FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Sensory Module to Support Industrial Maintenance Using AMRs

**Samuel Carvalho Loureiro**

FINAL VERSION




Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Vítor Hugo Machado Oliveira Pinto

July 31, 2024

© Samuel Carvalho Loureiro, 2024

# Resumo

No contexto da Indústria 4.0, a manutenção de equipamentos industriais está a avançar significativamente devido à crescente acessibilidade de hardware capaz de capturar e processar grandes volumes de dados complexos para analisar o estado e o desempenho dos equipamentos. Este progresso levou ao desenvolvimento da Manutenção Preditiva (PdM), uma estratégia proativa que utiliza análise de dados em tempo real para prever falhas nos equipamentos. Esta abordagem otimiza manutenções programadas, minimiza o tempo de inatividade e reduz os custos através da substituição de componentes apenas quando necessário.

Esta dissertação explora estratégias de PdM para Robôs Móveis Autónomos (AMRs) em ambientes industriais, com foco em abordagens *data-driven* e *Machine Learning*. O objetivo é desenvolver um módulo de sensores, para equipar num AMR, capaz de determinar quando é necessária manutenção no próprio robô e nas máquinas circundantes num cenário industrial, recolhendo dados do chão de fábrica e do AMR.

O documento começa por introduzir aspetos fundamentais, como os tipos de manutenção e técnicas de processamento de sinal, para uma melhor compreensão da revisão da literatura e da própria dissertação.

Foi feita uma revisão que explora as arquiteturas de sistemas de PdM, as abordagens mais comuns a esta técnica, e analisa os estudos e projetos relacionados.

O módulo foi concebido para incluir acelerómetros, um microfone ultrassónico, um sensor de temperatura, um sensor de gás e um sensor de poeiras; para equipar o AMR e para efetuar a recolha de dados. Foi criada uma bancada de teste ligando um motor a rolamentos em diferentes estados. Foram criados três conjuntos de dados identificados diferentes: um de rolamentos saudáveis, um de rolamentos danificados, e outro apenas de ruído.

Foram desenvolvidos vários modelos com diferentes técnicas de pré-processamento com o objetivo de detetar rolamentos danificados. A *Gradient Boosted Decision Tree* juntamente com uma *Short Time Fourier Transform* alcançou um resultado de 0.27 de *loss* e 0.98 de *accuracy*; e a *Random Forest* juntamente com vinte *Mel Frequency Cepstral Coefficients* alcançou 0.24 de *loss* 0.93 de *accuracy*. Foram também testados modelos não supervisionados para o mesmo efeito, e modelos para detetar a presença de rolamentos em ruído. Estes resultados foram obtidos usando um conjunto de dados diversificado à escala laboratorial. Apesar disso, o conjunto de dados foi registado num ambiente controlado que não corresponde a um ambiente industrial, pelo que os resultados podem indicar que o modelo será uma boa aposta para um conjunto de dados futuro mais próximo da realidade, mas os resultados não devem ser tomados como vinculativos.

**Palavras-Chave:** Manutenção Preditiva, Microfone, Acelerómetro, Alta-Frequência, *Machine Learning*, Processamento de Sinal, Rolamentos, MFCC, STFT, Árvores de Decisão, *Autoencoder*.

# Abstract

In the context of Industry 4.0, equipment maintenance is advancing significantly due to the increasing affordability of hardware capable of capturing and processing large volumes of complex data for analyzing equipment health and performance. This progress has led to the development of Predictive Maintenance (PdM), a proactive strategy that utilizes real-time data analysis to predict equipment failures. This approach optimizes maintenance schedules, minimizes downtime, and reduces costs by replacing components only when necessary.

This dissertation explores PdM strategies for Autonomous Mobile Robots (AMRs) in industrial settings, focusing on data-driven approaches and machine learning. It aims to develop a sensor module to equip an AMR capable of determining when maintenance is required on the robot itself and its surrounding machines in an industrial scenario by collecting data from the factory floor and from the AMR.

The document starts by introducing fundamental aspects, such as types of maintenance and signal processing techniques, to a better understanding of the related work review and the dissertation itself.

A comprehensive review was done exploring the system architectures of PdM, the most common approaches to this technique, and analyzing related studies and projects.

The sensor module was designed with accelerometers, an ultrasonic microphone, a temperature sensor, a gas sensor, and a dust sensor to equip the AMR to collect data. A bench test was created by connecting a motor to bearings in different health states. Three sets of identified data were made: one of healthy bearings, one of damaged bearings, and one of noise only.

Several models were developed with different pre-processing techniques to detect damaged bearings. The Gradient Boosted Decision Tree together with a *Short Time Fourier Transform* achieved a result of 0.27 of *loss* and 0.98 of *accuracy*, and the *Random Forest* together with twenty *Mel Frequency Cepstral Coefficients* achieved 0.24 of *loss* 0.93 of *accuracy*. We also tested unsupervised models for the same purpose and models to detect the presence of bearings in noise. These results were obtained using a diverse dataset on a laboratory scale. Despite this, the dataset was recorded in a controlled environment that does not correspond to an industrial environment, so the results may indicate that the model will be a good bet for a future dataset closer to reality. However, the results should not be taken as binding.

**Keywords:** Predictive Maintenance, Microphone, Accelerometer, High-Frequency, Machine Learning, Signal Processing, Bearings, MFCC, STFT, Decision Trees, Autoencoder.

# United Nations' Sustainable Development Goals (SDGs)

From September 25 to 27, 2015, the Heads of State, Government leaders, and High Representatives gathered at the United Nations (UN) Headquarters to celebrate the organization's 70th anniversary. During this event, they adopted the 2030 Agenda for Sustainable Development. This agreement consists of 17 main goals further divided into 169 specific targets. This universal agenda aims to balance the three dimensions of sustainable development: social, economic, and environmental.

This research aligns with several United Nations Sustainable Development Goals described in Table 1. All the information about these goals and their specific targets can be found on the UN website [1].

Table 1: Contributions of the work to the Sustainable Development Goals

| **ODS** | **Target** | **How the target is met** |
|---|---|---|
| 4 | 4.4 | Promotes quality education and skill development by requiring specialized knowledge to implement advanced predictive maintenance technologies. |
| 8 | 8.2 | Increases economic productivity through PdM processes, optimizing maintenance schedules, minimizing downtime, and reducing operational costs. |
| 9 | 9.4 | Promotes the modernization of industrial infrastructure by adopting new technologies such as predictive maintenance and the use of AMRs. |
| 12 | 12.5 | Reduces waste generation by replacing components only when necessary, promoting sustainable consumption and production patterns. |
| 13 | 13.2 | Contributes to climate action by reducing greenhouse gas emissions associated with production processes by optimizing machine performance. |
| 17 | 17.16 | Strengthens global partnerships for sustainable development by fostering collaboration between academic institutions, industries, and technology providers to develop and implement predictive maintenance strategies. |

# Agradecimentos

Quero começar por dar o meu mais sincero obrigado aos que tornaram possível a realização desta Dissertação, orientando-me e prestando ajuda sempre que necessário: ao orientador da minha Dissertação, Investigador Doutor Vítor Hugo Pinto; a todos os membros do PPS18: Investigadores e Estudantes de Doutoramento, Diogo Gomes e Geoffrey Spencer, e Investigadores Ricardo Silva e João Pinheiro.

Acredito que cada vez mais os estudantes entram para a FEUP somente para tirarem um curso com elevada empregabilidade e serem apenas mais um licenciado ou mestre no mercado de trabalho percário português.

Por isto, quero agradecer às inúmeras pessoas que me envolveram na tradição académica do Porto, passando valores e conhecimentos que muitos desprezam.

Agradeço então aos meus amigos Chico, Viana e Morgado.

Aos que aguentaram cinco anos ao meu lado de pura loucura e alucinações coletivas, matendo viva a chama do curso de Electro: Papelão, Xauxixas, SeiSer, Azamat, Mozzarella, Acliclas, Eusébios, Minion Bob, Era e Machachuxas; do fundo do meu coração, obrigado, sem vocês teriam sido só 1827 dias iguais e depressivos neste cojunto de blocos cinzento.

À minha Raquelinha, por estares sempre ao meu lado. Mesmo nos meus momentos mais difíceis, e estando a mais de 1700 km de ti, conseguiste prestar-me apoio e dar-me força para me reerguer.

Ao Roberto, por fazeres todos os trabalhos comigo, e juntos acabarmos este curso com a certeza que demos tudo o que tínhamos de nós para não sermos só mais dois jovens com um papel bonito na mão, mas sim dois jovens que têm muito para dar enquanto Engenheiros.

Aos meus pais, Manuel e Arminda, e avó, Augusta, um muito obrigado por me terem suportado financeiramente e emocionalmente durante estes anos, fazendo sacrifícios para que nunca me faltasse nada, e ter as melhores condições para maximizar o meu conforto longe de casa.

Samuel Carvalho Loureiro

*"It's going to be legen—wait for it, and I hope you're not dairy allergic,*
*because the second half of the word is—dary! Legendary!"*

Adapted from Barney Stinson

# Contents


**1 Introduction** **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 1
- 1.3 Objectives . . . 2
- 1.4 Challenges . . . 2
- 1.5 Document Structure . . . 3

**2 Background and Fundamental Aspects** **4**
- 2.1 Nomenclature Clarification . . . 4
- 2.2 Types of Maintenance and Respective Use Cases . . . 4
  - 2.2.1 Reactive Maintenance/Run to Failure/Corrective Maintenance . . . 5
  - 2.2.2 Preventive Maintenance . . . 5
  - 2.2.3 Predictive Maintenance . . . 6
- 2.3 Signal Processing . . . 7
  - 2.3.1 Discrete Fourier Transform (DFT) . . . 8
  - 2.3.2 Fast Fourier Transform (FFT) . . . 8
  - 2.3.3 Short-Time Fourier Transform (STFT) . . . 9
  - 2.3.4 Wavelet . . . 10
  - 2.3.5 Empirical Mode Decomposition (EMD) . . . 11
  - 2.3.6 Hilbert Transform (HT) . . . 12
  - 2.3.7 Hilbert-Huang Transform (HHT) . . . 13
  - 2.3.8 Mel-Frequency Cepstral Coefficients (MFCCs) . . . 14
- 2.4 Comparison of DSP Techniques . . . 16

**3 Related Work** **18**
- 3.1 Predictive Maintenance 4.0 . . . 18
- 3.2 Approaches to Predictive Maintenance . . . 20
  - 3.2.1 Knowledge-Based Approaches . . . 20
    - 3.2.1.1 Ontology-Based Approaches . . . 21
    - 3.2.1.2 Ruled-Based Approaches . . . 21
    - 3.2.1.3 Model-Based Approaches . . . 21
    - 3.2.1.4 Summary . . . 22
  - 3.2.2 Traditional Machine Learning Approaches . . . 22
    - 3.2.2.1 Artificial Neural Networks . . . 22
    - 3.2.2.2 Decision Tree . . . 23
    - 3.2.2.3 Support Vector Machine . . . 23
    - 3.2.2.4 One-Class Support Vector Machine . . . 23
    - 3.2.2.5 Summary . . . 24

3.2.3 Deep Learning Approaches . . . 24
3.2.3.1 Autoencoders . . . 24
3.2.3.2 Convolutional Neural Network . . . 25
3.2.3.3 Summary . . . 25
3.3 Related PdM Studies and Projects . . . 25
3.3.1 Works related to PdM in AMRs or similar . . . 25
3.3.2 Other PdM Related Works . . . 26
3.3.3 Conclusion . . . 29

**4 Sensor Module Prototype** **31**
4.1 Sensor Module . . . 31
4.1.1 Accelerometer . . . 33
4.1.2 Microphone . . . 35
4.1.3 Temperature Sensor . . . 37
4.1.4 Gas Sensor . . . 38
4.1.5 Dust Sensor . . . 39
4.1.6 Microcontroller and Audio Shield . . . 40
4.2 Rolling Bearing Test Bench . . . 41
4.3 Software Structure . . . 42
4.3.1 Data Collection . . . 42
4.3.2 Dataset . . . 44
4.3.3 Pre-Processing . . . 45
4.3.3.1 Parameters Choice . . . 46
4.3.4 Prediction Models . . . 47
4.3.4.1 Decision Trees . . . 48
4.3.4.2 Autoencoder . . . 52
4.3.4.3 Model's Overview . . . 55

**5 Experimental Results** **57**
5.1 Evaluation Metrics . . . 57
5.1.1 Confusion Matrix . . . 57
5.1.2 Accuracy . . . 58
5.1.3 Precision . . . 58
5.1.4 Recall (Sensitivity) . . . 58
5.1.5 F1-Score . . . 58
5.1.6 ROC-AUC . . . 58
5.2 First Analysis of the Bearings . . . 59
5.3 Supervised Models for Bearing State Classification . . . 60
5.3.1 Random Forest . . . 60
5.3.2 Gradient Boosted Decision Tree . . . 63
5.3.3 Model's Tuning . . . 63
5.3.3.1 GBDT + STFT . . . 63
5.3.3.2 RF + MFCC . . . 66
5.3.4 Model Comparison and Conclusion . . . 68
5.4 Supervised Models for Bearing Detection . . . 68
5.4.1 GBDT + STFT . . . 69
5.4.2 RF + MFCC . . . 70
5.4.3 Model Comparison and Conclusion . . . 72
5.5 Unsupervised Methods . . . 72

5.5.1 Autoencoder . . . . . 72
5.5.1.1 Classification . . . . . 72
5.5.1.2 Detection . . . . . 75
5.5.2 Model Comparison and Conclusion . . . . . 78
5.6 Internal Sensor Testing . . . . . 78
5.7 Conclusion . . . . . 79

**6 Conclusion and Future Work** **80**
6.1 Future Work . . . . . 81

**References** **82**

**A Auxiliary Tables** **88**

# List of Figures


2.1 Main maintenance methods . . . . . 4
2.2 Reactive Maintenance workflow . . . . . 5
2.3 Preventive Maintenance workflow . . . . . 6
2.4 Predictive Maintenance workflow . . . . . 7
2.5 Fast Fourier Transform example . . . . . 9
2.6 Short-Time Fourier Transform example . . . . . 10
2.7 Morlet Wavelet function . . . . . 11
2.8 Wavelet Transformation example . . . . . 11
2.9 Empirical Mode Decomposition example . . . . . 12
2.10 Hilbert Transform example . . . . . 13
2.11 Hilbert-Huang Transform example . . . . . 14
2.12 Mel-Spectrogram and MFCCs examples . . . . . 15

3.1 Current Predictive Maintenance maturity level . . . . . 19
3.2 System architecture for PdM 4.0 . . . . . 20
3.3 Best Sensors to Predict Failures Earlier in Motors . . . . . 30

4.1 *GoMouse-x* inside . . . . . 32
4.2 *GoTugger* with the sensor module already installed . . . . . 32
4.3 Adafruit breakout board MPU-6050 . . . . . 33
4.4 MPU-6050 bellow one of the engines in *GoMouse-x* . . . . . 34
4.5 MPU-6050 above one of the traction wheels in *GoTugger* . . . . . 35
4.6 SPU0410LR5H-QB PCB from Teensy Bat project . . . . . 36
4.7 SPU0410LR5H-QB installed in *GoMouse-x* . . . . . 36
4.8 SPU0410LR5H-QB installed in *GoTugger* . . . . . 36
4.9 Adrafuit Breakout Board TMP235 . . . . . 37
4.10 TMP235 installed in *GoMouse-x* . . . . . 38
4.11 TMP235 installed in *GoTugger* (sensor on the right) . . . . . 38
4.12 MQ-2 Analog Sensor . . . . . 39
4.13 Sharp GP2Y1010AU0F Sensor . . . . . 39
4.14 GP2Y1010AU0F installed in *GoTugger* . . . . . 40
4.15 Teensy 4.1 and its Audio Shield . . . . . 40
4.16 Rolling bearing test bench . . . . . 41
4.17 Two healthy bearings (above) and two damaged bearings (below) . . . . . 42
4.18 Simplified flowchart of data collection code . . . . . 43
4.19 Example of how parameters can be changed in setup file . . . . . 45
4.20 Illustrative example of an Autoencoder . . . . . 53
4.21 Model's overview . . . . . 56

5.1 FFT of Slightly Damaged Bearing *versus* Good Bearing . . . . . 59
5.2 Denoised FFT of Slightly Damaged Bearing *versus* Good Bearing . . . . . 59
5.3 Probability distribution for the "RF + STFT" model . . . . . 61
5.4 Probability distribution for the "RF + MFCC" model . . . . . 61
5.5 Probability distribution for the "RF + FFT" model . . . . . 62
5.6 Probability distribution for the "RF + SP" model . . . . . 62
5.7 Evolution of accuracy and loss during the train of the "GBDT + STFT" model . . 64
5.8 Probability distribution for the "GBDT + STFT" tuned model . . . . . 65
5.9 Confusion matrix of cross-validation evaluation for the "GBDT + STFT" tuned model . . . . . 65
5.10 Probability distribution for the "RF + MFCC" tuned model . . . . . 67
5.11 Confusion matrix of cross-validation evaluation for the "RF + MFCC" tuned model 67
5.12 Accuracy and loss during the train of the "GBDT + STFT" bearing detection model 69
5.13 Probability distribution for the "GBDT + STFT" detection bearing model . . . . 70
5.14 Accuracy and loss during the training for "GBDT + STFT" bearing detection model 70
5.15 Probability distribution for the "RF + MFCC" detection bearing model . . . . . 71
5.16 Accuracy and loss during the training for "RF + MFCC" bearing detection model 71
5.17 Validation metrics along the thresholds . . . . . 73
5.18 Loss evolution during the train of the "AE + STFT" classification model . . . . . 73
5.19 Error distribution for the "AE + MFCC" classification model . . . . . 74
5.20 Loss evolution during the train for the "AE + MFCC" classification model . . . . 74
5.21 Loss evolution during the train for the "AE + MFCC" bearing detection model . . 75
5.22 Error distribution for the "AE + MFCC" bearing detection model . . . . . 76
5.23 Confusion matrix for the "AE + MFCC" bearing detection model . . . . . 76
5.24 Loss evolution during the train for the "AE + STFT + ACCEL" bearing detection model . . . . . 77
5.25 Error distribution for the "AE + STFT + ACCEL" bearing detection model . . . . 77
5.26 Confusion matrix for the "AE + STFT + ACCEL" bearing detection model . . . . 78

# List of Tables


1 Contributions of the work to the Sustainable Development Goals . . . . . iii

2.1 DSP Techniques - Part 1 . . . . . 16
2.2 DSP Techniques - Part 2 . . . . . 17

4.1 Comparison of key features between selected accelerometers . . . . . 34
4.2 Suggested and predefined values for pre-processing parameters . . . . . 47
4.3 Comparison of Tree Depth and Number of Trees in RF and GBDT . . . . . 50
4.4 Default and predefined values for DT models - Part 1 . . . . . 52
4.5 Default and predefined values for DT models - Part 2 . . . . . 52
4.6 Hypertuning values STFT Dense Model . . . . . 55
4.7 Hypertuning values MFCC Dense Model . . . . . 55

5.1 Confusion Matrix Disposition . . . . . 57
5.2 Results of the RF models test metrics . . . . . 60
5.3 Results of the GBDT models test metrics . . . . . 63
5.4 Results of "GBDT + STFT" tuned model test metrics . . . . . 64
5.5 Top 10 STFT feature importance scores . . . . . 66
5.6 Results of "RF + MFCC" tuned model test metrics . . . . . 66
5.7 Top 10 MFCC feature importance scores . . . . . 68
5.8 Results of the “GBDT + STFT” detection model test metrics . . . . . 69
5.9 Results of the “RF + MFCC” detection model test metrics . . . . . 71
5.10 Results of the AE for the bearing classification model . . . . . 72
5.11 Results of the AE for the bearing detection model . . . . . 75

A.1 Summary of Key Terms . . . . . 88
A.2 Comparison of Model-Based, Data-Driven, and Hybrid PdM Approaches . . . . 89
A.3 Teensy 4.1 Features . . . . . 90
A.4 Statistical Properties Definitions . . . . . 91
A.5 Laboratory Desktop Specifications . . . . . 91

# Abreviaturas e Símbolos

| | |
|---|---|
| 1DCNN | 1-Dimensional Convolutional Neural Network |
| ACE | Acoustic Emission |
| AE | Autoencoder |
| AGV | Automated Guided Vehicle |
| AI | Artificial Intelligence |
| AMR | Autonomous Mobile Robot |
| AMS | Amplitude Modulation Spectrogram |
| ANN | Artificial Neural Network |
| ASD | Anomalous Sound Detection |
| AUC | Area Under The Curve |
| BA | Balanced Accuracy |
| BFCC | Bark Frequency Cepstral Coefficient |
| Bi-LSTM | Bidirectional Long Short-Term Memory |
| CAE | Convolutional Autoencoder |
| CNN | Convolutional Neural Network |
| CPS | Cyber-Physical Systems |
| CSV | Comma-Separated Values |
| CUDA | Compute Unified Device Architecture |
| DCT | Discrete Cosine Transform |
| DCNN | Deep Convolutional Neural Network |
| DDM | Degradation Detection Models |
| DFT | Discrete Fourier Transform |
| DL | Deep Learning |
| DM | Data Mining |
| DNN | Deep Neural Network |
| DSP | Digital Signal Processing |
| DT | Decision Tree |
| EMD | Empirical Mode Decomposition |
| ES | Expert System |
| FEUP | Faculty of Engineering of the University of Porto |
| FFT | Fast Fourier Transform |
| FN | False Negative |
| FNR | False Negative Rate |
| FP | False Positive |
| FPR | False Positive Rate |
| GAN | Generative Adversarial Network |
| GBDT | Gradient Boosting Decision Tree |

| | |
|---|---|
| GRU | Gated Recurrent Unit |
| H-FBANK | High-Mel Filterbank |
| HHT | Hilbert-Huang Transform |
| HSA | Hilbert Spectral Analysis |
| HT | Hilbert Transform |
| IDFT | Inverse Discrete Fourier Transform |
| IMF | Intrinsic Mode Function |
| IoS | Internet of Services |
| IoT | Internet of Things |
| ISO | International Organization for Standardization |
| LSTM | Long Short-Term Memory |
| MAPE | Mean Absolute Percentage Error |
| MEMS | Microelectromechanical System |
| MFCC | Mel-Frequency Cepstral Coefficient |
| MFPT | Machinery Failure Prevention Technology |
| ML | Machine Learning |
| MSE | Mean Squared Error |
| NR | Noise Reduction |
| OOB | Out Of Bag |
| Pauc | Partial Area Under Curve |
| PdM | Predictive Maintenance |
| PHM | Predictive Health Monitoring |
| PM | Preventive Maintenance |
| RAE | Recurrent Autoencoder |
| RF | Random Forest |
| RNN | Recurrent Neural Network |
| RM | Reactive Maintenance |
| RME | Root Mean Square |
| RMSE | Root Mean Square Error |
| ROC | Receiver Operating Characteristic |
| ROC-AUC | Receiver Operating Characteristic - Area Under The Curve |
| SNR | Signal-to-Noise Ratio |
| SP | Statistical Properties |
| STFT | Short-Time Fourier Transform |
| SVM | Support Vector Machine |
| SVR | Support Vector Regression |
| TF | TensorFlow |
| TP | True Positive |
| TNR | True Negative Rate |
| TPR | True Positive Rate |
| ViT | Vision Transformer |
| WAV | Waveform Audio File Format |

# Chapter 1

# Introduction

## 1.1 Context

In the ever-evolving world of industrial productivity, where Industry 4.0 demands that every process needs to be optimized to the fullest, the maintenance of the machinery supporting this surge cannot be left behind. Traditional maintenance practices, often characterized by reactive approaches and preventive replacements, have frequently resulted in excessive downtime, escalating repair costs, and inefficient resource utilization. Fortunately, increasing attention is being given to this issue, paving the way for a transformative shift in industrial maintenance: Predictive Maintenance (PdM).

This innovative paradigm, harnessing the power of cutting-edge technology, is revolutionizing the way maintenance is conducted. With the rise of smart manufacturing and the advancement of the Internet of Things (IoT), coupled with the progress in data mining and Artificial Intelligence (AI), PdM enables us to predict when a specific machine component will require repair or replacement. Unlike conventional maintenance systems that either react to anomalies after they occur or engage in preventive replacements, often without confirming their actual necessity, Predictive Maintenance empowers us to proactively address impending issues, preventing downtime and minimizing costly repairs [2].

Recent advancements in Machine Learning (ML) research have significantly enhanced the ability to predict complex processes with remarkable accuracy. These breakthroughs extend into the realm of maintenance, where various methods are being investigated and assessed, consistently demonstrating superior results compared to conventional approaches.

## 1.2 Motivation

The motivation behind this endeavor stems from the transformative potential of predictive maintenance in enhancing industrial operations. The envisaged sensor-equipped AMR serves as a proactive tool, allowing for the prediction of machinery failures before they occur and having the capability to have substantial implications for various industries. An example of an industry that

will benefit from the growth and capabilities of predictive maintenance is the aviation industry, as, according to [3], 42% of delayed flights are caused primarily by airline processes, such as maintenance, causing huge monetary losses, airport's congestion and people's discontentment, potentially leading to the loss of company's credibility.

According to [4], maintenance costs in the United States are estimated to be, on average, US$222 billion. The companies that rely more heavily on PdM and Preventive Maintenance (PM) had 52.7% less unplanned downtime and 78.5% fewer defects than those that rely on Reactive Maintenance (RM). The comparison between the first two shows that there is 18.5% less unplanned downtime and 87.3% fewer defects for those that rely more on predictive than preventive (each type of maintenance is explained in Section 2.2). This suggests significant advantages in shifting from a reactive maintenance approach to adopting preventive and/or predictive maintenance strategies.

## 1.3 Objectives

The proposed work falls under the development of a sensor module to equip an AMR (Autonomous Mobile Robot) capable of determining when maintenance is required on the robot itself and its surrounding machines in an industrial scenario by collecting data from the factory floor and from the AMR. This data should be used to detect differences in the regular operation of equipment, indicating that a failure may be imminent. In this way, it becomes possible to optimize maintenance processes, reducing costs and maximizing production. The project was developed at the Faculty of Engineering of the University of Porto (FEUP) under a Research and Development project - "3D navigation system for mobile robotic equipment".

## 1.4 Challenges

This project faces challenges in filtering sensor signals within the noisy factory shop floor, including differentiating impending machinery failures from sporadic events influenced by external factors. Overcoming these challenges hinges on creating and training a robust model, requiring a substantial volume of sensor-collected data for high predictive accuracy. The model's efficacy relies on significant computing power, emphasizing the need for sophisticated algorithms, extensive datasets, and formidable computing resources to train and effectively refine the predictive model.

Besides that, it is really important to study the cost-benefit of implementing PdM because, although these advanced technologies have promoted enhanced maintenance practices within the factory, the added complexity of smart manufacturing technologies has led to new faults and failures. As stated in research [4], PdM tends to be more financially advantageous for larger firms. This is primarily attributed to their extensive infrastructure requiring maintenance and the greater financial resources available for modernization investments. Additionally, there is a likelihood that the costs associated with implementing this maintenance approach will decrease over time due to technological advancements.

## 1.5 Document Structure

This document is composed of six chapters. Chapter 2 discusses some theoretical concepts that will be pertinent for the rest of the dissertation, such as the types of maintenance and signal processing techniques. Chapter 3 reviews articles containing work related to the objectives, including system architectures of PdM, their approaches, and the review of identical projects. Chapter 4 delves into the sensor module design, the dataset creation, the chosen pre-processing techniques, and the selected prediction models. Chapter 5 presents the results of the chosen ML models with different pre-processing techniques while explaining the metrics used to evaluate them. Chapter 6 closes the document by introducing the conclusions drawn during the project and possible future improvements.

# Chapter 2

# Background and Fundamental Aspects

This chapter explains theoretical aspects and fundamental knowledge essential to better understanding the dissertation.

## 2.1 Nomenclature Clarification

Understanding key terms related to predictive maintenance and system health monitoring is crucial for effective asset management. Table A.1 provides a concise summary of terms such as Condition Monitoring, Remaining Useful Life, Fault Detection, Fault Diagnostics, Fault Prognostics, and Anomaly Detection.

## 2.2 Types of Maintenance and Respective Use Cases

Maintenance refers to the activities and practices undertaken to ensure the proper functioning, reliability, and longevity of equipment, machinery, infrastructure, or any physical asset. There are three main maintenance methods, as illustrated in Figure 2.1. Each method has its application field, as might be seen in the following sections.

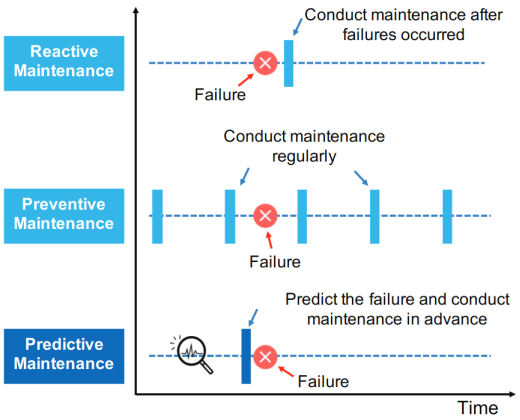



Figure 2.1: Main maintenance methods [2]

### 2.2.1 Reactive Maintenance/Run to Failure/Corrective Maintenance

Reactive Maintenance is the most straightforward maintenance method and consists of only fixing the failure when it occurs: *"If it ain't broke, don't fix it"*, as shown in Figure 2.2. This causes the production to stop to fix the problem, leading to increased downtime, production delays, and higher repair costs. The unpredictability of equipment failures may result in drops in operational efficiency and hinder the long-term reliability of assets. As a result, many industries are seeking to move away from reactive maintenance and adopt more proactive maintenance approaches to optimize equipment performance and minimize unplanned disruptions [2].

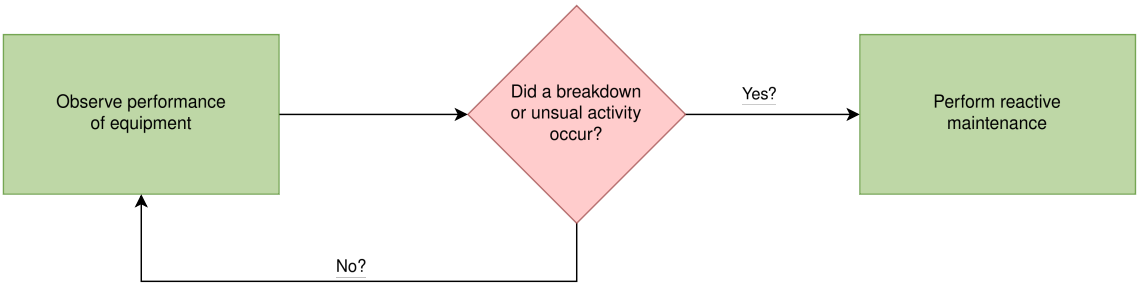



Figure 2.2: Reactive Maintenance workflow

An example of this type of maintenance is changing a light bulb. Sometimes, the luminescence may start blinking as an early indication of impending failure. More frequently, the replacement is only necessitated when it stops working completely. As changing a light bulb is an easy and quick task, the impact of its sudden breakdown is very low.

The same logic applies to the industry, RM remains viable for failures that occur infrequently and are quick to repair, without causing a major impact on production.

### 2.2.2 Preventive Maintenance

Preventive Maintenance constitutes a methodology designed to proactively avert failures by conducting scheduled inspections. This approach is extensively employed because it emphasizes sustaining equipment in optimal operational condition, thereby identifying and rectifying potential issues before they escalate into significant breakdowns or cause operational downtime. Consequently, this strategy proves instrumental in cost savings associated with maintenance activities [2].

It demonstrates a greater efficacy in cost reduction compared to RM; however, it involves the replacement of repairs and machine parts irrespective of the current state of deterioration of the equipment, consequently giving rise to avoidable corrective measure [2].

A daily-life example of PM is vehicle maintenance, like changing the car motor oil. Usually, the motor oil is changed every year, or after a certain number of kilometers, without proactively analyzing its viscosity, following the workflow of this type of maintenance, shown in Figure 2.3. By carrying out this preventive maintenance, the vehicle owner helps to ensure that the engine runs

efficiently, reducing premature wear on internal parts. This contributes to the vehicle's consistent performance and can prevent more serious damage if the oil is neglected for a prolonged period. Determining when the oil will enter the wear-out phase relies on theoretical failure rates rather than actual condition data specific to the equipment. Consequently, this approach often leads to costly and unnecessary maintenance procedures occurring preemptively or, conversely, after potentially catastrophic damage has already been done [2].

PM can be maintained without significant production efficiency losses in situations that occur a significant number of times, are faster to resolve, and do not cause too much disturbance.

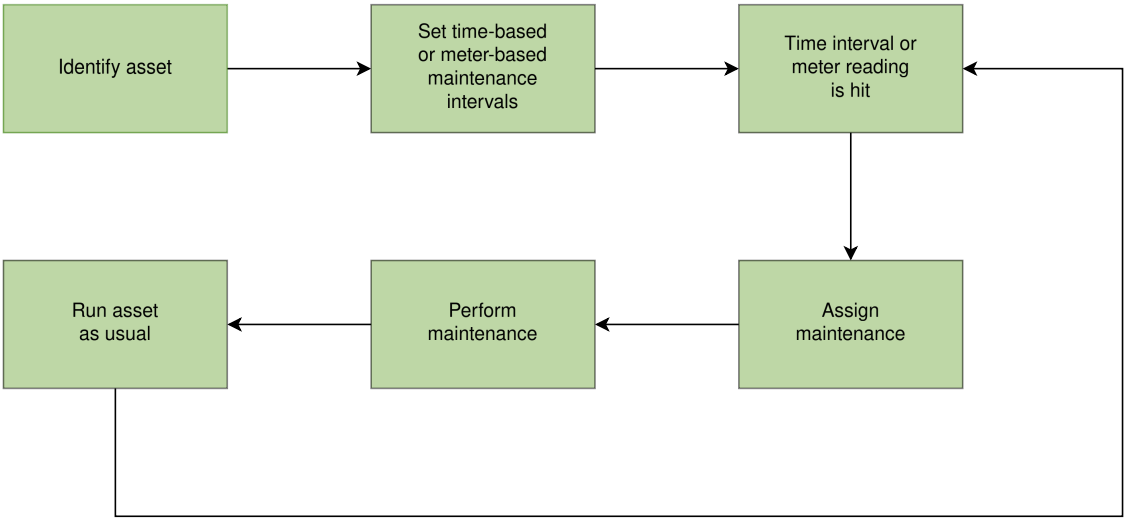



Figure 2.3: Preventive Maintenance workflow

### 2.2.3 Predictive Maintenance

Predictive Maintenance represents a proactive maintenance strategy that uses data analytics, sensor technologies, and machine learning methodologies to forecast and mitigate potential equipment failures before their manifestation. In contrast to conventional maintenance practices reliant on fixed schedules, predictive maintenance relies on real-time data and analytical insights to monitor the operational condition of machinery. By analyzing data patterns and trends, predictive maintenance systems can anticipate impending component failures, enabling the scheduling of maintenance activities with precision. This methodology enhances equipment longevity, minimizes operational downtime, and optimizes maintenance costs by addressing issues precisely when warranted instead of adhering to predetermined maintenance intervals [2]. The workflow of this strategy can be visualized schematically in Figure 2.4.

PdM enables the maintenance frequency to be minimized to the extent necessary for preventing unplanned RM while simultaneously avoiding the costs associated with excessive PM activities [2].

Although PdM has existed for many years, only recently emerging technologies have become seemingly capable and inexpensive enough to make PdM widely accessible [2].

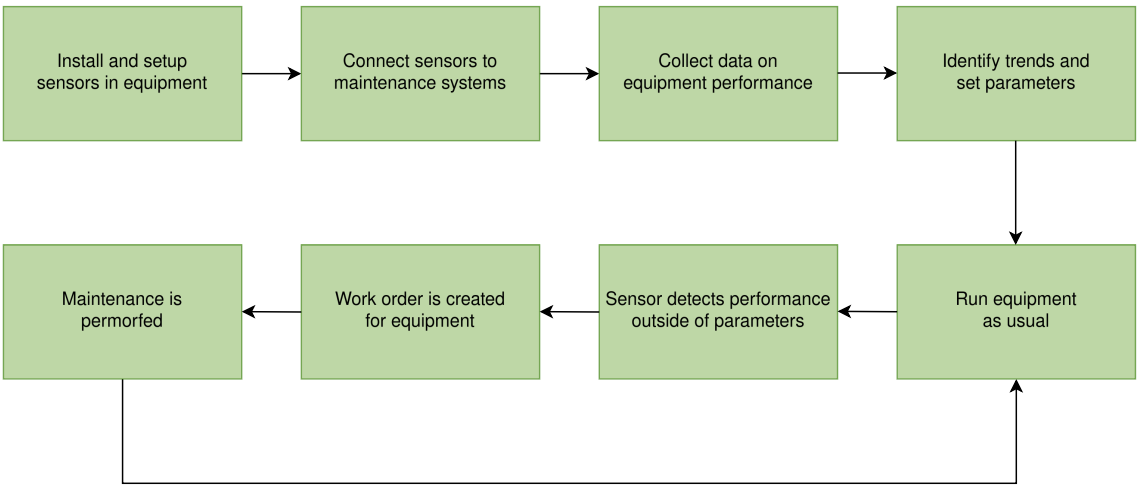



Figure 2.4: Predictive Maintenance workflow

A daily-life example of PdM usage is breast cancer diagnosis based on Machine Learning techniques and Artificial Intelligence. Although the screenings are scheduled, which can be associated with preventive maintenance, the images are analyzed by a program capable of predicting if a cancer will evolve. The program learned with the help of ML methods, such as neural networks and convolutional neural networks (CNN), based on thousands of diagnostic images and their results over time [5].

PdM is the ideal solution for faults that occur infrequently but demand extended downtime when faults occur. The infrequent occurrences of faults do not justify the replacement of these components during the design phase, signifying that their influence has the most significant effect in terms of potential downtime. Failures that happen frequently and demand extended downtime should be excluded in the design phase, but if it is not possible, PdM is the key to avoiding catastrophic damages.

With the continuous advancements in technology, PdM will improve their reliability, and the cost of implementing it will be reduced, which means that this maintenance approach is poised to become the preferred maintenance strategy in most situations that nowadays are still more practicable with RM and PM.

## 2.3 Signal Processing

Signal processing is a field of study involving manipulating, analyzing, and interpreting signals. In this context, a signal refers to any time-varying phenomenon that conveys information, and it can take various forms, such as audio, video, images, sensor readings, or any data that varies with time.

The primary objective of signal processing is to extract, enhance, compress, or interpret information contained within signals, involving filtering unwanted components, amplifying desired features, extracting relevant patterns, or transforming signals from one representation to another.

While recent studies concentrate on advancing ML and Deep Learning (DL) techniques for predictive health monitoring (PHM), a notable gap exists in exploring critical data preprocessing steps integral to training these models. This lacuna is particularly pronounced in the context of degradation detection models (DDMs), despite an abundance of literature focusing on algorithm development using standardized datasets such as those from the Machinery Failure Prevention Technology (MFPT) society [6] or Case Western Reserve University's Bearing dataset [7], these sources often fall short in mirroring real-life scenarios. Real-world monitoring data introduces challenges such as noise contamination, missing data, feature redundancy, and overall poor data quality complications that underscore the pivotal role of signal processing in refining and preparing raw data for subsequent analysis. The tendency to emphasize model complexity adjustments, spurred by rapid advancements in ML and DL algorithms and hardware capacity, has occasionally diverted attention from the indispensable role of meticulous data preprocessing, ultimately impacting the fidelity of the results derived from the predictive models. Recognizing the intrinsic connection between signal processing and the reliability of data-driven approaches is crucial for advancing the efficacy and real-world applicability of predictive health monitoring systems [8].

The following sections explore some of the most popular Digital Signal Processing (DSP) techniques. All the images in those sections are created and plotted using Python.

### 2.3.1 Discrete Fourier Transform (DFT)

The Discrete Fourier Transform is a mathematical technique used in signal processing to transform a sequence of complex numbers into another sequence of complex numbers, enabling the analysis and manipulation of signals in the frequency domain. It is widely used in audio processing, communications, image processing, and almost everything related to signal processing. Computing the DFT directly using the definition requires $O(N^2)$ operations, which might be computationally heavy for large calculations [9]. The Fast Fourier Transform (FFT) was created to solve this.

### 2.3.2 Fast Fourier Transform (FFT)

The Fast Fourier Transform algorithm, shown in Equation 2.1, calculates the DFT of a sequence or its inverse, the Inverse Discrete Fourier Transform (IDFT). Fourier analysis converts a signal from its original domain, often time or space, to a representation in the frequency domain and vice versa. The DFT is obtained by breaking down a sequence of values into components of different frequencies. This operation is helpful in many fields, but computing it directly from the definition is often too slow. The FFT rapidly computes such transformations by breaking down the DFT matrix into sparse (mostly zero) factors. As a result, it reduces the complexity of computing the DFT from $O(N^2)$, which occurs when one applies the definition of DFT, to $O(n \cdot log(n))$, where n is the data size. The difference in speed can be significant, especially for long data sets where n may be in the thousands or millions. In the presence of round-off error, many FFT algorithms

are much more accurate than evaluating the DFT definition directly or indirectly [9]. In the PdM field, it is extremely useful in vibration analysis [10].

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi}{N}kn} \tag{2.1}$$

In 1994, Gilbert Strang described the FFT as "the most important numerical algorithm of our lifetime" [11]. The majority of the other signal processing techniques take advantage, directly or indirectly, of the Fourier transform, using FFT to perform it to accelerate processing times.

In Figure 2.5, an example of an FFT of a trumpet sound is performed using Librosa [1] Python library.

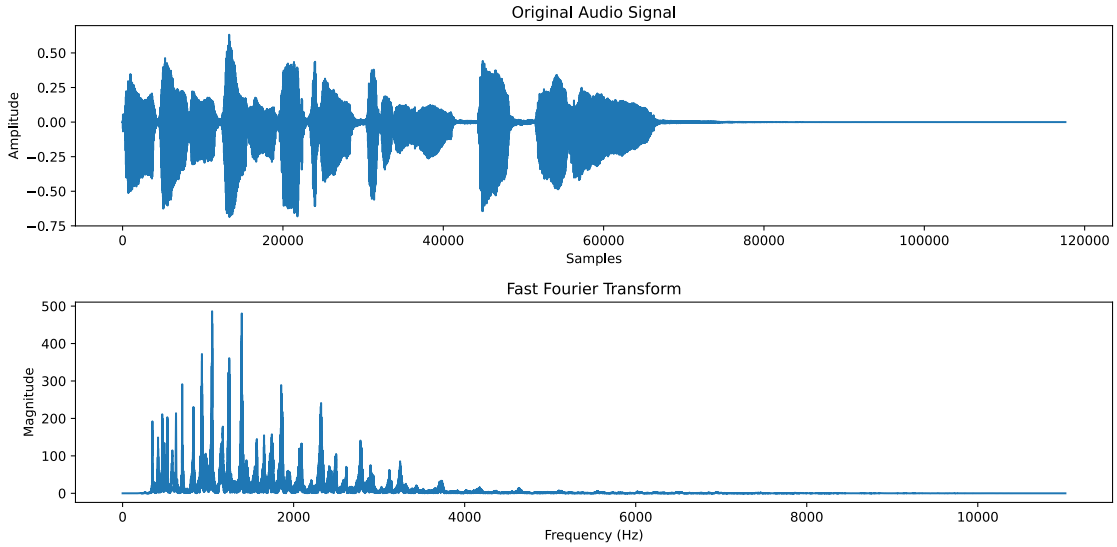



Figure 2.5: Fast Fourier Transform example

### 2.3.3 Short-Time Fourier Transform (STFT)

The Short-Time Fourier Transform is a method closely related to the Fourier transform. It is used to analyze the frequency and phase characteristics of localized segments of a signal as they evolve over time. To calculate the STFT, the longer input signal is divided into shorter, equally sized segments, and the FFT is computed separately for each segment. This helps to reveal the frequency spectrum for each segment. Typically, these changing spectra are plotted over time, resulting in a spectrogram [12]. This way, analyzing how the signal frequencies change over time is possible. Figure 2.6 shows an example of an STFT of the original signal from Figure 2.5.

[1] https://librosa.org/

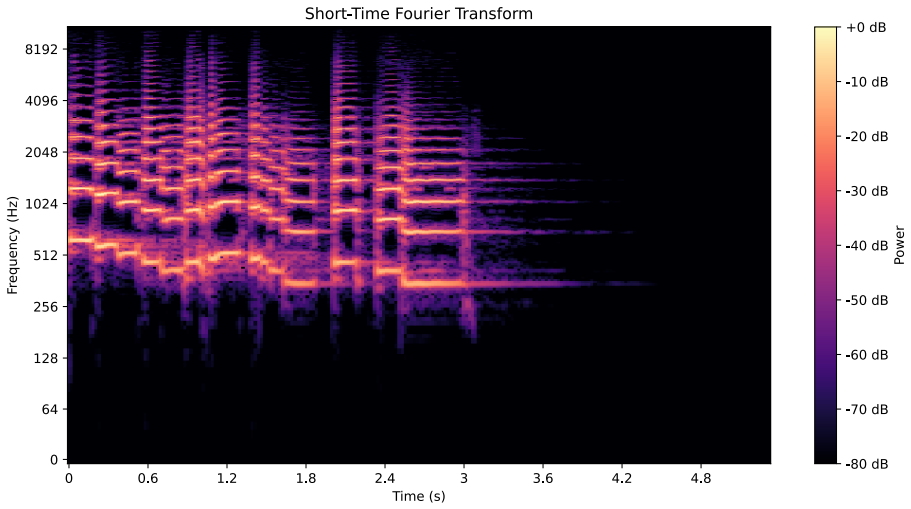



Figure 2.6: Short-Time Fourier Transform example

### 2.3.4 Wavelet

The wavelet transformation is an efficient method for analyzing small waves. It can be used in various applications such as data compression, noise removal, pattern recognition, and fast computation. This method involves altering the time extension by shifting and scaling the signals. Wavelet is used to evaluate a given signal from a mother wavelet and a linear combination of functions of expanded and shifted versions [13].

Wavelets offer advantages over Fourier transforms in reducing computations when analyzing specific frequencies. However, they are not typically more sensitive, and the commonly used Morlet wavelet, displayed in Figure 2.7, is mathematically [14] identical to an STFT. The exception is when searching for signals with a known, non-sinusoidal shape (e.g., heartbeats); in such cases, using matched wavelets can outperform standard STFT/Morlet analyses [15].

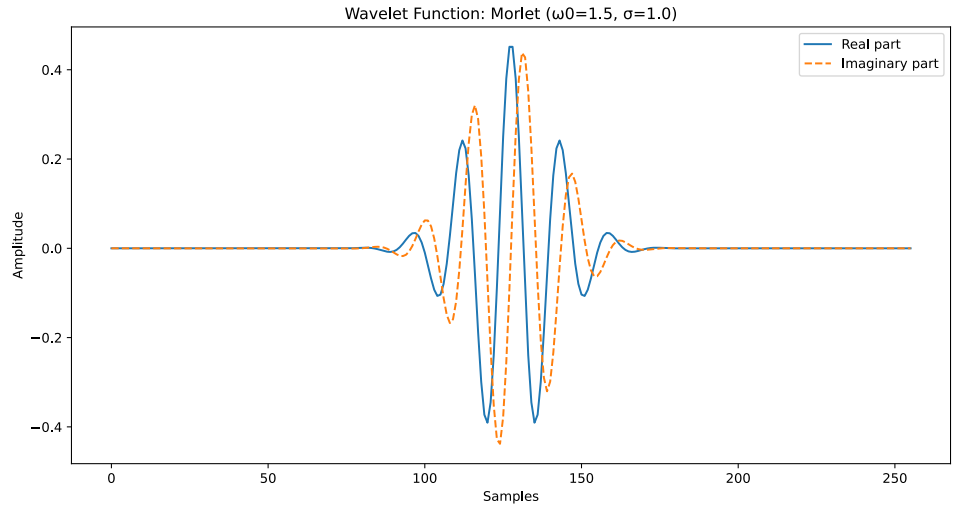



Figure 2.7: Morlet Wavelet function

Figure 2.8 shows the result of a Wavelet transformation using the Morlet Wavelet function applied to the original signal from Figure 2.5.

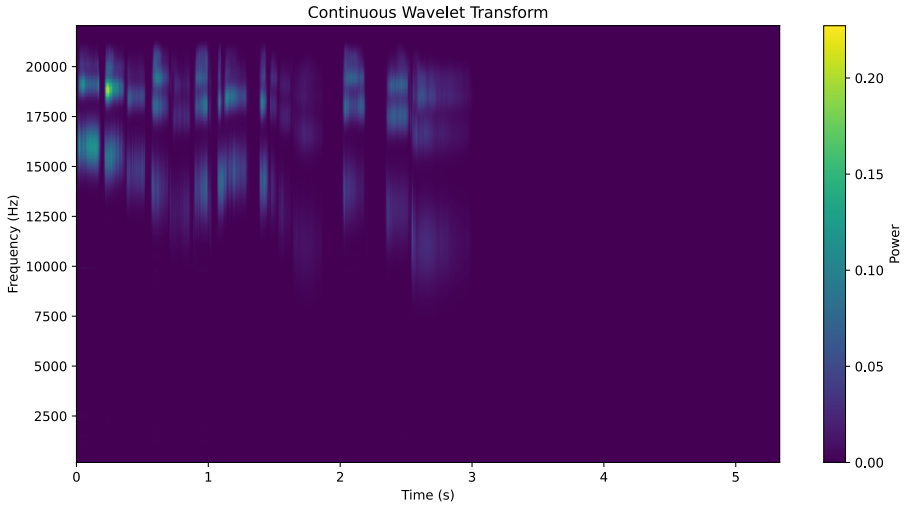



Figure 2.8: Wavelet Transformation example

### 2.3.5 Empirical Mode Decomposition (EMD)

Empirical Mode Decomposition is a signal processing technique that decomposes a signal into its underlying oscillatory components, called Intrinsic Mode Functions (IMFs). This data-driven method does not require prior knowledge of the signal's frequency content. The EMD algorithm decomposes the signal through a process called sifting, which involves identifying local

extrema, fitting an envelope to these extrema, and computing the difference between the signal and its envelope. This process is repeated until a smooth IMF is obtained. The resulting IMFs represent different oscillation scales, with the first IMF capturing the highest frequencies and the last capturing the lowest. The final residual signal represents the trend of the original signal. EMD is useful in signal denoising, feature extraction, and trend analysis applications and is particularly effective for analyzing non-stationary signals with varying frequency content over time [16].

Other methods have been developed to complement EMD, such as Ensemble Empirical Mode Decomposition, Local Mean Decomposition, and Empirical Wavelet Transform.

A basic example [2] of an EMD transformation can be observed in figure 3.3.

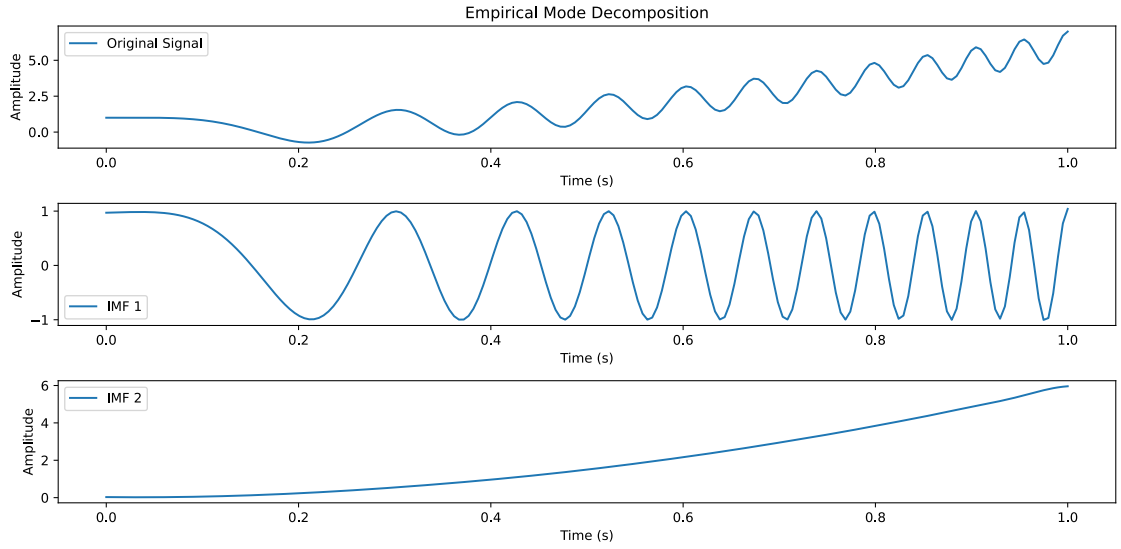



Figure 2.9: Empirical Mode Decomposition example

### 2.3.6 Hilbert Transform (HT)

The Hilbert Transform is a mathematical operation utilized in signal processing to obtain the analytic representation of a real-valued signal. This analytic signal is a complex-valued function where the real part represents the original signal, and the imaginary part is the Hilbert Transform of the signal. It is a valuable tool in signal processing for obtaining the analytic signal from a real-valued signal, allowing for the extraction of instantaneous amplitude, phase, and frequency [17]. The Hilbert Transform is extensively applied in areas such as modulation, demodulation, and biomedical signal analysis, and it plays a pivotal role in comprehending the underlying characteristics of signals.

An example of the HT of the input signal from Figure 2.9 can be observed in Figure 2.10.

[2] The original signal from Figure 2.5 was not used for this case, as in the examples above, because it was computationally heavy to perform its EMD

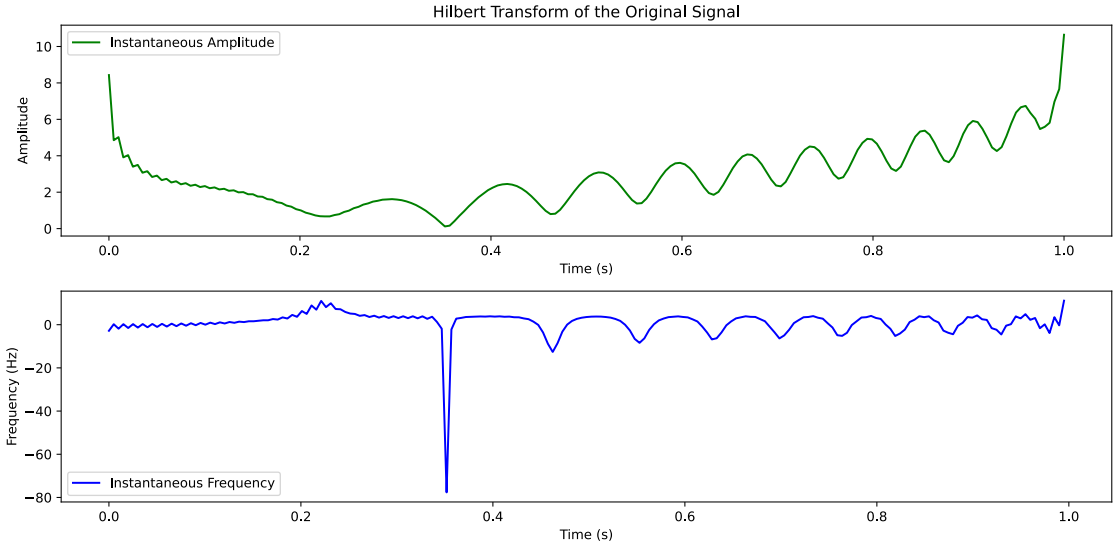



Figure 2.10: Hilbert Transform example

### 2.3.7 Hilbert-Huang Transform (HHT)

The Hilbert-Huang Transform is a signal processing technique developed by Norden E. Huang in the late 1990s for analyzing non-stationary and nonlinear time-series data. It consists of two main components [16]:

- Empirical Mode Decomposition: This method decomposes a signal into a finite set of IMFs with well-defined frequency ranges through a sifting and averaging process, as mentioned in 2.3.5;
- Hilbert Spectral Analysis (HSA): The Hilbert transform is applied to each IMF to obtain the instantaneous frequency and amplitude of the signal.

HHT is particularly effective for analyzing signals with non-linear and non-stationary characteristics, such as those with rapidly varying frequency components or abrupt changes in amplitude. It has been widely used in fields such as biological signal analysis, financial time series analysis, and climate data analysis [18].

Figure 2.11 shows the result of the HHT applied to the input signal of Figure 2.9.

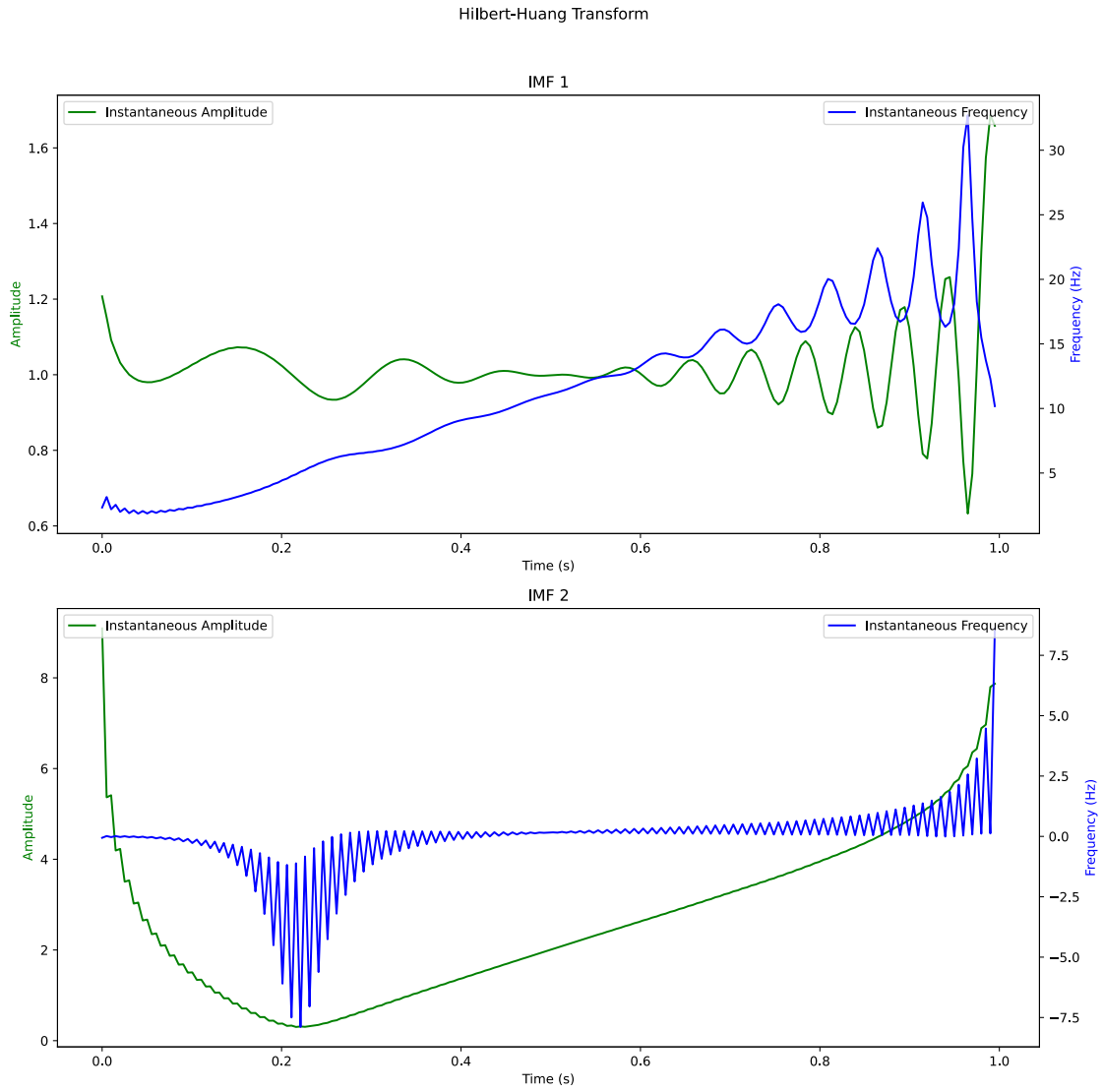



Figure 2.11: Hilbert-Huang Transform example

### 2.3.8 Mel-Frequency Cepstral Coefficients (MFCCs)

Mel-Frequency Cepstral Coefficients represent the spectral characteristics of a sound signal, especially in speech recognition systems. The process starts with taking the FFT of the sound signal and then mapping the resulting spectrum onto the mel scale to compute MFCCs. The logarithm of the powers at each mel frequency is taken, and the discrete cosine transform (DCT) is applied. A subset of resulting coefficients, typically between 10 and 20, is kept to capture the most significant features of the sound signal [19, 20]. The advantages of using MFCCs include their ability to efficiently encode useful phonetic features, while the disadvantages include encoding other acoustic features like speaker and environmental conditions [21]. Other variations of MFCCs

have been proposed, such as Bark-frequency cepstral coefficients (BFCCs) and autocorrelation MFCCs, which are noise-robust extensions of MFCCs [22].

Figure 2.12 shows a Mel-Spectrogram and forty MFCCs of the original sound from Figure 2.5.

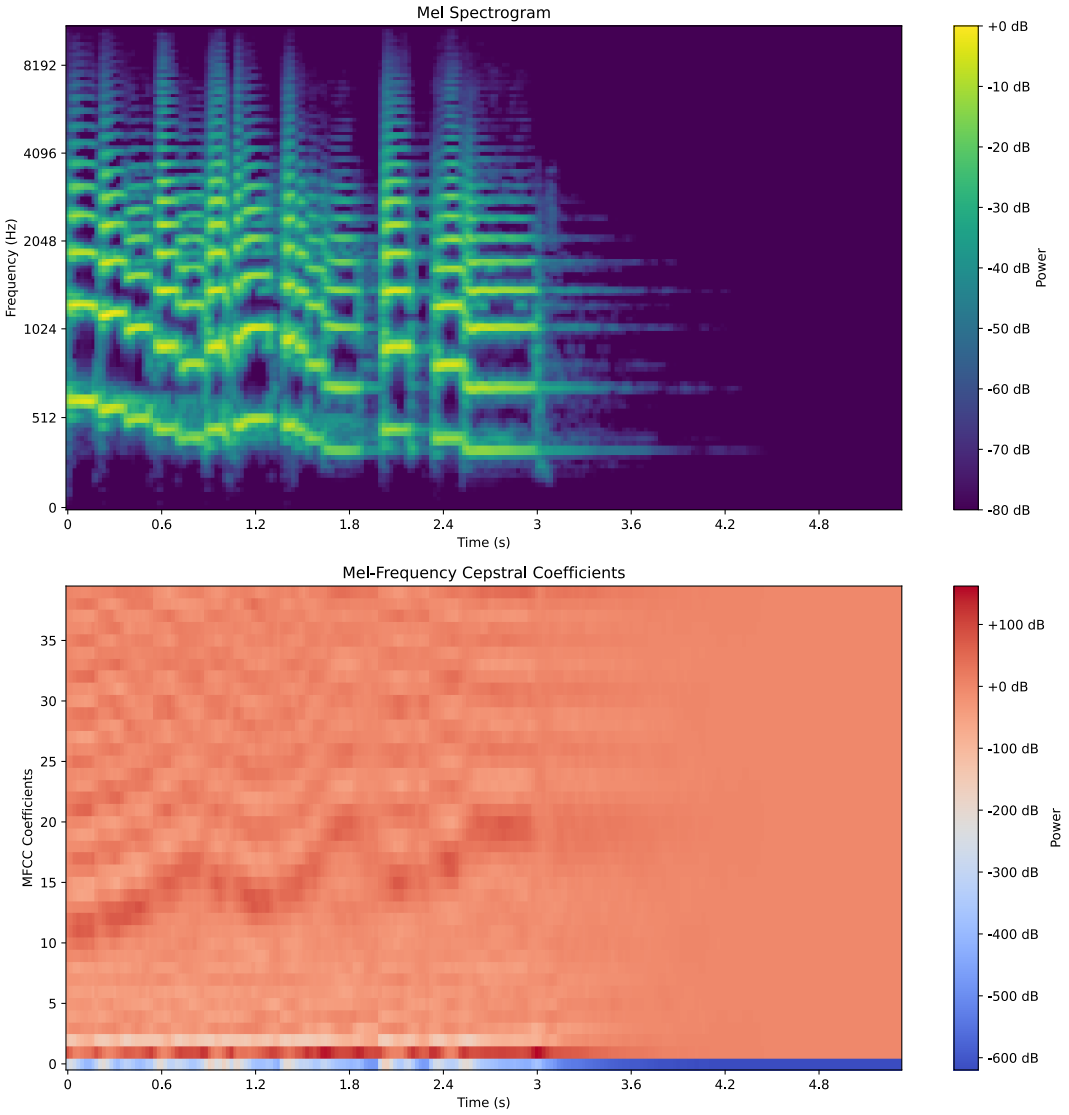



Figure 2.12: Mel-Spectrogram and MFCCs examples

## 2.4 Comparison of DSP Techniques

Tables 2.1 and 2.2 summarize and compare the analyzed DSP techniques based on the aforementioned references.

Table 2.1: DSP Techniques - Part 1

| Technique | Description | Advantages | Disadvantages | Applications |
|---|---|---|---|---|
| **DFT** | Transforms a sequence of time domain to the frequency domain. | Accurate frequency analysis; applicable to any signal length. | Computationally intensive for large datasets. | Signal processing, spectral analysis. |
| **FFT** | An optimized algorithm to compute the DFT efficiently. | Much faster than DFT, especially for large datasets; reduces computational complexity from $O(n^2)$ to $O(n \log n)$. | Requires signal length to be a power of two (though there are variations that handle other lengths). | Real-time signal processing, audio, and image processing. |
| **STFT** | Analyzes small sections of a signal over time by applying the FFT to overlapping windowed segments. | Provides time-frequency representation of the signal. | Trade-off between time and frequency resolution. | Speech processing, music analysis, and time-frequency analysis. |
| **Wavelet** | Decomposes a signal into components with both time and frequency information using wavelets. | Good time-frequency localization; handles non-stationary signals well. | Choice of wavelet function can significantly affect results. | Image compression, denoising, feature extraction. |

Table 2.2: DSP Techniques - Part 2

| Technique | Description | Advantages | Disadvantages | Applications |
|---|---|---|---|---|
| **EMD** | Decomposes a signal into intrinsic mode functions through iterative sifting. | Adaptive, data-driven method; effective for non-linear and non-stationary signals. | Computationally intensive; mode mixing issues. | Biomedical signal processing, climate data analysis. |
| **HT** | Provides the analytic signal of a real-valued signal, highlighting its amplitude and phase. | Useful for instantaneous frequency and phase analysis. | Only applicable to real-valued signals; edge effects. | Speech processing, modulation techniques, envelope detection. |
| **HHT** | Combines EMD and Hilbert Transform for analyzing non-stationary and non-linear signals. | Provides precise time-frequency representation; adaptive to signal characteristics. | Computationally intensive; requires effective implementation of EMD. | Biomedical signal analysis, mechanical fault detection, and geophysical data analysis. |
| **MFCC** | Represents the short-term power spectrum of a sound, using a mel scale of frequency. | Mimics human auditory system; useful for speech and audio processing. | May discard some information not relevant to human perception. | Speech recognition, audio compression, music genre classification. |

# Chapter 3

# Related Work

This chapter provides a comprehensive review and analysis of existing literature, research, and relevant studies to the dissertation's subject matter. The primary focus of the research encompasses the analysis of the PdM architecture "PdM 4.0" and prevalent approaches to it, followed by exploring some related studies.

## 3.1 Predictive Maintenance 4.0

Industry 4.0 represents a paradigm shift in industrial processes and products, driven by intelligent information processing, communication systems, and future-oriented techniques. The goal is to make machines and factories smarter, reducing equipment downtime through advanced technologies like networking, connected devices, data analytics, machine learning, and artificial intelligence [2].

PdM 4.0 employs sophisticated and real-time analysis of collected data to detect potential machine failures at an earlier stage and also supports technicians during maintenance interventions by providing intelligent decision support. In [23], maintenance strategies are categorized into four levels applicable in modern industries:

- **Level 1 – Visual inspections:** This involves periodic physical inspections and maintenance strategies rely solely on the expertise of inspectors;
- **Level 2 – Instrument inspections:** This level conducts periodic inspections and maintenance strategies are based on a combination of inspector expertise and instrument read-outs;
- **Level 3 – Real-time condition monitoring:** Continuous real-time monitoring of assets is conducted, and alerts are activated in response to predefined rules or critical thresholds;
- **Level 4 – PdM 4.0:** Continuous real-time monitoring of assets occurs at this level, and alerts are delivered based on predictive techniques, such as regression analysis.

According to [23], in a survey they conducted, they found that more than half of respondents are still below maturity level 4, and only 11% reached level 4, as shown in figure 3.1.

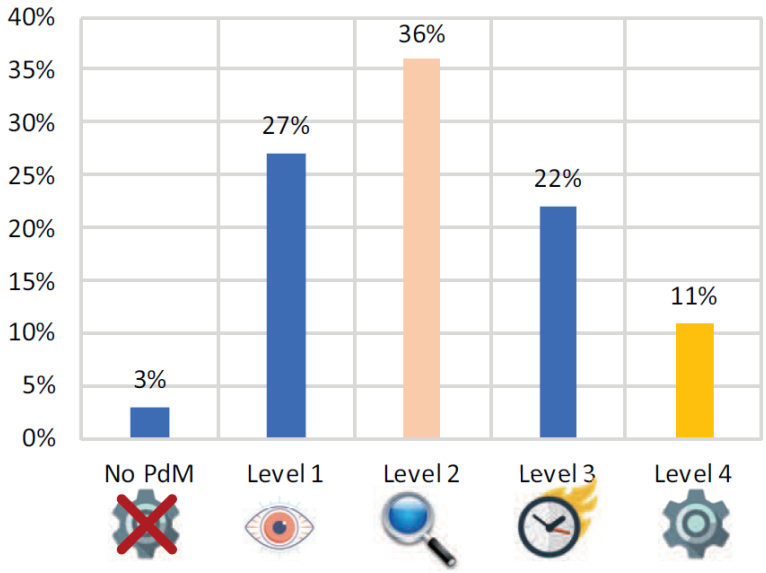



Figure 3.1: Current Predictive Maintenance maturity level [23]

PdM 4.0 architecture is divided into 5 layers [24]:

- **Cyber-Physical Systems (CPS):** Refers to a novel generation of systems that combine integrated computational and physical capabilities to interact with humans through computation, communication, and control. CPS can seamlessly translate the physical world into the virtual realm;
- **IoT:** Facilitates widespread access to entities on the Internet through a diverse array of sensing, location tracking, and monitoring devices. This enables "objects" to interact with one another and collaborate with their "smart" components to accomplish the shared objectives of PdM;
- **Big Data:** It involves how to store and process the large and complex data collected from sensors, e.g., filtering, data integration, data transformation, data validation, and feature extraction;
- **Data Mining (DM):** Aims to analyze and unveil patterns, rules, and knowledge from big data collected from various sources. Subsequently, optimal decisions can be timely and appropriately made based on the outcomes of the analysis;
- **Internet of Services (IoS):** Empowers service vendors to provide maintenance functions as services through the Internet. IoS encompasses business models, service infrastructure, the services themselves, and participants.

This architecture can be observed schematically in Figure 3.2.

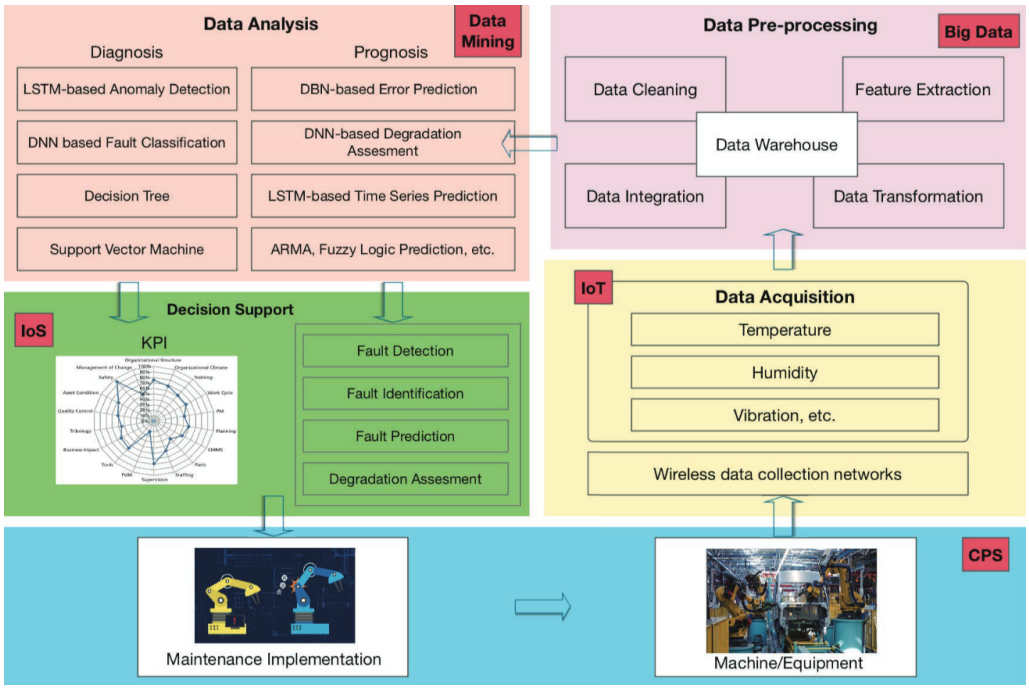



Figure 3.2: System architecture for PdM 4.0 [2]

## 3.2 Approaches to Predictive Maintenance

PdM approaches for industrial environments can be categorized into three main types: model-based, data-driven, and hybrid. Model-based approaches require a detailed understanding of machinery physics, making them challenging for real-world applications. On the other hand, data-driven approaches use empirical models and historical data to predict machinery health without relying on explicit knowledge of the system's physical behavior. Hybrid approaches aim to combine both strengths but may be challenging to implement. Data-driven approaches, particularly those based on Machine Learning or Deep Learning, are gaining attention as they offer reliable prognostic solutions without requiring extensive prior knowledge [25]. Table A.2 compares these approaches in different features.

These approaches can be roughly classified into Knowledge-Based, Machine Learning, or Deep Learning-based solutions, which are explored in the next sections.

### 3.2.1 Knowledge-Based Approaches

Many conventional PdM techniques for fault diagnosis and prognosis rely on pre-existing expert knowledge and deductive reasoning processes, such as expert systems and model-based reasoning [2]. This section provides a brief overview of knowledge-based approaches employed in various systems. These approaches are typically categorized into three main types: ontology-based, rule-based, and model-based approaches.

#### 3.2.1.1 Ontology-Based Approaches

Ontology-based approaches in the context of fault diagnosis and prognosis involve using structured frameworks to represent and organize knowledge about a particular domain. An ontology is a formal and explicit specification of concepts, entities, and their interrelationships within a specific domain. In PdM, ontology-based approaches leverage these structured representations to understand better system components, relationships, and potential faults [2].

By providing a structured and standardized representation of knowledge, this approach facilitates early fault detection by analyzing interrelated components. Integrating data from diverse sources into the ontology ensures a holistic understanding of the system's health, reducing the risk of overlooking interconnected issues [26].

Ran et al. [2] refers to various studies where ontology is employed to establish a knowledge base for diverse machinery systems and devices and can be combined with various existing reasoning algorithms (e.g., rule-based reasoning, Kalman filter, and Artificial Neural Networks (ANN)) to achieve fault diagnosis and prognosis. A practical example where this can be helpful in the dissertation project is, i.e., if a sensor outside of the AMR measures a high environment temperature, it is likely that the temperature inside the AMR also increases, not necessarily being a fault. Applying the ontology approach, relations between sensors can be established, avoiding false positive faults.

#### 3.2.1.2 Ruled-Based Approaches

Rule-based approaches in fault diagnosis and PdM involve evaluating online monitored data according to predetermined rules established by expert knowledge, constituting an Expert System (ES). These systems store domain knowledge, represented as rules in the form of "IF condition, THEN consequence," with conditions typically based on specific facts and consequences influencing the external environment. Building ESs encompasses knowledge acquisition, representation, and model verification, and they are commonly employed for monitoring, interpreting, and diagnosing systems, as well as planning PdM activities.

Berredjem et al. [27] introduce a fuzzy expert system in their paper that effectively localizes bearing faults and tackles distributed faults. The system's intelligent fuzzy rules are automatically extracted from numerical data using an enhanced range overlaps method, bolstering its accuracy and robustness.

To extract diagnosis knowledge or mine rules from raw data, DM techniques can be employed in an ES, e.g., an ANN [28] or a Decision Tree (DT) [29].

#### 3.2.1.3 Model-Based Approaches

Usually, model-based approaches establish a direct connection between mathematical models and the underlying physical processes that, either directly or indirectly, influence the well-being of relevant systems or components. In fault diagnosis and prognosis, model-based strategies employ residuals as features, scrutinizing the conformity between measured outcomes and the anticipated

behavior outlined by the analytical model. Leveraging this explicit model, residual generation methods derive residuals, such as Kalman filters, system identification, and parity relations. These residuals serve as indicators, revealing the presence or likelihood of faults in the targeted systems or components [2].

These models are employed to solve the maintenance problems in diverse systems and components, e.g., gearboxes, bearings, rotors, lithium-ion batteries, or aircraft redundant systems [30]. Furthermore, to improve the effectiveness of model-based approaches, several studies incorporate machine learning techniques [31].

#### 3.2.1.4 Summary

In general, ontology offers a promising approach for integrating, sharing, and reusing contextual knowledge in a system, but to achieve predictive maintenance, it must be combined with other reasoning methods. Rule-based approaches are useful when there is extensive experience but limited data for precise modeling, especially in complex industrial environments where linear approximations may cause errors. However, rule-based systems struggle to adapt to new faults and acquire comprehensive knowledge. While model-based approaches work well when accurate mathematical models can be derived, creating such models for complex systems is often impractical due to changing dynamics and operating conditions.

It is worth considering that the ontology method can be useful in the dissertation project if we choose appropriate sensors for the AMR. This method can help us research the relationships between sensors, as discussed in Section 3.2.1.1. Ruled-Based and Model-Based approaches may be challenging to integrate due to the AMR components' unknown behavior and the unpredictable industrial environment.

### 3.2.2 Traditional Machine Learning Approaches

In data-driven prognostics, ML methods have been applied, with noticeable success [25], to extract useful knowledge and make appropriate decisions from big data. This section reviews and briefly summarizes a subset of well-developed ML algorithms.

#### 3.2.2.1 Artificial Neural Networks

An Artificial Neural Network is a computational model that simulates the structure of biological neural networks. It is made up of interconnected nodes that are organized into layers, which include input, hidden, and output layers. The network processes information by transmitting signals between nodes, and each connection has a weight adjusted during learning. Unlike traditional systems, ANNs do not rely on rules defined by human experts; instead, they acquire fundamental principles by learning from a set of symbolic situations provided in examples given in the training data. Due to their ability to learn solely from examples, ANNs are ideal for handling random, fuzzy, and nonlinear data, and variations of this technology are widely used in Condition Monitoring [32].

In the article [33], an ANN model is developed for diagnosing faults in rolling element bearings using time-domain features, with experimental results showing that the accuracy can reach between 62.5% and 100% with different numbers of input signals, features, and levels of preprocessing data.

#### 3.2.2.2 Decision Tree

Decision Trees are a fundamental supervised learning algorithm used for classification and regression tasks. They work by recursively splitting the dataset into subsets based on the value of input features, creating a tree-like model of decisions. Each internal node of the tree represents a decision based on a feature, each branch represents the outcome of the decision, and each leaf node represents a final class label or continuous value. Decision trees are simple to understand and interpret, making them popular for many practical applications. However, they can be prone to overfitting, especially with noisy data or when the tree is too deep [34]. Nevertheless, some variants of this algorithm have appeared to solve these disadvantages.

Another method is the Random Forest, which is not a decision tree but an ensemble of multiple decision trees used in supervised learning for classification and regression tasks. It constructs multiple decision trees through bootstrap sampling and random feature selection. Each tree is trained independently and contributes to the final prediction by voting (for classification) or averaging (for regression). This method enhances robustness against overfitting, improves accuracy, and handles large datasets with high dimensionality. Additionally, it offers insights into the importance of features. Widely used and versatile, Random Forests are effective in various applications, particularly when dealing with complex datasets containing noise or irrelevant features [35].

#### 3.2.2.3 Support Vector Machine

A Support Vector Machine (SVM) is a supervised ML technique, particularly valuable when the underlying process of a real-world system is unknown or when obtaining the mathematical relation is prohibitively expensive due to the influence of numerous interdependent factors. In classification tasks, where the samples are assumed to have a positive and a negative class, an SVM training algorithm constructs a model for categorizing new examples into one of the two classes. This makes SVM a non-probabilistic binary linear classifier.

Due to the high classification accuracy, even for nonlinear problems, SVM has proven successful in various applications, including face detection, verification, and the recognition of machine fault diagnosis [25]. In [36], a Support Vector Regression (SVR) model is built to represent battery aging dynamics. Experimental results show that the SVR-based model outperforms an ANN-based model with considerably less estimation error.

#### 3.2.2.4 One-Class Support Vector Machine

One-class SVMs distinguish themselves from traditional SVMs as they aim to establish a boundary between normal and anomalous data. The fundamental process involves evaluating each

test instance to determine if it falls within the learned region. If a test instance is within this region, it is classified as normal; otherwise, it is deemed anomalous [37]. This specialization makes One-Class SVMs primarily suitable for Anomaly or Outlier Detection. In article [38], a One-Class SVM was constructed using 300 samples from a simulated system, considering factors like noise, instrument failure, formatting errors, and non-representative samples (e.g., misreadings). The results demonstrated the algorithm's robustness to outliers while maintaining sensitivity to discern whether a test instance was anomalous or normal, indicating its appropriateness for Fault Detection.

#### 3.2.2.5 Summary

ANNs, DTs, and SSVMs are among the most commonly used techniques in PdM literature, each with distinct strengths and use cases. ANNs are highly effective for handling nonlinear, random, and fuzzy data, making them suitable for complex systems where explicit rules are hard to define. DTs offer a clear and interpretable model for classification tasks. However, they can overfit without careful tuning, a problem addressed by methods like Random Forests, which enhance accuracy and are ideal for datasets with noise or irrelevant features. SVMs, particularly adept at binary classification and fault detection, work well in scenarios where the relationship between variables is unclear. One-class SVMs further specialize in anomaly detection, distinguishing normal from faulty conditions. While these are the most common methods discussed in PdM, many other approaches, such as k-nearest Neighbors and Bayesian networks, contribute to the field.

### 3.2.3 Deep Learning Approaches

In recent years, DL has demonstrated remarkable efficiency in feature learning, fault classification, and fault prediction by applying multilayer nonlinear transformations. Several DL models such as Autoencoder (AE), Convolutional Neural Network (CNN), Deep Belief Network (DBN), and others have found extensive application in the domain of PdM. In this section, some of these DL models are considered.

#### 3.2.3.1 Autoencoders

Autoencoders are an unsupervised type of neural network designed for feature extraction and data compression that aim to replicate their input in the output. This involves learning a compressed input representation in a lower-dimensional space, reducing dimensionality through encoding layers, capturing essential patterns in the least informative layer, and reconstructing the output through decoding layers. Trained solely on normal data during training, AEs excel at recognizing anomalies by detecting higher reconstruction errors for dissimilar patterns. By setting a threshold on the error, anomalies can be flagged [2, 32]. In [39], a hybrid model integrating AE with STFT effectively diagnosed faults in a hydro generator rotor using vibration signal, outperforming other comparison models, i.e., a modified AE that uses Softmax Regression and a CNN, in terms of accuracy, showcasing its effectiveness in fault diagnosis.

#### 3.2.3.2 Convolutional Neural Network

A Convolutional Neural Network is a supervised deep learning model designed for visual data tasks, such as image recognition. CNNs consist of multiple layers, including convolutional layers that apply filters to detect patterns, pooling layers that reduce dimensionality, and fully connected layers for classification. CNNs excel at feature extraction through convolution and automatically learning hierarchical representations, making them particularly effective for tasks like image classification and object detection.

In the field of PdM, CNN has shown dramatic capability in extracting useful and robust features from monitoring data. For one-dimensional monitoring signals, [40] build an end-to-end 1D-CNN that reflects the raw vibration signals to fault types. The result shows that the proposed model can achieve about 99% accuracy through hyperparameter tuning.

#### 3.2.3.3 Summary

AEs and CNN are only two of a growing list of DL methods. However, these are the two most popular approaches in the field of PdM, and many others are just variations of CNNs, e.g., Recurrent Neural Networks (RNNs), Deep Belief Networks, and Generative Adversarial Networks (GANs).

## 3.3 Related PdM Studies and Projects

Despite the extensive research and projects in predictive maintenance, no studies have been found where a sensor module is installed in an AMR or similar to collect environmental data for predicting failures in the evolving machinery.

Nevertheless, few studies have been carried out on the predictive maintenance of autonomous robots. These studies, as well as other potentially useful related ones on data collection and processing techniques and subsequent failure prediction, will be analyzed in the following sections.

### 3.3.1 Works related to PdM in AMRs or similar

Bohdan Shubyn et al. [41] used the energy measurements, motors status (turned on/off), odometry, brakes status (turned on/off), payload lifting, natural navigation and path control data, safety signals, weight strains, and driving mode to make an unsupervised anomaly detection model by predicting and comparing the real and predicted overall power consumption of an Automated Guided Vehicle (AGV).

They had four AGVs in different plants to collect data in different environments and make a robust dataset. They used modified RNN architectures based on Long Short-Term Memory (LSTM) cells to make the model, obtaining a Mean Absolute Percentage Error (MAPE) of 1.32%. To improve the performance, they did another experiment using Federated Learning (FL), which consists of dividing the dataset of each AGV into an arbitrary number of parts, making local models

of each, and then averaging them to make a global model, and based on the first global model conduct local training by using the second part of the local data to then make another global model by averaging, and do this the number of arbitrary parts of the dataset. By doing this, they gain an advantage over RNN by close to 5%.

This was the only article found to date on the predictive maintenance of AMRs or similar.

### 3.3.2 Other PdM Related Works

In [42], an unsupervised deep learning model was developed to predict failures in the bearings of rotating machinery using accelerometer data. The dataset consisted of vibration data collected by four sensors on a rotor test bench over approximately 140 seconds. The data included readings for a normal state (Normal) and three abnormal states (Type1, Type2, and Type3). Before analysis, the data was preprocessed into two-dimensional spectrograms using the STFT to visualize the frequency characteristics over time. Feature extraction was carried out using a CNN, specifically the AlexNet architecture [43] known for its effectiveness in image processing tasks. This step was crucial for extracting relevant information from the STFT images. Anomaly detection was accomplished using a Long Short-Term Memory Autoencoder, where the LSTM layers captured the temporal dynamics inherent in the time series data, and the autoencoder framework was utilized to detect anomalies by assessing reconstruction errors.

The model was trained and validated using a dataset encompassing normal and abnormal operating conditions to ensure its generalizability to real-world scenarios. The model's performance was evaluated using precision and recall, standard metrics in classification tasks. An optimal threshold was established based on the reconstruction error rates to provide a reliable criterion for anomaly detection. This threshold was strategically set where the difference between precision and recall was minimized [42].

Experimental validation was carried out using a separate test dataset to confirm the model's effectiveness in accurately distinguishing between normal and abnormal conditions, reaching 1.0 precision and recall. It is worth noting that this performance was reached in a specific simulation scenario with a test bench; the model was not tested in a real industrial scenario [42].

Kreuzer et al. [44] stated that most approaches for classifying bearing faults rely on large amounts of labeled training data for both healthy and damaged bearings; however, such data is often unbalanced because damaged bearing data is hard to obtain due to safety concerns, while healthy bearing data is plentiful. They also add that labeling naturally occurring damages is challenging as it is difficult to pinpoint the exact time when the first signs of a fault appear. Therefore, methods that do not depend heavily on damaged bearing data or require minimal amounts are particularly effective for bearing fault classification. This classification problem is known as outlier or anomaly detection.

Considering this, Kreuzer et al.introduced MFCCs and features extracted from the Amplitude Modulation Spectrogram (AMS) to improve bearing fault detection classification performance.

The study involved experiments on a commuter railway induction engine with a challenging scenario, including varying rotational speeds and torque levels, closely resembling real-world operating conditions. The evaluation metrics for classification performance focused on a binary classification problem distinguishing between healthy and damaged bearings. The True Positive Rate (TPR), True Negative Rate (TNR), Balanced Accuracy (BA), False Negative Rate (FNR), and False Positive Rate (FPR) were considered for assessing classifier performance. A One-Class SVM was trained using different feature sets, including time-domain features, spectral descriptors, amplitudes at characteristic fault frequencies, and scalar features. The results showed that MFCCs and AMS features are highly effective for detecting bearing faults, outperforming traditional time and frequency-domain features [44].

Wang et al. [45] introduced the High-Mel Filterbank (H-Fbank) feature, specifically designed for acoustic emission (ACE) signals, as opposed to conventional Mel-Frequency Cepstral Coefficients, which are more suited for speech signals. This innovative feature extraction method enhances the ability to capture the unique spectral characteristics of ACE signals. The paper proposes a 1DCNN-ViT ensemble model that combines the strengths of one-dimensional convolutional neural networks (1DCNN) and Vision Transformers (ViT). This hybrid model leverages the local feature extraction capabilities of 1DCNN and the dynamic attention mechanism of ViT, providing a comprehensive approach to identifying engine faults based on ACE signals. The used dataset comprises ACE signals collected from different locations and simulated damage sources in an engine fault test rig with an ACE sensor with a resonant frequency of 150 kHz. The ACE signals are preprocessed using the H-Fbank feature extraction method to prepare them for input into the proposed 1DCNN-ViT model. This model is designed to extract features from the preprocessed ACE signals effectively. By combining the feature extraction capabilities of 1DCNN with the global feature extraction and attention mechanisms of ViT, the model can accurately identify engine faults based on ACE signals. The model's robustness was tested by inputting a test set containing ACE signals with different signal-to-noise ratios (SNRs). The results showed that as the signal-to-noise ratio increased, the classification accuracy of the proposed model improved faster than other models. This indicates the robustness of the proposed model in handling ACE signals with varying noise levels. The ensemble model also exhibited better performance and robustness than other methods and models used in the experiments, being able to identify engine faults based on ACE signals accurately, and its high classification accuracy and robustness highlight its effectiveness in fault diagnosis without needing prior knowledge or diagnostic experts.

The proposal in the article [46] introduces a novel methodology for Anomalous Sound Detection (ASD) in predictive maintenance using deep learning techniques. Their approach stands out by incorporating machine identifiers, Mel-Spectogram analysis, and a unique loss function to enhance the training process of an autoencoder model. By conditioning the autoencoder on the relationships between Mel-Spectogram features and machine identifiers, the model can more effectively compute anomaly scores related to input sequences. The study evaluates the methodology on the DCASE dataset [1], demonstrating its superior performance in terms of Area Under the

[1] https://dcase-repo.github.io/dcase_datalist/

Curve (AUC) and partial AUC (pAUC) metrics for various types of machinery compared to other state-of-the-art approaches.

The proposal in the study [47] revolves around creating a cost-effective solution for small- and medium-sized factories to predict the operational status of manufacturing machines using add-on triaxial accelerometers. Unlike traditional Industry 4.0 implementations that require expensive sensor-embedded machines and high-performance computers, this approach aims to make the technology accessible to smaller factories by utilizing inexpensive sensors and a simplified computational framework. The key innovation lies in developing a dimensionality reduction method with low computational overhead to extract essential information from collected data and a neural network for automatic analysis.

This approach differs from previous works by focusing on smaller factories' specific needs and constraints, where conventional machines lack built-in sensors for data collection. The proposed system balances affordability and efficiency, while existing methods may use basic machine learning algorithms or complex neural networks with high costs. By conducting experiments at an actual spring factory, the study validates the effectiveness of their algorithm in predicting machine status accurately and promptly.

The study results showcase the proposed approach's potential in enabling small- and medium-sized factories to leverage Industry 4.0 technologies for improved operational efficiency and quality control. By demonstrating higher prediction accuracy than other models, the research highlights the feasibility of implementing low-cost solutions for machine status prediction, paving the way for broader adoption of advanced manufacturing technologies in smaller industrial settings.

Suawa et al. [48] focused on developing noise-robust machine learning models for predictive maintenance applications in industrial settings using sensor fusion. Firstly, data collection was conducted through experiments on a milling machine and a motor with loads mounted, capturing sound, vibration, and ultrasound data to monitor equipment conditions. Two distinct datasets were created: one comprising sound and vibration data for the milling machine and the other consisting of ultrasound data for the motor.

Next, the investigation utilized ensemble and deep learning methods to build the monitoring models. The noisy convolutional deep learning method was a significant component, integrating a noisy Gaussian layer with decreasing noise levels to enhance model performance in terms of accuracy and noise robustness. The deep convolutional neural network (DCNN) algorithm was applied to raw signals for milling machine health detection, while the deep neural network (DNN) and ensemble algorithms were used for training specific features extracted from the raw data to monitor the motor.

Furthermore, the models were evaluated for their noise resistance using white Gaussian noise added to the test data at various levels. The results indicated that the proposed models maintained high-performance levels, with an accuracy of at least 95%, even when subjected to noise levels up to 15 dB added to the signals. This robustness to noise was consistent across different models, showcasing the effectiveness of the noise-robust machine learning approach in industrial equipment monitoring tasks.

The methodology employed in the study by Jankauskas et al. [49] for early PdM in wind turbines involves several key steps. Firstly, the researchers selected input features for a temperature prediction model that accurately represented the external factors influencing the gearbox-bearing temperature in healthy turbine components. Recurrent Neural Networks, known for their ability to analyze time-series data, were chosen for the analysis; specifically, three different models – LSTM, Gated Recurrent Unit (GRU), and Bidirectional Long Short-Term Memory (bi-LSTM) – were utilized to predict gearbox bearing temperature. The models were tested with different structures, including recurrent layers with 128 cells and two recurrent layers with 128 + 100 cells. Training and testing were conducted multiple times using data from various turbines, and anomaly detection thresholds were proposed by comparing predicted temperatures with actual measurements. The study's conclusions highlighted the effectiveness of recurrent neural networks, particularly the bi-LSTM model, in predicting temperature anomalies indicative of impending gearbox failures in wind turbines. This model outperformed other regression models, demonstrating strong predictive precision with a Root Mean Square Error (RMSE) of 1.61, a standard deviation of 1.10, and minimal errors in both minimum (0.0021) and maximum (7.17) values.

### 3.3.3 Conclusion

Some selected studies about PdM in industrial scenarios were reviewed in the above Section 3.3.2. Other hundreds of articles about PdM can be found online in renowned libraries. To date [2], considering only 2024 articles, 1468 studies with the words "predictive maintenance" can be found on the ScienceDirect [3] website.

In this highly explored topic, it is important to consider the most common points of failure. Rolling element bearings are crucial parts of various industrial machinery, especially mechanical power transmission systems. They play a significant role in the reliability of rotating machines, and even minor bearing failures can lead to unforeseen issues with industrial equipment, causing extended periods of inactivity, resulting in financial loss, or, in the worst-case scenario, human injuries [50]. According to [51], defective bearings cause between 40% and 70% of electromagnetic drive system failures. Recent advances in bearing fault detection have extensively utilized vibration signals due to their rich information content about various bearing fault states.

The most common method for collecting these vibrations is using accelerometers, but this can also be done using microphones, eliminating the need for direct contact with the surface. Additionally, temperature [52] or acoustic signals [53] can be analyzed.

Based on [54], ultrasonic microphones and vibration sensors (such as accelerometers) have the best performance predicting failures earlier in motors, as might be seen in figure 3.3, taken from this same article. According to [52], the main fault in motors is also related to their bearings.

[2] 01/07/2024

[3] https://www.sciencedirect.com/search?qs=%22predictive%20maintenance%22

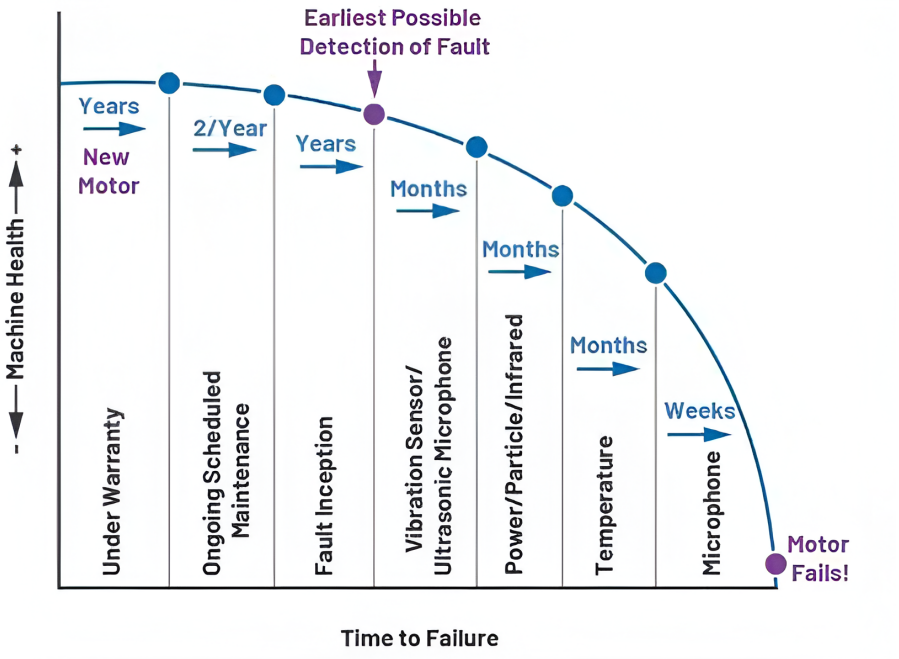



Figure 3.3: Best sensors to predict failures earlier in motors [54]

# Chapter 4

# Sensor Module Prototype

This chapter outlines the approach and methods used to achieve the proposed goal. Considering the information from chapter 3, an assessment will be conducted to determine the most suitable sensors for the proposed objective, considering both a low-cost and effective sensor module.

Additionally, the chapter will detail the software design for data collection from the sensors, the process of dataset creation, and the models to be tested.

## 4.1 Sensor Module

As mentioned in section 1.3, a fundamental goal of this dissertation is to develop a sensor module to equip an AMR capable of determining when maintenance is required on the robot itself and its surrounding machines in an industrial scenario.

Concerning the AMR, the critical points of possible failure detected were:

- Bearings of the wheels;
- Engines responsible for moving the wheels;
- A short, or overload, in the electrical system.

During the majority of the dissertation, the available AMR was a *GoMouse-x* [1] from *Flowbotic*. An image of its inside is visible in Figure 4.1. Considering the critical points, it has two freewheels, two traction wheels, one engine above each traction wheel, and an electrical and electronic system. The onboard computer already manages the battery lifetime and status.

[1] https://www.flowbotic.eu/#products

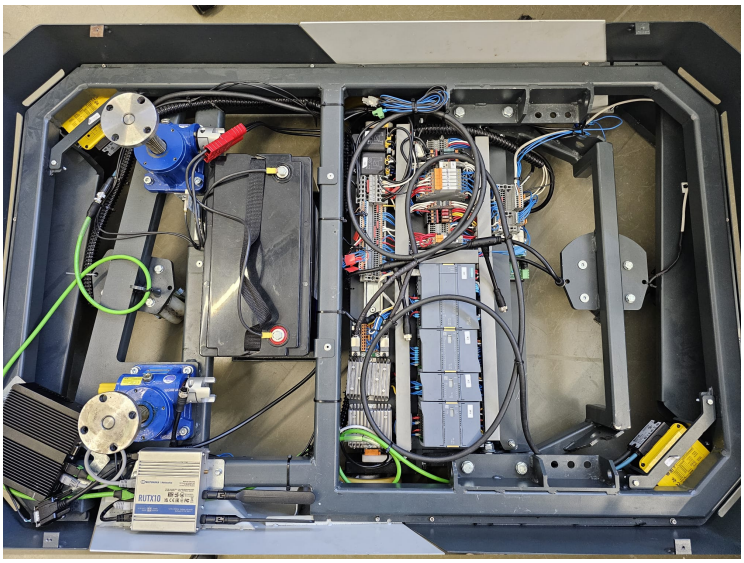


Figure 4.1: *GoMouse-x* inside

However, in the last two months of dissertation time, due to unforeseen circumstances, the AMR had to be exchanged for a *GoTugger*, shown in Figure 4.2. In terms of critical points of failure, what changes is that this robot has 2 rear wheels with traction and a front support wheel.

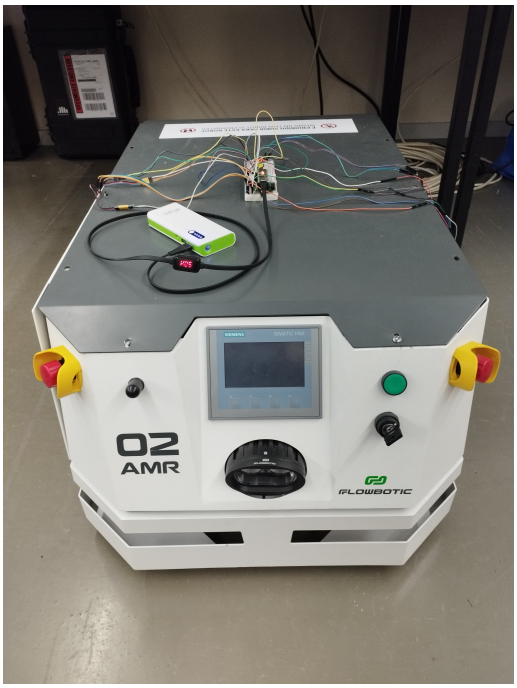



Figure 4.2: *GoTugger* with the sensor module already installed

Regarding the surrounding environment, the most common failures to analyze are related to bearings, as stated in Section 3.3.3.

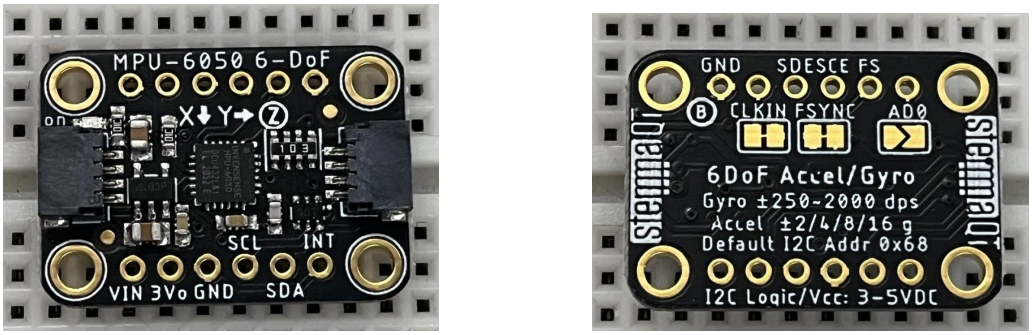
The following sections explore and justify the chosen sensors for the module.

### 4.1.1 Accelerometer

Most issues about condition-based monitoring and PdM in bearings involve using test benches and accelerometers to measure vibrations. Although they can effectively predict various failures, they must be attached to the machine and positioned close to the rotating shaft to detect the vibration.

In our case, they are not particularly helpful for collecting data from the surrounding environment, as the module is in the robot; however, they might be valuable for reading engine and wheel vibrations from the AMR and helping ML models distinguish vibrations from vehicle movement and the environment.

Table 4.1 compares key features of three popular choices of accelerometers. As one of the objectives of the sensor module is to be cost-effective, *ADXL355* [55] was discarded despite having better features overall. *MPU-6050* [56] was selected instead of *MMA84541* [57] because it can be found in an Adrafuit Breakout Board with an embedded gyroscope and temperature sensor (Figure 4.3), which compensates for the price difference, it has official and community libraries in several programming languages, and we already had it in stock.




(a) MPU-6050 - front

(b) MPU-6050 - back

Figure 4.3: Adafruit breakout board MPU-6050

It combines a 3-axis accelerometer, as well as a gyroscope, and a temperature sensor, simplifying system design, reducing component count, and minimizing potential calibration issues between separate sensors. While a gyroscope by itself would not be the primary sensor for detecting wheel and motor faults, fusing its data with an accelerometer can be beneficial as gyroscopes measure rotational orientation, helping distinguish between faults and issues related to overall system orientation, imbalance, and or floor irregularities. To configure, calibrate, and read all the outputs, we used the "Adafuit MPU6050" for Arduino [58].

Table 4.1: Comparison of key features between selected accelerometers

| **Accelerometer** | **MMA8451** | **ADXL355** | **MPU-6050** |
|---|---|---|---|
| **Num. Axis** | 3 | 3 | 3 |
| **Sensitivity** ($LSB/g$) | 4 096 | 256 000 | 16 384 |
| **Noise Density** ($\mu g/\sqrt{Hz}$) | 99 | 22,5 | 400 |
| **Bandwidth** ($Hz$) | 400 | 2 100 | 1 000 |
| **Price**(€)[2] | 7,39 | 43,46 | 12,04 |

With this in mind, in the *GoMouse-x*, an Adafruit *MPU-6050* was placed below each engine, as shown in Figure 4.4, with the same orientation to make the data consistent.

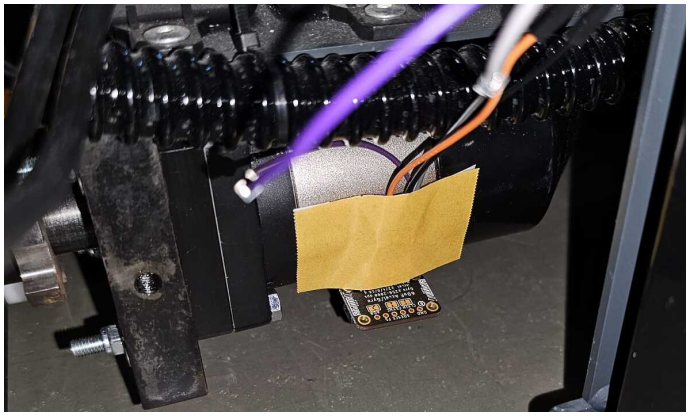


Figure 4.4: MPU-6050 bellow one of the engines in *GoMouse-x*

On the *GoTugger*, the engine wheels were not easily accessible, so we decided to install the accelerometers just above each traction wheel, as evidenced in Figure 4.5.

[2]Price on 21/06/2024 at Mouser Electronics

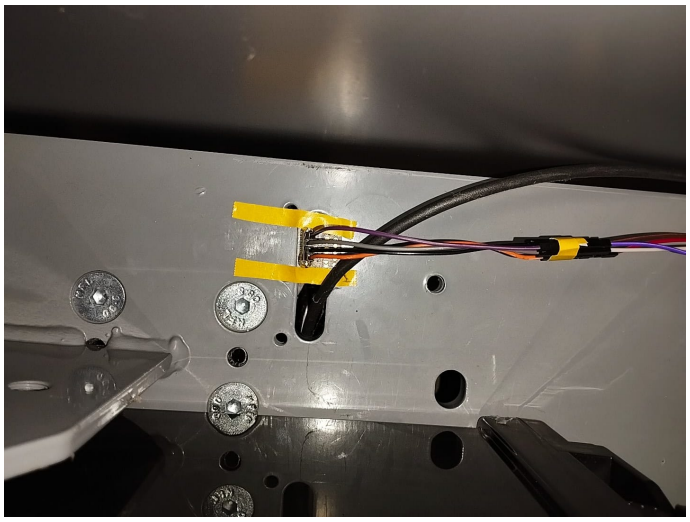


Figure 4.5: MPU-6050 above one of the traction wheels in *GoTugger*

It is important to recognize that the way the sensors were attached to the AMRs was not ideal; it was just a way of making it easy to assemble and disassemble, as the AMRs were on loan, and we had to remove everything before returning them.

### 4.1.2 Microphone

Unlike accelerometers, a microphone does not need to be in contact with the surface to pick up vibrations. As stated in 3, microphones can be used to predict failures in several machinery elements, particularly in bearings and motors.

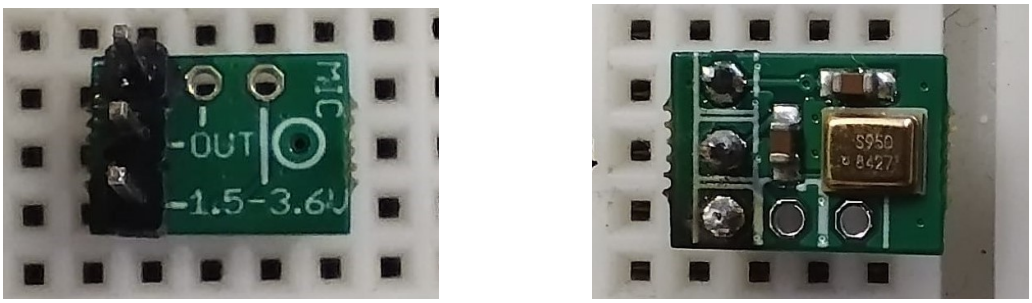
The incorporated microphone will be the main sensor to capture data from the environment, making it a very important choice. According to [54], a Microelectromechanical System (MEMS) ultrasonic microphone is the most suitable and cost-effective in the PdM context.

There are some popular Adafruit Breakout Boards with integrated MEMS microphones, such as *Adafruit I2S MEMS Microphone Breakout - SPH0645LM4H* [59], *Adafruit Silicon MEMS Microphone Breakout - SPW2430* [60], or *Adafruit PDM MEMS Microphone Breakout* [61], but they only pick up to around 20 KHz.

The analog MEMS microphone *SPU0410LR5H-QB* [59] was chosen for being able to capture frequencies up to 80 KHz, and it was tested in this high frequencies in *Teensy Bat* project [3]; a project that developed a device capable of detecting the high frequencies emitted by bats during their locomotion. In addition, it only costs 4€ [4], already integrated into a PCB, and it has analog to digital conversion using *Teensy Audio Shield*, and a very useful and powerful library to *Teensy Microcontroller*. The microcontroller and the audio shield are explored in Section 4.1.6.

[3] https://www.teensybat.com/

[4] Price on 02/07/2024 at Teensy Bat Project Shop




(a) SPU0410LR5H-QB PCB - front (b) SPU0410LR5H-QB PCB - back

Figure 4.6: SPU0410LR5H-QB PCB from Teensy Bat project

It was installed, in both AMRs, just above the front wheel using a 3D-printed support to fix it, as shown in Figures 4.7 and 4.8, to try to collect data, not only from the environment machines but also from the wheel.

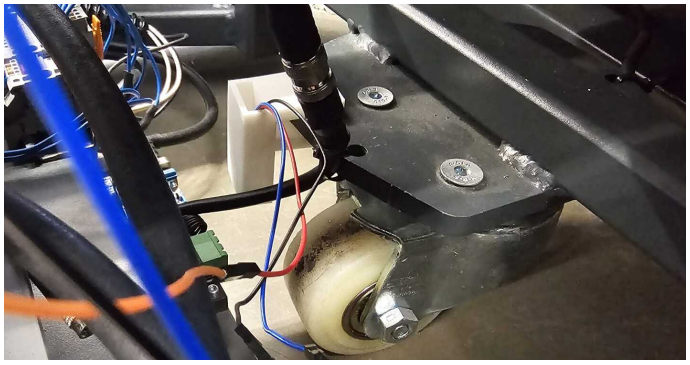


Figure 4.7: SPU0410LR5H-QB installed in *GoMouse-x*

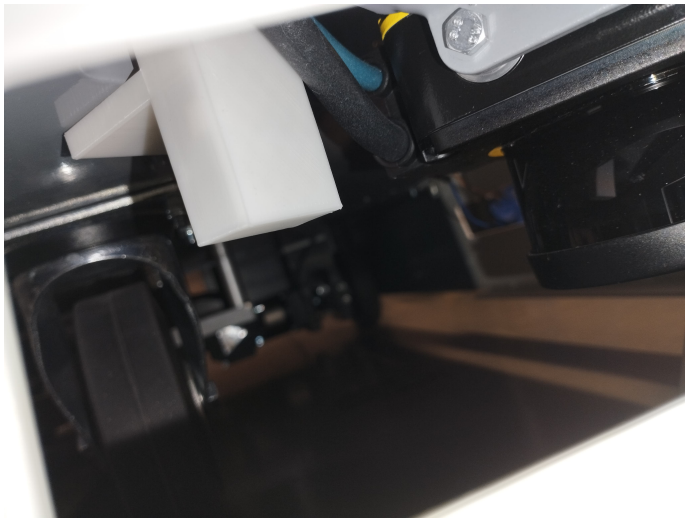


Figure 4.8: SPU0410LR5H-QB installed in *GoTugger*

### 4.1.3 Temperature Sensor

Even though the *MPU-6050* breakout boards already have embedded temperature sensors that can help detect motor faults by identifying overheating, the temperature rise may be caused by the surrounding environment. Placing a temperature sensor in the center of the AMR, away from the motors, can provide readings to assist machine learning models in distinguishing false positives. Additionally, positioning the sensor in an area close to numerous electrical cables and electronics can aid in detecting overloads and short circuits in the electrical system.

There are dozens of temperature sensors available on the market. Still, we chose the Adafruit TMP235 [62] analog sensor, displayed in figure 4.9, due to its cost-effectiveness, with an error of less than +/- 1º Celsius (C) for readings between 0º C and 70º C, and +/- 2º C for the other ranges [63], costing only 2.12€ [5].




(a) TMP235 - front (b) TMP235 - back

Figure 4.9: Adrafuit Breakout Board TMP235

To transform the output voltage in a temperature value, we used the "Adafruit Sensor" Arduino library [64].

Images of *TMP235* sensor installed in the AMRs can be observed in Figures 4.10 and 4.11.

[5] Price on 21/06/2024 at Mouser Electronics

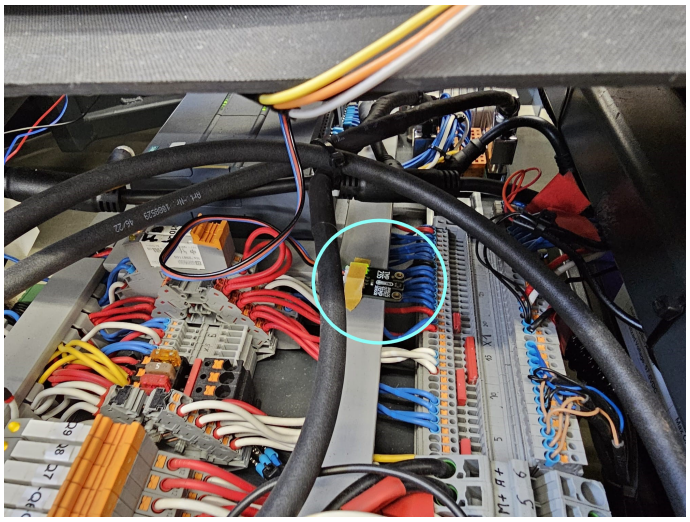


Figure 4.10: TMP235 installed in *GoMouse-x*

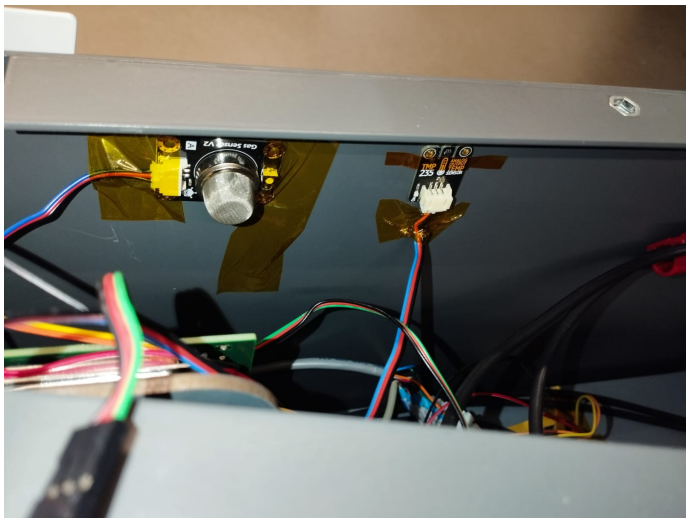


Figure 4.11: TMP235 installed in *GoTugger* (sensor on the right)

### 4.1.4 Gas Sensor

This sensor and the dust sensor were only installed on *GoTugger* due to errors and delays in ordering these sensors.

Even with all the safety measures required at the industrial level, which AMRs usually have, electrical faults are always liable to happen (e.g., short circuits or overloads). With this in mind, we decided to install a flammable gas sensor. We chose the *MQ-2 analog sensor* [65], shown in Figure 4.12, because it can detect carbon monoxide concentration, a gas typically released when

wire coatings are burnt, and also smoke, for only 5.65€ [6]. The "MQUnifiedsensor" library [66] was used to calibrate and read the sensor.




(a) MQ-2 - front (b) MQ-2 - back

Figure 4.12: MQ-2 Analog Sensor

The Figure from the temperature sensor 4.11, also shows the *MQ-2* installed. We set up the sensor next to a cluster of electrical cable connections, and at the same time, it is in a superior part of the AMR since hot air, such as smoke, tends to go up.

### 4.1.5 Dust Sensor

Something that can contribute to the electrical system overheating is accumulated dust. To avoid problems related to this, we have installed a dust detection sensor to alert you when the AMR needs to be cleaned internally. There are not many options of this type on the market, so we opted for the *Sharp GP2Y1010AU0F* [67], visible in Figures 4.13, because it is relatively cheap (10€ [6]), has open-source libraries, can be found in various air quality monitoring projects, and can also detect smoke in the air.



(a) GP2Y1010AU0F - front (b) GP2Y1010AU0F - back

Figure 4.13: Sharp GP2Y1010AU0F Sensor

We placed it above the battery, as shown in Figure 4.14, because it is a large, easy-to-install area where dust will likely accumulate. To make the interface between the hardware and software, we used the "GP2YDustSensor" public library [68].

[6]Price on 21/06/2023 at DFRobot

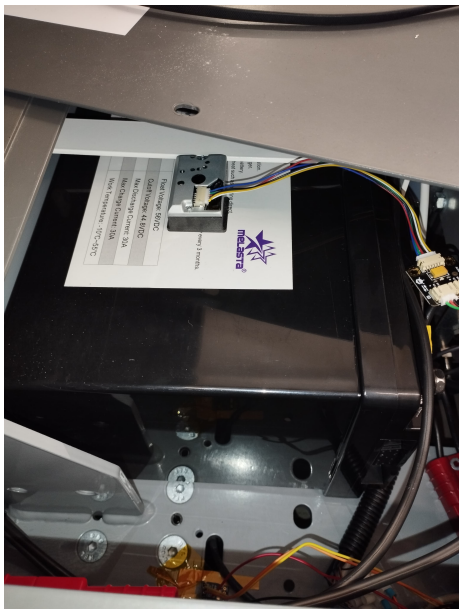
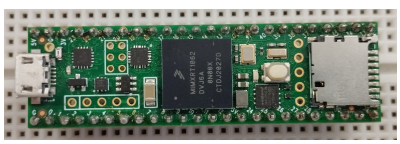


Figure 4.14: GP2Y1010AU0F installed in *GoTugger*

### 4.1.6 Microcontroller and Audio Shield

There are dozens of microcontrollers with different architectures to choose from. However, *Teensy 4.1* (Figure 4.15a) caught the eye for being equipped with an ARM Cortex-M7. Initial benchmarks have shown that C++ code compiled by Arduino tends to achieve 2 instructions about 40% to 50% of the time while performing numerically intensive work using integers and pointers. It was more than five times the benchmark score of popular boards, such as *ESP32*, *Metro M4*, and *Arduino Due*. Besides that, it also can be equipped with an *Audio Shield* (Figure 4.15b) capable of converting analog signals to I2S digital signals and vice versa, as well as providing anti-aliasing filtering. *Teensy* generates three clocks, but *LRCLK* is the most important for this project, as it defines the sample rate, which by default is 44.1 kHz [69]. The full hardware specifications of this board can be seen in table A.3.



(a) Teensy 4.1




(b) Audio Shield Rev 2 for Teensy 4.1

Figure 4.15: Teensy 4.1 and its Audio Shield

Arduino's IDE [7] software with the *Teensyduino* [8] add-on is the primary programming environment for *Teensy*, including a large collection of libraries which are tested and optimized for this microcontroller. Other libraries may be installed manually or by Arduino's library manager [69].

## 4.2 Rolling Bearing Test Bench

Most existing literature uses test benches with bearings attached to motors to simulate industrial rotating machines to collect data easily. We did not make things differently by connecting two DC motors [9] to bearings in different health states. Using two 3D printed parts, the outer part of the bearing was fixed, and only the inner component rotated. We connected the motors to a driver [10] and the driver to an *Arduino Uno*. The *Arduino* was powered by a power bank and the DC motor through a DC battery. This setup can be observed in Figure 4.16.

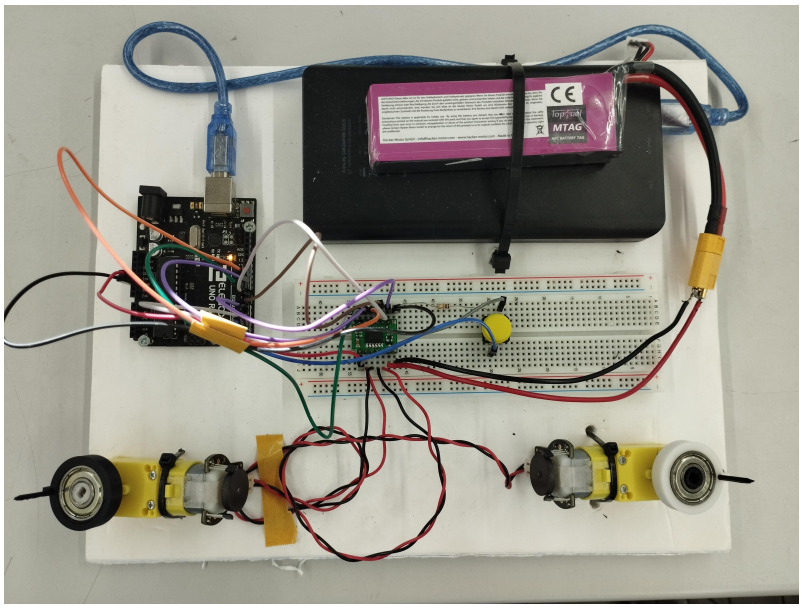


Figure 4.16: Rolling bearing test bench

Four bearings were chosen: two healthy bearings and two damaged bearings, visible in Figure 4.17.

[7] https://www.arduino.cc/en/software
[8] https://www.pjrc.com/teensy/td_download.html
[9] https://www.dfrobot.com/product-1457.html
[10] https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf



Figure 4.17: Two healthy bearings (above) and two damaged bearings (below)

## 4.3 Software Structure

The software was a crucial part of this dissertation, and a well-designed software structure and seamless integration significantly increased the efficiency and effectiveness of the research. A robust software framework streamlines the implementation process and ensures reliability, scalability, and ease of maintenance.

The software structure can be divided into two groups:

- **Data collection** software using Arduino IDE to program the *Teensy* board and the chosen sensors;

- **ML and DL models** using Python programming language.

Some auxiliary software was created, such as code to rotate the DC motor in Arduino language and Python scripts to divide the recorded data into smaller samples to create datasets.

All the code are available on our Dissertation GitHub repository.

### 4.3.1 Data Collection

As stated, sensor data was collected by programming *Teensy* with Arduino IDE. A very simplified diagram of the created code can be observed in Figure 4.18.

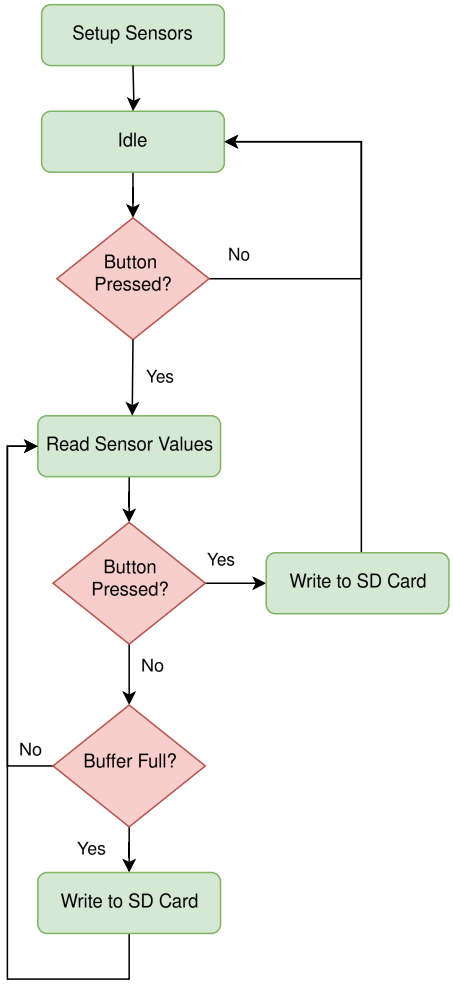



Figure 4.18: Simplified flowchart of data collection code

The open-source "audio" library for *Teensy* [70], created by Paul Stoffregen, was used to record the audio from the microphone. This library makes everything audio-related very simple, even for a layman in the field.

As mentioned in Section 4.1.6, the default rate of *LRCLK* is 44.1 kHz, which, according to the Nyquist Theorem, can pick up up to 22.05 kHz, i.e., the maximum audible frequency. However, we changed the clock frequency to 192 kHz to theoretically pick up frequencies of up to 96 kHz to study its emissions from damaged bearings and try to anticipate faults before they are audible. Nevertheless, as mentioned in Section 4.1.2, the microphone can only reliably pick up frequencies up to 80 kHz, so the remaining frequencies will be ignored. The clock change was made with a library created by a *Teensy* Forum user, FrankB [71].

The microphone data is recorded in an independent thread and saved in Waveform Audio File Format (WAV). All the other sensor data is saved in a comma-separated values file (CSV) and recorded at a sample rate of 50 Hz. If a sensor loses communication with the microcontroller or some operation does not occur as expected, a red LED is turned on, and the recording is stopped

and saved. A CSV error log file is also created to help debug what happened. While recording, the *Teensy* built-in LED is turned on.

### 4.3.2 Dataset

To simplify, three things should be taken into account from now on:

- All recorded data were divided into 1-second samples;
- The AMR was arbitrarily considered near the test bench when the front of the AMR was within 3 meters of the test bench. This was used to label the dataset;
- The bearings were lubricated before the recordings.

With this in mind, two main labeled datasets were created: the "Bearings Dataset" and the "Floor Noise Dataset".

The Bearings Dataset includes several subsets:

- **Two damaged bearings -** It comprises samples with the two damaged bearings rotating in both motors;
- **Two healthy bearings -** Equal to the previous one, but applied to the healthy bearings;
- **One damaged bearing -** It includes samples with the damaged bearings rotating, one at a time, in each of the motors, i.e., each damaged bearing rotates in both motors separately;
- **One healthy bearing -** Equal to the previous one, but applied to the healthy bearings;
- **One damaged and one healthy bearing -** It comprises samples with one damaged bearing and one healthy bearing rotating simultaneously.

In all these subsets, in most of the samples, the AMR was moving close to the test bench; in the others, it was stationary in different positions. To maximize the robustness of the dataset, several combinations were recorded within each subset, e.g., the subset “One damaged and one healthy bearing” contains samples with the two healthy bearings and the two damaged bearings, also changing the motor that rotates each bearing type. Around 10% of the dataset was recorded with the AMR moving on a tiled floor to cause more noise and increase the robustness of the models. The rest was recorded on a smooth floor, identical to typical factory floors.

The "Floor Noise dataset" contains samples in which the robot moves on the two types of floor mentioned in the previous paragraph, with an identical distribution, but without any interference from the test bench.

We also recorded the sound from one healthy bearing and from the heavily damaged bearing close to the microphone to analyze the frequencies they emitted.

In total, we had 3329 samples from healthy bearings, 4607 from damaged bearings, and 4296 from floor noise.

### 4.3.3 Pre-Processing

As mentioned in Section 2.3, applying pre-processing techniques can significantly improve the performance of ML and DL models.

A configuration Python file was created where the parameters of the signal-processing techniques can be changed, and the techniques we want to perform can be turned on and off, making it easier to compare the models' performances with different levels of pre-processing without changing the main code. It is also possible to force the dataset to be perfectly balanced by limiting the largest class or by using Smote [11] to generate synthetic data for the minority class. An example of how it works is shown in Figure 4.19.

```python
# Preprocessing options
preprocessing_options = {
    'sp': False,
    'noise_reduction': False,
    'fft': True,
    'mfcc': False,
    'stft': False,
    'sp_accel': True
}

# Noise reduction parameters
noise_reduction_params = {
    'n_fft': 2048,        # Number of FFT points
    'hop_length': 512,    # Hop size between successive frames
    'prop_decrease': 0.5  # Property for noise reduction
}

# FFT parameters
fft_params = {
    'n_fft': 2048,  # Number of FFT points
    'fmin': 500,    # Minimum frequency (Hz)
    'fmax': 80000   # Maximum frequency (Hz)
}
```

(a) Example of signal processing parameters - 1

```python
# MFCC parameters
mfcc_params = {
    'n_fft': 2048,       # Number of FFT points
    'hop_length': 512,   # Hop size between successive frames
    'n_mels': 106,       # Number of Mel filter banks
    'fmin': 500,         # Minimum frequency (Hz)
    'fmax': 80000,       # Maximum frequency (Hz)
    'n_mfcc': 40         # Number of MFCC coefficients
}

# STFT parameters
stft_params = {
    'n_fft': 2048,       # Number of FFT points
    'hop_length': 512,   # Hop size between successive frames
    'fmin': 500,         # Minimum frequency (Hz)
    'fmax': 80000        # Maximum frequency (Hz)

# Frequency resolution = sampling rate / window size = 192000
# Time resolution = hop size / sampling rate = 512 / 192000 ≈
# This means you can observe changes in the frequency content
}

# Balance dataset options (if both are True, the smote will be

# Use Smote to balance dataset
smote = True

# Force 50%-50% dataset
force_balanced_dataset = False
```

(b) Example of signal processing parameters - 2

Figure 4.19: Example of how parameters can be changed in setup file

The pre-processing techniques chosen to test were:

- **Noise Reduction (NR) -** To reduce the base noise of the microphone, an audio file with minimal audible external interference was recorded. Then, a spectrogram of the signal and the noise signal is computed, and a noise threshold for each frequency band of that signal/noise is estimated. That threshold is used to compute a mask, which gates noise below the frequency-varying threshold. The "Noise Reduced" [72] Python library was used to perform this;
- **FFT -** Useful to compare the frequencies emitted from the different bearings;
- **STFT -** Valuable to analyze how the emitted frequencies change over time;

[11] Smote is a commonly used library to generate synthetic data to balance datasets or to generate more data when needed. More information is available at https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

- **MFCC -** Despite being a method that is not new, it is little cited in the existing literature about PdM. Therefore, we decided to explore it;
- **Statistical Properties (SP) -** Mean, Standard Deviation (STD), Root Mean Square (RMS), Kurtosis, and Skew were performed in the 192000 frames captured in each sample;
- **SP for the accelerometer data -** The same properties from the previous point were performed in the accelerometer data. The idea behind these methods is to help the models distinguish random events and floor noise from an anomaly. The definition of these properties can be found in Table A.4;

FFT, STFT, and MFCC are explained in more detail in Section 2.3.

#### 4.3.3.1 Parameters Choice

All the pre-processing techniques mentioned, except for the statistical properties, are highly configurable, with dozens of customizable parameters. We considered the ones with more potential impact and added them to the config file to make changing it easier. For instance, parameters such as the number of FFT points, hop length, Mel filter banks, and minimum/maximum frequencies can significantly influence the outcome of the pre-processing steps. By providing clear explanations and default values for these parameters, we aim to improve the flexibility and reproducibility of our audio processing pipeline. All of the parameters we chose to configure for each technique are described below.

- **Noise Reduction**
  - *n_fft*: The number of points used in the FFT. Larger values enhance frequency resolution but increase computational cost.
  - *hop_length*: The number of samples between successive frames. Smaller values enhance time resolution but increase computational cost.
  - *prop_decrease*: A property used in the specific noise reduction algorithm, the exact impact depends on the algorithm implementation.
- **FFT**
  - *n_fft*: Same as in noise reduction.
  - *fmin*: The minimum frequency to consider in the FFT analysis.
  - *fmax*: The maximum frequency to consider in the FFT analysis.
- **MFCC**
  - *n_fft*: Same as in noise reduction.
  - *hop_length*: Same as in noise reduction.

  - *n_mels*: The number of Mel filter banks used to represent the frequency spectrum. More Mel filter banks capture more detailed spectral information.
  - *fmin*: Same as in FFT.
  - *fmax*: Same as in FFT.
  - *n_mfcc*: The number of MFCC coefficients extracted. More coefficients capture more detailed information about the audio signal.

- **STFT**
  - *n_fft*: Same as in noise reduction.
  - *hop_length*: Same as in noise reduction.
  - *fmin*: Same as in FFT.
  - *fmax*: Same as in FFT.

Considering that the sampling frequency of our audio recording is 192 KHz and that the used libraries suggested default parameters for CD quality (44.1 KHz), we adjusted the default values to those shown in Table 4.2 to account for the higher resolution of our data.

Table 4.2: Suggested and predefined values for pre-processing parameters

| Parameters | n_ftt | hop_length | prop_decrease | fmin (KHz) | fmax (KHz) | n_mels | n_mfccs |
|---|---|---|---|---|---|---|---|
| Suggested | 1024 | 256 | 1 | - [12] | - | 128 | 10 - 20 |
| Predefined | 2048 | 512 | 0.5 | 0.5 | 80 | 106 [13] | 40 |

By increasing the number of FFT points, hop length, and the number of MFCCs, we increase the number of features extracted by the respective techniques while decreasing the proportion of decrease to apply to the mask with the goal of not attenuating potential useful frequencies in the process. The minimum and maximum frequency were chosen based on Section 5.2 results.

Statistical summaries are often computed to transform MFCC and STFT results into feature vectors suitable for machine learning models. For MFCCs, each coefficient's mean and standard deviation were calculated across the number of Mel bands. Similarly, for STFT spectrograms, these statistics were computed for each frequency bin across time.

### 4.3.4 Prediction Models

To find the most suitable model with the best performance, several models were tested with different levels of pre-processing. Although there are several ML libraries, the "TensorFlow" (TF) library [73] was chosen due to the possibility of converting the model into a light model capable of running on microcontrollers and small single-board computers, i.e., a Raspberry Pi [14]. This can

[12] "-" means not applicable

[13] This value was calculated iteratively to obtain the maximum value of n_mels without having empty bands. This value is lower because the samples we are using are 1 second long, and the samples used in the examples are 10 seconds long.

[14] https://www.raspberrypi.com/

be particularly useful for running the model in real time on a small computer inside the AMR. The code was designed to use Compute Unified Device Architecture (CUDA) from NVIDIA whenever a compatible graphics card was available to accelerate model training and inference.

The developed models will be used for classifying a bearing as either "healthy" or "damaged" and for distinguishing between noise ("noise") and the presence of a bearing ("bearing"). The rationale behind this approach is to optimize the use of the sensory module based on the robot's location. If the robot's position relative to machines with bearings is known, the sensory module can be activated only when it is near the machine, eliminating the need to differentiate between industrial noise and the presence of a bearing.

However, if the location is unknown, the sensory module will remain active continuously, classifying incoming samples as either noise or bearing sounds. In the case of a bearing detection, the sample will then be passed to the second model, which will determine whether the bearing is healthy or damaged. This tiered approach ensures efficient use of computational resources while maintaining accuracy in different operational scenarios.

The following sections will explore how the code was structured to optimize it and make it easily scalable, as well as the tested models.

#### 4.3.4.1 Decision Trees

As stated in Section 3.2.2.2, Decision Trees are a supervised ML algorithm that works by recursively splitting the dataset into subsets based on the value of input features, creating a tree-like model of decision, which can be valuable for evaluating the effectiveness of different pre-processing techniques. We chose the "Yggdrasil Decision Forests" library [74] to work with these types of models as it is fully compatible with "TensorFlow" and it performs faster [75] and better than other popular choices [74]. These models are also lighter when compared to Neural Networks [74], which is a factor to consider in our scenario, as mentioned in Section 4.3.4.

There are several variants of this type of ML method, with its advantages and disadvantages. In our scenario, we chose the following:

- **Random Forest** [15] **-** As mentioned in Section 3.2.2.2, RFs operate through bagging (bootstrap aggregating), where multiple decision trees are trained independently on different random subsets of the data, and their predictions are averaged to reduce variance. Each tree is grown to its full depth, making it a strong learner, but the averaging process mitigates overfitting. This method aims to minimize variance, which makes it robust to overfit, particularly on small datasets or when there is a high number of input features [74];

- **Gradient Boosted Decision Tree (GBDT) -** GBDTs use boosting, where trees are added sequentially, each one attempting to correct the mistakes of the previous tree. They are trained on the residual errors of the previous ensemble. This makes GBDTs capable of minimizing both bias and variance but also prone to overfitting if not properly regularized. This

[15] Although it is not a decision tree but an ensemble of multiple decision trees - as mentioned in Section 3.2.2.2 - we have decided to group it here for simplicity.

model tends to build smaller trees, making each iteration more computationally efficient while focusing on improving the model's weaknesses. [74, 76].

Both algorithms have hyperparameters in common, such as the number of trees and the maximum tree depth. However, these parameters play a different role in each model and should be tuned accordingly.

The two hyperparameters that vary the most between these two types of decision trees are the number of trees and the maximum depth. The next four paragraphs explain in detail the impact and differences of each of these in each model. Table 4.3 summarizes it.

In RF, the number of trees represents how many individual decision trees are created, each trained on different random subsets of the data (via bootstrapping). Since each tree is trained independently, adding more trees helps improve the stability and accuracy of the model by reducing variance. The more trees you have, the more reliable the final prediction will be. Typically, RFs require a large number of trees (often in the hundreds or even thousands) to achieve good performance, as each tree is relatively deep and independent.

In contrast, in GBDTs, trees are added sequentially, with each new tree trying to fix the mistakes made by the previous ones (i.e., boosting). As a result, GBDT can often achieve high accuracy with fewer trees than RF because each tree contributes directly to improving the model. However, too many trees can lead to overfitting, especially if the learning rate (another GBDT-specific hyperparameter) is not appropriately set.

Regarding maximum depth, each tree is grown independently in RF and typically to a larger depth (often greater than 16). The large depth allows each tree to capture complex patterns in the data. Since RF averages over many deep trees, it can afford to have overgrown individual trees without heavily overfitting, as the randomness introduced by bagging helps counteract overfitting. Deep trees in RF can help the model capture intricate relationships between features.

In GBDT, trees are built sequentially, and typically, each tree is relatively shallow (with a depth between 3 and 8). Shallow trees are preferred because each tree is only meant to correct the mistakes of the previous ones rather than model the entire problem from scratch. Shallow trees also help prevent overfitting since each tree adds incremental information to the ensemble.

Based on these values, we chose 100 and 300 trees for the GBDT and RF models, respectively. The maximum depth was left to be defined by the model because of our choice of the parameter "growing strategy", as will be explained below.

Table 4.3: Comparison of Tree Depth and Number of Trees in RF and GBDT

| **Hyperparameter** | **RF** | **GBDT** |
|---|---|---|
| **Number of Trees** | Requires more trees because each tree is independent. The model benefits from averaging many deep trees to reduce variance. | Requires fewer trees, as each new tree improves on the last. Too many trees can lead to overfitting if not regularized properly. |
| **Max Depth** | Deeper trees are used (often $>$ 16) to fully model the data's complexity. Bagging helps to prevent overfitting. | Shallower trees (depth 3-8) are used to make incremental improvements to the model. Deeper trees can lead to overfitting. |

Yggdrasil documentation [77] explains every hyperparameter for each model and gives us hypertuning templates for improving the algorithms for each scenario. Based on this, we changed some parameters in order to improve our results. Starting with the common parameters between the two decision trees:

- **Split Axis -** By default, trees are "orthogonal" or "axis aligned"; that is, each split/condition tests a single feature. By opposition, conditions in oblique trees ("sparse oblique") can use multiple features, generally improving performance with the cost of being slower to train. We chose "sparse oblique" to refine our model results;

- **Number of Projections Exponent -** Oblique splits try out the maximum between the number of numerical features raised to the power of the number of projections exponent and the maximum number of projections (6000 by default) random projections for choosing a split. Therefore, increasing this number and possibly the maximum number of projections may improve model quality and significantly increase training time. We will start with "1" and then increase if needed;

- **Growing Strategy -** Decision trees can be grown using two strategies: "Local" and "Best First Global". In the "Local" approach (default strategy), each node is split independently based on predefined constraints, such as maximum depth or the minimum number of observations. As long as a node satisfies these criteria, it is split, following the traditional method for growing decision trees. In contrast, the "Best First Global" approach selects the node that provides the best overall loss reduction across the entire tree for splitting rather than splitting nodes independently. This strategy slightly increases training time but generally enhances the model. When chosen, it is recommended to tune the maximum number of nodes instead of the maximum tree depth. We chose "Best First Global";

- **Maximum Number of Nodes -** When using the "Best First Global" strategy for growing decision trees, the maximum number of nodes controls the number of nodes the tree can have. A typical value for this parameter often depends on the size and complexity of the

dataset. Still, for small/medium datasets, it generally ranges from 100 to 500 for a GBDT and from thousands to even tens of thousands for RFs. By focusing on this hyperparameter, we directly control the total complexity of the tree based on the number of nodes, allowing the tree to grow dynamically to its optimal structure. Constraining the maximum depth can interfere with this process, as the best splits may occur deeper in the tree, and limiting depth could prematurely halt important splits. Therefore, leaving the maximum depth unconstrained gives the model more flexibility to find the globally optimal tree structure while managing complexity through the number of nodes. We will start with 100 for GBDT and 1000 for RF and increase if needed.

Regarding the RF model, we changed an additional specific hyperparameter:

- **Winner Takes It All -** It controls how classification trees vote. If true (default), each tree votes for one class. If false, each tree votes for a distribution of classes. According to [77], setting it to false is often preferable. For this reason, we set it to true.

Concerning the GBDT model, we changed three additional parameters:

- **L1 Regularization -** L1 regularizer introduces a penalty on the absolute size of the coefficients, encouraging sparsity in the model and potentially improving interpretability by driving some feature weights to zero. We changed it from 0 (default) to 0.01. By applying this small regularization value, we aim to balance model complexity and performance while preventing overfitting;

- **L2 Regularization -** L2 regularizer penalizes the square of the coefficients' values, promoting smaller weights overall. Introducing L2 regularization can help mitigate issues related to multicollinearity and enhance the model's generalization capabilities. Setting it from 0 (default) to 0.01 balances underfitting and overfitting, aiming for optimal predictive performance;

- **Shrinkage -** Analogous to neural network learning rate, it is a coefficient applied to each tree prediction. A smaller value tends to give more accurate results (assuming enough trees are trained) but results in larger models. The default value is 0.1, but we reduced it to 0.01 to try to make the model more robust and to generalize better.

These three values might need to be adjusted according to the preliminary results.

Despite being enabled by default, it is important to highlight that GBDT models are trained with "Early Stopping" based on the "Loss Increase" criterion. This means that if the loss (error) does not decrease for a set number of consecutive trees — 30 trees by default — the training process will stop. At this point, the model will retain the number of trees it had when the loss last improved, preventing overfitting and unnecessary additional trees. This mechanism helps to optimize both model performance and training efficiency. RF model does not have this feature, but it provides logs during the training, making it possible to optimize the number of trees manually.

Another significant parameter important to refer to is the "Loss Function". Loss, or cost function, quantifies the error between the model's predicted and actual values. Essentially, the loss function is the objective function that the model aims to minimize during training. In GBDT, the default loss function is the "Binomial Log-Likelihood"; it is used in binary classification problems, where the goal is to predict probabilities for two classes. It measures the accuracy of probabilistic predictions by comparing the predicted probability for the correct class to the actual label. The loss increases as the predicted probability diverges from the true class label. The mathematical expression that describes the function is shown in Equation 4.1, where $y$ is the true label and $\hat{y}$ is the predicted probability of the positive class.

$$\text{Log Loss} = -\left(y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})\right) \tag{4.1}$$

In contrast, RFs for classification problems do not directly minimize a specific loss function during training, unlike GBDT. Instead, they rely on majority voting across the individual decision trees in the forest. However, implicitly, the algorithm is trying to minimize classification error (misclassification rate) by using individual decision trees to reduce errors and make more accurate predictions. This error is often called Out of Bag (OOB) Loss, and it is the proportion of times that the out-of-bag samples are incorrectly classified.

These initial hyperparameters, summarized in Tables 4.4 and 4.5, were chosen to try to maximize the results without significantly increasing the training time and, more importantly, the inference time. These parameters will be adapted, if necessary, in the results chapter.

Table 4.4: Default and predefined values for DT models - Part 1

| Parameters | Number of Trees | Maximum Depth | Maximum Number of Nodes | Growing Strategy |
|---|---|---|---|---|
| Default GBDT | 300 | 6 | - | Local |
| Default RF | 300 | 16 | - | Local |
| Predefined GBDT | 100 | Not Limited | 100 | Best First Global |
| Predefined RF | 300 | Not Limited | 1000 | Best First Global |

Table 4.5: Default and predefined values for DT models - Part 2

| Parameters | Split Axis | Number of Projections Exponent | Winner Takes It All | L1 | L2 | Shrinkage |
|---|---|---|---|---|---|---|
| Default GBDT | Axis Aligned | - | - | 0 | 0 | 0.1 |
| Default RF | Axis Aligned | - | True | - | - | - |
| Predefined GBDT | Sparse Oblique | 1 | - | 0.01 | 0.01 | 0.01 |
| Predefined RF | Sparse Oblique | 1 | False | - | - | - |

#### 4.3.4.2 Autoencoder

As mentioned in Section 3.2.3.1, an AE is an unsupervised DL approach that aims to compress the input data into a lower-dimensional space and then reconstruct it back to its original form. The difference between the input and the output is called *reconstruction error*, and the higher the error,

the higher the chance of anomalous data being present. Those classified as “bearings” would then be submitted to the supervised classification model. AE was chosen over, for example, a One-Class SVM, because it is compatible with "TensorFlow". "Keras" library [78] was used to design it as it is widely used and compatible with TF and other popular useful libraries.

A popular and straightforward architecture is a Dense Autoencoder (DAE), also known as a fully connected or feed-forward architecture. DAEs can effectively compress data into lower-dimensional representations, often capturing essential features crucial for downstream tasks. This makes them useful for data compression, visualization, and feature extraction tasks. Furthermore, unlike convolutional or recurrent layers that excel in processing spatial or sequential data, dense autoencoders work well with tabular data where features do not have a specific spatial structure. This is ideal for our scenario where the pre-processed data shape is basically a table. An illustrative image of an Autoencoder can be observed in image 4.20.

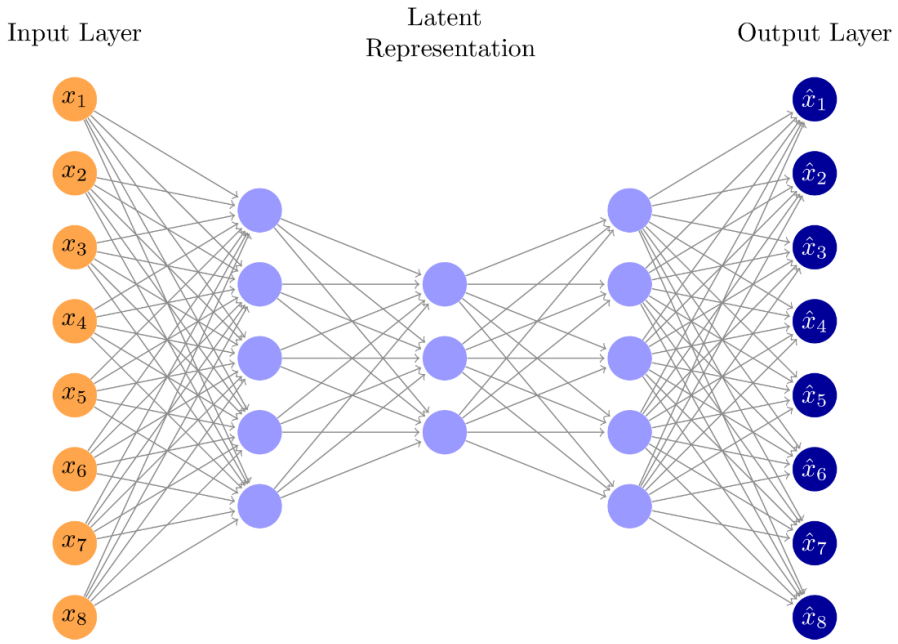



Figure 4.20: Illustrative example of an Autoencoder [79]

There are a handful of other different popular AE architectures. However, some need special attention to their input shape, e.g., Recurrent Autoencoders (RAEs) and Convolutional Autoencoders (CAEs). RAEs are designed for sequential data like time series or text, needing inputs in the form of sequences, while CAEs are specially designed for image data, requiring inputs with spatial dimensions, such as height, width, and channels. This specificity allows these architectures to leverage patterns inherent in the data, such as temporal dependencies in RAEs and spatial hierarchies in CAEs, enhancing their performance in their respective domains. Still, they are incompatible with our pre-processing approach.

Bearing this in mind, we designed a DAE with the following architecture:

**Input Layer -** Layer responsible for converting the data into tensors [16];

**Gaussian Layer -** Added to the input to enhance the autoencoder's robustness by forcing it to learn more generalized patterns rather than memorizing noise. This approach improves the model's ability to denoise inputs and prevents overfitting;

**Encoder with Regularization -** The encoder consists of one or two Dense layers with progressively decreasing units and ReLU activations, enabling the model to capture complex non-linear relationships in the data. A larger number of neurons increases the capacity of the network to capture complex patterns but also heightens the risk of overfitting. To mitigate this, each Dense layer is followed by a batch normalization layer to stabilize learning and a dropout layer to randomly deactivate neurons, further reducing overfitting. Additionally, L1 and L2 regularization is applied to each Dense layer to penalize overly complex weight patterns and encourage simpler, more generalizable models;

**Bottleneck Layer -** It is a critical component that compresses the input into a lower-dimensional space, capturing the most significant features of the data. This layer acts as the main feature extractor and serves as the core of the autoencoder's learning capabilities;

**Decoder -** It mirrors the encoder's structure, gradually reconstructing the data from the bottleneck representation to upsample the compressed data;

**Output Layer -** The output is another Dense layer with ReLU activation function to reconstruct data back to the original dimensionality;

**Loss -** The model is compiled with a Mean Squared Error (MSE) loss function, which measures the reconstruction error between the input and the output, guiding the network to minimize the difference between them. This loss was chosen because it is the most popular in AEs in the literature;

**Optimizer -** It is responsible for updating the weights of a neural network to minimize the loss function, guiding the learning process by adjusting the model's parameters. The Adam optimizer, with a starting learning rate of 0.001 (default learning rate for Adam), was employed due to its adaptive learning rate capabilities, which help balance speed and convergence stability.

The number of layers and units in the encoder depends on the dimensionality of the output features. For example, when using MFCCs with twenty coefficients, the output consists of forty features representing the average and standard deviation for each coefficient. In contrast, for an STFT with a length of 1024, the output is determined by half of the length (since only positive frequency bins are used), resulting in 512 features for the average and 512 for the standard deviation.

The encoder must be tailored to accommodate the varying dimensionality of these feature sets. For smaller feature sets, like MFCCs, the architecture can be relatively shallow, but for larger inputs, such as STFTs, deeper networks with more units per layer may be required to effectively compress the input data. The objective is to capture the essential features while reducing dimensionality in a way that preserves information necessary for reconstruction.

[16] A tensor is a multi-dimensional array of data, generalizing scalars, vectors, and matrices, and is used in machine learning to represent and manipulate input data, weights, and outputs in models

Tables 4.6 and 4.7 contain the hyperparameters that will be searched for during Bayesian Optimization for the STFT and MFCC preprocessing techniques. The unsupervised models will only be tested with these two techniques for the reasons that will be explained in Section 5.5.1.

Table 4.6: Hypertuning values STFT Dense Model

| Layer | Minimum | Maximum | Step |
|---|---|---|---|
| Dense Layer 1 | 256 | 512 | 64 |
| Dense Layer 2 | 128 | 256 | 32 |
| Bottleneck | 32 | 128 | 16 |

Table 4.7: Hypertuning values MFCC Dense Model

| Layer | Minimum | Maximum | Step |
|---|---|---|---|
| Dense Layer | 16 | 32 | 4 |
| Bottleneck | 8 | 16 | 2 |

Finding the optimal values for the parameters of each layer — such as the standard deviation of the Gaussian noise, the number of neurons in each dense layer, regularization coefficients, dropout rates, bottleneck units, and learning rate — can be highly challenging. Hypertuning will be employed to identify the most suitable hyperparameter settings. This process systematically searches various parameter combinations to optimize the model's performance, ensuring the autoencoder is balanced between complexity and generalization. There are some types of hypertuning techniques, but we chose Bayesian Optimization due to its efficiency and effectiveness in finding optimal hyperparameters with fewer evaluations. Bayesian Optimization intelligently explores the parameter space, balancing exploration and exploitation, and adapts based on past results. This approach reduces computational cost and time compared to traditional methods like grid search or random search, making it ideal for complex models like autoencoders.

#### 4.3.4.3 Model's Overview

Figure 4.21 illustrates the overall workflow of bearing classification models. Before this process, data from each sample undergoes pre-processing to enhance features.

The classification model then processes the pre-processed data and categorizes each sample as either a "healthy bearing" (0) or a "damaged bearing" (1). For traditional classifiers, such as decision trees, this involves learning patterns associated with each class directly.

For the Autoencoder model, the classification approach differs. The Autoencoder is trained to replicate the input data; classification is then inferred based on the reconstruction error, as mentioned in Section 4.3.4.2.

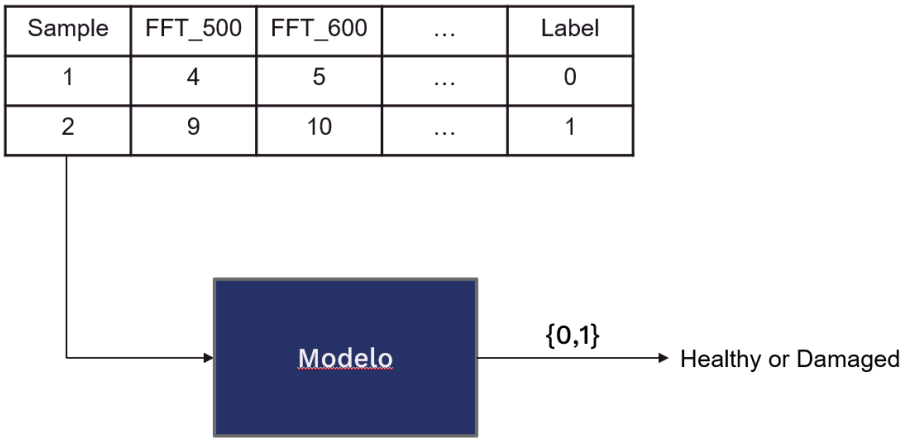



Figure 4.21: Model's overview

# Chapter 5

# Experimental Results

In line with the previous chapter, the next chapter will test the created models to evaluate their performance under various conditions.

## 5.1 Evaluation Metrics

Before delving into the most used metrics, it is important to consider that, for the decision trees, the dataset was split with 20% for the test set and 80% for the training set.

As the Autoencoder is an unsupervised method, the model is only trained with regular data. The noise data was also split 20% for the test set and 80% for the training set. However, this time, the test set was split again with 65% for the validation set and 35% for the test set, and the same number of bearing samples were added to each set to be perfectly balanced for a better interpretation of the metrics. With the validation set, an optimal threshold is chosen to maximize an objective, e.g., accuracy, recall, f1-score... Then, the test is performed to evaluate the model with the chosen threshold.

Evaluating the performance of a machine learning model is crucial for understanding its effectiveness and making informed decisions. Several metrics are commonly used to assess model quality. The following subsections explore the chosen metrics.

### 5.1.1 Confusion Matrix

A confusion matrix is a tabular representation of the performance of a classification model on a test dataset. It provides a breakdown of predictions and actual values, allowing for a detailed model performance analysis. A disposition of the matrix is visible in Table 5.1.

Table 5.1: Confusion Matrix Disposition

| Predicted | Actual Positive | Actual Negative |
|---|---|---|
| Positive | True Positive (TP) | False Positive (FP) |
| Negative | False Negative (FN) | True Negative (TN) |

### 5.1.2 Accuracy

Accuracy measures the overall correctness of the model's predictions. It is calculated as the ratio of correct predictions to total predictions [80], as shown in Equation 5.1.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

While accuracy is a straightforward metric, it can be misleading in imbalanced datasets.

### 5.1.3 Precision

Precision measures the proportion of positive predictions that were actually correct [80], as indicated in Equation 5.2. It focuses on the accuracy of positive predictions.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

### 5.1.4 Recall (Sensitivity)

Recall measures the proportion of actual positive cases that were correctly identified [80], as implied in Equation 5.3. It focuses on the model's ability to find all relevant cases.

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

### 5.1.5 F1-Score

The F1-score is the harmonic mean of precision and recall, balancing the two metrics [80]. It is useful when there's an imbalance between positive and negative classes. Its formula can be seen in Equation 5.4.

$$F1 - Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5.4}$$

### 5.1.6 ROC-AUC

The ROC-AUC (Receiver Operating Characteristic - Area Under The Curve) is calculated by integrating the area under the ROC curve, generated by plotting the True Positive Rate against the False Positive Rate for different classification thresholds [80]. The AUC can be computed using numerical methods, such as the trapezoidal rule, which approximates the area as the sum of trapezoids formed between points on the ROC curve. The formula for calculating the AUC using the trapezoidal rule is given in Equation 5.5.

$$ROC - AUC = \sum_{i=1}^{n-1} \frac{(FPR_{i+1} - FPR_i) \cdot (TPR_{i+1} + TPR_i)}{2} \tag{5.5}$$

In this formula, $FPR_i$ and $TPR_i$ represent the False Positive Rate and True Positive Rate at the $i$-th threshold, respectively, and $n$ is the total number of thresholds used. The AUC summarizes the classifier's performance across all thresholds, with an AUC of 1 indicating a perfect classifier and 0.5 being equivalent to random guessing.

## 5.2 First Analysis of the Bearings

In order to conduct a first analysis of the frequencies emitted by the bearings considered in Section 4.2, we performed an FFT of a 15-second sample of one of the healthy bearings and the less damaged bearing, and then applied the noise reduction function, explained in Section 4.3.3.

The FFTs are shown in Figures 5.1 and 5.2, and there are apparent differences between the three bearings. The main differences seem to be between 500 Hz and 40 KHz, with less marked differences up to around 80 KHz.

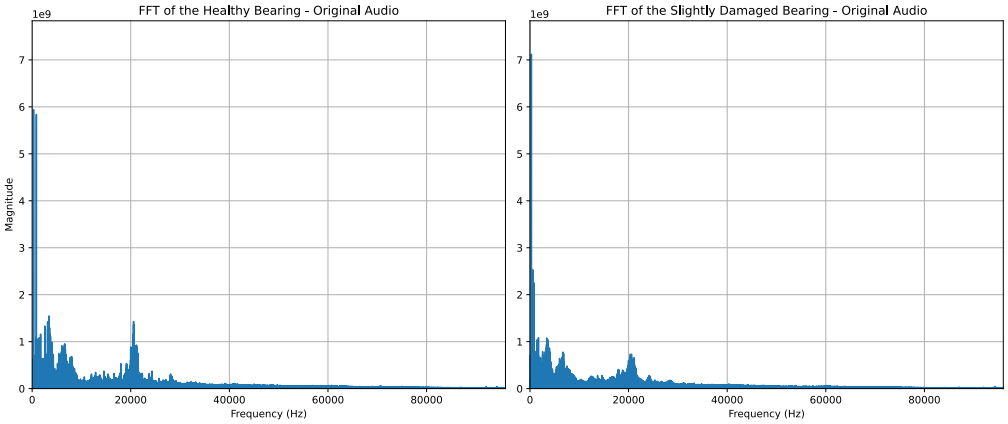



Figure 5.1: FFT of Slightly Damaged Bearing *versus* Good Bearing

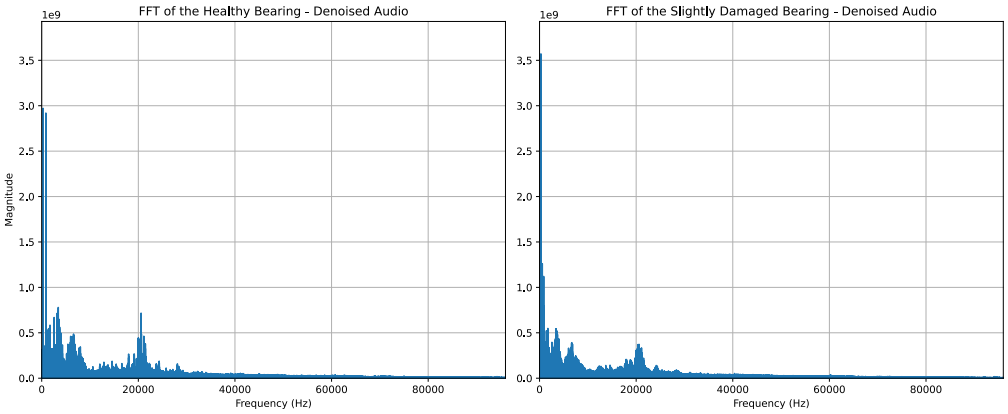



Figure 5.2: Denoised FFT of Slightly Damaged Bearing *versus* Good Bearing

## 5.3 Supervised Models for Bearing State Classification

In this section, the chosen supervised methods will be tested and evaluated with different levels of pre-processing to infer their ability to distinguish between a healthy bearing and a damaged one.

In addition to the metrics mentioned in Section 5.1, the average inference time ($Inf_t$) and pre-processing time per sample ($Pre-Proc_t$) were also considered. These two additional metrics are important to understand the impact they have on the total processing time of a sample, which on the computer where it is being tested does not affect much due to its specifications (shown in Table A.5), but which on a small computer can make a difference. All metrics were calculated in the test set except for the cross-validation [1] that was performed on all the data with ten k-folds.

The Smote technique was used so that both classes had exactly the same number of samples.

### 5.3.1 Random Forest

The RF model was trained separately with each of the pre-processing techniques chosen in Section 4.3.3, with and without the noise suppression technique, to analyze its effect. The results rounded to two decimal places are available in Table 5.2. These results are calculated without changing the parameters defined in Section 4.3.4.1

Table 5.2: Results of the RF models test metrics

| Model | Accuracy | Precision | Recall | F1-score | ROC-AUC | OBB Loss | $Pre-Proc_t$ (ms) | $Inf_t$ (ms) |
|---|---|---|---|---|---|---|---|---|
| RF + SP | 0.73 | 0.74 | 0.72 | 0.73 | 0.80 | 0.53 | 7.74 | 0.07 |
| RF + SP + NR | 0.70 | 0.71 | 0.69 | 0.70 | 0.80 | 0.54 | 46.08 | 0.07 |
| RF + FFT | 0.90 | 0.89 | 0.91 | 0.90 | 0.97 | 0.35 | 6.88 | 0.05 |
| RF + FFT + NR | 0.90 | 0.90 | 0.91 | 0.91 | 0.97 | 0.36 | 51.88 | 0.06 |
| RF + MFCC | 0.95 | 0.95 | 0.95 | 0.95 | 0.99 | 0.23 | 18.28 | 0.04 |
| RF + MFCC + NR | 0.95 | 0.94 | 0.96 | 0.95 | 0.99 | 0.23 | 72.27 | 0.04 |
| RF + STFT | 0.99 | 0.99 | 0.99 | 0.99 | 1.00 | 0.07 | 9.62 | 0.04 |
| RF + STFT + NR | 0.99 | 0.99 | 0.99 | 0.99 | 1.00 | 0.08 | 50.34 | 0.04 |

For this case study, it should be noted that the results show that the noise suppression technique did little or nothing to help the model; on the contrary, it more than doubled the total pre-processing time per sample.

Time-frequency techniques (STFT and MFCC) obtained the best results, with STFT achieving an almost perfect result. The probability distribution of the RF combined with STFT, displayed in Figure 5.3, shows that the model clearly distinguishes between the two groups, indicating a good classification performance. However, the high concentration of probabilities at the extremities suggests that the model is very confident in its predictions, which can be risky in the event of noise in the data or changes in conditions, impairing its ability to generalize.

[1] Cross-validation involves dividing the available data into multiple folds or subsets, using one of these folds as a validation set, and training the model on the remaining folds

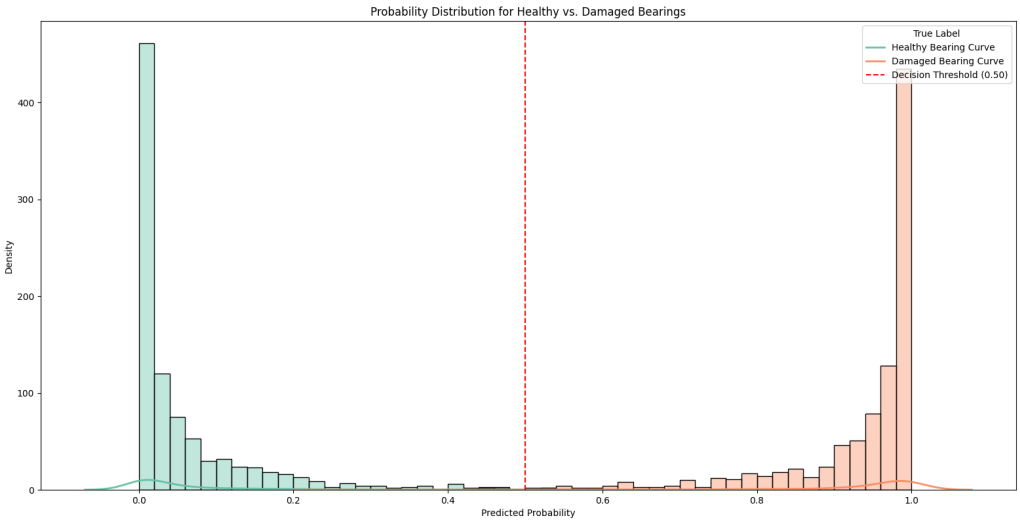



Figure 5.3: Probability distribution for the "RF + STFT" model

Despite the considerably longer pre-processing time, the RF combined with the MFCC pre-processing technique obtained more dispersion in the probability distribution, maintaining a more significant number of classifications at the extremities, as seen in Figure 5.4. This fact may make it more robust to noise and have a greater capacity for generalization.

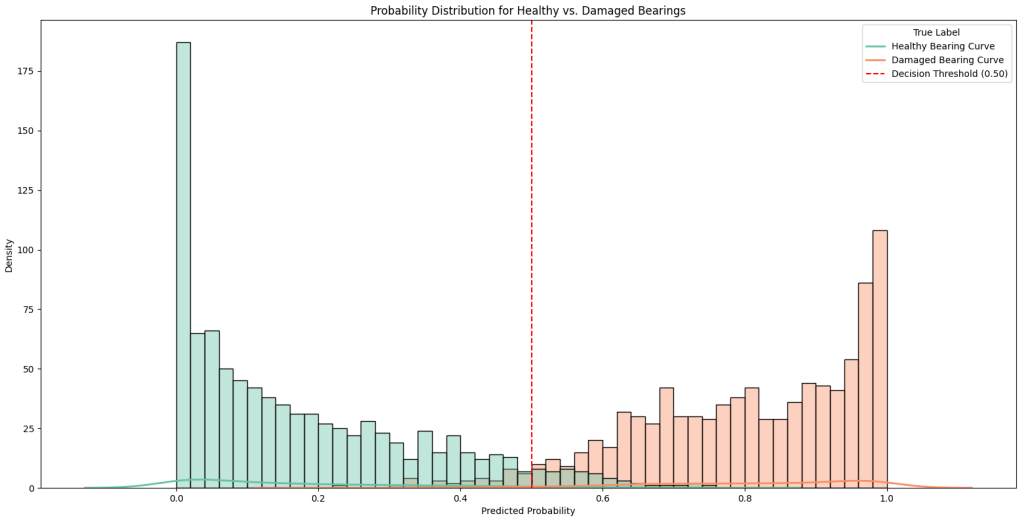



Figure 5.4: Probability distribution for the "RF + MFCC" model

The “RF+FFT” also obtained a solid result, reaching 0.97 of ROC-AUC. However, its probability distribution, shown in Figure 5.5, has a high concentration of classifications close to 0.5, which shows lower confidence than the other two models already analyzed, which was expected given its higher loss.

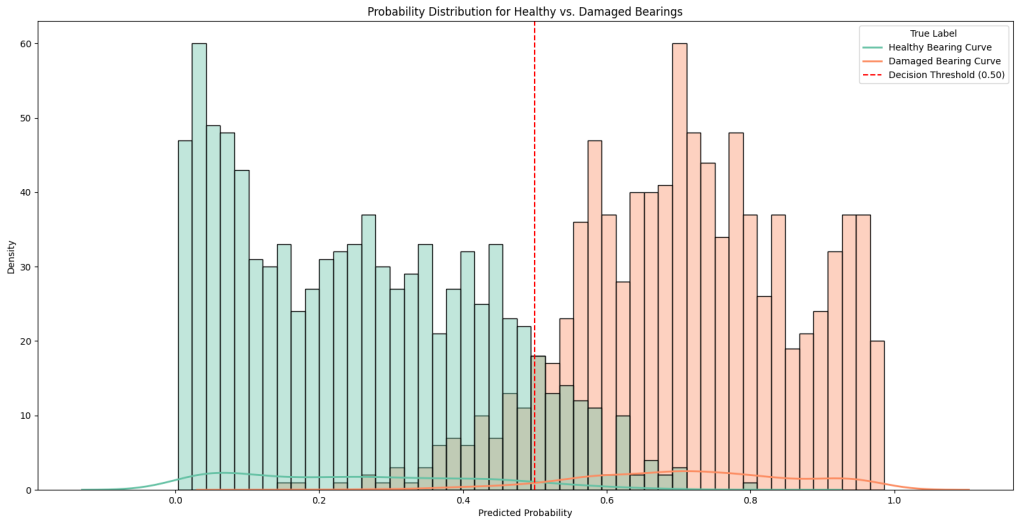



Figure 5.5: Probability distribution for the "RF + FFT" model

The “RF+SP” model achieved considerably worse results, with a considerable share of misclassification, as might be concluded through Figure 5.6

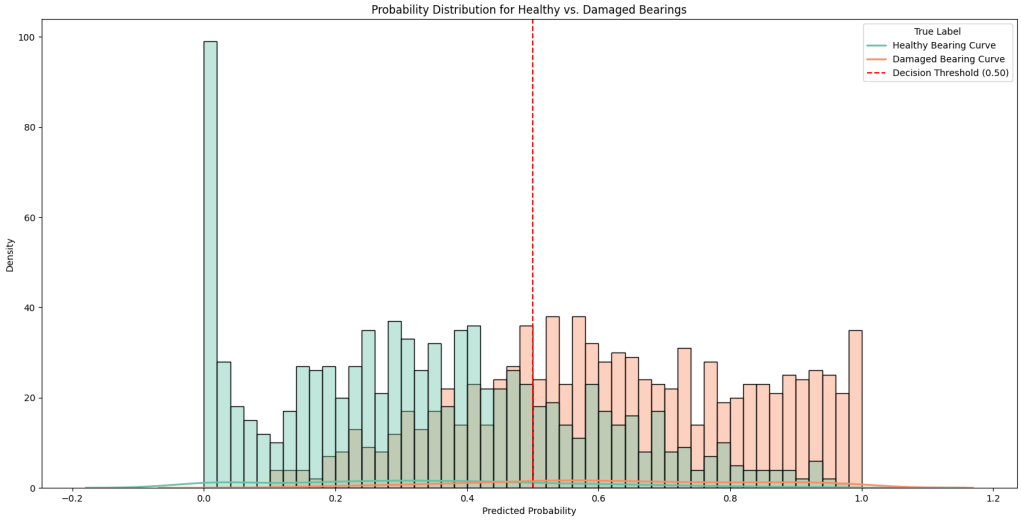



Figure 5.6: Probability distribution for the "RF + SP" model

We cross-validated all the combinations with 10 folds to corroborate the results, which were the same.

The loss dropped in all models along the 300 trees, but from 120 - 150, it dropped very slowly.

Every model was also tested with the accelerometer and gyroscope features, but the results were identical. These results and other combinations of pre-processing techniques results are available in our GitHub repository.

### 5.3.2 Gradient Boosted Decision Tree

The results for the GBDT under the same conditions as for the RF are exhibited in Table 5.3

Table 5.3: Results of the GBDT models test metrics

| Model | Accuracy | Precision | Recall | F1-score | ROC-AUC | Loss | $Pre-Proc_t$ (ms) | $Inf_t$ (ms) |
|---|---|---|---|---|---|---|---|---|
| GBDT + SP | 0.70 | 0.70 | 0.68 | 0.69 | 0.78 | 0.59 | 7.74 | 0.01 |
| GBDT + SP + NR | 0.69 | 0.71 | 0.63 | 0.67 | 0.77 | 0.58 | 46.08 | 0.01 |
| GBDT + FFT | 0.90 | 0.90 | 0.90 | 0.90 | 0.96 | 0.46 | 6.88 | 0.02 |
| GBDT + FFT + NR | 0.90 | 0.90 | 0.90 | 0.90 | 0.96 | 0.46 | 51.88 | 0.02 |
| GBDT + MFCC | 0.95 | 0.94 | 0.96 | 0.95 | 0.99 | 0.37 | 18.28 | 0.01 |
| GBDT + MFCC + NR | 0.95 | 0.94 | 0.96 | 0.95 | 0.99 | 0.37 | 72.27 | 0.01 |
| GBDT + STFT | 0.99 | 0.99 | 0.99 | 0.99 | 1.00 | 0.25 | 9.62 | 0.02 |
| GBDT + STFT + NR | 0.99 | 0.99 | 0.99 | 0.99 | 1.00 | 0.25 | 50.34 | 0.03 |

The conclusions are similar to the RF's, with almost identical results and probability distributions but with a shorter inference time. The losses are complicated to compare for the reasons explained in Section 4.3.4.1. Once again, we cross-validated all the models, and the results were the same as in the test.

### 5.3.3 Model's Tuning

In order to optimize the performance and generalization of our classification models, we have selected two models for tuning: one focused on maximizing performance and the other on enhancing generalization. For the performance-oriented model, we chose the "GBDT + STFT", due to its demonstrated effectiveness and efficiency in classification tasks. For the generalization-oriented model, we selected the "RF + MFCC", as its more diffuse probability distribution suggests a higher resistance to overfitting.

In this section, we will manually adjust both models. The pre-processing parameters, such as those related to STFT and MFCC, as well as the hyperparameters of the decision trees, will be systematically modified. We aim to strike an optimal balance between the models' potential for generalization and their evaluation metrics while minimizing inference time and pre-processing overhead.

#### 5.3.3.1 GBDT + STFT

Figure 5.7 shows that, although the loss looks like it could continue to decrease if the number of trees is increased, the accuracy is already very close to 1.0 from ten trees onwards.

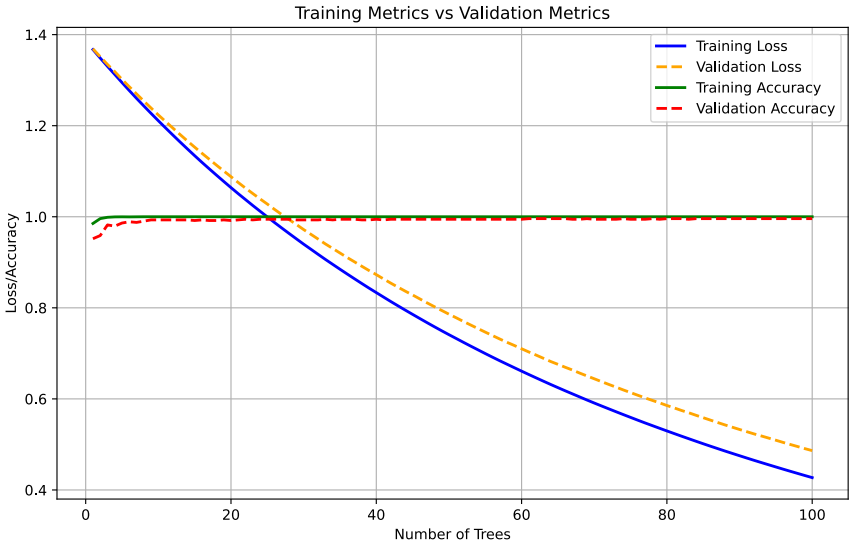



Figure 5.7: Evolution of accuracy and loss during the train of the "GBDT + STFT" model

Intending to lower the pre-processing time, we reduced the number of FFT points and the hop length to half, i.e., an FFT of size 1024 and a hop length of 256, thus reducing the number of features and the complexity of the model. Keeping the GBDT architecture, we achieved almost equal results with a third of the pre-processing time, as shown in Table 5.4. We decided to maintain the architecture, as our tests revealed that reducing the number of trees or the maximum number of nodes significantly increased the loss. Given that this model is focused on performance, any such adjustments compromised its effectiveness.

Table 5.4: Results of "GBDT + STFT" tuned model test metrics

| Model | Accuracy | Precision | Recall | F1-score | ROC-AUC | Loss | $Pre-Proc_t$ (ms) | $Inf_t$ (ms) |
|---|---|---|---|---|---|---|---|---|
| GBDT + STFT | 0.98 | 0.99 | 0.98 | 0.98 | 0.99 | 0.27 | 3.03 | 0.02 |

Figure 5.8 shows the probability distribution of the tuned model, demonstrating its effectiveness as a classifier even with fewer features.

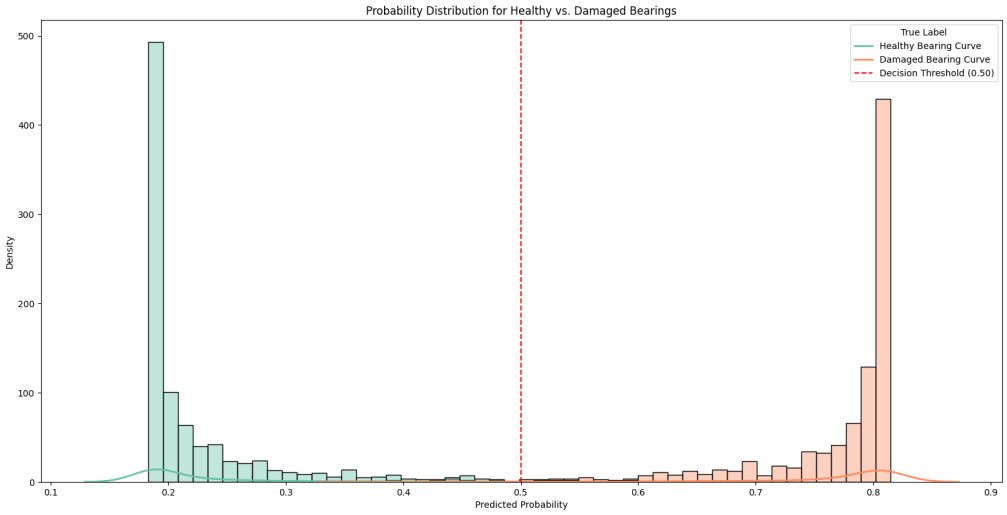



Figure 5.8: Probability distribution for the "GBDT + STFT" tuned model

We conducted cross-validation to confirm the outcomes, resulting in the same metrics. The confusion matrix of this evaluation can be observed in Figure 5.9.

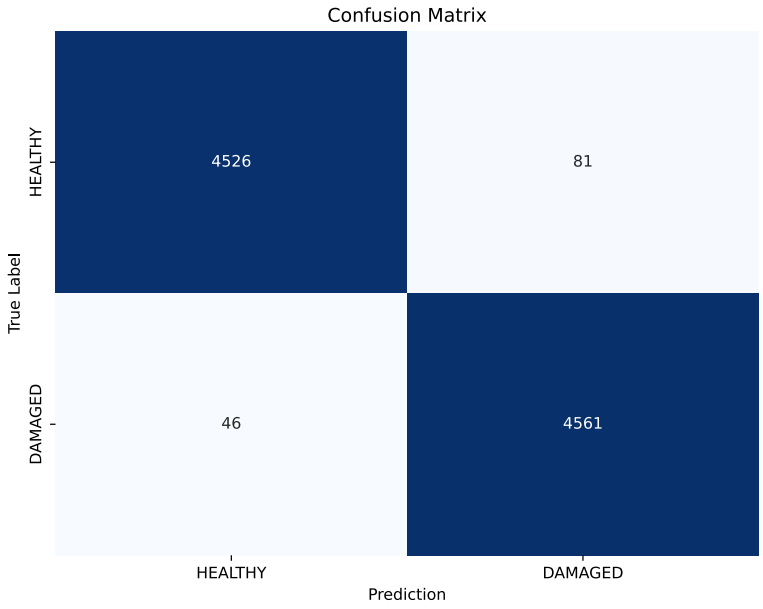



Figure 5.9: Confusion matrix of cross-validation evaluation for the "GBDT + STFT" tuned model

Decision trees offer multiple metrics to assess feature importance. In this context, we chose the sum of split scores because it reflects the frequency and effectiveness of a feature's contribution across all splits, delivering a more comprehensive evaluation of its significance. With the new parameters, the model received 848 features. Table 5.5 shows the most important ten variables.

Except for elements six, seven and eight, the remaining features are in the ultrasonic range, with frequencies above twice the audible frequency (> 40 kHz). The features names in the table have the following structure: “pre-processing technique_statistical property_frequency”. The information of how these features are calculated can be found in the Section 4.3.3.

Table 5.5: Top 10 STFT feature importance scores

| # | Feature | Score |
|---|---|---|
| 1 | `stft_std_70312.50_Hz` | 3545.71 |
| 2 | `stft_mean_79687.50_Hz` | 2756.65 |
| 3 | `stft_std_54187.50_Hz` | 2620.55 |
| 4 | `stft_std_73687.50_Hz` | 2371.84 |
| 5 | `stft_std_70125.00_Hz` | 1961.73 |
| 6 | `stft_mean_15000.00_Hz` | 1159.36 |
| 7 | `stft_mean_6375.00_Hz` | 904.07 |
| 8 | `stft_mean_6562.50_Hz` | 866.13 |
| 9 | `stft_std_55500.00_Hz` | 827.89 |
| 10 | `stft_std_45000.00_Hz` | 751.60 |

#### 5.3.3.2 RF + MFCC

We began by attempting to reduce the pre-processing time, a crucial factor for this method due to its computational intensity. To achieve this, we reduced the FFT size to 1024, the hop length to 256, the number of mel bands to 50, and the number of MFCCs to 20.

Although the Random Forest algorithm does not provide a direct way of visualizing the evolution of accuracy and loss during training, analysis of the logs indicated that after 200 trees, performance improvements became insignificant. Cutting the tree here while keeping the rest of the structure allowed it to maintain its balance between performance and generalization capacity, as well as more than halving the pre-processing time and slightly lowering the inference time, as shown in Table 5.6.

Table 5.6: Results of "RF + MFCC" tuned model test metrics

| Model | Accuracy | Precision | Recall | F1-score | ROC-AUC | Loss | $Pre-Proc_t$ (ms) | $Inf_t$ (ms) |
|---|---|---|---|---|---|---|---|---|
| RF + MFCC | 0.93 | 0.93 | 0.94 | 0.93 | 0.98 | 0.24 | 7.03 | 0.03 |

Figure 5.10 shows the probability distribution of the tuned model, demonstrating the balance between performance and possible generalization ability.

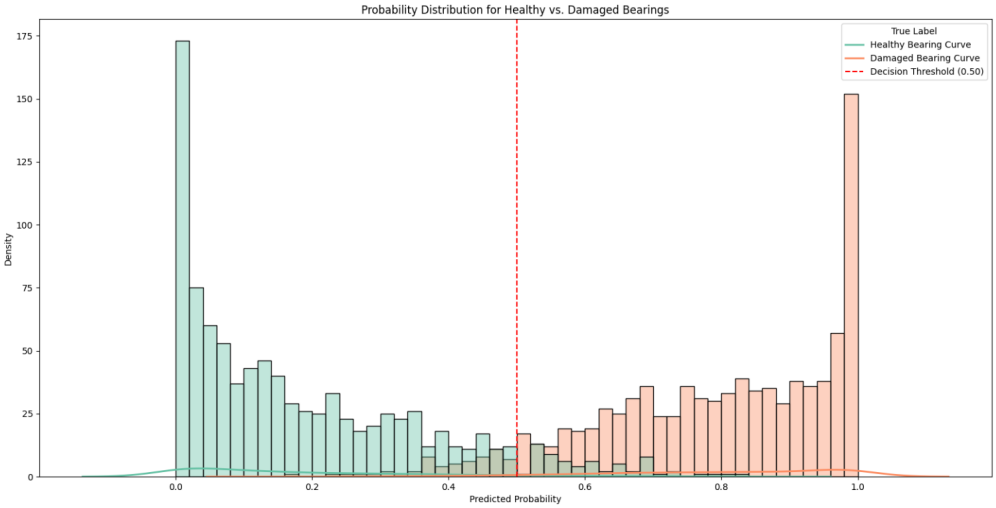



Figure 5.10: Probability distribution for the "RF + MFCC" tuned model

We performed cross-validation to confirm the outcomes, resulting in the same metrics. The confusion matrix of this evaluation can be seen in Figure 5.11.

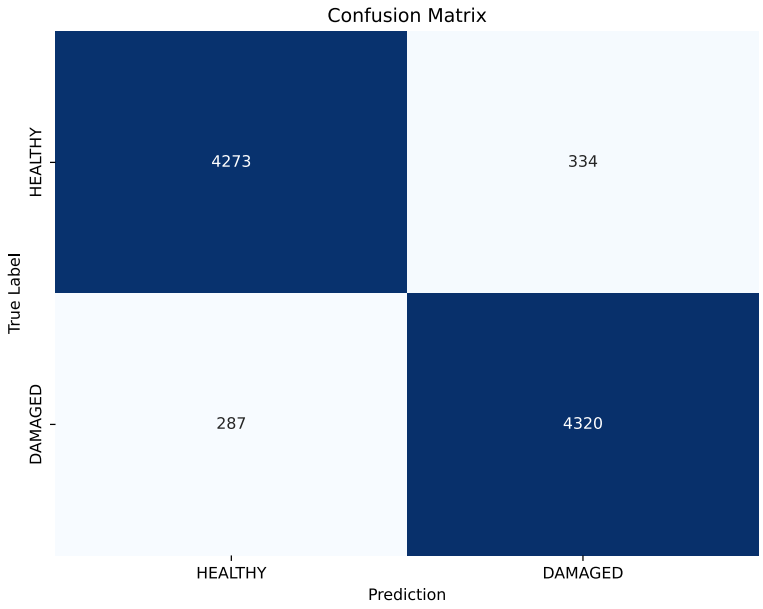



Figure 5.11: Confusion matrix of cross-validation evaluation for the "RF + MFCC" tuned model

From Table 5.7, we can conclude that the most important features in this dataset are predominantly the averaged MFCC coefficients. These coefficients, especially the lower-order ones, reflect broad spectral characteristics, which seem more critical for the task than the finer details. The features names in the table have the following structure: “pre-processing technique_statistical

property_number of MFCC Filter". The information of how these features are calculated can be found in the Section 4.3.3.

Table 5.7: Top 10 MFCC feature importance scores

| # | Feature | Score |
|---|---|---|
| 1 | `mfcc_avg_3` | 76820.69 |
| 2 | `mfcc_avg_2` | 65686.98 |
| 3 | `mfcc_avg_1` | 55219.57 |
| 4 | `mfcc_avg_4` | 54103.36 |
| 5 | `mfcc_avg_7` | 53747.00 |
| 6 | `mfcc_avg_0` | 46089.44 |
| 7 | `mfcc_std_0` | 41764.04 |
| 8 | `mfcc_avg_5` | 36179.74 |
| 9 | `mfcc_std_1` | 36177.51 |
| 10 | `mfcc_avg_6` | 32605.25 |

### 5.3.4 Model Comparison and Conclusion

Although NR seemed helpful in theory, it proved the contrary, probably because it eliminated noise and frequencies that were important for distinguishing the samples.

The tested time-frequency techniques proved effective with both models, validating the predefined hyperparameters.

The tuned models managed to maintain their characteristics with a considerable reduction in pre-processing time and model complexity, one being more performance-oriented and the other more generalization-oriented.

While "GBDT + STFT" took advantage of high frequencies to classify samples, "RF + MFCC" took advantage of lower coefficients, which extract properties from lower frequencies in the audible and near audible region since this pre-processing technique penalizes high frequencies more.

While the dataset is diverse and collected under controlled laboratory conditions, it remains limited to a non-industrial environment. Therefore, the results should be viewed as potential avenues for exploration rather than definitive conclusions. Further validation with real-world data is necessary to confirm the applicability of these findings in industrial settings. This topic will be deeply discussed in Section 6.1.

## 5.4 Supervised Models for Bearing Detection

In this section, the two tuned models from bearing classification will be retrained to evaluate their capacity to distinguish between noise and the presence of a bearing.

The Smote technique was used again to balance the dataset perfectly.

### 5.4.1 GBDT + STFT

The results of the bearing detection with the "GBDT + STFT" model were unrealistically perfect, as can be inferred from Table 5.8.

Table 5.8: Results of the “GBDT + STFT” detection model test metrics

| Model | Accuracy | Precision | Recall | F1-score | ROC-AUC | Loss | $Pre-Proc_t$ (ms) | $Inf_t$ (ms) |
|---|---|---|---|---|---|---|---|---|
| GBDT + STFT | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.01 | 3.03 | 0.02 |

From Figures 5.12 and 5.13, we can conclude that the model shows high confidence in its predictions. However, despite the validation loss closely mirroring the training loss, the model has likely overfitted to the noise dataset. This overfitting may reduce generalization performance, particularly when exposed to real-world or unseen data.

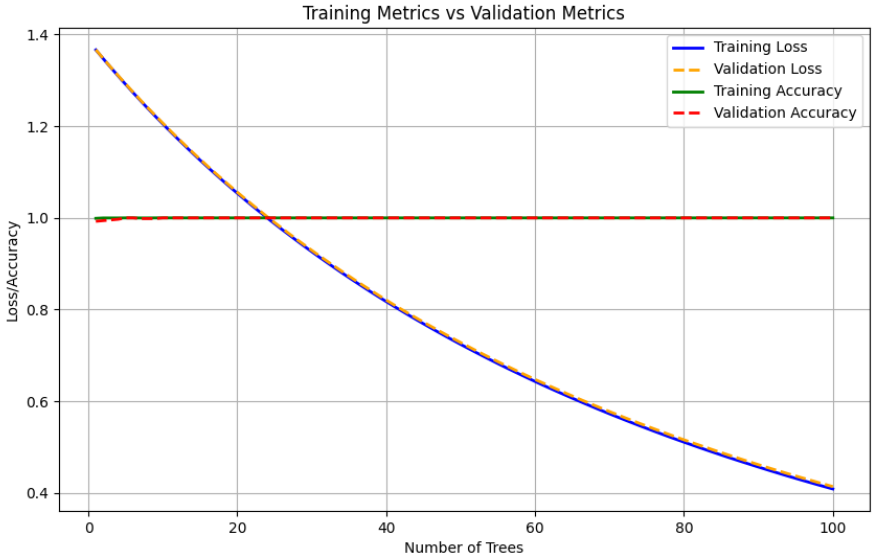



Figure 5.12: Accuracy and loss during the train of the "GBDT + STFT" bearing detection model

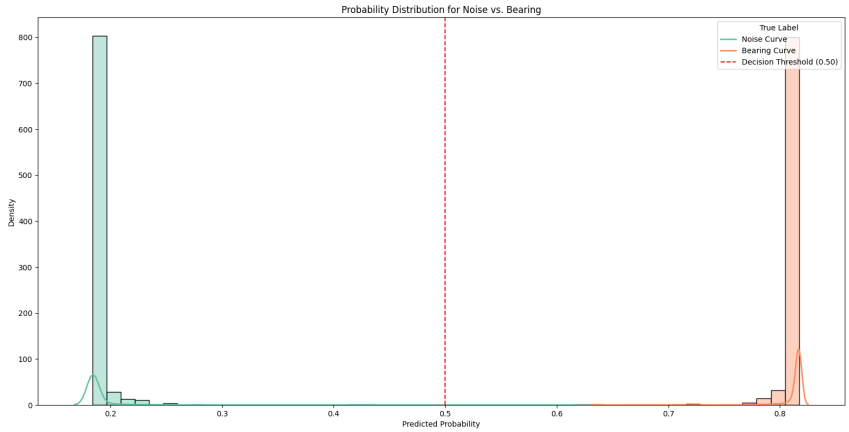



Figure 5.13: Probability distribution for the "GBDT + STFT" detection bearing model

The cross-validation confusion matrix can observed in Figure 5.14.

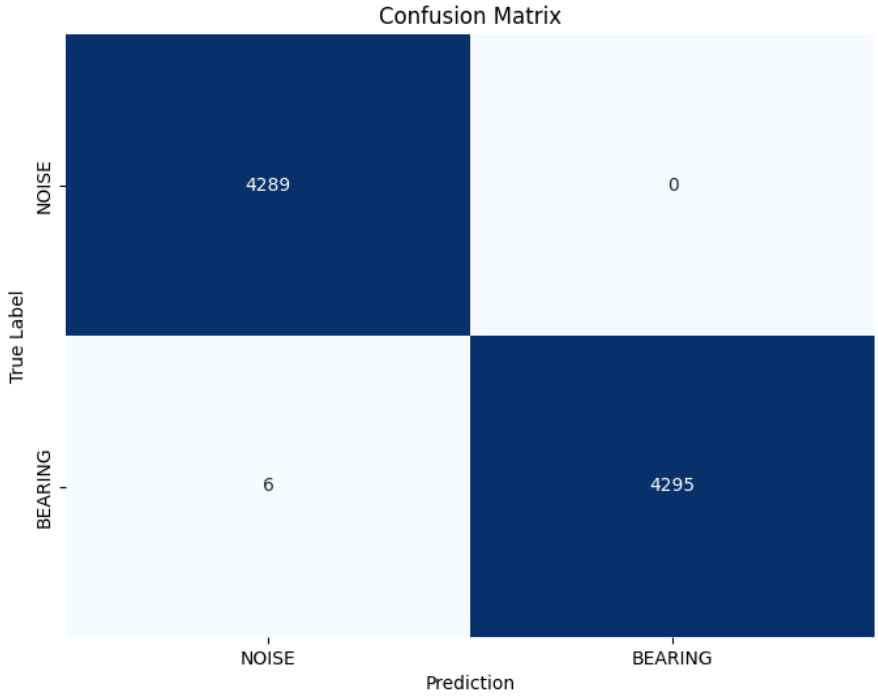



Figure 5.14: Accuracy and loss during the training for "GBDT + STFT" bearing detection model

### 5.4.2 RF + MFCC

The results for this model were the same, except for time metrics, as shown in Table 5.9.

Table 5.9: Results of the “RF + MFCC” detection model test metrics

| Model | Accuracy | Precision | Recall | F1-score | ROC-AUC | Loss | $Pre-Proc_t$ (ms) | $Inf_t$ (ms) |
|---|---|---|---|---|---|---|---|---|
| RF + MFCC | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.01 | 7.03 | 0.03 |

The probability distribution, visible in Figure 5.15, is also identical.

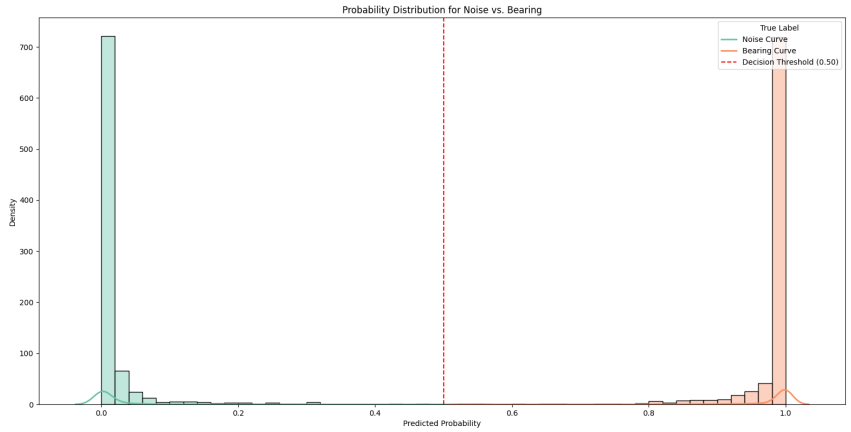



Figure 5.15: Probability distribution for the "RF + MFCC" detection bearing model

The cross-validation confusion matrix can observed in Figure 5.16.

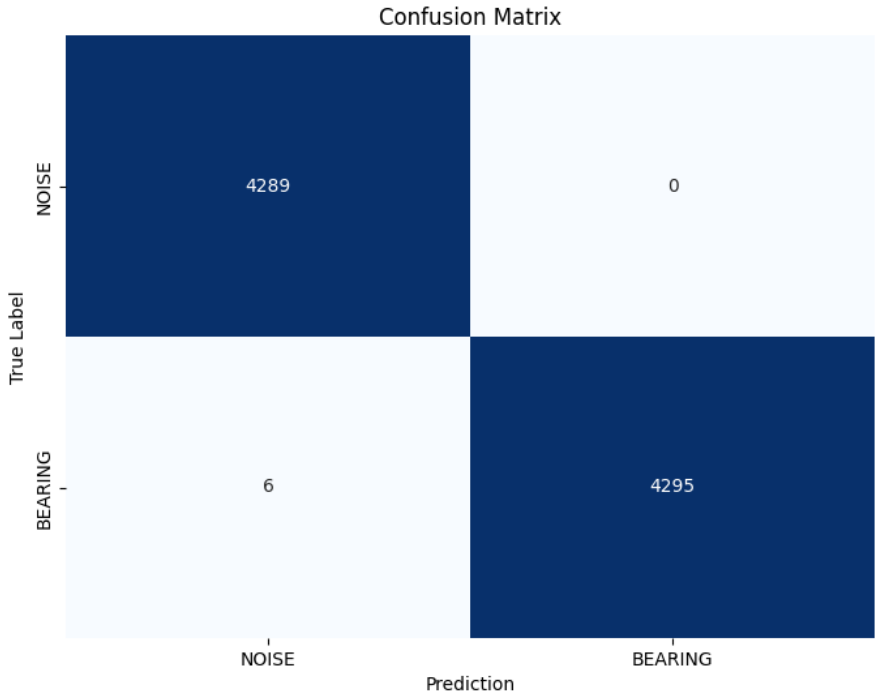



Figure 5.16: Accuracy and loss during the training for "RF + MFCC" bearing detection model

### 5.4.3 Model Comparison and Conclusion

Both models obtained almost equal results, and although the models appear to have overfitted the noise dataset, we opted not to perform hyperparameter tuning at this stage because the noise in the current dataset is highly unrealistic. Substantial adjustments to the hyperparameters would likely be necessary with a dataset more reflective of real-world industrial noise, making any current tuning irrelevant.

## 5.5 Unsupervised Methods

This section will analyze the results of unsupervised methods for detecting the presence of a bearing in the midst of noise and their effectiveness in classifying bearings.

### 5.5.1 Autoencoder

As Section 4.3.4.2 mentioned, Bayesian Optimization was performed on all the tested autoencoders to improve their results and optimize their hyperparameters. As this optimization is very time-consuming, we only did it with the STFT and MFCC preprocessing techniques, which had the best results in the bearing classification model. In addition, we tested both models with the accelerometer features to infer their impact on the unsupervised methods.

Since AEs do not directly classify a sample, the classification will be derived based on an error threshold established from a validation dataset. During testing, samples will be classified as "damaged"/"bearings" if the reconstruction error exceeds this threshold and "healthy"/"noise" if the error remains below the threshold.

The training was performed with 500 epochs and 256 batch size, with early stopping.

#### 5.5.1.1 Classification

The results of each best model from the Bayesian Optimization for unsupervised bearing classification with the optimized pre-processing parameters from Section 5.3.3 are shown in Table 5.10.

Table 5.10: Results of the AE for the bearing classification model

| Model | Accuracy | Precision | Recall | F1-score | ROC-AUC | Loss | $Pre-Proc_t$ (ms) | $Inf_t$ (ms) |
|---|---|---|---|---|---|---|---|---|
| AE + STFT | 0.55 | 0.55 | 0.53 | 0.54 | 0.55 | 0.50 | 3.03 | 6.04 |
| AE + MFCC | 0.58 | 0.58 | 0.56 | 0.57 | 0.58 | 0.59 | 7.03 | 1.83 |
| AE + STFT + ACCEL | 0.54 | 0.54 | 0.54 | 0.54 | 0.42 | 0.54 | 17.47 | 6.48 |
| AE + MFCC + ACCEL | 0.43 | 0.41 | 0.39 | 0.40 | 0.42 | 0.72 | 26.10 | 1.97 |

These results were performed with a threshold that minimizes the difference between precision and recall. In the end, to visually analyze the validation metrics, they were plotted along the threshold. Every metric can be disabled in the plot, and test metrics can also be abled, but the

threshold should never be optimized based on the test metrics to avoid overfitting with unseen data. An example of this plot can be seen in Figure 5.17.

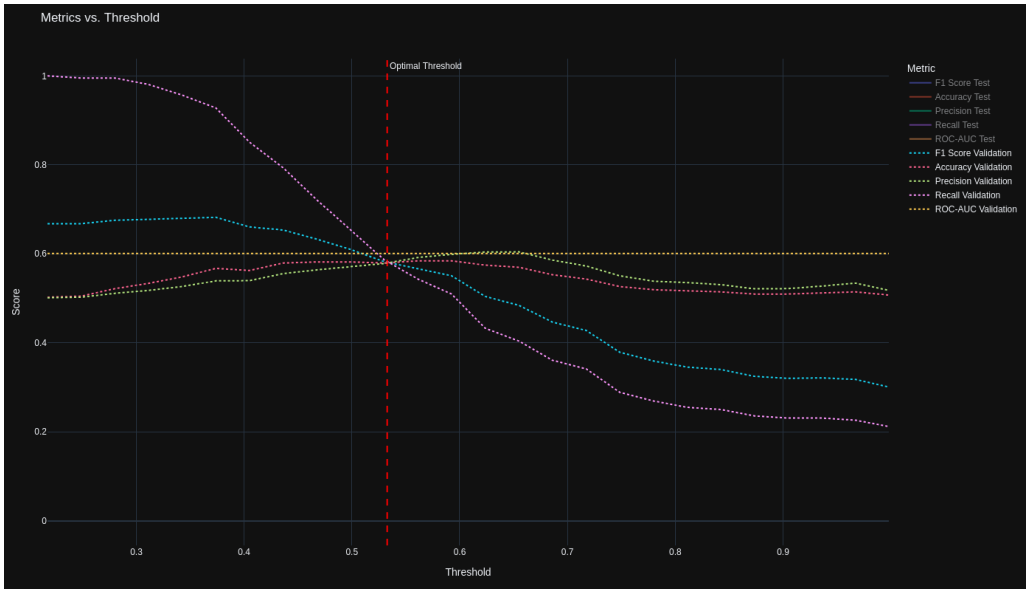



Figure 5.17: Validation metrics along the thresholds

The results of the four models were insufficient, being close to a random guess. Figure 5.18 shows the loss and validation loss evolution during the "AE + STFT" model training. The loss evolution for the "AE + STFT + ACCEL" model was identical.

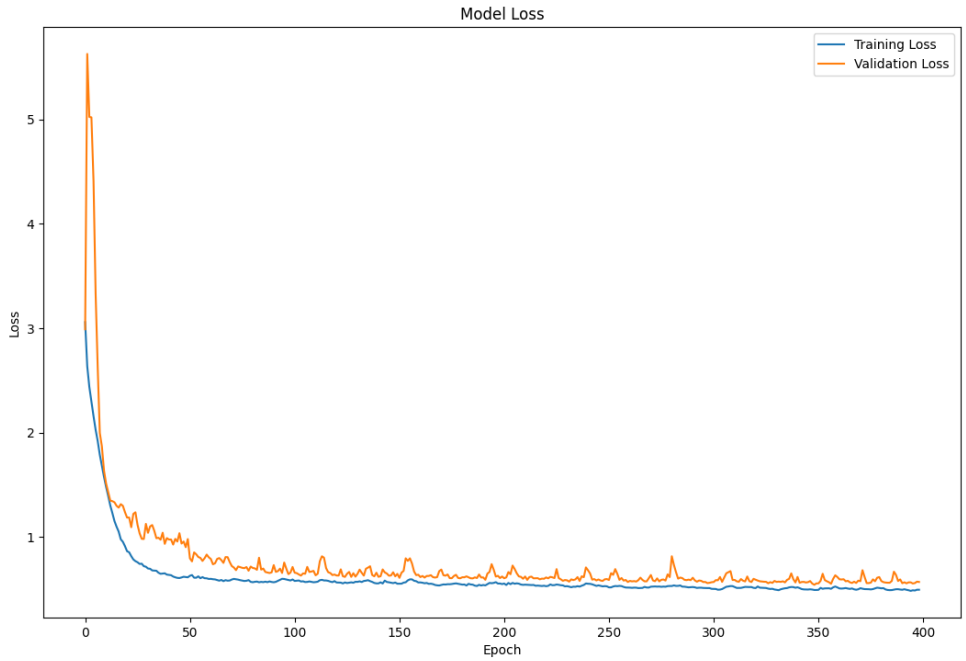



Figure 5.18: Loss evolution during the train of the "AE + STFT" classification model

The error distribution was also identical between the four models, which had a significant overlap between the two classes.

Figure 5.19, illustrates that for the "AE + MFCC" model.

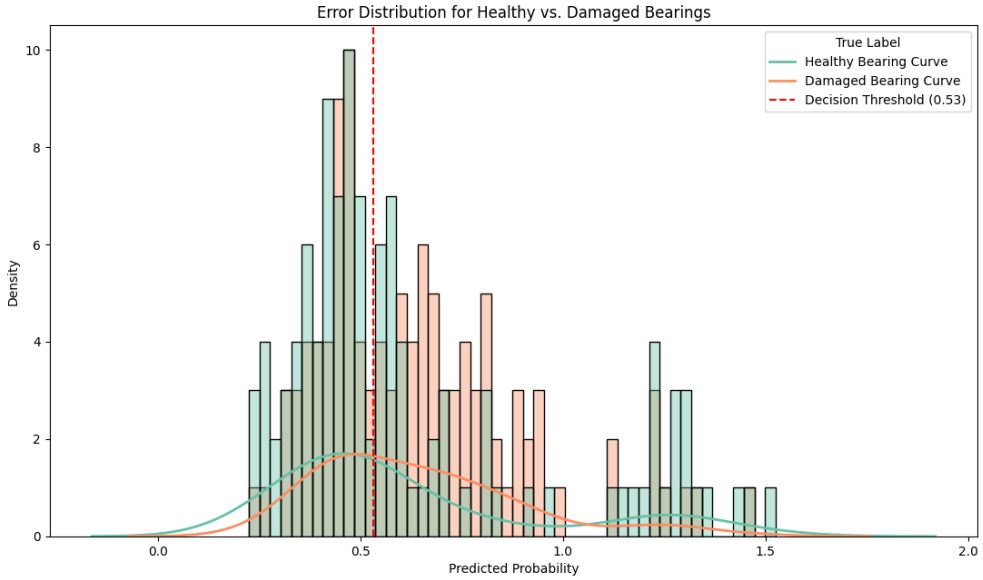



Figure 5.19: Error distribution for the "AE + MFCC" classification model

The models with the MFCC technique overfitted on the validation dataset, as can be stated in Figure 5.20. The plot for "AE + MFCC + ACCEL" was identical.

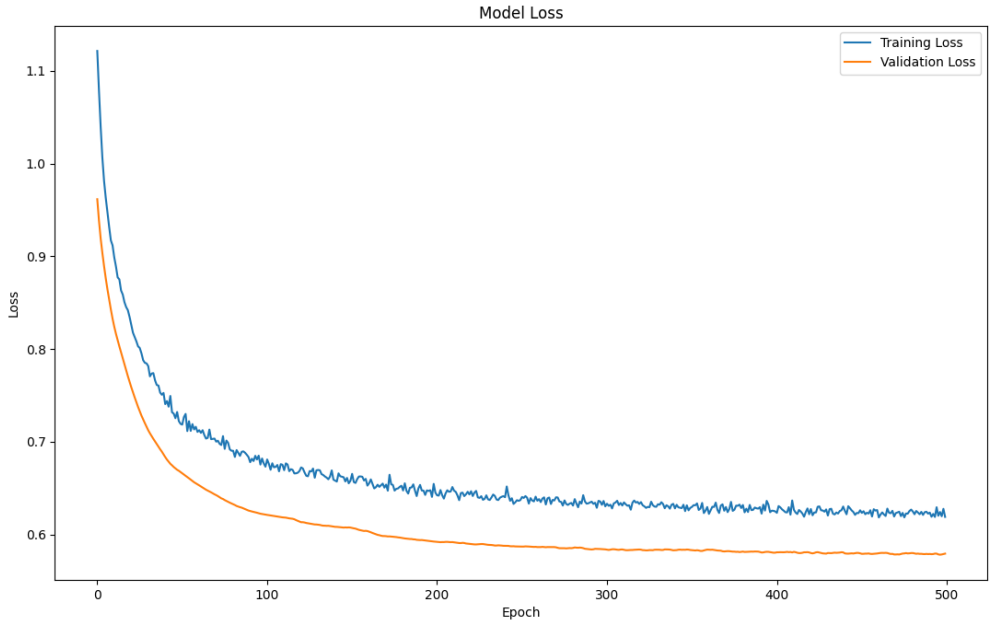



Figure 5.20: Loss evolution during the train for the "AE + MFCC" classification model

#### 5.5.1.2 Detection

The results of each best model from the Bayesian Optimization for unsupervised bearing detection with the optimized pre-processing parameters from Section 5.3.3 are shown in Table 5.11.

Table 5.11: Results of the AE for the bearing detection model

| Model | Accuracy | Precision | Recall | F1-score | ROC-AUC | Loss | $Pre-Proc_t$ (ms) | $Inf_t$ (ms) |
|---|---|---|---|---|---|---|---|---|
| AE + STFT | 0.75 | 0.78 | 0.78 | 0.73 | 0.75 | 0.32 | 3.03 | 6.33 |
| AE + MFCC | 0.82 | 0.82 | 0.82 | 0.82 | 0.82 | 0.38 | 7.03 | 0.69 |
| AE + STFT + ACCEL | 0.79 | 0.81 | 0.75 | 0.78 | 0.79 | 0.46 | 17.47 | 2.86 |
| AE + MFCC + ACCEL | 0.87 | 0.86 | 0.88 | 0.87 | 0.87 | 0.41 | 26.10 | 0.75 |

From the table, we can infer that the models with the MFCC pre-processing technique achieved relatively good results compared to the Autoencoders with STFT. Figure 5.21 illustrates the evolution of the loss during the train for the "AE + MFCC" model, showing overfitting in the validation set. The evolution loss for the "AE + MFCC + ACCEL" was almost equal to the "AE + MFCC".

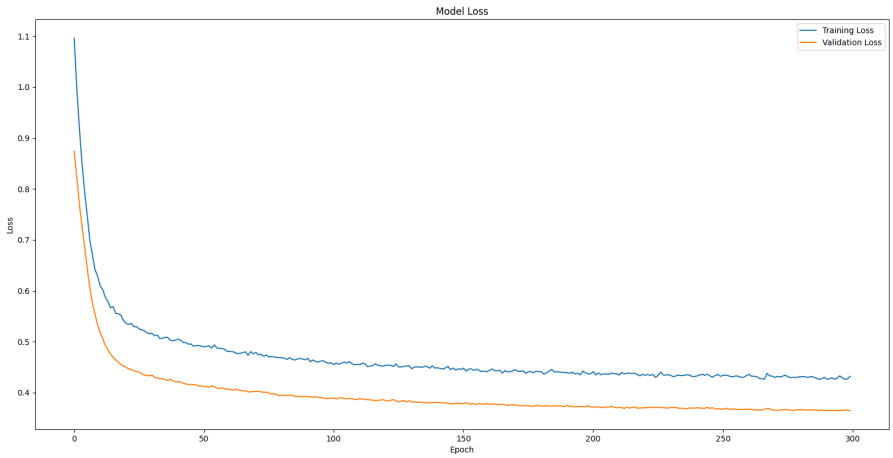



Figure 5.21: Loss evolution during the train for the "AE + MFCC" bearing detection model

Figure 5.22 shows that the “AE + MFCC” model for bearing detection could distinguish the two classes with little overlap compared to the unsupervised bearing classification models.

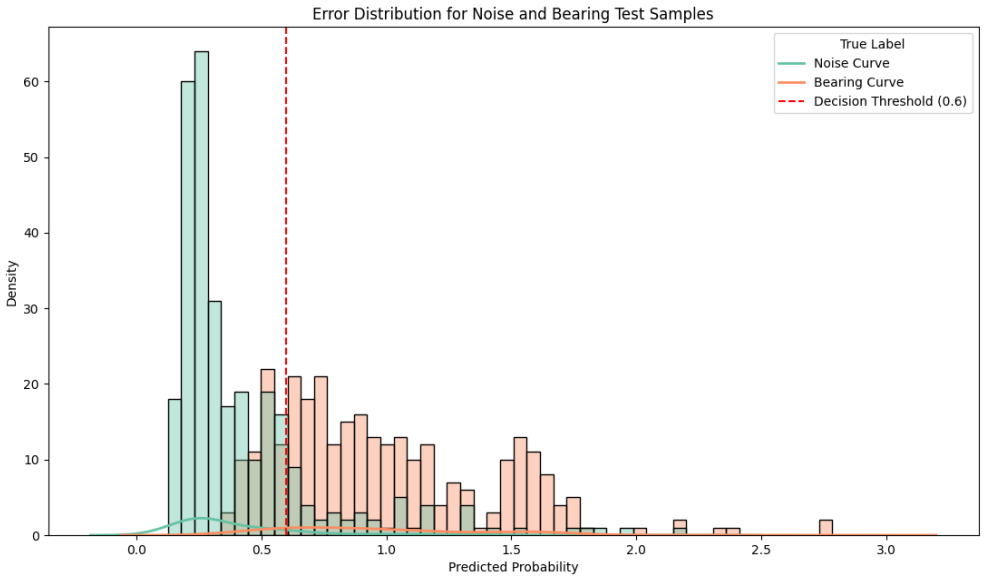



Figure 5.22: Error distribution for the "AE + MFCC" bearing detection model

The resultant confusion matrix for this same model can be observed in Figure 5.23.

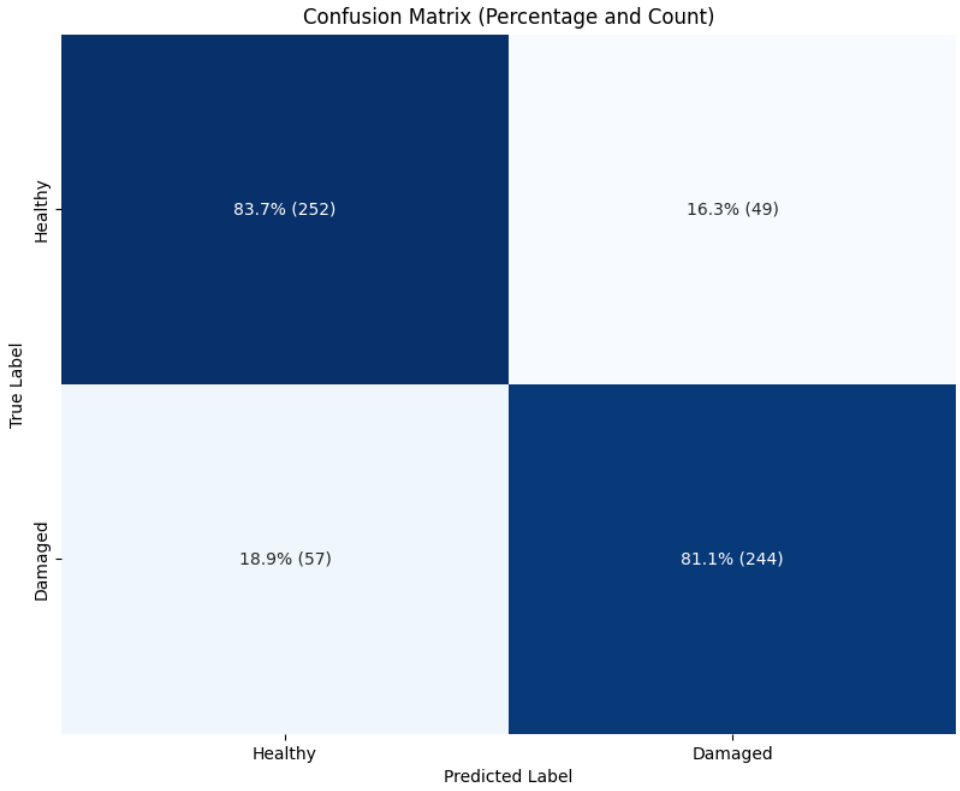



Figure 5.23: Confusion matrix for the "AE + MFCC" bearing detection model

Despite having lower metrics, the models with STFT preprocessing techniques showed more resistance to overfitting, as can be seen in Figure 5.24. The evolution shows some peaks, but these can probably be eliminated by increasing the regularization. The loss evolution is identical for the "AE + STFT" model.

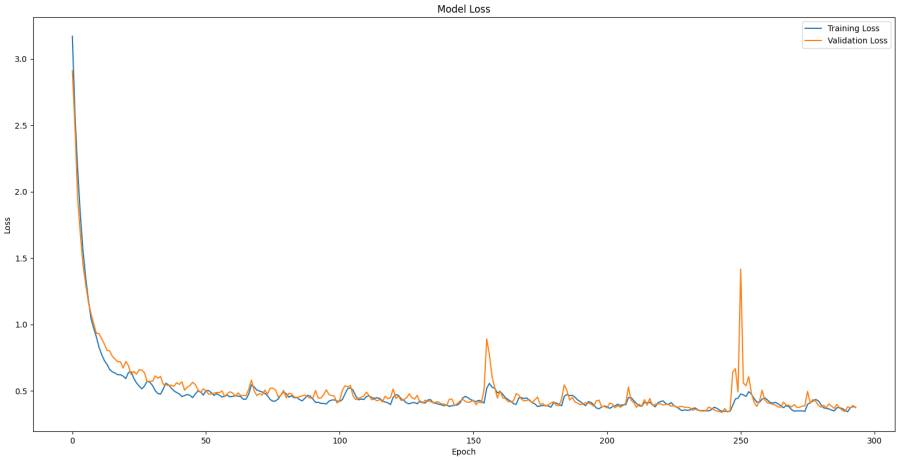



Figure 5.24: Loss evolution during the train for the "AE + STFT + ACCEL" bearing detection model

Figure 5.25 shows that some bearing samples had a very high error compared to the majority, which is also the case in the version of this model without the accelerometer data. The division between the two classes is also visible, although some overlap exists.

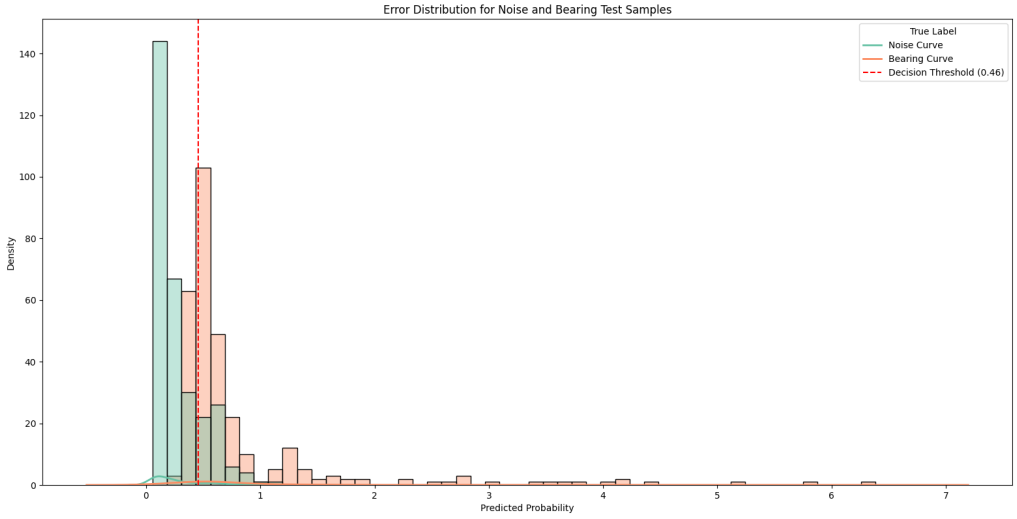



Figure 5.25: Error distribution for the "AE + STFT + ACCEL" bearing detection model

The resulting confusion matrix of this model is displayed in Figure 5.26.

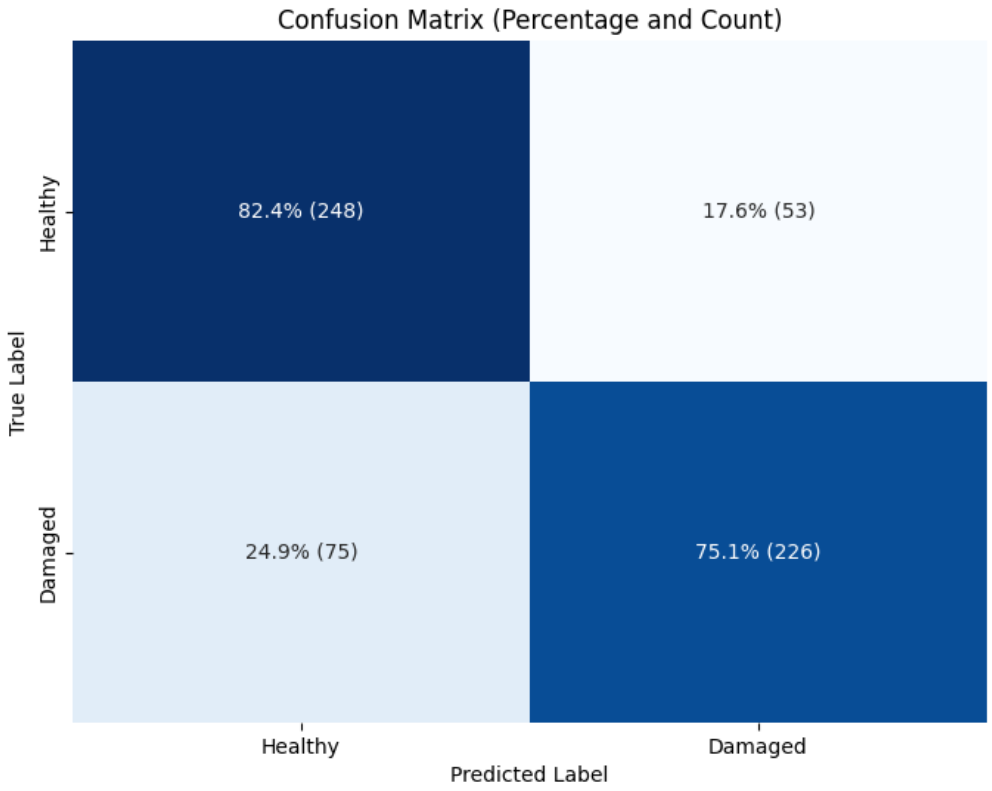



Figure 5.26: Confusion matrix for the "AE + STFT + ACCEL" bearing detection model

### 5.5.2 Model Comparison and Conclusion

The results of the bearing classification models were poor, probably because more data would have been needed for training and testing.

However, the bearing detection models obtained better results, namely “AE + MFCC”, which, despite the overfit, obtained good results in the test. The "AE + STFT + ACCEL" also obtained a result that may indicate a path to continue to explore, as it was more resistant to overfitting.

The results for all the tested models in this section are available in our GitHub in the "Unsupervised Results" folder.

## 5.6 Internal Sensor Testing

To commission the sensors responsible for detecting electrical faults, we collect baseline data during normal operation to establish thresholds for fault detection.

We conducted tests by burning electrical wires near the MQ-2 sensor and the GP2Y1010AU0F sensor to detect smoke. These sensors effectively identified the presence of smoke in close proximity.

The temperature sensor’s readings can vary significantly depending on the operating environment. However, data collected from the *GoMouse-x*, which utilized accelerometers placed on the

motors and temperature sensors integrated into the MPU6050, indicated that the temperature near the motors consistently measured about 2 degrees higher than that of the sensor located in the center of the AMR. Based on this observation, we established an arbitrary threshold of a 5-degree difference to indicate potential engine failure.

To enhance our measurements' accuracy, replacing one of the functioning motors with an identical faulty motor would be beneficial. This setup would allow us to more precisely assess any deviations and develop a model based on the accelerometer data. A similar approach should be employed to detect faults in the AMR's wheels, ensuring a comprehensive fault detection strategy across the entire system.

These thresholds, though based on preliminary observations, could benefit from further refinement with additional data, particularly from faulty motors, to improve fault detection accuracy.

In our scenario, if any of these thresholds were exceeded, an LED would start flashing; in an industrial situation, an alert would be sent to someone in charge.

## 5.7 Conclusion

The supervised methods outperformed unsupervised ones, as expected. However, autoencoder performance could improve with more realistic and larger datasets, especially when unavailable labeled data. For the unsupervised classification model, collecting more bearing data in healthy conditions would be necessary, ensuring sufficient variability to avoid overfitting in slightly different situations.

Among the supervised methods, the GBDT model combined with STFT and RF combined with MFCC showed promising results. The Yggdrasil decision tree library, although still relatively unexplored, has proved effective and efficient, with exceptionally fast inference times.

The time-frequency preprocessing techniques, STFT and MFCC, proved effective, as did using a high-frequency microphone. The STFT, in particular, was able to capture and take advantage of these high-frequency components.

# Chapter 6

# Conclusion and Future Work

This dissertation aimed to develop a sensory module to equip an AMR to predict failures in the surrounding environment and in the robot itself.

A study of related works was carried out to understand the best choice of sensors and methods. However, as mentioned in Section 3.3.3, no article was found where a robot, in this case, an AMR, is equipped with sensors to perform PdM of the surrounding environment.

Using a high-frequency microphone proved to be highly effective, allowing the GBDT model, combined with the STFT, to take advantage of the higher frequency range to detect bearing anomalies. Although MFCC is typically applied in models for human voice detection, it also proved helpful in this task, contributing to a model with generalizing capacity when combined with an RF.

The unsupervised classification models did not yield satisfactory results, likely due to data limitations. However, we anticipate that performance would improve with a larger dataset, as evidenced by successful applications in PdM literature, as stated in Section 3.2.3.1. It is essential to note that our chosen method, the Autoencoder, is a deep learning approach, which inherently requires more data for effective training compared to traditional machine learning methods. Expanding the dataset will be a focus of future work to enhance model performance.

As highlighted in Section 5.7, the developed models are not yet ready for real-world implementation. However, they provide promising directions for further exploration in this field.

We have achieved results that demonstrate high efficiency in detecting bearing faults in the environment with our sensory module and software. To validate the effectiveness of the methods, the dataset should be expanded with data from an industrial environment.

It is important to note that, although this case focused on bearings, these techniques can be readily adapted to monitor other components, such as valves or drills, expanding their applicability across various machinery.

To clarify how the work in this dissertation can be applied, let us consider an imaginary situation, though entirely plausible. Consider a machine responsible for drilling parts where the bearing gradually deteriorates, leading to imperfections and damage. Using an AMR with our sensor module, we can collect data when the machine is operating correctly. Then, we can explore ways to

extend the bearing's lifespan by inducing various failure conditions—such as damaged bearings or insufficient lubrication. Once the model is built, the robot will continuously pass collected samples through the classification model using data from the AMR's position and the machine. If the model detects multiple false positives, it could trigger an anomaly alert only after five consecutive anomaly classifications. However, a single detection could be enough if the system has high confidence. A similar approach can be applied using unsupervised models, which focus on recording data exclusively when the machine is in optimal condition. If it is not possible to have the position of the robot and machines, bearing detection models should be used.

Despite this, due to time constraints, the prediction of failures in the robot was not implemented past the laboratory prototype. Successive delays and setbacks with the arrival of the AMR did not help to meet the final objective, as did other external factors related to obtaining industrial data.

## 6.1 Future Work

The most important to continue and improve this project is to obtain data from an industrial setting. Collecting data from damaged bearings with healthy bearings enclosed would also be essential to studying their interference.

Adapting and expanding the models to more types of component failures, such as valves, drills, and gears, will be addressed to maximize the potential of this module.

Executing the PdM in the AMR, adding new sensors to this proposal (e.g., current sensors), and duplicating the existing sensors to record data from different points, i.e., adding more microphones and accelerometers at other points on the AMR, will also be part of the continuation of this project.

Continuing to test and validate anomaly detection algorithms using unsupervised ML methods, especially classification models, is also crucial to making this idea more adaptable to various scenarios.

Finally, implementing the tested methods and exploring the feasibility of running the models directly on a microcontroller — converting them into TinyML or streaming data in real-time to a small onboard computer — will be a key focus in the project's next phase. Notably, the Yggdrasil Decision Forests library offers a significant advantage by enabling the conversion of models to C++ code via a simple Python function, enhancing efficiency and speed. This capability will undoubtedly streamline the integration of the models into embedded systems.

# References


[1] U. Nations, "United nations sustainable development goals." [Online]. Available: https://sdgs.un.org/goals

[2] Y. Ran, X. Zhou, P. Lin, Y. Wen, and R. Deng, "A survey of predictive maintenance: Systems, purposes and approaches," 12 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1912.07383

[3] Z. Liu, N. Meyendorf, and N. Mrad, "The role of data fusion in predictive maintenance using digital twin," *AIP Conference Proceedings*, no. 1, 04 2018.

[4] D. S. Thomas and B. Weiss, "Maintenance costs and advanced maintenance techniques: Survey and analysis," *International Journal of Prognostics and Health Management*, vol. 12, no. 1, 2021.

[5] M. A. Naji, S. E. Filali, K. Aarika, E. H. Benlahmar, R. A. Abdelouhahid, and O. Debauche, "Machine learning algorithms for breast cancer prediction and diagnosis," *Procedia Computer Science*, vol. 191, pp. 487–492, 2021. [Online]. Available: https://doi.org/10.1016/j.procs.2021.07.062

[6] D. Verstraete, A. Ferrada, E. L. Droguett, V. Meruane, and M. Modarres, "Deep learning enabled fault diagnosis using time-frequency image analysis of rolling element bearings," *Shock and Vibration*, vol. 2017, p. 5067651, 2017. [Online]. Available: https://doi.org/10.1155/2017/5067651

[7] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *2008 International Conference on Prognostics and Health Management*, 2008, pp. 1–9.

[8] S. Cofre-Martel, E. Lopez Droguett, and M. Modarres, "Big machinery data preprocessing methodology for data-driven models in prognostics and health management," *Sensors*, vol. 21, no. 20, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/20/6841

[9] M. T. Heideman, D. H. Johnson, and C. S. Burrus, "Gauss and the history of the fast fourier transform," *Archive for history of exact sciences*, pp. 265–277, 1985.

[10] S. S. Patil and J. A. Gaikwad, "Vibration analysis of electrical rotating machines using fft: A method of predictive maintenance," in *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*, 2013, pp. 1–6.

[11] G. Strang, "Wavelets," *American Scientist*, vol. 82, no. 3, pp. 250–255, 1994. [Online]. Available: http://www.jstor.org/stable/29775194

[12] E. Sejdić, I. Djurović, and J. Jiang, "Time–frequency feature representation using energy concentration: An overview of recent advances," *Digital Signal Processing*, vol. 19, no. 1, pp. 153–183, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S105120040800002X

[13] M. M. Milani, P. E. Abas, and L. C. De Silva, "A critical review of heart sound signal segmentation algorithms," *Smart Health*, vol. 24, p. 100283, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352648322000186

[14] A. Bruns, "Fourier-, hilbert- and wavelet-based signal analysis: are they really different approaches?" *Journal of Neuroscience Methods*, vol. 137, no. 2, pp. 321–332, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165027004001098

[15] S. G. Krantz, *A panorama of harmonic analysis*. American Mathematical Soc., 2019, vol. 27.

[16] A. Bruns, "What is empirical mode decomposition," *Medium*, 2023. [Online]. Available: https://samanemami.medium.com/what-is-empirical-mode-decomposition-3ec89115db6b

[17] M. Feldman, "Hilbert transform in vibration analysis," *Mechanical systems and signal processing*, vol. 25, no. 3, pp. 735–802, 2011.

[18] N. E. Huang, *Hilbert-Huang transform and its applications*. World Scientific, 2014, vol. 16.

[19] H. N. Hoang, E. Granero-Fernandez, S. Yamada, S. Mori, H. Kagechika, Y. Medina-Gonzalez *et al.*, "Modulating biocatalytic activity toward sterically bulky substrates in co2-expanded biobased liquids by tuning the physicochemical properties," *ACS Sustainable Chemistry & Engineering*, vol. 5, no. 11, pp. 11 051–11 059, 2017.

[20] D. Mitrović, M. Zeppelzauer, and C. Breiteneder, "Features for content-based audio retrieval," in *Advances in computers*. Elsevier, 2010, vol. 78, pp. 71–150.

[21] H. Aldarmaki, A. Ullah, S. Ram, and N. Zaki, "Unsupervised automatic speech recognition: A review," *Speech Communication*, vol. 139, pp. 76–91, 2022.

[22] D. Mitrović, M. Zeppelzauer, and C. Breiteneder, "Features for content-based audio retrieval," in *Advances in computers*. Elsevier, 2010, vol. 78, pp. 71–150.

[23] M. Haarman, M. Mulders, and C. Vassiliadis, "Predictive maintenance 4.0: predict the unpredictable," *PwC and Mainnovation*, vol. 4, 2017.

[24] Z. Li, Y. Wang, and K.-S. Wang, "Intelligent predictive maintenance for fault diagnosis and prognosis in machine centers: Industry 4.0 scenario," *Advances in Manufacturing*, vol. 5, no. 4, pp. 377–387, 2017.

[25] A. Jaenal, J.-R. Ruiz-Sarmiento, and J. Gonzalez-Jimenez, "Machnet, a general deep learning architecture for predictive maintenance within the industry 4.0 paradigm," *Engineering Applications of Artificial Intelligence*, vol. 127, p. 107365, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095219762301549X

[26] M. Wang, "Ontology-based modelling of lifecycle underground utility information to support operation and maintenance," *Automation in construction*, vol. 132, p. 103933, 2021.

[27] T. Berredjem and M. Benidir, "Bearing faults diagnosis using fuzzy expert system relying on an improved range overlaps and similarity method," *Expert Syst. Appl.*, vol. 108, pp. 134–142, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:49619808

[28] V. Martínez-Martínez, F. J. Gomez-Gil, J. G. Gil, and R. Ruiz-Gonzalez, "An artificial neural network based expert system fitted with genetic algorithms for detecting the status of several rotary components in agro-industrial machines using a single vibration signal," *Expert Syst. Appl.*, vol. 42, pp. 6433–6441, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:17343422

[29] S. Liu, L. Xu, Q. Li, X. Zhao, and D. Li, "Fault diagnosis of water quality monitoring devices based on multiclass support vector machines and rule-based decision trees," *IEEE Access*, vol. 6, pp. 22 184–22 195, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:13716431

[30] S. Hong and Z. Zhou, "Remaining useful life prognosis of bearing based on gauss process regression," in *2012 5th International Conference on BioMedical Engineering and Informatics*. IEEE, Oct. 2012. [Online]. Available: http://dx.doi.org/10.1109/BMEI.2012.6513123

[31] J. Luo, M. Namburu, K. R. Pattipati, L. Qiao, and S. Chigusa, "Integrated model-based and data-driven diagnosis of automotive antilock braking systems," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 40, no. 2, p. 321–336, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1109/TSMCA.2009.2034481

[32] A. C. E. Gonçalves, "Sensor system for predictive maintenance in industrial environment," Master's thesis, Faculty of Engineering of the University of Porto, 2023. [Online]. Available: https://hdl.handle.net/10216/153610

[33] B. SAMANTA and K. AL-BALUSHI, "Artificial neural network based fault diagnostics of rolling element bearings using time-domain features," *Mechanical Systems and Signal Processing*, vol. 17, no. 2, p. 317–328, Mar. 2003. [Online]. Available: http://dx.doi.org/10.1006/MSSP.2001.1462

[34] T. Plapinger, "What is a decision tree?" Sep 2017. [Online]. Available: https://towardsdatascience.com/what-is-a-decision-tree-22975f00f3e1

[35] A. B. Nassif, M. A. Talib, Q. Nasir, and F. M. Dakalbab, "Machine learning for anomaly detection: A systematic review," *IEEE Access*, vol. 9, pp. 78 658–78 700, 2021.

[36] J. Wei, G. Dong, and Z. Chen, "Remaining useful life prediction and state of health diagnosis for lithium-ion batteries using particle filter and support vector regression," *IEEE Transactions on Industrial Electronics*, vol. 65, no. 7, p. 5634–5643, Jul. 2018. [Online]. Available: http://dx.doi.org/10.1109/TIE.2017.2782224

[37] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey. acm computing surveys," *vol*, vol. 41, p. 15, 2009.

[38] X. Wu, Y. Zhang, C. Cheng, and Z. Peng, "A hybrid classification autoencoder for semi-supervised fault diagnosis in rotating machinery," *Mechanical Systems and Signal Processing*, vol. 149, p. 107327, 2021.

[39] ——, "A hybrid classification autoencoder for semi-supervised fault diagnosis in rotating machinery," *Mechanical Systems and Signal Processing*, vol. 149, p. 107327, 2021.

[40] H. Qin, K. Xu, and L. Ren, "Rolling bearings fault diagnosis via 1d convolution networks," in *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*. IEEE, Jul. 2019. [Online]. Available: http://dx.doi.org/10.1109/SIPROCESS.2019.8868646

[41] B. Shubyn, D. Kostrzewa, P. Grzesik, P. Benecki, T. Maksymyuk, V. Sunderam *et al.*, "Federated learning for improved prediction of failures in autonomous guided vehicles," *Journal of Computational Science*, vol. 68, p. 101956, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877750323000169

[42] S. Gong, J. Lee, C.-g. Lee, J. Jeong, W. Choi, C. Kim *et al.*, "Cnn-lstm-ae based predictive maintenance using stft for rotating machinery," in *2023 IEEE/ACIS 8th International Conference on Big Data, Cloud Computing, and Data Science (BCD)*, 2023, pp. 80–85.

[43] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin *et al.*, "The history began from alexnet: A comprehensive survey on deep learning approaches," *arXiv preprint arXiv:1803.01164*, 2018.

[44] M. Kreuzer, A. Schmidt, and W. Kellermann, "Novel features for the detection of bearing faults in railway vehicles," in *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*, vol. 263, no. 3. Institute of Noise Control Engineering, 2021, pp. 3833–3844.

[45] S. Wang, T. Liu, K. Luo, and G. Yang, "Identification of engine faults based on acoustic emission signals using a 1dcnn-vit ensemble model," *Measurement Science and Technology*, vol. 34, no. 2, p. 024007, nov 2022. [Online]. Available: https://dx.doi.org/10.1088/1361-6501/aca041

[46] E. Di Fiore, A. Ferraro, A. Galli, V. Moscato, and G. Sperlì, "An anomalous sound detection methodology for predictive maintenance," *Expert Systems with Applications*, vol. 209, p. 118324, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422014518

[47] C.-J. Kuo, K.-C. Ting, Y.-C. Chen, D.-L. Yang, and H.-M. Chen, "Automatic machine status prediction in the era of industry 4.0: Case study of machines in a spring factory," *Journal of Systems Architecture*, vol. 81, pp. 44–53, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762117301133

[48] P. F. Suawa, A. Halbinger, M. Jongmanns, and M. Reichenbach, "Noise-robust machine learning models for predictive maintenance applications," *IEEE Sensors Journal*, vol. 23, no. 13, pp. 15 081–15 092, 2023.

[49] M. Jankauskas, A. Serackis, M. Šapurov, R. Pomarnacki, A. Baskys, V. K. Hyunh *et al.*, "Exploring the limits of early predictive maintenance in wind turbines applying an anomaly detection technique," *Sensors*, vol. 23, no. 12, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/12/5695

[50] M. Sohaib, C.-H. Kim, and J.-M. Kim, "A hybrid feature model and deep-learning-based bearing fault diagnosis," *Sensors*, vol. 17, no. 12, 2017. [Online]. Available: https://www.mdpi.com/1424-8220/17/12/2876

[51] P. N. Mueller, L. Woelfl, and S. Can, "Bridging the gap between ai and the industry — a study on bearing fault detection in pmsm-driven systems using cnn and inverter measurement," *Engineering Applications of Artificial Intelligence*, vol. 126, p. 106834, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0952197623010187

[52] S. Gawde, S. Patil, S. Kumar, P. Kamat, and K. Kotecha, "An explainable predictive maintenance strategy for multi-fault diagnosis of rotating machines using multi-sensor data fusion," *Decision Analytics Journal*, vol. 10, p. 100425, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2772662224000298

[53] J. Pacheco-Chérrez, J. A. Fortoul-Díaz, F. Cortés-Santacruz, L. María Aloso-Valerdi, and D. I. Ibarra-Zarate, "Bearing fault detection with vibration and acoustic signals: Comparison among different machine leaning classification methods," *Engineering Failure Analysis*, vol. 139, p. 106515, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1350630722004897

[54] C. Murphy, "Choosing the most suitable predictive maintenance sensor," *Analog Devices, Inc*, 2020.

[55] *Datasheet ADXL354/ADXL355*, Analog Devices, 2016, available at https://www.analog.com/media/en/technical-documentation/data-sheets/adxl354_adxl355.pdf.

[56] *MPU-6000 and MPU-6050 Product Specification Revision 3.4*, 3rd ed., TInvenSense, 8 2013, available at https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.

[57] *MMA8451Q, 3-axis, 14-bit/8-bit digital accelerometer*, 10th ed., NXP Semiconductors, 2 2017, available at https://www.nxp.com/docs/en/data-sheet/MMA8451Q.pdf.

[58] siddacious, "Adafruit mpu6050," Nov 2023. [Online]. Available: https://github.com/adafruit/Adafruit_MPU6050?tab=readme-ov-file

[59] *I2S Output Digital Microphone*, Knowles Electronics, 7 2015. [Online]. Available: https://www.ti.com/lit/ds/symlink/tmp235.pdf?ts=1712800949460

[60] A. Industries, "Adafruit silicon mems microphone breakout - spw2430," Dec 2023. [Online]. Available: https://www.adafruit.com/product/2716

[61] ——, Aug 2023. [Online]. Available: https://www.adafruit.com/product/3492

[62] ——, "Adafruit tmp235 - plug-and-play stemma analog temperature sensor," Nov 2022. [Online]. Available: https://www.adafruit.com/product/4686

[63] *TMP23x Low-Power, High-Accuracy Analog Output Temperature Sensors*, Texas Instruments, 5 2019. [Online]. Available: https://www.ti.com/lit/ds/symlink/tmp235.pdf?ts=1712800949460

[64] siddacious, "Adafruit sensor: Common sensor library," Nov 2023. [Online]. Available: https://github.com/adafruit/Adafruit_Sensor

[65] M. A. Califa Urquiza, G. Contreras Contreras, and Y. R. Carrillo Amado, "miguel5612/mqsensorslib: Arduino preview v1.03," Sep. 2019. [Online]. Available: https://doi.org/10.5281/zenodo.3384301

[66] DFRobot, "Gravity: Analog gas sensor (mq2)." [Online]. Available: https://www.dfrobot.com/product-681.html

[67] *Sharp GP2Y1010AU0F*, SHARP Corporation, 12 2006. [Online]. Available: https://dfimg.dfrobot.com/enshop/image/data/SEN0144/gp2y1010au_e.pdf

[68] Luciansabo, "Gp2ydustsensor," Jul 2019. [Online]. Available: https://github.com/luciansabo/GP2YDustSensor

[69] P. Stoffregen, "Teensy 4.1 development board." [Online]. Available: https://www.pjrc.com/store/teensy41.html

[70] ——, "Teensy audio library," 11 2015. [Online]. Available: https://github.com/PaulStoffregen/Audio

[71] alfa95139, "Alfa95139/pt8211-with-teensy-4.1: Example from frankb (i2sfrequency) to change the sampling frequency of teensy 4.1." [Online]. Available: https://github.com/alfa95139/pt8211-with-Teensy-4.1/

[72] T. Sainburg, M. Thielk, and T. Q. Gentner, "Finding, visualizing, and quantifying latent structure across diverse animal vocal repertoires," *PLoS computational biology*, vol. 16, no. 10, p. e1008228, 2020.

[73] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[74] M. Guillame-Bert, S. Bruch, R. Stotz, and J. Pfeifer, "Yggdrasil decision forests: A fast and extensible decision forests library," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2023, Long Beach, CA, USA, August 6-10, 2023*, 2023, pp. 4068–4077. [Online]. Available: https://doi.org/10.1145/3580305.3599933

[75] H. Guan, M. R. Dwarampudi, V. Gunda, H. Min, L. Yu, and J. Zou, "A comparison of decision forest inference platforms from a database perspective," 2023. [Online]. Available: https://arxiv.org/abs/2302.04430

[76] Ruchi, "Gradient boosted decision tree - clearly explained," Dec 2023. [Online]. Available: https://medium.com/@ruchi.awasthi63/gradient-boosted-decision-tree-clearly-explained-bd1d8c7d9923

[77] M. Guillame-Bert, S. Bruch, R. Stotz, and J. Pfeifer, "Yggdrasil decision forests," 2023. [Online]. Available: https://ydf.readthedocs.io/en/stable/

[78] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[79] J. Riebesell, "Autoencoder," Apr 2022. [Online]. Available: https://tikz.net/autoencoder/

[80] J. Miao and W. Zhu, "Precision–recall curve (prc) classification trees," *Evolutionary intelligence*, vol. 15, no. 3, pp. 1545–1569, 2022.

# Appendix A

# Auxiliary Tables

Table A.1: Summary of Key Terms

| Term | Definition |
|---|---|
| **Condition Monitoring** | Continuous assessment and tracking of parameters to evaluate system or equipment health, aiming to detect deviations from normal conditions for Predictive Maintenance. |
| **Remaining Useful Life** | Duration remaining from the present moment until the end of an asset or system's productive operational span, estimated by analyzing historical and condition monitoring data. |
| **Fault Detection** | Determination of faults in the monitored system, detecting deviations from standard conditions in processes, actuators, and sensors. |
| **Fault Diagnostics** | Identification and understanding of the root cause behind a detected fault or anomaly, involving comprehensive analysis of available data. |
| **Fault Prognostics** | Predicting future behavior of a system based on its current state and historical data, estimating time remaining before a specific failure, considering factors like wear and tear. |
| **Anomaly Detection** | Identifying deviations or outliers from normal patterns within a dataset or system behavior; commonly used for industrial damage detection, with a focus on moving parts analysis. |

Table A.2: Comparison of Model-Based, Data-Driven, and Hybrid PdM Approaches, based on [2, 23, 32, 25]

| **Feature** | **Model-Based** | **Data-Driven** | **Hybrid** |
|---|---|---|---|
| **Basis** | Analytical (Physics-of-Failure) | Empirical models from historical data | Combination of both |
| **Knowledge Requirement** | Extensive understanding of machinery physics and evolution | No explicit knowledge of the physical system required | Requires consideration of physical models, challenging without sufficient prior information |
| **Accuracy** | Theoretically more accurate, if precise understanding is available | Relies on historical data, may be influenced by data quality | Aims to leverage strengths of both, but accuracy may depend on the quality of physical models |
| **Interrelation of Components** | Tricky to define for complex systems with multiple components | Not explicitly considered relies on learning from data | Requires consideration of physical models for defining interrelations |
| **Implementation Challenges** | Accessibility of extensive prior knowledge | Potential sensitivity to data quality and representation | Challenging design, limited to scenarios with sufficient prior information |
| **Focus Area** | Machinery physics and evolution | Historical data from monitored machinery | Combination of physical models and historical data |
| **Research Trend** | Traditional but facing challenges | Growing attention due to reliability and independence from explicit physical knowledge | Emerging, addressing limitations of both model-based and data-driven approaches |
| **Examples** | Physics-of-Failure models | Machine Learning and Deep Learning models | Techniques combining physical models with data-driven methods |

Table A.3: Teensy 4.1 Features

| **Feature** | **Description** |
|---|---|
| Processor | ARM Cortex-M7 at 600 MHz |
| Floating Point Unit | 64-bit and 32-bit float point math unit |
| Memory | 7936K Flash, 1024K RAM (512K tightly coupled), 4K EEPROM (emulated) |
| Memory Expansion | QSPI memory expansion, locations for 2 extra RAM or Flash chips |
| USB | USB device 480 Mbit/sec & USB host 480 Mbit/sec |
| Digital I/O | 55 digital input/output pins, 35 PWM output pins |
| Analog Inputs | 18 analog input pins |
| Serial Communication | 8 serial, 3 SPI, 3 I2C ports |
| Audio | 2 I2S/TDM and 1 S/PDIF digital audio port |
| CAN Bus | 3 CAN Bus (1 with CAN FD) |
| SD Card | 1 SDIO (4 bit) native SD Card port |
| Ethernet | Ethernet 10/100 Mbit with DP83825 PHY [1] |
| DMA Channels | 32 general purpose DMA channels |
| Cryptography | Cryptographic Acceleration & Random Number Generator |
| RTC | RTC for date/time |
| Programmable I/O | Programmable FlexIO |
| Graphics Processing | Pixel Processing Pipeline |
| Peripheral Control | Peripheral cross triggering |
| Power Management | Power On/Off management |

[1] https://www.pjrc.com/store/ethernet_kit.html

Table A.4: Statistical Properties Definitions

| **Statistic** | **Description** |
|---|---|
| **Mean** $(\mu)$ | The average value of all elements in the data. |
| **Standard Deviation** $(\sigma)$ | A measure of how spread out the data is from the mean. |
| **Root Mean Square** | The square root of the average of squared elements. Often used to measure the magnitude of a fluctuating signal. |
| **Kurtosis** | It measures the "tailedness" of the distribution compared to a normal distribution. Not reliable for distributions with very low variance. |
| **Skew** | It measures the asymmetry of the distribution relative to a normal distribution. Not reliable for distributions with very low variance. |

Table A.5: Laboratory Desktop Specifications

| **Component** | **Specification** |
|---|---|
| **Processor** | AMD Ryzen 9 7950X3D 16-Cores 32-Threads |
| **Motherboard** | TUF GAMING X670E-PLUS |
| **Memory (RAM)** | 32 GB |
| **Graphics Card** | RTX 4090 24 GB |