Abstract

The ultimate objective of every industrial plant or vital utility plant is to increase output quantity and quality while keeping production costs as low as feasible. To do this, plants must be kept in peak operating conditions in order for the system's throughput to be maximized. The system must be properly maintained in order to remain completely functional. The plant's efficiency is maintained by a variety of maintenance measures. Maintenance has an impact on the cost of items produced in any sector. To avoid breakdowns, maintenance plans should be developed so that maintenance chores are completed at the appropriate times. Unnecessary maintenance chores raise maintenance expenses while also lengthening the time it takes to do them. Through this paper, the aim is to build an MES system that leverages the advantages of technologies such as Digital Twins, Machine Learning, Industrial internet of things, and Predictive Maintenance (PdM) together with Industry 4.0 principles. The goal is to optimize plant operation, i.e. reducing system downtime, which would result in lower production costs.

Contents

List of Tabels

List of Figures

List of Abbreviations

Sr. No.	Abbreviation	Expanded form
1	OEE	Overall Equipment Effectiveness
2	M2M	machine-to-machine
3	PdM	Predictive Maintenance
4	SME	Small and medium-sized enterprises
5	MES	Manufacturing execution system
6	OEE	Overall equipment effectiveness
7	MVP	Minimum viable product
8	RUL	Remaining Usable Life
9	LSTM	Long short-term memory
10	PV	Photovoltaics
11	RNN	Recurrent neural networks
12	GRU	Gated recurrent unit
13	IoT	Internet of Things
14	PLC	Programmable logic controller
15	SCADA	Supervisory control and data acquisition
16	GUI	Graphical user interface
17	CNN	Convolutional neural network
18	PCA	Principal component analysis
19	GAF	Gramian Angular Field
20	SVM	Support Vector Machine
21	VS	Vibration speed
22	MT	Motor torque

23	ACC	Acceleration
24	MS	Motor speed
25	AP	Air pressure
26	PW	Product weight
27	DEC	Deceleration
28	CUR	Current
29	BT	Belt tension
30	MT-S	Motor tension
31	TMP	Temperature
32	TCP/IP	Transmission Control Protocol/Internet Protocol
33	IIoT	Industrial internet of things
34	DFD	Data flow diagram
35	RMSE	Root-mean-square deviation
36	JSON	JavaScript Object Notation
37	MQTT	Message Queuing Telemetry Transport

Chapter 1

Introduction

With the constant evolution in the industry we have seen in the past from the introduction of steam power and mechanisation of production, electricity and assembly line production and partial automation using memory-programmable controls we are currently in the implementation of the Fourth Industrial Revolution. This is characterised by the application of information and communication technologies to industry and is also known as "Industry 4.0".

Production systems with computer technology are enhanced by a network link and, in a sense, have a digital twin on the Internet. These enable communication with other systems as well as the output of data about themselves. This is the next phase in the automation of production. All systems are connected, resulting in "cyber-physical production systems" and, as a result, smart factories, in which production systems, components, and people communicate across a network and production is practically autonomous.

Applying Industry 4.0 principles in the manufacturing lifecycle makes the process transparent, reduces downtime and unexpected maintenance related issues and this improves efficiency and reduces cost. Maintenance workers may now receive equipment documentation and service history more quickly and at the point of use, thanks to the growing use of digital devices inside factories and out in the field. Maintenance experts prefer to spend their time solving problems rather than waste time looking for technical information.

1.1 Description

The goal of this project is to monitor a factory floor and provide reports to the end user about the production status, machine health using Industry 4.0 principles. We will use an array of sensors to keep track of machine statistics and in case of faults report it to the user. This collected data allows us to run machine learning algorithms to predict future machine failures. An integrated production schedule will provide the completion status of a production order at any time eliminating the need for excessive paperwork and improving efficiency.

This will help us achieve the following objectives:
1) Production scheduling and full visibility through a web based system
2) Machine status and alarms enabled by IIoT helps with faster issue resolution
3) Predictive Maintenance leading to improvement in Overall Equipment Effectiveness (OEE) enabled by Machine Learning and Digital Twins

1.2 Problem Formulation

Traditional Paper based manufacturing systems need to transport reports to the plant floor from the shop floor and vice versa. This system is prone to human error, also requiring a report to physically arrive on the plant floor to begin manufacturing.

Organizations cannot afford any large or little breakdowns. It has an impact on the efficiency of corporate processes as well as the rate of growth. A single loose bolt leads to excessive wear and tear, shortened equipment life, and unplanned downtime.

As a result, you'll have higher recovery costs, lower production quality, no simultaneous device analysis, and no transparency in your manufacturing process. Smart technologies like big data, the internet of things, and machine-to-machine (M2M) communication methods are at the heart of the fourth industrial revolution, which aims to provide you with optimal automation.
Predictive Maintenance (PdM) combines three of 4IR's primary elements.

It employs real-time data intake to study the nature and extent of deterioration of an organization's amenities, extracts insights using machine learning algorithms, and generates reports on the machine's current state and upcoming difficulties.
With the right PdM tools, you can successfully address such issues before they become a problem, saving time, money, and quality.

The proposed system will provide a transparent system where the user can view the status of the machine and accordingly take predictive action.

1.3 Motivation

Larger industry players are actively implementing Industry 4.0 principles in their manufacturing process. Compared to the current industrial practices, application of Industry 4.0 principles shows to have a good impact with benefits such as greater productivity and better management of resources, more efficient decision-making based on real information, optimized and integrated productive processes. reduced downtime and reduced costs.

However these are players in the industry with large budgets and market capitalization.
The biggest threat to the SME sector in the event of failure to comply with the requirements of industry 4.0 is: loss of competitiveness due to a cost advantage that will not be as important as, for example, time of completion or quality. A big threat to SMEs is the lack of specialists who are taken over by large enterprises, and the education system is currently not adapted to
the new requirements of the labor market. The main barrier hindering the wider and systematic implementation of components of the Industry 4.0 concept is the lack of financial resources, lack of knowledge in the field of industry 4.0 and data protection.

Therefore keeping this in mind our project aims to make a minimum viable product to help convince SMEs to undertake the digital transformation using Industry 4.0 principles by showcasing an MES system that is based on a digital twin and also enables OEE through predictive maintenance.

1.4 Scope of the project

The proposed solution is aimed towards companies belonging to the small scale manufacturing industry sector. In this project the goal is to design a MVP (Minimum viable product) that will enable floor managers and maintenance technicians to know the status of the machines and maintenance requirements well in advance.

Chapter 2

Review of Literature

The following paragraphs contain the literature review of past papers and documentation on predictive maintenance using IIoT and Industry 4.0 principles.

Loubna Benabbou et. al. (2019) [1] wrote this paper based on the topic of predictive maintenance in the field of renewable energy. It describes the predictive maintenance process as well as the ML predictive maintenance framework for multi-component systems.

Predictive maintenance:

PdM seeks to forecast when a system will fail and to prevent it from happening by doing maintenance. It enables for the lowest possible maintenance frequency to avoid unplanned corrective repair while avoiding the costs of undertaking too much systematic preventive maintenance. PdM, according to experts, could cut maintenance costs by 10% to 40%. Critical components' availability, reliability, and security are also improved.

ML predictive maintenance framework for multi-components system

1. **Data Acquisition and Processing.**

   Data is collected through a set of sensors using various technologies. Special importance has to be given to data cleaning to eliminate errors that might affect the interpretation of degradation phenomena.

2. **Failure Detection**

   The initial step in the detection process is to distinguish between two types of failure: failure and no-failure. One can estimate the degree of the failure based on the assessment of any variation between normal and abnormal behaviour. This step helps you to isolate and identify the component that has ceased working.

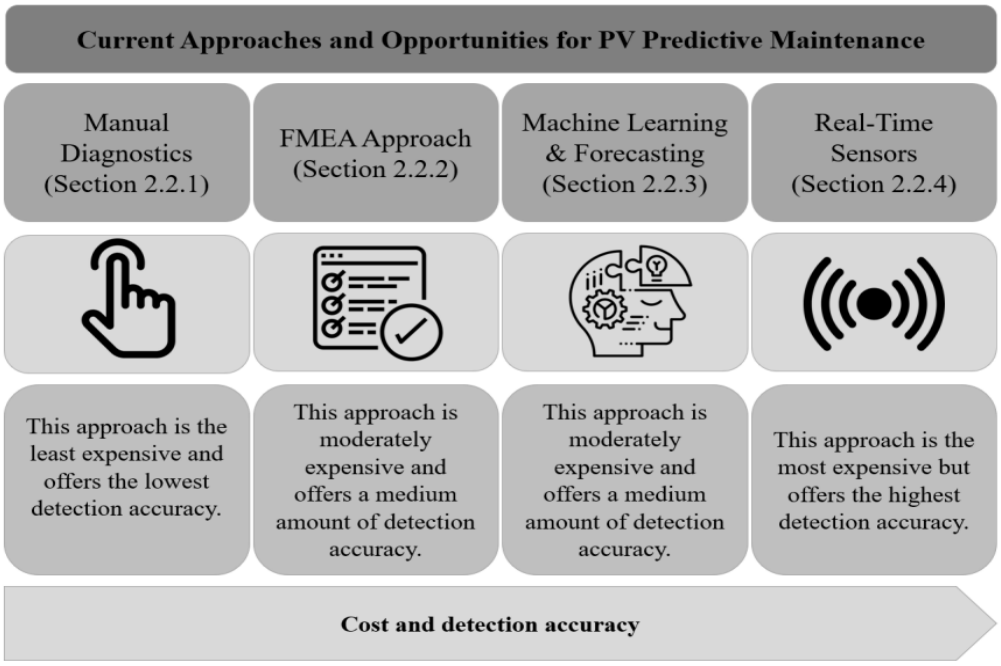
3. **RUL prediction**

They pointed out that the availability of large-scale datasets and the view of the RUL(Remaining Usable Life) problem as a time-series prediction are strong incentives to explore deep learning techniques in general, and LSTM in particular. In a number of applications of LSTM in RUL prediction, significant progress has been made.

Lisa B. Bosman et. al. (2020) [2] wrote a paper that contained information on how by installing wireless sensors and sending real-time data to a web platform, preventive maintenance, historical data analysis, and fault detection are facilitated. In another case of real-time monitoring with wired and wireless sensors and using programmable controllers, in this way, it is possible to monitor the performance of all the components of the PV system.

It proposed that PV system owners need an unbiased third-party off-the-shelf system-level predictive maintenance tool to optimize return-on-investment and minimize time to warranty claim in PV installations.

It also helped us understand the use of the system and its advantages to an industry but in this paper they specifically spoke about the PV cells.

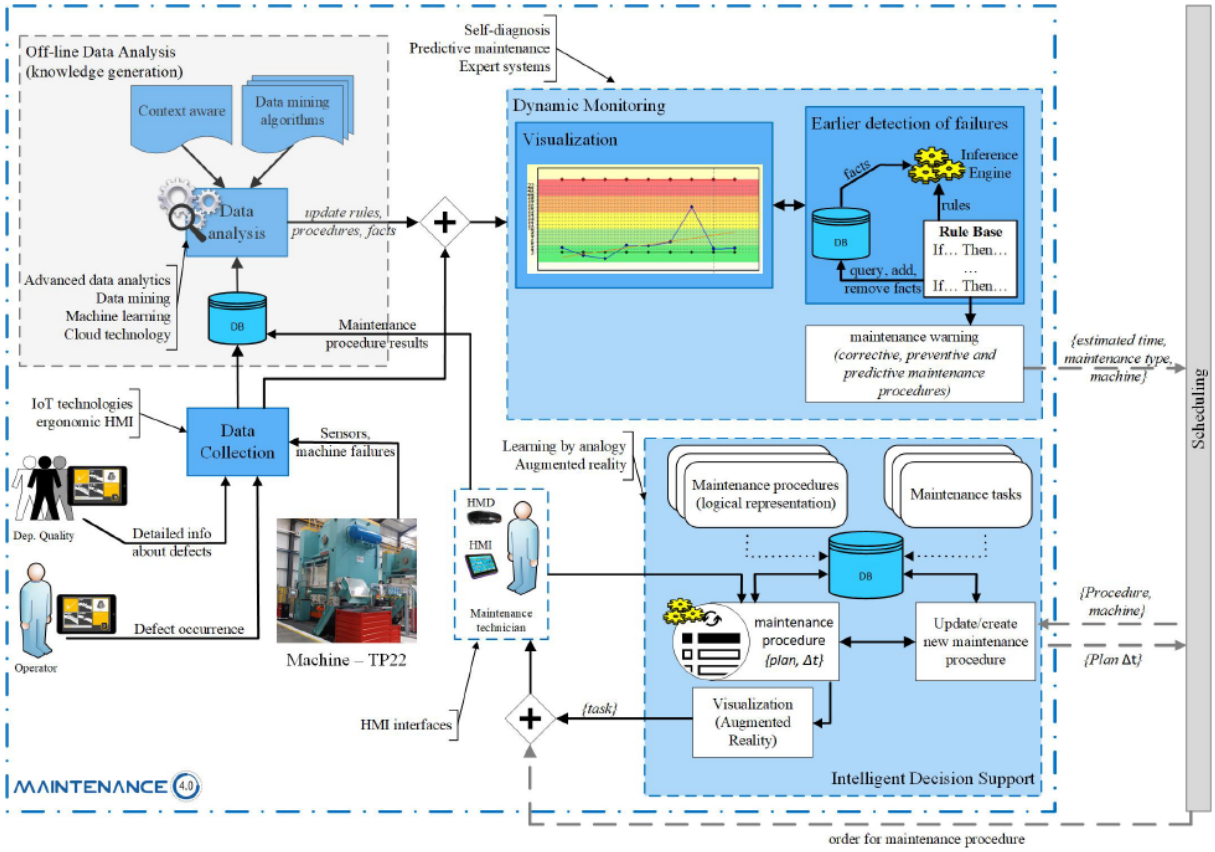


[2] Lisa B. Bosman, Walter D. Leon-Salas, William Hutzel and Esteban A. Soto, "PV System Predictive Maintenance: Challenges, Current Approaches, and Opportunities", Energies 2020, 13, 1398; doi:10.3390/en13061398. 2020

Fig. 2.1 Current Approaches and Opportunities for PV Predictive maintenance

Ana Cachada et. al. (2018) [3] proposed an architecture diagram that used the technology stack namely Internet of Things (IoT), cloud computing, advanced data analytics and augmented reality to perform predictive maintenance aligned with Industry 4.0 principles. They developed a deep machine learning approach with supervised learning for the early fault prediction and predictive maintenance using two types of recurrent neural networks (RNN), the long short-term memory (LSTM) and the gated recurrent unit (GRU) . The model was trained on 43000 past events with the Adam optimizer and binary cross entropy as loss function through 50 epochs. They had used pressure, temperature, humidity, vibration and operational noise when training the model on time series data. They also implemented an inference engine whose rules follow a simple structure based on the *If Condition then Action* syntax, which are processed by the inference engine and trigger an action when the rule is fired.

The visualization in their system was through the use of Charts and also through Augmented Reality.



Ana Cachada, José Barbosa et. al. , "Maintenance 4.0: Intelligent and Predictive Maintenance System Architecture", 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), 2018

Fig. 2.2 Maintenance 4.0: Intelligent and Predictive Maintenance System Architecture

Kahiomba Sonia Kiangala1 et al. (2018) [4] devised a decentralized model where Vibration data from a motor in a Bottling plant is collected through a PLC which communicates with a supervisory control and data acquisition (SCADA) graphical user interface (GUI) which instantaneously displays maintenance schedules and allows, whenever required, flexible configuration of new maintenance rules.

The vibration information is used to predict the health of the machine and based on the intensity a warning or alarm is issued on the GUI.

The vibration of the motors for various types of motor and their corresponding health classes was also gathered.

Class I: small-sized machines (from 0 to 15 kW) had a tolerance of 0.28 mm/s to 0.71 mm/s for an optimal health category. Vibration intensity above that created a warning displayed on the GUI.

This paper also proposes a decentralized monitoring system from which vibration speed states can be monitored on a cloud-based report accessible via the Internet; the decentralized monitoring system also sends instant email notifications to the intended supervisor for every maintenance schedule generated.

Kahiomba Sonia Kiangala1 et al. (2020) [5] built a classification model using a combination of time-series imaging and convolutional neural network (CNN) for better accuracy. They applied a feature extraction approach called principal component analysis (PCA) to reduce their dimensions to a maximum of two channels. The time-series are encoded into images via the Gramian Angular Field (GAF) method and used as inputs to a CNN model. In this paper they used both SVM and a CNN based model and got an accuracy of 55.2% and 100% respectively. The model incorporated the following features Vibration speed (VS), Motor torque (MT), Acceleration (ACC), Motor speed (MS), Air pressure (AP), Product weight (PW), Deceleration (DEC), Current(CUR), Belt tension (BT), Motor tension (MT-S) and Temperature (TMP).

Chapter 3

System Analysis

3.1 Functional Requirements:

The functional requirement describes the core functionality of the system. According to the functions of the system it's divided into three subsystems as listed below. The functional requirement of each subsystem is mentioned along with it. This section includes the data and functional process requirements.

3.1.1 Input subsystem

- Data is generated from sensors :
    - Vibration sensor
    - Rotary encoders
    - Temperature and humidity sensor.

3.1.2 Processing subsystem

- Fetch Real Time data
- Periodically perform data analysis on historical data to calculate RUL
- Fault detection

3.1.3 Output subsystem

- Display Remaining usable life.
- Display Data visualization in form on Chart
- Fault Alert and schedule maintenance

3.2 Non-Functional Requirements

The non-functional requirements of the system are as follows:

- Accessible – Accessible as it is a web application.

- Performance: The response time of the system will be relatively fast i.e., the system adapts to parameter changes and detect faults
- Reliable – Consistent and reliable quality of service.
- Maintainability: The system will be developed such that it can easily extend to incorporate additional functionalities.
- Contents and Design – Design and contents of the system are satisfactory to users.
- Availability: The system will be available to the user 24x7
- Usability: The system will provide a user-friendly interface.

3.3 Specific Requirements

3.3.1 Software Requirements:

- Operating System: Windows or Linux
- Languages used: Python 3, Docker, Nodejs, Ditto, Postgres database
- Frameworks used: React

3.3.2 Hardware Requirements:

- CPU: Intel i3/Ryzen 3 or above
- Clock speed: 2.5 GHz or above
- GPU: NVIDIA Geforce GTX 960 or above
- RAM size: 4GB or above Hard Disk capacity: 100GB or above
- Component: ESP8266 micro controller
    - Vibration Sensor.
    - Temperature Sensor.
    - Motor
    - Wifi Module (ESP8266)
    - Rotary encoders

ESP8266

The ESP8266 is a built-in TCP/IP networking software, and microcontroller capability. This small module allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections using Hayes-style commands. The ESP8285 is a similar chip with a built-in 1 MiB flash memory, allowing the design of single-chip devices capable of connecting via Wi-Fi.

Fig. 3.1 ESP8266

The pinout is as follows for the common ESP-01 module:

- GND, Ground (0 V)
- GPIO 2, General-purpose input/output No. 2
- GPIO 0, General-purpose input/output No. 0
- RX, Receive data in, also GPIO3
- VCC, Voltage (+3.3 V; can handle up to 3.6 V)
- RST, Reset
- CH_PD, Chip power-down
- TX, Transmit data out, also GPIO1

Rotary encoder

A rotary encoder, also called a shaft encoder, is an electro-mechanical device that converts the angular position or motion of a shaft or axle to analog or digital output signals. There are two main types of rotary encoder: absolute and incremental. The output of an absolute encoder indicates the current shaft position, making it an angle transducer. The output of an incremental

encoder provides information about the motion of the shaft, which typically is processed elsewhere into information such as position, speed and distance

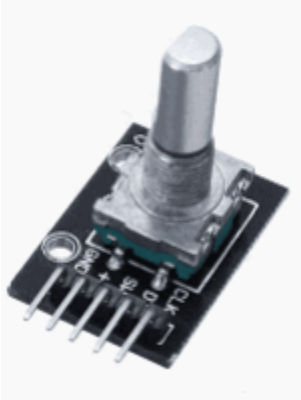


Fig. 3.2 Rotary Encoder

Vibration Sensor

Vibration Sensor (SW-420) is a high sensitivity non-directional vibration sensor. When the module is stable, the circuit is turned on and the output is high. When the movement or vibration occurs, the circuit will be briefly disconnected and output low.

Fig. 3.3 SW-420

DHT11

The DHT11 is a commonly used Temperature and humidity sensor that comes with a dedicated NTC to measure temperature and an 8-bit microcontroller to output the values of temperature and humidity as serial data. The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed).

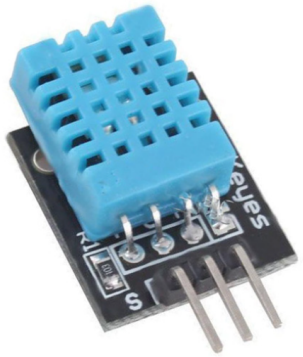


Fig. 3.4 DHT 11

3.4 Use-Case diagram and description

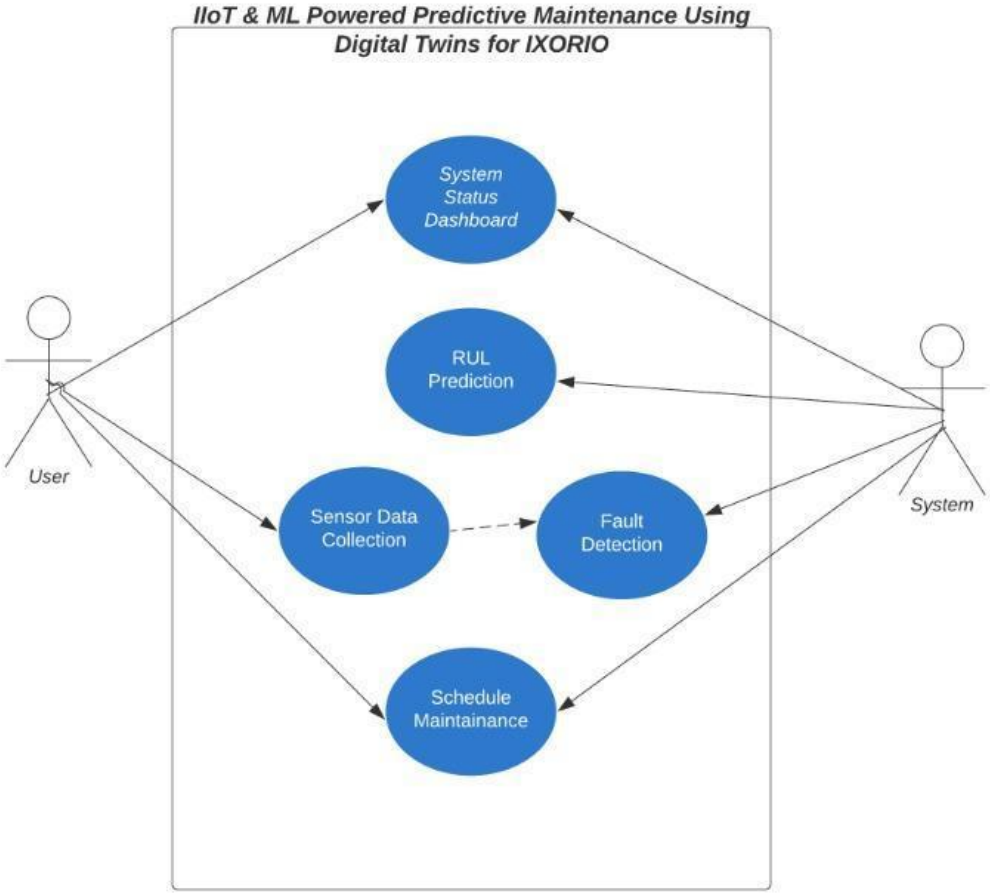


Fig. 3.5 Use-Case Diagram

Use Case Diagram:

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and the use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a “System” is something being developed and operated, such as a website. The “Actors” are people or entities operating under defined roles within the system. An example of this with reference to this system is shown in Fig. 3.5.

Use Case Specifications:

Use case: System status dashboard

Brief Description: The system is expected to input several data from IIoT sensors, analyze and visualize with the help of web application using E-chart and graphs. Whenever system finds fault or abnormality it will display alert on dashboard and send message/notification to user

Primary Actor: System, User

Main Flow:

- The user will be able to view the current status of the machine based on readings from a digital twin.
- The system displays a chart comparison based on data records.

Use case: Sensor data collection.

Brief Description: The system is expected to collect data from sensors installed on the machine, analyze the data and detect the fault.

Primary Actor: System

Main Flow:

- The system will collect sensor data
- The system will detect faults by analysing past and current data .

Use case: Schedule Maintenance.

Brief Description: The user will be able to view the current schedule and make changes to the schedule.

Primary Actor: User, System

Main Flow:

- Enter an order list and auto scheduling will be done.
- The user will be able to schedule maintenance.

Chapter 4

Analysis Modeling

4.1 DFD

Fig. 4.1.1 shows the level 0 DFD of the overall system for yield enhancement. The web interface is the interaction point for the user, from here the user can view the dashboard and any updates from the system such as RUL prediction and Fault Detection. The motor is also another entity in the system which has sensors that enable real time data updates to the MES system.

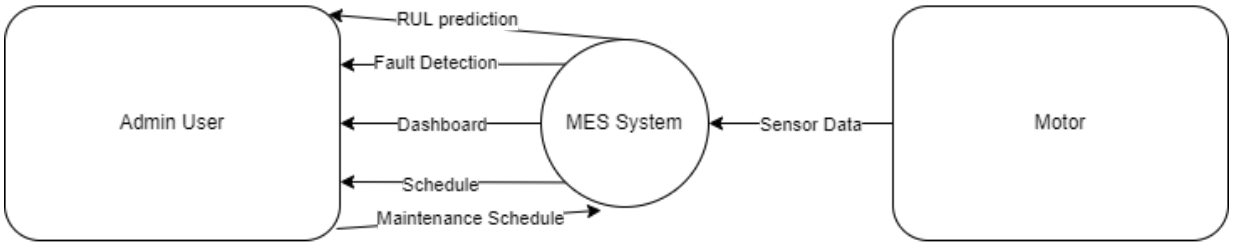


Fig 4.1 Level 0 DFD

Level 1 DFD provides a deeper understanding of the complete flow of data throughout the system. The user can access the Dashboard full of interactive charts. The motor has sensors attached that sends real time data to the MES system and this is then stored in the database fro model learning. This same data is updated in the digital twin which is used for both Fault detection and RUL prediction.

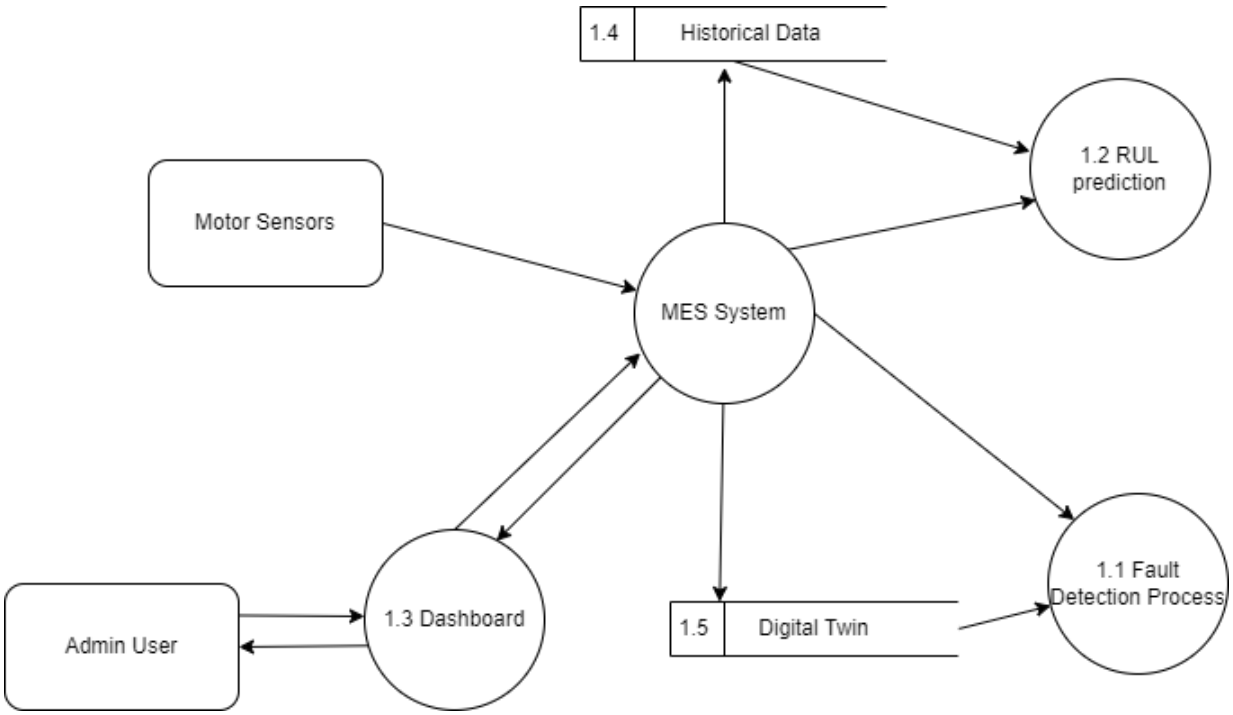


Fig 4.2 Level 1 DFD

The level 2 diagram is a more detailed explanation of the system. The Admin user uses the Web Interface that can perform four functionalities. The interface displays statistical data, warnings, results of RUL prediction and also can display the current schedule. The Cloud interface gets live sensor data from the machine and maps it to the digital twin and stores it in the database. This data is then fed to the maintenance, RUL and fault detection system and the analysis is done on the input data. This data is then displayed on the Dashboard.

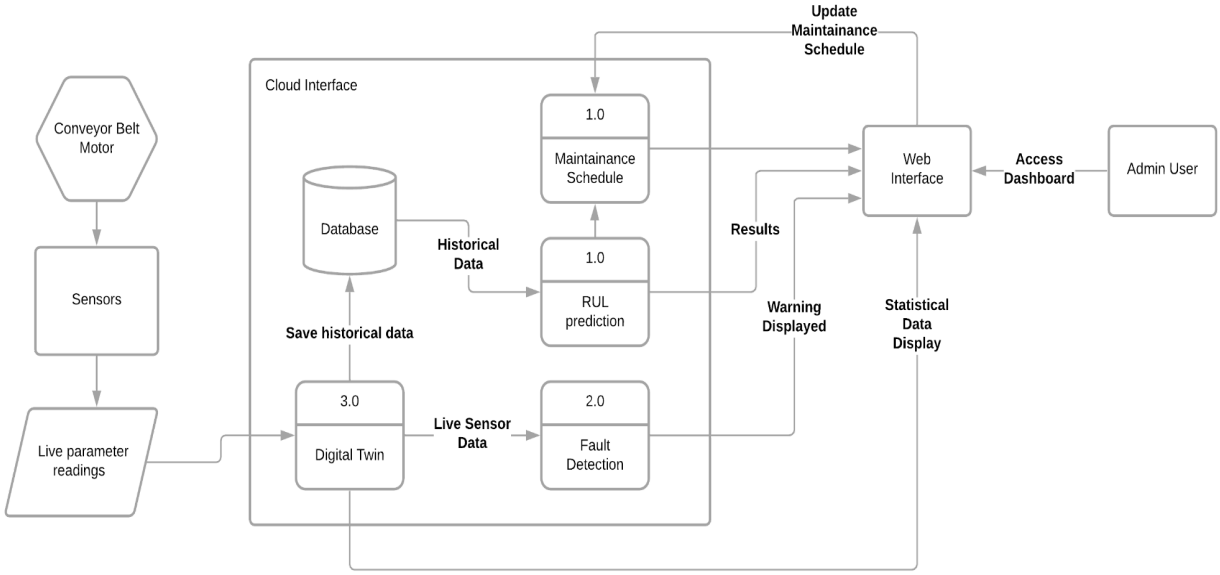


Fig. 4.3 Level 2 DFD

4.2 Activity Diagram

This is the activity diagram which shows the dynamic view of the system. The authenticated user will be able to view machine status,view production schedule meanwhile the machine will collect sensor data and send real time data to the end user.

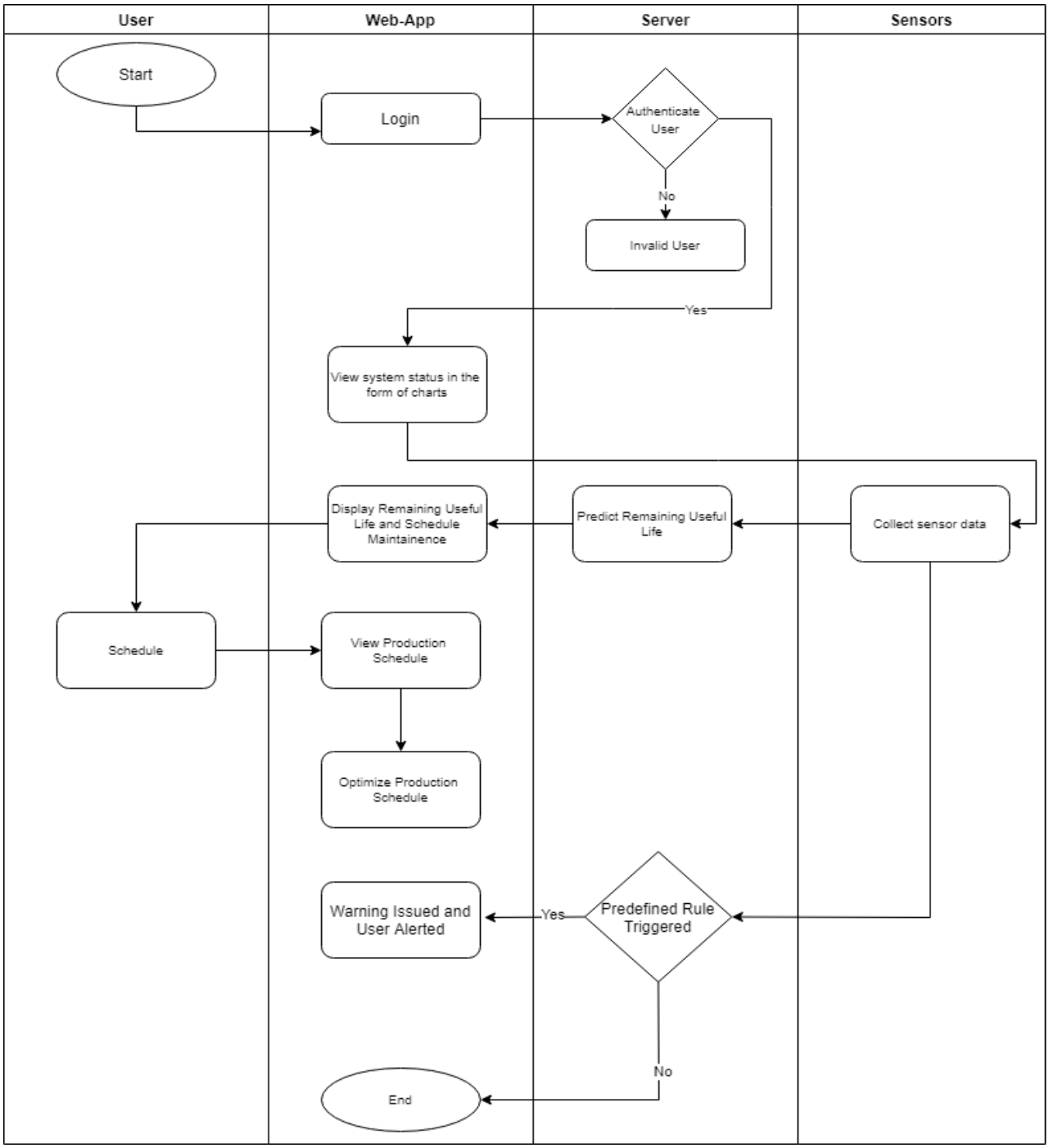


Fig. 4.4 Activity diagram

4.3 Timeline Chart

Timeline Chart that consists of the list of events in chronological order. It is a visualization of the tasks involved in the development of the system. The first column lists all the tasks. The adjoining column consists of long bars explaining the duration of the task.

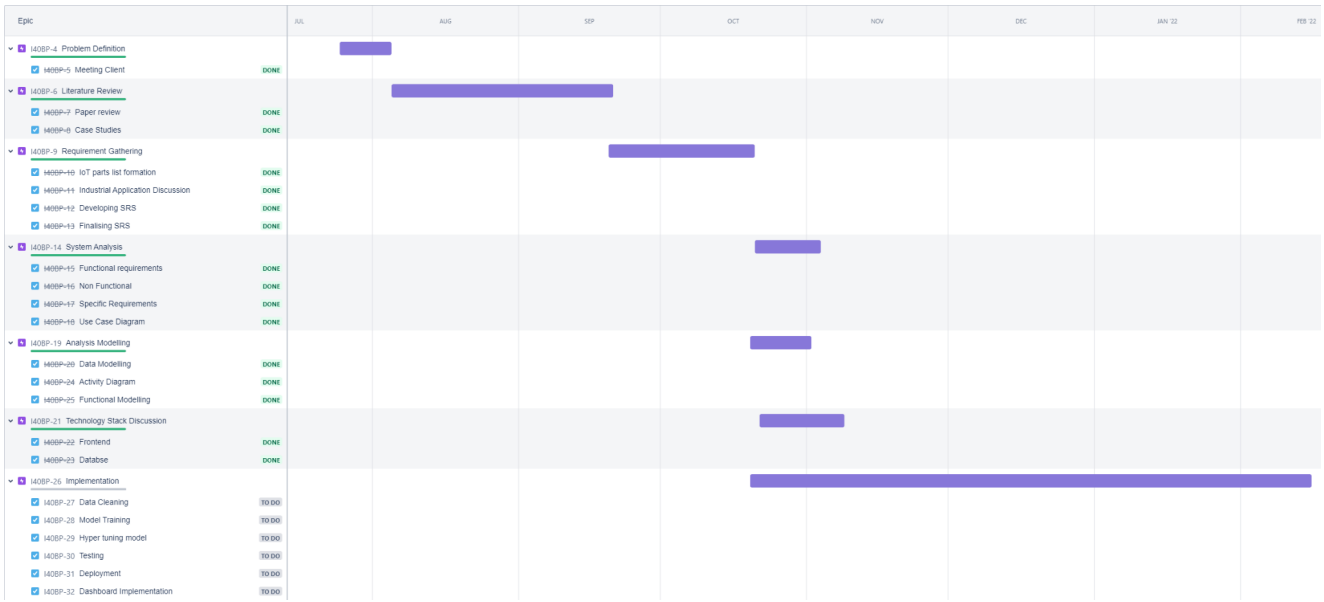


Fig. 4.5 Timeline Chart

Chapter 5

Design

5.1 Architectural Design

This is the architecture diagram which shows the abstract overview of components in the system and the relationship between them. Sensors connected to a machine send real-time data to the digital twin, this data is saved in a database to be used in data analysis. The user interacts with the dashboard to remotely monitor the system vitals. Any faults in the system will be displayed through notifications and alerts.

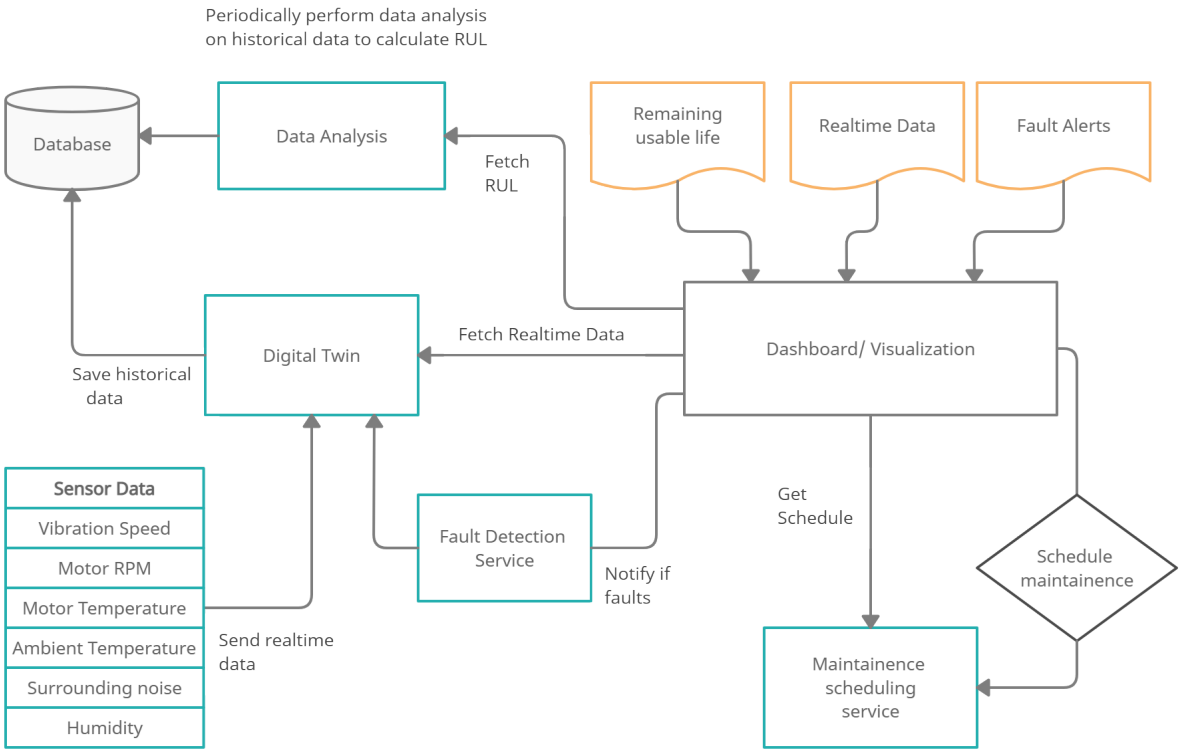


Fig. 5.1 Architecture design

5.2 User Interface

In Fig. 5.2 The system's dashboard is displayed, showing the temperature, speed, and vibration of the machine along with the predicted Remaining Useful Life

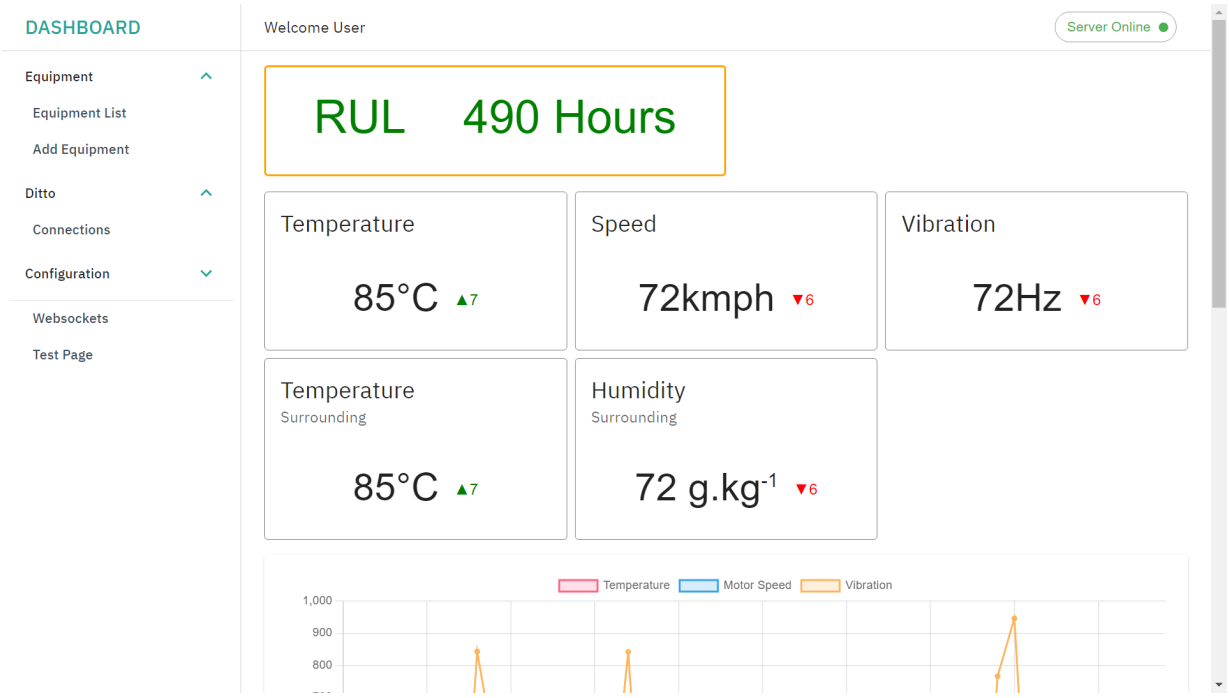


Fig. 5.2 Dashboard Display - Sensor Data and RUL

In Fig. 5.3 The system's dashboard is presented, with graphs depicting the machine's temperature, speed, and vibration.

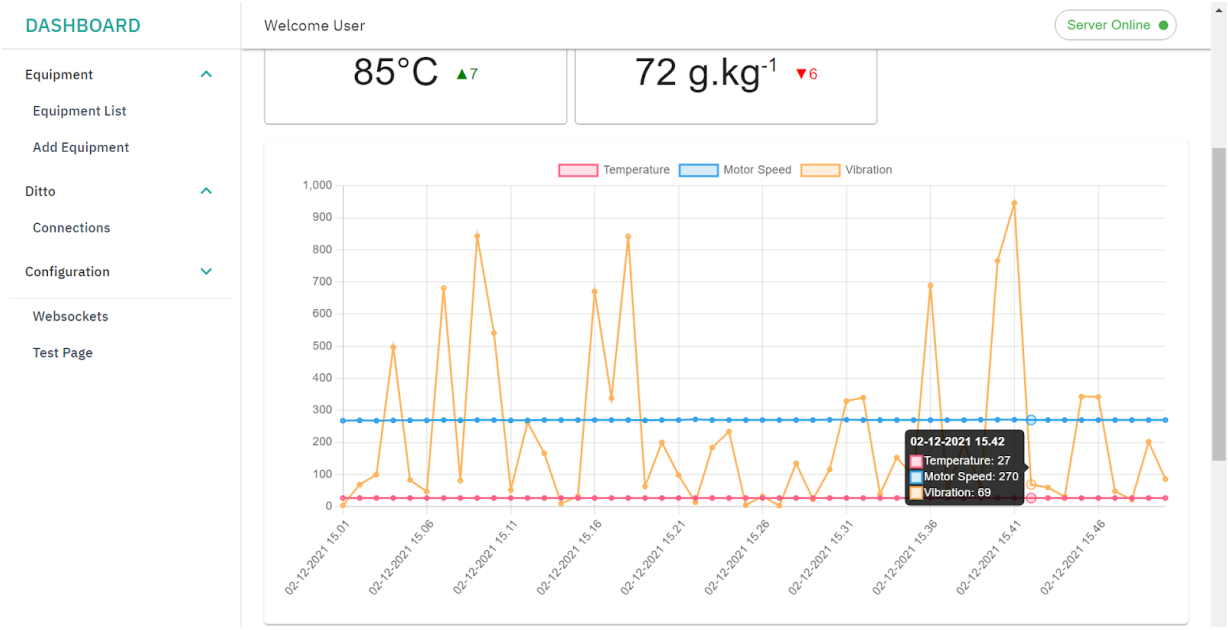


Fig. 5.3 Dashboard Display - Sensor Data Plot

The Fig. 5.4 The system displays the equipment list.

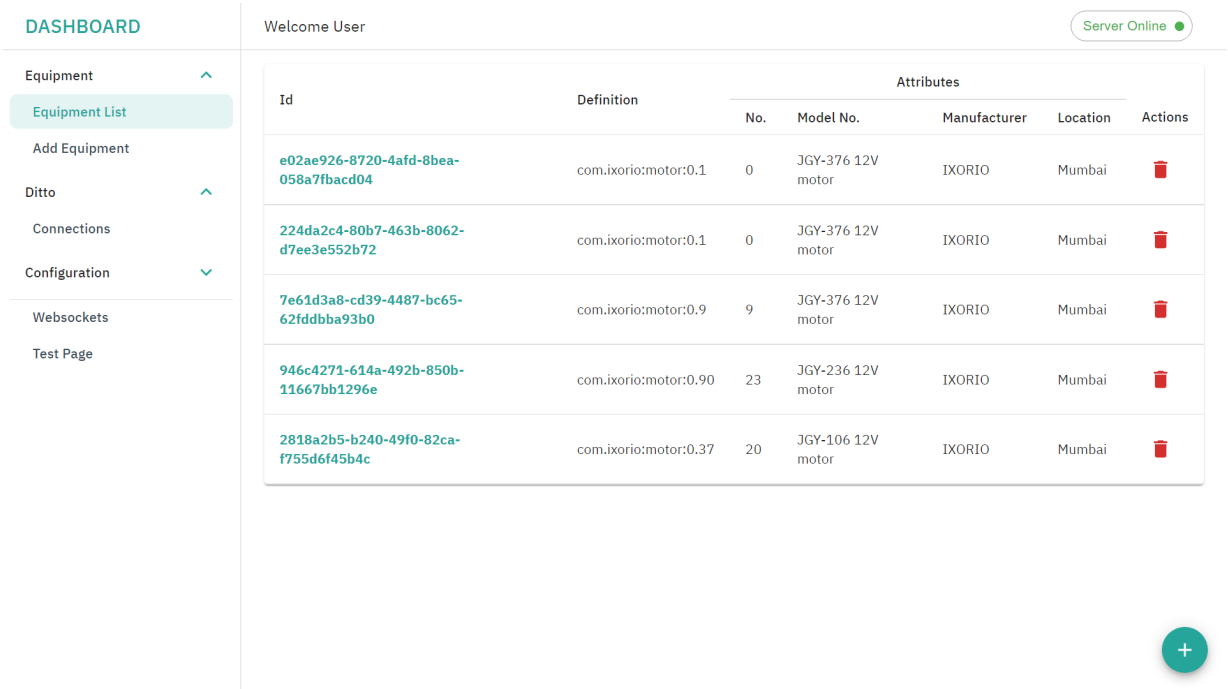


Fig. 5.4 Dashboard Display - Equipment List of Digital Twin

InFig.5.5 The system displays the maintenance schedules for various machines.

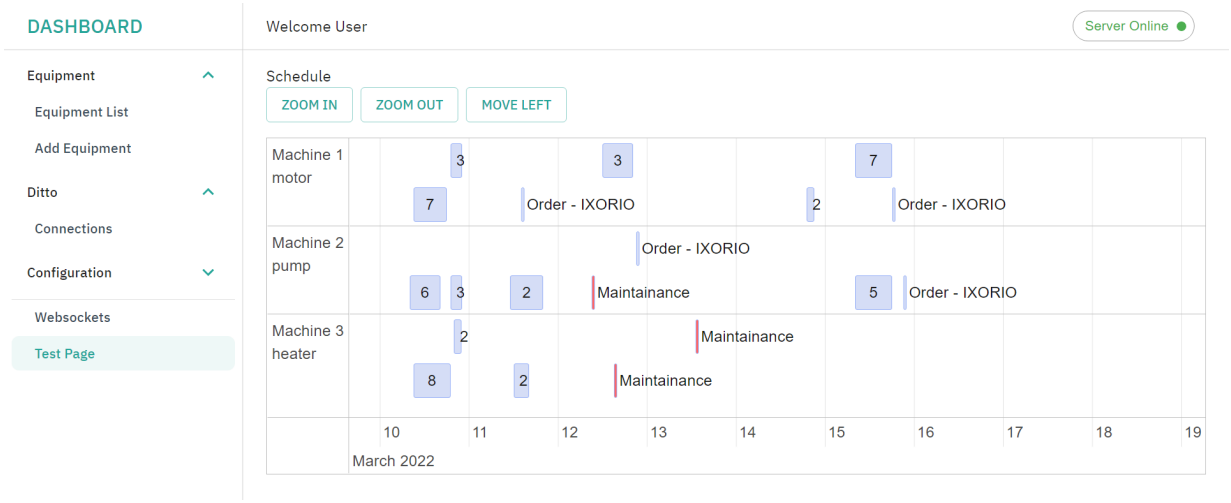


Fig. 5.5 Schedule of orders

Chapter 6

Implementation

6.1 Algorithms / Methods Used

Data Preprocessing-

Scaling - One of the most well-known ways of standardizing data is MinMaxScaler standardization. For each component, the base guess is set to 0, the most extreme value is set to 1, and all other values are set to a decimal between 0 and 1. Since the data only contained the timestamp and the sensor readings, and the methodology used in this project is that of supervised learning therefore the RUL had to be imputed from the time stamp at data read and the timestamp at the time of failure.

LSTM- The neural network model utilized in this study is based on the LSTM model, which uses memory cell neurons to tackle the problem of vanishing gradient and contains a chain of neural network modules in the shape of a chain. The input gate regulates the amount of fresh data that may be added to the cell state, the output gate regulates the cell's output data, the forget gate regulates the information that should be stored by the cell state, and the cell state is used to store relevant data.

Pseudo Code:

Step 1, in each iteration the past n values are passed as an array into the LSTM network. Each iteration produces a value that is the predicted RUL.

Step 2, take the predicted RUL given , the output by the LSTM network , and the real RUL value taken from the dataset and find the mean squared error loss

Step 3, Adjust the weight parameter of the network to minimize the mean squared error loss.

Step 4, repeat step 2-3, and set a callback such that the training stops when the MSE value does not change or increase for the last 5 epochs

Step 5, The performance data is passed to the trained network and the results are stored. The RMSE is calculated between the predicted RUL and the real RUL. Through the RMSE

evaluate the trained model is good or bad. If the RMSE is not satisfactory then the model is trained with different hyperparameters.

```
def model_builder(hp):
    model = Sequential()
    model.add(LSTM(hp.Int('input_unit',min_value=32,max_value=512,step=32), return_sequences=True, input_shape= (trainX.shape[1], trainX.shape[2])))
    model.add(Dropout(0.1))
    model.add(BatchNormalization())
    for i in range(hp.Int('n_layers', 1, 4)):
        model.add(LSTM(hp.Int(f'lstm_{i}_units',min_value=32,max_value=512,step=32),return_sequences=True))
        model.add(Dropout(hp.Float('Dropout_rate',min_value=0,max_value=0.5,step=0.1)))
        model.add(BatchNormalization())
    model.add(LSTM(hp.Int('layer_2_neurons',min_value=32,max_value=512,step=32)))
    model.add(Dropout(hp.Float('Dropout_rate',min_value=0,max_value=0.5,step=0.1)))
    model.add(BatchNormalization())
    model.add(Dense(30, activation='tanh'))
    model.add(Dense(1, activation='tanh'))

    model.compile(loss='mse', optimizer='adam',metrics = ['mse'])

    return model
```

Fig. 6.1 Code for LSTM model

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 52, 224)           228480
_________________________________________________________________
dropout (Dropout)            (None, 52, 224)           0
_________________________________________________________________
batch_normalization (BatchNo (None, 52, 224)           896
_________________________________________________________________
lstm_1 (LSTM)                (None, 52, 320)           697600
_________________________________________________________________
dropout_1 (Dropout)          (None, 52, 320)           0
_________________________________________________________________
batch_normalization_1 (Batch (None, 52, 320)           1280
_________________________________________________________________
lstm_2 (LSTM)                (None, 512)               1705984
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 512)               2048
_________________________________________________________________
dense (Dense)                (None, 30)                15390
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 31
=================================================================
Total params: 2,651,709
Trainable params: 2,649,597
Non-trainable params: 2,112
```

Fig 6.2 LSTM model summary

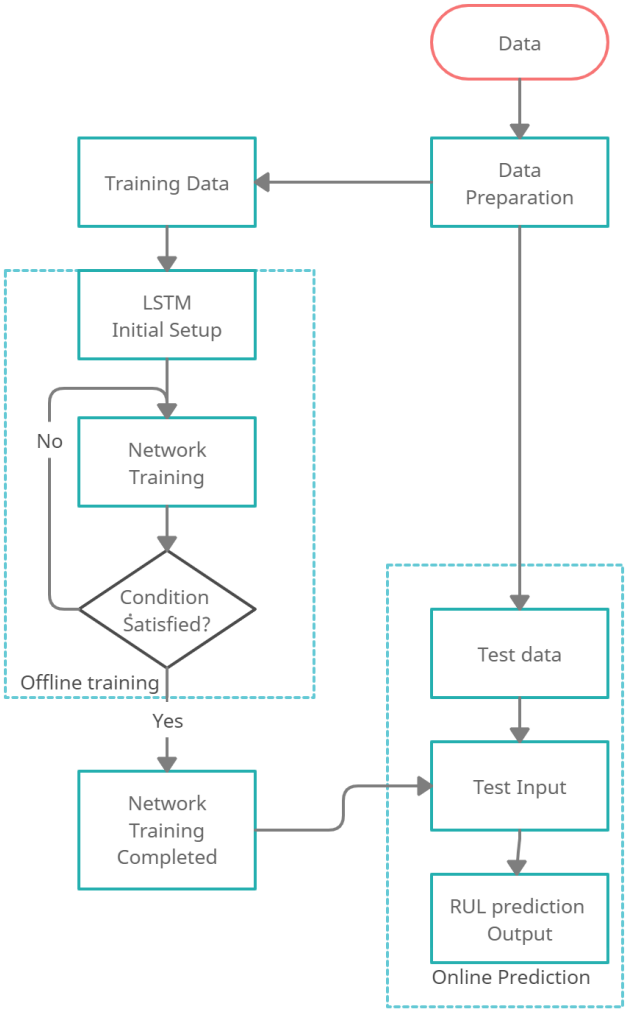


Fig. 6.3 Flow chart for model training

Ditto

Eclipse Ditto is an open source framework that helps you create and manage digital twins(thing). A digital twin is a digital replica of a physical system ranging from a small switch to a complex machine. It is used to represent the real time state of a machine in our case a small motor.

The temperature, vibration and speed of a motor along with additional metadata is represented in JSON format in ditto. Operations used are to list things, add/ delete things. Connections in ditto allow us to listen to external sources for state changes eg, listening to mqtt broker.

MongoDB

MongoDB is a source-available cross-platform document-oriented database program. It acts as a historical database for our real time data. All state changes are saved in MongoDB as time series data.
MongoDB as a primary database is being used to store equipment information, state changes and scheduling operations. All data is stored in JSON format.

Node.js

Node.js is an open-source, cross-platform, back-end. It is being used as our primary backend to handle the majority of api requests. Api Requests to other services are routed through node server to maintain consistency.

6.2 Working of the project

6.2.1 Predictive Maintenance api code

```
from tensorflow.keras.models import load_model
from typing import Optional
import numpy as np
import json
from fastapi import FastAPI
import pickle as pikl
from keras import backend as K
import uvicorn

import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
app = FastAPI()
```

```
def root_mean_squared_error(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true)))


loaded_model = load_model("D:\Sem7\BE_Project\machine_learning\\final_model",
custom_objects={
                    'root_mean_squared_error': root_mean_squared_error})
@app.get("/predict")
def scale_predict(text: str):
    try:
        scaler = pikl.load(open("D:\Sem7\BE_Project\machine_learning\scalerfinal.pkl", 'rb'))
        scaler1 = pikl.load(open("D:\Sem7\BE_Project\machine_learning\scalerlabfinal.pkl",
'rb'))
        # print(text)
        d = json.loads(text)
        # d.reshape(1,5,5)
        d = scaler.transform(d)
        print(d)
        numpy_2d_arrays = np.array(d, dtype=object)
        print(numpy_2d_arrays.shape)
        print(numpy_2d_arrays)
        d=d.reshape(1,5,5)


          res = loaded_model.predict(d)
        print(res)
        # return {"result_ns":f'{res}', "result_s":f'{scaler1.inverse_transform(res)}'}
        return {"pred": f'{scaler1.inverse_transform(res)[0][0]}'}
    except Exception as e:
        return {"error": f'{e}'}
```

```
if __name__ == '__main__':
    uvicorn.run(app,host='127.0.0.1', port=8000, debug=True)
```

6.2.2 Optaplanner Constriaints and API code

Domain of Optaplanner

```
from optapy import problem_fact, planning_id, planning_entity, planning_variable, \
    planning_solution, planning_entity_collection_property, \
    problem_fact_collection_property, \
    value_range_provider, planning_score
from optapy.types import HardSoftScore
from datetime import datetime, time, date, time, timedelta
from read_xlsx import read_xlsx
# import datetime

@problem_fact
class Machine:
    def __init__(self, id, type, name):
        self.id = id
        self.type = type
        self.name = name

    @planning_id
    def get_id(self):
        return self.id

    def __str__(self):
        return f"Machine(id={self.id}, type={self.type}, name={self.name})"
```

```
@problem_fact
class Timeslot:
    def __init__(self, id, day_of_week, date, start_time, end_time, begin_time, stop_time):
        self.id = id
        self.day_of_week = day_of_week
        self.date = date
        self.start_time = start_time
        self.end_time = end_time
        self.begin_time = begin_time
        self.stop_time = stop_time

    @planning_id
    def get_id(self):
        return self.id

    def __str__(self):
        return (
                f"Timeslot("
                f"id={self.id}, "
                f"day_of_week={self.day_of_week}, "
                f"date={self.date}, "
                f"start_time={self.start_time}, "
                f"end_time={self.end_time}),"
                f"begin_time={self.begin_time}, "
                f"stop_time={self.stop_time})"
        )

@planning_entity
class MachineAssignment:
```

```
    def __init__(self, id, task, customer, priority, ordernumber, deadline, machine_type,
timeslot=None, machine=None ):
        self.id = id
        self.task = task
        self.customer = customer
        self.priority = priority
        self.machine_type = machine_type
        self.timeslot = timeslot
        self.machine = machine
        self.ordernumber = ordernumber
        self.deadline = deadline

    @planning_id
    def get_id(self):
        return self.id

    @planning_variable(Timeslot, ["timeslotRange"])
    def get_timeslot(self):
        return self.timeslot

    def set_timeslot(self, new_timeslot):
        self.timeslot = new_timeslot

    @planning_variable(Machine, ["machineRange"])
    def get_machine(self):
        return self.machine

    def set_machine(self, new_machine):
        self.machine = new_machine

    def __str__(self):
```

```
        return (
                f"MachineAssignment("
                f"id={self.id}, "
                f"task={self.task}, "
                f"customer={self.customer}, "
                f"priority={self.priority}, "
                f"timeslot={self.timeslot}, "
                f"machine={self.machine},"
                f"ordernumber={self.ordernumber},"
                f"deadline={self.deadline})"
        )

def format_list(a_list):
    return ',\n'.join(map(str, a_list))

@planning_solution
class Schedule:
    def __init__(self, timeslot_list, machine_list, machine_assignment_list,score = None):
        self.timeslot_list = timeslot_list
        self.machine_list = machine_list
        self.machine_assignment_list = machine_assignment_list
        self.score = score

    @problem_fact_collection_property(Timeslot)
    @value_range_provider('timeslotRange')
    def get_timeslot_list(self):
        return self.timeslot_list

    @problem_fact_collection_property(Machine)
    @value_range_provider('machineRange')
    def get_machine_list(self):
```

```
        return self.machine_list

    @planning_entity_collection_property(MachineAssignment)
    def get_machine_assignment_list(self):
        return self.machine_assignment_list

    @planning_score(HardSoftScore)
    def get_score(self):
        return self.score

    def set_score(self, score):
        self.score = score

    def __str__(self):
        return (
            f"Schedule("
            f"timeslot_list={format_list(self.timeslot_list)},\n"
            f"machine_list={format_list(self.machine_list)},\n"
            f"machine_assingment_list={format_list(self.machine_assignment_list)},\n"
            f"score={str(self.score.toString()) if self.score is not None else 'None'}"
            f")"
        )

def generate_timeslots(start_date, end_date, start_time, end_time, time_interval):
    start_time = datetime.strptime(start_time, '%H:%M').time()
    og_start_time = start_time
    end_time = datetime.strptime(end_time, '%H:%M').time()

    (datetime.combine(date(1,1,1),start_time) + timedelta(hours=time_interval)).time()
    print(start_time,end_time)
```

```
    timeslot_list = []
    while start_date <= end_date:
        while start_time <= end_time:
            timeslot_list.append(Timeslot(
                len(timeslot_list),
                start_date.strftime("%A"),
                start_date,
                start_time,
                (datetime.combine(date(1,1,1),start_time) + timedelta(hours=time_interval)).time(),
                datetime.combine(start_date,start_time),
                (datetime.combine(start_date,start_time) + timedelta(hours=time_interval))

            ))
            start_time = (datetime.combine(date(1,1,1),start_time) +
timedelta(hours=time_interval)).time()
        start_date += timedelta(days=1)
        start_time = og_start_time
    return timeslot_list

def next_weekday(d, weekday):
    days_ahead = weekday - d.weekday()
    if days_ahead <= 0:  # Target day already happened this week
        days_ahead += 7
    return d + timedelta(days_ahead)

def generate_problem(start_date, end_date, start_time, end_time, time_interval):
    timeslot_list = []

    timeslot_list = generate_timeslots(start_date = next_weekday(date.today(), start_date),
end_date = next_weekday(date.today(), end_date), start_time = start_time, end_time =
end_time, time_interval = time_interval)
```

```
    machine_list = []
    for x in read_xlsx("./documents/machine.xlsx"):
        machine_list.append(Machine(int(x[0]), str(x[1]), str(x[2])))

    machine_assignment_list = []
    i=1
    for x in read_xlsx("./documents/orders.xlsx"):
        if x[6]!=None:
            k=int(x[6])
        while k>0:
            if x[0] != None:
                machine_assignment_list.append(MachineAssignment(int(i), str(x[0]), str(x[1]),
int(x[2]), str(x[3]), x[4].date(),str(x[5]), None, None))
                i+=1
                k-=1
    machine_assignment = machine_assignment_list[0]
    machine_assignment.set_timeslot(timeslot_list[0])
    machine_assignment.set_machine(machine_list[0])

    return Schedule(timeslot_list, machine_list, machine_assignment_list)
```

Constraints

```
from optapy import constraint_provider, get_class
# from optapy.types import Joiners, HardSoftScore
from domain import MachineAssignment, Machine, Timeslot, Schedule
from datetime import datetime, date, timedelta
from optapy.score import HardSoftScore
from optapy.constraint import Joiners, ConstraintFactory

MachineAssignmentClass = get_class(MachineAssignment)
```

```
MachineClass = get_class(Machine)
timeslots = get_class(Timeslot)
schedule = get_class(Schedule)

# Trick since timedelta only works with datetime instances
today = date.today()



def time_diff(machineassignment1, machineassignment2):
    between = datetime.combine(today, machineassignment1.timeslot.end_time) -
datetime.combine(today, machineassignment2.timeslot.start_time)
    return between

def before_deadline(machineassignment):
    return machineassignment.timeslot.date < machineassignment.deadline

def sel_machine(machineassignment):
    if machineassignment.task == 'maintainence':
        return False
    else:
        return machineassignment.machine.type == machineassignment.machine_type

def maintainence_assingment(machineassignment):
    if machineassignment.task == 'maintainence':
        if (machineassignment.machine.type == machineassignment.machine_type) and
((machineassignment.deadline - machineassignment.timeslot.date).days<timedelta(days=1)):
            return True
        else:
            return False
    else:
        return False
```

```
@constraint_provider
def define_constraints(constraint_factory: ConstraintFactory):
    return [
        # Hard constraints
        machine_conflict(constraint_factory),
        deadline_constraint(constraint_factory),
        select_machine(constraint_factory),
        maintainence_conflict(constraint_factory),
        machine_time_efficiency(constraint_factory),
        # Soft constraints
        customer_priority(constraint_factory),
        avoid_empty_timeslots(constraint_factory)
    ]


def machine_conflict(constraint_factory: ConstraintFactory):
    # A machine can do one task at one time.
    return
constraint_factory.forEachUniquePair(MachineAssignmentClass,[Joiners.equal(lambda
machineassignment: machineassignment.machine),
                                    Joiners.equal(lambda machineassignment:
machineassignment.timeslot)]) \
        .penalize("Machine conflict", HardSoftScore.ONE_HARD)

        # .filter(lambda machineassignment1, machineassignment2: machineassignment1.timeslot
==  machineassignment2.timeslot) \


def maintainence_conflict(constraint_factory: ConstraintFactory):
    # maintainance has to be done on the same machine
```

```
    return constraint_factory \
        .forEach(MachineAssignmentClass) \
        .filter(maintainence_assingment) \
        .reward("Machine maintainance", HardSoftScore.ONE_HARD)


def machine_time_efficiency(constraint_factory: ConstraintFactory):
    # A machine is more efficient in doing the same task back to back.
    return constraint_factory.forEachUniquePair(MachineAssignmentClass,\
            [
                Joiners.equal(lambda machineassignment: machineassignment.machine),
                Joiners.equal(lambda machineassignment: machineassignment.customer),
                Joiners.equal(lambda machineassignment: machineassignment.priority),
                Joiners.equal(lambda machineassignment: machineassignment.ordernumber)
            ]) \
        .filter(lambda machineassignment1, machineassignment2:
machineassignment1.timeslot.start_time == machineassignment2.timeslot.end_time) \
        .reward("Machine time efficiency", HardSoftScore.ONE_SOFT)

        # .filter(lambda machineassignment1, machineassignment2:
machineassignment1.ordernumber == machineassignment2.ordernumber) \

def customer_priority(constraint_factory: ConstraintFactory):
    # An order with higher priority must be completed first.
    return constraint_factory.forEachUniquePair(MachineAssignmentClass,\
            [
                Joiners.equal(lambda machineassignment: machineassignment.customer),
                Joiners.equal(lambda machineassignment: machineassignment.priority),
                Joiners.equal(lambda machineassignment:
machineassignment.timeslot.day_of_week)
            ]) \
```

```
        .filter(lambda machineassignment1, machineassignment2: machineassignment1.priority
>= machineassignment2.priority) \
        .reward("Customer priority", HardSoftScore.ONE_SOFT)

def deadline_constraint(constraint_factory:ConstraintFactory):
    # An order must be completed before the deadline.
    return constraint_factory.from_(MachineAssignmentClass) \
        .filter(before_deadline) \
        .reward("Deadline", HardSoftScore.ONE_SOFT)


def select_machine(constraint_factory: ConstraintFactory):
    return constraint_factory.from_(MachineAssignmentClass) \
        .filter(sel_machine) \
        .reward("Select Machine", HardSoftScore.ONE_HARD)


def avoid_empty_timeslots(constraint_factory: ConstraintFactory):
    return constraint_factory.forEach(MachineAssignmentClass) \
        .join(timeslots,[Joiners.equal(lambda machineassignment: machineassignment.timeslot,
lambda timeslot: timeslot)]) \
        .filter(lambda timeslot: timeslot.is_empty) \
        .penalize("Avoid empty timeslots", HardSoftScore.ONE_HARD)
```

6.2.3 Ditto definition and Ditto MQTT code

```
const searchThingsByNameSpace = async (nameSpace = defaultNamespace) => {
  const response = await axiosInstance({
    method: 'get',
    url: '/search/things',
    params: {
```

```
      namespace: nameSpace,
    },
  });
  if (response.status === 200) {
    return response.data;
  }
  throw new Error('Unexpected Error');
};
const thingDetail = async (id: string | undefined): Promise<Thing> => {
  if (!id) {
    throw new Error('No Id');
  }
  const _url = `${baseUrl}/things/${id}`;
  const response = await axios({
    method: 'get',
    url: _url,
    auth: {
      username: dittoUsername,
      password: dittoPassword,
    },
  });
  if (response.status === 200) {
    if (response.data === []) {
      throw new Error('No Id');
    }
    return response.data;
  }
  throw new Error('No Id');
};
const establishWebsocketConnection = () => {
```

```
  const websocketEndpoint =
`ws://${dittoUsername}:${dittoPassword}@localhost:8080/ws/2`;
  let socket = new WebSocket(websocketEndpoint);
  return socket;
};
export { searchThingsByNameSpace, thingDetail, establishWebsocketConnection };
import { connect } from 'mqtt';
import ThingModel from './models/thing.model';
interface MQTTData {
  surrounding_temperature: number;
  motor_temperature: number;
  vibration: number;
  humidity: number;
  speed: number;
  thingId: string;
}
function main() {
  console.log('in main');
  const client = connect('mqtt://broker.hivemq.com');

  const projTopic = 'anselalanaaronsherwin/';

  client.on('connect', () => {
    client.subscribe(projTopic);
  });
  client.on('message', (topic, message: Buffer) => {
    switch (topic) {
      case projTopic: {
        return handleIncommingData(message);
      }
    }
```

```
});
const handleIncommingData = async (message: Buffer) => {
  try {
    const data: MQTTData = JSON.parse(message.toString());

    const outline = {
      thingId: data.thingId,
      metadata: {
        timestamp: new Date().toString(),
      },
      features: {
        properties: {
          temperature: data.motor_temperature,
          speed: data.speed,
          vibration: data.vibration,
        },
        surrounding: {
          properties: {
            temperature: data.surrounding_temperature,
            humidity: data.humidity,
          },
        },
      },
    };

    const thing = new ThingModel(outline);
    await thing.save();
    console.log('Saving thing');
  } catch (e) {
    console.log(e);
  }
```

```
   return;
  };
}
export default main;
```

6.2.4 ESP8266 code

```
#include <ESP8266WiFi.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <EEPROM.h>
#include <PubSubClient.h>
#include <Wire.h>
#include "DHT.h"

void ICACHE_RAM_ATTR handleInterrupt();

String  DATA1;
String  DATA2;
String  DATA3;
WiFiClient espClient;
PubSubClient client(espClient);

int vs =0;// Pin 0
const int encoder_a = 2; // Pin 2
const int encoder_b = 3; // Pin 3
long encoder_pulse_counter = 0;
long direction = 1;

String speed_str;
String vib_str;
```

```
String temp_str;
String hum_str;
String ran;
char temp[50];
char hum[50];
char sp[50];
char vib[50];

long randsurrtemp;
long randmotortemp;
long randvibration;
long randhumidity;
long randspeed;
DHT dht(1,DHT11);

const char *mqtt_server = "broker.hivemq.com";
const char *device_id = "esp8266";
void reconnect(unsigned int speed, unsigned int vib_measurement, unsigned int temperature_v,
unsigned int humidity_v )
{
 Serial.println("START");
 while (!client.connected())
 {
  Serial.print("Attempting MQTT connection...");
  if (client.connect(device_id, "", ""))
  {
  Serial.println("connected");
  }
  else
  {
   Serial.print("failed, rc=");
```

```
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}


void setup()
{
  DATA1 = "";

  DATA2 = "";

  Serial.begin(9600);

  WiFi.disconnect();
  delay(1000);
  Serial.println("START");
  WiFi.begin("Ansel2000","anselamanda");
  while ((!(WiFi.status() == WL_CONNECTED))){
    delay(300);
    Serial.print("...");
  }
  Serial.println("Connected!!");
Serial.println("Your IP is:");
Serial.print((WiFi.localIP().toString()));

  client.setServer(mqtt_server, 1883);
  pinMode(encoder_a, INPUT_PULLUP);
  pinMode(encoder_b, INPUT_PULLUP);
```

```
  attachInterrupt(0, encoderPinChangeA, CHANGE);
  attachInterrupt(1, encoderPinChangeB, CHANGE);
Serial.begin(115200);
pinMode(pin5,INPUT_PULLUP);
pinMode(LED_BUILTIN,OUTPUT);
attachInterrupt(digitalPinToInterrupt(pin5),ISRoutine,FALLING);
pinMode(pin4,INPUT);
}
void loop()
{
   Serial.println("START");
long speed = encoder_pulse_counter/1024.00*60;
DATA1 = String(direction*speed);
encoder_pulse_counter = 0;
delay(1000);
 long vib_measurement =vibration();
DATA2 = String(vib_measurement);
delay(1000);
String temperature_v=String(dht.readTemperature( ));
String humidity_v=String(dht.readHumidity());
   Serial.println("MID");
   if (!client.connected())
     {
 reconnect(speed, vib_measurement, temperature_v, humidity_v);
     }
   if(client.connected()){
 speed_str = String(speed);
speed_str.toCharArray(sp,speed_str.length()+1);
vib_str = String(vib_measurement);
vib_str.toCharArray(vib,vib_str.length()+1);
temp_str = String(temperature_v);
```

```
 temp_str.toCharArray(temp,temp_str.length()+1);

hum_str = String(humidity_v);
hum_str.toCharArray(hum,hum_str.length()+1);

client.publish("esp/speed",(char*)speed_str.c_str());
client.publish("esp/vibration",(char*) vib_str.c_str());
 client.publish("esp/temperature_sur",(char*) temp_str.c_str());
client.publish("esp/humidity_sur",(char*) hum_str.c_str());
client.publish("esp","esp");
ran = "{ \"surrounding_temperature\" : "+String(randsurrtemp)+",
\"motor_temperature\":"+String(randmotortemp)+", \"vibration\" :"+String(randvibration)+",
\"humidity\" :"+String(randhumidity)+", \"speed\":"+String(randspeed)+", \"thingId\": " +
"\"ixorio:12vmotor\"" + '}';
Serial.println(String(ran));
client.publish("anselalanaaronsherwin/", (char*)ran.c_str());
delay(10000);
}
client.loop();
}
long vibration(){
long measurement=pulseIn (vs, HIGH);
return measurement;
}
void encoderPinChangeA()
{
   encoder_pulse_counter += 1;
   direction = digitalRead(encoder_a) == digitalRead(encoder_b) ? -1 : 1;
}

void encoderPinChangeB()
```

```
{
   encoder_pulse_counter += 1;
   direction = digitalRead(encoder_a) != digitalRead(encoder_b) ? -1 : 1;
}
void ISRoutine () {
int value;
Serial.println("I am in ISR");
}
```

6.2.5 Node Server Code

```
import { Router, Request } from 'express';
import {
  getDummyData,
  getEquipment,
  putEquipment,
  putState,
  deleteEquipment,
  getRul,
} from '../controllers/api.controller';

const router = Router();

import fs from 'fs';

router.get('/state', getDummyData);

router.get('/rul', getRul);

router.get('/equipment', getEquipment);
```

```
router.get('/equipment/:id', getEquipment);

router.delete('/equipment/:id', deleteEquipment);

router.post('/equipment', putEquipment);

export default router;



import { Router } from 'express';
import { writeFile } from 'fs';

const router = Router();

export default router;

router.use((req, res, next) => {
  console.log('New Request ----------');
  console.log(req);
  console.log(req.headers);
  console.log(req.method);

  writeFile('test-connection.json', JSON.stringify(req.body), (err) => {
    console.log(err);
  });

  res.send('Hello World');
});



import { Request, Response, NextFunction } from 'express';
```

```
import axios, { AxiosError } from 'axios';
import EquipmentModel from '../models/equipment.model';
import ThingModel from '../models/thing.model';
import { putThing } from '../services/ditto';
import dummyData from '../dummy.json';

export const getDummyData = async (
  req: Request,
  res: Response,
  next: NextFunction
): Promise<void> => {
  res.json(dummyData.slice(0, 50));
  return;
};

export const getState = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {};

export const putState = async (
  req: Request,
  res: Response,
  next: NextFunction
): Promise<void> => {};

export const getRul = async (
  req: Request,
  res: Response,
  next: NextFunction
```

```
): Promise<void> => {
  try {
    // fetch from mongod
    const thingStates = await ThingModel.find().sort({ _id: -1 }).limit(5);

    const requestBody = thingStates.map((thing) => {
      return [
        thing.features.surrounding?.properties?.temperature,
        thing.features.surrounding?.properties?.humidity,
        thing.features.properties?.temperature,
        thing.features.properties?.vibration,
        thing.features.properties?.speed,
      ];
    });

    const rulResponse = await axios.get(`${process.env.RUL_URL}/predict`, {
      params: {
        text: JSON.stringify(requestBody),
      },
    });

    res.json(rulResponse.data);
  } catch (err) {
    res.status(500).send(err);
  }
};

export const getEquipment = async (
  req: Request,
  res: Response,
  next: NextFunction
```

```
): Promise<void> => {
  if (req.params.id) {
    let equipment = await EquipmentModel.findById(req.params.id);
    res.json(equipment);
  } else {
    let equipments = await EquipmentModel.find();
    res.json(equipments);
  }
};

export const putEquipment = async (
  req: Request,
  res: Response,
  next: NextFunction
): Promise<void> => {
  const body = req.body;
  try {
    const response = await putThing(body);
    console.log(response);

    const equipment = new EquipmentModel(response);
    const newDoc = await equipment.save();

    res.json({
      status: 'Created',
      data: newDoc,
    });
  } catch (err) {
    console.log(typeof err);
    if (axios.isAxiosError(err)) {
      if (err.response) {
```

```
        res.status(500).send(`${err.response.status}: Ditto/Server Error`);
      } else {
        if (err.code === 'ECONNREFUSED') {
          res.status(500).send(`500: Ditto/Connection Error`);
        }
      }
    } else {
      res.status(500).send(err);
    }
  }
};

export const deleteEquipment = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  try {
    if (req.params.id) {
      await EquipmentModel.findByIdAndDelete(req.params.id);
      return res.status(200).json({ status: 'SUCCESS' });
    }
    throw 'ERROR';
  } catch (err) {
    return res.status(500).json({ status: err });
  }
};
```

6.2.6 MongoDB

```
import { Schema, model, Document, SchemaTypes } from 'mongoose';
```

```
interface Equipment extends Document {
  thingId: string;
}

const EquipmentSchema = new Schema<Equipment>(
  {
    thingId: {
      type: String,
      required: true,
      index: true,
    },
  },
  {
    strict: false,
    timestamps: true,
  }
);

EquipmentSchema.set('toObject', { virtuals: true });
EquipmentSchema.set('toJSON', { virtuals: true });

EquipmentSchema.virtual('id').get(function (this: Equipment) {
  return this._id;
});

const EquipmentModel = model<Equipment>('Equipment', EquipmentSchema);

export default EquipmentModel;
```

```
import { Schema, model, Document, SchemaTypes } from 'mongoose';

type stateProperty = number | string;

export interface Thing extends Document {
  thingId: string;
  definition?: string;
  attributes?: object;
  metadata: {
    timestamp: string;
  };
  features: {
    properties?: {
      temperature: stateProperty;
      speed?: stateProperty;
      vibration?: stateProperty;
    };
    surrounding?: {
      properties?: {
        temperature?: stateProperty;
        humidity?: stateProperty;
      };
    };
  };
}

const ThingSchema = new Schema<Thing>(
  {
    thingId: { type: String, required: true, index: true },
    definition: String,
    attributes: {
```

```
      type: Map,
      of: SchemaTypes.Mixed,
    },
    metadata: {
      timestamp: { type: String, index: true },
    },
    features: {
      properties: {
        temperature: { type: Number },
        speed: { type: Number },
        vibration: { type: Number },
      },
      surrounding: {
        properties: {
          temperature: { type: Number },
          humidity: { type: Number },
        },
      },
    },
  },
  { strict: false }
);

const ThingModel = model<Thing>('Thing', ThingSchema);

export default ThingModel;
```

Chapter 7

Testing

For testing the model the Remaining Useful Life on validation data was simulated and compared with the actual results.

The below table shows the value the Training loss and the Validation loss that were obtained when training the model with the training and validation data.

Training Loss (MSE)	8E-4
Validation Loss (MSE)	5E-2

Table 1. The values of Training and Validation Loss

Fig. 7.1 the error between the true and the forecasted Remaining Useful Life was also plotted for the number of epochs trained and it also was satisfactorily on the lower end of the spectrum.

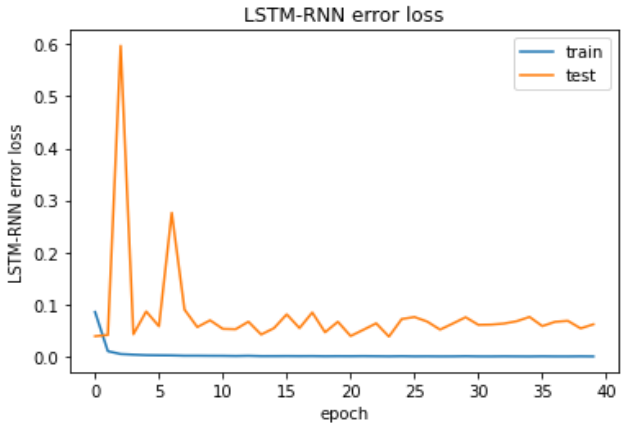


Fig. 7.1 The Root mean square error for each epoch during training

In Fig. 7.2 the forecasted result was plotted against the true validation data. It showed a great degree of similarity therefore verifying that the model was able to predict the Remaining Useful Life with a great degree of accuracy.

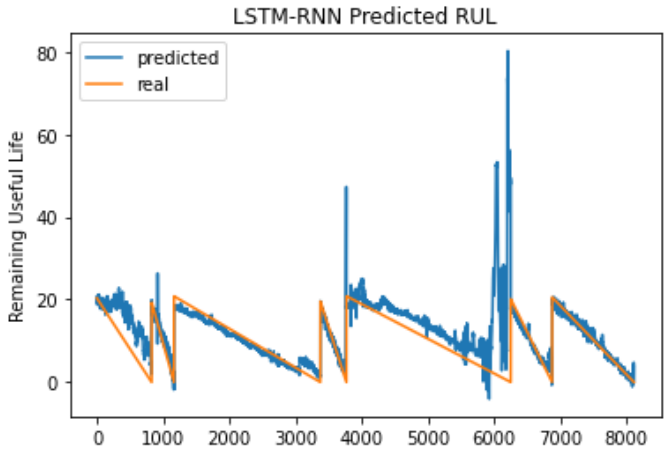


Fig. 7.2 The plot of forecasted RUL vs True RUL

Chapter 8

Results and Discussion

With the help of data acquired from sensors, the system displays the status of machinery by calculating the remaining useful life of machines. The plant manager can see the list of equipment and add new ones to the system. The plant manager determines which equipment require maintenance and plans maintenance for those machines by calculating the remaining usable life of the machines.

Fig 8.1 With the description of machinery and attributes such as manufacturer, serial number, location, and model number, the plant manager can add new equipment to the system.
First the user will add a new machine with the description of the machinery and attributes such as manufacturer, serial number, location, and model number, the plant manager can add new equipment to the system.

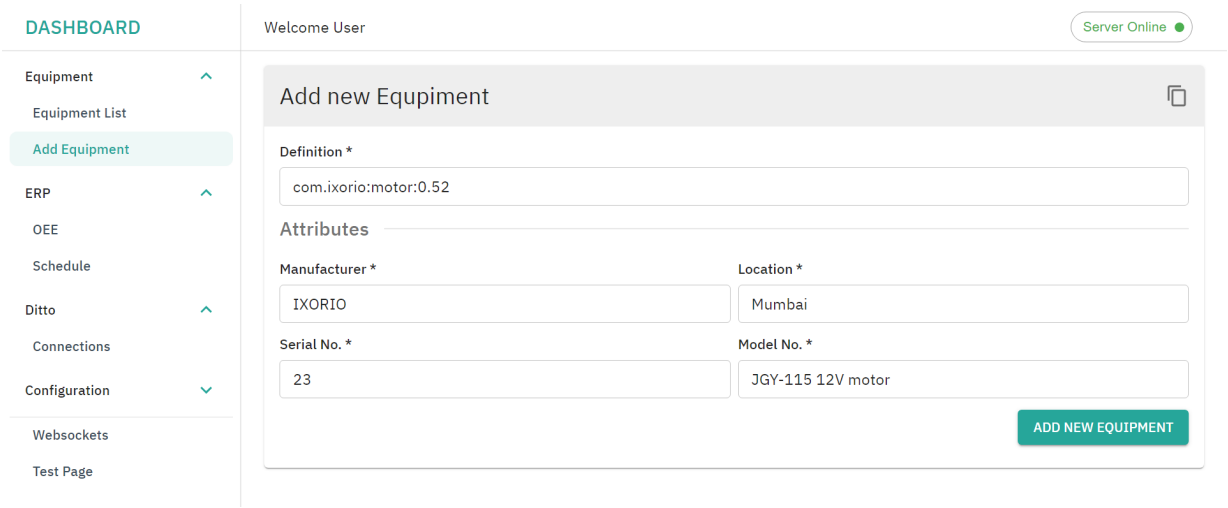


Fig. 8.1 Dashboard Display - Add New Equipment

In Fig 8.2 The plant management has access to the equipment list. The id, description, model number, manufacturer, and location of the machine are all included. The plant manager receives detailed information on the new equipment that has been added to the system.

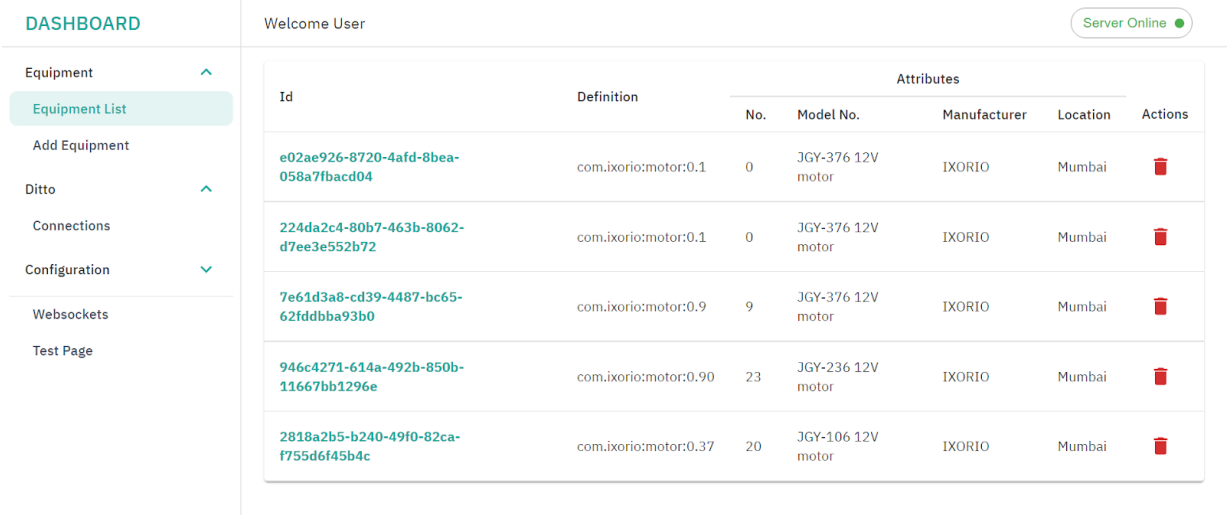


Fig. 8.2 Dashboard Display - Equipment List of Digital Twin

In Fig 8.3 the setup of a sample machine is shown with the sensors connected to the PLC. This PLC then publishes data to the MQTT broker.

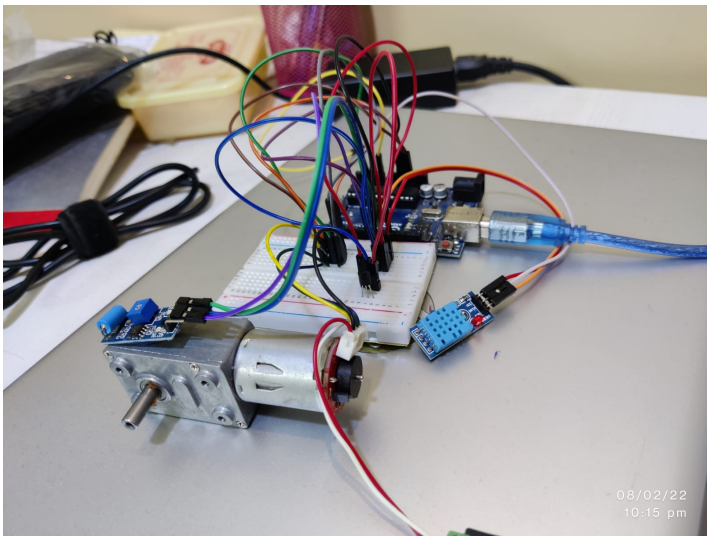


Fig. 8.3 Setup of sampe machine

In Fig 8.4 Here we see the dashboard. Once the data is sent to the broker we subscribe to the same topic through Eclipse Ditto and the data received is published on the dashboard in realtime. This data is also stored in the Databse and the past 5 reading are sent to the Machine Learning Model to forecast the Remaining Useful Life of the machine.

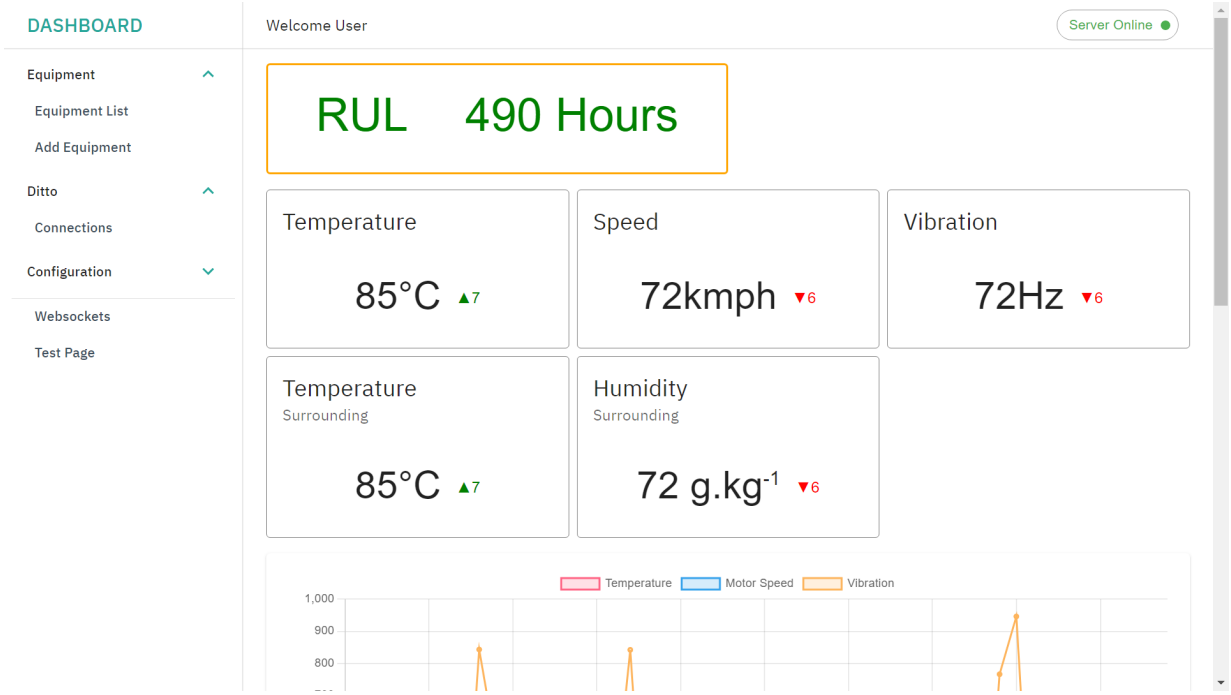


Fig. 8.4 Dashboard Display - Sensor Data and RUL

The user is then able to view and compare the readings in the form of a plotted graph as shown in Fig. 8.5

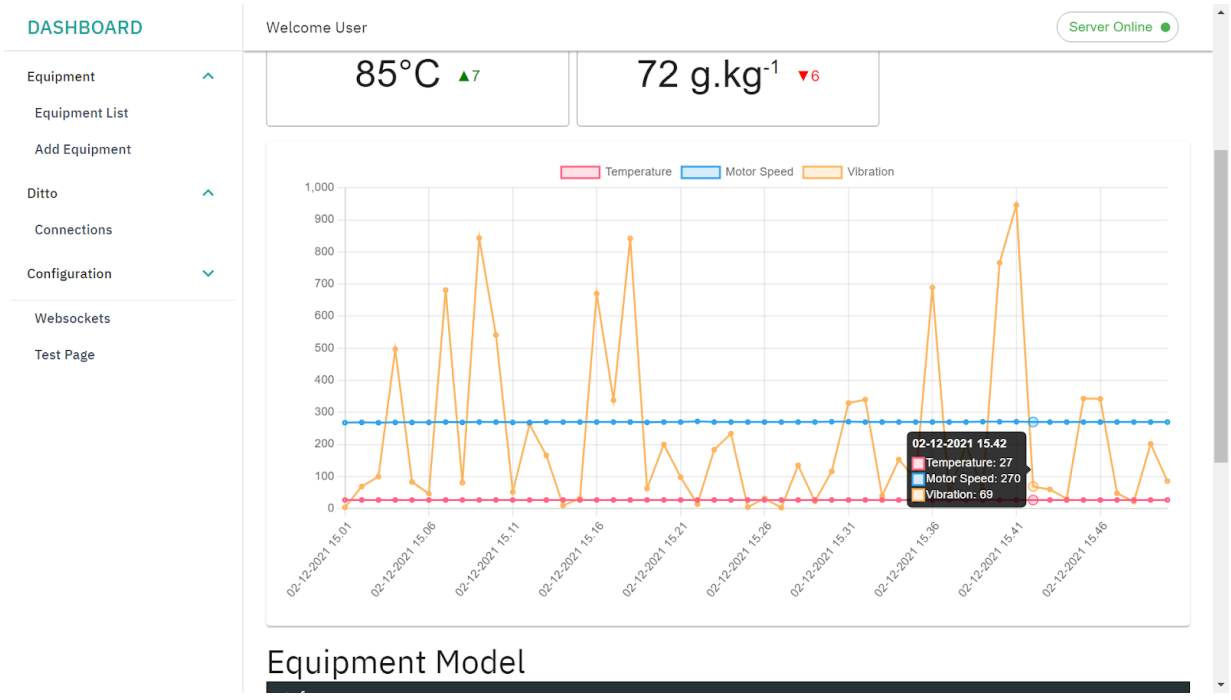


Fig. 8.5 Dashboard Display - Sensor Data Plot

In Fig 8.6 the user can then upload the details of the machine and orders that need to be scheduled. The Optaplanner problem solver then uses these details and based on the constraints that have been predefined and generates a schedule.

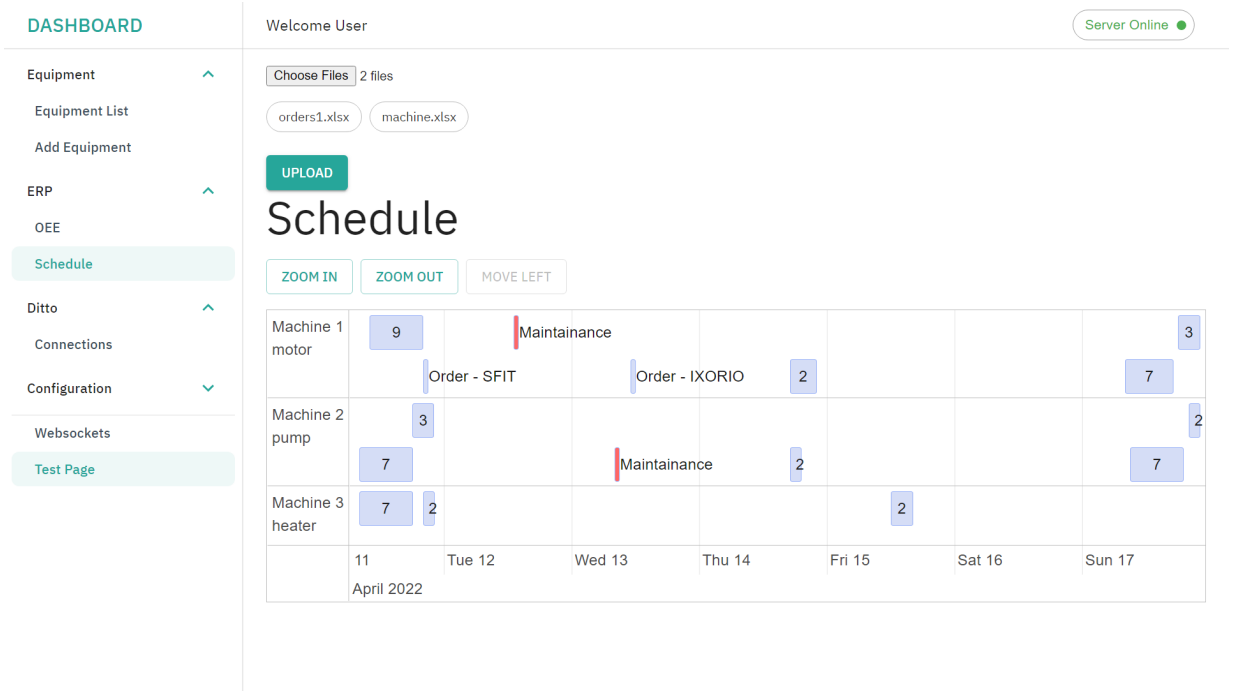


Fig. 8.6 Dashboard Display - Schedule of Orders

Chapter 9

Conclusion and Future Scope

Maintenance is considered as an integral part of the manufacturing process that contributes to the product quality, reducing production cost and ability to meet delivery schedules. Traditional Paper-based manufacturing systems need to transport reports to the plant floor from the shop floor and vice versa. This system is prone to human error, also requiring a report to physically arrive on the plant floor to begin manufacturing. Organizations cannot afford any large or little breakdowns. A single loose bolt leads to excessive wear and tear, shortened equipment life, and unplanned downtime. As a result, you'll have higher recovery costs, lower production quality, no simultaneous device analysis, and no transparency in your manufacturing process.

The current system works on reactive or preventive maintenance which is unpredictable or increases cost respectively. The proposed system, as per Industry 4.0 protocols, will be able to improve system reliability, prevent system failures, reduce maintenance costs, and have full visibility through a web-based system.

The proposed solution is aimed at companies belonging to the small-scale manufacturing industry sector. In this project, the goal is to design an MVP (Minimum viable product) that will enable floor managers and maintenance technicians to know the status of the machines and maintenance requirements well in advance. This was made possible through interactive dashboards which were dynamically generated to guard the conditions of the machinery. This prototyping model which was made for the company IXIRIO can be incorporated into the company’s working model.

In future this system can be integrated with other business management softwares such as ERP, Inventory management, production communications systems to efficiently divide workload between machines for obtaining maximum efficiency and reducing downtime.The software can further be improved with the use of machine learning to schedule preventative maintenance as well as fine-tune productivity such that the system will be able to detect faulty parts in a complex machine.

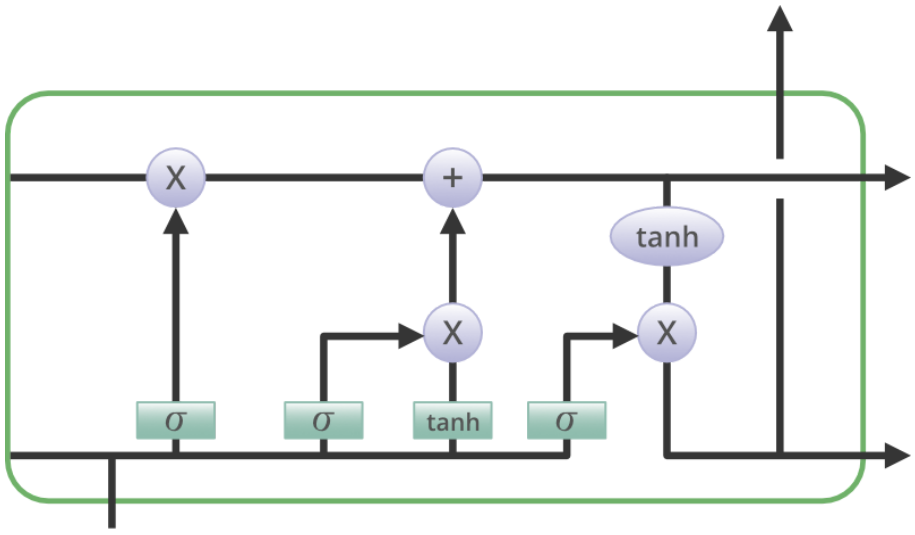
Appendix

LSTM-

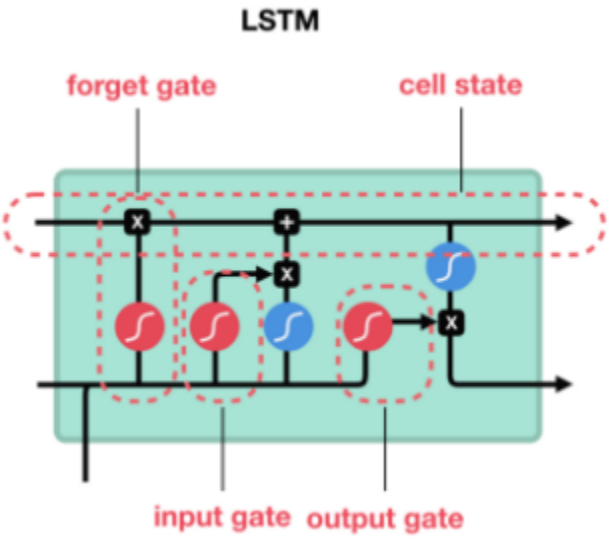
Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. LSTM networks are well-suited to classifying, processing and making predictions based on time series data.

In RNN output from the last step is fed as input in the current step. LSTM was designed by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give an efficient performance. LSTM can by default retain the information for a long period of time. It is used for processing, predicting, and classifying on the basis of time-series data.

With LSTMs, the information flows through a mechanism known as cell states. This way, LSTMs can selectively remember or forget things.



Deep Learning | Introduction to Long Short Term Memory - GeeksForGeeks



Illustrated Guide to LSTM's and GRU's: A step by step explanation - Michael Phi

Pseudo Code:

1. First, the previous hidden state and the current input get concatenated. We'll call it combine.
2. Combine gets fed into the forget layer. This layer removes non-relevant data.
4. A candidate layer is created using combine. The candidate holds possible values to add to the cell state.
3. Combine also gets fed into the input layer. This layer decides what data from the candidate should be added to the new cell state.
5. After computing the forget layer, candidate layer, and the input layer, the cell state is calculated using those vectors and the previous cell state.
6. The output is then computed.
7. Pointwise multiplying the output and the new cell state gives us the new hidden state.

```
def model_builder(hp):
   model = Sequential()
                  model.add(LSTM(hp.Int('input_unit',min_value=32,max_value=512,step=32),
return_sequences=True, input_shape= ( trainX.shape[1], trainX.shape[2])))
   for i in range(hp.Int('n_layers', 1, 4)):
```

```
model.add(LSTM(hp.Int(f'lstm_{i}_units',min_value=32,max_value=512,step=32),return_sequences=True))
    model.add(LSTM(hp.Int('layer_2_neurons',min_value=32,max_value=512,step=32)))
    model.add(Dropout(hp.Float('Dropout_rate',min_value=0,max_value=0.5,step=0.05)))
                    model.add(Dense(30,      activation=hp.Choice('dense_activation',values=['relu', 'sigmoid'],default='relu')))
    model.add(Dropout(hp.Float('Dropout_rate',min_value=0,max_value=0.5,step=0.05)))
                     model.add(Dense(1,      activation=hp.Choice('dense_activation',values=['relu', 'sigmoid'],default='relu')))

    model.compile(loss='mean_squared_error', optimizer='adam',metrics = ['mse'])

    return model

tuner = kt.RandomSearch(model_builder, objective="mse", max_trials = 3, executions_per_trial =1,directory = "./")

tuner.search(x=trainX, y=trainY, epochs = 150, batch_size =128, validation_data=(testX, testY), shuffle=False)
```

The Remaining Useful Life (RUL)-

The Remaining Useful Life (RUL) is a subjective estimate of the number of remaining years that an item, component, or system is estimated to be able to function in accordance with its intended purpose before warranting replacement.

Digital Twin -

- A Digital Twin is a virtual instance of a physical system (twin) that is continually updated with the latter's performance, maintenance, and health status data throughout the physical system's life cycle.

- The Digital Twin has the potential to give real-time status on machines performance as well as production line feedback.
- It gives the manufacturer the ability to predict issues sooner.
- Digital Twin use increases connectivity and feedback between devices, in turn, improving reliability and performance.
- AI algorithms coupled with Digital Twins have the potential for greater accuracy as the machine can hold large amounts of data, needed for performance and prediction analysis.

References

[1] Loubna Benabbou 1 Zouheir Malki 2 Kris Sankaran 3 Hicham Bouzekri 4, “Machine Learning-based Predictive Maintenance for Renewable Energy:The Case of Power Plants in Morocco”, ICML 2019 Workshop Climate Change: How Can AI Help?, 2019

[2] Lisa B. Bosman, Walter D. Leon-Salas, William Hutzel and Esteban A. Soto, “PV System Predictive Maintenance: Challenges, Current Approaches, and Opportunities”, Energies 2020, 13, 1398; doi:10.3390/en13061398. 2020

[3]Ana Cachada, José Barbosa et. al. , “Maintenance 4.0: Intelligent and Predictive Maintenance System Architecture”, 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), 2018

[4] Kahiomba Sonia Kiangala & Zenghui Wang , “Initiating predictive maintenance for a conveyor motor in a bottling plant using industry 4.0 concepts”, Springer-Verlag London Ltd., part of Springer Nature 2018, The International Journal of Advanced Manufacturing Technology

[5] Kahiomba Sonia Kiangala & Zenghui Wang, “An Effective Predictive Maintenance Framework for Conveyor Motors Using Dual Time-Series Imaging and Convolutional Neural Network in an Industry 4.0 Environment”, South African National Research Foundation under Grant 112108 and Grant 112142, and in part by the Tertiary Education Support Program (TESP) of South African ESKOM, July 2020

[6] A. Fuller, Z. Fan, C. Day and C. Barlow, "Digital Twin: Enabling Technologies, Challenges and Open Research," in IEEE Access, vol. 8, pp. 108952-108971, 2020, doi: 10.1109/ACCESS.2020.2998358.

[7] Fabian Timm (2017), Predictive Maintenance Taking pro-active measures based on advanced data analytics to predict and avoid machine failure, Position Paper - Deloitte Analytics Institute

[8] YANG, Wen & Haider, Naeem & ZOU, Jian-hong & Zhao, Qianchuan. (2017). Industrial Big Data Platform Based on Open Source Software. 10.2991/cnct-16.2017.90.

[9]V. Kamath, J. Morgan and M. I. Ali, "Industrial IoT and Digital Twins for a Smart Factory : An open source toolkit for application design and benchmarking," *2020 Global Internet of Things Summit (GIoTS)*, 2020, pp. 1-6, doi: 10.1109/GIOTS49054.2020.9119497.

[10]Industry 4.0 - "Smart Factory" explained - Pilz Belgium Mar 30, 2016
https://www.youtube.com/watch?v=h9t06cyC7Es

[11]Besutti, Rangel & de Campos Machado, Vanessa & Cecconello, Ivandro. (2019). Development of an open source-based manufacturing execution system (MES): industry 4.0 enabling technology for small and medium-sized enterprises. Scientia cum Industria. 7. 1-11. 10.18226/23185279.v7iss2p1.

[12]Nabti, Mohamed & Bybi, Abdelmajid & Chater, El Ayachi & Garoum, Mohammed. (2022). Machine learning for predictive maintenance of photovoltaic panels: cleaning process application. E3S Web of Conferences. 336. 00021. 10.1051/e3sconf/202233600021.

[13]How to create predictive maintenance software for wind turbines using machine learning algorithms - Boldare Machine Learning team
https://www.boldare.com/work/case-study-predictive-maintenance/