Fault Detection of Inter-Turn Short-Circuited Stator Windings in Permanent Magnet Synchronous Machines

An analysis of current data from a Permanent Magnet Synchronous Machine with an applied Inter-Turn Short-Circuited fault using data processing and Machine Learning model in Python

VILDE SEIM SUNDE

SUPERVISOR
van Khang Huynh

University of Agder, 2023
Faculty of Engineering and Science
Department of Engineering and Sciences

Obligatorisk erklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Jeg erklærer herved at min besvarelse er mitt eget arbeid, og at jeg ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Jeg erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Jeg er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Jeg er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Jeg er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Jeg har satt meg inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Jeg gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Abstract

Hydroelectric Powerplant (HEP) delivers green and reliable energy to the population of Norway, contributing to around 88 % of the yearly power demand. Unexpected shutdowns and downtime of the power-producing units can occur and will result in large economic losses, as well as the HEPs being unable to deliver essential power to the grid. With the rise of Industry 4.0, industries are employing newer technologies like cloud computing, Artificial Intelligence (AI) and Internet of Things (IoT) to improve their operations. In the hydropower industry, AI-based systems are proposed to be used as a foundation for predictive maintenance. Today, most maintenance operations are performed on a scheduled basis and the industry is looking into using Machine Learning (ML) methods for Fault Detection Identification (FDI) of the power-producing units. This thesis looks into the application of ML algorithms to predict early-stage Inter-Turn Short-Circuit (ITSC) faults of a Permanent Magnet Synchronous Motor (PMSM) using three-phase current data. Data A was collected in an in-house laboratory using a Permanent Magnet Synchronous Generator (PMSG) with an implemented 4.8 % ITSC fault, and consisted of healthy and faulty RMS-values of the three-phased current. Data B was obtained from a previously collected setup using the same Permanent Magnet Synchronous Motor (PMSM) with a 6.0 % ITSC fault, that generated signal-based three-phased current values.

In Python, the two datasets were visually inspected and pre-processed using the Z-score method of removing outliers. This step did however not have a noteworthy effect on improving the accuracy of the ML models. Simple signal processing in the time domain was applied to the current data, but could not detect the ITSC fault implemented on the second phase winding. Statistical features like absolute mean, Standard Deviation (STD), skewness, kurtosis, crest factor, peak-to-peak, RMS, clearance factor, shape factor and impulse factor, was calculated for all three phases. A Principal Component Analysis (PCA) algorithm was applied to the datasets with the calculated features and reduced Data A from 18 features to three Principal Components. Data B was reduced from 33 features to four Principal Components. Prepping the data for the ML models, fault indicators that flag values outside the 95th percentile of the mean feature values were added to the data frame. For this thesis, four supervised ML models - Random Forest, Decision Trees, k-Nearest Neighbour (k-NN), and Naive Bayes - were applied to the datasets. The Random Forest and Decision Tree models tended to over-fit the predictions on the datasets containing the features. The data frames with the PCA components reduced the over-fitting of these models and especially improved the accuracy of the Naive Bayes model. As the Naive Bayes model yielded varying results and was deemed inconsistent, as well as the over-fitting tendencies of the Random Forest and Decision Tree, the k-NN model was reviewed as the most reliable ML model for the datasets. The best fault indicators for Data A were the kurtosis and skewness indicators, whereas the clearance factor and shape factor yielded the best accuracies for Data B.

For further work, using data containing RMS values should be avoided, and the focus should be on using signal-based values like Data B. The data processing and labelling should also be performed in the frequency domain, as a great weakness of the thesis is that the methodology was only applied in the time domain. Performance indicators like robustness should also be applied to review the performance of the ML models.

Sammendrag

Vannkraftverk leverer grønn og pålitelig energi til befolkningen i Norge, og bidrar med rundt 88 % av landets årlige strømbehov. Uventede avbrudd og stans for kraftverkene vil resultere i store økonomiske tap, samt at kraftverkene ikke får levert nødvendig kraft til nettet. Med fremveksten av Industri 4.0 benytter industriene nyskapende teknologier som skytjenester, Kunstig Intelligens (KI) og tingenes internett for å forbedre de ulike operasjonene i selskapet. Innen vannkraft-industrien vil KI-baserte systemer bli brukt som grunnlag for prediktive vedlikehold. I dag utføres det meste av vedlikeholdsarbeid i henhold til en planlagt tidsplan, og industrien ser derfor på bruk av maskinlærings-metoder for tidlig feilgjenkjenning i vannkraftverkene. Denne masteroppgaven ser på anvendelsen av maskinlærings-algoritmer for å tidlig forutsi kortslutninger i aramturviklingene i en Permanent Magnet Synkronmaskin (PMSM), ved bruk av trefaset strøm-data. Data A ble samlet inn i et internt laboratorium med en Permanent Magnet Synkrongenerator (PMSG) som hadde en implementert 4.8 % kortslutning i aramturviklingen. Dataen bestod av sunne og defekte datasett med RMS-verdier for den trefasede strømmen. Data B ble hentet fra et tidligere arbeid av den samme typen PMSM med en 6.0 % kortslutning i aramturviklingen. Data B bestod av signal-verdier for den trefasede strømmen.

Ved bruk av Python ble de to datasettene visuelt inspisert og forbehandlet ved hjelp av 'Z-score'-metoden for å fjerne avvikende verdier. Denne prosessen hadde imidlertid ingen merkbar effekt på nøyaktigheten til maskinlærings-modellene. Enkel signalbehandling i tidsplanet ble anvendt på strømdataene, men klarte ikke å oppdage kortslutningsfeilen implementert på den andre faseviklingen. Statistiske parameter som gjennomsnitt, standard avvik, skjevhet, kurtose, toppverdifaktor, peak-to-peak, RMS, klaringsfaktor, formfaktor og impulsfaktor ble beregnet for alle tre fasene. En Principal Component Analysis (PCA)-algoritme ble anvendt på datasettene med de statistiske parameterne og reduserte Data A fra 18 parameter til tre Principal Components. Data B ble redusert fra 33 parametere til fire Principal Components. Før dataen kjøres i maskinlørings-modellene, ble feilindikatorer som flagger verdier utenfor den 95. persentilen av gjennomsnittsverdiene til parameterne lagt til i datasettet . Fire overvåkede maskinlærings-modeller – 'Random Forest', 'Decision Trees', 'k-NN' og 'Naive Bayes' – ble kjørt for datasettene. Random Forest- og Decision Tree-modellene hadde en tendens til å overtilpasse maskinlærings-prediksjonene på datasettene som inneholdt de statistisk parameterne. Datasettet med PCA-komponentene reduserte overtilpasningen av disse modellene og forbedret nøyaktigheten til Naive Bayes-modellen. Ettersom Naive Bayes-modellen ga varierende resultater og ble ansett som inkonsekvent, samt overtilpasnings-tendensene til Random Forest og Decision Tree, ble k-NN-modellen vurdert som den mest pålitelige av maskinlærings-modellene. De beste feilindikatorene for Data A var kurtose- og skjevhet-indikatorene, mens klaringsfaktor og formfaktor ga best nøyaktighet for Data B.

Videre arbeid bør unngå bruk av data som inneholder RMS-verdier, og fokusere på bruk av signalbaserte verdier slik som i Data B. Dataprosessering og feilmerking bør også utføres i frekvensplanet, ettersom en stor svakhet ved avhandlingen er at metodikken kun ble anvendt i tidplanet. Andre ytelsesindikatorer som robusthet bør også brukes for å vurdere ytelsen til maskinlærings-modellene.

Acknowledgements

This master's thesis was written in the spring of 2023 at the University of Agder (UiA), campus Grimstad. It was written as a part of the research group on Condition Monitoring (CM), for a collaboration between UiA, NORCE, Å Energi (prev. Agder Energi), and Volue. I would like to thank all the people who helped and guided me through the work of this thesis. I would like to thank my supervisor Professor van Khang Huynh and the CM research group for guiding me through the thesis. I would also like to give special thanks to Mahmoud Sayed Mahmoud Eid for great guidance and much-needed help in the laboratory, and Hoang Nam Du Nguyen for his knowledge of Python coding. Thanks to the lab engineers at UiA for answering my endless questions on the setup and equipment, Jan Christian Bjerke Strandene, Harald Sauvik and Steve Schading. I would like to give special thanks to Victoria for proofreading and giving useful feedback on the writing. Lastly, I would like to thank my co-students for making all the hours spent at the master-laboratory worth the while, and my family for always supporting me.

Contents

List of Figures

List of Tables

Nomenclature

$\boldsymbol{A}$	Matrix
A	Cross-sectional area of the cable
d	Sampling frequency
Δ	Difference
f	Frequency
f_s	Fundamental frequency
$\boldsymbol{I}$	Identity matrix
I	Current
I_p	Peak current
I_{RMS}	RMS value of the current
L	Length
λ	Eigenvalue
μ	Mean value
$\mu_{f,ITSC}$	Severity of the ITSC fault
N	Number of samples
n_s	Synchronous speed
p	Number of poles
ϕ	Phase angle
$Proj_{P_i}$	Projected value
R_{cable}	Resistance of cable
ρ	Resistivity of cable
σ	Standard deviation
t	Time
θ	Angular tilt
TN	True Negatives
TP	True Positives
$\vec{u}$	Unit vector

$\vec{v}$	Eigenvector
x	Raw data value
x	Variable (I, U)
$\bar{X}$	Mean value
X_{clear}	Clearance Factor of X
X_{crest}	Crest Factor of X
x_i	ith value of variable
X_{imp}	Impulse Factor of X
X_{kurt}	Kurtosis of X
X_{max}	Maximum value of X
X_{min}	Minimum value of X
X_{P2P}	Peak-to-Peak value of X
X_{RMS}	RMS-value of X
X_{shape}	Shape Factor of X
X_{skew}	Skewness of X
X_{STD}	Standard Deviation of X
Z	Deviation from the mean

Abbreviations

AI Artificial Intelligence

BRB Broken Rotor Bar

CM Condition Monitoring

EFA Exploratory Factor Analysis

FDI Fault Detection Identification

HEP Hydroelectric Powerplant

IoT Internet of Things

ITSC Inter-Turn Short-Circuit

k-NN k-Nearest Neighbour

KI Kunstig Intelligens

LSPMSM Line Start Permanent Magnet Synchronous Motor

ML Machine Learning

MMF Magnetic-Motive Force

NaN Not a Number

NN Neural Networks

PCA Principal Component Analysis

PMSG Permanent Magnet Synchronous Generator

PMSM Permanent Magnet Synchronous Motor

SC Short Circuit

STD Standard Deviation

Chapter 1

Introduction

In this introductory chapter, the background motivation of the thesis is presented. A short insight into some of the more important, basic theoretical aspects of hydropower generation will be explained. To give further insight into the span of the thesis, the goals and limitations will be given. Lastly in this chapter, the structure of the thesis will be described.

1.1 Background

Industries today are going through a technical revolution, often referred to as the Forth Industrial Revolution [1]. Industry 4.0 represents the enhancement of industrial processes and applications by a higher level of automatisation and intelligent optimisation [2]. This too, is applied in the hydropower sector, especially when it comes to Condition Monitoring (CM) and maintenance of the Hydroelectric Powerplant (HEP). Today, maintenance is usually executed in set intervals on a scheduled basis, or during downtime due to system failure [3]. The un-scheduled and unexpected maintenance are costly affairs as the downtime often causes large revenue losses [2]. For instance, in the hydropower station in Rygene, Norway, an unforeseen failure accounted for a revenue loss of 60 million NOK, and a downtime of 18 months [4]. The downtime of the HEP is also a threat to the energy security of the grid, as a steady flow of energy is halted from the HEP. With increasing effectiveness and precision, data processing techniques like signal processing, modelling and machine learning algorithms can detect faults using online sensors and equipment. The aim is to detect the fault and severity at an early stage, so a maintenance plan can be laid. The industry is in other words moving towards a predictive type of maintenance, where maintenance can be scheduled when needed, and thus eliminate any unnecessary downtime [5].

HEPs have been central in Norway's green power production, producing around 88 % of the Norwegian yearly power production [6]. The many rivers and lakes of Norway have been the main supplier of energy for many decades. Hydropower is a green source of energy and can be used to balance the grid as a lot of the production is flexible. Approximately 77 % of the Norwegian hydropower is flexible in means of power stored in reservoirs [7]. Despite the constant build-out of HEPs today, many of the stations in operation are built in the 60s, 70s, and 80s, as the systems have a long lifespan [5]. A predictive type of maintenance will also help predict the residual lifetime of the system, as well as prevent the destruction of the components if a fault is caught early [2].

1.2 Goals

The goal of the thesis is to look into Fault Detection Identification (FDI) of a Inter-Turn Short-Circuit (ITSC) fault in a Permanent Magnet Synchronous Motor (PMSM), using signal-processing and Machine Learning (ML) methods in a Python program. To be able to do this, extensive research on the topics has to be undergone, as well as gathering Data **A** using in-house equipment in a laboratory setup. Data from a PMSM gathered the previous year for a PhD research, called Data **B**, will also be used in the Python program. The results will show what features and methods yield the best accuracy of the fault prediction. The overall aim of the thesis is to contribute to the research that is required for moving the hydropower industry alongside Industry 4.0 and towards predictive maintenance.

1.3 Limitations

The biggest limitation for this thesis was time. The initial idea of the thesis was to obtain data from a real-life hydropower station to ensure that the findings could be applied and tested in the industry. However, this was not possible in the time frame of the thesis, and the data had to be obtained through data collection in an in-house laboratory using a smaller PMSM. The experimental setup required ordering new parts that took longer time to arrive than expected, and thus, less time was spent on the data analysis part of the thesis. The chosen data acquisition system did also limit the type of data that was gathered, where RMS-data of the Permanent Magnet Synchronous Generator (PMSG) was obtained. Preferably, the data should have been three-phase sinusoidal values. Luckily, data from previous work performed in the laboratory was obtained later on, and some analysis could be applied to the datasets. Due to the time limit, the data analysis was therefore only performed in the time domain and not the frequency domain.

To form an understanding of the topics ahead, some background information on HEPs and the Permanent Magnet Synchronous Machines is going to be presented.

1.4 Hydropower Generation

A typical HEP unit consists in broad terms of a turbine driven by the water from the inlet, which again drives a hydroelectric generator, see Figure 1.1. Using pipes, the potential energy of the water is transformed into kinetic energy that is sent through a turbine. A generator transforms the mechanical energy of the rotating turbine into electrical energy that is delivered to the grid. The type of generator is selected by parameters like the fall height of the water, the discharge range of the reservoir or river, the type of turbine blade, and many more parameters. The generators need to handle varying loads, as the water supply fluctuates throughout the seasons and years [8]. Most hydropower stations regulate the varying rotational load with a gear so that the synchronous generator is kept at a constant rotational speed. Other than the hydropower units delivering green energy and having a long lifetime, a great advantage of HEP is that they can quickly be connected to the grid and help to regulate the frequency of the grid [8]. The stations with the most regulatory power are reservoir stations. However, most river-based HEPs have some regulatory power, where a limited range of the water height is defined for each river system.

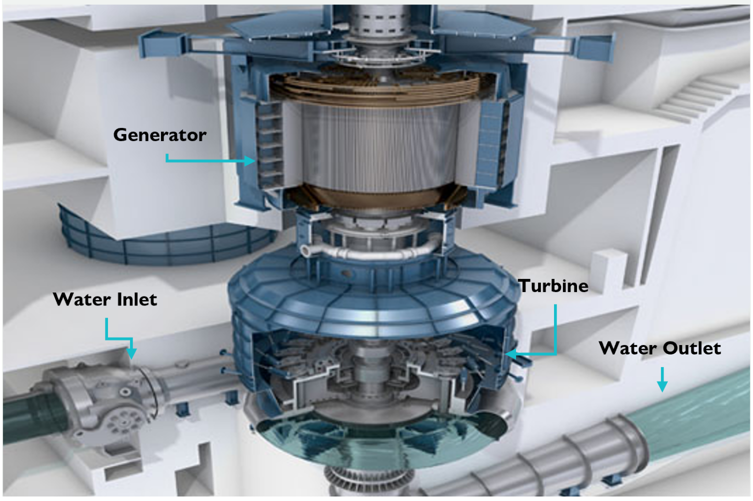


Figure 1.1. Illustration of the power producing system in a Hydroelectric Powerplant (HEP) ©GE.

1.4.1 Permanent Magnet Synchronous Machines

Most hydropower generators are synchronous machines, and a PMSM was used for this thesis. Permanent Magnet Synchronous Machines (PMSM) are rotating machines, and the working principle is the same for Permanent Magnet Synchronous Motor (PMSM) and Permanent Magnet Synchronous Generator (PMSG). The only difference is whether the power is applied to the machine, or collected from the machine. Figure 1.2 shows the internal components of a PMSG, where the stator is the stationary part of the machine and the rotor is the rotating component. The basic working principle of all rotating machines is that electromagnetic energy is generated by mechanically turning windings through a magnetic field [9]. For the PMSM, the windings are placed in the stator, and the permanent magnet rotor is excited by a field current, creating a magnetic flux field. When the rotor rotates, this magnetic field is passed through the windings, and thus a voltage is induced. Usually, the windings are coiled in groups of two or three, and these are called armature windings [9]. The PMSM in question have three armature windings, generating three-phase signals. In the figure below, the windings are of the bar wound wire type, whereas the in-house PMSM have wired-wound armature windings. The bearings allows for smooth rotation of the rotor, and gives support to the components.

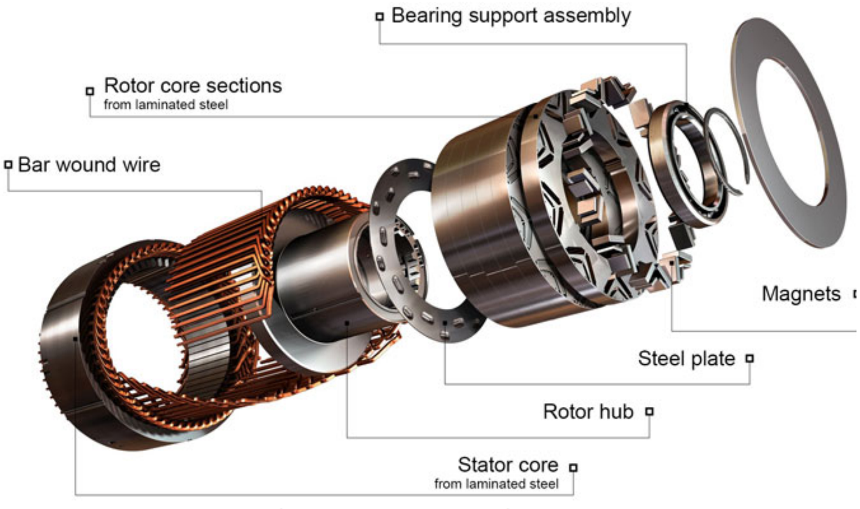


Figure 1.2. Illustration of a Permanent Magnet Synchronous Generator (PMSG) internal components ©EngineeringSolutions.

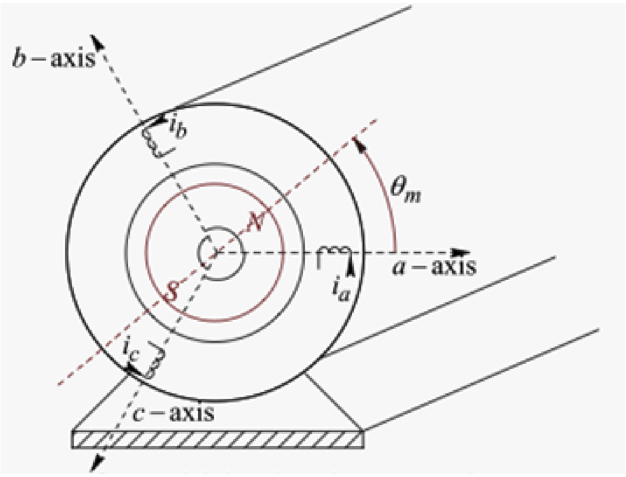


Figure 1.3. Schematic of a Permanent Magnet Synchronous Motor (PMSM).

In an idealized version of the synchronous machine, the magnetic air-gap flux created by the field winding is assumed to be sinusoidal in its distribution. As the flux field rotates with the rotor, the flux induces a voltage in each of the armature windings as it passes, creating a sinusoidal voltage and current signal. The sinusoidal waves of the three phases are shown in Figure 1.4 and the mathematical expression for sine-waves is described in Equation 1.1 [9]. $I(t)$ represent the current value of the wave at time t; I_p is the peak value of the current; f is the frequency; and ϕ is the phase angle. Ideally, the phases are spaced with 120 °between them, as they also are physically in the stator (see Figure 1.3).

$$I(t) = I_p \sin(2\pi f t + \phi) \tag{1.1}$$

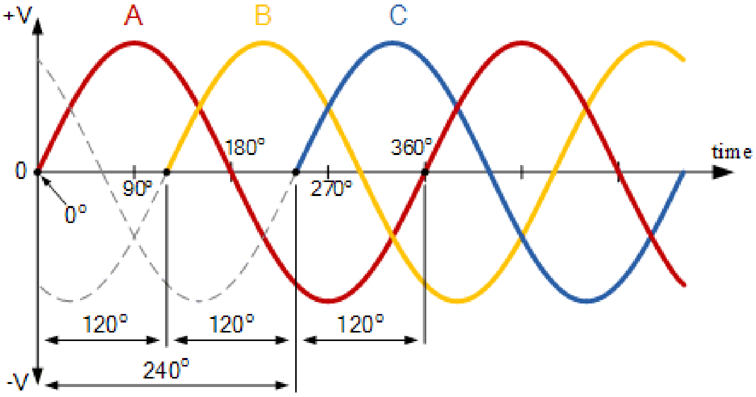


Figure 1.4. Plot of a three-phase sinusoidal wave for phase a, b and c ©ElectronicTutorials.

Another parameter that is important for the understanding of Data A, is the RMS-value of the current. The Root Mean value of the Square function, or the RMS, is essentially a mean value of a certain number of values. A general expression for the RMS value of the current is shown in Equation 1.2 [9], where I_{RMS} is the RMS value of the current; N is the number of samples; and i_i is the i^{th} value of the current.

$$I_{\text{RMS}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} i_i^2} \tag{1.2}$$

1.5 Structure of Thesis

The further structure of the thesis will be as follows:

Chapter 2 will give a look into the available literature on relevant topics, starting with the different types of faults in a PMSM and some of the diagnostic methods. The literature review is based on a submission for a previous course [10], and will also provide theoretical background on the subjects of the thesis. The topic of CM and Fault Detection Identification (FDI) methods machine learning methods will also be reviewed. Then lastly, the topic of data quality in ML methods will be shortly reviewed. Based on the performed reviews, the scope of the thesis will be stated.

Chapter 3 will describe the methodology for the entire thesis. The chapter will go through how the research was conducted, the describe shortly Data A and Data B. Then, the entire analysis of both datasets in Python will be described, as well as the theoretical framework for pre-processing, signal-processing, feature extraction, Principal Component Analysis (PCA) and Machine Learning (ML) models.

Chapter 4 goes in further depth of the experimental method of the data gathering of Data A. Potential sources of errors attached to the data gathering is also described in this chapter.

Chapter 5 presents the final results and findings of the thesis, as well as discusses them.

Chapter 6 summarises the main findings and concludes the thesis. Lastly, propositions and thoughts on further work will be given.

Chapter 2

Literature Review and Theoretical Background

The Operation and Maintenance (O&M) post can be further developed to further ensure the hydropower industry's economic strength. Condition Monitoring (CM) and Machine Learning (ML) plays a big role in the future of O&M of Hydroelectric Powerplant (HEP) as well as other power producing industries. This chapter will present a literature review of relevant topics, before summarising the findings and defining the scope of the thesis. The literature review will be divided into two main sections, where firstly the common faults in synchronous generators will be described. Then, the literature on the use of machine learning methods in condition monitoring in the hydropower industry will be presented.

2.1 Common Faults of Synchronous Generators

The faults and inevitable failures of synchronous generators HEPs have been well documented. A study from the early 2000s by CIGRE categorised 69 incidents of the breakdowns of over 1 200 hydro generators [11]. The root causes of these breakdowns were documented to be 56 % due to the fault in the electrical insulation, 24% due to mechanical defects, 17 % thermal issues, and 3 % due to fault in the bearings, as seen in Figure 2.1. The collected data of CIGRE have been widely used in other studies, many of which are relevant for forming a basic knowledge of CM in hydropower generators [12], [13]. An extensive review of CM for rotating electrical machines was published by Tavner (2008), where the author states that published studies in IEEE and IEE link 21 % of failed sub-assemblies due to bearing faults, 35 % from stator-related faults and the remaining 44 % from rotor related faults [13]. The author was surprised by the few root causes and noted that the research papers were generic. Another common way of classifying the stresses that electrical machinery is exposed to, is TEAM - Thermal, Electric, Ambient and Mechanical Stresses [3]. Considering the fact that the hydropower stations in Norway are quite old and that the technology of the components has not changed much over the last 20-30 years, the CIGRE classification of root causes is a great indication of the causes of failures.

Further on, some of the most common generator faults will be reviewed, starting with eccentricity and moving on to broken damper bars, bearing faults and short circuit faults.

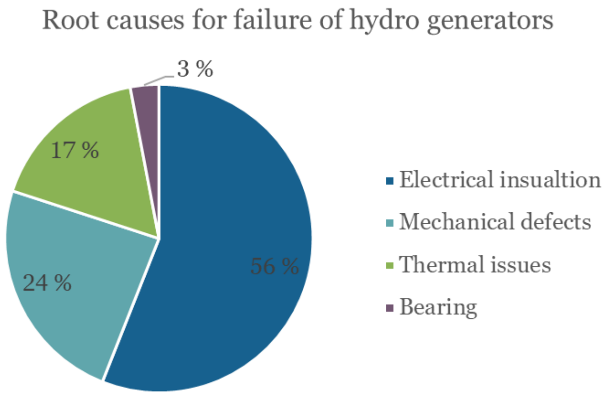


Figure 2.1. Root causes for failure of hydropower generators [11].

2.1.1 Eccentricity Faults

Eccentricity is a prevalent mechanical fault of the rotor that causes unwanted movement and vibration in the generator. It is usually due to misalignment and unbalanced mass centers stemming from the incorrect installment or production of the components, or vibration and movement from other parts of the generator [10]. Other causes for eccentricity can be from oil whip, thermal sources, unbalanced magnetic pull (UMP), or from different stiffness in the components [14]. Eccentricity faults can cause huge revenue losses as an eccentricity fault would metastasize further issues in the generator, such as more vibrations, UMP, and induce currents in generator parts [14]. Eccentricity occurs as static or dynamic eccentricity as seen in Figure 2.2, or a mix of the two [15]. Static eccentricity is the identical, but displaced, rotational axis of the rotor and stator. Dynamic eccentricity is the displacement of the rotor's rotational axis, making it asymmetrical to the stator's rotational axis.

Ehya et. al [15] state that over the last decade, eccentricity faults have been diagnosed using a range of metrics but that the diagnostic methods have been flawed. Methods using magnetic flux density are invasive and costly; current harmonics does not work under nonlinear load and cannot indicate severity; phase voltage harmonics is not useful under nonlinear load and susceptible to noise; shaft voltage and flux uses sensitive equipment and fails to detect the type of eccentricity fault as well as the severity [15]. The article suggests stator terminal phase voltage is the most reliable signal as it is independent of load harmonics. So far, the struggle with eccentricity is detecting it at its early stages and the severity of the fault. In addition, off-line methods have been mostly implemented in the industry, which

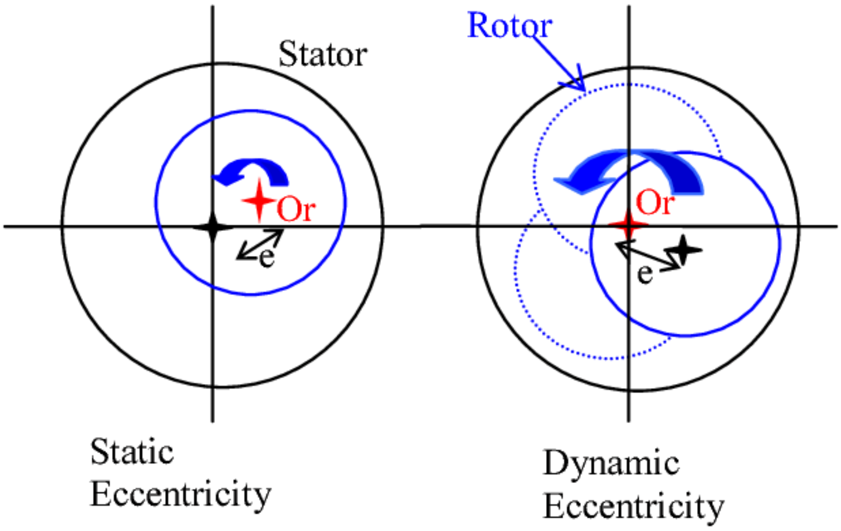


Figure 2.2. Visualisation of static and dynamic eccentricity [16].

is often associated with an expensive approach [17]. A newer approach by the same author was published in 2020 that proposes a statistical method using air-gap magnetic field data to detect static eccentricity in the early stages [17]. The reviewed literature on CM and fault detection of eccentricity faults use statistical approaches [17] or mathematical approaches [18]. Another paper that used a mathematical-based neural network to detect eccentricity faults in a Line Start Permanent Magnet Synchronous Motor (LSPMSM), showed promising fault detection results on unseen values as well as the tested data [19].

2.1.2 Broken Damper Faults

Damper bars in electrical rotating machinery have the purpose of ensuring stability in the transient mode of the operation, where the asynchronous behavior of the generator will be induced in the rotor bars. Thus, the damper bars prevent short circuits in other important windings [20]. When a damper bar breaks, it can cause hot spots in the machinery due to electrical arcs being created in the broken parts [21]. These hotspots can further cause unbalance in the machinery, in addition to the decreased transient stability due to the broken bars. As the damper bars only operate at the start and during transient operation and have a relatively low occurrence of broken damper bars (BDB) faults, the research is not widespread [22]. However, the causes of the faults are known to normally be due to stresses such as dynamic stresses from rotating forces, thermal stresses, or stresses from other broken components [23].

A research article from (2022) notes the current method of diagnosing a broken damper bar today is either offline and visual inspections, or by invasive sensors that operate at transient operations [20]. These approaches are however not an all-around solution, as they are either too costly or too narrow of a search during operation. A proposed method of non-invasive sensors with a broader working spectrum was covered in [20]. The method uses wavelet entropy analysis of stray magnetic fields caused by the broken damper bar, and it was found that the location of the fault had quite an impact on the magnitude of the created criterion function. In addition to the relatively limited research on condition monitoring on BDBs in electrical machinery, there are few articles on using machine learning to detect faults [24]. One paper from 2019 on the other hand, uses data mining on time series data to detect BDB faults in the induced voltages in the rotor windings [22].

2.1.3 Bearing Faults

A more studied topic is bearing faults of rotary electrical machinery, a fault that affects many industries due to the wide application of bearings [25]. Bearings often fail to live the expected life and often lead to system failures [26]. A review of fault diagnosis of bearing faults using machine learning and signal processing (2022) [26] finds that the majority of the extensive research on the topic is focused on vibration analysis with further machine learning steps. Most of the research looks specifically at the wear on the inner and outer rings, and the authors detect a research gap on the impacts on machine vibration from ball wear and cage wear [10]. For signal processing and machine learning, vibration and noise is the most commonly used datatype for condition monitoring using ML, but temperature, pressure, and oil monitoring are other common monitoring techniques [27]. A comprehensive review of bearing fault detection was published in 2022 [28]. Khan et al. reviews a range of fault detection methods ranging from conventional approaches to statistical approaches, as well as artificial intelligence-based method. The review raises the data generation, lack of studies in the frequency spectrum, and noisy data to be the current challenges for the topic of bearing fault detection.

2.1.4 Short Circuit Faults

Short Circuit (SC) faults are prevalent faults of synchronous machines and can occur in all of the electrical wiring [10]. The main reason for a SC in electric machinery is related to damage to the insulation system of the machine [29], [30]. The CIGRE report [11] found that insulation damage occurs due to aging; contamination of the windings; internal partial discharges; loosening of bars; thermal cycling and overload; and over-voltages, as seen in Figure 2.3. These factor causes the insulation to degrade, and when the insulation on the windings are extensively degraded a SC of the winding occurs. When this happens in the windings of either the stator or rotor, this is called Inter-Turn Short-Circuit (ITSC), and this type of fault has a higher severity than others [29]. When an SC occurs, the large current that is sent through the SC winding will generate overheating locally to the fault. This will, in turn, create more damage to the insulation, and worsen the condition of the fault. Due to the high severity of the SC, it is important to detect these faults in the early stages [29], more so than the other faults mentioned in this chapter. [29] describes a SC in the rotor windings as a demagnetizer component. It acts like a demagnetizer as the SC winding causes a current in the opposite direction than the current flow, which creates a Magnetic-Motive Force (MMF) opposite to the main MMF.

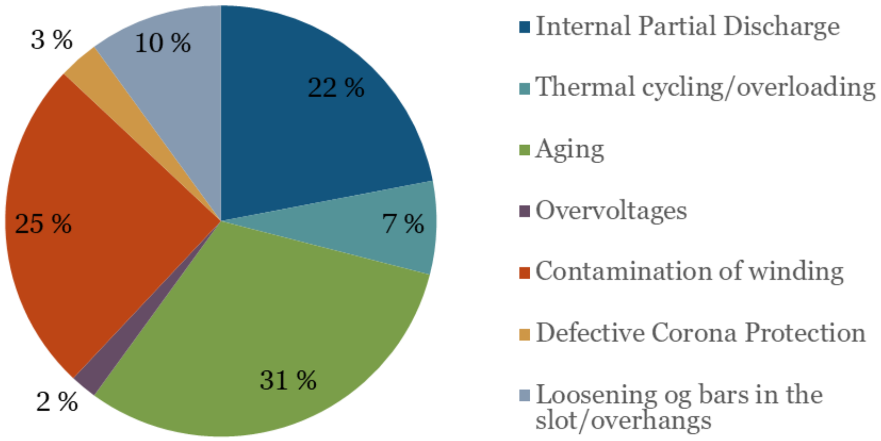


Figure 2.3. Root causes for insulation damage, reported by CIGRE [11].

There are many diagnostic methods and data types used to diagnose SC faults. One review on rotor-side short circuit faults goes in-depth on seven different indices of early fault detection and evaluates them at eight different parameters [29]. The methods evaluated are the MMF index, impedance index, air gap magnetic field index, current frequency spectrum, electromagnetic power index, electromotive force index, and shaft flux index. The author proposed the MMF and stator-rotor current to be the most advantageous as early fault indicators. There is also some research on fault diagnosis of PMSM ITSC faults that uses models of the motor to obtain data, and tests the ML models at various conditions [31]–[33]. The data of these methods are however model-based and will thus have an uncertainty about them, and thus the usability of the detection algorithms is somewhat weakened. Other research on SC fault detection using ML methods does exist, for instance, [34] proposes a method using the Random Forest classifier to detect SC fault in a squirrel-cage motor.

2.1.5 Diagnostic Methods

Some of the diagnostic methods have already been mentioned, but this chapter will shortly go through the most common detection methods. According to a review from 2021 on trends and challenges in CM of electrical machines, Kudelina et al. [3] categorises the current diagnostic methods for fault detection of electrical machines to be:

- Noise and vibration monitoring
- Motor current signature analysis
- Temperature measurements
- Electromagnetic field monitoring
- Chemical analysis
- Radio Frequency (RF) emission monitoring
- Acoustic noise measurement
- Modelling and AI-based techniques

The authors further describe which of the different signature datatypes can be used to detect Short Circuit (SC) faults, Broken Rotor Bar (BRB), eccentricity, and bearing faults [3]. Vibration data can be used to detect all four fault types but is most preferable for bearing fault detection. Current data is preferable for all types except eccentricity, although it can be used for that as well. Temperature can be used for all types except bearing faults, and chemical analysis can only be used for SC faults. Magnetic flux changes are an indicator for all four types but are preferable for SC, BRB and eccentricity faults. Lastly, torque data can be used for eccentricity fault detection, but is a preferable indicator for SC and BRB.

2.2 Machine Learning in Condition Monitoring

One of the diagnostic methods described in the list in Chapter 2.1.5 is modeling and AI-based techniques. For fault detection, the diagnostic methods are based on either signal processing, modeling, or Machine Learning Algorithms. Signal-processing techniques are often performed in the frequency domain, with techniques like Periodogram, Spectrogram, Scalogram, Discrete Wavelet Transform or the Wavelet Packet Transform [35], [36]. Signal processing can also be applied in the time domain, but these methods are simple and not as advanced as the frequency-based methods. Machine Learning is a powerful tool that has a wide application. The topic of ML in CM will be further explained later in this chapter.

2.2.1 Machine Learning in Hydropower Industry

As of today, using machine learning for condition monitoring in Hydroelectric Powerplant (HEP) is not quite at the predictive state, but more so in the developing state. Some sources state that the largest bottleneck today is the lack of data to train the models on [1]–[3]. Raja et al. [1] goes into further detail and states that the limitations come from the development of micro-controller boards and fault prognostics, in addition to the limitation of available data. Installation of sensors for specific fault detection usage is also needed. Overall, the result is a lack of fault diagnostic models and statistics. [1] states that the few, but current CM systems using ML are quite expensive, which is a reflection that the systems are not at a commercial level yet. The authors also state that identifying the faults and their early signs demands a lot of testing, statistics, and analysis before the technology has reached commercial levels.

Machine learning approaches to condition monitoring have been applied more extensively to the wind turbine industry, autonomous vehicles, weather station operations, and load forecasting [1], [3]. In the instances such a system is used to maximize the potential of the electrical machines, the cost and weight of the equipment itself is a central issue [37].

2.2.2 Machine Learning Types

Machine learning is a field of study where algorithms are used to solve problems by learning from other similar problems [3]. The aim of the ML model is to predict parameters and create models that can detect the researched fault. Mainly, machine learning can be divided into supervised learning; unsupervised learning; Reinforcement Learning; and Neural Networks [3]. Examples of models for the four categories are shown in Table 2.1.

Supervised learning and Unsupervised learning use a set of training data to create predictions [38]. The main difference between the two types is that supervised learning uses pre-defined labels on the data, whereas unsupervised learning creates its own. Supervised learning is either regression or classification-based [39]. Regression is used for the prediction of continuous values, where the output will be a real value. Classification is used to predict a classifier, where the output will be the class of which it fits into [40]. Unsupervised learning is divided into clustering and association methods [39]. Clustering is when the unlabelled data is clustering together based on their similarities. Association is based on the relationship between the data points, and when these points occur simultaneously [41].

Reinforcement learning is ML models that assign positive values to behaviours of the data that are found to be desired based on a pattern, whereas a negative value will be assigned to unwanted behaviours. This type of ML is often used for real-time decision-making for applications like robot navigation, gaming, and resource management [3] Neural Networks (NN) are made to resemble the biology of a neural network, as the neural system is able to learn and correct itself [3]. A subcategory of NNs is deep learning, and a great advantage is that the algorithm itself will extract features from the data, and there is no need for the time-costly labour of manual feature extraction [30].

Table 2.1. Categorisation of Machine Learning (ML) methods [3].

Types of Machine Learning Methods			
Supervised Learning	Unsupervised Learning	Reininforcement Learning	Neural Networks (Deep Learning)
Decision Tree Discriminant Analysis Linear Regression Logistic Regression Naive Bayes Nearest Neighbor Random Forest Similarity Learning Support Vector Machines	Fuzzy C-Means Principal Component Cluster Analysis K-Means	Control Theory Game Theory Genetic Algorithms Multi-Agent Systems Simulation-Based Optimization Statistics Swarm Intelligence Q-Learning	Autoencoder NN Convolutional NN Perceptron NN Probabilistic NN Recurrent NN

2.2.3 Data Quality in Machine Learning

For the application of ML in CM of the hydropower sector, data quality becomes an important topic. Many architectures [39] and libraries [38] have been proposed and implemented for large datasets to ensure the quality of the data. In this chapter, some key factors and thoughts on the data quality in ML models will be explained.

An article on data quality considerations for machine learning states that the ML models' accuracy is highly dependent on biases, variances and irreducible errors [39]. These factors affect the accuracy of the models and can cause wrong predictions and over-fitting [39]. Over-fitting is the phenomenon where the model creates great predictions for the training data but does not predict well for testing, or unknown, data [42]. If the predictions are over-fitted due to the type of model used, the error stems from biases. For instance, it would be hard to predict values with a nonlinear relationship using a linear model. Variances affect the model if the models have a sensitivity to larger variances in the dataset, as decision trees have. Fortunately, the decision trees are robust in the sense that they are broadly applicable, and thus not that prone to biases. The last factor that contributes to the wrongful predictions in the testing data is the noise that exists in the dataset, or as the article calls it, irreducible errors. The article also proposes the PCA and the Exploratory Factor Analysis (EFA) methods to reduce the dimensions of the dataset.

Over-fitting is a common issue for supervised ML models and is usually caused by the presence of noise, limitations on the training set size, and the complexity of the predictors [42]. Over-fitting is when the model predicts the training set with high accuracy but fails to predict the testing set with the same high level. x. Ying [42] presents four main solutions to over-fitting, where the first one stops the training before the performance is fully optimised. The second solution is a strategy that removes noise in the training set so that the generalised model is not affected by noise in training. The third strategy is for more complex models, where over-fitting is reduced by expanding the training set and thus fine-tuning the model. The last proposed strategy is using feature selection on the datasets, where an algorithm is used to extract the most important features, thus reducing the dimensions of the dataset.

2.3 Scope of the Thesis

The literature review in Chapter 2 gave an insight into what sections of the field have been studied and researched from a machine learning perspective. For instance, bearing and eccentricity faults are well-studied fields, whereas broken damper bars have not been heavily researched. As described in Chapter 2.1.2, the lack of research can be linked to low occurrences and limited operation. Short circuit faults are also a researched topic. However, considering the high severity of the fault as well as the high occurrence, fault detection of the short-circuited windings should be further developed. Through the literature research, it is apparent that the main bottleneck of using machine learning in condition monitoring is not the number of research on specific types of faults, but the data quality available. In machine learning, there isn't a one-size fits all, and at the stage that the technology is for condition monitoring, each fault needs to be inspected individually [3].

Based on the review of the literature and the accessible in-house equipment, the focus of this thesis is on the Inter-Turn Short-Circuit (ITSC) faults in the stator windings of a PMSM. Usually, these faults are due to partial discharges and degradation of the insulation, and a great indicator of such faults is temperature monitoring. This method has however been proven ineffective and costly due to the invasive placement of the sensors. The paper will use current data to detect the ITSC faults, as recommended by [3]. Based on the authors' previous knowledge and available time, supervised Machine Learning (ML) models will be applied to the data in Python.

Chapter 3

Methodology

This chapter will go through the methods applied in this thesis and describe some theoretical aspects of the processes. The methodology of creating a fault detection model is summed up in the pipeline schematic in Figure 3.1. For Data A, a data gathering step is required, whereas Data B is obtained from another work at UiA [43]. After obtaining the data, the datasets are processed in Python. The first Data Processing step is pre-processing, then features are extracted and the data is labelled. A PCA is also performed on the data, and both the original features and PCA features are sent through simple signal processing. In addition to the signal-processing step, the data frames are trained and tested in four ML models. The following chapters will go through the steps in further detail, but firstly, the methodology of the research will be explained.

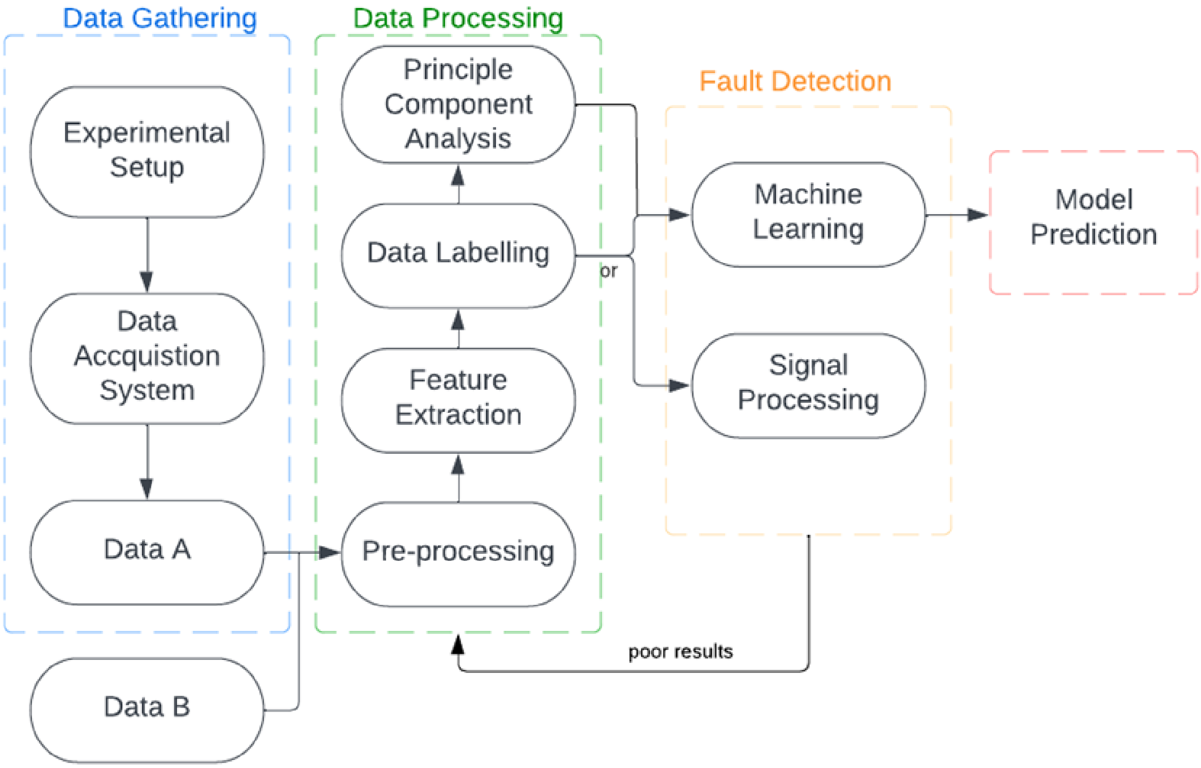


Figure 3.1. Pipeline of the methodology of the thesis.

3.1 Research of the Sciences

The method of the research part of the thesis consisted mainly of a literature study on the field, as shown in Chapter 2. The findings were also used as a basis for the theoretical aspects of this chapter. The researched articles and studies were mainly found in the IEEE Xplore and Research Gate databases. Google Scholar also proved to be an efficient search engine when looking for specific sources. The first targets in the searches were on the different specific faults such as eccentricity, broken damper, bearing, short circuit, thermal and mechanical faults. The second targets were machine learning techniques and big data issues with search words like supervised learning, unsupervised learning, neural networks, Internet of Things, Big Data, and Data Quality. Adding keywords to the searches like condition monitoring, synchronous machines/generator, fault diagnostics, machine learning, hydropower and predictive maintenance, narrowed the search and led to research in similar sections of the industry and academia. This traditional literature study gave a broad insight to the subject of the thesis, and the following methodology descriptions will be on the data gathering and data analysis processes. These processes were influenced by the findings in the literature study and the in-house resources.

3.2 Data Gathering

The data used in this thesis was obtained from an experimental setup (Data A), and from S. Attestog's PhD dissertation [43] which was recorded the previous year (Data B). The collection of Data A will be further explained in Chapter 4, and the gathering resulted in current and voltage data in RMS-form from the PMSG. Data B was collected from a motor of the same type of machine as Data A, and yielded three-phased current signals. Both machines are the VEM 2.2 kW PMSMs (IE5-PS2R 90 L). The specifications of the machine can be viewed in Table 4.1 in Chapter 4. The most important differences between Data A and B can be viewed in Table 3.1. Both datasets were run at 50 % torque load, but Data A was run at 500 rpm and Data B at 750 rpm. Another important parameter to note is the sampling frequency, where Data A has a frequency of 100 Hz, and Data B, has 10 kHz. The difference in sampling frequency is due to the data acquisition system, where the MicroLabBox collects data at a higher rate than the Power Analyser. A descriptive parameter when it comes to synchronous machines is the frequency. The fundamental frequency, f_s, of the motor and generator are calculated using Equation 3.1, and is determined by the synchronous speed, n_s, and the number of poles, p.

$$f[Hz] = \frac{n_s[rpm] \cdot p}{120} \tag{3.1}$$

Table 3.1. The most important specifications of Data A and Data B.

Data Input		
Parameters	**Data A**	**Data B**
Source	Experimental setup (Chapter 4)	PhD dissertation [43]
Data Type	I, U (three-phase RMS values)	I (three-phase signal)
Sampling Frequency	100 Hz	10 kHz
Data Aquistion System	HIOKI Power Analyser (PW6001)	MicroLabBox
Rotational speed, n_s	500 rpm	750 rpm
Fundamental frequency, f_s	16.67 Hz	25 Hz
Torque Load	50 %	50 %

Data A was imported directly into Python as .csv files and converted into `pd.DataFrames` by concatenating the different files. Data B, however, was obtained as a MATLAB file, and thus, the .mat file needed to be exported into a flat structure when imported to Python. This is done using the `scipy.io` library and extracting the wanted strings into lists. The data import in Python is shown in Appendix A.1.

3.3 Pre-Processing and Data Transformation

The quality of the data used in condition monitoring for predictive maintenance is a big bottleneck, whether the method is signal-based, model-based, or machine learning-based. To ensure good quality of the data, besides ensuring little sources of error in the gathering of the data, a pre-processing step can help improve the quality. In this section, the methods applied in Python will be described, all with the purpose of enhancing the quality.

Although pre-processing can increase the accuracy of the ML mode and thus help better the data representation, the process can also be time-consuming, contribute to data losses, and be the cause of over-fitting [39].

3.3.1 Pre-Processing

The datasets can be pre-processed, or cleaned, using various methods. This is especially important as the quality of the dataset will have an impact on the accuracy of the ML models [38], where the accuracy of the model can be viewed as an indirect indicator on the quality of the data [39]. The pre-processing step in most machine learning models consists mainly of removing outliers; filling of NaNs; removal of noise; data balancing; standardization of the data; and normalisation of the data. Outliers and noise appear often in a dataset due to the data acquisition, where wrong data is recorded during the measurement and transmission of the signal. Noise can also exist due to the surrounding of the setup, and stems from excessive noise, vibrations, flux fields, and such. NaNs usually appear if the recorded data is outside the bounds of the data acquisition system. In this thesis, the data is pre-processed by removing outliers and filling in Not a Number (NaN)s. The Python code can be viewed in Appendix A.2.

Outlier Removal and NaN filling

The method for outlier detection and removal will in this thesis be performed using the statistical feature Z-score. The z-score describes the distance, in steps of standard deviation, to the mean value of a normal distribution of a dataset, see Figure 3.2. It is calculated by Equation 3.2, where Z represents the deviation from the mean; x is the raw data value; μ is the mean value; and σ is the standard deviation.

$$Z = \frac{x - \mu}{\sigma} \tag{3.2}$$

The outliers detected above $\mu + 3\sigma$ and below $\mu - 3\sigma$ are replaced with NaN in the datasets. Both pre-existing and added NaNs are replaced by filling in the mean value of the nearest actual value before and after the NaN. The Python code for this is in Appendix A.2. It is important to note that the outliers in the dataset might actually be legitimate data, and might represent the fault that the model is trying to predict [39]. It can thus be wrong to remove the outliers, and the effect of using processed and unprocessed data on the model accuracy will be tested in the Results Chapter. Outliers can also be caused by the acquisition system, where the sampling rate has a great impact on the data.

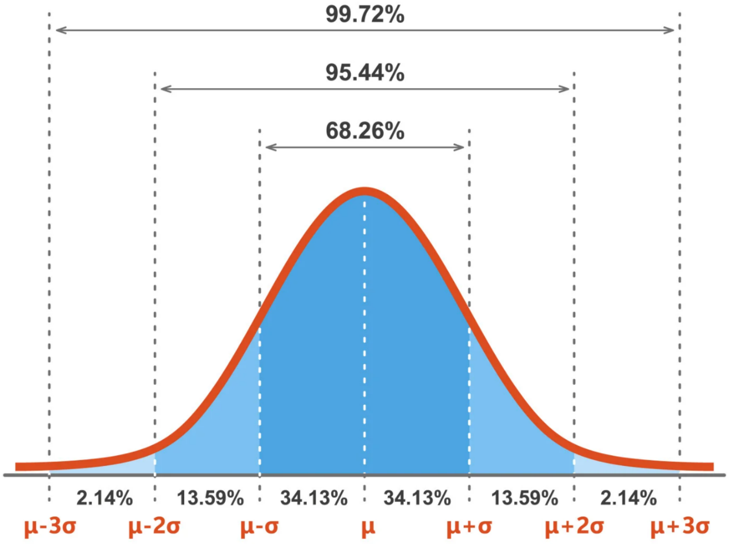


Figure 3.2. Visualisation of Z-score using the standard normal distribution [44].

Data Balancing and Variable Standardisation

The next pre-processing step will be balancing the datasets, to reduce the possibility of over-fitting or under-fitting the ML models. This is simply done by limiting the number of data points by the length of the shortest dataset, by using the `.iloc[:,len(df)]` function. The data should also be standardised or normalised before applied in the ML models so that one feature does not outweigh another. The features are thus scaled from 0 to 1 using `MinMaxScaler` from the `sklearn.preprocessing` library.

Noise removal

The removal of noise is performed in the frequency domain using a low or high-pass filter. As the datasets are not transformed into the frequency domain and noise removal will not be performed, this might remain a weakness of the thesis.

3.4 Feature Extraction

Data A and Data B do not have that many features, only the three-phase current signals. To gain more knowledge of the data, some statistical features will be calculated for the parameters. Feature extraction will be performed on the data in the time domain, and how they were calculated in Python is shown in Appendix A.3. The features that have been selected are shown in Table 3.2 [45], and as Data A consists of RMS values, fewer features will be calculated for Data A. The RMS value will be calculated for Data B as well, and additional features based on RMS and the raw signal will be calculated. To generate continuous values for the features and not a constant value for the entire variable, the `.rolling` function was used to calculate the different features for the 10 nearest rows of the target row. The variables in Table 3.2 are described in the Nomenclature chapter.

Table 3.2. Equations for Statistical Parameters

Parameter	Equation	Extracted in...
Absolute Mean	$\overline{X} = \frac{\sum_{i=1}^{N} \lvert x_i \rvert}{N}$	Data A and B
Standard Deviation (STD)	$X_{STD} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2}$	Data A and B
Skewness	$X_{skew} = \frac{\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^3}{(\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2)^{3/2}}$	Data A and B
Kurtosis	$X_{kurt} = \frac{\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^4}{(\frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2)^2}$	Data A and B
Crest Factor	$X_{crest} = \frac{x_{max}}{\sqrt{\frac{1}{N} \sum_{i=1}^{N} x_i^2}}$	Data A and B
Peak to Peak (P2P)	$X_{P2P} = x_{max} - x_{min}$	Data A and B
RMS	$X_{RMS} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} x_i^2}$	Data B
Clearance Factor	$X_{clear} = \frac{\max_{i=1}^{N} x_i - X_{RMS}}{X_{RMS}}$	Data B
Shape Factor	$X_{shape} = \frac{X_{RMS}}{\frac{1}{N} \sum_{i=1}^{N} \lvert x_i \rvert}$	Data B
Impulse Factor	$X_{imp} = \frac{x_{max}}{\frac{1}{N} \sum_{i=1}^{N} \lvert x_i \rvert}$	Data B

3.5 Signal Processing

For fault diagnosis in condition monitoring, some of the faults can be exposed by some simple signal processing. The method of signal processing used in this thesis is a very basic one, where different parameters are plotted to see if there are any visual signs of the fault. The fault is, as will be described in Chapter 4, applied to the second phase of the machine, and the signal processing will be used to see if this is indicated in the time domain. Preferably, this method would have been applied to the frequency domain, but there was not enough time for this. It is generally harder to spot the faults in the time domain, compared to the frequency domain, as faults often occur as frequency peaks or harmonics in the frequency.

The first calculation is on the difference between the three phases themselves, to see if the second phase differs from the others. The different parameters are Δa in Eq. 3.3, Δb in Eq. 3.4, and Δc in Eq. 3.5. I_1 represents I_{rms1} or I_a, I_2 represents I_{rms2} or I_b, and I_3 represents I_{rms3} or I_c.

$$\Delta a = I_1 - I_2 \tag{3.3}$$

$$\Delta b = I_1 - I_3 \tag{3.4}$$

$$\Delta c = I_2 - I_3 \tag{3.5}$$

The second part of signal processing applies simple calculations to view the differences between the phases of the healthy and faulty cases. The applied calculations are shown in Equation 3.6 and 3.7, where I is the current [A]; Φ represents the phases; and $\_H$ or $\_F$ denotes if the variable is from the healthy or the faulty data set, respectively.

$$\Delta I_{\Phi} = I_{\Phi\_H} - I_{\Phi\_F} \tag{3.6}$$

$$\Delta I_{\Phi} = \lvert (I_{\Phi\_H})^2 - (I_{\Phi\_F})^2 \rvert \tag{3.7}$$

3.6 Principal Component Analysis

PCA is used as part of the signal-processing part of the data, as a way to reduce the computation and number of features used in the ML algorithms. PCA is a dimensionality reduction technique, which falls under the category of unsupervised learning algorithms [46]. It differs from feature selection, where feature selection is selecting features extracted from a target value, and dimensionality reduction is extracting the most important information from the features. The goal is to reduce the dataset into a number of principal components. PCA uses an orthogonal transformation to transform the correlated features into uncorrelated features and applies a regression code to find out the best fit for the dataset. This will help ease the computational time for the future ML model, as it contains fewer features with the highest likeliness to provide statistically significant results. A visualisation of the PCA method can be viewed in Figure 3.3.

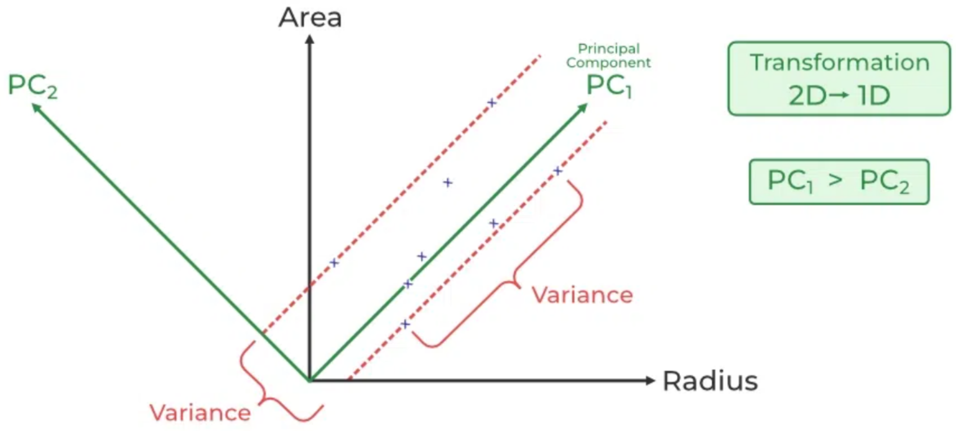


Figure 3.3. Visualisation of Principal Component Analysis (PCA) plot [46].

PCA is often used to prevent under-fitting that occurs due to an over-dimensioned dataset. As the goal is to perform a ML algorithm on the dataset, a PCA is used for dimensionality reduction, whereas the Auto Encoder method is usually used for Deep Learning algorithms. The Python code that is seen in Appendix A.4 is based on the tutorial in [46], and the theoretical background for the analysis is as follows:

The first step in PCA is to normalize the features into a standard scalar form, which in terms means that the mean value is 0 and the standard deviation is 1. For the PCA, this is done manually by Z-score, as shown in Equation 3.8. Z represents the new scaled value; X is the un-scaled value; $\bar{X}$ is the mean of the un-scaled value; and σ is the standard deviation of X.

$$Z = \frac{X - \bar{X}}{\sigma} \tag{3.8}$$

The second step is then to calculate the covariance of the features and detect how they are related. The principle here is to use the covariance as an indicator of how much information the variable contains, where a higher value represents more information. The formula for covariance can be found in Equation 3.9, where cov is the covariance of $x1$ and $x2$; x_{1i} and x_{2i} represent the ith observations of the variables; $\bar{x_1}$ and $\bar{x_2}$ represent the sample means of the variables; and n represents the sample size.

$$cov(x_1, x_2) = \frac{\sum_{i=1}^{n}(x_{1i} - \bar{x_1})(x_{2i} - \bar{x_2})}{n-1} \tag{3.9}$$

The third step is to calculate the eigenvalue and corresponding eigenvector, which will denote which PCA the value corresponds to. The number of PCA depends on the number of features and the level of covariance they have and is usually called PCA1, PCA2, PCA3, etc. The eigenvector is calculated as shown in Equation 3.10, where $\boldsymbol{A}\vec{v}$ is the matrix; λ is the eigenvalue of the matrix; and $\vec{v}$ is the eigenvector. The eigenvalue, *lambda* can be found by using the conditions for the eigenvector, which is stated in Equation 3.11. $\boldsymbol{I}$ is the identity matrix of matrix $\boldsymbol{A}$. The eigenvalue was gathered in Python using the `np.linalg.eig` code, extracted from the covariance matrix.

$$\boldsymbol{A}\vec{v} = \lambda\vec{v} \tag{3.10}$$

$$|\boldsymbol{A} - \lambda\boldsymbol{I}| = 0 \tag{3.11}$$

It is possible to calculate how many Principal Components are best for the code by finding the explained variance of the eigenvalues. This is done by taking the cumulative sum of the eigenvalues, that describes the covariance, and dividing them by the total sum of the eigenvalues. Then, a limit on the explained variance is set to 50 %, and the principle components with a variance above 50 % are selected. The number of components is calculated as described in the Python code in Appendix A.4. The PCA components are extracted from the eigenvectors in a matrix with [number of features, number of components] as the size.

The last step is to project the dataset by calculating the dot-product of the scaled variables and the found PCA components, as shown in Equation 3.12. The projection essentially takes a high-dimension dataset and projects it onto the Principal Components in a lower dimension. This can be portrayed as a single value between each feature and each Principal Component, usually visualised in a heatmap. In the projecting equation, $Proj_{P_i}$ is the projected value in the ith row; $\vec{u}$ is the unit vector; and $|u|$ is the length of the unit vector. If the number of components is more than two, this might be hard to visualise in a plot, but a typical result for two principal components is shown in Figure 3.3.

$$Proj_{P_i}\vec{u} = \frac{P_i \cdot \vec{u}}{|u|} \tag{3.12}$$

All of the steps above can be performed using the `PCA` package from the library from `sklearn.decomposition`. However, a manual approach is preferred, as the number of components in the sklearn-method is a setting, and the manual method will generate an indication as to how many is preferred. The output of the method will be a data frame with calculated PCAs, which can be inserted as the input of the ML models. The PCA code was applied to the dataset containing all the features from the previous chapter, and reduced into 3 and 4 PCAs for Data A and B, respectively.

3.7 Machine Learning Algorithm

The machine learning models can have a regression or classification-based approach. Regression is often used to predict continuous values, whereas classification is used to determine whether the data is one value or another, often binary. In this thesis, classification algorithms are preferred as the goal is to see if the values represent either Healthy or Faulty data. Four classification ML methods were chosen for this thesis - Decision Tree, Random Forest, k-Nearest Neighbours, and Gaussian Naive Bayes. The methods fall under the category of supervised learning, so they require a binary target value as described in the previous chapter. The four ML methods will be described in the following chapter, and then the chosen parameters will be described.

3.7.1 Target Variables - Data Labelling

The aim of the ML code is to see how well the model can determine whether something is healthy or faulty with the ITSC fault. The recorded data have only been flagged by which dataset it originates from, and it is desired to have some other indicators on the healthy and faulty data. Based on the features that were selected in an earlier step, such indicators can be created. Rows in the faulty dataset with values exceeding 95 % of the total mean value of the different features are flagged as faulty. This is done for each of the three phases and all of the features selected for Data A and Data B.

3.7.2 Decision Trees

Decision Trees create easy decision rules from the features and predict the target variable based on those rules [47]. A simplified example of this is shown in Figure 3.4 [48]. This method does not require a lot of data processing, although the pre-processing steps are applied to the data for this as well. This type of model is somewhat sensitive to biases, where scaling the features and balancing the datasets are important. Decision trees are also sensitive to outliers, as variations in the data can cause a new tree (classifier) to be created. The Python code for decision trees is based on `DecisionTreeClassifier` from the `sklearn.tree` library [49].

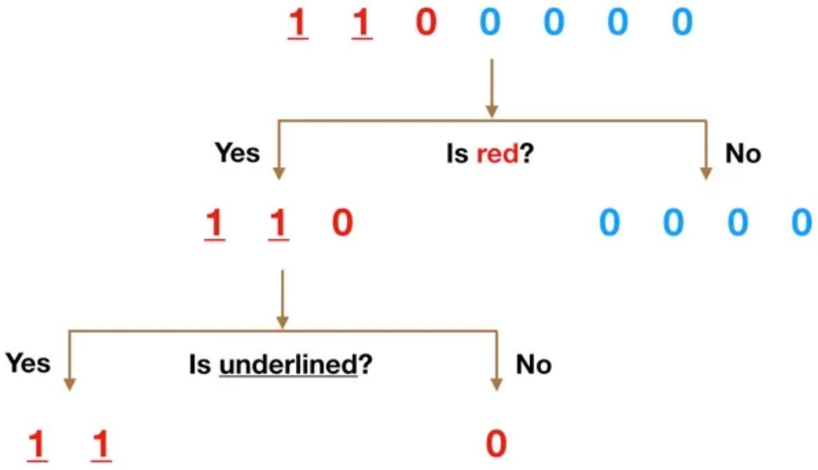


Figure 3.4. Example of how a decision tree model works [48].

3.7.3 Random Forest

Random Forest is an ensemble method, which means that the algorithm uses multiple predictors and combines them into one, which yields high robustness and accuracy [50]. In other words, the Random Forest model creates multiple decision trees that each creates predictors, and the prediction with the most number of votes are chosen as the final output. The Random Forest model is a perturbed-and-combined method, which means that the classifiers are created with randomness in their data baseline. These classifiers can often be a cause of over-fitting as the method is not generalized, so high accuracy in this model can be expected. Compared to decision trees, the ensemble method is not that sensitive to outliers, as the output is determined by the popular vote. The result from the Random Forest can also be statistically tested to see if the model over-fits the predictions. The Random Forest Python code uses `RandomForestClassifier` from the `sklearn.ensemble` library.

3.7.4 k-Nearest Neighbor

k-NN is a Nearest Neighbor method, where the model predicts the likelihood of the value to be in a group of data based on the group that the nearest neighbours are in [51]. The k-integer represents a chosen number of samples, which in this case, is 5 numbers of neighbours. k-NN differs from many ML methods as it does not try to create a generalized structure, but stores the most important information for the nearest groups. For the Nearest Neighbor classification methods, k-NN is preferred over Radius Neighbor as the sampling rate is constant. The neighbour groups are weighed uniformly, meaning that the neighbours contribute equally to the prediction. The k-NN model is affected by the dimensionality curse, which states that the larger the number of features in the data, the larger amount of input variables are needed to train the model accurately [39]. When the dimensionality of the data is very large, the distance to the nearest neighbour begins to resemble the distance to the neighbour farthest away. For this reason, it will be interesting to see if the PCA have an effect on the accuracy of the k-NN model.

k-NN is performed in Python using `kNeighborsClassifier` from `sklearn.neighbors` library [52]. By using the default algorithm, 'auto', the model chooses the most appropriate algorithm of the three available - 'ball_tree', 'kd_tree' or 'brute.

3.7.5 Gaussian Naive Bayes

The Gaussian Naives Bayes method is based on the Bayes' Theorem that states that there exists a conditional independence between the features, given the value for the class predictor [53]. The theorem is considered to be naive, hence the name, and the mathematical approach is given in Equation 3.13 where $P(x_i|y$ is the conditional probability; y is the class variable; x_i is the dependent feature vector; and σ_y and μ_y are parameters calculated using the maximum likelihood.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi{\sigma_y}^2}} exp\Big(- \frac{(x_i - \mu_y)^2}{2{\sigma_y}^2}\Big) \tag{3.13}$$

The Naive Bayes method does not need a lot of training data to create its classifiers and is generally a fast ML method. However, the estimators are not that good as the model assumes that all predictors are independent and that the probabilistic output should be used with care. The predictions are calculated in Python using `GaussianNB` from the `sklearn.naive_bayes` library [54].

3.7.6 Chosen Parameters in Python

The parameters chosen for the classifiers are shown in Table 3.3. The chosen parameters are standard inputs for the classifiers, and when using other parameters than the default ones, it is normal to use a code that will test and find the best fit of parameters. Grid search and random search are common methods for such parameter fitting. This would however require a lot of computational power and time, and since the code will run through a variety of features and target indicators, it was chosen not to apply such a code.

Table 3.3. The chosen parameters for the different Machine Learning (ML) models

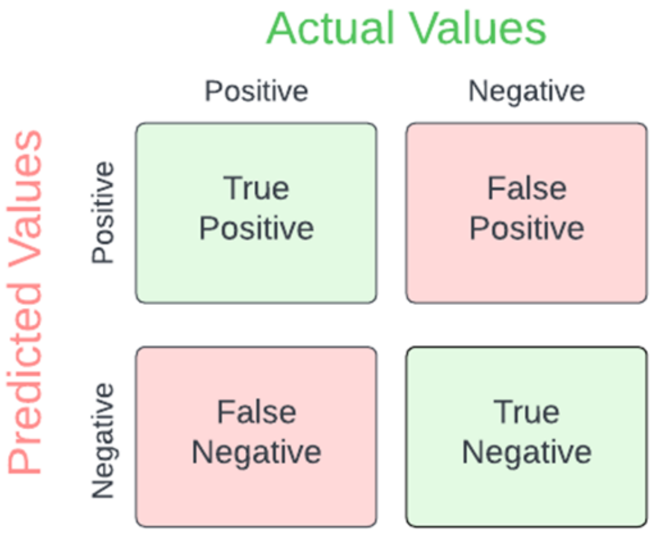
Parameters for Machine Learning Classifiers	
Random Forest [55]	(n_estimators=100, random_state=0)
k-Nearest Neighbour	(n_neighbors=5, metric='minkowski', p=2)
Naive Bayes (Gaussian)	(priors=None, var_smoothing=1e-9)
Decision Tree	(criterion='entropy', random_state=0)

3.7.7 Evaluating the performance

The main indicator for how well the models predict will be described by the accuracy. Accuracy is how well the model calculates actual faults and is calculated by Equation 3.14.

$$Accuracy = \frac{TP + TN}{N} \tag{3.14}$$

The accuracy is calculated by the output of the confusion matrix that is based on the training set of the ML model. The principle of a confusion matrix is shown in Equation 3.5, where, for instance, if the prediction states a fault, the prediction can be either an actual fault, True Positive (TP), or a False Positive (FP). The same goes for the Healthy prediction, where the model predicts either a True Negative (TN) or a False Negative (FN). N represents the total number of predictions, or $TP + FP + TN + FN$.



		Actual Values	
		Positive	Negative
Predicted Values	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Figure 3.5. Principle of a confusion matrix.

Chapter 4

Experimental Method

In this chapter, the experimental method of the in-house data collection of Data **A** will be explained. Firstly, the laboratory setup and the components will be described. The implementation of the short circuit fault will also be recounted from the research it was made for. Then, the process of aligning the setup will be presented, before discussing possible sources of errors that might occur in the data collection.

4.1 Laboratory setup

The data collection will be conducted in two collections using the experimental setup under the same conditions, one with a healthy PMSG and one with an ITSC. The purpose of the first set of data is to benchmark the generator, gathered from a healthy motor-generator system. The second set of data was collected from a generator with a short-circuited winding in the stator. This type of fault is also known as an inter-turn fault and is in this case modified to short-circuit 4.8 % of the stator winding. How the generator has been modified to have an ITSC fault will be further described in Chapter 4.1.2. A schematic of the complete setup and components can be viewed in Figure 4.1.

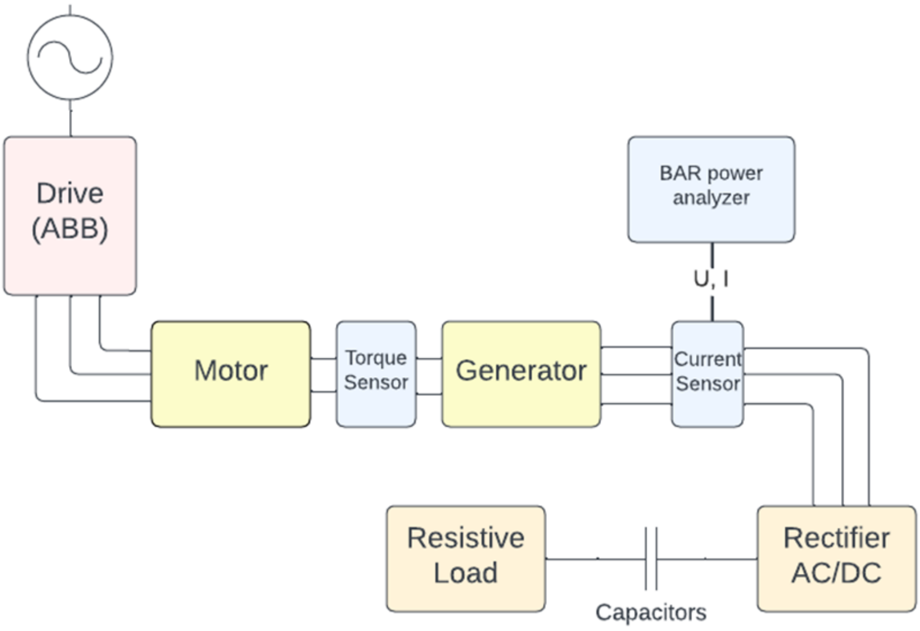


Figure 4.1. Schematic of the experimental setup.

A photograph of the experimental setup can be viewed in Figure 4.2. The setup consists of a PMSM (IE5-PS2R 90 L) (1) that is connected with flexible coupling to a generator (2) of the same machine. A torque sensor (3) is installed between the couplings, but data from this was not collected. Three HIOKI current sensors (4) measure the 3-phase current on the generator side, as well as the voltages. The current and voltage is measured with a Power Analyser (PW6001) (9), from which the data will be collected with a USB drive. The generator is connected to a rectifier (5) that converts the current from AC to DC, and sends the current through a set of capacitors. The capacitors are connected to a resistive load (7) that can be manually adjusted. To help ease adjusting the load, a multi-meter (8) is used to measure the resistance and to check if it is kept constant. The entire system is powered and controlled by an ABB drive (10) that is powering the motor. The entire setup is mounted on a T-bar bench that has been machined straight, and the plates that the PMSMs are mounted on are fitted to the T-bench.

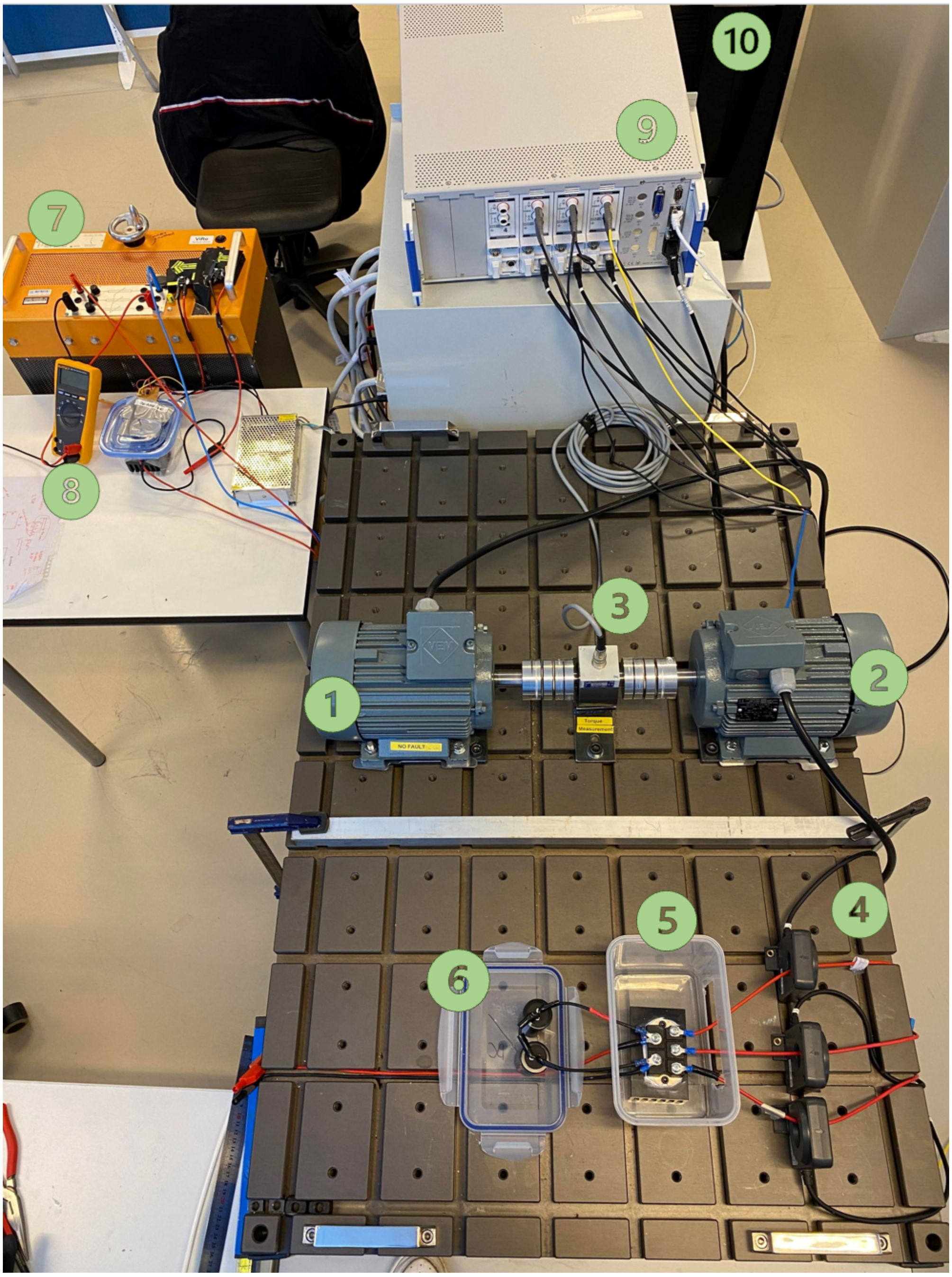


Figure 4.2. Photograph of the experimental setup for Data A.

4.1.1 Motor-Generator Configuration

The motor and the generator are both PMSMs, where one acts as a motor that drives the other one as a generator. The machine specifications can be seen in Table 4.1, and the faulty PMSM is shown in Figure 4.4. The generator and motor are star-connected, as shown in Figure 4.3, where the N terminals are shorted. The earth of the power analyser is placed in the N-terminal, which enables the recorded data to have the same frame of reference. A multimeter can be used between the phases and the N-terminals to check if the generator wiring is faulty. As described, the data is collected in a healthy setup as well as a faulty setup with an implemented ITSC fault.

Permanent Magnet Synchronous Machines	
Model name	IE5-PS2R 90 L
Number of poles	4
Output power	2.2 kW
Nominal current	5 A
Nominal voltage	280 V
Nominal speed	3000 rpm
Nominal torque	7.0 Nm
Phase resistance	0.8 ohm
Direct axis inductance	6 mH
Quadrature axis inductance	6 mH
Nominal efficiency	90.2%
Weight	18 kg

Table 4.1. Parameters of the PMSM.

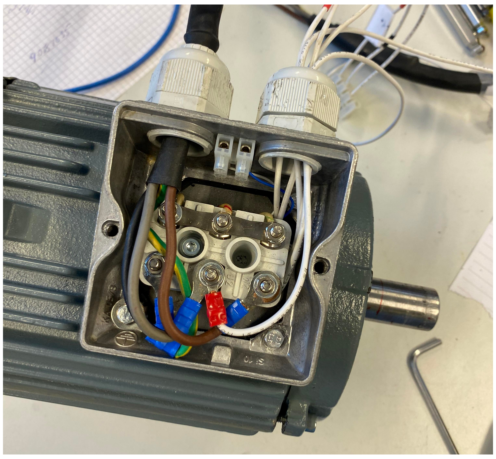


Figure 4.3. The star connection of the PMSM, here with the ITSC-fault.

To prevent faults in the setup that are due to misalignment, eccentricity or other common installation-caused faults, the assembly of the setup is important. Although the couplings are flexible, a high level of accuracy is wanted to obtain as good data as possible. The motor setup can be aligned by, for instance, a shaft alignment tool (like the SKF TKSA 11 [56]), dial gauge, manual measurements, shims and lasers. The process of aligning the axles of the motor, torque sensor and generator will be further explained in Chapter 4.2.

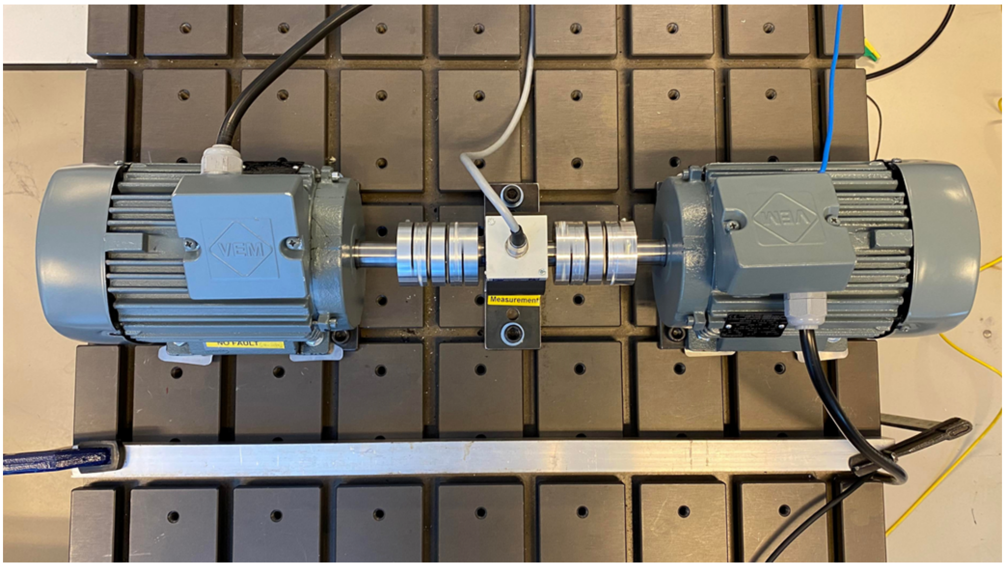
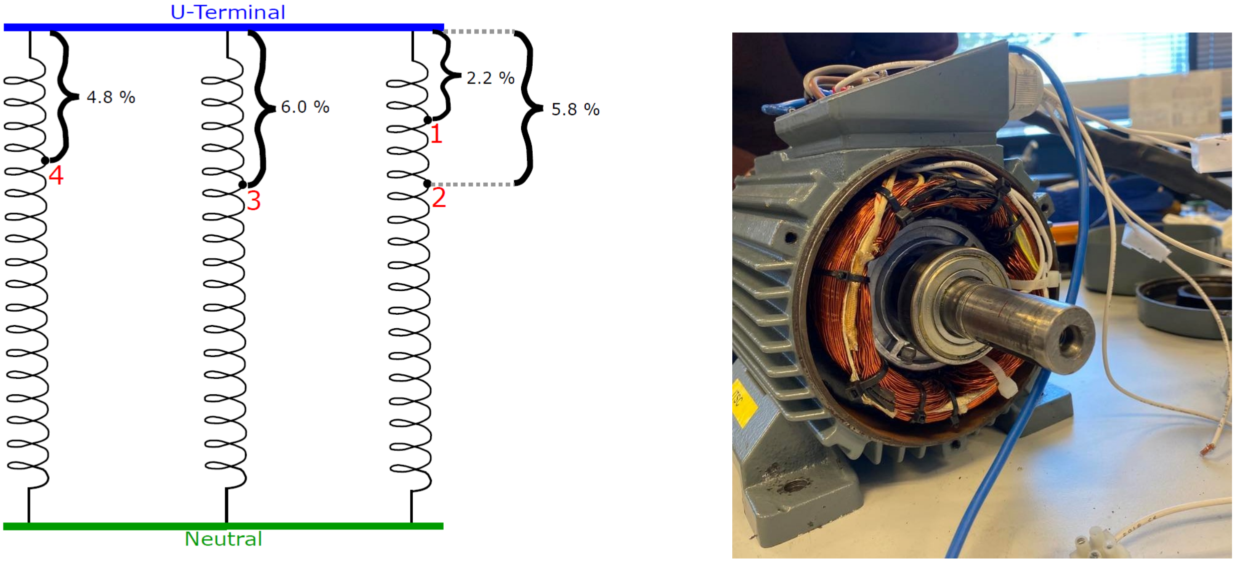


Figure 4.4. The setup of the IE5-PS2R 90 L motor and generator.

The setup is controlled by a manual load regulator and the ABB drive. The setup is run at constantly 500 rpm, regulated at the ABB drive. The speed was chosen as hydropower generators are usually low-speed machines. 500 rpm is the equivalent rotational speed to a 12 pole generator delivering 50 Hz to the grid, as per Equation 3.1. The electrical machines used in this setup have 4 poles, and will at 500 rpm deliver 16 Hz from the generator, running at approximately 17 % of the nominal speed. Running such an experimental setup at lower speeds will also reduce the need to perfectly align the setup, as the potential eccentricity and vibration will not be substantial.

4.1.2 Inter-Turn Short-Circuited Machine

The PMSG that have been short-circuited in the stator winding have been modified for previous research done at the university for a PhD dissertation [43]. Each phase of the PMSM consists of 3 windings connected in parallel. The short circuits are implemented in the U terminal, which is connected as the second phase to the power analyser. A total of 4 taps were installed in the windings, as shown in the schematic in Figure 4.5a). The exposed stator with the ITSC fault is shown in Figure 4.5b).



(a)

(b)

Figure 4.5. (a) The ITSC fault implementation at the U-terminal. (b) The PMSM with an exposed stator and ITSC fault.

The severity of the ITSC fault is calculated by Equation 4.1, where $\mu_{f,ITSC}$ is the severity of the ITSC fault, $n_{shorted}$ is the number of shorted turns, and n_{tot} is the number of total windings. As illustrated in Figure 4.5a, four taps were installed to yield ITSC fault severities of 2.2 %, 4.8 %, 5.8 %, and 6.0 %. The setup for Data **A** selected a ITSC severity of 4.8 %, whereas Data **B** was selected to be 6.0 %.

$$\mu_{f,ITSC}[\%] = \frac{n_{shorted}}{n_{tot}} \tag{4.1}$$

4.1.3 Sensors and Power Analyser

In the setup, three types of signals can be extracted - current, voltage and torque. The torque sensor will not be used in this collection, and the voltage and current data is measured by the HIOKI (CT6862-05) sensor. The parameters of the sensor can be found in Table 4.2. It is important that the sensors are positioned with the arrows towards the load, and ideally that the wires of the three phases are in the middle of the holes.

	Current Sensor
Model	HIOKI CT6862-05
Supply voltage	2 V
Measuring range	± 19.2 A
Sensitivity	104.16 mV/A
Linearity error	<0.1 %
max. temperature	80 °C
min. temperature	-40 °C

Table 4.2. Parameters for the HIOKI sensors [57].

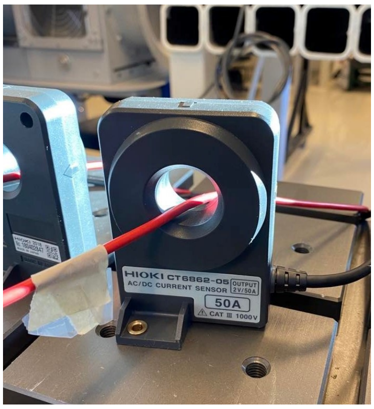


Figure 4.6. The HIOKI current sensor (CT6862-05) [57].

The current and voltage sensors are connected to the HIOKI Power Analyser (PW6001) [58]. The Power Analyser is measuring under the setting of 3P4W, which in short terms dictates that the average and RMS-values are calculated based on all three phases [57]. The current sensor is high-accuracy measurement equipment, so the limiting factor in terms of sampling rate will be the Power Analyser. With the current and voltage values as the input, the Power Analyser can calculate a range of other parameters, but these are not used further.

4.1.4 Load Regulator

As the setup is run at a constant speed, the load regulator can be a simple configuration and a general schematic of the circuit is shown in Figure 4.7. The three-phase current is sent through a full-bridge rectifier (IXYS VUO82-16NO7) that converts the signal from AC to DC, see Figure 4.7. Two $1000\mu F$ capacitors are connected to the rectifier at the output terminals and removes possible ripple voltages in the system. The rectifier is connected to the $3.3kW$ load resistor bank (MV1100), which can be manually adjusted. For the data collection, a load of 50 % torque is chosen. The resistor bank is monitored by a multi-meter during the data collection to ensure that a 50 % torque is obtained. From other measurements, this is achieved with a resistance of 22.5Ω.

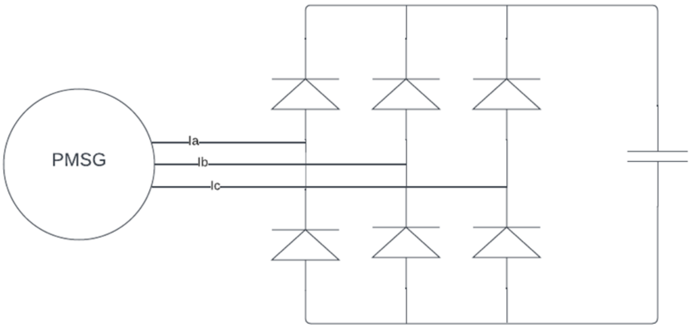


Figure 4.7. Schematic of the rectifier in the experimental setup.

An electric load regulator would be more precise than a manual regulator like the one used in the setup. However, this would require a change in the setup where everything would be controlled through a computer via the MicroLabBox. As the setup is quite simple and is ran at constant speed and load, a manual load regulator controlled by the ABB drive and a multi-meter is enough to control the load and keep the protective system of the generator to kick in.

4.2 Aligning the Setup

To collect the data from the experimental setup, the setup needs to be aligned and ready for a long run. Although there was a pre-existing motor generator setup, the previous generator and couplings needed to be replaced. With the new parts, the setup had to be remade as the new couplings were longer than the previous ones. Utilising the T-bar bench's slots and T-nuts to fasten the components to the table, a degree of freedom in the axial direction is obtained. Due to the difference in the size of the slots and nuts, some degree of freedom does, unfortunately, exist in the trans-axial direction as well. This increases the importance of aligning the setup precisely, but also makes it easier to do so.

4.2.1 Flexible Couplings

The couplings were standard Ø24H7 flexible couplings from VEM motors that fit the Ø24 axles of the motor and generator. Due to some tolerance differences between the axles and the internal diameter of the couplings, the couplings were a tight fit on the motor side of the setup. A tight fit is great to prevent slip of the axles, but makes the assembly somewhat tougher. To install the tight couplings, a custom-made wedge tool was used to widen the internal diameter of the coupling, see Figure 4.8, and assembled on one side of the axles. For both the motor and generator side of the setup, the couplings were first installed, with no more than 10Nm torque, on either the machine axle or the torque axle. Then the tool was used on the other side of the coupling and pushed onto the other axle.

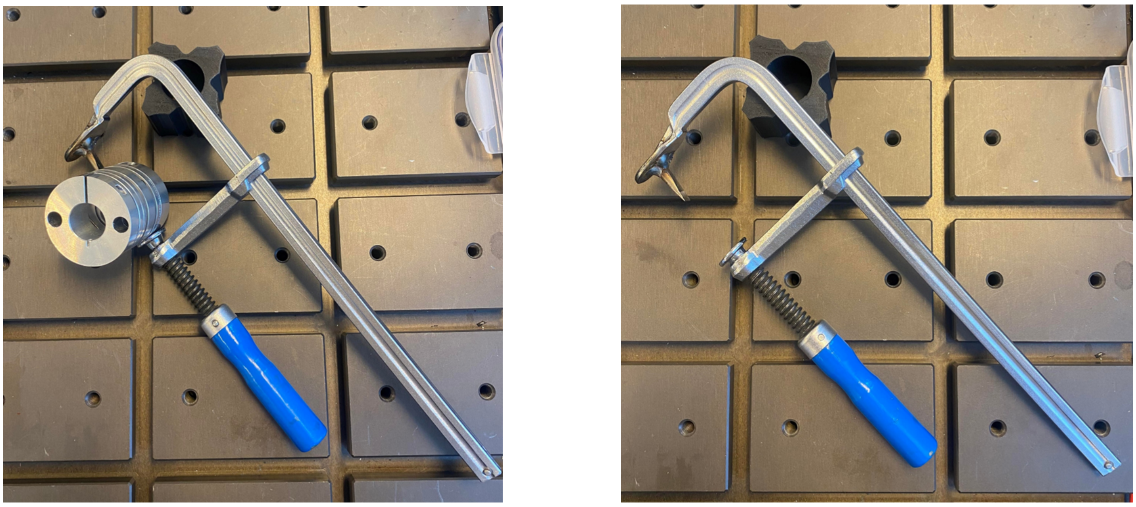


Figure 4.8: Tool specially made to open tight coupling.

4.2.2 Permissible Misalignment

For the setup, the permissible misalignment of the flexible couplings will be used to indicate if the setup is aligned correctly. These tolerances can be viewed in the datasheet for the couplings in Appendix B. The types of misalignment are visualised in Figure 4.9 [59], and shows axial offset, radial offset, angular tilt, and synthesized misalignment. The flexible couplings can tolerate up to 1.2 mm axial offset, 0.27 mm radial offset and 1.0 °of angular tilt. These are the parameters that will be measured and adjusted in the setup, according to the permissible limits.

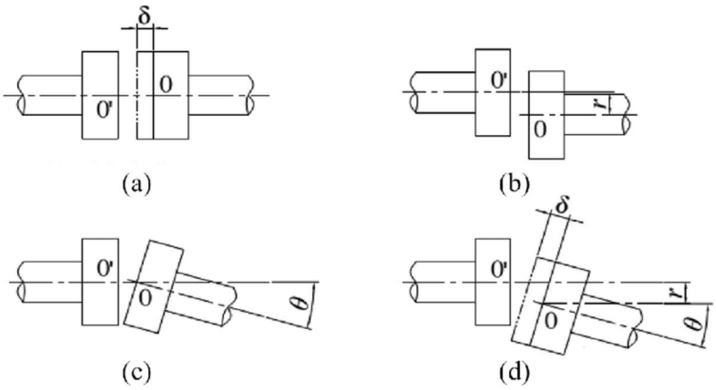


Figure 4.9. Misalignment in a) axial offset b) radial offset c) angular tilt d) mixture of all (synthesized misalignment) [59].

Precise and modern tools like the SKF Shaft Alignment Tool TKSA 11 [56] yield accurate and quick alignment of the axles. However, the TKSA does not fit between the coupling and torque sensor house, and therefore, other in-house methods had to be used to align the setup. The primary measuring tool to align the setup is a TESATEST dial test indicator that measures with a 0.01 mm precision and a 0.8 mm range. A calliper and regular measuring tape were also used. As the T-bench is machined straight, its slots can be used as a benchmark for the TESATEST dial gauge. To align the setup with the tracks of the table, an angle piece made of extruded straight aluminium was fastened along the table tracks, as seen in Figure 4.10. This angle piece will be used as a benchmark to slide an arm with the gauge clock along the axles. It should be noted that the measurements are taken with varying types of tools, with varying uncertainties. However, due to the size of the permissible margin of error in the couplings, this will not be much of an issue. A complete description of the measuring method will be given in the list below and will describe how the parameters of the permissible misalignment are measured and calculated:

Before assembling the couplings:

1. **Radial Offset (height):** measured by pushing the axles together and measuring the height difference between the machine axle and torque sensor axles with the TESATEST dial gauge, see Figure 4.10. A radial offset higher than the permissible limit is adjusted with shims between the four corners of the motor and the mounting plate it is placed upon.

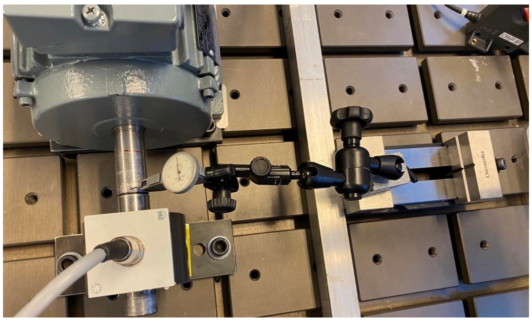


Figure 4.10. Setup of measuring the radial offset with the TESATEST dial.

2. **Angular tilt (horizontal plane):** Pushes the machine back to about the position it will have when the setup is complete, and measures the height difference with the TESATEST dial at the machine axle and the torque sensor axle. The distance between the two points is also measured. The angle of the tilt is calculated by trigonometry as

in Equation 4.2, where θ is the angular tilt; Δx is the height difference; and L is the distance between the measuring points.

$$\theta = sin^{-1}\left(\frac{\Delta x}{L}\right) \tag{4.2}$$

A tilt higher than the permissible limit is adjusted with shims on either the front or back part of the machine.

After assembling the couplings:

3. **Angular Tilt (vertical plane):** Now that the couplings are installed, the tilt will be measured by turning the axles half of a rotation and measuring the difference at two points on the axle. The TESATEST measures the difference of the placement during the turning (see Fig. 4.11), and the distance between the measurements are also noted. The angular tilt is calculated by Equation 4.3, where θ is the tilt angle; Δm is the machine-side difference; Δt is the torque-side difference; and L is the distance between the measurements.

$$\theta = sin^{-1}\left(\frac{\Delta m - \Delta t}{L}\right) \tag{4.3}$$

A tilt higher than the permissible limit is adjusted by gently banging on the side of the machine in the opposite direction of the tilt.

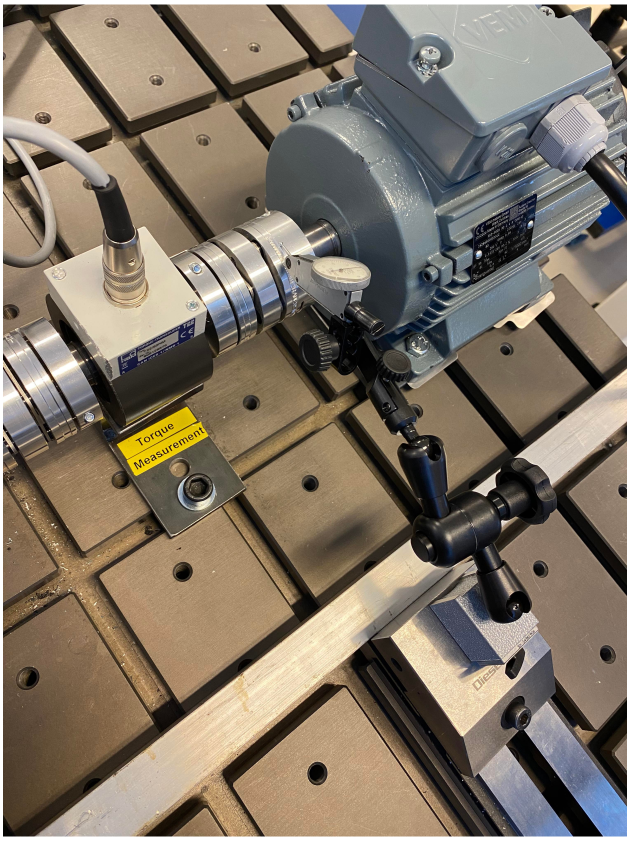


Figure 4.11. Setup of measuring the angular tilt in the vertical plane using the TESATEST dial.

4. **Axial Offset:** The axial offset is found by measuring the length of the couplings before and after the instalment. If the axial offset is bigger than the permissible limit, the axial position of the machine is adjusted, and step 3. and 4. is repeated.

Table 4.3. The permissible misalignment of the couplings and the measured values.

Alignment of experimental setup			
	Permissible Misalignment	Motor-Side	Generator-Side
		Healthy | Faulty	Healthy | Faulty
Radial Offset	0.27 mm	0.18 mm | 0.18 mm	0.07 mm | 0.1 mm
Angular Tilt, Horisontal Plane	1.0 °	0.226 °| 0.226 °	0.08595 °| 0.076 °
Angular Tilt, Vertical Plane	1.0 °	0.0596 °| 0.0596 °	0.143 °| 0.143 °
Axial Offset	1.2 mm	0.2 mm | 0.2 mm	0.3 mm | 0.03 mm

The tightening and loosening of the PMSMs to the table with T-bolts affected the measurements noticeably. During all of the measurements, the PMSMs are tightened to the table, so that the measurements are similar to the finished assembly. Rotating the axles during the alignment can also indicate faults by clicking noises or visual inspection. These methods can yield somewhat inaccurate results, but will give an indication of the alignment corrections that are needed. The final alignment measurements of the healthy and faulty setup can be viewed in Table 4.3.

4.3 Running the Setup

When the setup is within the permissible misalignment, the next step is data collection. The aim is to have an hour's worth of data from both the healthy and the faulty setup. The healthy setup is run for an hour straight while monitoring indicators of faults such as overheating, vibration and noise. The faulty setup is run for 10 minutes for six rounds, with 10 minutes breaks in between. The breaks are necessary as the setup has a 4.8 % ITSC fault, which can create some heating and/or asymmetry in the flux field, which can potentially ruin the insulation and create more, uncontrolled, insulation-related faults. In a worst-case scenario, the entire PMSG could fail, but this is not expected as the severity of the fault is quite small and the speed is low. The wanted variables are selected in the Power Analyser settings and automatically uploaded in a pre-made file in a connected USB drive.

4.4 Sources of Error

One of the largest sources of errors in this thesis lies in the data collection, or more specifically - the instalment and assembly of the components. The setup is deconstructed between the healthy and the faulty data collection, and differences and errors between the disassembly and re-assembly can easily occur. For instance, potential shaft misalignment should be similar for both cases, as the potential differences in the data might not be apparent, and can thus be misinterpreted as a short-circuit error. Another aspect of ensuring good data collection is the quality of the resulting data. Good data quality will lessen the need for data-processing, and therefore make the ML algorithm work more efficiently.

Another source of error in the instalment is the electrical wiring. Wires might be wrongly connected, but this is often noticeable in the data acquisition system as signals that are not as expected. The plugs or clamps of the wires can also be loose, which in most cases will show up as a signal fault as well. The cables and wiring might be different lengths and/or lifespans due to the cable type, which can cause the recorded phases to be slightly different. The length, type and age of a cord affect the resistance in the cord, as described in Equation 4.4. R_{cable} is the resistance of the cable [Ω]; ρ is the resistivity of the cable; L is the length; and A is the area.

$$R_{cable} = \rho \frac{L}{A} \tag{4.4}$$

In a setup with different wires, differences in the amplitudes of the three phases that are within the mA range are not concerning and are normal. Phase-amplitude differences can also be caused by the diode resistance in the rectifier connected to the load regulator. The rectifier for the three-phase circuit contains 6 diodes, which are not perfectly equal, and will thus apply different resistances to the three phases. In summation, differences in the amplitudes of the three phases of the generator might stem from different resistances in the setup.

The recorded values of the phases are also affected by the usage of the sensors. The three phase wires are placed inside the holes of the HIOKI sensors, as seen in Figure 4.6. These wires should not be in tension and should be in the middle of the hole. The current sensor is considered a high-accuracy measurement equipment so the usage might be noted in the recorded data. However, the power analyser has a sampling frequency of 100 Hz (equivalent to a sampling rate of 10 ms) [58], whereas the current sensors have a sampling frequency of 1 MHz (equivalent to 1 μs) [57]. The limiting factor will thus be the power analyser's sampling rate, and differences in the phases due to misuse of the current sensors will, most likely, not be noticed.

Lastly, there is the potential for the setup to be misaligned even though the permissible misalignments are upheld. Depending on the severity of the misalignment, it can develop vibrations and eccentricity-related faults. However, a misalignment issue will often be noticed as vibration or noise during the data collection. In addition, the system is not run for a long period of time and at low speed, so the impacts of these potential sources of errors are significantly reduced compared to real-life usage, such as in hydropower stations.

Chapter 5

Results and Discussion

The results of the performed data analysis described in Chapter 3 are presented in this chapter, and will be discussed as they are presented. The results will be presented for Data A and Data B separately, and compared in broader terms later in the chapter. Firstly, the pre-processing step will be presented, where the raw data will be inspected and the outlier detection method will be discussed. Then, with some simple signal processing, the data will be evaluated for visual fault detection. The findings of the Principal Component Analysis (PCA) for Data A and Data B will be presented and discussed before the final results of the Machine Learning (ML) models will be evaluated.

5.1 Pre-processing of Data A

The pre-processing step consists of inspecting and correcting the raw data, and then removing and filling the detected outliers.

5.1.1 Inspecting the Raw Data A

A three-phased current from a Permanent Magnet Synchronous Generator (PMSG) behaves in a sinusoidal wave, and the effects of the transformation to RMS-value are displayed in Figure 5.1. As the RMS calculated the value for a chunk of the data, the plots will not resemble a rhythmic sinewave.

The first step of cleaning the data is inspecting the raw data and removing parts that look out of place. While inspecting Data A, some of the recorded data have an offset in the y-axis that is very clear, as seen in the top plot in Figure 5.2. The offset between the healthy and faulty data will not be edited, whereas the offset in the healthy data between index point 147 000 to 178 490, and 328 000 to 330 512 will be removed. The first 10 minutes of the faulty data will also be removed as it has somewhat lower values than the rest of the dataset, as the aim is to have a seamless time series of the healthy data recording. The values are removed by refraining to import the first .csv file of the faulty data recording.

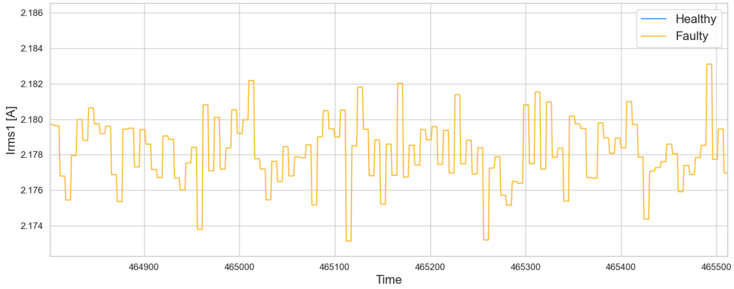


Figure 5.1. The raw data of 'Irms1' in Data A, zoomed in.

The raw data of ’Irms1’ before and after removing the data points can be viewed in Figure 5.2. The ’Irms2’ and ’Irms3’ plots are not shown as they have the same offsets as ’Irms1’. The difference in the mean value between the healthy and faulty datasets is approximately 0.02 A. This difference is most likely due to the sources of error described in Chapter 4, and is not assumed to be an indication of the ITSC fault. The length of the datasets is now uneven for the two datasets, and to ensure balanced datasets, the longest dataset will be limited to the length of the shortest.

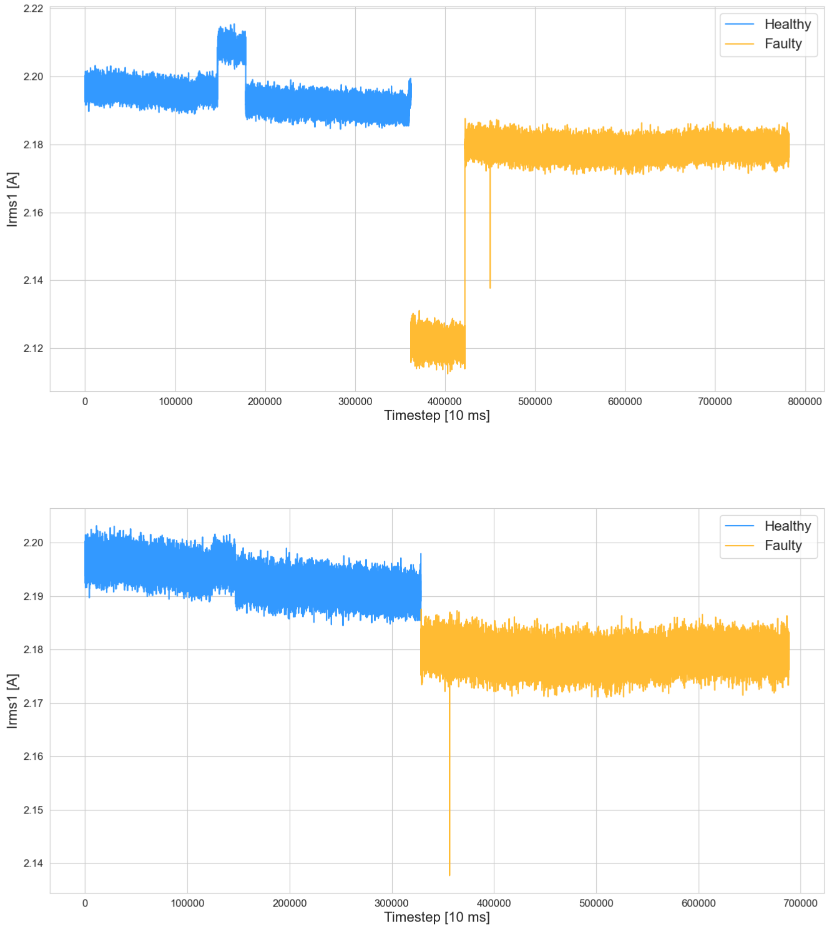


Figure 5.2. The raw Data A for ’Irms1’ before and after the initial cleaning.

The currents seem to decrease slightly over time for the healthy dataset, although the change is minuscule. This is most likely due to the temperature increase inside the Permanent Magnet Synchronous Motor (PMSM) during the data gathering, as the cooling from the machine fans is not as effective at lower speeds. The current of the faulty data does not seem to decrease over time as much as the healthy data. The faulty data recording had pauses in between the short recordings, whereas the healthy is gathered in one sitting.

As a result of the difference in the current values between the healthy and faulty data, the normal distribution of the combined dataset will have two peaks (see Figure 5.3). The next step is the removal of the outliers using the Z-score, and as the normal distribution of the data has two peaks, the Z-score method will detect more outliers than it should. This is a result of the centre of the distribution will not be at the peaks, and will thus detect a wrong amount of data outside the limit. For this reason, the Z-score needs to be applied to the faulty and healthy datasets separately.

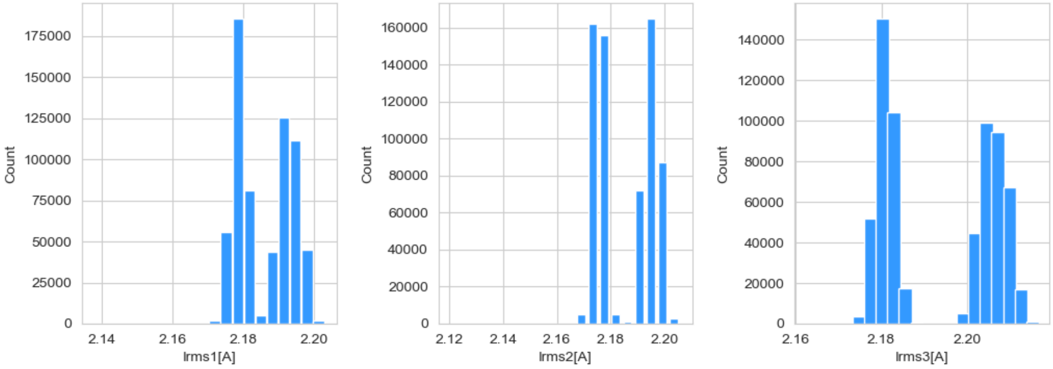


Figure 5.3. The normal distribution of combined dataset of healthy and faulty for Irms1, Irms2 and Irms 3 of Data A.

5.1.2 Outlier removal

The method of removing the outliers in the data is using the Z-score, and was described in Chapter 3. The data before and after filing the outliers with NaNs is displayed for 'Irms1' in the healthy (top) and faulty (bottom) dataset in Figure 5.4. The plots for 'Irms2' and 'Irms3' can be viewed in Appendix C, and are not shown due to the similarity to 'Irms1'.

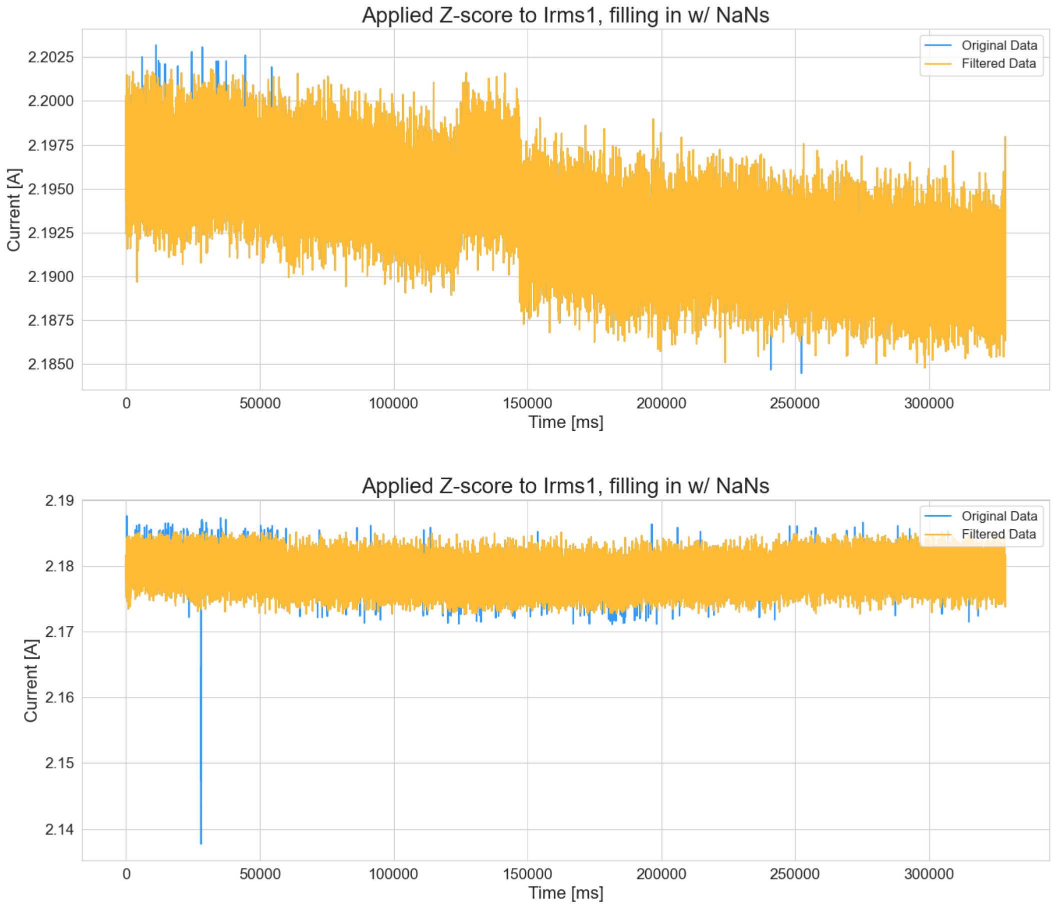


Figure 5.4. Data A before and after applying Z-score filtering on 'Irms1'.

Upon first inspection, it seems like the faulty dataset has more data points filtered with the Z-score method than the healthy one. This is also the case for the other phase-currents, and the suspicion is further confirmed by the number of NaNs documented in Table 5.1. If the Inter-Turn Short-Circuit (ITSC) fault is represented in the data that was removed as outliers, the ML models can have a hard time predicting the faults as they are replaced. Later in the result, the models will be applied to the pre-processed data and compared to the unprocessed data, to see if this might affect the accuracy of the models.

Table 5.1. NaNs detected for Data A using Z-score.

NaNs detected using Z-score : Data A		
	Healthy	**Faulty**
Irms1	90 NaNs	852 NaNs
Irms2	85 NaNs	692 NaNs
Irms3	102 NaNs	937 NaNs

5.2 Pre-Processing of Data B

The results of the first step in the data analysis have been performed for the RMS data in Data A, and the pre-processing results for the signal data in Data B will now be presented.

5.2.1 Inspecting the Raw Data B

The healthy and faulty data for Data B was uploaded in Python using the `scipy.io` library and `loadmat` function. The entire combined raw data for 'Ia' is plotted in Figure 5.5, as well as a zoomed-in version in Figure 5.6. Data B was collected from a Permanent Magnet Synchronous Motor (PMSM) and the MicroLabBox data acquisition system, which generated three-phased signals at a much higher sampling rate than Data A. The mean of the signal lies at around 0 A, contrary to Data A which has a mean value of around 2.18 A. For some ML models, having a signal with a mean value of 0 might help establish a prediction, as this can simplify the patterns that the model bases its prediction on.

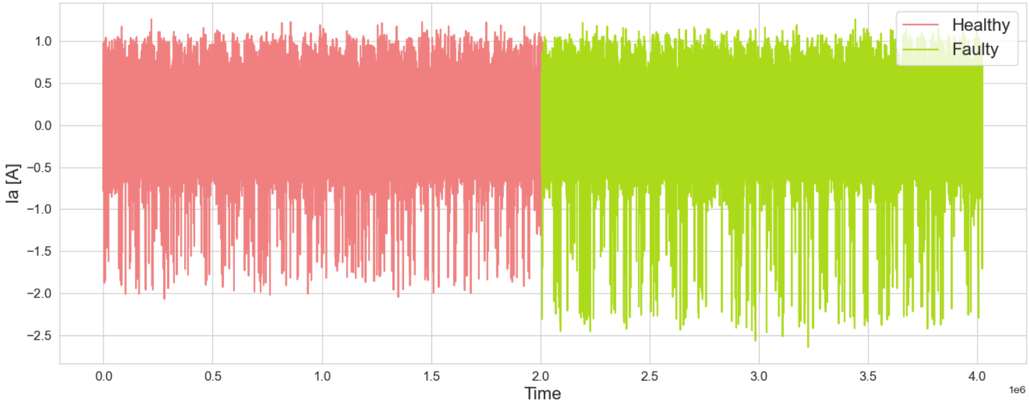


Figure 5.5. The raw data of 'Ia' from Data B for the entire dataset.

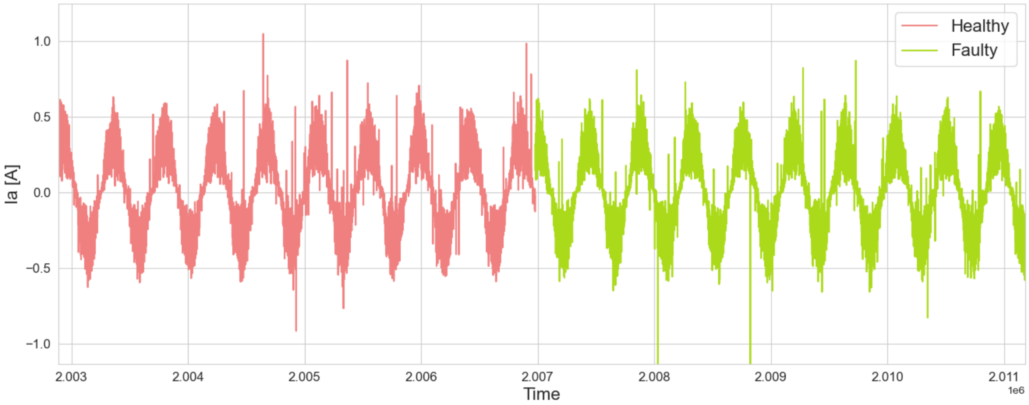
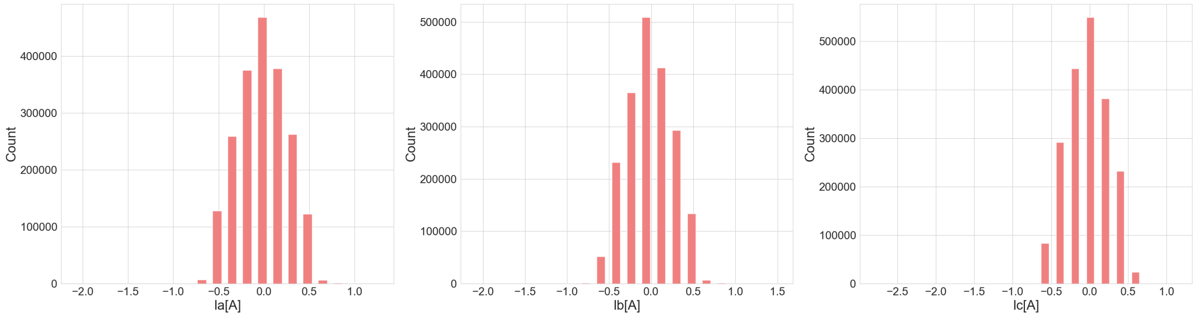


Figure 5.6. The raw data of 'Ia' from Data B, zoomed in.

Looking at the normal distribution of the combined healthy and faulty dataset, in Figure 5.7, the distribution has only one peak and thus the outlier removal can be performed on the combined dataset. Furthermore, the dataset does not have any obvious outliers that should be removed manually or any abnormalities in the normal distribution graph.



(a) (b) (c)

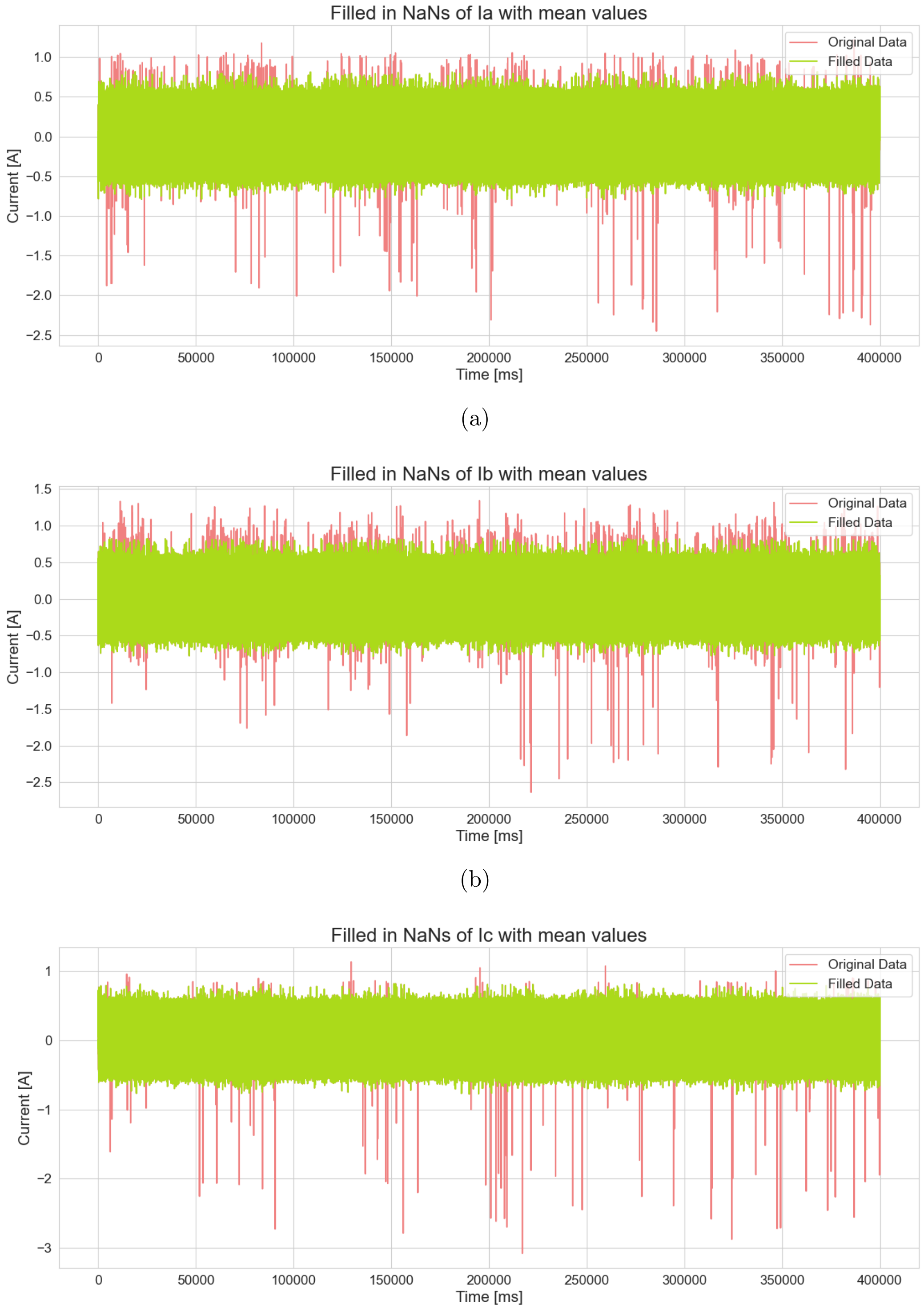
Figure 5.7. The normal distribution of the combined Data B for (a) Ia, (b) Ib, and (c) Ic.

5.2.2 Outlier Removal

The removal of the outliers was applied to the combined dataset of Data B using the Z-score method. The results are shown in Figure 5.8, and the number of replaced outliers are shown in Table 5.2. Comparing how many data points were replaced with the Z-score method, Data B has 1 outlier for each 456^{th} data point, whereas Data A has 1 outlier for each 236^{th} data point. This can be an indication that there are more sources of errors present in Data A than in Data B, which can stem from the errors in the experimental setup, the data acquisition system. It is also possible that Data A is more sensitive to the Z-score outlier detection method.

Table 5.2. NaNs detected for Data B using Z-score.

NaNs detected using Z-score : Data B	
Ia	301 NaNs
Ib	460 NaNs
Ic	113 NaNs



(a)

(b)

(c)

Figure 5.8. Data B before and after applying Z-score filtering for (a) Ia, (b) Ib, and (c) Ic.

5.3 Signal-Processing Data A

The signal-processing of Data A was executed as described in Chapter 3.5. The aim is to see a clear ITSC fault implemented in the second phase through some visualisation of the phases.

5.3.1 Differences of the Irms1, Irms2, and Irms3

Figure 5.9 shows a small time frame of the three phases of Data A, and the phases do not alternate around the same baseline. The phase in Data A that differs the most from the others, is 'Irms3'. This should not be interpreted as the ITSC fault, as it is implemented in 'Irms2', and is most likely due to the different resistances in the cables, diodes and equipment as described in Chapter 4.4 in the Experimental Setup. The fault is thus not apparent in the raw data, and the calculated difference between the phases will be looked at next.

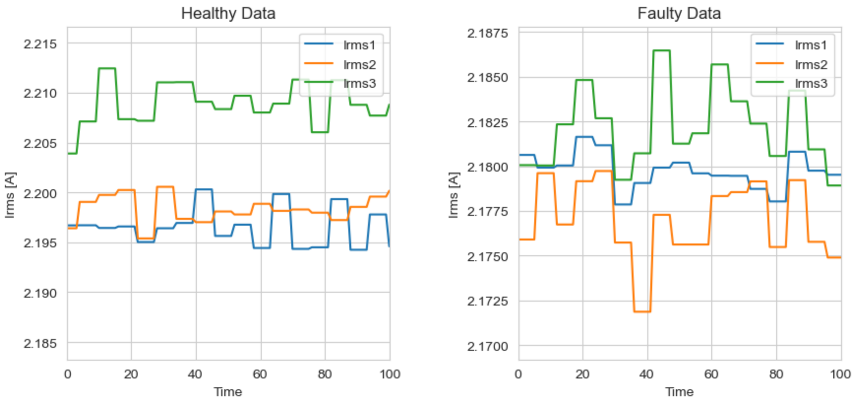


Figure 5.9. Detailed view of the three phases of Data A.

Plotting the calculated Δa, Δb, and Δc as described in Equations 3.3-3.5, respectively, yields the plots for the healthy and faulty datasets in Figure 5.10. Here too, there is not much difference between the phases in either dataset. Δb and Δc have higher values than Δa, which is due to 'Irms3' being larger than the other two phases. The ITSC fault would be evident if Δb differed from the rest, as this parameter is the only one without the second phase. The phases are more similar for the faulty dataset, and thus no fault is detected from these calculations either.

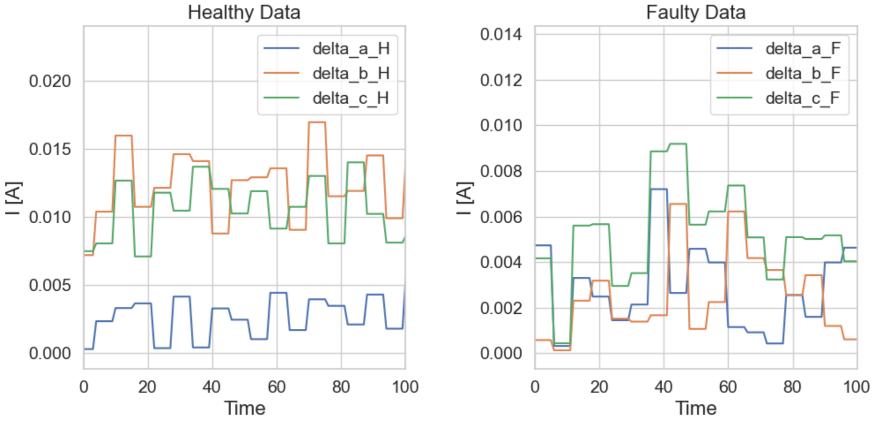
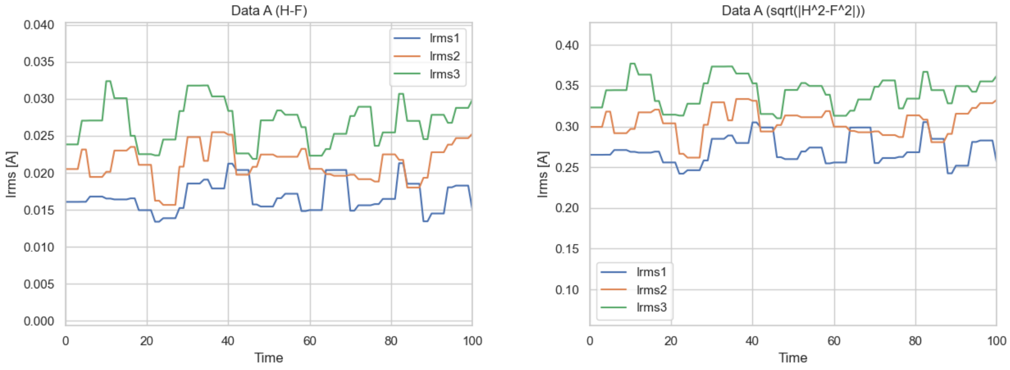


Figure 5.10. The differences between the three phases of Data A in the healthy case and faulty case

5.3.2 Calculated differences

The last step of the signal processing of Data A is to observe some calculated differences between the healthy and faulty phases. The plots for Equation 3.6 and Equation 3.7 are shown in Figure 5.11. Both equations yield similar-looking plots. The second phase does not stand out compared to the other two, and so this signal processing step for Data A did not yield fault detection either. A mathematical detection method could have been applied, but as the signal-processing calculations are simple, it is not likely that another mathematical or statistical approach would detect the fault.



(a) (b)

Figure 5.11. (a) Equation 3.6, and (b) Equation 3.7, applied to the healthy and faulty datasets of Data A for each phase.

5.4 Signal-Processing Data B

The signal-processing of Data B was correspondingly executed as described in Chapter 3.5, with the aim to see a clear ITSC fault implemented in the second phase.

5.4.1 Differences of the Ia, Ib, and Ic

Figure 5.12 shows the three phases of Data B, all alternating around 0 A. From this plot, the phases have no obvious differences as all of them follow the same path with 120°between the signals. Data B should have been converted to the frequency domain and inspected there, as the figure shows a lot of noise in the signal.

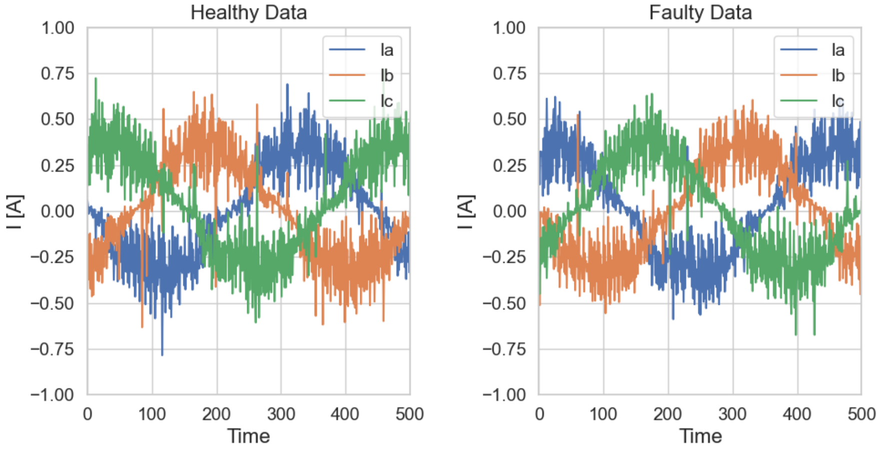


Figure 5.12. The three phases of the healthy and faulty Data B datasets.

Nevertheless, the same procedure as the signal-processing of Data A was performed for Data B, starting with calculating Δa, Δb, and Δc. The calculated values are shown in Figure 5.13 for the healthy and faulty datasets. The findings for Data B are the same as Data A so far, where no apparent fault is detected between the phases.

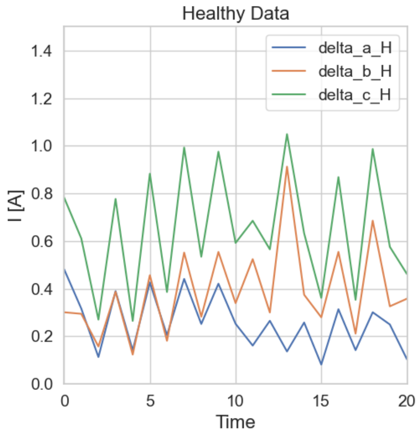
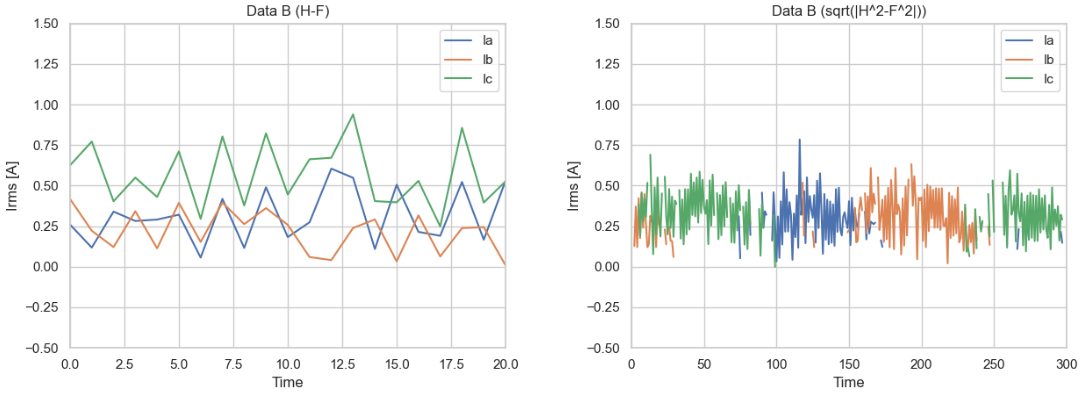


Figure 5.13. The differences of the three phases of Data B in a) the healthy case and b) faulty case

5.4.2 Calculated differences

Performing the last signal-processing in the time domain for Data B generated the plots for Equation 3.6 and Equation 3.7 in Figure 5.14. The third phase in Figure 5.14a shows that 'Ic' differs the most from the other two phases, but as the fault is implemented on 'Ib', this is not an indication of a ITSC fault. It is more likely that the gathering setup of Data B supplied small variations between the phases due to the collection, equipment and data acquisition. The plot for Equation 3.6 in Figure 5.14a looks normal, whereas the plot for Equation 3.7 in Figure 5.14b has some abnormalities in it. The calculation yielded a lot of NaNs which appear in a cyclic behaviour for the three phases. By looking into the data frame, the NaNs appear as the program could not compute the square and square-root of values close to zero. Using a mathematical or statistical approach to the signal-processing of Data B could have produced some fault detection, but this would demand human expert input and further research. A faster approach to fault detection of an ITSC fault is, therefore, machine learning modelling.



(a)

(b)

Figure 5.14. (a) Equation 3.6, and (b) Equation 3.7, applied to the healthy and faulty datasets of Data A for each phase.

5.5 Feature Extraction

The features that were calculated for each phase of Data A were mean value, Standard Deviation (STD), kurtosis, skewness and crest factor, as described in Chapter 3.4. The features were calculated for the 10 nearest rows of the target row. The features were also used to create the fault indicators, as described in Chapter 3.7.

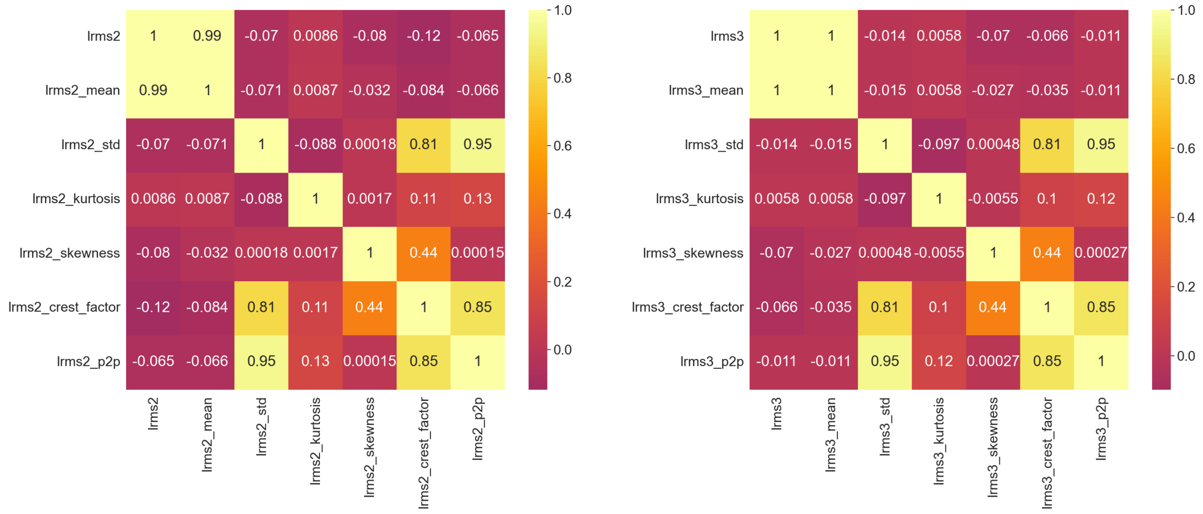
As Data B consists of the three-phased signal at a higher sampling rate, there were more features that can be extracted from the dataset. As described in the methodology in Chapter 3.4, the features calculated in Python are as follows: absolute mean, standard deviation, skewness, kurtosis, crest factor, peak-to-peak, RMS-value, clearance factor, shape factor, impulse factor. The Python code is shown in Appendix A.3, and the features are calculated for the 10 nearest rows. As for Data A, the fault indicators for Data B were calculated as described in Chapter 3.7. The new data frame with all the features and feature-based indicators is referenced as 'All_Features' for both Data A and Data B.

5.6 Principal Component Analysis

The code for the Principal Component Analysis (PCA), seen in Appendix A.4, is applied to the 'All_features' data frame for Data A and Data B. The following subchapters are split into PCA for Data A and Data B, and will show the covariance matrix of the features, the scatter plot and a heatmap of the found Principal Components.

5.6.1 PCA for Data A

The first step is to create a covariance matrix for the features in Data A and calculate the eigenvalue and eigenvector. The code suggested a number of three Principal Components for Data A, and so the data frame will be reduced from 18 features to three. The covariance matrix for all of the features is seen in Appendix C.3, and a cutout of the covariance matrices for 'Irms2' and 'Irms3' are shown in Figure 5.15. A value close to 1 represents a high positive covariance, whereas a value on the negative side of the scale represents a negative covariance. A negative covariance means that when one feature increases, the other tends to decrease. Values close to 0 represent little to no covariance between the features.



(a)

(b)

Figure 5.15. The covariance heat map for Data A for (a) Irms2 and (b) Irms3.

As the three phases are very similar, the separated covariance matrices are similar in terms of the pattern of the covariances. From this plot, the strongest covariance is found between the original values of the three phases, as well as between the mean values and the phases. This is expected, as the mean value is calculated in groups of 10 and is thus strongly related to the original Irms values. The standard deviation, crest factor and peak-to-peak values all have a strong covariance with each other. The phases do have small variations in the seperated covariance matrices, where, for instance, the skewness and kurtosis have a positive covariance of 0.0017 in ’Irms2’ and a negative covariance of -0.0055 for ’Irms3’. These values are however both positive for ’Irms1’ and ’Irms2’, and thus, the biggest difference is not found in the second phase in which the fault is implemented. The PCA tool is powerful, but it is hard to understand how the Principal Components are selected. Regardless, based on the covariance matrices, it can be suspected that the components will have something to do with the original phase values, mean values, crest factor, standard deviation and peak-to-peak values.

A heatmap of the resulting three Principal Components is shown in Figure 5.16. As described in Chapter 3.6, the heatmap shows a projection of the dataset onto the principal components. A higher value, in both the positive and negative direction, represents a high covariance between the Principal Component and the features. Here too, the standard deviation, crest factor and peak-to-peak features yield the highest score. The original Irms data and the consecutive mean values do also score high, with values around 0.4. The heatmap is a lower-dimension projection, and a data frame with the PCA values for the higher-dimension data is created. This data frame will be applied in the ML models and will be further referenced as the PCA data frame.

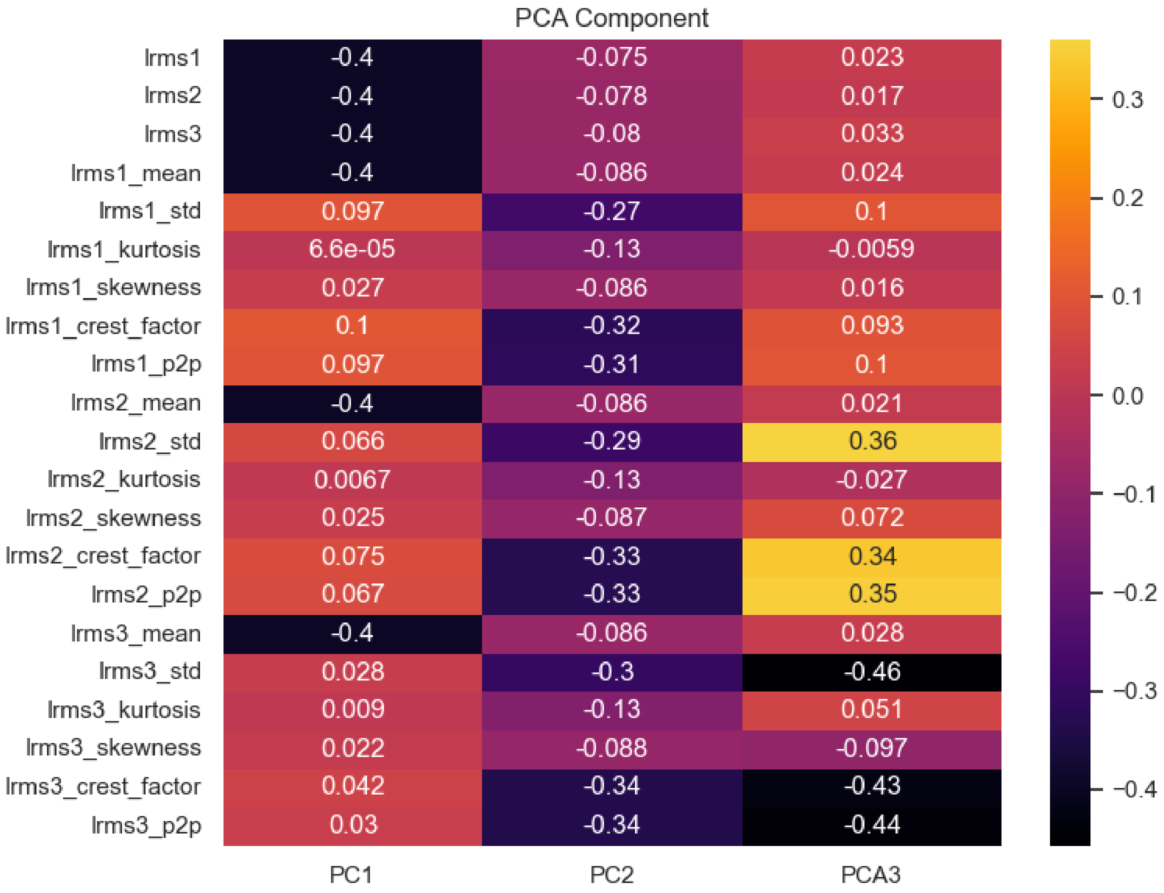
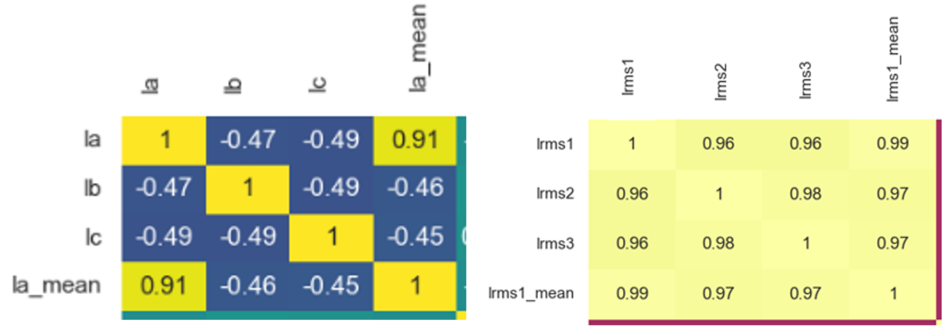


Figure 5.16. The projection matrix as a heatmap for the three PCAs of Data A.

5.6.2 PCA for Data B

The covariance heatmap for all the features in Data B is shown in Figure C.4 in Appendix A.4. The PCA for Data B was not performed on the phases separately, so the resulting covariance matrix is large. Generally, the covariance for Data B has a bigger range than Data A, where Data B ranges from 1 to -0.6, and Data A varies from 1 to -0.1. This means that Data B has more negative covariance than Data B, which is not strange considering that the signal values for Data B behave in a sinusoidal wave with 120 °separating the three phases. This shows a different covariance between the phases themselves, as seen in the clip-out of the covariance matrices for Data A and B in Figure 5.17.



	Ia	Ib	Ic	Ia_mean
Ia	1	-0.47	-0.49	0.91
Ib	-0.47	1	-0.49	-0.46
Ic	-0.49	-0.49	1	-0.45
Ia_mean	0.91	-0.46	-0.45	1

(a)

	Irms1	Irms2	Irms3	Irms1_mean
Irms1	1	0.96	0.96	0.99
Irms2	0.96	1	0.98	0.97
Irms3	0.96	0.98	1	0.97
Irms1_mean	0.99	0.97	0.97	1

(b)

Figure 5.17. Closer look of the covariance matrices of (a) Data A and (b) Data B.

Reviewing Figure C.4 in the Appendix A.4, all of the original data for Ia, Ib and Ic have a strong covariance with the calculated phase means and phase crest factor. In that sense, Data A and Data B have similar results for the covariance. The standard deviation of Data B shows the biggest covariance with the peak-to-peak values, the RMS values, and the impulse factor. The crest factor does also have a covariance of 0.53 with the clearance factor. Out of all the features, the impulse factor has covariance with the most number of features, and it is suspected that this feature will be represented in the principal components.

The Python code calculated the explained variance and found that the data frame should be reduced from 33 features to four Principal Components. The Principal Components for both Data A and Data B represent the features with an explained variance above 50 %. The projected matrix of the Principal Components and some of the features for Data B is shown in Figure 5.18. The Principal Components have the strongest covariance with the original phases, the standard deviation, the RMS values and the impulse factor. The new data frame with the four Principal Components will be referenced as the PCA data frame.

5.7 Machine Learning Models

The four ML models were run for three different data frames for Data A and Data B. The largest data frame was the 'All Features' data frame containing all the features and their consecutive feature indicators - six features for Data A and ten features for Data B. The second, and smallest, data frame was the 'PCA' data frame containing the PCAs presented in the previous chapter - three principal components for Data A and four for Data B. The last data frame is essentially the same as 'All Features', only that the data have not been pre-processed. The 'PCA' and 'Raw' data frames were applied to the ML models with the two best indicators as the target values (y) from the 'All Features' data frame.

5.7.1 ML Using 'All Features' for Data A

The 'All Features' data frame was passed through the ML code, as shown in Appendix A.5. The performance of the six feature indicators for each phase of Data A is shown in Table 5.3.

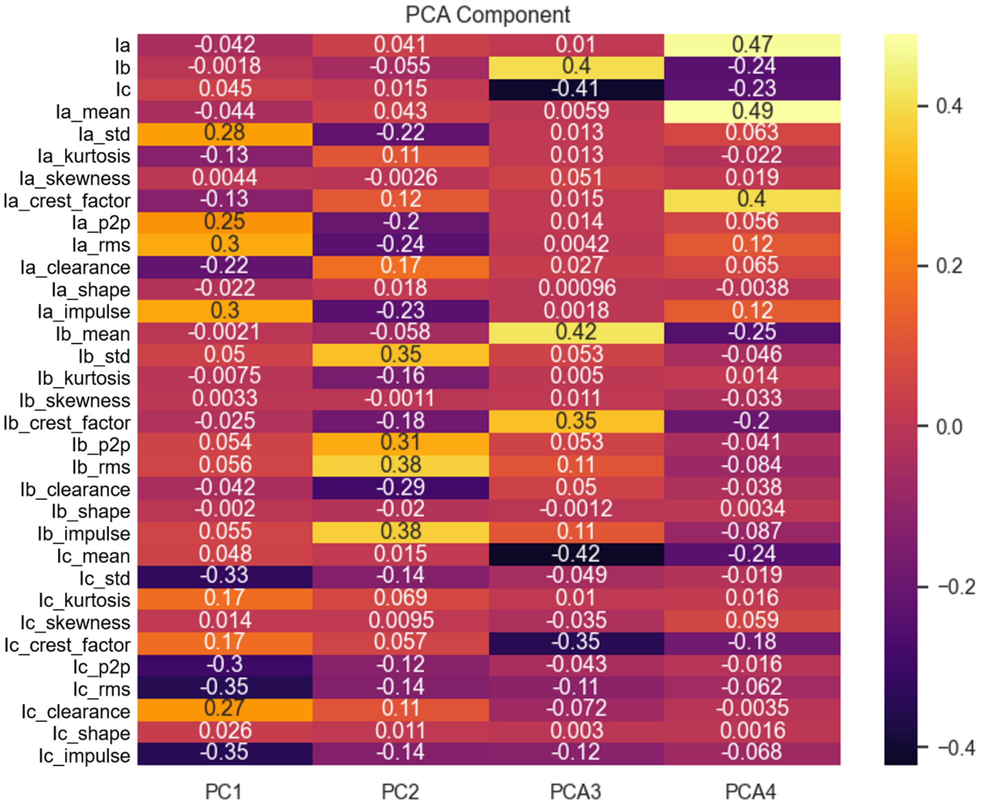
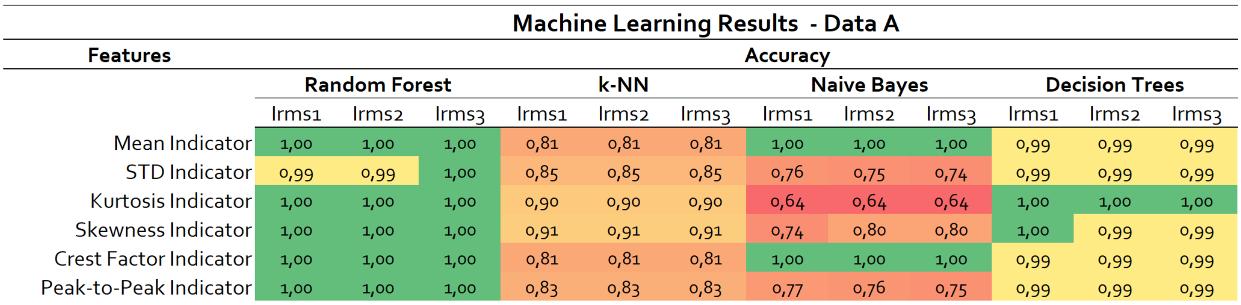


Figure 5.18. The projection matrix as a heatmap for the four PCAs of Data B.

The accuracy ranges from 0 to 1, where 1 represents 100 % accuracy. Upon first inspection, it seems that the Random Forest and Decision Tree model yields great results. However, over-fitting is a common issue for the two decision tree-based models, and the cause of the high accuracy might stem from this. It is very uncommon for a ML model to predict true values in all instances, so it is hard to trust a model with such results. The k-Nearest Neighbour (k-NN) model seems to yield the most realistic results for all indicators compared to the Naive Bayes, as it seems to over-fit the mean and crest factor indicators.

Table 5.3. The accuracy results of the ML methods applied to all of the features in Data A.



Machine Learning Results - Data A

Features	Accuracy											
	Random Forest			k-NN			Naive Bayes			Decision Trees		
	Irms1	Irms2	Irms3	Irms1	Irms2	Irms3	Irms1	Irms2	Irms3	Irms1	Irms2	Irms3
Mean Indicator	1,00	1,00	1,00	0,81	0,81	0,81	1,00	1,00	1,00	0,99	0,99	0,99
STD Indicator	0,99	0,99	1,00	0,85	0,85	0,85	0,76	0,75	0,74	0,99	0,99	0,99
Kurtosis Indicator	1,00	1,00	1,00	0,90	0,90	0,90	0,64	0,64	0,64	1,00	1,00	1,00
Skewness Indicator	1,00	1,00	1,00	0,91	0,91	0,91	0,74	0,80	0,80	1,00	0,99	0,99
Crest Factor Indicator	1,00	1,00	1,00	0,81	0,81	0,81	1,00	1,00	1,00	0,99	0,99	0,99
Peak-to-Peak Indicator	1,00	1,00	1,00	0,83	0,83	0,83	0,77	0,76	0,75	0,99	0,99	0,99

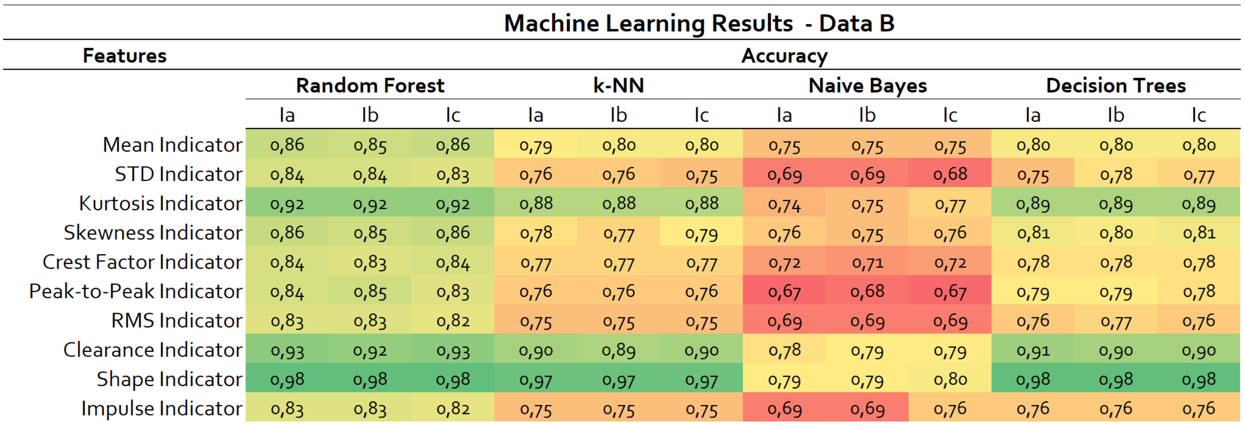
The Naive Bayes does additionally deliver the worst performance, where the kurtosis indicator is predicted with 64 % accuracy for all of the phases. Another noteworthy result from the Naive Bayes model is the differences between Irms1, Irms2, and Irms3 using the skewness indicator. The other models tend to yield approximately the same accuracy for the three phases, but here Irms1 have 74 % accuracy, whereas the two other phases have 80 %. Looking at all of the Naive Bayes models, the model has a lot of variations. Due to the fact that the features are all related and stem from the original phase values, the diversity in the accuracy of the Naive Bayes model is suspected to occur due to over-and under-fitting.

Excluding the Naive Bayes model, the ML models predicts with the highest accuracy using the kurtosis and skewness indicators.

5.7.2 ML Using 'All Features' for Data B

The ML models were applied to Data B with the 33 features, and the target variables were set in a for-loop for all the indicators obtained from the data labelling. The results from the models are gathered in Table 5.4. The Random Forest and Decision Tree models seem to generate more realistic accuracies in the predictions, compared to Data A. As the results are not tested for over-fitting, it is hard to say if the models over-fit or not. Regardless, the Random Forest and Decision Tree yield the highest accuracies out of the four models. The Random Forest model generally outperforms the Decision Tree model for all of the features in Data B. This is expected as the Random Forest is an ensemble version of the Decision Tree, and will thus be more robust to variations in the data.

Table 5.4. The accuracy results of the ML methods applied to all of the features in Data B.



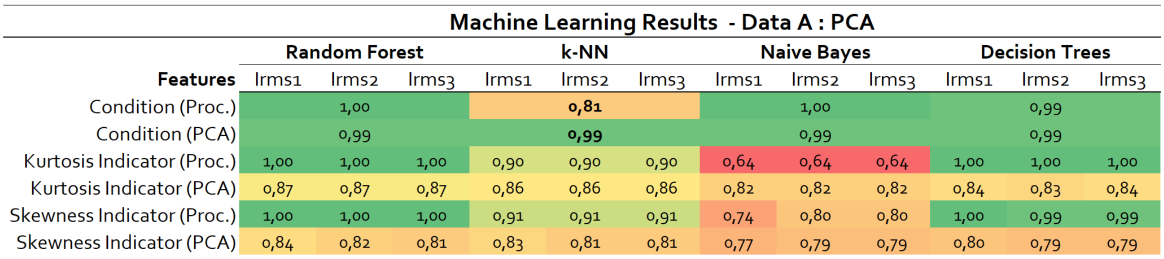
Machine Learning Results - Data B												
Features	Accuracy											
	Random Forest			k-NN			Naive Bayes			Decision Trees		
	Ia	Ib	Ic	Ia	Ib	Ic	Ia	Ib	Ic	Ia	Ib	Ic
Mean Indicator	0,86	0,85	0,86	0,79	0,80	0,80	0,75	0,75	0,75	0,80	0,80	0,80
STD Indicator	0,84	0,84	0,83	0,76	0,76	0,75	0,69	0,69	0,68	0,75	0,78	0,77
Kurtosis Indicator	0,92	0,92	0,92	0,88	0,88	0,88	0,74	0,75	0,77	0,89	0,89	0,89
Skewness Indicator	0,86	0,85	0,86	0,78	0,77	0,79	0,76	0,75	0,76	0,81	0,80	0,81
Crest Factor Indicator	0,84	0,83	0,84	0,77	0,77	0,77	0,72	0,71	0,72	0,78	0,78	0,78
Peak-to-Peak Indicator	0,84	0,85	0,83	0,76	0,76	0,76	0,67	0,68	0,67	0,79	0,79	0,78
RMS Indicator	0,83	0,83	0,82	0,75	0,75	0,75	0,69	0,69	0,69	0,76	0,77	0,76
Clearance Indicator	0,93	0,92	0,93	0,90	0,89	0,90	0,78	0,79	0,79	0,91	0,90	0,90
Shape Indicator	0,98	0,98	0,98	0,97	0,97	0,97	0,79	0,79	0,80	0,98	0,98	0,98
Impulse Indicator	0,83	0,83	0,82	0,75	0,75	0,75	0,69	0,69	0,76	0,76	0,76	0,76

Compared to Data A, the k-NN model for Data B predicts the ITSC faults with lower accuracy. However, the k-NN model and the Decision Tree model have very similar accuracies. The Naive Bayes model is the worst predictor overall and does not seem to be a good fit for either Data A or B. This is not that surprising, as the Naive Byes is known for delivering untrustworthy predictors [53]. Out of all the indicators, the kurtosis, clearance factor and shape factor indicator delivers the best accuracies. The two latter indicators are not applied to Data A, but both datasets share the kurtosis indicator as one of the best indicators. The RMS and impulse factor indicators yield the worst accuracies for Data B. If the results are accepted as actual fault indicators, it weakens the argument of using Data A for fault detection as it is RMS-values.

5.7.3 Performance of PCA Data Frame

From the 'All Features' data frame, it was evident that Data A possibly had some over-fitting issues for the Random Forest and Decision Tree models. One of the methods for reducing over-fitting presented in [42] is using feature selection to reduce the dimensions of the dataset. The effects of the feature selection method PCA showed great results for Data A, as seen in Table 5.5. The kurtosis and skewness indicator yielded 100 % accuracy for the Random Forest and Decision Tree models previously, which was reduced for the PCA data frame to 87 % and 81 %, respectively. This is the first indication that by using the PCA data frame, the over-fitting-prone models will generate more realistic performances. Contrary to this, it can be argued that the models merely have worse predictions for the PCA data frame than the previous. On another note, the PCA data frame did not reduce, but increased, the accuracy for Naive Bayes using the kurtosis indicator from 64 % to 82 % for all the phases. It seems that for Data A, the PCA created more realistic results when it comes to over-fitting, but also increased the performance for the models with the worst accuracy.

Table 5.5. Accuracy of ML models for the two best indicators and condition of Data A, comparison between PCA and Features Dataframe.

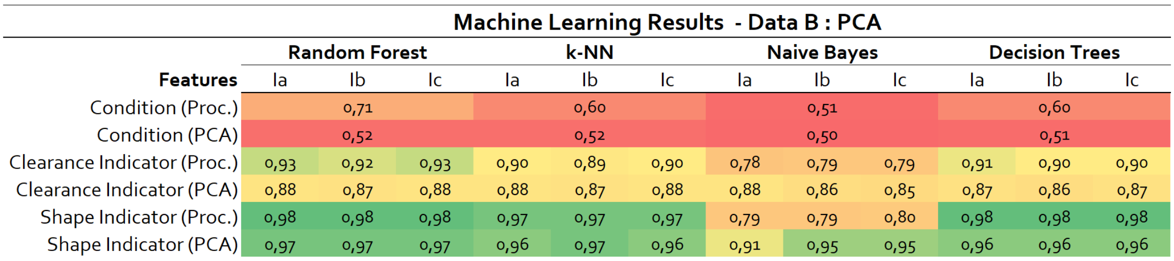


Machine Learning Results - Data A : PCA												
	Random Forest			k-NN			Naive Bayes			Decision Trees		
Features	Irms1	Irms2	Irms3	Irms1	Irms2	Irms3	Irms1	Irms2	Irms3	Irms1	Irms2	Irms3
Condition (Proc.)	1,00			**0,81**			1,00			0,99		
Condition (PCA)	0,99			**0,99**			0,99			0,99		
Kurtosis Indicator (Proc.)	1,00	1,00	1,00	0,90	0,90	0,90	0,64	0,64	0,64	1,00	1,00	1,00
Kurtosis Indicator (PCA)	0,87	0,87	0,87	0,86	0,86	0,86	0,82	0,82	0,82	0,84	0,83	0,84
Skewness Indicator (Proc.)	1,00	1,00	1,00	0,91	0,91	0,91	0,74	0,80	0,80	1,00	0,99	0,99
Skewness Indicator (PCA)	0,84	0,82	0,81	0,83	0,81	0,81	0,77	0,79	0,79	0,80	0,79	0,79

The PCA did however not reduce the over-fitting of the 'Condition' indicator, which states if the value originates from the healthy or faulty dataset. This is initially a poor target variable as it does not represent a real fault, only in which dataset the fault can be found in. Nevertheless, it is interesting that the PCA did not reduce the accuracy for the models using 'Condition' as a target. It is hard to distinguish if the 'Condition'-variable is prone to over-fitting, or if it actually is a good indicator of a fault. Whether or not the prediction over-fits can be inspected by applying the k-fold-cross-validation test, but this was not done in the work for the thesis. Another factor that plays a role for Data A is that the mean value for the faulty data is about 0.02 A lower than the healthy dataset, and it is therefore relatively easy for the models to predict which dataset the value originates from. This factor further weakens the argument that Data A is a good dataset for fault detection, and that 'Condition' is a good target value.

The PCA algorithm did also somewhat reduce the accuracy for the indicators of Data B, as seen in Table 5.6. The biggest difference compared to Data A is the results using the 'Condition' variable, where the PCA data frame of B reduced the accuracy by 10 to 20 % for the Random Forest, k-NN and Decision Tree models. The 'Condition' variable is not a good indicator for Data B, as seen by the poor performance. Apart from that, using the PCA data frame reduces the accuracy of the clearance indicator more than the shape indicator, which stays more or less the same. This suggests that the shape factor is a better fault indicator for the PCA data frame than the clearance indicator. Another noticeable factor of the results is that the PCA data frame yields a higher performance for the Naive Bayes models, increasing the accuracy by approximately 10 % for both indicators. This increase was the same as for Data A, and it proves that the Naive Bayes model has a greater necessity for a PCA step, at least more so than the other three models.

Table 5.6. Accuracy of ML models for the two best indicators and condition of Data B, comparison between PCA and Features Dataframe.

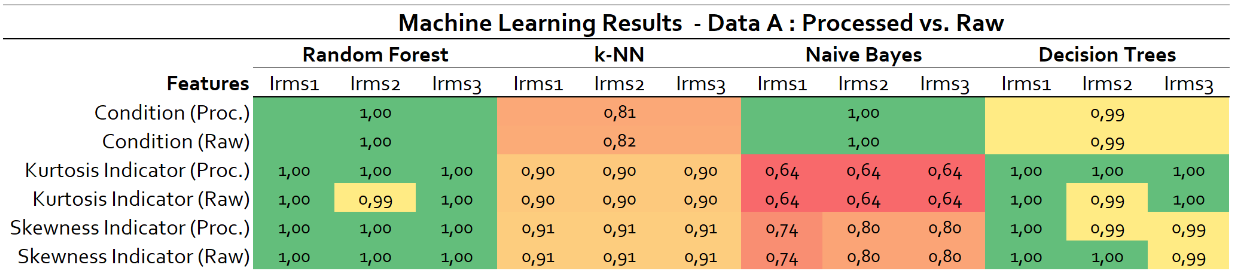


Machine Learning Results - Data B : PCA												
	Random Forest			k-NN			Naive Bayes			Decision Trees		
Features	Ia	Ib	Ic	Ia	Ib	Ic	Ia	Ib	Ic	Ia	Ib	Ic
Condition (Proc.)	0,71			0,60			0,51			0,60		
Condition (PCA)	0,52			0,52			0,50			0,51		
Clearance Indicator (Proc.)	0,93	0,92	0,93	0,90	0,89	0,90	0,78	0,79	0,79	0,91	0,90	0,90
Clearance Indicator (PCA)	0,88	0,87	0,88	0,88	0,87	0,88	0,88	0,86	0,85	0,87	0,86	0,87
Shape Indicator (Proc.)	0,98	0,98	0,98	0,97	0,97	0,97	0,79	0,79	0,80	0,98	0,98	0,98
Shape Indicator (PCA)	0,97	0,97	0,97	0,96	0,97	0,96	0,91	0,95	0,95	0,96	0,96	0,96

5.7.4 Pre-processed Data vs. Raw Data

The two best indicators for Data A were also run through the ML code using the raw data frame. The results are compared to the processed data in Table 5.7. It is apparent that the performance of the ML models are not affected by the pre-processing step for Data A. The biggest deviation in the results of the raw and processed data frames is 1 % and is for most cases due to round-off differences. Taking this into consideration, the processing step of Data A was most likely not effective enough. Other outlier detection methods, especially techniques performed in the frequency domain, should have been applied to the dataset to increase the performance.

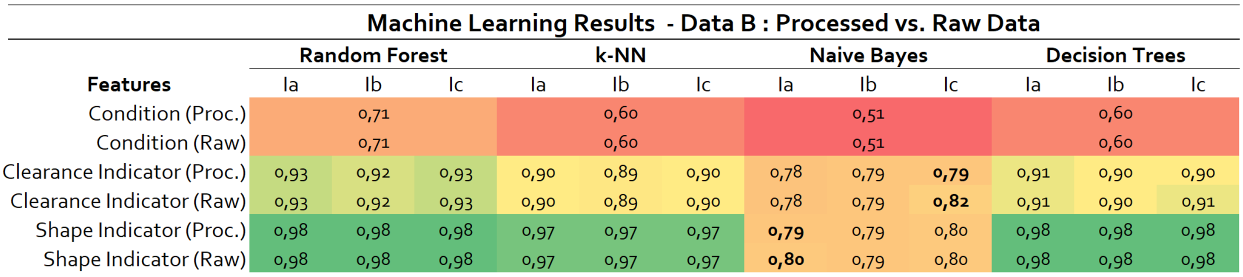
Table 5.7. The accuracy of the ML methods for the two best indicators and Condition, processed data compared to raw data.



Machine Learning Results - Data A : Processed vs. Raw												
	Random Forest			k-NN			Naive Bayes			Decision Trees		
Features	Irms1	Irms2	Irms3	Irms1	Irms2	Irms3	Irms1	Irms2	Irms3	Irms1	Irms2	Irms3
Condition (Proc.)		1,00			0,81			1,00			0,99	
Condition (Raw)		1,00			0,82			1,00			0,99	
Kurtosis Indicator (Proc.)	1,00	1,00	1,00	0,90	0,90	0,90	0,64	0,64	0,64	1,00	1,00	1,00
Kurtosis Indicator (Raw)	1,00	0,99	1,00	0,90	0,90	0,90	0,64	0,64	0,64	1,00	0,99	1,00
Skewness Indicator (Proc.)	1,00	1,00	1,00	0,91	0,91	0,91	0,74	0,80	0,80	1,00	0,99	0,99
Skewness Indicator (Raw)	1,00	1,00	1,00	0,91	0,91	0,91	0,74	0,80	0,80	1,00	1,00	0,99

The ML models were also applied for the two best indicators from Data B on the unprocessed, raw data. The results from the raw data frames are compared to the results from the processed data in Table 5.8. The findings from Data A does also apply to Data B, where the pre-processing has little influence on the performance of the ML models. Unlike Data A, the accuracy of the Naive Bayes using the clearance indicator is increased from 79 % to 82 % in the raw data frame. This could have been an indication of the faults, but as the ITSC fault is implemented on the Ib-phase, this is very unlikely. As discussed earlier, the Naive Bayes model has big variations in the accuracy of the predictions and is most likely the cause of the differences.

Table 5.8. The accuracy of the ML methods for the two best feature indicators and 'Condition', processed data compared to raw data.



Machine Learning Results - Data B : Processed vs. Raw Data												
	Random Forest			k-NN			Naive Bayes			Decision Trees		
Features	Ia	Ib	Ic	Ia	Ib	Ic	Ia	Ib	Ic	Ia	Ib	Ic
Condition (Proc.)		0,71			0,60			0,51			0,60	
Condition (Raw)		0,71			0,60			0,51			0,60	
Clearance Indicator (Proc.)	0,93	0,92	0,93	0,90	0,89	0,90	0,78	0,79	**0,79**	0,91	0,90	0,90
Clearance Indicator (Raw)	0,93	0,92	0,93	0,90	0,89	0,90	0,78	0,79	**0,82**	0,91	0,90	0,91
Shape Indicator (Proc.)	0,98	0,98	0,98	0,97	0,97	0,97	**0,79**	0,79	0,80	0,98	0,98	0,98
Shape Indicator (Raw)	0,98	0,98	0,98	0,97	0,97	0,97	**0,80**	0,79	0,80	0,98	0,98	0,98

5.8 General Discussion

5.8.1 Biases

The results of Data A and B for the processed data frame have some similarities and some differences. The Random Forest and Decision Tree models seem to yield the highest accuracy for both Data A and B, but as discussed previously, this is suspected to be due to over-fitting. Another factor to the high accuracy, which applies to all the models, is the fact that the features are directly calculated from the phase values. The target variables in the models are also the indicators which are based on the calculated features. In other terms, the variables, features and indicators used in the models are strongly correlated, not necessarily by a true fault, but by biases.

Looking at the cases for Data A and B where ’Condition’ is used as a target variable, the models yield high accuracy for Data A, but very low for Data B. The suspicion was that the ’Condition’ variable was a poor indicator of faulty or healthy data as it only represents where the data originate from. The difference between the healthy and faulty datasets in Data A is very apparent in terms of the mean values of the phase currents. The distinction between the healthy and faulty data is visible in Figure 5.2. For the faulty and healthy data sets in Data B, there is not an apparent difference between them. Due to this particular reason, Data A yields high accuracies and Data B yields low for the ’Condition’ target variable. Considering these points, the suspicion that the ’Condition’ variable is not a good indicator of a fault, is confirmed.

5.8.2 Importance of Data Labelling

Regarding the issue of labelling the data, the ML methods are essentially predicting if the datasets are within the 95th percentile of the different calculated features. Whether or not this actually represents a fault is hard to say, and was not proven. The type of data labelling used in this thesis is a combination of normalisation theory and guessing. Most computational data labelling that is conducted after the data gathering, requires human expert inputs on the training set. For instance, for datasets where the ML model is trained on photos of cats or dogs, it would be easy for a human to label the image a cat or a dog. The issue of this thesis lies with the understanding of the datasets, where the aim is for the machine learning to understand when the signals represent a ITSC fault, and when it does not. For the ’Condition’ indicator, the data is not labelled specifically enough, as all of the faulty datasets are not faulty values. The ITSC fault is implemented to short circuit 4.8 % of the second phase, and more knowledge on how this would appear in the data is needed. Data labelling is one of the greater weaknesses of the thesis, and with more focus on the labelling - the results could have been more factual and applicable on fault diagnosis of the specific PMSM.

Data labelling and signal processing for three-phased current signals are usually performed in the frequency domain. Thus, another weakness of the thesis is that the analysis is only performed in the time domain. The method used for signal processing is essentially an inspection of the data and should have been based on sounder statistical or theoretical approaches. For instance, signal processing in the frequency domain could have rendered more useful data labelling. Signal processing in the frequency domain would be most applicable to Data B as it has a sinusoidal signal, whereas Data A would have limitations due to the fact that it is comprised of RMS values. As Data B was obtained late in the works of this thesis, there was not enough time to perform an in-depth analysis of the frequency domain. If given the time, signal processing on Data B could have given indications on an ITSC fault. Previous work on ITSC faults have reported specific frequencies at which the fault appears as [43], [60], [61], and this should have been inspected further.

5.9 Key Findings and Reflections

The results of the thesis are most likely not directly applicable for Fault Detection Identification (FDI) of real-life hydro-generators, however a summary of the key findings and reflection are summed up in the list below.

- The data gathering should have been more technical and used other data acquisition systems, where a method for flagging the ITSC fault should have been implemented.
- The RMS values will limit the extent of the type of features that can be extracted from the data, and thus, signal data at a higher sampling rate is preferred. This is mostly due to the fact that the sinusoidal behaviour of the three-phased signal is not kept in the RMS-transformation, and is a weakness of Data A.
- Data A is further weakened by the ML findings using the RMS feature in Data B, where the feature did not generate good predictions.
- The pre-processing step using the Z-score method proved to be insignificant when it comes to improving the accuracies of the ML models.
- Removal of noise is usually performed in the frequency domain and can be transformed back into the time domain. This additional pre-processing step could add some more accuracy to the models, even for the time domain analysis.
- Although the time domain can produce some data labelling based on statistical features and guessing, inspecting Data B in the frequency domain is suspected to yield more precise data labelling.
- The over-fitting of Random Forest and Decision Tree could be checked by using the k-fold-cross-validation method. However, applying PCA to the datasets proved to yield more realistic predictions, especially for Data A.
- The Naive Bayes model is a poor predictor for the datasets, as the model assumes that all the features are independent of each other. This is not the case for the datasets used in this thesis, where all the features are calculated from the same parameters.
- Other ML methods than supervised learning should have been tested as the data labelling proved to be a big weakness of the thesis. Methods like unsupervised and neural networks do not need a label and could have provided more reliable fault detection. These methods do however require more inputs than supervised learning, and these inputs should often be independent. All the data and features of this thesis are dependent of each other.
- Accuracy is a good indicator of the models, but it can also paint a wrong picture. Additional parameters like robustness, precision and prevalence should have been performed given more time.

Chapter 6

Conclusion

In this thesis, a Fault Detection Identification (FDI) method using four Machine Learning (ML) models has been proposed to indicate an Inter-Turn Short-Circuit (ITSC) fault of a Permanent Magnet Synchronous Motor (PMSM). Three-phased current data was gathered from a 2.2 kW IE5 Permanent Magnet Synchronous Motor (PMSM). Data A was collected for this thesis in an experimental setup, whereas Data B was previously collected for a dissertation using the same PMSM. Pre-processing, feature extraction, signal processing, Principal Component Analysis (PCA), and four ML models were performed in Python on both datasets and produced fault detection for different variants of the data.

The sources of errors in the experimental setup such as misalignment; electrical setup; resistance in the cables and components; differences between the collections; and faulty equipment, are suspected to have caused poor quality of Data A. This was however shadowed by the fact that the data acquisition system gathered RMS values of the PMSM. The preferred format of the data is sinusoidal signals such as in Data B, and the usefulness of Data A was strongly limited due to this. The data quality of the data from Hydroelectric Powerplant (HEP) is a big bottleneck for FDI using ML to be successful, and thus the data acquisition system used in the collection of Data A is not recommended for further work.

Regarding the domain that the datasets were inspected in, it can be concluded that for the work done in this thesis, the ITSC fault was not found in the time domain by visual inspection. Based on the literature and other research, the suspicion is that the frequency domain would tell more about the ITSC fault. Therefore, it is recommended that the data inspection and signal processing should be performed in the frequency domain. For the data pre-processing, the Z-score method of outlier detection proved to be insignificant for the performance of the ML models. For this reason, it is recommended to use a more advanced approach for the outlier detection of the datasets. For Data B and other signal-based current data, noise removal in the frequency domain should be applied.

The four ML models applied in this thesis, Random Forest, k-Nearest Neighbour (k-NN), Naive Bayes and Decision Trees, perform differently from each other. For Data A and Data B, the Random Forest and Decision Tree models tend to over-fit the prediction. However, the effect of over-fitting is reduced noticeably by applying a Principal Component Analysis (PCA) step to the datasets. The Naive Bayes model generated the lowest accuracies of the ML models and is generally considered an unreliable predictor. This model was positively affected by the PCA. The k-Nearest Neighbour (k-NN) model provided decent predictions and was considered a more trustworthy model compared to the other three.

Using 'Condition' as a target variable yielded great results for Data A, but poorly for Data B. The healthy and faulty datasets that Data A was made up of, had a 0.02 A difference in the mean value of the current, and for this reason, the ML models could easily make predictions using 'Condition' as a target value. The other fault indicators used in the ML models were created from flagging values above the 95th percentile of the calculated features. Whether these indicators represent a fault is doubted, and other data labelling techniques should be applied in further work. For Data A, the kurtosis and skewness features indicator yielded the highest prediction accuracy, whereas for Data B, the clearance and shape factor indicator rendered the highest accuracy.

Bibliography

[1] H. A. Raja, K. Kudelina, B. Asad, and T. Vaimann, "Perspective chapter: Fault detection and predictive maintenance of electrical machines," in *New Trends in Electric Machines*, M. Delgado-Prieto, J. A. A. Daviu, and R. A. O. Rios, Eds., Rijeka: IntechOpen, 2022, ch. 1. DOI: 10.5772/intechopen.107167. [Online]. Available: https://doi.org/10.5772/intechopen.107167.

[2] B. Ristić and I. Bozic, "A short overview on industry 4.0 in maintenance of hydropower plants," University of Belgrade Faculty of Mechanical Engineering, Sep. 2022, ISBN: 978 86 6060 131 7. [Online]. Available: https://machinery.mas.bg.ac.rs/handle/123456789/6156.

[3] K. Kudelina, T. Vaimann, B. Asad, A. Rassõlkin, A. Kallaste, and G. Demidova, "Trends and Challenges in Intelligent Condition Monitoring of Electrical Machines Using Machine Learning," *Applied Sciences 2021, Vol. 11, Page 2761*, vol. 11, no. 6, p. 2761, Mar. 2021, ISSN: 2076-3417. DOI: 10.3390/APP11062761. [Online]. Available: https://www.mdpi.com/2076-3417/11/6/2761/htm%20https://www.mdpi.com/2076-3417/11/6/2761.

[4] J. Andreassen, "Agder energi kan tape 60 millioner kroner på feilen," *Agderposten*, Feb. 7, 2020. [Online]. Available: https://www.agderposten.no/naeringsliv/i/JQW5l6/agder-energi-kan-tape-60-millioner-kroner-paa-feilen, (accessed: 05.02.2023).

[5] A. Betti, E. Crisostomi, G. Paolinelli, A. Piazzi, F. Ruffini, and M. Tucci, "Condition monitoring and predictive maintenance methodologies for hydropower plants equipment," *Renewable Energy*, vol. 171, pp. 246–253, 2021, ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2021.02.102. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0960148121002810.

[6] NVE. "Kraftproduksjon." (2023), [Online]. Available: https://www.nve.no/energi/energisystem/kraftproduksjon/. (accessed: 01.05.2023).

[7] NVE, "Hvor stor andel av vannkraften i norge er fleksibel?" *NVE FAKTA*, vol. 1/2023, 2023, (accessed: 10.02.2023.

[8] M. Skoric, "Falling water, rising power," *IEEE Potentials*, vol. 36, no. 5, pp. 30–31, 2017. DOI: 10.1109/MPOT.2017.2700238.

[9] *Electric Machinery* (McGraw-Hill Series in Electrical and Computer Engineering), 6th Edition. McGraw-Hill Higher Education, 2003, ISBN: 0-07-123010-6.

[10] V. S. Sunde, "Preliminary study of common failures of hydro power generators," *internal source UiA*, 2022, [Energy Research Project].

[11] "Hydrogenerator failures – results of the survey cigre study committee sc11 eg 11.02," in CIGRE AI group, 2009, p. 22. [Online]. Available: https://e-cigre.org/publication/392-survey-of-hydrogenerator-failures, [Brochures].

[12] C. Sumereder, "Statistical lifetime of hydro generators and failure analysis," *IEEE Transactions on Dielectrics and Electrical Insulation*, vol. 15, no. 3, pp. 678–685, 2008. DOI: 10.1109/TDEI.2008.4543104.

[13] P. Tavner, "Review of condition monitoring of rotating electrical machines," *IET Electric Power Applications*, vol. 2, no. 4, pp. 215–247, 2008. DOI: 10.1049/iet-epa:20070280.

[14] L. T. Rosenberg, "Generator eccentricity, vibration, and shaft currents," *Electrical Engineering*, vol. 74, no. 6, pp. 465–465, 1955. DOI: 10.1109/EE.1955.6439405.

[15] H. Ehya, I. Sadeghi, and J. Faiz, "Online condition monitoring of large synchronous generator under eccentricity fault," in *2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2017, pp. 19–24. DOI: 10.1109/ICIEA.2017.8282807.

[16] H. C. Dirani, A. Merkhouf, A.-M. Giroux, and K. Al-Haddad, "Study of the impact of eccentricity in large synchronous generator with finite elements," in *2014 International Conference on Electrical Machines (ICEM)*, 2014, pp. 277–282. DOI: 10.1109/ICELMACH.2014.6960193.

[17] H. Ehya, A. Nysveen, and R. Nilssen, "A practical approach for static eccentricity fault diagnosis of hydro-generators," in *2020 International Conference on Electrical Machines (ICEM)*, vol. 1, 2020, pp. 2569–2574. DOI: 10.1109/ICEM49940.2020.9270675.

[18] C. Klein, M. Palmieri, M. Nienhaus, and E. Grasso, "Effect of static eccentricity on the mean values of the inductances of pmsms," in *2022 IEEE 21st Mediterranean Electrotechnical Conference (MELECON)*, 2022, pp. 348–353. DOI: 10.1109/MELECON53508.2022.9843071.

[19] I. M. Hussein and Z. Al-Hamouz, "Neural network based detection technique for eccentricity fault in lspms motors," in *2018 Innovations in Intelligent Systems and Applications Conference (ASYU)*, 2018, pp. 1–5. DOI: 10.1109/ASYU.2018.8554007.

[20] H. Ehya and A. Nysveen, "Comprehensive broken damper bar fault detection of synchronous generators," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 4, pp. 4215–4224, 2022. DOI: 10.1109/TIE.2021.3071678.

[21] P. J. Berry and E. S. Hamdi, "An investigation into damper winding failure in a large synchronous motor," in *2015 50th International Universities Power Engineering Conference (UPEC)*, 2015, pp. 1–4. DOI: 10.1109/UPEC.2015.7339924.

[22] H. Ehya, A. Nysveen, R. Nilssen, and U. Lundin, "Time domain signature analysis of synchronous generator under broken damper bar fault," in *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, 2019, pp. 1423–1428. DOI: 10.1109/IECON.2019.8927529.

[23] S. Nandi, H. Toliyat, and X. Li, "Condition monitoring and fault diagnosis of electrical motors—a review," *IEEE Transactions on Energy Conversion*, vol. 20, no. 4, pp. 719–729, 2005. DOI: 10.1109/TEC.2005.847955.

[24] Y. Park, S. B. Lee, J. Yun, M. Sasic, and G. C. Stone, "Air gap flux-based detection and classification of damper bar and field winding faults in salient pole synchronous motors," *IEEE Transactions on Industry Applications*, vol. 56, no. 4, pp. 3506–3515, 2020. DOI: 10.1109/TIA.2020.2983902.

[25] A. Hasan and J. Singh, "Fault detection in ball bearing through machine learning models," in *2022 6th International Conference on Electronics, Communication and Aerospace Technology*, 2022, pp. 1–6. DOI: 10.1109/ICECA55336.2022.10009579.

[26] V. Barai, S. Ramteke, V. Dhanalkotwar, Y. Nagmote, S. Shende, and D. Deshmukh, "Bearing fault diagnosis using signal processing and machine learning techniques: A review," *IOP Conference Series: Materials Science and Engineering*, vol. 1259, p. 012 034, Oct. 2022. DOI: 10.1088/1757-899X/1259/1/012034.

[27] A. Patange, J. R, N. Bajaj, A. Khairnar, and N. Gavade, "Application of machine learning for tool condition monitoring in turning," *Sound and Vibration*, vol. 56, pp. 127–145, Mar. 2022. DOI: 10.32604/sv.2022.014910.

[28] M. A. Khan, B. Asad, K. Kudelina, T. Vaimann, and A. Kallaste, "The Bearing Faults Detection Methods for Electrical Machines&mdash;The State of the Art," *Energies 2023, Vol. 16, Page 296*, vol. 16, no. 1, p. 296, Dec. 2022, ISSN: 1996-1073. DOI: 10.3390/EN16010296. [Online]. Available: https://www.mdpi.com/1996-1073/16/1/296/htm%20https://www.mdpi.com/1996-1073/16/1/296.

[29] I. Sadeghi, H. Ehya, J. Faiz, and A. A. S. Akmal, "Online condition monitoring of large synchronous generator under short circuit fault — a review," in *2018 IEEE International Conference on Industrial Technology (ICIT)*, 2018, pp. 1843–1848. DOI: 10.1109/ICIT.2018.8352465.

[30] H. Lee, H. Jeong, G. Koo, J. Ban, and S. W. Kim, "Attention recurrent neural network-based severity estimation method for interturn short-circuit fault in permanent magnet synchronous machines," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 4, pp. 3445–3453, 2021. DOI: 10.1109/TIE.2020.2978690.

[31] P. Naderi, "Magnetic-equivalent-circuit approach for inter-turn and demagnetisation faults analysis in surface mounted permanent-magnet synchronous machines using pole specific search-coil technique," *IET Electric Power Applications*, vol. 12, no. 7, pp. 916–928, 2018. DOI: https://doi.org/10.1049/iet-epa.2017.0403.

[32] Y. Qi, E. Bostanci, M. Zafarani, and B. Akin, "Severity estimation of interturn short circuit fault for pmsm," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 9, pp. 7260–7269, 2019. DOI: 10.1109/TIE.2018.2879281.

[33] M. A. Mazzoletti, G. R. Bossio, C. H. De Angelo, and D. R. Espinoza-Trejo, "A model-based strategy for interturn short-circuit fault diagnosis in pmsm," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 9, pp. 7218–7228, 2017. DOI: 10.1109/TIE.2017.2688973.

[34] T. dos Santos, F. J. T. E. Ferreira, J. M. Pires, and C. Damásio, "Stator winding short-circuit fault diagnosis in induction motors using random forest," in *2017 IEEE International Electric Machines and Drives Conference (IEMDC)*, 2017, pp. 1–8. DOI: 10.1109/IEMDC.2017.8002350.

[35] E. Ahmar, V. Choqueuse, M. Benbouzid, *et al.*, "Advanced signal processing techniques for fault detection and diagnosis in a wind turbine induction generator drive train: A comparative study," Oct. 2010, pp. 3576–3581. DOI: 10.1109/ECCE.2010.5617707.

[36] Z. Germán-Salló and G. Strnad, "Signal processing methods in fault detection in manufacturing systems," *Procedia Manufacturing*, vol. 22, pp. 613–620, 2018, 11th International Conference Interdisciplinarity in Engineering, INTER-ENG 2017, 5-6 October 2017, Tirgu Mures, Romania, ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2018.03.089. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2351978918303858.

[37] M. L. Wymore, J. E. Van Dam, H. Ceylan, and D. Qiao, "A survey of health monitoring systems for wind turbines," *Renewable and Sustainable Energy Reviews*, vol. 52, pp. 976–990, 2015, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2015.07.110. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032115007571.

[38] L. Ehrlinger, V. Haunschmid, D. Palazzini, and C. Lettner, "A daql to monitor data quality in machine learning applications," in *Database and Expert Systems Applications*, J. Hartmann Svenand Küng, S. Chakravarthy, G. Anderst-Kotsis, A. M. Tjoa, and I. Khalil, Eds., Cham: Springer International Publishing, 2019, pp. 227–237. DOI: 10.1007/978-3-030-27615-7_17.

[39] V. Gudivada, A. Apon, and J. Ding, "Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations," *International Journal on Advances in Software*, vol. 10, pp. 1–20, Jul. 2017. [Online]. Available: https://www.researchgate.net/publication/318432363_Data_Quality_Considerations_for_Big_Data_and_Machine_Learning_Going_Beyond_Data_Cleaning_and_Transformations.

[40] V. Nasteski, "An overview of the supervised machine learning methods," *HORIZONS.B*, vol. 4, pp. 51–62, Dec. 2017. DOI: 10.20544/HORIZONS.B.04.1.17.P05.

[41] A. Stetco, F. Dinmohammadi, X. Zhao, *et al.*, "Machine learning methods for wind turbine condition monitoring: A review," *Renewable Energy*, vol. 133, pp. 620–635, 2019, ISSN: 0960-1481. DOI: https://doi.org/10.1016/j.renene.2018.10.047. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S096014811831231X.

[42] X. Ying, "An overview of overfitting and its solutions," *Journal of Physics: Conference Series*, vol. 1168, p. 022 022, Feb. 2019. DOI: 10.1088/1742-6596/1168/2/022022.

[43] S. Attestog, "Modelling and detecting faults of permanent magnet synchronous motors in dynamic operations," [Doctoral Dissertation], University of Agder, Faculty of Engineering and Science, 2022. [Online]. Available: https://hdl.handle.net/11250/3025635.

[44] S. Psychology. "Z-score: Definition, formula, calculation & interpretation." (2022), [Online]. Available: https://www.simplypsychology.org/z-score.html%5C#:~:text=The%5C%20formula%5C%20for%5C%20calculating%5C%20a,by%5C%20the%5C%20population%5C%20standard%5C%20deviation.. [Online Tutorial].

[45] M. S. Mahmoud, H. V. Khang, J. Senanayaka, and R. P. Panadero, "Fault diagnosis of low-severity demagnetization in permanent magnet synchronous motors using numerical data," in *2022 25th International Conference on Electrical Machines and Systems (ICEMS)*, 2022, pp. 1–6. DOI: 10.1109/ICEMS56177.2022.9983411.

[46] A. Warya. "Principal component analysis (pca)." (2019), [Online]. Available: https://www.geeksforgeeks.org/ml-principal-component-analysispca/.

[47] scikit-learn.org. "1.10. decision trees." (2023), [Online]. Available: https://scikit-learn.org/stable/modules/tree.html%5C#decision-trees. [Online Resource]. (accessed: 17.04.2023).

[48] T. Yiu. "Understanding random forest." (2019), [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2. (accessed: 19.04.2023).

[49] scikit-learn.org. "Sklearn.tree.decisiontreeclassifier." (2023), [Online]. Available: scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. [Online Resource]. (accessed: 17.04.2023).

[50] scikit-learn.org. "1.11. ensemble methods." (2023), [Online]. Available: https://scikit-learn.org/stable/modules/ensemble.html%5C#forests-of-randomized-trees. [Online Resource]. (accessed: 17.04.2023).

[51] scikit-learn.org. "1.6. nearest neighbors." (2023), [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html. [Online Resource]. (accessed: 17.04.2023).

[52] scikit-learn.org. "Sklearn.neighbors.kneighborsclassifier." (2023), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html. [Online Resource]. (accessed: 17.04.2023).

[53] scikit-learn.org. "1.9. naive bayes." (2023), [Online]. Available: https://scikit-learn.org/stable/modules/naive%5C_bayes.html%5C#gaussian-naive-bayes. [Online Resource]. (accessed: 17.04.2023).

[54] scikit-learn.org. "Sklearn.naive_bayes.gaussiannb." (2023), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive%5C_bayes.GaussianNB.html. [Online Resource]. (accessed: 17.04.2023).

[55] scikit-learn.org. "Sklearn.ensemble.randomforestclassifier." (2023), [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Online Resource]. (accessed: 17.04.2023).

[56] SKF. "Shaft alignment tool tksa 11." (2023), [Online]. Available: https://www.skf.com/id/products/maintenance-products/alignment-tools/shaft-alignment/shaft-alignment-tool-tksa-11. (accessed: 06.02.2023).

[57] HIOKI. "Ac/dc current sensor ct6862-05." (2023), [Online]. Available: https://www.hioki.com/sg-en/products/current-probes/high-precision/id_6024. (accessed: 01.02.2023).

[58] HIOKI. "Power analyzer pw6001." (2023), [Online]. Available: https://www.hioki.com/in-en/products/power-meters/power-analyzer/id_6029. (accessed: 08.02.2023).

[59] F. Chen, F. Bai, C. Chen, S. Wang, and H. Zhang, "Research on double span rotor system driven by motorized spindle with coupling misalignment," *Advances in Mechanical Engineering*, vol. 11, p. 168 781 401 882 100, Apr. 2019. DOI: 10.1177/1687814018821009.

[60] J. A. Rosero, L. Romeral, J. A. Ortega, and E. Rosero, "Short-circuit detection by means of empirical mode decomposition and wigner–ville distribution for pmsm running under dynamic condition," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 11, pp. 4534–4547, 2009. DOI: 10.1109/TIE.2008.2011580.

[61] J.-K. Park and J. Hur, "Detection of inter-turn and dynamic eccentricity faults using stator current frequency pattern in ipm-type bldc motors," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 3, pp. 1771–1780, 2016. DOI: 10.1109/TIE.2015.2499162.

Appendix A

Python Script

A.1 Importing and arranging DataFrames

```
#%%#############     Importing & Combining the dataframes      ##############

# ##_________________________________DATA A:
# ### HEALTHY  - uses the 1 hr recording that is split into 3 files
# sampling_rate = 0.01
# c1=’#3399ff’
# c2=’#ffbb33’

# #Define the file names and directory path
# directory = r’C:\Users\Vilde\Documents\MASTENE\4. semester - MASTER\Python\MASTER coding’
# file_prefix = "0412020"
# file_suffix = ".csv"

# #Define the start and end file numbers
# start_file_num = 0
# end_file_num = 2
# file_nums = range(start_file_num, end_file_num + 1, 1)

# #Load the data from each file into a list of DataFrames
# data_frames_H = []
# for file_num in file_nums:
#     file_name = f"{file_prefix}{file_num}{file_suffix}"
#     df_H = pd.read_csv(f"{directory}/{file_name}", delimiter=’;’, decimal=’,’)
#     data_frames_H.append(df_H)

# #Concatenate the list of DataFrames into a single DataFrame
# unprocessed_healthy = pd.concat(data_frames_H, axis=0, ignore_index=True)
# print(’Concatenated Healthy (before): ’, unprocessed_healthy.shape)

# #Removing datapoints and then resetting the index
# unprocessed_healthy = unprocessed_healthy.drop(list(range(360000, 362002))
                                               + list(range(147000, 178491)))
# unprocessed_healthy = unprocessed_healthy.reset_index(drop=True)

# #Add a time column
# unprocessed_healthy[’Time (s)’] = unprocessed_healthy.index * sampling_rate
# print(’Concatenated Healthy (after): ’, unprocessed_healthy.shape)

# #Getting the length of the dataframe as healhty is shortest
# healthy_len = len(unprocessed_healthy)


# ### FAULTY DATA - 6x10 min recordings
# # Define the file names and directory path
# directory=r'C:\Users\Vilde\Documents\MASTENE\4. semester - MASTER\Python\MASTER coding'
# file_prefix = "04150"
# file_suffix = ".csv"

# #Define the start and end file numbers
# # start_file_num=100                    #comment out for cleaner data
# start_file_num = 200
# end_file_num = 700
# file_nums = range(start_file_num, end_file_num + 100, 100)

# #Load the data from each file into a list of DataFrames
# data_frames_F = []
# for file_num in file_nums:
#     file_name = f"{file_prefix}{file_num}{file_suffix}"
#     df_F = pd.read_csv(f"{directory}/{file_name}", delimiter=';', decimal=',')
#     data_frames_F.append(df_F)

# #Concatenate the list of DataFrames into a single DataFrame
# unprocessed_faulty = pd.concat(data_frames_F, axis=0, ignore_index=True)
# print('Concatenated Faulty (before): ', unprocessed_faulty.shape)

# #Limit the dataframe by the length of the healthy (balancing data)
# unprocessed_faulty = unprocessed_faulty.iloc[:healthy_len]

# #Add a time column
# unprocessed_faulty['Time (s)'] = unprocessed_faulty.index * sampling_rate
# print('Concatenated Faulty (after): ', unprocessed_faulty.shape)


##____________________________________DATA B:
##______FAULTY

#Defining variables
sampling_rate = 0.0001
c1='#F08080'
c2='#ABDA1A'
#Importing the .mat files
#Details: no demag, yes ITSC 6, 750 rpm, 50% load
directory = r'C:\Users\Vilde\Documents\MASTENE\4. semester - MASTER\Python\MASTER coding'
X_mat_F = sio.loadmat('ITSC_750_50_x.mat')
Ia_mat_F = sio.loadmat('ITSC_750_50_Ia')
Ib_mat_F = sio.loadmat('ITSC_750_50_Ib')
Ic_mat_F = sio.loadmat('ITSC_750_50_Ic')

#Transform into lists
X_list_F = X_mat_F['x'].tolist()[0]
Ia_list_F = Ia_mat_F['Ia'].tolist()[0]
Ib_list_F = Ib_mat_F['Ib'].tolist()[0]
Ic_list_F = Ic_mat_F['Ic'].tolist()[0]

#Combine lists in a dict to create new DataFrame
tot_dict_F = {'x': X_list_F, 'Ia': Ia_list_F, 'Ib': Ib_list_F, 'Ic': Ic_list_F}
raw_faulty = pd.DataFrame.from_dict(tot_dict_F, orient='columns')
print('Concatenated Faulty (before): ', raw_faulty.shape)

#Add a time column
raw_faulty['Time (s)'] = raw_faulty.index * sampling_rate

#Limit the length of the dataset
unprocessed_faulty = raw_faulty.head(200000)
print('Concatenated Faulty (after): ', unprocessed_faulty.shape)
```

```
##______HEALTHY
#Importing the .mat files
#Details: no demag, yes ITSC 6, 750 rpm, 50% load
X_mat_H = sio.loadmat('H_750_50_x.mat')
Ia_mat_H = sio.loadmat('H_750_50_Ia.mat')
Ib_mat_H = sio.loadmat('H_750_50_Ib.mat')
Ic_mat_H = sio.loadmat('H_750_50_Ic.mat')

#Transform into lists
X_list_H = X_mat_H['x'].tolist()[0]
Ia_list_H = Ia_mat_H['Ia'].tolist()[0]
Ib_list_H = Ib_mat_H['Ib'].tolist()[0]
Ic_list_H = Ic_mat_H['Ic'].tolist()[0]

#Combine lists in a dict to create new DataFrame
tot_dict_H = {'x': X_list_H, 'Ia': Ia_list_H, 'Ib': Ib_list_H, 'Ic': Ic_list_H}
raw_healthy = pd.DataFrame.from_dict(tot_dict_H, orient='columns')
print('Concatenated Healthy (before): ', raw_healthy.shape)

#Add a time column
raw_healthy['Time (s)'] = raw_healthy.index * sampling_rate

#Limit the length of the dataset
unprocessed_healthy = raw_healthy.head(200000)
print('Concatenated Healthy (after): ', unprocessed_healthy.shape)

## ______________COMBINING THE HEALTHY AND FAULTY CASE, rowwise
#Add a new column to the that indicactes if the data is Healthy/Faulty
unprocessed_healthy["Condition"] = "Healthy"
unprocessed_faulty["Condition"] = "Faulty"

#Concatenate the healthy and faulty dataframes
combined_unprocessed = pd.concat([unprocessed_healthy, unprocessed_faulty],
                                axis=0, ignore_index=True)
combined_unprocessed = combined_unprocessed.drop('Time (s)', axis=1)

#Add a new time column
combined_unprocessed['Combined Time (s)'] = combined_unprocessed.index * sampling_rate
```

A.2 Pre-Processing

```
#%%#############     Pre-processing the data      ##############

##### 1. Removing outliers with fillna

def zscore(column):
    upper_limit = column.mean() + 3 * column.std()
    lower_limit = column.mean() - 3 * column.std()
    #zscore = stats.zscore(column, axis=0, ddof=0)
    #print(zscore)
    return np.where((lower_limit <= column) & (column <= upper_limit), column, np.nan)

# # DATA A
# processed_healthy = unprocessed_healthy[['Irms1', 'Irms2',
                                          'Irms3', 'Time (s)']].copy()
# processed_faulty = unprocessed_faulty[['Irms1', 'Irms2',
                                        'Irms3', 'Time (s)']].copy()
# zscore_list = ['Irms1', 'Irms2', 'Irms3']
# df_list = [processed_healthy, processed_faulty]

# for DF in df_list:
#     for cols in zscore_list:
#         DF[cols] = zscore(DF[cols])
#         NaNs = DF[cols].isna().sum()
#         print(f’Zscore applied for {cols}, {NaNs} NaNs filled’)

#         plt.figure(figsize=(16,6))
#         if DF is processed_healthy:
#             plt.plot(unprocessed_healthy.index, unprocessed_healthy[cols],
                                                  label=’Original Data’, c=c1)
#         elif DF is processed_faulty:
#             plt.plot(unprocessed_faulty.index, unprocessed_faulty[cols],
                                                  label=’Original Data’, c=c1)

#         plt.plot(DF.index, DF[cols], label=’Filtered Data’, c=c2)
#         plt.xticks(fontsize=14)
#         plt.yticks(fontsize=14)
#         plt.xlabel(’Time [ms]’, fontsize=16)
#         plt.ylabel(’Current [A]’, fontsize=16)
#         plt.title(f’Applied Z-score to {cols}, filling in w/ NaNs’, fontsize=20)
#         plt.legend(loc=’upper right’, fontsize=12)
#         plt.show()


# DATA B
combined_processed=combined_unprocessed.copy()
processed_healthy = unprocessed_healthy.copy()
processed_faulty = unprocessed_faulty.copy()
zscore_list = [’Ia’, ’Ib’, ’Ic’]

for cols in zscore_list:
    combined_processed[cols] = zscore(combined_processed[cols])
    NaNs = combined_processed[cols].isna().sum()
    print(f’Zscore applied for {cols}, {NaNs} NaNs filled’)

    plt.figure(figsize=(16,6))
    plt.plot(combined_processed.index, combined_processed[cols],
                                        label=’Original Data’, c=c1)
    plt.plot(combined_processed.index, combined_processed[cols],
                                        label=’Filtered Data’, c=c2)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.xlabel(’Time [ms]’, fontsize=16)
    plt.ylabel(’Current [A]’, fontsize=16)
    plt.title(f’Applied Z-score to {cols}, filling in w/ NaNs’, fontsize=20)
    plt.legend(loc=’upper right’, fontsize=12)
    plt.show()

# #### 2. Filling in NaNs  -  this step takes a while

# # Data A:
# for DF in df_list:
#     for cols in zscore_list:
#         mean_cols_before = DF[cols].mean()
#         #create a list of the nans in the cols
#         nan_indices = DF[cols].index[DF[cols].isna()]
#         print(nan_indices.shape)
#         # Loop through each NaN value and fill with the mean of the non-NaN values
#         for nan in nan_indices:
```

```
#             non_nan_indices = DF[cols].index[(DF[cols].index < nan) & (DF[cols].notna())]
#             if len(non_nan_indices) == 0:
#                 non_nan_indices = DF[cols].index[(DF[cols].index > nan) & (DF[cols].notna())]
#             if len(non_nan_indices) > 0:
#                 left_value = DF.loc[non_nan_indices[-1], cols]
#                 right_value = DF.loc[non_nan_indices[0], cols]
#                 DF.loc[nan, cols] = (left_value + right_value) / 2

#         print(f’{DF}:’)
#         nans_after = DF[cols].isna().sum()
#         print(f’NaNs filled for {cols}, {nans_after} NaNs remaining’)
#         mean_cols_after = DF[cols].mean()
#         print(f’Mean value of {cols} before: {mean_cols_before}’)
#         print(f’Mean value of {cols} after: {mean_cols_after}’)


#Data B:
for cols in zscore_list:
    mean_cols_before = combined_processed[cols].mean()
    #create a list of the nans in the cols
    nan_indices = combined_processed[cols].index[combined_processed[cols].isna()]
    print(nan_indices.shape)
    # Loop through each NaN value and fill with the mean of the non-NaN values
    for nan in nan_indices:
        non_nan_indices = combined_processed[cols].index[(combined_processed[cols].index
                                            < nan) & (combined_processed[cols].notna())]
        if len(non_nan_indices) == 0:
            non_nan_indices = combined_processed[cols].index[(combined_processed[cols].index
                                            > nan) & (combined_processed[cols].notna())]
        if len(non_nan_indices) > 0:
            left_value = combined_processed.loc[non_nan_indices[-1], cols]
            right_value = combined_processed.loc[non_nan_indices[0], cols]
            combined_processed.loc[nan, cols] = (left_value + right_value) / 2

    print(f’{combined_processed}:’)
    nans_after = combined_processed[cols].isna().sum()
    print(f’NaNs filled for {cols}, {nans_after} NaNs remaining’)
    mean_cols_after = combined_processed[cols].mean()
    print(f’Mean value of {cols} before: {mean_cols_before}’)
    print(f’Mean value of {cols} after: {mean_cols_after}’)

####   5. Combining the dataset in a processed df (Data A)

#Data A
## ______________COMBINING THE HEALTHY AND FAULTY CASE, rowwise
# Add a new column to the that indicactes if the data is Healthy/Faulty
processed_healthy["Condition"] = "Healthy"
processed_faulty["Condition"] = "Faulty"

#Drop the old time columns
processed_healthy = processed_healthy.drop(’Time (s)’, axis=1)
processed_faulty = processed_faulty.drop(’Time (s)’, axis=1)


# Concatenate the healthy and faulty dataframes
combined_processed = pd.concat([processed_healthy, processed_faulty],
                                axis=0, ignore_index=True)

#Adding a new time columns
combined_processed[’New Time (s)’] = combined_processed.index * sampling_rate
```

A.3 Feature Extraction

```
#%%#############     Feature Selection      ##############

####          T I M E      D O M A I N     F E A T U R E S

# ##  D A T A    A:
# #Creating a new DataFrame
# Features = combined_processed.loc[:, [ 'Condition', 'Irms1', 'Irms2', 'Irms3']].copy()

# #Define a list of the target variables
# variable_list_Features = ['Irms1', 'Irms2', 'Irms3']
# window_size = 10

# #Add the features to each of the variables
# from scipy.stats import kurtosis, skew

# for col in variable_list_Features:
#     rolling_mean = Features[col].rolling(window_size, center=True).mean()
#     rolling_std = Features[col].rolling(window_size, center=True).std()
#     rolling_kurt = Features[col].rolling(window_size, center=True).apply(kurtosis)
#     rolling_skewness = Features[col].rolling(window_size, center=True).apply(skew)
#     rolling_crest_factor = Features[col].rolling(window_size,
                          center=True).apply(lambda x: x.max() / np.sqrt(np.mean(np.square(x))))
#     rolling_p2p = Features[col].rolling(window_size, min_periods=1,
                              center=True).apply(lambda x: x.max() - x.min(), raw=False)

#     #Add these features to the original DataFrame as new columns
#     Features[f'{col}_mean'] = rolling_mean
#     Features[f'{col}_std'] = rolling_std
#     Features[f'{col}_kurtosis'] = rolling_kurt
#     Features[f'{col}_skewness'] = rolling_skewness
#     Features[f'{col}_crest_factor'] = rolling_crest_factor
#     Features[f'{col}_p2p'] = rolling_p2p
#     print(f'Features for {col} added')

# #Dropping the columns with NaNs, as the rolling function creates 10 rows with NaNs
# print('Features containing Nans before: ', Features.isna().sum())
# Features.dropna(inplace=True)
# Features.reset_index(drop=True, inplace=True)
# print('Features containing Nans after: ', Features.isna().sum())

##  D A T A    B:
#Creating a new DataFrame
#TODO: change from unproccessed to processed
Features = combined_unprocessed.loc[:, [ 'Condition', 'Ia', 'Ib', 'Ic']].copy()

#Define a list of the target variables
variable_list_Features = ['Ia', 'Ib', 'Ic']

window_size = 10

#Add the features to each of the variables
from scipy.stats import kurtosis, skew

for col in variable_list_Features:
    #Define and add these features to the df
    rolling_mean = Features[col].rolling(window_size, center=True).mean()
    Features[f'{col}_mean'] = rolling_mean
    rolling_std = Features[col].rolling(window_size, center=True).std()
    Features[f’{col}_std’] = rolling_std
    rolling_kurt = Features[col].rolling(window_size, center=True).apply(kurtosis)
    Features[f’{col}_kurtosis’] = rolling_kurt
    rolling_skewness = Features[col].rolling(window_size, center=True).apply(skew)
    Features[f’{col}_skewness’] = rolling_skewness
    rolling_crest_factor = Features[col].rolling(window_size,
                    center=True).apply(lambda x: x.max()/np.sqrt(np.mean(np.square(x))))
    Features[f’{col}_crest_factor’] = rolling_crest_factor
    rolling_p2p = Features[col].rolling(window_size, min_periods=1,
                            center=True).apply(lambda x: x.max() - x.min(), raw=False)
    Features[f’{col}_p2p’] = rolling_p2p
    rolling_rms = Features[col].rolling(window_size,
                                    center=True).apply(lambda x: np.sqrt(np.mean(np.square(x))))
    Features[f’{col}_rms’] = rolling_rms
    rolling_clearance = Features[f’{col}_crest_factor’] / Features[f’{col}_rms’]
    Features[f’{col}_clearance’] = rolling_clearance
    rolling_shape = Features[f’{col}_rms’] / np.abs(Features[f’{col}_mean’])
    Features[f’{col}_shape’] = rolling_shape
    rolling_impulse = Features[col].rolling(window_size,
                                center=True).apply(lambda x: np.trapz(np.abs(x)), raw=False)
    Features[f’{col}_impulse’] = rolling_impulse

    print(f’Features for {col} added’)
    print(Features.columns.to_list())


#Dropping the columns with NaNs, as the rolling function creates 10 rows with NaNs
print(’Features containing Nans before: ’, Features.isna().sum())
Features.dropna(inplace=True)
Features.reset_index(drop=True, inplace=True)
print(’Features containing Nans after: ’, Features.isna().sum())
```

A.4 Principal Component Analysis

```
#%%#############     Principle Component Analysis      ##############
#Change Condition for PCA_Features
PCA_Features = Features.copy()

condition_mapping = {’Healthy’: 0, ’Faulty’: 1}
PCA_Features[’Condition’] = PCA_Features[’Condition’].map(condition_mapping)
df_list=[PCA_Features]

for df in df_list:
    #Import and scale the features
    X = df.iloc[:, 1:]

    #Standardization of the features
    X_mean = X.mean()
    X_std = X.std()
    Z = (X - X_mean) / X_std

    #Covariance Matrix
    cov_matrix = Z.cov()

    #Calculating eigenvalue and eigenvector
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
    print(’Eigenvalues:\n’, eigenvalues)
    print(’Eigenvalues Shape:’, eigenvalues.shape)
    print(’Eigenvector Shape:’, eigenvectors.shape)
```

```
#Sorting the values in descending order and calculating eigenvector
idx = eigenvalues.argsort()[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:,idx]

#Calculate the explained variance
explained_var = np.cumsum(eigenvalues) / np.sum(eigenvalues)
print(explained_var)

#Finding the number of PCA, edit the limitations of expl_var
n_components = np.argmax(explained_var >= 0.50) + 1           #above 50% explained variance
print(n_components)


#Creating the PCA component as a matrix
u = eigenvectors[:,:n_components]

#Ix_Features
# pca_component = pd.DataFrame(u, index=[’I’, ’Mean’,’STD’,
                                        # ’Kurtosis’, ’Skewness’,’Crest Factor’,
                                        # ’Peak to Peak’], columns = [’PC1’,’PC2’])
# #All_Features
# #DATA A
# pca_component = pd.DataFrame(u, index=[’Irms1’,’Irms2’,’Irms3’,’Irms1_mean’,
#                                         ’Irms1_std’,’Irms1_kurtosis’,’Irms1_skewness’,
#                                         ’Irms1_crest_factor’,’Irms1_p2p’,’Irms2_mean’,
#                                         ’Irms2_std’,’Irms2_kurtosis’,’Irms2_skewness’,
#                                         ’Irms2_crest_factor’,’Irms2_p2p’,’Irms3_mean’,
#                                         ’Irms3_std’,’Irms3_kurtosis’,’Irms3_skewness’,
#                                         ’Irms3_crest_factor’,’Irms3_p2p’],
#                               columns = [’PC1’,’PC2’,’PC3’])

#DATA B
pca_component = pd.DataFrame(u,
index=[’Ia’,’Ib’,’Ic’,’Ia_mean’,’Ia_std’,’Ia_kurtosis’,
                                ’Ia_skewness’,’Ia_crest_factor’,’Ia_p2p’,’Ia_rms’,
                                ’Ia_clearance’,’Ia_shape’,’Ia_impulse’,’Ib_mean’,
                                ’Ib_std’,’Ib_kurtosis’,’Ib_skewness’,’Ib_crest_factor’,
                                ’Ib_p2p’,’Ib_rms’,’Ib_clearance’,’Ib_shape’,’Ib_impulse’,
                                ’Ic_mean’,’Ic_std’,’Ic_kurtosis’,’Ic_skewness’,
                                ’Ic_crest_factor’,’Ic_p2p’,’Ic_rms’,’Ic_clearance’,
                                Ic_shape’,’Ic_impulse’], columns = [’PC1’,’PC2’,’PC3’,’PC4’])

#Plotting a heatmap
plt.figure(figsize =(8, 6))
sns.heatmap(pca_component,  cmap=’inferno’,
        center = 0,
        annot=True)
plt.title(’PCA Component’)
plt.tight_layout()
plt.show()


###Perfroming a PCA analysis using sklearn PCA
from sklearn.decomposition import PCA

#Set the number of components
pca = PCA(n_components = 3)
pca.fit(Z)
x_pca = pca.transform(Z)
```

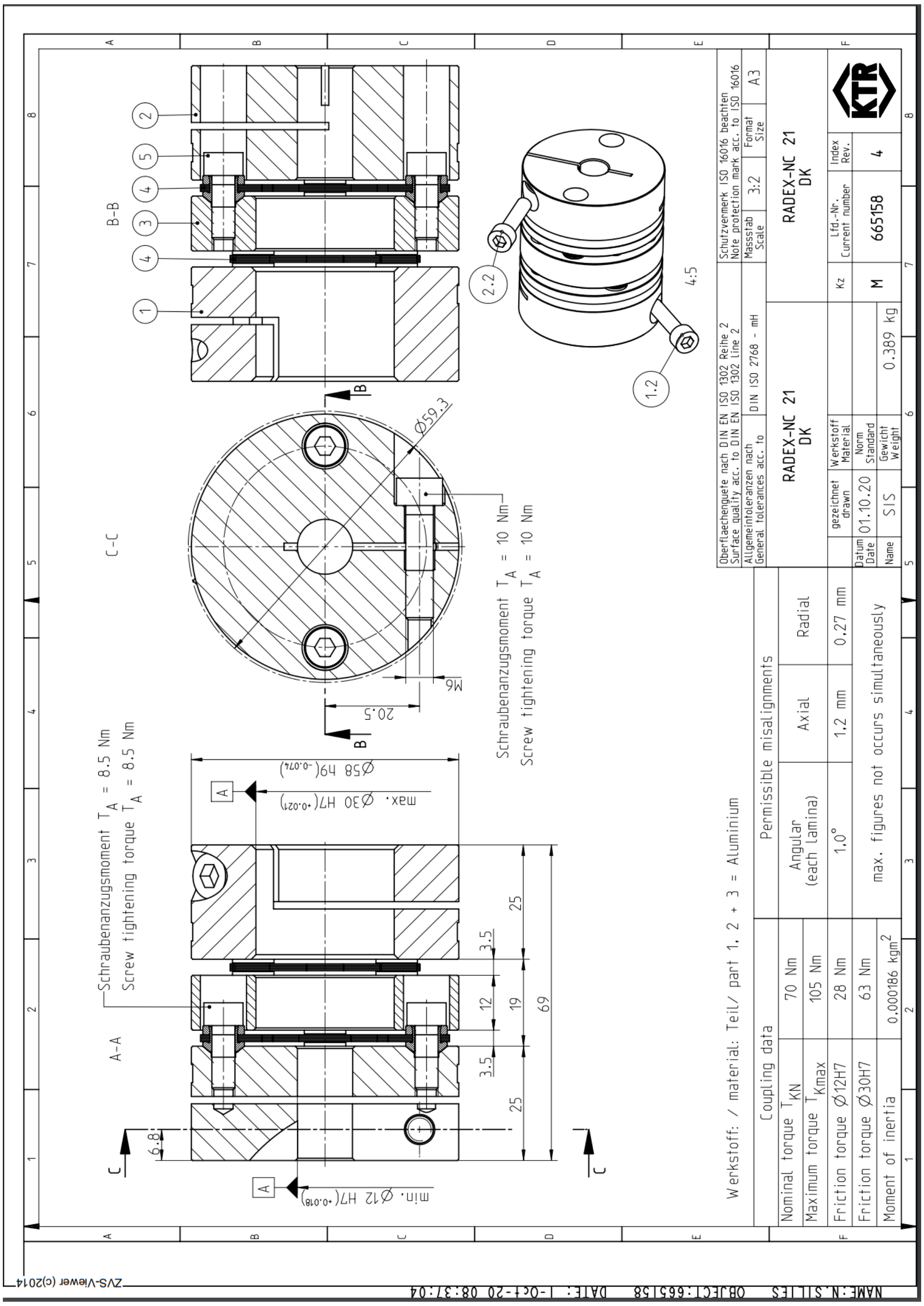
```
    #Create a new DataFrame
    X_pca = pd.DataFrame(x_pca, columns = ['PCA1','PCA2', 'PCA3'])
    # X_pca = pd.DataFrame(x_pca, columns = ['PCA1','PCA2','PCA3','PCA4'])
    print(X_pca.head())


    #Projecting the dataset by dot product
    Z_pca = Z @ X_pca
    Z_pca = pd.DataFrame(Z_pca.values, columns = ['PCA1','PCA2'])
    print(Z_pca.head())


#Change me later after running all features
#Add the PCA columns to X_pca
X_pca['Condition'] = Features['Condition']
# columns_to_add = ['Ia_clearance_indicator', 'Ib_clearance_indicator', 'Ic_clearance_indicator',
                  # 'Ia_shape_indicator', 'Ib_shape_indicator', 'Ic_shape_indicator']
columns_to_add = ['Condition', 'Irms1_kurtosis_indicator', 'Irms2_kurtosis_indicator',
                  'Irms3_kurtosis_indicator', 'Irms1_skewness_indicator',
                  'Irms2_skewness_indicator', 'Irms3_skewness_indicator' ]
X_pca[columns_to_add] = All_Features[columns_to_add]
```

A.5 Machine Learning Models

```
#%%#######     MACHINE LEARNING - Training and testing the data     ##########
##    C L A S S I F I C A T I O N

from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB

#DATA A:
# ML_df_list = [All_Features]
ML_df_list = [X_pca]

for data in ML_df_list:
    # 1) All_Features
    # y_list_ML =
    #   ['Condition', 'Irms1_mean_indicator','Irms1_std_indicator','Irms1_kurtosis_indicator',
    #    'Irms1_skewness_indicator','Irms1_crest_factor_indicator','Irms1_p2p_indicator',
    #    'Irms2_mean_indicator', 'Irms2_std_indicator', 'Irms2_kurtosis_indicator',
    #    'Irms2_skewness_indicator', 'Irms2_crest_factor_indicator', 'Irms2_p2p_indicator',
    #    'Irms3_mean_indicator', 'Irms3_std_indicator', 'Irms3_kurtosis_indicator',
    #    'Irms3_skewness_indicator','Irms3_crest_factor_indicator', 'Irms3_p2p_indicator']


    #2)For the two best features
    y_list_ML = ['Condition', 'Irms1_kurtosis_indicator', 'Irms2_kurtosis_indicator',
                 'Irms3_kurtosis_indicator', 'Irms1_skewness_indicator',
                 'Irms2_skewness_indicator', 'Irms3_skewness_indicator' ]


    for target in y_list_ML:
        # Defining and scaling X (0-1)
        # X = data[[ 'Irms1', 'Irms2', 'Irms3','Irms1_mean','Irms1_std', 'Irms1_kurtosis',
        #            'Irms1_skewness','Irms1_crest_factor', 'Irms1_p2p', 'Irms2_mean',
        #            'Irms2_std','Irms2_kurtosis','Irms2_skewness','Irms2_crest_factor',
        #            'Irms2_p2p', 'Irms3_mean', 'Irms3_std', 'Irms3_kurtosis', 'Irms3_skewness',
        #            'Irms3_crest_factor', 'Irms3_p2p']].values
        X = data [['PCA1', 'PCA2', 'PCA3']].values
        # ##Defining and scaling X (0-1) for PCA
        scaler = MinMaxScaler()
        X_scaled = scaler.fit_transform(X)

        #Defining target values
        y = data[target].values

        #Encoding the Dependent Variable
        from sklearn.preprocessing import LabelEncoder
        labelencoder_y = LabelEncoder()
        y = labelencoder_y.fit_transform(y)

        #Splitting the data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

        #Creating the classifiers
        classifiers = {
            'Random Forest': RandomForestClassifier(n_estimators=100, random_state=0),
            'k-NN': KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2),
            'Naive Bayes': GaussianNB(),
            'Decision Tree': DecisionTreeClassifier(criterion='entropy', random_state=0)}

        #Training the classifiers
        for name, clf in classifiers.items():
            clf.fit(X_train, y_train)

        #Predicting y_pred using the classifiers
        for name, clf in classifiers.items():
            y_pred = clf.predict(X_test)
            print(f'{name} ran with {target} as y-value: ')
            print(f'{name} accuracy: {accuracy_score(y_test, y_pred)}')
            print(f'{name} confusion matrix:\n{confusion_matrix(y_test, y_pred)}\n')


# #DATA B

# ML_df_list = [All_Features]
# # ML_df_list = [X_pca]
# for data in ML_df_list:
#     #1) All_Features
#     # y_list_ML = ['Ia_mean_indicator','Ia_std_indicator','Ia_kurtosis_indicator',
#     #              'Ia_skewness_indicator','Ia_crest_factor_indicator','Ia_p2p_indicator',
#     #              'Ia_rms_indicator','Ia_clearance_indicator','Ia_shape_indicator',
#     #              'Ia_impulse_indicator','Ib_mean_indicator','Ib_std_indicator',
#     #              'Ib_kurtosis_indicator','Ib_skewness_indicator','Ib_crest_factor_indicator',
#     #              'Ib_p2p_indicator','Ib_rms_indicator','Ib_clearance_indicator',
#     #              'Ib_shape_indicator','Ib_impulse_indicator','Ic_mean_indicator',
#     #              'Ic_std_indicator','Ic_kurtosis_indicator','Ic_skewness_indicator',
#     #              'Ic_crest_factor_indicator','Ic_p2p_indicator','Ic_rms_indicator',
#     #              'Ic_clearance_indicator','Ic_shape_indicator','Ic_impulse_indicator']
#     #2) Condition
#     # y_list_ML = ['Condition']

#     #3)For the two best features
#     y_list_ML = ['Condition', 'Ia_clearance_indicator', 'Ib_clearance_indicator',
#                  'Ic_clearance_indicator','Ia_shape_indicator',
#                  'Ib_shape_indicator', 'Ic_shape_indicator']

#     for target in y_list_ML:
#         # Defining and scaling X (0-1)
#         X = data[[’Ia’,’Ib’,’Ic’,’Ia_mean’,’Ia_std’,’Ia_kurtosis’,’Ia_skewness’,
#         ’Ia_crest_factor’,’Ia_p2p’,’Ia_rms’,’Ia_clearance’,’Ia_shape’,’Ia_impulse’,
#         ’Ib_mean’,’Ib_std’,’Ib_kurtosis’,’Ib_skewness’,’Ib_crest_factor’,’Ib_p2p’,
#         ’Ib_rms’,’Ib_clearance’,’Ib_shape’,’Ib_impulse’,’Ic_mean’,’Ic_std’,’Ic_kurtosis’,
#         ’Ic_skewness’,’Ic_crest_factor’,’Ic_p2p’,’Ic_rms’,’Ic_clearance’,’Ic_shape’,
#         ’Ic_impulse’]].values

#         # ##Defining and scaling X (0-1) for PCA
#         # X = data [[’PCA1’, ’PCA2’, ’PCA3’, ’PCA4’]].values

#         scaler = MinMaxScaler()
#         X_scaled = scaler.fit_transform(X)

#         #Defining target values
#         y = data[target].values

#         #Encoding the Dependent Variable
#         from sklearn.preprocessing import LabelEncoder
#         labelencoder_y = LabelEncoder()
#         y = labelencoder_y.fit_transform(y)

#         #Splitting the data into training and testing sets
#         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)


#         #Creating the classifiers
#         classifiers = {
#             ’Random Forest’: RandomForestClassifier(n_estimators=100, random_state=0),
#             ’k-NN’: KNeighborsClassifier(n_neighbors=5, metric=’minkowski’, p=2),
#             ’Naive Bayes’: GaussianNB(),
#             ’Decision Tree’: DecisionTreeClassifier(criterion=’entropy’, random_state=0)
#         }
#         #Training the classifiers
#         for name, clf in classifiers.items():
#             clf.fit(X_train, y_train)

#         #Predicting y_pred using the classifiers
#         for name, clf in classifiers.items():
#             y_pred = clf.predict(X_test)
#             print(f’{name} ran with {target} as y-value: ’)
#             print(f’{name} accuracy: {accuracy_score(y_test, y_pred)}’)
#             print(f’{name} confusion matrix:\n{confusion_matrix(y_test, y_pred)}\n’)
```

Appendix B

Data sheet Couplings


A-A
C-C
B-B
Schraubenanzugsmoment T_A = 8.5 Nm
Screw tightening torque T_A = 8.5 Nm
Schraubenanzugsmoment T_A = 10 Nm
Screw tightening torque T_A = 10 Nm
6.8
min. Ø12 H7(+0.018)
max. Ø30 H7(+0.021)
Ø58 h9(-0.074)
A
B
C
3.5
12
3.5
25
19
25
69
20.5
M6
Ø59.3
1
4
3
4
5
2
2.2
1.2
4:5
Werkstoff: / material: Teil/ part 1, 2 + 3 = Aluminium
Oberflaechenguete nach DIN EN ISO 1302 Reihe 2
Surface quality acc. to DIN EN ISO 1302 line 2
Allgemeintoleranzen nach
General tolerances acc. to
DIN ISO 2768 - mH
Schutzvermerk ISO 16016 beachten
Note protection mark acc. to ISO 16016
Massstab Scale 3:2
Format Size A3
RADEX-NC 21 DK
RADEX-NC 21 DK
gezeichnet drawn
Werkstoff Material
Datum Date 01.10.20
Norm Standard
Name SIS
Gewicht Weight 0.389 kg
Kz M
Lfd.-Nr. Current number 665158
Index Rev. 4
KTR
ZVS-Viewer (c)2014
NAME: N.SILIES OBJECT: 665158 DATE: 1-Oct-20 08:37:04

Coupling data	
Nominal torque T_{KN}	70 Nm
Maximum torque T_{Kmax}	105 Nm
Friction torque Ø12H7	28 Nm
Friction torque Ø30H7	63 Nm
Moment of inertia	0.000186 kgm^2

Permissible misalignments		
Angular (each lamina)	Axial	Radial
1.0°	1.2 mm	0.27 mm
max. figures not occurs simultaneously		

Figure B.1. Data sheet for the VEM coupling.

Appendix C

Results

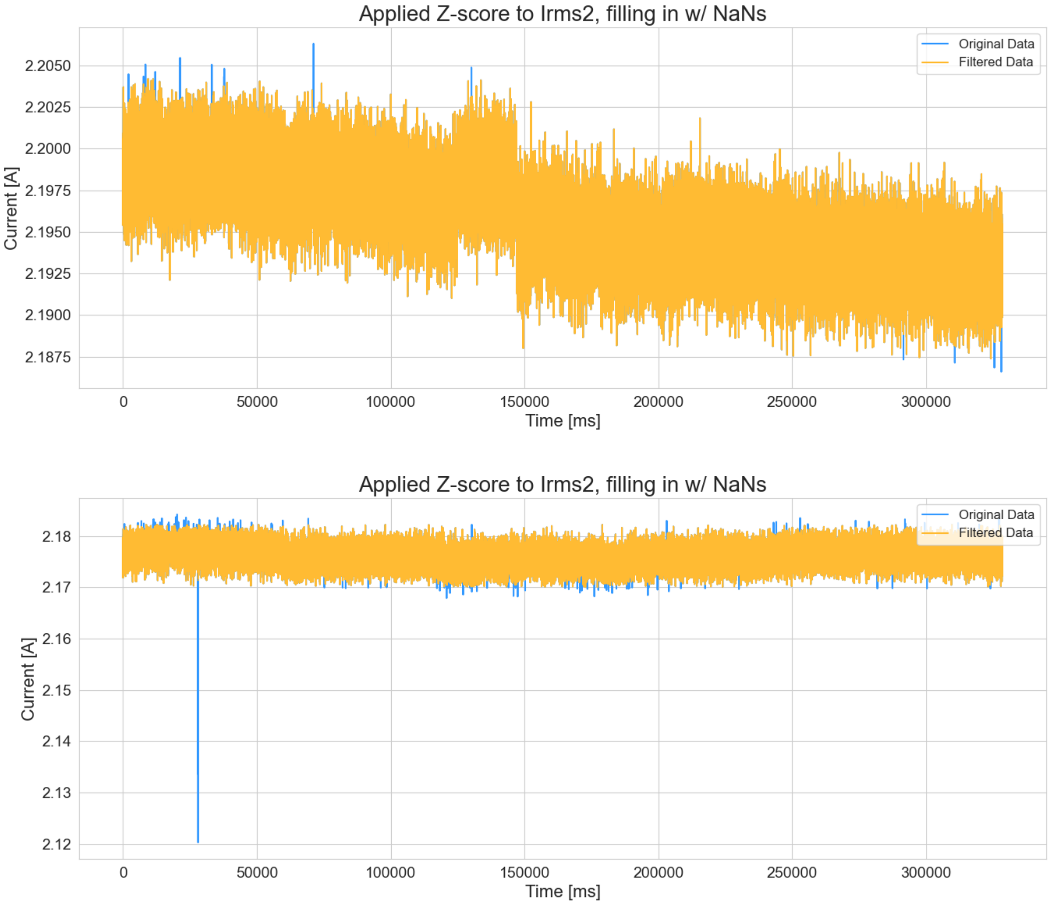


Figure C.1. Data A before and after applying Z-score filtering for 'Irms2'.

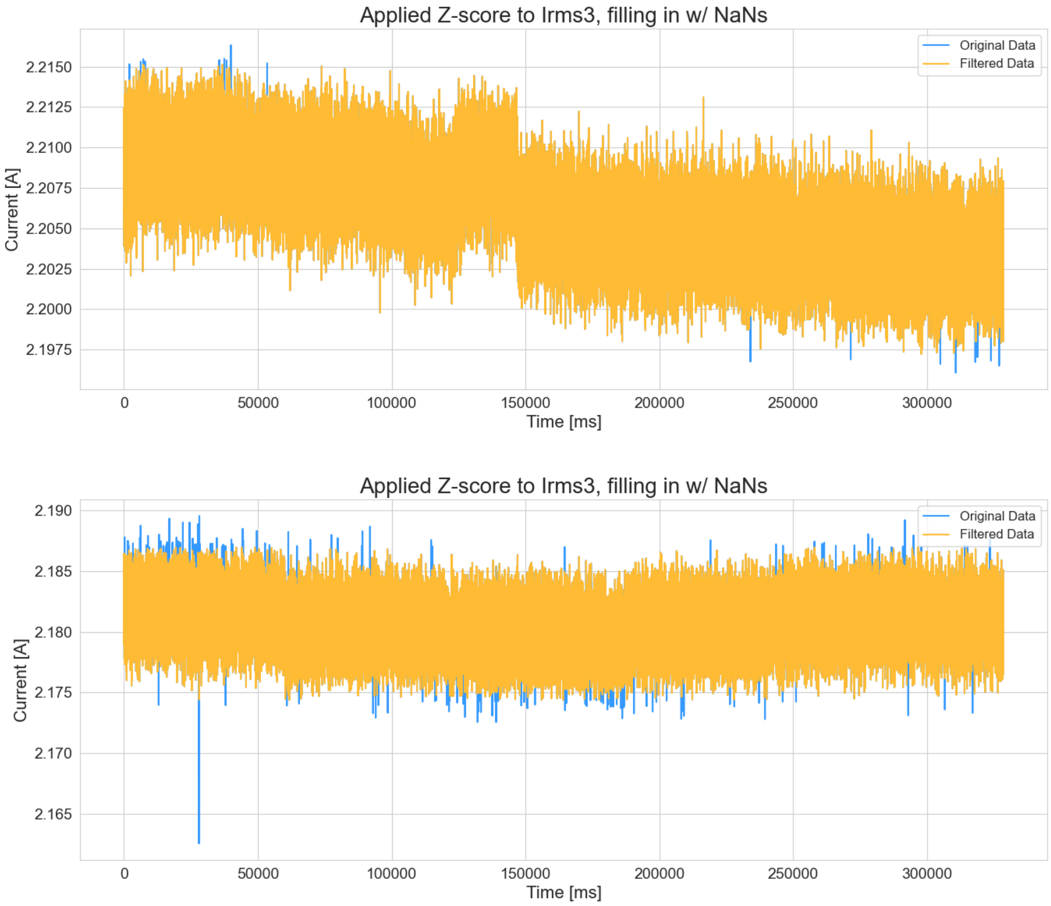


Figure C.2. Data A before and after applying Z-score filtering for 'Irms3'.

C.1 Covariance Heatmap Data A

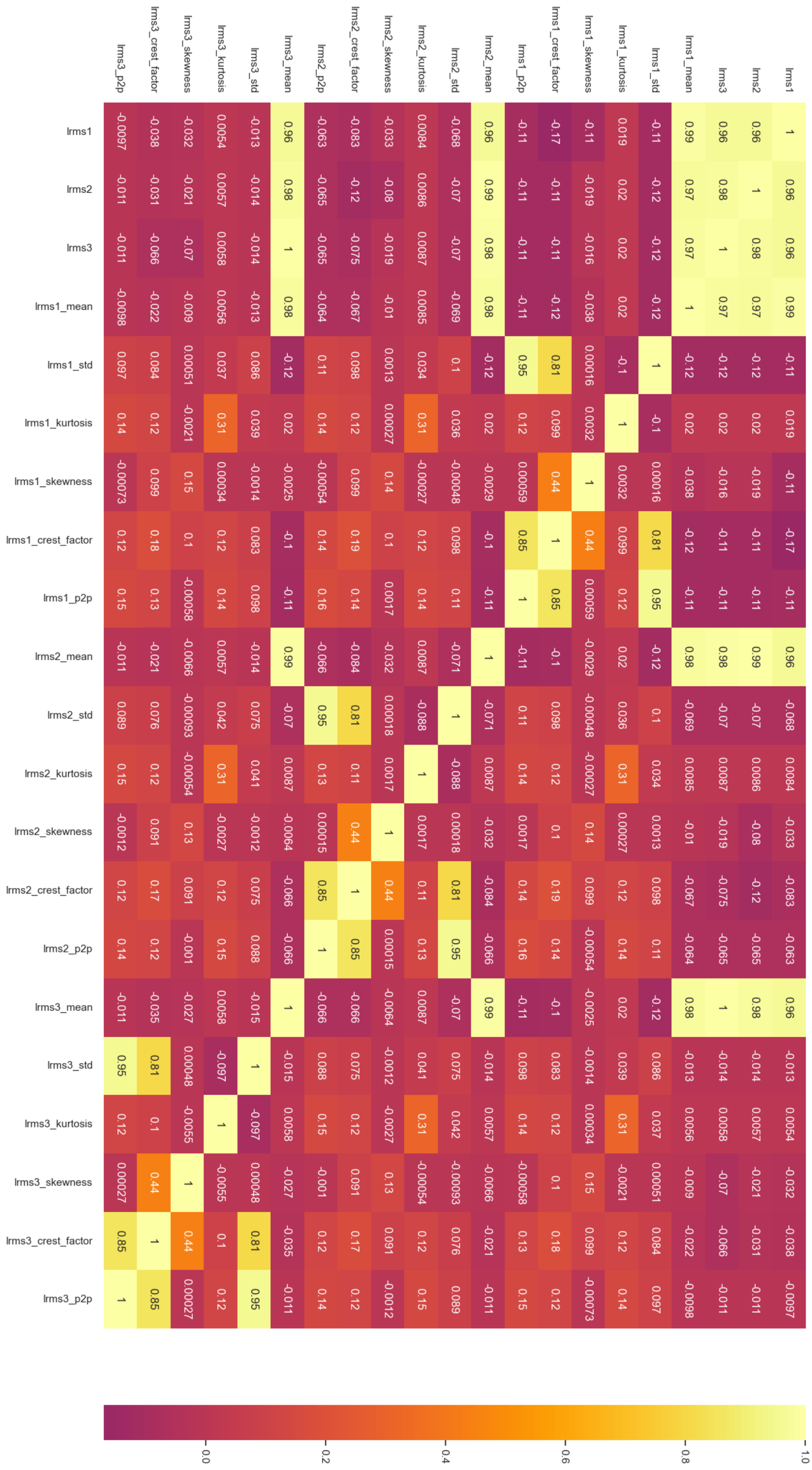


Figure C.3. Covariance heatmap for the applied features of Data A.

C.2 Covariance Heatmap Data B

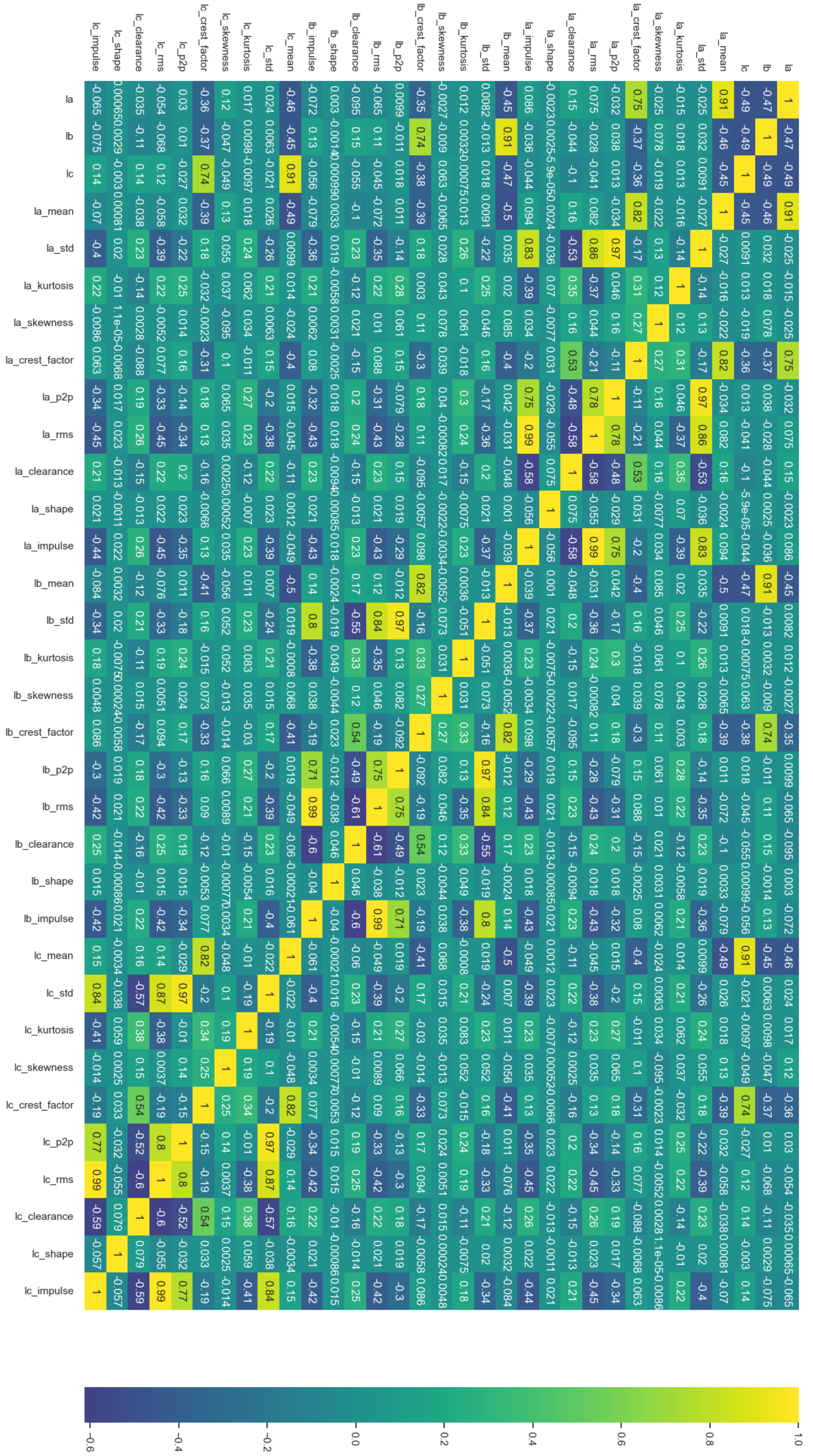


Figure C.4. Covariance heatmap for the applied features of Data B.