



ENGINEERING DEPARTMENT

MASTER'S DEGREE IN
ARTIFICIAL INTELLIGENCE AND DATA ENGINEERING

INTERNET OF THINGS

“IDIOT”
***An InDustrial IoT Network for Predictive
Maintenance***

[GitHub Repository](#)

Gemelli Mattia - Namaki Ghaneh Daniel

UNIVERSITÀ DI PISA
ACADEMIC YEAR 2023/2024

Summary

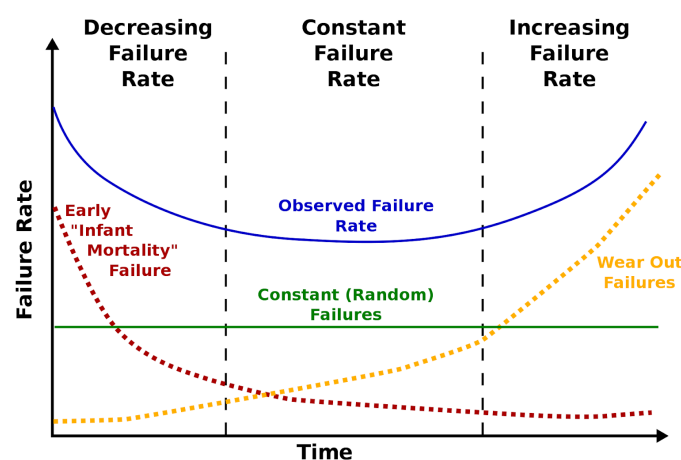
Abstract.....	2
Architecture.....	4
CoAP Network.....	5
Sensor nodes.....	5
Actuator node.....	5
Database.....	5
Sensor.....	6
Telemetry.....	6
Failures.....	7
Data Encoding.....	7
Deployment and Execution.....	8
CoAP Network.....	8
Sensors.....	8
Actuator.....	8
Remote Control Application.....	9
health.....	10
monitor.....	10
failures.....	10
switch.....	11
help.....	11
exit.....	11
Grafana.....	12
Machine Learning model.....	13
Predictive Maintenance Use Case.....	14
Scenario.....	14
Implementation.....	14
Conclusion.....	14

Abstract

Maintenance costs are crucial in manufacturing and production companies, ranging from 15 to 60 percent of the cost of goods produced, depending on the industry. Industrial and process plants typically use two types of maintenance management: run-to-failure and preventive maintenance.

Run-to-Failure Management: this reactive technique waits for machine or equipment failure before taking any maintenance action. Essentially, it is a “no maintenance” approach.

Preventive Maintenance: this proactive approach involves regular and routine maintenance of equipment and assets to keep them running and prevent costly unplanned downtime due to unexpected failures. The statistical life of a machine-train, illustrated by the mean-time-to-failure (MTTF) or bathtub curve, shows that a new machine has a high probability of failure due to installation issues in the first few weeks. After this initial period, the failure probability is relatively low for an extended time, followed by a sharp increase as the machine ages. In preventive maintenance, repairs or rebuilds are scheduled based on MTTF statistics.



However, preventive maintenance cannot entirely eliminate equipment failure due to the complexity of machines, where many components work together and influence each other's lifespan. This raises the question: How can we determine the optimal moment for maintenance so that components are not prematurely replaced, yet the system remains reliable?

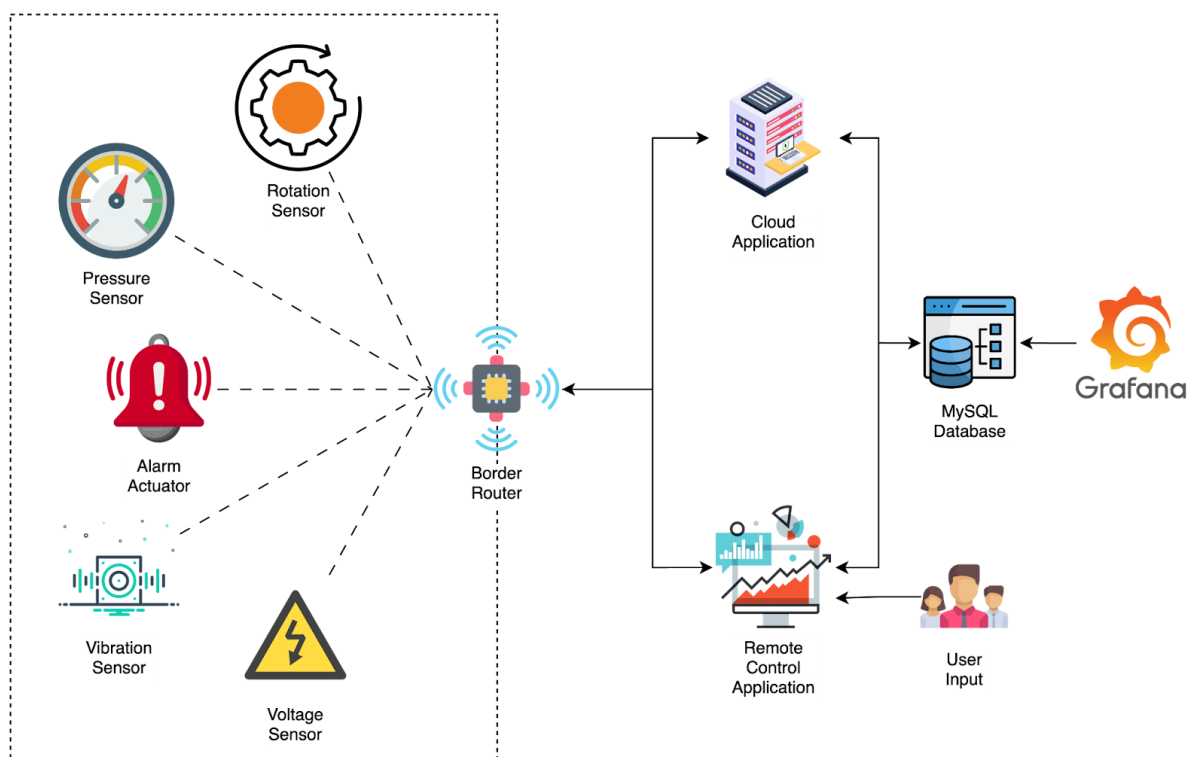
Predictive Maintenance aims to answer this by building predictive models based on observed data such as vibration, pressure, voltage, and rotation to quantify the risk of failure at any moment. This information improves maintenance scheduling.

¹ By Bath tub _curve.jpg: Wyattsderivative work: McSush (talk) - Bath tub _curve.jpg, Public Domain, <https://commons.wikimedia.org/w/index.php?curid=7458336>

Predictive maintenance is not a substitute for traditional maintenance methods but a valuable addition to a comprehensive plant maintenance program. While traditional methods rely on routine servicing and quick responses to failures, predictive maintenance schedules specific tasks based on the actual needs of plant equipment. It cannot replace the need for run-to-failure or preventive maintenance entirely but can reduce unexpected failures and provide a more reliable tool for scheduling routine preventive maintenance tasks.

Architecture

The system architecture is designed around a CoAP (Constrained Application Protocol) network to ensure efficient communication between sensor nodes and the central server. Each sensor node in the Low Power and Lossy Network is equipped with a sensing module such as rotation, voltage, pressure, and vibration, that monitors the machine's parameters which they are attached to. These sensor nodes are programmed to send their readings to a CoAP server, which stores the data, and to an actuator node. The actuator node is responsible for processing the incoming sensor data, including storing it in queues and using it to make predictions about potential system failures using a **machine learning model**. The nodes use CoAP observe mechanism to enable the server to receive real-time updates from the sensors. This setup ensures low-power, reliable communication suitable for IoT (Internet of Things) environments. Additionally, the system includes a remote control application to manage the operational state of the sensors and actuators and retrieve metrics from the database. The Cloud Application allows node registration and retrieves the IP addresses of the sensors, initiating and managing observations, periodically checking for their status. The architecture ensures robust and scalable monitoring and control of the system components, leveraging CoAP's lightweight communication protocol to maintain efficient operation. CoAP network is deployed using real sensors nRF52840 Dongle. Networks are connected to a border router that allows them external access.



CoAP Network

Sensor nodes

- **Pressure** sensor: this sensor measures the pressure exerted on the machine, providing real-time data on pressure fluctuations during the entire process.
- **Voltage** sensor: this sensor measures the electrical voltage applied to the machine. By tracking voltage variations, it helps to detect possible anomalies or damages caused by overvoltage or undervoltage.
- **Vibration** sensor: this sensor detects vibrations produced by the machine's operations. Some strange vibrations may be caused by some issues such as, imbalance, misalignment or wear of components.
- **Rotation** sensor: this sensor monitors the rotational speed of the machinery's moving components.

All the sensor nodes deployed in the Low Power and Lossy Network expose an *observable resource* which is the sensing module. Every hour they collect and send data both to the actuator node and to the cloud application.

Actuator node


- **Alarm**: the system collects sensor data over a 24-hour period, feeds it into the machine learning model, and based on the model's output, determines whether there's a high likelihood of a component failure within the subsequent 24 hours. If it predicts such a failure, it activates an alarm to notify relevant personnel to take preventive measures or perform maintenance. In addition to that, it sends the data about predictions to the cloud application to allow monitoring of failures.

This setup essentially enables predictive maintenance, where potential failures can be anticipated and addressed before they occur, minimizing downtime and preventing costly breakdowns.

Database

It is essential to store data collected with sensors, also to be able to analyze them through Grafana. The database ("iot") is particularly simple, we have defined different tables:


Sensor



Field	Type	Null	Key	Default	Extra
ip_address	varchar(45)	NO	PRI	NULL	
type	varchar(45)	YES		NULL	
status	int	YES		NULL	
registration_timestamp	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

This table is used to store information about sensors and actuators registered in the system. It maintains status to detect possible network failures and react accordingly.

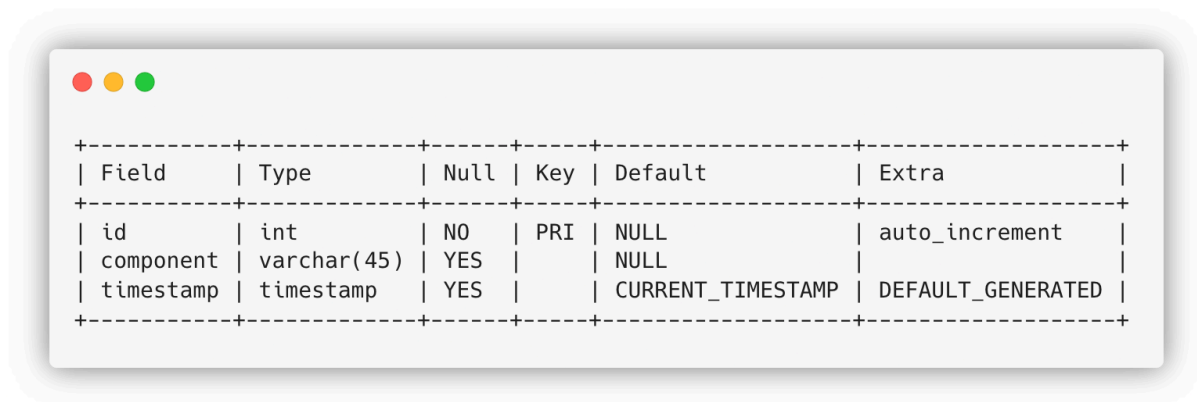
Telemetry



Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
timestamp	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
vibration	float	NO		NULL	
rotation	float	NO		NULL	
pressure	float	NO		NULL	
voltage	float	NO		NULL	

The telemetry table is a crucial component of the IoT system designed to capture and store real-time sensor data. This table logs the measurements collected from various sensors deployed within the network, providing a historical record of the environmental and operational parameters being monitored.

Failures



Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
component	varchar(45)	YES		NULL	
timestamp	timestamp	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

The failures table is designed to log instances where a component in the IoT system is identified as needing replacement to prevent potential failure. This proactive approach to maintenance helps to ensure system reliability and minimizes downtime by addressing issues before they result in actual failures.

There are no dependencies between the tables.

Data Encoding

In resource-constrained IoT scenarios, the efficiency of the chosen protocol is crucial. Since we do not have constraints that force us to use XML (there are no other devices already deployed to interact with using XML), we opted to use JSON, which offers a simpler and more lightweight syntax compared to XML. As a result, JSON generally yields smaller data payloads since it utilizes concise syntax without the inclusion of verbose tags and attributes found in XML. This characteristic leads to reduced bandwidth usage and reduce the resource used by the sensor for computing the packet.

Deployment and Execution

In order to be reachable from the remote control application both sensors and actuator must register themselves to a CoAP Registration Server. Each node sends a payload containing the name of the node. The CoAP registration server then will store information regarding the IP address and the status of each node.

CoAP Network

Sensors

The sensors deployed in the system behave as CoAP (Constrained Application Protocol) servers, each exposing an observable resource to notify the collected data:

- Pressure: “**res-pressure**”
- Voltage: “**res-voltage**”
- Rotation: “**res-rotation**”
- Vibration: “**res-vibration**”

Additionally, they expose another resource to monitor and modify their operational state:

- Pressure: “**res-pressure-status**”
- Voltage: “**res-voltage-status**”
- Rotation: “**res-rotation-status**”
- Vibration: “**res-vibration-status**”

Data collection is simulated by generating random values from each sensor every hour. These random numbers are obtained from a normal distribution given *mean* and *standard deviation*.

The operational state of each sensor indicates whether it is currently operating or not. By toggling this state from 0 to 1, allows the sensor to be turned on or off. The state simulation can be triggered either by physically pressing the button present on the sensor or remotely via a user application.

The payload sent by any sensor follows this format: {“**sensor**”: (type), “**value**”: number} for the telemetry and {“**sensor**”: (type), “**status**”: 1/0} for the state.

Actuator

The actuator in the system serves a dual role, functioning both as a client and a server. As a server, it exposes an observable resource to notify the predictions obtained from the machine learning model:

- Alarm: “**res-alarm**”

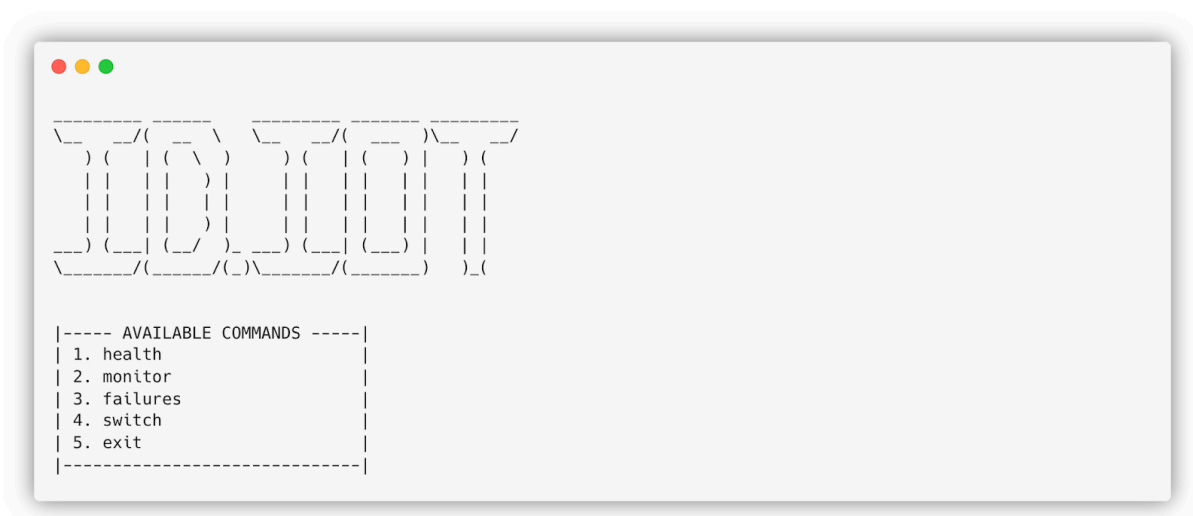
As a client, it requests the IPv6 addresses of the sensors from the cloud application and subsequently registers as an observer to each sensor. Upon receiving telemetry notifications from sensors, the actuator records the values in a queue. After a 24-hour period, the actuator can make a prediction. It first calculates the average of the values obtained from each sensor and utilizes these average values to make the prediction. Each prediction is then sent to the cloud application to be stored in the database. If the model predicts a potential failure (class 0, 1, 2, or 3) of a component, the actuator triggers an alarm by blinking a red LED.

Periodically, the actuator also sends requests to each sensor to monitor their operational status. If a sensor is found to be offline, the actuator cannot make predictions, as it relies on data from all sensors to generate accurate predictions.

In order to make accurate predictions, the actuator stores the number of errors of each type (1, 2, 3, 4, 5) that have occurred in the previous 3 hours. For each change or increment in these values, it is simulated by pressing the button present on the actuator. Each press sequentially increments the value of each counter: the first press increases the first counter (error1_count), the second press increases the second counter (error2_count), and so on. After the fifth press, the incrementation cycle restarts with the first counter.

This mechanism allows the actuator to keep track of the occurrence of different types of errors over time, enabling it to adjust its predictions based on recent error trends. By simulating changes through button presses, the actuator can mimic real-world scenarios and ensure that its predictive capabilities remain up-to-date and reflective of current system conditions.

Remote Control Application



This application allows users to interact with the system and perform the following operations.

health

This command retrieves and displays the current status of all sensors and actuators in the system. It connects to the database, fetches the status and IP addresses of the sensors and actuators, and prints them in a formatted output.

```
COMMAND> health

|----- SYSTEM HEALTH -----|
| Pressure:    1 at fd00::202:2:2:2 |
| Vibration:   1 at fd00::204:4:4:4 |
| Voltage:     1 at fd00::205:5:5:5 |
| Rotation:    1 at fd00::203:3:3:3 |
| Alarm:       1 at fd00::206:6:6:6 |
|-----|
```

monitor

This command starts monitoring the system by observing the sensors in real-time. It creates observer objects for each active sensor (excluding the alarm) and keeps them under observation. This is useful for continuously checking the sensors' status and values.

```
COMMAND> monitor
System is being monitored. Press 'Ctrl + C' to stop monitoring.
Starting observation at fd00::202:2:2:2 and port 5683, resource pressure
Starting observation at fd00::204:4:4:4 and port 5683, resource vibration
Starting observation at fd00::205:5:5:5 and port 5683, resource voltage
Starting observation at fd00::203:3:3:3 and port 5683, resource rotation

🟡🟡🟡🟡 VIBRATION TELEMETRY 🟡🟡🟡🟡 : 42.97
🟢🟢🟢🟢 PRESSURE TELEMETRY 🟢🟢🟢🟢 : 83.87
🟢🟢🟢🟢 VOLTAGE TELEMETRY 🟢🟢🟢🟢 : 144.09
🟢🟢🟢🟢 ROTATION TELEMETRY 🟢🟢🟢🟢 : 430.04
🟢🟢🟢🟢 PRESSURE TELEMETRY 🟢🟢🟢🟢 : 89.04
🟡🟡🟡🟡 VIBRATION TELEMETRY 🟡🟡🟡🟡 : 32.93
🟢🟢🟢🟢 VOLTAGE TELEMETRY 🟢🟢🟢🟢 : 144.51
🟢🟢🟢🟢 ROTATION TELEMETRY 🟢🟢🟢🟢 : 390.47
```

failures

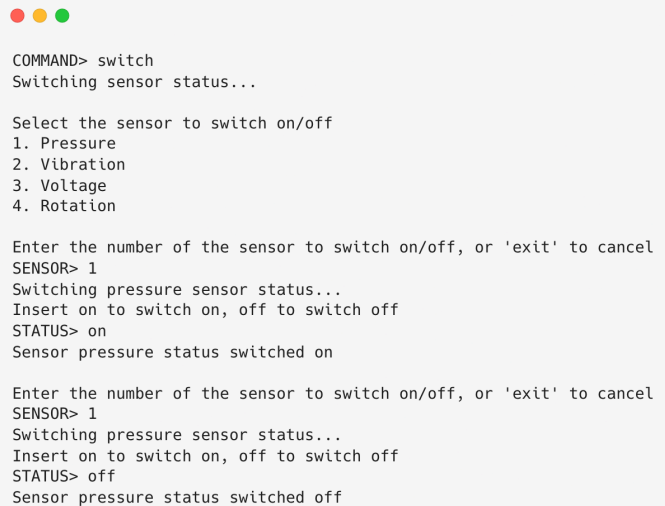
This command retrieves and displays a list of components that need to be replaced to prevent potential failures. It queries the failures table in the database and prints the components along with their respective timestamps.

```
COMMAND> failures

|----- COMPONENT HEALTH CHECK ----|
| STATUS OK:   2024-05-29 22:00:39 |
| STATUS OK:   2024-05-29 22:00:53 |
| Component 1: 2024-05-30 14:14:56 |
| Component 2: 2024-05-30 14:16:04 |
| Component 2: 2024-05-30 14:16:34 |
| Component 2: 2024-05-30 14:17:04 |
| Component 2: 2024-05-30 14:20:11 |
| Component 3: 2024-05-30 14:20:14 |
| Component 3: 2024-05-30 14:20:17 |
| Component 3: 2024-05-30 14:20:20 |
| Component 2: 2024-05-30 14:20:23 |
| Component 2: 2024-05-30 14:20:26 |
| Component 5: 2024-05-30 14:20:35 |
| Component 5: 2024-05-30 14:20:38 |
| Component 5: 2024-05-30 14:20:41 |
| STATUS OK:   2024-05-31 15:48:09 |
|-----|
```

switch

This command allows you to switch the status of a specific sensor on or off. You will be prompted to select a sensor from the list, and then specify whether to switch it on or off. This command sends a CoAP request to the sensor to change its status.



```
COMMAND> switch
Switching sensor status...

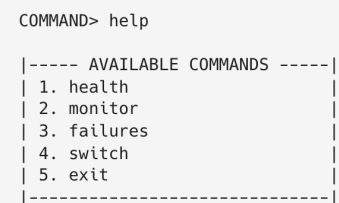
Select the sensor to switch on/off
1. Pressure
2. Vibration
3. Voltage
4. Rotation

Enter the number of the sensor to switch on/off, or 'exit' to cancel
SENSOR> 1
Switching pressure sensor status...
Insert on to switch on, off to switch off
STATUS> on
Sensor pressure status switched on

Enter the number of the sensor to switch on/off, or 'exit' to cancel
SENSOR> 1
Switching pressure sensor status...
Insert on to switch on, off to switch off
STATUS> off
Sensor pressure status switched off
```

help

This command displays the list of available commands and their brief descriptions. It is useful for reminding users of the available functionalities and how to use them.

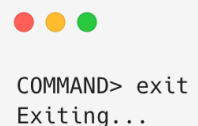


```
COMMAND> help

|----- AVAILABLE COMMANDS -----|
| 1. health                          |
| 2. monitor                         |
| 3. failures                        |
| 4. switch                          |
| 5. exit                            |
|-----|
```

exit

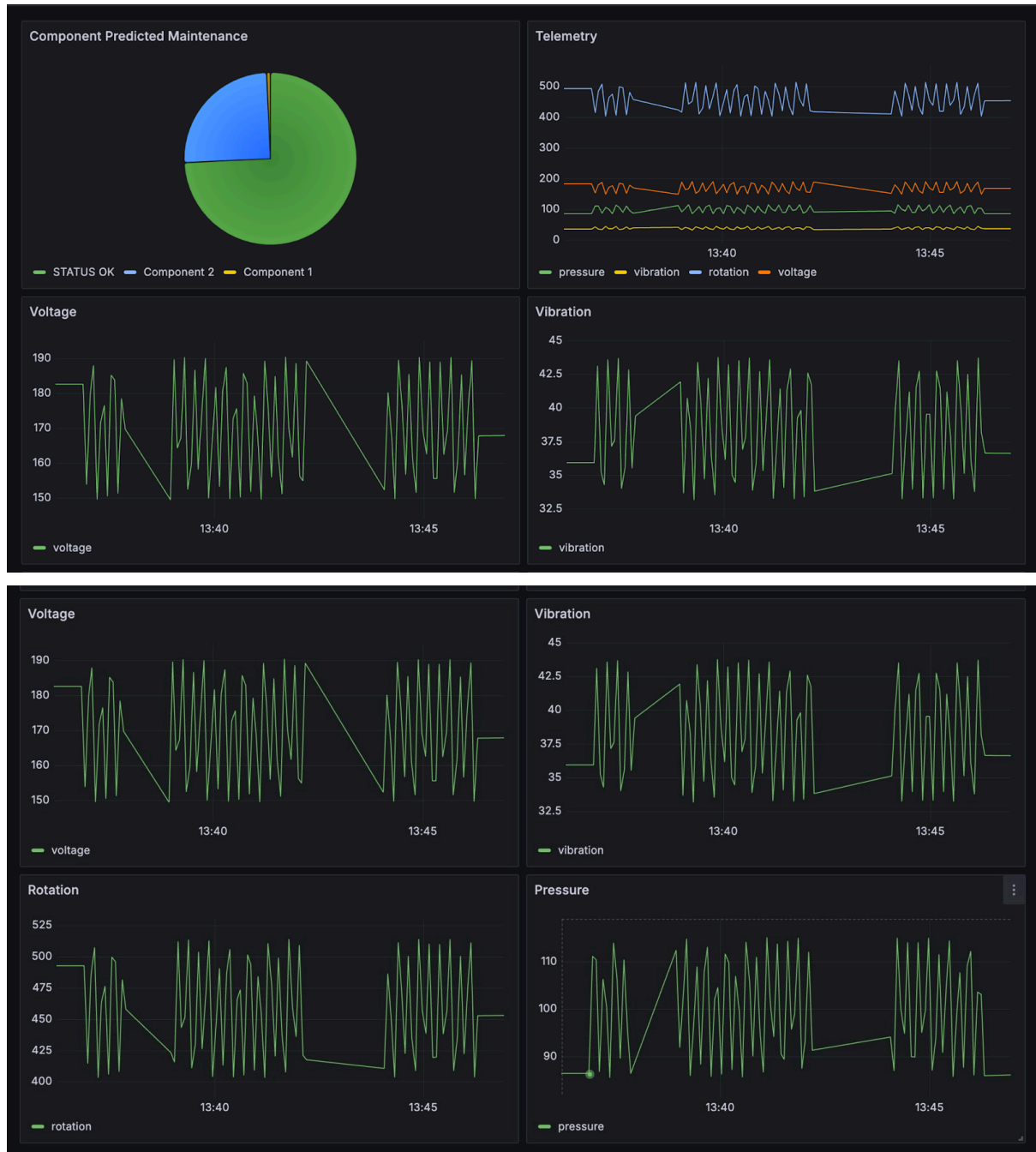
This command exits the application. It terminates the program gracefully, ensuring any ongoing processes are properly stopped.



```
COMMAND> exit
Exiting...
```

Grafana

Grafana dashboard is used to compose some analytics based on the values that are stored on the database.



Machine Learning model²

The implemented model represents a robust predictive maintenance system designed to anticipate and prevent component failures within an industrial IoT environment. Anchored by a decision tree classifier, the model harnesses the power of machine learning to analyze telemetry data collected from sensors. Initially, the dataset³ is meticulously curated to focus exclusively on relevant data pertaining to a specific model (“model1”), while superfluous columns are omitted to streamline processing. Leveraging time series cross-validation ensures that temporal dependencies within the data are preserved, a critical aspect in accurately evaluating the model’s performance. The choice of the F1 score as the evaluation metric underscores the model’s ability to strike a balance between precision and recall, crucial in the context of predictive maintenance. The model undergoes iterative refinement, with one iteration utilizing all available features and another employing a reduced feature set based on meticulous feature importance analysis. This iterative approach not only enhances the model’s interpretability but also optimizes its predictive accuracy. Visualization techniques, such as boxplots, offer clear insights into the comparative performance of the model under different feature configurations, aiding in informed decision-making. Finally, the trained decision tree classifier is converted into a deployable format using the *emlearn* library, facilitating seamless integration into operational workflows.

² [Predictive Maintenance using Machine Learning | by Medini Kumar Bora | Medium](#)

[Machine Learning/Predictive Maintenance Modelling Guide Python Notebook.ipynb at master · vikasgupta1812/Machine Learning \(github.com\)](#)

³ [Microsoft Azure Predictive Maintenance \(kaggle.com\)](#)

Predictive Maintenance Use Case

Predictive maintenance in the industrial environment aims to anticipate and prevent potential failures before they occur, ensuring smooth operation and minimizing downtime. By leveraging data from various sensors and actuators, the system can predict and address maintenance needs proactively.

Scenario

In an industrial environment, the predictive maintenance system continuously monitors machinery and equipment using sensors to collect data such as pressure, voltage, rotation, and vibration. Actuators remain ready for activation and are visible within the network.

Implementation

1. Data Monitoring:
 - Sensors continuously sample data of pressure, voltage, rotation speed, and vibration of machines
 - Actuators such as alarms, remain ready for activation as needed.
2. Predictive Action:
 - Depending on the prediction, appropriate actions are taken, such as turning on the alarm system.
3. Preventive Maintenance:
 - By addressing potential issues proactively and implementing preventive measures, the system aims to prevent equipment failures or breakdowns that could disrupt operations.

Conclusion

Through predictive maintenance, the industrial system can anticipate and mitigate potential equipment failures, ensuring reliable operation and maximizing productivity. By proactively addressing maintenance needs and optimizing equipment performance, the system enhances operational efficiency and reduces the risk of costly downtime.