. 21, p. 581, Jan. 15, 2021. DOI: `10.3390/s21020581`.

[66] A. Bondyra *et al.*, *Fault Diagnosis and Condition Monitoring of UAV Rotor Using Signal Processing*. Sep. 1, 2017, p. 238, 233 pp. DOI: `10.23919/SPA.2017.8166870`.

[67] G. Iannace *et al.*, "Fault Diagnosis for UAV Blades Using Artificial Neural Network," *Robotics*, vol. 8, p. 59, Jul. 20, 2019. DOI: `10.3390/robotics8030059`.

[68] K. H. Park *et al.*, "Unsupervised Fault Detection on Unmanned Aerial Vehicles: Encoding and Thresholding Approach," *Sensors*, vol. 21, no. 6, p. 2208, 6 Jan. 2021. DOI: `10.3390/s21062208`. [Online]. Available: `https://www.mdpi.com/1424-8220/21/6/2208` (visited on 08/24/2021).

[69] H. Loewen. "Isolating Components from UAV Vibration." (), [Online]. Available: `https://www.micropilot.com/pdf/isolating-components-uav-vibration.pdf` (visited on 04/27/2022).

[70] F. Riccardi, *DESIGN AND ANALYSIS OF ELASTOMERIC VIBRATIONS DAMPERS FOR IMU ON-BOARD RUAV*. Jul. 22, 2015. DOI: `10.13140/RG.2.1.4395.7603`.

[71] "Vibration Damping — Copter documentation." (), [Online]. Available: `https://ardupilot.org/copter/docs/common-vibration-damping.html` (visited on 04/27/2022).

[72] "Blip," Crowd Supply. (), [Online]. Available: `https://www.crowdsupply.com/electronut-labs/blip` (visited on 04/28/2022).

[73] "LIS2DH12 - MEMS digital output motion sensor: Ultra-low-power high-performance 3-axis "femto" accelerometer - STMicroelectronics." (), [Online]. Available: `https://www.st.com/en/mems-and-sensors/lis2dh12.html` (visited on 04/28/2022).

[74] "HobbyKing 20A BlueSeries Brushless Speed Controller," Hobbyking. (), [Online]. Available: `https://hobbyking.com/en_us/hobbyking-20a-blueseries-brushless-speed-controller.html?___store=en_us` (visited on 04/28/2022).

[75] "X500 Kit." (), [Online]. Available: `https://shop.holybro.com/x500-kit_p1180.html` (visited on 04/23/2022).

[76] Nordic Semiconductor, director, *Introduction to the Zephyr RTOS*, Oct. 6, 2020. [Online]. Available: `https://www.youtube.com/watch?v=jR5E5Kz9A-k` (visited on 04/29/2022).

[77] "Zephyr Security Overview — Zephyr Project Documentation." (), [Online]. Available: `https://docs.zephyrproject.org/latest/security/security-overview.html` (visited on 04/29/2022).

[78] A. V. Oppenheim *et al.*, *Discrete-Time Signal Processing*, 3rd ed. Upper Saddle River: Pearson, 2010, 1108 pp., ISBN: 978-0-13-198842-2.

[79] S. W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego, Calif.: California Technical Pub., 1999, ISBN: 978-0-9660176-7-0 978-0-9660176-4-9 978-0-9660176-6-3.

[80] "Sampling rate and aliasing effect." (), [Online]. Available: `https://www.kistler.com/es/glosario/termino/sampling-rate-and-aliasing-effect/` (visited on 05/08/2022).

[81] T. M. Mitchell, *Machine Learning*, ser. McGraw-Hill Series in Computer Science. New York: McGraw-Hill, 1997, 414 pp., ISBN: 978-0-07-042807-2.

[82] C. M. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. New York: Springer, 2006, 738 pp., ISBN: 978-0-387-31073-2.

[83] I. Goodfellow *et al.*, *Deep Learning*, ser. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: The MIT Press, 2016, 775 pp., ISBN: 978-0-262-03561-3.

Acronyms

ADC analog-to-digital conversion. 36

DFT discrete Fourier transform. 39

DTW dynamic time warping. 43

ESC electronic speed controller. 6, 9, 10, 11, 13, 24, 30, 52

FC flight controller. 6, 11, 13, 23

FD fault detection. 19, 22

FDD fault detection and diagnostics. 19

FDI fault detection and isolation. 19, 21

FFT fast Fourier transform. 22, 39, 57, 58, 59

FNN feedforward neural network. vii, 45, 47, 56, 57, 58, 59

FPGA field programmable gate array. 26

GCS ground control station. 3

GPS global positioning system. 3, 13

I2C inter-integrated circuit. 24

IMU inertial measurement unit. 13, 22, 23

kNN k-nearest neighbors. vii, 43, 56, 57, 59

Li-ion lithium-ion. 11, 12

LiPo lithium-ion polymer. vi, 11, 12, 13, 54

LSTM Long and Short-Term Memory. 21, 22

MCU microcontroller. 26

MEMS micro-electro-mechanical systems. 18, 24

ML machine learning. 40, 41, 42

MLP multilayer perceptron. 45

MPU microprocessor. 26

NiMH nickel metal hydride. 11, 12

OS operating system. 26

PWM pulse-width modulation. 11, 30, 52

RMSE root-mean-square energy. 57, 58, 59

RPM revolutions per minute. 7, 10, 56

RTOS real time operating system. 26

SoC system on a chip. 24, 26

SPI serial peripheral interface. 24

SQNR signal to quantization ratio. 36

SVM support vector machine. 22

UART universal asynchronous receiver/transmitter. 50

UAS unmanned aircraft system. 3

UAV unmanned aerial vehicle. vi, 1, 2, 3, 4, 5, 7, 9, 11, 19, 21, 22, 23, 25, 29, 61

WPD wavelet packet decomposition. 21, 22

Contents of attached SD card

`REAME.md`	README
`thesis.pdf`	the thesis text in PDF format
`zephyr_app`	the zephyr daat acquisition application
`algorithms`	notebooks and models