Computer Engineering Department

Faculty Of Engineering

Helwan University

Maintenance Assistance System

(M.A.S.)

Supervised by:

Assoc. Prof. Amr El-Sayed Mohamed.

BY

Ahmed Mohamed Ahmed El-Nassag.

Solwan Shokry Ahmed Mohamed.

Mahmoud Reda Ezz El-Arab.

Marwa Abdelghany Ibrahim Elfeqy.

Mostafa Farid Saad Ayad.

Table of Contents

Table of Abbreviations 7

Abstract 9

Chapter 1 : Introduction 10

1.1. Problem Statement 10

1.2. Goals & Objectives 11

1.3. Previous Solutions 12

1.4. Proposed Solution 13

1.4.1. Block Diagram 13

Chapter 2 : Background 15

2.1. Machine Learning (ML) 15

2.2. Predictive Maintenance 17

2.3. Firmware Over The Air 17

2.4. Internet of Things (IoT) 19

2.5. Boot Loaders 23

2.6. Graphical User Interface (GUI) 25

2.7. CAN Bus 31

Chapter 3 : Previous Related Work 37

Chapter 4 : System Requirements and Specification 44

4.1. System Requirements 44

4.1.1. Enumerated Functional Requirements 44

4.1.2. Enumerated Non-Functional Requirements 45

4.2. Functional Requirements Specification 45

4.3. System Design 46

4.4. Use Case Diagram 50

4.5. Use Cases Description 50

4.6. Sequence Diagrams 58

4.7. Tools and Technologies 59

4.8. Project Hardware 60

Chapter 5 : Predictive Maintenance System Implementation 61

5.1. Use Case 62

5.2. Model Architecture 62

5.2.1. Convolutional Neural Networks (CNN) 63

5.2.2. Long Short-term Memory (LSTM) 64

5.2.3. Hybrid CNN-LSTM Model for Pd.M. 65

5.3. Pd.M. Model 67
5.3.1. Data Set 67
5.3.2. Exploratory Data Analysis 68
5.3.3. Label Generation 71
5.3.4. Model Implementation 72
5.3.5. Pd.M. Model Training and Evaluating 73
5.4. Engine Condition Monitoring 74
5.4.1. Sensors' Readings 74
5.5. CAN in Our Project 79
5.5.1. The Specification Transmitting of CAN: 79
5.5.2. The Specification Receiving of CAN: 80
5.5.3. Transmit Handling 80
5.5.4. Reception Handling. 81
5.5.5. CAN in PDM 81
5.5.6. CAN in FOTA 81
5.5.7. Steps to Connect CAN Network. 82
Chapter 6 : FOTA System Implementation 83
6.1. Firmware Over The Air (FOTA) 83
6.1.1. Hexadecimal File (HEX File) 83
6.1.2. Flashing Firmware 85
6.1.3. Boot Loader Design 89
6.1.4. Boot Loader Sequence 90
6.1.5. Boot Loader Software Architecture 91
6.1.6. Internet of Things (IoT) 91
6.1.7. ESP8266 Sequence of Operations 92
6.1.8. ESP8266 Hardware Connection 92
6.1.9. Data Integrity 93
6.1.10. Recovery Mechanism 94
6.2. Website 96
6.3. HMI ECU 103
6.3.1. How To Connect Raspberry Pi to CAN Bus 104
6.3.2. GUI Design 105
6.3.3. GUI Description and Features 105
6.3.3.1. User Sign In and Sign Up 106
6.3.3.2. Engine Diagnosis 107

6.3.3.3. PDM (Predictive Maintenance) 108
6.3.3.4. FOTA 110
Chapter 7 : Conclusion 114
7.1 Results 114
7.2 Future Work 115

Table of Figures

Figure 1.1 M.A.S. Flowchart 11
Figure 1.2 Block Diagram 14
Figure 2.1 Internet of Things Applications 20
Figure 2.2 IoT systems 22
Figure 2.3 Bootloader operation flowchart 24
Figure 2.4 Booting sequence 24
Figure 2.5 GUI layers 26
Figure 2.6 GUI in cars 26
Figure 2.7 GUI applications 27
Figure 2.8 Signals and slots 29
Figure 2.9 QWidget 30
Figure 2.10 QLayOut 30
Figure 2.11 CAN network 31
Figure 2.12 CAN transmission handling 33
Figure 2.13 Receive FIFO states 34
Figure 2.14 Error frame 35
Figure 2.15 Overload Frame 35
Figure 2.16 Data and Remote frames 36
Figure 3.1 framework for self-verification of firmware updates over the air in vehicle ECUs 39
Figure 3.2 The base network topology 40
Figure 3.3 OTA security 41
Figure 3.4 OTA Update Process and Architecture 41
Figure 4.1 PDM Flow Chart 46
Figure 4.2 Update over the air 47
Figure 4.3 update flow chart 48
Figure 4.4 Roll back flow chart 49
Figure 4.5 Use case Diagram 50
Figure 4.6 FOTA sequence diagram 58
Figure 4.7 Failure prediction sequence diagram 59
Figure 5.1 CNN structure 64
Figure 5.2 LSTM structure 65
Figure 5.3 Hybrid CNN-LSTM structure 66
Figure 5.4 Distribution of the different machine features 68
Figure 5.5 Vibration of Machine 1 for 2015 69
Figure 5.6 Voltage of Machine 2 for 1st two weeks of 2015 69
Figure 5.7 Frequency of different error conditions along with a set of 100 machines 69
Figure 5.8 Machine to type of error distribution 70

Figure 5.9 Number of Maintenance Issues raised per day....70
Figure 5.10 Distribution of age of the Machines....70
Figure 5.11 Processing and features generation....71
Figure 5.12 Model confusion matrix....73
Figure 5.13 BMP180....75
Figure 5.14 Interfacing with Application ECU....75
Figure 5.15 Measurement flow BMP180....76
Figure 5.16 801S vibration sensor....77
Figure 5.17 Interfacing 801S....77
Figure 5.18 MLX90217....78
Figure 5.19 Hall effect sensor operation....79
Figure 5.20 CAN transmit configuration....79
Figure 5.21 CAN receive configurations....80
Figure 5.22 Transmitting handling....80
Figure 5.23 Reception handling....81
Figure 5.24 CAN in FOTA....82
Figure 5.25 Loop back mode....82
Figure 6.1 Hex file....84
Figure 6.2 Hex record structure....85
Figure 6.3 Off-circuit Programming....86
Figure 6.4 Debugger....87
Figure 6.5 In-circuit programming....87
Figure 6.6 Automotive industry....88
Figure 6.7 System network....89
Figure 6.8 Flash memory design....89
Figure 6.9 Bootloader design....90
Figure 6.10 Boot loader sequence....90
Figure 6.11 Boot loader software architecture....91
Figure 6.12 ESP8266....91
Figure 6.13 AT Commands....92
Figure 6.14 ESP sequence of operations....92
Figure 6.15 Internal connection....93
Figure 6.16 Hex file fields....93
Figure 6.17 Checksum algorithm....94
Figure 6.18 Banking design....95
Figure 6.19 Recovery algorithm....95
Figure 6.20 IOT internal connection....96
Figure 6.21 Web server....97
Figure 6.22 OEM server....98
Figure 6.23 services page....99
Figure 6.24 Home page....100
Figure 6.25 Firmware upload page....100
Figure 6.26 Sign page....101
Figure 6.27 Back-end flowchart....102
Figure 6.28 Raspberry Pi....103
Figure 6.29 SPI to CAN module....104

Figure 6.30 Raspberry pi connection 105
Figure 6.31 Home Screen 106
Figure 6.32 Sign In 106
Figure 6.33 Sign up 107
Figure 6.34 Engine Diagnosis 107
Figure 6.35 PDM 108
Figure 6.36 PDM flow chart 109
Figure 6.37 FOTA notifications tab 110
Figure 6.38 FOTA flow chart 111
Figure 6.39 FOTA update history 112
Figure 6.40 Roll back flow chart 113
Figure 7.1 Future work 115

List of Tables

Table 0.1: Table of Abbreviations 7
Table 5.1: A detailed description of different data sources. 68
Table 5.2: Display the first few rows of the Telemetry data for Machine 1 69
Table 5.3:Train, validation, and test data splits 71
Table 5.4: Features of the generated dataset 71
Table 5.5: Different parameter settings. 72
Table 5.6: Model evaluation metrics 73

Table of Abbreviations

Table 0.1: Table of Abbreviations

ACK	Acknowledgement	**HMI**	Human-Machine Interface
ANN	Artificial Neural Networks	**ICP**	In-Circuit Programming
AT Commands	Attention Commands	**IOT**	Internet Of Things
BL	Boot Loader	**IP**	Internet Protocol
CAN	Controller Area Network	**JTAG**	Joint Test Action Group
CNN	Convolutional Neural Network	**LSTM**	Long Short-term Memory
CNN-LSTM	Hybrid CNN-LSTM Model	**MAS**	Maintenance Assistance System
DBN	Deep Belief Networks	**MCAL**	Micro-Controller Abstraction Layer.
DL	Deep Learning	**OEM**	Original Equipment Manufacture
DNN	Deep Neural Networks	**OS**	Operating System
ECU	Electronic Control Unit	**OTA**	Over The Air
FIFO	First In First Out	**Pd.M.**	Predictive Maintenance
FOTA	Firmware Over The Air	**ReLU**	Rectified Linear Unit
FPEC	Flash Program and Erase Controller	**RTOS**	Real-Time Operating System
GUI	Graphical User Interface	**SFOTA**	Secured Firmware Over The Air
HAL	Hardware Abstraction Layer	**SPI**	Serial Peripheral Interface
Hex File	Intel Hexadecimal Object File Format	**UART**	Universal Asynchronous Receiver-Transmitter

Acknowledgment

We would like to express our heartfelt appreciation to our chair of committee, especially our Professor Dr. Amr El Sayed, our project supervisor, for his enthusiasm, patience, insightful comments, useful information, practical advice, and never-ending ideas, all of which have been invaluable to us throughout the course of our research and writing for this thesis. Working and studying under his supervision was a huge honour and privilege. We are grateful for everything he has done for us. We sincerely hope he is satisfied with our performance. We could not have finished this project without his guidance and assistance. We could not have asked for a better study supervisor.

Lastly, we would be remiss in not mentioning our family, especially our parents. Their belief in us has kept our spirits and motivation high during this process.

Abstract

Vehicle breakdowns are dangerous and expensive, despite the recent upgrades in the quality of automobile designs that have improved protection and safety, car breakdowns still occur on roads. In truth, they represent a chief percentage of street incidents, on freeways. They can negatively affect individuals, companies, and countries due to time and cost losses, or worse, such as accidents, injuries, and potential deaths. Also, with the evolution in automotive industry vehicles have become more connected. On average, an automotive vehicle today comprises of approximately one hundred ECUs, the potential for errors in the software in that vehicle increases. It could be a simple error affecting the user experience negatively, or it could be a fault in a critical part such as the brakes or engine that could expose the driver and passengers to danger, or it could put their lives at risk.

The increasing demand for advanced safety, comfort, and persuasion systems has pushed us to move forward with the Maintenance Assistance System (MAS). which can help drivers by sending a warning to the driver if a failure is expected, to inform him of the need to go to the maintenance center. Also, if the failure can be prevented by the software, our system provides the ability to install bug fixes and firmware updates without unnecessary visits to the maintenance center.

By using Machine learning algorithms, and IoT technologies our system provides the ability to processes the condition of the vehicle's critical part of interest, -in our case; the engine-, monitored with the use of various sensors to get periodic readings of the engine's pressure, vibration, and speed. And the ability to remotely update firmware of vehicles' ECUs so that the driver is not required to visit maintenance centres to resolve this issue.

Chapter 1 : Introduction

We all use cars and transportation in our daily lives as drivers or passengers. This exposes us to many possibilities that may affect our lives significantly. Some of us may be exposed to a malfunction or failure in the car that may lead to its breakdown, and we end up on the side of the road with a broken car, wasted time, and a loss of money. This malfunction may be in a critical part of the car such as the brakes or the engine and the tires due to unprofessional or delayed car maintenance, which puts us at risk of an accident [1]. As a result, our lives and the lives of others are at risk.

There had been around 20,000 yearly reported on-road car breakdowns in comparison to the 27,000 yearly crashes in New South Wales (NSW) between 2012 and 2015 [3]. The total number of road breakdowns may be higher due to underreporting.

Road Safety Statistics

- Commercial vehicles are experiencing more on-road failures than ten years ago. Estimates of downtime costs for fleets range from $448-$760 a day.
- According to the World Health Organization (WHO) approximately 1.35 million people die in road crashes each year.
- Road crashes are the leading cause of death in the U.S. for people aged 1-54 [2].

1.1. Problem Statement

Road accidents frequently involve a mechanical part sudden failure, which wasn't anticipated, and which could have been avoided if the vehicle owner had been able to predict it earlier. Also, the rate of errors and failures in the vehicle increases as a result of the development and advancement of technology in the fields of software and the design and manufacture of automobiles. This can be seen as the primary contributing factor to most traffic accidents since vehicles now have a number of complex systems that are controlled by software. The causes of failure are divided into two main parts:

1. Hardware Failures

   A failure due to a mechanical part in the vehicle in which a drop in performance or its lifetime has expired and needs maintenance or replacement.

2. Software Failures

   A vehicle failure that occurred due to an error in the software that controls the vehicle.

After we discussed the main causes of road accidents, we need to be able to prevent them in order to provide on road safety, and better user experience.

1.2. Goals & Objectives

Our main goal is to develop a MAS (Maintenance Assistance System) to reduce or prevent vehicle downtime that can negatively affect individuals, companies, and countries due to time and cost losses, or worse, such as accidents, injuries, and potential deaths. We will deal with these cases by addressing and preventing the causes that may lead to a vehicle failure. By alerting the driver that a failure is about to occur so that he can take the necessary steps to prevent it and thus reduce losses and provide a better customer experience.

We started working on our project with the intention of creating a system that can make the maintenance process of vehicles critical parts a lot easier. With that in mind our main objective was to satisfy the requirements of our system, and the sequence shown in Figure below was what we kept in mind throughout the entire planning, designing, and development processes.

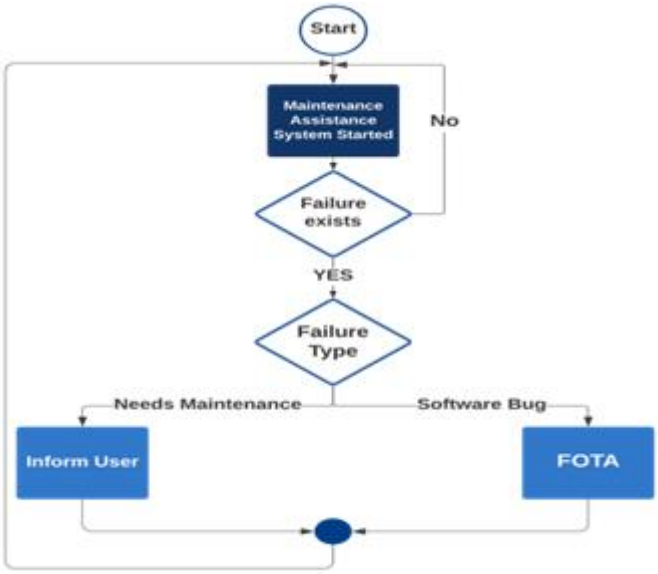

Figure 1.1 M.A.S. Flowchart

To build a system that starts and monitor the critical parts conditions periodically, with the use of various sensors, and keeps providing the data read from those sensors to our Pd.M. model. The ML model keeps processing the data it received from the sensors' ECU to detect whether a failure is predicted to be occurring or not. Once the system detected a failure it should then start classifying its type. If it's a failure detected in the critical mechanical part, our system being a software system hasn't got the ability to solve the problem, but it can always notify the user with the information at hand; A

failure is expected in a vehicles critical part! The user is then aware of the critical part condition and is responsible to head to the nearest maintenance centre to replace the degraded, worn-out mechanical part of the vehicle.

On the other hand, if the failure occurred to be of a one-time-failure type, typically an error in the software in the form of a software bug, error, or even an abnormal functionality that may lead to a serious risk of the user's life. The second phase of our system shall handle those types of failures with the use of our FOTA system, by remotely providing the appropriate bug fix or even a new firmware version.

That way we can achieve timely maintenance experience for the user, and saving unnecessary costs, time waste, avoid the risk of vehicles down time and the occurrence of accidents resulting from it, and provide better user experience for the customer.

1.3. Previous Solutions

Previous Solutions for Software Failures

In the automotive industry, Tesla released the first FOTA software upgrade for their "Model S" automobiles in September 2012. The company released an upgrade and utilised either the built-in 3 G network link in the customer's automobile or their home network's Wi-Fi signal. From 2017 to 2018, updates were started to be delivered via OTA by several car OEMs, including General Motors, BMW, Volvo, the Detroit Diesel Organization, and Mercedes-Benz. By 2020, Ford also intends to send updates via OTA. Along with automobile OEMs, agricultural machinery manufacturers including John Deere, AGCO Company, and CNH Industrial have started offering straightforward OTA updates for their farm equipment.

According to The Research and Economies, the market for Automotive Over the Air (FOTA) updates is anticipated to grow at a CAGR of 58.15% from 2018 to 2022. It's also a truth that many of the biggest firms in the automotive industry are already moving toward a FOTA upgrade that is stable and smooth.

There are protocols for secure download, but none for ensuring proper firmware installation and memory verification [67]. As for the detailed work-done in similar projects, various case of studies has been introduced in multiple research frameworks which were published in the form of IEEE papers.

Previous Solutions for Hardware Failures

Deep learning models have provided effective solutions in fault diagnosis due to their powerful feature learning abilities. They build a set of representation methods using

numerous layers and learn the non-linear representation of any time series to a higher level of complexity and abstraction [35].

"A review on the application of deep learning in system health management" paper proposed a simple auto-encoder, CNN and RNN-based machine health monitoring system Deep learning has been applied to fault diagnosis and prediction, and this expansion has spanned from mechanical equipment monitoring to electrical systems, power installations, and aviation specialties. This includes solutions for electromechanical equipment fault diagnosis, deterioration classification and pattern recognition, and component RUL prediction.

The most recent presented is a synthesis of articles reviewed during an on-going research project on the application of deep learning. The authors adopted a pragmatic approach and focused their efforts on the discipline health management system; Rather than providing a comprehensive deep learning survey file [36].

We decided to apply these algorithms to apply them in the automotive field to make the most of the ability to monitor the wear of mechanical parts and use them as solutions to predict the life of the monitored part.

1.4. Proposed Solution

The increasing demand for advanced safety, comfort and persuasion systems has pushed us to move forward with a Maintenance Assistant System solution. By sending a warning to the driver if a failure is expected, inform him of the need to go to the maintenance center.

Users achieve timely maintenance towards increased up-times, better maintenance plan by reducing unnecessary field service calls, optimizing repair parts replacement, reducing unplanned downtime, improving vehicle performance, avoiding human casualties, and lowering maintenance cost.

1.4.1. Block Diagram

Our proposed solution consists of 4 main controllers as shown in the figure below.

1- Main ECU: it is connected to the ESP8266 Wi-Fi module and acts as the gateway to the OEM server.

2- Engine ECU: It’s connected with sensors, and it’s responsible for collecting data from the car’s engine.

3- Target ECU: it acts as a faulty node whose software needs to be updated or has a feature added to it.

4- HMI ECU: it consists of Raspberry-pi model 3B and HDMI touch screen it acts as an interface to the user and contains predictive maintenance model.

The four ECUs of the system connected through a CAN network.

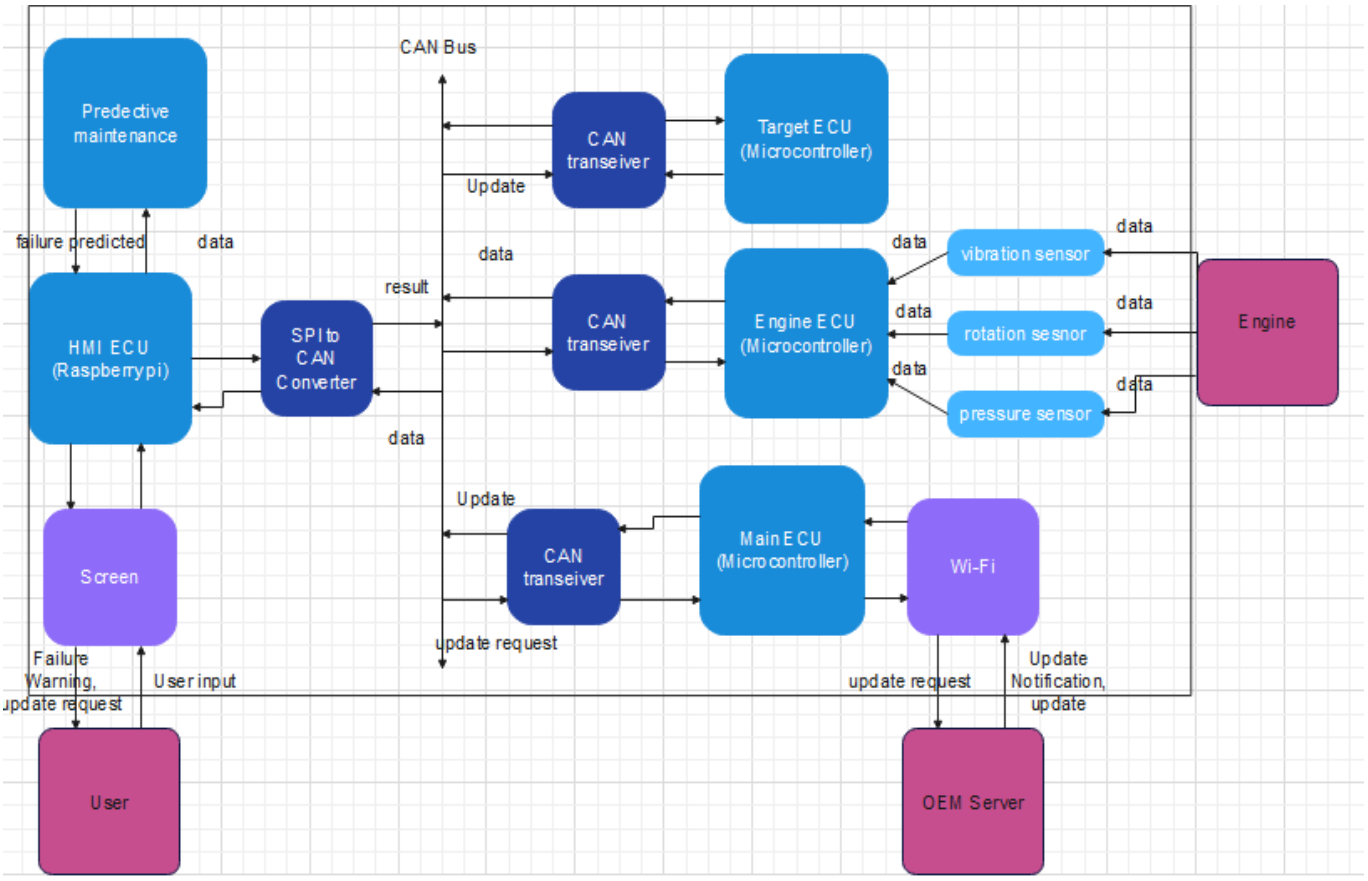

Figure 1.2 Block Diagram

The remainder of this thesis is structured as follows:

Chapter 2 discusses Background of the project. In Chapter 3, we provide a detailed view on the previous related work. In chapters 4 we provide our system requirements and specifications. In chapter 5 and 6 we discuss in detail our proposed MAS project. We conclude our project in chapter 7 presenting our results and future work to be done.

Chapter 2 : Background

2.1. Machine Learning (ML)

ML is a field of inquiry devoted to understanding and building methods that 'learn', that is, methods that leverage data to improve performance on some set of tasks. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as training data, to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks [37].

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers, but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory, and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning. Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain.

Supervised learning

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as training data and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an array or vector, sometimes called a feature vector, and the training data is represented by a matrix. Through iterative optimization of an objective function, supervised learning algorithms learn a function that can be used to predict the output associated with new inputs [38].

An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task. Types of supervised-learning algorithms include active learning, classification, and regression.

Classification algorithms are used when the outputs are restricted to a limited set of values, and regression algorithms are used when the outputs may have any numerical value within a range. As an example, for a classification algorithm that filters emails,

the input would be an incoming email, and the output would be the name of the folder in which to file the email.

Similarity learning is an area of supervised machine learning closely related to regression and classification, but the goal is to learn from examples using a similarity function that measures how similar or related two objects are. It has applications in ranking, recommendation systems, visual identity tracking, face verification, and speaker verification.

Convolutional neural network

In deep learning, a convolutional neural network (CNN, or Conv-Net) is a class of artificial neural network (ANN), most applied to analyse visual imagery. CNNs are also known as Shift Invariant or Space Invariant Artificial Neural Networks (SIANN), based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation-equivariant responses known as feature maps [56].

Counter-intuitively, most convolutional neural networks are not invariant to translation, due to the down sampling operation they apply to the input. They have applications in image and video recognition, recommender systems, image classification, image segmentation, medical image analysis, natural language processing, brain–computer interfaces, and financial time series [40].

Long short-term memory

Long short-term memory (LSTM) is an artificial neural network used in the fields of artificial intelligence and deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. Such a recurrent neural network can process not only single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition, machine translation, robot control, video games, and healthcare LSTM has become the most cited neural network of the 20th century [44].

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. LSTM networks are well-suited to classifying, processing, and making predictions based on time series data, since there can be lags of unknown duration between important events in a time series [45].

LSTMs were developed to deal with the vanishing gradient problem that can be encountered when training traditional RNNs. Relative insensitivity to gap length is an advantage of LSTM over RNNs, hidden Markov models and other sequence learning methods in numerous applications [39].

2.2. Predictive Maintenance

Predictive maintenance techniques are designed to help determine the condition of in-service equipment to estimate when maintenance should be performed. This approach promises cost savings over routine or time-based preventive maintenance because tasks are performed only when warranted. Thus, it is regarded as condition-based maintenance carried out as suggested by estimations of the degradation state of an item [31].

The main promise of predictive maintenance is to allow convenient scheduling of corrective maintenance, and to prevent unexpected equipment failures. The key is "the right Info equipment lifetime, increased plant safety, fewer accidents with negative impact on environment, and optimized spare parts handling. Predictive maintenance differs from preventive maintenance because it relies on the actual condition of equipment, rather than average or expected life statistics, to predict when maintenance will be required [32].

Typically, Machine Learning approaches are adopted for the definition of the actual condition of the system and for forecasting its future states. Some of the main components that are necessary for implementing predictive maintenance are data collection and pre-processing, early fault detection, fault detection, time to failure prediction, maintenance scheduling and resource optimization. Predictive maintenance has also been one of the driving forces for improving productivity and one of the ways to achieve "just-in-time" in manufacturing.

2.3. Firmware Over The Air

What if we predict that the failure is imminent and it turns out that this failure can be fixed with a software update, why does the driver have to go to the service center and waste time and money? We need to be able to send car bug fixes in the form of a remote update to the car and not force the driver to go to service centers to solve this problem.

Vehicles have evolved rapidly from being mechanical machines to smart connected machines that define luxury, safety, and comfort. We now have various complex systems in a car that are driven by software pieces.

On average, an automotive vehicle today comprises of approximately 100 Electronic Control Units (ECU) and over 100 million lines of software code. With our increasing need to develop advanced functionality, there is a need to update the software running on these embedded devices. Updates could be issued to improve existing functionality or to remedy discovered bugs.

Recalling vehicles in the field to correct field problems, to introduce new features and to upgrade vehicle performance could cause us to lose our reputation as an OEM and incurrence of huge costs.

To provide superior driving experience and reliability for our customers, it is important to keep the software and firmware pieces up to date. When the driver must go to the service station to update the software, it not only increases cost, but it also becomes a time consuming and tedious affair.

This is exactly what OTA updates can do for regular as well as autonomous cars. An emerging trend for vehicle manufacturers is to perform firmware updates over the air (FOTA) [72].

FOTA provides:

- **Cost- Efficient and better managed firmware update**.

  It involves minimal customer inconvenience since there exists no need for the customer to bring the vehicle to a service station for a firmware update

- **Upgrade the firmware safely, anytime & anywhere**.

  As soon as the firmware is released it can be downloaded and installed in the vehicle. The firmware is downloaded over a wireless network connection from a trusted portal to the vehicle and then flashed to the correct ECU [71].

FOTA took 4 management stages process:

- **Initiation:** OEMs initiate the main services.
- **Planning**: Plan the main network flow according to the available resources.
- **Execution:** Execute different methodologies and technical flows.
- **Closure**: Customer review and application feedback.

Through these stages FOTA shall facilitate:

- Enables fabricators to patch glitches in new systems.

- Allows OEMs to be able to send and install new software updates and features remotely after customers have bought the cars.

In the automobile industry, in September 2012, Tesla delivered the first FOTA software update for their 'Model S' vehicles. The business published an update and used either the car's integrated 3 G network link or a Wi-Fi signal from the customer's home network [10].

Other automotive OEMs such as General Motors, BMW, Volvo, Detroit Diesel Organization and Mercedes-Benz have begun to deliver updates to OTA from 2017-2018. Ford also plans to deliver updates to OTA by 2020. Agricultural machinery manufacturers including John Deere, AGCO Company, and CNH Industrial have begun providing simple OTA updates for their farm machinery along with automotive OEMs.

In the automotive industry, FOTA updates go hand in hand with self-driving and connected vehicles. IoT updates will be an important part of the near future, according to industry analysts, by 2030, 98 percent of cars sold worldwide are projected to be connected vehicles. The value produced by next-generation cars (in terms of the cost of the vehicle) will be totally different than the current ones; the software will be a key selling point in next-generation cars. To sell their vehicles soon, automotive OEMs will deliver robust software along with hardware systems.

Industry forecasts say that FOTA 's future in automotive is looking promising. This is relevant to the automotive industry which is shifting its gear towards software-driven autonomous systems development.

The Research and Economies reported that the Automotive Over the Air (FOTA) updates market is expected to expand at a CAGR of 58.15 per cent over the period 2018-2022.And many of the largest automobile players are already on the path of making FOTA in automotive systems seamless and stable upgrade, a fact.

2.4. Internet of Things (IoT)

The term "Internet of Things," or "IoT," represents the billions of physical devices currently linked to the Internet. All gathering and distributing data. With the introduction of incredibly affordable computer chips and the widespread adoption of wireless networks, anything can become a component of the Internet of Things, from small objects to massive machinery [33].

Connecting all these various objects and equipping them with sensors gives the devices a level of digital intelligence that enables them to exchange data in real time

without the need for human interaction [9]. The world around us is becoming smarter and more responsive thanks to the Internet of Things, which combines the digital and physical domains.

What sort of thing falls under the Internet of Things?

Any physical object that can be connected to the Internet for data communication or control can be transformed into an IoT device.

IoT devices include anything that can be controlled by a smartphone app, such as a linked streetlight, smart thermostat at your office, or motion sensor. A self-driving truck can be as deadly as an IoT gadget, or it might be as delicate as a child's toy [9].

Many smaller Internet of Things components could be placed inside some of the same huge things. Such as a jet engine, which is now equipped with thousands of sensors that gather data and transmit it back to ensure that it is operating well. To better understand and manage the environment, smart city projects are covering every region with sensors [5].

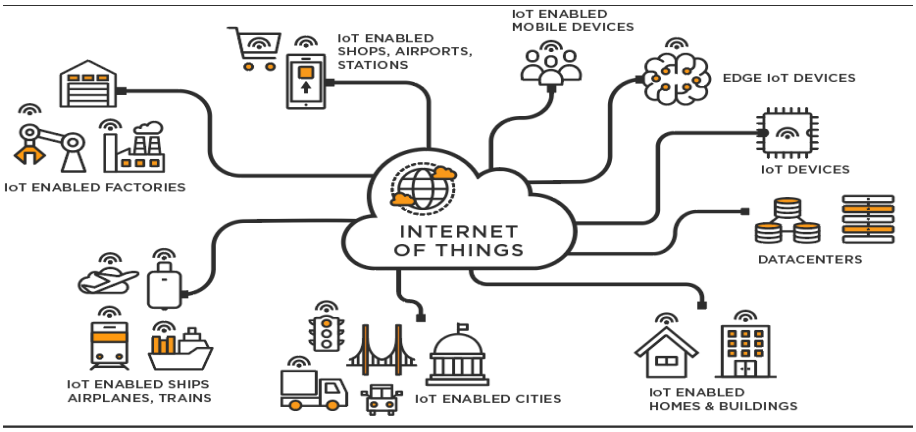

Figure 2.1 Internet of Things Applications

Internet-of-Things Classifications and Characteristics

The key characteristics that will define the Internet of Things have evolved along with its span over time. The Internet-of-Things was initially described in general terms by what were known as the five "C"s:

- Convergence – any ‘thing’, any device
- Computation – anytime, always on
- Collection - any data, any service
- Communication - any path, any network

- Connectivity - any place, anywhere

Later, these generic traits expanded to incorporate specifics that specifically reflected the logical operations of IoT [4]:

- Data distributed (storage and portability)
- Scalability (infrastructure)
- Abstraction (rapid prototyping)
- Accessibility (networks)
- Error tolerance (user-friendliness)
- Event-based (modular architecture)
- Entity-based concept (physical and virtual objects)
- Communication (machine and users)
- Distributed execution (design and processing)
- Operates instantly (speed and performance)

What innovations have enabled the Internet of Things?

Although the concept of the Internet of Things has been around for a while, it has only just become a reality thanks to a variety of recent technological advancements [7].

- Access to low-cost, low-power sensor technology.

More manufacturers may now use IoT technology thanks to reasonably priced and trustworthy sensors.

- Connectivity.

It is now simple to link sensors to the cloud and other "things" for effective data transfer thanks to a variety of network protocols for the internet.

- Cloud computing infrastructures

Businesses and consumers may now get the infrastructure they need to scale up without having to manage it all thanks to the expansion of cloud platforms.

- Analytics using machine learning.

Businesses may acquire insights more quickly and easily thanks to improvements in machine learning and analytics, as well as access to diverse and enormous volumes of data stored in the cloud. The development of these complementary technologies pushes the limits of IoT, and the data generated by IoT feeds these complementary technologies.

- Artificial intelligence with speech (AI).

Natural language processing (NLP) is now available on Internet of Things (IoT) devices, including digital personal assistants like Alexa, Cortana, and Siri. This has made IoT devices more appealing, practical, and inexpensive for usage at home.

The significance of embedded systems in IoT

Environment's supporting structures, which include embedded systems, are essential. The importance of an embedded system in the IoT is that many aspects of the IoT as we currently know them would not be possible without the embedded systems that support device functionality.

Even while the internet is necessary for data transmission to and from IoT devices to online (cloud) services, embedded systems are what will allow this data to be sent and frequently analyzed locally [6].

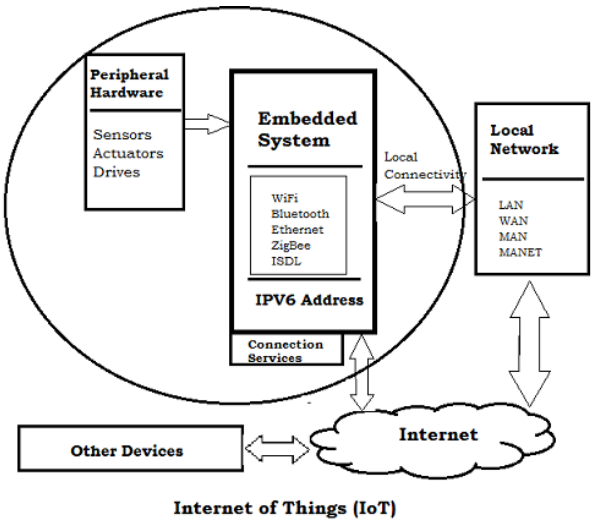

Figure 2.2 IoT systems

The data that embedded systems interpret can come from online services, nearby PCs, and smartphone applications. The multitude of sensors that continuously input real-world data, however, remain the most important source of information [8]. The decision to communicate this data over whatever connectivity an IoT device has will thereafter be made by embedded systems.

Automotive Industry

The deployment of IoT applications has the potential to significantly benefit the automotive industry. Sensors can detect impending equipment failure in vehicles that are already on the road and can warn the driver with details and advice, in addition to the advantages of applying IoT to production processes. IoT-based apps' pooled data has allowed automotive suppliers and manufacturers to understand more about how to keep vehicles running and car owners informed.

2.5. Boot Loaders

The embedded boot loader is a piece of software that takes the system from a running state to a usable state. It is the first part of the firmware to be executed once the embedded system boots/resets. The primary goal of the Boot loader is to initialize the embedded system and provide control for the application/RTOS.

The boot loader performs various hardware checks, configures the processor and peripherals, and performs other tasks such as partitioning or configuring registers. Besides starting the system, the boot loader is also used to update the MCU firmware later. Boot loaders must be able to communicate with the outside world in order to get updates, so most boot loaders can communicate with some form of interface. Another goal of boot loader is also to support data loading feature.

Embedded systems are designed to forget everything when shutdown (except for data already stored in their non-volatile memory such as flash or EEPROM.) The moment the power is used up, the microprocessor inside the SoC does not remember any of its neighbors. In a typical microcontroller, the microprocessor's neighbors include SRAM, DRAM, flash, EEPROM, timers, GPIO controllers, and serial controllers for serial communication protocols such as USB, SPI, Ethernet, UART, and I2C. These neighbors are usually referred to as on-chip peripherals.

The only thing the embedded microprocessor remembers is the instructions that must be executed once it is turned on. This code is usually a single instruction, usually called a reset code. Usually, this code does nothing but move the execution to another piece of code called the device initialization code. The hardware initialization code is where the magic happens.

1. Hardware initialization

The name of this piece of code is the hardware initialization code and it does all the hard work to get the system ready for the next stage of booting. It wakes up the

processor, which in turn wakes up some other peripheral that the microprocessor needs to get into the next stage.

A critical point to note here is that only the peripherals needed for the next stage of the boot process are configured. These are also known as critical peripherals.

2. Choose the mode

The next decision is the choice of mode. Once the individual components and critical peripherals of the microcontroller. Start working together and give us the "embedded system", the second stage of the boot loader begins. In this second stage of the boot process, a decision must be made by the user of the system. Either you go to

1. Application mode
2. Boot loader mode

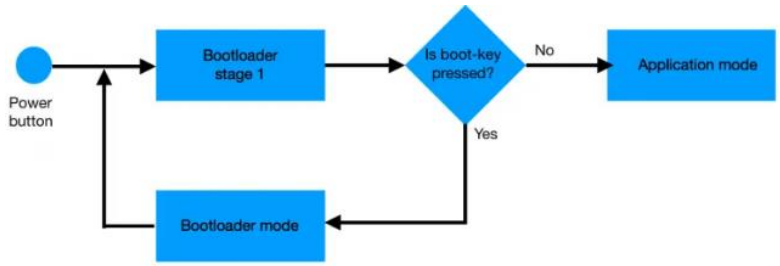

Figure 2.3 Bootloader operation flowchart

3. Startup Code

After the hardware initialization phase, the system is still not ready to execute the application program. On embedded systems, the microprocessor executes the startup code to make the system ready. The main task of this program is to prepare the execution environment for the application written in high-level languages like C, C++, and Rust.

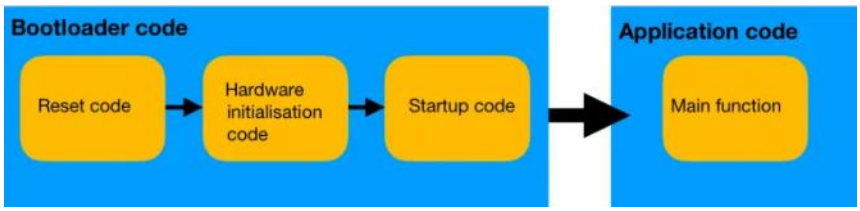

Figure 2.4 Booting sequence

A generic startup code has the following functions.

- It allocates space for and copies all the global variables in the code into the RAM.
- Stack and Stack pointer is initialized.
- Heap is initialized.
- Call the main function in your program.

2.6. Graphical User Interface (GUI)

When operating machines needed an encyclopedic knowledge of code and the inner workings of a program, they had to become more user-friendly for the computers to take off.

The graphical user interface (GUI) has heralded a new age for user interaction with computers and devices. This breakthrough gave people a different way to communicate with systems, so they didn't need to know any coding concepts. This destroyed the entry point into a previously steep learning curve.

A GUI (the graphical user interface) is a computer software framework with interactive visual elements. An interface shows objects conveying knowledge and representing actions that the user may take. When the user communicates with them, the objects change color, size, or visibility.

GUI objects include keys, icons, and cursors. Occasionally, these visual elements are enhanced with sounds or visual effects such as transparency or falling shadows.

The big advantage of a GUI is that systems that use one are accessible to people of all knowledge levels, from a beginner to an advanced developer or other tech-savvy individual. We make it easy for anyone to open menus, transfer files, launch programs, or check the internet without having to tell the machine to perform a task through the command line.

Instant feedback is also provided by GUIs. For example, if you click an icon, it will open, and that can be seen in real time. Using a command line GUI, once you hit return, you won't know if it's a legitimate entry; if it isn't correct, nothing will happen.

GUI Layers

From the user side he deals with the graphical interface which appears in front of him on a hardware display where icons, gestures, and tools which he can choose from all appears on the display server as windows manager in (windows interfaces) which is

linked to the operating system (kernel) such Linux then the hardware which is the whole container which the user use it for controlling.

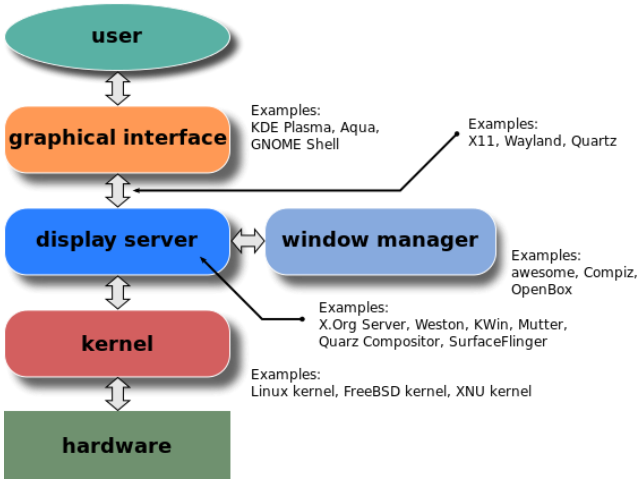

Figure 2.5 GUI layers

GUI in automotive

Figure 2.6 GUI in cars

Tier 1 and OEM manufacturers understand the need to offer groundbreaking digital instrument clusters, driver information screens, and in-vehicle infotainment systems to keep up with rapidly changing market demands. Storyboard is the only platform that brings together designers and developers, treating them as equals without sacrificing resources or performance to design rich, intuitive in-vehicle experiences.

Automotive human machine interface (HMI) solutions enable drivers to interact with touchpads, multi-touch dashboards, built-in screens, control panels, pushbuttons, and traditional keypads. Since a car is an entire ecosystem of interconnected components, high-quality HMI software is crucial to the automotive industry.

The automotive HMI program renders cars of the next generation versatile and customized using the new technologies. Modern consumers are demanding a seamless

experience and HMI solutions can meet the demand for smooth car interactions. Human machine interfaces, enabled by smart systems and embedded sensors, ensure vehicles react to driver's intent and preferences.

The User Experience is all about on-screen GUI (Graphic User Interfaces) in cars. In the latest car models the on-screen user experience could be divided into these categories:

- Entertainment and communication: Music, television, email, internet, third-party applications UI-related. Such features are often accessed through the monitor in the center console.
- The tachometer dashboard UI, alarm, driving assist and other driving-related functions.
- Anything else, such as climate control, general settings, and other vehicle settings.

Figure 2.7 GUI applications

Different programming languages are used in developing desktop applications and graphical user interfaces for multi-tasking applications. C++ and Python are mainly used in those kinds due to the ease of interaction with those languages when dealing with heavy-duty applications, which need you as a developer to be able to deal with different kinds of complexities.

So, with these kinds of differences, companies have challenged each other around the efficiency of a GUI and the comfort of the user who is handling it.

Apple and Google are both allies and competitors of car companies in this race, and automakers are being increasingly forced to avoid providing their services because customers already have a

long history with the ecologies of those firms. Car companies have recognized that they can deliver excellent digital user experiences by investing in digital design and maximizing the added benefit of deep integration with key driving experience elements.

Python in GUI

Python is used in the design due to its simplicity where you can handle multi modules at the same time using 3 main modules which make it more comfortable for the developer to use it in GUI applications. Most of the python modules used in GUI are built using c/c ++ but using it in python makes it easier to handle.

PyQt5 vs TKINTER

PyQt is being chosen over Tkinter due to these reasons.

1) **Stability**: PyQt is used in many large-scale applications and has stood the test of time.
2) **PyQt has a straightforward API** with its classes corresponding to Qt C++'s, and as such, the API documentation for C++ works for Python — the namespaces, properties, methods are all the same. If you have experience working with Qt and/or C++, you will find PyQt easy to work with.
3) **The signal/slot mechanism allows for code flexibility**: GUI programming with Qt is designed around the notion of signals and slots for object communication. When an event occurs (e.g., a button is pressed), a signal is emitted, and slots are callable functions which handle the event (e.g., when a button is clicked, display a pop-up). This provides versatility when dealing with Interface events and results in a cleaner codebase.
4) **Many learning resources are available**: PyQt is one of the most popular UI frameworks for Python. It has an active community with many third-party code examples and tutorials available [16].

PyQt5

PyQt5 has 3 main modules.

- QtCore Module contains the core non-GUI classes, including the event loop and Qt's signal and slot mechanism. It also includes platform independent abstractions for Unicode, threads, mapped files, shared memory, regular expressions, and user and application settings.

- The **QEventLoop** class provides a means of entering and leaving an event loop.
- **Signals and slots** are used for communication between objects. The signals and slots mechanism are a central feature of Qt and probably the part that differs most from the features provided by other frameworks. In GUI programming, when we change one widget, we often want another widget to be notified. More generally, we want objects of any kind to be able to communicate with one another.

For example, if a user clicks the Close button, we probably want the window's close () function to be called.

Other toolkits achieve this kind of communication using callbacks. A callback is a pointer to a function, so if you want a processing function to notify you about some event you pass a pointer to another function (the callback) to the processing function. The processing function then calls the callback when appropriate. While successful frameworks using this method do exist, callbacks can be unintuitive and may suffer from problems in ensuring the type-correctness of callback arguments.

Signals are emitted by objects when they change their state in a way that may be interesting to other objects. This is all the object does to communicate. It does not know or care whether anything is receiving the signals it emits. This is true information encapsulation and ensures that the object can be used as a software component.

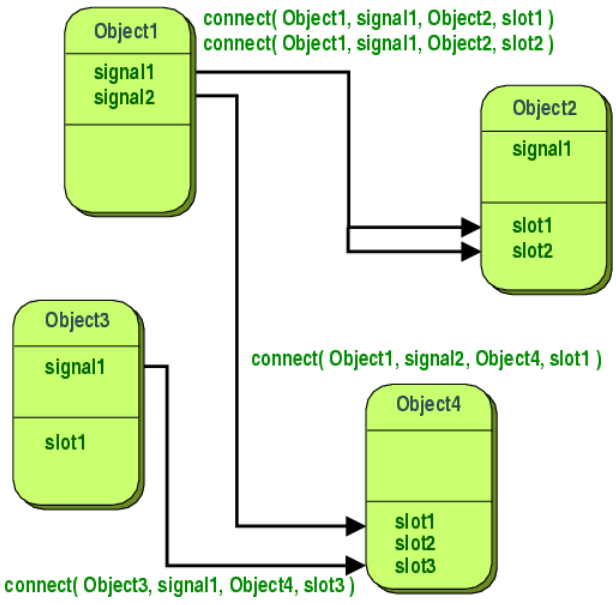

Figure 2.8 Signals and slots

Slots can be used for receiving signals, but they are also normal member functions. Just as an object does not know if anything receives its signals, a slot does not know if it has any signals connected to it. This ensures that truly independent components can be created with Qt. You can connect as many signals to a single slot as you want, and each signal can be connected to as many slots as you need. It is even possible to connect a signal directly to another signal. (This will emit the second signal immediately after the first. Together, signals and slots make up a powerful component programming mechanism. [17]

- QtGui Module contains most of the GUI classes. These include several tables, trees and list classes based on the model–view–controller design pattern. Also provided is a sophisticated 2D canvas widget capable of storing thousands of items including ordinary widgets. [18]
- QtWidgets Module provides a set of UI elements to create classic desktop-style user interfaces, for example:

- **Widgets (QWidget)** which are the primary elements for creating user interfaces in Qt.

Figure 2.9 QWidget

- **Layouts (QLayout)** are an elegant and flexible way to automatically arrange child widgets within their container [18]

Figure 2.10 QLayOut

2.7. CAN Bus

What is a CAN Bus?

CAN stands for Controller Area Network.

The CAN Bus is a serial two-wire half-duplex communication protocol that allows multiple nodes in a system to communicate efficiently with each other. Each node can send and receive messages; not all nodes can communicate at once. All nodes receive all broadcast data from the bus, then each node decides whether that data is relevant according to the filter masks of each node, as shown in the following figure.

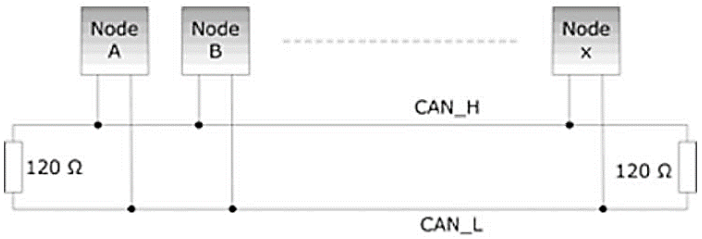

Figure 2.11 CAN network

If any node wants to send a message, it puts it on the bus with a specific ID and then each node from the other nodes will check whether this ID is included in its filters or not. If yes, it reads the message from the bus, otherwise it ignores it.

The CAN protocol is already widely used in vehicle and vessel internal components. In recent years, it has seen adoption in data communications and control in industrial fields.[11]

Why did we choose CAN?

• Variation in speed

- Low Speed:

  The CAN network offers low signal transfer rates that range between 40 kbps and 125 kbps.

- High Speed

CAN offers high signal transfer rates of between 40 kbps and 1 Mbps. High-speed CAN networks are terminated at both ends of the bus line with a 120-ohm resistor between the CAN high and CAN low lines.

The lower signaling rates allow communication to continue on the bus, even when a wiring failure takes place, while high-speed doesn't.

• **Low price**

When the CAN protocol was created, its aim and primary goal was to enable faster communication between modules and electronic devices in vehicles while simultaneously decreasing the amount of wiring (and the amount of copper) necessary.

• **Built-in Error Detection**

Error handling is built in the CAN protocol, each node can check for its errors during transmission and maintain its own error counter. When errors are detected, nodes transmit a special Error Flag message and will destroy the offending bus traffic to prevent the error from spreading through the whole network. Even the node which is generating the fault will detect its own error in transmission, and raise its error counter, this eventually led the device to "bus off" and cease participating in the CAN network traffic. In this way, CAN nodes can both detect errors and prevent faulty devices from creating any useless bus traffic.

• **Flexibility**

To understand the flexibility of the CAN bus protocol in communications, we need to know the difference between message-based protocols and address-based ones. In an address-based communication protocol, nodes communicate directly with each other by configuring themselves at the same address.

The CAN bus protocol is a message-based communication protocol. In this type of protocol, nodes on the bus have no address to identify them. As a result, nodes can easily be added or removed without performing any software or hardware updates on the system.

This feature makes it easy to integrate new electronic devices into the CAN bus network without significant programming overhead and supports a modular system that is easily modified to suit your specs or requirements.[12]

CAN Specification

- Wired
- Serial
- Asynchronous
- Multi-Master No Slave (MMNS)
- Half duplex [13]

CAN Transmission Handling

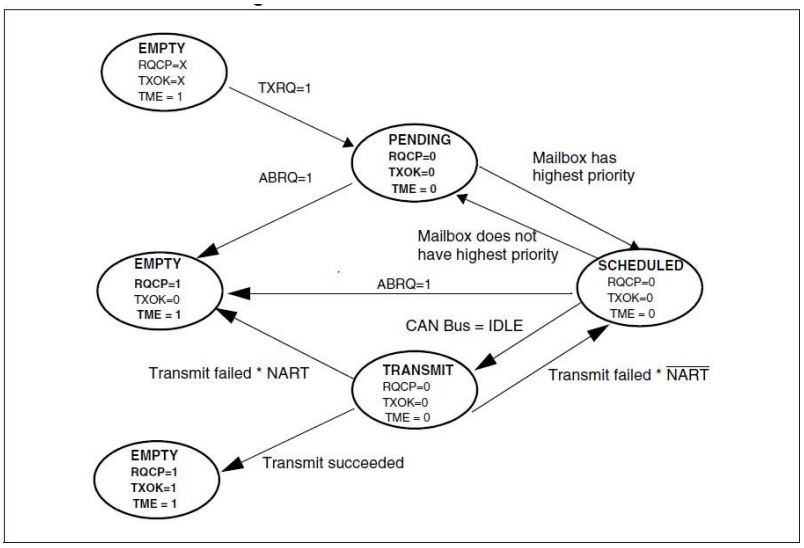

Figure 2.12 CAN transmission handling

1. In order to transmit a message, the application must select one empty transmit mailbox, setup the identifier, the data length code (DLC), and the data before requesting the transmission by setting the corresponding TXRQ bit (Transmit mailbox request) in the CAN_TIxR.
2. When the mailbox is left in an empty state and after the TXRQ bit has been set, the mailbox enters a pending state and waits to become the highest priority mailbox (in our case, the highest priority is defined by the transmit request order). As soon as the mailbox has the highest priority, it will be scheduled for transmission.
3. The transmission of the message will start (enter transmit state) when the CAN bus becomes idle, and once the mailbox message has been successfully transmitted by setting the RQCP (Request Complete) and TXOK (Transmission Ok) bits in the CAN_TSR register, the mailbox becomes empty, and we can use it again. [14]

CAN Reception Handling

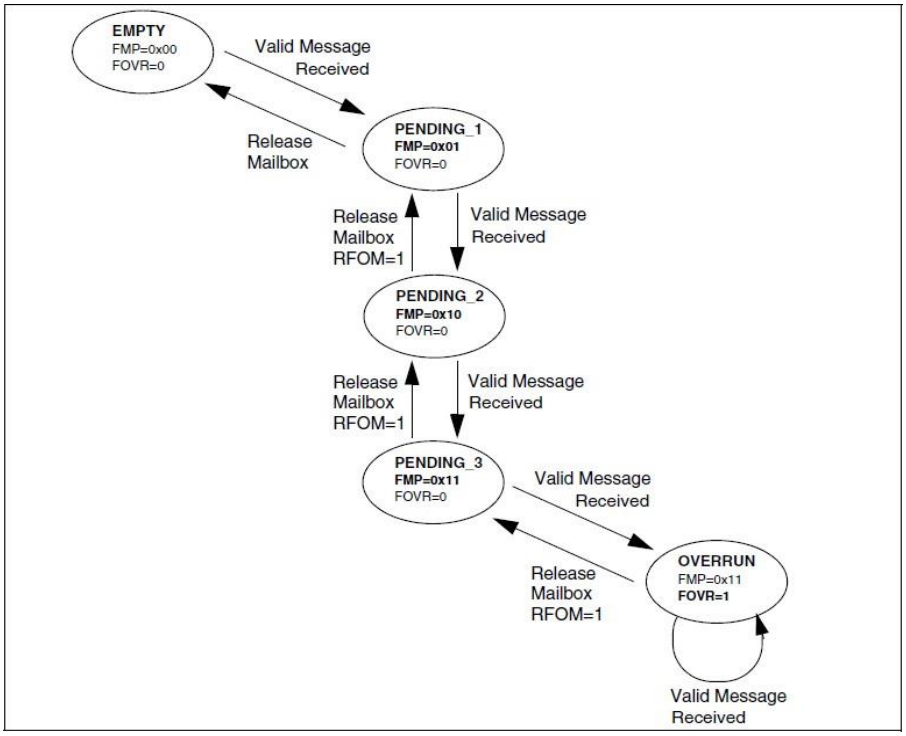

Figure 2.13 Receive FIFO states

We can receive up to three messages in the FIFO mailbox. Then, if we start from the empty state, the first valid message received is stored in the FIFO, which becomes pending_1. If we receive the next valid message, it will be stored in the FIFO and enter pending_2. We must release the FIFO mailbox to receive another message. [14]

CAN Filters

We have 14 configurable and scalable filter banks (13-0) for the application in order to receive only the messages the software needs, and each filter bank consists of two 32-bit registers.

Masking Mode:

Filter masks are used to know which bits in the message identifier (ID) should be compared with the filter bits, as follows:

The mask bit may be set to one or zero. If it is set to one, the corresponding identifier bit will be compared with the filter bit. If they are matched, the messages will be accepted, otherwise they will be rejected. When a mask bit is set to zero, regardless of the value of the filter bit, the corresponding identifier bit is automatically accepted.

CAN Frames

Frame types:

- **Error Frame** – Reports error condition

Any node in the CAN network sender or receiver may signal an error condition at any time during a data or remote frame transmission.

Figure 2.14 Error frame

- **Overload Frame** – Reports node overload

A node sends an overload frame to request a delay between two remote or data frames, so the overload frame can only occur between data or remote frame transmissions.

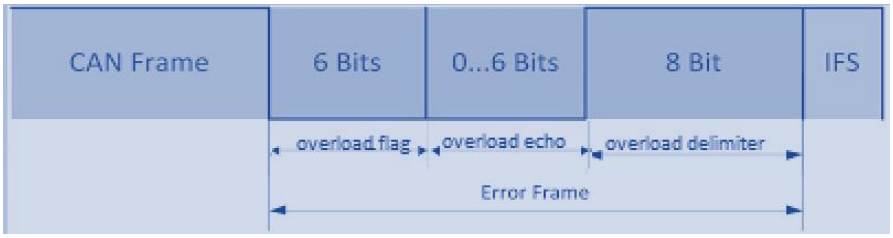

Figure 2.15 Overload Frame

- **Data Frame** – Sends data

Data transfer from node to the CAN bus and the nodes that are interested in the message can read it (This is done according to the filters of the node).

- **Remote Frame** – Requests data

Any node may request data from other nodes. The remote frame represents the request so its consequently followed by a data frame containing the requested data. Both data and remote frame have the same frame but in remote frame data = 0

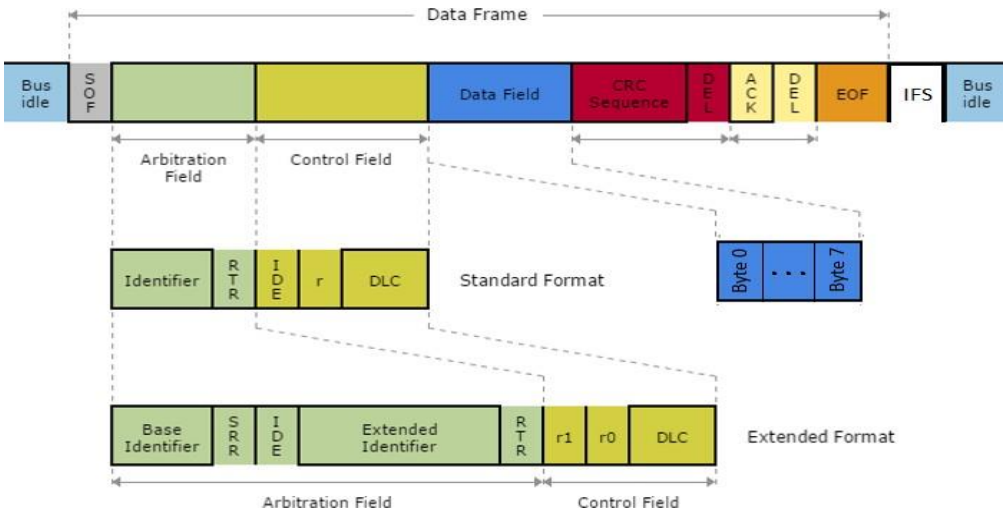

Figure 2.16 Data and Remote frames

- SOF (Start of Frame) (1 bit) – Marks the beginning of data and remote Frames Arbitration
- Field – Includes the message ID (11bits in standard ID and 29 bits in extended ID) and RTR (Remote Transmission Request) bit, which distinguishes data and remote-Control Field.
- DLC to determine data size (4 bits) and IDE (1 bit) to determine message ID
- Data Field – The actual data (which is zero in remote frame and contains the actual data frame)
- CRC Field (16 bits) – Checksum, CRC stands for Cyclic Redundancy Check, it contains Checksum and delimiter.
- ACK Field (2 bits) – Acknowledge and EOF (End of Frame) (7 bits)– Marks the end of data and remote frames
- IFS Field (3 bits) – Inter-frame space

ACK delimiter: The acknowledgement is sent from the receiving node to the transmitting node, and it needs some time, so ACK delimiter is used. CRC delimiter: The ECU needs some time to calculate the CRC, so a delimiter bit is introduced to make some delay for the ECU.[15]

Chapter 3 : Previous Related Work

Flashing Over the Air (FOTA)

Firmware updates over the air (FOTA) as a service is an upcoming trend for automobile manufacturers [67]. Some systems, such as mobile networks, support over-the-air (OTA) firmware updates [66].

The vehicle must be brought back to the dealership if an OEM finds a bug in the firmware that manages a crucial feature like the brakes or an airbag. Deeper within the vehicle network architecture, Electronic Control Units (ECUs) that regulate functions like the engine, brakes, and steering are often Microcontrollers (MCUs) with minimal amounts of embedded flash and RAM. This introduces restrictions that call for a different update strategy [69].

Vehicle owners won't put up with their vehicles being unavailable while upgrades are being made, unlike users of mobile devices or computers. Therefore, it is excellent for upgrades that are necessary for proper vehicle functioning to happen seamlessly and discreetly in the background [69].

This approach has several advantages. It causes minimal customer inconvenience because the customer is not required to bring the vehicle to a maintenance centre for a firmware update installation. It also allows for faster updates; as soon as the firmware is released publicly, the customer can download and install the new firmware image in the vehicle. The firmware image is downloaded to the vehicle via a wireless network connection from a trusted portal and then flashed to the correct ECU [67].

Because firmware controls a vehicle's functionality, security is critical [67], and to ensure that the firmware update process is successful in the face of potential threats. As a result, mobile devices may be useful in assisting the OTA firmware update process for other devices, such as those used in automotive applications [66].

The first paper proposed a secure OTA firmware update (FOTA) protocol to offer flexibility to the firmware update process, while meeting the required security requirements. The protocol was formally analysed using Scyther and CasperFDR and no known attack was found [66].

In the automotive industry, security plays an important role to ensure the safety and reliability of the car. If any of the ECUs are attacked, the safety of the car and passengers may be at risk [66]. The problem faced was that firmware updates of automotive subsystems are security critical, but there is yet to be a secure solution that

can be accessible to all the involved entities. A secure solution with flexibility in the process is important to ensure its acceptance, especially by the users [66].

The contributions of this paper were as follows:

1. It proposed a secure OTA automotive firmware update protocol with the use of a mobile device. The mobile application to conduct the firmware update ensures the authentication of the parties involved and the confidentiality of the firmware [66].
2. It provided analysis of the OTA firmware update protocol using Scyther and CasperFDR [66].

The second paper is A Framework for Self-Verification of Firmware Updates over the Air in Vehicle ECUs We propose a framework where a firmware binary is downloaded to an ECU from a trusted portal using a secure FOTA protocol. We apply hardware virtualization techniques on the ECU, allowing two guest systems to run simultaneously on the underlying virtual machine monitor. The control system, consisting of L4, performs simple operations such as to provide memory verification of the flashed firmware. The functional system handles all the services or functionality of the corresponding ECU. The binary is then flashed to the ROM of the ECU, which constitutes the functional system. Next, the control system locally verifies the correct flashing on the ECU itself. The self-verification allows the ECU to assure that the proper firmware has been installed [67]. The main contributions of this paper are as follows:

1. They have developed a framework for self-verification of remote firmware updates in ECUs [67].
2. They have applied virtualization techniques to support a functional and a control system. The control system verifies the flashing of the functional system [67].
3. They have designed an efficient protocol for memory verification of the flashed firmware. The verification is performed locally on the embedded device [67].
4. The protocol has been analysed in terms of security and cost overhead [67].

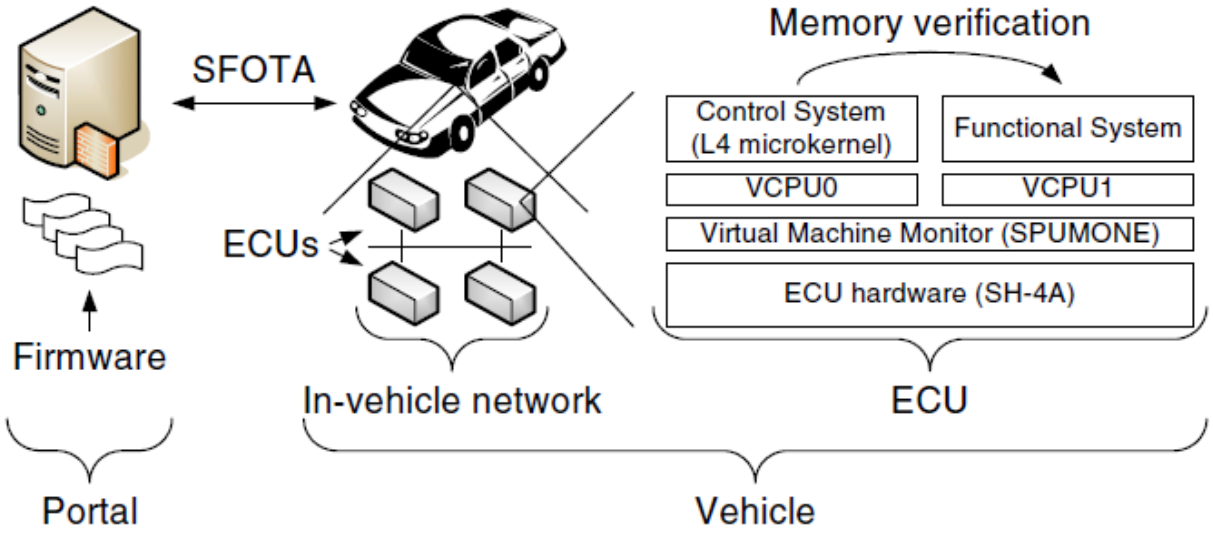

Figure 3.1 framework for self-verification of firmware updates over the air in vehicle ECUs

The third paper was An Over the Air Update Mechanism for ESP8266 Microcontrollers. It presented a concept for building and publishing cryptographically secure Over the Air updates for embedded devices based on ESP8266 microcontrollers. A proof-of-concept implementation has been developed, which is now an essential part of the home-automation development and deployment in the Magrathea Laboratories e.V. hackerspace. All the devices running the OTA-enabled firmware have undergone multiple major updates without any problems. This includes a major network configuration change and an important stability fix for the network communication stack. All devices applied the update successfully and started to work without any manual interaction required afterwards. While the devices from various manufacturers in the hackerspace are all delivered with a pre-installed firmware, which is thought to be ready for smart home application, none of them has been provided with updates by the manufacturer so far. It is not visible to the users if the current firmware of these devices is at the latest version nor which versions are installed or how to update them [68].

The update infrastructure has been the crucial point for most of their members towards the framework. Enabling the developers to do updates in combination with the shared configuration and behaviour provided by the framework resulted in a massive speedup when it comes to project deployment. Before that, the cost for applying changes after deployment was estimated so high, that most projects tend to delay deployment until all required and wanted features were implemented. Now, as the devices are deployed as soon as the hardware is considered stable, these devices start to provide

functionality early and therefore the developers can get better feedback on the provided functionality [68].

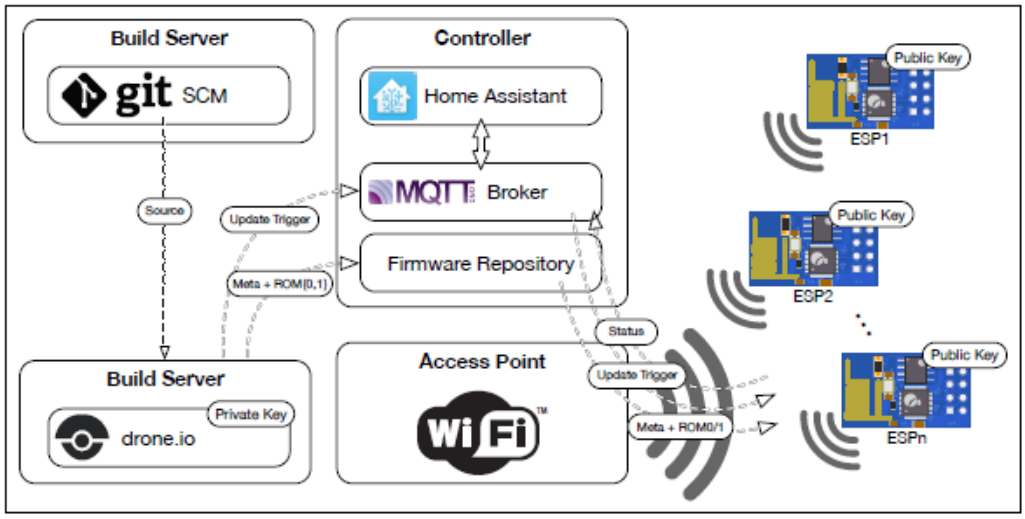

Figure 3.2 The base network topology

The fourth paper was conducted by BU Automotive, NXP Semiconductors to achieve Making-Full-Vehicle-OTA-Updates-Reality. The diagram below shows the key components which take the update file from the OEM's servers to the specific ECU within a vehicle. A secure connection is set up over a cellular network between an individual vehicle and the server. This allows the new, updated, firmware to be sent securely to the vehicle's Telematics Unit, and then on to the OTA Manager. The OTA Manager manages the update process for all ECUs within the vehicle. It controls the distribution of firmware updates to ECUs and will tell the ECU when to perform the update. This is important in the case when multiple ECUs need to be updated simultaneously – e.g., to add a new feature to the vehicle which involves multiple ECUs. Once the update process is complete, the OTA manager will send confirmation to the OEM.

An external NAND flash can be fitted to the ECU which runs the OTA Manager to allow for the firmware update to be stored until they are required. The external flash can also be used to store backup copies of the firmware for other vehicle ECUs which can be called upon in the case of a major fault in an ECU update, which leaves an ECU without any working firmware. These backup copies would be secured via encryption and authentication protection to prevent any tampering of the firmware whilst stored in the external memory module.

The OTA Manager contains a table of every ECU within the vehicle including information such as serial numbers and current firmware version. This allows the OTA

Manager to verify firmware updates which arrive and ensure that they are authorised for use in this vehicle. If the ECU being updated does not have security functionality, then the OTA manager would also be responsible for decrypting and authenticating the incoming update.0

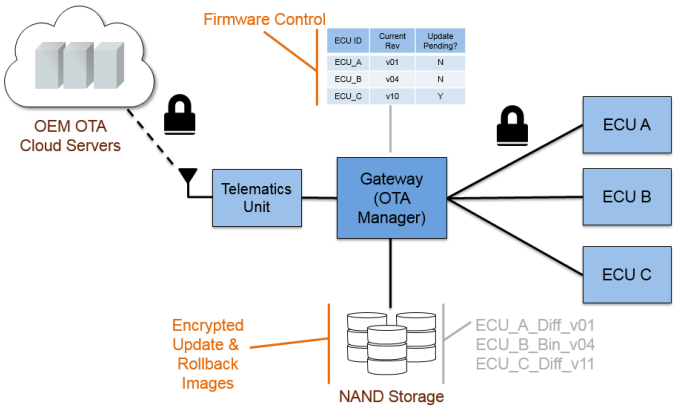

Figure 3.3 OTA security

The fifth paper discussed Automotive OTA, the potential, and the challenge that worked on a principle that in order to achieve a successful FOTA update, an update manager should handle the FOTA operations, distributes the updates using the most appropriate methods for each package as specified [70].

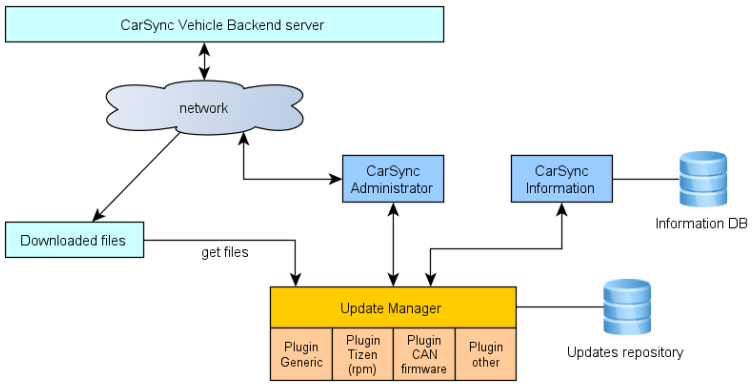

Figure 3.4 OTA Update Process and Architecture

Update Manager has no knowledge of the required modifications to the data or resources used by respective packages, neither of the actual operations that need to be executed to ensure the proper level of update integrity. Update Manager provides means of logging, so that each step is logged, allowing for the update process to be traced, or even span across device reboots [70].

Strandberg, Kim, Dennis Kengo Oka and Tomas Olovsson have contributed with a comprehensive and novel unified software update framework named UniSUF, well

aligned with industry stakeholders. UniSUF is made to accommodate various scenarios for the automotive domain by encapsulating needed data into one single file, a Vehicle Unique Update Package (VUUP). This vehicle unique file can be processed within a vehicle ECU, using a workshop computer, at factory production, with a diagnostic update tool, or in other compositions. Moreover, the complete update process can be performed without any external communication dependencies since all files are inherently secured. A continuous secure software update process is a prerequisite for facilitating vehicle resilience towards cyber-attacks in a rapidly changing environment [74].

Predictive maintenance

The implementation of Pd.M. solutions needs to consider key aspects, such as the underlying scenario, the performance key indicators, and the metrics that should be defined to measure them [80].
Besides those, some other aspects are important to build an efficient Pd.M. system, such as forecasting the key indicators, enabling institutions to make decisions according to the upcoming problems [81].
The Pd.M. system requires the identification of the input variables, a target application, and the model that creates a relationship between them. After the development of the model, it is possible to obtain results that will be analysed and validated to make decisions that can help with the evaluation and optimization of the overall system.
This evaluation is very important for the analysis of the system reliability since companies build Pd.M. solutions to reduce costs, and thus, the system should be economical and low risk. It is important to evaluate the amount and quality of the available data.
The existence of large amounts of data may require the application of reducing and normalization methods. Another common issue is the lack of information about the target application, which will affect the relationship between the application and the inputs.
Deep learning has been studied to predict an unknown time series using historical data [41]. A variety of modern machine learning algorithms ranging from multi-layer perceptron's (MLP) and long short-term memory (LSTM) to the combinations of RNNs and dynamic Boltzmann machines [41] have been used in time-series forecasting.
All these algorithms have their own strengths and weaknesses, depending on the type of data or the task at hand. However, due to the complex nature of predictive

maintenance use cases, none of the above noted algorithms have provided excellent results. Moreover, hybrid methods [42] are a new development trend in the field of time-series forecasting.

The authors of [43] introduced long short-term memory (LSTM), a special kind of RNN, capable of dealing with complex data and learning long-term dependencies. Over time, these networks were further optimized in other work [44,45].

LSTMs have demonstrated their success in time-series forecasting [46]. Recently in [47], a CNN combined with K-means, which was proposed for time-series load forecasting, achieved impressive results.

The main reason for this is the capability of CNN to extract features from input data at different levels. Because CNN can learn from a vast range of non-linear problems as explained in [48], it is well suited for time-series production data.

In contrast, LSTMs are effective in modelling time-series data because of their ability to remember the states of input data in their memory [46].

Abbasi et al. [49] introduced DL algorithm based on RNN using long short-term memory (LSTM) to design a predictive maintenance of air booster compressor motor.

Rahhal et al. [50] presented a system for Pd.M. based on IoT and deep learning to predict the failure time of equipment. The Pd.M. model is built using LSTM and RNN deep neural networks and used to predict the RUL for light bulbs with a minimum error rate of 0.79%.

Bamboula et al. [51] utilized LSTM autoencoders deep learning method to build a model for planning maintenance in Hyperphysical Production Systems. The autoencoder deplaning model is used to classify real-world machine status based on the machine's sensor data. The model is evaluated using data collected from the steel industry production process. The model obtained an average accuracy of 94.2%

Deep learning method with LSTM combined with support vector machine (SVM) proposed in "Vibration-based anomaly detection using LSTM/SVM approaches" paper[52] are utilized to distinguish aberrant data from normal vibration signals gathered during a reduction gearbox endurance test and helicopter test flight data acquired by many sensors located throughout the aircraft

Deep learning and LSTMs offer to train hierarchical models over input data that describe probability distributions. Artificial intelligence has become a vibrant subject with many practical applications such as machine prognosis, fault detection, predictive maintenance, etc., thanks to recent advancements in both hardware and brain models, particularly in the previous decade.

Chapter 4 : System Requirements and Specification

4.1. System Requirements

4.1.1. Enumerated Functional Requirements

Requirement	Description
1. The system should provide the ability to observe the engine condition.	Multiple sensors shall be implemented on the engine to gather various data, such as vibration, pressure, voltage, and rotation.
2. Application ECU should gather the data from the engine.	Data should be sent to the ML model, error free.
3. ML model should pre-process Data.	We shall handle missing data and filter the noisy data.
4. Model will classify the type of error.	Accordingly, the history of failures of the machine, the model will classify the type of failure.
5. The ML model should be able to predict the failures in the engine.	The prediction must be more than 94% accurate.
6. ECU shall report to the driver the expected failure.	ECU shall send the error to the Driver less than one second from the failure prediction.
7. FOTA Master ECU Detects the new update.	FOTA Master must check for the new updates every 24 hours.
8. The system should allow the user to accept the new update.	The user should be able to accept or refuse the new firmware through the UI system.
9. The system should be able to request and download the firmware from the OEM Server.	The Firmware must be downloaded with respect to the firmware integrity.
10. The system must be able to install new firmware.	The bootloader must be able to complete the firmware installation process successfully.
11. Capabilities of FOTA ECU for rollback.	FOTA Target ECUs shall be capable to internally recovering the SW image, being active before last (FOTA) activation. This may happen on a trigger by the FOTA Master instance.
12. Activation of new software in vehicle’s safe state only	FOTA Target ECU shall accept activation of new software in vehicle safe state only.

4.1.2. Enumerated Non-Functional Requirements

Requirement	Description
1. Performance	System must increase equipment uptime by 10-20% and reduce recalls due to software glitches by 100%.
2. Accuracy	The ML model must be more than 94% accurate, FOTA System must be 100% accurate.
3. Safety	No image can be downloaded until it has been verified as it's authentic.

4.2. Functional Requirements Specification

- **System stakeholders**
  i. Developer
  ii. OEM
  iii. User
- **Actors and Goals**
  i. Developer (Participating)
    - Develop and monitoring the system.
  ii. OEM (Participating)
    - Sending the firmware to the vehicle.
  iii. User (Participating)
    - uses the vehicle and accept or reject the update.

4.3. System Design

We will achieve this by using a machine learning model that is applied to the data of the engine that we will extract using sensors. As the model will depend on the performance of the engine and through it. it will be able to predict whether a failure is about to happen or not.

Flow chart diagram of predictive maintenance is shown in PDM Flow Chart

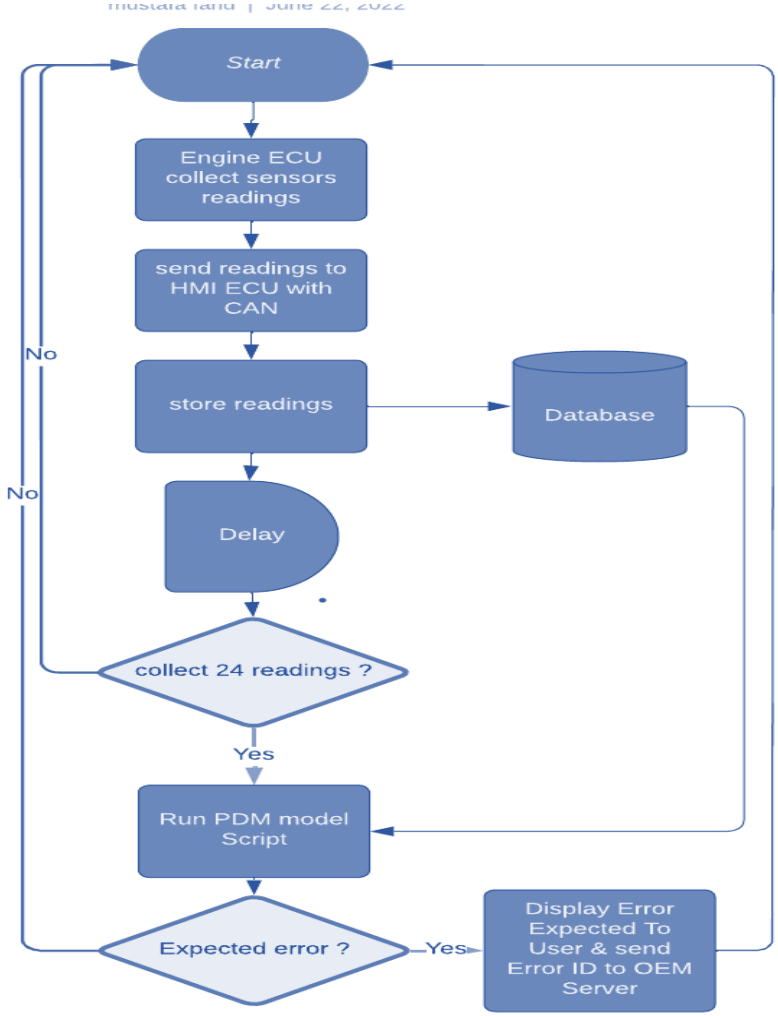

Figure 4.1 PDM Flow Chart

One of the ECUs is the Engine ECU, which is responsible for collecting data from the engine sensors. It will collect data each hour during the day. The data is sent over the CAN network to the Raspberry Pi, on which predictive maintenance operates. The PMS will store and process this data and based on it, will inform the user if there is a failure that is expected to occur through the user interface system.

This will enable us to predict hardware failures before they occur and alert the driver through the user interface long enough to allow them to go to the service center and take the necessary actions.

Firmware Updates

Our project will provide this solution as we will build a FOTA system which will consist of a vehicle subsystem containing multiple electronic controllers connected via CAN bus and IoT device.

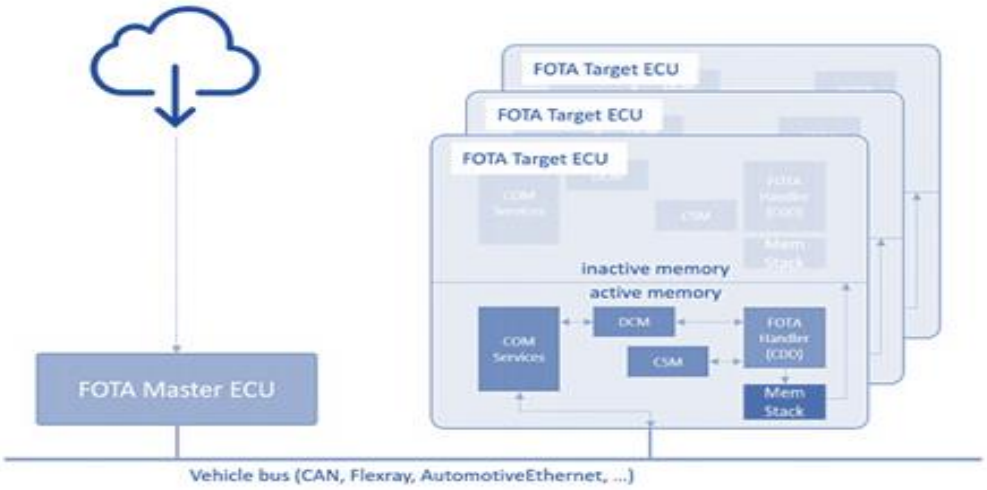

Figure 4.2 Update over the air

One of the ECUs is known as the main ECU, and it is linked to the ESP8266 WIFI module, which will constantly check the OEM server for new versions of updates. If a new update is available on the server, the main ECU will send a notification to the HMI ECU via the CAN bus, and the user will be notified via the GUI. If the user accepts the update, the HMI ECU will send this acceptance to the main ECU, and the main ECU will download the new update. If the car is in a safe state (not moving), the main ECU will send the update to the target ECU, which will flash the update to its flash memory. and will display that the update occurred successfully in the GUI to the user.

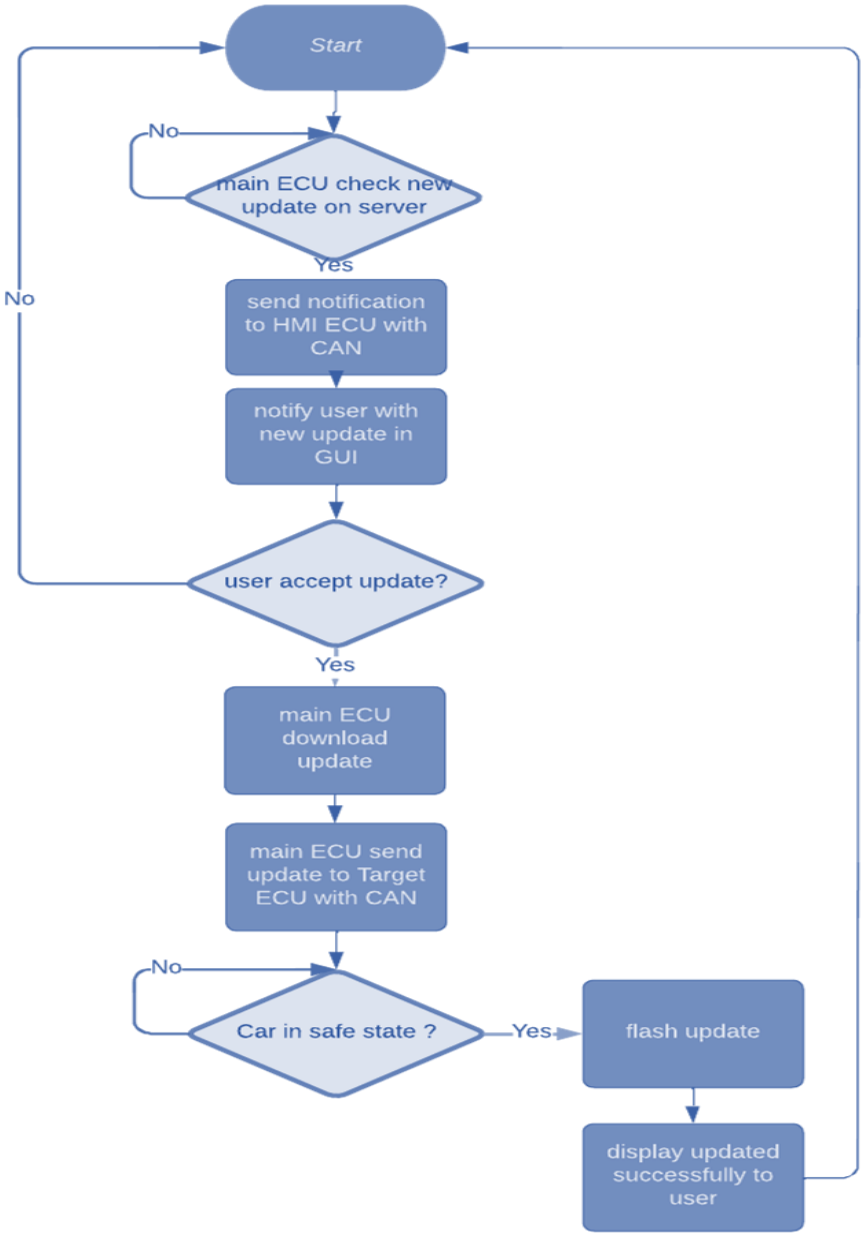

Figure 4.3 update flow chart

Roll back

If the user doesn't like it or there is a bug in the current software version, the user can roll back to the previous software version of the specified ECU by choosing the version of software in the GUI. It will be sent by HMI ECU on a CAN bus to the main ECU. If this chosen version is available on the server, it will be downloaded. The main ECU will send the update to the target ECU, which will flash the update to its flash memory. and will display that the update occurred successfully in the GUI to the user. Roll back flow chart is shown in below.

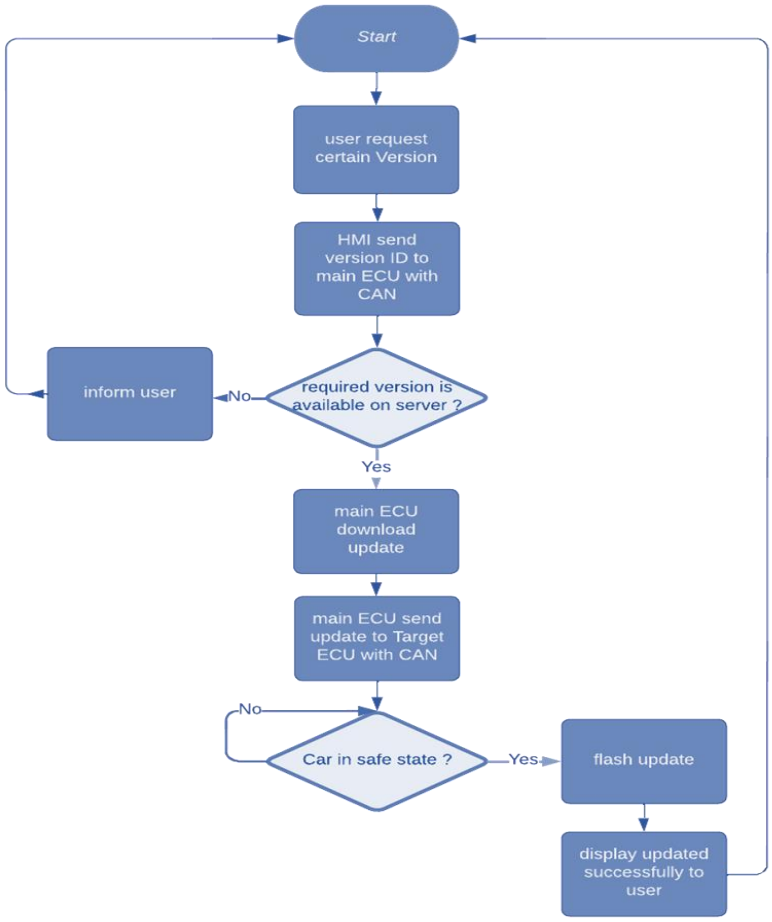

Figure 4.4 Roll back flow chart

4.4. Use Case Diagram

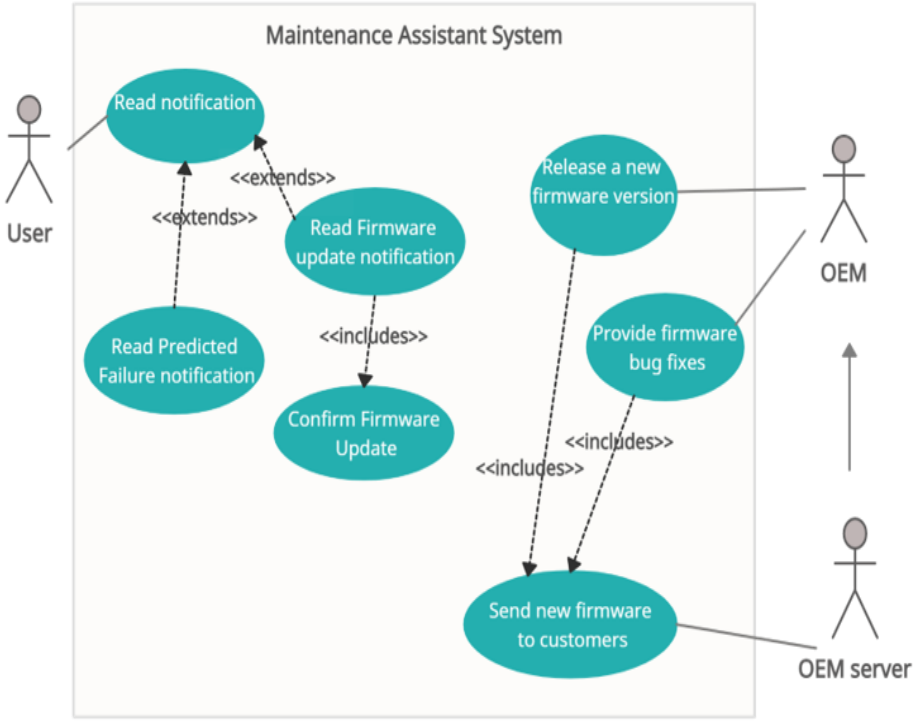

Figure 4.5 Use case Diagram

4.5. Use Cases Description

Use Case ID: 01
Title: Read notification
Actor: User
Stakeholders and their needs: • Users need to read notifications and know if there is any update from OEM • Users need to know if Predictive system find a failure and interaction with messages • Users need to know messages containing information about firmware features • Users need to know messages containing information about the type of failure that predicted
Trigger: Inform user with updates from MAS system
Pre-conditions: • The system must be active on the Driver's Car • The model is ready and takes input from sensors, and sends notifications based on what predicted • OEM server is ready to send notifications based on OEM release updates

Post-conditions:

- Users see notification and interaction with it,
- If it is from “firmware update” confirm the update or delay the confirmation
- If it is from “predict failure” know the type of failure
- Totally ignore the notifications .

Basic flow:

1. The OEM sends firmware updates to the server and the server will notify the vehicles
2. Sever connected with MAS system and send notifications to the user
3. Notifications will pop up to the driver and wait for his response
4. The user can confirm the installation of update if car is stable
5. The user can delay installation until a specific time

Use Case ID: 02

Title: Read Firmware update notification

Actor: User

Stakeholders and their needs:

- User -vehicle owner- needs to know if OEM launched firmware version.
- User -vehicle owner- needs to interact with this update notification.
- User -vehicle owner- needs to know what features of firmware and what the problem that it fixed.
- User -vehicle owner- needs to install a new firmware version that solves some problems.

Trigger: Inform the user that a new firmware has been released and know its features.

Pre-conditions:

- Systems are working and connected with OEM Server.
- OEM server is ready to send notifications based on OEM release updates.
- Cars are in stable condition if the user confirmed installation.

Post-conditions:

- Users read the message and know what firmware will be fixed.
- Users can confirm the firmware update and begin installing.
- Users just ignore the messages but for specific time.

Basic flow:

1. The OEM sends updates to server and notify the vehicles.
2. Sever connected with MAS system and send notifications.
3. Notifications will pop up to the driver and the user can interact with this message.
4. Massage containing information about firmware features.

Use Case ID: 03

Title: Read predicted failure notification

Actor: User

Trigger: Inform the user that a specific type of failure will happen in future.

Pre-conditions:

- Predictive Systems are working and connected Sensors.
- Engine state is ready to send its reading based on Sensors.
  Cars are in stable condition and all sensors read data well.

Post-conditions:

- User read the message and know what type of failure.
- Users can take actions to replace the component that will fail.

Users just ignore the messages but for specific time.

Basic flow:

1. The predictive system takes input feature from engine state
2. The input will be preprocessed to enter the model
3. The predictive system will predict the result and type of failure
4. Notifications will pop up to the driver and the user can interact with this message

Massage containing information about failure type

Extensions:

1. Users go to fix the part that will fail in future
2. Users can ignore the notification, but it will pop up from time to time

Priority of replacement reach max then update must be installed

Use Case ID: 04

Title: Confirm Firmware update.

Actor: User

Stakeholders and their needs:

- OEM needs to ensure that the new version of the firmware reaches all its customers.
- OEM server needs to deliver the new firmware version to all the vehicles connected to server.
- User -vehicle owner- needs to access the newly launched firmware version to

install it.

Trigger: A new firmware version is sent from the OEM server.

Pre-conditions:

- Vehicles have functional firmware running beforehand.
- OEM has a remotely accessible server in which they can upload the new firmware.
- The running firmware is not up to date.
- The vehicle has a working Wi-Fi connection to download the update.

Post-conditions:

- The vehicle's system is up to date with the latest firmware version available on the server.
- The new firmware is running, and all the modules are compatible and error free.

Basic flow:

1. The OEM launches the new firmware version.
2. The OEM then uploads the new firmware on their server.
3. Once the new firmware is available on the server the customers are sent a notification informing them of the availability of a new update.
4. If the user wishes to download and install the new firmware, the download begins.
5. After the firmware is successfully downloaded the user chooses when to begin the installation process.
6. With the completion of installation, the new firmware is ready to be running on the vehicle's ECUs.

Extensions:

1. The user chooses to ignore the newly available firmware due to the lack of network access.
2. The user insures there's a proper network connection.
3. After successfully connected to the server, the user downloads the firmware.
4. After the firmware is successfully downloaded the user chooses when to begin the installation process.
5. With the completion of installation, the new firmware is ready to be running on the vehicle's ECUs.

Use Case ID: 05

Title: Provide engine status.

Actor: Engine

Stakeholders and their needs:

- Models need data and engine state from sensors
- Models need a stage for data preprocessing and feature scaling
- Models need to ensure the range of sensors reading and that is like training set.

Trigger:

- Inform Model with Input feature to make prediction.

Pre-conditions:

- The software is running, and the sensor is active.
- Sensor is adjusted and ready for reading.

Post-conditions:

- Sending the engine state and sensor reading to the model.

Basic flow:

1. Sensor reading data and know engine state.
2. Data is preprocessed and makes Feature scaling process.
3. Data is ready as a feature for Model.
4. Model Take this Data as input and make prediction process.
5. The fixed firmware is released and uploaded to the server.

Extensions:

1. The sensor has some issues and is reading wrong.
2. Some range of what sensor read need to be adjusted as training set.
3. The predictive is successfully predicted as we want and find type of failure based on state of engine.

Use Case ID: 06

Title: Release a new firmware version

Actor: OEM

Stakeholders and their needs:

- OEM's firmware developers add new features and fix existing issues into the current firmware version.
- OEM needs to ensure that the new version of the firmware reaches all its customers.
- OEM server needs to deliver the new firmware version to all the vehicles connected to server.
- Customers -vehicle owners- need to access the newly launched firmware version to install it.

Trigger: A new firmware version is launched by OEM and provided on the OEM server.

Pre-conditions:

- Vehicles have functional firmware running beforehand.
- OEM has a remotely accessible server in which they can upload the new firmware.
- Vehicle owners and Maintenance centers have access to said server.

Post-conditions:

- New firmware is available for downloading and installation.

Basic flow:

1. The OEM's developers add new features to the previous firmware versions and fix existing bugs discovered by continuous software tests throughout the development process.
2. The OEM launches the new firmware version.
3. The OEM then uploads the new firmware on their server.
4. Once the new firmware is available on the server the customers are sent a notification informing them of the availability of a new update.
5. It's then up to the customer's choice whether they want to download and install the new firmware or not.

Extensions:

1. The customers have a problem downloading the new firmware from the server.
2. The OEM finds a problem with the authentication settings and fixes it.
3. The customers are then able to download the new firmware and install it.

Use Case ID: 07

Title: Provide firmware bug fixes

Actor: OEM

Stakeholders and their needs:

- OEM's developers need to fix existing bugs in the currently available firmware.
- OEM needs their customers satisfaction of provided services, so they can't afford to have an existing bug in their vehicles.
- Vehicle owners -users- need to fix any unexpected glitches in the system.

Trigger: A bug is found in the firmware and must be fixed.

Pre-conditions:

- Vehicles have functional firmware running beforehand.
- OEM has a remotely accessible server in which they can upload the new firmware.
- Vehicle owners and Maintenance centers have access to said server.
- Software testing teams run multiple tests to detect the exact problem in the current firmware versions.
- A software bug is found in the current firmware version.

Post-conditions:

- New firmware is available for downloading and installation.
- The software bug is fixed and now the firmware is errors and bugs free.
- The bug fixed firmware update is compatible with currently running firmware.

Basic flow:

1. The OEM's software validation teams find a bug in the current firmware in a specific module.
2. The bug is reported to the software development team.
3. The developers work on fixing the bug.
4. A new version of the corrupted module is available with no trace to the previously unleashed error.
5. The software validation team perform intensive tests on the new firmware update.
6. The fixed firmware is released and uploaded to the server.
7. Any vehicle containing the mentioned firmware module will then be able to download and install the bug fix from the server.

Extensions:

1. The development team provided a new version with the fixed bug.
2. The software validation team discovers compatibility errors between the newly edited module and the rest of the firmware.
3. The errors are reported to the development team.
4. The development team re-edit the module to ensure its compatibility with the firmware.
5. The final tests are performed**,** and the firmware bug fix is ready to be provided to the server.

Use Case ID: 08

Title: Send new firmware to customers

Actor: OEM server

Stakeholders and their needs:

- OEM needs to ensure that the new version of the firmware reaches all its customers.
- OEM server needs to deliver the new firmware version to all the vehicles connected to server.
- Customers -vehicle owners- need to access the newly launched firmware version to install it.

Trigger: A new firmware is uploaded to the server and needs to be sent to the customers.

Pre-conditions:

- Vehicles have functional firmware running beforehand.
- OEM has a remotely accessible server in which they can upload the new firmware.
- Vehicle owners and Maintenance centers have access to said server.
- Vehicles must have a connection to the internet to be able to receive the firmware update from the server.

Post-conditions:

- New firmware is available for downloading and installation.
- The vehicle's system is now up to date with the latest firmware version available on the server.

Basic flow:

1. The OEM launches the new firmware version.
2. The OEM then uploads the new firmware on their server.
3. Once the new firmware is available on the server the customers are sent a notification informing them of the availability of a new update.
4. It is then up to the customer's choice whether they want to download and install the new firmware or not.
5. If the customer chooses to download the firmware, they can download it from the server.

Extensions:

1. The download is interrupted due to the loss of Wi-Fi connection inside the vehicle.
2. The customer then provides a connection and resumes the download process.
3. The firmware is successfully sent to the customer.

4.6. Sequence Diagrams

Software Bug Fix Use case

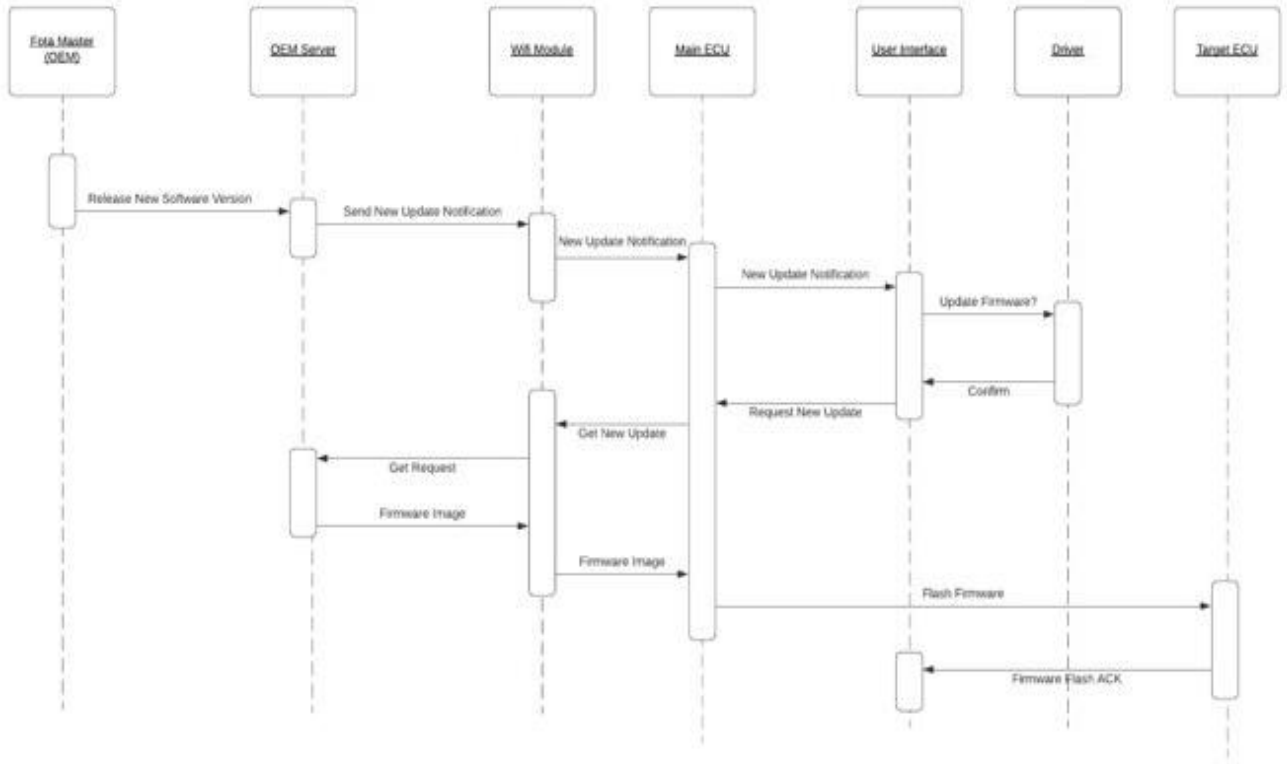

Figure 4.6 FOTA sequence diagram

Steps for updating the new software version:

1. OEM will upload a new software version in OEM Server.
2. OEM server will send notification to WIFI module
3. WIFI module will send it to main ECU.
4. Main ECU will send this notification to User interface.
5. User interface will alert driver.
6. User accepts the new update.
7. User interface requests new version from main ECU.
8. Main ECU send request to WIFI module.
9. WIFI module get request from OEM server.
10. OEM server will send firmware image to WIFI module.
11. The WIFI module will send this image to the main ECU.
12. Main ECU will send this firmware image to Target ECU.
13. Target ECU will send ACK to user interface to let Driver know of update completion.

Predicted Failure Use case

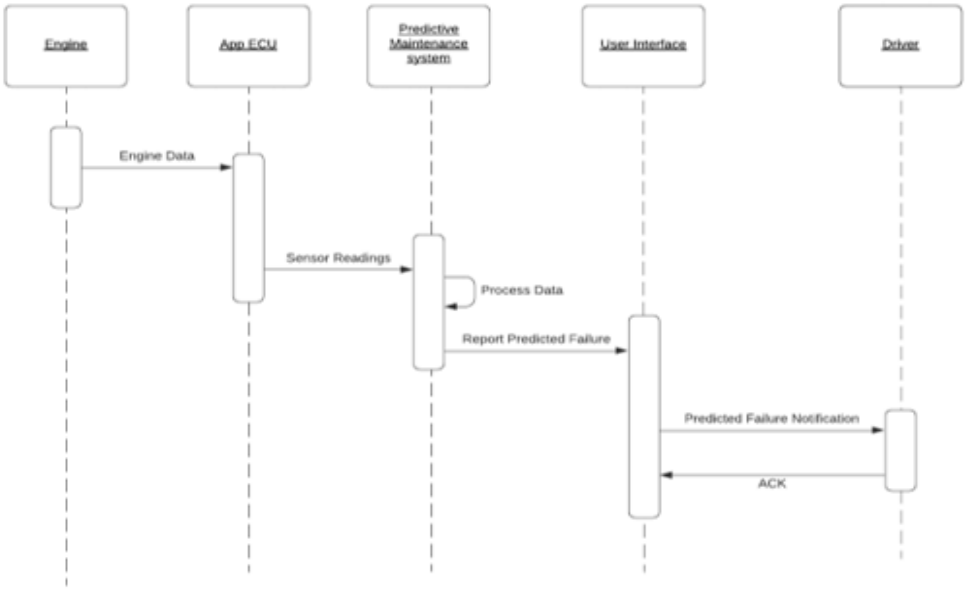

Figure 4.7 Failure prediction sequence diagram

Steps to predict failure:

1. Engine ECU will collect data from the engine using several sensors.
2. Engine ECU will send data to the predictive maintenance system in HMI ECU through CAN bus.
3. Predictive maintenance system will process this data.
4. In case of error detected, Predictive maintenance system will report GUI.
5. GUI will alert drivers.
6. The driver will send acknowledge to user interface.

4.7. Tools and Technologies

1. Eclipse: is an integrated development environment (IDE) for developing applications using the C programming language.
2. Keil: is Development Tools are designed to solve the complex problems facing embedded software developers.
3. Real Term: It can be used for monitoring, controlling, and viewing serial data and useful for sending and receiving data sent by serial communication protocols, most notably UART.
4. Anaconda: is the data science platform for data scientists, IT professionals and business leaders of tomorrow. It is a distribution of Python, R, etc. With more than 300 packages for data science.
5. Jupyter notebook: can use Jupyter Notebooks for all sorts of data science tasks including data cleaning and transformation, numerical simulation, exploratory

data analysis, data visualization, statistical modelling, machine learning, deep learning, and much more.

6. Google Colab: allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education.
7. FileZilla: is a utility for transferring files to or from a remote computer by a standard method known as FTP (File Transfer Protocol). FileZilla is open-source software that is installed and runs on Windows.
8. St-Link utility: is a full-featured software interface for programming STM32 microcontrollers. It provides an easy-to-use and efficient environment for reading, writing, and verifying a memory device.
9. Atom: is enables users to install third-party packages and themes to customize the features and looks of the editor.
10. Visual studio code: Use it to code in any programming language, without switching editors and support for many languages, we are using python.

4.8. Project Hardware

1. MCP 2551(Can transceiver).
2. MCP 2515(Can transceiver interface module).
3. Stm32 Blue Pill (ARM 32-bit processor).
4. Wi-Fi ESP module.
5. Raspberry pi 3 model b+.
6. St-link.
7. Pressure sensor.
8. Vibration sensor.
9. Hall Effect sensor.

Languages C, Python, Html, CSS, Bootstrap, PHP, jQuery, and Java script**.**

Chapter 5 : Predictive Maintenance System Implementation

Maintenance is an essential part of modern manufacturing systems [26]. The complexity of manufacturing machines that consist of multiple interdependent components requires an advanced maintenance model [27]. The massive amount of data that results from the integration between the physical and digital systems of the production process makes it possible for deep learning (DL) algorithms to be applied for fault prediction and diagnosis. Machines are now equipped with sensors that continuously collect information about their status. Predictive maintenance (Pd.M.) is one type of maintenance strategy that is applied to production equipment based on an estimate of the status of that piece of equipment [28].

The estimation aims to avoid breakdowns and maximize the service life of the equipment [29]. By connecting the devices with the sensors, the data of the devices can be used to identify patterns that lead to discovering failures before it happens [30].

These machines can monitor their status by continuously observing the conditions of various important features such as voltage, pressure, vibration, rotation, temperature, motor speed, etc. These features of a modern machine are used to train deep learning algorithms that can detect and predicting faults even before they occur [54].

This allows us to augment or even replace an ordinary/regular maintenance schedule through predictive maintenance, which is more reliable and advanced, reduces unnecessary maintenance and its associated costs, and guarantees the proper and timely functioning of machines.

This project used a hybrid deep learning framework based on CNN-LSTM for Pd.M. The method combines the advantages of convolutional neural networks (CNN), which extract the effective features and patterns among multi-variate input variables, and long short-term memory (LSTM), which captures the complex long-term dependencies and automatically selects the mode best suited for relevant time-series data.

The hybrid CNN-LSTM model is designed to make optimization easier by extracting the effective features using CNNs (Convolutional Neural Networks) and then incorporating LSTM in parallel to predict machine failures.

The main contributions of our project are summarized as follows:

- We used a hybrid deep learning model (CNN-LSTM) for Pd.M. The model uses CNN to extract features from the time series and LSTM for prediction. It uses a

time sequence of different errors and analyses the correlation between different input variables for better prediction.

- On comparing [34] the evaluation indexes of CNN, LSTM, and hybrid CNN-LSTM, it was found that hybrid CNN-LSTM has the highest prediction accuracy and is more reliable and suitable for Pd.M. forecasting.

A predictive maintenance model for a multi-component production system should consider the known state and the degradation threshold of each component [26].

5.1. Use Case

With the advent of connected sensors (IoT), data from equipment is continuously collected and data preprocessing techniques are used to pre-process and analyze the data. The data analysis can be presented through rich visualizations.

Once the data is prepared, deep learning algorithms such as LSTMs are used, which identify the correlations between different features and take the appropriate steps based on the data.

LSTMs are applied to the sensor data or time-series data to find anomalies within a production line. The LSTMs predict critical errors in advance to avoid a shutdown, fatal accident, or any unwanted event. The basic difference between LSTMs and other existing models is in terms of being more proactive and reactive. LSTMs are responsible for predictive maintenance.

CNNs perform automatic feature extraction and can learn a high-order representation of data (e.g., time-series or image-based data) [53]. However, CNNs alone when applied to time-series data are highly reliant on data pre-processing and tuning of a large pool of hyperparameters. In contrast, CNNs combined with LSTMs identify the most important features that contribute to the output. The combined model does not require heavy feature engineering, which makes the model computationally more reliable and efficient when compared to standalone CNNs.

5.2. Model Architecture

Deep learning (DL), as a subclass of artificial intelligence, has emerged as a powerful tool for developing intelligent algorithms in many applications [55]. DL is a technique inspired by the human nervous system and the structure of the brain [57].

With the ability to handle high dimensional and multivariate data, DL becomes an attractive methodology for practitioners for Pd.M. applications. Examples of DL algorithms are deep neural networks (DNN), convolutional neural network (CNN), deep belief networks (DBN), and recurrent neural networks (RNN). However, the

performance of DL algorithms depends on the appropriate choice of the DL technique for a given problem.
Therefore, this section presents a model based on hybrid convolutional neural network and long short-term memory network (CNN-LSTM) approach to identify possible failures.

5.2.1. Convolutional Neural Networks (CNN)

NN is a popular deep neural network (DNN) that takes this name based on the mathematical linear operation between matrixes called convolution. CNN is multiple layers fully connected layers. The most beneficial aspect of CNN is reducing the number of parameters [58], [59].
It was first introduced by Lacuna with the LeNet-5 architecture in the early 1980s [60]. This architecture consists of an input layer, several convolutional layers, pooling, and output layers. The output layer of the architecture can be bonded to fully connected layers or classifier layers such as the sigmoid layer.
Since it may be multimedia data such as image, sound, video, it is preferred by researchers working in many signals processing fields because of its high performance. To reduce the margin of error, a strategy by using a backpropagation algorithm is implemented. This strategy adjusts the CNN architectures to update their learning weights with a margin of error throughout the training process [61].

The details of the CNN algorithm used in this project are described in Algorithm 1.

Algorithm 1: CNN model

Input: x input features vector
F filter with size k x d
Output: c' output features vector
1: For I=1 to N
2: $w\,i\ = [x\,i\ , x\,i+1, \ldots. x\,i+k-1]$
3: $c\,i\ = (w\,i \odot F\ )$
4: End
5: $c' = M\,a\,x\,(c\ )$

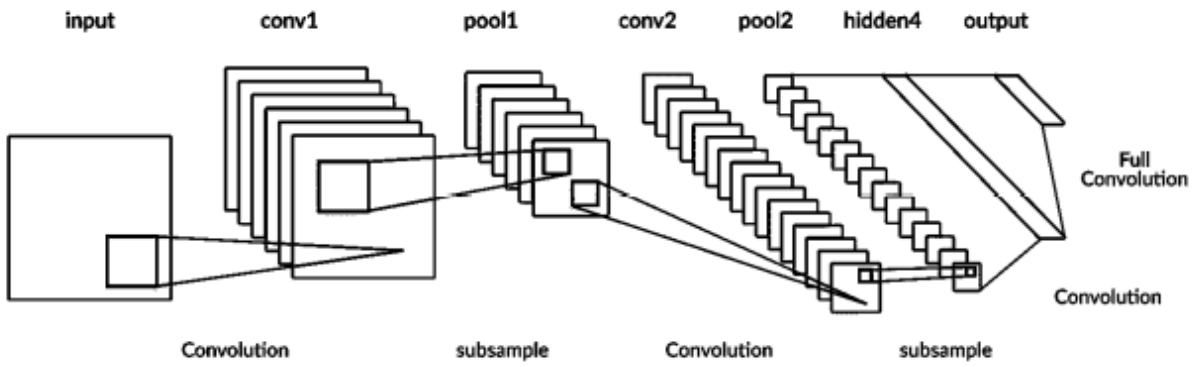

Figure 5.1 CNN structure

5.2.2. Long Short-term Memory (LSTM)

LSTM is a type of RNN network for sequence learning tasks. It does not have long-term time dependency problems where due to the sequential nature of sensor data, information can be reminded for long periods of time [62]. It can be defined as an iterative neural network algorithm that can learn long-term dependencies. Standard iterative algorithms such as neural networks consist of long-term information and are designed to avoid dependency problems remembering at this time. One of the main problems in RNN structure comes from vanishing gradients over time.

Since the learning network created during the training becomes complex, and the backward weight values of the network are updated because of zero or close to zero values, an update cannot be performed, and training may stop. This problem of backward coherence in RNN structure is presented as a solution by accompanying a memory cell RNN structure in LSTM structure.
With this memory cell, information from the previous time can be taken and transferred to the next [63], [64]. These units in the LSTM network remember long or short time periods. The values kept being remembered in these units do not interact in any way or experience change and disappear.
The structure of the LSTM unit. In the LSTM unit structure shown in figure below: the input X(t) takes the current input value, h(t-1) takes the previous hidden state and c(t-1) takes the previous memory state values. The output h(t) generates the current latent state and c(t) the current memory state. Algorithm 2 describes the LSTM algorithm used in this work.

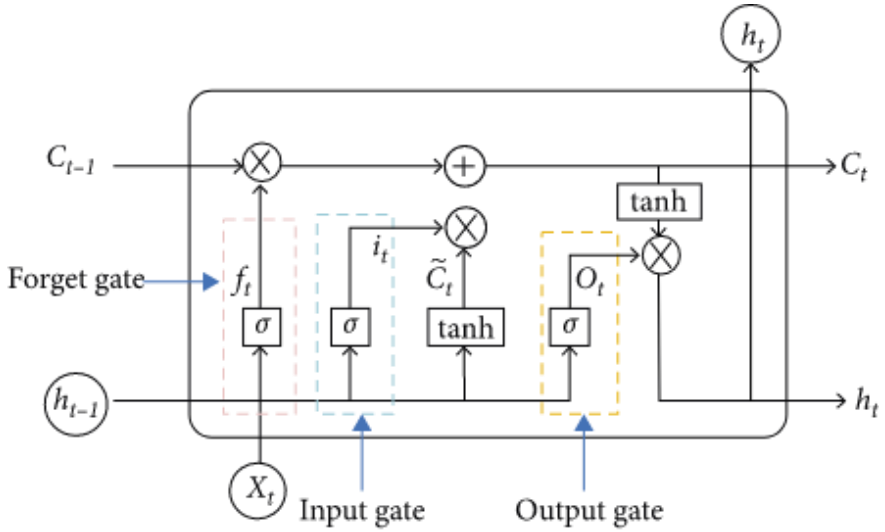

Figure 5.2 LSTM structure

Algorithm 2: LSTM model

Input: features vector c', Output: h vector

1: For j=1 to t

2: $i\,j\ = (W\,i\ .\ [ht-1 + b\,i\ ])$

3: $f\,j\ = (W\,f\ .\ [ht-1, x\,t\ ] + b\,f\ )$

4: $q\,j\ = t\,a\,n\,h\,(W\,q\ \ [ht-1, x\,t\ ] + b\,q\ )$

5: $o\,j\ = (W\,o\ \ [ht-1, x\,t\ ] + b\,o\ )$

6: $c\,j\ = f\,j \odot c\,j-1 + i\,j \odot q\,j$

7: $hj\,o\,j\ =$ tanh $\odot\ (c\,j\ )$

8: End

5.2.3. Hybrid CNN-LSTM Model for Pd.M.

C hybrid CNN-LSTM framework, which is a two-stage framework for predictive maintenance use cases. The overall structure of the proposed model is depicted below.

Algorithm 3: Hybrid CNN-LSTM model

Input X set x features

Output: Y prediction

1: Foreach x in X

2: $C\,x$ =CNN $(X\,x\ )$

3: End

4: Foreach x in C

5: $O\,x$ = LSTM $(C\,x\ )$

6: End

7: Foreach x in O

8: $Y\,x$ =sigmoid$(O\,x\ )$

9: End

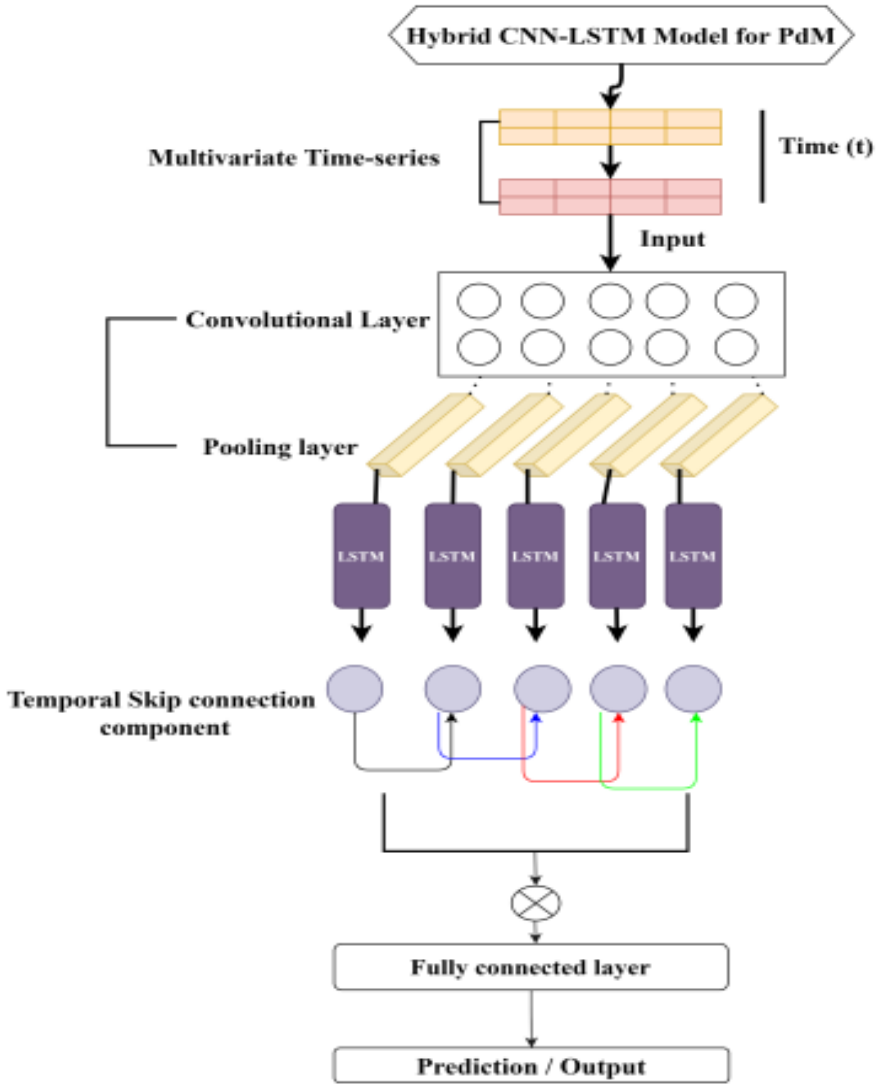

Figure 5.3 Hybrid CNN-LSTM structure

The main steps are as follows:

1. Input data: input the data required for CNN-LSTM training.
2. Data standardization: as there is a large gap in the input data,
3. Initialize network: initialize the weights and biases of each layer of the CNN-LSTM.
4. CNN layer calculation: the input data are successively passed through the convolution layer and pooling layer in the CNN layer, the feature extraction of the input data is carried out, and the output value is obtained.
5. LSTM layer calculation: the output data of the CNN layer are calculated through the LSTM layer, and the output value is obtained.
6. Output layer calculation: the output value of the LSTM layer is input into the full connection layer to get the output value.

7. Calculation error: the output value calculated by the output layer is compared with the real value of this group of data, and the corresponding error is obtained.
8. To judge whether the end condition is satisfied: the conditions for the end are to complete a predetermined number of cycles, the weight is lower than a certain threshold, and the error rate of the forecasting is lower than a certain threshold. If one of the conditions for the end is met, the training will be completed, update the entire CNN-LSTM network, and go to step 10; otherwise, go to step 9.
9. Error back propagation: propagate the calculated error in the opposite direction, update the weight and bias of each layer, and go to step 4 to continue to train the network.
10. Save the model: save the trained model for forecasting.
11. Input data: input the input data required for the forecasting.
12. Data standardization: the input data are standardized according to formula (8).
13. Forecasting: input the standardized data into the trained model of CNN-LSTM, and then get the corresponding output value.
14. Data standardized restore: the output value obtained through the model of CNN-LSTM
15. Output result: output the restored results to complete the forecasting process.

5.3. Pd.M. Model

All process to implement model in real word.

5.3.1. Data Set

The data used in this experiment is from Microsoft's case study. The dataset provides information about the failure history, maintenance history, error conditions, and machine features and telemetry, which consists of information such as voltage, pressure, vibration, and rotation sensor values. The entire dataset is summarized in Table shown below.

Table 5.1: A detailed description of different data sources.

Data Source	Description
Telemetry	Time-series data of voltage, rotation, pressure, and vibration measurements recorded in real time from 100 machines and averaged across every hour throughout 2015.
Errors	Non-breaking errors that machines throw while in operation
Maintenance	Scheduled and unscheduled records corresponding to inspection and failures of different components.
Machines	Includes machine summary such as model, age, and years in service
Failures	Records of the replacements due to failures

The data are collected for a total of 100 machines from 1 January 2015 to 1 January 2016. For each machine, data is collected on an hourly basis for one complete year. Therefore, for a total of 100 machines and each machine, we have $(1 * 365 * 24)$ 8760 and $(8760 * 100)$ 876,000 observations, respectively.

5.3.2. Exploratory Data Analysis

Distribution of the different machine features

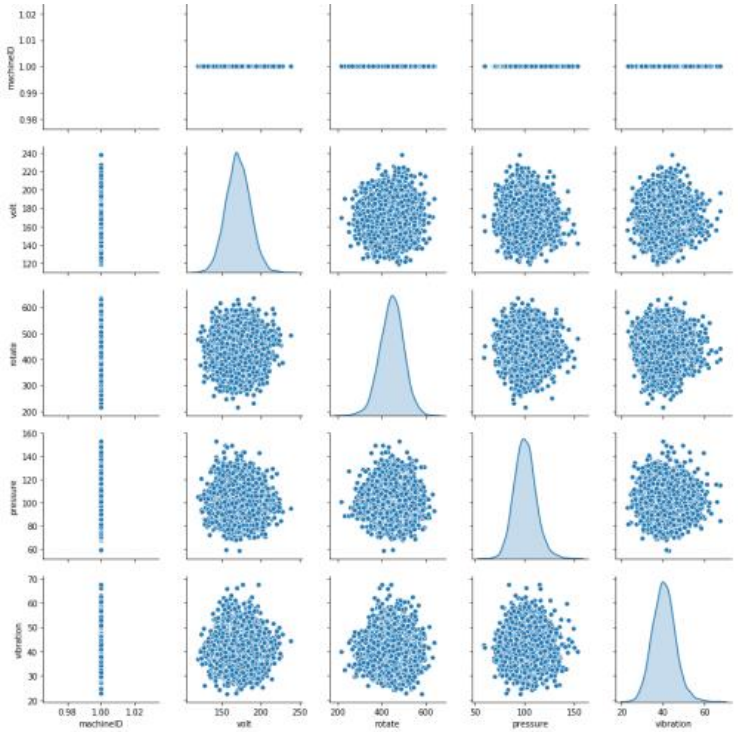

Figure 5.4 Distribution of the different machine features

Table 5.2: Display the first few rows of the Telemetry data for Machine 1

	datetime	machineID	volt	rotate	pressure	vibration
0	2015-01-01 06:00:00	1	176.217853	418.504078	113.077935	45.087686
100	2015-01-01 07:00:00	1	162.879223	402.747490	95.460525	43.413973
200	2015-01-01 08:00:00	1	170.989902	527.349825	75.237905	34.178847
300	2015-01-01 09:00:00	1	162.462833	346.149335	109.248561	41.122144
400	2015-01-01 10:00:00	1	157.610021	435.376873	111.886648	25.990511

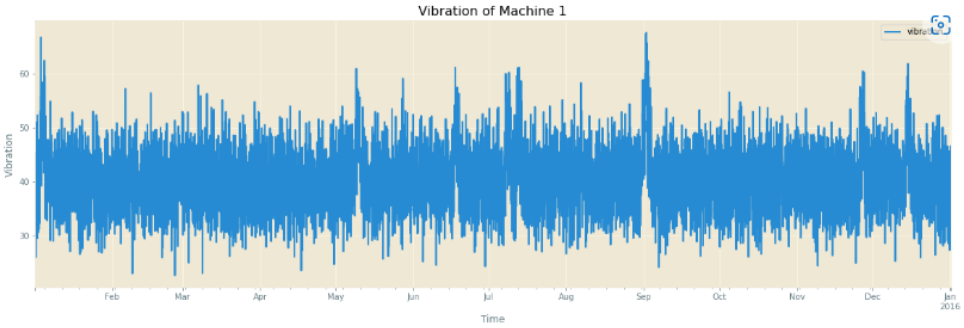

Figure 5.5 Vibration of Machine 1 for 2015

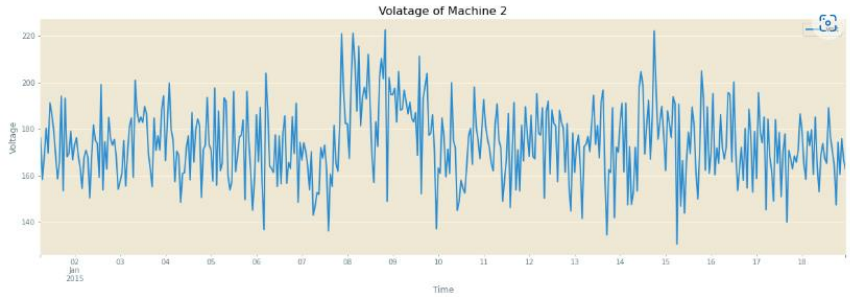

Figure 5.6 Voltage of Machine 2 for 1st two weeks of 2015

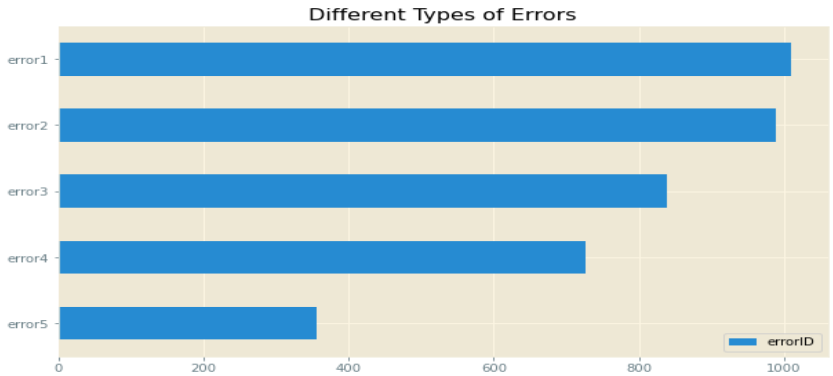

Figure 5.7 Frequency of different error conditions along with a set of 100 machines

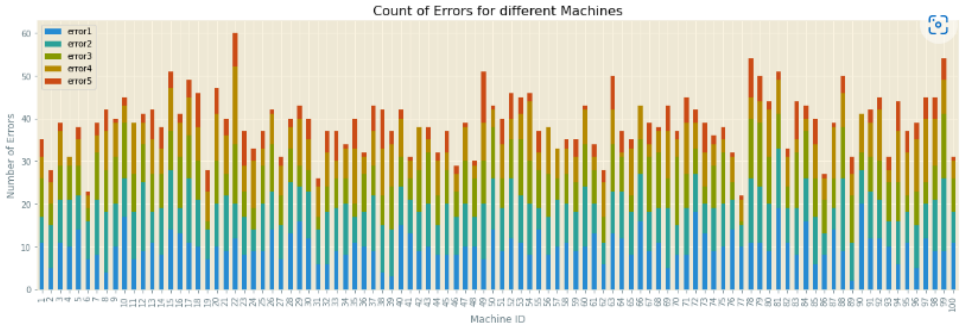

Figure 5.8 Machine to type of error distribution

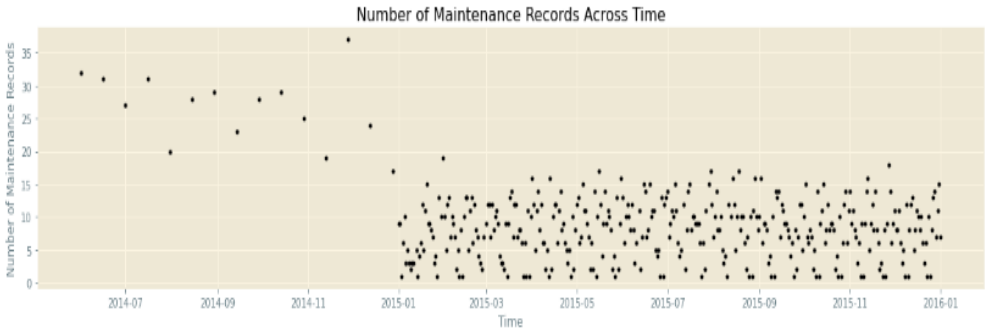

Figure 5.9 Number of Maintenance Issues raised per day

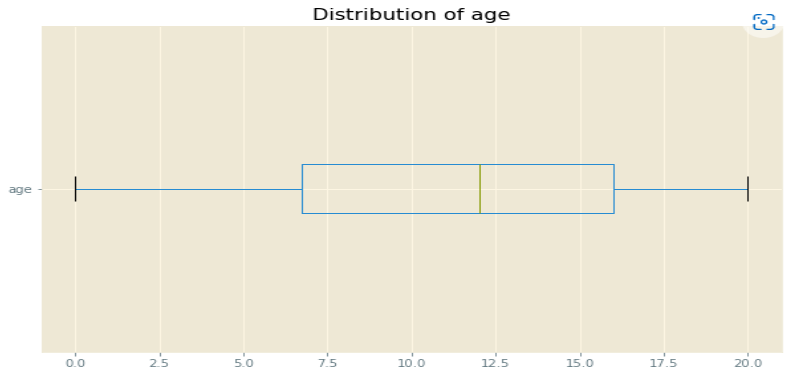

Figure 5.10 Distribution of age of the Machines

It was also observed that machine breakdowns are usually rare events of the assets as compared to normal operation in predictive maintenance. This generates an imbalance in the label distribution, resulting in poor performance because algorithms prefer to classify majority class examples over minority class examples, as the total misclassification error is greatly reduced when the majority class is correctly classified.

Although accuracy can be high, this results in low recall rates, which becomes a bigger issue when the cost of false alarms is considerable. To address this issue, simple strategies such as **oversampling minority cases** [65], were considered. The data are split: 70% training set, 10% validation set, and 20% test set, as shown in the Table below. Finally, the proposed model is evaluated on the test dataset.

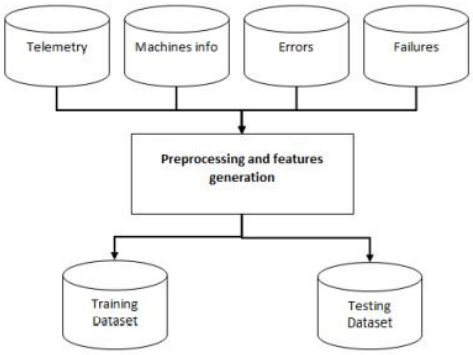

Figure 5.11 Processing and features generation

Table 5.3:Train, validation, and test data splits

Training Set	Validation	Test Set
70%	10%	20%

5.3.3. Label Generation

When utilizing multi-class classification to predict asset failure due to an issue, labeling is done by selecting a time window before the asset failure and labeling the records that fail within that window as "failure", whereas all other records are labeled as "normal".

The time range is chosen based on the business use case, such as in some cases predicting failure hours in advance may be sufficient, whereas in others, days or weeks may be required to allow for the arrival of new components. In our study we used a time-window of 24 h as our goal is to determine the likelihood that a machine would fail soon due to component breakdown.

The dataset provides information of specific component failures such as component 1, 2, 3, and 4. We create a categorical feature that serves as label like failure = comp1, failure = comp1, and so on.

Feature Engineering

Table 5.4: Features of the generated dataset

Feature	Description
1	Time stamp (Date Hour)
2	Machine ID (1-100)
3-6	Sensors values (voltage, rotation speed, pressure**,** and vibration)
7	Machine information (age)
10	Error ID (Error1, Error2, Error3, Error4)
12	Failure ID (Comp1, Comp2, Comp3, Comp4)

5.3.4. Model Implementation

The general structure of the proposed hybrid CNN-LSTM model for Pd.M. is shown in Figure 5.3. This structure takes the advantages of CNN and LSTM by combining them together. CNN deep learning method is well-known for its robustness and effectiveness for extracting features from the data, while LSTM method is effective and powerful for classifying time series. The structure of the hybrid CNN-LSTM model is described as follows: The model accepts input features shown in the table above and output decision for the machine failure state for the next hour based on the machine state for the last 24 hours.

The algorithm of the proposed Hybrid CNN-LSTM model for Pd.M. is described in Algorithm 3. For CNN model the 1-dimensional convolution layer is used. This layer has a filter size of sixty-four and kernel size of 1 and uses rectified linear unit (ReLU) as an activation function. This layer is followed by a 1-dimensional maxpooling layer with a pool size equal to 2, and then a dropout layer is added to overcome overfitting issues.

The output from this layer is projected and fed to the LSTM model. On other hand, LSTM model consists of two hundred hidden layers using ReLU as activation function and binary_crossentropy as loss function. The output of the LSTM model is fed into a fully connected layer with sigmoid activation. This layer is responsible for providing the final binary-classification decision for a given input to the model. The details of the proposed hybrid CNN-LSTM model for Pd.M.

Table 5.5: Different parameter settings.

Parameters	**Value**
Convolutional layer filters	64
Convolutional kernel size	1
Convolutional layer activation function	Relu
Pooling layer pool size	1
Pooling layer activation function	Relu
Number of LSTM hidden cells	200
Number of skip connections	2
LSTM activation function	Tanh
Batch size	32
Loss function	binary_crossentropy
Optimizer	Adam
Learning rate	0.0001
Epochs	100

5.3.5. Pd.M. Model Training and Evaluating

The proposed CNN-LSTM model for Pd.M. is trained using the training dataset generated in the previous steps. The model is implemented in the TensorFlow platform [34] with GPU acceleration is active. The hyper-parameters for the model are used based on the values shown in the table above. After the training of CNN-LSTM model is completed, the model is evaluated using the testing dataset prepared earlier.

Since the training data is unbalanced, the performance of the model is evaluated based on the model accuracy metrics precision, recall, and F-score as given in (1), (2) and (3). These metrics are calculated based on the confusion matrix shown in the table below as follows:

Precision measures the exactness of the model and can be calculated by (1):

$$Pression = TP / (TP + FP) \quad (1)$$

Recall measures the completeness of the model and can be calculated as shown in (2):

$$Recall = TP / (TP + FN) \quad (2)$$

F-score is the weighted average of Recall and Precision. It's wildly used to measure the model accuracy where imbalanced data is used for training. F-score can be calculated as in (3):

$$F - score = (2 * Pression * Recall) / (Pression + Recall) \quad (3)$$

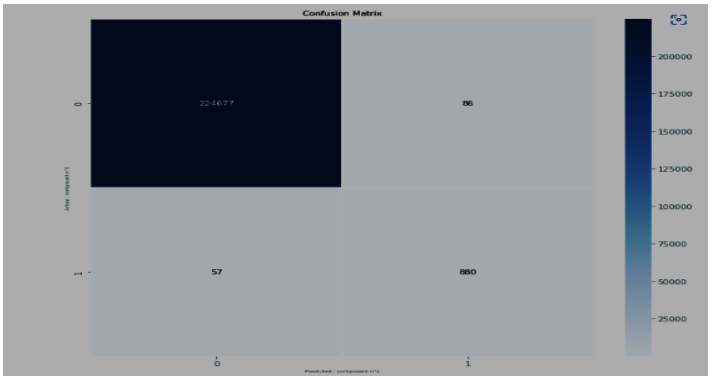

Figure 5.12 Model confusion matrix

Table 5.6: Model evaluation metrics

Accuracy:	**0.9993664155959238**
Recall:	**0.9391675560298826**
Precision:	**0.9109730848861284**
F1 score:	**0.9248554913294798**

5.4. Engine Condition Monitoring

Traditionally, predictive maintenance is being done using rule-based techniques. With the advent of connected sensors (IoT), data from equipment is continuously collected and fed to Machine Learning based systems to predict its future health.

The proliferation of sensing technologies such as sensors has resulted in vast amounts of time-series data being produced by machines in industrial plants and factories. There is much information available that can be used to predict machine breakdown and degradation in a given factory.

The downtime of industrial equipment accounts for heavy losses in revenue that can be reduced by making accurate failure predictions using the sensor data. Internet of Things (IoT) technologies have made it possible to collect sensor data in real-time. We found that hybrid modelling can result in efficient predictions as they are capable of capturing abstract features which facilitate better predictions.

In addition, developing an effective optimization strategy is difficult because of the complex nature of different sensor data in real-time scenarios. This project proposes a method for multivariate time-series forecasting for predictive maintenance (Pd.M.) based on a combination of convolutional neural networks and long short-term memory.

The data used in this experiment are from Microsoft's case study. The dataset provides information about the failure history, maintenance history, error conditions, and machine features and telemetry, which consists of information such as voltage, pressure, vibration, and rotation sensor values recorded between 2015 and 2016.

5.4.1. Sensors' Readings

Engine pressure

We use BMP180 Digital Pressure Sensor to collect pressure of the engine readings. The BMP180 is the function compatible successor of the BMP085, a new generation of high precision digital pressure sensors for consumer applications. The ultra-low power, low voltage electronics of the BMP180 is optimized for use in mobile phones, PDAs, GPS navigation devices and outdoor equipment. With a low altitude noise of merely 0.25m at fast conversion time, the BMP180 offers superior performance. The I2C interface allows for easy system integration with a microcontroller. The BMP180 is based on piezo-resistive technology for EMC robustness, high accuracy, and linearity as well as long term stability. Robert Bosch is the world market leader for pressure sensors in automotive applications. Based on the experience of over 400million pressure sensors in the field, the BMP180 continues a new generation of micro-machined pressure sensors.

Figure 5.13 BMP180

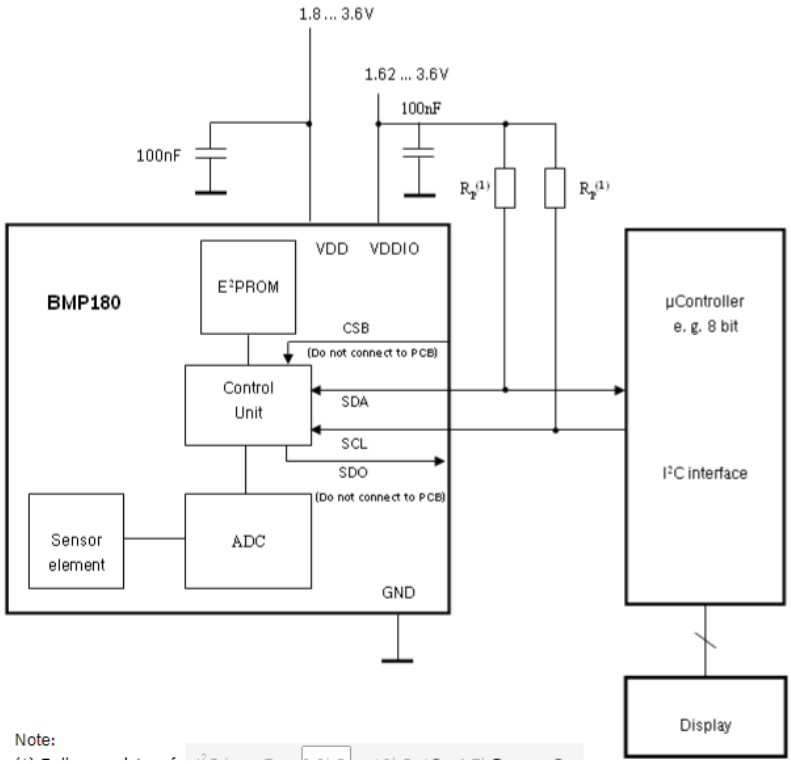

Figure 5.14 Interfacing with Application ECU

Measurement of pressure

For all calculations presented here an ANSIC code is available from Bosch Sensortec ("BMP180 _API"). The microcontroller sends a start sequence to start a pressure or temperature measurement. After converting time, the result value (UP or UT, respectively) can be read via the I2C interface. For calculating temperature in °C and pressure in hPa, the calibration data must be used. These constants can be read out from the BMP180 E2PROM via the I2C interface at software initialization. The sampling rate can be increased up to 128 samples per second (standard mode) for dynamic measurement. In this case, it is sufficient to measure

the temperature only once per second and to use this value for all pressure measurements during the same period.

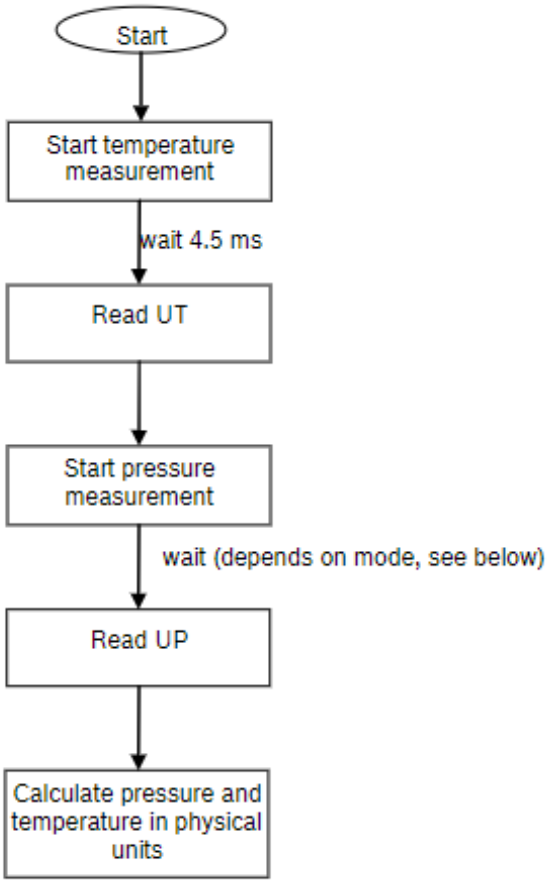

Figure 5.15 Measurement flow BMP180

Calculating pressure and temperature

The mode (ultra-low power, standard, high, ultra-high resolution) can be selected by the variable oversampling setting (0, 1, 2, 3) in the C code. Calculation of true temperature and pressure in steps of 1Pa (=0.01hPa = 0.01mbar) and temperature in steps of 0. 1°C.The following figure shows the detailed algorithm for pressure and temperature measurement.

Engine vibration

We used 801S vibration sensor to monitor the vibration of the engine. 801S is a vibration sensor with a gold alloy plated shock sensor and an onboard comparator IC. The sensor can provide real-time vibrational data along with micro shock detection and about 60,000,000 times shock guaranteed.

Figure 5.16 801S vibration sensor

Features and Specifications of 801S Vibration Sensor

Here are some features and specifications of the vibration sensor module:

1. Voltage Input: 5V.
2. Analog and TTL level signal output.
3. Adjustable sensitivity.
4. Micro shock detection.
5. Gold alloy plated Shock sensor.

Connections of 801S Vibration sensor are shown in the image below. 801S is a 4-pin module having power pins along with digital and analogue output pins. The VCC pin of the sensor module is to be connected to the +5V pin, the ground is connected to the ground terminal of the microcontroller. The analogue pin (A0) and digital pin (D0) on the sensor module are connected to the analogue and digital terminals of the microcontroller, respectively.

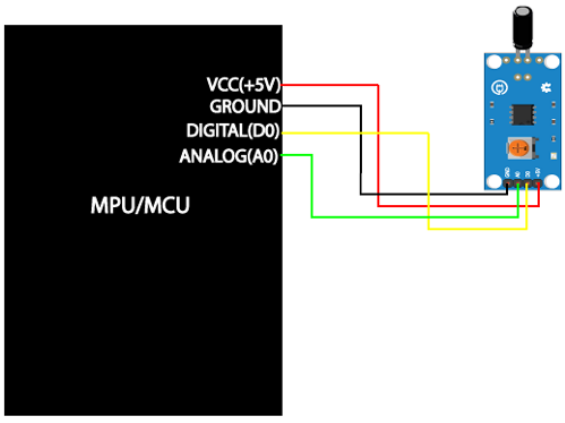

Figure 5.17 Interfacing 801S

Engine speed

The Melexis MLX90217 is a digital gear-tooth hall effect sensor designed for use in automotive camshaft sensing as well as other speed sensing applications. It is designed to be used with a magnet to sense the rotating speed (incremental position) of a ferrous tooth-wheel.

Figure 5.18 MLX90217

Hall-effect gear-tooth sensors detect changes in a magnetic field. When a gear tooth or other ferrous object approaches a magnet, it distorts the field surrounding that magnet. The presence or absence of ferrous gear tooth are detected by measuring this distortion. The most common method of measurement places a sensor on one magnet poles and observes an increase in the magnetic field as a target approaches the pole.

Parallax reported using the Melexis 90217 sensor in applications including CNC milling machine spindle speed measurement and feedback on motor RPM. With the CNC milling machine it reliably measured the top spindle speed of 7,500 RPM.

Features

- On-chip 10-bit A/D converter.
- Self-adjusting magnetic range.
- Voltage: 3.5-24 VDC.
- Micropower Operation.
- Current: ~3 mA.
- Rotary position gear tooth sensing ability.
- High speed operation.
- Zero speed detection.

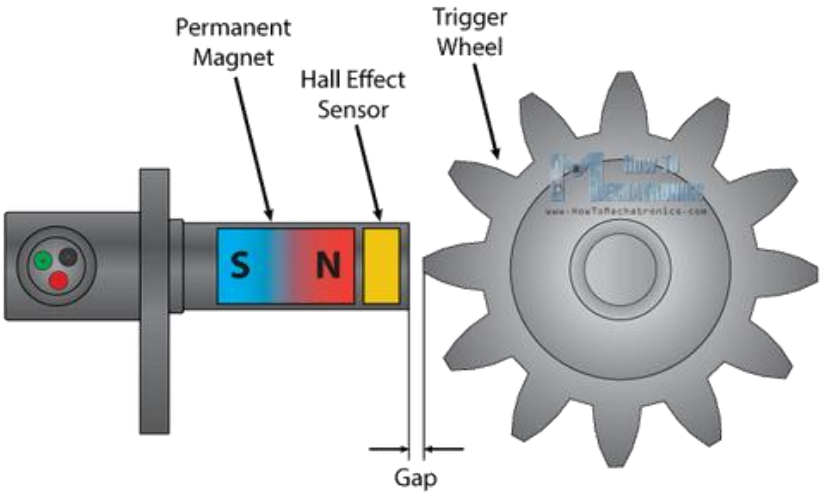

Figure 5.19 Hall effect sensor operation

Applications

- Gear tooth rotation speed detection.
- Shaft rotation speed detection.
- Proximity detection.

5.5. CAN in Our Project

5.5.1. The Specification Transmitting of CAN:

1. Speed: 500Kbs.
2. Priority by transmit request order.
3. Auto retransmission in case of message fails to be transmitted.
4. Using 3 mailboxes each can have 8 bytes of data

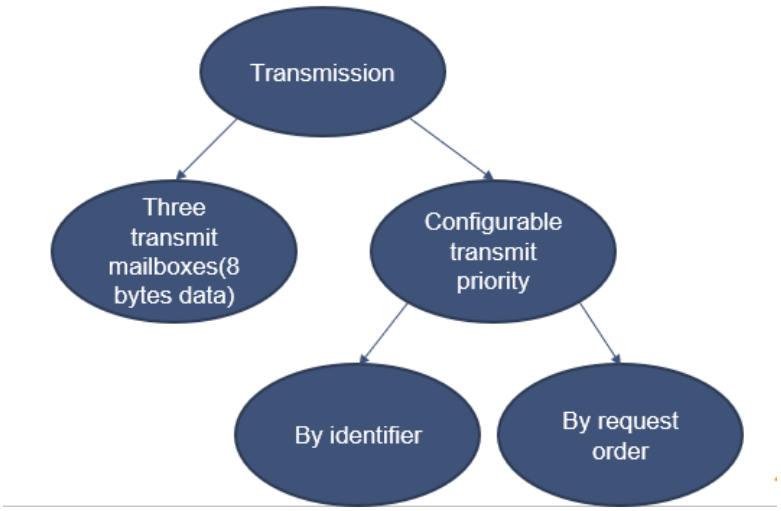

Figure 5.20 CAN transmit configuration

5.5.2. The Specification Receiving of CAN:

1. Speed: 500Kbs.
2. Two receive FIFOs, each one with three stages.
3. Used Identifier list mode method for filtering.

Means writing a list of message IDs that we are interested in receiving.

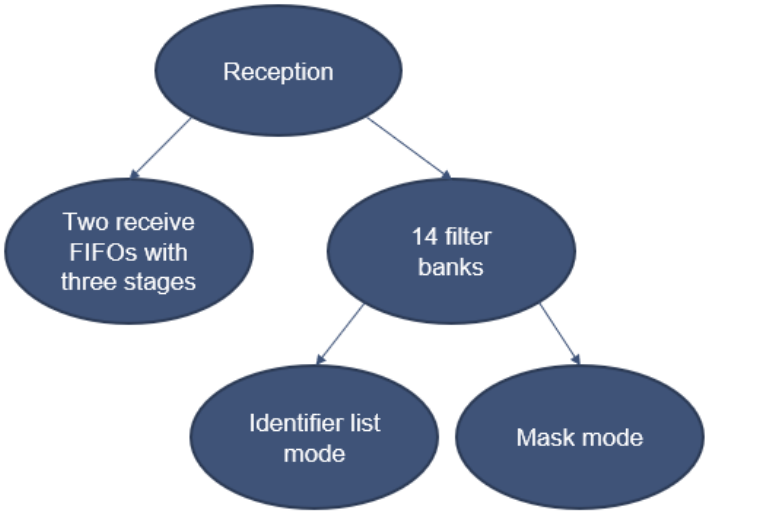

Figure 5.21 CAN receive configurations

5.5.3. Transmit Handling

First, we have an empty mailbox. When writing to it, it turns into a pending state, and as we discussed, we configured transmitting priority by transmitting request order. So, the highest priority mailbox of the 3 mailboxes will be sent first if the CAN bus is idle. Then we empty this mailbox to be ready again.

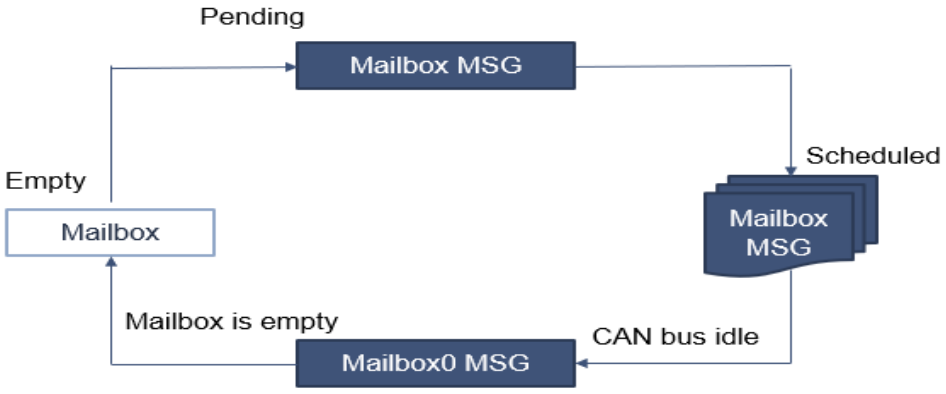

Figure 5.22 Transmitting handling

5.5.4. Reception Handling.

As we discussed later, we have 2 FIFO mailboxes to receive messages. Each one can have 3 messages. We have used 1 FIFO in reception. So, if we have received one message, FIFO will turn to pending one state. If we receive another one, it will be pending 2 states. If we receive another message, it will be pending 3 states. And it will be full then. And if we have received another message, it will be lost, so we have to empty the FIFO mailbox.

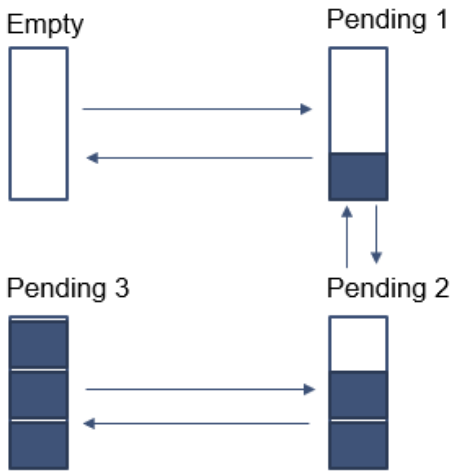

Figure 5.23 Reception handling

5.5.5. CAN in PDM

The engine ECU will collect readings from the engine using different sensors like pressure, rotation, and pressure. Everyone hour each day, which means twenty-four readings each day. So, we need to send this data to the HMI ECU to be processed. That's why we need CAN. We set the filter of HMI node (65). As we mentioned before, we chose to use the identifier list mode method for filtering. So, we put ID 65 to read CAN messages to be accepted by HMI node filters. Data frames are used in this process.

5.5.6. CAN in FOTA

The main ECU checks if there is a new version on the OEM server. If there is a new update, the main ECU will send a message with ID 75 to the HMI ECU. And we have set a filter with ID 75 in the HMI ECU to accept this message. When the user accepts this update, the main ECU will download the update and it will need to send it to the target ECU to be flashed. At first, we divide the hex file into packets and send the first packet in a data frame with ID 10. So, we set the filter in the target ECU with ID 10. When the target ECU receives the first packet, it sends an ACK in the remote frame to

the main ECU, instructing it to send the next packet, and so on. As shown in Figure below.

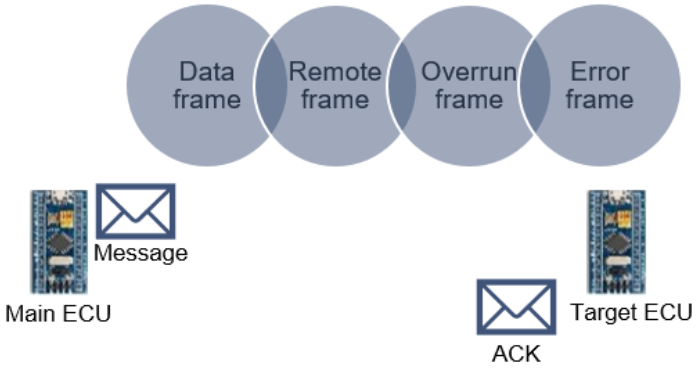

Figure 5.24 CAN in FOTA

In the case of an error, it is detected automatically as CAN has built-in-error detection as we know from chapter 2. The node will send an error frame to inform the rest of the network. And in the case of an overrun in the receiving node in its FIFO mailbox, it will send an overrun frame.

5.5.7. Steps to Connect CAN Network.

1. First put a list of the ID's that each node can accept in the filter registers.

2. Try with only two nodes and only one mailbox and one FIFO.

3. Work with loopback mode first to check software written.

4. If step 3 worked well, that's great, now try normal mode.

5. If step 5 worked now you should only scale the driver to use the 3 mailboxes and 2 FIFOs.

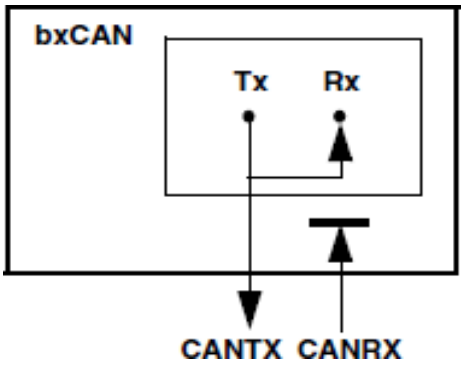

Figure 5.25 Loop back mode

Chapter 6 : FOTA System Implementation

6.1. Firmware Over The Air (FOTA)

With the FOTA firmware update method, the boot loader and application both manage the upgrade procedure on their own without the need for a technician. The new firmware replaces the old firmware in several ways. The new firmware entirely replaces the old firmware. The updated firmware is often sent with IOT technologies over cellular networks, typically LTE 4G, 3G, or 2.5G. In some enterprise applications, LORA WANs is also used to provide OTA firmware updates. For the embedded device to obtain the necessary update, a secure communication channel is established between the server and the system. In a Secure Firmware Update, the agent frequently transmits the updated or patched firmware code to the gateway or microcontroller. Both the firmware and the source of the recipient are verified. If the authentication process is successful, the new firmware will be flashed onto the flash memory by the boot loader, replacing the old firmware. The memory of the current firmware stays locked if any of the source firmware or updated firmware is not approved [78].

6.1.1. Hexadecimal File (HEX File)

We write the executable instructions for embedded devices in a high-level language like C, C++, or Rust. We must translate the high-level language into machine language that the processor can understand in order for the microcontroller to know what to perform. We compiled our own code because of this. After the compilation, we obtained an output that was converted into a HEX file.

Hex file refers to Intel hexadecimal object file format, The Intel hex format is a file format that presents binary data as ASCII text. [24] It is frequently used to program hardware emulators, programmable logic devices, EPROMs, and other kinds of programmable logic.

Intel HEX is made up of ASCII text lines that are separated by line feed, carriage return, or both. Hexadecimal characters are used to encode several binary numbers on each text line. Depending on their location in the line, the type and length of the line, and the data they represent, the binary integers may be used to represent data, memory addresses, or other values. A record is a single text line.

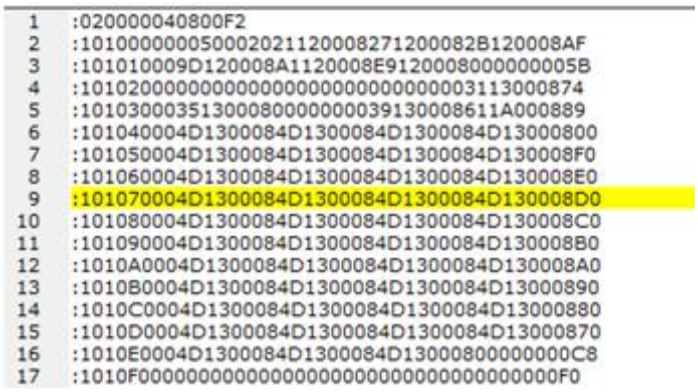

Figure 6.1 Hex file

Six fields (parts) make up a record (a line of text), and they are arranged from left to right as follows: [25]

1. **ASCII colon ':'**

As the first character in the code.

2. **Byte Count**

The data field is indicated by the byte count, which are two hex digits (or one hex digit pair). The maximum number of bytes is 255. (0xFF).

3. **Address**

Four hexadecimal digits that represent the data's 16-bit starting memory offset. Memory addressing is made possible beyond the 64-kilobyte restriction of 16-bit addresses by computing the physical address of the data by adding this offset to a previously determined base address. Different kinds of records have the ability to alter the base address, which starts at zero.

4. **Record Type**

two hex digits from 00 to 05, describing the purpose of the data field (see record types below).

5. **Data**

A collection of n bytes of information represented by 2n hexadecimal digits. Depending on the application, data bytes have different meanings and interpretations.

6. **Checksum**

A two-digit hex value that can be used to confirm that the record is error-free.

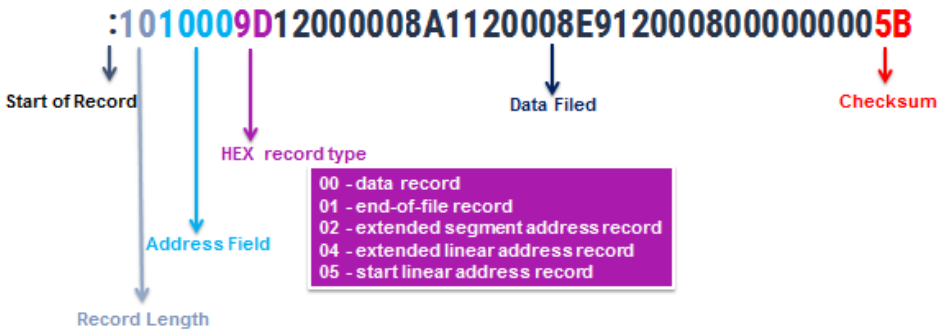

Figure 6.2 Hex record structure

Getting to know the hex file and its properties is very important because we will need it a lot during the flashing and data checking phase.

6.1.2. Flashing Firmware

Firmware is the term used to describe the small programs found on ROM (Read Only Memory) modules that house low-level (hexadecimal, machine-level) software. They give the device they are on the ability to evaluate its capabilities and make those capabilities useful. The data stored in the ROM is non-volatile, which means that it is not erased when the power is turned off.

Despite the hardware functioning normally, the firmware also coordinates those actions and provides the programming language necessary to perform those operations and includes the computer language elements necessary to carry out such actions. Compared to using pure hardware circuitry, using firmware offers more flexibility.

The ability to upgrade the firmware of a device can be achieved by connecting it to your PC in a specific way and then executing the software that the manufacturer supplies. Some firmware cannot be upgraded, however others can.

Flashing is the term used to describe this process. This is required either to improve the performance of the device or because the device is no longer compatible with more recent operating systems. It's crucial to adhere to the manufacturer's guidelines when updating firmware in order to avoid damaging the device.

There are various procedures that must be followed in order to write the new instructions to the flash memory. Flash memory requires a particular voltage to function, and typically the processor's ability to create that voltage is insufficient for this usage. Therefore, flashing requires an external device called Flash Programmer.

The executable file can be flashed to flash memory in three ways.

1. OFF Circuit Programming

OFF Circuit Programming is a flashing technique that requires the microcontroller to be linked to a different board called burner. The burner is used to write firmware to flash memory using a flash programmer, a piece of hardware that can read and write to the flash memory of the microcontroller by applying the appropriate voltage necessary for the flash memory to respond to the write operations.

Figure 6.3 Off-circuit Programming

This is due to the fact that the CPU is unable to provide this voltage level, making an external device necessary in order for flash memory to be written to. The microcontroller is put back on the system board when the flashing procedure is complete to begin the service for which it was developed.

Every time the firmware needs to be updated, we must remove the microcontroller from the system board, flash the new firmware, and then reattach it. The issue is that since the microcontroller must typically be soldered to industrial boards, moving it around every time a firmware update is required is an ineffective way.

2. In Circuit Programming

In-Circuit Programming (ICP) is a technology in which the programmable device is programmed after the device has been attached or soldered to a circuit board. In in-circuit programming, the flash programmer is installed on the microcontroller, and we can communicate with it by using a communication protocol that the flash programmer can understand and send the executable file (HEX file) to the flash programmer inside the microcontroller to flash it on the flash memory.

The hex file is typically on a computer or other device, so we need to utilize a flasher/debugger to transfer it from the device to our flash programmer. Since one of

them may use a different communication protocol than the other, we need a translator between them so that they can understand one another and successfully transfer the data of the Hex file. In this case, the debugger is not used to write binaries to flash memory but rather as a translator between the computers containing the hex file and the flash programmer on the microcontroller.

We used ST-LINK / V2 to be able to program the nodes in our system. The ST-LINK / V2 is an in-circuit debugger and programmer for the STM8 and STM32 microcontrollers. Which allowed us to flash the required HEX file from our computer on flash memory for each ECU. This is for contracts that do not need to change the firmware during runtime and for code such as a boot loader, which will be able to change the application firmware in the future but must be located on flash memory using the in-circuit programming method only for the first time.

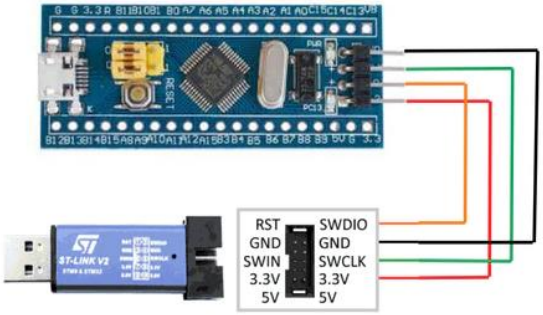

Figure 6.4 Debugger

Flexibility is increased by in-circuit programming because previously, firmware updates required removing the microcontroller and attaching it to a flasher.

Now that microcontrollers can be attached to industrial boards and reprogrammed whenever necessary, all that is required is to connect the microcontroller to the computer that contains the hex file using the debugger.

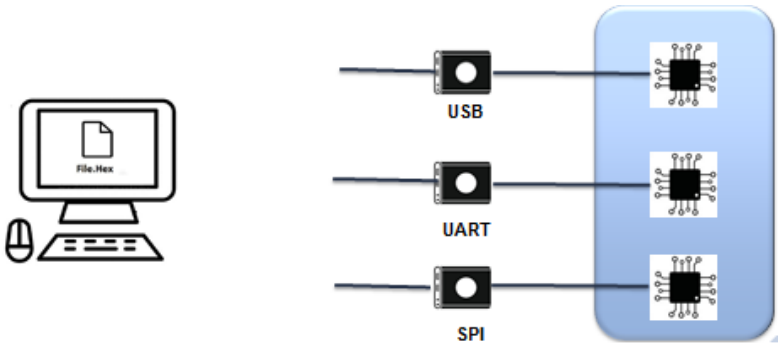

Figure 6.5 In-circuit programming

If our system only has a few electronic control units, this is a simple and effective way. However, this approach is completely ineffective in industrial applications and in large systems. Today's vehicles may contain 100 ECUs or more.

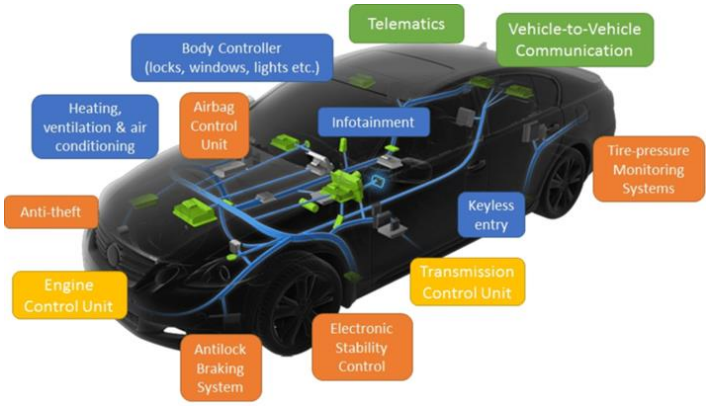

Figure 6.6 Automotive industry

What happens to these microcontrollers and a software "enhancement" needs to be made? Do we have to connect a cable to each ECU to be connected to the debugger to update the software?

Most definitely not, as it would be impractical to attach a connection from each ECU to the debugger each time the software needs to be updated. And that is why most systems have boot-loader on-board.

3. Boot loader

The boot loader is the most crucial component of the FOTA system since it represents FOTA's primary purpose, which is to automatically flash fresh firmware automatically without human intervention. The boot loader is independent software running on flash memory that receives the Hex file and writes it line by line to the flash memory. It is also in charge of navigating to the section of the program that performs the main system functionality.

The main benefit of the boot loader we used in the vehicle system is that it overcomes the issues with in-circuit programming that occurred because of the system's numerous electronic control units and enables us to upgrade the firmware without any physical connections. By integrating all the ECUs on one network, we can transmit the hex file to the desired ECU using the network's communication protocol when we need to update the firmware on one of the ECUs, doing away with the debugger, the physical connection, and any human involvement during flashing the firmware.

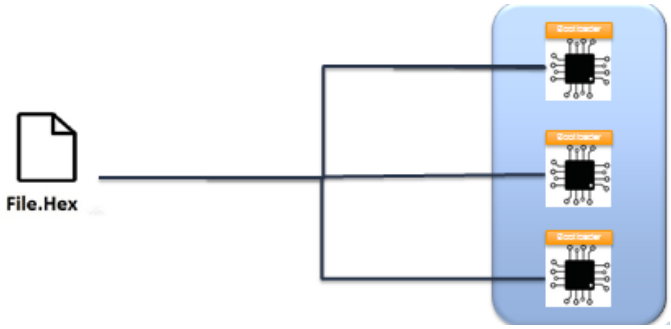

Figure 6.7 System network

6.1.3. Boot Loader Design

To find the boot loader design that best satisfies the needs of our system, we went through several prototypes. The flash memory is totally dedicated to the application part before the boot loader. This section holds the instructions that the microprocessor uses to carry out the function for which it was designed.

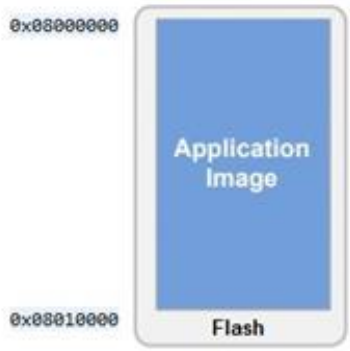

Figure 6.8 Flash memory design

But after using the boot loader, the internal design of the flash memory is different because it is originally a program and a set of instructions that must be present on the flash memory for the microprocessor to be able to execute them.

The internal design of the flash becomes two separate parts,

- **Boot loader Part**

  In-circuit programming method is used to write the boot loader firmware on the flash,

- **Application Part**

  This Firmware is written using the boot loader.

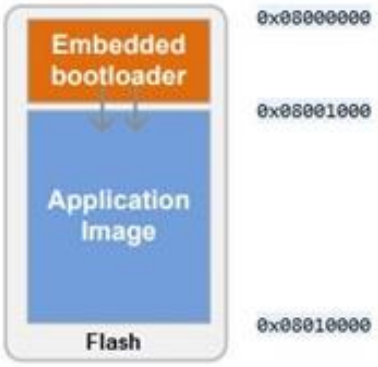

Figure 6.9 Bootloader design

Our microcontroller STM32F103C8 has 64KB flash memory. We allocate a part at the beginning of the flash memory for the boot loader with a size of 4 KB, starting from address 0x8000000 and ending at 0x8001000, and the rest of the flash memory 60 KB is dedicated to the application.

6.1.4. Boot Loader Sequence

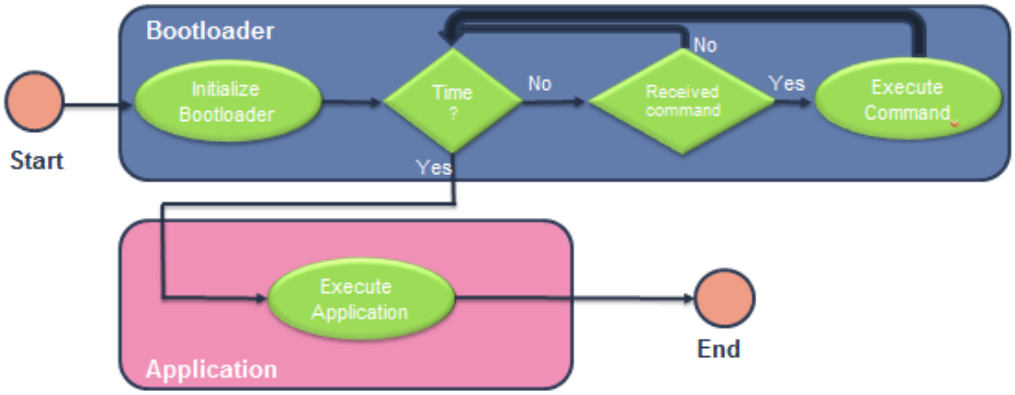

Figure 6.10 Boot loader sequence

After resetting the microcontroller, the execution of this sequence begins

1. The processor jumps to the boot loader section

2. Check whether the time specified for receiving instructions has passed or not

3. If the specified time does not pass, the processor starts waiting for commands (HEX File) to write them in the flash

4. When the command is received, the processor starts writing it in the flash and comes back again to check if the time has passed or not

5. When the specified time has passed, the processor jumps on the application part and starts executing it.

6.1.5. Boot Loader Software Architecture

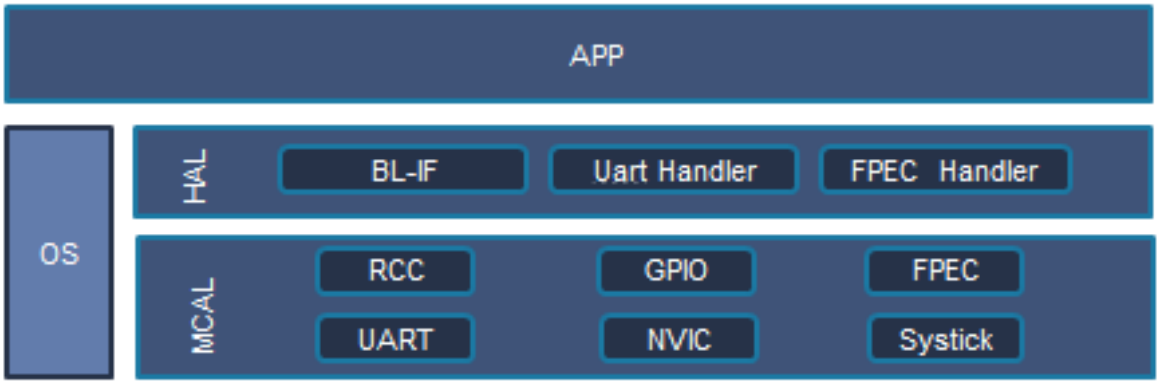

Figure 6.11 Boot loader software architecture

6.1.6. Internet of Things (IoT)

This component oversees converting the system from a simple embedded system to an Internet-connected system; it connects to the OEM server, searches for new firmware versions, and notifies the user when new versions are found. IOT PART can request the hex file from the server, download it, and then transmit it over CAN bus to the boot loader if the user requests to download the updated version.

The ESP8266 is a low-cost Wi-Fi microchip that has microcontroller and TCP/IP networking software capabilities built in.

With the use of Hayes-style commands, this tiny module enables microcontrollers to join a Wi-Fi network and establish straightforward TCP/IP connections.

The ESP8266 is a Serial (UART) to Wi-Fi SoC (System on a Chip) built around a Tensilica Xtensa LX3 DPU and was created by Espressif Systems, based in Shangai. This tiny IC has an onboard TCP/IP stack, RAM, and an RF front end, so it can connect to an access point nearby, function as an access point by itself, or do both.[76]

Figure 6.12 ESP8266

AT Commands

The ESP8266's basic firmware, which enables it to process any AT commands that it receives over its Serial UART interface. The main benefit of this solution is that we don't need to be experts in any particular language or framework in order to use the module. To accomplish our objective, we only need to provide it with a series of orders. The drawback of this is that in order to deliver the required commands, we either need to use a USB to Serial adapter or another microcontroller [75].

Basic	
Command	**Description**
AT	Test AT startup
AT+RST	Restart module
AT+GMR	View version info
AT+GSLP	Enter deep-sleep mode
ATE	AT commands echo or not
AT+RESTORE	Factory Reset
AT+UART	UART configuration,
AT+UART_CUR	UART current configuration
AT+UART_DEF	UART default configuration, save to flash
AT+SLEEP	Sleep mode
AT+RFPOWER	Set maximum value of RF TX Power
AT+RFVDD	Set RF TX Power according to VDD33

Figure 6.13 AT Commands

6.1.7. ESP8266 Sequence of Operations

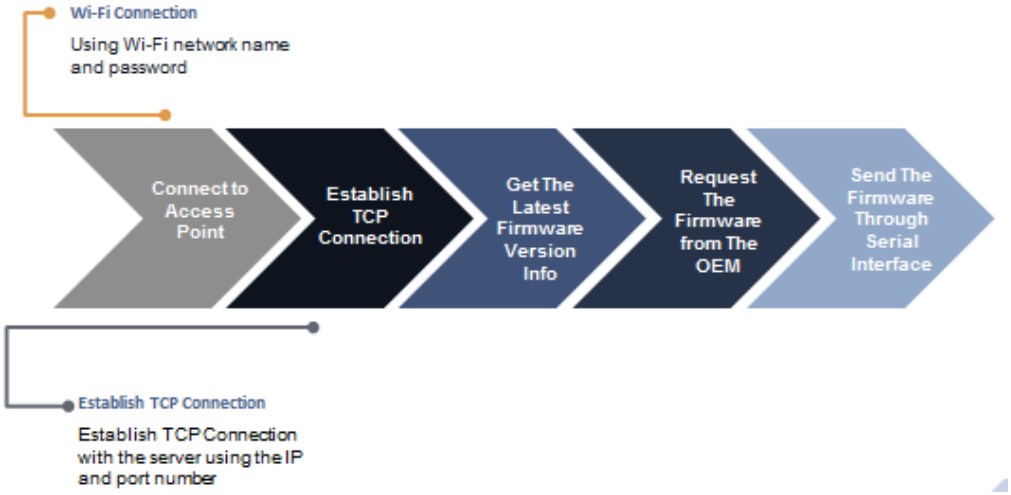

Figure 6.14 ESP sequence of operations

6.1.8. ESP8266 Hardware Connection

The ESP8266's firmware included an interrupter for AT instructions because it was designed as a Serial to Wi-Fi converter. We used ESP and connected it to the gateway

ECU which is responsible for the internet connection and OEM server transactions. The Gateway ECU controls the ESP using UART. If the user requested a new version of the server, the Main ECU performs a request and downloads this new version of the firmware from the OEM server via the Wi-Fi module and sends it via a CAN bus to the Target ECU (boot loader) that performs the flashing process.

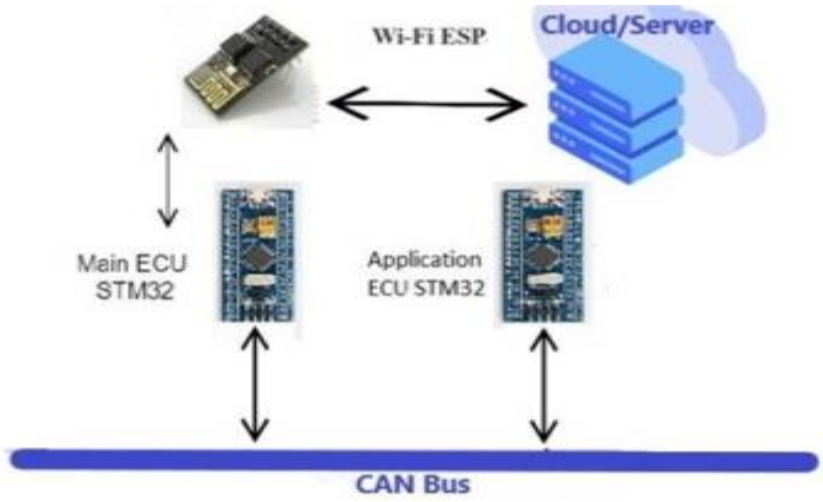

Figure 6.15 Internal connection

6.1.9. Data Integrity

Now we have a system that can check for new firmware copies on the OEM server, and if the user requests it, the system can make a request to the server and download the new firmware and send it to the boot loader. This flashes the firmware on the target ECU. But what if an error occurred during the process of transferring data to the firmware and an error occurred in one byte or bit?[79]. There must be an algorithm that is able to identify this error and report the error to the user and also act and make the system into a stable state.

Hex File Check Sum Byte

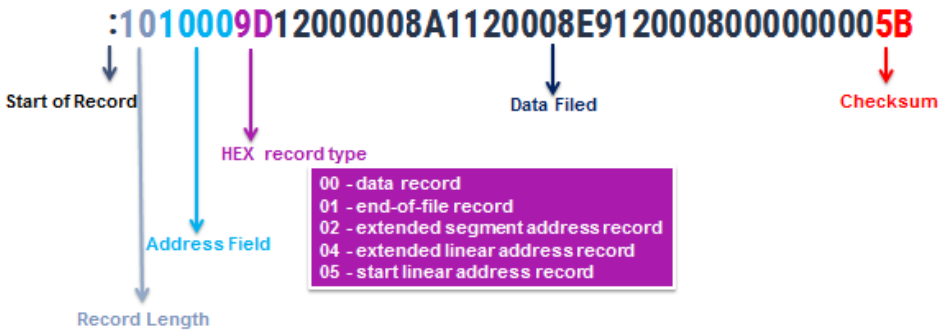

Figure 6.16 Hex file fields

As we said earlier, a hex file consists of a number of lines called records. Each record contains a byte called check sum, with which we can test and validate the incoming data for each record individually [77].

Check Sum Algorithm

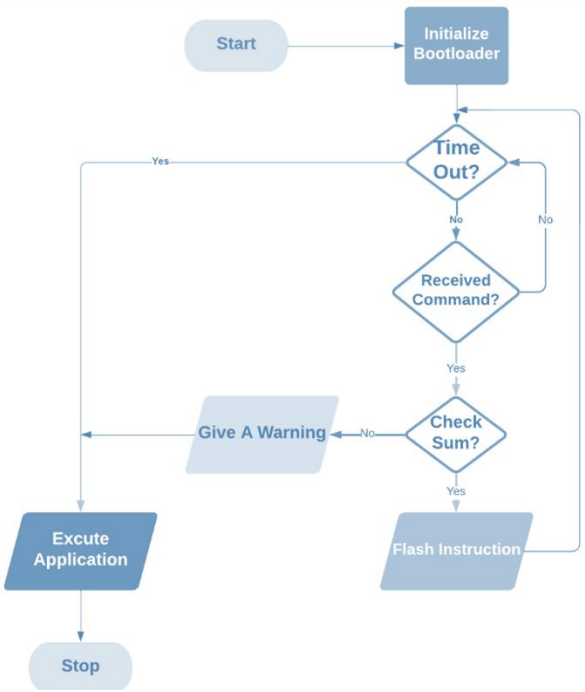

Figure 6.17 Checksum algorithm

If any error occurs in the received data, the system can identify this error immediately and give a warning of the occurrence of the error.

6.1.10. Recovery Mechanism

When the boot loader boots and starts receiving Hex file data for the new firmware that starts writing to flash memory, the boot loader must first clear the contents of the flash memory (previous firmware) to be able to write the new data to it.

What if there is a crash while installing software using the boot loader after erasing the contents of the flash memory? Like cutting off the power to the electronic control unit or an error that has been reported in the check sum algorithm.

What will happen after the boot loader part ends and the microprocessor jumps on the application part that has been erased?

In fact, the processor will implement undefined instructions, and this will make the system in a completely unstable state. And this is unacceptable.

That's why we came up with the banking technique.

Boot loader Banking

We changed the design of the old boot loader, which had two parts, one for the boot loader and the other for the application. In the event that the boot loader flashed a new application, it would erase the old application part and write on it.

Now the design is different, as it consists of three parts, a boot loader and two banks.

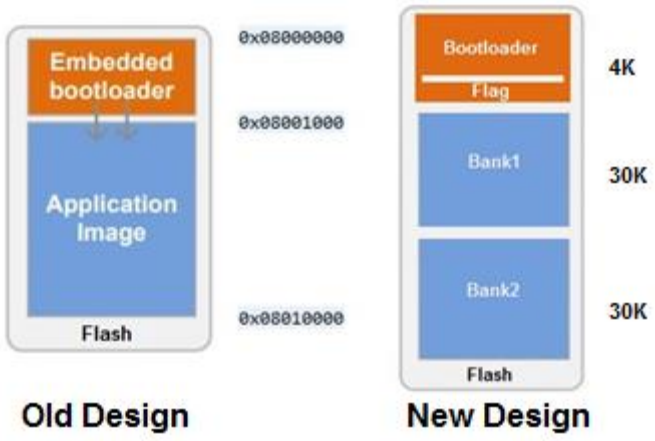

Figure 6.18 Banking design

One of the two banks is called a valid bank, which contains the current version of the application. The other is called Invalid bank, and it has been designated as a space for the new version of the application. When the firmware version is updated, the invalid bank is erased, and a new valid version is written to it and the two banks are switched so that the bank containing the old version becomes the invalid bank and the other with the new version is valid bank.

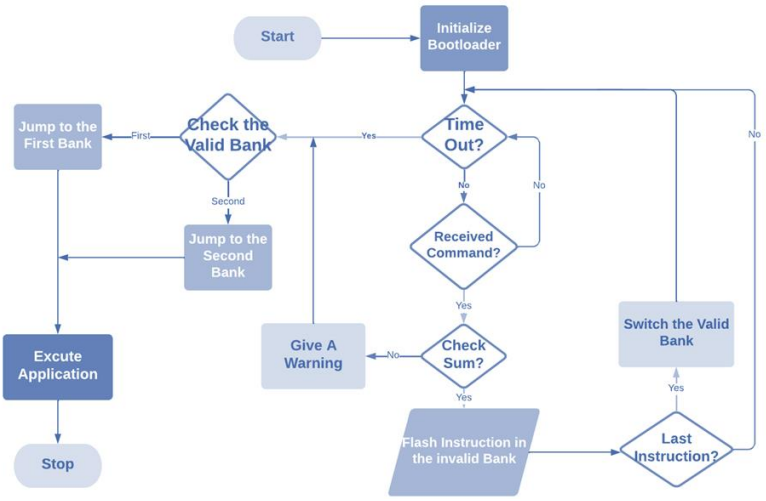

Figure 6.19 Recovery algorithm

This means that if any error occurs while downloading the new firmware, the current version will be safe, and the boot loader can identify this error and return it to the valid bank, since its contents have not been erased.

6.2. Website

We have a problem when we need to update the new firmware, we need a lot of physical connection between the MAS and the machine that has the update of the new firmware, we need to update the new firmware and installation it from anywhere without go to the maintenance centre, like a smartphone when there is a new update, we need internet only to instillation this update from any place.

To achieve this goal, we need something our MAS have the access of it and communicate to update the new firmware without any physical connection, so we decided to use the web server.

In web server, we can upload the new firmware and the MAS can request server to download this update from anywhere and therefore, you do not need to go to maintenance centres.

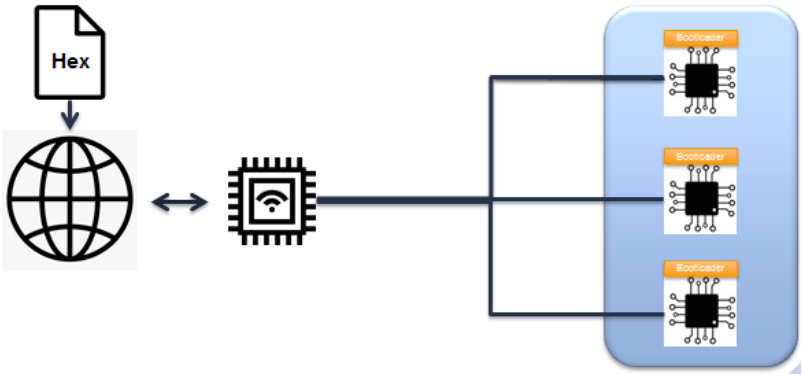

Figure 6.20 IOT internal connection

To design web server, we pass through three stages, front end, back end and make web server global, we will discuss in detail.

Front end

Front-end refers to the user interface and to anything that you see in a browser when view a website, Front-end developers design and construct the user experience elements on the web page including buttons, menus, pages, links, graphics and more.

To design our front-end web server, we are using a combination of technologies such as Hypertext Mark-up Language (HTML), Cascading Style Sheet (CSS), Bootstrap, JavaScript (JS), jQuery.

HTML

What is language html?

HTML is a markup language that defines the structure of your content. HTML consists of a series of element, which you use to enclose, or wrap, different parts of the content to make it appear a certain way or act a certain way. The enclosing tags can make a word or image hyperlink to somewhere else, can italicize words, can make the font bigger or smaller [19].

Example Explained to the element of HTML

1. The <html> element is the root element of an HTML page.
2. The <head> element contains meta information about the HTML page.
3. The <title> element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab).
4. The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
5. The <h1> element defines a large heading.
6. The <p> element defines a paragraph.

In our web server we are using HTML to write all text.

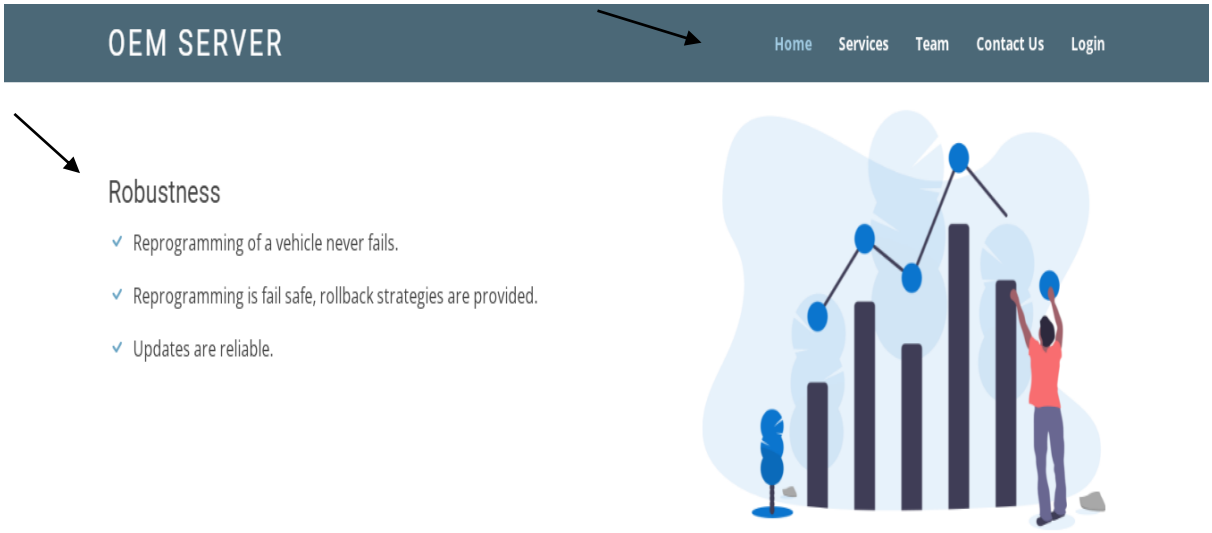

Figure 6.21 Web server

CSS

What is language CSS?

CSS (Cascading Style Sheets) is used to style and layout web pages for example, to alter the font, colour, size, and spacing of your content, split it into multiple columns, or add animations and other decorative features. This module provides a gentle beginning to your path towards CSS mastery with the basics of how it works, what the syntax looks like, and how you can start using it to add styling to HTML [20].

In our web server we are using CSS to style and layout web pages and control in the font, colour, and size.

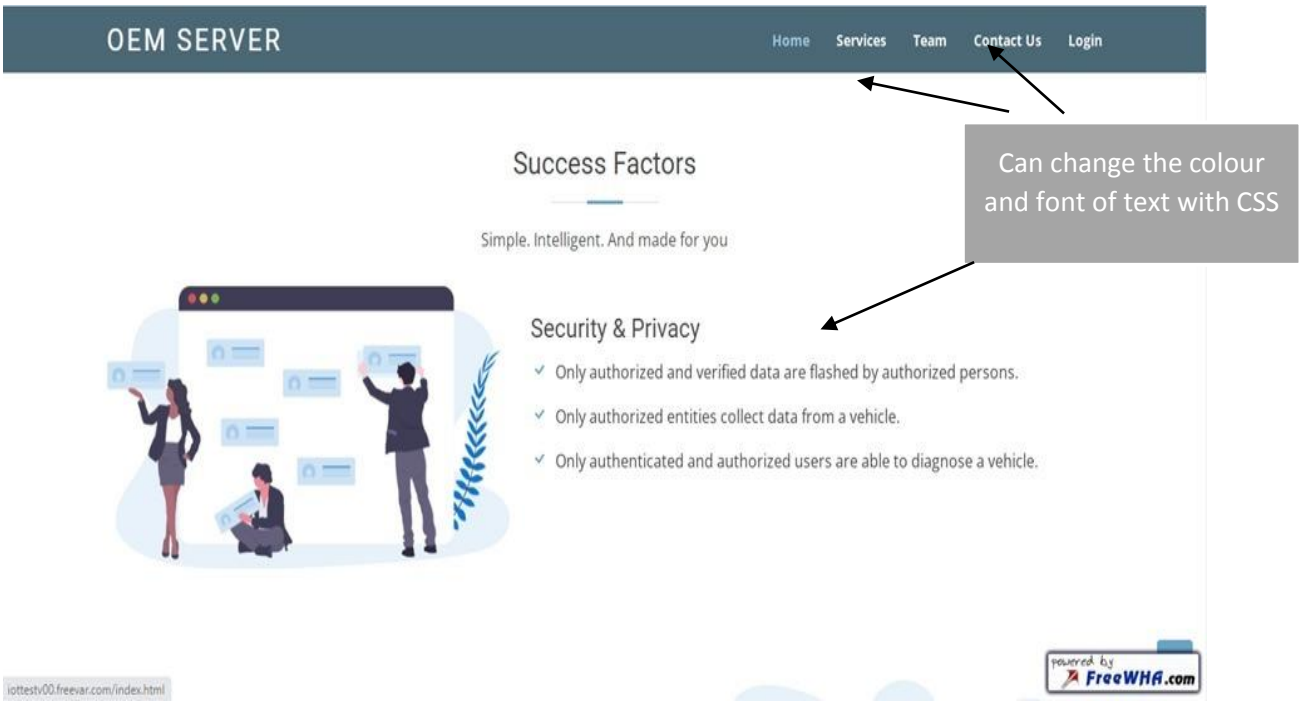

Figure 6.22 OEM server

Bootstrap

What is Bootstrap?

Bootstrap is the most popular HTML, CSS, and JavaScript framework for developing a responsive and mobile friendly website, it is framework used for easier and faster web development.

It includes HTML and CSS based design templates for typography, forms, buttons, tables, navigation, modals, image carousels and many others it can also use JavaScript plug-ins, it facilitates you to create responsive designs.

In our web server, we use the bootstrap which have a lot of templates to build card as shown in Figure below, it was possible to use HTML and CSS to build this card, but it is more complex, and bootstrap make it easier.

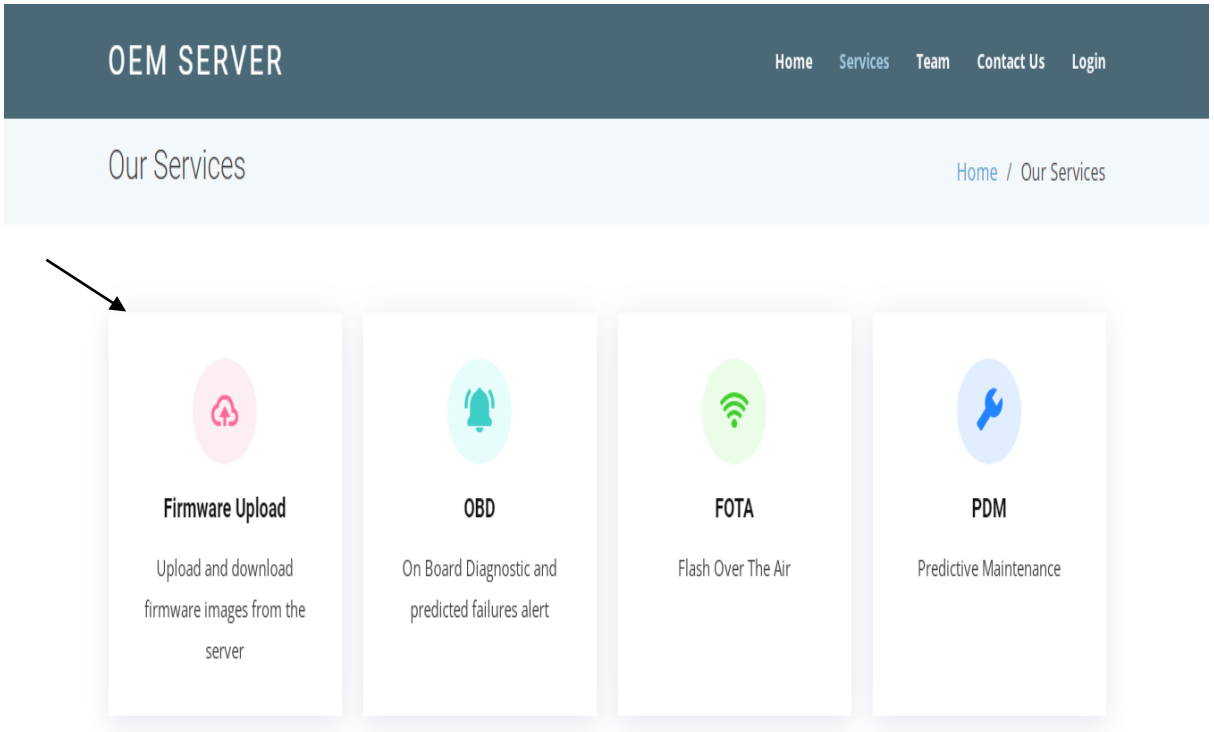

Figure 6.23 services page

The user can choose which service he wants to use.

JavaScript

What is JavaScript?

JavaScript is a lightweight, interpreted, object-oriented language with first-class functions, and is best known as the scripting language for Web pages, but it's used in many non-browser environments as well. It is a prototype-based, multi-paradigm scripting language that is dynamic, and supports object-oriented, imperative, and functional programming styles.

JavaScript runs on the client side of the web, which can be used to design or program how the web pages behave on the occurrence of an event. JavaScript is an easy to learn and powerful scripting language, widely used for controlling web page behaviour [21].

In our web server, we use JavaScript to add photo which make the web server more beautiful, and JavaScript can make slider as shown in Figure below.

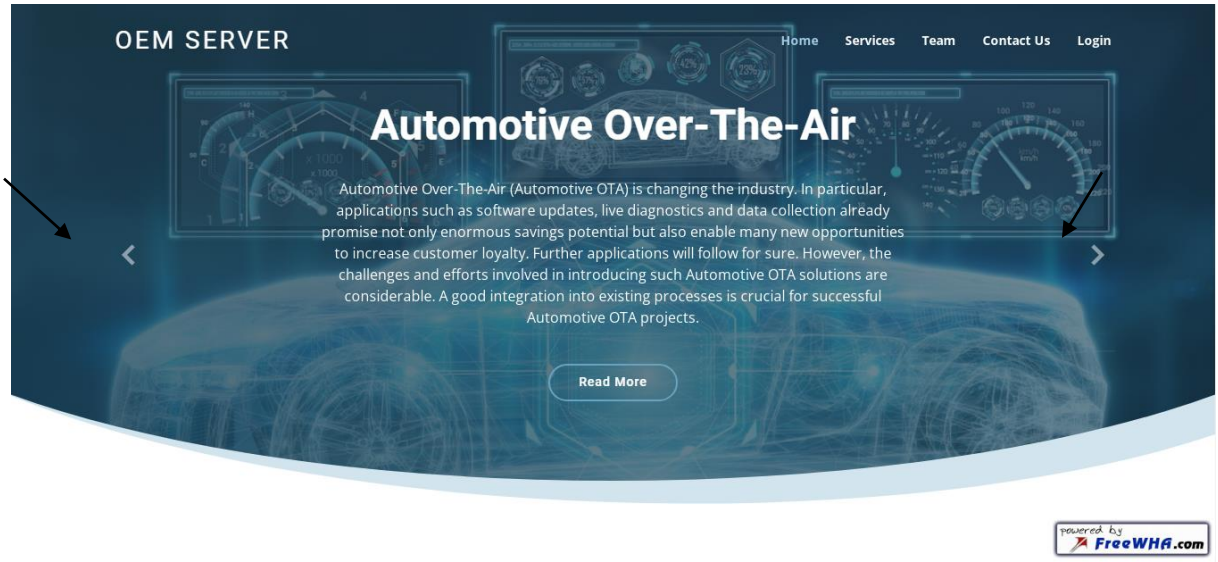

Figure 6.24 Home page

Details of the web server and the services that can user be able to use it.

jQuery

What is jQuery?

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript [22].

In our web server, we use jQuery because the bootstrap depends on it, jQuery make the JavaScript faster and easier to make slider.

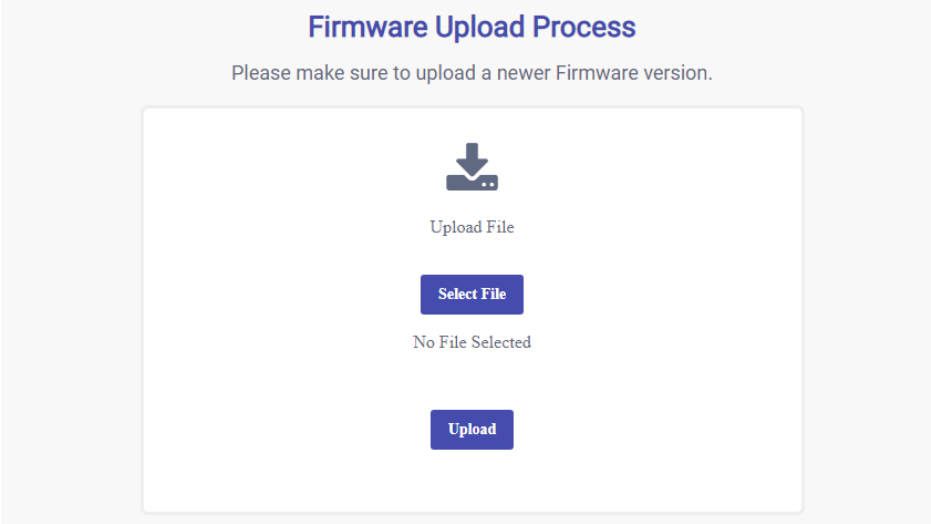

Figure 6.25 Firmware upload page

The Engineer of OEM can upload the new firmware update on the web service which the MAS can be able to request server and download the new firmware update.

Back end

The back end means the server, application and database that work behind the scenes to deliver information to the user, the user enters a request through the interface, it is consists of the server which provides data on request, the application that channels it, and the database which organizes the information is kept on the server.

To design our back-end web server, we are using a PHP.

PHP

What is PHP?

PHP mean for PHP: Hypertext Pre-processor, it is a widely used, open-source scripting language, scripts are executed on the server, it is especially suited for web development, PHP can collect form data, add, delete, modify data in your database, be used to control user-access, it can support a wide range of databases [23].

In our web server, we use PHP to link the data that the user enters on user interface to the web server database and if user make sign up the PHP is responsible to send this data to web server data base and save it as shown in Figure below.

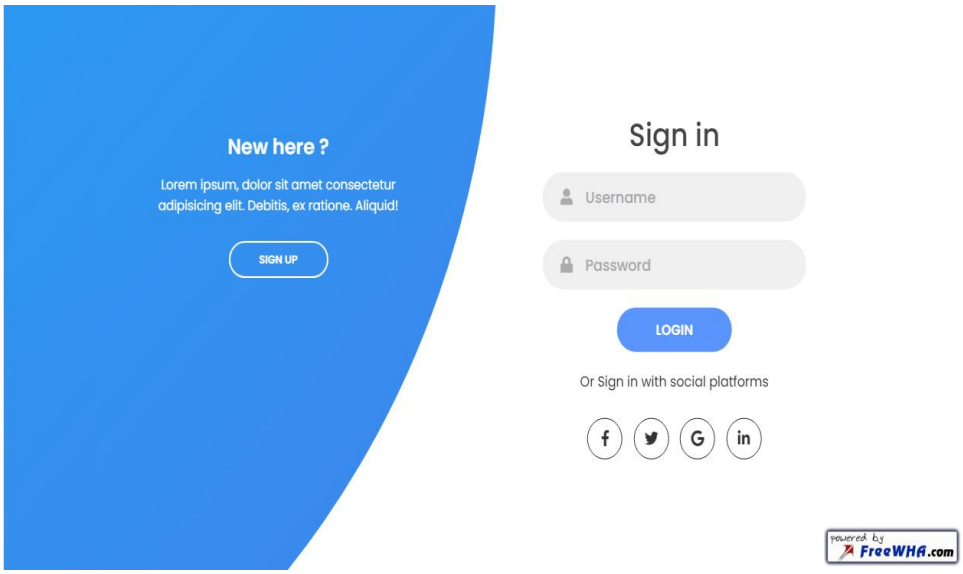

Figure 6.26 Sign page

Back-end flowchart

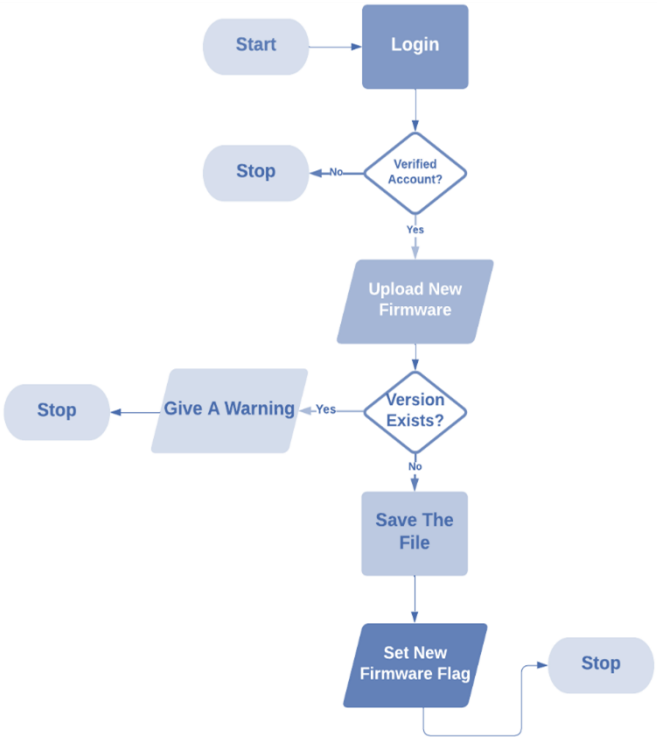

Figure 6.27 Back-end flowchart

Step of back end

1. The user must be logged in to be able access to release the firmware.
2. The system checks if this user is authorized to upload the new firmware.
3. The user can upload the hex file on the web server.
4. The system checks if this version is on the server, if we found it on server, the system warning user to get the latest firmware version, if we don't find it on the server, we got the latest version of firmware.
5. We will save the new firmware on the server.
6. Set flag to indicate that we have a new firmware update version on the web server, to the MAS knows and makes a request.

Web server global

We have a problem the web server on the computer which developed it, we need to make it public, and any user can browse the web server, we have two tools to convert the web server from private to public, FileZilla, and free web hosting area.

FileZilla

What is FileZilla?

The FileZilla software program is a free-to-use open-source FTP utility, allowing a user to transfer files from a local computer to a remote computer. FileZilla is available as a client version and a server version.

In our web server, we are using FileZilla to transfer the date of web server in our computer to the web in browser.

Free web hosting area

What is Free web hosting area?

Free web hosting area is a service that can servers connected on the internet, it allowing to user, organizations, and company to serve content or host services connected to the Internet.

A common kind of hosting is web hosting. Most hosting providers offer a combination of services - e-mail hosting, website hosting, and database hosting.

In our web server, the free web hosting area is responsible to make our web server is public to any user to browser it.

6.3. HMI ECU

The controller used in this ECU is Raspberry Pi 3 model B+.

Figure 6.28 Raspberry Pi

The Raspberry Pi hardware has evolved through several versions that feature variations in the type of the central processing unit, amount of memory capacity, networking support, and peripheral- device support, which help in developing different application programming interfaces (APIs), using python and Linux OS with its Raspbian system makes it easy to install different libraries and work through different structures.

6.3.1. How To Connect Raspberry Pi to CAN Bus

We build our GUI system using touch screen connected to raspberry pi so that we can tell the user that there is a new update and ask him to accept the update requests or refuse them and show the progress of the updating process if the user accepts the update. Also, for receiving data from sensors and alert users if there is a predicted failure to occur. Because we want all ECUs in our project to be connected to the same CAN network, we use SPI TO CAN module because Raspberry Pi doesn't have a built-in CAN Bus, but its GPIO includes SPI Bus, that is supported by large number of CAN controllers. So, we will use a bridge between Raspberry Pi and CAN Bus which is SPI Bus.

SPI TO CAN Module consists of CAN controller (**MCP2515**) because CAN Bus is a multi- master protocol, each node needs a controller to manage its data. And CAN transceiver (**MCP2551**) because CAN controller needs a send/receive chip to adapt signals to CAN Bus levels. Controller and Transceiver are connected by two wires TxD and RxD.

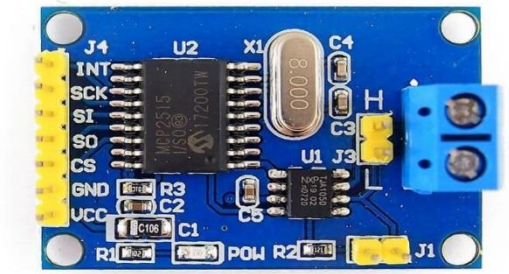

Figure 6.29 SPI to CAN module

So, we will show how we connected our Raspberry Pi to SPI to CAN Module.

SPI to CAN Module has 5 connections.

- MOSI (Master Out Slave In).
- MISO (Master in Slave Out).
- SCLK (Serial Clock).
- CS or SS (Chip Select, Slave Select) to enable and disable the chip
- INT (Interruption) to interrupt the chip when needed

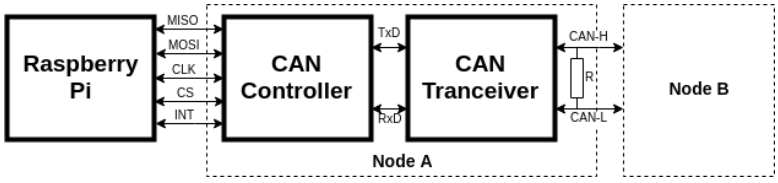

Figure 6.30 Raspberry pi connection

6.3.2. GUI Design

Our GUI contains 3 main files

1. MAS.py: The whole flow and interaction of the GUI between its main parts and between the GUI and the other components & processes of the project as we are going to mention these main tasks.
2. MainWindow.py: it includes over 2000 lines of CSS code, it is used to control the appearance of the GUI, Widgets, shapes, and organization of the design.
3. Image.qrc: this file contains all media used in the GUI such as images and icons, which is transformed to .py to be read.

It includes:

- Signaling and slots concept.
- Virtual lining between tabs and widgets
- Receiving different kinds of text to appear on the GUI
- Sending and Receiving CAN messages
- Security is handled by using a process of log in and sign up.

6.3.3. GUI Description and Features

Here we will discuss the design of our GUI we have written our GUI using Python (PyQt5). This is the Home Screen of our GUI.

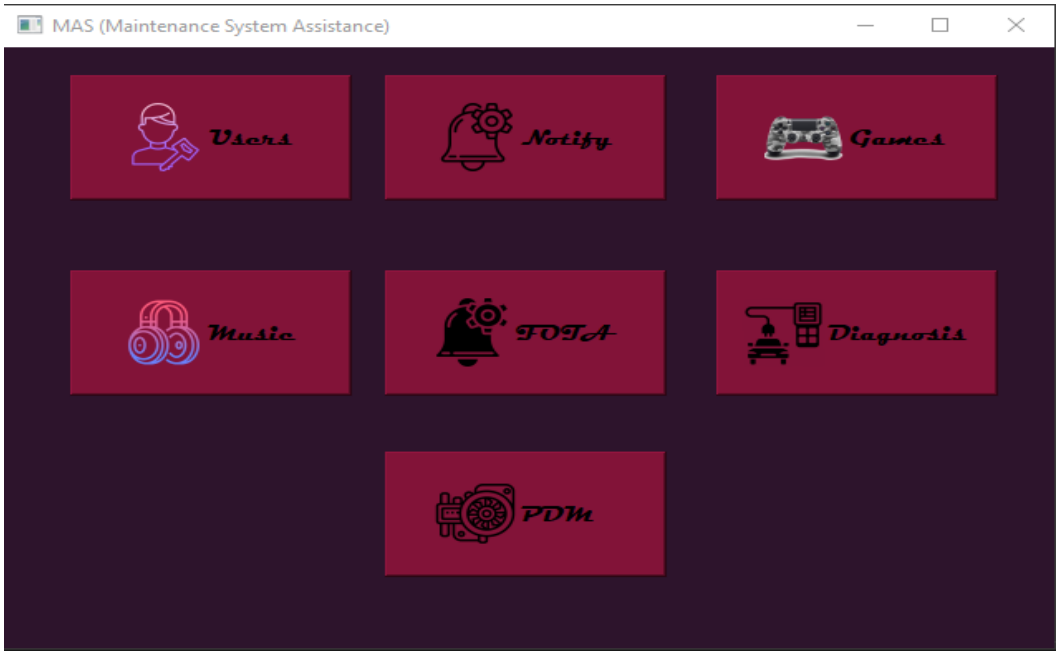

Figure 6.31 Home Screen

We have focused on four tabs

5. Users: it is responsible for sign in and sign up.
6. FOTA: it is responsible for updates.
7. Diagnosis: it is responsible for receiving data from car's engine.
8. PDM: alerting users in case of predicted failure will happen in future.

6.3.3.1. User Sign In and Sign Up

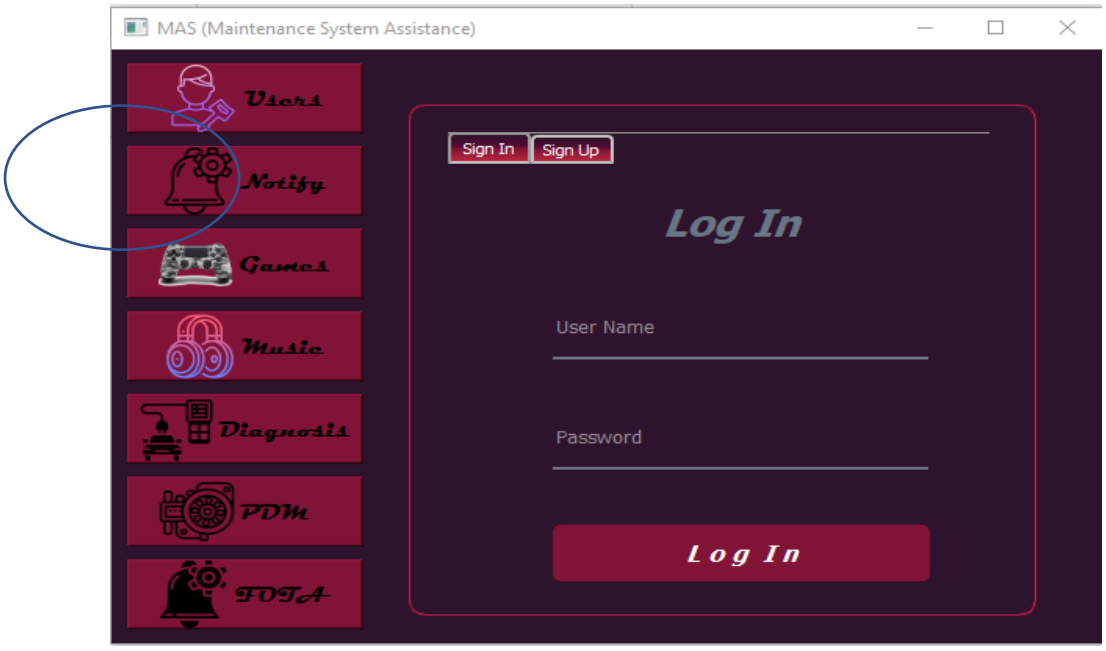

Figure 6.32 Sign In

Here as we can see, this is the user tab where the user can login with his username and password fields.

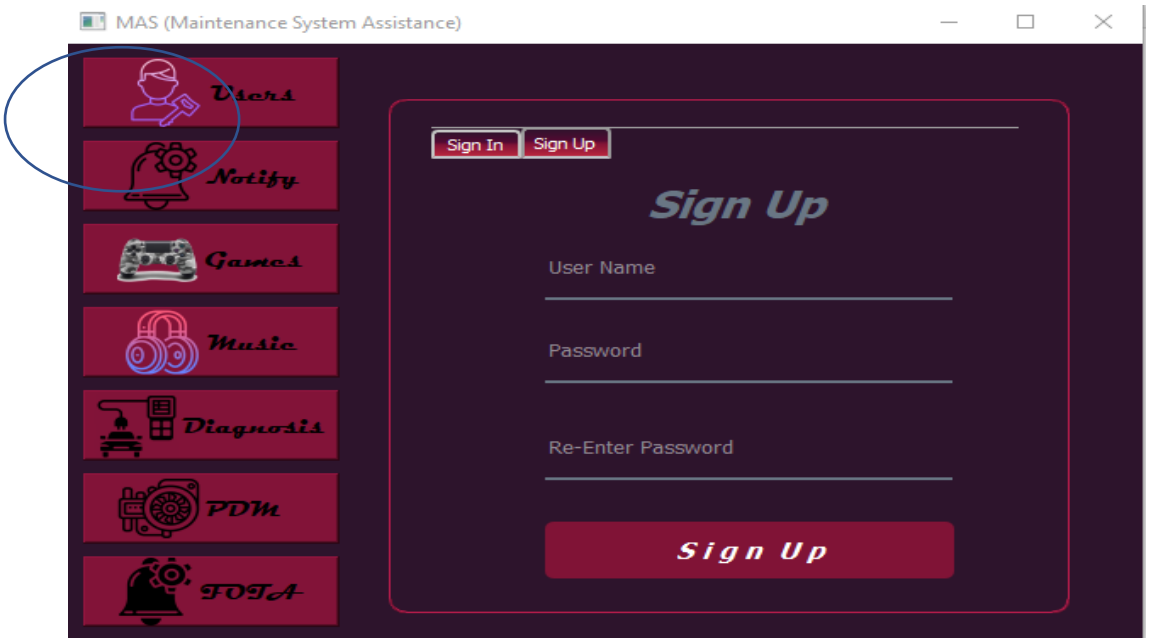

Figure 6.33 Sign up

Here as we can see, sign up tab where 3 fields are existing to serve a new user to enter his name and password twice for safety.

6.3.3.2. Engine Diagnosis

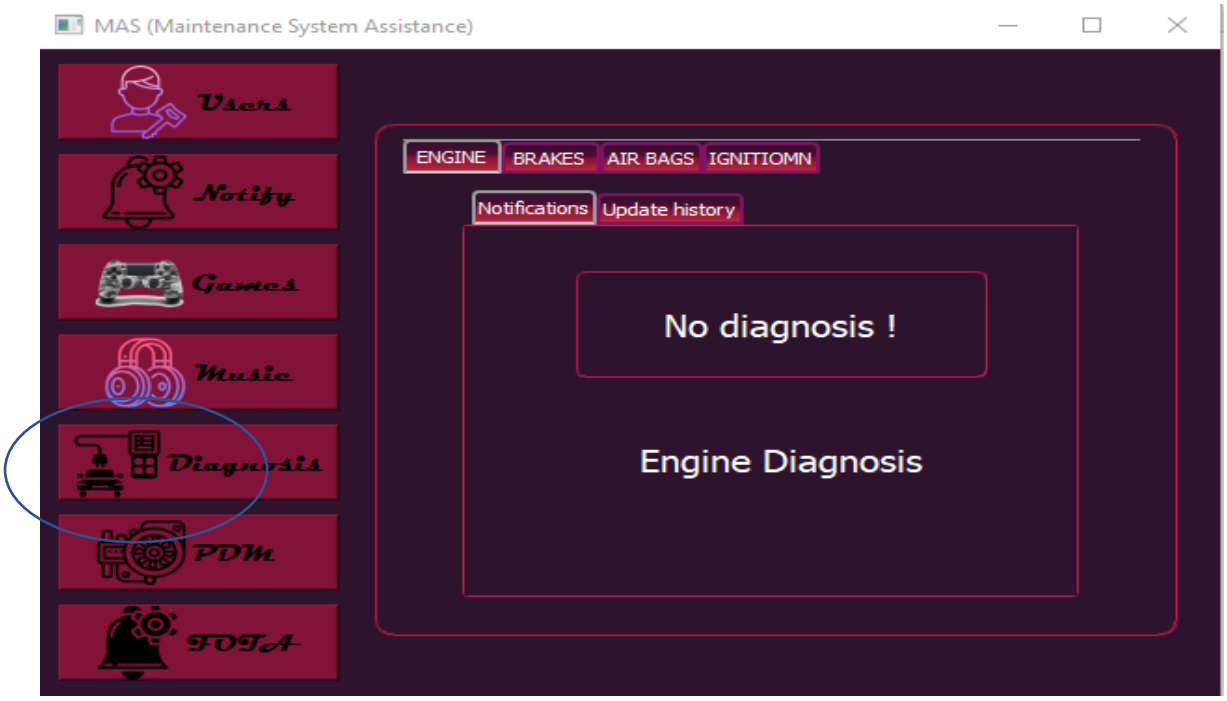

Figure 6.34 Engine Diagnosis

The user is shown for each part of the car when data is collected from sensors by the Target ECU and sent over the CAN bus to the HMI ECU to be processed. And then this data will be stored in an excel sheet and entered into the predictive maintenance model to be processed. But we have focused on the engine only as proof of concept. And after collecting data it can be sent to OEM server.

6.3.3.3. PDM (Predictive Maintenance)

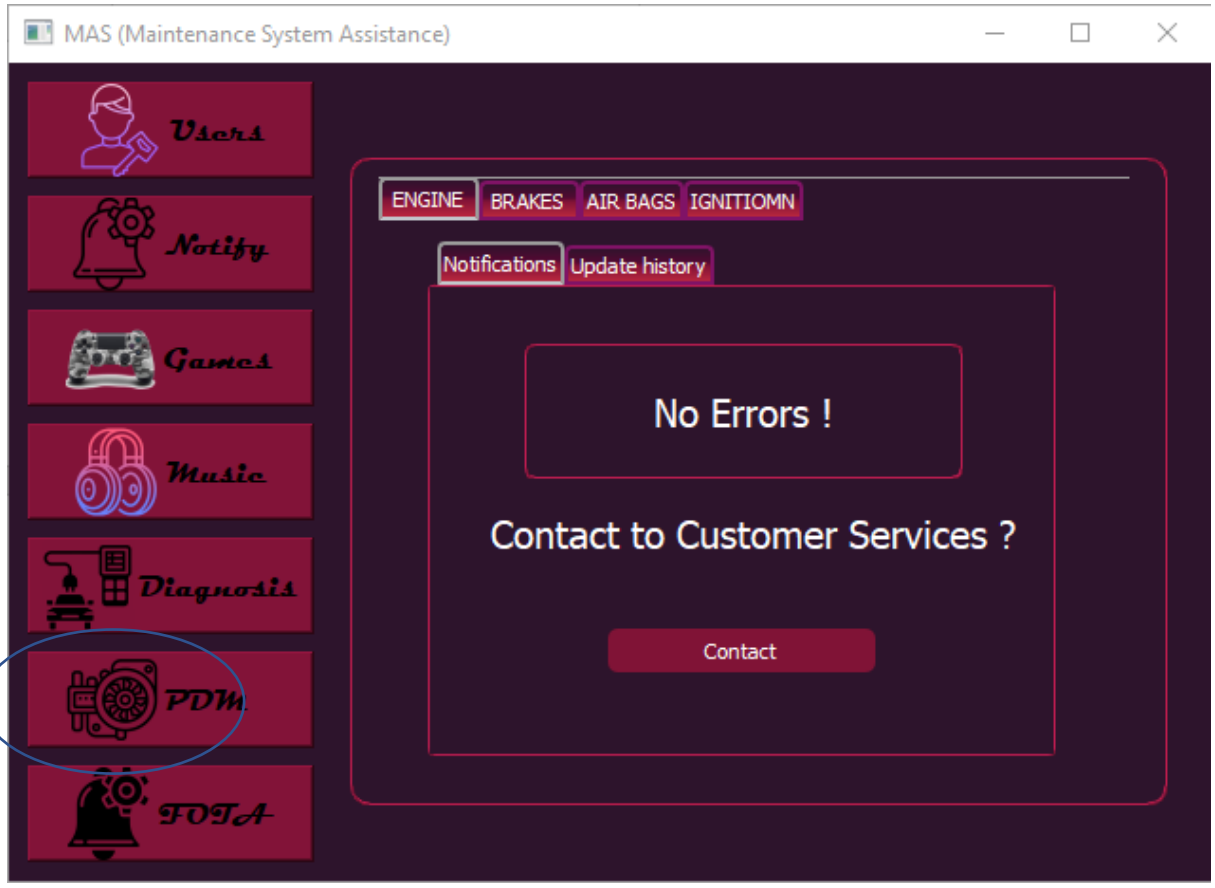

Figure 6.35 PDM

The user is shown for each part of the car if there is an error is predicted to occur by the machine learning model will alert user in this tab and user will be able to send code of error to OEM server. But we have focused on the engine only as proof of concept.

Here is a flow chart of how part of diagnosis and predictive maintenance works.

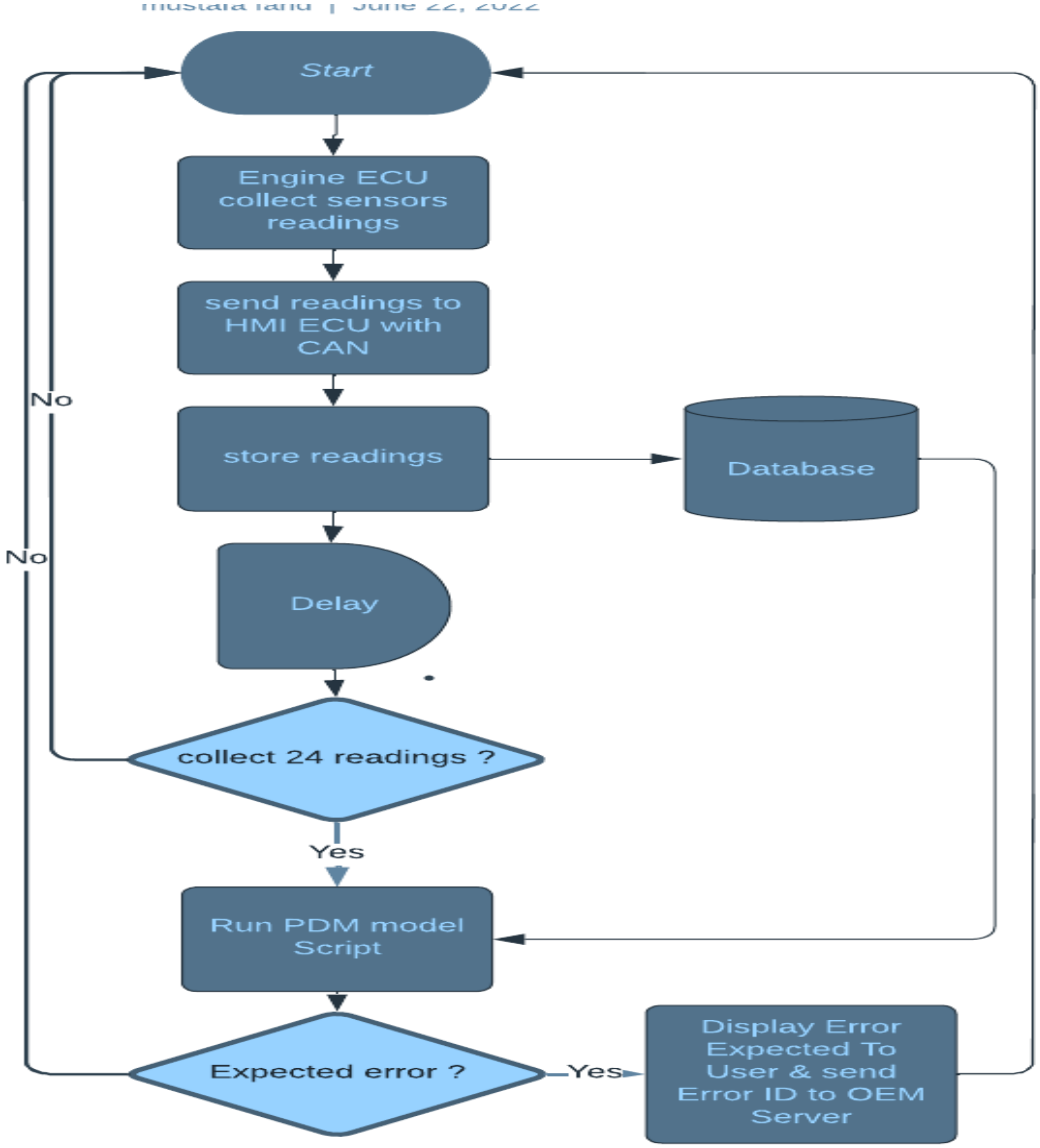

Figure 6.36 PDM flow chart

One of the ECUs is the Engine ECU, which is responsible for collecting data from the engine sensors. It will collect data each hour during the day. The data is sent over the CAN network to the Raspberry Pi, on which predictive maintenance operates. The PMS will store and process this data and based on it, will inform the user if there is a failure that is expected to occur through the user interface system.

This will enable us to predict hardware failures before they occur and alert the driver through the user interface long enough to allow them to go to the service center and take the necessary actions.

6.3.3.4. FOTA

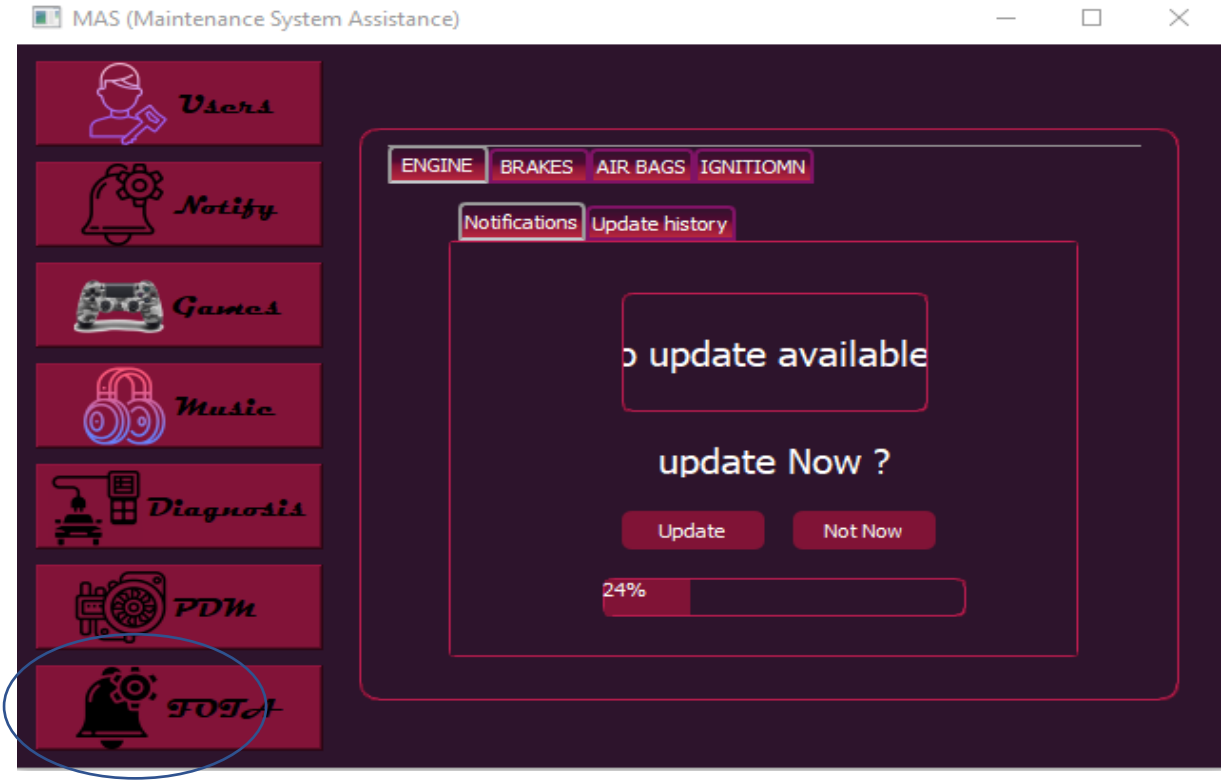

Figure 6.37 FOTA notifications tab

Here as we can see, our key role the FOTA main tab which contains our Notifications tab & and Updates history tab, The notification tab contains the different ECUs updates which the user can check it any time there is a text box which will allow the user to receive its update message and asking the user whether yes or no for the update with 2 push buttons, And a loading bar to indicate the percentage completed of update.

Flow chart of update over the air

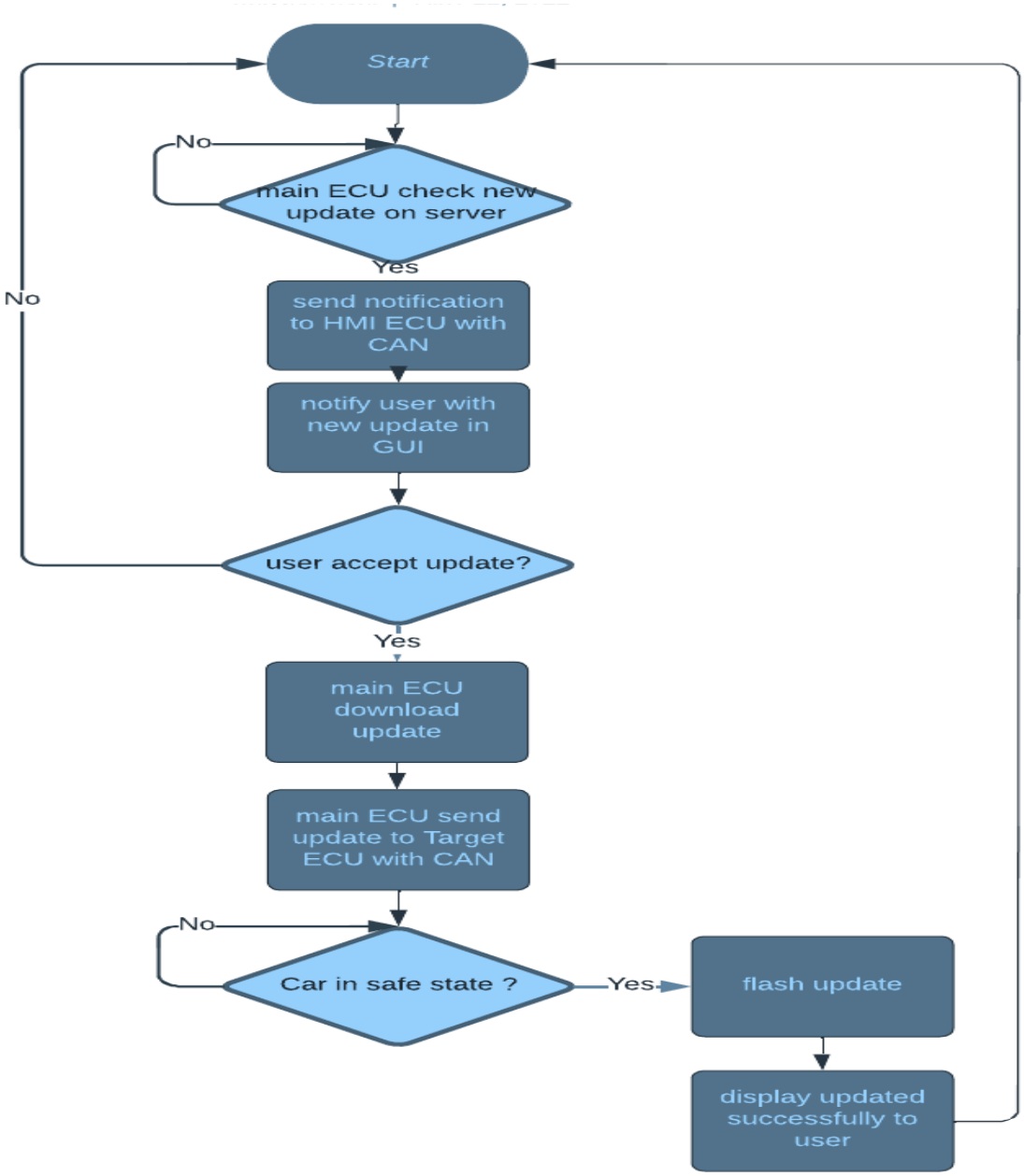

Figure 6.38 FOTA flow chart

One of the ECUs is known as the main ECU, and it is linked to the ESP8266 WIFI module, which will constantly check the OEM server for new versions of updates. If a new update is available on the server, the main ECU will send a notification to the HMI ECU via the CAN bus, and the user will be notified via the GUI. If the user accepts the update, the HMI ECU will send this acceptance to the main ECU, and the main ECU will download the new update. If the car is in a safe state (not moving), the main ECU will send the update to the target ECU, which will flash the update to its flash memory. and will display that the update occurred successfully in the GUI to the user.

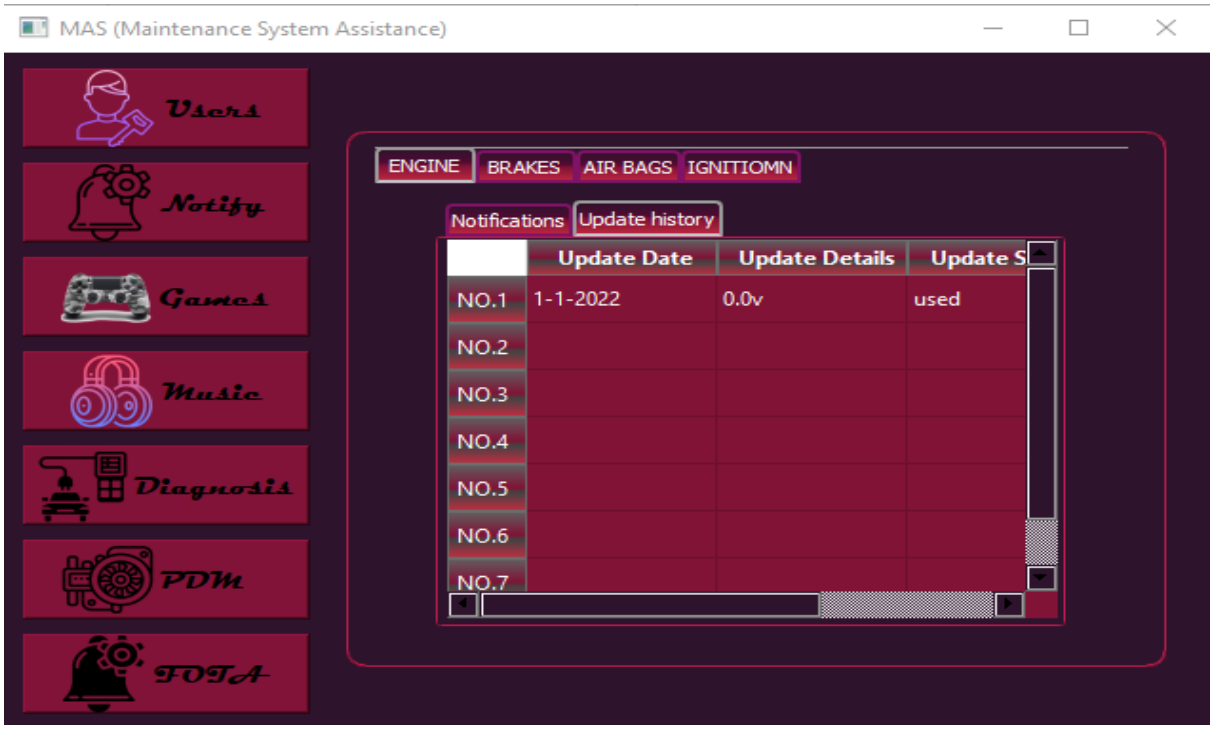

Figure 6.39 FOTA update history

Here, as we can see, the updates history tab which handles the history of the updates and its details in order for the user. And user can choose previous version to roll back if there's a problem or bug in last version installed.

The rollback flow chart is shown in the Figure below.

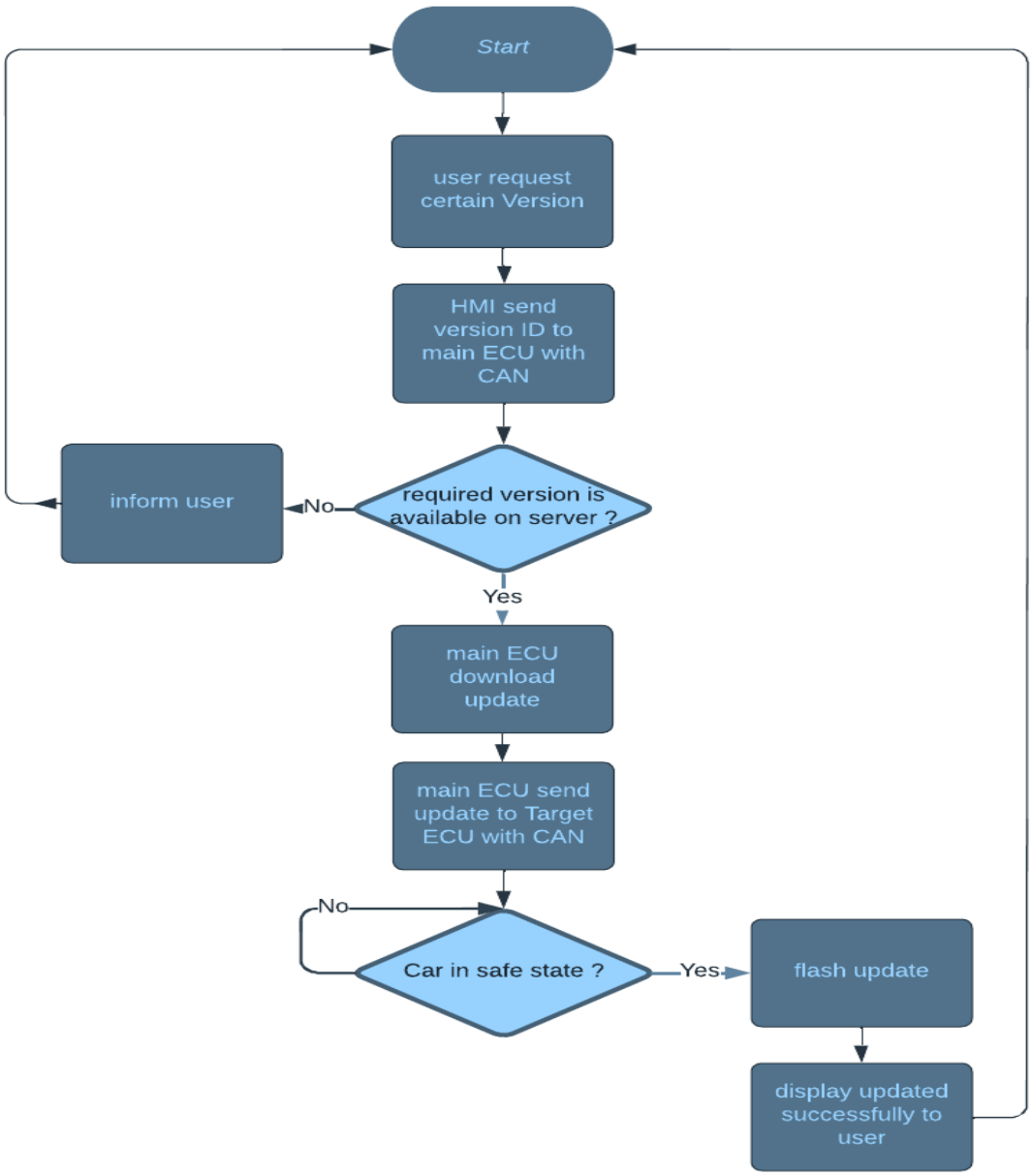

Figure 6.40 Roll back flow chart

If the user doesn't like it or there is a bug in the current software version, the user can roll back to the previous software version of the specified ECU by choosing the version of software in the GUI. It will be sent by HMI ECU on a CAN bus to the main ECU. If this chosen version is available on the server, it will be downloaded. The main ECU will send the update to the target ECU, which will flash the update to its flash memory. And will display that the update occurred successfully in the GUI to the user.

Chapter 7 : Conclusion

7.1 Results

- We have developed an entire system that has the ability to deal with all sudden failures and breakdown times that may happen to the vehicle.
- We Designed and Implemented Predictive maintenance system the uses Machine learning algorithms to predict failures in critical mechanical parts in a vehicle before they occur and issue a warning to prevent them and provide car owners with a better maintenance plan.
- We proposed approach uses unique features such as voltage, pressure, vibration, rotation, machine age, error type, number of components, model type, and failure as inputs within a full time-sequence of the data.
- We built the FOTA System to offer the ability to install bug fixes and update the vehicle's software without having to visit the service center.
- We have specially designed the system with a performance monitoring system for critical vehicle parts using dedicated sensors
- We have made it possible for the user to stay in touch with his fleet of vehicles and offer real-time diagnostics via the Internet of Things.
- We have ensured stable system performance by offering the ability to recover from any error in the current firmware version and the ability to roll back to any previous version.
- In this project we proposed a predictive maintenance system (PdM) based on a hybrid CNN-LSTM model for multivariate time-series classification.
- CNN extracts the features from the input data, and LSTM learns from those extracted features. Together the hybrid model is able to capture the long-term and short-term data patterns for robust predictions.
- The results show that the model achieves an average of 91.6% Precision, 93.4% Recall, and 92.48% F-Score on the testing dataset

7.2 Future Work

1. Implement a full Diagnostic system by applying a predictive maintenance system to all critical parts of the vehicle. Like the battery, brakes, and tires.
2. Provide a two-way diagnostics utility in which the OEM provides an Over the air bug fixes, and the car owner allows the Critical part monitor data to be shared for enhancement purposes of MAS.
3. Invest more in a secure FOTA system and expand the mission of protecting transmitted data.
4. Re-Integrate the system using Autosar Layered Software Architecture.

Figure 7.1 Future work

References

[1] Wang, W.; Chen, H.; Bell, M.C. (2005) "Vehicle Breakdown Duration Modeling", J. Transp. Stat. 2005.

[2] National Highway Traffic Safety Administration: Annual United States

Road Crash Statistics. 2018.

[3] Sai Chand, Emily Moylan, S. Travis Waller and Vinayak Dixit. "Analysis of Vehicle Breakdown Frequency: A Case Study of New South Wales", Australia, 2020

[4] United Nation International Telecommunications Union - ITU, "The Internet of Things", ITU Strategy and Policy Unit (SPU), Geneva, Switzerland, November 2005

[5] Jeannette China, *, Vic Callaghanb and Somaya Ben Allouchc "The Internet of Things: Reflections on the Past, Present and Future from a User Centered and Smart Environment Perspective"

[6] Chin J, Callaghan V., "Embedded-Internet Devices: A Means of Realising the Pervasive Computing Vision", Proceedings of the IADIS International Conference WWW/Internet 2003, ICWI 2003, Algarve Portugal, 2003

[7] V. Callaghan (editor), "Intelligent Embedded Agents", Journal of Information Sciences – Special issue: Intelligent embedded agents. Volume 171 Issue 4, 12 May 2005

[8] ETSI Technical Specification, "Machine-to-Machine Communications (M2M); M2M Service Requirements", Technical Specification. ETSI TS 102 689, V1.1.1(2010-08)

[9] Jeannette Chin and Vic Callaghan, "A Show Me by Example Approach to Teaching Programming the Internet-of-Things in Immersive", 2nd European Immersive Education Summit, Paris, 26th to 27th November 2012

[10] Jan Bauwens, Peter Ruckebusch, Spilios Giannoulis, Ingrid Moerman, and Eli De Poorter "Over-the-Air Software Updates on the Internet-of-Things: An Overview of Key Principles"

[11] "Cancheck.- Easy to Use Software for CAN Device Detection and Diagnosis", Comes with USB Keypro. Converts CAN Messages and Eliminates the Hassle of Wading through Logs and Manually Interpreting Results. Supports Math Functions, https://www.icpdas-usa.com/cancheck.html.

[12] Di Natale, Marco, et al. "Understanding and using the controller area network communication protocol: theory and practice", Springer Science & Business Media, 2012.

[13] HPL, Steve Corrigan. "Introduction to the controller area network (CAN)", Application Report SLOA101 (2002): 1-17.

[14] "STM32F107xx advanced ARM-based 32-bit MCUs", Reference manual. STMicroelectronics (2010).

[15] “Controller Area Network (CAN Bus) - Message Frame Architecture”, Copperhill, https://copperhilltech.com/blog/controller-area-network-can-bus-message-frame-architecture/.

[16] Polo, Guilherme. "PyGTK, PyQT, Tkinter and wxPython comparison", 2017.

[17] Meier, Burkhard. "Python GUI Programming Cookbook: Develop functional and responsive user interfaces with tkinter and PyQt5", Packt Publishing Ltd, 2019.

[18] “All Classes”, Qt Documentation, https://doc.qt.io/qt-5/classes.html.

[19] developer.mozilla.org/en-US/docs/Web/HTML

[20] "CSS developer guide". MDN Web Docs. Archived from the original on 2015-09-25. Retrieved 2015-09-24

[21] javatpoint.com/javascript-tutorial

[22] jquery.com

[23] w3schools.com/php/php_intro.asp

[24] "Appendix D. MCS-86 Absolute Object File Formats: Hexadecimal Object File Format". 8086 Family Utilities - User's Guide for 8080/8085-Based Development Systems.

[25] “Hexadecimal Object File Format Specification”. Revision A. Intel. 1998 [1988-01-06].

[26] C. Chen, Y. Liu, X. Sun, C. D. Cairano-Gilfedder, and S. Titmus, “Automobile maintenance prediction using deep learning with GIS data”, Procedia CIRP, vol. 81, pp. 447-452, 2019, doi: 10.1016/j.procir.2019.03.077.

[27] A. Van Horenbeek, and L. Pintelon, “A dynamic predictive maintenance policy for complex multi-component systems”, Reliability Engineering & System Safety, vol. 120, pp. 39-50, 2013, doi: 10.1016/j.ress.2013.02.029.

[28] G. A. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi, “Machine learning for predictive maintenance: A multiple classifier approach”, IEEE Transactions on Industrial Informatics, vol. 11, no. 3, pp. 812-820, 2014, doi: 10.1109/TII.2014.2349359.

[29] T. Rieger, S. Regier, I. Stengel, and N. L. Clarke, “Fast Predictive Maintenance in Industrial Internet of Things (IIoT) with Deep Learning (DL): A Review”, CERC, 2019, pp. 69-80.

[30] B. Hrnjica, and S. Softic, “Explainable AI in Manufacturing: A Predictive Maintenance Case Study”, IFIP International Conference on Advances in Production Management Systems, 2020, pp. 66-73, doi: 10.1007/978-3-030-57997-5_8.

[31] K. Wang, and Y. Wang, “How AI affects the future predictive maintenance: a primer of deep learning”, International Workshop of Advanced Manufacturing and Automation, vol. 451, 2017, pp. 1-9, doi: 10.1007/978-981-10-5768-7_1.

[32] Mobley, R.K. "An Introduction to Predictive Maintenance", Elsevier: Amsterdam, The Netherlands, 2002.

[33] Ur Rehman, M.H.; Ahmed, E.; Yaqoob, I.; Hashem, I.A.T.; Imran, M.; Ahmad, S. "Big data analytics in industrial IoT using a concentric computing model", IEEE Commun. Mag. 2018, 56, 37–43.

[34] Abdul Wahid 1, *, John G. Breslin 1 and Muhammad Ali Intizar "Prediction of Machine Failure in Industry 4.0: A Hybrid CNN-LSTM Framework".

[35]. Zhao, R.; Yan, R.; Chen, Z.; Mao, K.; Wang, P.; Gao, R.X. "Deep learning, and its applications to machine health monitoring", Mech. Syst. Signal Process, 2019, 115, 213–237. [CrossRef]

[36]. Khan, S.; Yairi, T. "A review on the application of deep learning in system health management", Mech. Syst. Signal Process. 2018, 107, 241–265. [CrossRef]

[37]. Remadna, I.; Terrissa, S.L.; Zemouri, R.; Ayad, S. "An overview on the deep learning-based prognostic", In Proceedings of the 2018 International Conference on Advanced Systems and Electric Technologies IC_ASET), Hammamet, Tunisia, 22–25 March 2018.

[38]. Sun, W.; Shao, S.; Zhao, R.; Yan, R.; Zhang, X.; Chen, X. "A sparse auto-encoder-based deep neural network approach for induction motor faults classification. Measurement", 2016, 89, 171–178. [CrossRef]

[39]. Zhao, R.; Wang, D.; Yan, R.; Mao, K.; Shen, F.; Wang, J. "Machine health monitoring using local feature-based gated recurrent unit networks", IEEE Trans. Ind. Electron. 2017, 65, 1539–1548. [CrossRef]

[40]. Wu, Y.; Yuan, M.; Dong, S.; Lin, L.; Liu, Y. "Remaining useful life estimation of engineered systems using vanilla LSTM neural networks", Neurocomputing 2018, 275, 167–179. [CrossRef]

[41]. Dasgupta, S.; Osogami, T. "Nonlinear dynamic Boltzmann machines for time-series prediction", In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31. 18. Jain, A.; Kumar, A.M. Hybrid neural network models for hydrologic time series forecasting. Appl. Soft Comput. 2007, 7, 585–592. [CrossRef]

[42]. Jain, A.; Kumar, A.M. "Hybrid neural network models for hydrologic time series forecasting", Appl. Soft Comput. 2007, 7, 585–592. [CrossRef]

[43] Sak, Hasim; Senior, Andrew; Beaufays, "Long Short-Term Memory recurrent neural network architectures for large scale acoustic modeling", 2018-04-24.

[44]. Dyer, C.; Ballesteros, M.; Ling, W.; Matthews, A.; Smith, N.A. "Transition-based dependency parsing with stack long short-term memory", arXiv 2015, arXiv:1505.08075.

[45]. Tai, K.S.; Socher, R.; Manning, C.D. "Improved semantic representations from tree-structured long short-term memory networks", arXiv 2015, arXiv:1503.00075.

[46]. Kong, W.; Dong, Z.Y.; Jia, Y.; Hill, D.J.; Xu, Y.; Zhang, Y. "Short-term residential load forecasting based on LSTM recurrent neural network", IEEE Trans. Smart Grid 2017, 10, 841–851. [CrossRef]

[47]. Dong, X.; Qian, L.; Huang, L. "Short-term load forecasting in smart grid: A combined CNN and K-means clustering ap_proach", In Proceedings of the 2017 IEEE international conference on big data and smart computing (BigComp), Jeju, Korea, 13–16 February 2017; pp. 119–125.

[48]. Wang, J.; Ma, Y.; Zhang, L.; Gao, R.X.; Wu, D. "Deep learning for smart manufacturing: Methods and applications", J. Manuf. Syst. 2018, 48, 144–156. [CrossRef]

[49] T. Abbasi, K. H. Lim, and K. S. Yam, “Predictive maintenance of oil and gas equipment using recurrent neural network”, Iop conference series: Materials science and engineering, vol. 495, no. 1, 2019, Art. no. 012067.

[50] J. S. Rahhal, and D. Abualnadi, “IOT Based Predictive Maintenance Using LSTM RNN Estimator”, 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), 2020, pp. 1-5, doi: 10.1109/ICECCE49384.2020.9179459.

[51] X. Bampoula, G. Siaterlis, N. Nikolakis, K. Alexopoulos, “A Deep Learning Model for Predictive Maintenance in Cyber-Physical Production Systems Using LSTM Autoencoders”, Sensors, vol. 21, no. 3, 2021, Art. 972, doi: 10.3390/s21030972.

[52]. Vos, K.; Peng, Z.; Jenkins, C.; Shahriar, M.R.; Borghesani, P.; Wang, W. "Vibration-based anomaly detection using LSTM/SVM approaches", Mech. Syst. Signal Process. 2022, 169, 108752. [CrossRef]

[53] A. Rivas, J. M. Fraile, P. Chamoso, A. González-Briones, I. Sittón, and J. M. Corchado, “A predictive maintenance model using recurrent neural networks”, International Workshop on Soft Computing Models in Industrial and Environmental Applications, 2019, pp. 261-270, doi: 10.1007/978-3-030-20055-8_25.

[54] H. M. Hasemian, “State-of-the-art predictive maintenance techniques”, IEEE Transactions on Instrumentation and measurement, vol. 60, no. 1, pp. 226-236, doi: 10.1109/TIM.2009.2036347.

[55] Duyar, Ahmet, Merrill, Walter (March 1992). "Fault diagnosis for the Space Shuttle main engine". Journal of Guidance, Control, and Dynamics.

[56] A. Shrestha, A. Mahmood, “Review of deep learning algorithms and architectures”, IEEE Access, vol. 7, pp. 53040-53065, 2019, doi: 10.1109/ACCESS.2019.2912200.

[57] X. K. Dang, X.K., Truong, H.N., Nguyen, V.C. and Pham, T.D.A., “Applying convolutional neural networks for limited-memory application”, TELKOMNIKA Telecommunication, Computing, Electronics and Control, vol. 19, no. 1, pp. 244-251, doi: 10.12928/telkomnika. v19i1.16232

[58] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network”, 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

[59] J. S. Ashwin, and N. Manoharan, “Convolutional neural network-based target recognition for marine search”, Indonesian Journal of Electrical Engineering and

Computer Science (IJEECS), vol. 8, no. 2, pp. 561-563, 2017, doi: 10.11591/ijeecs. v8.i2. pp561- 563.

[60] Y. L. Cun et al., “Handwritten digit recognition with a back-propagation network”, Proceedings of the 2nd International Conference on Neural Information Processing Systems, 1989, pp. 396-404.

[61] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, Advances in neural information processing systems, vol. 25, pp. 1097-1105, 2012.

[62] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, “Long short-term memory network for remaining useful life estimation”, 2017 IEEE international conference on prognostics and health management (ICPHM), 2017, pp. 88-95, doi: 10.1109/ICPHM.2017.7998311.

[63] S. Hochreiter, and J. Schmidhuber, “Long short-term memory”, Neural computation, vol. 9, no. 8, pp. 1735-1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[64] C. Hwang, K. Lee, and H. Jung, “Improving data quality using a deep learning network”, Indonesian Journal of Electrical Engineering and Computer Science (IJEECS), vol. 20, no. 1, pp. 306-312, doi: 10.11591/ijeecs. v20.i1.pp306-312.

[65] Zhu, T.; Luo, C.; Zhang, Z.; Li, J.; Ren, S.; Zeng, Y. "Minority oversampling for imbalanced time series classification", Knowl.-Based Syst. 2022, 108764. [CrossRef]

[66] Mansor, Hafizah, Konstantinos Markantonakis, Raja Naeem Akram, and Keith Mayes. “Let's Get Mobile: Secure FOTA for Automotive System”, NSS (2015).

[67] Nilsson, Dennis K., Lei Sun and Tatsuo Nakajima. “A Framework for Self-Verification of Firmware Updates over the Air in Vehicle ECUs”, 2008 IEEE Globecom Workshops (2008): 1-5.

[68] Frisch, Dustin, Sven Reissmann and Christian Pape. “An Over the Air Update Mechanism for ESP8266 Microcontrollers”, 2017.

[69] Daniel Mckenna, BU Automotive, NXP Semiconductors, “Making-Full-Vehicle-OTA-Updates-Reality-WP”.

[70] Dr. Walter J. Buga, CEO, “Automotive OTA The potential and the challenge”, Tokyo, Japan, May 28, 2013.

[71] Idrees, Muhammad Sabir, Hendrik Schweppe, Yves Roudier, Marko Wolf, Dirk Scheuermann and Olaf Henniger. “Secure Automotive On-Board Protocols: A Case of Over-the-Air Firmware Updates”, Nets4Cars/Nets4Trains (2011).

[72] Pedroza, Gabriel, Muhammad Sabir Idrees, Ludovic Apvrille and Yves Roudier. “A Formal Methodology Applied to Secure Over-the-Air Automotive Applications”, 2011 IEEE Vehicular Technology Conference (VTC Fall) (2011): 1-5.

[73] J. A. Garay and L. Huelsbergen, “Software Integrity Protection Using Timed Executable Agents,” in ASIACCS ’06: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security. New York, NY, USA: ACM, 2006, pp. 189–200.

[74] Strandberg, Kim, Dennis Kengo Oka and Tomas Olovsson. “UniSUF: a unified software update framework for vehicles utilizing isolation techniques and trusted execution environments”, 2021.

[75] "ESP8266 AT Instruction Set ", Espressif Systems IOT Team 2016.

[76] " ESP8266 Technical Reference”, Espressif Systems IOT Team 2016.

[77] Formaniak, Peter G.; Leitch, David, "A Proposed Microprocessor Software Standard", BYTE - the Small Systems Journal. Technical Forum. Peterborough, New Hampshire, USA: Byte Publications, Retrieved 2021-12-06.

[78] S. M. Mahmud, S. Shanker, and I. Hossain, S. M. Mahmud, S. Shanker, and I. Hossain, “Secure Software Upload in an Intelligent Vehicle via Wireless Communication Links”, in Proceedings of IEEE Intelligent Vehicles Symposium. IEEE, 2005.

[79] Dennis Kengo Oka, “Simulated Attacks on CAN Buses: Vehicle virus,” Proceedings of the Fifth IASTED Asian Conference on Communication Systems and

Networks (ASIACSN). ACTA Press, 2008.

[80]. Satta, R.; Cavallari, S.; Pomponi, E.; Grasselli, D.; Picheo, D.; Annis, C." A dissimilarity-based approach to predictive maintenance with application to HVAC systems", arXiv 2017, arXiv:1701.03633.

[81] Groba, C.; Cech, S.; Rosenthal, F.; Gossling, A. "Architecture of a Predictive Maintenance Framework", In Proceedings of the 6th International Conference on Computer Information Systems and Industrial Management Applications (CISIM’07), Minneapolis, MN, USA, 28–30 June 2007. [CrossRef].