Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electric Power Engineering

Markus Fredrik Johansen

Image processing and machine learning application for fault diagnosis in synchronous motors

Master's thesis in Energy and Environmental Engineering
Supervisor: Arne Nysveen
Co-supervisor: Hossein Ehya
June 2022

NTNU
Norwegian University of
Science and Technology

Markus Fredrik Johansen

Image processing and machine learning application for fault diagnosis in synchronous motors

Master's thesis in Energy and Environmental Engineering
Supervisor: Arne Nysveen
Co-supervisor: Hossein Ehya
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electric Power Engineering

NTNU
Norwegian University of Science and Technology

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING
Master thesis

Image processing and machine learning application for fault diagnosis in synchronous motors

Supervisor:
Arne Nysveen
Co-Supervisor:
Hossein Ehya

Candidate: Markus Fredrik Johansen

June 27, 2022

Abstract

English:
Synchronous motor fault diagnosis provides insight into motor health and operational reliability. Diagnostic analysis is typically performed by examining changes in frequency components. This thesis presents an image processing and machine learning approach to detect faults in the motor automatically. The image processing method would use continuous wavelet transformation plots as input, generated on experimental and simulation data. The machine learning method, gradient boosting on decision trees, would use spatial and statistical image region descriptors as input. The effectiveness of the developed method produced a classification accuracy of 92%. The method is dependent on healthy data, but it is argued that with a more extensive sampling size, the method could become independent of healthy data and could potentially become applicable as a diagnostic tool in practical applications
Norsk:
Synkronmotor feildiagnose gir innsikt i motorhelse og driftssikkerhet. Diagnostisk analyse utføres vanligvis ved å undersøke endringer i frekvenskomponenter. Denne oppgaven presenterer en bildebehandlings- og maskinlærings tilnærming for å oppdage feil i motoren automatisk. Bildebehandlingsmetoden vil bruke kontinuerlige wavelet-transformasjonsplott som input, generert på eksperimentell og simuleringsdata. Maskinlærings metoden, gradientforsterkning på beslutningstrær, vil bruke romlige og statistiske bildeområde beskrivelser som input. Effektiviteten til den utviklede metoden ga en klassifiseringsnøyaktighet på 92%. Metoden er avhengig av sunn data, men det argumenters for at med en mer omfattende datainnsamling vil metoden kunne bli uavhengig av sunn data og potensielt bli anvendelig som et diagnostisk verktøy i praktiske anvendelser

0. Contents

0. List of Figures

1. Introduction

This section will contain a short and comprehensive summary of the faults that will be investigated later. These faults include inter-turn shortcircuit faults with increasing severity in the field winding. It will also contain information about the experimental laboratory setup at the NTNU campus. For this project, the experimental setup will change from a generator to a motor operation. Furthermore, it will contain information on the FE model and its simulation program. Finally, a summary of previous project work done by students at NTNU will be presented.

1.0.1 Scope of project

The objective of this Master's project is to develop an image processing method that produces data that feeds into a machine learning algorithm, which will provide an estimate of the type of fault and how severe the fault is. The method will be applied to a salient pole synchronous motor. The data will be gathered both experimentally and through simulations. The experiments will be carried out on a synchronous laboratory machine found at the faculty of information technology and electrical engineering at the NTNU campus. The simulation data will be gathered using a modified FE model created by a previous student. The collected data will be processed through signal processing. The signal processed data will serve as the input for image processing. The faults will cover increasing inter-turn shortcircuit(ITSC) faults. The fault severities of ITSC will range from 1ITSC, 2ITSC, 3ITSC, 7ITSC and 10ITSC. The data will be acquired during no-load operations and various full-load operations. The project will provide theory on image processing and machine learning. The theory will be based on relevant theory to explain the methods used in this task aptly. Furthermore, a literary survey on fault detection of electrical machines, with specific detail on machine learning based on image processing data, will be conducted. A survey on previous work on fault detection in synchronous motors will be performed as well. The conclusion of this work will hopefully be an algorithm that can extract the fault and its severity with machine learning.

1.0.2 Background

There has been a great expanse of electrical motors, of these, synchronous motors fit a particular role in the industry. Synchronous motors are estimated to reach a market value of 30.1 million USD by 2027[11]. They are typically used in cases where high performance and high reliability are necessary. These range from medical, aerospace, military and automotive[17], fulfilling roles such as compressors, rolling mills and reciprocating pumps. It is desirable to keep these running as efficiently as possible. During prolonged operation of the motor, faults may arise. If the machines continue to be utilized during faulty operation, it may come at the detriment of the efficiency. If the motor continues to work with faults, unexpected failure may arise. Unforeseen events like these can have tremendous consequences. Depending on the application, the consequences can range from costly maintenance or loss of life[41].

Efforts for the early detection of faults have been researched, and it is an ongoing process. Methods include, but not limited to, online magnetic flux monitoring[13], application of signal processing[37], stray magnetic flux monitoring[38], flux Monitoring for Demagnetization Diagnosis[35], by means of the zero-Sequence voltage component[42], supervised machine learning for temperature estimation[22][32], non-intrusive leakage flux-based methods[33], air gap flux-based detection and classification of damper bar and field winding faults[30] and so forth. This work develops a fault detection method based on image processing and machine learning. The goal is to be able to process plots generated by the signal processing of stray magnetic flux data.

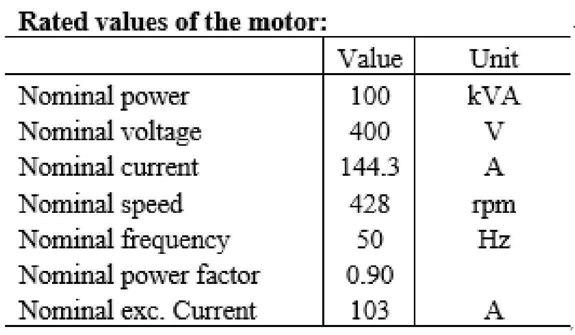
The work done in this thesis is a continuation of the work done in a specialization project performed in fall 2021[21]. That project was oriented around synchronous generators instead of synchronous motors. Due to the symmetry of the synchronous machine operation, the same methods proposed in the previous work should, in theory, also function for motor operation.

1.0.3 Experimental setup and the Finite Element Model

For this master thesis, data will be acquired experimentally and through simulation. The experimental data will be gathered using magnetic flux sensors from a 100 Kva synchronous machine. These sensors will register the stray magnetic flux generated during machine operation. During the tests, the machine will be operating as a motor. Henceforth in this thesis, the synchronous machine will be referred to as a motor. The following sections present the specifications of the test motor and the finite element model(FEM) that has been modified from previous work[13].

The experimental motor

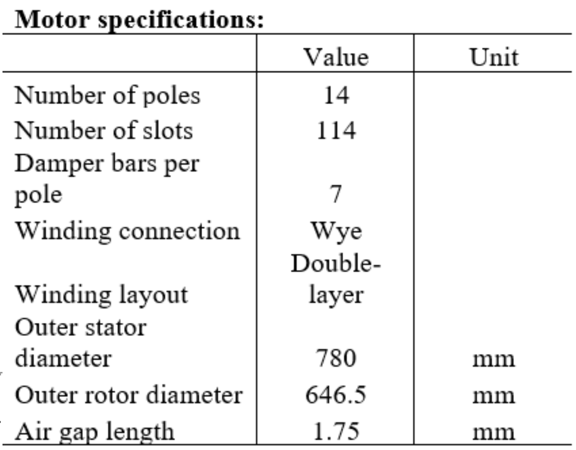
The experimental data are acquired using an experimental synchronous machine located in the laboratories at NTNU. The machine is modelled as a scaled-down version of a standard hydropower generator to achieve results that apply to real-life situations. Therefore, as with synchronous motors, it is a three-phase salient-pole synchronous machine. A unique feature of the machine is that it is modifiable. This allows for testing under different conditions that can resemble faulty operations in synchronous motors. The desired rotor windings can be short-circuited to resemble inter-turn short circuit faults. The stator frame can be moved and misaligned with the rotor axis to resemble an eccentricity fault. Whole damper bars can be removed to resemble broken damper bar faults, and so forth. Table 1.1 and 1.2 show design and geometry specifications of the motor[13]. Three sensors around the stator core will collect data on the stray flux during motor operation. The goal is to observe



Rated values of the motor:

	Value	Unit
Nominal power	100	kVA
Nominal voltage	400	V
Nominal current	144.3	A
Nominal speed	428	rpm
Nominal frequency	50	Hz
Nominal power factor	0.90	
Nominal exc. Current	103	A

Figure 1.1: Nominal values of the test motor



Motor specifications:

	Value	Unit
Number of poles	14	
Number of slots	114	
Damper bars per pole	7	
Winding connection	Wye	
Winding layout	Double-layer	
Outer stator diameter	780	mm
Outer rotor diameter	646.5	mm
Air gap length	1.75	mm

Figure 1.2: Motor specifications

their corresponding waveforms in order to detect faults.

FE model

The FE model used in this thesis was created by a prior student[13] and used in previous projects[38][13]. The purpose of those reports was to simulate faults in hydropower generators. The FEM model has been slightly modified in this report to simulate faults in synchronous motors. The building blocks are the same, but a voltage is supplied to the three phases. The FE model is compatible with and used in the ANSYS Maxwell2D 2021 release program. According to the ANSYS Maxwell webpage, the program is "an EM field solver for electric machines, transformers, wireless charging, permanent magnet latches, actuators and other electro-mechanical devices. It solves static, frequency-domain and time-varying magnetic and electric fields"[4]. Based on this description, the program can be an effective tool for solving low-frequency magnetic field simulations. The complete FE model can be shown in fig.1.3. The outer green circle represents the stator core. The stator core is filled with 38 blue(phase A), 38 red(phase B) and 38 green(phase C) rectangles that represent the three-phase stator windings. The blue circles surrounding the stator frame represent the stray magnetic flux sensors. The large inner green circle represents the rotor core. The reddish-brown rectangles in the rotor core represent the field windings surrounding the rotor poles. The reddish-brown circles within the rotor poles represent the damper bars.

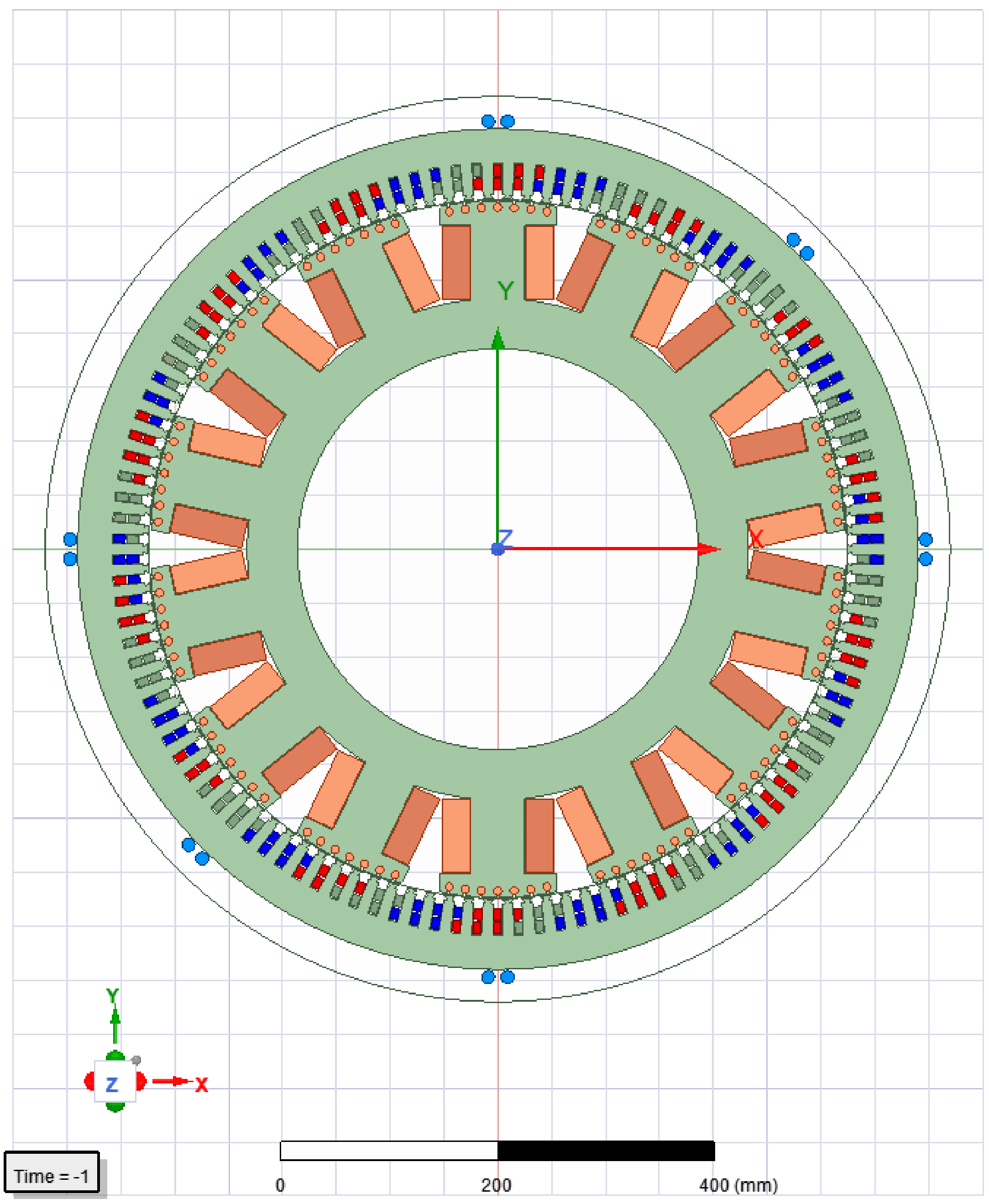


Figure 1.3: The complete FE model

1.0.4 Faults in sychronous motors

The following sections will summarise typical faults that may arise during the operation of synchronous motors[41]. The faults presented will range from inter-turn shortcircuit, air-gap eccentricity and broken damper bar faults. During faulty operation, these faults produce specific symptoms that can be observed through the investigation of the motor. Symptoms include a change in operating temperature, the motor's vibration, audible noise from the motor, increased losses, lower average torque, unbalanced air-gap voltages and unbalanced line currents[41].

Air-gap eccentricity

Air-gap eccentricity in a generator is a misalignment between the rotor and the stator. There are numerous causes for air-gap eccentricity[41], such as improper mounting, bearing wear, and bent rotor shaft. Once this misalignment occurs, the distance between the rotor and the stator is no longer uniform. As such, these faults cause changes in the stator currents[28][41].

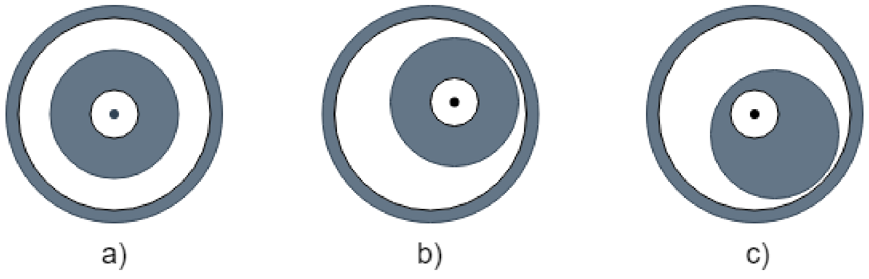


Figure 1.4: Types of air-gaps a) Concentric b) Static eccentricity c) Dynamic eccentricity, figure reproduced with permission[21]

Static eccentricity is characterized by its fixed minimal radial air gap, while in dynamic eccentricity, the minimal radial air gap follows with the turning of the rotor[28]. In some cases, there might be a combination of static and dynamic eccentricity, called mixed eccentricity. The asymmetry that occurs during eccentricity faults induces magnetic flux harmonics, which in turn induces harmonics in the current and voltage of the machine[38].

Shorted turns in stator windings

A typically occurring fault in salient pole synchronous motors is related to stator windings. In the case there are shorted turns, they may cause excessive stress to the machinery by producing heat in the stator coil and cause an imbalance in the current. The faults may occur due to insulation deterioration[40]. The deterioration occurs due to thermal ageing and mechanical force caused by high-speed rotation. This fault might not necessarily lead to machine failure, but prolonged operation with faulty stator windings could, over time, aggravate into machine failure as increased fault leads to increased bearing vibration and limit output. A fault might begin with a single winding fault. Over time, this fault might increase in severity as degradation and stress cause additional windings to fail. It is therefore beneficial to detect and repair this fault early.

Broken damper bars

Broken damper bars typically arise as a consequence of the transient time of a synchronous motor. During this time, the damper bars may be subjected to excessive amounts of current. This significant excess is prevalent during start-stop cycles, load changes and speed changes. If these changes of states occur frequently, it may cause a fault in the damper bars[41].

1.0.5 Previous work

This project continues the work done by the three previous specialization projects[13][37][38] conducted at a laboratory at NTNU. This laboratory contains an experimental generator that can be modified to mimic faults that typically occur in salient pole synchronous generators. In the report by Groth[13], an investigation of fault detection by on-line magnetic flux monitoring was performed. That analysis was done on the experimental laboratory generator and a finite element model simulation. That included faults for inter-turn short circuits and static eccentricity for both no-load and full-load operations. In the report by Skreien[37] and Sørheim[38], signal processing for fault detection in synchronous generators was examined. That included signal processing methods such as fast Fourier transform, Hilbert Huang transform, discrete wavelet transforms, continuous wavelet transform and short-time Fourier transform.

1.0.6 Limitations

Due to time constraints and knowledge limitations, no data was gathered on broken damper bar and eccentricity faults. Furthermore, simluated data on full-load motor operation could not be produced. Therefore, this thesis will focus primarily on analysing inter-turn shortcircuit faults with increasing severity

2. Signal processing

2.0.1 Continous wavelet transform

Wavelet analysis deals with the analysis of signals within short-duration finite energy functions. In other words, shorter waveforms at higher frequencies[26]. These waveforms are obtained by scaling the mother wavelet(also known as the reference wavelet). Wavelets are oscillating signals with zero average and energy, and these construct what is known as a wavelet dictionary. The relative transient wavelength used varies depending on the frequency component under study. This is achieved by shifting the scale and time of the mother wavelet, thus creating a cross-correlation between a signal and a family of wavelets. A large transformation value is obtained if the wavelet fits well with the signal at a specific scale and position. However, if the wavelet and signal do not match well, a low value for the transformation is obtained[19].

The notation for the continuous wavelet transfrom for a continuous signal is given in eq.2.1. Here a denotes the scale of the wavelet $\Psi^*_{a,b}$, while b is the temporal centre, while $X(t)$ is the signal that shall be analysed.

$$X(a,b) = \int_{-\infty}^{\infty} x(t)\Psi^*_{a,b}\,dt \tag{2.1}$$

The equation for the mother wavelet is as follows:

$$\Psi_{a,b}(t) = \frac{1}{\sqrt{a}}\Psi(\frac{t-b}{a}) \tag{2.2}$$

By plotting the wavelet transformations, it is possible to generate pictures based on the correlation between the wavelets at various scales and locations within the signal. In image processing, wavelet local maxima indicate the placement of edges, which are sharp and sudden changes in image intensity. The plots generate a scale-space approach for the continuous signal from which exact image approximations are reconstructed. At different scales, the geometry of these local maxima gives the contours of image structures of various sizes. This image processing trait is particularly effective for pattern recognition in the field of computer vision[26].

We have various wavelets used for signal processing. The selection of a specific wavelet depends on the application used[2][26]. We manipulate the wavelets in two ways: The first is scaling, and the second one is translation. For high frequencies, shorter wavelets are employed to improve time localization at the expense of frequency localization. For low frequencies, longer wavelets are used to improve frequency localization at the expense of time localization. This scheme allows us to achieve a variable time-frequency localization. A wavelet dictionary can then be constructed by varying the scales and translations of the mother wavelet[26]. The translations are chosen to meet the "admissibility" condition, meaning it is zero average and compact and approximate the type of signal contained by the time series. The transformation is calculated for various signal locations and wavelet scales, thus filling up the transformation plane. If the process is done smoothly and continuously (i.e., if scale and position are varied very smoothly), then the transform is called continuous wavelet transform[19]. If the scales and positions are changed discretely, the transform is called discrete wavelet transform. On the other hand, Fourier transform is a spectrum of one-dimensional array values. At the same time, wavelet transform is a spectrum of two-dimensional array values.

For those interested in the mathematical approach of the continuous wavelet transform, the proof and the resolution of its identity can be found in the "Ten Lectures on Wavelets"[2].

3. Image processing

Disclaimer

This chapter will present a brief and concise introduction to image processing. It will present how a computer interprets images, what a filter is and how to apply them to an image matrix, morphological filters, thresholding methods and how to find regions within an image. Most of this theory was researched during the specialization report in fall 2021[21]. Some formulations have been edited, some figures have been updated, and irrelevant material has been removed. Due to the implementation of machine learning, a particular focus on shape descriptors has been given. None of the information presented in this section is presented as new information. This is all based on a previous interpretation of the image processing theory that was done in fall 2021[21]. The theory was acquired through textbooks such as [27][29] and [31].

3.1 Introduction

Most people have acess to a device that allows for digital image capture and the ability to manipulate the images. This development has propelled the field of image processing significantly and it continues to do so. In the early days, only a handful of specialists with expensive equipment could feasibly process images. The specialists would usually be stationed at laboratories, earning image processing an exclusive position amongst the academics. For comparison, an image acquired on a common camera today would be too big for the most powerful personal computers back in the early 1990s. At that time, image processing techniques were invented and reinvented as there was no unified guidance for the field. Azrial Rosenfeld wrote the first textbook on image processing in 1969[23], and it acted as a collection of known methods in order to solidify the image processing theory. As an advancing field, image processing achieved great success when it helped map the surface of the moon. This mapping would later be used during the Apollo 13 mission[31]. With victories like these, great interest gathered and researchers wanted to tame the potential of image processing.

Today, image processing is considered one of the core disciplines in computer science and engineering. It is critical in research and is, therefore, a rapidly growing technology. Today, image processing is a technique or method for operating on an image. These operations are applied to extract useful information from characteristics or features. Because of this, image processing is widely used in medicine, remote sensing, media, entertainment and geological applications[7].

Programming with digital images

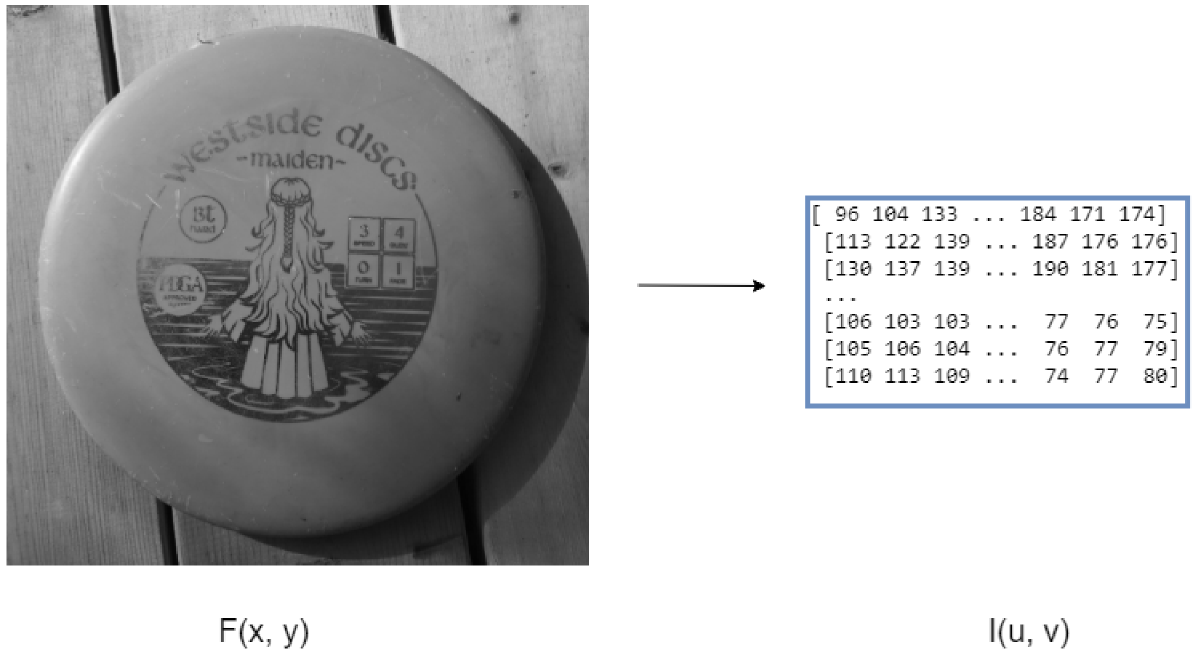
Programming is no longer locked behind specialists through great strides and technological advancement. Almost everyone has the opportunity to manipulate images, be it by employing filters, algorithms or extracting features. It can be intriguing to develop programs that can progress through an image and discover structures hidden from the human eye. Uncover features that aid in the early detection of symptoms, faults and environmental change can prove bountiful. To the trained eye, an image is nothing more than a matrix of values[6]. With the proper knowledge and expertise, it is possible to manipulate these matrixes to fit any demand

Image analysis in computer vision

The first challenge with image processing is that the computer does not intuitively know what is essential in an image and what is not. This challenge can seem trivial to the human eye, but the matrix the computer is comprised of an array of numbers. How can a computer tell the difference between a cat and a dog? A frown from a smile? Careful manipulation of the image will aid the computer in realizing the foreground and the background, which structures are essential and which structures are considered noise, and how to segment the structures in the image in a helpful way.

3.2 Image matrix

As stated before, an image is nothing more than a matrix to the computer. The size of a matrix is determined by its width M and height N. The width represents the number of columns in the matrix, while the height represents the number of rows. The resolution of an image is the direct result of how many array elements are present per measurement. Different applications have different resolution requirements. While print production might require a resolution of "dots per inch", satellite imaging might need "pixels per kilometre". However, in most cases, the elements of the image are square. This is because the resolution is the same horizontally and vertically[16].

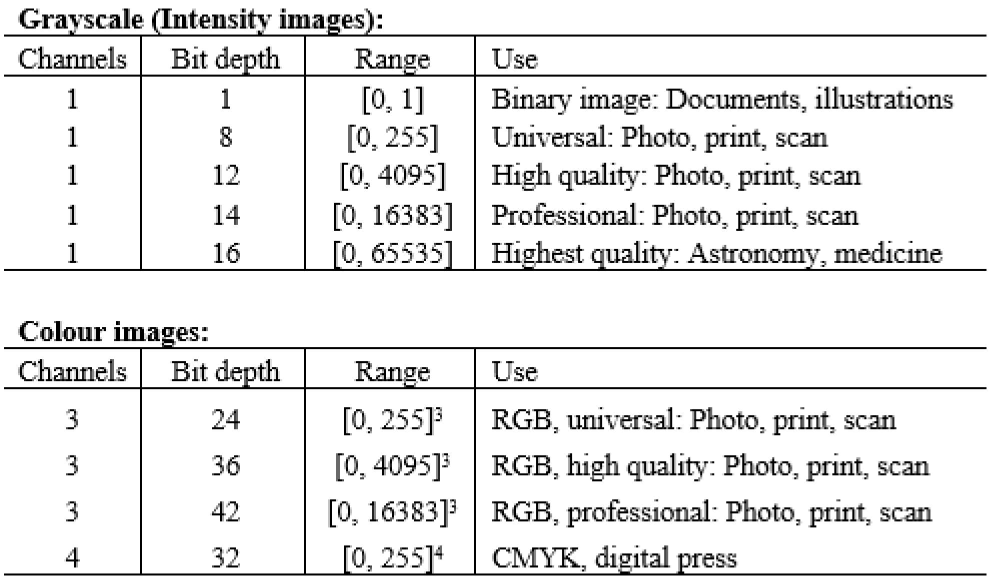


```
[ 96 104 133 ... 184 171 174]
 [113 122 139 ... 187 176 176]
 [130 137 139 ... 190 181 177]
 ...
 [106 103 103 ...  77  76  75]
 [105 106 104 ...  76  77  79]
 [110 113 109 ...  74  77  80]
```

Figure 3.1: Gray values matrix, shows the transformation from continous grayscale image F(x, y) to a discrete image I(u, v) with corresponding gray level intensity values

3.2.1 Bits and gray levels

The individual elements of an image matrix are referred to as pixels. Pixels represent intensity values within an image. The higher the pixel value, the greater the intensity. Within an image, a pixel can have a value of 2^k where k is the "bit depth". A binary image has a bit depth of 1 with the range [0, 1] where the images are either black or white. Grayscale images have a bit depth of 8 with the range [0, 255], where 0 is black, and 255 is white. RGB colour images typically have a bit depth of 24, comprising of three layers; red, green and blue. Each layer in RGB images has a bit depth of 8[31]. Various image types can be seen listed in fig.3.2



Grayscale (Intensity images):

Channels	Bit depth	Range	Use
1	1	[0, 1]	Binary image: Documents, illustrations
1	8	[0, 255]	Universal: Photo, print, scan
1	12	[0, 4095]	High quality: Photo, print, scan
1	14	[0, 16383]	Professional: Photo, print, scan
1	16	[0, 65535]	Highest quality: Astronomy, medicine

Colour images:

Channels	Bit depth	Range	Use
3	24	$[0, 255]^3$	RGB, universal: Photo, print, scan
3	36	$[0, 4095]^3$	RGB, high quality: Photo, print, scan
3	42	$[0, 16383]^3$	RGB, professional: Photo, print, scan
4	32	$[0, 255]^4$	CMYK, digital press

Figure 3.2: Common image types, shows image types based on amount of channels and bit depth, as well as common uses. Table has been reproduced with permission[21]

3.2.2 Grayscale images

Grayscale images are typically referred to as intensity images[27]. This type of image mainly focuses on highlighting an image's brightness or density. It is composed exclusively of different shades of grey. An advantage of grayscale images over RGB images is that less information is needed to produce the image. This, in turn, means that it requires less computing power to process in later stages. In order to convert an RGB image to grayscale, the following equation is applied to the pixels in the matrix:

$$Y = 0.2125R + 0.7154G + 0.0721B \tag{3.1}$$

Here, Y is the new grey pixel intensity value calculated by the R red pixel intensity value, the G green intensity value and the B blue pixel intensity value at that point in the image. This formula is used because the human eye perceives red, green and blue differently in terms of brightness[1].

Binary images

Binary images are generally classified as images with a bit depth of 1 with the value 0 or 1. This gives them the attribute that they are black or white. A grayscale image can be converted to binary with a thresholding operation by setting a threshold for the pixel intensity; see sec.3.4 for more on thresholding. Binary images are instrumental in cases requiring a distinct pixel value representation. This can be roads on a map, written documents, electronic printing and barcodes on the packaging.

Color images

To further elaborate on what was stated in sec.3.2.1, colour images are an additive colour system. These images start at black, with the RGB values set at 0. Colour gradually appears as more and more primary colour is added. Each colour channel in an RGB image has [0, 255] values. Mixing and combining different values for each colour channel creates the different colors[39].

3.3 Image statistics

A way to gauge the state of an image is through image statistics. It is possible to see if an image is underexposed, overexposed, its dynamic range, and even if image processing techniques have been applied to it. Analyzing the picture before applying image processing techniques is vital through methods such as histograms. The histogram can quickly tell if an image permanently loses information due to poor lighting and exposure. Any loss of information will skew the rest of the image processing process and prevent proper structural segmentation from occurring. The histogram can function as a "forensic tool" to aid in seeing if an image has too many errors, gaps and spikes. If this happens to be the case, it is recommended to reconsider the image acquisition method[29].

3.3.1 Histogram definition

Histograms are considered in image processing as the frequency distributions of intensity values that occur in the image[29]. The main function of a histogram is to provide clear statistical information in a condensed form. A grayscale image with its corresponding pixel intensity histogram is shown in fig.3.3

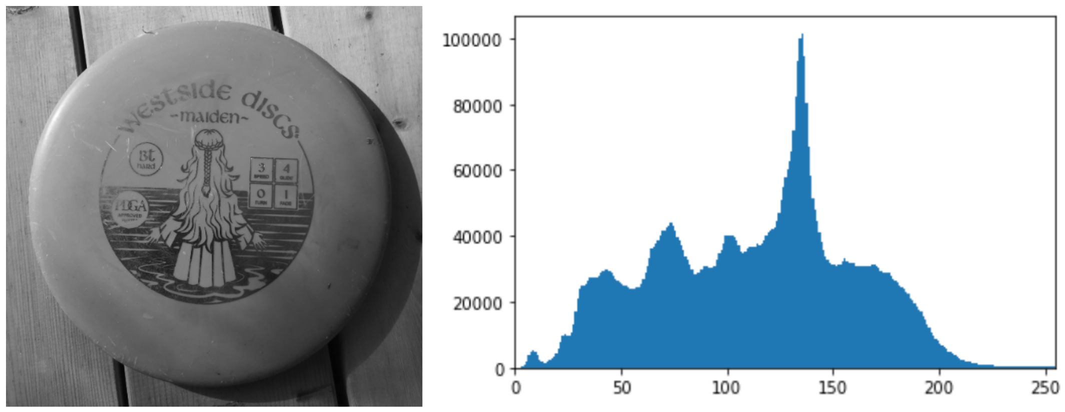


Figure 3.3: Histogram of a grayscale image, various peaks show in a condensed format the amount of pixels that share the same pixel intensity value

The following sections contain theory that is explicitly tailored towards grayscale images. The reasoning is that in image processing, many methods rely on converting images to grayscale before applying image processing techniques. So to generalize the theory as much as possible, grayscale images will be the type of images that will be considered.

For a grayscale image, the histogram contains K entries, where $K = 2^8 = 256$ in the case of an 8-bit depth image. This means that the image I will have pixel intensity values in the range $I(u, v) \in [0, K1]$. Since the histogram is an intensity plot, the entries can be defined as $h(i)$ with $h(i)$ being the number of pixels with a certain intensity value i for all $0 \leq i < K$. This can be reiterated more formally as:

$$h(i) = card(u, v)|I(u, v) = i, 0 \leq i < K \tag{3.2}$$

All the values at $h(0)$ have a value of 0, corresponding to black pixel intensities. As you increase $h(i)$, the entries will increase in pixel intensity values and appear brighter and brighter. This continues until $h(255)$ where all the pixel intensity values are 255, corresponding to white. The result of eq.6.1 is a 1-dimensional vector h with a length K

One significant drawback of histograms is the loss of the spatial dimension. A histogram cannot tell where a specific pixel is located in an image. Attempting to reconstruct an image based on its histogram is futile[29]. A comparison has been made in fig.3.4. If you were to compare the histogram of the different images, they would appear nearly identical.

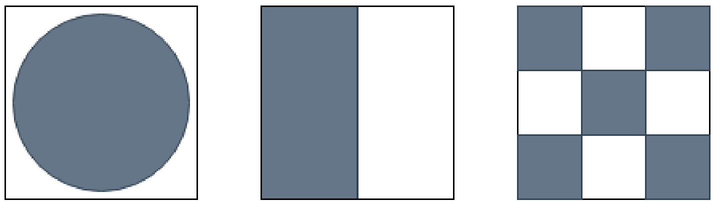


Figure 3.4: Three different images that have similar, if not identical, histograms. Reproduced with permission from[21]

An example of histogram analysis is provided in fig.3.5 where it is seen how contrast affects the original image. Higher contrast corresponds to a broader range of pixel intensity values. Lower contrast corresponds to pixels that share similar pixel intensity values to a greater extent. This can be seen in the figure as the high contrast image has the values evenly spread across the various possible i values. The low contrast histogram gathers the i values more densely and is displayed as significant peaks.

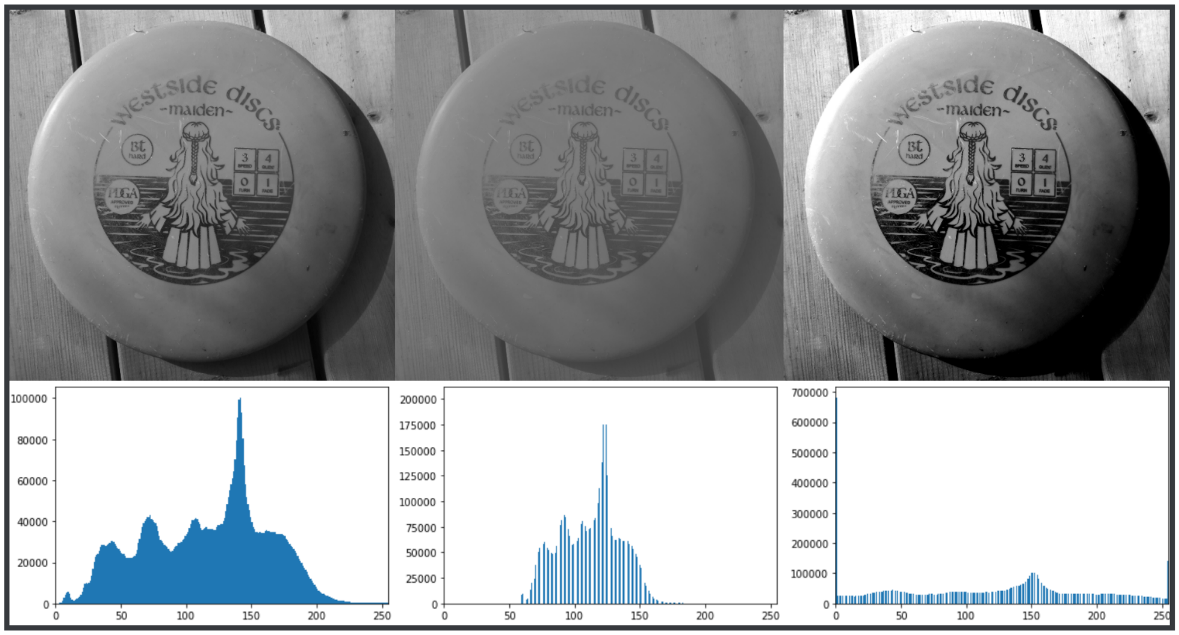


Figure 3.5: The histogram of the image, image on the left is unprocessed, image in the middle has reduced contrast, image on the right has increased contrast

3.4 Points operations

Point operation creates an empty image and uses the values of the original image as inputs. The operator will go through the matrix elements individually, perform operations, and gather the new values onto the empty image. It moves through the image from left to right, top to bottom. This will eventually form a different version of the old image, but with a change based on the point operator. Point operators get their name because they only require a single element as input. They do not rely on neighbouring pixels in any way. This attribute means that the structures, geometry or size of the image remains untouched[29]. The fig.3.5 shows the result of a point operation. The low contrast image was processed by a point operation where all the pixel elements were scaled down. The high contrast image was processed by a point operation where all the pixel elements were scaled up. Mathematically, a point operation can be presented as follows:

$$g(u, v) = T(f(u, v)) \tag{3.3}$$

Where $g(u, v)$ is the output image, T is the operator of intensity transformation and $f(u, v)$ is the input image. These operations typically come in two forms. Homogenous or nonhomogenous. Homogenous point operations are independent of image coordinates. This allows for operations such as gamma correction, colour transformations, inverting images and applying arbitrary intensity transformations. On the other hand, nonhomogenous point operations rely on the current image coordinate (u, v). This is useful for fixing uneven lighting during image acquisition[29]

Clamping to limit intensity values

When performing point operations, increasing the intensity values above or below the permissible values is theoretically possible. An 8-bit grayscale image can only contain the intensity values [0, 255]. Therefore it is necessary to contain any exceeding values within the permissible range. This can be done by setting an upper and lower limit. Any value exceeding the upper limit will be set to 255, while any value exceeding the lower limit will be set to 0.

This can be seen in the high contrast grayscale histogram in fig.3.5. The point operation for increasing contrast ran the risk of exceeding boundaries, so the program used for this operation set all the exceeding values to either 0 or 255. Therefore, we see almost 70,000 pixel elements with 0 intensity.

Threshold operations

Thresholding is a type of point operation that aims at "binarizing" an image. It can result in any two values, not just black and white. The concept of thresholding is to place an arbitrary boundary value. This could be any value between [0, 255]. Any intensity value higher than or equal to the thresholding value gets set to one value. Any intensity lower than the thresholding value gets set to another value. The end result is an image with only two intensity values, see fig.3.6. This method is excellent for separating the foreground from the background. Careful application of thresholding techniques can find the optimal thresholding value to preserve as much of the foreground structure as possible. Such techniques will be revisited later in sec.3.7

Point operations and histograms

Different point operations have different effects on the resulting histogram. As seen in fig.3.5, varying the contrast flattens and raises the intensity values. Inverting an image would flip the histogram. Raising the brightness by a constant value will shift all the intensity values to the right. An example of how binarizing an image through thresholding is shown in fig.3.6.

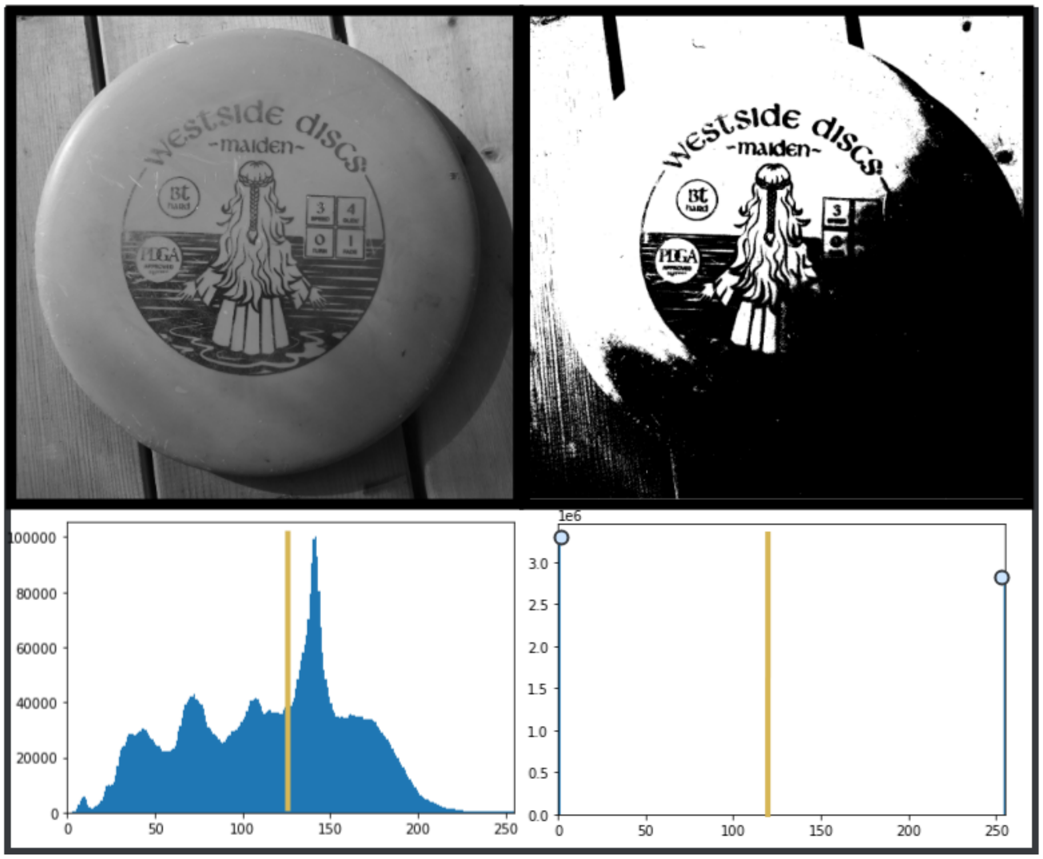


Figure 3.6: Shows the image before the thresholding, where the optimal threshold was found, with the image and its corresponding histogram after the threshold operation

In practicality, the binarization point operation acts as shown in fig.3.7.

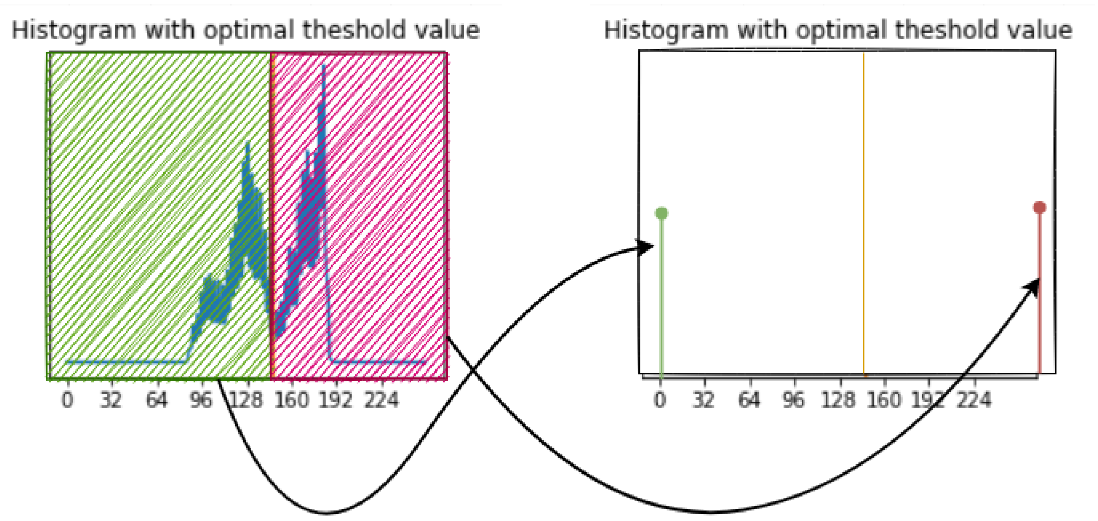

Histogram with optimal theshold value
Histogram with optimal theshold value

Figure 3.7: The histogram before and after binarization, reproduced with permission from[21]

A downside to point operations is their irreversibility. Some point operations lead to the merging of i values. This merging is irreversible as it is nearly impossible to predict which matrix elements used to belong to which i value. The point operations have then reduced the image's dynamic range and caused permanent loss of information[29].

3.5 Filters

Filter operations are a step further than point operations. The same logic applies to creating new pixel intensity values based on the old intensity values. This time, new pixel intensity values are created based on their neighbouring pixels instead of focusing on individual pixels simultaneously. Operations like these move around the image as a "stamp" that produces new pixel intensity values based on the weights on the stamp. These operations do not remove or destroy the geometry of the image, but they can sharpen edges, blur the images and much more[29]

A case of filter operation is to smooth images. Structures and geometry within an image are created by what is perceived as edges and lines. Edges and lines signify when the pixel intensity value rapidly shifts from one pixel to another. However, if the pixel intensity values have a smooth transition, the edges and lines appear to fade. This effect will make the image appear blurred. A simple implementation of smoothing filters is to make the "stamp" take the average of the neighbouring pixels and then create the new pixel intensity value.

In most cases of the image, a pixel will have eight neighbouring pixels. A typical filter can have a stamp with a size of 3x3. Therefore nine pixels in total are included in the calculations. A smoothing filter could theoretically look like the following[29]:

$$I'(u,v) \leftarrow \frac{1}{9} \cdot \sum_{j=-1}^{1} \sum_{i=-1}^{1} I(u+i, v+j) \tag{3.4}$$

Although the previous example used a stamp of size 3x3, filter stamps can have a wide range of sizes and shapes. It can be 4x4, 5x5, 6x6, even NxN. It could be circular, unevenly weighted, and the central pixel does not have to be a part of the new pixel intensity values calculations.

This variety in shape, size and weights allows for tremendous amounts of freedom when determining a filter. For the sake of clarity, it is vital to classify filters into categories. The broadest classifications are linear and nonlinear filters. Linear filters calculate the source pixel with a linear method. Nonlinear filters calculate the source pixel with a nonlinear method[29].

3.5.1 Linear filters

It is called a linear filter when computing a source pixel intensity based on the neighbouring intensities in a weighted summation. Linear filters can also be the product of linear convolutions. The weights can be adjusted to meet a plethora of demands. The smoothing filter in eq.3.4 is such a linear filter, where all the neighbouring pixels have equal weights[31].

Filter kernel

What has been referred to so far as a filter "stamp" is, in theory, a filter kernel. The kernel is an instruction manual on calculating a new value for the source pixel(marked with parenthesis in eq.3.7) using weights. "The filter kernel can be defined as the filter matrix $H(i, j)$ where the size is equal to the filter region and the elements contained in that region specify the weights for the corresponding pixel intensity values on the computation"[31][21]. The smoothing filter in eq.3.4 can be represented by the filter kernel:

$$H = \frac{1}{9} \cdot \begin{bmatrix} 1 & 1 & 1 \\ 1 & (1) & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.5}$$

The averaging factor here is nine as the filter kernel itself is 3x3. An example of the averaging filter can be seen in the A.1. The source pixel is the central one in the kernel, but this does not have to be the case for other filters. In order to apply the filter for a typical 3x3 filter kernel, we can combine Eq.3.4 and Eq.3.7 into a new equation[29]:

$$I'(u, v) \leftarrow \frac{1}{9} \cdot \sum_{j=-1}^{1} \sum_{i=-1}^{1} I(u + i, v + j) \cdot H(i, j) \tag{3.6}$$

Gaussian filter

A gaussian filter is considered a low pass filter than can smooth an image and reduce noise. This functionality is due to its odd sized symmetric kernel as seen here:

$$H = \frac{1}{16} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 2 & (4) & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3.7}$$

This type of kernel will have a smoothing effect on the image because the elements further away from the kernel centre have less weight. Therefore, it goes easy on drastic brightness changes, indicating edges. The Gaussian filter approximates the Gaussian function and contains the characteristic "bell-shaped" function[29]. An example of the Gaussian filter can be seen in the A.2.

3.5.2 Non-linear filters

Nonlinear filters are any filter that is not considered a linear filter. The procedure for applying the filter remains the same as with the linear filters. The filter kernel will move through the image and calculate new pixel intensity values based on the desired kernel. The result will be a new, filtered version of the original image. The most straightforward application for the nonlinear filters is the maximum and minimum filters[31]. The equations for the maximum and minimum filters are as follows[29]

$$I'(u,v) = \min_{(i,j)\in R}(I(u+i, v+j)) \tag{3.8}$$

$$I'(u,v) = \max_{(i,j)\in R}(I(u+i, v+j)) \tag{3.9}$$

Where $I'(u,v)$ is the filtered image, $I(u,v)$ is the old image, and R is the filter region. The maximum and minimum filters can be seen in A.3 and A.4. In order to accentuate the effects of the maximum and minimum filter, the noise was applied to the grayscale image. This noise is also referred to as "salt and pepper", a technique for adding noise to an image. Salt and pepper add random white and black dots across the image. When using the maximum filter, all the bright spots in the image get expressed to a greater degree. Hence all the white dots from the salt and pepper technique will appear brighter and dominate the image more. Reversibly, when using the minimum filter, all the black dots get expressed to a greater degree. Therefore the image will appear overall darker[31].

Median filter

Filtering away all the noise of an image is a near-impossible task. Some techniques allow for good approximations, but the noise removal operation cannot tell what is noise and what is structures within an image. One powerful noise removal technique is the median filter. This filter calculates the median by first sorting all the pixel intensity values in ascending order and then proceeds to replace the source pixel with the median pixel intensity value. The median filter is particularly useful against salt and pepper noise[31], as seen in fig.3.8

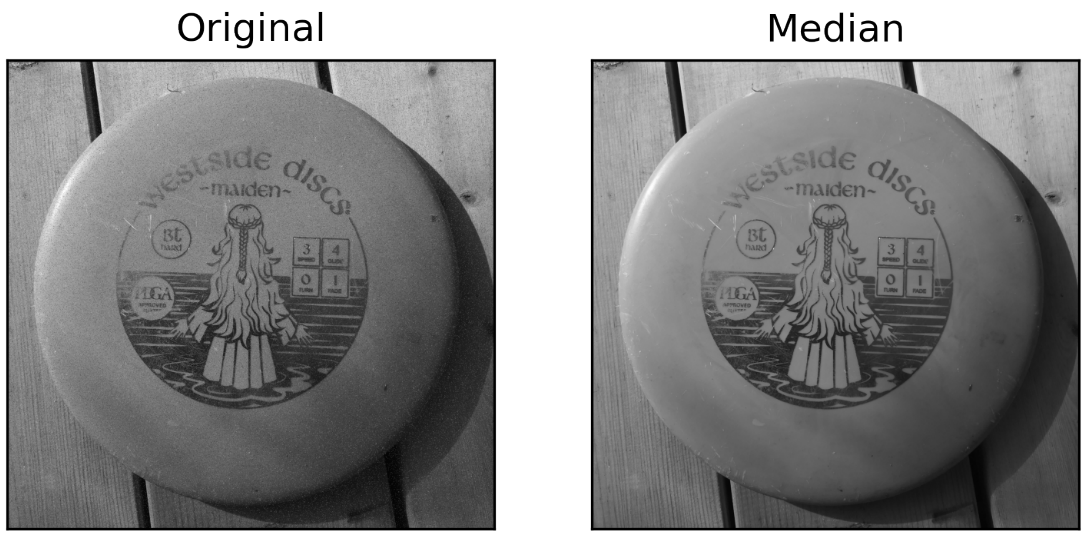


Figure 3.8: Median filtering and noise removal

3.5.3 Edges and contours

Edges and contours are what create the structures within images. Therefore, in image processing, it is important to properly apply edge detection methods to express the desired image structures clearly. For an edge detection operation, edges and contours are characterised by sudden and rapid shifts in pixel intensity values. When such a rapid change occurs, an edge has likely been located. The greater the shift, the greater the probability of an edge. The pixel intensity values can be considered the rate of change in intensities and are therefore susceptible to mathematical operations such as derivation. Edge detection operators such as Sobel, Prewitt, Canny and Laplace consider the derivation of pixel intensity values[29]. Examples of the four edge operators can be seen in A.[5 - 8]

3.5.4 Grayscale morphology

Grayscale morphology falls under the category of morphological transformations. These types of transformations aim at accentuating image structures by manipulating the structures within an image. This is typically done with binary images, but it is feasible to perform morphological operations on grayscale and colour images. Examples of morphological operations are erosion, dilation, opening and closing.

Erosion makes detected structures smaller by slightly reducing their edges. Dilation makes detected structures more significant by slightly increasing their edges. Opening is a combination of erosion and dilation where the resulting image will have structures appear more individualistic. Closing is also a combination of dilation and erosion where the resulting image will have more unified structures. In order to apply morphological filters on grayscale images, the min and max operators are used, and the kernel is a set of 2D functions instead of numerical weights. In order to apply morphological filters on colour images, the same operation as with grayscale images is performed on all the colour channels before being layered on top of each other again[31].

3.6 Regions in binary images

3.6.1 Finding connected regions

By finding connected regions in an image, it is possible to discern structures. The structures are typically used for investigating properties by analyzing them for various properties such as shapes and sizes. In order to do an analysis of an image, it is vital to properly find the connected regions. This can be achieved with methods such as iterative flood-filling, sequential region labeling method and recursive flood filling. These methods are chosen based on their complexity, computational costs and results. They differ greatly in terms of theory, but when it comes to practical application, they produce similar results.

When locating connected regions, it is important to decide what is considered a pixel neighbroughood. In most cases, a pixel neighbourhood is either 4-pixel connected or 8-pixel connected, see fig.3.9

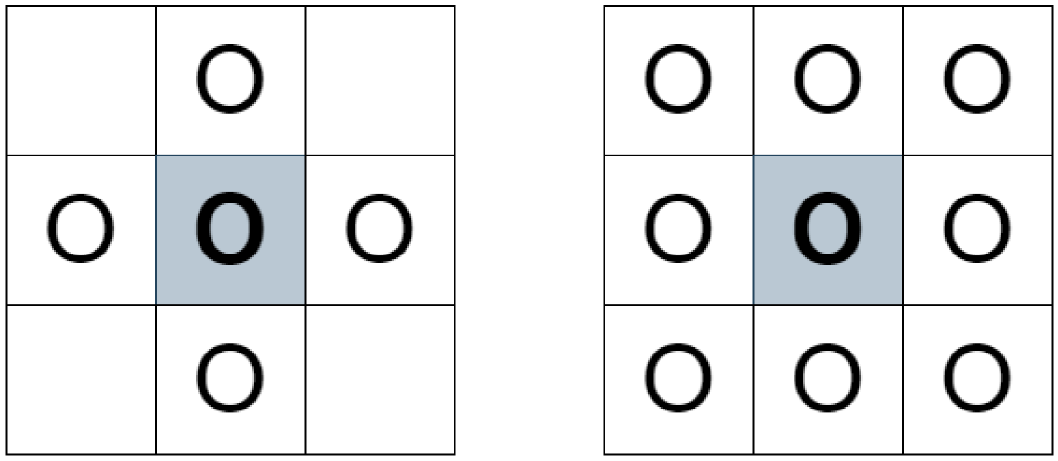


Figure 3.9: 4-pixel connected region to the left, 8-pixel connected region to the right

Based on the pixel neighourhood, different results are provided. Is is therefore vital to choose the pixel neighbourhood that provides the most appropriate results for a given demand[27].

3.6.2 Combining labeling and contour finding

Even region detection programs are prone to faults. The algorithm might run into some issues if the image region is of extreme enough complexity. It might correctly detect a region edge, but due to the region's complexity, it might also detect regions within the structure. This might even progress recursively so that the algorithm keeps detecting regions within subregions. This is where labelling becomes a powerful tool. With labelling, it is possible to assign markers that tell the algorithm as it moves through the image where it has been before, what it found, and where it should progress next. A combination of labelling and contour finding can be described as follows[31]: A region finding algorithm progresses throughout an image from the top left to the bottom right. As it progresses, it assigns one of three labels depending on three different scenarios. If there is an abrupt transition from a background pixel to an unmarked foreground pixel, it will be marked as an edge. The neighbouring background pixel will then be marked with a -1. If there is an abrupt transition from a marked foreground pixel to an unmarked background pixel, it will be marked as an inner contour. Once again, the neighbouring background pixel will be marked with a -1. Lastly, if there is a transition from a marked foreground pixel to an unmarked foreground pixel, it will be marked as a foreground pixel. The algorithm recognizes that the new unmarked pixel is a foreground pixel as there is no abrupt shift in pixel intensity values. The value of the marked foreground pixel will then be propagated forward. This process will continue until all the pixel elements have been marked. The resulting image will contain labels that indicate which pixels are structures and which pixels are background.

3.6.3 Shape features

A shape feature is a qualitative measure that can be extracted from a detected region. These measures form the basis of region features and are pivotal in the field of pattern recognition and feature extraction[27]. As the name suggests, shape features are the analytical extraction of the geometrical shapes of regions. Here is a list of shape features that are useful for feature extraction[43]

Area: Number of pixels in a detected region

Bounding box area: Number of pixels in the bounding box surrounding a detected region

Filled area: Area of a detected region with all subregions filled in

Convex area: The area of the smallest convex polygon enclosed by the detected region

Perimeter: Approximates the contour of a detected region as a line through the border pixels using 4-connectivity

Crofton perimeter: Calculates the perimeter approximation using the Crofton formula

Maximum Feret's diameter: The longest distance between points around a detected region's convex hull contour

Extent: The ratio between the detected pixels within a region and its corresponding bounding box

Statistical properties

Furthermore, it is also possible to gather statistical properties of the detected regions. Here is a list of beneficial statistical properties for feature extraction[43]

Moments: A statistically weighted average of the pixel intensities in the detected regions

Normalized moments: A moments calculation with consideration of the standard deviation

Weighted moments: Moments defined by the cumulative distribution function

Central moments: Translation invariant moments

Hu moments: Moments that are invariant to translation, scale and rotation

Centroids: Average coordinate of a region

Local centroids: Average coordinate of a region with relation to that regions bounding box

Weighted centroids: Average coordinate of a region in relation to that region's intensity image

Eccentricity: The ratio of the focal distance over the major axis length of a detected region

Euler number: The number of connected components subtracted by the number of holes in a detected region

Inertia tensor: An ellipse of a detected region that signifies the rotation around its mass

Inertia tensor eigenvalues: The eigenvalues of the inertia tensor

Minimum intensity: The minimum pixel intensity value of a detected region

Maximum intensity: The maximum pixel intensity value of a detected region

Average intensity: The average pixel intensity value of a detected region

Orientation: The angle between the rows and the major axis of the inertia tensor

Solidity: Ratio of the pixels in the detected regions to the region's convex hull image

3.7 Automatic thresholding

As explained in sec.3.4, the goal of thresholding is to separate the pixel elements into the foreground or background. Previously, this was done manually by setting a thresholding value. However, it is also possible to perform an unsupervised automatic thresholding operation. This is excellent for cases where multiple images must be processed to save time. Furthermore, it can also provide local thresholding values. Local thresholding values can, in most cases, extract even more helpful structural information from an image[27].

3.7.1 Otsu's method

Otsu's method is an automatic thresholding method that was created early in the field of image processing. It might be old, but it is pretty effective, even competing with newer and computationally heavier methods. Otsu's method finds the threshold value that maximizes the separation of the image histogram. The intra-variance of the resulting distributions will be at its minimum, while the centres of the distributions will be as distant from each other as possible[27].

3.7.2 Local adaptive threshold

In the case where the image has a low variability in the background and high variability of the foreground pixel intensities, a local adaptive threshold is a suitable thresholding method. This method calculates new thresholding values for every image position the operator moves through. The new threshold value is calculated through statistical properties of neighbouring pixels using a kernel, similar to filter kernels. This ensures that the operator correctly detects potential edges and contours, making the rest of the image processing process more accurate. Considering the thresholding step is performed so early in the process, choosing the local adaptive threshold can bring forth structures of an image that other thresholding techniques would otherwise lose. This is particularly true for images containing uneven exposures[18].

4. Machine learning

This section will provide a brief introduction to some of the central topics in machine learning. The topics will continue until they reach CatBoost, which is the algorithm used in this work to determine the outcome of the fault severities

4.1 Machine learning

A successful and effective machine learning algorithm typically consists of three key steps. The steps are data, feature and model, as shown in fig.4.1. It is imperative for an algorithm that sufficient amounts of training data are collected and, if necessary, labelled. Data is the information an algorithm will base its model and estimations on. Labelled data is provided with some tag, name or number to classify. An example of labelled data in machine learning can be found in object detection as boxes around objects with tags such as "car", "cyclist", and "pedestrian". Unlabelled data is data without any tags, names or numbers to classify them, leaving the classification to the algorithm. The concepts of labelled and unlabelled data will be further discussed later in the supervised and unsupervised learning sections. Having tremendous amounts of data is the most powerful way to boost the algorithm's performance. Furthermore, having data collected in the same domain as the machine learning tool will be deployed is a sure-fire way to ensure that the algorithm is trained and applicable to the user's needs. Therefore the quantity and quality of the data are essential[20].

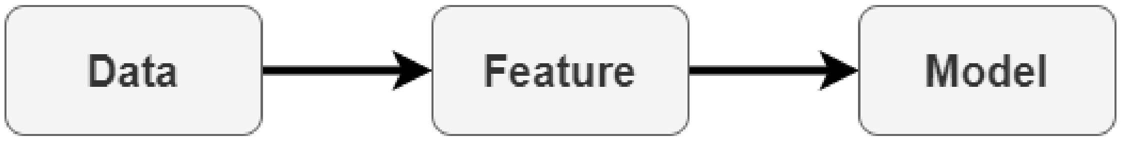


Figure 4.1: Typical machine learning pipeline

The second stage is the feature extraction stage. Here, domain-specific procedures must be used to extract relevant features from the raw data. What constitutes a feature is not always known, but it is up to the machine learning algorithm to sort it out. A feature can be considered a measurable characteristic of a trait in the data. The features should be compact while still containing critical data from the raw dataset. Creating a machine learning algorithm adapted to the dataset and the domain from which the dataset was gathered will ensure that the algorithm can identify the essential features. The issue is that domains vary widely in nature, and the algorithms will vary widely in design. The needs of a speech recognition problem require different methods than that of finding structures within images.

The final stage consists of choosing the proper learning algorithm to construct models from the extracted features of the training data. These models can range from classification, regression, supervised and unsupervised learning[20].

4.1.1 Classification versus regression models

Classification and regression models are two major categories in machine learning. They both have their uses, depending on the system outputs. Regression models can solve continuous outputs, while classification models can solve discrete outputs. This is because discrete outputs can only have a predefined output value, hence classification. Continuous outputs can have any real value within a given set of limits[20].

Classification models

Classification models predict a class-type outcome. Class type outcomes come in many different shapes and forms. An example could be vegetables, such as carrots and broccoli. Here carrots would be one class type, while broccoli is another distinct class type. A classification algorithm would look for attributes that define what broccoli is and what a carrot is. This could be its colour, its shape and its dimensions. Collecting data on various attributes could predict the likelihood of the outcome being a carrot or broccoli. A classification model can learn that broccoli is more likely green with stems spreading out of a stalk, while carrots are more likely orange and have a long cylindrical shape. It will, over time, learn that these attributes are more likely to belong to one category than another and will thus attach more data points belonging to that category, forming a classification[20].

Regression models

Regression is a method for predicting continuous outcomes from data, also known as predictive modelling. It learns and understands the relationship between features and an outcome. Once the algorithm estimates the relationship between dependent and independent variables, an outcome can be predicted. The input and output data are typically in the form of labelled training data. This training data must represent the domain in which the model will be applied. If done well, the model can predict outcomes from unseen input data. Otherwise, it will lead to inaccurate predictions. This is because the regressive model will attempt to learn from the nonrepresentative data and overfit logic that does not represent new data. Examples of usages for regression models are predicting house prices, the future success of retail sales, predicting customer trends, predicting stock prices, and predicting the success of marketing campaigns[20].

Supervised learning

Both classification and regression models fall under the category of supervised learning. This means that the training data has been labelled for a specific output. The models are subsequently trained until they can learn the underlying patterns that connect the input data and the premade output labels. The goal is for such an algorithm to yield accurate labels when used on new and unseen data. In other words, to make sense of new and unseen data within the domain in which the algorithm has been deployed.

A typical supervised learning algorithm will be deployed in the following way. It is introduced to a labelled training set. The algorithm uses this set to learn the relationship between the data set and the corresponding labels that are assigned to the data. Once the algorithm has formed some estimations, it is presented to a test dataset. This data set is also labelled, but the algorithm does not reveal the labels. This is because the testing dataset should act like new and previously unseen data. As a result, the test dataset allows the algorithm to measure how well it performs on unlabeled data.

How well an algorithm performs is dependent on what algorithm is used and the quality/quantity of the available training data. There needs to be enough data so that the algorithm can develop robust estimations, but it needs to be crafted well enough, so it does not skew the algorithm in favour of specific outputs[20].

Unsupervised learning

While supervised draws benefit from labelled data, unsupervised learning aims at bridging the gap between machine learning and human learning. Just as we do not know what something is before we have learned about it, the unsupervised method uses unlabeled data. This grants the unsupervised algorithm the freedom to act on the data without any outside influence. Similarily to supervised learning, unsupervised learning aims to find connections and structures within the dataset. It will attempt to gather the data according to the similarities and then attempt to provide apt representations of the data. This can be done with either clustering or association. The main reasons to use unsupervised learning versus supervised learning are to more closely mimic human learning and to have a model more adapted to the real world, as it is not always possible to have labelled data

4.1.2 Gradient boosting

Gradient boosting is a particular case in machine learning in which the deployment of many weak learners can iteratively create a robust model. The concept of boosting is that weak learners could be trained to become better. A good model could result from the emergence of many lesser models. The predictive function is built slowly and constantly checked to see if it is strong enough. This is where the gradient in gradient boosting comes in. The algorithm will gradually minimize the loss function by iteratively choosing the model that points towards the negative gradient. In practicality, it works like this: The algorithm starts with a series of weak learners who are barely better than random chance. These learners will attempt to create a predictive model based on their testing and hypotheses. For each iteration, the new hypothesis will be tested based on the classifications and misclassifications of the previous iteration. Over time, this will create a single robust model with a proper predictive function[20].

4.1.3 Decision trees

Decision trees are considered one of the more popular supervised learning algorithms. It is particularly used in classification problems, but it also works with regression problems. It is structured much like a tree with a stem that spreads out in branches, hence the name decision trees. Each branch junction, or node, represents a feature of the dataset. The branches that spring out of the node are the rules set by the decision tree algorithm. The end nodes, or leaves, represent the various outcomes.

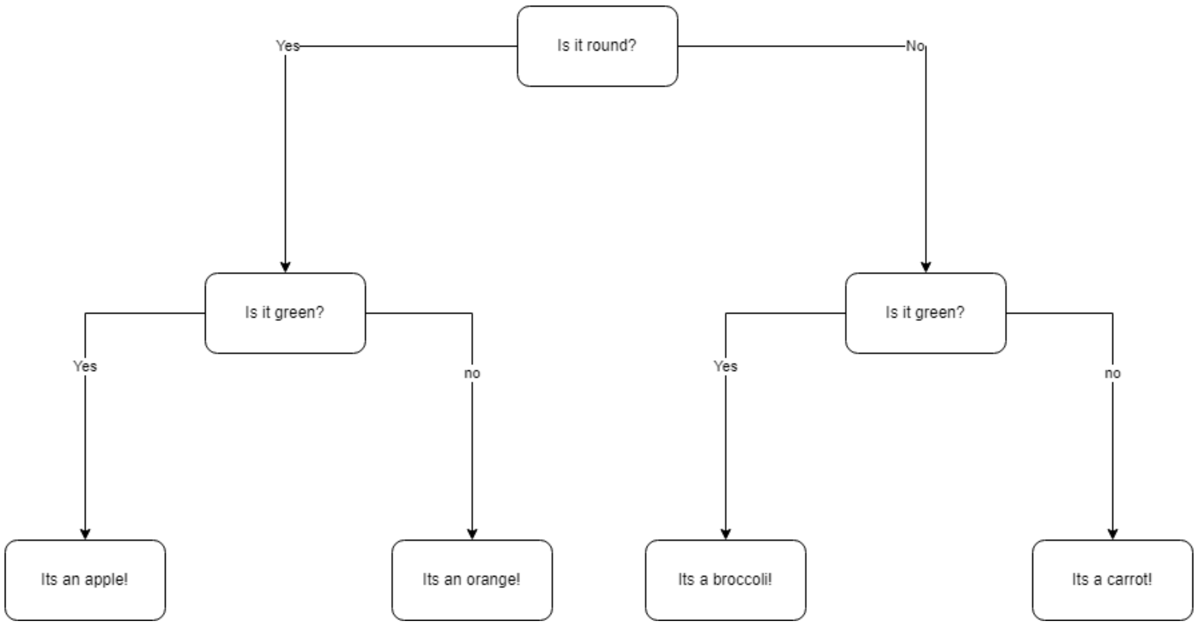


Figure 4.2: Illustration showing the branch like structure of the decision tree algorithm

The decisions are a direct result of the features in the dataset. These decision trees can keep expanding into a full tree-like structure depending on the complexity of the issue

4.1.4 Gradient boosting decision trees

As stated before, gradient boosting deploys many weak learners to create one strong learner. In the case of gradient boosting decision trees, the weak learners are individual decision trees. The decision trees are connected in a series where each tree aims at minimizing the previous tree's error. This is done by taking in the residuals of the previous step into the new decision trees. Once the final model has been achieved, the result of each step is aggregated, and a strong learner is created. It is a highly accurate algorithm, if not relatively slow due to its sequential nature.

4.1.5 CatBoost

The chosen algorithm for this work is the CatBoost algorithm. It uses gradient boosting on decision trees as its method. It is an open-source machine learning library which provides excellent results, especially for categorical data. It is relatively straightforward to implement as it only requires some dataset preprocessing. It has well-developed diagnostic tools, so it is possible to investigate which features affect the prediction function the most. It is also compatible with other libraries such as Keras and TensorFlow, which allows for a multitude of different approaches to solving a machine learning problem[5].

5. Literary review

This literary review will go along with the exploration and previous attempts at using machine learning to perform online fault detection of synchronous motors, methods for improving synchronous motors efficiency and fault detection methods of synchronous motors in general. Furthermore, an investigation into image processing techniques suitable for fault detection in synchronous motors was conducted. The literature will be presented chronologically to create a coherent red thread on how this field has gradually developed over time. Most, if not all, of the papers, are found and gathered from the Institute of Electrical and Electronics Engineers(IEEE). More specifically, it is the IEEE Xplore digital library.

5.1 Synchronous motors and efficiency optimization review

This literary review begins in 2005 with J. Habibi and S. Vaez-Zadeh's paper on optimizing the efficiency of a permanent magnet synchronous motor using direct torque control[14]. The study is motivated by increasing the PMSM's efficiency due to the widespread usage of these types of motors and the pressing matters of climate change. An offline optimization is first proposed using the optimum stator flux linkage values. These values are then stored in a table. Once this table is filled, you can use online calculations to acquire the desired rotation speed. Thus, a solution for optimum stator flux has been calculated for salient pole machines. This method would then be confirmed with simulation results. Another paper[46] proposed in 2008 a novel approach to control the direct torque system of a permanent magnet synchronous motor. This control system would directly control the magnetization factor of the stator current. Results indicated that the static and dynamic performances had improved, as well as a reduction in the torque ripple. The study[44] proposed in 2016 an efficiency optimization of the direct torque control with particular consideration of the iron losses. A mathematical approach is performed of the rotor frame with iron losses, which leads to a formula for power loss minimalization considering the relationship between output torque, speed, power losses and stator flux. This method preserved the fast direct torque control dynamic, as well as improved the operating efficiency of the driving system

The study by Yifa Sheng, Shou-Yi Yu and Wen-Zhen Zhou in 2010[34] aimed at combining torque compensation with intelligent integrated controls. This was achieved by a combination of fuzzy logic and the golden search method in order to accelerate the search process for the torque correction control. This was done in order to resolve the torque pulsation problem. In the paper, fuzzy logic and the golden section method are presented as methods of adaptively making the step size smaller to shorten the convergence time efficiently. They are both presented with advantages and disadvantages. The fuzzy method can adaptively make the step size smaller but runs the risk of increasing torque oscillations at the optimal efficiency point. The golden section method has a significantly faster convergence speed but may cause torque ripples at the beginning of the search. Therefore, combining these two methods might compensate for each other's weaknesses. The advancement of adding intelligent integrated control with the previously presented work proved to improve energy consumption, system efficiency, reduction in torque pulsations and overall robustness is improved.

An approach with outstanding results within online efficiency optimization in 2014 was the non-model-based optimization method[48]. It is an extremum-seeking control method that aims at optimizing the control values to make the system as efficient as possible. This setup is robust against any parameter changes and allows for a quick convergence speed as it is based on the principle of a numerical gradient-based optimization method.

Finally, the paper [47] aims to improve the control efficiency of PMSMs using machine learning methods. The chosen model was gradient boosting on decision trees. The study used the open-source machine learning library "CatBoost" for the experiments. The model was used to evaluate the stator yoke, tooth temperature, winding temperature and rotor temperature. It was deployed on an open base dataset library "Electric motor temperature", consisting of 998070 entries across 13 parameters. The study concludes that the CatBoost algorithm provided sufficiently accurate results that could be used in control systems in order to detect overheating early. This could potentially improve and increase the PMSM's operational life.

5.2 Image processing investigation

The book by Sklansky[36] was written in the 1980s to provide some structure and guidelines for the next generations of image processing developments. It stated that grayscale conversion was performed on analogue devices such as logarithmic amplifiers. Image segmentation techniques were developed, and feature extractions were performed. Filter operations were done through digital convolutions and were nearly as good as methods today in reducing image noise. Concepts such as "growing" and "shrinking" were early iterations of erosion and dilation. Histogram manipulation had progressed to the point where histogram equalization was considered new. For edge detection, gradient and Laplacian operators were the leading methods. The major obstacles that image processing faced in the 1980s were that previous knowledge of the images was necessary for many of the operations, as well as computation time and costs. The previous knowledge required was what structures and features were of interest and by what method the image had been acquired. As the next step, Hong et al.[15], and Liu et al.[24] argued that due to the nature of digital images being identical to matrixes, digital images could be subjected to algebraic transformations.

A study was done by Wu et al.[3] which proved that a couple of implementations could "the first can reduce the scan time by about half in many test cases, and the second can reduce the total execution time by a factor of five or more. If the image size is large, the speedup could be more than a factor of 100."[3]. The first implementation is to use arrays instead of pointer-based decision trees when performing the Union-find operations. The second implementation was to remove the flatten operation after lines had been scanned. This study was inspired by the works of Fiorio and Gustedt[10], which had proposed two linear time Union-find strategies for optimizing connected component labelling algorithms. Previously, Union-find algorithms had been performed using pointer-based decision trees[9][12]. The method of using decision trees came with the disadvantage that the amount of data needed to be handled would negatively impact computation time[3]. The strategy of storing the data in arrays instead of decision trees would significantly reduce the stored data, which would speed up the computation time

A study using the empirical cumulative distribution function to analyze the morphological data on nanoparticles was performed by Odziomek et al.[25]. The nanoparticles that were researched were tricalcium phosphate and hydroxy phosphate. They were investigated for features in the form of shape descriptors. Shape descriptors could range from aspect ratio, circularity, solidity, area, perimeter and roundness. The framework for this analysis was to choose the most representative image statistically and use the empirical cumulative distribution function based on the Kolmogorov-Smirnov test. This framework could significantly reduce the number of images required to confirm features sufficiently

A study that involved signal processing and image processing was the study on "Spectral feature extraction based on continuous wavelet transform and image segmentation for peak detection" by Yang et al.[45]. The paper followed the standard image processing steps, as well as using the continuous wavelet transform. This combination could reduce the effects of noise and eliminate interfering signals. A new development in terms of thresholding was the usage of the fuzzy Otsu method. A comparison between the Otsu and fuzzy Otsu threshold can be seen in fig.5.1. It is clear that the fuzzy Otsu method can detect a broader range of segments

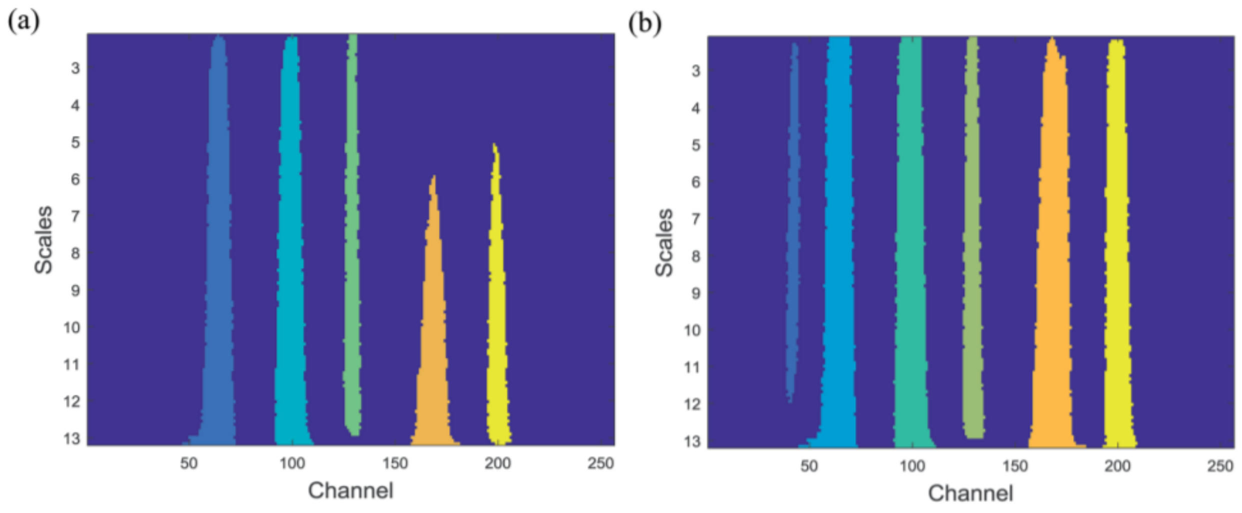


Figure 5.1: The segmentation performance (a) adopting the Otsu threshold, (b) adopting the fuzzy Otsu threshold[45]

An important point made in the study[45] is that the continuous wavelet scales have already been determined before any image processing can take place. It is advised to carefully choose the wavelet scale values to preserve as much helpful information as possible.

Another automatic thresholding technique that would challenge the fuzzy Otsu method is the adaptive local threshold method[31][18]. This method calculates multiple threshold values across the image using local pixel intensity structures. It is also more straightforward and fairly easy to implement compared to fuzzy Otsu. In the study by Issac et al.[18], adaptive local thresholding was used to investigate glaucoma. This method was decided because glaucoma imaging varies greatly in noise, grey levels and image structures. The results of that paper indicated that the adaptive local thresholding method could sufficiently improve glaucoma detection and classification

6. Method

The work in this thesis can be separated into four categories. Data acquisition with simulated and experimental data, signal processing through continuous wavelet transform, image processing through feature extraction and machine learning through gradient boosting on decision trees. The simulated data was acquired through ANSYS electronics desktop on a FE model created by a previous student[13]. The experimental data were acquired with a synchronous test generator at the department of electrical engineering at NTNU with the valuable help of research scientist Anyuan Chen. The signal processing was performed in python using an adapted version of code written by Tarjeu Nesbø Skreien[37]. The image processing is performed by edge sharpening, thresholding, region segmentation and feature extraction with powerful tools such as scikit-image[43], in a similar way as done on synchronous generators in the specialization project in fall 2021[21]. Machine learning was performed by extracting numerous properties of the image processing results and gathering them into datasets, which would be used in the robust gradient boosting on decisions trees open source library CatBoost[5].

The main contribution in this work compared to the specialization project[21] is an improved image processing method that has increased accuracy in spotting lower fault severities. A secondary contribution is a machine learning algorithm that can accurately predict fault severities, as well as an investigative analysis to see what features of the continuous wavelet plots contribute the most to detecting and classifying the faults.

6.1 Data acquisition

6.1.1 Simulation data acquisition

The simulation of the synchronous motor will be performed in the program ANSYS electronics desktop at the Ørstedt computer lab provided by the faculty of Information Technology and Electrical Engineering. A FE model for a synchronous generator has been previously created by the previous student Ingrid Linnea Groth[13]. In this work, this model has been modified to operate as a motor instead of a generator. The method for modification was provided by the ANSYS workshop: UDPS example. The modifications were relatively simple. Change the R, S and T phases from current based to voltage based. This was done by adding a low resistance and adding phase voltages in the three phases while also taking into account the phase shifts.

$$\frac{\sqrt{2}}{\sqrt{3}} * V * \sin(2\pi f t - \phi) \tag{6.1}$$

The simulation would run for $1100ms$ with a $0.1ms$ time step. The voltage $V = 400V$, and the frequency $f = 50Hz$. The information would be gathered by simulated magnetic flux sensors placed at the spots shown in the figure.1.3

The simulation was first run for a no-load healthy case. The simulation would run for 16 hours. In order to simulate an inter-turn shortcircuit fault. A modification to the model was performed. This was done by reducing the turns in the rotor winding N2 and rotor winding P1 in the FEM. Winding N2 and P1 correspond to the winding on a rotor pole's left and right sides. For every increase in severity, the winding was reduced accordingly, which means that for a two inter-turn short circuit, the winding was reduced from 35 to 33. For a three inter-turn short circuit, it was reduced from 35 to 32 and so on. This was performed for 1ITSC, 2ITSC, 3ITSC, 7ITSC and 10ITSC. The data would be collected and processed only to preserve the steady-state data.

In order to simulate a full-load case, some investigation into the FE model had to be conducted. First, it was essential to determine the direction the torque rotated in during generator operation. Second, to see if the motor operation inverted the torque. Once this was completed, the full-load simulation could be achieved by changing the phase angles of the three phases. By correctly manipulating these values, you could change the load angle, which in turn would simulate a loaded operation.

Further investigation had to be done. The testing would attempt to get a positive torque by either adding a voltage source or adjusting the phase angles. Positive torque, in this case, implies loaded motor operation. This would require abundant testing, so adjustments to the model were made to reduce the calculation time. Once an estimate for the max positive torque was achieved, the time-saving adjustments would be reversed, and the new parameters would be kept. These adjustments include reducing the mesh quality, increasing the time step and reducing the stop time. A voltage source would be added to the Excitations or in the External Circuit of the simulation. Alternatively, a current source could be added. Particularly for the current source excitation, an adjustment to the initial position along the q-axis had to be made. If the intended motor operation was not achieved by setting the initial position, an option was to check multiple different phase angles in the voltage or current source and observe the torque. With this, it is possible to identify the max positive, max negative and zero torque at specific phase angles.

Changing the initial position in order to simulate a full-load operation proved futile. The remaining alternative was to sweep through different phase angles to detect which phase angle values would achieve maximum positive torque. Due to time constraints, the sweep was not completed, and full-load data could not be acquired.

6.1.2 Experimental data acquisition

The experimental data for this report was collected with the synchronous test generator at the department of electrical engineering at NTNU. It is a 400V, 100kvA synchronous machine designed as a scaled-down version of a hydropower generator. This machine has a unique feature that allows for different cases of faulty operation. This is achieved by the machine's ease of use in terms of structure manipulation. It is designed, so it is possible to short circuit inter-turns, misalign the rotor to cause eccentricity, remove damper bars etc. This feature allows for the data acquisition of a multitude of faults and their severities. Perhaps the most unique feature is that it is possible to have data for faulty and healthy cases on the same machine. The most notable alteration that will be performed in this thesis compared to the specialization report[21], is the change from generator operation to motor operation.

Experimental setup

The setup for this experiment has been developed by Hossein Ehya and Arne Nysveen. The setup for data acquisition has been specifically developed by Hossein Ehya. The data acquisition is performed by using magnetic flux sensors on the stator core. This allows us to gather the stray magnetic field. The sensors are then connected to a RHODE SCHWARZ RTO 2044 oscilloscope which allows for easy gathering and exporting of data.

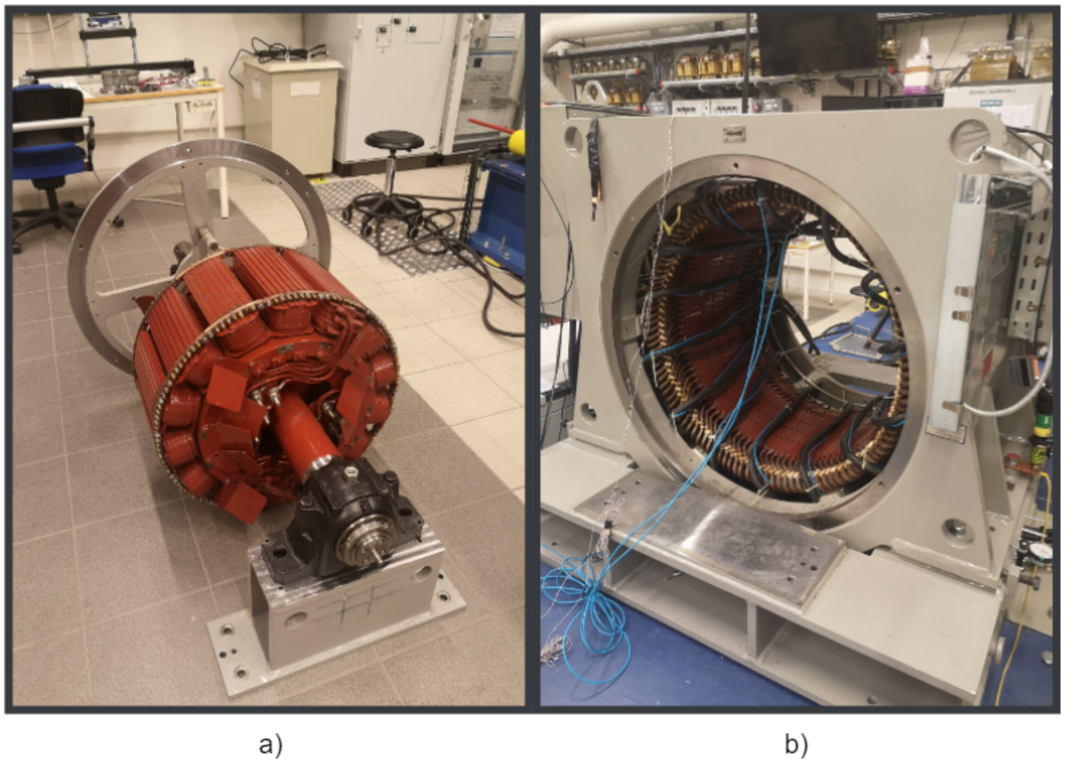


Figure 6.1: The laboratory rotor core a), and stator core b)

No-load ITSC proceedure

The proceedure for no load ITSC experiments are as follows: Remove the machine cage so we have access to rotor and the stator of the machine. Apply the desired fault by short circuiting the rotor using a metal plate. This plate would get secured tight to avoid any loose parts in the machine during operation. Reassemble the machine cage.

Turn on the rotor motor to initiate rotation. Slowly increase the rotor to the desired RPM, in our case it was until we reached 50Hz which was at around 1710RPM. Apply an excitation current and slowly raise it to the desired current, in our case it was 53.2A. Once the motor reaches steady state, we measure the voltage across the three sensors on a RHODE SCHWARZ RTO 2044 oscilloscope. Because we have three sensors, we have three channels of data to acquire. Once the oscilloscope has produced enough information, we stop it and save the data to a csv file. This file is then extracted using one of the USB slots of the oscilloscope. This process is repeated for different severities of ITSC fault ranging from healthy, 1ITSC, 2ITSC, 3ITSC, 7ITSC and 10ITSC. The differences in setup can be seen in the A..2

Full-load ITSC proceedure

The full-load procedure is an extension of the no-load proceedure. The startup of the motor by adjusting the RPM, the excitation current, the data collection and the adjustments made to induce a fault are all the same. The main difference is that the setup is connected to the main grid in order to apply a breaking torque. The phase angles and the frequencies of the grid and the machine needs to be synchronized. This is achieved by having a phase angle difference of zero and by the two systems having the same frequency. This was done by using a synchroscope and adjusting the RPM of the machine in order to align with the grid

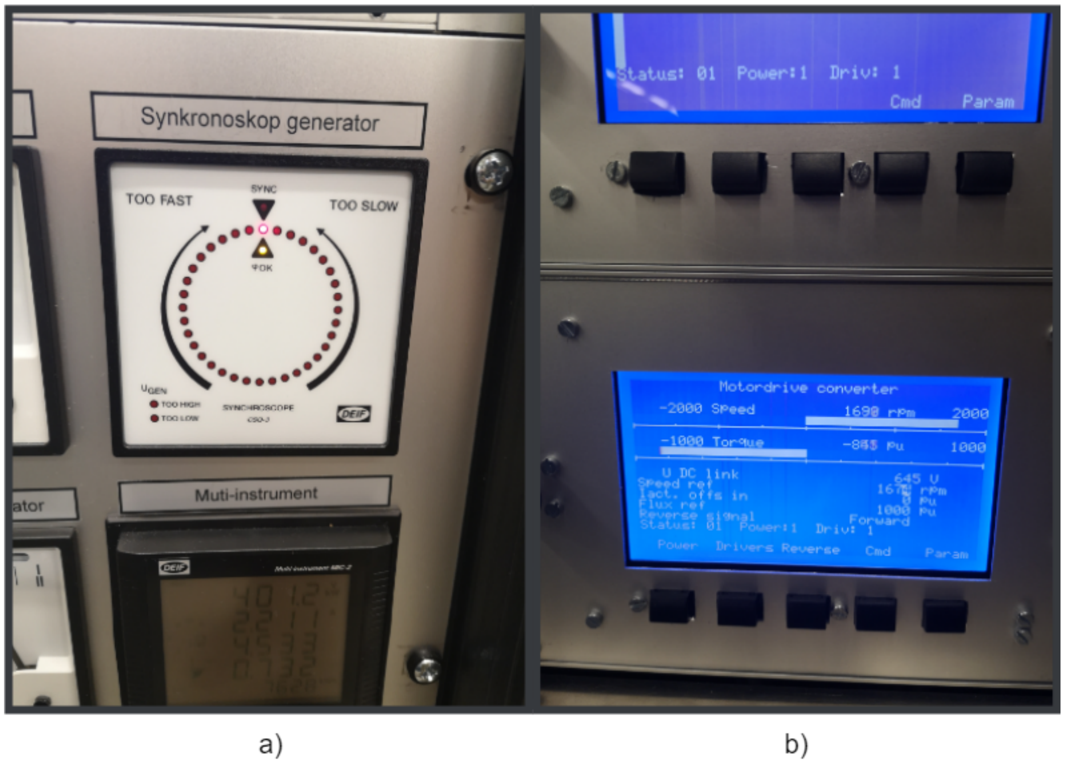


Figure 6.2: Synchroscope a), and speed reference control panel b). Note that b) already shows negative torque. This is due to the picture being taken after the synchronization had occured

The dial of the synchronoscope would rotate clockwise or counterclockwise depending on the difference in frequency and phase angle. We would adjust the motors RPM until it was possible to safely synchronize the two systems and lock them in. In order to apply a load to the motor, we reduced the speed reference to use the torque limit as a control for the actual breaking torque. As seen in fig.6.2, the torque is negative. Normally for synchronous machines, a positive torque indicates motor operation and a negative torque indicates generator operation. Because this system is primarily designed as a generator, negative torque here is equivalent to motor operation.

Possible sources of errors in the data

Some faults in the data can be explained by the following. Initial testing with the oscilloscope showed that the scope produced an error of 4V in one of its channels. The remaining three channels of the scope had an error of less than 200mV. Considering our results are in the range of 4V, this error may be responsible for up to 5% deviation in the data. The data export of the oscilloscope was done by manually pausing the data acquisition. An attempt to pause the oscilloscope during the same part of the phases for the different cases was made. This led to a difference in which initial part of the phase is captured. The data was gathered over multiple rotational cycles of the motor, so the long term effects of the difference is minimal. How this affects the method down the line is uncertain. After the signals have been processed, a CWT plot of the signal will be produced. In this plot, the fault will be shown in the image depending on where in the phase cycle the data acquisition begins. This means that the fault will appear in different places for different cases. The way the image processing program works, it will take the CWT plot and segments it into areas. Typically the program detects 28 areas, and their size and shape varies on where the fault is in the cycle. These areas are analyzed for various properties and then averaged. One could argue that because the areas get averaged in the end, the placement of the faults is negligible. Another argument is that the faults might appear more often, therefore there might be more calculated areas for the fault. This might result in a skew in the data, which might affect the machine learning later. A more indepth explanation of the image processing and machine learning applications will be provided in sec.6.4

6.2 Signal processing

The acquired data would then be processed through signal processing. Multiple methods for signal processing are available, such as fast fourier transform, short time fourier transform, discrete wavelet transform, hilbert-huang transfrom and continuous wavelet transform(CWT). For this thesis, the CWT was chosen based on its ability to produce plots which clearly shows a pattern with increasing fault severity. A set of intensity plots can be seen in fig.6.3.

The process of producing a CWT plot is built on calculing scales based on the frequencies of interest. The code takes in the continuous wavelet, the sampling frequency of the signal and the frequencies that are of interest and returns a list of scales. The plot generated by this process is dependent on a heatmap with varying levels of intensity throughout. Normally the intensity is automatically detected, but in order to have a comparative analysis between the healthy cases and the faulty cases, it was necessary to limit the heatmap values to an upper boundary. This boundary was decided by first running a healthy case with automatic intensity detection and noting the upper boundary. This upper boundary would then be set as the upper boundary for all the other cases. This approach circumvents the issue with scales as observed in [45].

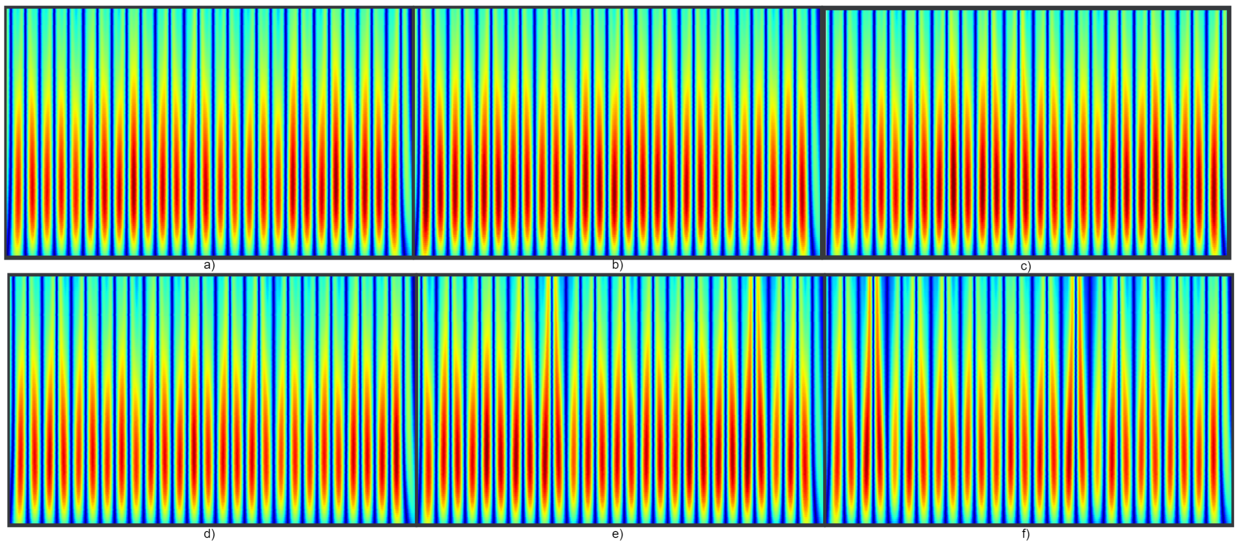


Figure 6.3: CWT intensity plots of stray magentic field, Full-load operation at a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

These plots show the CWT of different inter-turn shortcircuit(ITSC) fault severities. The upper left plot is the healthy case, with no induced faults. The faults range from lowest, 1ITSC, to highest(10ITSC). All the CWT plots can be found in A.3. The CWT was calculated over three mechanical rotations of the machine in both the experimental and simulated case. It is observed that with increasing fault severity, the intensity plot exhibits sudden spikes in intensity values periodically. These periodic spikes are assumed to be where the sensors register a stray magnetic field from where the fault is located. From section.2.1, we know that in CWT that sharp and sudden changes of image intensity in the plot are considered wavelet local maxima. This is because CWT produces large transformation values where the wavelet fits well with the scales at their corresponding frequencies. Whereas if it doesn't fit well, the CWT will produce low transformation values. This trait of CWT produces structures in the plot that we can take advantage of with image processing.

6.3 Image processing

This section of the method mirrors in large part the work I've done previously in the specialization report[21]. Some paragraphs have been rephrased and some explanations have been added to provide more clarity in the work presented. The figures have been updated to account for the change in data, as the thesis diverges from the specialization project in terms of machine operation.

Grayscale conversion

The first step of image processing is to convert the plots to grayscale using point operations, see sec.3.4. The plots are considered RGB images meaning they have three color channels of 8-bits as noted in section.3.2.2. In order to convert it, we need to combine the pixel intensity values in each channel into a single 8-bit channel. This is done by using eq.3.1. Images are converted to grayscale to reduce the amount of color channels which in turn reduces computation time. It is possible to perform image processing on RGB images, but all the operations needs to be performed on each channel separately, before being combined together again.

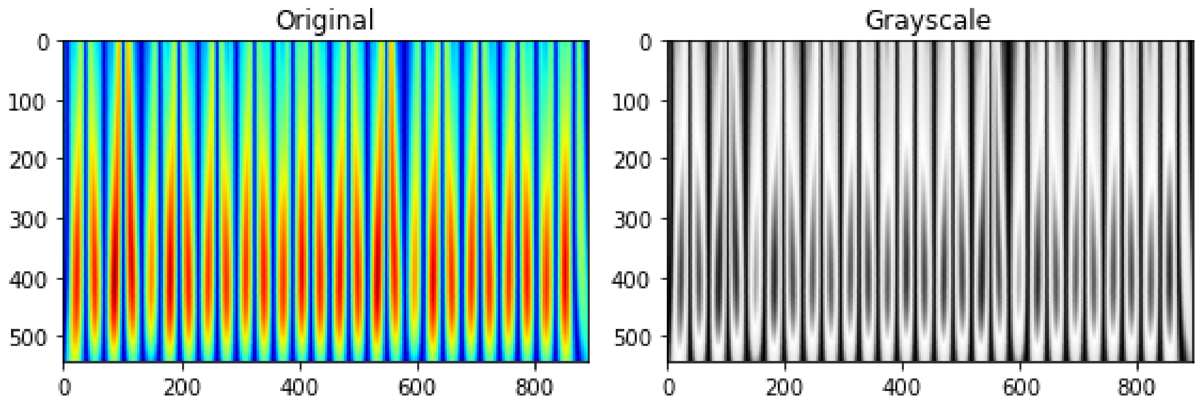


Figure 6.4: Original CWT plot of Full-load 10ITSC fault on the left, grayscale converted image on the right

Noise reduction

The grayscale images are then filtered using a median filter in order to remove any potential outliers in the image. These outliers, often referred to as hot pixels, are pixels with a sudden spike in brightness. If none of the surrounding pixels contain a similar spike, it is assumed that the hot pixel isn't part of any structure within the image and can be removed. Afterwards, a series of operations to sharpen the edges of the image is performed. First, a copy is made of the filtered grayscale image. The copy is then smoothed with the gaussian filter. The filtered grayscale image will then have its pixel intensity values subtracted with the smoothed copy's pixel intensity values. The resulting image will then have less noise and more even edges.

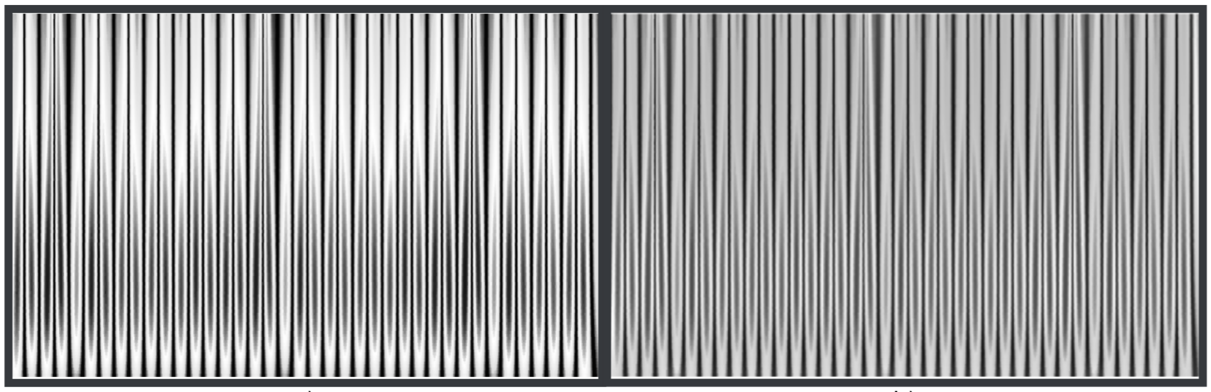


Figure 6.5: Grayscale image of Full-load 10ITSC fault with median filter a) and noise reduced with gaussian substraction b)

Thresholding

The next step is to perform a thresholding operation. This is done to seperate the foreground from the background, essentially isolating the objects we're interested in from the rest of the image. The tresholding method used in this work is the adaptive treshold. This was decided after doing a comparative analysis of three different thresholding techniques. The first approach was a manual thresholding method. The goal with the manual approach is to estimate which tresholding value will preserve the most useful information in the image. This was done iteratively until an optimal result was produced. The second approach was the otsu thresholding method. This approach automatically determines the thresholding value, so no iteration is necessary. However, this method produced subpar results which typically lacked significant portions of valuable image information. Lastly, the adaptive thresholding method is both automatic and it takes into consideration the neighbouring pixels to generate multiple threshold values. This is effective for images with great variation in pixel intensities in the foreground and low variability in the pixel intensities of the background, see section.3.7.2. The comparison can be seen in fig.6.6

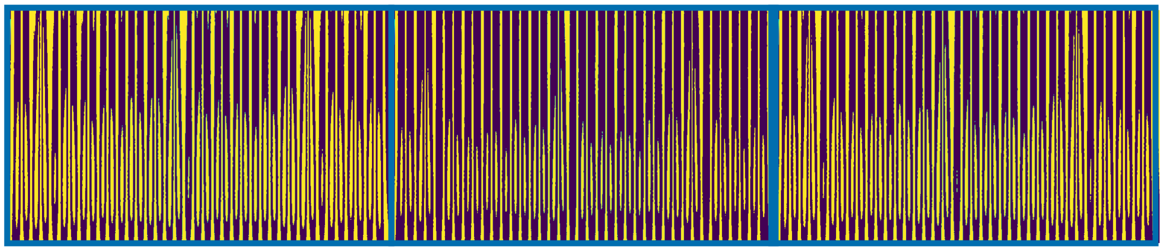


Figure 6.6: Comparison of threshold values on Full-load 10ITSC CWT plots, a) Manual given threshold constant of t = 0.05, b) Otsu threshold, c) Adaptive threshold

Image segmentation and labeling

With the images thresholded, they are now prepared to be segmented and labeled. A brief introduction to how this is done theoretically is given in section.3.6.2 and section.3.6.1. The method used in this work is based on the papers by Dillencourt et al.[9] and Gabow et al.[12]. Their methods have been adapted to this work in order to operate with array-based values as presented in[10][3]. The adaptation allows for a significant reduction in computation time and resources. The segmentation and labeling process is performed twice. The first operation labels all the areas it can find. After this, areas beneath a set value will be considered as noise and removed. This helps cleaning up the borders of the image. Afterwards, the remaining structures in the image are segmented and labeled once again. To further expand on the choice of thresholding method, a comparison can be seen in fig.6.7 to see how much the thresholding value affects the segmentation and labeling process. It is therefore recommended to carefully choose the thresholding value before continuing with the image processing.

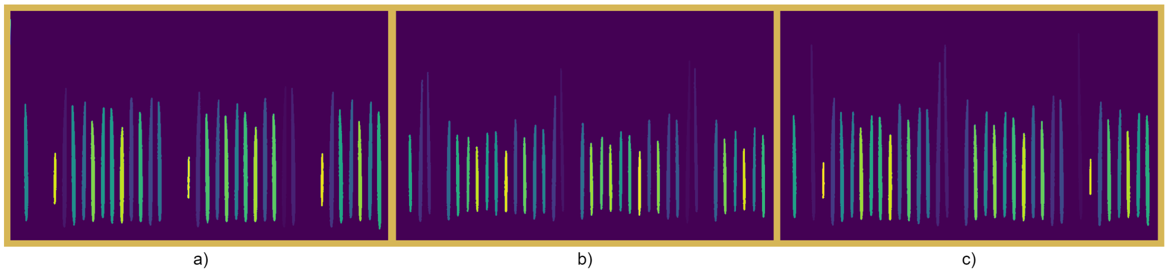


Figure 6.7: Comparison of threshold values on labeling process for Full-load 10ITSC, a) Manual given threshold constant of t = 0.05, b) Otsu threshold, c) Adaptive threshold

Detected regions test

A test is conducted to see if the algorithm can adequately detect the labeled regions. The results of that test can be seen in fig.6.8. Observations indicate that this method is sufficintly able to detect the areas and can surround the areas with bounding boxes. The boxes are the simplest figures to enclose a space and act as a reference point for later processing.

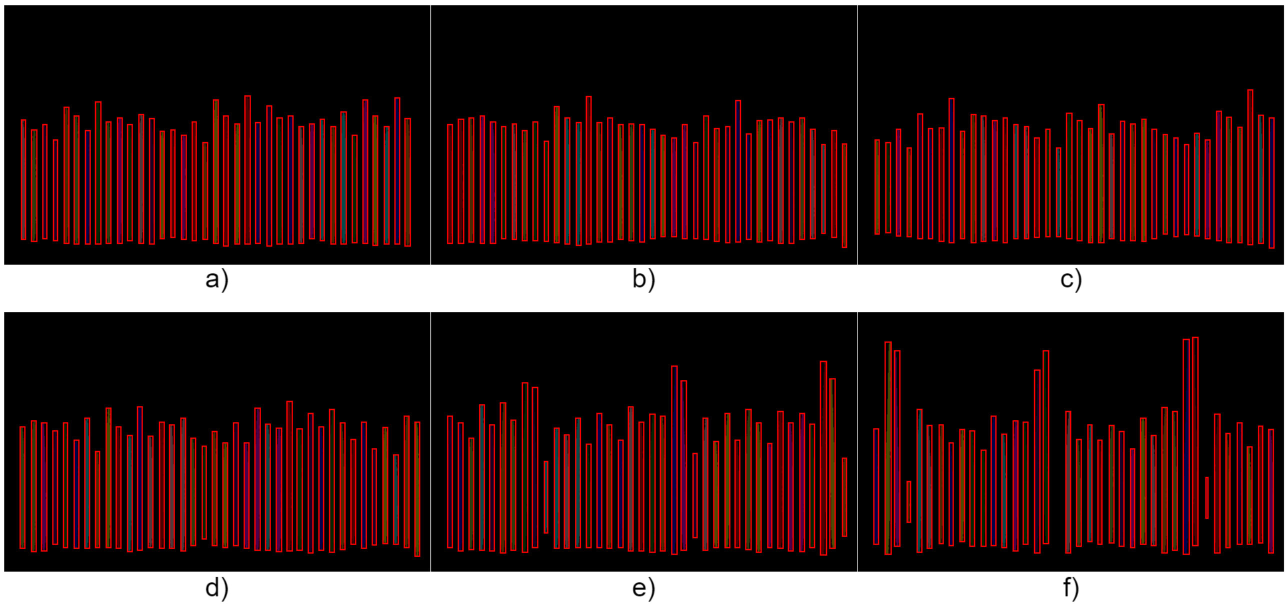


Figure 6.8: Detected regions in stray magnetic field during Full-load a) Healthy case b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

Empirical cumulative distribution function of the mean intensities

This conlcudes the preprocessing of the images. With the regions detected and labeled, it is time to extract useful information from them. The algorithm will take advantage of the fact that various fault severities have various levels of intensities in the CWT plots. It is therefore possible in theory to extract information that can differentiate the fault cases from one another. In order to take advantage of this trait, the algorithm takes the detected regions and layer them on top of the original, unprocessed intensity CWT plots. Afterwards, the algorithm will use the areas as reference when extracting information from the intensity plots. The intensities within the now referenced areas will be the main focus of the analysis. The algorithm then calculates the average intensity in each area and gathers this data in an array and saves it with the corresponding fault severity, sensor channel and load case. This information is then processed with the empirical cumulative distribution function(ECDF) in a similar fashion as done in [25]. In this report[25], the ECDF analysis requires a cumulative image set. This means that we normally take the average of the intensity values between fault severity plots in order to see how each plot differ from the average. For the sake of measuring fault severity, this work uses the ECDF for the healthy case as the "baseline". In other words, how much does the other cases differ from the healthy case in terms of mean intensity of the labeled regions. The analysis will focus on a healthy case, 1ITSC, 2ITSC, 3ITSC, 7ITSC and 10ITSC for no-load and full-load.

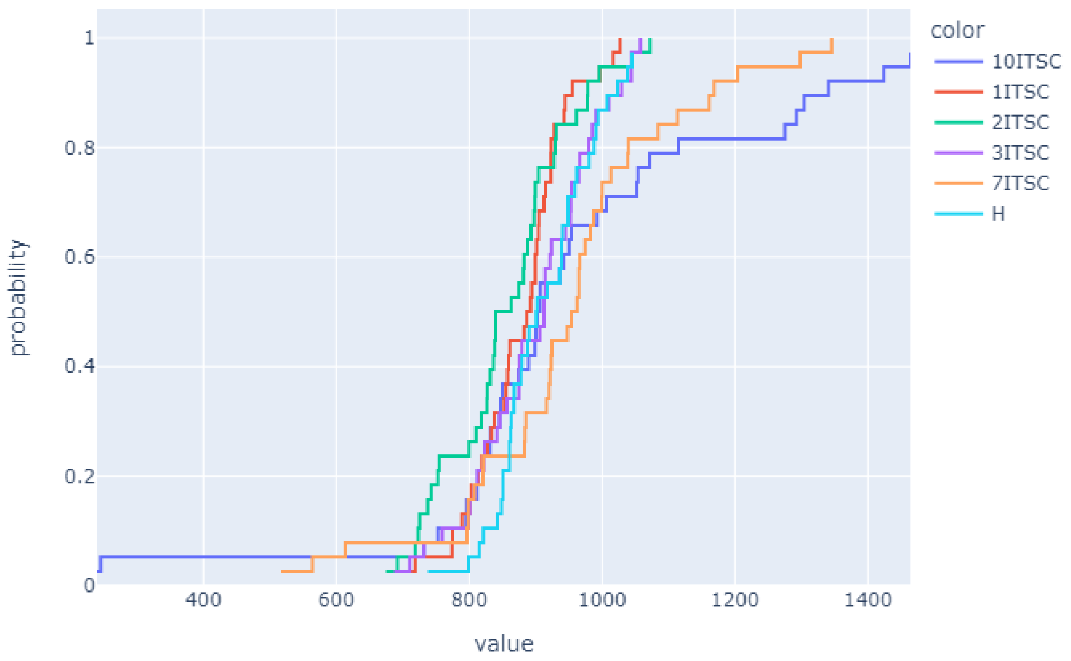


Figure 6.9: Mean intensity ECDF of image regions during experimental Full-load operation with different ITSC fault severities

The X-axis of the ECDF plot shows the observed average intensity values in the regions, while the Y-axis shows the expected probability for observing a value less than or equal to a given average intensity value[25]. Each of the coloured lines represent a fault severity.

The difference method

The previous method would use the labeled areas and place them on top of the original intensity CWT plots. This would mean that, for example, the labeled areas of the 10ITSC case would get layered on top of the 10ITSC CWT plot when calculating the mean intensity values.

The difference method instead takes the labeled areas of the different fault severities and layer them on top of the healthy CWT plot. Now all the severity cases are directly measured against the healthy CWT intensity plot.

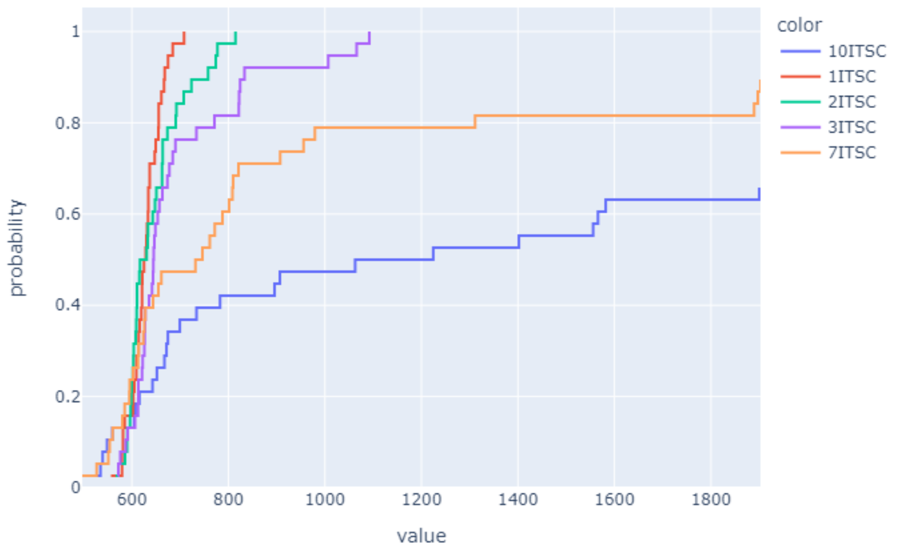


Figure 6.10: Mean intensity ECDF of image regions during simulated No-load operation with different ITSC fault severities

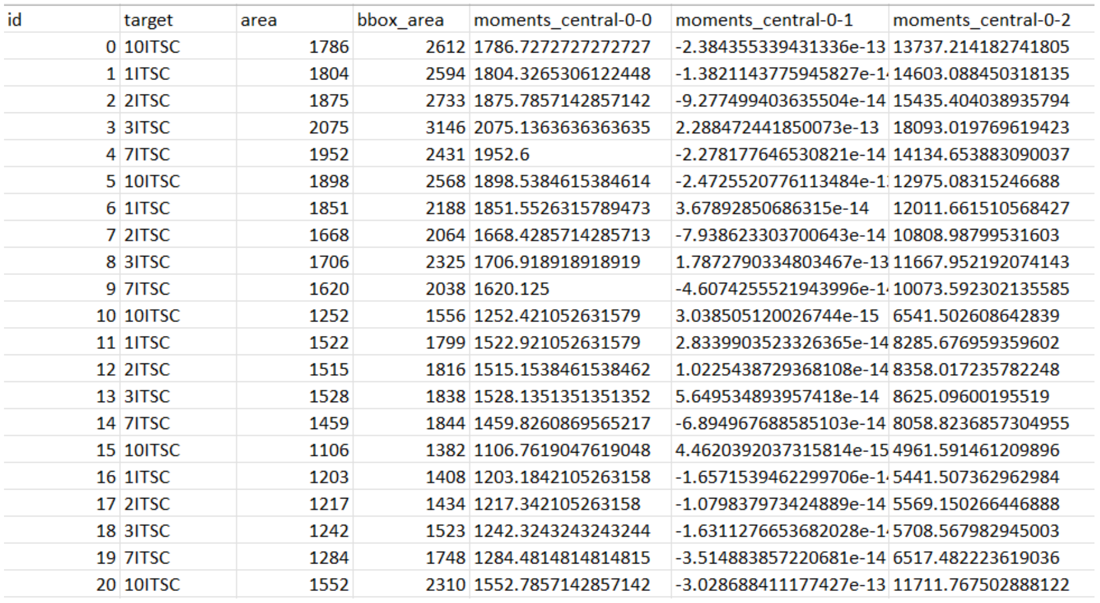
With this method, the healthy data is "lost" in the sense that we won't be calculating the properties of the segmented regions in the healthy CWT plot. Furthermore, by using the healthy CWT plot as the intensity plot for the faulty cases, it is more of a direct comparison. This means that the ECDF shows how similar each severity case is to the healthy case. If the severity case is low, and therefore differs only slightly, the ECDF will plot a straighter vertical line. If the severity case is high, and therefore differs significantly, the ECDF will plot a more vertical line.

6.4 Machine learning

For the machine learning part, the open-source library CatBoost was used. This is due to the study[47], which proved effective in diagnosing PMSMs. For the dataset, data from both experimental and simulation was mixed and used. This means the dataset contains data on healthy, 1ITSC, 2ITSC, 3ITSC, 7ITSC and 10ITSC faults for no-load and full-load operations. The dataset was also filled with data from each sensor. The experimental setup had three sensors for both full-load and no-load, while the simulated setup had 6 for no-load. The properties of the segmented regions would be calculated on the difference method. By using the optimal setup for generating plots, the dataset would contain $6x5 + 3x5 + 3x5 = 60$ entries. Early testing showed that this was not enough entries to achieve a good predictive function. Therefore, some manipulation of the plot generation was done. By doing slight tweaks to the maximum intensity values allowed and small tweaks to the maximum frequency of the plots, it was possible to generate a significant amount of new plots. To the human eye, these plots would seem to be slight variations of the same plots.

To the eye of the machine learning algorithm, these plots would contain new and previously unseen data.

With these tweaks, the dataset grew from 60 entries to 720 entries and thus forming the training dataset. The training dataset would be filled with information on the properties of the segmented regions in the CWT plots. These properties range from the area, bbox area, moments central, centroid, convex area, eccentricity, Euler number, extent, feret diameter max, filled area, moments hu, inertia tensor, inertia tensor eigvals, local centroid, max intensity, mean intensity, min intensity, moments, moments normalized, orientation, perimeter, Crofton perimeter, solidity, weighted centroid and weighted moments. The properties and their sub-properties would sum up to a total of 98 different categories. These are all properties that can be handled by the Scikit-image library[43]. Once all the values were filled in, the training dataset was split into training and testing datasets. The testing dataset will be unlabeled and act like new and unseen data to confirm the accuracy of the algorithm's predictive function.



id	target	area	bbox_area	moments_central-0-0	moments_central-0-1	moments_central-0-2
0	10ITSC	1786	2612	1786.7272727272727	-2.384355339431336e-13	13737.214182741805
1	1ITSC	1804	2594	1804.3265306122448	-1.3821143775945827e-1	14603.088450318135
2	2ITSC	1875	2733	1875.7857142857142	-9.277499403635504e-14	15435.404038935794
3	3ITSC	2075	3146	2075.1363636363635	2.288472441850073e-13	18093.019769619423
4	7ITSC	1952	2431	1952.6	-2.278177646530821e-14	14134.653883090037
5	10ITSC	1898	2568	1898.5384615384614	-2.4725520776113484e-1	12975.08315246688
6	1ITSC	1851	2188	1851.5526315789473	3.67892850686315e-14	12011.661510568427
7	2ITSC	1668	2064	1668.4285714285713	-7.938623303700643e-14	10808.98799531603
8	3ITSC	1706	2325	1706.918918918919	1.7872790334803467e-13	11667.952192074143
9	7ITSC	1620	2038	1620.125	-4.6074255521943996e-1	10073.592302135585
10	10ITSC	1252	1556	1252.421052631579	3.038505120026744e-15	6541.502608642839
11	1ITSC	1522	1799	1522.921052631579	2.8339903523326365e-14	8285.676959359602
12	2ITSC	1515	1816	1515.1538461538462	1.0225438729368108e-14	8358.017235782248
13	3ITSC	1528	1838	1528.1351351351352	5.649534893957418e-14	8625.09600195519
14	7ITSC	1459	1844	1459.8260869565217	-6.894967688585103e-14	8058.8236857304955
15	10ITSC	1106	1382	1106.7619047619048	4.4620392037315814e-15	4961.591461209896
16	1ITSC	1203	1408	1203.1842105263158	-1.6571539462299706e-1	5441.507362962984
17	2ITSC	1217	1434	1217.342105263158	-1.079837973424889e-14	5569.150266446888
18	3ITSC	1242	1523	1242.3243243243244	-1.6311276653682028e-1	5708.567982945003
19	7ITSC	1284	1748	1284.4814814814815	-3.514883857220681e-14	6517.482223619036
20	10ITSC	1552	2310	1552.7857142857142	-3.028688411177427e-13	11711.767502888122

Figure 6.11: A snippet showing how the dataset is built up, the full CSV file will be added to the submission

For CatBoost to work effectively, some preprocessing of the data is needed. First, the training dataset is sorted into numerical and categorical columns. This is done when there is a mix of different data types. By isolating the categorical columns, it is possible to convert them to a usable numerical format. The columns are then combined and a target is set. In this case, the targets are 1ITSC, 2ITSC, 3ITSC, 7ITSC and 10ITSC. The healthy data is indirectly present in the form that all the calculated properties for the faulty cases are in relation to the healthy case. The training dataset is then split into training and evaluation dataset. Evaluation data acts as a sort of temporary testing data in order to prevent overfitting of the model. Afterwards, a learning rate and a random state is chosen, and the CatBoost classifier can be deployed. In this case, CatBoost ran for 1000 iterations with a leaf depth of 8.

7. Results

Mean intensity method

The first image processing method of attempting to use the mean intensity values of the original continuous wavelet plots produced the following results with the empirical cumulative distribution function(ECDF) plot

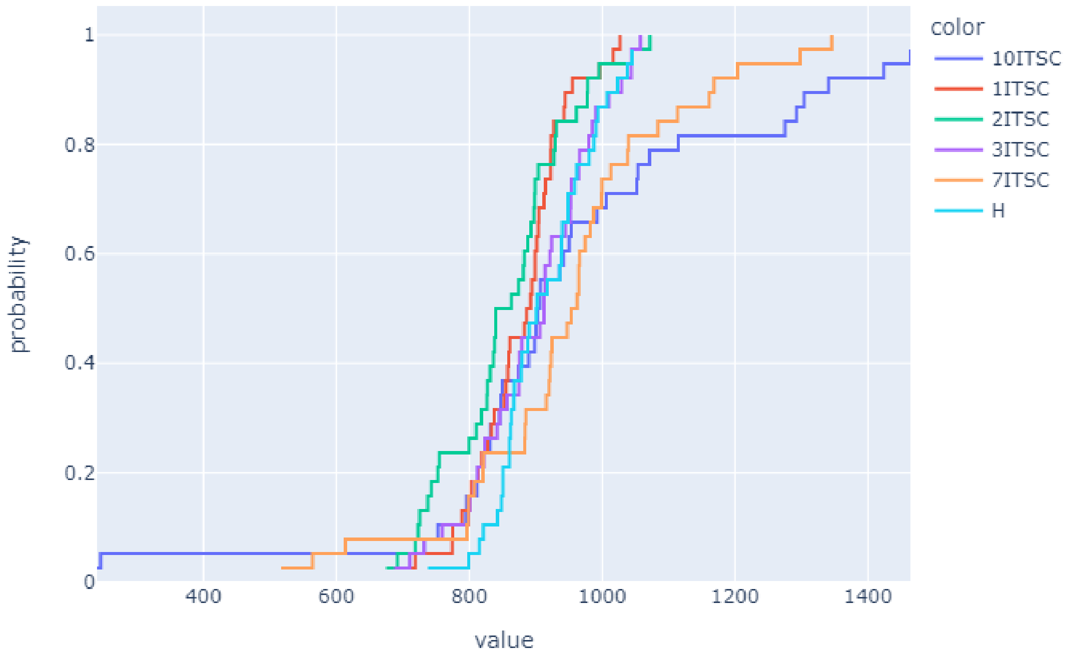


Figure 7.1: Experimental result of mean intensity ECDF plot for full-load ITSC faults sensor 1

The mean intensity results for all the other cases can be found in the A.4

Each line in the plot represents a fault severity case from healthy(H) to inter-turn shortcircuit(ITSC) fault of severity 1ITSC, 2ITSC, 3ITSC, 7ITSC and 10ITSC. The lines form a rudementary "S"-shape, which is common with the cumulative distribution function[25]. The X-axis of the ECDF plot shows the observed average intensity values in the regions, while the Y-axis shows the expected probability for observing a value less than or equal to a given average intensity value[25]. In the case of fig.27, it is observed that the 7ITSC and 10ITSC case starts to deviate from the trajectory of the healthy case. Meanwhile, the lower severity cases of 1ITSC, 2ITSC and 3ITSC appear to have similar trajectory as the healthy case. This makes it difficult to discern the fault at lower fault severities

Difference method

The second image processing method of attempting to use the healthy CWT plot case as the intensity image for the faulty cases produced the following ECDF plots

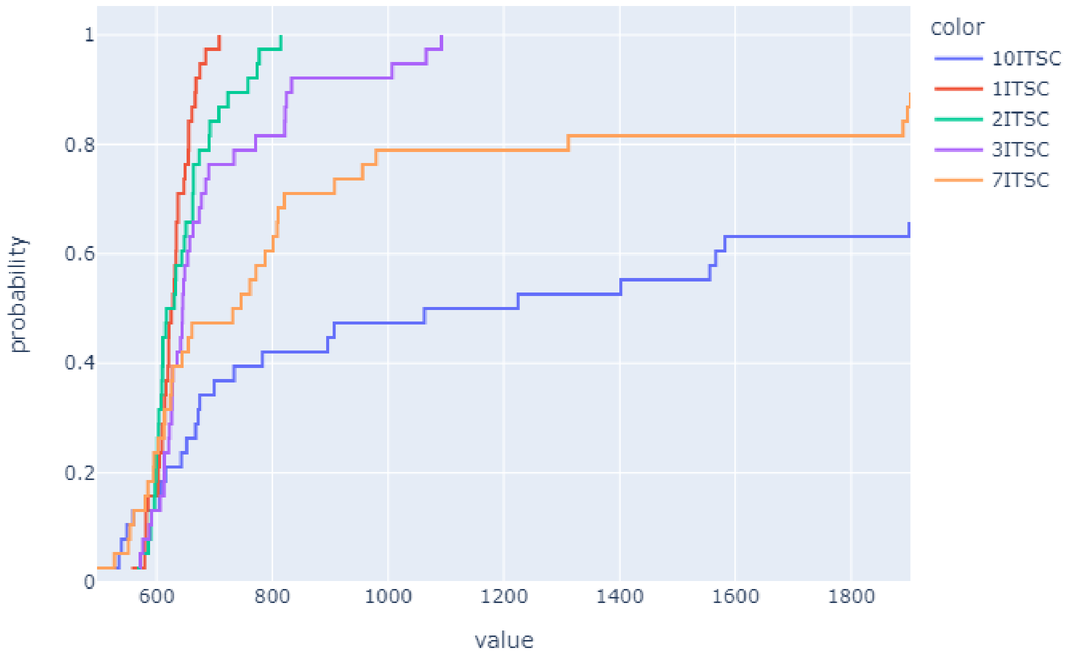


Figure 7.2: Simulated result ECDF difference plot for no-load ITSC faults sensor 5

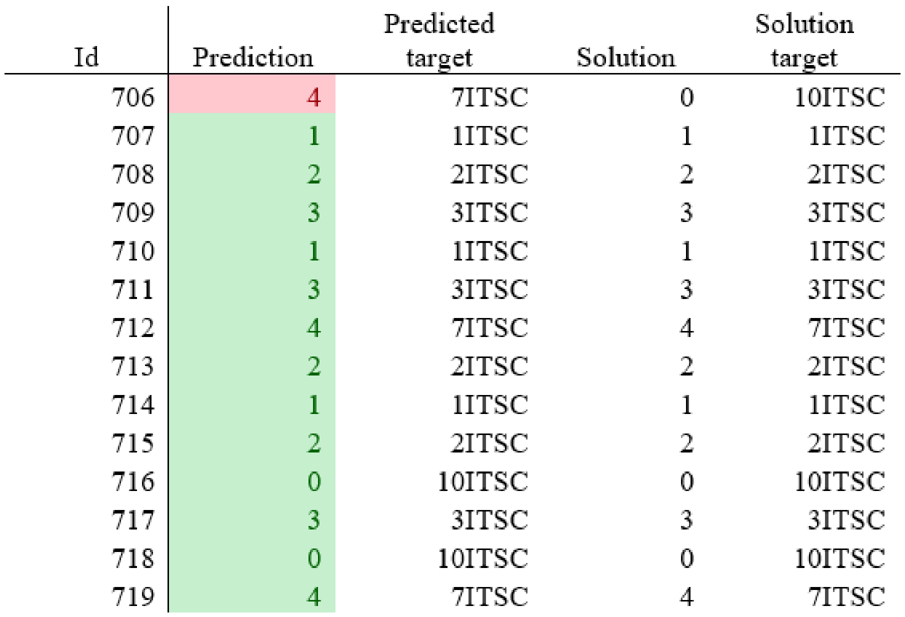
The difference method results for all the other cases can be found in the A.5

Similarily to the mean intensity method, each line in the plot represents a fault severity case. This time, the healthy case is not present in the plot. This is because the healthy data is being used as the comparison for all the other cases. While the mean intensity plot was a graph showing differences in intensity values, the difference method shows how similar each fault case is to the healthy case. The more a severity case differs, the more horisontal the plot lines become. In terms of detecting fault severities, the difference method is able to detect high severities as well as low severities. A clear difference between the cases is observed and a tendency for higher severity cases to go horizontal can be seen. It would appear that for the experimental full-load cases such as fig.39 and fig.41, the algorithm struggles between

1ITSC and 2ITSC. It consistently assumes that the 2ITSC case is more similar to the healthy case than the 1ITSC case

Machine learning results

The image processing method used for creating the region property data which would form the training dataset for the machine learning algorithm was the difference method. This was decided because the difference method produced more consistent and desired results than the mean intensity method. Using CatBoost on this dataset produced a R^2 value of 0.8. According to the book by Moore et al.[8], a $R^2 > 0.7$ is considered a strong effect size. In this case, it means that 80% of the variation in the output can be explained by the input variables. The results of the signal processing, image processing and machine learning application produced a model that could accurately predict 14 out of 15 severity cases



Id	Prediction	Predicted target	Solution	Solution target
706	4	7ITSC	0	10ITSC
707	1	1ITSC	1	1ITSC
708	2	2ITSC	2	2ITSC
709	3	3ITSC	3	3ITSC
710	1	1ITSC	1	1ITSC
711	3	3ITSC	3	3ITSC
712	4	7ITSC	4	7ITSC
713	2	2ITSC	2	2ITSC
714	1	1ITSC	1	1ITSC
715	2	2ITSC	2	2ITSC
716	0	10ITSC	0	10ITSC
717	3	3ITSC	3	3ITSC
718	0	10ITSC	0	10ITSC
719	4	7ITSC	4	7ITSC

Figure 7.3: Predicted results versus the solution template

Fig.7.3 shows the result on the test sample. The test sample is unlabeled, so the machine learning algorithm attempted to classify the data entries using its trained model. The predictions are in the shape of nummerical values. This is due to the targets being converted from categorical(strings) to nummerical(integer) values. The conversion of the target values is 1ITSC = 1, 2ITSC = 2, 3ITSC = 3, 7ITSC = 4 and 10ITSC = 0. Due to creating the testing dataset, the solution is known. Therefore it is possible to investigate how well the algorithm worked. It appears that the predictive function was able to accurately detect all the cases except one, where it predicted the entry to be a 7ITSC fault when in reality it was a 10ITSC fault.

An investigation into seeing which region properties affected the predictive function the most was conducted. The result of that investigation is shown here:

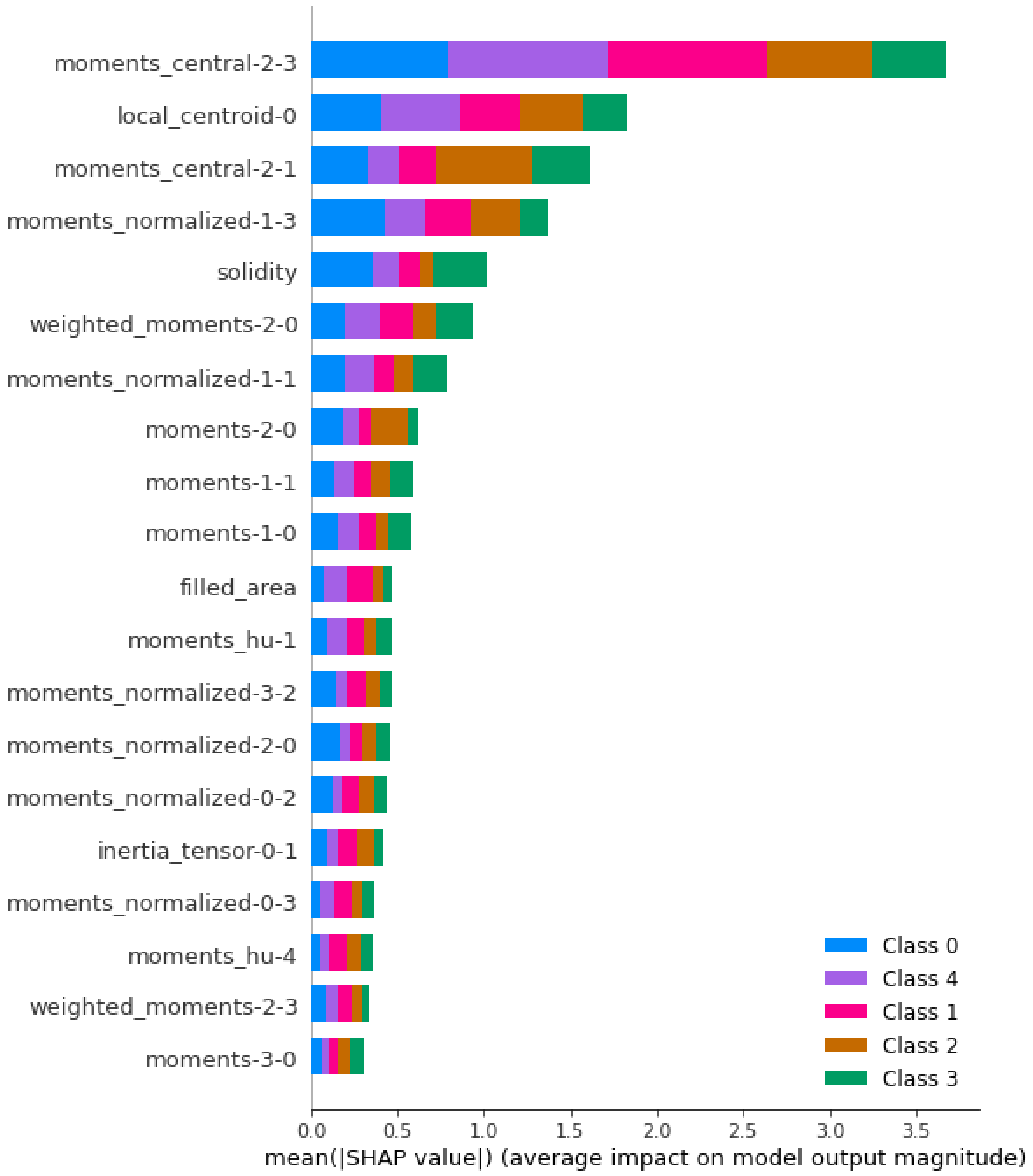


Figure 7.4: Comparison of region properties and their effect on the predictive function

Each bar in the figure shows the total of how much a region property impacted the classes. It is observed that the moments central-2-3 and local centroid 0 impacted the output magnitude the most

The model was trained for a second time with a larger testing dataset. This time the test dataset contained 63 entries compared to the 15 entries in the previous set. This also meant that the training dataset would be 48 entries shorter. The test dataset consisted of entries from both simulation and experimental operation. This time the model achieved a R^2 of 0.73 with an accuracy of 92%. Continued testing revealed that the accuracy was impacted by the training set size the most. By increasing the test dataset, the evaluation dataset had to be reduced accordingly. The investigation into region properties showed this plot:

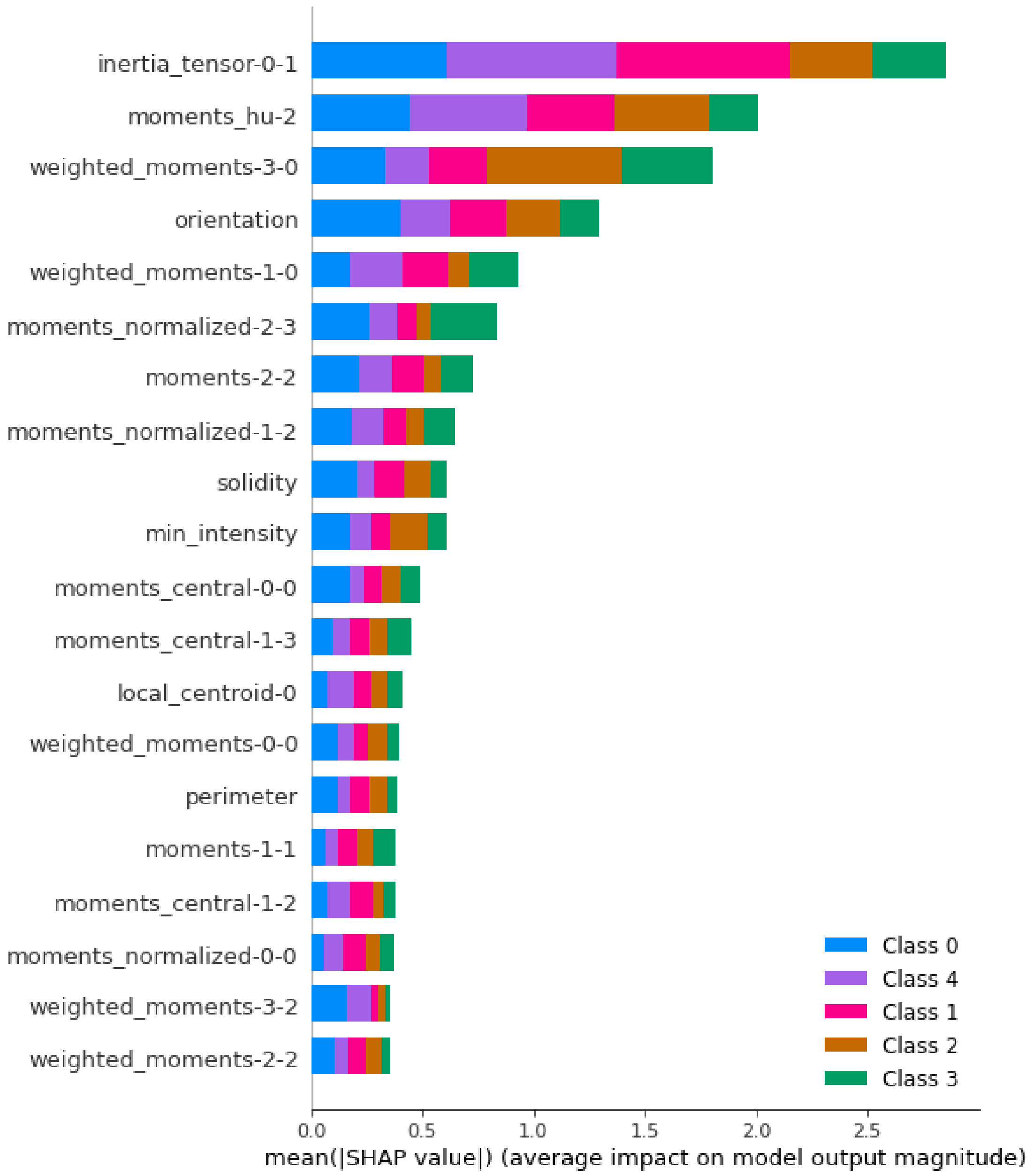


Figure 7.5: Comparison of region properties and their effect on the predictive function with a larger testing dataset

Which shows a drastic change compared to the previous region property investigation. This could suggest that the region properties themselves aren't as decisive as previously thought

8. Discussion

The results of this work indicate that using image processing and machine learning could be a viable method for fault diagnosis in synchronous motors. By using the difference method, it was possible to see an increasing difference between the healthy and faulty cases with increasing severity, even at lower fault severities. Using the mean intensity method could accurately detect higher intensity faults but not the lower fault severities. Using gradient boosting on decision trees as a method for machine learning could detect the fault and its severity in 92% of the cases, but is entirely dependent on healthy data.

The mean intensity method appears to have the worst results in accurately detecting faults. It was only able to detect higher severity faults accurately. Despite this, it is also the only method in this work that does not directly compare with the healthy data. All the cases are compared with themselves and then added to the empirical cumulative distribution function(ECDF). This means that healthy data is not necessary for performing computations. This could be an advantage because, in real-life applications, it is rare to find a complete healthy dataset for a synchronous motor. Then again, the ECDF plot plots the lines according to the levels of intensities found in the continuous wavelet transform(CWT) plot. Analyzing and determining that there is a fault based on the CWT intensities is difficult without knowing what a regular, healthy operation is supposed to look like.

The difference method showed that the ECDF plot could be used to measure fault severities more accurately. This included both low severity and high severity faults. This came at the cost of using the healthy data as a source of comparison. The fact that the healthy data is used for the difference comparison makes this method dependent on healthy data. Or at least less faulty data. With this analysis method, the ECDF plot went from being a measure of intensity to measuring how different the fault severities are from the healthy plot. A benefit to this method is that the most important properties for determining the difference between healthy and faulty cases are made more evident.

The machine learning algorithm produced surprisingly good results. Using the method proposed by the paper[47], a model with 92% accuracy on a bigger test dataset was produced in this thesis. This means that when presented with new and unseen information, the algorithm will be able to detect the fault severity 92% of the time. Of course, as long as it is within the same domain as the algorithm was deployed. Despite the high accuracy, this model will probably perform worse in real-life applications. The data used for training was data produced through the difference method. In other words, the algorithm is dependent on healthy data. What this method can be used for is to investigate what region properties are the most effective at predicting the various severities apart. This should have been reflected in fig.7.4 and fig.7.5 with the same region properties dominating the top of the impact scale for both cases. This does not mean the algorithm did not work as intended necessarily. This could mean that the algorithm saw some underlying logic early on in its iterations that worked and then proceeded to build a predictive model around that logic. It is fair to argue that the "moments" properties take up most of the spots on the plots. This is because out of the 98 properties, various moments constitute 70 of them. If we can assume that the algorithm can find underlying logic that works, regardless of the property, we can assume that the properties, in reality, have the same amount of potential for predicting the severities apart. Then, by the law of averages, it is fair to assume most of the top properties in the investigation will be moments.

In order to produce a better predictive model, more data is needed. This could have been easily solved in the data acquisition stage, but I lacked the foresight to see it at the time. When performing the experiments, the data would fill up the oscilloscope monitor and then get exported. After the data was exported, the experimental motor would be turned off and modified for the next faulty case. What should have been done was to take more samples per fault case. The data would be similar but different enough to aid a machine learning algorithm to learn better. Instead, data was "artificially" created to fill up the dataset. This was done by varying the max frequency and the max allowed intensity values of the CWT plots. This would be done on the same data, but the plots would be technically different with these changes. It would also be incredibly beneficial for the algorithm if it were fed data from different sources. Otherwise, the predictive function might become overfitted for detecting faults in the laboratory motor and the FE model. This would prevent the algorithm from being useful in settings other than the laboratory and the FEM.

Verifying the experimental full-load results with the simulation data is impossible in this thesis as there are no full-load data from the simulation. In theory, the no-load operation should appear the same for motor and generator operations. Full-load data would be needed for the FE model to be an accurate simulation of motor operation. Attempts at converting the FEM to a motor operation yielded little results.

Although the method proved effective in detecting inter-turn shortcircuit faults, it is necessary to expose the algorithm to other faults such as eccentricity and vibration faults to make it real-life applicable. It is rare for the real world to be so kind as to follow laboratory settings. It is therefore essential to gather data from different sources. Once a solid foundation of experimental and simulation data has been trained, the algorithm can use that previous experience to detect faults in real-life applications. It would be interesting to see how the algorithm would handle cases with mixed faults. Perhaps in the future, the algorithm will be able to detect cases with multiple faults accurately.

8.0.1 Future work

Based on the discussion, it is recommended to increase the dataset size for future work. It is also recommended to simulate full-load motor operation to verify the experimental data. Gather more samples of the same fault intensities to reinforce the underlying logics the algorithm finds. Experiments and simulations of various faults such as static eccentricity, dynamic eccentricity, mixed eccentricity, broken damper bar and vibration expose the algorithm to a broader range of faults. Data from real-life sources such as reciprocating pumps and compressors could be beneficial. Use this data on the method proposed in this thesis until a clear dominating region property is found. If this property is found, the original mean intensity method can be improved to the point where it can accurately detect lower severity faults. This can be done by swapping the "mean intensity" with the new region property. If this is done, the proposed machine learning method could become independent of healthy data and provide a better predictive accuracy.

9. Conclusion

In this thesis, a machine learning application, gradient boosting on decision trees, is explored to detect inter-turn shortcircuit faults in synchronous motors. The machine learning library, CatBoost, would use image processing data for automatic classification of motor fault. The image processing data would be acquired through feature extraction of region properties. The region properties would be both spatial and statistical descriptors. The image regions would be detected through a series of image processing operations which can be summarised to grayscale conversion, image sharpening, thresholding, segmentation and labeling. The images are produced as a result of performing continuous wavelet transformations on magnetic field data, acquired both experimentally and through simulations. The resulting investigations can be categorised as two types. One is dependent on healthy data, while another is independent of healthy data. Due to the small dataset sample available, the healthy data dependent method proved the most accurate with a predictive accuracy of 92%. An investigation into which descriptors had the greatest weight on the predictive function was performed. The investigation could aid in the pursuit of which descriptors is beneficial for fault detection. It is argued that with enough data, the method can become independent of healthy data and become more applicable as a diagnostic tool.

9. Bibliography

[1] Michael Stokes (Hewlett-Packard) Matthew Anderson (Microsoft) Srinivasan Chandrasekar (Microsoft) Ricardo Motta (Hewlett-Packard). *A Standard Default Color Space for the Internet - sRGB*. URL: https://www.w3.org/Graphics/Color/sRGB. (accessed: 17.06.2022).

[2] "2. The Continuous Wavelet Transform". In: *Ten Lectures on Wavelets*, pp. 17–52. DOI: 10.1137/1.9781611970104.ch2. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611970104.ch2. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611970104.ch2.

[3] Wu K. Otoo E. Shoshani A. *Optimizing connected component labeling algorithms*. URL: https://escholarship.org/uc/item/7jg5d1zn. (accessed: 03.01.2021).

[4] ANSYS. *Ansys Maxwell Low Frequency EM Field Simulation*. URL: https://www.ansys.com/products/electronics/ansys-maxwell. (accessed: 25.05.2022).

[5] CatBoost. *Open source CatBoost GitHub page*. URL: https://github.com/catboost/catboost.git. (accessed: 16.06.2022).

[6] Pragnan Chakravorty. "What Is a Signal? [Lecture Notes]". In: *IEEE Signal Processing Magazine* 35.5 (2018), pp. 175–177. DOI: 10.1109/MSP.2018.2832195.

[7] Eduardo A.B. da Silva and Gelson V. Mendonça. "4 - Digital Image Processing". In: *The Electrical Engineering Handbook*. Ed. by WAI-KAI CHEN. Burlington: Academic Press, 2005, pp. 891–910. ISBN: 978-0-12-170960-0. DOI: https://doi.org/10.1016/B978-012170960-0/50064-5. URL: https://www.sciencedirect.com/science/article/pii/B9780121709600500645.

[8] William I. Notz David S. Moore. *The Basic Practice of Statistics*. 9th ed. Macmillan Learning, 2021. ISBN: 9781319383695; 1319383696. URL: libgen.li/file.php?md5=dc4be0a736a0fdb1709cd6af9f59eea9.

[9] Michael B. Dillencourt, Hanan Samet, and Markku Tamminen. "A General Approach to Connected-Component Labeling for Arbitrary Image Representations". In: *J. ACM* 39.2 (1992), pp. 253–280. ISSN: 0004-5411. DOI: 10.1145/128749.128750. URL: https://doi.org/10.1145/128749.128750.

[10] Christophe Fiorio and Jens Gustedt. "Two Linear Time Union-Find Strategies for Image Processing". In: *Theor. Comput. Sci.* 154 (1996), pp. 165–181.

[11] Market Research Future. *Synchronous Motor Market*. URL: https://www.marketresearchfuture.com/reports/synchronous-motor-market-8318. (accessed: 28.05.2022).

[12] Harold N. Gabow and Robert Endre Tarjan. “A Linear-Time Algorithm for a Special Case of Disjoint Set Union”. In: *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*. STOC ’83. New York, NY, USA: Association for Computing Machinery, 1983, pp. 246–251. ISBN: 0897910990. DOI: 10.1145/800061.808753. URL: https://doi.org/10.1145/800061.808753.

[13] I. L. Groth. *On-line magnetic flux monitoring and incipient fault detection in hydropower generators*. 2018.

[14] J. Habibi and S. Vaez-Zadeh. “Efficiency-Optimizing Direct Torque Control of Permanent Magnet Synchronous Machines”. In: *2005 IEEE 36th Power Electronics Specialists Conference*. 2005, pp. 759–764. DOI: 10.1109/PESC.2005.1581712.

[15] Zi-Quan Hong. “Algebraic feature extraction of image for recognition”. In: *Pattern Recognition* 24.3 (1991), pp. 211–219. ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(91)90063-B. URL: https://www.sciencedirect.com/science/article/pii/003132039190063B.

[16] ImageJ. *What is a digital Image?* URL: https://sites.google.com/site/learnimagej/image-processing/what-is-a-digital-image. (accessed: 16.12.2021).

[17] Rakib Islam et al. “Inter winding short circuit faults in permanent magnet synchronous motors used for high performance applications”. In: *2012 IEEE Energy Conversion Congress and Exposition (ECCE)*. 2012, pp. 1291–1298. DOI: 10.1109/ECCE.2012.6342667.

[18] Ashish Issac, M. Partha Sarathi, and Malay Kishore Dutta. “An adaptive threshold based image processing technique for improved glaucoma detection and classification”. In: *Computer Methods and Programs in Biomedicine* 122.2 (2015), pp. 229–244. ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2015.08.002. URL: https://www.sciencedirect.com/science/article/pii/S0169260715001996.

[19] R. H. Herrera J. B. Tary and M. van der Baan. “Analysis of time-varying signals using continuous wavelet and synchrosqueezed transforms”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*. Vol. 376. 2126. 2018. DOI: 10.1098/rsta.2017.0254.

[20] Hui Jiang. *Machine Learning Fundamentals: A Concise Introduction*. New. Cambridge University Press, 2022. ISBN: 9781108837040; 1108837042. URL: libgen.li/file.php?md5=64070d5f00d8518d82edc7399315cc58.

[21] M. F. Johansen. *Image processing application for fault diagnosis in hydropower generators*. 2021.

[22] Wilhelm Kirchgässner, Oliver Wallscheid, and Joachim Böcker. “Data-Driven Permanent Magnet Temperature Estimation in Synchronous Motors With Supervised Machine Learning: A Benchmark”. In: *IEEE Transactions on Energy Conversion* 36.3 (2021), pp. 2059–2067. DOI: 10.1109/TEC.2021.3052546.

[23] Lewis Lipkin. “Pictorial Information: <i>Picture Processing by Computer</i>. Azriel Rosenfeld. Academic Press, New York, 1969. x + 198 pp., illus. $11.50. Computer Science and Applied Mathematics.” In: *Science* 169.3941 (1970), pp. 166–167. DOI: 10.1126/science.169.3941.166. eprint: https:

//www.science.org/doi/pdf/10.1126/science.169.3941.166. URL: https://www.science.org/doi/abs/10.1126/science.169.3941.166.

[24] Ke Liu, Yong-Qing Cheng, and Jing-Yu Yang. “Algebraic feature extraction for image recognition based on an optimal discriminant criterion”. In: *Pattern Recognition* 26.6 (1993), pp. 903–911. ISSN: 0031-3203. DOI: https://doi.org/10.1016/0031-3203(93)90056-3. URL: https://www.sciencedirect.com/science/article/pii/0031320393900563.

[25] ODZIOMEK K. USHIZIMA D. OBERBEK P. KURZYDŁOWSKI K. J. PUZYN T. HARANCZYK M. “Scanning electron microscopy image representativeness: morphological data on nanoparticles. Journal of Microscopy”. In: *Journal of Microscopy* 265(1) (2016), pp. 34–50.

[26] Stephane Mallat. *A wavelet tour of signal processing. The sparse way.* 3ed. Academic Press, 2009. ISBN: 0123743702; 9780123743701. URL: libgen.li/file.php?md5=6869ff67f0daf4cbe8cf5861e47add6f.

[27] Alberto Aguado Mark Nixon. *Feature Extraction and Image Processing for Computer Vision.* 4th Edition. Academic Press, 2019. ISBN: 9780128149775.

[28] Dubravko Miljković. “Brief Review of Motor Current Signature Analysis”. In: *CrSNDT Journal* 5 (June 2015), pp. 14–26.

[29] Mark S. Nixon and Alberto S. Aguado. “3 - Image processing”. In: *Feature Extraction and Image Processing for Computer Vision (Fourth Edition).* Ed. by Mark S. Nixon and Alberto S. Aguado. Fourth Edition. Academic Press, 2020, pp. 83–139. ISBN: 978-0-12-814976-8. DOI: https://doi.org/10.1016/B978-0-12-814976-8.00003-8. URL: https://www.sciencedirect.com/science/article/pii/B9780128149768000038.

[30] Yonghyun Park et al. “Air Gap Flux-Based Detection and Classification of Damper Bar and Field Winding Faults in Salient Pole Synchronous Motors”. In: *IEEE Transactions on Industry Applications* 56.4 (2020), pp. 3506–3515. DOI: 10.1109/TIA.2020.2983902.

[31] Richard E Woods Rafael C Gonzalez. *Digital image processing.* 4th Edition. Pearson, 2018. ISBN: 9780133356724 0133356728.

[32] Rushank Savant, Abhiram Ajith Kumar, and Aditya Ghatak. “Prediction and Analysis of Permanent Magnet Synchronous Motor parameters using Machine Learning Algorithms”. In: *2020 Third International Conference on Advances in Electronics, Computers and Communications (ICAECC).* 2020, pp. 1–5. DOI: 10.1109/ICAECC50550.2020.9339479.

[33] Muhammad Faizan Shaikh, Jongsan Park, and Sang Bin Lee. “A Non-Intrusive Leakage Flux Based Method for Detecting Rotor Faults in the Starting Transient of Salient Pole Synchronous Motors”. In: *IEEE Transactions on Energy Conversion* 36.2 (2021), pp. 1262–1270. DOI: 10.1109/TEC.2020.3021207.

[34] Yifa Sheng, Shou-yi Yu, and Wen-zhen Zhou. “Research on efficiency optimization of interior permanent magnet synchronous motor based on intelligent integrated control”. In: *2010 Chinese Control and Decision Conference.* 2010, pp. 3059–3063. DOI: 10.1109/CCDC.2010.5498664.

[35] G. A. Skarmoutsos, K. N. Gyftakis, and M. Mueller. “MCSA versus Flux Monitoring for Demagnetization Diagnosis in Axial-Flux PM Generators”. In: *2021 IEEE 13th International Symposium on Diagnostics for Electrical Machines,*

Power Electronics and Drives (SDEMPED). Vol. 1. 2021, pp. 119–125. DOI: `10.1109/SDEMPED51010.2021.9605530`.

[36] J. Sklansky. *Image Segmentation and Feature Extraction*. Pattern recognition research: Technical report. School of Engineering, University of California, 1977. URL: `https://books.google.no/books?id=8SXsGwAACAAJ`.

[37] T. N. Skreien. *Application of signal processing and machine learning tools in fault detection of synchronous generators*. 2019.

[38] O. Sørheim. *Fault detection of inter-turn short-circuit in synchronous generator using stray magnetic flux*. 2020.

[39] Stanford. *Introduction to digital images*. URL: `https://web.stanford.edu/class/cs101/image-1-introduction.html`. (accessed: 16.12.2021).

[40] Greg C. Stone et al. "Using magnetic flux monitoring to detect synchronous machine rotor winding shorts". In: *2011 Record of Conference Papers Industry Applications Society 58th Annual IEEE Petroleum and Chemical Industry Conference (PCIC)*. 2011, pp. 1–7. DOI: `10.1109/PCICon.2011.6085862`.

[41] Hamid A. Toliyat et al. *Electric Machines: Modeling, Condition Monitoring, and Fault Diagnosis*. CRC Press, 2013. ISBN: 0849370272; 9780849370274. URL: `libgen.li/file.php?md5=3054d5917b00228dd70ef53662b4fb8b`.

[42] Julio Urresty et al. "Detection of Demagnetization Faults in Surface-Mounted Permanent Magnet Synchronous Motors by Means of the Zero-Sequence Voltage Component". In: *Energy Conversion, IEEE Transactions on* 27 (Mar. 2012), pp. 42–51. DOI: `10.1109/tec.2011.2176127`.

[43] Stéfan van der Walt et al. "scikit-image: image processing in Python". In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: `10.7717/peerj.453`. URL: `https://doi.org/10.7717/peerj.453`.

[44] Zhang Xinghua and Chen Pengfei. "Efficiency optimization of direct torque controlled interior permanent magnet synchronous motor considering iron losses". In: *2016 19th International Conference on Electrical Machines and Systems (ICEMS)*. 2016, pp. 1–5.

[45] Guofeng Yang et al. "Spectral feature extraction based on continuous wavelet transform and image segmentation for peak detection". In: *Anal. Methods* 12 (2 2020), pp. 169–178. DOI: `10.1039/C9AY02052G`. URL: `http://dx.doi.org/10.1039/C9AY02052G`.

[46] Sheng Yifa, Yu Shouyi, and Hong Zhennan. "A novel control method for permanent magnetism synchronous motor direct torque control system". In: *2008 27th Chinese Control Conference*. 2008, pp. 672–675. DOI: `10.1109/CHICC.2008.4605147`.

[47] E.V. Zagoskina et al. "Improving of Permanent Magnet Synchronous Machine Control Efficiency". In: *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*. 2020, pp. 0411–0413. DOI: `10.1109/USBEREIT48449.2020.9117688`.

[48] Tong Zhao, Xiangming Shen, and Ronghui Zhou. "An online efficiency optimization control method for permanent magnet synchronous motor". In: *2014 IEEE Conference and Expo Transportation Electrification Asia-Pacific (ITEC Asia-Pacific)*. 2014, pp. 1–4. DOI: `10.1109/ITEC-AP.2014.6940773`.

Appendices

.1 Image processing theory figures

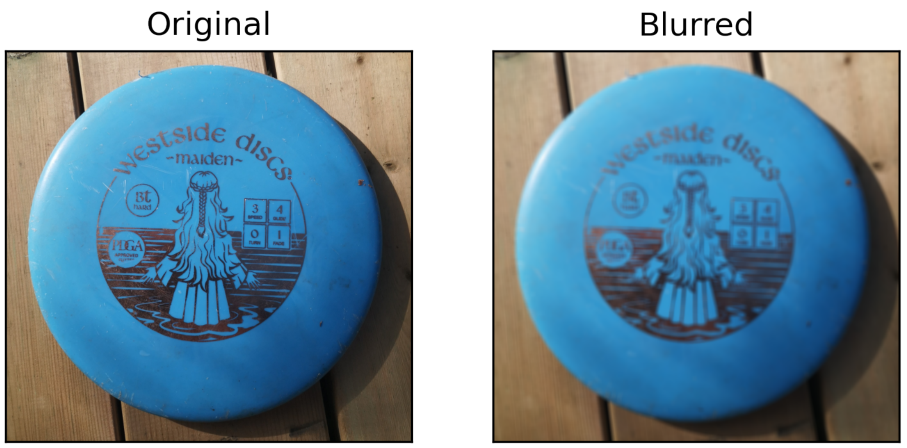


Figure 1: Base image applied with the averaging blurring filter

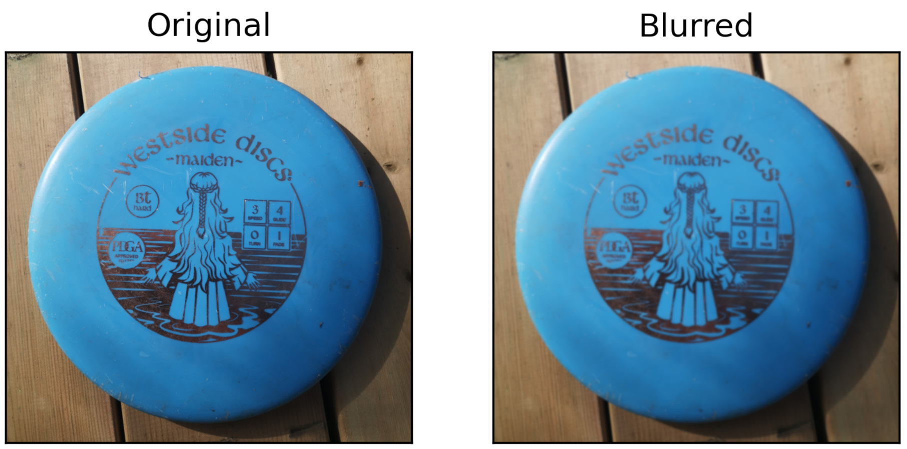


Figure 2: Base image applied with the gaussian blurring filter

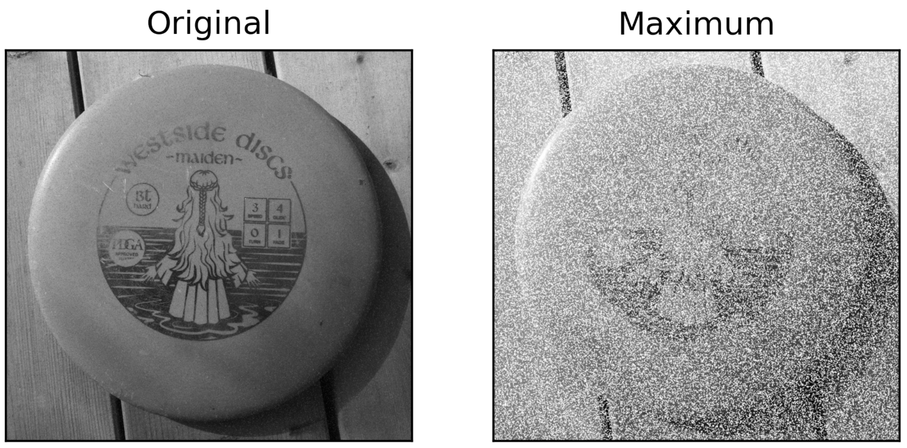


Figure 3: Grayscale image with the max filter

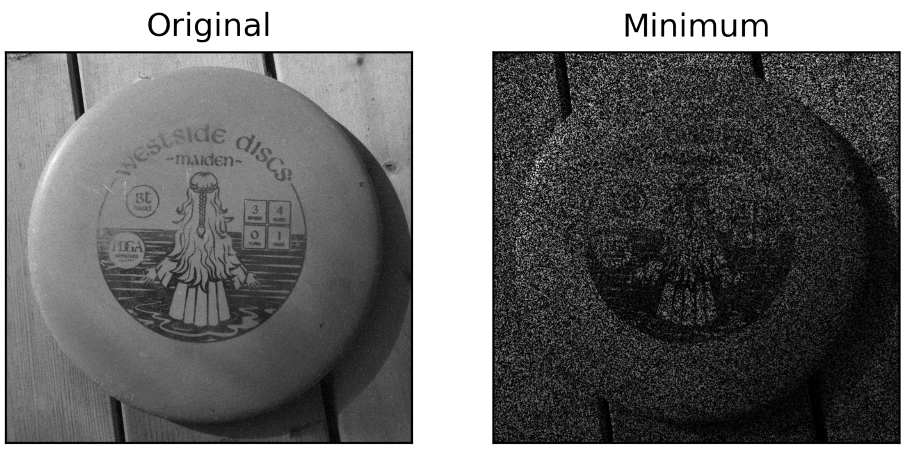


Figure 4: Grayscale image with the min filter

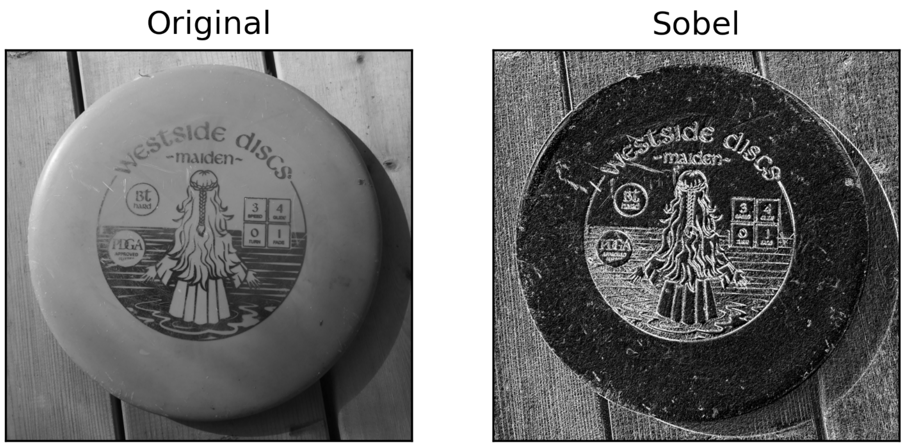


Figure 5: Sobel edge operator on grayscale image

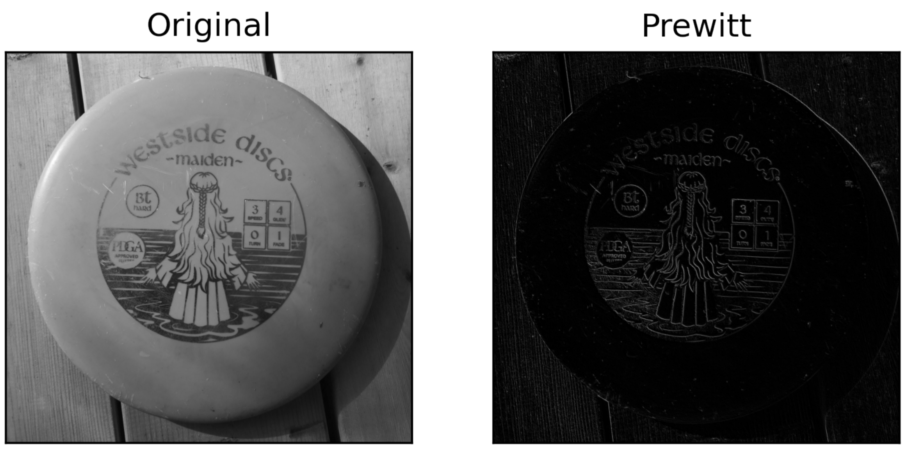


Figure 6: Prewitt edge operator on grayscale image

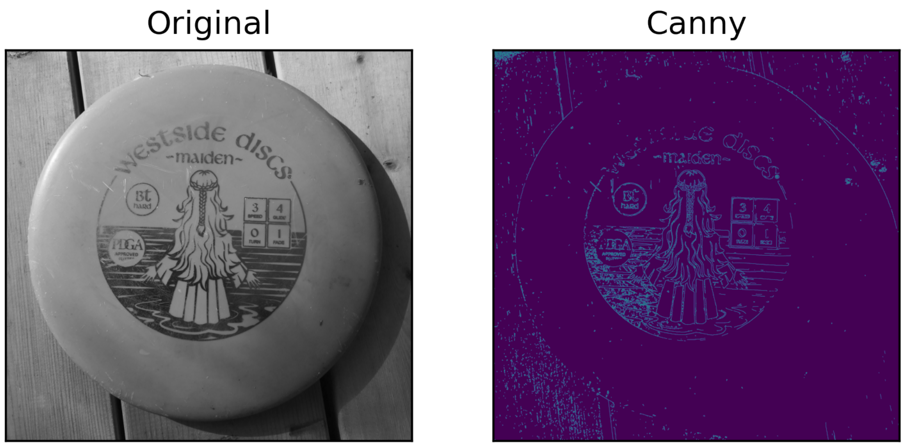


Figure 7: Canny edge operator on grayscale image

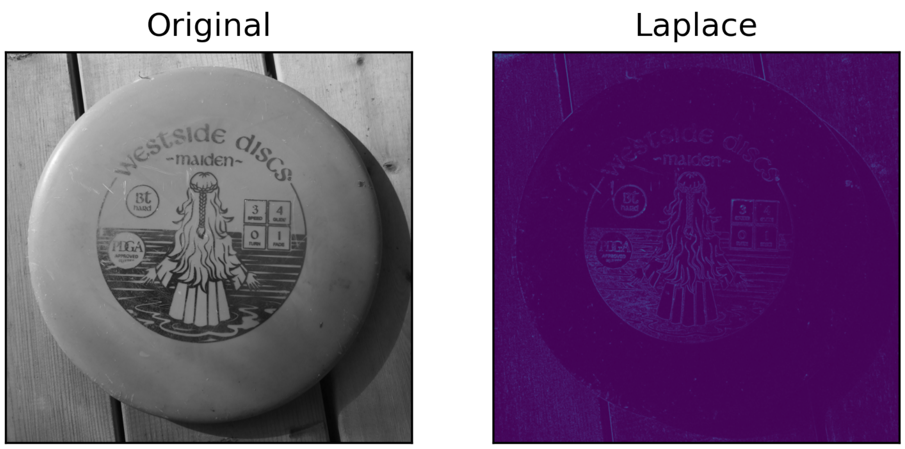


Figure 8: laplace edge operator on grayscale image

.2 Experimental interturn shortcircuit modification

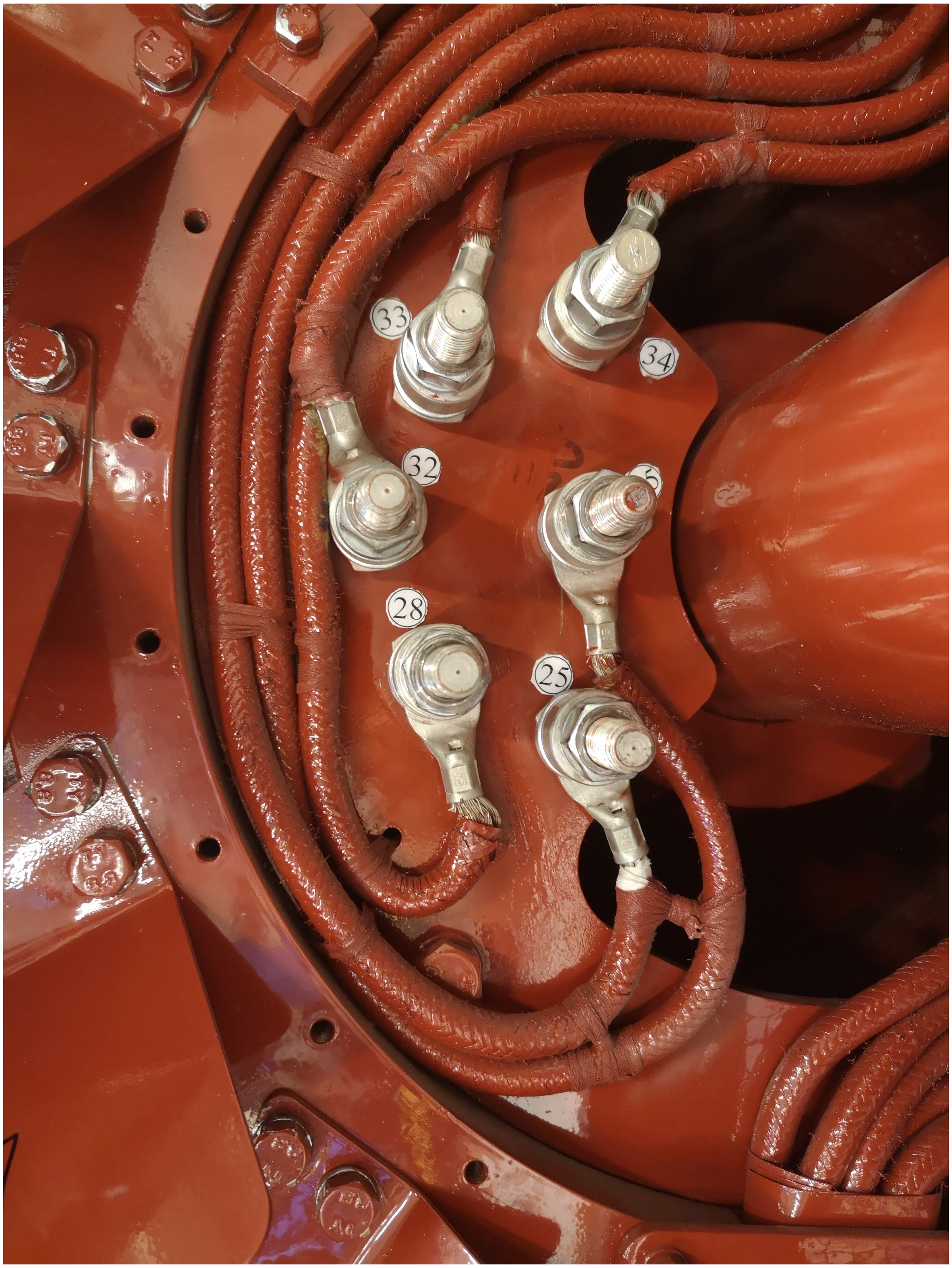


Figure 9: Setup for NLH case

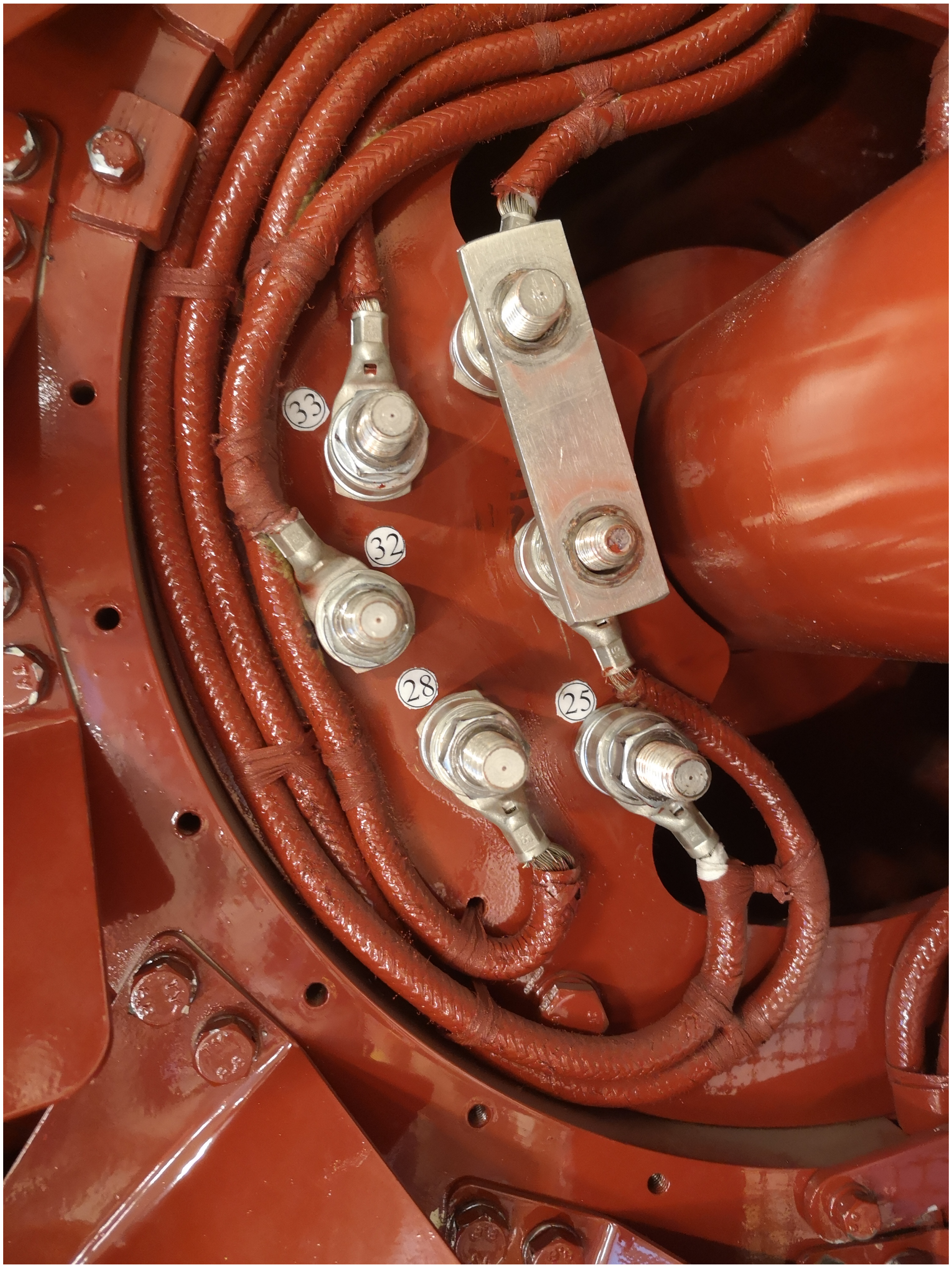


Figure 10: Setup for NL1ITSC case

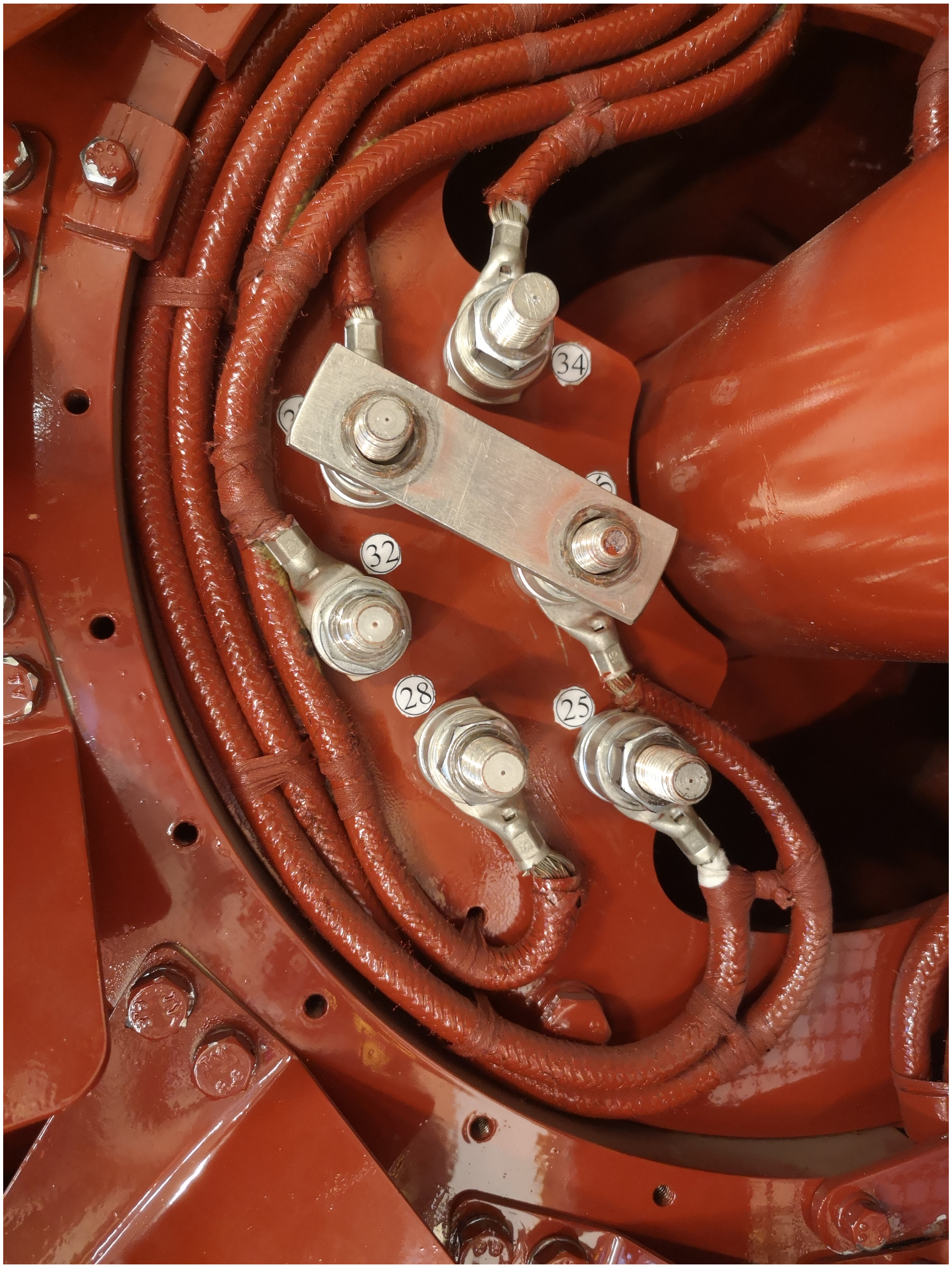


Figure 11: Setup for NL2ITSC case

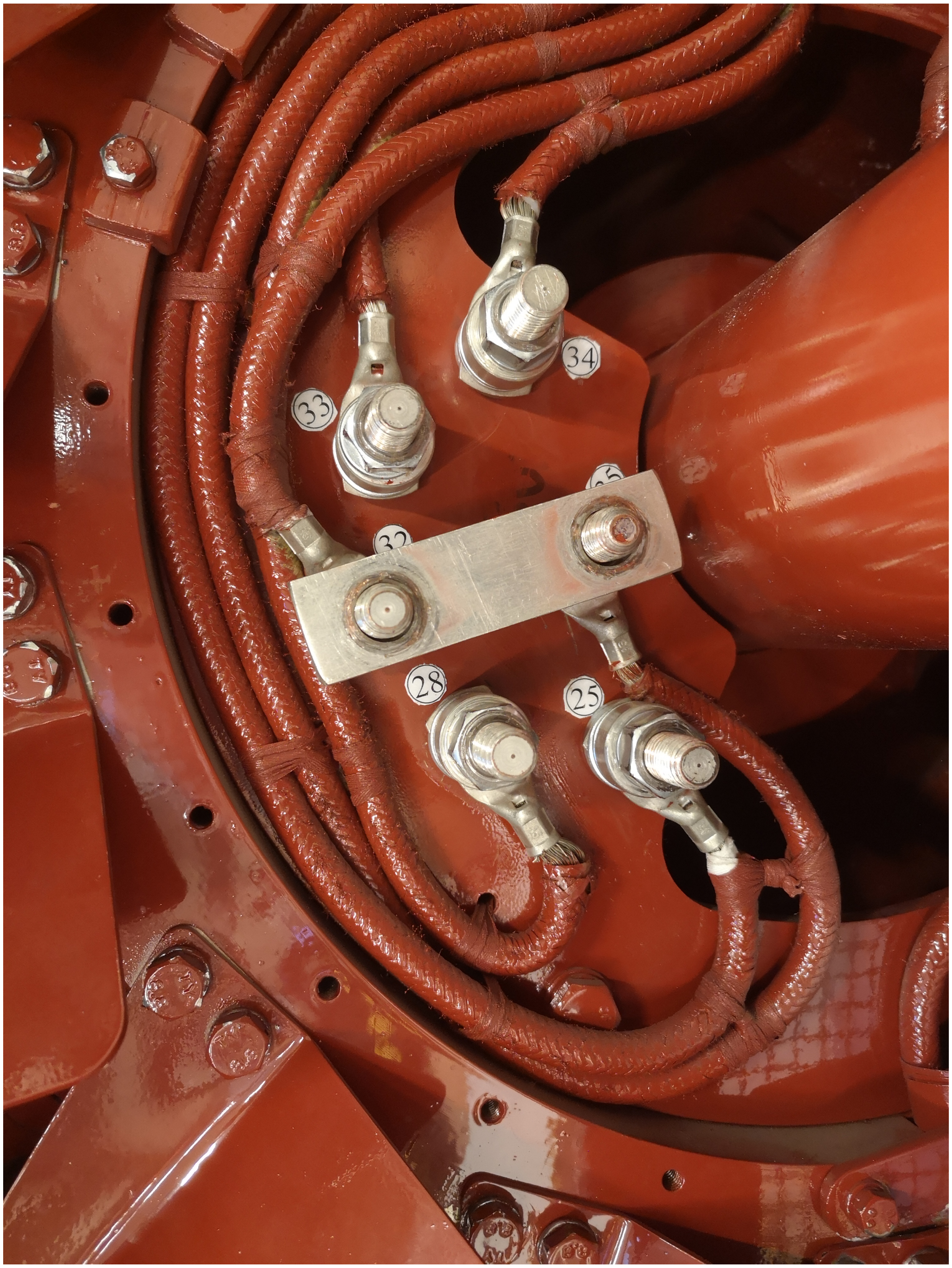


Figure 12: Setup for NL3ITSC case

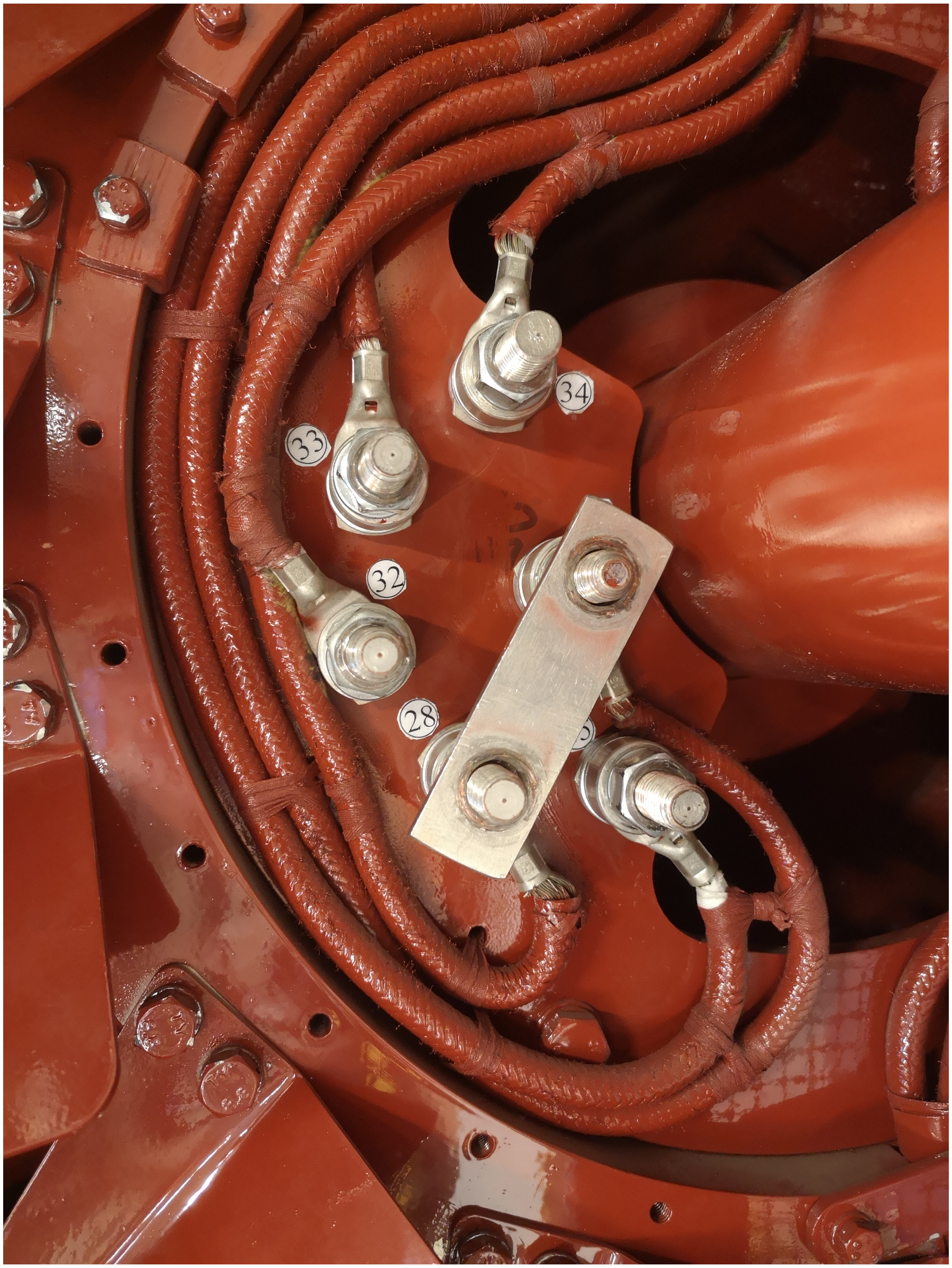


Figure 13: Setup for NL7ITSC case

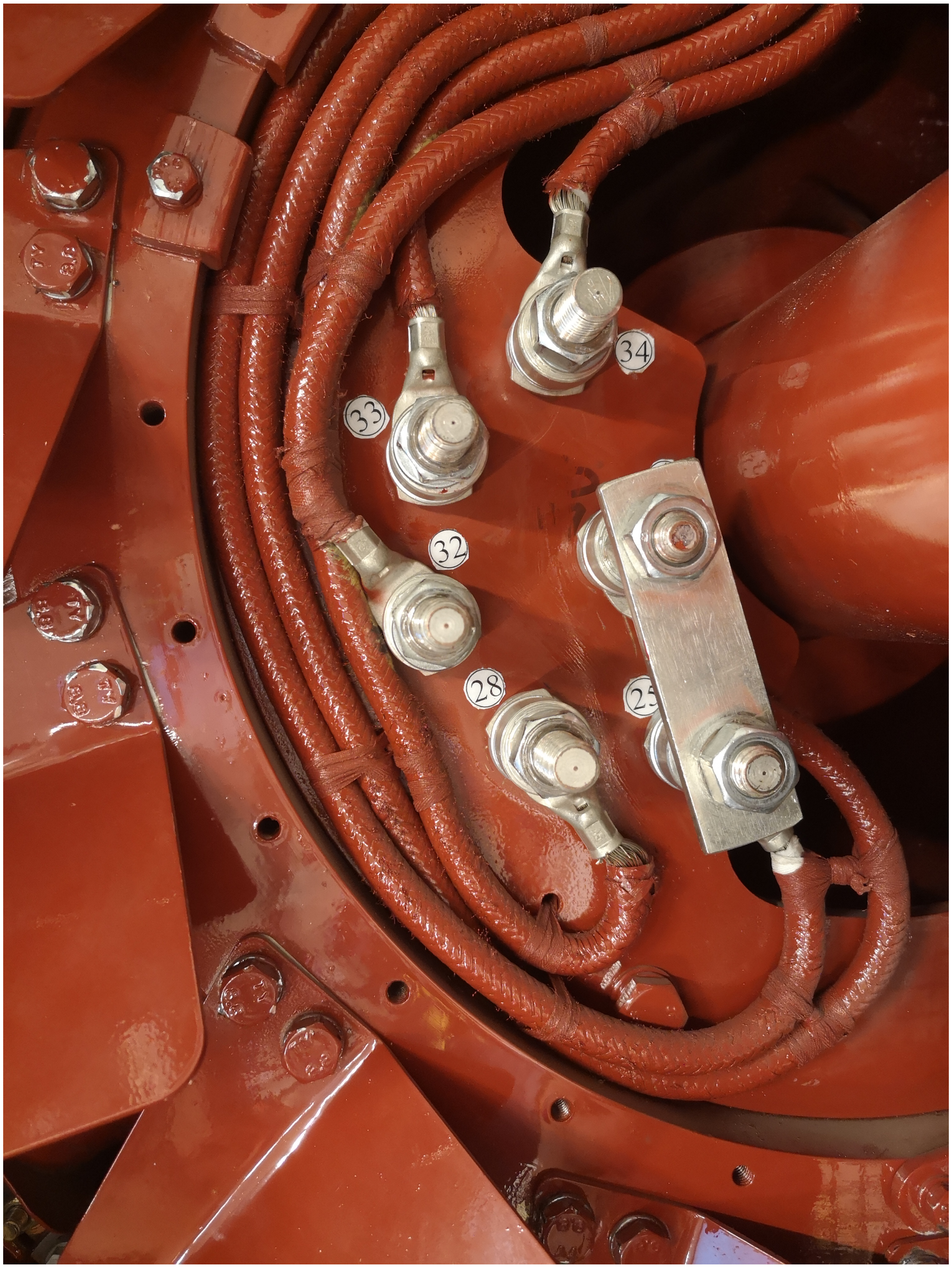


Figure 14: Setup for NL10ITSC case

.3 Continuous wavelet transformation plots

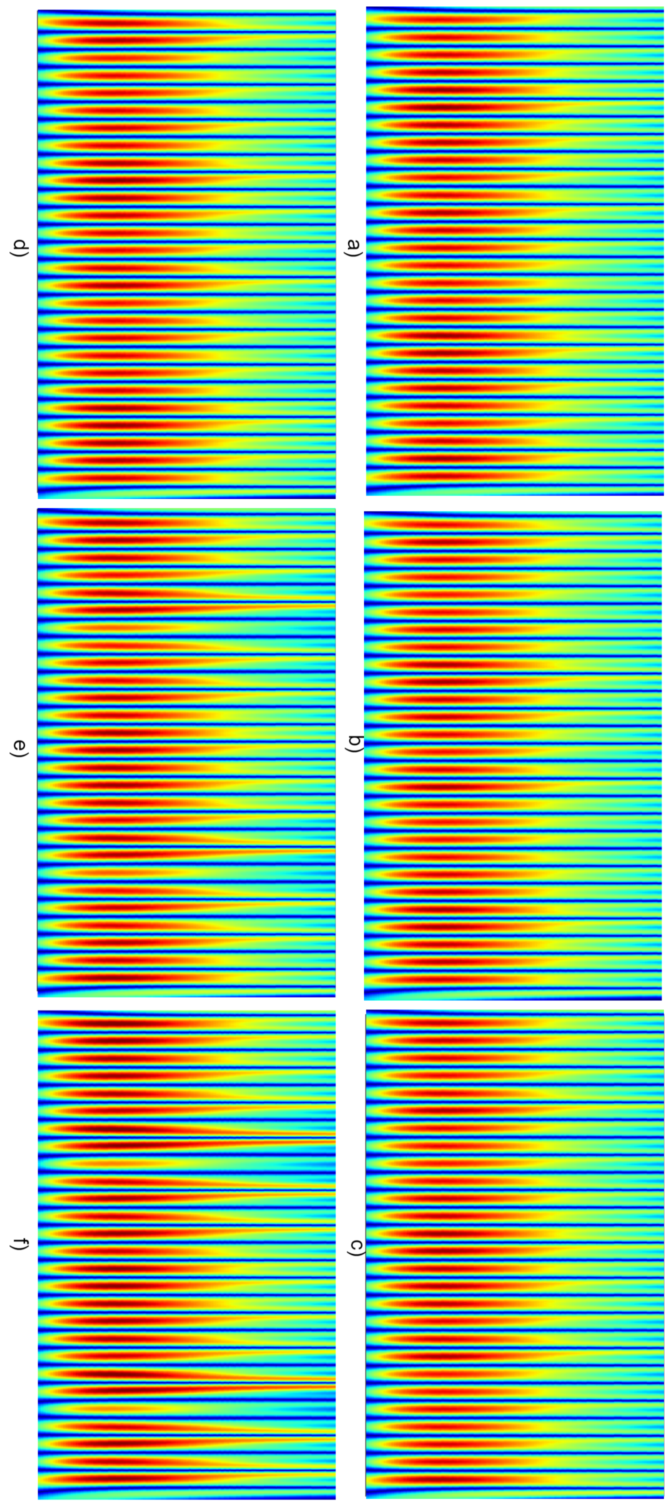


Figure 15: Experimental No-load sensor 1 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

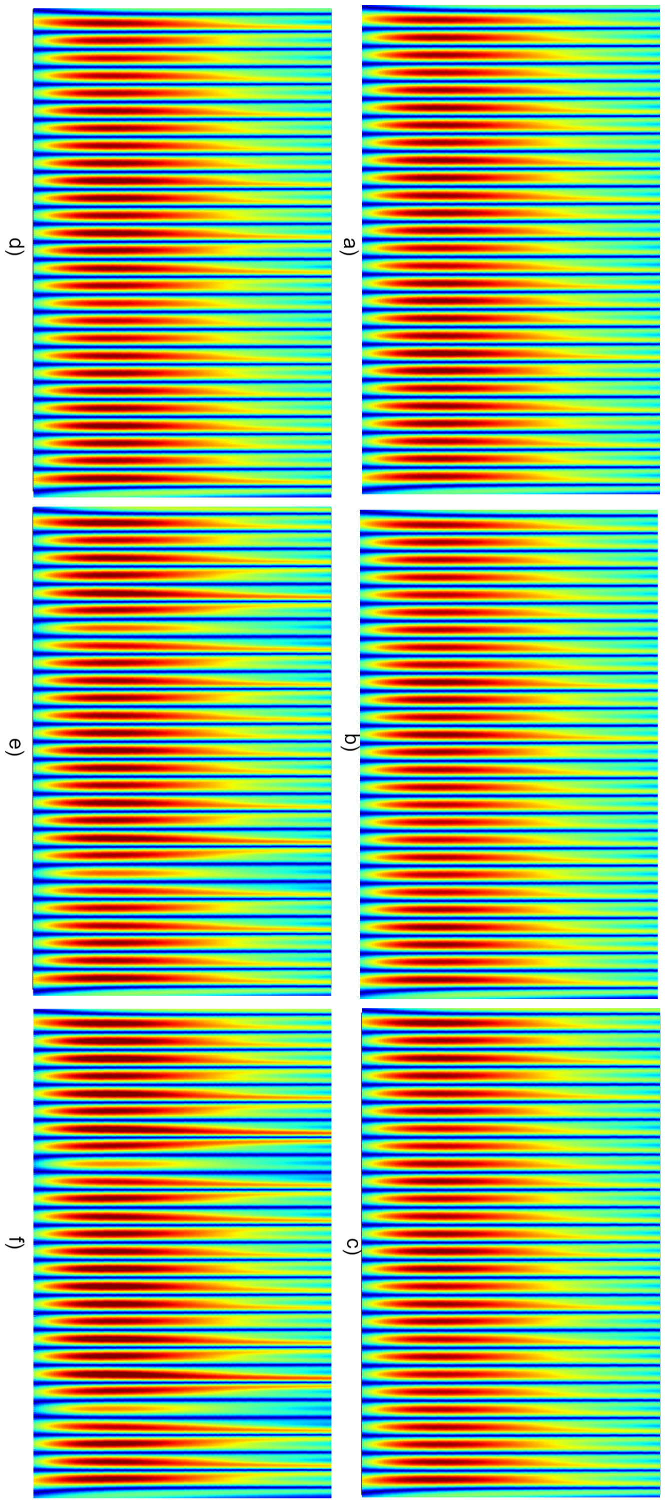


Figure 16: Experimental No-load sensor 2 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

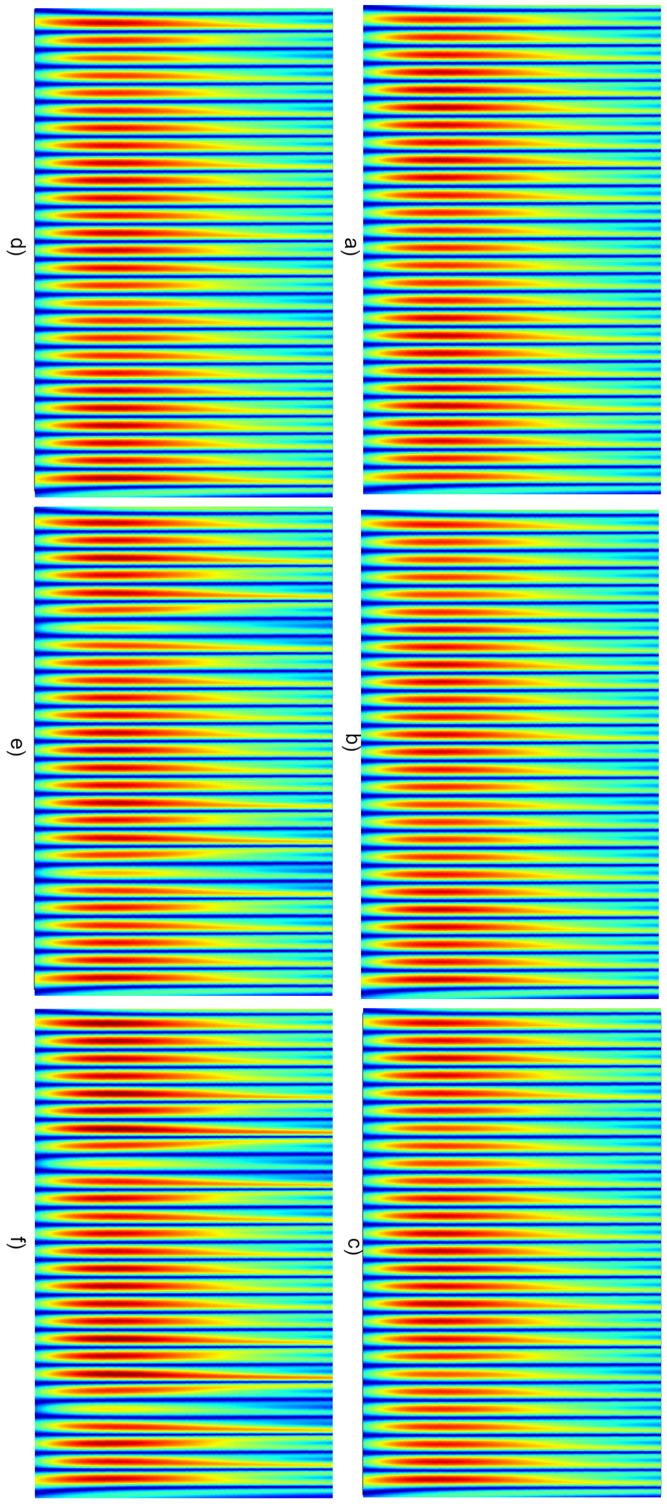


Figure 17: Experimental No-load sensor 3 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

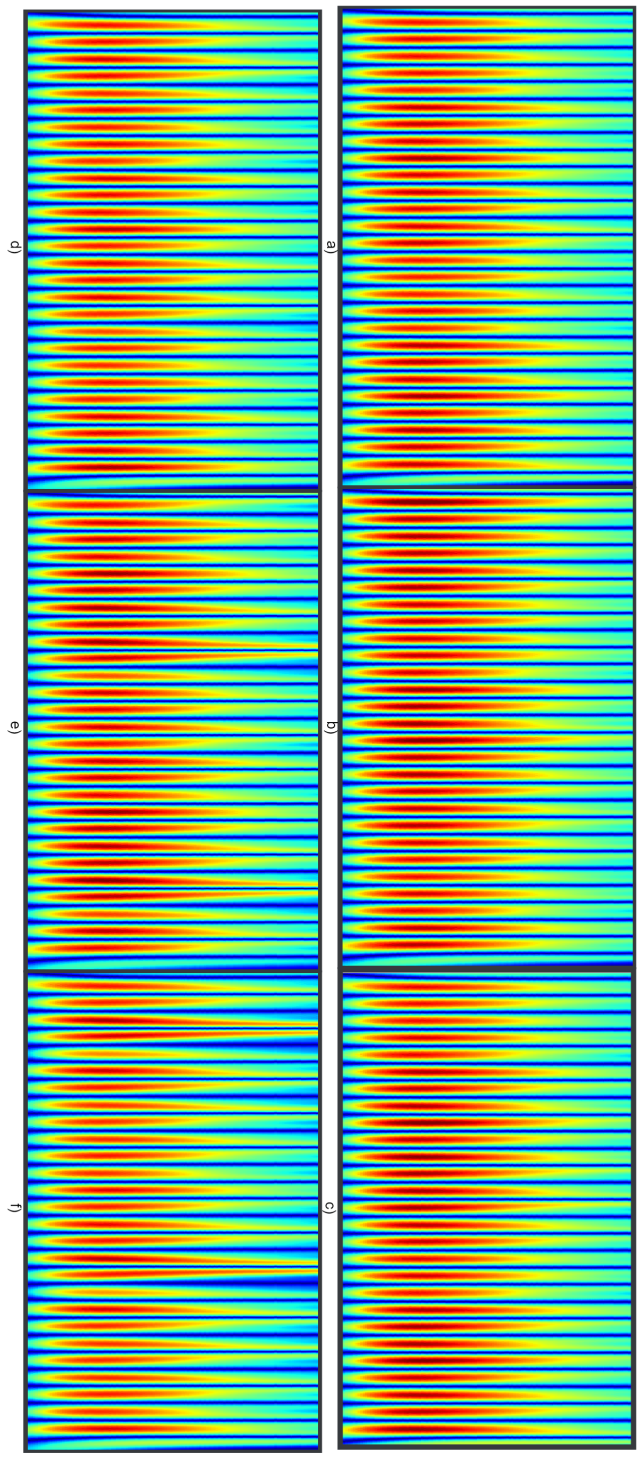


Figure 18: Experimental Full-load sensor 1 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

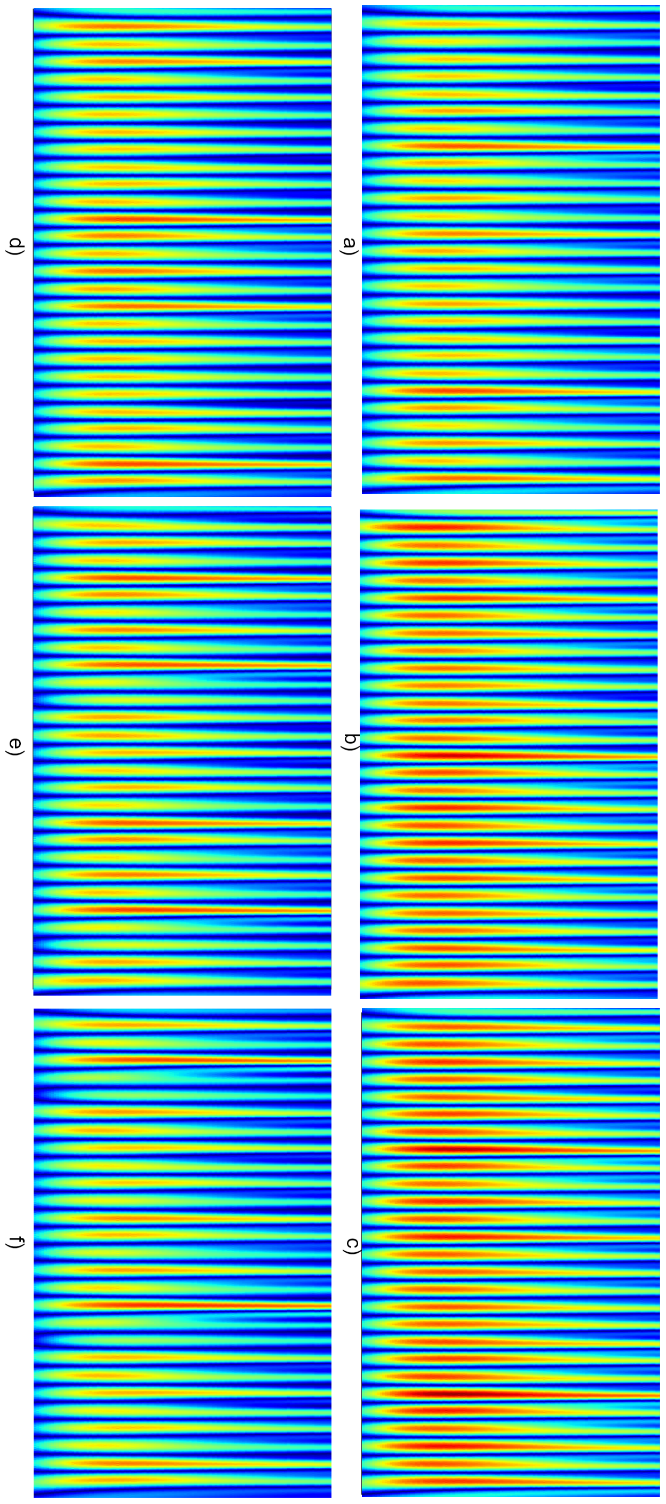


Figure 19: Experimental Full-load sensor 2 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

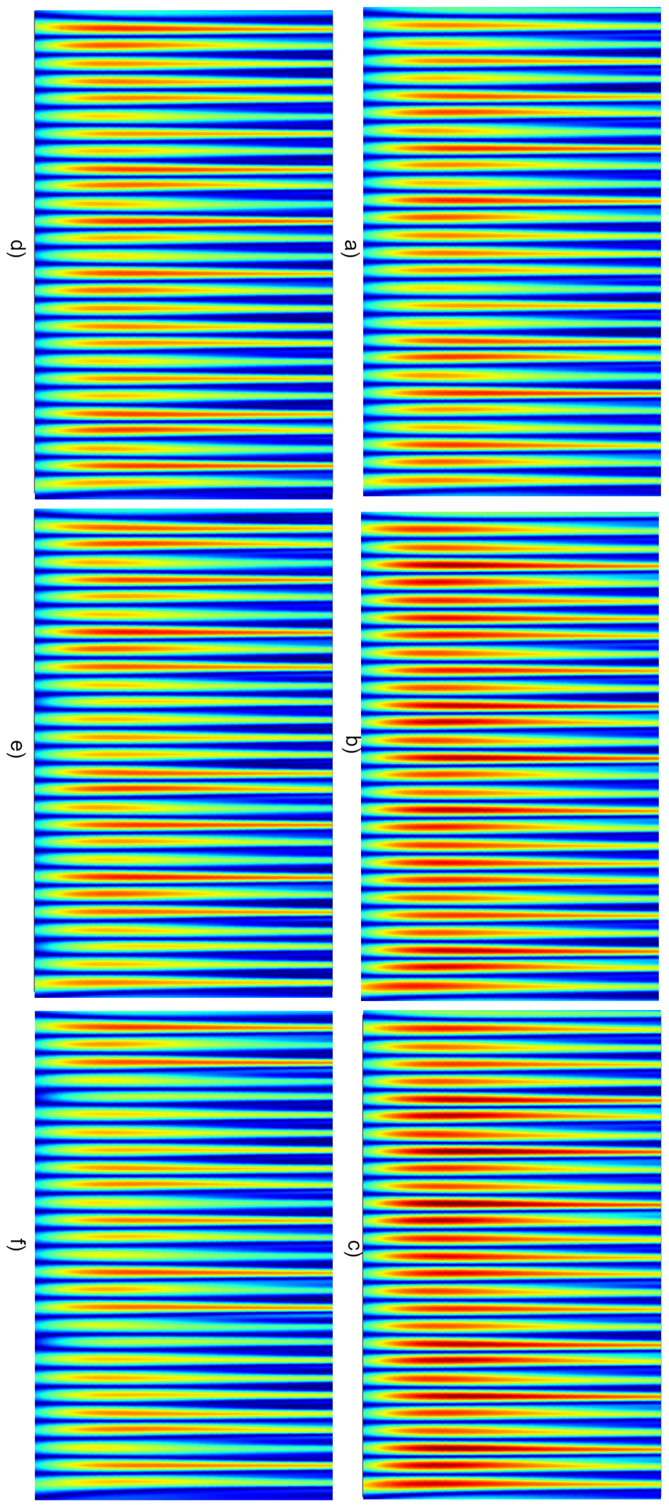


Figure 20: Experimental Full-load sensor 3 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

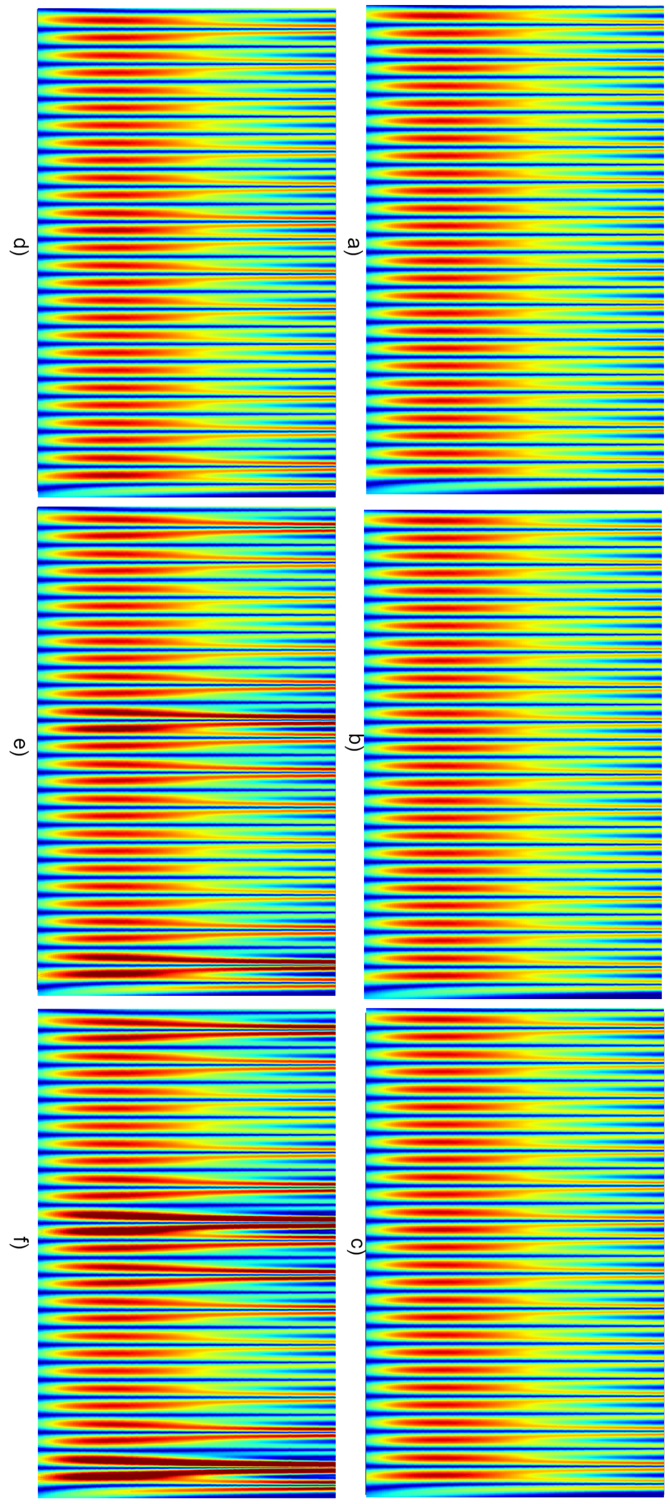


Figure 21: Simulated No-load sensor 1 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

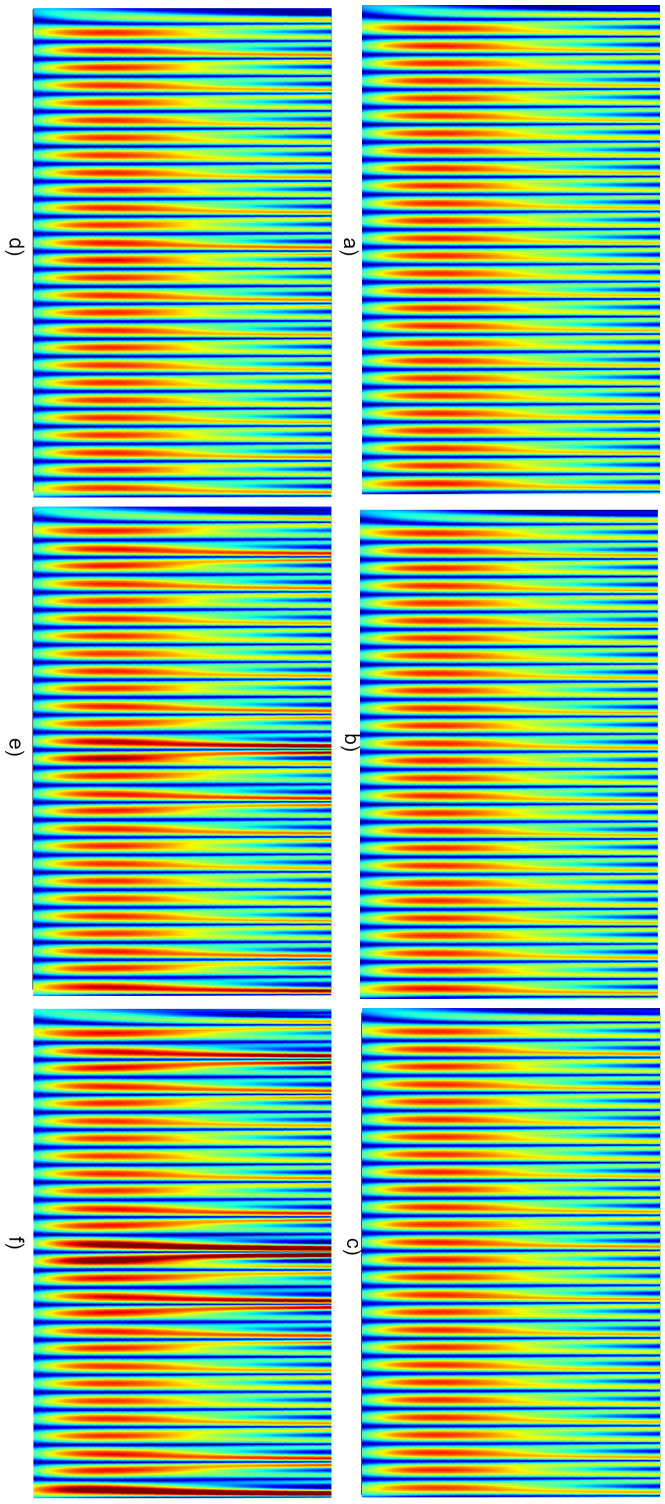


Figure 22: Simulated No-load sensor 2 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

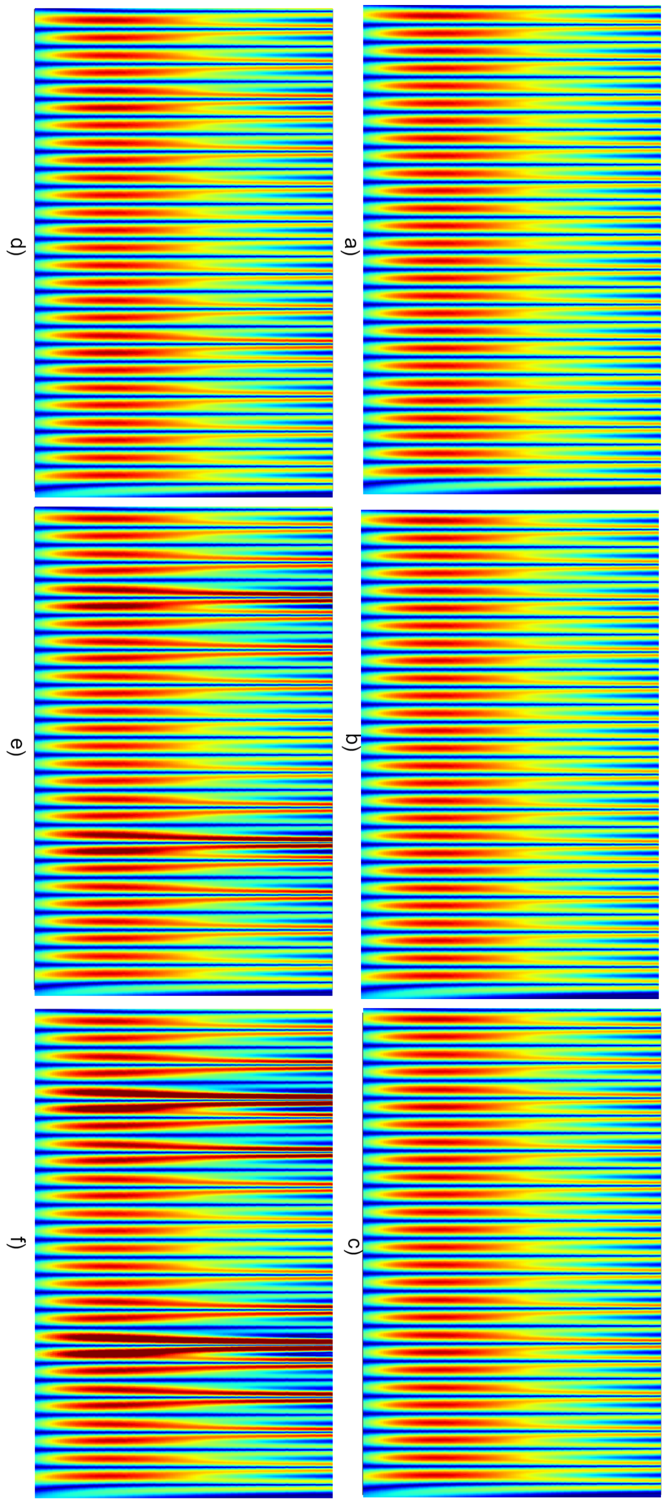


Figure 23: Simulated No-load sensor 3 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

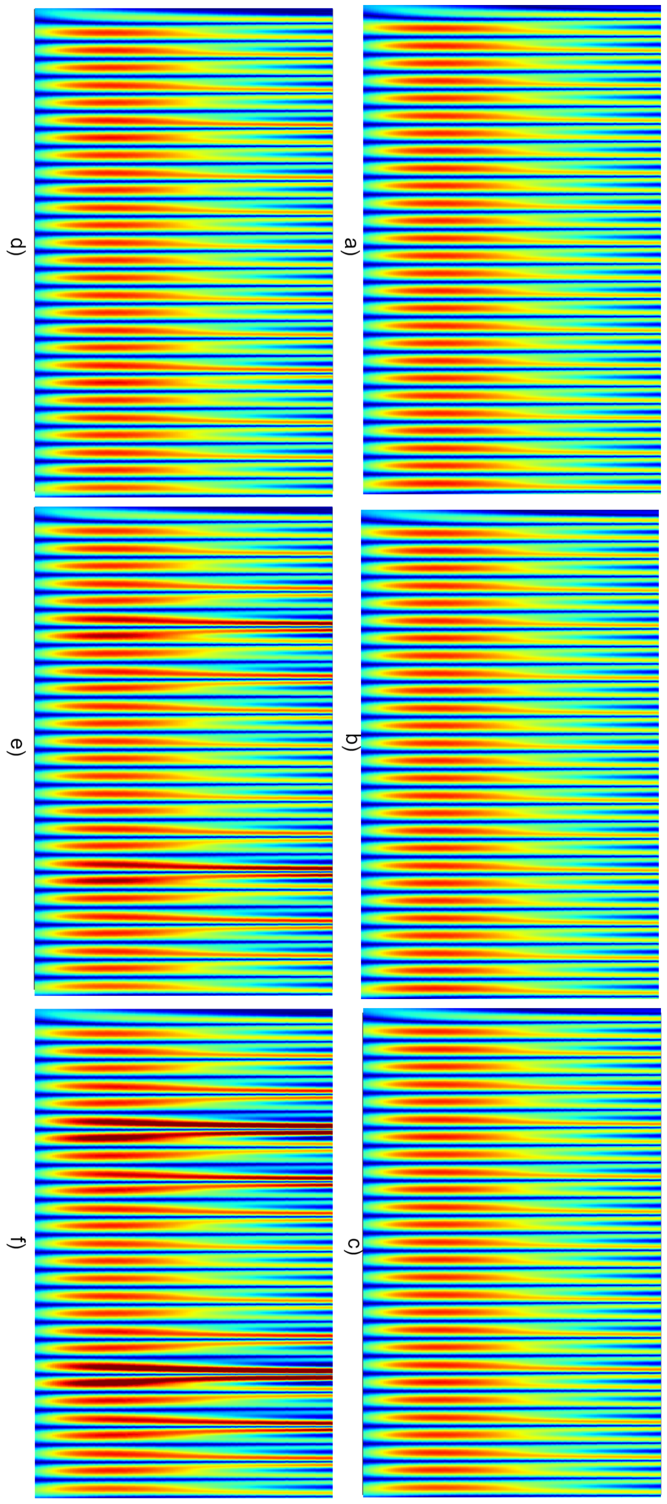


Figure 24: Simulated No-load sensor 4 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

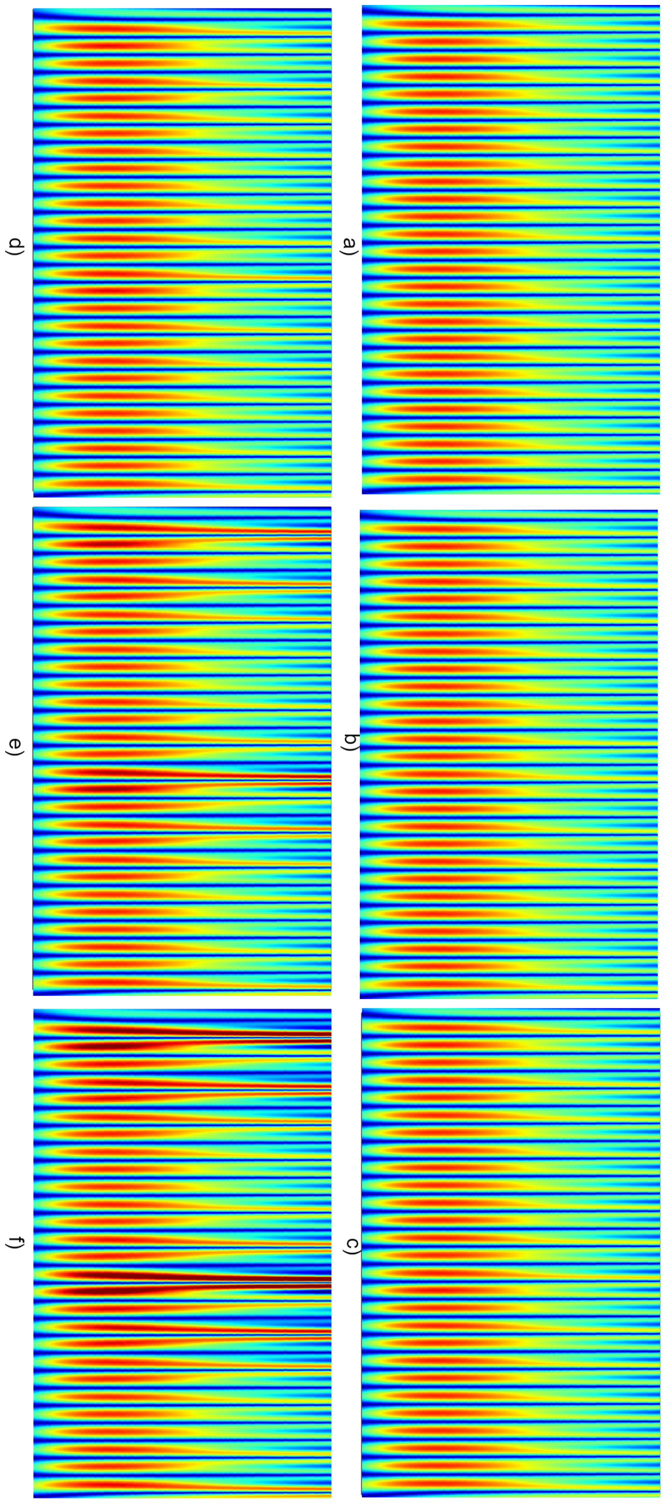


Figure 25: Simulated No-load sensor 5 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

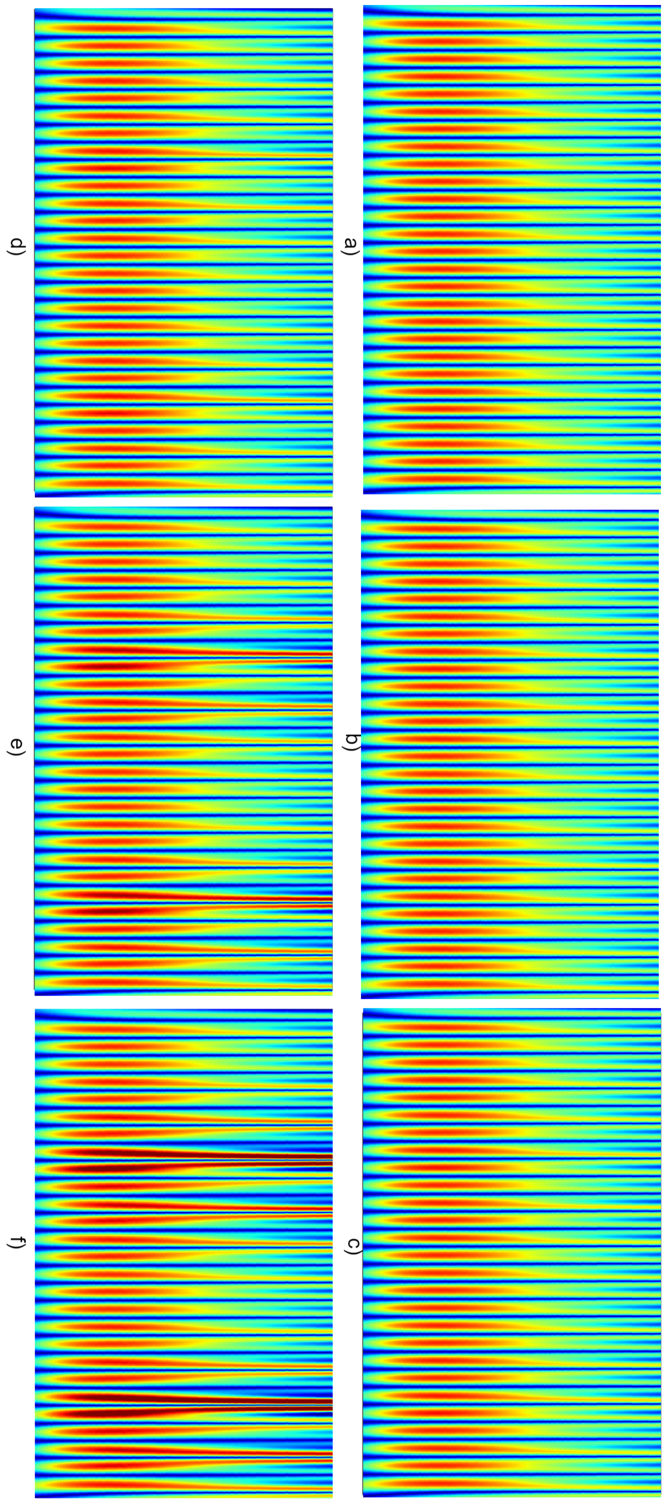


Figure 26: Simulated No-load sensor 6 data CWT ITSC comparison where a) Healthy b) 1ITSC c) 2ITSC d) 3ITSC e) 7ITSC f) 10ITSC

.4 Mean intensity method

.4.1 Experimental results Full-load

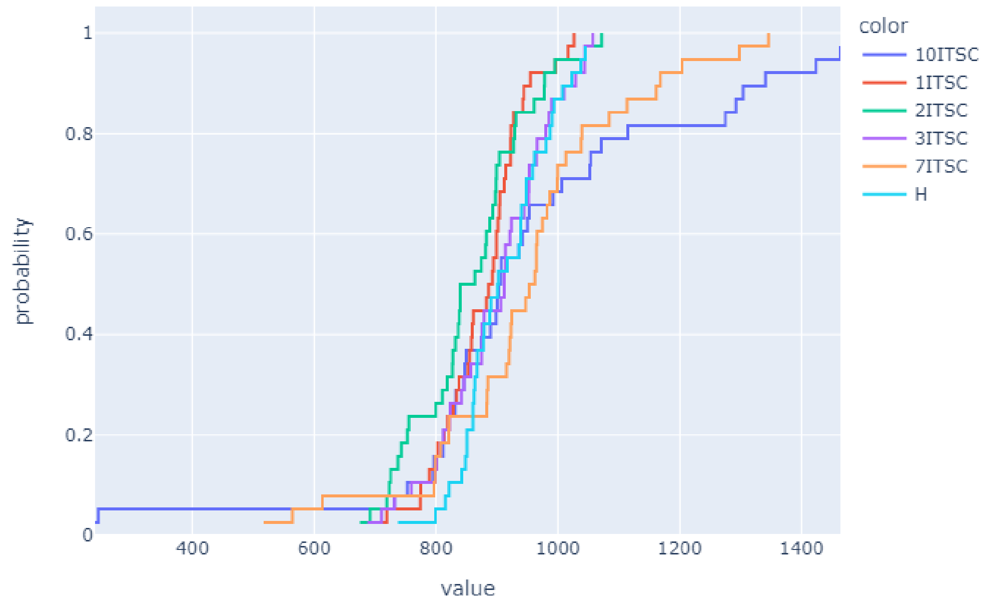


Figure 27: Experimental result of mean intensity ECDF plot for full-load ITSC faults sensor 1

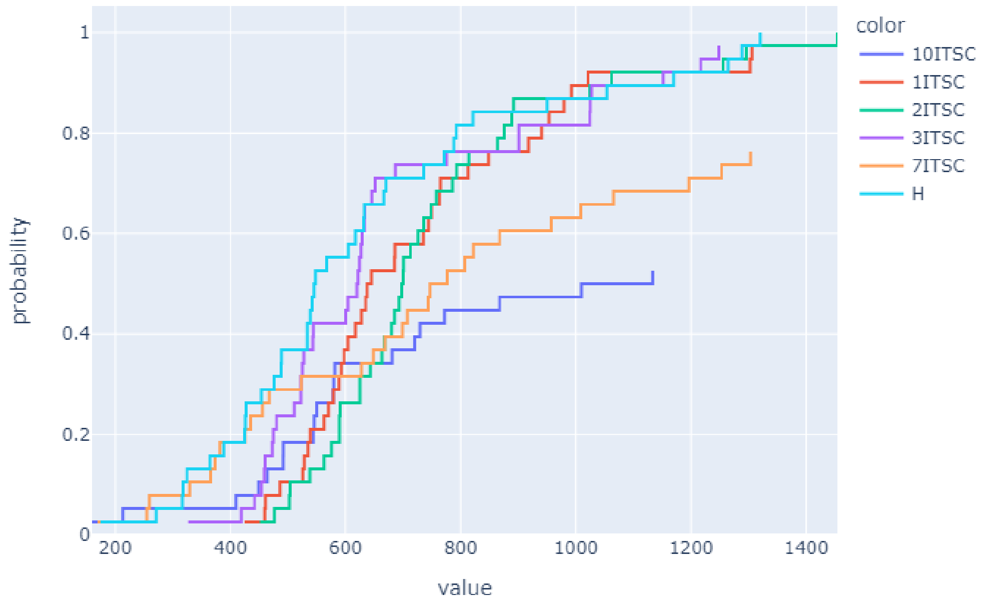


Figure 28: Experimental result of mean intensity ECDF plot for full-load ITSC faults sensor 2

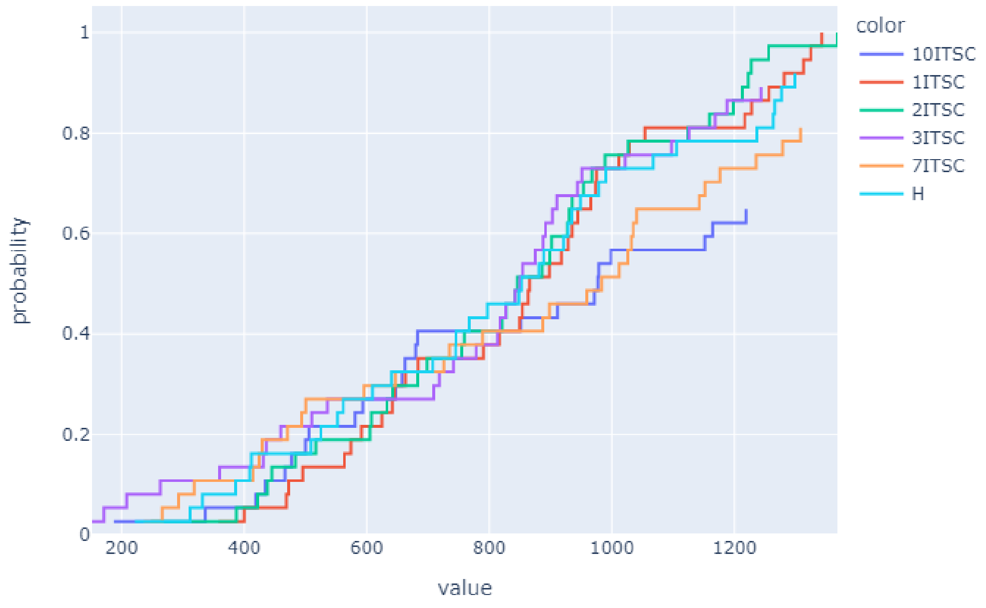


Figure 29: Experimental result of mean intensity ECDF plot for full-load ITSC faults sensor 3

.4.2 Experimental results No-load

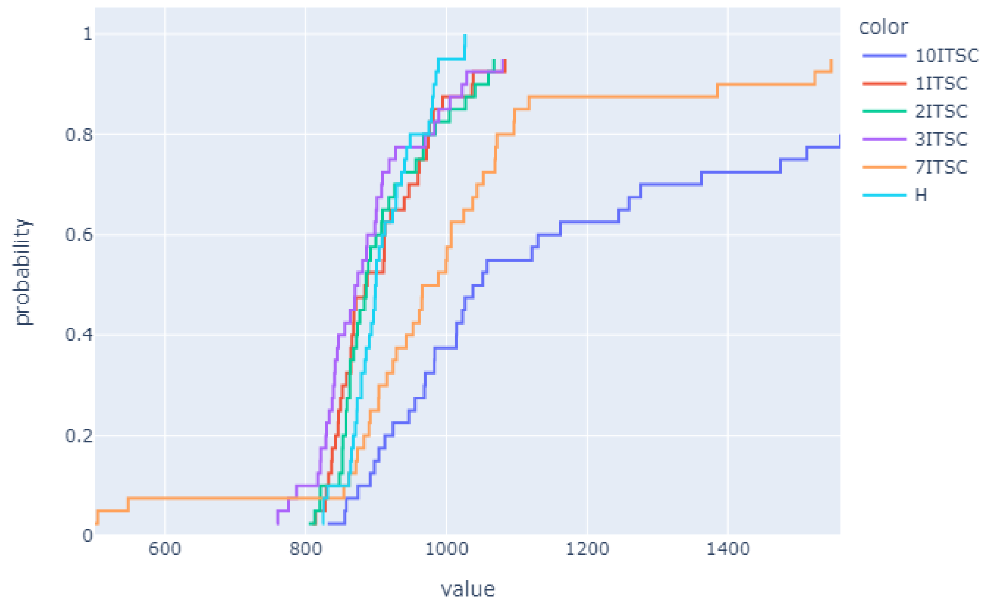


Figure 30: Experimental result of mean intensity ECDF plot for no-load ITSC faults sensor 1

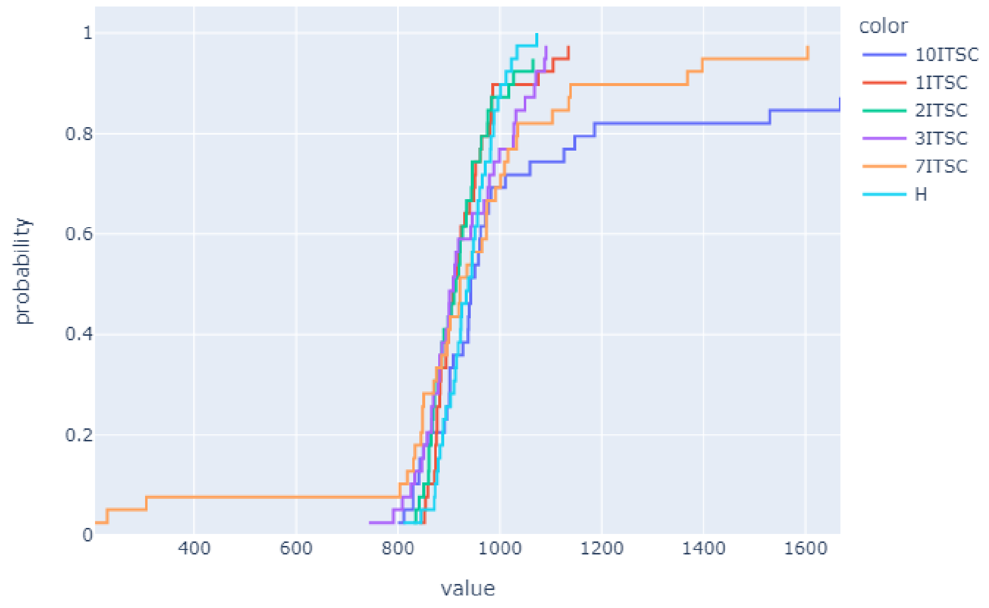


Figure 31: Experimental result of mean intensity ECDF plot for no-load ITSC faults sensor 2

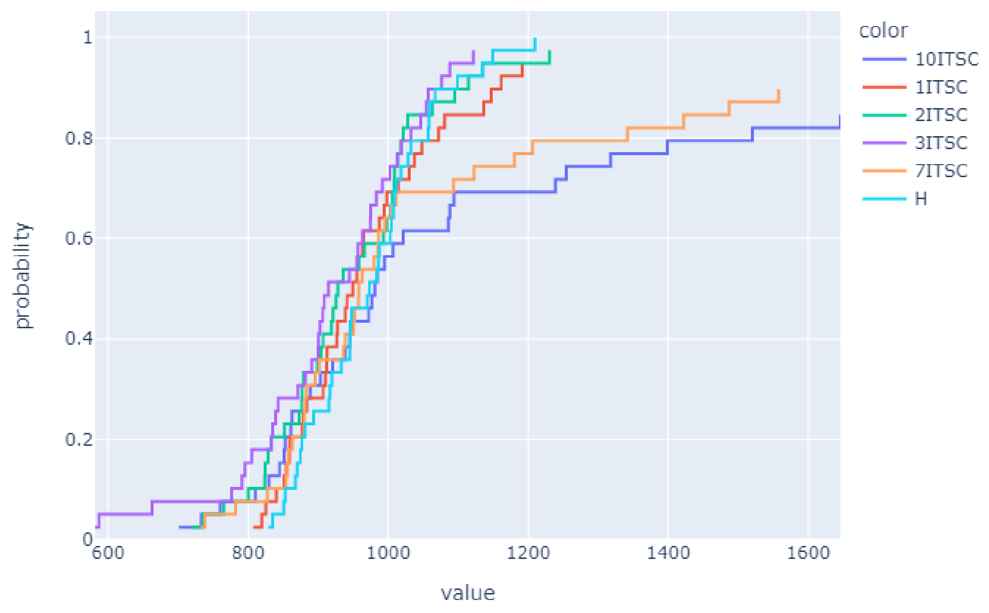


Figure 32: Experimental result of mean intensity ECDF plot for no-load ITSC faults sensor 3

.4.3 Simulation results no-load

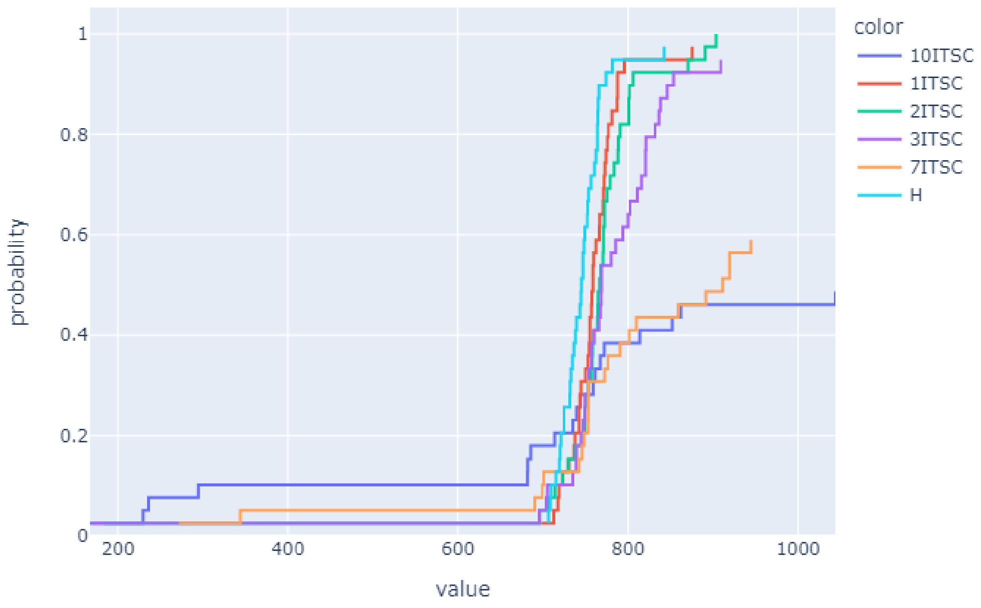


Figure 33: Simulated result of mean intensity ECDF plot for no-load ITSC faults sensor 1

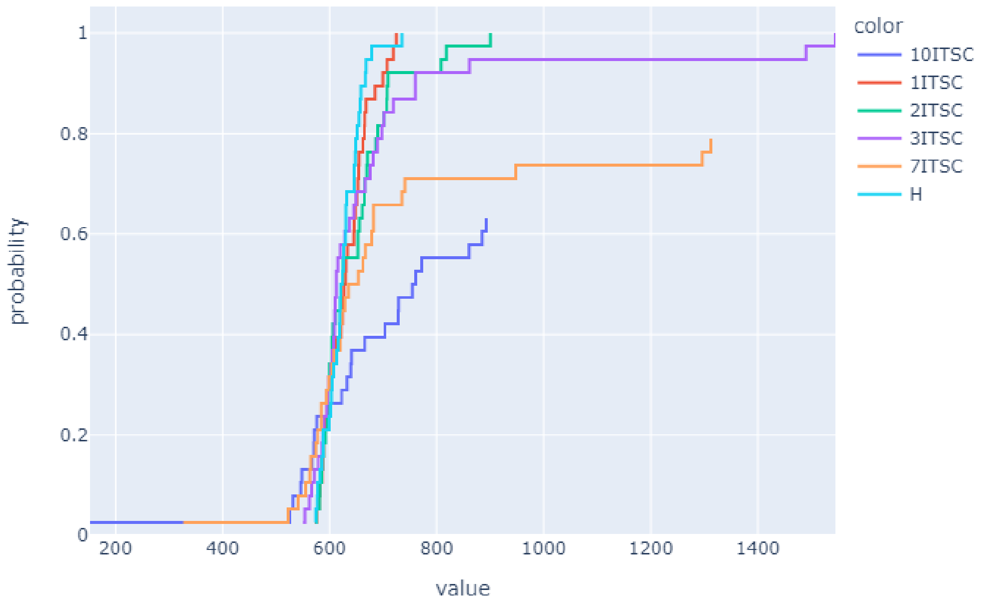


Figure 34: Simulated result of mean intensity ECDF plot for no-load ITSC faults sensor 2

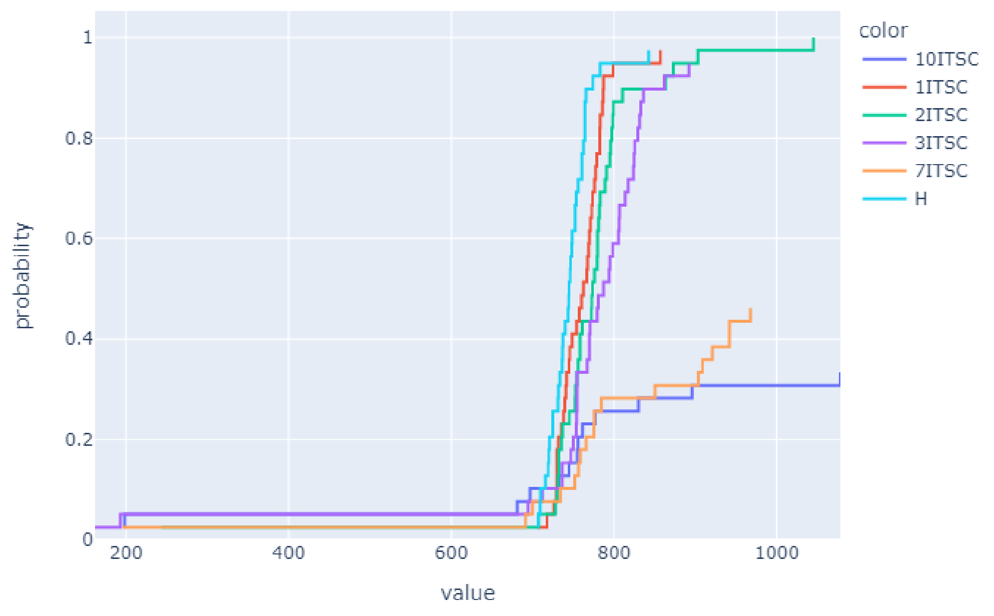


Figure 35: Simulated result of mean intensity ECDF plot for no-load ITSC faults sensor 3

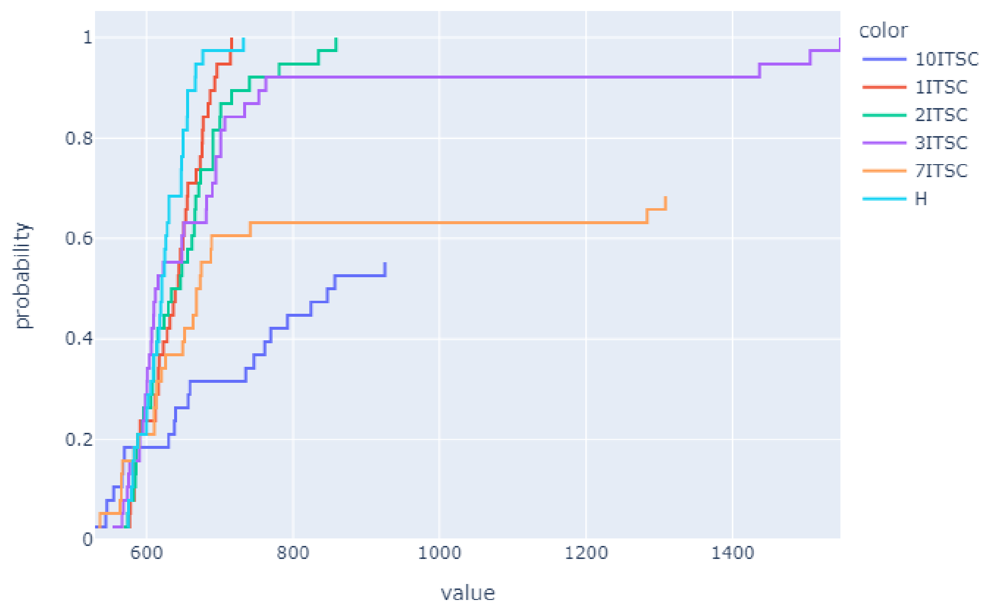


Figure 36: Simulated result of mean intensity ECDF plot for no-load ITSC faults sensor 4

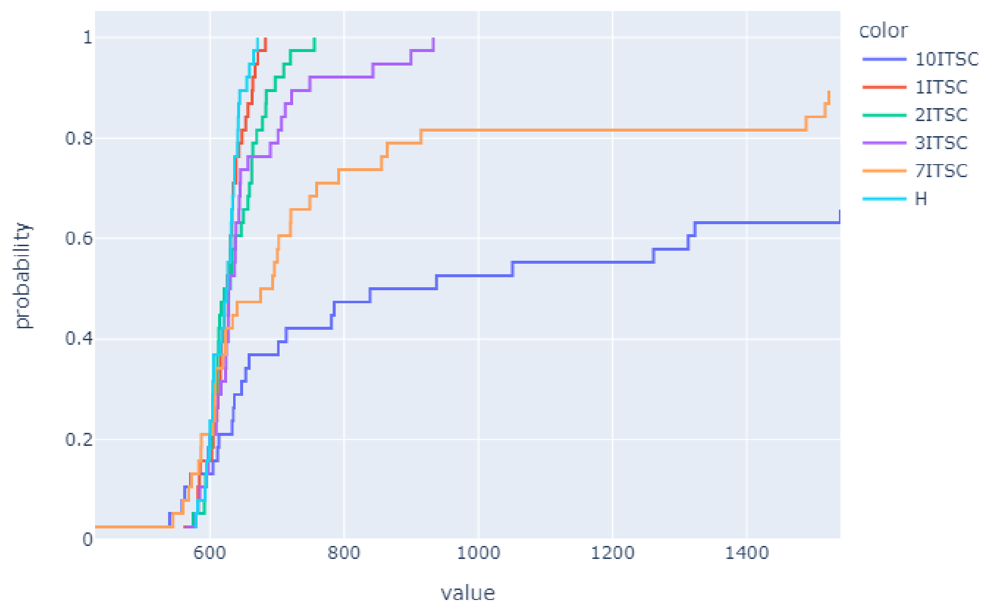


Figure 37: Simulated result of mean intensity ECDF plot for no-load ITSC faults sensor 5

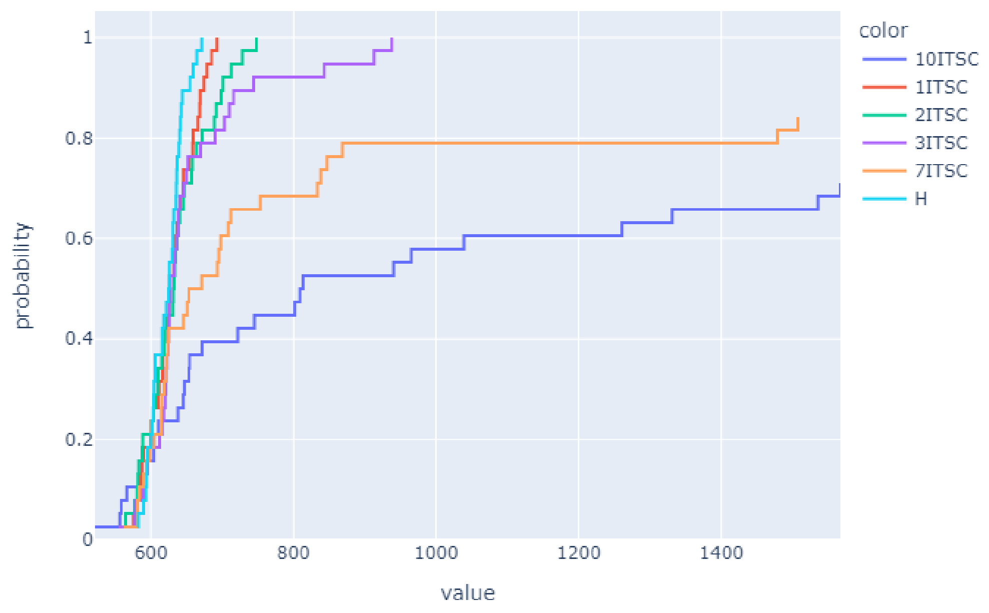


Figure 38: Simulated result of mean intensity ECDF plot for no-load ITSC faults sensor 6

.5 Difference method

.5.1 Experimental difference results Full-load

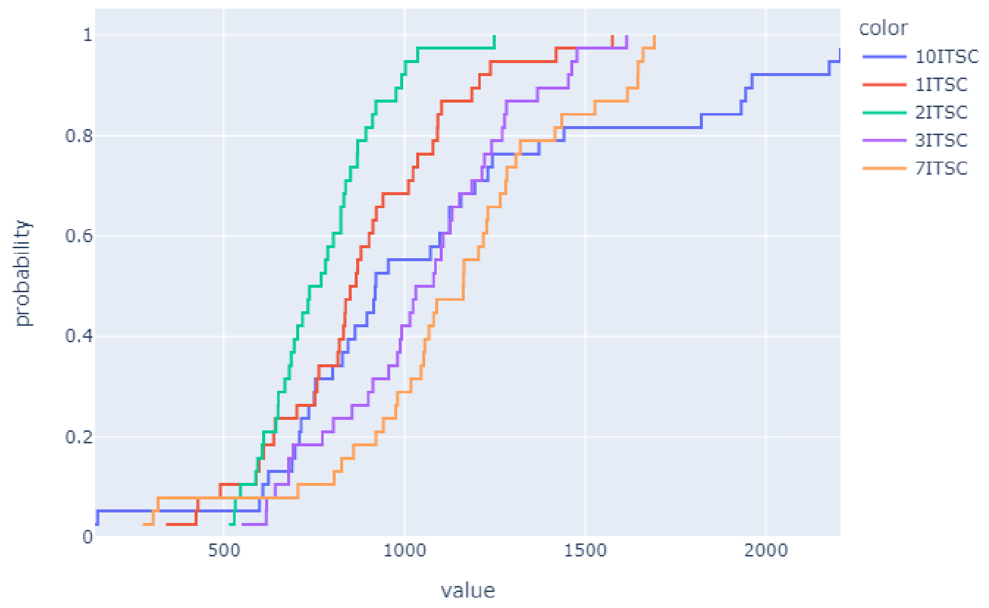


Figure 39: Experimental result ECDF difference plot for full-load ITSC faults sensor 1

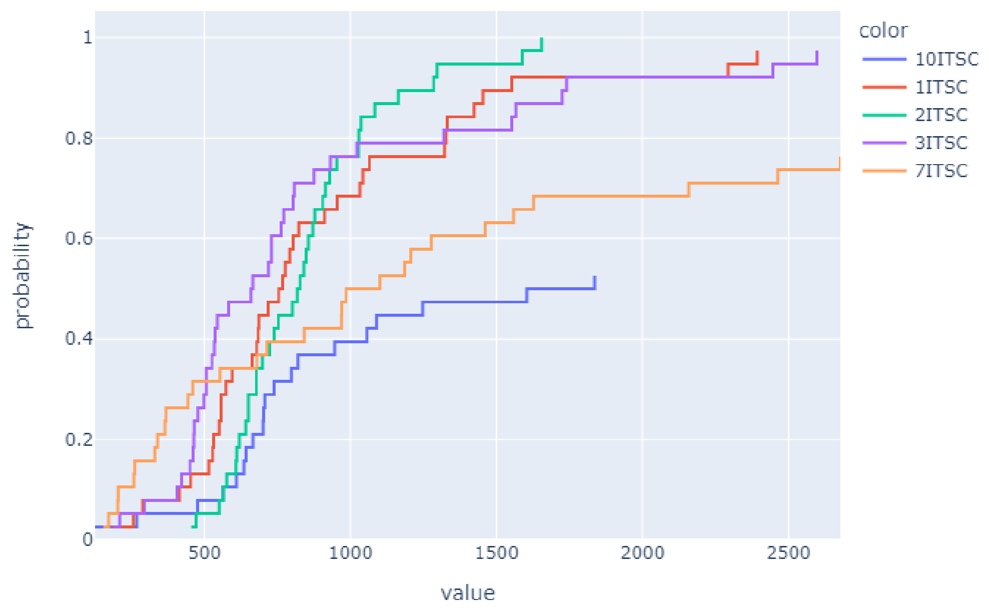


Figure 40: Experimental result ECDF difference plot for full-load ITSC faults sensor 2

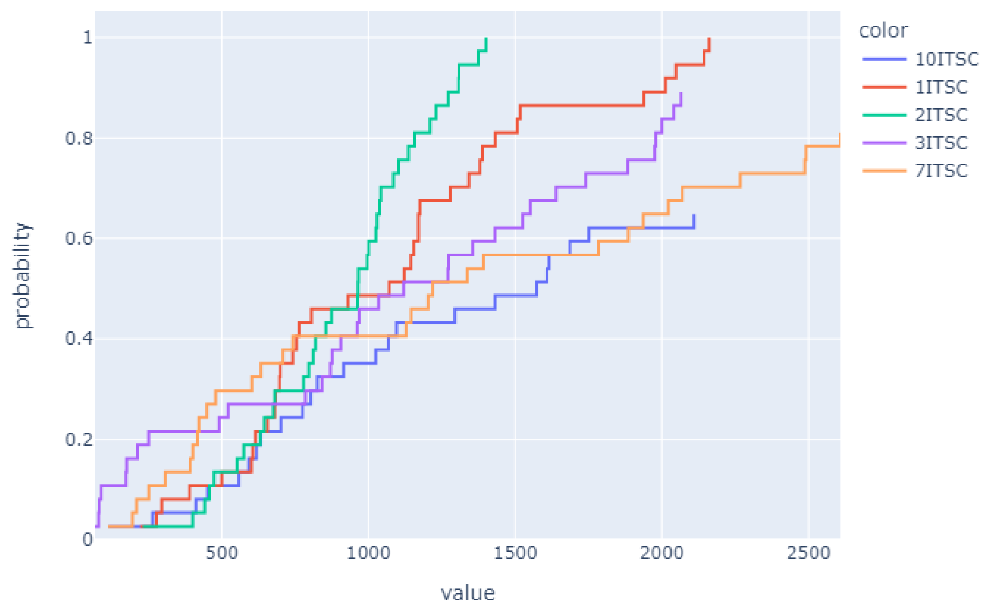


Figure 41: Experimental result ECDF difference plot for full-load ITSC faults sensor 3

.5.2 Experimental difference results no-load

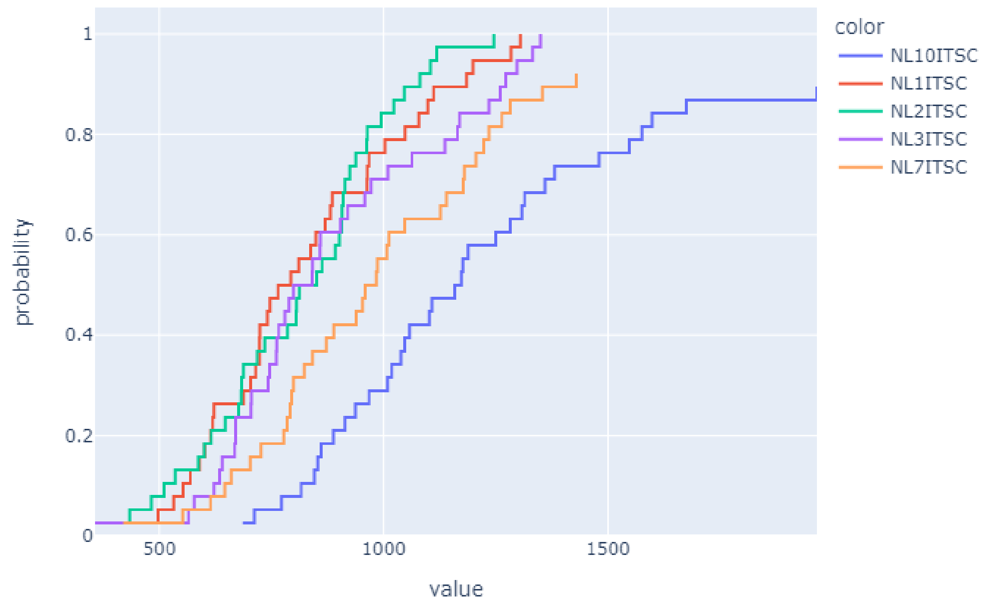


Figure 42: Experimental result ECDF difference plot for no-load ITSC faults sensor 1

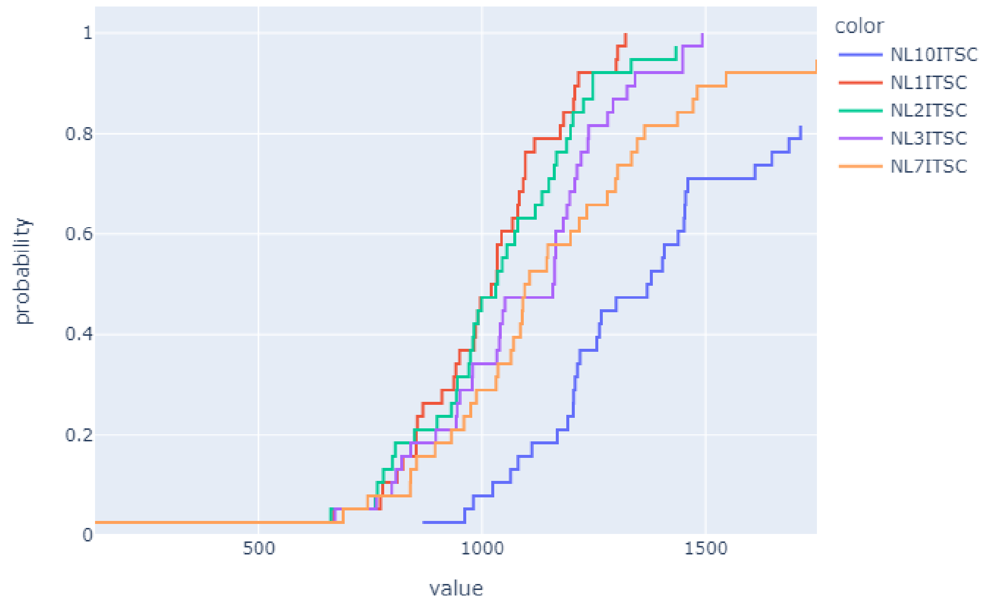


Figure 43: Experimental result ECDF difference plot for no-load ITSC faults sensor 2

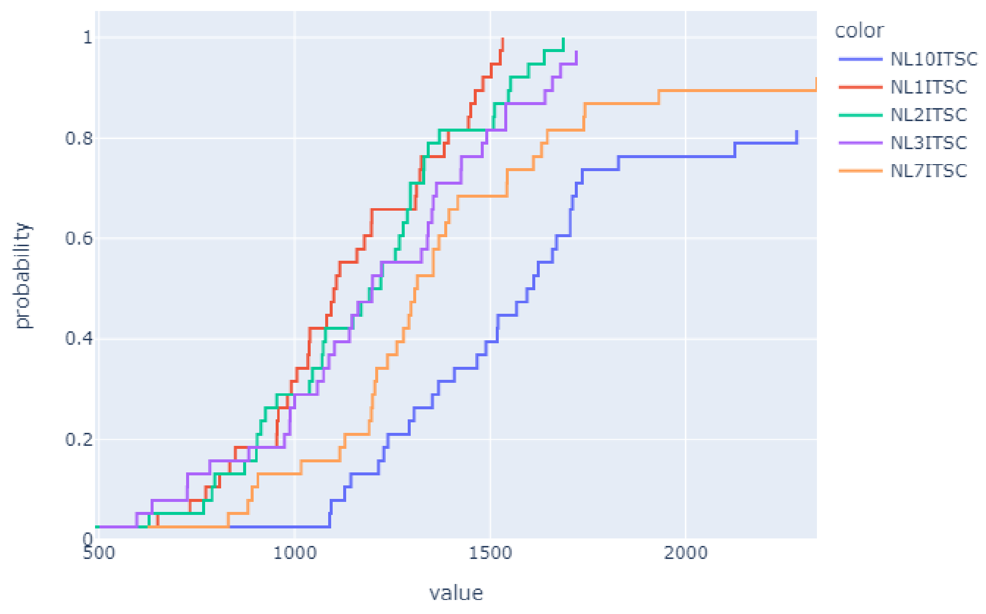


Figure 44: Experimental result ECDF difference plot for no-load ITSC faults sensor 3

.5.3 Simulation difference results no-load

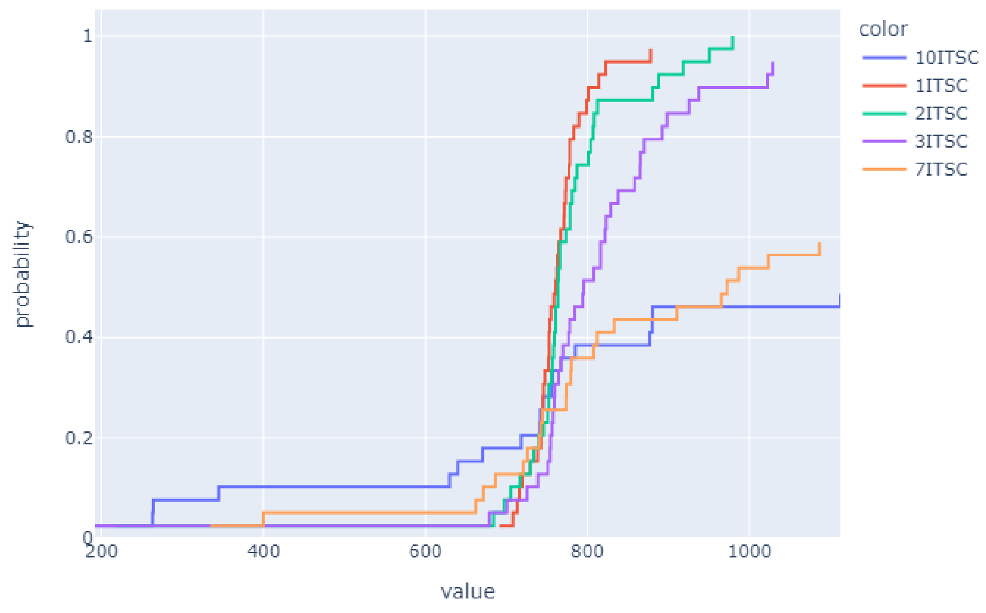


Figure 45: Simulated result ECDF difference plot for no-load ITSC faults sensor 1

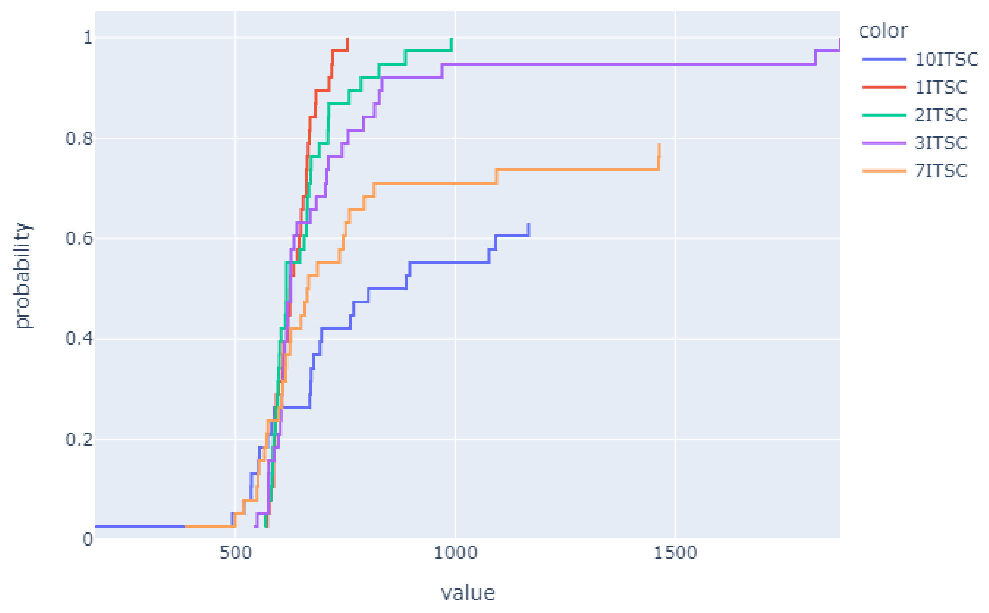


Figure 46: Simulated result ECDF difference plot for no-load ITSC faults sensor 2

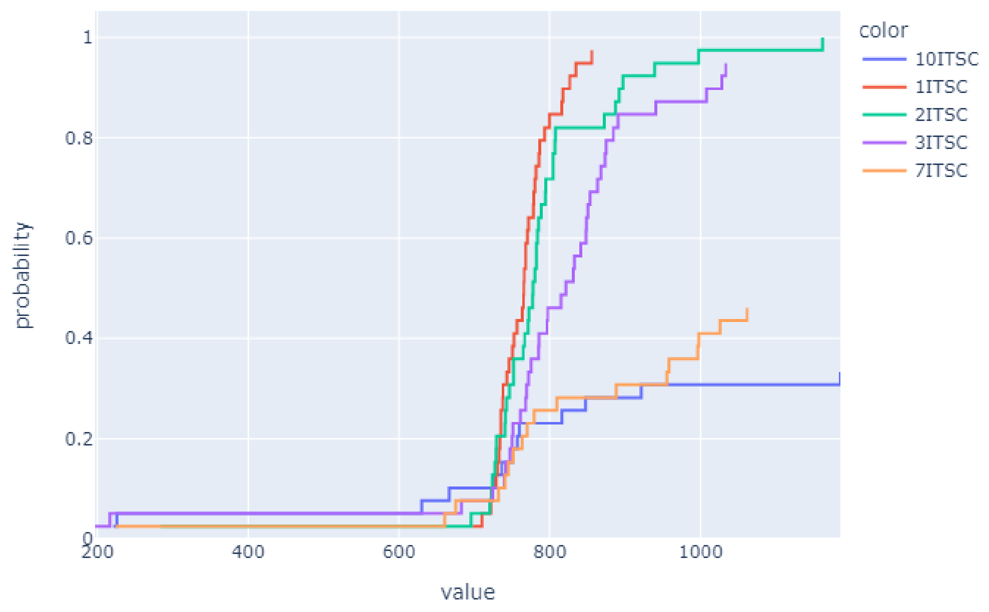


Figure 47: Simulated result ECDF difference plot for no-load ITSC faults sensor 3

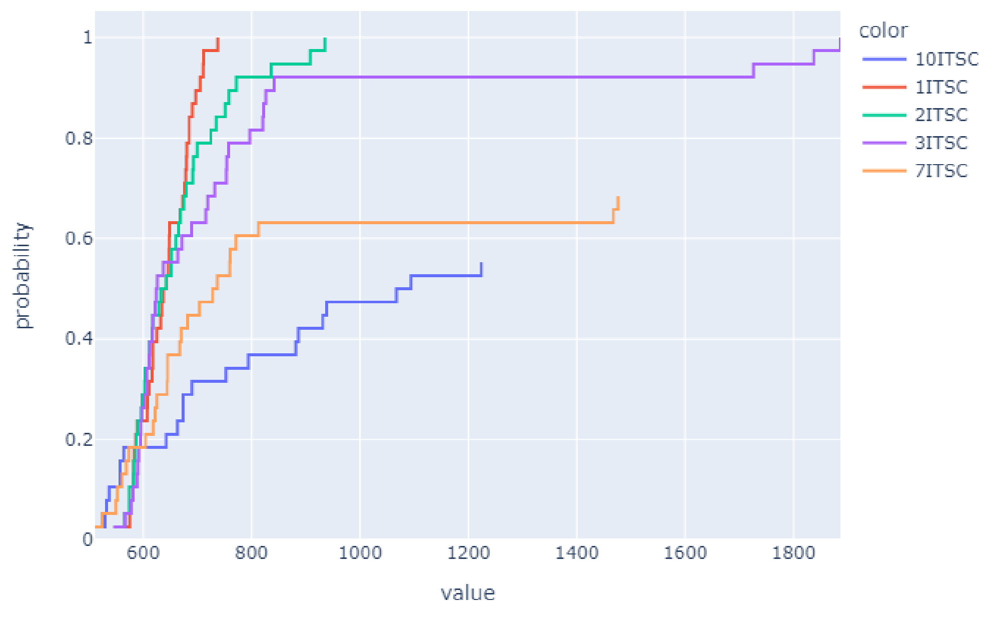


Figure 48: Simulated result ECDF difference plot for no-load ITSC faults sensor 4

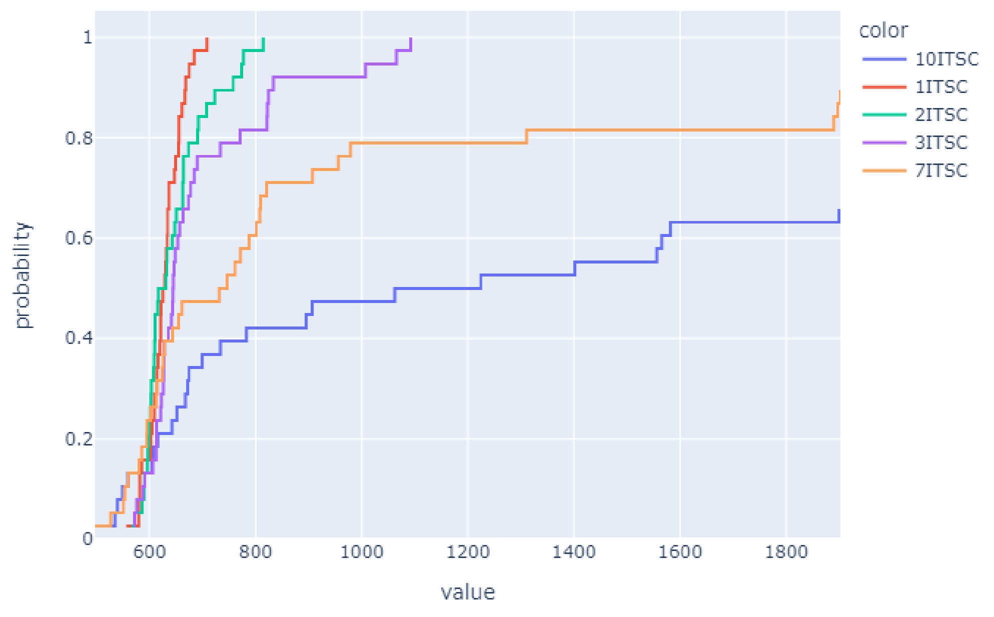


Figure 49: Simulated result ECDF difference plot for no-load ITSC faults sensor 5

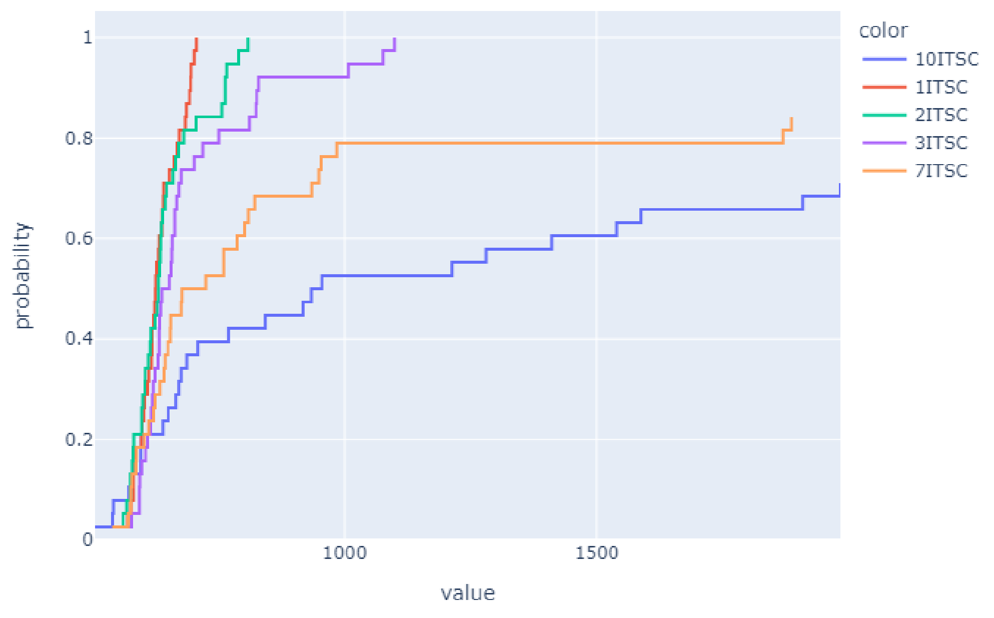


Figure 50: Simulated result ECDF difference plot for no-load ITSC faults sensor 6

.6 Python code

.6.1 Continuous wavelet transform plot acquisition

This code is the result of work done by a previous student Tarjeu Nesbø Skreien[37]. It was slightly modified in order to be able to be iterable in the specialization report[21]

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.colors as colors
import pyhht
import pywt
from PIL import Image
import skimage.io
import skimage.color
import skimage.filters
from skimage.filters import threshold_otsu, threshold_local
import skimage.measure
from skimage.morphology import square
```

```python
from skimage.morphology import reconstruction
from skimage import data
from skimage import img_as_float
import skimage.segmentation
import skimage.morphology
import os
import cv2
import scipy.ndimage
from scipy.ndimage import gaussian_filter
import functools
import plotly.express as px
from pathlib import Path
import csv
from skimage.feature import blob_dog, blob_log, blob_doh
import matplotlib.patches as mpatches
from skimage import data
from skimage.segmentation import clear_border
from skimage.measure import label, regionprops, ...
    regionprops_table
from skimage.morphology import closing, square
from skimage.color import label2rgb
get_ipython().run_line_magic('matplotlib', 'inline')
plt.rcParams.update({'font.size': 20}) # Setting the font size...
     of the plots to 20. Use plt.rcdefaults() to reset to ...
    default.


# # Importing the data

# In[2]:


def normalize_describe(df):
    df_norm = (df - df.mean()) / (df.max() - df.min())
    print(df_norm.describe())
    return df_norm


# # Windowing the data

# In[3]:


def is_valid_crossing(position, data_series, validation_length...
    ):
    # Checks that the series of samples after "position" are ...
    all positive
    is_valid = True
    validation_position = position
    sum_of_samples = 0.0
    for i in range(validation_length):
        current_sample = data_series[position + i] # sampling ...
    freq --> current_pos
        sum_of_samples += current_sample
        if ((current_sample < 0) or (sum_of_samples < 0)):
            validation_position += i
            is_valid = False
            break

    return is_valid, validation_position


def find_zero_crossing(search_from, data_series, ...
   sampling_period, validation_ratio):
    # Finds and returns the first rising zero crossing in the ...
   signal after ...
    # the time "search_from", using zero-crossing.
    # "data_series" is the array with the signal.
    # "sampling_period" is the signal's sampling period.
    # "validation_ratio" is the length of the validation check...
   , given in ...
    # "synchronous_periods". Should be between 0.1 and 0.35.

    current_pos = int(search_from/sampling_period)


    while (data_series[current_pos] > 0): # Fast forward to a ...
   lightly zero-crossing point.
        current_pos += 1
    while (data_series[current_pos] < 0):
        current_pos += 1
    current_pos -= 10

    validated_crossing = False # True if "current_pos" is a ...
   validated zero-crossing, False otherwise.
    validation_length = int((0.02/sampling_period)*...
   validation_ratio)

    while not validated_crossing: # Iterate through the ...
   samples looking for a zero-crossing
        has_crossed_upwards = False # True if "current_pos" ...
   has just risen above zero, reset every cycle.

        while not has_crossed_upwards:
            if (data_series[current_pos] > 0):
                has_crossed_upwards = True
            else:
                current_pos += 1

        validated_crossing, current_pos = is_valid_crossing(...
   current_pos, data_series, validation_length)

        if not validated_crossing:
            current_pos += 1

    crossing_int = current_pos
    crossing_time = crossing_int*sampling_period

    return crossing_time, crossing_int

def return_periods(df, search_from=0, sampling_freq=50000, ...
   synchronous_periods=4,  channel_of_interest=1):
    # synchronous_periods refer to the electrical periods. The...
     mechanical rotation period of the machine is 7.1333 Hz. We...
     need at least 50/7.13 = 7 to capture an entire period of ...
    rotation.
    time_window = synchronous_periods*0.02 # The number of ...
    electrical periods within the window we're looking at
    sampling_period = 1/sampling_freq
    start_time, start_int = find_zero_crossing(search_from, [i...
     for i in df.iloc[:,channel_of_interest] ], sampling_period...
    , 0.1)
    end_time, end_int = find_zero_crossing(start_time + ...
    time_window - 0.005, [i for i in df.iloc[:,...
    channel_of_interest] ], sampling_period, 0.1)

    channels = np.array([[i for i in df.iloc[start_int:end_int...
    ,j]] for j in range(df.shape[1])])

    time_series = np.linspace(start_time, end_time, len(...
    channels[0]) )
    return (start_time, end_time), (start_int, end_int), ...
    channels, time_series


# # Formatting

# In[4]:


def formatting_data(synchronous_periods = 14):
    time_window = synchronous_periods*0.02
    yes_start_end_times, yes_start_end_ints, yes_channels, ...
    yes_time_series = return_periods(has_fault, ...
    channel_of_interest=0, search_from=0,
                                                                  ...
                               sampling_freq=sampling_freq, ...
    synchronous_periods=synchronous_periods)
    yes_adjusted_sampling_freq = len(yes_time_series)/...
    time_window
    return yes_adjusted_sampling_freq, yes_channels


# # Continuous wavelet transform

# In[5]:


def plot_CWT(f, t, Zxx, figsize=(16,10),
             cmap='magma', max_freq=None,
             norm=colors.PowerNorm(gamma=2./8.),
             time_shaving=None, output=False, save_figure=None...
    ):
    # Plots an CWT transform.
    # figsize is the size of the resulting figure.
    # cmap is the colour scheme, by default magma. Reference: ...
    https://matplotlib.org/3.1.1/gallery/color/...
    colormap_reference.html
    # max_freq is the maximum frequency to include in the plot...
    .
    # time_shaving is the amount to shave off of the edges to ...
    avoid edge artifacts, must be between 0 and 0.5.
    # output is if the function is to return an outputm by ...
    default False.
    # save_figure will save the figure to disk if a string is ...
    given. It is saved in the format of the filename.

    fig = plt.figure(figsize=figsize)
    ### Different methods can be chosen for normalization: ...
    PowerNorm; LogNorm; SymLogNorm.
    ### Reference: https://matplotlib.org/tutorials/colors/...
    colormapnorms.html
    spec = plt.pcolormesh(t, f, np.abs(Zxx),
                          norm=norm,
                          cmap=plt.get_cmap(cmap))
    #cbar = plt.colorbar(spec)
    ax = fig.axes[0]
    ax.grid(False)
    plt.axis('off') #Turn off for image processing


    #ax.set_title('CWT Magnitude')
    ax.set_ylabel('Frequency [Hz]')
    ax.set_xlabel('Time [sec]')



    if (max_freq is not None):
        ax.set_ylim(0,max_freq)
    if (time_shaving is not None):
        shaved_time_indices = int(time_shaving*len(t))
        xMinimum = t[shaved_time_indices]
        xMaximum = t[len(t)-shaved_time_indices]
        ax.set_xlim(xMinimum,xMaximum)

    #fig.show
    if (save_figure is not None):
        fig.savefig(save_figure, dpi=None,
                    facecolor='w', edgecolor='w',
                    orientation='portrait',
                    papertype=None, format=None,
                    transparent=False, bbox_inches='tight',
                    pad_inches=0, metadata=None)
        #fig.clf()
        #plt.close()

    if output:
        return f,t,Zxx
    else:
        return

def find_scale(wavelet, frequency, scale,
               sampling_frequency, max_deviation=1):
    # Takes in the continuous wavelet, the sampling frequency ...
    of the signal , ...
    # the prior scale , and a frequency and finds its ...
    corresponding scale .
    # max_deviation is the convergence criteria of the ...
    frequency
    is_found = False # A boolean denoting if the correct scale...
     is found
    top_scale = scale
    bot_scale = scale
    top_frequency = pywt.scale2frequency(wavelet , top_scale)*...
    sampling_frequency
    bot_frequency = pywt.scale2frequency(wavelet , bot_scale)*...
    sampling_frequency
    while not is_found:
        top_frequency = pywt.scale2frequency(wavelet , ...
    top_scale)*sampling_frequency
        bot_frequency = pywt.scale2frequency(wavelet , ...
    bot_scale)*sampling_frequency
        if (((top_frequency - frequency)<max_deviation) and
            ((frequency - bot_frequency)<max_deviation)):
            scale = (top_scale + bot_scale)/2
            is_found = True

        if ((top_frequency < frequency) or
            (bot_frequency > frequency)):
            if (top_frequency < frequency):
                top_scale -= 1
            if (bot_frequency > frequency):
                bot_scale += 1
        else:
            if ((pywt.scale2frequency(wavelet , (top_scale + ...
    bot_scale)/2)*sampling_frequency)>frequency):
                top_scale = (top_scale + bot_scale)/2
            else:
                bot_scale = (top_scale + bot_scale)/2
    return scale

def make_scales(wavelet , frequencies ,
                sampling_frequency , max_deviation=1):
    # This function takes in the continuous wavelet , the ...
    sampling frequency of the signal ...
    # and the frequencies that are of interest and returns a ...
    list of scales .
    # wavelet is a string with the name of the continuous ...
    wavelet .
    # frequencies is a numpy array with the frequencies or ...
    interest .
    # samplingFrequency is the sampling frequency of the ...
    signal to be analysed .
    # max_deviation is the convergence criteria of the ...
    frequency
    scales = []
    scale = 2500
    for frequency in frequencies:
        scale = find_scale(wavelet , frequency , scale
                           , sampling_frequency ,
                                max_deviation)
        scales = np.append(scales,scale)

    return scales


# # Scales for wavelets of interest

# In[6]:


def scales_of_interest(max_freq, min_freq):
    center_frequency = (50/50000)
    bandwidth = center_frequency/2
    wave_list_continuous = ['gaus1']

    yes_adjusted_sampling_freq, yes_channels = formatting_data...
    (synchronous_periods)
    desired_frequencies = np.arange(max_freq,min_freq, -0.5)
    scales_list = []
    for wavelet in wave_list_continuous:
        resultant_scales = make_scales(wavelet, ...
    desired_frequencies, yes_adjusted_sampling_freq, ...
    max_deviation=0.1)
        scales_list.append(resultant_scales)
    return scales_list[0], wave_list_continuous[0]


# # Continous wavelet calculation

# In[ ]:


def CWT_plot(channel, vmin = None, vmax = None, save_figure = ...
    None):

    scales, wavelet = scales_of_interest(max_freq, min_freq)
    yes_adjusted_sampling_freq, yes_channels = formatting_data...
    (synchronous_periods)
    yes_start_end_times, yes_start_end_ints, yes_channels, ...
    yes_time_series = return_periods(has_fault, ...
    channel_of_interest=0, search_from=0,
                                                                   ...
                                sampling_freq=sampling_freq, ...
    synchronous_periods=synchronous_periods)
    coeff,freqs = pywt.cwt(yes_channels[channel],
                              scales, wavelet,
                              sampling_period=(1/...
    yes_adjusted_sampling_freq))
    plot_CWT(freqs, yes_time_series, coeff,
                              save_figure=(save_figure),
                              cmap='jet', norm= colors.Normalize...
    (vmin=vmin, vmax=vmax))
    return save_figure
```

.6.2 Image area segmentation

This code is the result of a practice done at MIT in 2016. It has been modified to include adaptive thresholding[21].

```
#!/usr/bin/env python
# coding: utf-8

# # Image area segmenter

# In[ ]:


def area_segmenter(image, thresh='otsu', radius=10.0, ...
    image_mode='greater',
                   area_bounds=(0,1e7), ecc_bounds=(0, 1)):
     """
     This function segments a given image via thresholding and ...
    returns
     a labeled segmentation mask.

     Parameters
     ----------
     image : 2d-array to be segmented
     thresh : Threshold value. Can be 'int', 'otsu' or '...
    adaptive'.
          If 'otsu' or 'adaptive', the threshold value will be ...
    automatically calculated
     radius : float. Radius for gaussian blur for background ...
    subtraction. Default value is 10.
     image_mode : 'lower' or 'greater'
          If 'lower', objects with
          intensity values *lower* than the provided threshold ...
    will be selected.
          If `greater`, values *greater* than the provided ...
    threshold will be
          selected. Default value is 'greater'.
     area_bounds : tuple of ints.
          Range of areas of acceptable objects. This should be ...
    provided in units
          of square pixels.
     ecc_bounds : tuple of floats
          Range of eccentricity values of acceptable objects. ...
    These values should
          range between 0.0 and 1.0.

     Returns
     -------
     image_labeled : 2d-array, int
          Labeled segmentation mask.
     """
     # Apply a median filter to remove hot pixels.
     med_selem = skimage.morphology.square(3)
     image_filt = skimage.filters.median(image, selem=med_selem...
    )

     # Perform gaussian subtraction
     image_sub = background_subtract(image_filt, radius)

     # Determine the thresholding method.
     if thresh is 'otsu':
          thresh = skimage.filters.threshold_otsu(image_sub)
     elif thresh is 'adaptive':
          thresh = skimage.filters.threshold_local(image_sub, ...
    block_size)

     # Determine the image mode and apply threshold.
     if image_mode is 'lower':
          image_thresh = image_sub < thresh
     elif image_mode is 'greater':
          image_thresh = image_sub > thresh
     else:
          raise ValueError("image mode not recognized. Must be '...
    lower'"
                         + " or 'greater'")

     # Label the objects.
     image_label = skimage.measure.label(image_thresh)

     # Apply the area and eccentricity bounds.
     image_filt = area_ecc_filter(image_label, area_bounds, ...
    ecc_bounds)

     # Remove objects touching the border.
     image_border = skimage.segmentation.clear_border(...
    image_filt, buffer_size=20)

     # Relabel the image.
     image_border = image_border > 0
     image_label = skimage.measure.label(image_border)
     image_label = skimage.morphology.dilation(image_label)
     plt.imshow(image_label)
     return image_label


def background_subtract(image, radius):
     """
     Subtracts a gaussian blurred image from itself smoothing ...
    uneven
     values.

     Parameters
     ----------
     image : 2d-array
          Image to be subtracted
     radius : int or float
          Radius of gaussian blur

     Returns
     -------
     image_sub : 2d-array, float
          Background subtracted image.
     """

     # Apply the gaussian filter.
     image_filt = skimage.filters.gaussian(image, radius)

     # Ensure the original image is a float
     if np.max(image) > 1.0:
          image = skimage.img_as_float(image)

     image_sub = image - image_filt

     return  image_sub


def area_ecc_filter(image, area_bounds, ecc_bounds):

     #Thresholds objects in an image based on their areas and ...
    eccentricities.

     #Parameters
     #image : Labeled image to be filtered
     #area_bounds : Acceptable area range. Provided in units of...
     square pixels
     #ecc_bounds : Acceptable range of eccentricities. Should ...
    be in the range of 0 to 1
     #
     #Returns
     #image_relab : 2d-array. The relabeled and filtered image


     # Extract the region properties of the objects.
     props = skimage.measure.regionprops(image)

     # Extract the areas and labels.
     areas = np.array([prop.area for prop in props])
     eccs = np.array([prop.eccentricity for prop in props])
     labels = np.array([prop.label for prop in props])

     # Make an empty image to add the approved cells.
     image_approved = np.zeros_like(image)

     # Threshold the objects based on area and eccentricity
     for i, _ in enumerate(areas):
          if areas[i] > area_bounds[0] and areas[i] < ...
    area_bounds[1]             and eccs[i] > ecc_bounds[0] and ...
    eccs[i] < ecc_bounds[1]:
             image_approved += image == labels[i]

     # Relabel the image.
     print(np.sum(image_approved))
     image_filt = skimage.measure.label(image_approved > 0)

     return image_filt
```

.6.3 Enhanced connected components difference method

This code applies the segmented faulty cases on top of the healthy case in order to see how significantly the faulty cases differ from the healthy case. It also saves the ECDF plot to the computer. This code is a variation of the code produced in the specialization project[21]

```
#!/usr/bin/env python
# coding: utf-8

# # Enhanced connected components

# In[12]:


pathlist = Path('A:\\Master thesis\\Data sets Motor\\...
    Experiment\\Difference FL').rglob('*.csv')
save_path = 'A:\\Master thesis\\Iterative image folder'
save_figure = 'A:\\Master thesis\\Iterative image folder\\new....
    png'
#----------------------------------------------
sampling_freq = 10000
synchronous_periods = 20 # Normally 20
#----------------------------------------------
max_freq = 80
min_freq = 20
channel = 1 # Corresponds to the columns in the csv datasets
vmin = None
vmax = None #NLH had a max intensity of 8.6
#----------------------------------------------
t = 0.05 #Normally 0.25
block_size = 101
#----------------------------------------------
area_bounds = (100, 10000)
ecc_bounds = (0, 1)
namelist = []
meanlist = []
image_seglist = []
healthy_image_seglist = []
#----------------------------------------------

healthy = str('A:\\Master thesis\\Data sets Motor\\Experiment...
    \\FL\\H.Wfm.csv')
has_fault = pd.read_csv(healthy, sep=';',header='infer')
healthy_plot = CWT_plot(channel = channel, vmin = vmin, vmax =...
     vmax, save_figure=save_figure)
healthy_test = skimage.io.imread(f'A:\\Master thesis\\...
    Iterative image folder\\new.png')
healthy_test = skimage.color.rgb2gray(healthy_test)

```

```
for path in pathlist:
    file = str(path)
    has_fault = pd.read_csv(file, sep=';',header='infer')

    return_plot = CWT_plot(channel = channel, vmin = vmin, ...
   vmax = vmax, save_figure=save_figure)

    test = skimage.io.imread(f'A:\\Master thesis\\Iterative ...
   image folder\\new.png')
    test = skimage.color.rgb2gray(test)

    # Pass all images through our function.
    image_seg = area_segmenter(test, thresh = 'adaptive', ...
   image_mode = 'lower', area_bounds=area_bounds, ecc_bounds=...
   ecc_bounds)
    image_seglist.append(image_seg)

    props = skimage.measure.regionprops(image_seg, ...
   intensity_image=healthy_test)
    mean_int = np.array([prop.moments_central for prop in ...
   props]) # mean_int is a numpy.ndarray

    # Creating list with corresponding filename
    name = file.rstrip('.Wfm.csv').split('\\')[-1]
    print(name)
    namelist.append(name)
    meanlist.append(mean_int)

# Performing the empirical cummulative distribution function ...
    and saving its plot
image_name = file.rstrip('.Wfm.csv').split('\\')[-2]
addon = file.rstrip('.Wfm.csv').split('\\')[-3]
save_path = os.path.join(save_path, addon + ' ' + image_name +...
     ' ' + 'channel' + ' ' + str(channel))
fig = px.ecdf(meanlist, color=namelist)
fig.write_image(f'{save_path}.png')
```

.6.4 Label image regions

This code is provided by scikit-image[43] and it is one of their applications of the regionprops tool

```
#!/usr/bin/env python
# coding: utf-8

# # Label image regions

# In[ ]:


import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

from skimage import data
```

```python
from skimage.filters import threshold_otsu
from skimage.segmentation import clear_border
from skimage.measure import label, regionprops, ...
    regionprops_table
from skimage.morphology import closing, square
from skimage.color import label2rgb

image = image_seglist[5]

# apply threshold
thresh = threshold_otsu(image)
bw = closing(image > thresh, square(3))

# remove artifacts connected to image border
cleared = clear_border(bw)

# label image regions
label_image = label(image)
# to make the background transparent, pass the value of `...
    bg_label`,
# and leave `bg_color` as `None` and `kind` as `overlay`
image_label_overlay = label2rgb(label_image, image=image, ...
    bg_label=0)

fig, ax = plt.subplots(figsize=(10, 6))
ax.imshow(image_label_overlay)

for region in regionprops(label_image):
    # take regions with large enough areas
    if region.area ≥ 100:
        # draw rectangle around segmented coins
        minr, minc, maxr, maxc = region.bbox
        rect = mpatches.Rectangle((minc, minr), maxc - minc, ...
    maxr - minr,
                                  fill=False, edgecolor='red',...
     linewidth=2)
        ax.add_patch(rect)

ax.set_axis_off()
plt.tight_layout()
plt.show()
```

.6.5 CatBoost implementation

This code is a variation of work done by the student Markus Fredrik Johansen in a specialization topic during fall 2021. It has been modified to work for image processing data as well as providing an investigation into region property importance

```python
#!/usr/bin/env python
# coding: utf-8

# # Necessary packages for the CatBoost implementation

# In[9]:
```

```


import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import catboost as cb
from catboost import CatBoostClassifier
from catboost import CatBoostRegressor
import catboost as cb
import numpy as np
import pandas as pd
import seaborn as sns
import shap
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.inspection import permutation_importance


# # Importing the training dataset and the test dataset
#
# The data is imported. For this application, the data is ...
    produced by the difference method and distributed between a...
     training set and a testing set

# In[10]:


training_set = pd.read_csv(r'A:\Master thesis\Data collection\...
    Final collection 2.csv', sep=';')
categories = training_df.columns # Get the names of all the ...
    categories
test_set = pd.read_csv(r'A:\Master thesis\Data collection\...
    Final test 2.csv', sep=';')
test_original = test_set # Will be used later for gathering ...
    results


# # Preview of data

# In[11]:


# Preview dataset
training_set.head()
test_set.head()

# Summary of dataframe
training_set.info()
test_set.info()


# # Sorting numerical and categorical columns
# In the case that there is a mix in data types, it is ...
    important to sort them into numerical and categorical ...
    columns. This is done so it is possible to convert the ...
    columns into an usable format

# In[12]:


# Sorting column into object and numerical
training_set_numerical = training_set.select_dtypes(exclude='...
    object')
test_set_numerical = test_set.select_dtypes(exclude='object')

training_set_categorical = training_set.select_dtypes(include=...
    'object')
test_set_categorical = test_set.select_dtypes(include='object'...
    )

# Names of columns
training_set_numerical_cols = training_set_numerical.columns....
    tolist()
training_set_categorical_cols = training_set_categorical....
    columns.tolist()

test_set_numerical_cols = test_set_numerical.columns.tolist()
test_set_categorical_cols = test_set_categorical.columns....
    tolist()


# # Converting categorical columns into numerical columns and ...
    filling in missing values

# In[13]:


# Converting object to category
training_set_category = training_set_categorical[...
    training_set_categorical_cols].astype('category')
test_set_category = test_set_categorical[...
    test_set_categorical_cols].astype('category')

# Converting category to int and filling missing values
training_set_category[training_set_categorical_cols] = ...
    training_set_category[training_set_categorical_cols].apply(...
    lambda col:pd.Categorical(col).codes)
test_set_category[test_set_categorical_cols] = ...
    test_set_category[test_set_categorical_cols].apply(lambda ...
    col:pd.Categorical(col).codes)

# Filling missing numerical values
numeric_cols = training_set_numerical.columns.values
for col in numeric_cols:
    missing = training_set_numerical[col].isnull()
    num_missing = np.sum(missing)
    if num_missing > 0:  # impute values only for columns that...
     have missing values
        med = training_set_numerical[col].median() #impute ...
    with the median
```

```
        training_set_numerical[col] = training_set_numerical[...
    col].fillna(med)

numeric_cols = test_set_numerical.columns.values
for col in numeric_cols:
    missing = test_set_numerical[col].isnull()
    num_missing = np.sum(missing)
    if num_missing > 0:  # impute values only for columns that...
     have missing values
        med = test_set_numerical[col].median() #impute with ...
    the median
        test_set_numerical[col] = test_set_numerical[col]....
    fillna(med)


# # Combining columns into a complete dataset
# Finishing the preparation by combining the datasets into the...
     training and testing dataset again

# In[14]:


# Combining columns
training_set = pd.concat([training_set_numerical, ...
    training_set_category], axis=1)
test_set = pd.concat([test_set_numerical, test_set_category], ...
    axis=1)


# # Preparing training dataset

# In[37]:


# Feature vector and target variable
X = training_set.drop('target', axis=1)
y = training_set['target']

# Split into training and validation set
test_size = 0.1
random_state = 9987
X_train, X_test, y_train, y_test = train_test_split(X, y, ...
    test_size=test_size, random_state=random_state)


# # Implementing CatBoost
# CatBoost was chosen for this task because of previous ...
    studies showing prominent results within diagnosing ...
    synchronous motors

# In[38]:


# Catboost implementation
iterations = 1000
learning_rate = 0.2
```

```

train_dataset = cb.Pool(X_train, y_train)
test_dataset = cb.Pool(X_test, y_test)

categorical_features_indices = np.where(X.dtypes != np.float)...
    [0]


clf = CatBoostClassifier(iterations=iterations, learning_rate=...
    learning_rate, depth = 8)
clf.fit(X_train, y_train, cat_features = ...
    categorical_features_indices, eval_set=(X_test, y_test), ...
    verbose=False, plot=True)



# # Finding the R-squared value of the predictive function

# In[39]:


r2 = r2_score(y_test, pred)
print("Testing performance")
print('R2: {:.2f}'.format(r2))


# # Investigating feature importance

# In[40]:


sorted_feature_importance = clf.feature_importances_.argsort()
plt.barh(categories[sorted_feature_importance],
        clf.feature_importances_[sorted_feature_importance],
        color='turquoise', height=0.5)
plt.xlabel("CatBoost Feature Importance")
plt.plot(figsize=(20,10))


# # A more concise representation of the investigation

# In[41]:


explainer = shap.TreeExplainer(clf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test, feature_names = ...
    categories[sorted_feature_importance])


# # Exporting data
# And here we see the result of the work. The predictions is ...
    shown here in the plot

# In[43]:

```

```

# Exporting data
probs = clf.predict(test_df)
sub = pd.DataFrame()
sub['id'] = test_orig['id']
sub['target'] = probs[:, 0]
sub['target'].plot.hist(bins=50)

sub.to_csv(r'A:\Master thesis\Data collection\results.csv', ...
    index=False, sep=';')
print('Test size =', test_size, 'Random state =', random_state...
    , 'Iterations =', iterations, 'Learning rate =', ...
    learning_rate)
```

Markus Fredrik Johansen

Image processing and machine learning application for fault diagnosis in synchronous motors

NTNU
Norwegian University of
Science and Technology