AI Based Industrial Motor Fault Prediction

A BS Final Year Project by

Badar E Alam
826-FET/BSEE/F20

Tayyab Ur Rehman
877-FET/BSEE/F20

Touseef Ameer
878-FET/BSEE/F20

Supervised by
Engr. Dr. Tayyab Ali

Co-supervised by
Engr. Dr. Aleem Khaliq

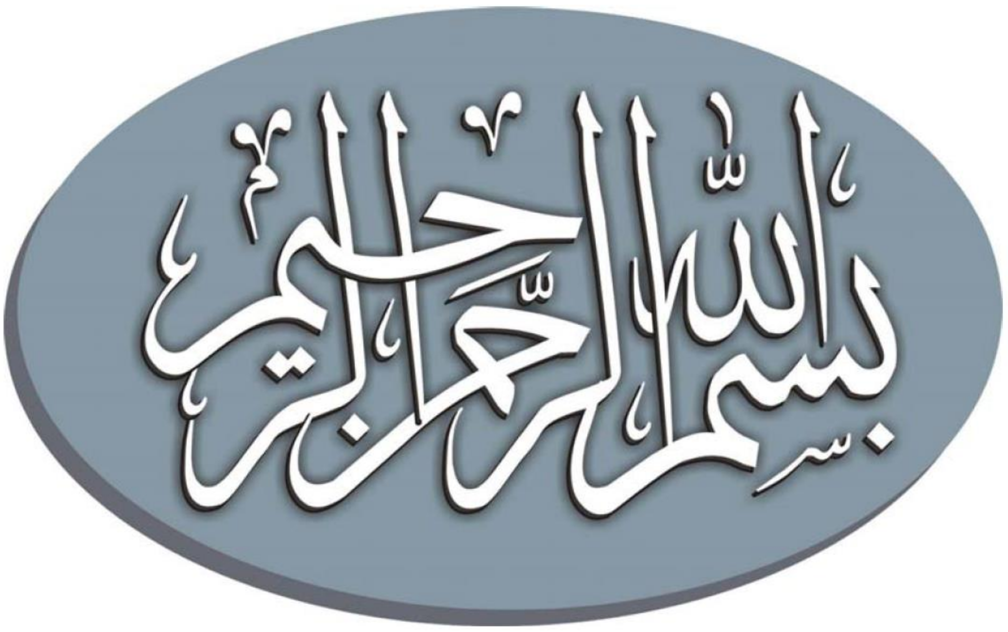
Department of Electrical and Computer Engineering
Faculty of Engineering and Technology
International Islamic University, Islamabad

July 2024

Certificate of Approval

It is certified that we have checked the project presented and demonstrated by **Badar E Alam 826-FET/BSEE/F20, Tayyab Ur Rehman 877-FET/BSEE/F20, Touseef Ameer 878-FET/BSEE/F20** and approved it.

External Examiner	*Internal Examiner*
Engr. Fahad Munir	**Engr. Dr. Athar Waseem**
Lecturer	Lecturer
Supervisor	*Co-supervisor*
Engr. Dr. Tayyab Ali	**Engr. Dr. Aleem Khaliq**
Lab Engineer	Lab Engineer



In the name of Allah (SWT), the most beneficent and the most merciful

A BS Final Year Project submitted to the

Department of Electrical and Computer Engineering

International Islamic University, Islamabad

In partial fulfillment of the requirements

For the award of the degree of

Bachelor of Science in Electrical Engineering

Declaration

We hereby declare that this work, neither as a whole nor as a part thereof has been copied from any source. No portion of the work presented in this report has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning. We further declare that the referred text is properly cited in the references.

Badar E Alam
826-FET/BSEE/F20

Tayyab Ur Rehman
877-FET/BSEE/F20

Touseef Ameer
878-FET/BSEE/F20

Acknowledgments

This BS thesis in Electrical Engineering has been conducted at Department of Electrical and Computer Engineering, Faculty of Engineering and Technology, International Islamic University, as part of the degree program. We would like to thank Engr. Dr. Tayyab Ali for providing us with an opportunity to work on this project, under his supervision and guidance throughout the project. We would also like to thank Engr. Dr. Aleem Khaliq and Engr. Aasar Ahmad for his help, efforts and dedicated support throughout the project. Further we are particularly thankful to Almighty Allah and grateful to our parents, brothers and sisters who always supported and encouraged us during our project and studies at IIUI.

Badar E Alam

Tayyab Ur Rehman

Touseef Ameer

Project Title:	AI Based Industrial Motor Fault Prediction	
Undertaken By:	Badar E Alam	(826-FET/BSEE/F20)
	Tayyab Ur Rehman	(877-FET/BSEE/F20)
	Touseef Ameer	(878-FET/BSEE/F20)
Supervised By:	**Engr. Dr. Tayyab Ali** Lab Engineer	
Co-Supervised By:	**Engr. Dr. Aleem Khaliq** Lab Engineer	
Date Started:	August, 2023	
Date Completed:	July, 2024	

Tools Used:

- **Arduino IDE**
- **MATLAB**
- **Simulink Simscape**
- **Python**
- **Jupiter Notebook**
- **Things Speak**

Abstract

This project embodies the development of an artificial intelligence (AI)-powered system that predicts motor failures, aiming to enhance industrial operations. Electric motors play a vital role in various industries, yet unforeseen faults can lead to significant periods of inactivity and financial setbacks. Our technology addresses these problems by using sensors, AI algorithms, and IoT connectivity to enable proactive maintenance. The system utilizes vibration, temperature, and current sensors to get instantaneous data from motors. Sophisticated artificial intelligence algorithms examine this data to forecast possible motor malfunctions. IoT connectivity allows for the remote monitoring and management of motor health, providing stakeholders with easily available data and alarms through user-friendly interfaces. The deployment of this system demonstrates considerable potential for improving industrial processes in industries such as manufacturing, energy, transportation, and essential infrastructure. Industries can save operational costs, increase productivity, and guarantee uninterrupted service provision by utilizing predictive maintenance strategies.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

SoC	System on Chip
FYDP	Final Year Design Project
OBE	Outcome Based Education
IoT	Internet of Things
AI	Artificial Intelligence
ML	Machine Learning
UN	United Nations
PLC	Programmable Logic Controller
SDG	Sustainable Development Goals
MSC	Multiscale Entropy
UDP	User Datagram Protocol

Chapter 1
Introduction

This chapter contain brief motivation/background information about the project i.e. "*AI Based Industrial Motor Fault Prediction*" problem statement, objectives, the methodology implemented for problem-solving, and the outlines of the results and future scope of the project.

1.1 Motivation

The inspiration for the report named "Artificial intelligence Industrial Motor Fault Prediction" comes from the basic need inside ventures to limit personal time, streamline support processes, and forestall exorbitant hardware disappointments. Modern engines assume a urgent part in different assembling processes, and their startling disappointments can prompt critical monetary misfortunes because of creation stops and fix costs. Customary upkeep approaches, like booked or responsive support, are frequently wasteful and can result in pointless personal time and asset wastage. By saddling the force of man-made consciousness (computer-based intelligence) and AI procedures, the point is to alter the support worldview by moving towards prescient support techniques. Prescient support uses progressed calculations to investigate ongoing information from sensors, PLCs, and correspondence conventions to anticipate potential gear disappointments before they happen.

This proactive methodology takes into consideration opportune intercessions, preventive measures, and improved planning of upkeep exercises, eventually upgrading hardware unwavering quality, limiting personal time, and diminishing functional costs. The report looks to investigate and assess the adequacy of simulated intelligence based prescient support frameworks, especially zeroing in on engine shortcoming expectation in modern settings. By utilizing AI calculations, for example, Irregular Timberland, Backing Vector Machines, or on the other hand Brain Organizations, the objective is to foster exact models fit for recognizing early signs of engine corruption, bearing shortcomings, or other execution oddities. Through exhaustive information investigation and trial approval, the report plans to show the capability of AI based intelligence driven prescient upkeep arrangements in improving functional productivity, amplifying efficiency, and guaranteeing the smooth working of modern hardware. Besides, the report means to add to the developing assemblage of examination in the field of modern robotization and Industry 4.0 by giving bits of knowledge into the down to earth execution and advantages of simulated intelligence based prescient upkeep frameworks. By showcasing2 genuine applications, exploratory outcomes, and contextual investigations, the report means to move industry partners to embrace AI based intelligence advances and take on prescient support techniques to remain ahead in an undeniably aggressive market scene.

At last, the overall inspiration is to enable businesses with the instruments and information expected to proactively deal with their resources, relieve chances, and open new degrees of effectiveness and efficiency in the time of advanced change.

1.2 Project Overview

In our project, we are changing industrial motor support by utilizing state of the art Artificial intelligence innovation to anticipate and forestall possible shortcomings before they grow into exorbitant breakdowns. By coordinating an organization of sensors, including IR temperature sensors, flow sensors, and piezoelectric vibration sensors, we catch continuous information on basic boundaries like temperature, flow, and vibration levels. These sensors persistently screen the engine's wellbeing, giving a complete comprehension of its functional state. Through cutting edge AI calculations, we investigate this information to identify early advance notice indications of issues like inordinate intensity, overcurrent issues, and undesirable vibrations.

By taking AI-driven predictive maintenance, we enable businesses to proactively deal with their motor frameworks, improving execution and lessening the gamble of impromptu margin time. Our answer improves functional productivity as well as limits upkeep costs by forestalling horrendous disappointments and broadening hardware life expectancy. With our inventive methodology, modern offices can change from responsive support practices to proactive techniques, guaranteeing continuous creation processes and expanding efficiency.

1.3 Problem Statement

Electric motors are the foundation of many operations in the industrial world, and their smooth functioning is vital to it. On the other hand, problems with severe vibrations, high rotor temperatures, and anomalies in current flow can cause serious problems with motor performance. Industries that depend on continuous production suffer significant financial losses and downtime as a result of these disruptions. This project aims to create an advanced AI-powered monitoring and prediction system in response to these problems.

The principal aim is to detect possible malfunctions in industrial motors in advance by keeping an eye on variables like vibrations, temperature fluctuations, and current flow. Maintenance teams can strategically intervene to reduce unscheduled downtime and decrease repair costs by identifying these issues early on. In addition, the project intends to develop customized Digital Twins for various motor variations in order to improve fault prediction and detection accuracy. In the end, our project aims to provide timely insights to industries so they can maintain production and operational effectiveness.

1.4 Project Objectives

Our project, which aims to transform industrial motor maintenance, is in line with the Sustainable Development Goals of the UN. Our objective is to improve real-time data collecting and visualization by using modern sensors and applying AI algorithms, hence encouraging proactive maintenance techniques. Our contribution to high-quality instruction in AI-driven predictive maintenance is the application of AI for early fault prediction. By guaranteeing dependable access to necessary services and reducing disruptions, our program promotes sustainable cities and communities. Furthermore, we optimize resource use in motor maintenance with simulated models such as Digital Twins, promoting responsible consumption and manufacturing. These ideas are explained separately as follow:

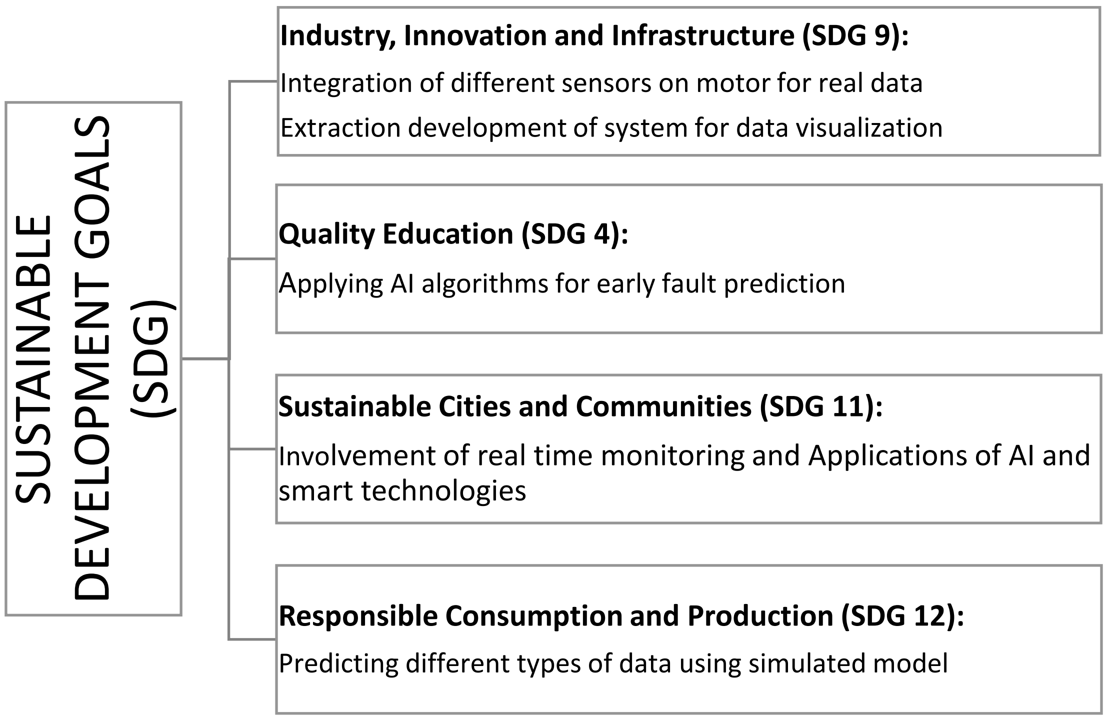

SUSTAINABLE DEVELOPMENT GOALS (SDG)
Industry, Innovation and Infrastructure (SDG 9):
Integration of different sensors on motor for real data
Extraction development of system for data visualization
Quality Education (SDG 4):
Applying AI algorithms for early fault prediction
Sustainable Cities and Communities (SDG 11):
Involvement of real time monitoring and Applications of AI and smart technologies
Responsible Consumption and Production (SDG 12):
Predicting different types of data using simulated model

Fig 1. 1 Represent Objectives of the project according to UN SDG

1.4.1 SDG 9: Industry, Innovation, and Infrastructure

- **Integration of different sensors on motor for real data:**

The objective is to obtain real-time data on essential parameters by integrating many sensors, including piezoelectric vibration sensors, current sensors, and infrared temperature sensors, onto industrial motors. A thorough understanding of motor performance will be made possible by this integration, which will also make preventative maintenance techniques possible.

- **Development of System for data Visualization:**

The main goal of this purpose is to create a reliable system for displaying the information gathered from the integrated sensors. Stakeholders may quickly understand the operating characteristics of the motor and spot any anomalies or patterns by producing clear and educational data visualizations. We utilize "MATLAB Things Speak" to visualize the data at this step.

1.4.2 SDG 4: Quality Education

- **Applying AI Algorithm for Early Fault Prediction:**

The goal of this project is to use artificial intelligence (AI) algorithms to evaluate the gathered data and forecast possible industrial motor failures. The system will use predictive analytics and machine learning to find trends that point to imminent breakdowns. This will advance knowledge of motor maintenance and provide chances for education in AI-driven predictive maintenance.

1.4.3 SDG 11: Sustainable Cities and Communities

- **Involvement of Real-Time Monitoring and Application of AI and Smart Technologies:**

The significance of real-time monitoring and the use of AI and smart technologies in industrial settings are emphasized by this goal. Through the use of smart monitoring systems and predictive maintenance approaches, enterprises can cultivate sustainable practices, mitigate disturbances to urban infrastructure, and enhance community welfare by guaranteeing dependable accessibility to crucial services that support sustainability.

1.4.4 SDG 12: Responsible Consumption and Production

- **Predicting different type of data using Simulations:**

  This goal is to use MATLAB Simscape to create a simulated model, or may be Digital Twin, that can be used to anticipate different kinds of data about industrial motor performance. Stakeholders can enhance their understanding of the behavior of motors, maximize resource utilization, and encourage responsible consumption and production practices by modeling various operating circumstances and scenarios.

1.5 Brief Project Methodology

Project methodology can be understandable according to the given points and figure:

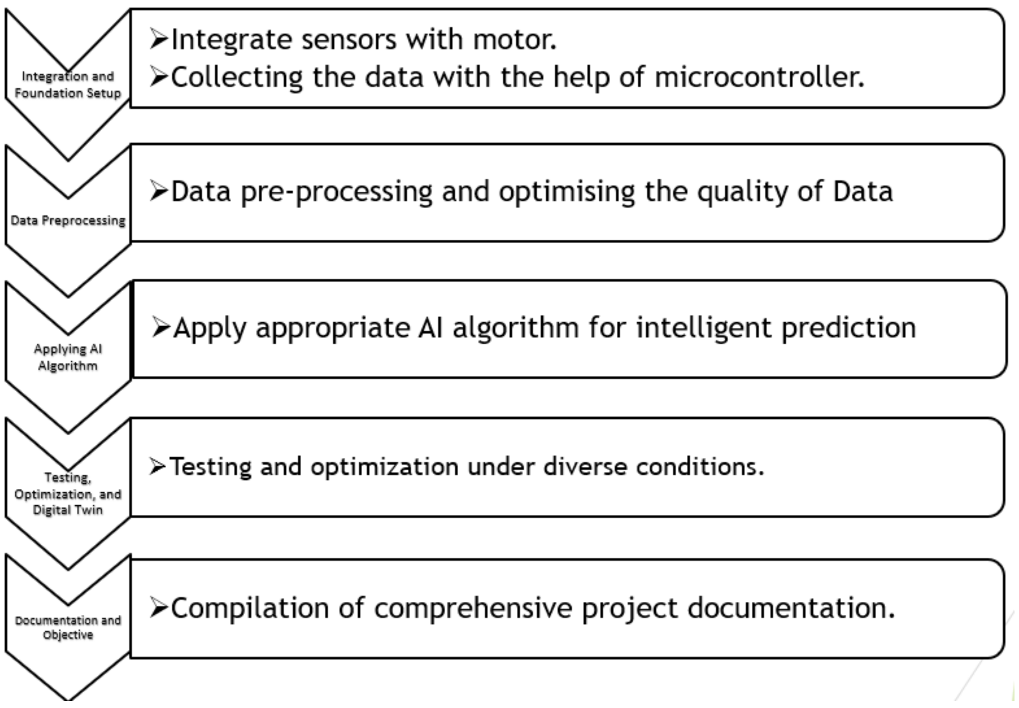

Integration and Foundation Setup
Integrate sensors with motor.
Collecting the data with the help of microcontroller.
Data Preprocessing
Data pre-processing and optimising the quality of Data
Applying AI Algorithm
Apply appropriate AI algorithm for intelligent prediction
Testing, Optimization, and Digital Twin
Testing and optimization under diverse conditions.
Documentation and Objective
Compilation of comprehensive project documentation.

Fig 1. 2 Represent the brief Methodology of the Project

1.5.1 Integration and Foundation Setup:

In this phase, we focus on integrating various sensors with the industrial motor to enable real-time data collection. Sensors such as IR temperature sensors, current sensors, and piezoelectric vibration sensors are strategically placed to monitor temperature, current, and vibrations, respectively. These sensors are connected to a microcontroller, which facilitates data collection and transmission to the central processing unit. By establishing this foundation, we ensure continuous monitoring of crucial motor parameters, laying the groundwork for predictive maintenance.

1.5.2 Data Preprocessing:

Once the data is collected from sensors, it undergoes preprocessing to enhance its quality and usability. Techniques such as noise removal, outlier detection, and data normalization are applied to optimize the data for further analysis. Additionally, missing data handling methods like interpolation or deletion are employed to ensure completeness. Through effective data preprocessing, we aim to improve the reliability and accuracy of subsequent analyses and predictions.

1.5.3 Applying AI Algorithm:

After preprocessing, the preprocessed data is fed into AI algorithms for intelligent prediction of motor faults. Various AI techniques such as anomaly detection and classification are utilized to analyze the data and identify patterns indicative of potential faults. Anomaly detection algorithms help in detecting deviations from normal behavior, while classification algorithms categorize data into different fault classes. By leveraging AI, we aim to develop a robust predictive maintenance system capable of proactive fault identification.

1.5.4 Testing, Optimization, and Digital Twins:

In this phase, the developed predictive maintenance system undergoes rigorous testing and optimization under diverse operating conditions. Real-world testing is conducted to evaluate the system's performance in different industrial environments and scenarios. Optimization techniques are applied to fine-tune the system parameters and algorithms for improved accuracy and efficiency. Additionally, digital twins are created to simulate predictive scenarios and analyze potential faulty data. These digital twins serve as virtual replicas of the industrial motors, enabling predictive simulations and enhancing fault prediction capabilities.

1.5.5 Documentation and Objective:

Finally, comprehensive documentation of the entire project, including the integration process, data preprocessing techniques, AI algorithms employed, testing results, and optimization strategies, is compiled. This documentation serves as a reference for future maintenance activities and provides insights into the project's objectives and achievements. By documenting the project's methodologies and outcomes, we ensure transparency and facilitate knowledge sharing within the organization and the broader industrial community.

1.6 Report Outline

The first section of the research explores the vital function that electric motors play in industrial processes, highlighting how vital they are for powering a variety of machinery that is necessary for both production and operation. These motors can, however, experience several problems, including overheating, overcurrent, and undesired vibrations, which can cause large amounts of downtime and financial losses for businesses. The project intends to use AI technology to create an enhanced monitoring and prediction system in order to address these issues. With the help of this system, maintenance personnel will be able to proactively identify motor issues and take strategic action to prevent unplanned downtime, which would guarantee continuous production for various sectors.

The goals of the project are in line with the Sustainable Development Goals of the UN, with special attention on Quality Education (SDG 4), Sustainable Cities and Communities (SDG 11), Industry, Innovation, and Infrastructure (SDG 9), and Responsible Consumption and Production (SDG 12). Real-time data for analysis can be obtained by integrating various sensors, such as piezoelectric vibration sensors, current sensors, and infrared temperature sensors, onto the motor. The creation of a data visualization system will improve our comprehension of motor performance and make maintenance task decision-making easier. The initiative intends to increase the efficiency and dependability of industrial processes by utilizing AI algorithms for early fault detection.

Starting with the integration and foundation setup, where sensors are connected with the motor and data is gathered via microcontrollers, the technique consists of multiple critical steps. After that, data pretreatment methods are used to maximize the quality of the data, and then AI algorithms are used for intelligent prediction. The robustness of the system is tested and optimized under various scenarios, and the process is aided by the construction of digital twins for predictive simulations and erroneous data. The study finishes with a thorough summary of the project's findings and goals, including insights into the successes, drawbacks, and directions for further research and development in the area of industrial motor defect prediction.

Chapter 2
Literature Review

2.1 Background of Project

The backbone of modern-day industrial operations, induction motors provide strong, dependable performance that powers a wide range of manufacturing processes. These motors power machinery and equipment used for production, processing, and other industrial processes while running constantly under exacting circumstances. But because of the constant use, the motors are subjected to several stresses, such as thermal stress, mechanical wear, and climatic conditions, which might eventually cause defects to arise.

Fig 2. 1 Represent the older methods of fault diagnosing

A wide range of problems, including mechanical vibrations, uneven stator windings, voltage imbalances, and thermal overloads, can cause defects in industrial motors. If these defects are not found or corrected, they have the potential to worsen quickly, leading to expensive repairs, production halts, and equipment failures. Stator winding breakdowns are among the most frequent and dangerous types of glitches, and they can seriously impair the dependability and performance of motors.

Conventional techniques for finding motor faults frequently involve human inspection, scheduled maintenance, and the examination of variables including voltage, current, power factor, and vibration signals. Although these techniques have demonstrated some efficacy, they are labor-intensive, time-consuming, and may not possess the sensitivity required to precisely identify early defects. Likewise, standard planned maintenance and physical inspection might not always be enough to find and fix emergent problems in a timely way.

Predicting industrial motor faults and performing preventative maintenance with the use of modern technologies, especially artificial intelligence (AI), has gained popularity in recent years. AI could completely transform defect detection by allowing massive volumes of sensor data to be analyzed in real-time and seeing minute patterns that point to future problems. Researchers want to improve problem detection accuracy, efficiency, and dependability while lowering dependency on manual inspection and routine periodic maintenance by utilizing AI algorithms and machine learning attitudes.

The dependability, effectiveness, and safety of industrial processes could be greatly enhanced by the incorporation of AI into industrial motor defect prediction. Artificial intelligence (AI) systems have the capability to examine data from many parameters, such as vibration, temperature, and current, to detect possible issues and initiate preventive maintenance measures. By extending the lifespan of industrial machinery and minimizing production downtime, this hands-on approach lowers repair costs and boosts overall operational efficiency and competitiveness.

Using sophisticated signal processing methods like multiscale entropy (MSE) and instantaneous current signature analysis (HT) in conjunction with grey fuzzy logic is a potential strategy for AI-based defect prediction. These techniques capture minute changes in motor activity that point to imminent errors, improving fault detection accuracy and reliability. Researchers hope to create complete failure prediction systems that are suited to the unique requirements and difficulties of industrial applications by fusing AI algorithms with multi-parameter monitoring techniques.

In short, there are promising prospects for improving industrial motor fault prediction and preventative maintenance practices due to the confluence of modern technologies like artificial intelligence, signal processing, and predictive analytics. In an increasingly competitive global landscape, researchers can drive innovation in predictive maintenance by utilizing insights from data-driven approaches, interdisciplinary collaborations, and innovative technologies. This will ultimately improve the reliability, efficiency, and safety of industrial operations.

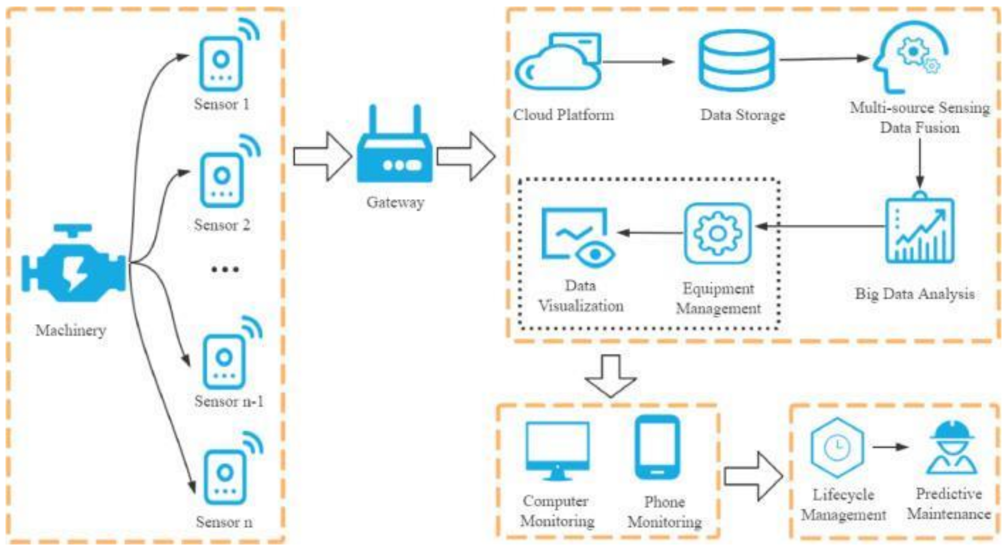


Fig 2. 2 Conceptual Block Diagram (taken form online resource)

2.2 Related Work/Projects

Our project is dedicated to revolutionizing industrial maintenance practices by developing advanced predictive maintenance solutions for industrial motors. By leveraging cutting-edge technologies and interdisciplinary collaboration, we aim to enhance industrial operations' reliability, efficiency, and safety.

At the core of our project is utilizing a multi-sensor approach to gather comprehensive data on the health and performance of industrial motors. We will deploy current, piezoelectric, and infrared (IR) temperature sensors to collect data on parameters such as current flow, mechanical vibrations, and temperature fluctuations. This multi-sensor setup lets us capture a holistic view of motor behavior under different operating conditions. We will utilize Arduino Nano and ESP32 microcontrollers to facilitate data collection and processing, which provide versatile and customizable platforms for sensor integration and data acquisition. These microcontrollers will be deployed alongside the sensor arrays to collect real-time data from industrial motors in diverse operating environments.

Once the data is collected, we will employ MATLAB ThingsSpeak, a cloud-based IoT platform, to aggregate and manage the data streams from multiple sensors. MATLAB ThingsSpeak offers robust data storage and visualization capabilities, enabling us to efficiently analyze and interpret the collected data. With the data gathered on MATLAB ThingsSpeak, we will apply advanced artificial intelligence (AI) techniques to analyze the data and develop predictive models for motor fault detection. By leveraging machine learning algorithms and pattern recognition techniques, we aim to identify subtle patterns and anomalies in the data that may indicate potential motor faults.

The application of AI to the collected data will enable us to detect emerging faults and predict maintenance needs proactively, thereby minimizing unplanned downtime and reducing repair costs. Furthermore, by integrating predictive maintenance strategies into existing workflows, we aim to optimize maintenance schedules and maximize the lifespan of industrial machinery.

Overall, our project represents a concerted effort to harness the power of technology and data-driven insights to transform industrial maintenance practices. By combining sensor technology, microcontroller platforms, cloud-based data management, and AI algorithms, we aim to develop practical solutions that deliver tangible benefits to industrial organizations, ensuring their critical assets' reliability, efficiency, and longevity.

2.3 Project Contribution

Alok **Verma**, Somnath **Sarangi**, and Maheshkumar **H. Kolekar** worked together on a research article called "Stator winding fault prediction of induction motors using multiscale entropy and grey fuzzy optimization methods." In the introduction, Verma emphasized the significance of promptly identifying faults in induction motors and outlined the widespread occurrence and repercussions of stator winding problems in industrial settings [1]. Sarangi provided a thorough summary of prior research efforts in the detection of faults in stator windings. The summary highlighted several approaches used for diagnosing faults. Meanwhile, Kolekar introduced an innovative method that utilized multiscale entropy (MSE) and grey fuzzy logic to detect faults and optimize motor parameters. He performed studies to confirm the effectiveness of the suggested methodology, ultimately contributing to the progress of defect diagnosis techniques in induction motors.

Yang **Huang**, Chiun-Hsun **Chen**, and **Chi-Jui Huang** authored the paper titled "Motor fault detection and feature extraction using an RNN-based variational autoencoder," published in IEEE Access in 2019. Their study creates a two-stage machine learning analysis architecture that can accurately guess motor fault modes by looking at time-domain signals of motor vibration without having to do a lot of complicated preprocessing [2]. In the first stage, they propose a method called RNN-based VAE for dimension reduction of time series data, achieving significant improvements in prediction accuracy compared to other methods such as autoencoders and variational autoencoders. The authors further employ Principal Components Analysis (PCA) and Linear Discriminant Analysis (LDA) in the second stage for additional dimension reduction, enabling clear visualization and detecting different or unknown fault modes. This innovative approach addresses the challenges of traditional fault detection methods and demonstrates promising results for application in smart manufacturing environments.

Anurag **Choudhary,** Deepam **Goyal,** and Shimi Sudha **Letha** wrote the study "Infrared thermography-based fault diagnosis of induction motor bearings using machine learning." It was published in the IEEE Sensors Journal in 2020 [3]. Their study presents an innovative method for detecting defects in induction motors (IM) by utilizing infrared thermography (IRT) in conjunction with machine learning algorithms. The authors propose a novel approach to overcome the drawbacks of conventional fault identification techniques. They introduce a two-dimensional discrete wavelet transform (2D-DWT)-based IRT method for effectively identifying inner and outer race flaws and detecting insufficient lubrication. The researchers employ principal component analysis (PCA) and Mahalanobis distance (MD) to reduce and select features and then use complex decision trees (CDT), linear discriminant analysis (LDA), and support vector machines (SVM) for classification. The results suggest that Support Vector Machines (SVM) perform better than Causal Decision Trees (CDT) and Linear Discriminant Analysis (LDA) in classifying faults. This demonstrates the usefulness of the proposed technique for recognizing bearing problems in induction motors (IM) and reducing unexpected shutdowns and maintenance expenses.

The paper titled "Machine learning approach for predictive maintenance in industry 4.0" by Marina **Paolanti** was presented at the 2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA). It introduces a machine learning framework for predictive maintenance in industrial environments [4]. The authors highlight the significance of condition monitoring and predictive maintenance in preventing unforeseen motor failures and improving the reliability of the system. The authors present a methodology based on Random Forest for Predictive Maintenance, which was evaluated using actual industry data obtained from several sensors and machine PLCs. The suggested approach facilitates real-time decision-making for maintenance management by training a Random Forest model using Azure Machine Learning Studio. Initial findings indicate a significant level of precision (95%) in forecasting various operational conditions of a cutting machine using real-time data. The paper's contributions encompass the presentation of a comprehensive cloud architecture for Industry 4.0, the application of machine learning techniques to actual machine data, and the successful attainment of a high level of accuracy in predicting the state of the main spindle rotor. The study is organized to offer a comprehensive overview of data analysis, elaborate on machine learning methodologies for predictive maintenance, and present empirical findings from a practical use case.

The study by Andrew **M. Knight** and Sergio **P. Bertani**, published in the IEEE Transactions on Energy Conversion, looks at how to use stator current monitoring to find mechanical faults in a medium-sized induction motor [5]. The writers talk about how important it is to keep an eye on the status of induction motors in factories and stress using stator current analysis as a cheap and non-intrusive way to find problems. The report highlights the significance of identifying mechanical flaws, particularly bearing-related issues, which are frequent causes of motor failures. Previous studies mainly concentrated on electrical faults. The paper outlines the creation of a testing facility that can replicate eccentricity faults and bearing fault situations. This facility allows for examining the possibility of detecting faulty bearing conditions by monitoring the stator current. The experimental findings indicate the possibility of identifying bearing deterioration using cost-effective and uncomplicated equipment, providing significant knowledge for industrial maintenance procedures. This research enhances the comprehension of mechanical fault identification in medium-sized induction motors and has practical implications for condition monitoring in industrial environments.

The article presents a fresh method for finding problems with machinery, especially induction motors. Contrary to conventional approaches that depend on evaluating a single signal, this method utilizes numerous signals concurrently. It is like analyzing a machine from various perspectives to comprehend more comprehensively. Researchers can efficiently analyze and understand these signals using an advanced computer program called a deep learning model. Through empirical investigation, they ascertained that this strategy surpasses traditional methodologies. It involves proactively identifying and resolving possible issues in a machine before they worsen. Not only does this improve safety, but it also results in substantial cost savings by preventing severe malfunctions.

2.4 Summary

The research articles examined novel techniques for problem diagnosis and predictive maintenance in industrial environments, specifically focusing on induction motors. The initial paper presents a machine learning framework for anticipatory maintenance employing a random forest methodology. The system can accurately forecast machine conditions by aggregating data from diverse sensors and machine PLCs and its subsequent analysis utilizing Azure Cloud architecture. The study demonstrates the efficacy of this strategy in actual industrial scenarios, emphasizing its capacity to reduce expenses and enhance productivity.

Conversely, the second paper investigates the identification of mechanical faults in induction motors through stator current monitoring. The system detects mechanical malfunctions by studying the interplay between permeance and magneto-motive force harmonics. It then predicts the harmonics of the air gap flux density and calculates the induced voltages and currents in the stator windings. The experimental findings indicate that it is possible to identify bearing deterioration using uncomplicated and affordable equipment. This emphasizes the significance of employing condition-monitoring methods for induction motors.

Finally, the final paper presents a method for diagnosing faults in induction motors using deep learning and multiple signals. The suggested model uses convolutional neural networks (CNNs) to learn from many different types of sensor signals simultaneously, making it reliable and accurate at finding faults. The experiments done on a machine fault simulator illustrate this technology's superiority over traditional fault diagnostic procedures, emphasizing its potential to enhance machine quality and safety and decrease maintenance expenses. These publications jointly contribute to advancing fault detection and predictive maintenance in the field, providing useful insights and practical solutions for industrial applications.

Chapter 3
System Design and Implementation

3.1 System Design

Industries can efficiently address the difficulties of unanticipated setbacks and downtimes by utilizing the capabilities of artificial intelligence (AI), the Internet of Things (IoT), and predictive maintenance. This approach facilitates more efficient motor management, resulting in substantial time and labor savings through the implementation of intelligent solutions. Artificial intelligence and predictive maintenance are advanced technologies that utilize new trends to improve operational efficiency. The objective of our research is to monitor the physical condition and real-time data of industrial motors using a range of sensors, such as IR temperature sensors, piezoelectric sensors, and current sensors. These sensors offer vital information on factors such as temperature, vibration, and current, which are necessary for evaluating the condition of the motor.

Our project involves strategically placing an infrared temperature sensor on the primary coil of the motor to monitor changes in temperature accurately during various levels of load. This enables us to observe and analyze the thermal performance of the motor and identify any irregularities in the heating patterns. Similarly, piezoelectric sensors are employed to record vibration data, aiding in the detection of mechanical imbalances or misalignments. Current sensors are devices that measure electrical properties, allowing for the detection of overcurrent conditions that may indicate the presence of short circuits or mechanical overloads. The data gathered by these sensors is forwarded to Google Colab for additional processing.

Inside Google Colab, the data goes through a meticulous cleaning and preprocessing stage to guarantee precision and pertinence for machine learning algorithms. The data is analyzed, and probable errors are predicted using techniques such as Linear Regression and K-nearest neighbors (KNN). We have collected data under three separate conditions: without any load (without a belt), with a load, and with an excessive load. For example, when we use a wheat grinder as a load, we can detect that a situation with no load could indicate a problem, possibly suggesting a damaged pulley belt. These observations enable well-informed decision-making on the maintenance and repair of motors.

Industries may achieve improved motor management and save substantial time by utilizing AI, IoT, and predictive maintenance to tackle unforeseen setbacks and downtimes efficiently. Artificial intelligence and predictive maintenance are advanced technologies that leverage new trends to enhance operational efficiency. The project utilizes a range of sensors, including IR temperature sensors, piezoelectric sensors, and current sensors, to monitor the physical condition and real-time data of industrial motors. These sensors provide vital information on temperature, vibration, and current parameters, which are necessary for evaluating the condition of the motor.

Our project involves strategically placing an infrared temperature sensor on the primary coil of the motor to monitor changes in temperature during various load circumstances accurately. This allows us to observe the thermal characteristics of the motor and identify irregular heating patterns. Piezoelectric sensors are used to detect and record vibration data, which can be used to discover problems caused by mechanical imbalances or misalignments. Current sensors are devices that measure electrical properties, such as the amount of current flowing through a circuit. They are used to detect overcurrent situations, which short circuits or mechanical overloads can cause. The data gathered by these sensors is forwarded to Google Colab for additional processing.

During the data processing step in Google Colab, a meticulous cleaning and preparation procedure is performed to guarantee the precision and pertinence of the data for machine learning algorithms. The data is analyzed, and probable errors are predicted using techniques such as Linear Regression and K-nearest neighbors (KNN). We have collected data under three separate conditions: without any load (without a belt), with a load, and with an excessive load. For example, if we use a wheat grinder as a load, we can detect that a situation with no load could indicate a malfunction, possibly suggesting a broken pulley belt. These observations enable well-informed decision-making on the maintenance and repair of motors.

The data obtained from this study is used as a basis for developing digital replicas of electric motors and industrial systems. Digital twins are computer-generated models that accurately represent physical systems, allowing for continuous monitoring and the ability to simulate future outcomes. Through the ongoing analysis of data collected by these sensors, we can create strong models that forecast the performance of motors in different situations, thereby improving the dependability and effectiveness of industrial operations. This data-centric methodology eliminates major malfunctions and enhances the overall sustainability of industrial operations by reducing downtime and optimizing the utilization of resources. The use of artificial intelligence (AI), the Internet of Things (IoT), and predictive maintenance represents a significant change in the management of industrial motors, establishing a higher benchmark for proactive maintenance and operational excellence.

3.1.1 System Architecture Block Diagram

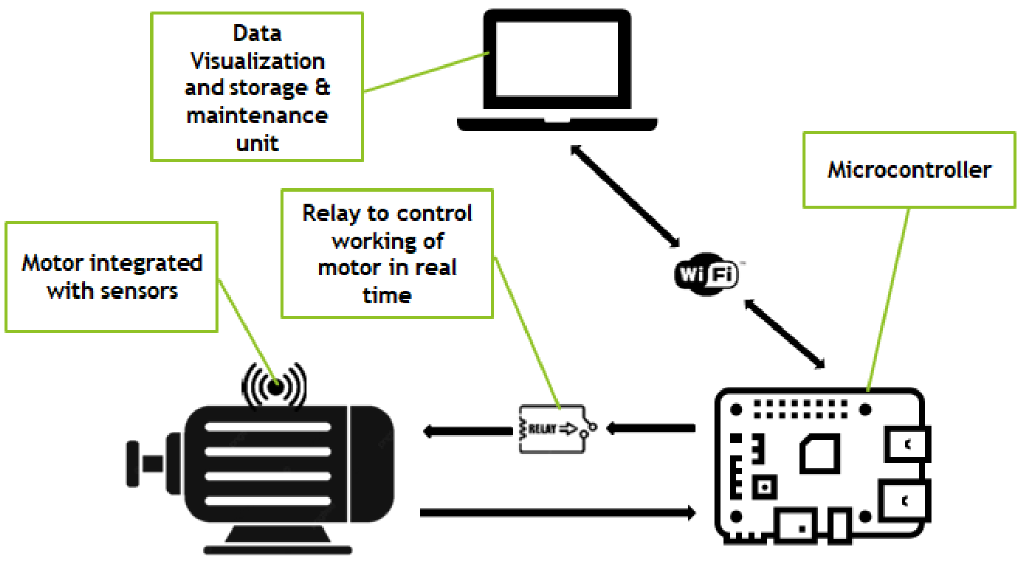

Data Visualization and storage & maintenance unit
Microcontroller
Relay to control working of motor in real time
Motor integrated with sensors
RELAY

Fig 3. 1 Represent the Block Diagram of our FYDP

3.1.2 Requirement Analysis

The primary prerequisites of this project are as follows:

- Conduct a comprehensive analysis of pertinent literature and prior studies related to the subject matter.
- Choose suitable sensors for effortless data acquisition, such as IR Temperature sensors for monitoring winding temperature, Piezo Electric sensors for detecting rotor vibrations, and Current sensors for measuring electrical current.
- Select a highly efficient microcontroller for collecting and transmitting data. Use the Arduino Nano specifically for gathering current data, vibrations and temperature while employing the ESP32 for handling other types of data and facilitating cloud-based transmission.
- Utilizing MATLAB ThingSpeak as the cloud-based platform for visualizing data.
- Identify the most advantageous location for each sensor to get most accurate data.
- Perform data cleansing and preprocessing to ensure the data's quality for machine learning algorithms.
- Identify appropriate machine learning algorithms for prediction tasks.
- Perform comprehensive testing and employ random selection to validate the correct operation of the control configuration.

3.2 Methodological/Implementation Details

Project implementation method includes the following points:

- Create an intelligent system that collects motor data using sensors (current, piezoelectric vibration, and infrared temperature).
- To forecast motor failures, sensors gather analogue signals on vibration, temperature, and current.
- The ESP32 and Arduino Nano microcontrollers are used to process and communicate the gathered data.
- Whereas the ESP32 measures temperature and vibrations, the Arduino Nano measures current.
- Microcontrollers, which comprise the primary data collection system, exchange data over Tx and Rx pins.
- Transferring data using the concept of UDP to our PC
- MATLAB ThingSpeak, an open-source platform from MathWorks, is used for data visualization.
- For visualization, ESP32 transmits motor data via the internet to ThingSpeak.
- For predictive maintenance, machine learning methods (K-Nearest Neighbor and Linear Regression) are used on Google Colab.

3.2.1 Hardware/Development Setup

To build an intelligent system that makes use of a variety of sensors for thorough data collecting and analysis from industrial motors, the project implementation approach entails a number of crucial elements.

- **Sensor Integration:** To monitor important characteristics like temperature, vibration, and current, the system integrates a range of sensors, such as current sensors, piezoelectric vibration sensors, and infrared temperature sensors. Because any metric might point to possible problems, these sensors are essential for anticipating motor failures. Analogue signals, which are crucial for real-time monitoring and analysis, are the data that the sensors gather.
- **Microcontroller Deployment:** We connect the sensors to microcontrollers to ensure optimal data processing. We employ an Arduino Nano to gather data from current, vibration, and temperature sensors. The Arduino Nano is responsible for collecting and overseeing data from these sensors, guaranteeing accurate monitoring under different operational circumstances. The extensive collection of this data is critical for

assessing the condition and functionality of the motor, allowing for precise and prompt study of variations in current, vibrations, and temperature.

- **System Communication:** The Arduino Nano and NodeMCU exchange data by utilizing their Tx (transmit) and Rx (receive) pins. The Arduino Nano's Tx pin communicates the sensor data it collects, while the NodeMCU's Rx pin receives this data from the Arduino Nano. The NodeMCU functions as a data transmitter, relaying the gathered data to ThingSpeak and utilizing the UDP protocol to communicate the data to a personal computer. This configuration facilitates seamless and prompt data transmission and monitoring, guaranteeing an accurate and efficient representation of the motor's operational state.
- **Data Visualization:** MATLAB ThingSpeak, an open-source platform from MathWorks, is used in this project to visualize the data that has been gathered. The data is sent by the ESP32 via the internet to ThingSpeak, where it is displayed in real time. Comprehensive data analysis and visualization made possible by ThingSpeak enable preemptive maintenance decisions and clear insights into the motor's state.
- **Machine Learning Implementation:** Google Colab, a free platform that supports a variety of machine learning tasks, is utilized by the project to implement machine learning algorithms. The data is subjected to linear regression and K-Nearest Neighbors (KNN) algorithms to forecast possible errors. By analyzing the patterns in the data, these algorithms help to prevent unplanned motor breakdowns and enable early detection of anomalies. By seeing problems before they get serious, this predictive maintenance strategy drastically lowers maintenance costs and downtime.

To produce a reliable predictive maintenance system, the project integrates real-time data visualization, machine learning algorithms, microcontroller integration, and advanced sensor technology. By reducing downtime and increasing overall productivity in industrial environments, this technology guarantees the effective and dependable running of industrial motors.

3.2.2 Hardware Details

The detail of the hardware components is as follows:

- **Arduino Nano**

The Arduino Nano is a compact embedded device that utilizes the ATmega328P microcontroller. This program utilizes current, vibration, and temperature sensors to gather data in various scenarios. It transforms the analogue impulses captured by these sensors into digital signals. We communicate this data to the NodeMCU's Rx pin on the Arduino Nano for further processing and cloud deployment.

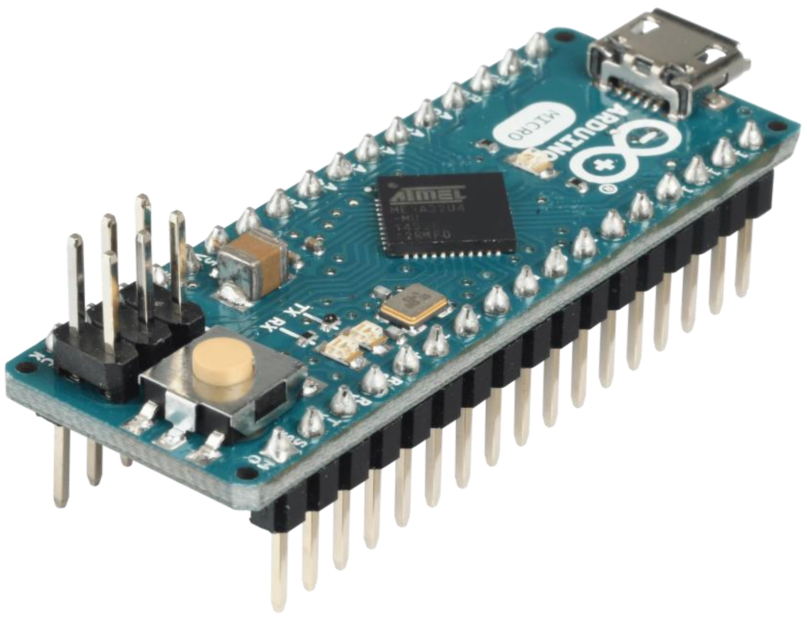


Fig 3. 2 Arduino Nano

- **NodeMCU**

The Tensilica Xtensa LX6 CPU powers the embedded microcontroller device NodeMCU, which also features integrated Wi-Fi and Bluetooth modules. The microcontroller is highly adaptable and commonly employed in Internet of Things (IoT) applications and control-oriented tasks. In this configuration, the NodeMCU acquires all sensor data (including current, vibration, and temperature) from the Arduino Nano through the Rx pin. The NodeMCU subsequently transfers this data to the cloud, utilizing ThingSpeak, and employs UDP to dispatch the data to a personal computer. In addition, the NodeMCU has the capability to simultaneously control a relay while performing its data transmission responsibilities.

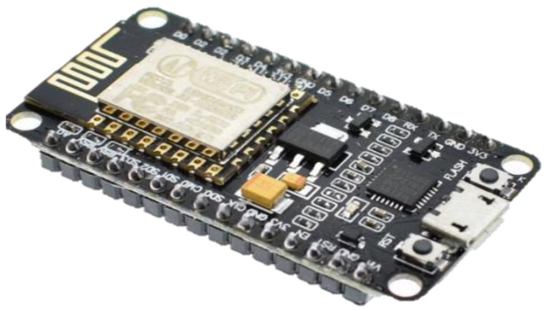


Fig 3. 3 NodeMCU: also known as esp8266

- **Contactless IR Temperature sensor**

We are using *MLX90515* for the measurement of temperature form winding. This sensor is deployed on the main winding of the induction motor where it will collect the data related to the temperature. It consists of signal processing unit (SPU) which amplifies the weak infrared signals and convert it into usable voltage.

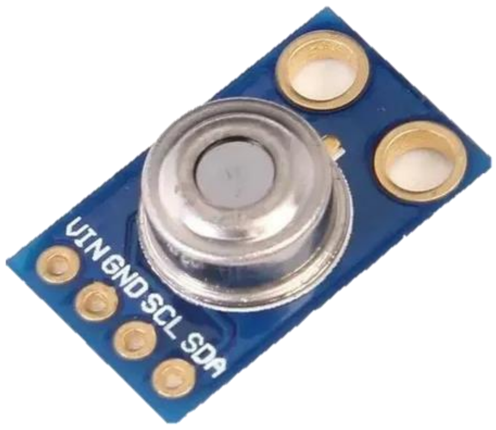


Fig 3. 4 Contactless IR Temperature sensor (MLX 90515)

- **Piezo Electric Sensor**

Mechanical vibrations i.e. *KS0272* are translated into electrical impulses via a piezoelectric sensor. This sensor is critical to our project since it measures the vibrations inside the motor, allowing us to watch and anticipate motor problems. It identifies possible problems by detecting imbalance, misalignment, and excessive load. The ESP32 microcontroller receives real-time vibration data from the sensor and processes it before sending it to the Matlab ThingSpeak platform for analysis.

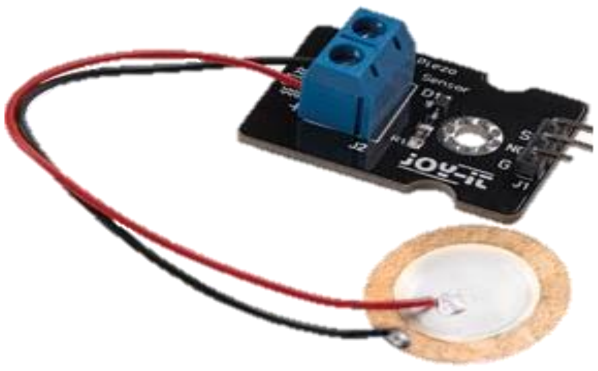


Fig 3. 5 Piezo electric sensor (KS 0272)

- **Current Sensor**

A current sensor i.e. *ACS712* quantifies the electrical current passing through a conductor and transforms it into a legible signal. In our project, this sensor is crucial for measuring the electrical current that the motor consumes under various operational conditions. It makes it easier to spot issues like excessive current from mechanical overload, faulty capacitors, or voltage swings. The Arduino Nano handles the current data and transmits it to the ESP32, which transfers it to the MATLAB ThingSpeak platform for analysis. The ability to monitor in real-time enables the early identification of malfunctions, assuring optimal motor performance and minimizing periods of inactivity.

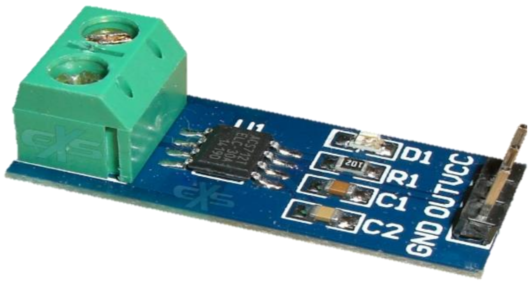


Fig 3. 6 Current Sensor (ACS 712)

- **Relay**

A relay is an electrically controlled switch that utilizes an electromagnet to activate a mechanical switch. In our project, relays regulate the power distribution to the motor, considering the information gathered from sensors. The relay can disconnect the motor to prevent harm when it detects abnormal conditions such as excessive current, high temperature, or odd vibrations. This guarantees the security and dependability of industrial activities, minimizing the likelihood of motor malfunctions and decreasing the time until operations are halted.

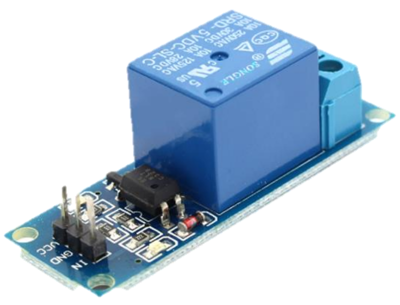


Fig 3. 7 Relay Module

- **Single Phase Induction Motor**

A single-phase induction motor uses a single-phase power supply to convert electrical energy into mechanical energy. In our project, the motor supplies energy to machinery under close observation to detect any issues. We evaluate the motor's performance by measuring current, temperature, and vibrations using sensors. Examining this data aids in predicting and averting faults, guaranteeing dependable industrial functioning.

Fig 3. 8 Single Phase 1hp induction motor

- **Wheat Grinder**

A single-phase induction motor uses a single-phase power supply to convert electrical energy into mechanical energy. In our project, the motor supplies energy to machinery under close observation to detect any issues. We evaluate the motor's performance by measuring current, temperature, and vibrations using sensors. Examining this data aids in predicting and averting faults, guaranteeing dependable industrial functioning.

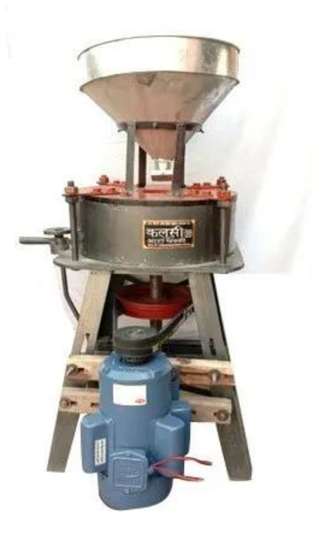


Fig 3. 9 Wheat Grinding Machine (as a load)

3.2.3 Software/Tools

We had used different tools and software in this project are the detail is as follows:

- **Arduino IDE**

The Arduino Integrated Development Environment (IDE) plays a crucial role. This software platform is crucial in simplifying the programming process and deploying code to Arduino microcontrollers, namely the Arduino Nano used in the project. The interface is designed to be easy to use, allowing developers to write, build, and publish code quickly and effectively, making the development process more efficient. Using the Arduino IDE, project engineers may effectively apply the required logic for processing sensor data and establish connectivity with other hardware components, including the ESP32 microcontroller and numerous sensors. The smooth integration of all hardware components allows the efficient functioning of the entire system, making a substantial contribution to the project's goals of predicting faults in industrial motors and performing maintenance in advance.

Fig 3. 10 Arduino IDE

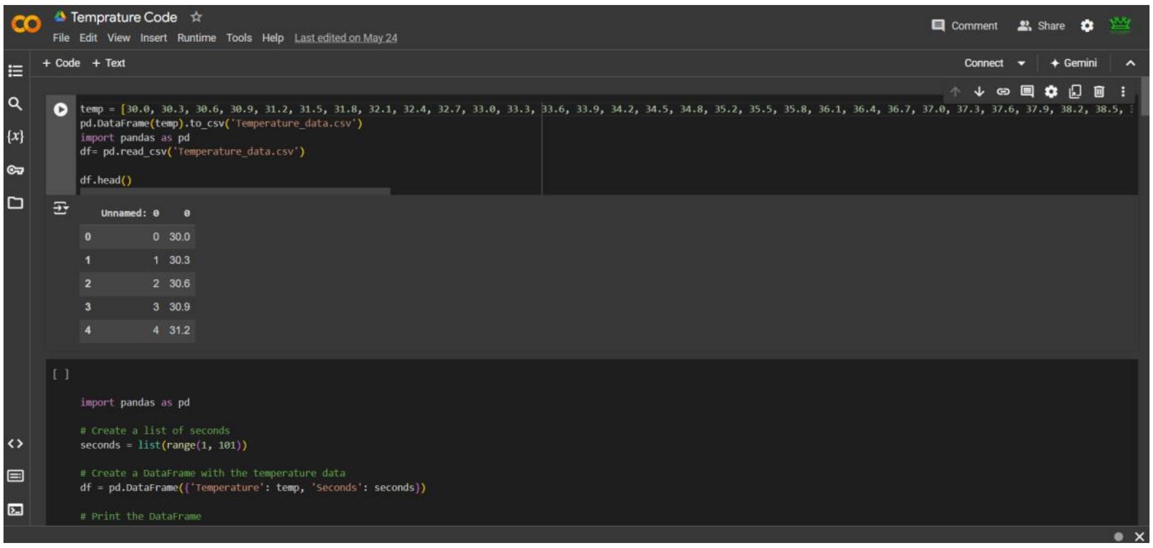
- **Python**

Python plays a crucial role in the software foundation of the project, especially when it comes to implementing machine learning. Python enables sophisticated data analysis and predictive modelling using Scikit-learn and Pandas. The scikit-learn library provides diverse machine-learning methods and utilities, enabling developers to construct and implement predictive maintenance models effortlessly. In addition, the Pandas package offers robust data structures and tools for manipulating and analyzing data, allowing for effortless preprocessing and modification of motor data. By raging Python's capabilities, developers may thoroughly loot the potential of machine learning approaches to store failures properly. Incorporating Python with sci-kit-learn and Panda highlights its crucial function in advancing the project's goals of predicting faults in industrial motors and implementing proactive maintenance.

Fig 3. 11 Python

- **Google Colab**

Google Colab is crucial for applying machine learning to the project. The platform offers a Python environment hosted on the cloud, granting users access to high-performance resources such as GPUs and TPUs. This enables the effective execution of activities, such as training models. Colab facilitates cooperation by allowing users to share notebooks, facilitating teamwork in developing predictive maintenance methods. Colab enables developers to implement machine learning algorithms such as linear regression and k-nearest neighbors to predict motor failures correctly. This component's integration is essential to accomplishing the project's objectives of predicting faults in industrial motors and implementing proactive maintenance measures.



Temprature Code
File Edit View Insert Runtime Tools Help Last edited on May 24
Comment Share
+ Code + Text
Connect
Gemini

```
temp = [30.0, 30.3, 30.6, 30.9, 31.2, 31.5, 31.8, 32.1, 32.4, 32.7, 33.0, 33.3, 33.6, 33.9, 34.2, 34.5, 34.8, 35.2, 35.5, 35.8, 36.1, 36.4, 36.7, 37.0, 37.3, 37.6, 37.9, 38.2, 38.5,
pd.DataFrame(temp).to_csv('Temperature_data.csv')
import pandas as pd
df= pd.read_csv('Temperature_data.csv')

df.head()
```

```
   Unnamed: 0     0
0           0  30.0
1           1  30.3
2           2  30.6
3           3  30.9
4           4  31.2
```

```
[ ]

import pandas as pd

# Create a list of seconds
seconds = list(range(1, 101))

# Create a DataFrame with the temperature data
df = pd.DataFrame({'Temperature': temp, 'Seconds': seconds})

# Print the DataFrame
```

Fig 3. 12 Google Colab

- **Things Speak**

ThingSpeak is essential to the project as it is a platform for visualizing real-time data. Created by MathWorks, it enables the representation of motor data communicated via the Internet. ThingSpeak facilitates thorough data analysis, assisting in making proactive maintenance decisions. The system enables real-time monitoring and analysis by transferring motor data from ESP32 microcontrollers to ThingSpeak. This integration guarantees efficient visualization and analysis of motor health, which supports the project's goals of predicting industrial motor faults and implementing proactive maintenance.

3.3 Algorithms/Codes

Here the detail of the codes and which we used in our project. It is divided into three parts i.e. code for Arduino Nano, ESP32 and Machine Learning.

3.3.1 Arduino Nano

We used an Arduino Nano to fetch the current, vibrations and temperature readings form motor Then, it was transmitted to esp32 using serial communication because Arduino Nano is much more efficient in current measurement. Here is the detailed code for the Arduino Nano:

```
#include "ACS712.h"
//  Arduino UNO has 5.0 volt with a max ADC value of 1023 steps
//  ACS712 5A  uses 185 mV per A
//  ACS712 20A uses 100 mV per A
//  ACS712 30A uses  66 mV per A
ACS712  ACS(A0, 5.0, 1023, 66);
//  ESP 32 example (might requires resistors to step down the logic
voltage)
//  ACS712  ACS(25, 3.3, 4095, 185);
void setup()
{
  Serial.begin(9600);
  while (!Serial);
  Serial.println(_FILE_);
  Serial.print("ACS712_LIB_VERSION: ");
  Serial.println(ACS712_LIB_VERSION);

  ACS.autoMidPoint();
  Serial.print("MidPoint: ");
  Serial.println(ACS.getMidPoint());
  Serial.print("Noise mV: ");
  Serial.println(ACS.getNoisemV());
  /////////////////////////////////////
  pinMode(A1, INPUT);
  Serial.println("---------Vibrations per Second------------------");

  while (!Serial);
}
void loop(){
  float average = 0;
  uint32_t start = millis();
  for (int i = 0; i < 100; i++)
  {
    //  select appropriate function
    //  average += ACS.mA_AC_sampling();
    average += ACS.mA_AC();
  }

  float mA = average / 100.0;
  uint32_t duration = millis() - start;
  //Serial.print("Time: ");
  //Serial.print(duration);
  Serial.print((mA)/1000);
  Serial.print(",");

  delay(700);

  int vibrationCount = 0;
  unsigned long startTime = millis();

  // Measure vibrations for 1 second
  while (millis() - startTime < 1000) {
    if (analogRead(A1) >= 10) {
      vibrationCount++;
      while (analogRead(A1) >= 10) {
        // Wait for the vibration to end
      }
    }
  }

  float vibrationsPerSecond = float(vibrationCount);
  Serial.println(vibrationsPerSecond);
  //Serial.print("V/s");

  //Serial.println(",");
  delay(1000 - millis() % 1000); // Update every second
}
```

3.3.2 NodeMCU

We used NodeMCU which is also known as esp8266 microcontroller. In our venture to fetch data and serially transfer it into the cloud to efficiently apply AI algorithms and then send the signal to relay according to the need of action. Here is the detailed code for the NodeMCU:

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include "ThingSpeak.h"
#include <Adafruit_MLX90614.h>
Adafruit_MLX90614 mlx = Adafruit_MLX90614();

const char* ssid = "alpha";
const char* password = "alpha1233";
////////////////////////////////////////////



// ThingSpeak parameters
unsigned long myChannelNumber = 2;
const char *myWriteAPIKey = "O1ATXJKNB473MZTW";
////////////////////////////////////////////
char data[70]; // for comma saperated data store coming from serail
monitor
char *token;
char current[20];
char vibrations[20];
char temperature[20];
int receivedValue;
#define RELAY_PIN D7

const int localUdpPort = 4425; // Local port to listen for incoming UDP
packets
const int remotePort = 4425;   // Remote port to send data to
IPAddress remoteIP(192,168,4,203); //// IP address of the destination
10.140.2.194 test wifi

unsigned long previousMillis = 0;
const long interval = 2000;
unsigned long previousMillis_1 = 0;
const long interval_1 = 4000;
////////////
WiFiClient client;
WiFiUDP udp;
void setup() {
  Serial.begin(9600);
  delay(1000);
  pinMode(D7, OUTPUT);
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  //////////
  ThingSpeak.begin(client);
 // Initialize ThingSpeak
  // Initialize UDP
  udp.begin(localUdpPort);
  Serial.println("UDP initialized");
}
void loop() {

  // Check for incoming UDP packets
  int packetSize = udp.parsePacket();
  if (packetSize) {
    char incomingPacket[255];
    int len = udp.read(incomingPacket, 255);
    if (len > 0) {
      incomingPacket[len] = 0; // Null-terminate the string
    }
    Serial.print("Received packet: ");
    Serial.println(incomingPacket);

    // Optionally, convert received packet to integer
     receivedValue = atoi(incomingPacket);
     Serial.print("Received value as integer: ");
     Serial.println(receivedValue);

    }
      if(receivedValue == 1){//receivedValue == 1 || receivedValue == 2
      digitalWrite(D7, LOW);
      Serial.println("high on d4 rely on/motor of");
      delay(3000);
      /*Serial.println("low on d4 rely off");
      digitalWrite(D7, LOW);
      delay(3000);*/
    } else{
      digitalWrite(D7, HIGH);
      Serial.println("low on d4 rely off/motor running");
      delay(3000);
  }
 unsigned long currentMillis = millis();
  // Non-blocking delay logic
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
   if (Serial.available() > 0) {
    // Read data from serial port
    memset(data,0,sizeof(data));
    Serial.readBytesUntil('\n', data, sizeof(data));
    // Get the first token
    token = strtok(data, ",");
 // Save the first token in current
    if (token != NULL) {
      strcpy(current, token);
    }
    // Get the next token
    token = strtok(NULL, ",");

    // Save the second token in vibrations
    if (token != NULL) {
      strcpy(vibrations, token);
  }
    token = strtok(NULL, ",");

      if (token != NULL) {
      strcpy(temperature, token);
  }
   }
    // Format the sensor values into a single string
    String sensorData = String(current) + "," + String(vibrations) + "," +
String(temperature);
    // Send sensor data over UDP
    udp.beginPacket(remoteIP, remotePort);
    udp.print(sensorData);
    udp.endPacket();
    Serial.println("Sensor data sent: " + sensorData);
    ThingSpeak.setField(1, current);
    ThingSpeak.setField(2, vibrations);
    ThingSpeak.setField(3, temperature);
  // Send data to ThingSpeak
  int x = ThingSpeak.writeFields(myChannelNumber,myWriteAPIKey);
  if (x == 200) {
    Serial.println("Channel update successful.");
  } else {
    Serial.println("Problem updating channel. HTTP error code " +
String(x));
     }
  }
  delay(1000);
}
```

3.3.3 Google Colab

We used Google Colab to apply learning algorithms and access using the Pandas library of Python. Here, we apply linear regression and the KNN classifier to the predictive maintenance. Here is the detail code of the collaboration:

- **KNN algorithm**

```
current_0=[5.11, 5.11, 5.11, 5.12, 5.1, 5.09, 5.09, 5.1, 5.11, 5.1, 5.09,
5.09, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.08, 5.09, 5.1, 5.09, 5.1, 5.1, 5.1,
5.09, 5.09, 5.1, 5.1, 5.1, 5.1, 5.11, 5.09, 5.09, 5.1, 5.09, 5.08, 5.1,
5.1, 5.11, 5.11, 5.1, 5.09, 5.1, 5.09, 5.09, 5.09, 5.07, 5.08, 5.09, 5.09,
5.08, 5.08, 5.08, 5.08, 5.07, 5.06, 5.06, 5.08, 5.06, 5.06]
vibrations_perSec_0= [45.0, 7.0, 11.0, 12.0,11.0, 12.0, 36.0, 21.0, 20.0,
26.0, 22.0, 28.0, 8.0, 0.0, 1.0, 0.0, 0.0, 6.0, 12.0, 2.0, 14.0, 5.0,
16.0, 20.0, 19.0, 19.0, 29.0, 11.0, 9.0, 21.0, 33.0, 13.0, 19.0, 16.0,
18.0, 12.0, 18.0, 16.0, 7.0, 16.0, 15.0, 34.0, 17.0, 20.0, 12.0, 11.0,
13.0, 15.0, 12.0, 3.0, 0.0, 1.0, 0.0, 0.0, 2.0, 3.0, 8.0, 4.0, 2.0, 0.0,
0.0]
label_0 = [0] * len(current_0)

current_1 = [5.4, 5.43, 5.42, 5.43, 5.42, 5.45, 5.46, 5.45, 5.48, 5.48,
5.49, 5.52, 5.54, 5.54, 5.55, 5.55, 5.56, 5.52, 5.49, 5.49, 5.48,
5.5,5.84,5.74,5.71,5.74,5.75,5.76,5.76,5.77,5.77,5.78,5.78,5.77,5.78,5.78,
5.77,5.77,5.75,5.77,5.77,5.77,5.77,5.79,5.77,5.78,5.79,5.81,5.79,5.79,5.77
,5.78,5.76,5.77,5.75,5.70,5.72,5.68,5.62,5.67,5.68]
vibrations_perSec_1 =
[259.00,251.00,256.00,254.00,260.00,296.00,248.00,253.00,216.00,258.00,247
.00,226.00,262.00,270.00,258.00,284.00,240.00,262.00,279.00,308.00,276.00,
274.00,252.00,246.00,262.00,297.00,246.00,228.00,214.00,207.00,198.00,178.
00,220.00,241.00,212.00,226.00,153.00,129.00,245.00,259.00,251.00,256.00,2
54.00,260.00,296.00,248.00,253.00,216.00,226.00,153.00,129.00,245.00,259.0
0,251.00,256.00,254.00,260.00,296.00,248.00,253.00,216.00]
label_1 = [1] * len(current_1)
current_2 = [7.5, 7.1, 6.9, 6.9, 7.9, 6.4, 7.0, 6.4, 7.7, 7.3,7.5, 7.1,
6.9, 6.9, 7.9, 6.4, 7.0, 6.4, 7.7, 7.3, 6.8, 7.8, 6.4, 7.1, 7.3, 7.9, 6.3,
6.5, 6.6, 6.9, 7.6, 7.7, 7.1, 7.2, 6.9, 7.3, 6.8, 7.8, 6.4, 7.2, 7.2, 7.2,
7.3, 6.5, 6.7, 6.2, 7.8, 7.2, 7.3, 7.6, 7.6, 7.2, 7.7, 6.3, 6.6, 6.2, 6.7,
7.1, 7.8, 6.2, 7.6]
vibrations_perSec_2 = [366.0, 436.0, 311.0, 416.0, 420.0, 343.0, 435.0,
382.0, 343.0, 396.0,366.0, 436.0, 311.0, 416.0, 420.0, 343.0, 435.0,
382.0, 343.0, 396.0, 426.0, 336.0, 312.0, 389.0, 442.0, 306.0, 426.0,
432.0, 437.0, 365.0, 413.0, 386.0, 370.0, 381.0, 378.0, 312.0, 321.0,
391.0, 440.0, 355.0, 437.0, 413.0, 305.0, 420.0, 418.0, 412.0, 340.0,
347.0, 350.0, 326.0, 412.0, 331.0, 355.0, 319.0, 444.0, 317.0, 421.0,
308.0, 353.0, 423.0, 307.0]
label_2 = [2] * len(current_2)
```

```
for i in range(len(current_1)):
 label_1.append(1)
current_0_1_2 = current_0+current_1+current_2
vibrations_perSec_0_1_2 =
vibrations_perSec_0+vibrations_perSec_1+vibrations_perSec_2
labe_0_1_2 = label_0+label_1+label_2
#distances[i] = (((newc1 - c1[i])(newc1 - c1[i]))+((newc2 - c2[i])(newc2 -
c2[i])))*1/2;
distances = []
n_curr = 6.36
n_vib = 90
for i in range(len(labe_0_1_2)-61):
    distances.append((((n_curr - current_0_1_2[i]) ** 2) + ((n_vib -
vibrations_perSec_0_1_2[i]) ** 2)) ** 0.5)
print(len(distances))
#now collecting labels of the smallest differences
smallest_1 = 999999
smallest_2 = 999999
smallest_3 = 999999
index_1 = 0
index_2 = 0
index_3 = 0
for i in range(len(distances)):
 if (distances[i]<smallest_1):
  smallest_3 = smallest_2
  index_3 = index_2
  smallest_2 = smallest_1
  index_2 = index_1
  smallest_1 = distances[i]
  index_1 = i
 elif (distances[i]<smallest_2):
  smallest_3 = smallest_2
  index_3 = index_2
  smallest_2 = distances[i]
  index_2 = i
 elif (distances[i]<smallest_3):
  smallest_3 = distances[i]
  index_3 = i
print("smallest distances
are",smallest_3,":",smallest_2,":",smallest_1,"and there indexes
are",index_3,":",index_2,":",index_1)
labels_of_smallest_distances =
[labe_0_1_2[index_1],labe_0_1_2[index_2],labe_0_1_2[index_1]]
print(labels_of_smallest_distances)
vote_for_0 = 0
vote_for_1 = 0
vote_for_2 = 0
for i in range(len(labels_of_smallest_distances)):
 if labels_of_smallest_distances[i] == 0:
  vote_for_0 += 1
 elif  labels_of_smallest_distances[i] == 1:
  vote_for_1 += 1
 elif  labels_of_smallest_distances[i] == 2:
  vote_for_2 += 1
print("vote/s for class o or Broken Belt is/are: ",vote_for_0)
print("vote/s for class 1 or Normal condition  is/are: ",vote_for_1)
print("vote/s for class 2 or Overload class is/are: ",vote_for_2)
if vote_for_0>vote_for_1 and vote_for_0>vote_for_2:
 print("new data belongs to class 0 or Broken Belt")
elif vote_for_1>vote_for_0 and vote_for_1>vote_for_2:
 print("new data belongs to class 1 or Normal condition")
else:
 print("new data belongs to class 2 or Overload class")

import matplotlib.pyplot as plt
# Separate data by class
class_0_data = [current_0, vibrations_perSec_0]
class_1_data = [current_1, vibrations_perSec_1]
class_2_data = [current_2, vibrations_perSec_2]
# Plot data for each class
plt.scatter(class_0_data[0], class_0_data[1], label='Class 0: Broken
Belt')
plt.scatter(class_1_data[0], class_1_data[1], label='Class 1: Normal
Condition')
plt.scatter(class_2_data[0], class_2_data[1], label='Class 2: Overload')

# Add labels and title
plt.xlabel('Current')
plt.ylabel('Vibrations per Second')
plt.title('Scatter Plot of Different Classes')

# Add legend and show plot
plt.legend()
plt.show()

# prompt: Generate DIfferent kinds of plots for the data
import matplotlib.pyplot as plt

# Scatter plot
plt.scatter(current_0_1_2, vibrations_perSec_0_1_2)
plt.xlabel("Current")
plt.ylabel("Vibrations per Second")
plt.title("Scatter Plot of Current vs. Vibrations per Second")
plt.show()

# Histogram of current
plt.hist(current_0_1_2)
plt.xlabel("Current")
plt.ylabel("Frequency")
plt.title("Histogram of Current")
plt.show()

# Histogram of vibrations per second
plt.hist(vibrations_perSec_0_1_2)
plt.xlabel("Vibrations per Second")
plt.ylabel("Frequency")
plt.title("Histogram of Vibrations per Second")
plt.show()

# Boxplot of current by class
plt.boxplot([current_0, current_1, current_2], labels=["Class 0", "Class
1", "Class 2"])
plt.xlabel("Class")
plt.ylabel("Current")
plt.title("Boxplot of Current by Class")
plt.show()

# Boxplot of vibrations per second by class
plt.boxplot([vibrations_perSec_0, vibrations_perSec_1,
vibrations_perSec_2], labels=["Class 0", "Class 1", "Class 2"])
plt.xlabel("Class")
plt.ylabel("Vibrations per Second")
plt.title("Boxplot of Vibrations per Second by Class")
plt.show()

# prompt: Generate DIfferent kinds of plots for the data
# Import necessary libraries
import matplotlib.pyplot as plt
# Create data for each class
class_0_data = [current_0, vibrations_perSec_0]
class_1_data = [current_1, vibrations_perSec_1]
class_2_data = [current_2, vibrations_perSec_2]

# Create a figure with three subplots
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

# Plot scatter plot for each class
axes[0].scatter(class_0_data[0], class_0_data[1], label='Class 0: Broken
Belt')
axes[0].set_title('Class 0')
axes[0].set_xlabel('Current')
axes[0].set_ylabel('Vibrations per Second')
axes[1].scatter(class_1_data[0], class_1_data[1], label='Class 1: Normal
Condition')
axes[1].set_title('Class 1')
axes[1].set_xlabel('Current')
axes[1].set_ylabel('Vibrations per Second')
```

```
axes[2].scatter(class_2_data[0], class_2_data[1], label='Class 2:
Overload')
axes[2].set_title('Class 2')
axes[2].set_xlabel('Current')
axes[2].set_ylabel('Vibrations per Second')
# Add legend and show plot
for ax in axes:
    ax.legend()
plt.tight_layout()
plt.show()
# Create a line plot for each class
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.plot(class_0_data[0], class_0_data[1], label='Class 0: Broken Belt')
plt.title('Class 0')
plt.xlabel('Current')
plt.ylabel('Vibrations per Second')
plt.subplot(1, 3, 2)
plt.plot(class_1_data[0], class_1_data[1], label='Class 1: Normal
Condition')
plt.title('Class 1')
plt.xlabel('Current')
plt.ylabel('Vibrations per Second')
plt.subplot(1, 3, 3)
plt.plot(class_2_data[0], class_2_data[1], label='Class 2: Overload')
plt.title('Class 2')
plt.xlabel('Current')
plt.ylabel('Vibrations per Second')
plt.tight_layout()
plt.show()
# Create a histogram for each class
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.hist(class_0_data[0], label='Class 0: Broken Belt')
plt.title('Class 0')
plt.xlabel('Current')
plt.ylabel('Frequency')
plt.subplot(1, 3, 2)
plt.hist(class_1_data[0], label='Class 1: Normal Condition')
plt.title('Class 1')
plt.xlabel('Current')
plt.ylabel('Frequency')
plt.subplot(1, 3, 3)
plt.hist(class_2_data[0], label='Class 2: Overload')
plt.title('Class 2')
plt.xlabel('Current')
plt.ylabel('Frequency')
plt.tight_layout()
plt.show()
```

- **For Temperature Prediction**

```
import pandas as pd
df= pd.read_csv('Temperature_data.csv')

df.head()

df = df.drop(df.columns[df.columns.str.contains('Unnamed', case=False)],
axis=1)
df.head(1)

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# Extracting features and target variable
x = df['Time'].values.reshape(-1, 1)
y = df['Temperature'].values

# Scatter plot of Temperature data vs Time
plt.scatter(x, y)
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.title('Scatter Plot of Temperature Data')
plt.show()

# prompt: suggest plots for the data
# Line plot of Temperature data vs Time
plt.plot(x, y)
plt.xlabel('Time')
plt.ylabel('Temperature')
plt.title('Line Plot of Temperature Data')
plt.show()

# Histogram of Temperature data
plt.hist(y)
plt.xlabel('Temperature')
plt.ylabel('Frequency')
plt.title('Histogram of Temperature Data')
plt.show()

# Boxplot of Temperature data
plt.boxplot(y)
plt.xlabel('Temperature')
plt.title('Boxplot of Temperature Data')
plt.show()

# Splitting the dataset into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

```
# Linear Regression model
model = LinearRegression()
model.fit(x_train, y_train)


# Predicting temperature for test data
y_pred = model.predict(x_test)

# Plotting actual vs predicted
plt.scatter(x_test, y_test, color='black', label='Actual data')
plt.plot(x_test, y_pred, color='blue', linewidth=3, label='Linear
regression line')
plt.xlabel('Time')  # Update x-axis label
plt.ylabel('Temperature')
plt.title('Linear Regression Example')
plt.legend()
plt.show()

# Model coefficients and intercept
print("Model coefficients:", model.coef_)
print("Model intercept:", model.intercept_)
time_to_be_predicted = 500

# Predicting temperature at a specific time (e.g., 50)
time_to_predict = np.array([[time_to_be_predicted]])  # Reshape into a 2D
array
predicted_temperature = model.predict(time_to_predict)
print("Predicted temperature at time :",(time_to_predict)/60,
predicted_temperature)
!pip install requests

#Model coefficients: [0.08839913]
#Model intercept: 19.960466458766604
#Predicted temperature at time : [[8.33333333]] [64.1600291]
Estimeted_time = []
i = 0
for i in range(10):
    i = i+120
    Estimeted_time.append(19.960466458766604+i*0.08839913)
print(Estimeted_time)

import requests
import time
# Replace with your ThingSpeak Write API Key
WRITE_API_KEY = 'O1ATXJKNB473MZTW'
# ThingSpeak API URL for updating a channel
THINGSPEAK_URL = 'https://api.thingspeak.com/update'
# Initial value for the data field
i = 0
pre_temp = 0

while True:
    # Update the data to be sent to ThingSpeak
    data = {
        'api_key': WRITE_API_KEY,
        'field4': pre_temp,  # Example field value (e.g., humidity)
    }
```

```
# Send the data to ThingSpeak
response = requests.post(THINGSPEAK_URL, data=data)

    # Check the response from ThingSpeak
    if response.status_code == 200:
        print('Data successfully sent to ThingSpeak.')
    else:
        print('Failed to send data to ThingSpeak. Status code:',
response.status_code)
    i = i +60
    pre_temp = 19.960466458766604+i*0.08839913
    # Update the data for the next iteration
    print(i)
    print(pre_temp)

    # Wait before sending the next update (e.g., 15 seconds)
    time.sleep(5)
i=0
while True:
  i = i + 1
  print(i)
  time.sleep(5)
```

- **For ML Algorithm in PyCharm**

```
Column1=[5.11, 5.11, 5.11, 5.12, 5.1, 5.09, 5.09, 5.1, 5.11, 5.1, 5.09,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.09, 5.1, 5.1, 5.1, 5.1, 5.1,
5.1, 5.08, 5.09, 5.1, 5.09, 5.1, 5.1, 5.1, 5.09, 5.09, 5.1, 5.1, 5.1,
5.1, 5.11, 5.09, 5.09, 5.1, 5.09, 5.08, 5.1, 5.1, 5.11, 5.11, 5.1, 5.09,
5.1, 5.09, 5.09, 5.09, 5.07, 5.08, 5.09, 5.09, 5.08, 5.08, 5.08, 5.08,
5.07, 5.06, 5.06, 5.08, 5.06, 5.06, 5.06, 5.06, 5.05, 5.05, 5.06, 5.06,
5.07, 5.04, 5.05, 5.05, 5.05, 5.05, 5.05, 5.05, 5.04, 5.04, 5.05, 5.06,
5.05, 5.04]
Column2= [45.0, 7.0, 11.0, 12.0, 36.0, 21.0, 20.0, 26.0, 22.0, 28.0, 8.0,
0.0, 1.0, 0.0, 0.0, 6.0, 12.0, 2.0, 14.0, 5.0, 16.0, 20.0, 19.0, 19.0,
29.0, 11.0, 9.0, 21.0, 33.0, 13.0, 19.0, 16.0, 18.0, 12.0, 18.0, 16.0,
7.0, 16.0, 15.0, 34.0, 17.0, 20.0, 12.0, 11.0, 13.0, 15.0, 12.0, 3.0,
0.0, 1.0, 0.0, 0.0, 2.0, 3.0, 8.0, 4.0, 2.0, 0.0, 0.0, 4.0, 8.0, 6.0,
9.0, 6.0, 2.0, 0.0, 1.0, 2.0, 1.0, 1.0, 4.0, 1.0, 1.0, 1.0, 0.0, 0.0,
1.0, 3.0, 3.0, 3.0, 3.0, 0.0, 3.0, 2.0, 1.0, 0.0, 0.0, 2.0, 1.0]
Column3  =[45.89, 46.01, 46.31, 46.35, 46.73, 46.85, 47.17, 47.23, 47.47,
47.55, 47.77, 47.65, 47.65, 47.09, 47.21, 47.19, 47.21, 47.23, 47.19,
48.03, 48.15, 48.39, 48.49, 48.61, 48.85, 48.99, 49.23, 49.31, 49.41,
49.49, 49.87, 50.11, 50.21, 50.41, 50.67, 50.47, 50.79, 51.15, 51.43,
51.37, 51.67, 51.85, 51.85, 57.15, 57.21, 57.59, 57.61, 57.89, 58.05,
58.25, 58.35, 58.21, 58.47, 58.59, 58.61, 59.05, 59.09, 59.51, 59.61,
59.77, 60.29, 60.47, 60.59, 60.75, 60.91, 60.89, 61.27, 61.61, 60.97,
61.65, 61.57, 61.99, 61.97, 62.25, 62.31, 62.53, 62.39, 62.73, 62.75,
62.73, 62.75, 63.01, 63.33, 63.35, 63.41, 63.57, 63.41, 63.73, 63.77]
```

```
#NOrmal condition
Column4=[5.4, 5.43, 5.42, 5.43, 5.42, 5.45, 5.46, 5.45, 5.48, 5.48, 5.49,
5.52, 5.49, 5.49, 5.48, 5.51, 5.53, 5.55, 5.53, 5.54, 5.54, 5.55, 5.55,
5.56, 5.52, 5.49, 5.49, 5.48, 5.51, 5.53, 5.55, 5.53, 5.54, 5.54, 5.55,
5.55, 5.56, 5.56, 5.55]
Column5= [204.0, 186.0, 202.0, 175.0, 193.0, 206.0, 173.0, 194.0, 202.0,
210.0, 216.0, 171.0, 221.0, 198.0, 196.0, 199.0, 180.0, 194.0, 225.0,
201.0, 231.0, 220.0, 238.0, 233.0, 171.0, 221.0, 198.0, 196.0, 199.0,
180.0, 194.0, 225.0, 201.0, 231.0, 220.0, 238.0, 233.0, 200.0, 215.0]
Column6= [56.13, 56.15, 56.33, 56.79, 56.91, 57.05, 57.17, 57.59, 57.81,
57.67, 57.97, 58.25, 58.39, 58.77, 58.85, 59.11, 59.31, 59.43, 59.71,
60.03, 59.71, 60.17, 60.23, 60.49, 58.25, 58.39, 58.77, 58.85, 59.11,
59.31, 59.43, 59.71, 60.03, 59.71, 60.17, 60.23, 60.49, 60.85, 60.85]
#Over load condition
col7 = [6.4, 6.4, 7.1, 7.5, 6.5, 7.2, 6.6, 7.5, 7.0, 6.4, 6.6, 6.5, 5.9,
5.9, 5.9, 6.3, 6.9, 6.5, 7.3, 7.1, 6.3, 7.1, 6.4, 6.2, 6.4, 5.9, 7.2,
6.2, 7.3, 7.2, 7.3, 7.4, 6.6, 6.9, 6.3, 6.7, 6.6, 6.3, 6.7, 7.1, 6.5,
7.1, 5.9, 6.1, 6.6, 6.4, 6.6, 6.4, 7.3, 6.5, 5.9, 6.7, 6.1, 6.2, 6.1,
6.7, 6.4, 7.3, 7.2, 6.4, 6.1, 6.5, 7.5, 7.4, 7.2, 7.0, 7.4, 7.2, 6.8,
7.3, 6.4, 6.9, 6.9, 7.1, 7.4, 7.2, 6.9, 6.6, 6.1, 6.0, 7.1]
col8 = [398.8, 351.8, 448.4, 369.9, 445.4, 420.4, 365.4, 478.8, 475.7,
463.6, 406.2, 379.1, 464.6, 464.4, 490.9, 404.6, 424.6, 486.7, 364.4,
370.0, 497.3, 458.9, 422.1, 476.9, 463.7, 392.2, 415.3, 425.9, 401.9,
466.2, 434.6, 384.6, 368.0, 350.9, 496.3, 471.9, 434.8, 421.3, 457.5,
483.1, 390.3, 496.5, 385.8, 397.9, 450.0, 385.8, 483.8, 372.2, 366.8,
362.7, 359.3, 476.4, 436.9, 400.9, 499.7, 409.8, 481.3, 470.2, 404.6,
413.6, 434.3, 356.9, 453.6, 403.7, 417.7, 455.8, 442.6, 366.4, 432.2,
354.1, 484.6, 380.9, 357.2, 413.8, 394.2, 426.3, 490.0, 397.5, 483.9,
455.4, 379.3]'''

import socket
import signal
import sys
# UDP server configuration
UDP_IP = "0.0.0.0"  # Bind to all available interfaces
UDP_PORT = 4425  # Port to listen on (must match the Arduino's
destination port)
ESP_IP = "192.168.4.190"  # IP address of the ESP8266 (replace with the
actual IP address)
ESP_PORT = 4425  # Destination port on the ESP8266

# Create a UDP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
if hasattr(socket, 'SO_REUSEPORT'):
    sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEPORT, 1)
    print("SO_REUSEPORT is available")

else:
    print("SO_REUSEPORT is not available")
```

```
      sock.bind((UDP_IP, UDP_PORT))
      print("UDP server up and listening...")

def signal_handler(sig, frame):
    print("\nExiting and closing the socket.")
    sock.close()
    sys.exit(0)
signal.signal(signal.SIGINT, signal_handler)
#current_0 vibrations_perSec_0

current_0=[5.11, 5.11, 5.11, 5.12, 5.1, 5.09, 5.09, 5.1, 5.11, 5.1, 5.09,
5.09, 5.1, 5.1, 5.1, 5.1, 5.1, 5.1, 5.08, 5.09, 5.1, 5.09, 5.1, 5.1, 5.1,
5.09, 5.09, 5.1, 5.1, 5.1, 5.1, 5.11, 5.09, 5.09, 5.1, 5.09, 5.08, 5.1,
5.1, 5.11, 5.11, 5.1, 5.09, 5.1, 5.09, 5.09, 5.09, 5.07, 5.08, 5.09,
5.09, 5.08, 5.08, 5.08, 5.08, 5.07, 5.06, 5.06, 5.08, 5.06, 5.06]
vibrations_perSec_0= [45.0, 7.0, 11.0, 12.0,11.0, 12.0, 36.0, 21.0, 20.0,
26.0, 22.0, 28.0, 8.0, 0.0, 1.0, 0.0, 0.0, 6.0, 12.0, 2.0, 14.0, 5.0,
16.0, 20.0, 19.0, 19.0, 29.0, 11.0, 9.0, 21.0, 33.0, 13.0, 19.0, 16.0,
18.0, 12.0, 18.0, 16.0, 7.0, 16.0, 15.0, 34.0, 17.0, 20.0, 12.0, 11.0,
13.0, 15.0, 12.0, 3.0, 0.0, 1.0, 0.0, 0.0, 2.0, 3.0, 8.0, 4.0, 2.0, 0.0,
0.0]
label_0 = [0] * len(current_0)

current_1 = [5.4, 5.43, 5.42, 5.43, 5.42, 5.45, 5.46, 5.45, 5.48, 5.48,
5.49, 5.52, 5.54, 5.54, 5.55, 5.55, 5.56, 5.52, 5.49, 5.49, 5.48,
5.5,5.84,5.74,5.71,5.74,5.75,5.76,5.76,5.77,5.77,5.78,5.78,5.77,5.78,5.78
,5.77,5.77,5.75,5.77,5.77,5.77,5.77,5.79,5.77,5.78,5.79,5.81,5.79,5.79,5.
77,5.78,5.76,5.77,5.75,5.70,5.72,5.68,5.62,5.67,5.68]
vibrations_perSec_1 =
[259.00,251.00,256.00,254.00,260.00,296.00,248.00,253.00,216.00,258.00,24
7.00,226.00,262.00,270.00,258.00,284.00,240.00,262.00,279.00,308.00,276.0
0,274.00,252.00,246.00,262.00,297.00,246.00,228.00,214.00,207.00,198.00,1
78.00,220.00,241.00,212.00,226.00,153.00,129.00,245.00,259.00,251.00,256.
00,254.00,260.00,296.00,248.00,253.00,216.00,226.00,153.00,129.00,245.00,
259.00,251.00,256.00,254.00,260.00,296.00,248.00,253.00,216.00]
label_1 = [1] * len(current_1)

current_2 = [7.5, 7.1, 6.9, 6.9, 7.9, 6.4, 7.0, 6.4, 7.7, 7.3,7.5, 7.1,
6.9, 6.9, 7.9, 6.4, 7.0, 6.4, 7.7, 7.3, 6.8, 7.8, 6.4, 7.1, 7.3, 7.9,
6.3, 6.5, 6.6, 6.9, 7.6, 7.7, 7.1, 7.2, 6.9, 7.3, 6.8, 7.8, 6.4, 7.2,
7.2, 7.2, 7.3, 6.5, 6.7, 6.2, 7.8, 7.2, 7.3, 7.6, 7.6, 7.2, 7.7, 6.3,
6.6, 6.2, 6.7, 7.1, 7.8, 6.2, 7.6]
vibrations_perSec_2 = [366.0, 436.0, 311.0, 416.0, 420.0, 343.0, 435.0,
382.0, 343.0, 396.0,366.0, 436.0, 311.0, 416.0, 420.0, 343.0, 435.0,
382.0, 343.0, 396.0, 426.0, 336.0, 312.0, 389.0, 442.0, 306.0, 426.0,
432.0, 437.0, 365.0, 413.0, 386.0, 370.0, 381.0, 378.0, 312.0, 321.0,
391.0, 440.0, 355.0, 437.0, 413.0, 305.0, 420.0, 418.0, 412.0, 340.0,
347.0, 350.0, 326.0, 412.0, 331.0, 355.0, 319.0, 444.0, 317.0, 421.0,
308.0, 353.0, 423.0, 307.0]
label_2 = [2] * len(current_2)
```

```
'''print(len(current_0))
print(len(vibrations_perSec_0))
print(len(label_0))
print(len(current_1))
print(len(vibrations_perSec_1))
print(len(label_1))
print(len(current_2))
print(len(vibrations_perSec_2))
print(len(label_2))
'''
current_0_1_2 = current_0 + current_1 + current_2
vibrations_perSec_0_1_2 = vibrations_perSec_0 + vibrations_perSec_1 +
vibrations_perSec_2
labels_0_1_2 = label_0 + label_1 + label_2

while True:
    try:
        # Receive data from the ESP8266
        data, addr = sock.recvfrom(1024)  # Buffer size is 1024 bytes
        sensor_data = data.decode()
        print(f"Received data: {sensor_data}")
        # Split the data based on commas
        sensor_values = sensor_data.split(',')
        # Ensure the data is in the expected format
        if len(sensor_values) != 3:
            print("Received data in unexpected format.")
            continue
        # Store the values in different variables
        current = float(sensor_values[0])
        vibrations_per_sec = float(sensor_values[1])
        temperature = float(sensor_values[2])

        # Print the values to verify
        print(f"Current in Amps Value: {current}")
        print(f"Vibrations/sec Value: {vibrations_per_sec}")
        print(f"Temperature value: {temperature}")

        # Calculate distances
        distances = []
        #n_curr = 5.36
        #n_vib = 80.0
        for i in range(len(labels_0_1_2)):
            distances.append((((current- current_0_1_2[i]) ** 2) +
((vibrations_per_sec - vibrations_perSec_0_1_2[i]) ** 2)) ** 0.5)
        # Find the three smallest distances
        smallest_1 = smallest_2 = smallest_3 = float('inf')
        index_1 = index_2 = index_3 = 0
        for i in range(len(distances)):
            if distances[i] < smallest_1:
                smallest_3 = smallest_2
                index_3 = index_2
                smallest_2 = smallest_1
                index_2 = index_1
                smallest_1 = distances[i]
                index_1 = i
            elif distances[i] < smallest_2:
                smallest_3 = smallest_2
                index_3 = index_2
                smallest_2 = distances[i]
                index_2 = i
            elif distances[i] < smallest_3:
                smallest_3 = distances[i]
                index_3 = i
        print("Smallest distances are", smallest_3, ":", smallest_2, ":", 
smallest_1, "and their indexes are", index_3, ":", index_2, ":", index_1)
        labels_of_smallest_distances = [labels_0_1_2[index_1], 
labels_0_1_2[index_2], labels_0_1_2[index_3]]
        print("Labels of smallest distances:", 
labels_of_smallest_distances)

        # Voting
        vote_for_0 = vote_for_1 = vote_for_2 = 0
        for label in labels_of_smallest_distances:
            if label == 0:
                vote_for_0 += 1
            elif label == 1:
                vote_for_1 += 1
            elif label == 2:
                vote_for_2 += 1
        print("Vote for Broken_belt (0): ", vote_for_0)
        print("Vote for Normal condition (1): ", vote_for_1)
        print("Vote for Overload condition (2): ", vote_for_2)

        if vote_for_0 > vote_for_1 and vote_for_0 > vote_for_2:
            print("New data belongs to class 0 (broken belt)")
            message = '1'
        elif vote_for_1 > vote_for_0 and vote_for_1 > vote_for_2:
            print("New data belongs to class 1 (normal condition)")
            message = '0'
        elif vote_for_2 > vote_for_0 and vote_for_2 > vote_for_1:
            print("New data belongs to class 2 (overload)")
            message = '2'
        # Send data to the ESP8266
        sock.sendto(message.encode(), (ESP_IP, ESP_PORT))
        print(f"Sent data to ESP: {message}")

    except Exception as e:
        print(f"An error occurred: {e}")
        break
sock.close()
```

Chapter 4
Testing and Validation/Discussion

This chapter comprehensively explains the meticulous testing and validation procedures implemented to guarantee the dependability and precision of our AI-powered motor fault prediction system. At first, we exposed the system to several controlled situations by using the wheat grinder machine as a load—this involved simulating scenarios with no, average, and overload loads. We meticulously observed every sensor, including the IR temperature, piezoelectric vibration, and current sensors, to gather extensive sets of data. We preprocessed and analyzed the data sets using linear regression and K-Nearest Neighbor (KNN) methods to detect patterns that suggest motor defects. We evaluated these algorithms by measuring their response time and accuracy in predicting outcomes. In addition, we have incorporated a digital replica to conduct more simulations and evaluate the system's ability to make accurate predictions in a virtual setting. We established that the system could accurately and promptly predict faults by conducting repeated testing and optimization, allowing for proactive maintenance interventions. The validation phase encompassed comparing anticipated problems with actual motor performance, enhancing the system based on received feedback, and guaranteeing its resilience in practical industrial environments.

4.1 Testing

Throughout the testing phase, we carried out a range of subcomponents, starting with calibrating each sensor. We smoothly incorporated these calibrated sensors into our project's prototype, enabling extensive data collection. During the last stage, our main objective was to improve the system's functionality by incorporating automated motor-tripping methods. This entailed creating algorithms and processes to identify deviations from standard patterns and initiate appropriate measures to protect the motor and adjacent equipment from potential harm or malfunction.

4.1.1 Prototypes

The prototype consists of a comprehensive sensor array strategically fitted into the motor assembly, which improves our capacity to monitor its performance and operational data. Following integration, we link the motor to a load, explicitly selecting a wheat grinder for our experimental setup. This experimentation phase entails exposing the motor to various operating situations, allowing us to gather comprehensive data and assess the motor's reaction in different scenarios. We aim to gain a more profound understanding of the motor's behavior and performance attributes by conducting thorough data collection and analysis. This will enable us to make educated decisions and develop optimization strategies for industrial applications.

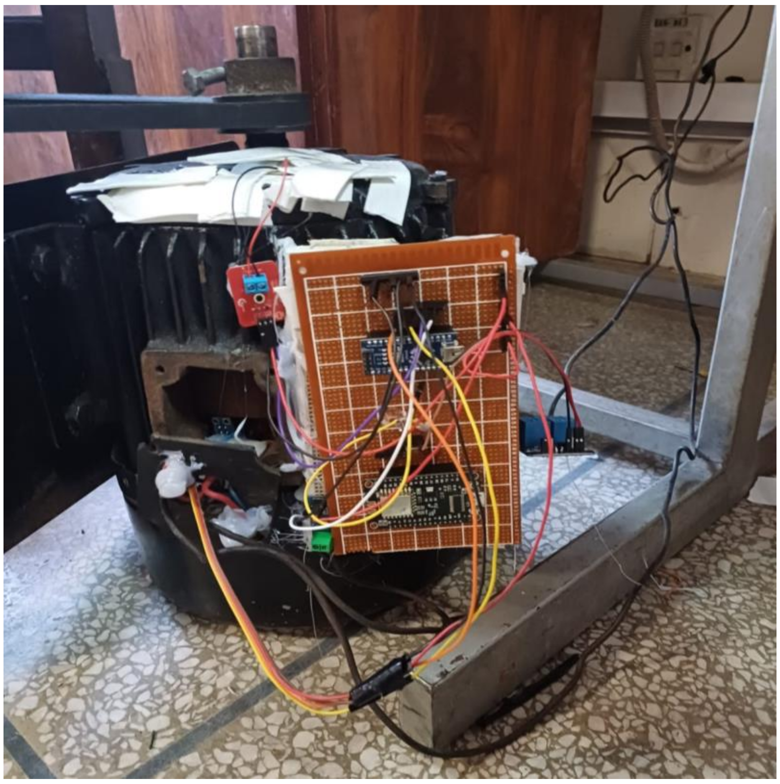


Fig 4. 1 Prototype with sensors integrated

4.1.2 Test Cases

Introduce the three conditions, i.e. with load, without load (anomaly). This data is collected using the thing Speak, and the concept of UDP, and then it is plotted. Since we drive the two cases.

- **Without Load**

  We labeled the data without load as an anomaly or fault, and our prototype consists of a motor with a pully, which is further attached to the load with the help of a belt. If the belt is broken in an industry, it is known as an anomaly or fault. When this kind of fault appears in the motor, there will be two instant changes in the motor, i.e., current and vibrations. Here is the table which will describe the readings in overload conditions.

Name of Sensor	Vibration	Current
Range of Readings	0-50 vib/sec	4.8-5.1 A

Table 1 Represent faulty data range

- **With Load**

When we put a load on the motor, we noticed dramatic changes in the vibrations and the current. Since the motor will make low vibrations, the current will raise to some point. Hence, the observed details of observed data:

Name of Sensor	Vibration	Current
Range of Readings	150-280 vib/sec	5.1-5.5 A

Table 2 Represent the range of non-faulty data

- **With Overload**

When we put an extra load on the motor with the help of the adjustable tightness bolt attached to the load, Then, at that stage, the vibration and current pattern will increase.. Hence, the observed details of observed data:

Name of Sensor	Vibration	Current
Range of Readings	300-600 vib/sec	Greater than 6 A

Table 3 Represent the range of overload conditioned data

4.2 Results/Output/Statistics

In the end we had visualize the data in the form of graph in things speak where stakeholders can easily view the real time data according to the conditions etc. Here is the graphical view of the data:

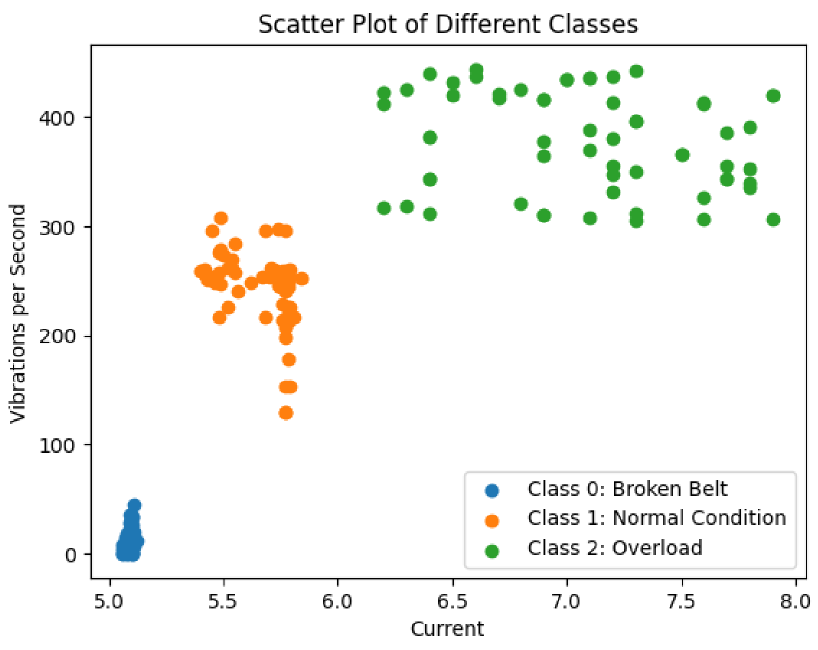


Fig 4. 2 Representation of Vibrations in different cases

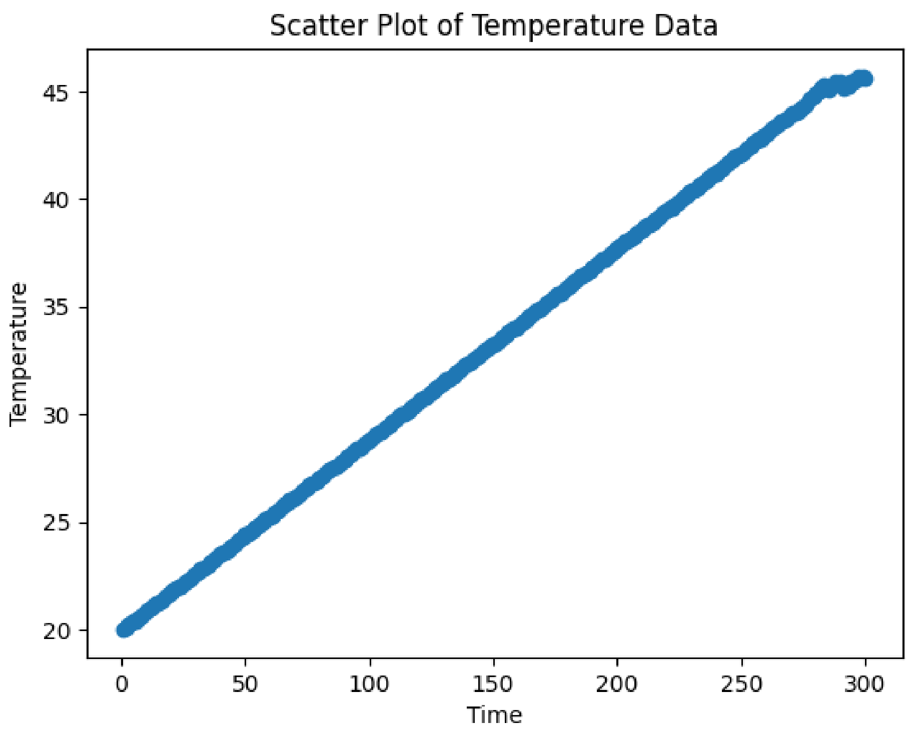


Fig 4. 3 Graph showing the behaviour of temperature

.

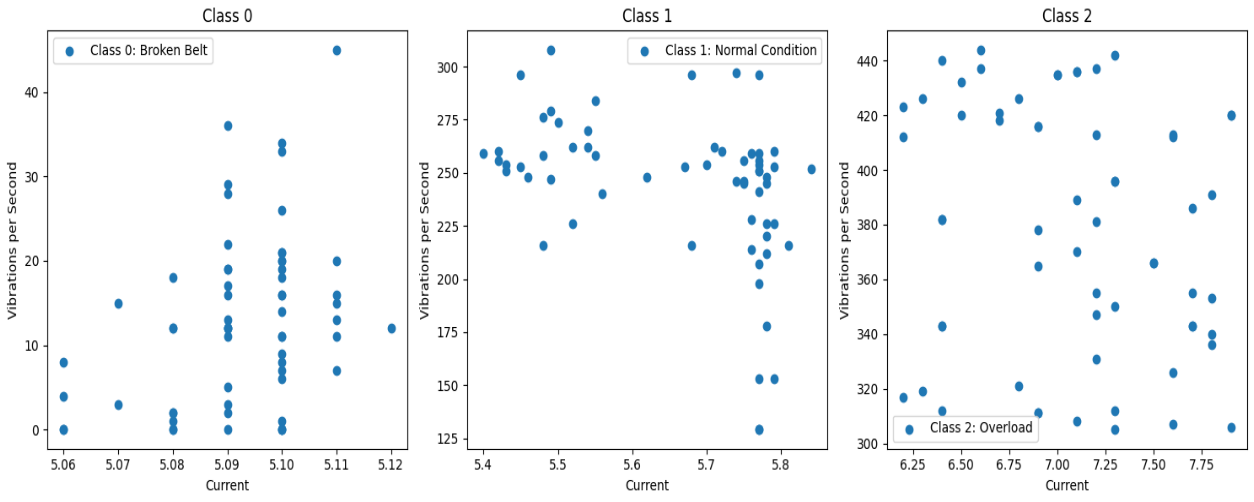
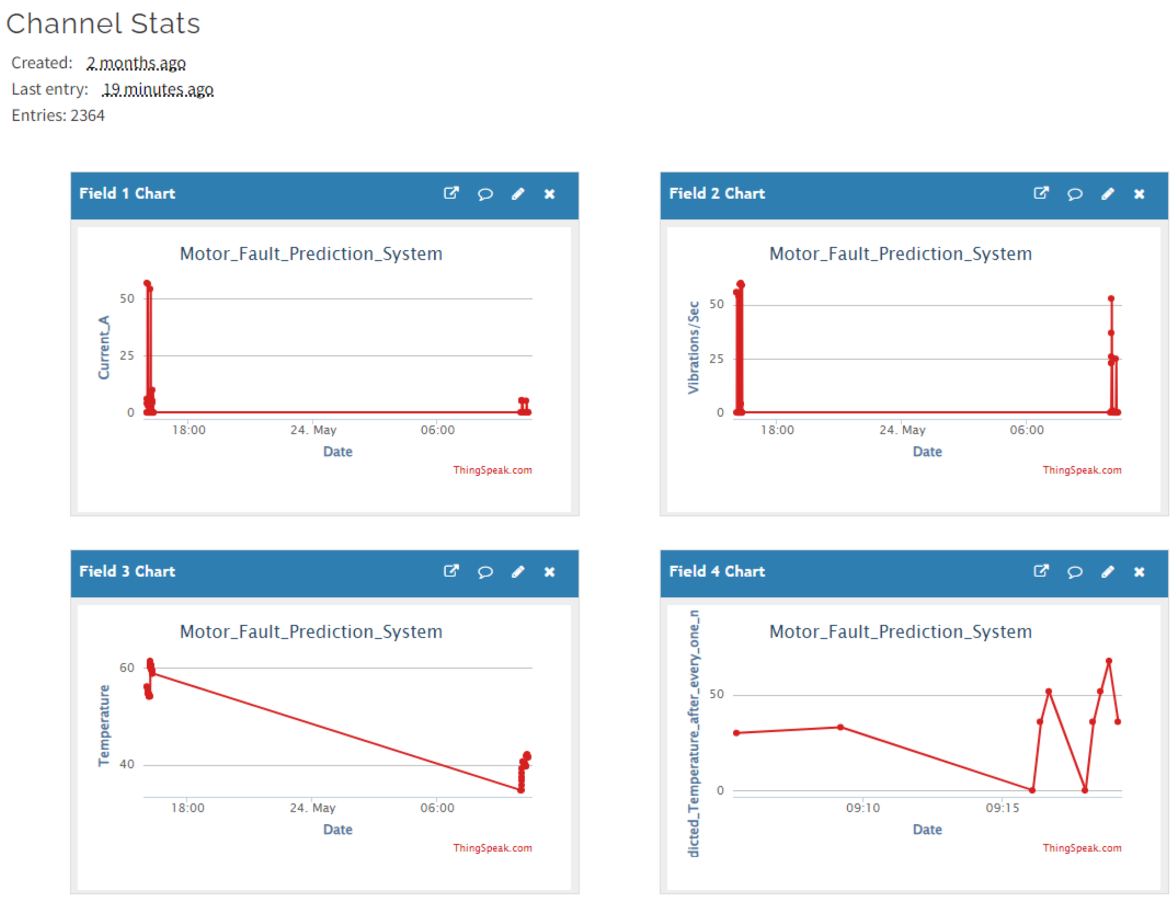


Fig 4. 4 Representation of Current in different cases


Channel Stats

Created: 2 months ago

Last entry: 19 minutes ago

Entries: 2364

Fig 4. 5 Plots showing the real time data on Things Speak

Chapter 5
Conclusions and Future Recommendations

5.1 Conclusions

This research introduces an AI-powered motor defect prediction system that aims to improve industrial processes by limiting setbacks and errors, resulting in reduced time and financial losses. The technology, which reduces the need for manual labor and minimizes the occurrence of human error, significantly reduces the expenses related to motor maintenance and repair. The system utilizes sensors for vibration, temperature, and current, along with advanced AI algorithms and IoT connectivity, to collect and analyze real-time data. This allows for the prediction of probable motor problems. This enables proactive maintenance and minimizes unexpected periods of inactivity.

AI algorithms analyze the sensor data to correctly predict motor problems, utilizing Python libraries such as sci-kit-learn and Pandas. Google Colab simplifies the creation and implementation of these machine-learning models by offering the required computational resources. By facilitating data visualization, MathWorks' open-source platform ThingSpeak enables stakeholders to monitor the condition of motors remotely using user-friendly interfaces. This groundbreaking approach offers substantial advantages for businesses in areas such as manufacturing, energy, transportation, and critical infrastructure. By implementing predictive maintenance solutions, companies can achieve cost reduction, increased productivity, and uninterrupted service. With the ongoing advancement of AI technology, industrial processes will experience more accuracy and precision, hence expanding the project's impact.

The AI-powered motor defect prediction system will assist industries in mitigating numerous setbacks and errors, resulting in significant reductions in time and financial waste. Due to its reduced susceptibility to human mistakes and lower requirement for manual labor, the implementation of this system will result in a decrease in the expenses associated with maintenance teams. In the future, artificial intelligence (AI) will undergo significant advancements, leading to enhanced accuracy and precision in various industrial processes. This project signifies a substantial advancement towards improving the efficiency and dependability of industrial operations.

5.2 Future Recommendations

Within our solution, we have thoroughly examined the data stored on the cloud. Although cloud computing has various benefits, it can be costly, particularly for small enterprises, because of the constraints of free cloud platforms and pay-as-you-go pricing models. To resolve this problem, we suggest executing the project directly on the ESP32 microcontroller. This would enable local data processing and minimize dependence on cloud services. Implementing this technique will enhance the cost-effectiveness and accessibility of the service for smaller enterprises.

Furthermore, the project has the potential to be applied in practical situations in large-scale industrial sectors. Implementing the technology in such circumstances can result in substantial reductions in both time and expense associated with motor maintenance. Enhancing the effectiveness of predictive maintenance and minimizing unplanned downtime will have a direct positive impact on industrial operations.

Finally, we suggest utilizing the obtained data to generate a simulated replica of the motor, sometimes referred to as a digital twin. Digital twin technology facilitates the generation of a precise digital duplicate of the motor, which enables the extraction of diverse data that is precisely calibrated to real-world scenarios. By utilizing this simulated data, we may improve our comprehension and control of motor performance, thus aiding in the attainment of Sustainable Development Goals (SDGs) through the promotion of effective resource allocation and the reduction of industrial waste.

References

[1]. Verma, Alok, Somnath Sarangi, and Maheshkumar H. Kolekar. "Stator winding fault prediction of induction motors using multiscale entropy and grey fuzzy optimization methods." Computers & Electrical Engineering 40.7 (2014): 2246-2258.

[2]. Huang, Yang, Chiun-Hsun Chen, and Chi-Jui Huang. "Motor fault detection and feature extraction using RNN-based variational autoencoder." IEEE access 7 (2019): 139086-139096.

[3]. Choudhary, Anurag, Deepam Goyal, and Shimi Sudha Letha. "Infrared thermography-based fault diagnosis of induction motor bearings using machine learning." IEEE Sensors Journal 21.2 (2020): 1727-1734.

[4]. Paolanti, Marina, et al. "Machine learning approach for predictive maintenance in industry 4.0." 2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA). IEEE, 2018.

[5]. Knight, Andrew M., and Sergio P. Bertani. "Mechanical fault detection in a medium-sized induction motor using stator current monitoring." IEEE Transactions on Energy Conversion 20.4 (2005)

[6]. Shao, Siyu, et al. "DCNN-based multi-signal induction motor fault diagnosis." IEEE Transactions on Instrumentation and Measurement 69.6 (2019)