Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Measurement

Edge machine learning-based industrial fault detection

Erik Pásztor

Supervisor: prof. Ing. Radislav Šmíd, Ph.D.
Field of study: Cybernetics and Robotics
May 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Pásztor Erik**	Personal ID number:	**483595**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Measurement**		
Study program:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Edge machine learning-based industrial fault detection

Master’s thesis title in Czech:

Detekce poruch v průmyslu s použitím strojového učení v koncovém zařízení

Guidelines:

Design and build a demonstrator for fault detection using local machine learning. Consider faults specific to industrial rotating machinery, like a damaged or missing tooth in spur gears. Use the incremental encoder (IRC) as the only signal source. In the edge device based on an STM32F413 microcontroller, implement signal preprocessing, feature extraction, fault detection, and network communication via Ethernet. Evaluate the usability and performance of NanoEdge AI Studio, an Automated Machine Learning (ML) tool for STM32, for both supervised and unsupervised learning.

Bibliography / sources:

Duda R.O., Hart, P.E.,Stork, D.G.: Pattern Classification, John Willey and Sons, 2nd ed, New York
NanoEdgeAIStudio - Automated Machine Learning (ML) tool, STmicroelectronics 2022

Name and workplace of master’s thesis supervisor:

prof. Ing. Radislav Šmíd, Ph.D.   Department of Measurement FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **07.02.2023**   Deadline for master's thesis submission: **26.05.2023**

Assignment valid until:
by the end of summer semester 2023/2024

prof. Ing. Radislav Šmíd, Ph.D.	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature		Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt   Student’s signature

Acknowledgements

Foremost, I would like to thank my supervisor, prof. Radislav Šmíd for guidance and valuable insights during the writing of this thesis.

I would also like to thank the company STMicroelectronics, who supplied the necessary hardware and whose employees often provided advice on many parts of the thesis.

Special thanks goes to my family, whose help, encouragement, and motivation made this work a pleasant endeavour.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 20. May 2023

Signature:

Abstract

This thesis describes the implementation of machine learning-based fault detection on an edge device. The main part of the system is built on an STM32F413ZH microcontroller that performs data acquisition and processing and inference by a pre-trained machine-learning model.

A gearbox was 3D printed for demonstration of this device with interchangeable undamaged and damaged wheels. Apart from this, the designed demonstration unit is composed of an electric motor, a microcontroller to control it, a separate microcontroller to enable Ethernet communication and a display.

The proposed system collects data from an incremental rotary encoder, preprocesses the signal, and extracts features based on both its frequency and time domain characteristics. Various models trained in NanoEdge AI Studio were tested and compared, and the limit where the system can still reliably detect faults was determined. With anomaly detection, a true positive rate of 1 and a true negative rate of 0.74 were achieved. With multiclass classification, a perfect score was obtained when considering only the healthy state and two large faults.

Keywords: edge device, machine learning, fault detection, anomaly detection, incremental rotary encoder, gear damage, Industry 4.0, NanoEdge AI Studio

Supervisor: prof. Ing. Radislav Šmíd, Ph.D.

Abstrakt

Táto práca sa zaoberá realizáciou koncového zariadenia s detekciou porúch založenou na strojovom učení. Hlavnou súčasťou systému je mikrokontrolér STM32F413ZH, ktorý vykonáva zber a spracovanie dát a vyhodnotenie pomocou dopredu naučeného modelu.

Pre demonštráciu tohto zariadenia bola 3D tlačou vyrobená prevodovka s vymeniteľnými nepoškodenými a poškodenými ozubenými kolesami. Okrem toho demonštračná jednotka obsahuje elektromotor, mikrokontrolér, ktorý ho ovláda, osobitný mikrokontrolér zabezpečujúci ethernetovú komunikáciu a displej.

Navrhnutý systém zbiera dáta z inkrementálneho rotačného enkodéru, predspracováva signál a extrahuje príznaky založené na jeho frekvenčných aj časových charakteristikách. Boli otestované a porovnané rôzne modely natrénované v NanoEdge AI Studio a určený limit, pri ktorom systém ešte dokáže spoľahlivo detekovať chyby. S projektom založenom na detekcií anomálií boli dosiahnuté hodnoty 1 pre mieru skutočne pozitívnych detekcií a 0.74 pre mieru skutočne negatívnych detekcií. Model klasifikujúci viacero tried mal bezchybné výsledky, keď bol vstup obmedzený iba na normálny stav a dve veľké poškodenia.

Kľúčové slová: koncové zariadenie, strojové učenie, detekcia chýb, detekcia anomálií, inkrementálny rotačný enkodér, poškodenie prevodov, priemysel 4.0, NanoEdge AI Studio

Preklad názvu: Detekcia porúch v priemysle s použitím strojového učenia v koncovom zariadení

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation and Task Description

The motivation behind this work is the growing popularity of machine learning (ML) and artificial intelligence (AI) algorithms in general and the growing need for precise identification of faults in industrial machinery. As shown in Figure 1.1, it is estimated that one-quarter of all manufacturing costs in the United States. can be traced to the time when production is stopped because of machine downtime. Accurate and early detection that a machine

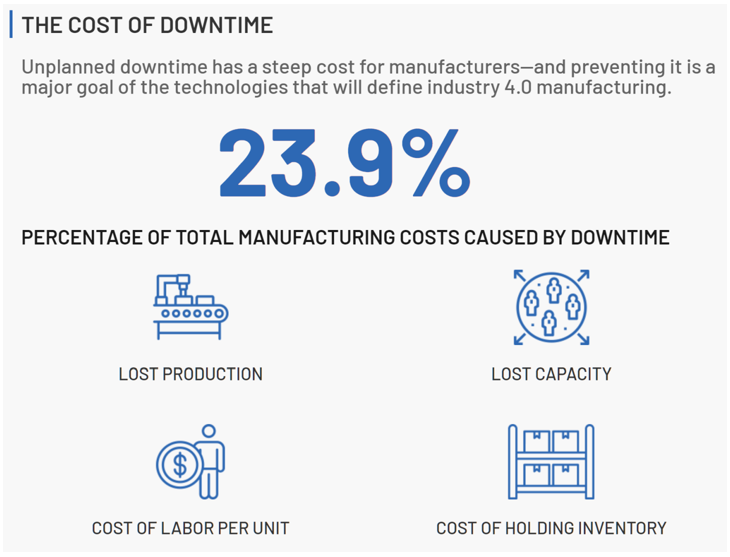

THE COST OF DOWNTIME
Unplanned downtime has a steep cost for manufacturers—and preventing it is a major goal of the technologies that will define industry 4.0 manufacturing.
23.9%
PERCENTAGE OF TOTAL MANUFACTURING COSTS CAUSED BY DOWNTIME
LOST PRODUCTION
LOST CAPACITY
COST OF LABOR PER UNIT
COST OF HOLDING INVENTORY

Figure 1.1: Estimation of downtime costs in the United States. Image adapted from Analog Devices, source: The Costs and Benefits of Advanced Maintenance in Manufacturing, U.S. Department of Commerce, April 2018.

is going to require maintenance or repair allows for careful scheduling and management and thus reducing the final cost. With many Internet of Things (IoT) devices already in use, a large amount of data needs to be transferred to the cloud for processing because a large number of these devices only serve to collect data. It is apparent that the volume of data and, thus, load on the internet infrastructure will increase in the coming years with more devices being deployed. Figure 1.2 shows that in a few years, almost one-third of all Internet traffic will be made up of IoT communication. To address this issue,

In 2030 it is estimated that the billions of sensors on IoT devices could be responsible for 30% of Internet data traffic*. Edge computing provides a solution.
*Andrae (2020) 'New perspectives on internet electricity use in 2030'

Figure 1.2: Estimation of IoT internet traffic. Image adapted from STMicroelectronics, source: [1].

devices that need to transfer as little data as possible should be designed. Industry 4.0 aims to incorporate smart automation and faster and more accurate devices into manufacturing infrastructures, and this work should contribute to that goal.

Paired with advances in the computing power of microcontroller units (MCUs), the consequence is that more intelligent and capable edge devices are being created. Edge devices are units that can collect data or act on their surroundings (or both) and are connected to a central unit that unifies them and can make decisions based on data. This architecture is covered by the Industry 4.0 concept.

Apart from collecting data, many devices also evaluate what has been collected, interpret the results of computations, and communicate with their central device only to indicate a state or report on an action, etc. One of the advantages of this approach is that the communication bandwidth is considerably reduced as only a small number of results need to be transferred instead of all the collected data. This is strongly related to the saving of energy consumption since if the device uses wireless communication, the energy expended for transmission can be an important part of its overall consumption. Another benefit is privacy; if the collected data are not transmitted between devices and stored in a database, there is a lower risk of an attacker gaining sensitive information. The time delay between a change in conditions and a critical action taken by the system can also be shortened because everything runs locally. Combining these factors, intelligent edge devices can operate fully independently at all times, without increasing risks or operating costs. In these devices, ML is used to interpret data and make decisions about the state of the observed system.

An important subcategory is made up of devices that implement predictive maintenance of industrial or other types of machinery. Detecting faults in this context is important, for example, to avoid unscheduled downtime in manufacturing. These devices can often perform some kind of real-time monitoring while sending all the data to a database, where they are processed further by cloud computing techniques. A detailed description of parts of this architecture is provided in article [2].

Using signal processing combined with AI directly on the edge device, fault detection systems can avoid the need for a central server and benefit from the advantages listed above. This is, in part, the aim of this work, to show how well the entire system can function without any input from other devices. The focus will be on detecting faults in rotating machinery that incorporates gears. These faults can include more or less pronounced damage to gear teeth, issues with bearings, or motor defects. The state of such systems can be monitored using various sensors, including accelerometers or microphones to observe vibrations or sounds originating in the gearbox or the motor itself. In this work, however, an incremental rotary encoder (IRC) will be used, which is already incorporated in a large portion of more complex motor control applications for the purposes of speed and position estimation. Thus this approach eliminates the need for additional sensors, which can save costs and complexity of the final design. Apart from this, IRC is much less susceptible to noise, which can affect other types of sensors, vibrations, or sound from external sources. And its direct connection to the drive shaft or even inclusion in the motor itself means that the signal travels only a short path from the source (e.g., the degraded tooth), as opposed to accelerometers which sense overall vibrations that can be influenced by the surrounding non-relevant sources.

1.2 Related Work

There has been a number of works that dealt with similar topics and resulted in prototypes or evaluation software that were more or less functional. Many of these works either utilised a computer for signal processing and AI algorithms implementation (and only used an MCU to collect data from a physical set-up) or their system worked completely on an MCU but the proposed methods were too simple, and they could not achieve substantial accuracy of detection.

In [3], the authors used a complex ML approach based on convolutional neural networks (CNN). They collected data from two IRCs (to get a measurement called transmission error) and from accelerometers. They used various methods for signal processing (denoising, enhancement, and segmentation) before using it as an input for their CNN. Several CNNs of different types were trained to classify three states of a gear wheel. The authors achieved near-perfect results when testing with the transmission error signal, but the trained networks had a considerable number of layers, and the software was designed to work on a computer with more resources than an MCU.

A team of authors in [4] analysed vibration signals and computed a representation called a Poincaré plot. They extracted several features from this, which were then used for classification by Support vector machines. They obtained accuracy in the range of 90% to 100% when classifying gear faults and roller bearing issues with various methods and settings. Although the algorithms are simple enough to implement on an MCU, the limitation of this work is that an external sensor was used.

A much more simplistic approach was taken in [5], where a team from France attempted to classify gear and bearing faults in a gear reducer system based purely on the three-phase current signals which are, in general, measured in all alternating current motors. This mitigates the need for any sensor, even an IRC, which does not need to be used in every application. After signal processing, they employed Fast Fourier transform to obtain a frequency spectrum from which they exported various features. Then, these were used for classification with an Adaptive Neuro-Fuzzy inference system with almost 100% accuracy. Even though a computer was used for computations in this work, a similar approach should be possible to employ on an MCU.

One of the many state-of-the-art solutions which have recently been introduced for practical use in the industry is the ADI OtoSense Smart Motor Sensor (described in [6]), created and sold by the company Analog Devices. This product won the Digitalization Automation Award in 2021. It uses signal analysis with AI algorithms to detect and classify nine different mechanical and electrical faults (including misalignment, bearing problems, etc.). Similarly to the system proposed in this work, it is a single-sensor solution but uses accelerometers instead of an IRC. This makes it potentially vulnerable to external noise, which the system needs to deal with. A system using an IRC should be more independent of the mechanical qualities of the motor's construction. An IRC can also be placed on any rotating shaft in a gearbox without losing any advantages or needing to reconfigure the system. In contrast, with an accelerometer, the placement affects signal properties due to different signal paths.

1.3 Aims

This work aims to develop a unit that would demonstrate the usage of ML algorithms in an edge device, particularly for detecting faults in a gearing system. Spur gears are used as the simplest type of gear. Other kinds (for example, herringbone gears) offer significant advantages, including less emitted noise or more efficient transfer of torque, but they are harder to assemble and would not be suitable for a demonstration where gear wheels need to be changed often.

From the wide range of faults that can occur in a gearbox, missing teeth (due to overloading or crack), an extreme case of mechanical damage, will be targeted for detection. Tests will also be performed to try to detect more minor faults represented by pitted teeth. An IRC sensor will be used as the only information source about the gearbox's state. This sensor is built-in to the brushless direct current motor, which will be used.

The whole detection pipeline will be implemented exclusively on an STM32F413ZH MCU from the acquisition of data from an IRC, signal processing, and feature extraction to inference using a pre-trained AI model. Training will be done in NanoEdge AI Studio (a product of STMicroelectronics) using real data collected by the MCU. The particular MCU was chosen to verify and show that such a system can work even on an older, less powerful device (compared to other available MCU series).

Various types of ML models will be trained and compared to each other - models that classify multiple states into corresponding classes and models whose output is only an indication of whether the system is in a healthy or damaged state. For models with multiple classes, a comparison with results achieved on a computer (with similar ML algorithms) will also be provided.

The main output of this work should be a demonstrator with added functions such as a graphical user interface and Ethernet communication to better illustrate how such a device could be implemented in a natural industrial setting. A secondary contribution will be an evaluation of the NanoEdge AI Studio tool, showing if the created models can produce satisfactory results in a complex scenario.

Chapter 2

Methods for Fault Detection Using Edge Computing

In this chapter, a general description will be provided of the systems and methods that were used to create an edge device capable of detecting gearbox faults. The theory of the target sensor and how its signal is obtained and processed will be discussed. Then several ML algorithms that can be used to evaluate the state of the gears will also be outlined.

Figure 2.1 shows the overall flowchart of the system. The gear state reflects the gearbox's physical condition. This is transferred through the gears onto the motor shaft, whose rotation can be monitored using an IRC sensor. Fluctuations in the IRC signal are caused by irregular motor movement, which can be traced back to an abnormal state of one of the gear wheels.

A hardware timer (an MCU peripheral) is used to collect data from the sensor. After gathering enough data points, signal processing is done, and features are computed. These features enable us to distinguish between various gear states using inference by an ML model. The result is then communicated to the system's operator (using an LCD) and to a central device via Ethernet.

More details on individual blocks will be given in the following sections, and details of the practical implementation will be provided in Chapter 3.

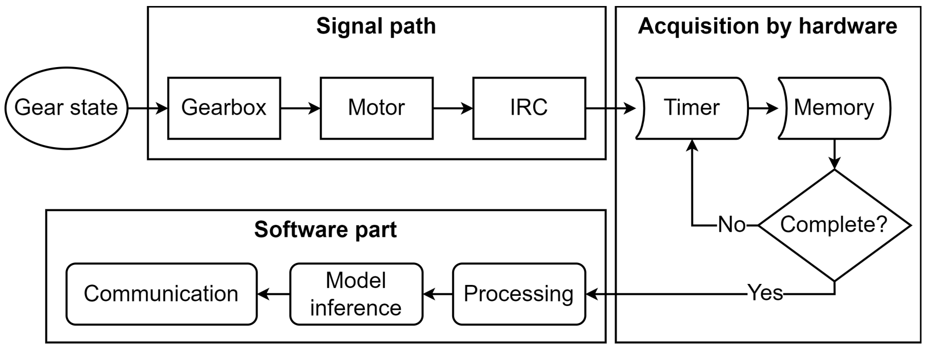


Figure 2.1: Flowchart of a gear state's propagation through the system

2.1 Signal Acquisition

This section describes the IRC sensor and the method employed to acquire its signal.

IRC Sensor

The principle of IRC sensors is quite simple compared to other popular sensors, such as accelerometers. At the same time, they can be manufactured on a larger scale, which makes them typically cheap, reliable, and widely used. Figure 2.2 shows the typically produced signals. The type of output depends on the sensor. The first type offers better resolution after calculating the motor position using trigonometry but requires fast and precise sampling and additional computation time. However, a pulse signal can be fed into a timer peripheral of an MCU. This provides a more straightforward but slightly less accurate measurement. Usually, there are two output signals (channels) physically shifted by 45 degrees. By measuring which channel rises first, the control system can determine the direction of rotation.

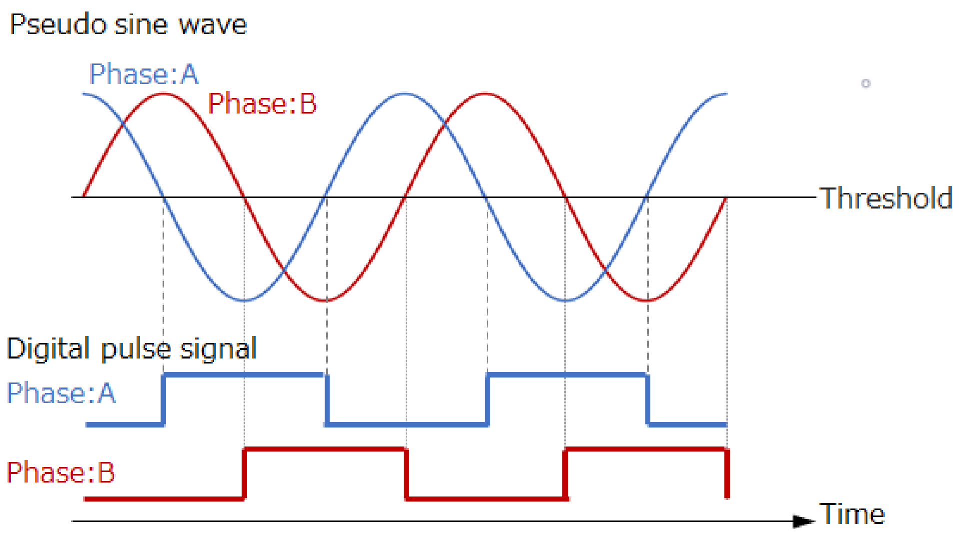


Figure 2.2: Output signals of a rotary encoder, courtesy of [7]

There are other advantages to these sensors besides their price and robustness. They are often already incorporated into the motor control applications, so there is no need to use additional external sensors. An IRC is connected directly to the motor's drive shaft. In contrast, accelerometers and similar sensors are mounted to an enclosure or a part of a support structure close to the gears. Because of this, an IRC is positioned much closer to the motor in the drivetrain and, therefore, should have lower susceptibility to external vibrations and noises. Because of this, the argument can also be made that an IRC does not provide as much information as an accelerometer positioned close to a gear far from the motor (with a high gear ratio). In this work, the results of fault detection will be compared with faults located both close and farther from the motor using an IRC.

Of course, there are disadvantages to this kind of sensor. They are not as informative about vibrations and movement in the gears because they only register a change in one dimension. On the contrary, accelerometers usually measure movement in all three axes. Also, to successfully record a fault with an IRC, the change in speed needs to last a sufficiently long time to be above the sensor's resolution. An IRC has a fixed resolution, in this case, given by the number of pulses it produces per revolution. When using other types of sensors, the time resolution can generally be increased by increasing the sampling rate.

Timer Input Capture

To track the sensor signal with the MCU, its timer is used in input capture mode. The timer keeps increasing its internal counter, and at every edge of the external signal, the current value is saved to a register, and an interrupt is generated. Only rising edges are used, as explained later in Section 3.4. This is illustrated in Figure 2.3. The timer values can be transferred directly to a defined memory location using direct memory access (DMA) by the peripheral's hardware without any input (and thus delay) from the MCU core.

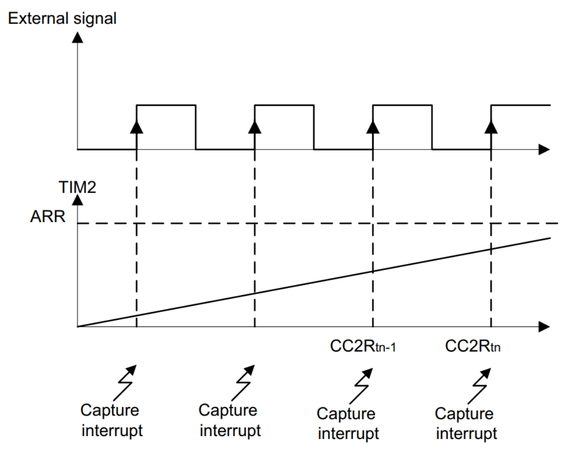


Figure 2.3: Input capture mode of a timer, taken from [8]

2.2 Detection of Fluctuations in IRC Signal

In this section, the output signal of an IRC sensor is described in relation to the given application. It also outlines how capturing the signal with a timer compares to doing acquisition with an ADC. The principle will be explained on an ideal signal.

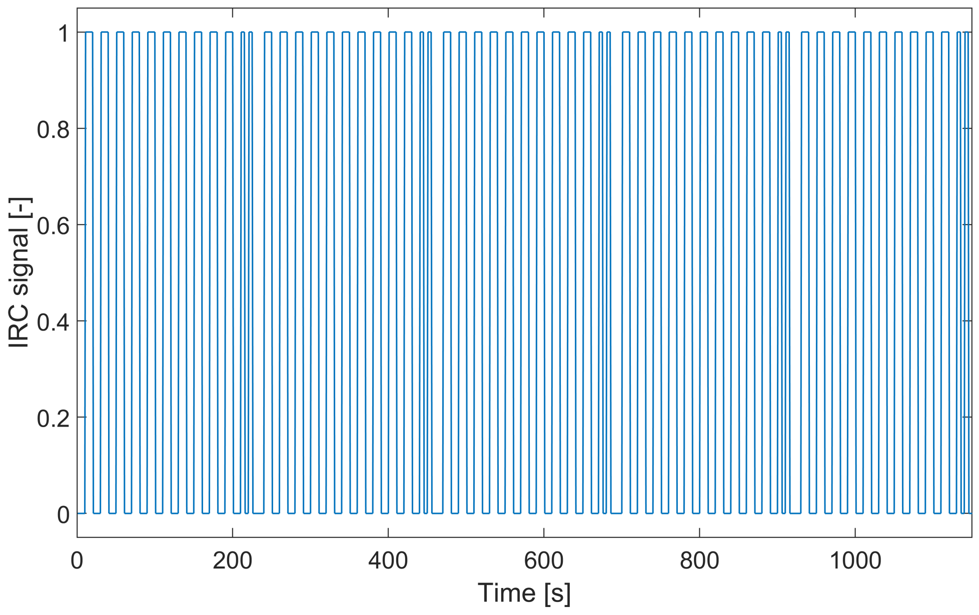
For the following analysis, a binary signal was created in Matlab with a sample rate of 1 Hz. It is made up of ten long pulses (each with a period of 20 seconds) and two slow ones (with a ten-second period); this pattern is repeated 100 times. The length of the signal is 2300 s, and it is shown in Figure 2.4. It can represent a case where the damage to a gear covers one-sixth of its circumference. In a scenario without mechanical faults, the gears would mesh perfectly, and the motor would rotate at a constant speed, thus resulting in a fixed length of pulses. Damage to a gear tooth creates a free space where the motor can move faster. This, of course, depends on the torque the controller produces, which increases with the applied load. As a result, there is a periodically repeating part of the signal where the pulses are shorter. This signal section in the time domain, combined with a known IRC resolution, corresponds to the fault location in the space domain. Commonly used ways to measure instantaneous angular speed (IAS) from a given IRC signal are described in article [9]. Here, the simplest timer-based method will be employed by measuring the time between successive pulses because the MCU timer can run with a clock frequency that is high enough not to create a considerable error.

When creating a system that is supposed to run on an MCU, an essential factor to consider is limited memory space. For signal analysis, multiple occurrences of an event must be recorded to properly determine its existence. This can lead to an excessively long signal when a high resolution is used together with the need to register rare faults - a damaged cog located directly on the shaft will produce more frequent signal changes than one farther away. This leads to a necessary trade-off between high resolution (which provides better accuracy in model creation) and adhering to memory constraints and lowering the computation time. As the sensor has a fixed resolution, only memory and time requirements must be considered.

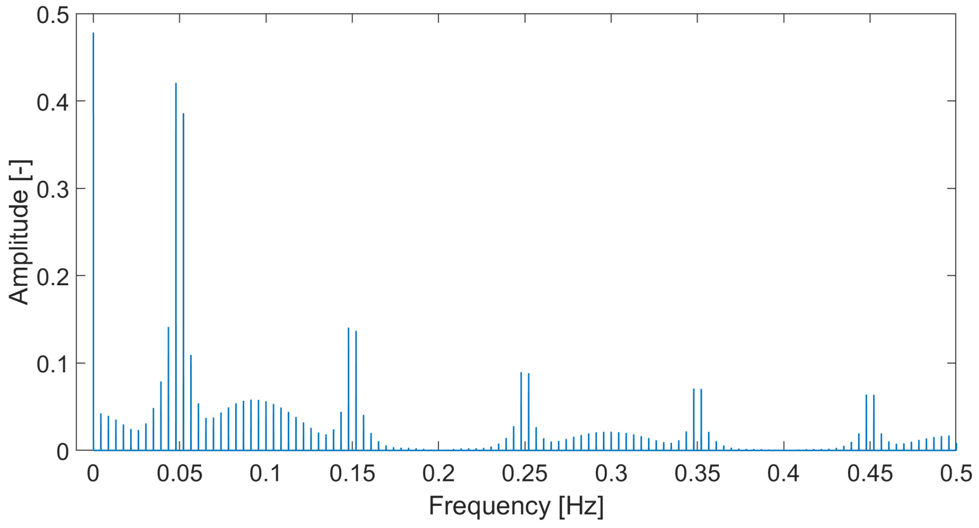
Considering the memory and computation time constraints, only one output channel of the IRC will be used throughout this work. The second channel would increase the spatial resolution twice but also multiply the required memory by the same factor. Generally, the main advantage of using both channels is that the direction of rotation can be easily found - which do not need to be considered for this application.

For the same reason, it is enough to capture only rising edges with the timer. Based on the assumption that the IRC resolution is high enough, the output signals should always have a duty cycle of 50 %. This means that complete information about the signal is obtained by measuring the signal time period between two rising edges. The theoretical exception to this would be a condition where the damage is minimal and covers only one pulse of the IRC. In such a case, capturing both rising and falling edges or using an IRC with a higher resolution would be necessary. However, it can be argued that such a slight change in signal (only one short pulse) would be below the level of measurement noise. This gives rise to possible modifications in a final application based on the necessary precision and the smallest detectable faults.

As mentioned in Section 2.1, one possible way to record such a signal would be with an ADC. Using the defined sample rate, it would directly acquire the signal as it is shown (at discrete points). Figure 2.4b shows the frequency spectrum of this signal. The signal will not be perfectly square in a real scenario, but the examined faults do not influence this type of fluctuation. So this method would not provide additional information while requiring more memory space.



(a) : Signal



(b) : Frequency spectrum

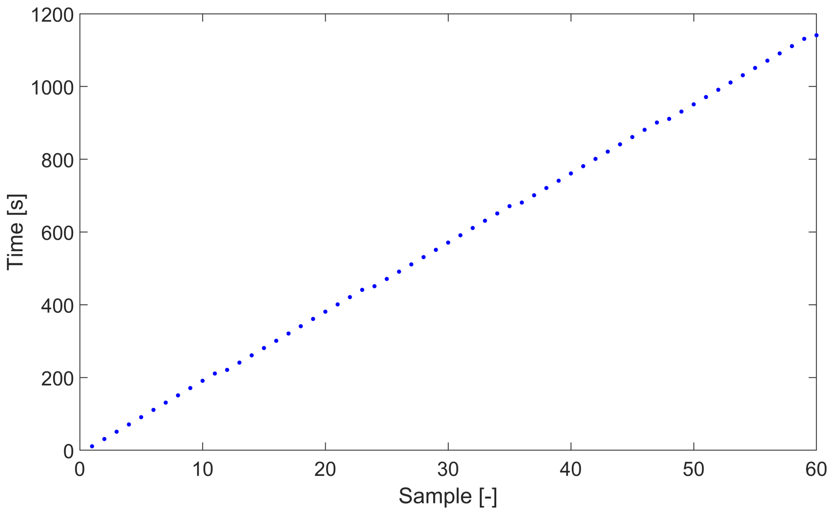
Figure 2.4: Idealised output of an IRC (one channel) and its frequency spectrum

On the other hand, as has been explained, the IAS signal computed from timer data should sufficiently reflect the desired phenomenon. This was confirmed, for example, by the authors of article [10], who found that changes in frequency components of the IAS signal correlate to variations in the dynamics of the rotor blades caused by damage, looseness, or misalignment.

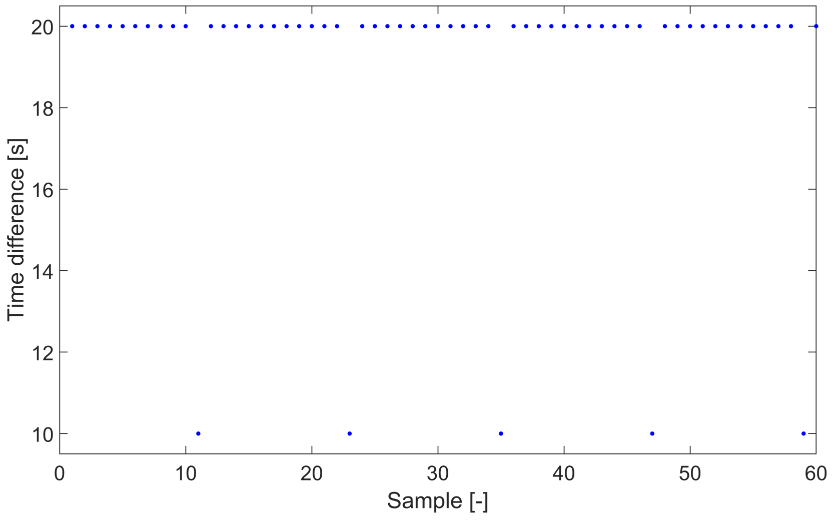
Figure 2.5a illustrates the times a timer would detect rising edges. Deviations of the signal slope can be seen at samples corresponding to intervals with higher speeds in the original signal. By calculating the differences between the timestamps, an IAS representation is obtained, as shown in Figure 2.5b. Several peaks can be clearly seen in its frequency spectrum in Figure 2.5c. In this idealised scenario, a gearbox without faults would produce a perfectly constant IAS, and so the frequency spectrum would only contain one non-zero component at zero frequency.

When comparing Figures 2.4b and 2.5c, it can be seen that, in the case of the timer acquisition method, the resulting spectrum contains considerably fewer peaks. They correspond to the base frequency of changes (errors) in the signal at $\frac{1}{6}$ of the sampling frequency and its harmonics. This can be attributed to the fact that the values are sampled at a constant angle frequency given by the IRC resolution. Therefore, it is not influenced by the motor's rotation speed, because the signal is always sampled at the same fixed points. It provides a kind of modulation to the resulting signal, which can be seen in the spectrum. This kind of sampling is beneficial because the sampling points are tied to the signal properties. In contrast, sampling points in the time domain are not related (in this case) to the measured signal, and so can produce additional noise and redundant information.

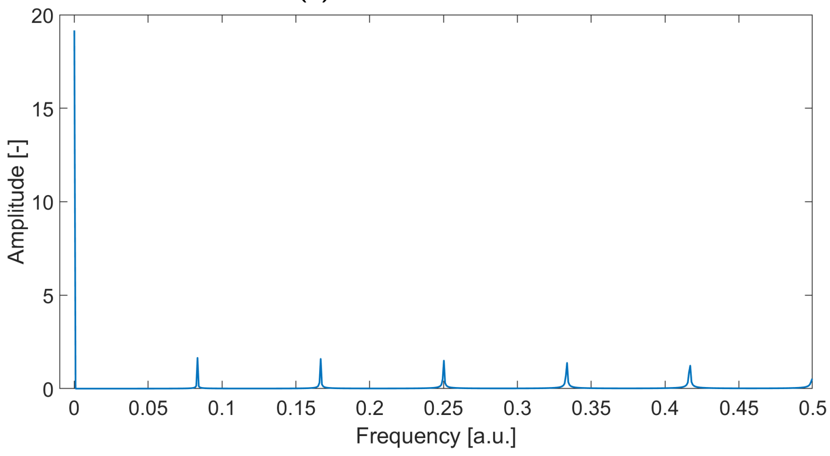
A combination of these approaches is termed synchronous sampling, where the sampling of one signal is triggered by another signal with additional information compared to sampling based on time. In the web article [11], the authors describe the differences between synchronous and fixed-time sampling. They explain that the frequencies, which are multiples of the base sampling frequency, are more well-defined when using synchronous sampling; this same conclusion can be seen from the example. It has been exploited in several works, namely [12] and [13] which both used software resampling of an acquired vibration signal from accelerometers. Article [12] also used other gearbox measurements, such as transmission error, to create features with which they later trained a multilayer perceptron algorithm. They also used true angular sampling for some features. With additional pre-processing, the success rate of their trained network was near 100 % in detecting several gearbox problems, such as gear failures (worn/chipped/missing tooth, crack), mechanical imbalances, misalignments, or bearing faults. They highlight the increased precision when using angular sampling. The authors of [13] worked similarly with frequency analysis and succeeded in showing how the spectrum changes dramatically when the correct type of sampling is used.



(a) : Timer values



(b) : Time differences



(c) : Frequency spectrum of differences. Auxiliary unit is used on the x-axis instead of Hz because the input signal of the frequency analysis is not time but time difference.

Figure 2.5: Ideal signal as captured by a timer

2.3 Signal Processing and Analysis

This section will explain the theoretical steps of the signal processing method. Figure 2.6 shows the individual stages together with illustrations of their outputs. It also contains the acquisition of the IRC signal, as explained in Section 2.2. After creating the feature vector, it is used to train an ML model or to infer a result with an already trained one.

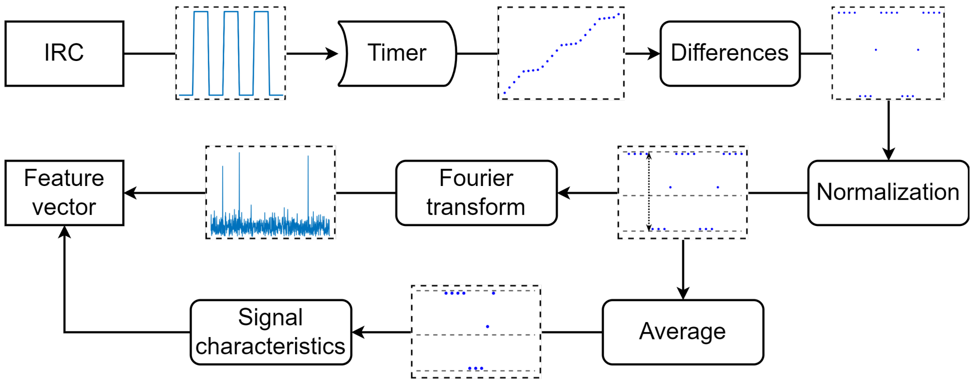


Figure 2.6: Flowchart of the proposed signal processing method

When trying to determine an appropriate approach, the Matlab software was employed to examine signals collected from the IRC and apply common processing techniques - practical results of this part and implementation details will be shown in chapter 4. Building on the memory savings, which were explained previously in Section 2.2, the computation time of the designed signal processing scheme also needs to be kept in mind. In practice, this means that complex analysis cannot be used as compared to other works that use computers for processing. Also, for this reason, while frequency features are included in the pipeline, only several predetermined frequency coefficients will be computed instead of the whole spectrum. Works showing that the use of computers for advanced processing can lead to excellent results include the article [12], in which the authors calculated the entire frequency spectrum and extracted features extracted from the spectrum and power spectrum density. They used these features in a multilayer perceptron model and achieved almost 100% successful classifications of several gearbox problems.

2.3.1 Time Differences and Normalisation

After acquiring the timestamps from the IRC signal, the differences between successive points are computed. By doing this, an approximation of the IAS signal is obtained that expresses the speed at each time. To get a proper representation of speed, the differences would need to be converted from timer ticks to real time (through the timer's sampling frequency) and divided by the fixed physical distance of pulses. But this is not necessary in this case as it only scales the values and does not provide additional information.

A similar signal is created by passing the timer data through a digital filter. This results in a low-pass filtered version of the IAS signal, which is shorter by $k-1$ samples (where k is the filter's length). A diagram of such a filter (here with $k=3$) is shown in Figure 2.7. Both of these are then used to construct a normalised signal according to the following equation:

$$N_i = \frac{D_{i+k-1}}{\frac{FD_i}{C}} \tag{2.1}$$

where D_i is a time difference value, FD_i is the filtered value, and C is a constant - this division helps to keep the resulting normalised values N_i large and better utilise the floating point range. The purpose of normalisation is to remove very low frequencies (around 0) and transform the values into the same range for all speeds tested.

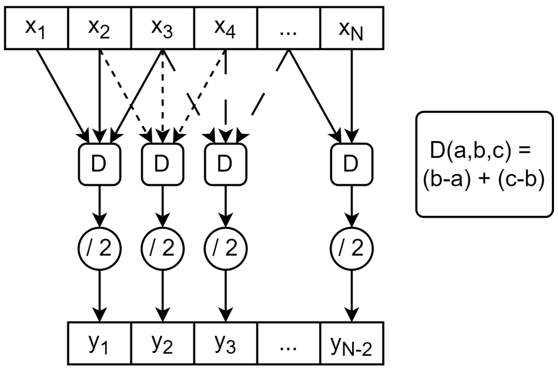


Figure 2.7: Block diagram of the proposed difference filter

2.3.2 Frequency-domain Features

The spectral analysis provides a unique perspective of a given signal. It is used to compute the signal's frequency spectrum, which can, for example, be useful when the application needs to determine if there is a component which creates a known particular pattern in the signal.

The basic mathematical tool for computing the frequency components is the Fourier transform. Notably, for finite sequences of discrete samples, the discrete Fourier transform (DFT) is used. When a series of samples $x_{0\ldots N-1}$ is given, each frequency coefficient X_k is defined by the relation:

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x_n \cdot \exp(\frac{-i2\pi}{N}kn) \\ X_k &= \sum_{n=0}^{N-1} x_n \cdot [\cos(\frac{2\pi}{N}kn) - i \cdot \sin(\frac{2\pi}{N}kn)] \end{aligned} \tag{2.2}$$

There are several advanced implementations derived from this definition, the most prominent being the Fast Fourier transform (FFT) - formulated by the authors of [14]. Despite the generally low computation time of this version, it involves the manipulation of the whole input signal and can only compute all the frequency coefficients, not just a few selected ones. That is why the choice was made to implement Equation 2.2 directly in the code.

As seen in equation 2.2, DFT works with complex input and produces complex numbers as output. It yields N equally spaced values, in the closed interval $\langle -\frac{f_s}{2}; \frac{f_s}{2} \rangle$ (f_s denotes the sampling frequency). For real-valued input signals, the algorithm can use the fact that the values for negative and positive frequencies are symmetric with respect to the y-axis. Therefore, to obtain the true values for the positive (real) coefficient k, simplification can be used

$$\bar{X_k} = 2\,|X_k| \tag{2.3}$$

where the absolute value operation is used to get the real amplitude of a complex number given by its phase and magnitude. Figure 2.8 shows the input and output of DFT and how the presence of a specific signal can be detected by its frequency in the spectrum. If only some frequencies need to be expressed, the values of k can be computed based on f_s and N.

After normalising the signal, DFT is applied to it with several selected frequencies. The calculated values are used directly as features for an ML model.

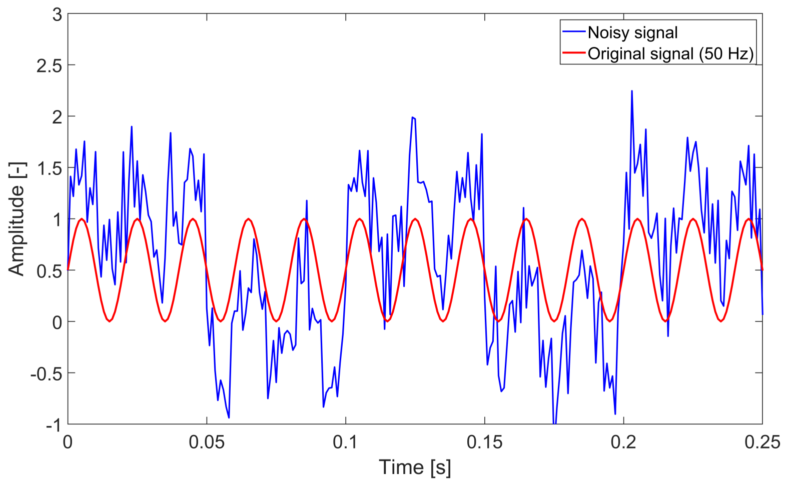
2.3.3 Time-domain Features

As opposed to frequency features, time-domain parameters are computed on the basis of the signal shape and other characteristics in its original form, without computing another representation. They express, for example, the shape or statistical qualities of a given signal. To reduce noise and reduce computation time, an average signal is calculated from the normalised signal specified in section 2.3.1. For an input given by samples $x_{1\ldots N}$ and a desired output length M, this can be represented by the following relation:

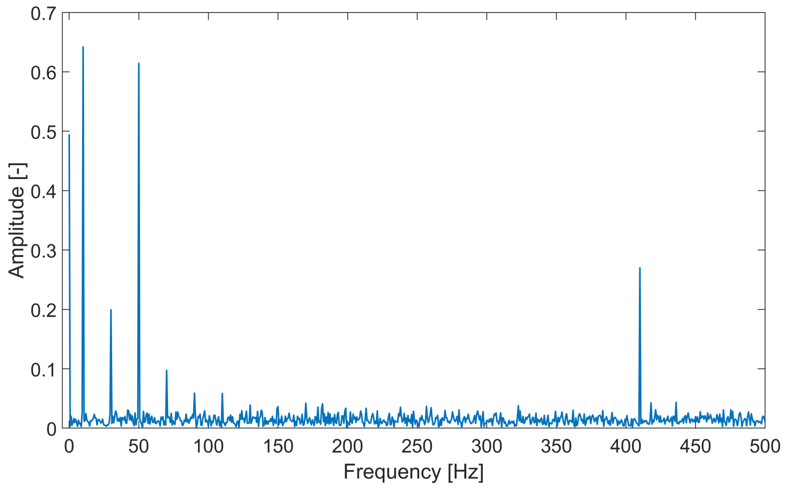
$$\begin{aligned} R &= \frac{N}{M} \\ X_m &= \frac{1}{R}\sum_{r=0}^{R-1} x_{m+r\cdot R},\ m = 1\ldots M \end{aligned} \tag{2.4}$$

Peak-to-RMS

The peak value of a signal is defined as the maximum absolute value that it reaches. Root-mean-square (RMS) is described as the effective stationary value of a nonstationary signal. By dividing these two measured parameters, a characteristic called peak-to-RMS is obtained. When many peaks occur in



(a) : Signal - sine waves with frequencies 50 Hz and 410 Hz, square wave with frequency 10 Hz and white noise



(b) : Frequency spectrum

Figure 2.8: Example of spectral analysis by DFT

a signal with a relatively wide shape and similar amplitudes, they contribute to the RMS value, and therefore the ratio will not be high. But if the peaks are infrequent, they do not have a reasonable impact on the RMS, and the ratio will be higher. The computation is given by the following formulas:

$$\begin{aligned} x_{peak} &= \max_n |x_n| \\ x_{RMS} &= \sqrt{\frac{1}{N} \sum_{n=1}^{N} |x_n|^2} \\ x_{p-rms} &= \frac{x_{peak}}{x_{RMS}} \end{aligned} \tag{2.5}$$

Shape Factor

Shape factor is an indication of a signal's effective value in relation to the mean absolute value, representing its shape but is independent of the absolute dimensions. It is simply the RMS compensated by the mean value. Because RMS is based on the absolute value of a signal, it will increase depending on the peaks and also the mean value. Division by mean removes this effect and leaves a representation of the value variability. Equations for this parameter are:

$$\begin{aligned} x_{MA} &= \frac{1}{N}\sum_{n=1}^{N}|x_n| \\ x_{SF} &= \frac{x_{RMS}}{x_{MA}} \end{aligned} \tag{2.6}$$

Kurtosis

Kurtosis is a measure of the number of outliers a signal contains. From a statistics point of view, it represents the size of side tails in the distribution (or histogram) of values compared to a normal Gaussian distribution. When a signal is perfectly periodic, and so its values repeat without any new ones being introduced (for example, a pure sine wave), its distribution will be narrow. After adding other signal components (noise or peaks caused by a recurring fault), the new values will be located on the sides of the distribution, with fewer occurrences. In this way, it can be seen how much of the signal is composed of irregular data points. Kurtosis increases with an increasing number of outliers. It is given by the relation:

$$\begin{aligned} \bar{x} &= \frac{1}{N}\sum_{n=1}^{N}x_n \\ x_{kurt} &= \frac{\frac{1}{N}\sum_{n=1}^{N}(x_n-\bar{x})^4}{[\frac{1}{N}\sum_{n=1}^{N}(x_n-\bar{x})^2]^2} \end{aligned} \tag{2.7}$$

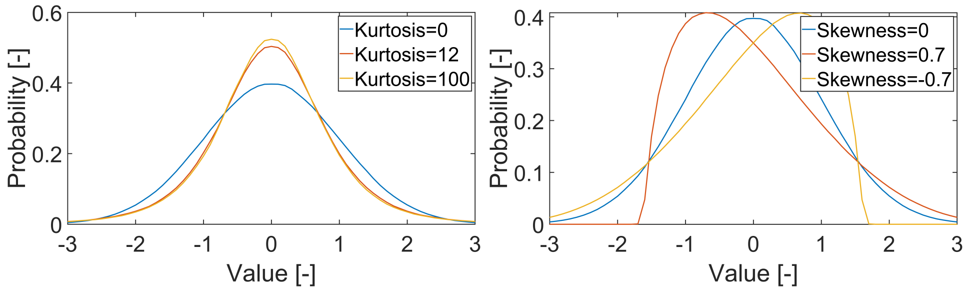
A normal distribution with different kurtosis values is shown in Figure 2.9b.

Skewness

Similarly to kurtosis, skewness is a characteristic of the signal value distribution. More specifically, it measures the symmetry of the distribution. It can be useful, for example, when a mechanical fault leads to a value similar to one that is already present because of normal operation. In this case, this value would have an increased number of occurrences and the peak in the distribution would start moving to one side, as opposed to the presumed original normal distribution. The skewness of a signal is expressed as:

$$x_{skew} = \frac{\frac{1}{N}\sum_{n=1}^{N}(x_n-\bar{x})^3}{[\frac{1}{N}\sum_{n=1}^{N}(x_n-\bar{x})^2]^{\frac{3}{2}}} \tag{2.8}$$

The effect of the skewness value is visualised in Figure 2.9a.



(a) : Different x_{kurt}, $x_{skew} = 0$ **(b) :** Different x_{skew}, $x_{kurt} = 0$

Figure 2.9: Normal distributions with various x_{skew} and x_{kurt}

2.4 Learning Models and Algorithms

In this section, the basic theory of several ML models that were used and tested is covered. It is not a complete list of all models available in NanoEdge AI Studio, only those that were tested in this work. Because NanoEdge AI Studio is licensed as an intellectual property of STMicroelectronics' partner, particular implementation details are not known. A general description will be provided where possible.

2.4.1 Support Vector Machines

The support vector machines (SVM) method was introduced in [15]. It was designed as a binary classifier which learns from a given training data divided into two categories and then assigns one of these classes to new data. Because each training sample has a known label, SVM belongs to the group of supervised ML algorithms.

The component that is learnt is a hyperplane that best separates the two sets of data. A hyperplane is a geometric expression, meaning a subspace whose dimension is one less than that of its ambient space, a line in 2D, a plane in 3D, etc. They can be represented by affine equations in the form:

$$a_1x_1 + a_2x_2 + a_{n-1}x_{n-1} = b \tag{2.9}$$

for n-dimensional space (where a_i and b are the hyperplane's parameters and at least one of a_i must be different from 0). Apart from binary classification, it can be used with multiple classes by determining hyperplanes for either each pair of groups or each group and the rest.

In the most simplistic form, this problem could be seen as finding a line separating two sets of points in a 2D plane. If there is such a line, one (or more) points can be defined from each set that is closest to the line. Then the distance between an arbitrary line and its corresponding nearest points can be measured to find the line that maximises this criterion while still dividing the classes. The two points are called support vectors because, when a line is set, removing either of them leads to changing the value of the criterion and potentially altering the best line.

If the points are not linearly separable, as is the case in most problems, the authors proposed a non-linear function (called a kernel) that maps the input vectors into a higher dimensional space where they can be separated. This can also involve warping of the original Euclidian space - for example, a circle can be seen as a linear object when viewed through polar coordinates. They also implemented the idea that the hyperplane does not have to divide two sets of vectors precisely, if it is not possible without transforming them into a space whose dimension is too high. In such a case, a soft-margin hyperplane is introduced. This is found by minimising the number of training errors (vectors that are separated from their corresponding set) while maximising the distance between the hyperplane and its support vectors.

Training and testing data can generally be of any type. However, in this case, simply the numerical representation of the signal parameters which were described in Section 2.3 can be used. In Figure 2.9, several data sets are shown with their separating hyperplanes.

2.4.2 Random Forest

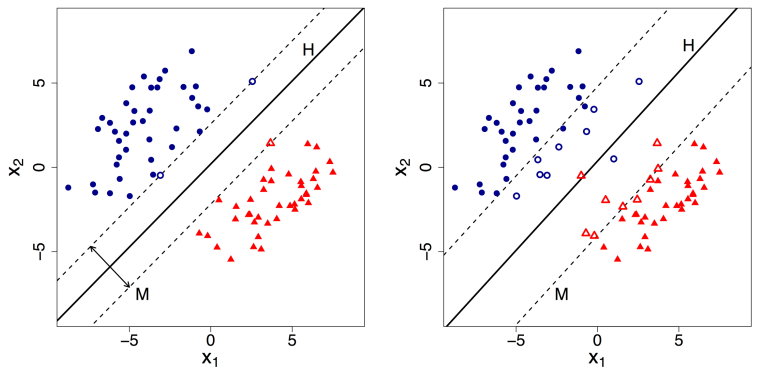
The basic part of the random forest (RF) algorithm, proposed in [17] and later improved in [18], is a decision tree. Similarly to SVM, this is also a supervised ML algorithm.

In decision trees, the input vector (a set of numbers) is subjected to a series of comparisons. Each node in the tree compares one of the input values to its learnt parameter. Based on the result, the following tree branches are explored until a decision about the input class is reached. Not all input values need to be used. It can naturally be used for multiclass as well as binary classification. Figure 2.11 shows the structure of a simple decision tree. In the learning stage, the input values and the fixed parameters are determined.

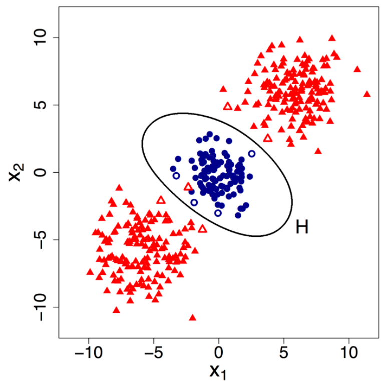
In RF learning, many decision trees are created, and each is assigned a random group of training data. Additionally, every tree uses a randomly specified subset of the input vector. This improves accuracy and prevents overfitting. The overall output prediction is given by a majority vote scheme of the individual predictions.

2.4.3 Multilayer Perceptron

A perceptron is the simplest ML model, which multiplies the input vector by a vector of learnt weights, optionally adds a bias, and passes the result



(a) : Linear hyperplane with hard (on the left) and soft (right) margin



(b) : Nonlinear hyperplane

Figure 2.10: Examples of SVM hyperplanes, courtesy of [16]

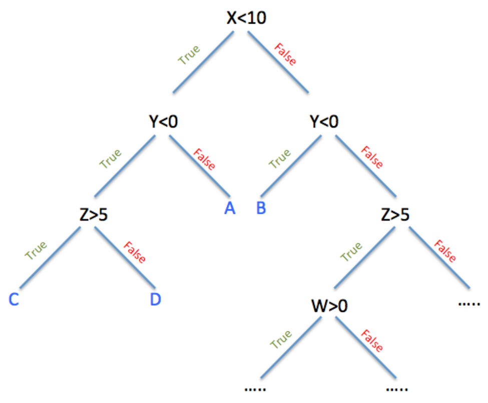


Figure 2.11: Example of a decision tree, from [19]

through an activation function. The last step is usually a linear function that serves to change the output's range. By comparing the output to a defined value, for example, using a step function, the input's class can be determined. During learning, backpropagation is used to update the weights based on the prediction compared to the actual class. The structure of a perceptron is illustrated in Figure 2.12.

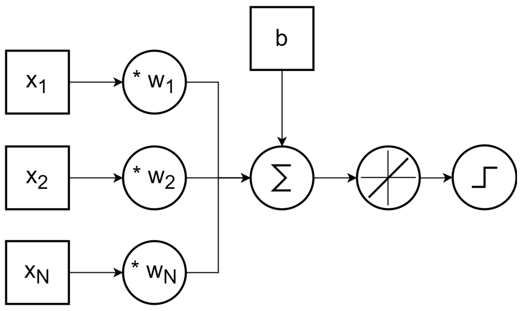


Figure 2.12: Schema of a single perceptron

As an extension of this, several perceptrons can be combined in layers to create a multilayer perceptron (MLP), which is described in [20]. MLP typically uses fully connected layers, meaning that the input vector is the same for each perceptron in the first layer, outputs of this layer are then connected to each input of the second layer, etc. As opposed to single perceptrons, MLP uses non-linear activation functions - if all the steps were linear, it would be possible to reduce the model to just one perceptron. The final layer can be composed of several perceptrons for multiclass problems.

2.4.4 Fast Incremental Support Vector Data Description

Support vector data description is a modification of SVM which can be used for single-class classification (also known as outlier detection). Single-class classification refers to the process of determining whether it is statically probable that a given sample is part of the same distribution as the learnt data.

The authors of [21] proposed a quick learning variation called fast incremental support vector data description (FISVDD). The advantage of this method is that, in each learning step, it focuses on the current support records, not all data points. It also utilises only matrix operations, which can be implemented quickly even with limited hardware options. This learning method can be described as unsupervised because the whole dataset is presumed to be part of one class, so individual labels do not need to be considered.

2.4.5 Z-score Model

In NanoEdge AI Studio, an algorithm called the Z-score model is used for anomaly detection. This task is similar to single-class classification. The difference is that, generally, the data which is considered abnormal can be part of the same distribution but with a different magnitude. The practical implication is that (for models in NanoEdge AI Studio) the model can be adjusted after learning. During training, the general "shape" of a separation function is found, and when new data (which is known to come from a correct class) is obtained, the dimension of this function can be moved.

For this algorithm, specific details about its function are unavailable because of its proprietary licence. But it is probably based on the statistical measurement of z-score - two normal distribution functions are built based on means and standard deviations of a good and an anomalous class. When dividing by a standard deviation, the score is normalised - this negates possible differences in the measured units or magnitudes. This is an unsupervised technique.

Chapter 3

Demonstrator Design and Realisation

In the following chapter, a description is provided of the implementation of the theory presented in Chapter 2 and the designed system. The individual components, such as a motor, MCUs or processing software, will be discussed. For the ML part of the system, the models that were used together with the basic usage of NanoEdge AI Studio will be outlined.

3.1 Used Software

In this work, the following STMicroelectronics development tools were used: CubeMX (version 6.6.1) for MCU configuration, CubeIDE (version 1.7.0) for code editing and compilation, Motor Control Software Development Kit (version 6.0.0) for creating and configuring the motor control firmware, and finally NanoEdge AI Studio (version 3.3.0) for training and exporting machine learning models. NanoEdge AI Studio and its features will be discussed in more depth in Section 3.6. The firmware packages for F4 (version 1.27.1), G4 (version 1.5.0), and H7 (version 1.10.0) MCUs were used in CubeMX. All tools are available at `https://www.st.com/`.

For gearbox design, Autodesk Fusion 360 was used (with an educational licence available on `https://www.autodesk.com/`) together with a free community-made plugin called GF Gear Generator (`https://apps.autodesk.com/FUSION/en/Detail/Index?id=1236778940008086660`). Several simple scripts were written in the Python programming language (`https://www.python.org/`) for communication with the MCU and data collection. Matlab software (version R2021a, with an educational licence, `https://www.mathworks.com/products/matlab.html`) was used for initial signal processing and analysis after collecting samples using the MCU.

3.2 Hardware Setup and Drivers

Here, a description of the hardware chosen and the proposed demonstrator will be provided.

3.2.1 Motor and Microcontrollers

The motor used is a Shinano LA052 brushless direct current (BLDC) electric motor with a built-in optical incremental rotary encoder (IRC). It is connected to an STM32 Nucleo-G431RB board (with an STM32G431RB MCU) through an X-NUCLEO-IHM16M1 power board. The power board features a BLDC driver, sensing resistors, protections, etc., and enables the MCU to work with high voltage levels. It is also equipped with an adapter to connect an external power supply to the motor. The motor is shown in Figure 3.1, and its basic parameters are summarised in Table 3.1.

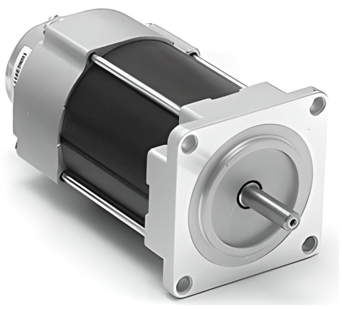


Figure 3.1: Shinano LA052 motor

Motor rated parameters					
Power	Torque	Speed	Voltage	Current	Pole-pairs
80 W	0.25 Nm	3000 r/min	24 V	4.6 A	2

Motor dimensions		
Width, height	Length	Mass
56.4 mm	86.1 mm	0.6 kg

Encoder (optical)			
Voltage	Current	Signals	Resolution
5 V	50 mA	2 square	400 pulse/r

Table 3.1: Motor properties

A separate STM32 F413 Discovery kit (with an STM32F413ZH MCU) is connected to the motor's IRC. This MCU was chosen for the fault detection part of the project because of its relatively low price and computing power compared to some newer models available. It is part of STMicroelectronics' high-performance access line. Table 3.2 summarises several important characteristics of this MCU. Apart from the MCU itself, another important requirement for this work was external RAM memory - the memory that is mounted on the Discovery kit (with a size of 4 Mb) was used. The AI libraries (and the entire proposed system) could be implemented on one MCU, along with motor control software. But the decision was made to use two MCUs to better demonstrate the ability to add a similar unit to an already established system.

For the purposes of the final demonstrator, an LCD display that is a part of the Discovery kit was also used. Because the F4 MCU does not support an Ethernet interface, an STM32 Nucleo-H753ZI board (with an STM32H753ZI MCU) to enable Ethernet communication was added.

Core	Voltage	Current	Clock speed
Arm 32-bit Cortex-M4	1.7 - 3.6 V	112 $\mu A/MHz$	100 MHz
Floating point unit	Flash	RAM	Price (approx.)
32-bit	1.5 MB	320 kb	9 $

Table 3.2: Basic parameters of STM32F413ZH MCU

3.2.2 IRC Sensor

As mentioned in Table 3.1, the LA052 motor has a built-in optical IRC sensor with a square wave output signal. Figure 3.2 shows the basic structure of an optical IRC. As the code wheel rotates, a light source shining through the slits creates two sine waves on the photosensors below the wheel. These signals can be sampled directly by an analogue-to-digital converter (ADC) or passed through a subsequent comparator circuit, which creates digital pulse signals.

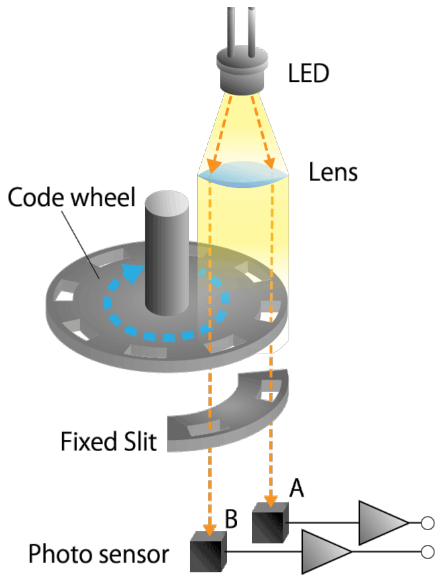


Figure 3.2: Function of an optical rotary encoder, courtesy of [7]

3.2.3 3D Printed Gearbox

After selecting the motor, a support structure was designed to which it can be attached from the inside with screws. The shafts located at the top are printed together with the support as one piece. Gear wheels of defined sizes can be mounted on these shafts. After the motor is fixed in place, the driven gear is installed using a simple coupling mechanism with a pinch bolt. Figures 3.3 and 3.4 show how the printed gearbox looks.

The whole development process was done using this gearbox, and then a second, slightly different, piece was designed and printed for the purpose of the final demonstration unit. This is described in Section 3.7.

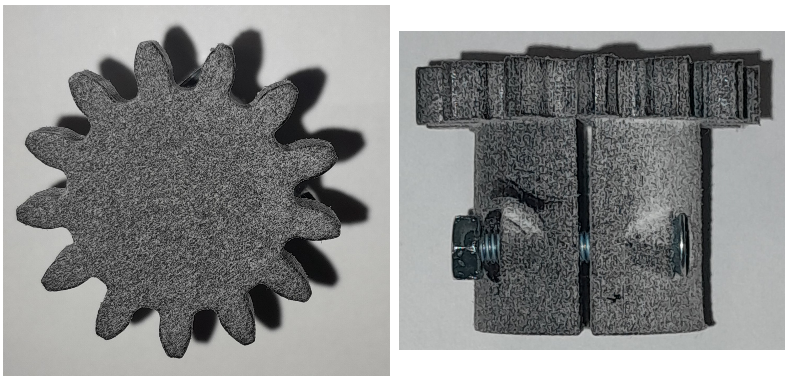


Figure 3.3: Close view of a gear

All gear wheels were designed as spur gears with a module of 0.3 mm, a pressure angle of 14.5 degrees and a height of 5 mm. Overall, the gearbox contains five wheels. The first one is connected to the motor, and the last one has a plate attached to it, which creates air resistance and therefore load for the motor. These two wheels have one gear ring each, and the other three have two rings each. This creates a larger overall gear ratio, which makes it possible to determine the limit of the trained models in terms of the distance of a fault to the sensor - the higher the gear ratio between them, the more information is lost in transmission. The numbers of teeth on individual wheels are shown in Table 3.3. The resulting ratio of 0.05 means that the motor has to turn 20 times, while the last wheel only turns once. In addition to the original undamaged gears, cogs with two types of faults were printed: a missing tooth and a dented one. To do this, a tooth was simply removed in the 3D sketch or used an ellipse-shaped cutout. Figure 3.5 shows the resulting drawings in the Fusion software. It also illustrates how the gears should mesh (in the case of ideal printing).

The supporting box has a length of 20 cm, a width of 18 cm, and a height of 10.7 cm. The top wall is 7 mm thick, while the sides are thinner at 3 mm. Everything was printed on an HP Multi Jet Fusion 3D printer with a layer height of 0.2 mm and 100% infill (no free space inside the structure). The material used for printing was PA12 nylon filament.

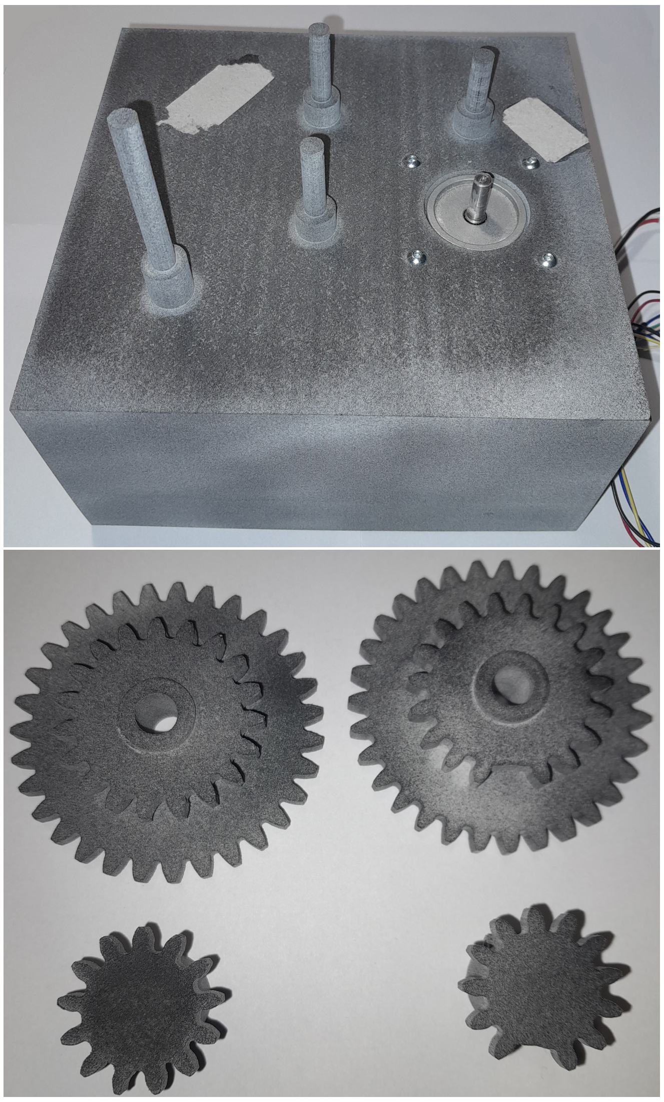


Figure 3.4: Printed gearbox parts

Wheel	1	2	3	4	5
N_1	-	31	43	37	39
N_2	13	19	22	17	-
Gear ratio to motor	1	0.42	0.19	0.11	0.05

Table 3.3: Gearing ratios. N_1 is the number of teeth on the ring connected to the previous gear, N_2 to the next one.

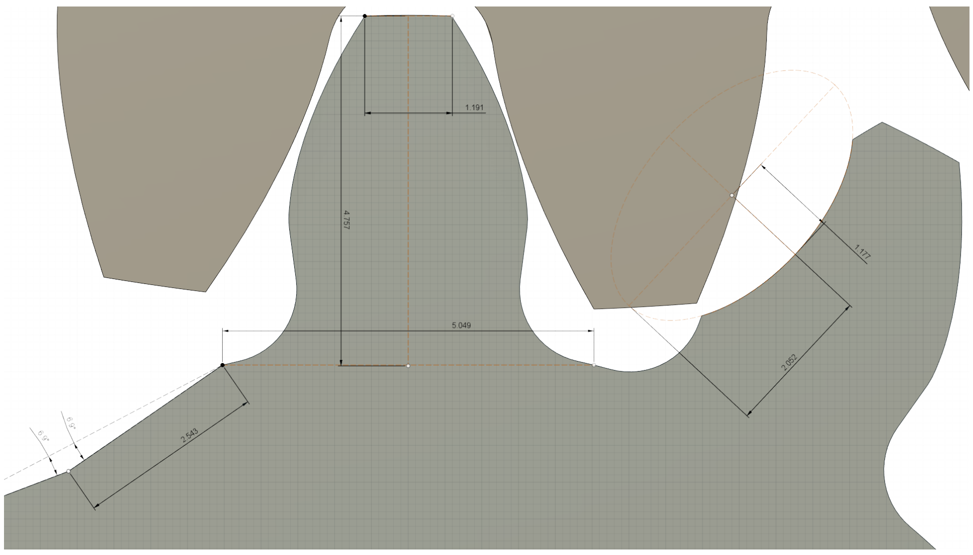


Figure 3.5: Design of gear faults. Dimensions in mm.

3.3 Basic Firmware

Before moving on to the description of how data is collected and other more complex parts of the software, this section describes this work's approach to motor control. This section will also characterise communication between the MCUs and several MCU peripherals, which will later be used to carry out the signal acquisition.

3.3.1 Field Oriented Control

One of the readily available motor control schemes was used in the Motor Control Software Development Kit (MCSDK). In particular, it uses Field Oriented Control ([22]), a method that uses mathematical transformations to convert three stator currents into two orthogonal components of a vector. The components represent the magnetic flux and torque of the motor at each time moment. These values are continuously used as input to proportional-integral (PI) controllers ([23]) that use feedback regulation to keep the values as close as possible to the set references. The flux reference point is fixed to be constant at zero. The torque controller's reference is given by the output of another PI controller whose input is the motor's instantaneous speed, and reference is set by the application. The outputs of the torque and flux regulators are mathematically transformed and then used to generate three pulse-width-modulated (PWM) signals that drive the motor.

To achieve good speed stability, 20 kHz is used as the frequency for the torque and flux regulators, which means that motor currents are measured, and PWM values are recomputed every 50 μs. The speed regulator works similarly at a frequency of 1 kHz. The real speed of the motor is estimated using the encoder. Figure 3.6 shows how the resulting speed is regulated using this method. The stability of the motor's rotation speed and its implications will be discussed in Section 4.1.

After setting the necessary parameters in the MCSDK, most of which have already been determined for this motor by STMicroelectronics, the MCSDK provides a compiled library that encapsulates the controller and all necessary measurements. It enables the user to control the motor with several simple instructions, for example, to rotate it to a certain degree or spin at a desired frequency. The higher-level application code is a simple state machine that supplies commands to the Motor Control.

3.3.2 Used Peripherals and Connections

To ensure there is enough memory space for other needs, the data captured by the MCU's timer are stored in an external pseudo-static random-access memory (PSRAM), which is a part of the F4 Discovery kit. A peripheral of the MCU called the Flexible static memory controller (FSMC) works as an

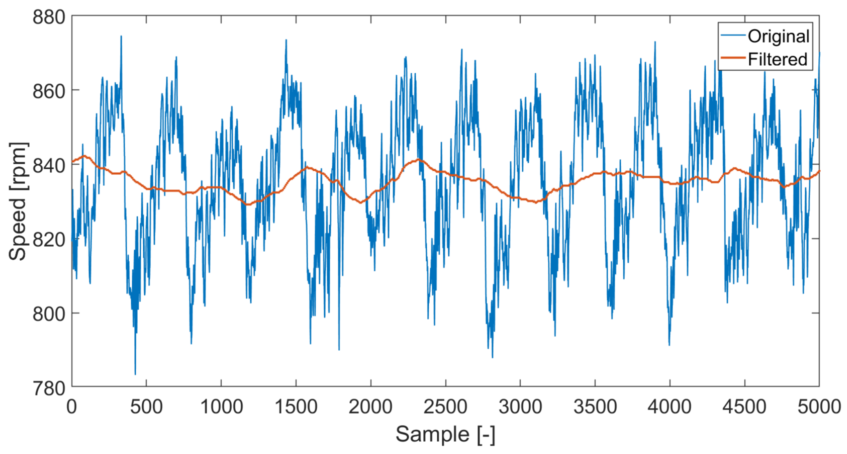


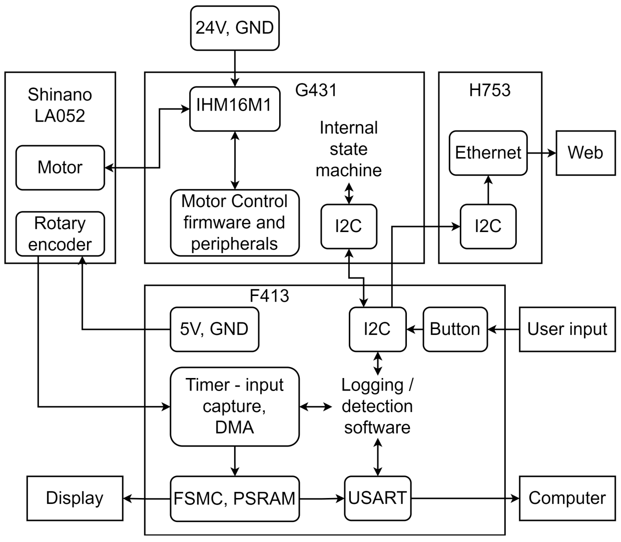
Figure 3.6: Measured speed of the motor controlled by the FOC algorithm. Reference set to 840 rpm.

interface between the MCU and the memory. All peripheral and hardware connections are symbolically shown in Figure 3.7a.

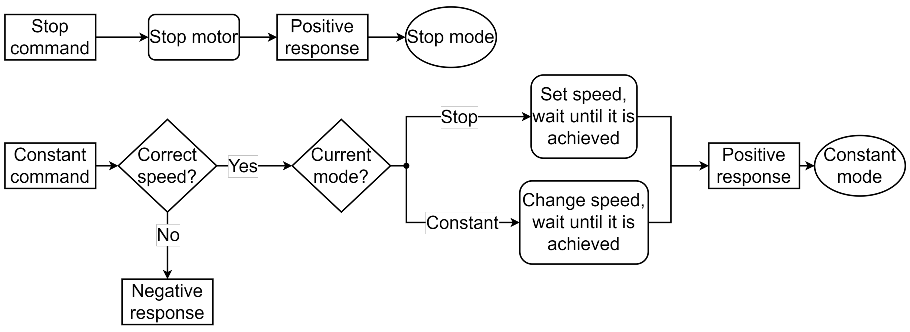
To send commands to the G4 MCU, the Inter-integrated circuit interface (I2C) peripheral is used. Stop and run commands (with an appropriate speed) can be sent. This is shown in Figure 3.7b. The connected button is used to start detection by a user action. Communication with a computer (transferring gathered data for logging purposes) is done with the help of the Universal synchronous/asynchronous receiver transmitter peripheral.

The I2C bus is also used to communicate with the H7 MCU, which then forwards the detected gearbox state to a website using its Ethernet interface. Apart from memory, FSMC is connected to a display embedded in the F4 Discovery kit.

Used pins of the F4 MCU and associated peripherals are listed in Table 3.4.



(a) : Hardware connections



(b) : Diagram of communication between MCUs

Figure 3.7: Symbolic diagrams of connections and communication between units

PE3	PF0	PF1	PF2	PF3	PF4	PF5
Output	AF	AF	AF	AF	AF	AF
Led	FSMC_A0	FSMC_A1	FSMC_A2	FSMC_A3	FSMC_A4	FSMC_A5
PA0	PA1	PC5	PF12	PF13	PF14	PF15
EXTI, PD	AF, PU	Output	AF	AF	AF	AF
Button	TIM2_CH2	Led	FSMC_A6	FSMC_A7	FSMC_A8	FSMC_A9
PG0	PG1	PE7	PE8	PE9	PE10	PE11
AF	AF	AF	AF	AF	AF	AF
FSMC_A10	FSMC_A11	FSMC_D4	FSMC_D5	FSMC_D6	FSMC_D7	FSMC_D8
PE12	PE13	PE14	PE15	PB10	PB11	PD8
AF	AF	AF	AF	AF	AF	AF
FSMC_D9	FSMC_D10	FSMC_D11	FSMC_D12	I2C2_SCL	I2C2_SDA	FSMC_D13
PD9	PD10	PD11	PD12	PD14	PD15	PG2
AF	AF	AF	AF	AF	AF	AF
FSMC_D14	FSMC_D15	FSMC_A16	FSMC_A17	FSMC_D0	FSMC_D1	FSMC_A12
PG3	PG4	PG5	PA15	PD0	PD1	PD4
AF	AF	AF	Output, PD	AF	AF	AF
FSMC_A13	FSMC_A14	FSMC_A15	ETH_SS	FSMC_D2	FSMC_D3	FSMC_NOE
PD5	PD7	PG9	PG10	PG14	PE0	PE1
AF	AF	AF	AF	AF	AF	AF
FSMC_NWE	FSMC_NE1	USART6_RX	FSMC_NE3	USART6_TX	FSMC_NBL0	FSMC_NBL1

AF - alternate function PU/D - pull-up/down resistor

Table 3.4: Used pins of STM32F413ZH MCU, their modes and functions

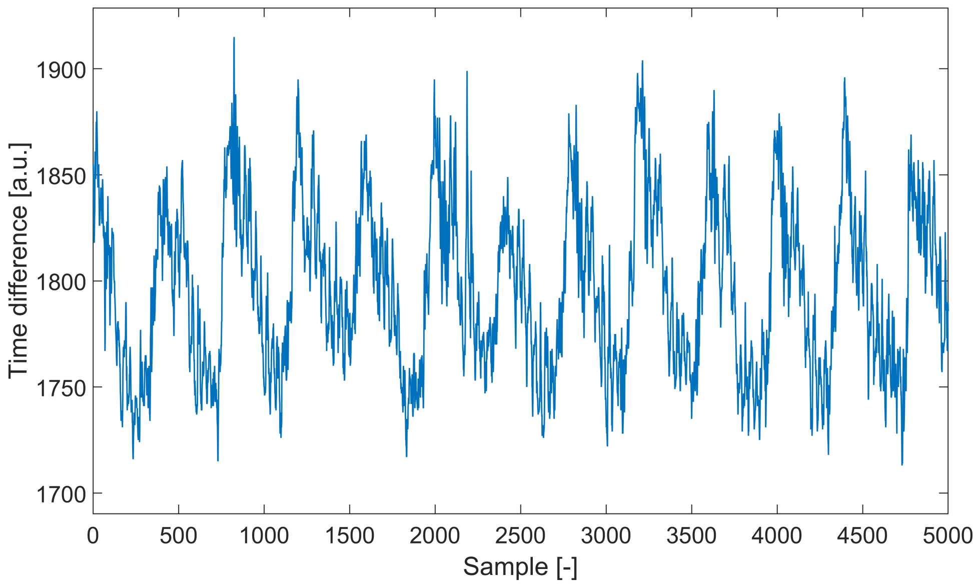
3.4 Timer Parameters and Real IRC Signal

The timer used in the F413 MCU has a 32-bit counter register. That means that every time a pulse from the motor is registered, it saves (through DMA) a 32-bit unsigned integer representing the current timestamp. The available 4 Mb of memory can store $N_{mem} = 4194304b = 2^{22}b = 2^{19}B = 2^{17}$ integers. Assuming a rotation speed of 1500 rpm and 400 pulses per rotation, the pulses will have a frequency $f_p = \frac{1500*400}{60} = 10kHz$ and a period $T_p = \frac{1}{f_p} = 100\mu s$. To properly determine the timestamp of a pulse when an error occurs, the sampling frequency f_s needs to be at least 10 times higher than f_p. This is the low limit for $f_{s,min} = 100kHz$. The high limit for f_s is given by the need for the timer not to overflow during collection. Based on the previous assumption, it can also be seen that to collect the maximum number of samples 2^{17}, the motor must spin for $T_{spin} = \frac{2^{17}}{10^4} \approx 13s$. At its maximum frequency $f_{s,max} = 100MHz$, the timer will overflow in $\frac{2^{32}}{10^8} \approx 43s$. If the timer overflows during collection, the data get corrupted, and either the samples recorded after the overflow would need to be thrown away or larger (64-bit) integers could be used. For verification of lower speeds, the minimum speed can be 500 rpm. Then $f_p \approx 3.33kHz$ and $T_{spin} \approx 39s$. To have a certain safety, $f_s = 10MHz$ will be used. This means that time shifts as small as $\frac{1}{f_s} = 100ns$ can be recorded, and overflow occurs after approximately 429 s. So, there is no risk of the timer overflowing in the given conditions, even during low-speed operation.

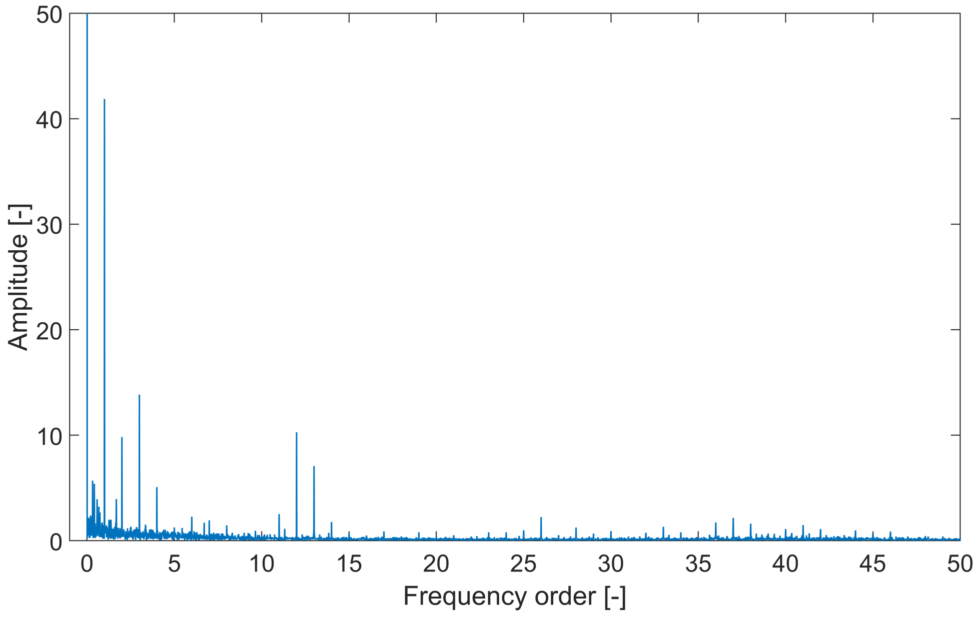
An estimate was made that to correctly detect an event; it should be registered at least five times in one input signal. So when aiming to recognise a fault on a given gear wheel, the signal needs to be long enough to cover five whole rotations of that wheel. Based on this, the number of pulses per motor revolution (400) and the overall gearing ratio of the designed system (20), signal length was set to $N = 5 \cdot 400 \cdot 20$ samples.

A real signal measured on the gearbox and its frequency spectrum are shown in Figure 3.8. It should be compared with Figures 2.5b and 2.5c.

In contrast to the analysis of an idealised signal, it can be seen that the signal's shape is quite dissimilar. The most obvious difference is the presence of a harmonic component, which causes near-sinusoidal changes in the signal with a period of 400 samples - which corresponds to one rotation of the motor. This is probably caused by the motor control algorithm and can make it difficult to precisely identify changes in the signal at this frequency. There are also many other frequencies, most of which are probably associated with measurement noise.



(a) : Signal. An auxiliary unit is used in place of the timer's internal counter state, tied to the real time through the timer's frequency.



(b) : Order spectrum. Values on the x-axis are multiples of motor rotations and were recomputed using the encoder's resolution. Only the first quarter of the x-axis is shown to keep the graph readable, and the y-axis is clipped as well.

Figure 3.8: Example of a real signal captured by a timer without a gearbox fault

3.5 Processing Implementation on MCU

To implement the algorithms described in Section 2.3, only the standard maths library in the C language was used. The MCU's Floating point unit (FPU), which is a part of the MCU core, also provides an advantage and enables the application to perform calculations with floating point numbers natively on the hardware instead of using the basic core operations. This offers a significant reduction in computation time.

In Section 2.3.1, a difference filter was defined - the specific length of this filter was $k = 400$, corresponding to the sensor resolution. The normalising constant in Equation 2.1 was set to $C = 10^{10}$. This proved effective in increasing the used floating point range while still avoiding overflow. For the averaged signal length, specified by equation 2.4, $M = 3600$ was used, which is equal to nine motor rotations. When computing the kurtosis and skewness values, overflows were encountered, caused by the multiplication of C and the power operations in their definition. This was subsequently avoided by dividing the number under the power operator by 10^4, so the expression $x_n - \bar{x}$ in Equations 2.7 and 2.8 would become $\frac{x_n - \bar{x}}{10^4}$. The output values were then multiplied by 10^6 to obtain a larger value closer in range to the frequency parameters.

The normalised signal is computed in place from the originally captured data to avoid the need to create two additional signals and save memory space. Pseudocodes of the implemented functions can be seen in the code snippets 1 - 5.

Algorithm 1 Definitions for the processing algorithm

```
$RECORD\_LEN = 5 \cdot 20 \cdot 400$
$FILTER\_N = 400$
$NORM\_LEN = RECORD\_LEN - FILTER\_N$
$MEAN\_LEN = 9 \cdot 400$
$MEAN\_PERIODS = NORM\_LEN/MEAN\_LEN$
$NORM\_MULT = 10^{10}$
(float) $FEATURE\_DIV = 10^4$
(float) $FEATURE\_NORM = 10^6$
```

Algorithm 2 Signal normalisation

```
function NORMALISE(signal)
    (float) $sum\_long = 0$, $sum\_short = 0$, $value = 0$

    # First round of filter, fill up buffer
    for $i = 1 \dots FILTER\_N$ do
        $sum\_long \mathrel{+}= signal[i] - signal[i-1]$
    end for
    $sum\_short = signal[FILTER\_N] - signal[FILTER\_N - 1]$

    $value = sum\_short \cdot NORM\_MULT \cdot FILTER\_N / sum\_long$

    # Rest of filter
    $j = 0$
    for $i = FILTER\_N \dots RECORD\_LEN - 1$ do
        $sum\_short = signal[i] - signal[i-1]$
        $sum\_long \mathrel{+}= sum\_short$
        $prev = signal[i - FILTER\_N] - signal[i - FILTER\_N - 1]$
        $sum\_long \mathrel{-}= prev$
        $signal[j] = value$
        $j \mathrel{+}= 1$
        $value = sum\_short \cdot NORM\_MULT \cdot FILTER\_N / sum\_long$
    end for
    $signal[j] = value$
end function
```

Algorithm 3 DFT implementation

```
function DFT(signal, $dft\_coeff[]$)
    (float) $Xre = 0$, $Xim = 0$
    (float) $k2pi\_const = 0$, $k2pinN\_const = 0$
    $dft\_n = length(dft\_coeff)$
    (float) $output[dft\_n]$

    for $i = 0 \dots dft\_n - 1$ do
        $k = dft\_coeff[i]$
        $Xre = 0$
        $Xim = 0$
        $k2pi\_const = 2 \cdot k \cdot \pi$

        for $n = 0 \dots NORM\_LEN - 1$ do
            $k2pinN\_const = k2pi\_const \cdot n \cdot NORM\_LEN$
            $Xre \mathrel{+}= (signal[n] \cdot cos(k2pinN\_const))$
            $Xim \mathrel{-}= (signal[n] \cdot sin(k2pinN\_const))$
        end for
        $output[i] = \sqrt{Xre^2 + Xim^2} \cdot 2/NORM\_LEN$
    end for
    return $output$
end function
```

Algorithm 4 Average signal calculation

```
function AVERAGE(signal)
    (float) $output[MEAN\_LEN]$
    for $i = 0 \dots MEAN\_LEN - 1$ do
        for $j = i \dots NORM\_LEN - 1, j \mathrel{+}= MEAN\_LEN$ do
            $output[i] \mathrel{+}= signal[j]$
        end for
        $output[i] \mathrel{/}= MEAN\_PERIODS$
    end for
    return $output$
end function
```

Algorithm 5 Features calculation

```
function FEATURES(signal)
    (float) $rms\_v = 0$,$mean\_v = 0$,$peak\_v = 0$,$value = 0$
    (float) $skew\_n = 0$,$skew\_d = 0$,$kurt\_n = 0$
    for $i = 0 \dots MEAN\_LEN - 1$ do
        $x = signal[i]$
        if $x > peak\_v$ then
            $peak\_v = x$
        end if
        $rms\_v \mathrel{+}= x^2$
        $mean\_v \mathrel{+}= x$
    end for
    $rms\_v = \sqrt{rms\_v/MEAN\_LEN}$
    $mean\_v \mathrel{/}= MEAN\_LEN$
    $shape\_f = rms\_v \cdot FEATURE\_NORM/mean\_v$
    $peak2rms = peak\_v \cdot FEATURE\_NORM/rms\_v$

    for $i = 0 \dots MEAN\_LEN - 1$ do
        $x = (signal[i] - mean\_v)/FEATURE\_DIV$
        $skew\_n \mathrel{+}= x^3$
        $skew\_d \mathrel{+}= x^2$
        $kurt\_n \mathrel{+}= x^4$
    end for
    $skew\_n \mathrel{/}= MEAN\_LEN$
    $kurt\_n \mathrel{/}= MEAN\_LEN$
    $kurt\_d = (skew\_d/MEAN\_LEN)^2$
    $skew\_d = \sqrt{skew\_d/MEAN\_LEN}$
    $skew\_d = skew\_d^3$
    $skew = skew\_n \cdot FEATURE\_NORM/skew\_d$
    $kurt = kurt\_n \cdot FEATURE\_NORM/kurt\_d$
    return $shape\_f, peak2rms, skew, kurt$
end function
```

3.6 NanoEdge AI Studio

In this section, the basic usage of NanoEdge AI Studio will be shown. It is not an exhaustive manual or a list of components but rather a brief description of the functionalities that were used in this work.

Direct data logging was not used as it is not available for the chosen MCU kit. After importing data into the software, it is possible to see statistical characteristics of signals, for example, minimum and maximum values, mean and standard deviation along with a frequency spectrum. This analysis is more useful for raw measured data and not really significant for already computed features.

The length of imported signals is limited to 2^{14}, which is probably set because the algorithm implementations use libraries made by ARM specifically for each MCU core that work with this limit as well. The learnt models can then include signal processing, for example, frequency spectrum calculation.

This limit would not be enough for importing the whole signal, which is 40000 samples. For this reason, all features were computed in code, and only the ML part of NanoEdge software was used.

3.6.1 Project Types

The software offers four types of ML projects for different detection purposes. Their main features, necessary input data, and possible models will be described. A project category called extrapolation will not be covered as this is used for inferring numerical values, not class labels, and was not relevant to this work.

Anomaly Detection

An anomaly detection project uses signals of two types for training: regular and abnormal. After training from these signals on a computer, the exported model can continue learning on the target hardware (MCU) - this is done by passing signals from normal conditions to the model. It is even expected that the MCU will go through a number of learning iterations to achieve the correct results. The suggested amount is specified after training.

The general advantage of this type of project is that a model can be trained centrally and then uploaded to many machines of the same type, each with slightly different parameters. The model then adapts to each particular machine. Another aim of anomaly detection can be seen as detecting sudden mechanical changes in a machine. By learning from new signals at regular intervals, the model can adjust to machine wear or similar slow effects. If there is a fault causing an abrupt alteration of the parameters, it is detected by inference. Of course, after completing the initial learning phase, it is not necessary to continue learning, in which case the system would be similar to 1-class classification.

A screen with the results of training an anomaly detection model is shown in Figure 3.9. The accuracy of the model and its 95% confidence interval can be seen along with the memory requirements and the classification of the training samples. The suggested number of learning iterations is visible on the right side.

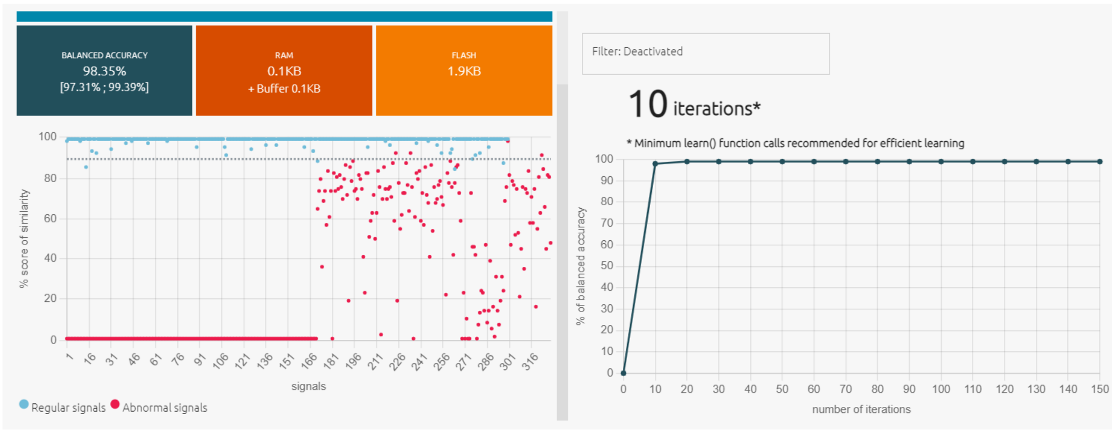


Figure 3.9: Anomaly detection training results in NanoEdge AI Studio

Algorithms such as the Z-score model (discussed in Section 2.4.5) are used here. Their output is a single number representing the input signal's similarity to the regular class. This is influenced by a parameter called sensitivity, which can be modified on the MCU between individual inferences. A sensitivity value between 0 and 1 (excluded) decreases the sensitivity of the model, while a value between 1 and 100 increases it. A higher value tends to decrease the percentages of similarity returned (the algorithm is more sensitive to perturbations), while the opposite is true for lower sensitivity values. To obtain a class label (regular or abnormal) for the signal, the output can be compared (by the user) to a defined threshold.

1-class Classification

A 1-class classification project needs only regular signals for training. It is similar to anomaly detection, except that the model's parameters are fixed after training, and it cannot learn on an MCU. It should detect all changes, sudden or slow, as long as they are pronounced enough.

Methods such as FISVDD (described in Section 2.4.4) are used. The output of a learnt model is a class label, either regular or abnormal.

N-class Classification

With n-class classification, the user can specify the number of classes that should be trained.

ML algorithms such as SVM or RF (detailed in Section 2.4) are trained. The model output is a set of probabilities corresponding to each class. A signal gets a label assigned based on the class with the highest probability. The output can also be a so-called "unknown" state, which means that the signal does not properly resemble any of the trained classes.

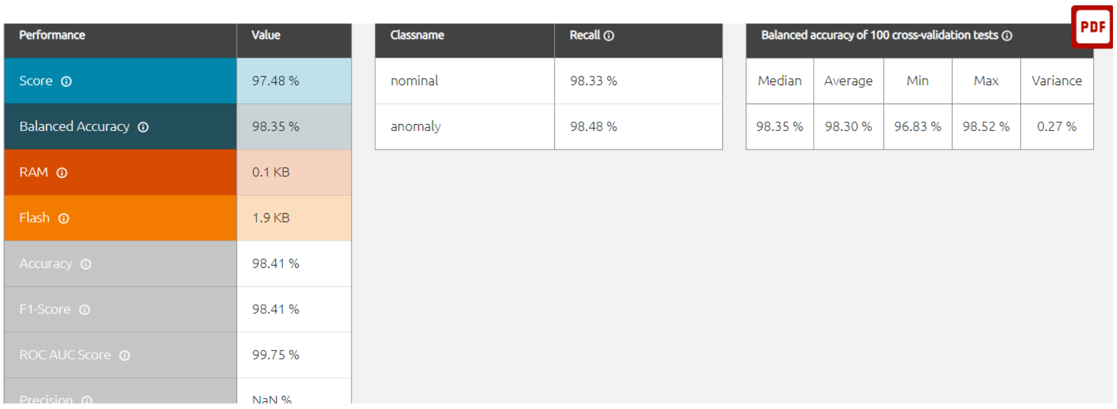
3.6.2 Creating a Model

After starting a benchmark, the software looks for suitable libraries and parameters and outputs the models that performed the best.

After training, the user can select a model to continue working with based on memory requirements, accuracy, etc. NanoEdge software also offers an emulator, which enables users to better estimate a model's performance. The emulator shows what results the model should provide for input data that were not seen during training. This is related to how the collected data should be split by the user. When training, the software uses all the available data. K-fold cross-validation (described in article [24]) is used to internally evaluate the performance of each model during training. The number of folds K is determined by the software based on the size of the dataset and project type.

A general recommendation was followed to save 30 signals from each class that were not included in the training set. These were then used to compare the models in both the emulator and the MCU. It is also important to compare several models that use various algorithms and have different memory requirements, as they can vary in accuracy when transferred to an MCU. When these metrics were combined, the most appropriate models (for this given scenario) for each type of project were selected. The summary of the data collected and the results of the evaluation on the MCU will be presented in Chapter 5.

Figure 3.10a shows the software's display of training results for a particular model. In Figure 3.10b, the emulator window can be seen for the selected anomaly detection model with the sensitivity set to 0.8 and after learning from 63 samples. The input data consisted of 30 signals of each state: without fault, fault on the first and second wheels. Only three of the regular signals were misclassified.

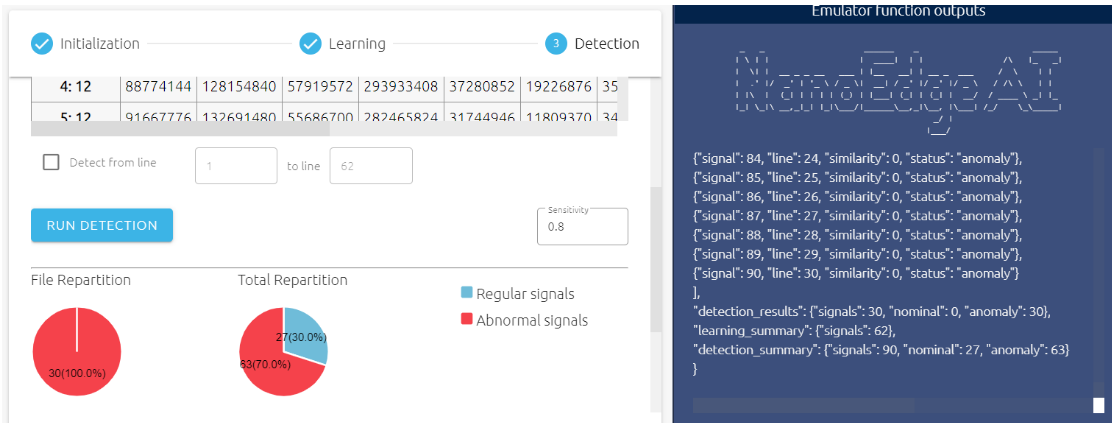


Performance	Value
Score	97.48 %
Balanced Accuracy	98.35 %
RAM	0.1 KB
Flash	1.9 KB
Accuracy	98.41 %
F1-Score	98.41 %
ROC AUC Score	99.75 %
Precision	NaN %

Classname	Recall
nominal	98.33 %
anomaly	98.48 %

Balanced accuracy of 100 cross-validation tests				
Median	Average	Min	Max	Variance
98.35 %	98.30 %	96.83 %	98.52 %	0.27 %

(a) : "Validation" view



4: 12	88774144	128154840	57919572	293933408	37280852	19226876	35
5: 12	91667776	132691480	55686700	282465824	31744946	11809370	34

☐ Detect from line 1 to line 62

Sensitivity 0.8

🟦 Regular signals
🟥 Abnormal signals

```
{"signal": 84, "line": 24, "similarity": 0, "status": "anomaly"},
{"signal": 85, "line": 25, "similarity": 0, "status": "anomaly"},
{"signal": 86, "line": 26, "similarity": 0, "status": "anomaly"},
{"signal": 87, "line": 27, "similarity": 0, "status": "anomaly"},
{"signal": 88, "line": 28, "similarity": 0, "status": "anomaly"},
{"signal": 89, "line": 29, "similarity": 0, "status": "anomaly"},
{"signal": 90, "line": 30, "similarity": 0, "status": "anomaly"}
],
"detection_results": {"signals": 30, "nominal": 0, "anomaly": 30},
"learning_summary": {"signals": 62},
"detection_summary": {"signals": 90, "nominal": 27, "anomaly": 63}
}
```

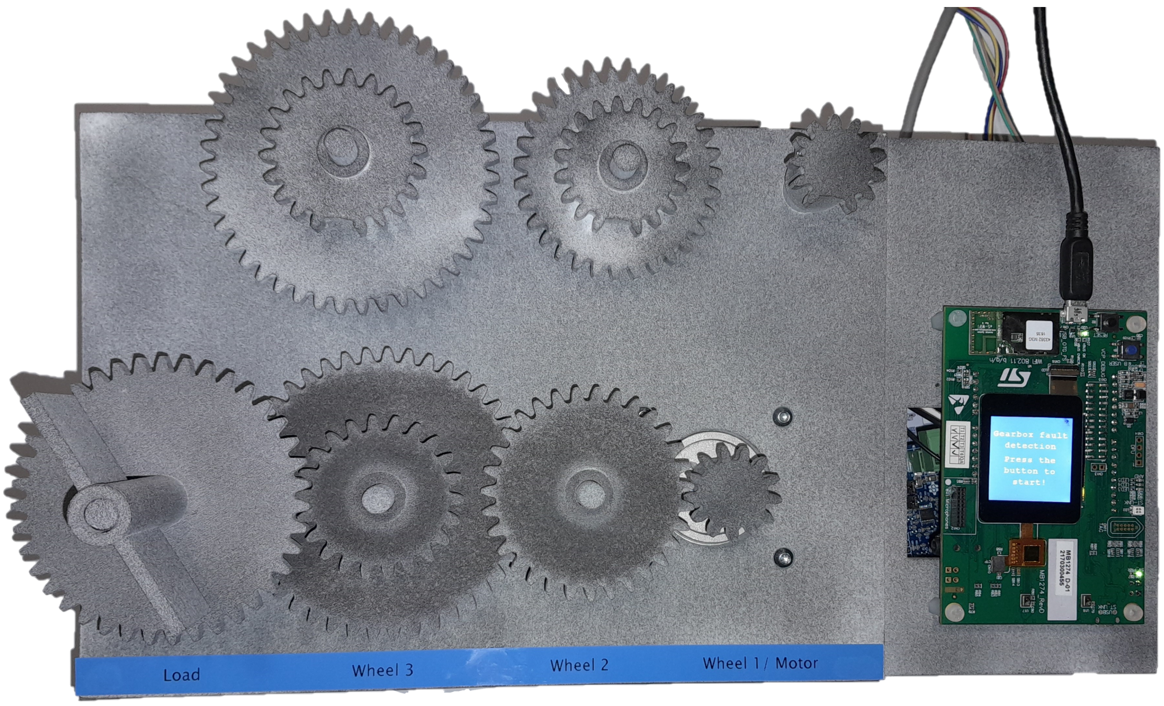
(b) : "Emulator" view

Figure 3.10: Model performance evaluation in NanoEdge AI Studio

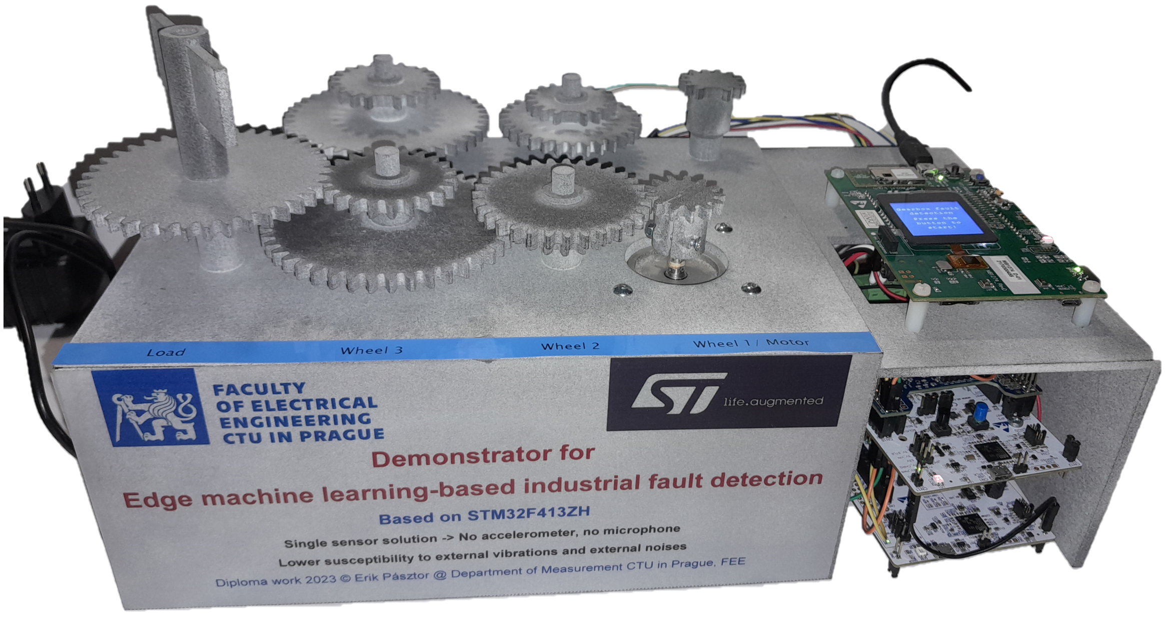
3.7 Final Demonstration System

In this section, the proposed demonstration system will be described, together with its implemented software and communication. Figure 3.11 shows the components of this system - the gearbox and microcontrollers. This only serves as a demonstration; no testing was done on this unit. But most of its components and parameters are the same as in the first gearbox.

Compared to the original development gearbox, one less wheel is used as successful detection of faults on the fourth wheel was not possible during tests. The wheels are also placed in a line rather than optimising space. The structure contains additional shafts that can hold damaged wheels and was printed using the same process and material as explained in Section 3.2.3. The thickness of the walls is also the same. The gears were used from the original gearbox, and the numbers of their teeth are summarised in Table 3.3 (apart from the unused fourth wheel).



(a) : Top view


Load
Wheel 3
Wheel 2
Wheel 1 / Motor
FACULTY OF ELECTRICAL ENGINEERING CTU IN PRAGUE
life.augmented
Demonstrator for
Edge machine learning-based industrial fault detection
Based on STM32F413ZH
Single sensor solution -> No accelerometer, no microphone
Lower susceptibility to external vibrations and external noises
Diploma work 2023 © Erik Pásztor @ Department of Measurement CTU in Prague, FEE

(b) : Front view

Figure 3.11: Demonstration gearbox

An extension is added to the side of the main box that supports the F4 MCU. The other two MCUs are laid below this, and all the connecting cables (between the MCUs and the motor) are routed inside this compartment. Only one USB cable is necessary to supply power to the F4 board as the remaining MCUs are supplied from the F4's 5 Volt output. Apart from this cable, a power supply for the motor and an Ethernet cable are connected from the back of the unit. Figure 3.12 shows a diagram of the connections made between the pins of the MCUs and the motor. The motor's supply voltage is not shown here, as it is provided by the extension board.

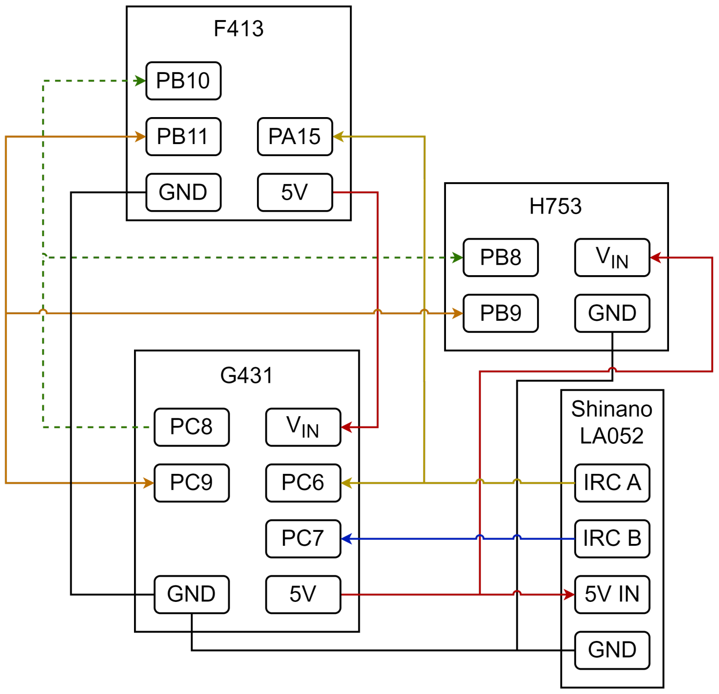


Figure 3.12: Simplified diagram of connections between the MCUs and the motor

3.7.1 Software and Display

The F4 MCU is programmed with a 3-class classification model, which can distinguish a regular gearbox state and large faults on the first and second wheels. The performance of this model is described in Section 5.3.1.

In addition to the detection system, a simple graphical application runs on the MCU. The user is asked to start data collection (by pressing the button). After that, the F4 MCU sends a command to the G4 to start the motor. The motor is stopped, the whole signal has been acquired, and inference starts automatically. The motor spins at 850 rpm, and 40000 edges of the IRC signal are collected the same way as in the original system.

After inference is completed, a message is displayed based on the result. The application can signal a healthy state, faults on the first or second wheel, and an unknown state. The last one should only be obtained when the model cannot, with enough confidence, classify the signal. This should happen when a fault is used that was not part of the training set, for example, with a damaged third wheel or if there is some other mechanical dissimilarity or issue. The possible messages are shown in Figure 3.13.

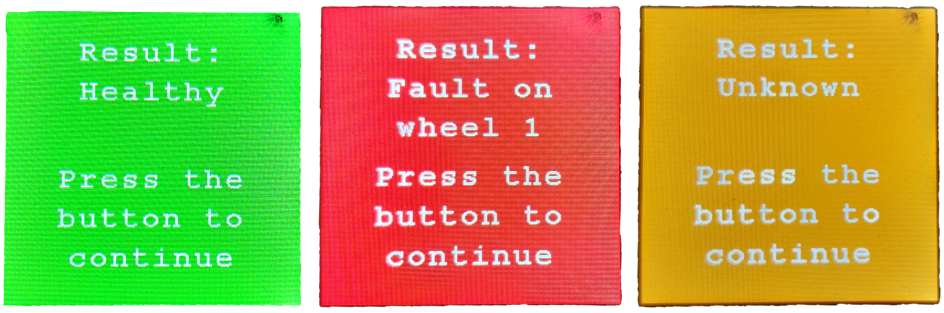


Figure 3.13: Inference result in the graphical application

3.7.2 Ethernet Communication

As an example of how it is possible to store and visualise the collected data, an Internet of Things (IoT) analytical platform called Thingspeak (`https://thingspeak.com/`), which is available as part of the Matlab licence, was employed for this purpose.

After inference, the gearbox state is transmitted to the H7 MCU via I2C. This then forwards the data via its Ethernet interface to a Thingspeak server. Hypertext Transfer Protocol (HTTP) is used to encode the necessary parameters of the target channel and the data. Figure 3.14 shows how the data can be visualised on a chart.

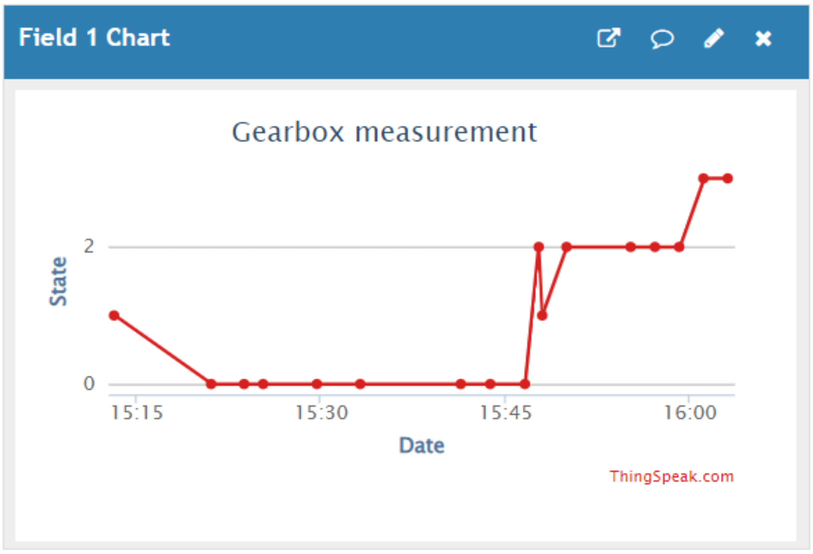


Figure 3.14: Thingspeak chart of data

Of course, other data could be sent to such an IoT service apart from the final gearbox state- individual computed features, whether the motor is currently running, etc.

Chapter 4

Analysis in Matlab and Selected Features

In this chapter, the results of the analysis of the collected signals will be shown. The goal was to confirm the validity of the theory from Chapter 2 - whether a gearbox fault can be distinguished from a correct state based on the proposed features. It also serves to display examples of real signals collected through this work and to illustrate the differences between classes of the obtained data. This was done using the Matlab software, mainly to avoid the need to implement all necessary tools on an MCU - this both simplifies the process and provides significant time savings. Those frequencies that showed the best distinction between classes were also identified and then used to train ML models in the NanoEdge AI Studio.

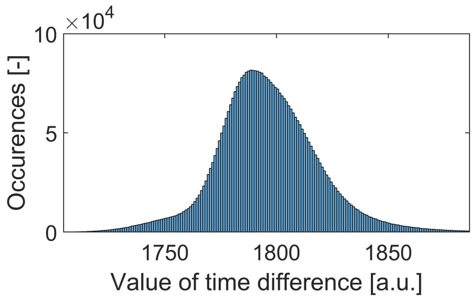
The signals were acquired with the timer parameters described in Section 3.4 and processing was performed using the various constants specified in Section 3.5. Of the two types of tooth damage, only gears with missing teeth were used to record signals for training. Later, during testing, both versions of damaged cogs were used.

4.1 Signal Processing

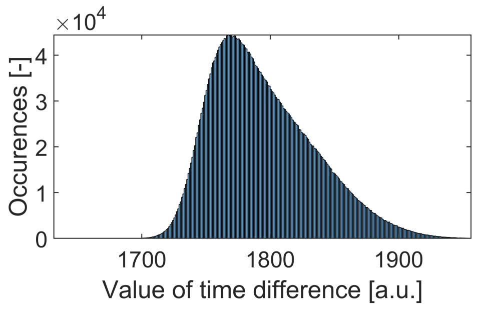
In the following sections, examples will be listed to show the individual steps of the processing pipeline. Based on the analysis of several testing speeds, ranging from 600 rpm to 2000 rpm, it was concluded (mainly from the tests in Section 4.1.2) that the best results should be possible with a motor speed of 850 rpm. The signals and features shown here were generated from data recorded at this speed. Due to the poor results in detecting faults on the last wheel, will not be shown signals and features recorded with these faults.

4.1.1 Normalised Signal

The time difference signal, which is the basis of further calculations, was previously depicted in Figure 3.8a. Figure 4.1 then shows the statistical distribution of time differences in the form of histograms, calculated over 100 signals from two classes. It can be seen that they are nearly normally distributed. This fact is beneficial for some of the employed methods, as they make this assumption to make the computations more effective and accurate. Moreover, it confirms a necessary prerequisite - the data are not random, so further analysis is possible. There is a notable difference between the standard deviations of the two data sets, where the class with a fault has almost twice the value of the fault-free state.



(a) : Without faults. $\mu = 1795.6$, $\sigma = 22.7$.



(b) : Fault on the first wheel. $\mu = 1794$, $\sigma = 40.6$.

Figure 4.1: Histograms of time differences with means and standard deviations.

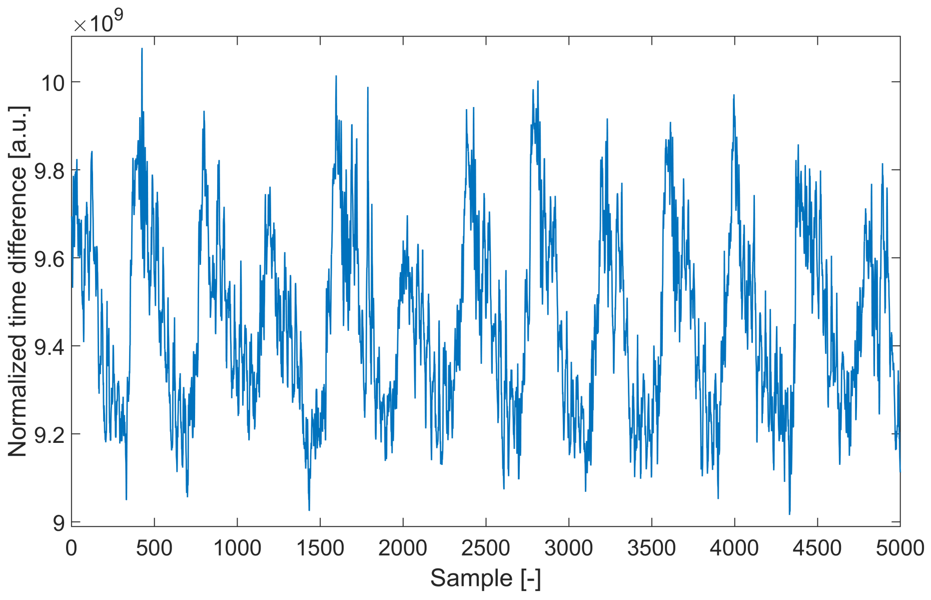
From the original values, normalised signals were computed. Examples of these can be seen in Figure 4.2. There are certain differences visible, mainly the more pronounced higher frequencies in the faulty signal.

4.1.2 Frequency-domain Features

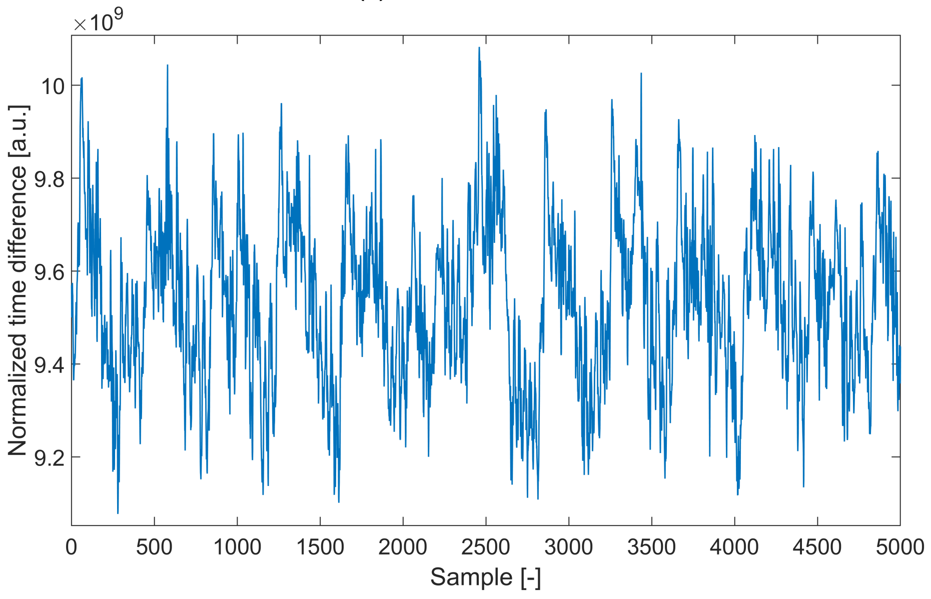
Figure 4.3 shows the spectra corresponding to the signals in Figure 4.2. They are limited to just above the motor frequency. This is because the main interesting events are those that occur at each rotation of the motor (in case of a fault on the first wheel) or less often (in case of other faults), so these low frequencies should be focused on. The farther the fault is from the sensor, the lower the frequency it generates. The higher frequency components of the signals are considered noise and are not taken into account for the analysis.

Clear dissimilarities can be seen between the two spectra. In a case where a fault is present, the peak at the motor frequency has a lower amplitude, and other peaks appear - namely, at orders 0.42, 0.69 and 0.84.

Figure 4.4 then provides a view of the spectra of all collected signals. Similarly to the comparison above, distinctions can be seen between the spectra of individual classes. Apart from a peak at the motor frequency, which is present in every signal, different combinations of peak locations arise. Differences between some signals of the same class are probably caused by changes in the gearbox's physical condition, the driven gear being tightened more or less, and a slightly shifted placement of the box . . . . This contributes to the so-called interclass variation, meaning that the individual data vectors



(a) : Without faults

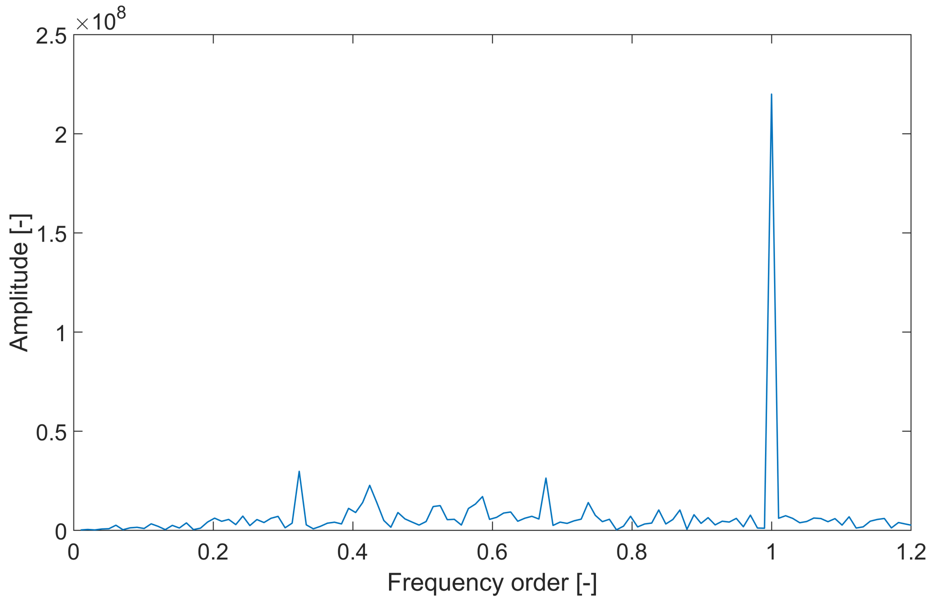


(b) : Fault on the first wheel

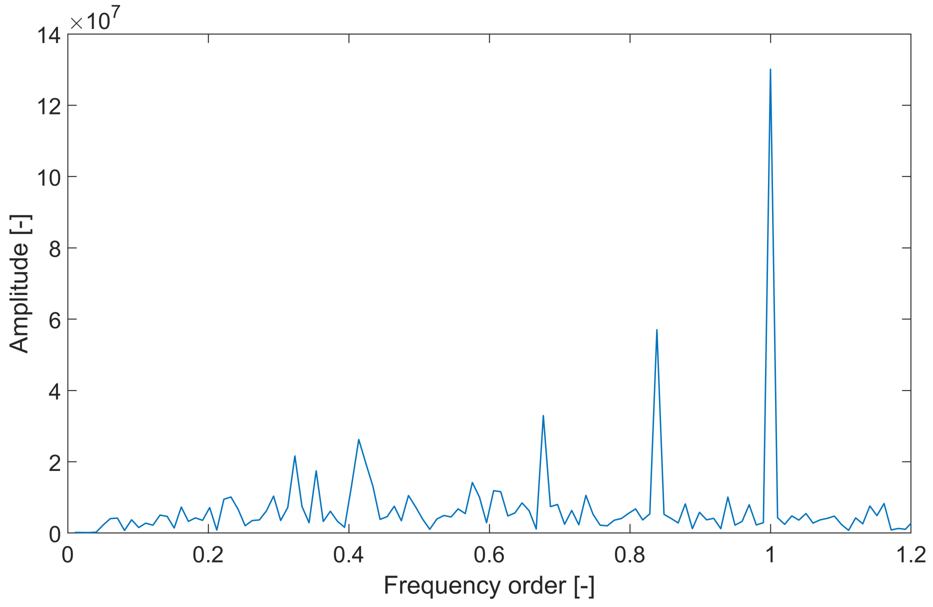
Figure 4.2: Examples of normalised signals

are noticeably different but belong to the same class. Commonly, having data that cover as many states of each class as possible can improve a model's quality because it is forced to generalise and learn well enough to deal with this.

It should be noted that, theoretically, there should be meaningful changes in the frequencies corresponding to the gear ratios listed in Table 3.3. But at the same time, these frequencies will also be present in signals without faults. This is caused by mechanical impacts of the gear teeth. So in Figure 4.4, a



(a) : Without faults



(b) : Fault on the first wheel

Figure 4.3: Examples of normalised signal spectra

notable change in frequency order 1 should be seen for a fault on the first wheel, 0.42 on the second wheel, etc. On order 1, there is an overwhelming amplitude caused by the motor regulator, but it is influenced by the fault. Similarly, order 0.42 has comparable amplitudes for signals without faults and with a fault on the first and second wheels. 0.19, 0.11 and 0.05 do not have a noteworthy presence. This is most likely caused by the mechanical distance of the corresponding faults to the sensor and the dampening and noise of the signals. This means that it would presumably not be possible to

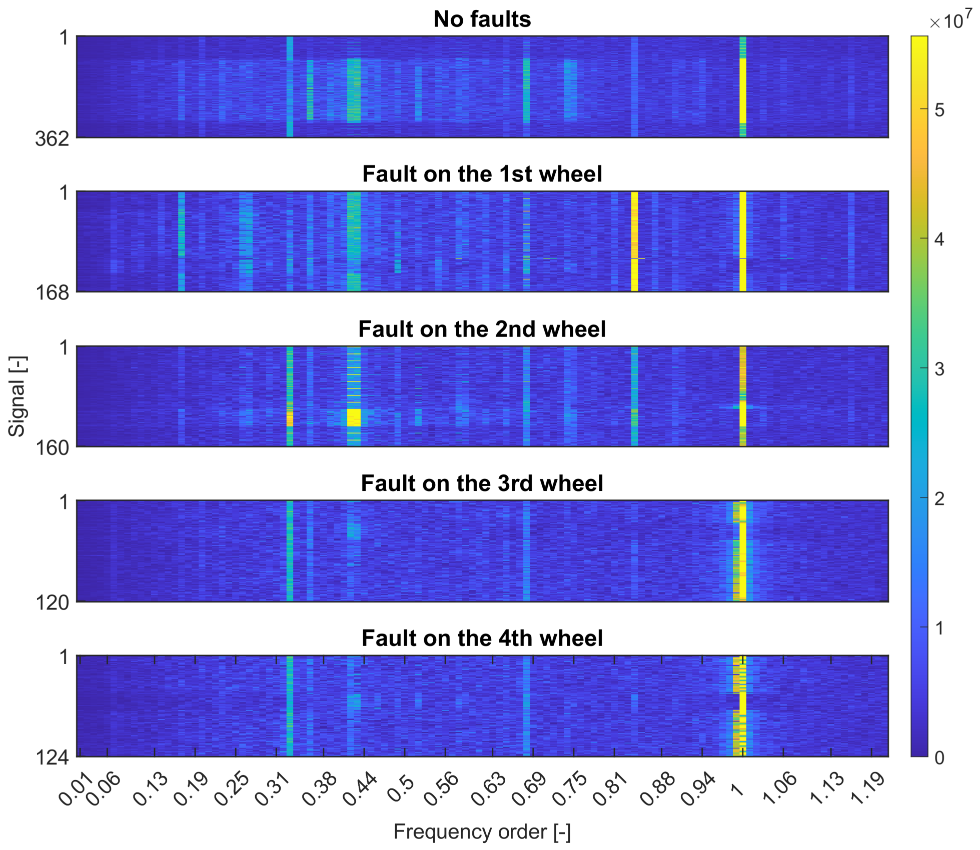


Figure 4.4: Spectra of the collected signals. Amplitudes clipped to one-fifth of the maximum value to improve readability.

create a successful model simply by relying on these basic frequencies.

Interestingly, the frequency order 0.84, which is the second multiple of the second wheel's gear ratio, appears distinctively for faults on the first and second wheels. This may be derived from certain mechanical properties that were not explored.

The histograms of the values at four frequencies are shown in Figure 4.5. This visually reinforces the hypothesis that the classes are separable on the basis of these frequency features.

4.1.3 Time-domain Features

Figure 4.6 shows a normalised signal and its averaged version. The effect of averaging is clearly visible, as the peaks are less pronounced, and the individual periods (nine in total) have shapes more resembling each other. In this part, the frequency parameters that would be influenced by fluctuations in the signal's periods, like different peak locations, are not important. With time-domain features, the focus is more on the general shape of the signal.

Similarly to frequency characteristics, the histograms of the characteristics

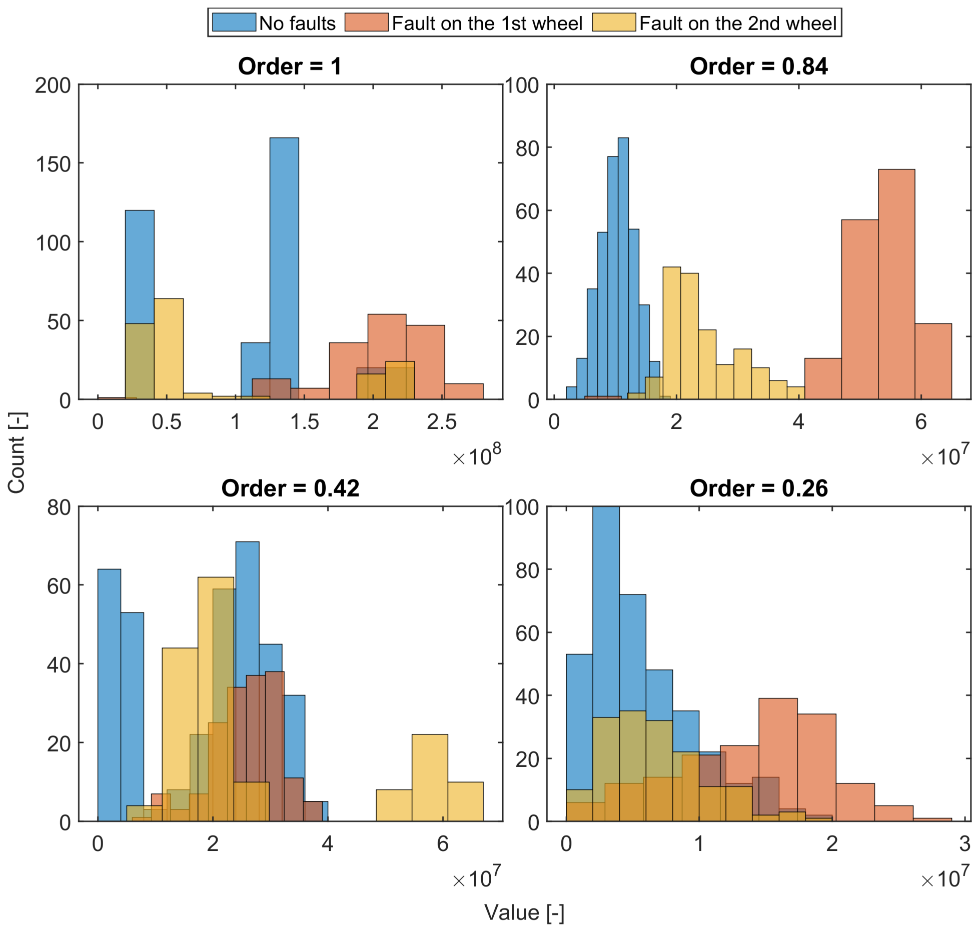


Figure 4.5: Histograms of amplitudes at several chosen frequencies

computed in Figure 4.7 show certain distinctions between the data classes.

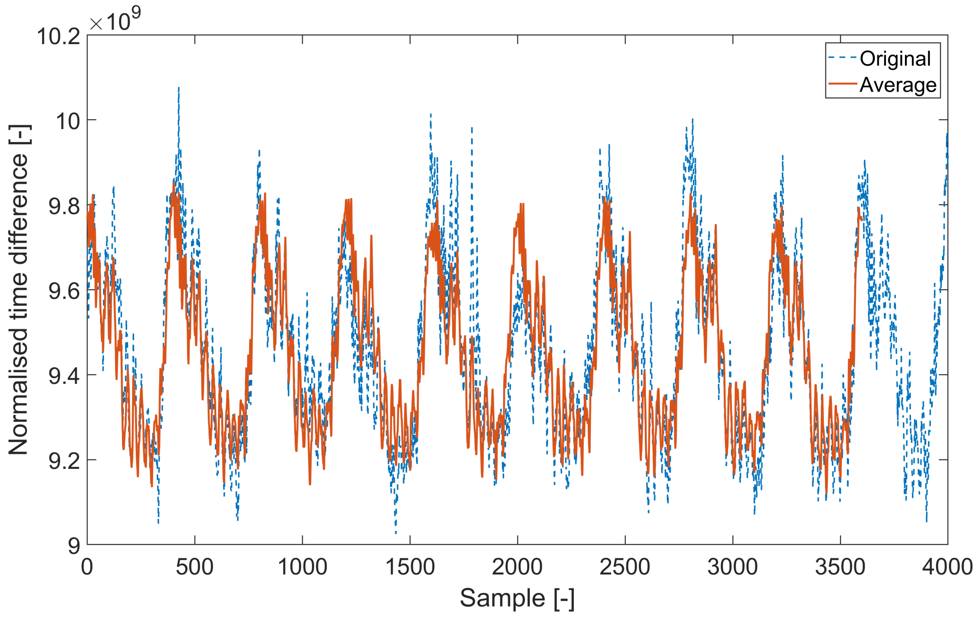


Figure 4.6: Example of an averaged signal

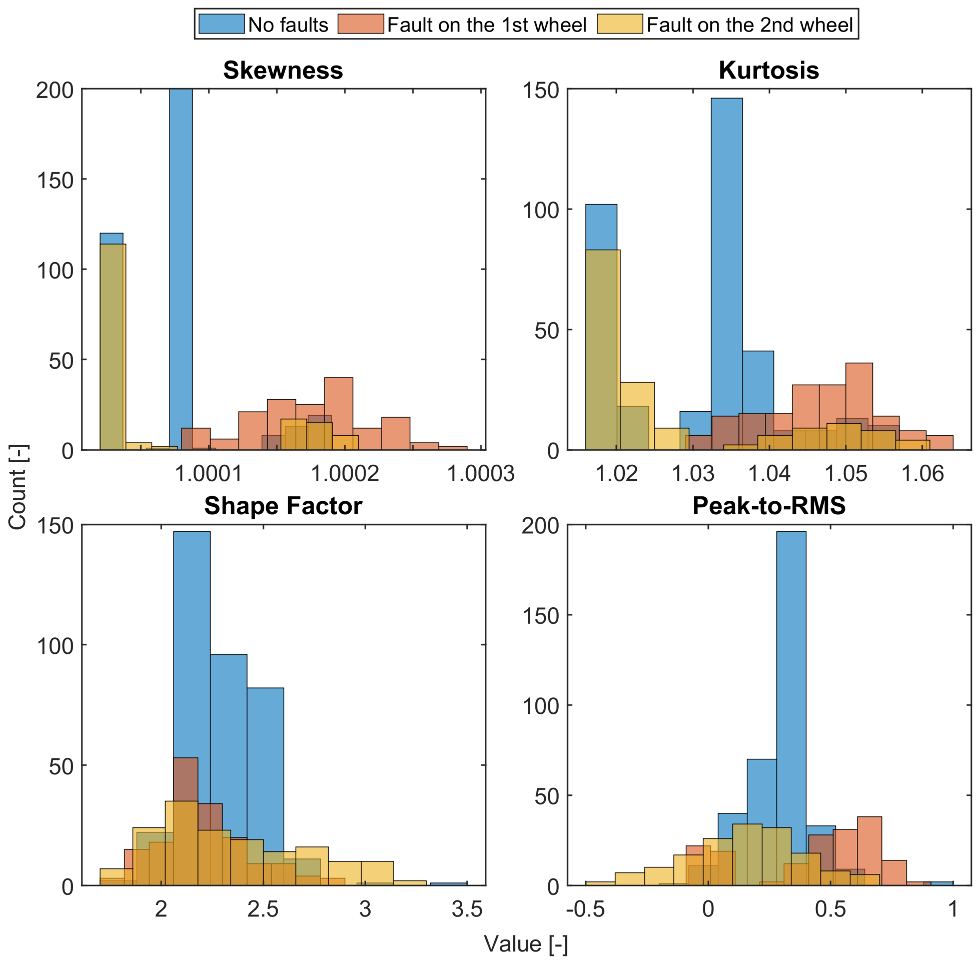
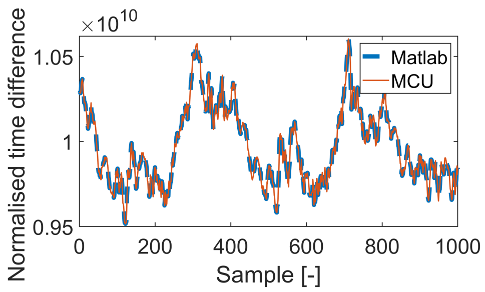


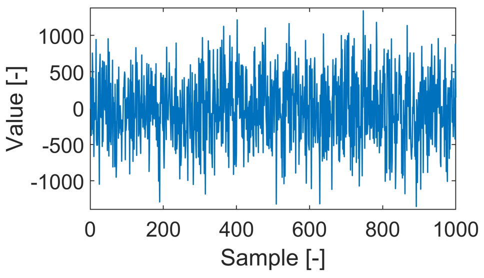
Figure 4.7: Histograms of values of the time-domain features

4.1.4 Comparison to MCU Processing

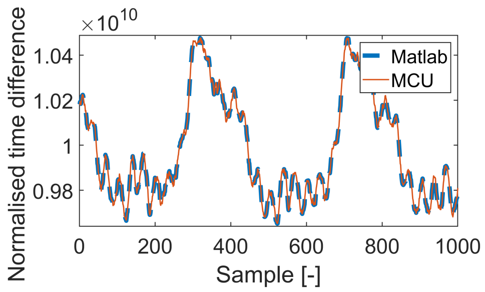
Figure 4.8 illustrates how the processing results on the MCU deviate from the computations executed in Matlab, which is considered to be far more precise. The two methods differ significantly in their computing capabilities. Whereas Matlab uses 64-bit representation, the FPU in the MCU core works only with 32-bit numbers. This reduces accuracy as the shorter format has a lower resolution. It is one of the reasons why multiplication by a large constant was employed in the proposed pipeline to avoid situations where changes in values would be under the MCU's resolution level. At the same time, Matlab uses proprietary algorithms, which probably include general enhancements and are more precise than the simple computations implemented here.



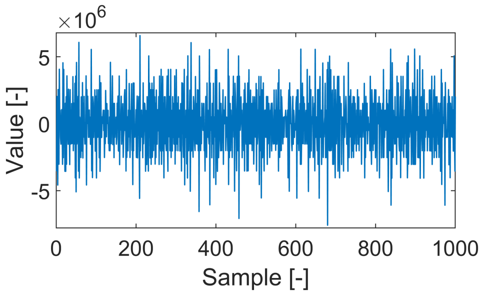
(a) : Normalised signals



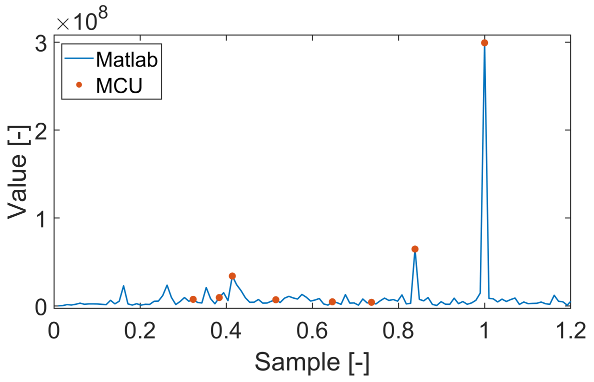
(b) : Differences between normalised signals



(c) : Averaged signals



(d) : Differences between averaged signals



(e) : Frequency components

Figure 4.8: Processing in Matlab and MCU comparison

Figure 4.8b shows the difference in normalised signal computation. The peak values of around 10^3 mean that the maximum error in the MCU implementation is close to $\frac{10^3}{10^{10}} = 10^{-7} = 0.1ppm$ (where 10^{10} is the approximate mean value of the normalised signal). In Figure 4.8d, the deviation of the averaged signal reaches $5 \cdot 10^6$. This would be translated as $\frac{10^6}{10^{10}} = 10^{-4} = 100ppm$ error. The error of several computed frequency values, shown in Figure 4.8e, reaches a maximum of 1270.1, while its mean value is -401.4. In terms of relative error, the maximum is at $101ppm$, and the mean is at $45ppm$. For time-domain characteristics, the average error computed over several signals was equal to $115ppm$, with the error being generally lowest for the peak-to-RMS value and highest for kurtosis - which is expected because kurtosis calculation involves raising values to the power of four.

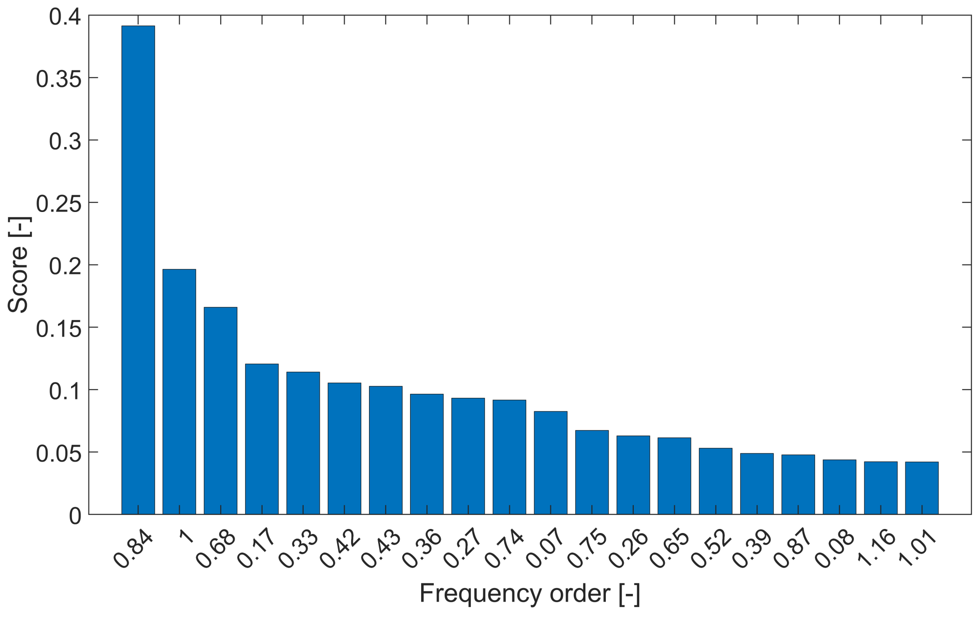
These values demonstrate (even when computed from one signal each) the numerical accuracy of the MCU implementation approaches that which is achieved on a computer with larger resources and more advanced algorithms.

4.2 Frequency Feature Selection

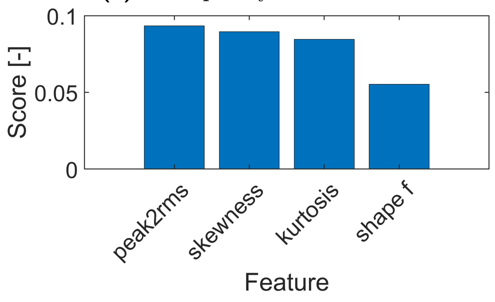
To determine which features are suitable for use in classification, their ranking was first calculated using the RELIEFF algorithm. RELIEFF was introduced in article [25]. It works by randomly selecting a vector of features with its corresponding class label, finding K nearest neighbours (in the sense of vector distance) from each class and updating the weights for all these neighbours. The update procedure is designed so that it penalises the features that have different values to neighbours of the same class and rewards those that give different values to neighbours of different classes. It is an iterative algorithm where the number of iterations should be fixed, but Matlab does not provide a way to set this value. $K = 10$ was used in the analysis. Although this algorithm is used primarily to work with classification models, the selected features were used with anomaly detection.

20 frequency features with the highest RELIEFF score can be seen in Figure 4.9a. For reference, sorted signal characteristics are also shown in Figure 4.9b. From the frequencies, ten orders were chosen (0.84, 1, 0.68, 0.33, 0.42, 0.36, 0.27, 0.74, 0.65, 0.52 and 0.87) based on this ordering and a visual inspection of the spectra in Figure 4.4 - for example, 0.17 was viewed as redundant because it was only significantly visible with the fault on the first gear wheel, which is better distinguished by frequency 0.84. A few other factors were also considered - not picking frequencies too close to each other (0.42 and 0.43), too low or high frequencies (0.07).

Eight frequencies were picked (while keeping all of the four time-domain characteristics) to keep the complexity low and avoid overfitting the model. This was done by training an SVM model for each set of parameters: a combination of eight frequencies together with four fixed characteristics. The used data set used for both of these analyses is summarised in Table 4.1.



(a) : Frequency coefficients



(b) : Time-domain features

Figure 4.9: Features ordered by RELIEFF algorithm

When comparing their results on the testing data set, one of the models with the best accuracy was trained with frequencies 0.84, 1, 0.33, 0.42, 0.36, 0.74, 0.65, 0.52. Confusion matrices for this model are shown in Table 4.2. It is visible that states without faults and faults and the first and second wheels can be classified correctly most of the time. But for faults on the third and fourth wheel, the results are worse, even on training data which should be classified correctly because it was seen in training.

Fault on wheel	-	1	2	3	4
Training signals	162	168	160	30	31
Testing signals	15	15	15	5	5

Table 4.1: Summary of the data set used for feature selection. Fault "-" signifies a state without faults. Only the larger faults were used.

		Predicted class				
		No F	F1	F2	F3	F4
True class	No F	159	0	0	0	3
	F1	0	168	0	0	0
	F2	0	0	160	0	0
	F3	1	0	0	29	0
	F4	10	0	0	0	21

(a) : On training data

		Predicted class				
		No F	F1	F2	F3	F4
True class	No F	15	0	0	0	0
	F1	0	15	0	0	0
	F2	0	0	15	0	0
	F3	1	0	0	4	0
	F4	1	0	0	0	4

(b) : On testing data

Table 4.2: Confusion matrices of the best SVM model. "No F" signifies a signal without faults, "F1" a fault on the first wheel, etc.

Chapter 5

Detection Performance

This chapter covers the achieved results of detecting faults on the designed gearbox using three approaches and trained models from NanoEdge AI Studio. This evaluation was done on the gearbox detailed in Section 3.2.3.

5.1 Collected Data

Table 5.1 shows how many signals were collected for each gearbox state. As explained in Section 4.1, the faults on the last wheel were not used after analysing the signals in Matlab. So this type of signal is not present for evaluation. All signals were recorded with the motor running at 850 rpm.

To have a more balanced data set when creating an anomaly detection model (which only classifies regular and abnormal states), more signals without faults were collected, and all the faulty signals were used as a representation of the abnormal state. For anomaly detection, 62 signals were also left out for later learning on the MCU and used 300 for training in the NanoEdge AI Studio. For 1-class classification, only signals without faults were used. Naturally, all five classes were used to train n-class classification models.

For testing, samples with smaller faults were also collected individually on the first and second wheels and with both faults present at the same time. This should more closely represent a real-life scenario, where a chipped tooth on one wheel would cause damage to a tooth on the neighbouring wheel.

Fault on wheel	-	1	2	3	4	1S	2S	1S+2S
Training signals	362	168	161	121	124	0	0	0
Testing signals	30	30	30	30	30	30	30	30

Table 5.1: Summary of the collected data set. Fault "-" signifies a state without faults, "1S" a smaller fault on the first wheel, "1S+2S" a small fault on both wheels.

5.2 Evaluation Approach and Metrics

After training in NanoEdge AI Studio, models were selected based on their training and validation performance, both of which can be seen, for example, in Figure 3.10a. Further exploration was done only for the models that had achieved the highest scores. These were then evaluated both using the emulator in NanoEdge AI Studio and on the MCU.

In the following sections, metrics used to describe anomaly detection and 1-class classification models are true positive count (TP), which is the number of signals correctly identified as being without faults, and false positive count (FP), which is the number of faulty signals classified as faultless. True negative count (TN) represents those faulty signals that were correctly identified, and false negative count (FN) those without a fault classified as faulty. From these, the true positive rate (TPR, also called sensitivity) and the true negative rate (FNR, also called specificity) are calculated as $TPR = \frac{TP}{TP+FN}$ and $TNR = \frac{TN}{TN+FP}$. Confusion matrices are employed to better visualise these numeric results.

To find the best-performing model of each type, the emulator was used first with features computed on the MCU. For anomaly detection models, a suitable value for sensitivity was also estimated. Then this model was exported and used for evaluation on the MCU - previously collected testing data were transferred back to the MCU from a computer using USART communication.

During this evaluation, the time requirements were observed by measuring the time on a computer starting after the transfer had finished and ending at the reception of the MCU's output. Average execution times, which include both signal processing and class inference, are listed for each project in the corresponding sections below. Processing time alone was measured in the same way, except that the MCU was programmed to perform only the processing part of the pipeline. The MCU was set to run at its highest possible clock frequency, 100 MHz.

All models were trained on features extracted from signals with larger faults. Smaller faults were used to test anomaly detection and 1-class classification.

5.3 Evaluation Results

Table 5.2 shows the measured processing time. This test was carried out on 50 signals.

Average time [ms]	Standard deviation [ms]
8277.2	7.01

Table 5.2: Processing time measurement

5.3.1 N-class Classification

For classification, the focus was on larger faults. This was done because there was a lack of a substantial training data set containing signals with smaller faults, which would be necessary for their classification.

Table 5.3 shows the results obtained when an n-class classification model was trained on all five signal classes. The trained model produced identical confusion matrices when the classification was simulated in the NanoEdge AI Studio emulator and when it was physically implemented on the MCU.

		Predicted				
		No F	F1	F2	F3	F4
	No F	16	0	0	2	12
	F1	0	30	0	0	0
True	F2	0	30	0	0	0
	F3	10	0	1	12	7
	F4	7	0	3	14	6

(a) : In emulator

		Predicted				
		No F	F1	F2	F3	F4
	No F	16	0	0	2	12
	F1	0	30	0	0	0
True	F2	0	30	0	0	0
	F3	10	0	1	12	7
	F4	7	0	3	14	6

(b) : On the MCU

Table 5.3: Confusion matrices for 5-class classification. "No F" signifies a signal without faults, "F1" a fault on the first wheel, etc. Matrices a) and b) are identical.

The accuracy of this model is quite low, especially compared to what can be achieved with the same data on a computer (as seen in Table 4.2). Most obviously, many errors are tied to faults on the more distant gear wheels.

To try and get around this issue, a similar model with only three classes was trained: faultless and a fault on the first and second wheels. Confusion matrices for this model are shown in Table 5.4. Again, the classification in the emulator matched the real results. But in this case, a perfect score was achieved, which shows that the differences between the undamaged gearbox state and damage on the third or fourth wheel are too small for the 5-class model to properly distinguish.

		Predicted		
		No F	F1	F2
	No F	30	0	0
True	F1	0	30	0
	F2	0	0	30

(a) : In emulator

		Predicted		
		No F	F1	F2
	No F	30	0	0
True	F1	0	30	0
	F2	0	0	30

(b) : On the MCU

Table 5.4: Confusion matrices for 3-class classification. "No F" signifies a signal without faults, "F1" is a fault on the first wheel, etc. Matrices a) and b) are identical.

Table 5.5 then lists the classification time and memory required by the model. The overall time needed is only approximately 20 ms longer than the processing duration seen in Table 5.2.

Average time [ms]	Standard deviation of time [ms]	Model type
8294.78	8.36	RF
	RAM [kB]	Flash [kB]
3-class	0.1	25.9
5-class	0.1	69.3

Table 5.5: N-class classification time and memory requirements and metrics. Time measured on the MCU.

It should also be noted that a model with three classes was tested (from the selection trained by NanoEdge AI Studio) which was ranked higher than the one presented here, based on the validation score during testing and the memory size. But in the evaluation, it classified signals with a fault on the second wheel as belonging to the class with a damaged second wheel. Therefore, it was not able to make a distinction between these two states. This underscores the importance of testing the models on clean data. An interesting difference between the models was that the worse-performing one needed significantly less flash memory space - 3.5 kB compared to 25.9 kB of the chosen model.

5.3.2 Anomaly Detection

The anomaly detection model was trained on a regular class with faultless signals and an abnormal class containing signals with four large faults. Then it was tested on these testing data with the smaller faults in addition to large faults and faultless signals. When evaluating it, the classification results were obtained as summarised in Table 5.6.

		Predicted	
		Regular	Abnormal
True	No F	27	3
	F1	0	30
	F2	0	30
	F3	30	0
	F4	30	0
	F1S	17	13
	F2S	8	22
	F1S+F2S	5	25

(a) : In emulator

		Predicted	
		Regular	Abnormal
True	No F	30	0
	F1	0	30
	F2	0	30
	F3	30	0
	F4	30	0
	F1S	28	2
	F2S	6	24
	F1S+F2S	5	25

(b) : On the MCU

Table 5.6: Confusion matrices for anomaly detection. "No F" signifies a signal without faults, "F1" a fault on the first wheel "F1S" a smaller fault, etc.

When considering large faults, the performance is similar to n-class classification with five classes - faults on the third and fourth wheels cannot be detected. But the remaining three states are recognised correctly in all cases. Small faults on the second wheel (and with a combination of the first and second wheels) are classified accurately in most instances. But the model failed to identify small faults on the first wheel.

The differences between the classifications using the emulator and the MCU can be in part due to the fact that the emulator has fixed values for model sensitivity at 1 and threshold at 90 %. For the evaluation on the MCU, a sensitivity of 0.8 and a threshold equal to 80 % were used.

Table 5.7 shows the model details and the summary of the evaluation. The measured inference time is 50 ms (after subtracting signal processing time). Faults on the third and fourth wheels are excluded from the metrics (TP, etc.).

Average time [ms]	Standard deviation of time [ms]
8327.5	5.63

Model type	RAM [B]	Flash [kB]
Z-score model	48	1.9

TP	FN	TPR
30	0	1
TN	FP	TNR
111	39	0.74

Table 5.7: Anomaly detection time and memory requirements and metrics. Measured on the MCU.

5.3.3 1-class Classification

First, the 1-class classification model was trained with an abnormal class made up of all four large-fault signals. With this setting, the models found by NanoEdge AI Studio were not accurate enough (for example, when compared to the anomaly detection model), particularly for faults on the third and fourth wheels. Because of this, the model whose results are shown in Table 5.8 was trained only with faultless signals and large faults on the first two wheels.

The results achieved were then comparable to the anomaly detection model. Table 5.9 shows that the memory and time requirements of this model are lower than for anomaly detection. Namely, the inference time, which is only around 1 ms.

		Predicted	
		Regular	Abnormal
True	No F	16	14
	F1	0	30
	F2	0	30
	F3	25	5
	F4	24	6
	F1S	4	26
	F2S	0	30
	F1S+F2S	0	30

(a) : In emulator

		Predicted	
		Regular	Abnormal
True	No F	25	5
	F1	5	25
	F2	0	30
	F3	30	0
	F4	28	2
	F1S	5	25
	F2S	8	22
	F1S+F2S	4	26

(b) : On the MCU

Table 5.8: Confusion matrices for 1-class classification. "No F" signifies a signal without faults, "F1" a fault on the first wheel "F1S" a smaller fault, etc.

Average time [ms]	Standard deviation of time [ms]
8278.37	3.27

Model type	RAM [kB]	Flash [kB]
FISVDD	0.2	0.5

TP	FN	TPR
25	5	0.83
TN	FP	TNR
128	22	0.85

Table 5.9: 1-class classification time and memory requirements and metrics. Measured on the MCU.

5.4 Performance Summary

From the evaluation results, it is apparent that there is a limit to the gearing ratio at which the system recognises faults. For this particular setup, this limit is after the second gear wheel. So the maximum gearing ratio where faults could reliably be detected was approximately 1/2.4. In general, this limit would probably be influenced by factors such as the sensor resolution used, mechanical load, or physical attributes of the gearbox (like material or size of teeth) which would determine how well the fault signal travels through the system.

From a data perspective, this could be interpreted in a way that the models are not complex enough to learn the shapes of decision boundaries between individual classes. With fewer classes, the patterns are simpler. But this same trend could also be seen in the test on a computer with more advanced models in Table 4.2, although the results there were slightly better.

When classifying multiple classes, the model was able to achieve a perfect score if only three classes were used. In this scenario, the result is comparable to the computer test in Table 4.2. This shows that equal accuracy can be reached even with simple hardware when the extent of classified states is limited. Adding more classes led to worse results, even in the classification of these three states. A large number (almost 50 %) of faultless signals were classified as faulty, which would not be acceptable in practice.

When excluding those more difficult faults, the anomaly detection model achieved scores of $TPR = 1$ and $TNR = 0.74$, which means that when the model classifies a signal as faulty, this is certainly consistent with the real state. And when a signal is classified as faultless, there is a chance that it is faulty, and there should be further investigation. This includes smaller faults, which are presumably harder to detect.

Some of the good results when detecting smaller faults may be caused by different physical conditions of the setup. As these signals were collected at a later time than the original data, the gearbox might have been placed differently, the motor gear might have been less tightened, etc. This probably resulted in changed signal properties, namely noise at frequencies significantly higher than those that were computed when extracting features. Because of this, the frequency feature should not be affected, but the time-domain features might have slightly changed values. But, as already explained in Section 4.1.2, this contributes to the overall generalisation possible by the model.

The 1-class classification model achieved slightly poorer accuracy with $TPR = 0.83$ and $TNR = 0.85$. At the same time, it is limited in its ability to be fine-tuned when compared to anomaly detection. Although 1-class classification does not require abnormal signal samples for training, anomaly detection is probably applicable in more general cases. A fact that can compensate for this is that it achieved a much better result when classifying signals with small faults on the first wheel - this caused its TNR score to be higher than for the anomaly detection model.

To improve performance, a buffer could be established that collects several inference results. Then the final results would be determined as either an average value or the state with the most occurrences. Another way is to run detection using different models and evaluate their outputs.

In all cases, the measured execution time is mainly made up of the signal processing time. The inference time was, at most, approximately 50 ms (for anomaly detection). This is insignificant in most practical cases compared to more than 8 s measured for processing.

In terms of memory requirements, the necessary Flash space was highest for the n-class classification, at almost 26 kB. Anomaly detection and 1-class classification took up 1.9 and 0.5 kB, respectively. 0.2 kB of RAM was needed for 1-class classification and 0.1 kB for the other two projects. This mostly came from the buffer which needs to be allocated for the computed features - ten 32-bit floats take up 40 bytes. With 1.5 MB of Flash and 320 kB of RAM available on the used MCU, this should not be a limitation in most applications.

Overall, the anomaly detection approach seems to be the most promising, if the user is only interested in detecting a fault, in general. It also has the advantage (over classification) that it does not need to be trained on a specific kind of fault. Faults that manifest slightly differently than training samples can also be detected, as proven by the fact that smaller gear damages were detected. If the aim is to have more information about the fault, then n-class classification can be used, perhaps in conjunction with anomaly detection.

Chapter 6

Conclusion and Future Work

6.1 Discussion and Conclusion

The objective of this work was to design and build a demonstrator for a machine-learning system on an edge device that would detect mechanical faults (specifically damaged or missing teeth) in industrial machinery with gears.

This was achieved after selecting an STM32F413ZH MCU as the primary device on which the detection software would be implemented and creating ML models in NanoEdge AI Studio. This also accomplishes the second goal of the work of evaluating the performance of the NanoEdge AI Studio tool. The premise that it can work well even in more complex cases was validated. The tool itself can produce code for signal processing and feature extraction, but this component was not relevant for raw IRC signals, as it is limited by the input length.

A gearbox was designed and 3D printed with three different sets of wheels: without faults, with small damage to one tooth and a missing tooth. After collecting data and analysing them in Matlab software, a pipeline was proposed to process the IRC signal and extract features from it, which was then programmed into the MCU.

The requirements led us to choose a suitable BLDC motor with a built-in IRC, an MCU with supplementary hardware which would control the motor, an MCU to facilitate Ethernet communication.

After testing the models and confirming that such a system is plausible, a second gearbox was designed with fewer wheels and an additional section to attach the MCUs. This, combined with the original motor and MCUs, can work as a complete, standalone demonstration system.

When integrating into existing systems, just one MCU can be added that will function as the detection unit. For a new system, the separate functions (detection, motor control, and Ethernet communication) could be implemented in a single, more powerful MCU, for example, one of the STMicroelectronics'

H7 line.

Compared to some other available research prototypes or even devices that have been implemented in practice, the advantage of this design is that it is a single-sensor solution that works with a sensor that is, already a part of many motor control projects. The chosen type of sensor leads to a lower susceptibility to external vibrations and noise as no accelerometers or microphones are used.

Further requirements for this demonstration unit were that it should work independently of other devices and so would need to collect and process the data by itself, the only signal source would be an IRC sensor, and that it should enable Ethernet communication with a central device to conform to industry standards. Using Ethernet, communication with an IoT data-collection platform was established.

From a timing point of view, which is also important for practical applications, the measured inference times were less than 50 ms. The processing pipeline took approximately 8 s to execute. The primary limiting factor is the time it takes to collect a signal of sufficient length. This could lead to a system with periodical inferences, for example, every second, with the motor still running.

Different ML models were trained in the NanoEdge AI Studio. They were evaluated both in the available emulator tool and on the MCU. The results of the tests show that, for this particular scenario, NanoEdge AI Studio can produce models with satisfactory results. When detecting faults (both large and small without distinction) on the first two wheels, scores of $TPR = 1$ and $TNR = 0.74$ were achieved with an anomaly detection model and $TPR = 0.83$ and $TNR = 0.85$ for 1-class classification. A classification model with three classes achieved a perfect score when focusing only on large faults. As expected, when detecting smaller faults, the models did not perform with such good results.

6.2 Future Work

The most beneficial area for future work is probably collecting more data from various mechanical conditions, which means different tightness of the motor gear attachment, placement of the gearbox on bases made of different materials, higher load on the motor, etc. This would help generalise the trained model.

Experiments with shorter signals could be done to determine the limit at which a model is able to detect faults.

The array of detected faults could be extended to include, for example, bearing issues, loose connection of the motor gear (which results in the gear not rotating with the shaft), and other mechanical problems.

Another interesting line of work would be to try to train an ML model on a computer and transfer it to an MCU with the help of X-CUBE-AI, another tool from STMicroelectronics. This approach would allow us to see and control all parameters of the trained model, which is something NanoEdge AI Studio does not offer.

Appendix A

Contents of the attached CD

Data	Used signals collected at 850 rpm and extracted features.
Datasheets	Datasheets of the used MCUs, development kits, and the motor.
F4 Detect	Project and source code for the detection software.
G4 Motor Control	Project and source code for the motor control part.
H7 Ethernet	Project and source code for Ethernet communication.
STL	.stl files for the final demonstrator and individual gear wheels.
thesis.pdf	Full text of the thesis.

Appendix B

Bibliography

[1] A. SG Andrae, "New perspectives on internet electricity use in 2030," *Engineering and Applied Science Letter*, vol. 3, no. 2, pp. 19–31, 2020.

[2] W. Yu, T. Dillon, F. Mostafa, W. Rahayu, and Y. Liu, "A global manufacturing big data ecosystem for fault detection in predictive maintenance," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 1, pp. 183–192, 2019.

[3] S. Kim and J.-H. Choi, "Convolutional neural network for gear fault diagnosis based on signal segmentation approach," *Structural Health Monitoring*, vol. 18, no. 5-6, pp. 1401–1415, 2019.

[4] R. Medina, J. C. Macancela, P. Lucero, D. Cabrera, R.-V. Sánchez, and M. Cerrada, "Gear and bearing fault classification under different load and speed by using Poincaré plot features and SVM," *Journal of Intelligent Manufacturing*, vol. 33, no. 4, pp. 1031–1055, 2022.

[5] M. Soualhi, K. T. Nguyen, A. Soualhi, K. Medjaher, and K. E. Hemsas, "Health monitoring of bearing and gear faults by using a new health indicator extracted from current signals," *Measurement*, vol. 141, pp. 37–51, 2019.

[6] "Adi otosense smart motor sensor." `https://otosense.analog.com/`, visited 2023-05-06.

[7] "Principle and advantages of optical encoder." `https://www.akm.com/eu/en/products/rotation-angle-sensor/tutorial/optical-encoder/`, visited 2023-03-04.

[8] "General-purpose timer cookbook for STM32 microcontrollers." `https://www.st.com/content/ccc/resource/technical/document/application_note/group0/91/01/84/3f/7c/67/41/3f/DM00236305/files/DM00236305.pdf/jcr:content/translations/en.DM00236305.pdf`, visited 2023-03-04.

[9] Y. Li, F. Gu, G. Harris, A. Ball, N. Bennett, and K. Travis, “The measurement of instantaneous angular speed,” *Mechanical Systems and Signal Processing*, vol. 19, no. 4, pp. 786–805, 2005.

[10] A. A. Gubran and J. K. Sinha, “Shaft instantaneous angular speed for blade vibration in rotating machine,” *Mechanical Systems and Signal Processing*, vol. 44, no. 1, pp. 47–59, 2014. Special Issue on Instantaneous Angular Speed (IAS) Processing and Angular Applications.

[11] “Order tracking: Fixed sampling versus synchronous sampling.” `https://community.sw.siemens.com/s/article/order-tracking-fixed-sampling-versus-synchronous-sampling`, visited 2023-03-17.

[12] S. Fedala, D. Rémond, R. Zegadi, and A. Felkaoui, “Contribution of angular measurements to intelligent gear faults diagnosis,” *Journal of Intelligent Manufacturing*, vol. 29, pp. 1115–1131, 2018.

[13] Y. Shao, D. Su, A. Al-Habaibeh, and W. Yu, “A new fault diagnosis algorithm for helical gears rotating at low speed using an optical encoder,” *Measurement*, vol. 93, pp. 449–459, 2016.

[14] J. W. Cooley and J. W. Tukey, “An algorithm for the machine calculation of complex Fourier series,” *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[15] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[16] A. Kirchner and C. S. Signorino, “Using support vector machines for survey research,” *Survey Practice*, vol. 11, no. 1, 2018.

[17] T. K. Ho, “Random decision forests,” in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1, pp. 278–282, IEEE, 1995.

[18] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.

[19] “Machine learning: Decision tree classifier.” `https://medium.com/machine-learning-bites/machine-learning-decision-tree-classifier-9eb67cad263e`, visited 2023-04-02.

[20] C. M. Bishop *et al.*, *Neural networks for pattern recognition.* Oxford university press, 1995.

[21] H. Jiang, H. Wang, W. Hu, D. Kakde, and A. Chaudhuri, “Fast incremental SVDD learning algorithm with the Gaussian kernel,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3991–3998, 2019.

[22] R. Gabriel, W. Leonhard, and C. J. Nordby, "Field-oriented control of a standard AC motor using microprocessors," *IEEE Transactions on Industry Applications*, vol. IA-16, no. 2, pp. 186–192, 1980.

[23] R. P. Borase, D. Maghade, S. Sondkar, and S. Pawar, "A review of PID control, tuning methods and applications," *International Journal of Dynamics and Control*, vol. 9, pp. 818–827, 2021.

[24] T. Fushiki, "Estimation of prediction error by using K-fold cross-validation," *Statistics and Computing*, vol. 21, pp. 137–146, 2011.

[25] I. Kononenko, E. Šimec, and M. Robnik-Šikonja, "Overcoming the myopia of inductive learning algorithms with RELIEFF," *Applied Intelligence*, vol. 7, pp. 39–55, 1997.