TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Priit Kullerkupp 040849IATM

5G NB-IoT Implementation for Predictive Car Maintenance

Master's thesis

Supervisors: Muhammad Mahtab Alam
PhD

Priit Roosipuu
MSc

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Priit Kullerkupp 040849IATM

5G IoT rakendus auto hoolduse ennustamiseks

Magistritöö

Juhendajad: Muhammad Mahtab Alam
Doktorikraad

Priit Roosipuu

Magistrikraad

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Priit Kullerkupp

15.12.2020

Abstract

Predictive maintenance (PdM) and connected cars are trends that are gaining in popularity year by year and are set to change our current perception of car maintenance. Massive machine type connections (mMTC) are becoming a reality with the introduction of 5G wireless communications which provides connectivity options for various use cases.

Modern vehicles produce a lot of data which is useful for supporting PdM, however the availability of relevant (historic) data and the means by which it can be collected, processed and communicated to cloud services is not something that is commonly implemented in vehicles. Moreover, fault detection needs to be modelled with machine learning (ML) algorithms to be able to give an early indication of a problem which poses additional hardware related requirements in itself.

In this thesis open solutions and standards are used to build a prototype data collection device which is integrated to cloud services to overcome limitations with many commercially available devices. Through practical experiments (air filter restriction, engine air leak) and test driving it is evaluated if Narrowband Internet of Things (NB-IoT) is able to support PdM for vehicles with data that is collected in real-time.

This thesis is written in English and is 64 pages long, including 5 chapters, 37 figures and 12 tables.

Annotatsioon

5G IoT rakendus auto hoolduse ennustamiseks

Ennustav hooldus ja ühendatud autod on märksõnad, mis koguvad aasta-aastalt populaarsust ning on muutmas meie tavapärast arusaama auto hooldusest. Viienda põlvkonna (*5G*) traadita side tehnoloogial põhinevad massilised masintüüpi ühendused (*massive machine type connections*) võimaldavad erinevates kasutusvaldkondades juurutada uusi sidelahendusi.

Ennustav hooldus on süsteemsel andmete kogumisel ja analüüsil põhinev teadusharu, mis võimaldab loodud mudelite alusel prognoosida vaadeldava süsteemi veatut toimimist. Kaasaegsed autod on varustatud erinevate sensoritega, millelt saadava info töötlusel tema juhtmoodulites tagatakse erinevate mehhanismide õige töö. Auto juhtmoodulite töövõime on piiratud, seepärast on autodes ennustava hoolduse rakendamiseks vaja täita nõudeid (andmete kogumine, töötlemine, võrgule edastamine) täiendava riistvara näol – tänaseni ei ole selliste lahenduste kaasamine autodes tavapärane.

Käesoleva magistritöö eesmärk on hinnata kitsaribalise asjade interneti (*Narrowband Internet of Things*) sobivust auto ennustava hoolduse tarbeks, mis tugineb virtuaalserveriga (*cloud server*) integreeritud seadmest reaalajas andmete kogumisel. Prototüüpseadme arendamisel on lähtutud süsteemi avatuse ja standardite kasutamise põhimõtteist. Kogutud andmeid on vaadeldud kahe eksperimendi (piiratud õhufiltri töövõime, mootori õhuleke) käigus loodud masinõppe (*machine learning*) mudelite abil. Lisaks on antud töös kogutud tavasõidu ajal *5G* võrgust parameetreid (paketikadu ja viide) ning saadud tulemused on esitatud tabeli kujul, mille põhjal on võimalik hinnata pakutava lahenduse töökindlust.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 64 leheküljel, 5 peatükki, 37 joonist, 12 tabelit.

List of abbreviations and terms

3GPP	3rd Generation Partnership Project
ANN	Artificial neural network
APP	Accelerator pedal position
ARIMA	Autoregressive integrated moving average
AT	ATtention
CAN	Controller area network
CDF	Cumulative distribution function
CSMA/CA	Carrier sense multiple access with collision avoidance
DLC	Datalink connector
DTC	Diagnostic troublecode
ECT	Engine coolant temperature
ECU	Electronic control unit
eDRX	Extended discontinuous reception
eMTC	Enhanced machine type communication
EOF	End of frame
FT	Fuel trim
GPIO	General purpose input-output
ID	Identifier
IoT	Internet of Things
IP	Internet protocol
LPWAN	Low power wide area network
LTE	Long-Term Evolution
LTFT	Long-term fuel trim
MAF	Mass airflow
MIL	Malfunction indicator light
ML	Machine learning
MLR	Multiple linear regression
MQTT	Message Queuing Telemetry Transport
MSE	Mean squared error

NB-IoT	Narrowband Internet of Things
NVRAM	Non-volatile random access memory
OBD	On-board diagnostics
OSI	Open Systems Interconnection
PCA	Principal component analysis
PCM	Powertrain control module
PDF	Probability density function
PdM	Predictive maintenance
PDP	Packet data protocol
PID	Parameteric identifier
PROM	Programmable read-only memory
PSM	Power save mode
QoS	Quality of Service
RAM	Random access memory
RF	Random forest
RPM	Revolutions per minute
RUL	Remaining useful life
SBC	Singleboard computer
SOF	Start of frame
SSH	Secure shell
STFT	Short-term fuel trim
SVM	Support vector machine
SVR	Support vector regression
TA	Timing advance
TCP	Transmission control protocol
TPS	Throttle position sensor
URC	Unsolicited return code
USB	Universal serial bus

Table of Contents

List of Figures

List of Tables

1 Introduction

Predictive maintenance (PdM), based on continuous condition monitoring, is closely related to Internet of Things (IoT) as data from sensors is used for estimating the remaining useful life (RUL) of user equipment. Implementation of a PdM system needs to consider the environment in which the user equipment is operating as well as data related requirements. In Figure 1 a general infrastructure of vehicle-based PdM system is depicted which can be categorized as follows:

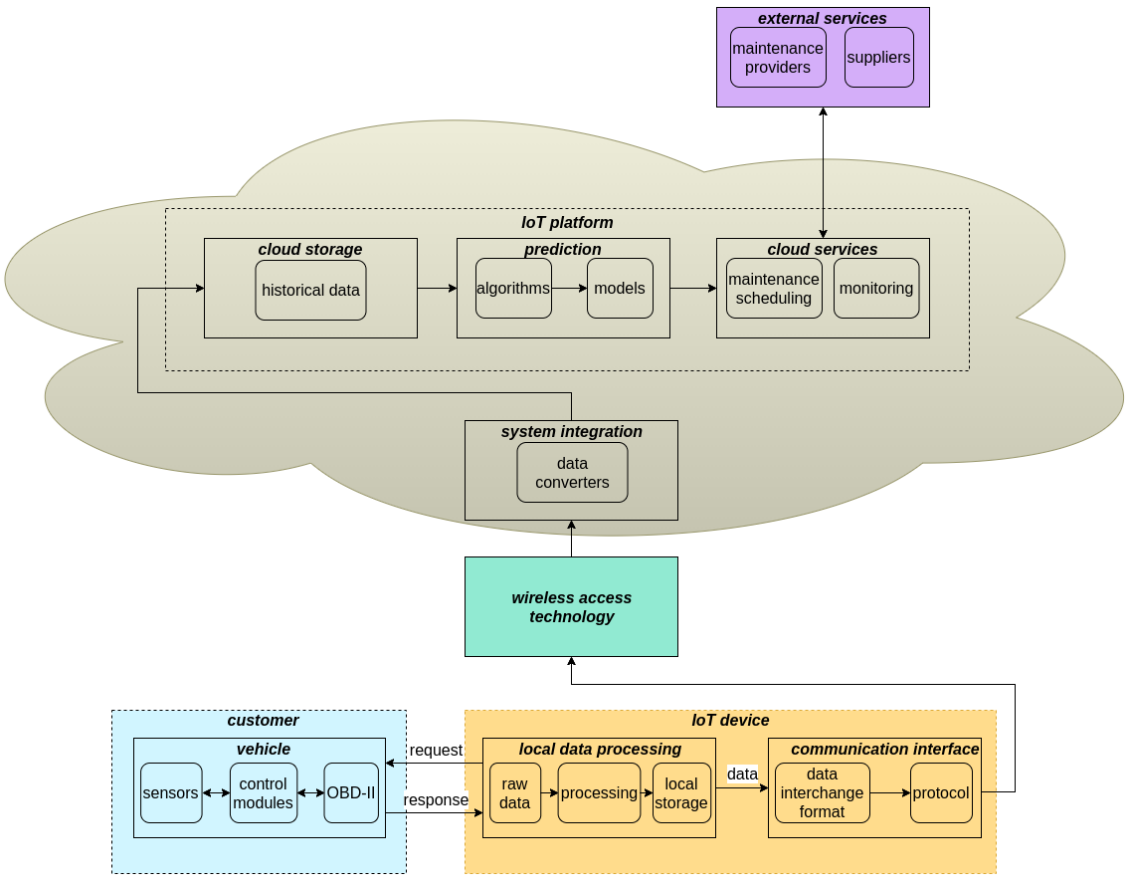


Figure 1. Cloud-based predictive maintenance infrastructure for vehicles.

- data source (sensors in vehicle)
- local data processing (IoT device inside vehicle)

- data transmission medium
- remote data processing (predictive algorithms)
- external services (maintenance scheduling)

Modern cars are equipped sensors which are monitored by various control modules and used as inputs for proper engine functionality. Access to these sensors is made available through standardized onboard diagnostic (OBD) connectors and signalling protocols which originate from engine emission control requirements. Control modules in vehicles are preprogrammed to detect irregular behaviour and will set fault codes in the module's memory. The processing and storage capabilities of vehicle control modules are limited as they are designed for robust operation, therefore additional hardware resources are needed for PdM purposes.

IoT device's main tasks are sensing, data processing and communication. In a vehicular application data must be requested from vehicle-bus, returned raw data is processed to convert it to human readable format and saved locally. Using appropriate data interchange format and protocol, the communication interface is used to forward data to cloud server for further processing over transmission medium.

Cloud services in PdM implementation are used for storing historical data, running predictive algorithms and informing relevant parties of maintenance actions (such as scheduling and pre-ordering of components). The benefits of vehicle based PdM solutions should be obvious, as the aim is to schedule maintenance actions when needed, thereby reducing costs for the user, but also in advance if a pending failure is detected. Implementation of such systems is not widespread due to increase in complexity (requirements for data collection, processing, storage, security and predictive modelling) compared to traditional maintenance strategies.

1.1 Problem statement

The research question this thesis aims to answer is if Narrowband Internet of Things (NB-IoT) can be used for data collection in a real-time vehicular setup and which

supervised machine learning algorithms are suitable for predictive maintenance in such use case.

1.2 Problem description

In PdM data is needed over a longer time period and depending on the use case some loss of information may not be critical as relevant data is accumulated during this period. On the other hand if data is collected too infrequently there may not be enough relevant samples based on which decisions could be made with required confidence levels. Some faults are more difficult to detect than others while in some cases it may be impossible. The goal of PdM is not the detection of immediate faults – it is assumed that based on historical data a model can be created which can be used to estimate the condition of the system with required accuracy. Compared to traditional maintenance strategies where maintenance is performed at certain intervals, PdM requires a more sophisticated approach as an understanding of the system is needed with a model. A question that arises from the modelling requirement is where data processing should take place – either in vehicle or cloud servers, because both options have their advantages and disadvantages. Availability of data and and the nature of it varies greatly from one vehicle to another, therefore it is not feasible to create an universal model which could work equally well on different types of vehicles.

Sensing IoT devices, unlike multimedia and streaming applications, do not have high requirements for throughput and latency as data generally needs to be transmitted infrequently. The emergence of low power wide area network (LPWAN) radio access technologies in recent years such as NB-IoT and eMTC (enhanced machine type communication) have broadened the possibilities for deploying IoT on a massive scale in industrial, medical, home and other areas. Compared to NB-IoT, eMTC has full mobility support and increased data rates, although at the expense of higher spectrum bandwidth usage (1,4 MHz). NB-IoT has the potential to serve larger amount of devices needing only 180 kHz of spectrum bandwidth and can offer extended coverage but currently mobility support in commercially available hardware is limited which makes it more challenging to use in a vehicular application.

In this thesis it is studied if NB-IoT can be used to support PdM in vehicular setup – for this purpose a prototype data collection device is developed and integrated with cloud services. Using the configured system, data is collected in real-time for two PdM case studies (air filter restriction and engine air leak) and the performance of the system for such use cases is evaluated. The focus of this work is on configuring and integrating the system with appropriate protocols and processing collected data with algorithms for PdM.

The chapters are structured as follows. Chapter 1 gives background information on relevant terminology. In chapter 2 overview for state of the art with related works is given. Chapter 3 describes an implementation of predictive maintenance system in detail and chapter 4 provides experimental results. Conclusion for the work is given in chapter 5.

1.3 Background

The following subchapter gives an overview of relevant terminology used in this thesis.

1.3.1 OBD-II

On-board diagnostics (OBD) standards originate from United States where a serious air quality problem was recognized in the 1970s. The first iteration of the standard (OBD-I), introduced in 1985, required only malfunction indicator light (MIL) to be set and was generally found insufficient because it did not define a standard way how various vehicles could be diagnosed, therefore in 1989 OBD-II was introduced which specified unified data link connectors (DLCs), signalling protocols and self-diagnosis (monitors) for engine control systems [1].

OBD-II is defined in a way which leaves some room for future expansion and implementation of new functionality – there are 10 defined operating modes (typically manufacturers add their own modes after those) and some of the DLC pins are unused by OBD-II standardized signalling protocols. An OBD-II DLC with 16 pinholes is shown in Figure 2, usage of the pins depends on the signalling protocol used by the

vehicle. Pins 4 and 16 (chassis ground and battery positive terminal) are commonly used by all five signalling protocols [2]:

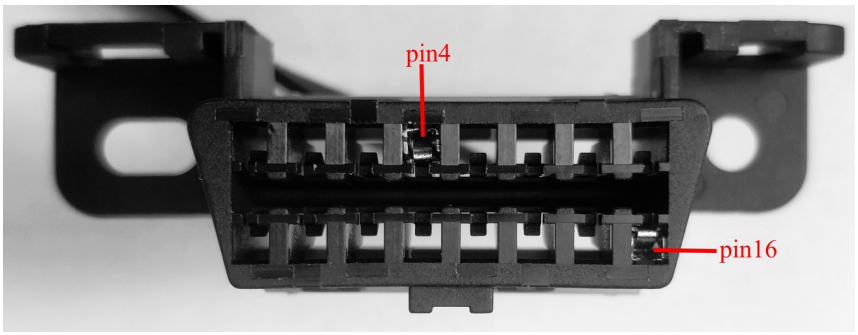


Figure 2. J1962 type-a female connector.

- SAE J1850 PWM
- SAE J1850 VPW
- ISO9141-2
- ISO14230-4
- ISO15765-4

1.3.2 Electronic Control Unit (ECU)

ECU is a broad term used to describe a system or subsystem which is electronically controlled. In automotive context ECUs with specific functions are also referred to as control modules such as powertrain control module (PCM), engine control module (ECM), transmission control module (TCM).

ECU is essentially a digital microprocessor which translates inputs (usually voltage signals) received from sensors or other ECUs to binary data, processes them and sends appropriate commands to actuators (such as solenoid driven valves, fuel pump relay, fuel injectors). ECUs use programmable read only memory (PROM) as lookup tables (maps) to assist in their decision making with regards to received input signals – for

example fuel injection pulse width maps, ignition timing maps. Random access memory (RAM) is used by ECU to store sensor values and diagnostic troublecodes (DTCs) [1].

1.3.3 Controller area network (CAN)

CAN is a serial communications protocol (maximum bitrate 1Mbit/s) developed by Bosch and standardized as ISO 11898-1 in 1993. Development of CAN was motivated from need for low-cost real-time control systems, new functionalities and to reduce the amount of wiring in automotive applications. Nowadays CAN is also used in industrial automation, medical equipment and other real-time embedded networking [3].

CAN protocol is defined in the first two layers of OSI (Open Systems Interconnection) model – physical and data link layer. Carrier sense multiple access with collision avoidance (CSMA/CA) is used in CAN as well as mechanisms for error detection. An arbitration process, which involves bitwise comparison of identifier (ID) fields (lower IDs are always prioritized), is held between nodes in CAN-bus to assign priorities before data transmission can start [3].

CAN-bus, as depicted in Figure 3, is a network of nodes on a pair of wires terminated with resistors at both ends of the bus where frames are delivered to all nodes on the bus because there is no recipient defined in the frame structure (Figure 4). The end of frame (EOF) is signalled with 7 recessive bits (logical 1) and the bus needs to be idle for the duration of interframe spacing (device specific) where only recessive bits are transmitted [4].

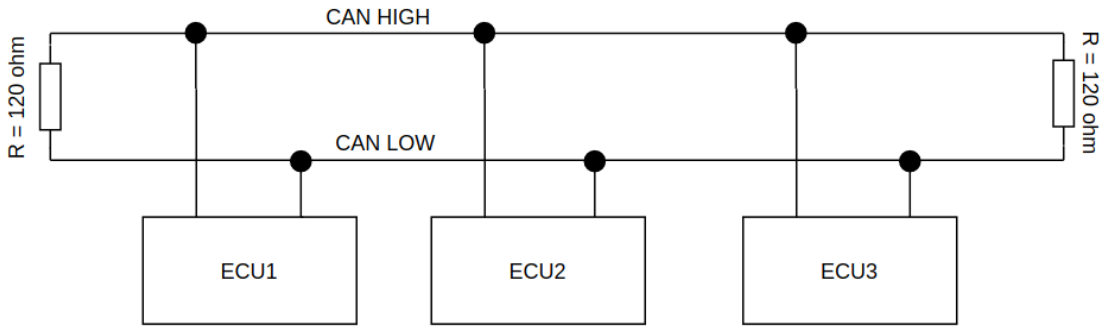


Figure 3. CAN-bus topology with three nodes.

Figure 4. CAN frame structure [7]: start of frame (SOF), identifier (ID), remote transmission request (RTR), control, data, cyclic redundancy check (CRC), acknowledge (ACK), end of frame (EOF).

Figure 5 depicts the arbitration process of two nodes with ID values 7E8 and 7E9 (in hexadecimal). CAN-bus start of frame (SOF) is one dominant bit (logical 0) followed by ID bits. Dominant bit has priority over recessive bit meaning that dominant bit is always written on the bus if both are transmitted at the same time. In this example ECU2 stops transmitting when it detects that dominant bit was written on the bus while it transmitted a recessive bit and must wait until the end of frame until it can try to retransmit.

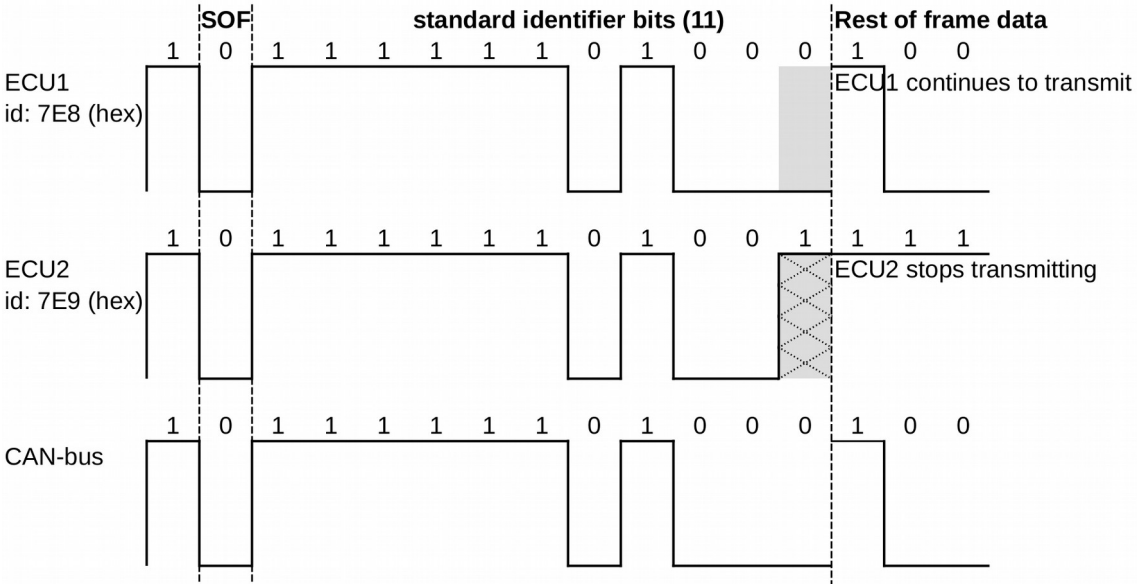


Figure 5. CAN-bus arbitration showing start of frame (SOF) bit and standard 11bit identifier field.

The increasing amount of ECUs and as a result increased load on CAN-bus in modern vehicles make it more and more difficult for nodes with lower priority to transmit as they need to keep waiting for longer time periods. A CAN message compression algorithm suitable for automotive applications is proposed in [5] to reduce the transmission data to up to 22%. Competing protocols for automotive applications are

LIN (Local Interconnect Network), CAN-FD (CAN with flexible data rate), FlexRay and TTCAN (time-triggered CAN). LIN is mostly used with low speed applications where FlexRay and CAN-FD were designed to support high speed networks up to 10 and 5 Mbit/s respectively [6].

1.3.4 Internet of Things (IoT) and cloud computing

IoT is a term used to describe a set of physical devices which contain embedded technology, use internet protocol (IP) to communicate with cloud services and are capable of sensing with external environment [7], [8]. IoT can be categorized into following layers:

- sensing
- processing
- communication
- application (backend and cloud services).

National Institution of Standards and Technology (NIST) defines cloud computing as a model for enabling on-demand network access to a shared pool of computing resources – its capabilities can be modified without requiring human interaction. The service models for cloud computing are [9]:

- software as as service (SaaS)
- platform as a service (PaaS)
- infrastructure as a service (IaaS).

There are numerous wireless technologies which could be used for IoT communication either in short range (Bluetooth, WiFi, ZigBee, Z-Wave) or long range (LoRa, Sigfox, NB-IoT, eMTC). Also various protocols for data transmission exist such as MQTT (Message Queueing Telemetry Transport), CoAP (Constrained Application Protocol) and more traditional such as HTTP (Hypertext Transfer Protocol) and FTP (File Transfer Protocol). The goal of this work is not to list or compare the features and

differences of various protocols as such works already exist [8],[10]. In this work NB-IoT was selected because the network coverage is provided by mobile network operators compared to technologies which operate in unlicensed spectrum bands (LoRa, Sigfox) and require the setup of dedicated infrastructure. Furthermore, the implementation and evaluation of NB-IoT for real-time communications in mobile environment is not well addressed as it is considered more a technology suitable for devices which are stationary in first few releases. MQTT is used in this work mainly because it is a light and already well established protocol in IoT which is integrated into many commercially available modems and cloud server platforms.

1.3.5 Narrowband Internet of Things (NB-IoT)

NB-IoT is a 5G radio access technology introduced in 3GPP release 13. The focus of NB-IoT is on the efficient use of radio spectrum – to support thousands of IoT devices over limited bandwitdh (180kHz) and it is predicted that for 2024 it will become a massive IoT technology (for smart cities, homes and industrial IoT) with the total number of connections exceeding 4 billion [11].

NB-IoT is related with LTE (Long-Term Evolution) technology, thus it can be deployed in unused guard intervals or in-band within an LTE carrier as well as in standalone mode [12]. NB-IoT is a LPWAN technology making use of features such as extended discontinuous reception (eDRX) and power save mode (PSM) for power efficiency [13]. 3GPP release 14 enhanced NB-IoT with Radio Resource Control (RRC) re-establishment allowing user equipment (UE) to resume data transfers while release 15 provided reduced transmission delays, release 16 adds improvements for idle mode and optional connected mode mobility [11],[14].

1.3.6 Message Queuing Telemetry Transport (MQTT)

MQTT is an application layer (OSI model layer 5-7) protocol aimed for unreliable and bandwidth limited networks working on top of TCP/IP (transmission control protocol / internet protocol). MQTT is commonly used in IoT for data transmissions – it is based on publish-subscribe communication model where clients subscribe to topics to receive messages published from other clients. There is no direct communication link between

clients – all messages are delivered through a broker (server) instance. Three quality of service (QoS) levels can be configured:

- "at most once" – packet loss can occur as no retransmission attempt is made
- "at least once" – packet delivery is guaranteed but duplicates may occur
- "exactly once" – packets are assured to be delivered only once .

MQTT has low packet overhead with a fixed header size of 2 bytes [15].

1.3.7 Machine learning (ML)

ML is described as a computerized method for solving learning problems, which, through the use of various algorithms applied to observed data, is able to improve the performance of the system [16]. Supervised ML algorithms use historical (labelled) data to create models that best represent the relationship between inputs and outputs, usually by assigning weights to features (inputs) [17]. Two main approaches used for training supervised ML models are physical and data-driven. While effectively both approaches are based on measurable data, all available data is used for model training in data driven approach. With physical approach a subset of available data is used by performing a feature selection with the assumption that the selected features have the most impact on the key parameter we are trying to predict. In unsupervised learning the training data is not labelled and it is generally applied to clustering tasks to identify patterns within data [18].

2 State of the art

In this chapter an overview is given for commonly used algorithms for predictive maintenance and OBD-II based data collection devices. At the end of the chapter related works are reviewed.

2.1 Algorithms for predictive maintenance

A recent systematic literature review [19] reveals that interest in PdM as a maintenance strategy is increasing based on the growth of yearly publications in the field which is attributed to the increased amount of data produced by industrial applications and advances in ML algorithms. According to the review most commonly used ML algorithms are random forest (RF), artificial neural networks (ANN), support vector machines (SVM) and K-means. A recent research paper [20] highlights that ML can be applied to improve the cybersecurity of next-generation vehicles – it is said that the most critical aspect to train a model is the availability of data not the ML algorithm itself, at the same time it is noted that due to constraints on computing hardware resources in vehicles, the models can not be overly complex.

Random forest (RF) is a machine learning algorithm based on decision trees and can be applied to classification and regression modelling. RF creates a number of decision trees (estimators) and for each tree uses random sampling (bootstrapping) of original data to create its training dataset – for each step in a decision tree randomly selected features are used (Figure 6). Prediction results from individual decision trees are averaged using statistical means with each decision tree having equal weight. Due to random sampling of training data, RF can produce good results even with reduced quality training datasets [21].

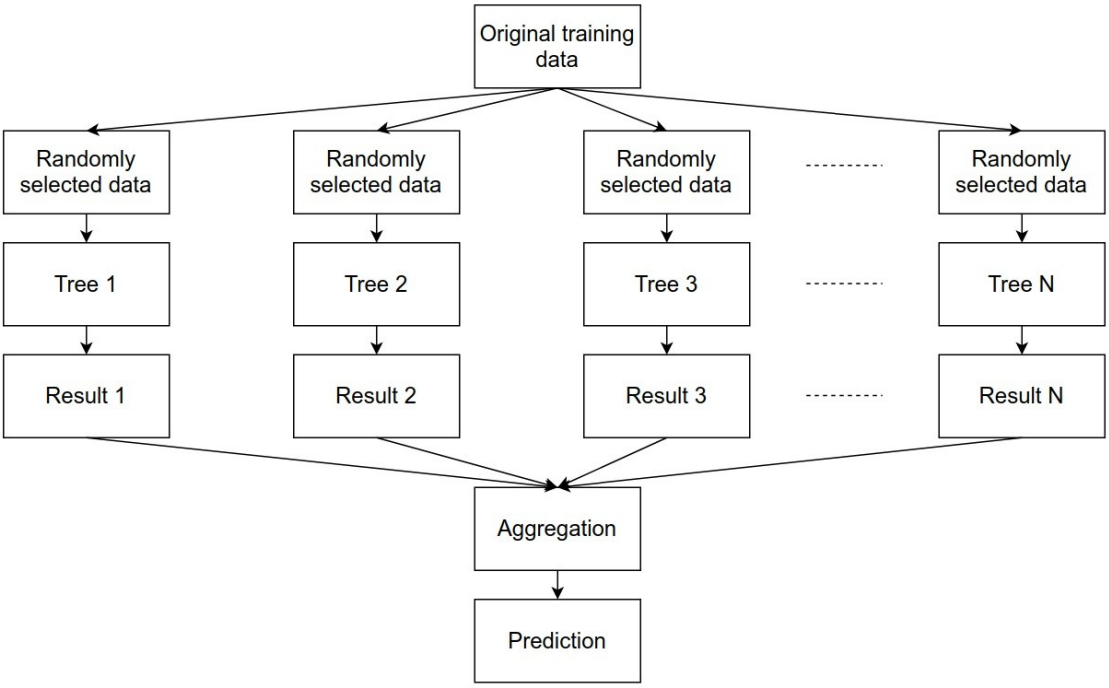


Figure 6. Random forest algorithm.

Support vector machine (SVM) was initially introduced as a classification algorithm but later also extended for regression tasks. SVM classifier tries to find a hyperplane which provides best separation between classes while maximizing the margin between the hyperplane and closest data points. The dimensions of the hyperplane depend on the number of input features – with two features the hyperplane is a line, as shown in Figure 7, and with three features it is a two-dimensional plane [22].

SVM uses a technique called kernel trick to map the input data into a space with higher dimensions to obtain a separating hyperplane if no separation is otherwise possible which is often the case with non-linear data. Radial basis, linear, polynomial and sigmoid are widely used SVM kernel functions. Proper tuning of kernel function parameters as well as scaling (standardizing features to have zero as mean value and unit variance) is essential for the best performance of SVM [22],[23].

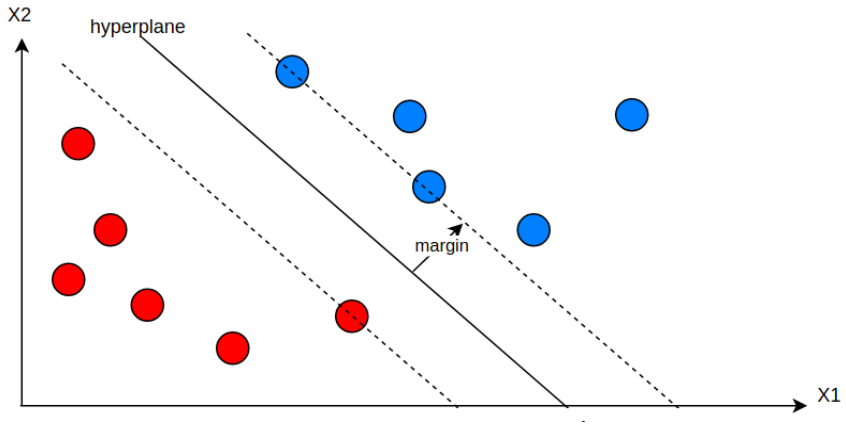


Figure 7. SVM hyperplane (line) with two input features.

Artificial neural networks (ANN) is a supervised ML algorithm where data is moved (in one direction) from nodes in bottom layers to upper layers forming a network of nodes, as shown on Figure 8. Linear combination of data from inputs is fed to nodes at multiple layers where they are summed and based on activation functions (sigmoid, hyperbolic, tangent, rectified linear unit) it is decided if the data should be forwarded to next layers. Weights are randomly selected when training is started and are adjusted while learning the model. ANNs with a large number of layers is called deep-learning [24],[25].

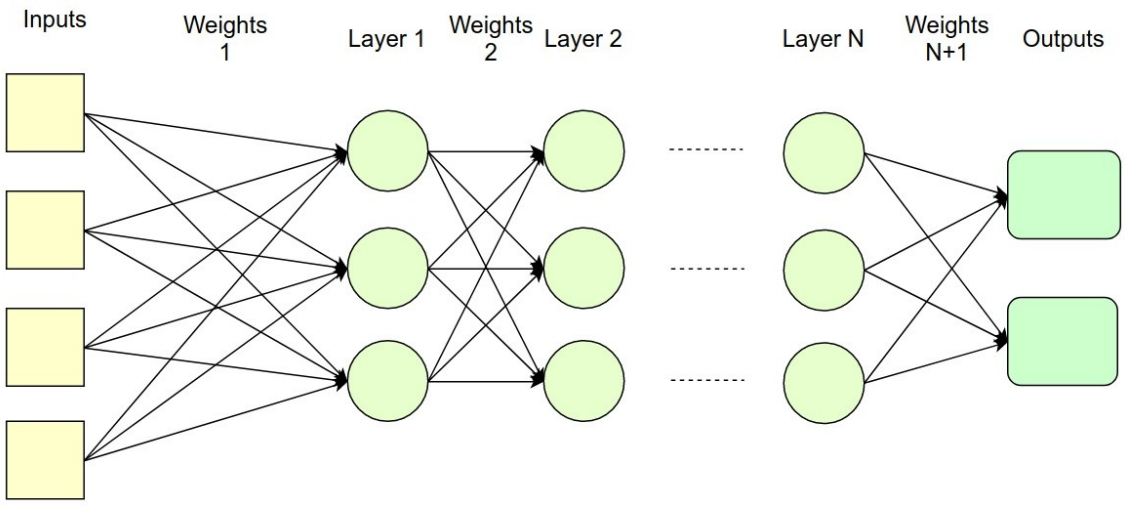


Figure 8. General structure of multilayered neural network.

Principal component analysis (PCA) is an unsupervised statistical method used to reduce the dimensionality of data which is beneficial in reducing the time required for ML model training. Principal components, calculated from covariance matrix, are a linear combination of all input data and try to maximize variance within the data while being uncorrelated to another – if the scores (explained variance) of first few principal components are significantly higher than the remaining ones', then the dataset can be represented with those components with minimal loss of accuracy [26].

2.1.1 Machine learning metrics

The performance of ML algorithms are compared based on their prediction accuracy – the use of metrics depends on the type of algorithm that is used: (a) classification, (b) regression, (c) clustering. Classification and regression models are based on supervised learning whereas clustering (unsupervised) models have no prior knowledge for what the output should be. Classification models are used when we are interested if the output belongs to a range of values separated by a classifier (1 and 0 for binary classification). Regression models are used when continuous output values need to be predicted (Figure 9).

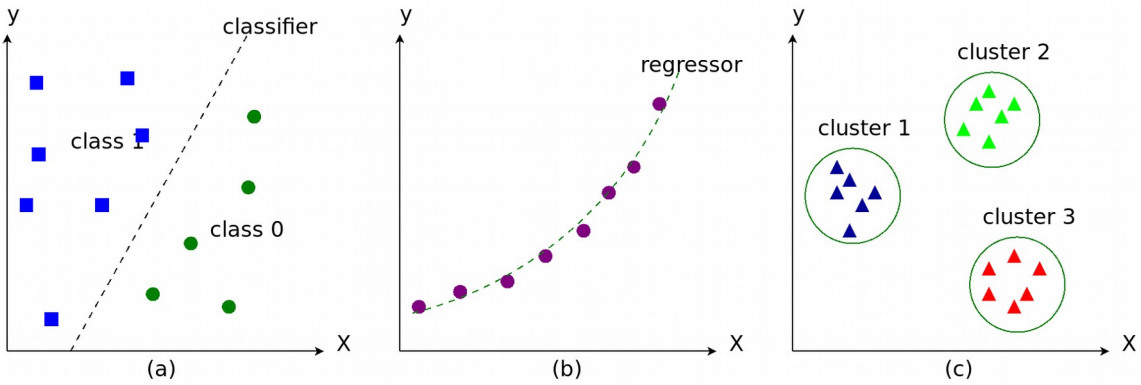


Figure 9. Algorithms based on: (a) classification, (b) regression, (c) clustering.

Classification metrics:

- **Accuracy** is the ratio of $\frac{Tp+Tn}{Tp+Tn+Fp+Fn}$,where Tp is the number of true positives, Tn true negatives, Fp false positives and Fn false negatives.

- **Precision** is the ratio of $\frac{Tp}{Tp+Fp}$, where Tp is the number of true positives and Fp false positives.
- **Recall** is the ratio of $\frac{Tp}{Tp+Fn}$, where Tp is the number of true positives and Fn false negatives.
- **F1 score** is the ratio of $\frac{2*Precision*Recall}{(Precision+Recall)}$

Regression metrics:

- **MSE** (mean squared error) is calculated as $\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$, where y_i is true value and $\hat{y}_i$ predicted value.
- **R^2 score** is defined as $R^2=1-\frac{u}{v}=1-\frac{\sum_{i=1}^{n}(y_i-\hat{y}_i)^2}{\sum_{i=1}^{n}(y_i-\bar{y})^2}$, where u is sum of squared residuals, v is total sum of squares, $\hat{y}_i$ is predicted value, $\bar{y}$ is statistical mean and y_i is true value. The maximum value for R^2 is 1 which denotes a perfect prediction.

2.2 Maintenance strategies

According to standard EN-13306, PdM is a preventive maintenance policy based on continuous condition monitoring which in IoT generally assumes the use of sensors. Historical data both from corrective and preventive maintenance is useful for modelling a PdM system (Figure 10). PdM aims to reduce operational costs by scheduling maintenance activities only when needed, thereby using equipment to its full potential. PdM assumes that it is possible to map gradual degradation or deviation processes to time scale to make a prediction about the future health state or some other key indicator of the system [27].

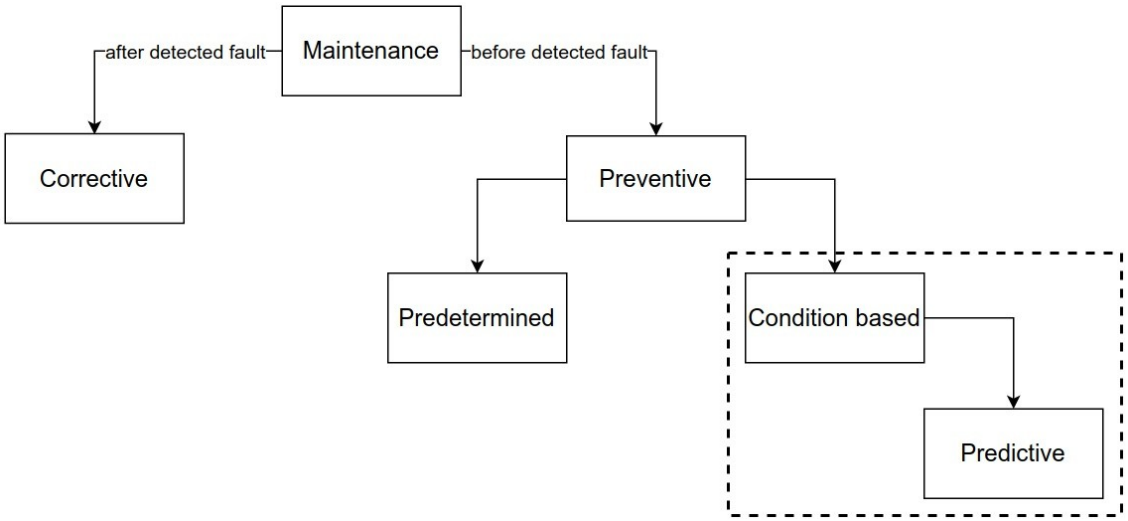


Figure 10. Maintenance strategies [27].

A recent literature survey in PdM [28] reveals that the use of remote data storage and access for PdM systems is not widely addressed. Furthermore, system integration is said to be problematic because of the effort needed to make different systems and devices work together [29].

2.3 Remaining useful life (RUL)

RUL prediction (Figure 11) is based on a set of models which try to connect degradation process to time scale. RUL prediction models are classified as follows [30]:

- knowledge based models
- life expectancy models
- ANN models
- physical models

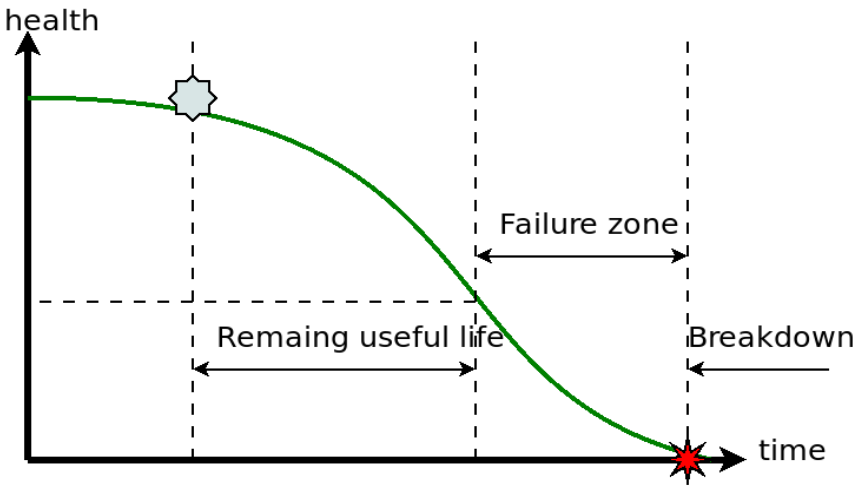


Figure 11. Remaining useful life [27],[30].

Knowledge based models are a set of rules created by experts based on their previous learned experiences – the advantage is that the model is relatively easy to create but requires an extensive set of rules to be defined. Life expectancy models use probability and reliability related methods to model failure data with statistical distribution functions and other metrics such as mean-time-to-failure (MTTF) and trend extrapolation. ANN models are useful for RUL prediction when a good physical understanding of the system can not be obtained whereas physical models require deep understanding of the system with mathematical models [30].

Weibull distribution (Weibull minimum extreme value) function is regularly used in life expectancy failure rate modelling, its probability density function (PDF) is defined with equation (1), where $x \geq 0$, $c > 0$ and $\lambda > 0$.

$$f(x) = \frac{c}{\lambda}\left(\frac{x}{\lambda}\right)^{c-1} \exp\left(-\left(\frac{x}{\lambda}\right)^{c}\right) \tag{1}$$

λ is the scale parameter of the function and in standardized form ($\lambda = 1$) Weibull distribution is described by equation (2), where c is the shape parameter of the function and values $c > 1$ indicate that the failure rate is increasing over time as depicted in Figure 12.

$$f(x) = cx^{c-1} \exp\left(-x^{c}\right) \tag{2}$$

In special cases of $c = 1$ and $c = 2$ Weibull distributions become respectively exponential and Rayleigh distributions [31].

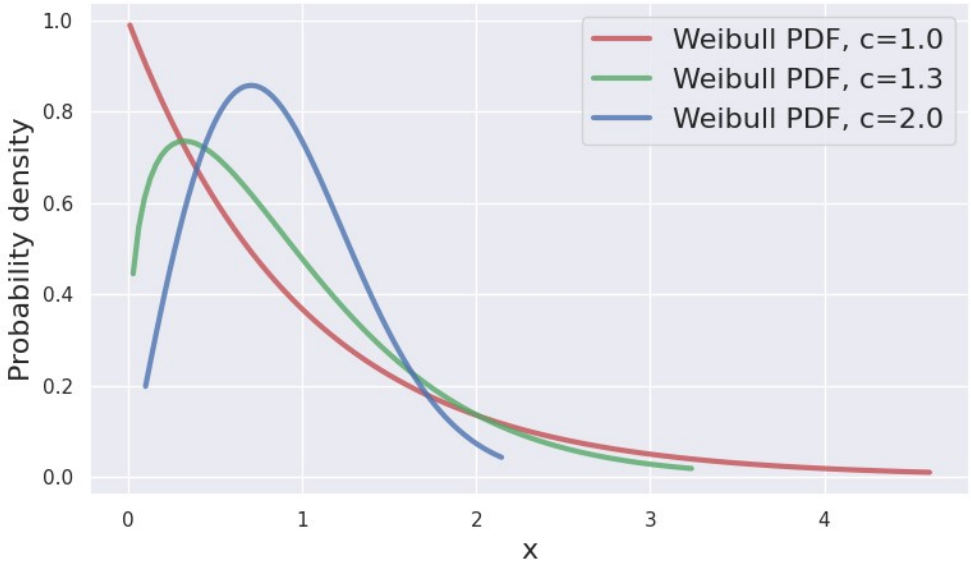


Figure 12. Weibull distribution probability density function.

ARMA (autoregressive moving average) and ARIMA (autoregressive integrated moving average) statistical modelling is another common method for time series forecasting where future states are defined as a linear combination based on previous observations [30].

2.4 Data collection devices

This subchapter gives an overview of devices that were considered and tested for this thesis. Various hardware solutions for OBD-II data collection with cellular connectivity options already exist. These devices are typically Arduino or Raspberry Pi based, some of them are listed in Table 1.

ELM327 is an integrated circuit (IC) from ELM Electronics which supports all five OBD-II protocols and found inside many scan tools. These devices act like a bridge between OBD-bus and computer serial interface (WiFi, Bluetooth or USB) and are configured using AT (ATtention) commands [32]. Many smartphone based apps compatible with ELM327 OBD-II scan tools exist which mainly communicate over a Bluetooth connection to monitor in vehicle signals. While WiFi and Bluetooth allow

convenient access to OBD-bus, they generally impose weak security. USB-based ELM327 scan tool was used in this thesis as a part of data acquisition system from vehicle CAN-bus while Bluetooth-based adapter was used in initial phases as the functionality of these devices is the same.

Table 1. OBD-II data collection devices.

Device name	**Description**
Huasheng HS-4000C	CAT M1/NB OBD GPS Tracker
Teltonika FMM001	LTE CAT M1/GNSS/BLE plug and play OBD tracker
Freematics ONE	OBD-II telematics prototyping platform
AutoPi Telematics Unit	WiFi/NB/Cat M1 telemetry device

Raspberry Pi is a singleboard computer (SBC) which comes with various connectivity options such as WiFi, Bluetooth, USB, Ethernet and GPIO (general purpose input-output). Raspberry Pi runs on Linux operating system and due to its low cost is widely used as an IoT device with the addition of sensor boards and other applications. PiCAN2 is an add-on board for Raspberry Pi which allows it to communicate with CAN-bus (CAN v2.0B at 1 Mbit/s) over a network interface provided by SocketCAN driver. With SocketCAN it is possible to monitor traffic or send messages to CAN-bus using provided application programming interface (API).

Arduino is a singleboard microcontroller which can be extended with various shields to act as an IoT device for data acquisition. The computational power of these devices is limited and they can generally be assigned to perform a specific task as running multiple threads is not supported, which complicates communication with multiple serial devices at the same time. Arduino MKR NB 1500 is a board integrated with narrowband modem which was tested initially for this thesis. The advantage of using a microcontroller such as Arduino is that they can start operating instantly when powered on as there is no overhead in terms of operating system which needs to boot and shut down properly as is the case with Raspberry Pi. The downside, however is that a lot of low-level code is needed (managing memory, buffers, data type conversions) and the

software needs to be compiled and uploaded every time a change is made which limits the rate at which a prototype can be developed.

HS-4000C is an OBD-based telematics device (microcontroller) which was tested with a view for data collection for this thesis. The configuration of this device is done over a serial connection and server side communication – everything had to be implemented on the cloud server:

- a TCP server instance was created with Python socketserver module
- all encoding and decoding functions were explicitly implemented following device protocol

A common approach with many commercially available devices appears to be to use messaging protocols where data is packed as hexadecimal strings using type-length-values (TLVs). On the one hand this approach can make data transmissions more efficient but on the other it makes cloud server integration more difficult because data needs to be parsed accordingly to each devices specification and firmware. In the end not enough usable data could be collected from the device for this thesis – at the time of writing this many of the standard PIDs (parametric identifiers) we were interested in were not supported.

2.4.1 Conclusion from tested devices

An existing solution which could be easily modified to perform various experiments and to answer the research question of this thesis was not found, therefore it was decided to develop a prototype device which could:

- work independently from a smartphone and not require interaction from the driver
- be easy to access, configure and monitor
- use open standards and frameworks which are easy to interpret and process such as JSON (JavaScript Object Notation) and MQTT.

ELM327-based scan tools and Raspberry Pi were found to be straightforward to work with and fulfilled the above requirements. In this thesis only standard OBD-II PIDs were considered to be used. Using standard OBD-II PIDs we are able to access only a part of sensors that are available in a vehicle – the manufacturers are not obligated to support all standard PIDs and as a result their availability depends greatly on the make and model of the vehicle. Access to vehicle specific non-standard PIDs is available using expert diagnostic software (such as VCDS, ODIS, Autel, Delphi Auto) and are beyond the scope of this thesis.

2.5 Related works

An automated fault detection approach for detecting engine pre-ignitions based on deep neural networks is proposed in [33]. According to the authors analytical methods could use all ECU signals to identify fault patterns more accurately – referred to as Deep Automotive Diagnostic Network (DADN), which is implemented in Python3 with Tensorflow and Keras frameworks. Their training dataset consists of 1600 test drives and in total 1681 ECU signals. They have compared the resulting models' accuracy with traditional classification machine learning models and shown that their model has higher accuracy for detecting pre-ignitions of an engine while the detection of no pre-ignitions performance was comparable with traditional models. They have shown that the performance of the classifier can be improved if 50% of statistically relevant signals are used. It is noted that future research is needed for implementing an onboard embedded microcontroller with the fault detection model.

Although the focus of [33] was for model generation, the methodology and hardware setup used for collecting ECU signals is not revealed nor which type of vehicle was used (it is said a vehicle fleet with the same engine). It is not explained which method has been used for obtaining statistically relevant signals to improve model classification accuracy.

The authors of [34] have applied statistical methods (mean, 25^{th} percentile, 75^{th} percentile and standard deviation) for data aggregation and used RF regression in combination with LIME (Local Interpretable Model-agnostic Explanations) algorithm for most relevant signals preselection to predict the ageing of a vehicle powertrain

component (exhaust gas recirculation cooler). They have said that based on manually selected signals it is difficult to evaluate the ageing process of vehicle components. They say extensive amount of data collected from one vehicle could be later used on other similar vehicles using a cloud based approach.

Renyi entropy analysis has been used in [35] to provide information about an upcoming fault with engine coolant temperature sensor. The authors suggest there is lack of automated solutions for collecting and analysing data from vehicles, thus they have reverse engineered an android app (Torque) to process its logfiles and publish the data to a NoSQL (not only structured query language) server using HTTP (hypertext transfer protocol) requests. The use of open source standards and technologies is encouraged in this work for easier access to data.

The system architecture in [35] is not truly automated because it is based on the existence of an android phone which requires the user to start an app every time for data collection. A more automated approach could have been to develop an android service which runs continuously in the background and a more suitable protocol with lower packet overhead than HTTP could have been used for telemetry upload.

Statistical and ML methods have been applied in [26] to CNC (computer numerical control) drilling machine to predict the flank wear of drillbit. They have calculated five statistical features for each sensor and among these features have selected those which are showing high correlation (using Spearman correlation coefficient) with measured tool wear. For the selected features they have applied PCA to reduce the dimensionality and the required training time of the ANN model, which yields very accurate results based on RMSE (root mean squared error). They highlight the importance of reducing the amount of data volume for modern industries.

A low cost PdM approach for a small manufacturing enterprise (SME) is proposed in [36] – data acquisition system based on Raspberry Pi and a sensor board is implemented to measure temperature and vibrations and compared to the roughness of each produced machine part. Recursive partitioning and regression tree ML models were used to predict machining quality.

They have used Dropbox in [36] as a cloud data storage due to easy integration with Raspberry Pi, although it would have made more sense to use a database management system instead of storing plain text files for production use.

Recently histogram data from a large fleet of heavy-duty Scania trucks was used in [37] to build a model based on RF classification to predict failure of NOx sensor. Relevant historical data, which is stored onboard each truck in aggregated form, is downloaded during every workshop visit – the final dataset included 16980 trucks (with 951 reported NOx failure data). It is noted that due to data aggregation, there is loss in information (especially sudden changes in data) which could be used to detect a fault. It is said that no clear improvements have not been made in the field to build predictive models from such aggregated data. The accuracy obtained for identifying faulty trucks was 84%.

R. Prytz and others have provided a method in [38] which could be used to detect anomalies in the operation of vehicles based on most interesting (relevant) signals. They have collected on-board data from a Volvo truck under normal and self-inflicted fault conditions (clogged air filter and exhaust, charge air cooler leak), in total 21 signals were collected during each drive. LASSO (Least absolute shrinkage and selection operator) and RLS (recursive least squares) methods have been used to obtain signals which show strong correlation among them and used as parameters to estimate particular signals. Classification accuracy of ML algorithms (linear regression, SVM, RF) has been compared and their results show that RF performs much better than others. It is said that some faults are hard to detect, attributed to the accuracy and availability of related measurements.

Work	**Research problem**	**Used algorithms**	**Data source**	**Data collection hardware**	**Cloud integration**
[33]	Engine pre-ignition detection	Deep neural networks	1600 test drives, in total 1681 ECU signals from vehicle fleet with same engine	Not revealed	no
[34]	Ageing of exhaust gas recirculation cooler	RF, LIME	Data from workshop in certain intervals	CAN logger	no

Work	Research problem	Used algorithms	Data source	Data collection hardware	Cloud integration
[35]	Engine coolant temperature sensor fault	Renyi entropy analysis	Torque android application	ELM327 Bluetooth adapter	yes
[26]	Drillbit flank wear of CNC machine	ANN, PCA	Multiple sensor monitoring system	National Instruments USB-6361 data logger	no
[36]	Machining quality prediction	Recursive partitioning, regression tree	CNC machine	Raspberry Pi with vibration and temperature sensor board	yes
[37]	NOx sensor failure prediction	RF	16980 Scania trucks (with 951 reported NOx failure data)	Data logger. Data is downloaded during workshop visits	no
[38]	Anomaly detection based on most interesting signals	RF, SVM, linear regression, LASSO, RLS	Volvo VN780 truck, 21 signals from 14 driving runs with normal and faulty conditions	Data logger	no

2.5.1 Conclusion

Review of PdM literature reveals that no best approach stands out that can be applied to a PdM system – the methods used for data acquisition, availability of historical and relevant data, computing power and expert domain knowledge all affect the accuracy of the predictive model. ML algorithms such as ANN, SVM and RF are commonly used for classification or regression tasks which are aided by statistical methods for feature selection to improve accuracy of the model. Future research is needed for deploying PdM algorithms in embedded devices. PdM can be applied to systems where gradual deterioration or a pattern which leads to failure of a component can be observed and identified.

3 Implementation of predictive maintenance system

One of the challenges faced when starting work on this thesis was to decide which hardware solution could be used for data collection and how the device could be integrated to cloud services for further data processing. Key aspects such as supported communication protocols, OBD-II PIDs and configurability of the hardware were considered. Implemented system architecture is shown in Figure 13 where the IoT device is permanently installed inside a vehicle and requires no interaction from the driver.

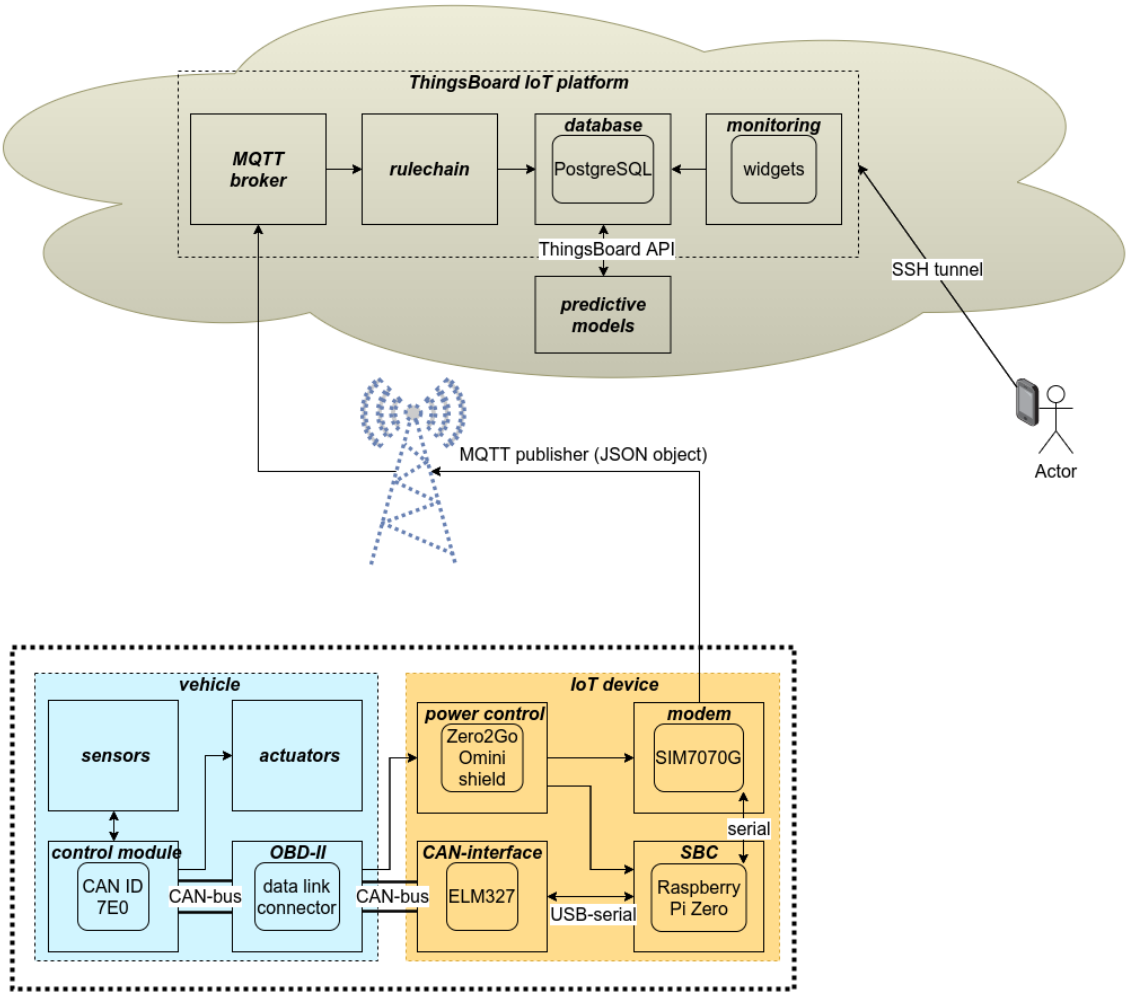


Figure 13. System architecture for predictive car maintenance.

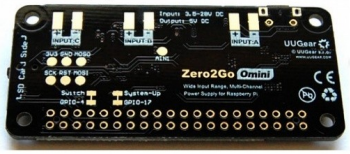
3.1 Hardware configuration

Table 2 lists the components selected for the IoT device – three stackable boards and CAN-serial bridge (Figure 14). Raspberry Pi, power supply and modem shields are connected using 40pin headers (Figure 15). Power is provided from J1962 connector pins 16 (battery positive) and 4 (chassis ground) to power management shield which is configured to automatically power on and off everything (from GPIO 5V and GND pins) based on high and low voltage thresholds. Configuration and install of power supply is done with an install script [39].

Table 2. Selected hardware components for IoT device.

Harware component	**Name**
CAN-serial bridge	ELM327 scan tool (with USB connection)
SBC	Raspberry Pi Zero WH
Modem	SIM7070G shield
Power supply	Zero2Go Omini shield
Storage	Sandisk Extreme Pro MicroSD 32GB

(a) (b) (c)



CPU	1GHz single core ARMv6	**Input**	DC 3.5-28V	**Access technology**	Cat-M, NB2, GPRS
GPU	VideoCore IV	**Output**	5V (max 2A)	**Interface**	40pin GPIO, microUSB
RAM	512MB	**Interface**	40pin GPIO	**Power chip**	MP1482, RT9193-33
Networking	Wifi, Bluetooth	**MCU**	ATtiny841		
Expansion	40pin GPIO, USB-OTG	**Power consumption**	0.36mA (V_{in}=12V)		
Storage	MicroSD				

Figure 14. Hardware components: (a) Raspberry Pi Zero, (b) Zero2Go Omini, (c) SIM7070G.

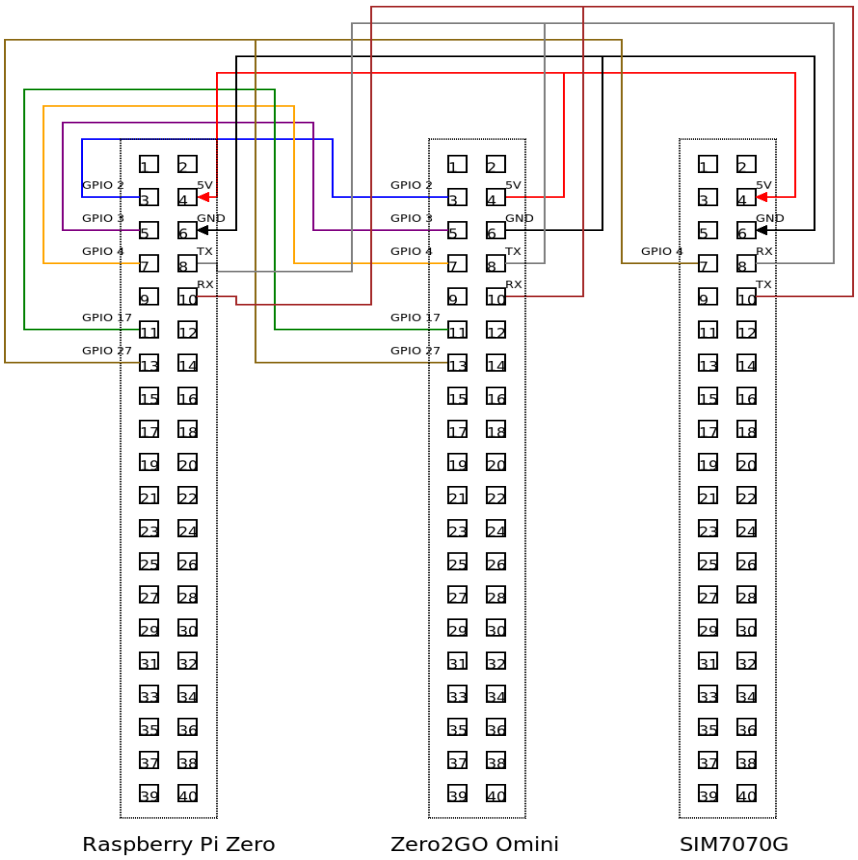


Figure 15. Wiring diagram for data collection device.

Modem and power shield both use GPIO4 pin as power on (switch) signal by default, therefore it was not possible to install modem shield directly on power shield – instead jumper wires were used for making connections to avoid the conflict (assigning GPIO27 to modem). Serial communication is used between modem and Raspberry Pi with GPIO14 and GPIO15 pins. ELM327 scan tool is connected to Raspberry Pi using microUSB OTG cable and accessed with USB-serial interface. MicroSD card is used for running the operating system – Raspberry Pi OS.

3.1.1 Modem configuration

All configuration and communication with the modem is done over serial port ('/dev/ttyAMA0') with baudrate 115200, all commands used are summarized in Table 3 and are referenced from SimCom manual [40]. Modem was configured for NB-IoT mode only and MQTT was used to publish data to cloud server. Engineering mode command was used to gather signal strength and quality parameters, also ping latency

data was collected to evaluate the performance of NB-IoT. There are commands which persist – they are saved in modem non-volatile memory (NVRAM) and are required to be set only once. Other commands which are saved in random access memory (RAM) will need to be executed every time modem is rebooted. AT prefix is set at the beginning of each command and carriage return (CR) is used to terminate the command. Once a command is executed we must wait for a final reply from modem (can be 'OK' or 'ERROR') before another command can be sent. Another aspect to keep in mind is that unsolicited return codes (URCs) could interrupt our program if not handled properly – URCs might be received at any time. With some modems it is possible to turn off URCs but nothing of the sort was found in relevant documentation, therefore the following procedure was used: (i) create a list of URCs which could be sent by the modem, (ii) keep reading data until a final reply was found, (iii) remove URCs from received data if present and handle them as necessary.

Table 3. SIM7070G configuration commands.

Command	**Description**
AT+CFUN=0	Put the modem in minimal functionality mode
ATE0	Turn off echo of sent commands
AT+CNMP=38	Set preferred mode selection to LTE only
AT+CMNB=2	Set NB-IoT mode only
AT+CBANDCFG="NB-IOT",1,2,3,4,5,8,12,13,18,19,20,25,26,28,66,71,85	Set NB-IoT radio frequency bands
AT+CGDCONT=1,"IP","internet.emt.ee"	Define PDP (packet data protocol) context to use IPV4 and set APN (access point name)
AT+COPS=0,0,"",9	Use automatic operator selection (operator field is ignored) and NB-IoT mode
AT+CSDP=2	Set service domain to circuit and packet switched networks
AT+CFUN=1	Put modem in full functionality mode
AT+CNACT=0,2	Auto activate application network
AT+CGPADDR	Check if IP address is obtained from network

Command	Description
AT+CNACT?	Check if PDP is activated
AT+CNSMOD?	Show current network mode
AT+COPS?	Show current network mode and operator
AT+CENG?	Show engineering mode data from serving and neighbor cells (signal strengths and quality parameters)
AT+CNTP=pool.ntp.org,12,0,0	Synchronize local time with NTP (network time protocol) server
AT+SNPING4=51.89.167.146,1,16,1000	Send one ping request with packet size 16 bytes and timeout of 1000ms
AT+SMCONF="URL",51.89.167.146,1883	Set MQTT broker server ip and port
AT+SMCONF="USERNAME",T1_TEST_TOKEN	Configure MQTT username (access token)
AT+SMCONF="CLEANSS",1	Use a clean MQTT session
AT+SMCONF="QOS",0	Set MQTT QoS to level 0
AT+SMCONF="RETAIN",1	Retain the last message on MQTT broker
AT+SMCONN	Establish connection to MQTT broker
AT+SMSTATE?	Query MQTT broker connection state
AT+SMPUB="v1/devices/me/telemetry",52,0,1	Publish MQTT message to specified topic and set content length (in bytes), QoS and retain policy
AT+SMDISC	Disconnect from MQTT broker

3.1.2 ELM327 configuration

ELM327 scan tool is configured in a similar manner using AT commands with baudrate 38400 over serial port ('/dev/obdscanner'). ELM327 commands are terminated with CR and it is necessary to wait for the final reply from the device before new requests can be made – returned '>' character means it is done processing a previous request. ELM327 has programmable parameters which are stored in NVRAM but no such commands were used. Setting up ELM327 is relatively simple as can be seen from Table 4 – in this

configuration a CAN filter is set to accept data only from engine ECU with CAN ID 7E0 (8 hexadecimal is added in response ID, hence 7E8) which is useful for speeding up how frequently requests can be made. Response speed from ELM327 can be improved if we specify how many frames (lines) of data is expected so it does not have to keep on waiting for data to arrive until timeout (200ms) is reached [32]. Service mode 1 PIDs 0-60 (hexadecimal) all return one data frame. We have already set CAN receive filter, so only one data frame is expected. We can therefore simply append "1" to the end of a PID request. In such configuration new data can be requested every 10-20ms.

Table 4. ELM327 configuration commands.

Command	**Description**
ATZ	Reset the device
ATE0	Turn off echo of sent commands
ATL0	Turn off linefeeds (newline characters)
ATS0	Turn off printing of spaces
ATH1	Turn on message headers (reveals CAN ID)
ATTP6	Set ISO 15765-4 CAN (11 bit ID, 500 kbaud) as OBD-II protocol
AT CRA 7E8	Set CAN receive filter which accepts responses only from engine ECU with CAN ID 7E8
ATRV	Get battery voltage

3.2 Software configuration

Software for data acquisition was programmed in Python3 and installed as system service for automatic startup. Figure 16 depicts the flow diagram of the main program which uses three classes: (a) MSimSerial, (b) MObdSerial, (c) MObdDecoder.

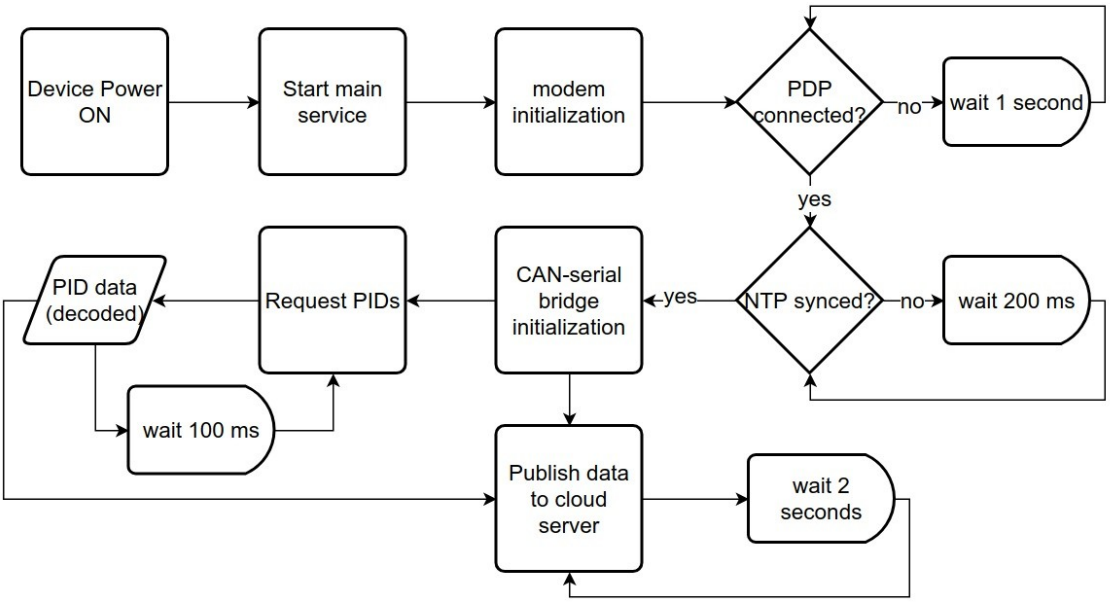


Figure 16. Flow diagram of created system service for publishing OBD-II data to cloud server.

MSimSerial class is used for modem setup – once it has connected to packet data and set system time from NTP (network time protocol) server (all other activity is blocked until then), an instance of MObdSerial class is started in a separate thread which in turn configures CAN-serial bridge and starts requesting data from engine ECU. Publishing data to server without correct timestamps would cause problems later when analysing data – due to network conditions data could arrive out of order. An instance of MObdDecoder class is created within MObdSerial to convert the raw PID data to human readable format and save it as a variable in program memory. After starting MObdSerial thread, MSimSerial will go into an infinite while loop which publishes decoded PID data to cloud server using MQTT protocol. The advantage of using threading is that in data publishing loop we do not introduce delays from requesting PIDs – latest data is instantly available. Additionally a support service (watchdog) was created which monitors the main service and restarts it if necessary.

To optimize the performance of the device unused kernel modules and system services were disabled in the operating system (Appendix 2). Raspi-config utility was used to enable hardware serial on Raspberry Pi and disable login shell over serial, also system time zone was set to UTC. Udev rule was created for CAN-serial bridge for device name persistence (Figure 17).

```
ACTION=="add", SUBSYSTEMS=="usb", ATTRS{idProduct}=="2303",
ATTRS{idVendor}=="067b", ATTRS{version}==" 1.10",
SYMLINK+="obdscanner"
```

Figure 17. Udev rule for CAN-serial bridge in /etc/udev/rules.d/80-rename-obdadapter.rules.

The prototype data collection device is depicted in Figure 18 with the engine turned on.

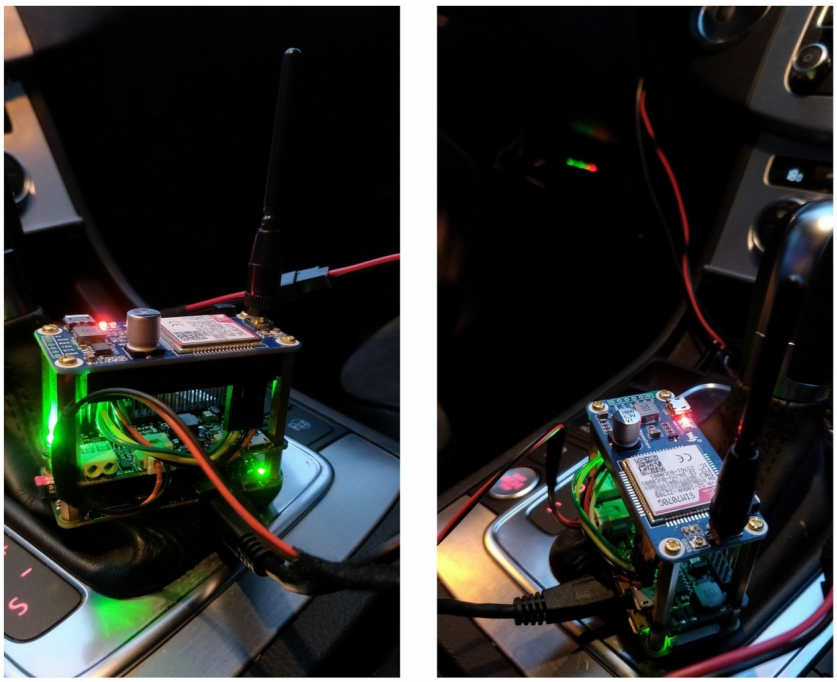


Figure 18. Prototype data collection device.

3.2.1 Cloud server configuration

Cloud server was set up on a virtual private server (VPS) host with specifications listed in Table 5. Additional information about server IP address and location is disclosed in Appendix 3. Access to the server and all configuration was done using secure shell (SSH) tunnel with private-public keypair (password authentication and root login were disabled). Firewall was set up with uncomplicated firewall (UFW) allowing access from Telia Eesti AS allocated dynamic IP range [41] to SSH and MQTT (Figure 19).

Table 5. Cloud server specification.

Operating system	Ubuntu 18.04.5 LTS
RAM	8GB
CPU cores	4
Storage	160GB Nvme SSD
Allocated bandwidth	1Gbps

```
sudo /sbin/ufw allow 9393/tcp
sudo /sbin/ufw allow from 37.157.64.0/18 to any port 1883 proto tcp
sudo /sbin/ufw allow from 46.131.0.0/16 to any port 1883 proto tcp
sudo /sbin/ufw enable
```

Figure 19. Cloud server firewall configuration.

ThingsBoard (3.0.1) open source IoT platform was installed on the server as a backend for database management, device integration and real-time monitoring (Figure 20). In ThingsBoard devices panel a new device named OmniPi_N1 was registered and its access token was used to configure MQTT username.

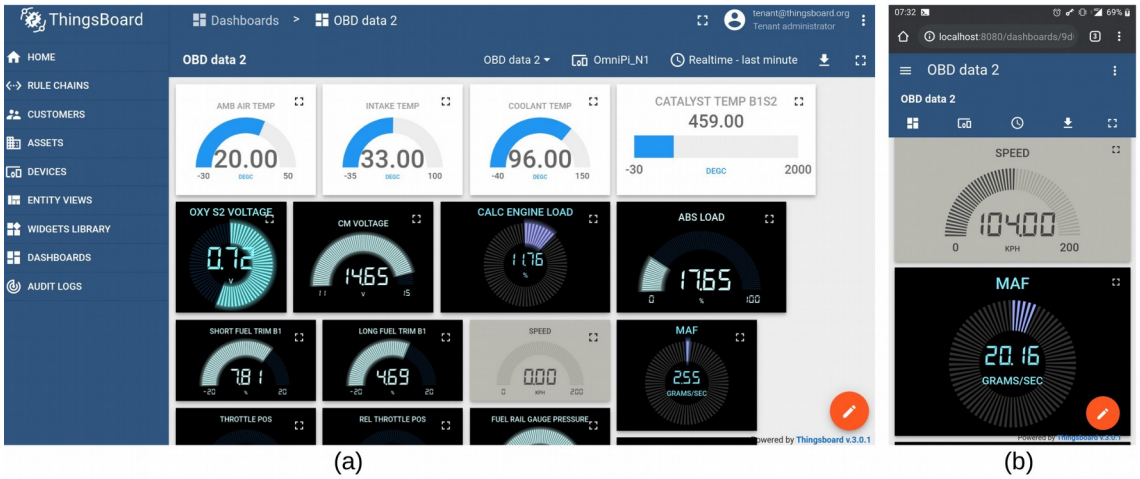


Figure 20. ThingsBoard real-time dashboard view with configurable widgets: (a) desktop view, (b) smartphone view.

In ThingsBoard the way messages (incoming data) are handled is highly configurable using rulechains. There are three types of messages:

- **Post attributes**, which is used for sending device attributes (firmware version, serial numbers etc.) to server. With MQTT API these are published to topic **v1/devices/me/attributes**
- **Post telemetry**, which is used for sending telemetry (data from sensors) to server. With MQTT API these are published to topic **v1/devices/me/telemetry**
- **RPC request**, which is used for listening or sending remote procedure calls (RPC) to server. With MQTT API the device can subscribe to topic **v1/devices/me/rpc/request/+** to listen for server RPCs.

Using message type switches incoming messages can be easily distinguished and processed in separate chains checking for existence fields in data, using scripts for data transformation, saving to database and performing other actions such as raising alarms. To process incoming messages a rule chain (Figure 21) was created to check for existence fields in data and then save data to time series.

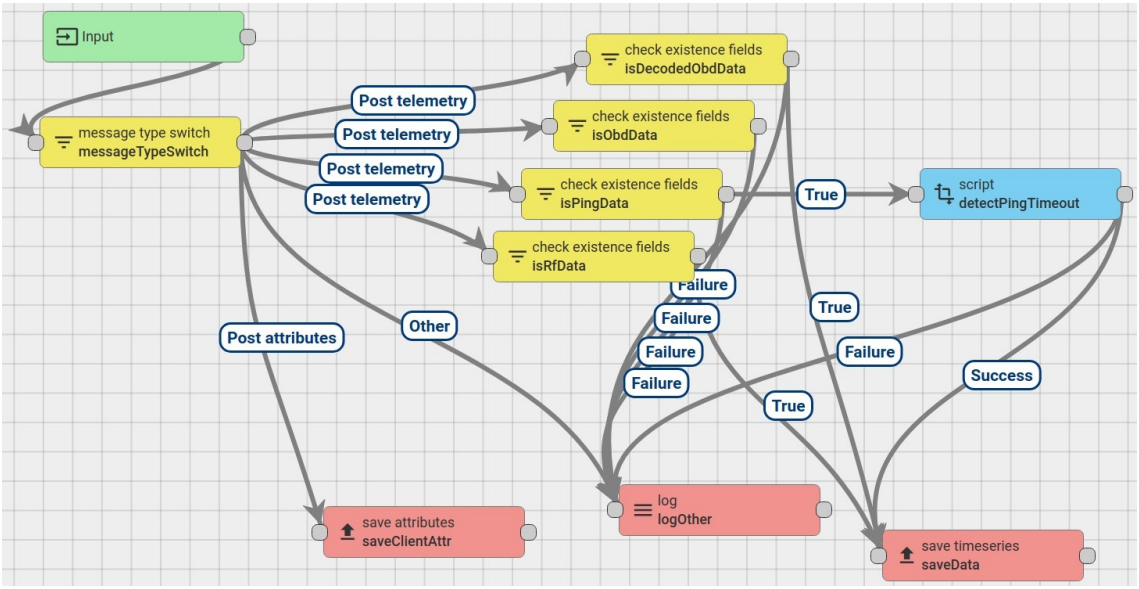


Figure 21. ThingsBoard rulechain for processing incoming data.

Initially ThingsBoard Javascript engine was used for transforming incoming OBD-II data (in hexadecimal) to human readable format, however when looking at saved data on some occasions high signal values were seen which should not have been possible. As a workaround to fix the issue all decoding functions were implemented on the Raspberry Pi itself.

The modem used (SIM7070G) for publishing data to cloud server did not have support for mobility management, as it was based on 3GPP release 14, and at the time of doing this work devices with newer release versions were not available to the author, which presented challenges with persisting the MQTT connection to the server while driving. During testing it was discovered that in some instances the modem would report the MQTT connection state incorrectly and would also hang forever waiting for a proper response for the connection state command. To overcome this problem an additional system service was created for restarting the main service which was triggered by following events (Appendix 4):

- modem has exceeded threshold for reporting invalid transmit power (device is in idle mode for certain time)
- threshold for unsuccessful MQTT connection establishments exceeded
- any AT command issued by the modem did not receive a proper response during 60 seconds
- any unhandled exception occurred during execution in the main program.

3.3 Methodology

In this thesis an implementation for a vehicular data collection device based on a singleboard computer is proposed and it is integrated with an open source IoT platform on a Linux cloud server instance over NB-IoT radio access technology. ML algorithms and statistical methods were used on collected driving data to build models based on following experiments:

1) airflow restriction
2) engine (vacuum) air leak

In the first experiment SVR (support vector regresion), RF and MLR (multiple linear regression) ML models for regression analysis were created and compared with simple linear regression model from single independent variable. The accuracy of the models were compared with the coefficient of determination, R^2.

The second experiment used histograms, cumulative distribution function (CDF) and RF classification for fault detection. In addition an ARIMA model was created from collected and simulated data for forecasts.

Development of the data collection device was based mainly on one vehicle – no emulators were used as it was necessary to learn how vehicle sensors react to real driving conditions and the limitations of using NB-IoT in a mobile application. Data was mostly collected in normal driving conditions while monitoring real-time data from cloud server using a smartphone – the aim was not to gather data in some specific way as done by car workshops where a vast amount of data is collected over a short time period and observed under certain conditions. No attempts were made to store the gathered data locally or for retransmissions in case of packet loss – it is assumed that over a longer time period enough samples are collected and stored on the cloud server. Another reason for collecting data this way was to make sure that the received data on cloud server was valid (included correct timestamps) and could be used as a data source for experiments. Data was collected from all supported standard OBD-II service mode 1 PIDs, in total 29.

Software for the data collection device was programmed in Python3 using standard software modules. RPi.gpio and pyserial modules were used for power control and serial communication with modem and ELM327 adapter. ML and statistical models were also created in Python3 with the following modules:

- scipy
- scikit-learn
- statsmodels

4 Experimental results

In this chapter two experiments were conducted with the device configured in previous step exploring airflow restriction and engine air leak. An overview is given for the sensors used in each experiment and conclusions from modelling results at the end. Finally, the reliability of the system architecture is estimated with packet loss rates obtained from various driving conditions.

4.1 Experiment 1 – airflow restriction

The goal of the following experiment was to evaluate how ML models could be applied to detect an anomaly with airflow rate. Sensors used for this experiment are listed in Table 6.

Table 6. Experiment 1 OBD-II PIDs.

OBD-II MODE + PID	**PID name**
01 10	Mass airflow
01 23	Fuel rail gauge pressure
01 0C	Engine speed (RPM)
01 0D	Vehicle speed
01 0F	Intake air temperature
01 11	Throttle position
01 49	Accelerator pedal position d

Throttle position sensor (TPS) is a sensor inside throttle body which gives feedback to engine control systems about the opening of throttle valve by generating a voltage signal. Throttle valve controls the airflow into the engine, it is linked to a DC motor which opens or closes the valve and its rotary motion is proportional to the opening

angle of the valve. Throttle body is equipped with two springs which are used in case of system failure to allow minimal airflow to the engine [42].

Accelerator pedal position sensor (APP) is used for providing torque demand to the engine control systems from the driver. For safety concerns two APP sensors are used and the output signal generated by both sensors are continuously compared – in case of an inconsistency between the signals, the systems will go into failsafe mode. Earlier vehicles had a direct mechanical link (cable) between accelerator pedal and throttle body but nowadays the opening of throttle valve is controlled electronically from APP and other engine systems (such as cruise control). In addition to input from APP sensor engine control system uses two maps for throttle valve control: (a) torque demand pedal map, (b) mass airflow map [42].

Mass airflow (MAF) sensor sits between air filter box and throttle body and is used by engine control systems to calculate the the mass of air (grams/sec) entering the engine. Typically a "hot wire" type MAF sensor is used where the amount of current needed to heat the wire is proportional to the airflow as air passing over the wire will cool it down [1]. A dirty MAF sensor will cause drivability problems as it will under report the airflow, in case of a faulty MAF sensor it could also over report the airflow.

Engine speed sensor is used by engine control system to determine the rotary speed of engine crankshaft wheel. Engine speed is expressed in revolutions per minute (RPM).

Fuel rail gauge pressure sensor signal is used by engine control system to measure the engine fuel pressure relative to atmospheric pressure (provided by high pressure fuel pump) [43]. An increase in engine speed and airflow will require higher fuel pressure to be provided to fuel injectors to maintain correct air-fuel ratio (stoichiometric) – for gasoline engines air-fuel ratio is 14,7:1 (14,7kg of air per 1kg of fuel).

4.1.1 Data collection

Data from sensors was collected during normal driving with: (i) new air filter, (ii) fault condition 1, (iii) fault condition 2. Data collection period for each scenario was one week and included driving in the city and highway (Figure 22).

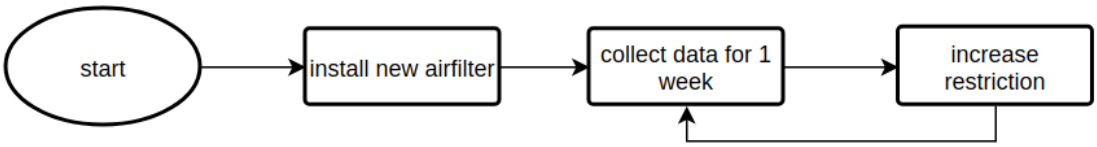


Figure 22. Experiment 1 data collection.

For fault condition 1, cardboard pieces of various size were placed under the air filter and for fault condition 2 aluminium foil tape was used to mask the air filter for additional restriction (Figure 23).

(a) (b)

Figure 23. Test setup: (a) fault condition 1, (b) fault condition 2.

With fault condition 1 there was a moderate lack of acceleration while driving compared to normal conditions but no warning lights were displayed. With fault condition 2 the lag under acceleration was significant, also a warning light regarding an error with tire pressure monitoring was displayed multiple times which did not seem related at the time, however when air filter restriction was removed later it did not appear again. Later, when checking engine ECU for fault codes, a pending fault code P0101 was discovered as show on Figure 24 – MAF sensor (G70) with an implausible signal [44]. When a fault is discovered for the first time, it is stored as a pending code, and if also detected during the next drive cycle it will be stored as a confirmed code and engine light will be illuminated, otherwise the pending fault will be deleted from memory. In this case engine light was not illuminated.

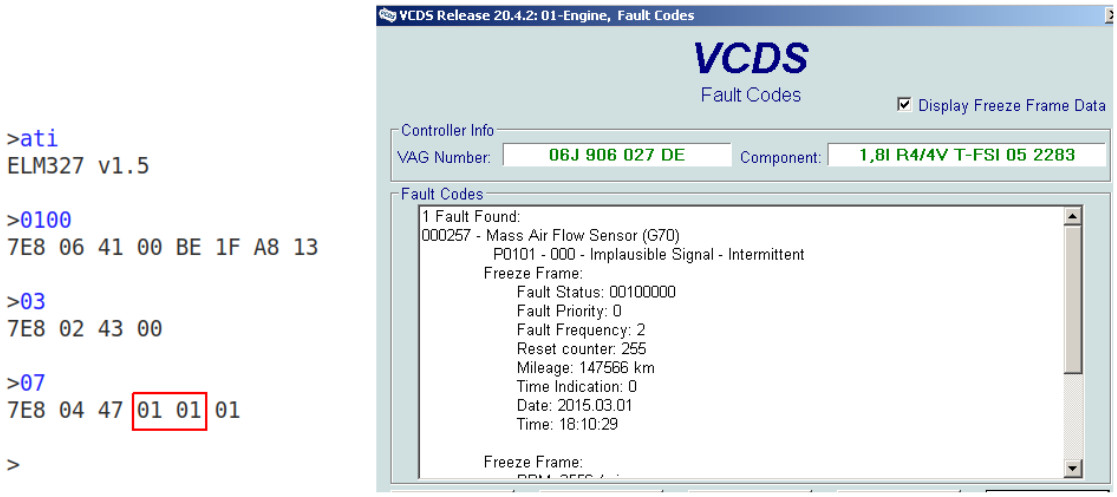


Figure 24. Fault code P0101: (a) ELM327, (b) VCDS.

4.1.2 Data processing

Individual PIDs were plotted against MAF to visualize the relationship between sensors. Collected samples were split into training (70%) and test (30%) sets – Pearson and Spearman correlation coefficients were calculated for each training set. Simple and robust (Theil-Sen, Siegel) linear regression models (estimators) were created based on training data and plotted for test data, R^2 score was used to compare the accuracy of the models. Figure 25 depicts data collected with a new air filter during one week – in total there were 9553 samples and the majority of them were collected under normal driving conditions which can be read from throttle position (12-30%) and engine speed (800-2500 RPM) ranges. Throttle position and fuel rail pressure seem to have a better linear relationship with MAF than engine speed. It is noticeable that robust linear regression models are less sensitive to outliers in data than simple linear regression, especially from throttle position plot. From accelerator pedal position plot it can be seen that correlation with throttle position is quite low – this is mainly because the the plot includes driving data with cruise control enabled (when accelerator pedal is released it reports constantly 14.9-15.3%).

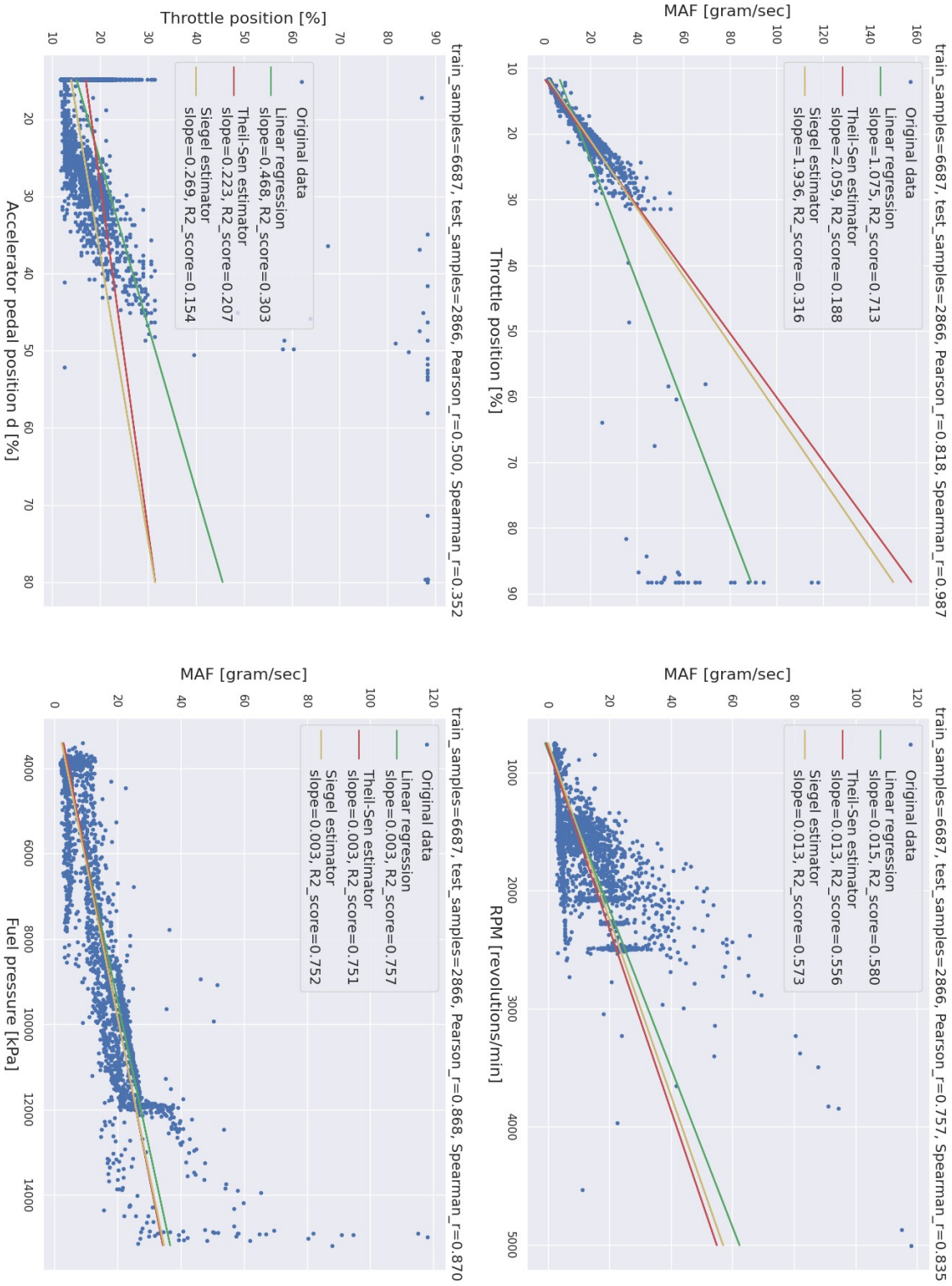


Figure 25. Linear regression plots for original data (no air filter restriction).

To obtain samples for moderate to heavy acceleration (impact of restricted airflow is noticeable under increased engine load) initially it was considered to use oxygen

(lambda) sensor voltage which under heavy acceleration should report more than 0.8V, however while driving and observing the recorded data it appeared that oxygen sensor along with fuel trims do not update at the same frequency as throttle position or fuel pressure, therefore they were not used. PIDs such as absolute and calculated engine load are available but these are calculated directly based on MAF sensor in engine ECU. After removing the outliers from original data (using throttle position, accelerator pedal position and vehicle speed) it can be seen in Figure 26 that linear and robust regression models give a more unified approximation for resulting 3271 samples. Vehicle speed and intake air temperature were also plotted individually but were eliminated from further analysis – vehicle speed correlation coefficient was below 0.3 and intake air temperature showed negative correlation (air density decreases with increased air temperature, therefore mass airflow will be negatively affected).

From Figure 26 it was concluded that based on engine speed alone it was not possible to predict mass airflow very accurately, however it could be improved by including fuel rail pressure and throttle position.

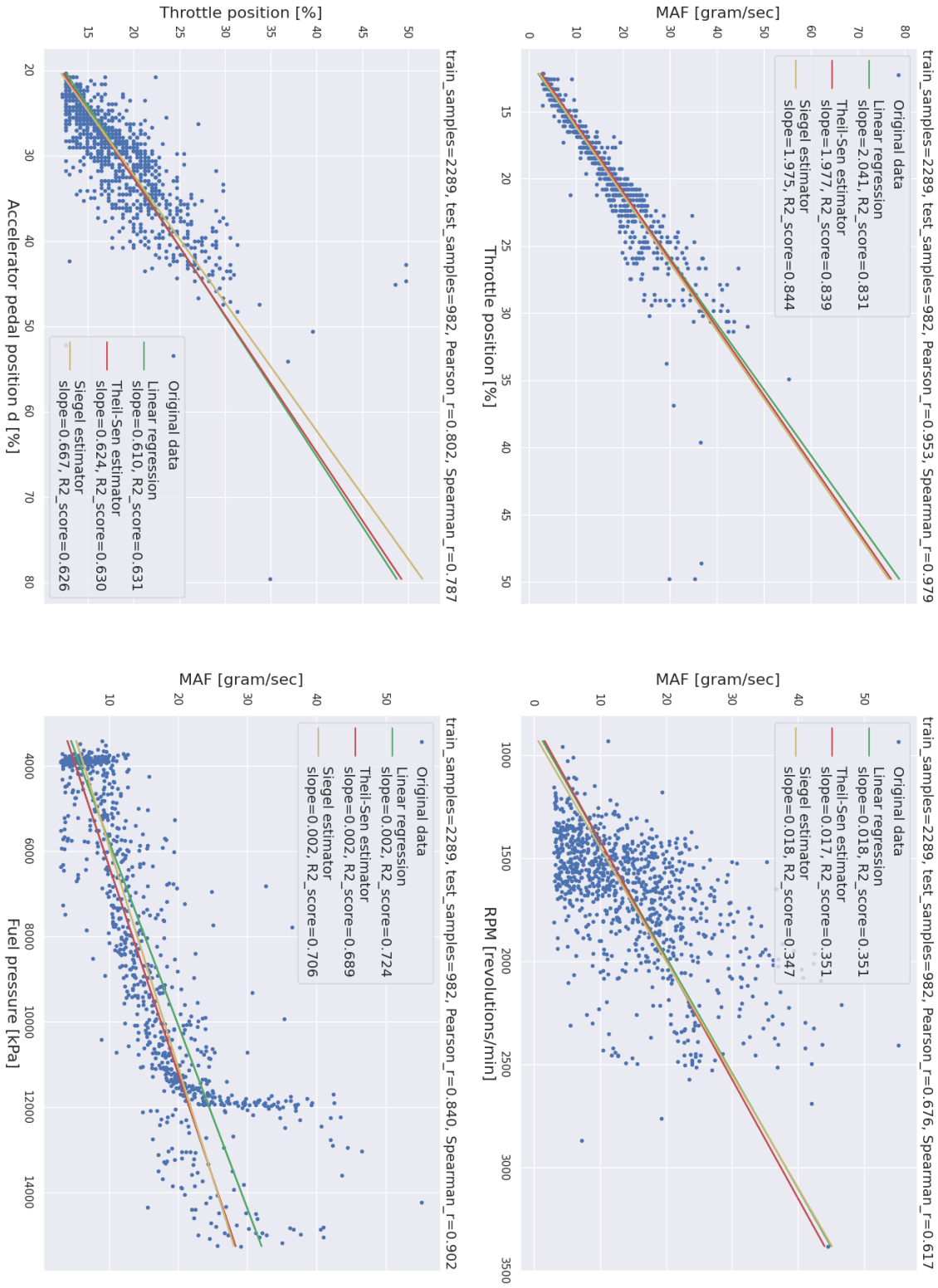


Figure 26. Linear regression plots for preprocessed data (no air filter restriction).

4.1.3 Modelling results

ML models were trained on the processed datasets (70% of samples for training data) using MLR, SVR and RF algorithms – SVR model used radial basis kernel function with scaling applied to data, RF model used 10 estimators. Figure 27 compares the prediction accuracy of the three models on test samples using R^2 score – SVR and RF both achieved close to 97%, MLR fared slightly worse with 93%. For all the created models we can observe that the residuals are scattered around 0 without forming any distinct shapes or patterns which is what we want to see.

Models were created based on dataset with new air filter (no added restriction) for MLR, SVR, RF (using throttle position, fuel pressure and engine speed for input) and simple linear regression (using only throttle position for input). Datasets with added restriction were used as inputs for created models and R^2 score was calculated in each case. To validate the created models additional samples were gathered with unrestricted air filter scenario which were not used for training and testing the model. The results are summarized in Table 7.

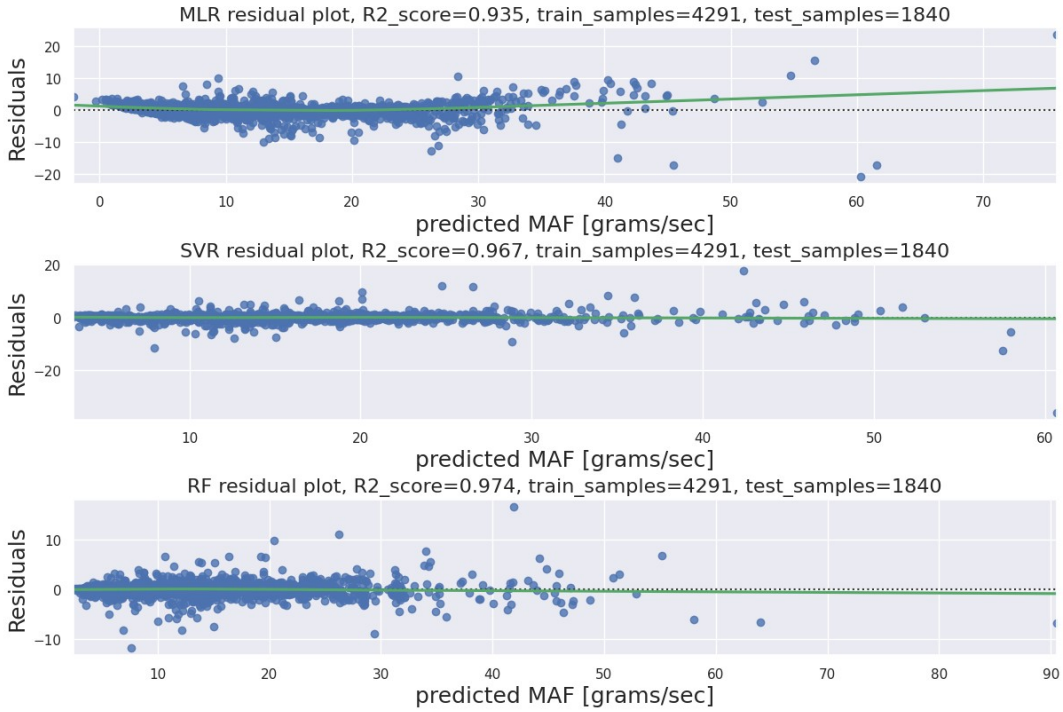


Figure 27. Residual plots comparison for MLR, SVR and RF models for MAF prediction using fuel rail pressure, throttle position and engine speed as input features with no air filter restriction.

Table 7. RF, SVR, MLR and simple linear regression R^2 score comparison.

Restriction amount	**Total samples (used samples)**	**RF R^2 score**	**SVR R^2 score**	**MLR R^2 score**	**simple linear regression R^2 score**
No restriction	9553 (3271)	0.97	0.97	0.93	0.88
“25%” cardboard piece	9844 (3625)	0.96	0.97	0.93	0.88
“75%” cardboard piece	5241 (2022)	0.97	0.97	0.93	0.89
“100%” cardboard piece	7281 (2788)	0.96	0.95	0.92	0.86
Aluminium foil tape	3329 (2098)	0.82	0.84	0.76	0.81

4.1.4 Conclusion from experiment

During this experiment we collected data with various amounts of restriction in air intake and stored it on Linux cloud server using NB-IoT. Data from cloud server was preprocessed to retain samples which were relevant to detecting issues with airflow and finally ML and simple linear regression models were used to predict mass airflow. Performance of the created models was compared with coefficient of degradation. It appeared that data collected with fault condition 1 (using various size cardboard pieces as restriction) did not have big enough of an impact on the prediction score to be clearly identified as a fault because deviation from normal mode was minimal. No warnings or fault codes were stored in engine ECU while performing tests for fault condition 1, although lack of acceleration was notable with larger cardboard pieces – this could be because cardboard pieces used as restriction were able to move around in the air filter box and still allowing adequate airflow for normal driving. With fault condition 2 (using aluminium foil tape as restriction), however, there was a severe loss of acceleration caused by limited airflow and this was also confirmed with increased deviation of the prediction scores for all models. Fault code P0101 related to implausible MAF sensor

signal was later found on engine ECU. All ML models that were used for this experiment were able to predict airflow more accurately than simple linear regression with only one input variable – SVR and RF earned highest prediction scores with 97%. A rolling ARIMA model based on obtained prediction scores could be used to forecast the RUL (demonstrated with the next experiment). Overall, from this experiment it was quite surprising to learn how limited the airflow needed to be in order for a fault code to be set by the vehicle and a significant change in the prediction scores to be seen.

4.2 Experiment 2 – air leak

The objective of this experiment is to provide an indicator which could be used to detect a gradually developing engine air (vacuum) leak. OBD-II PIDs used in this experiment are listed in Table 8.

Table 8. Experiment 2 OBD-II PIDs.

OBD-II MODE + PID	**PID name**
01 03	Fuel system status
01 05	Engine coolant temperature
01 06	Short term fuel trim (bank1)
01 07	Long term fuel trim (bank1)
01 0C	Engine speed
01 0D	Vehicle speed
01 0E	Timing advance
01 10	Mass airflow

Fuel system status indicates the current operating mode of the engine – either open loop or closed loop mode. In open loop mode oxygen (lambda) sensors are not used as feedback for air-fuel ratio (stoichiometric) correction because the sensors need to be heated up before they start sending valid signals to engine ECU. The purpose of oxygen sensors is to provide engine ECU with information regarding unburned oxygen levels in exhaust gases, based on which engine ECU will know if air-fuel mixture is rich (too

much fuel) or lean (not enough fuel). Once oxygen sensors are up to temperature (in modern vehicles a heating element is included which allows to heat them up as quickly as 19-30 seconds) the engine will go into closed loop mode and include oxygen sensors for fuel injection timings (pulse widths) adjustment [1].

Engine coolant temperature (ECT) sensor provides a voltage signal to engine ECU, based on which the temperature of engine coolant liquid can be calculated – voltage drop is measured by voltage sensing circuit across a thermistor and converted to a temperature value [1]. ECT sensor is used during engine warm-up (open loop mode) for air-fuel enrichment by increasing fuel injector pulse width [45].

Short term fuel trim (STFT) is calculated from oxygen sensors and is used for fuel injection corrections (to maintain air-fuel ratio) in closed loop mode only. STFT is expressed in percentages – (i) 0% indicates that no corrections are needed, (ii) positive STFT values indicate a rich command (engine is running lean and more fuel is needed), (iii) negative STFT values indicate a lean command (engine is running rich and less fuel is needed). STFT readings in open loop mode are invalid and should be ignored. STFT values for normal operation should generally be in range -10% to 10% [1].

Long term fuel trim (LTFT) is a learned value and used together with STFT for fuel injection corrections – difference being that it is adapted over a longer time period than STFT. The purpose of LTFT is to keep STFT values close to 0% – for example if STFT is constantly showing values above 10%, LTFT will be increased (as compensation) which allows STFT to be lowered closer to 0%. LTFT, as opposed to STFT, is also used in open loop engine mode. Normal operation range for LTFT is from -10% to 10%. Absolute value of total fuel trim (FT_{total}), in equation (3), exceeding 15% indicates an abnormal operation [1].

$$FT_{total} = STFT + LTFT \tag{3}$$

Ignition timing advance (TA) maps are used by engine ECU to set ignition timings based on various engine working conditions to optimize engine performance and fuel economy. TA is specified as degrees before top dead center (BTDC) of the compression stroke – in order for air-fuel combustion to happen at desired position (10 to 20 degrees after top dead center during the combustion stroke), spark must be given early because

of a 3ms ignition delay. When ignition happens too late maximum combustion efficiency is not achieved, however when too early it results in detonation [1].

4.2.1 Data collection

Data from sensors was collected during normal operation and thereafter by creating air leaks. Air leaks are most easily detectable with lower engine loads because the ratio of unmetered airflow to measured airflow reported by MAF sensor will be higher. Oil rod was removed from the car and a fuel hose with a ball valve was attached – various degrees of air leaks could then be simulated by opening the ball valve (Figure 28).

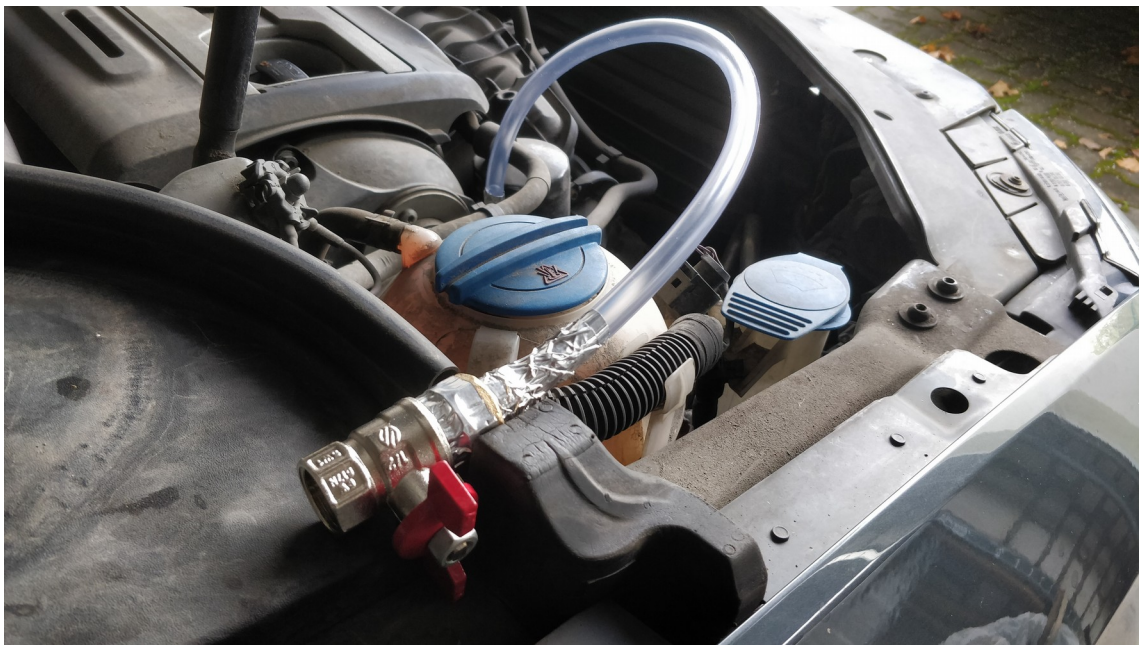


Figure 28. Setup for air leak experiment.

The experiment lasted for 90 minutes and the air leak amount was increased in 15 minute intervals, during which the car was driven for 5 minutes with slow speeds (below 50 km/h) and standing still for 10 minutes (Figure 30). Air conditioning was turned off to reduce engine load during this experiment and the engine had been running long enough earlier to bring it into closed loop operating mode. Observations made while doing the experiment are summarized in Table 9.

Table 9. Observations from air leak experiment.

Interval	**Observations**
int1 (14:30-14:45)	There is no air leak and engine is running normally. LTFT is below 10%.
int2 (14:45-15:00)	Small air leak is present but LTFT is still in normal range (around 10%).
int3 (15:00-15:15)	Air leak is increased and LTFT is just below 15%
int4 (15:15-15:30)	Air leak is increased further. During first 5 minutes there is no change in LTFT but after driving LTFT goes above 35% when car is standing still.
int5 (15:30-15:45)	Air leak is increased again and straight away the engine starts idling roughly with increased RPM. During first 5 minutes there is no change in LTFT but after driving LTFT goes above 45% when car is standing still.
int6 (15:45-16:00)	Air leak is eliminated. LTFT is just below 45% during first 5 minutes while STFT is at -22% indicating a rich condition. The last 5 minutes show engine has adapted to the rich condition and LTFT is lowered to 13%.

During the experiment a trend with LTFT and TA could be observed as depicted in Figure 29. Collected data confirms that the effect of air leak is most noticeable with low engine loads (standstill).

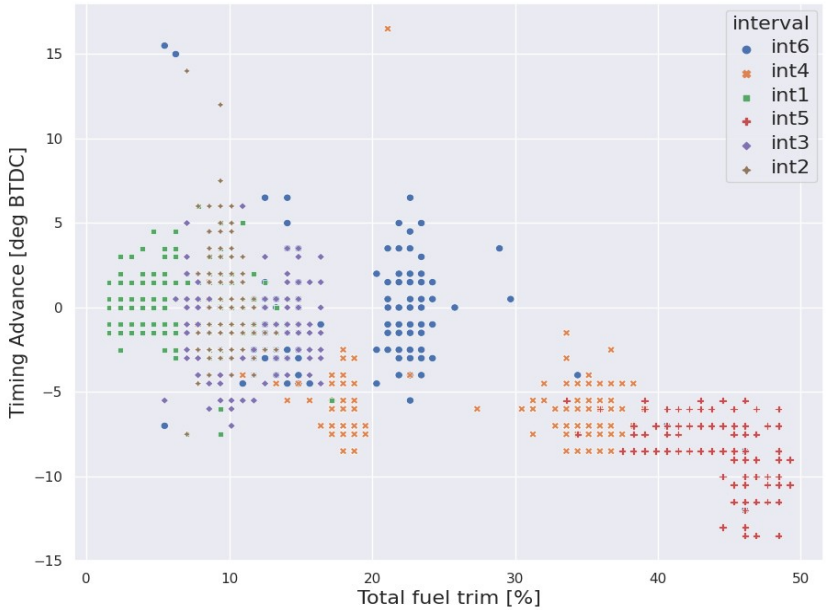


Figure 29. Data from air leak experiment during engine idle.

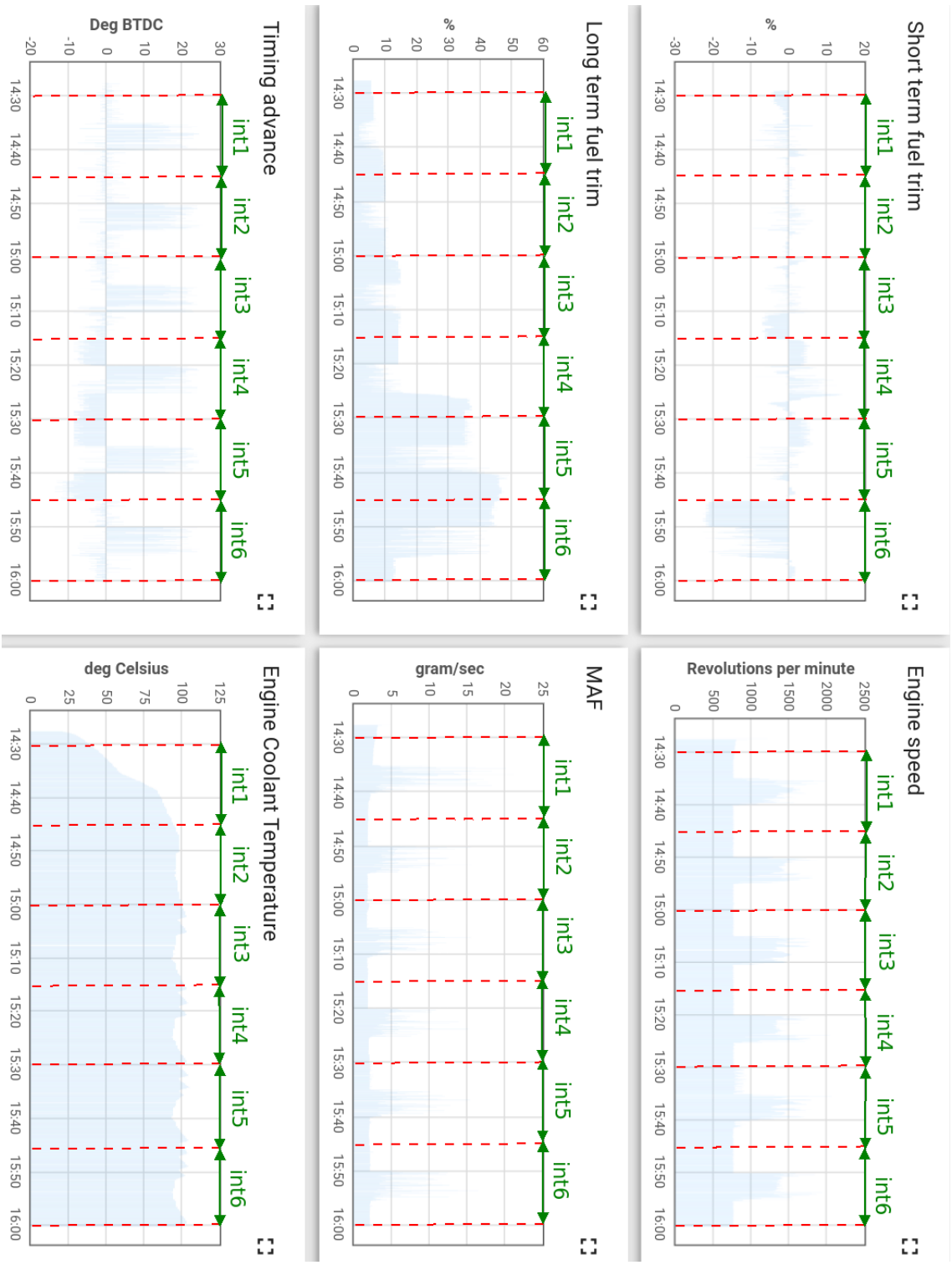


Figure 30. Data from STFT, LTFT, TA, RPM, MAF and ECT sensors during air leak experiment.

According to [46] TA is retarded because of the greater combustion rate of lean air-fuel mixture (increased oxygen from air leak) which requires less timing. Driving data with slow speeds showed a significant change for LTFT only with the last interval where air

leak was at its highest. MAF showed just a slight increase within the last air leak interval caused by rough idling (increased engine speed). At the end of the experiment engine ECU was scanned for fault codes but none were found, although LTFT was clearly outside the boundaries of normal operation – this was attributed to the fact that the engine ECU needs to detect this fault condition during multiple drive-cycles.

4.2.2 Modelling results

Experimental data collection shows that most relevant data for detecting an air leak can be obtained from fuel trim and timing advance during engine idle – other parameters such as RPM, MAF and ECT showed little to no change. Collected data was filtered based on vehicle speed and fuel system status to indicate whether the car was moving or stationary and engine operating in closed loop mode.

The following two methods were used for fault detection:

1. Histograms for FT_{total} and cumulative distribution function (CDF)
2. RF classifier

In the first method a set of probabilities was calculated for each observation interval as given in equations (4). The probabilities were obtained from CDF, which was constructed from FT_{total} histogram data, and describe how FT_{total} is distributed between normal and abnormal operating ranges. The probabilities should be saved as time series for each day when the vehicle was used (or a sufficient amount of samples were gathered) as illustrated in Figure 31.

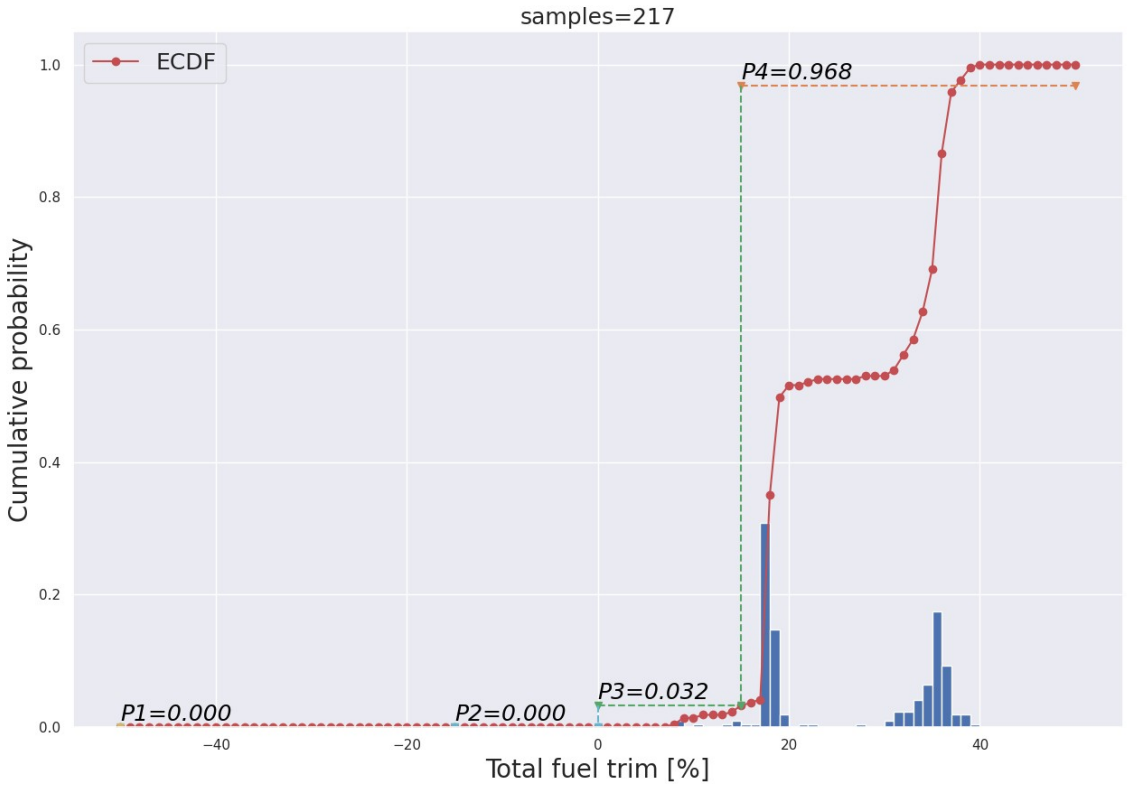


Figure 31. Total fuel trim histogram and empirical CDF on collected driving data.

$$\begin{cases} P_1 = P(FT_{total} \le -15) = CDF(-15), \\ P_2 = P(-15 < FT_{total} \le 0) = CDF(0) - CDF(-15), \\ P_3 = P(0 < FT_{total} \le 15) = CDF(15) - CDF(0), \\ P_4 = P(FT_{total} > 15) = 1 - CDF(15) \end{cases} \quad (4)$$

Table 10 summarizes the obtained FT_{total} probabilities. Under normal operating conditions the sum of probabilities P_2 and P_3 should be close to 1, which it is before creating leaks and also during leaks 1 and 2 because FT_{total} is mostly below 15%. During leak 3 approximately 96% of collected samples were outside normal operating range as indicated by P_4 which increased to 100% with leak 4 when car was standing still. From samples that were gathered while car was moving at slow speeds we can observe that it is slower to react to changing conditions with P_4 probabilities for leaks 3 and 4 for being 22% and 85%. FT_{total} did not return to normal values straight away when air leak was eliminated due to engine ECU needing more time to adapt.

Table 10. Total fuel trim probabilities.

	Samples$_{idle}$ (Samples$_{moving}$)	**P$_{1_idle}$ (P$_{1_moving}$)**	**P$_{2_idle}$ (P$_{2_moving}$)**	**P$_{3_idle}$ (P$_{3_moving}$)**	**P$_{4_idle}$ (P$_{4_moving}$)**
No air leak (int1)	218 (99)	0 (0)	0 (0.293)	0.995 (0.707)	0.005 (0)
Air leak 1 (int2)	221 (88)	0 (0)	0 (0.011)	1 (0.989)	0 (0)
Air leak 2 (int3)	212 (98)	0 (0)	0 (0)	0.915 (0.980)	0.085 (0.020)
Air leak 3 (int4)	217 (90)	0 (0)	0 (0.011)	0.032 (0.767)	0.968 (0.222)
Air leak 4 (int5)	218 (83)	0 (0)	0 (0)	0 (0.145)	1 (0.855)
No air leak (int6)	215 (89)	0 (0)	0 (0)	0.484 (0.663)	0.516 (0.337)

In the second method binary RF classifier was used where for simplicity observation intervals 1-3 and 6 were labelled as 0 and intervals 4-5 were labelled as 1 indicating a fault condition. The reason for such labelling is that small air leaks are more difficult to detect as illustrated in Figure 29. Only samples during which the vehicle was not moving (1301) were used and 70% of those were used for training RF classifier (using 10 estimators) while the remaining 30% were used for testing the model. The classification accuracy of the model was 98%, as shown in Figure 32.

In this experiment time series forecasting could be based on either P_4 probability from method 1 or fault probability, equation (5), in second method

$$P_{fault} = \frac{S_{fault}}{S_{total}} \tag{5}$$

where S_{fault} is number of samples classified as fault and S_{total} is the total number of used samples. Fault probabilities for both methods are shown in Figure 33 which are based on collected driving data – probabilities were calculated only for days which had at least 50 relevant samples. Both methods show a similar level of performance.

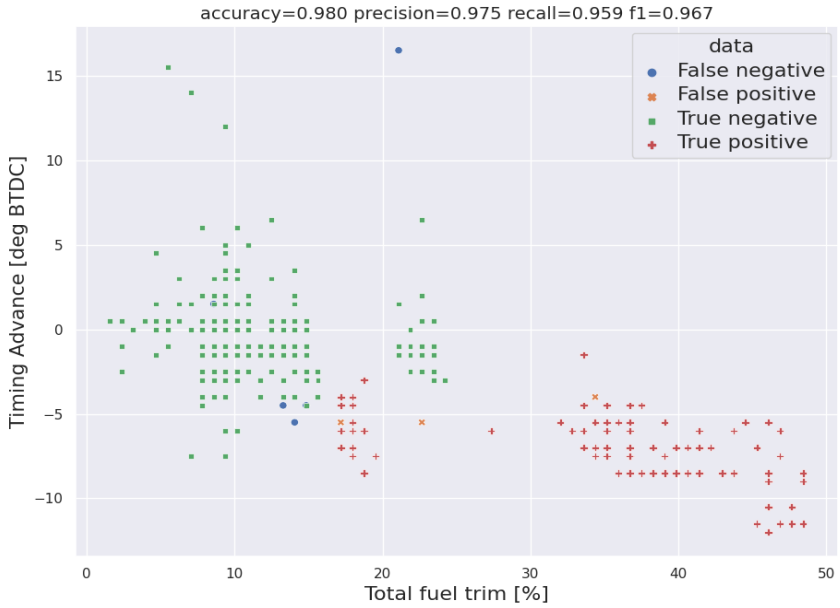


Figure 32. RF classification for air leak detection during engine idle.

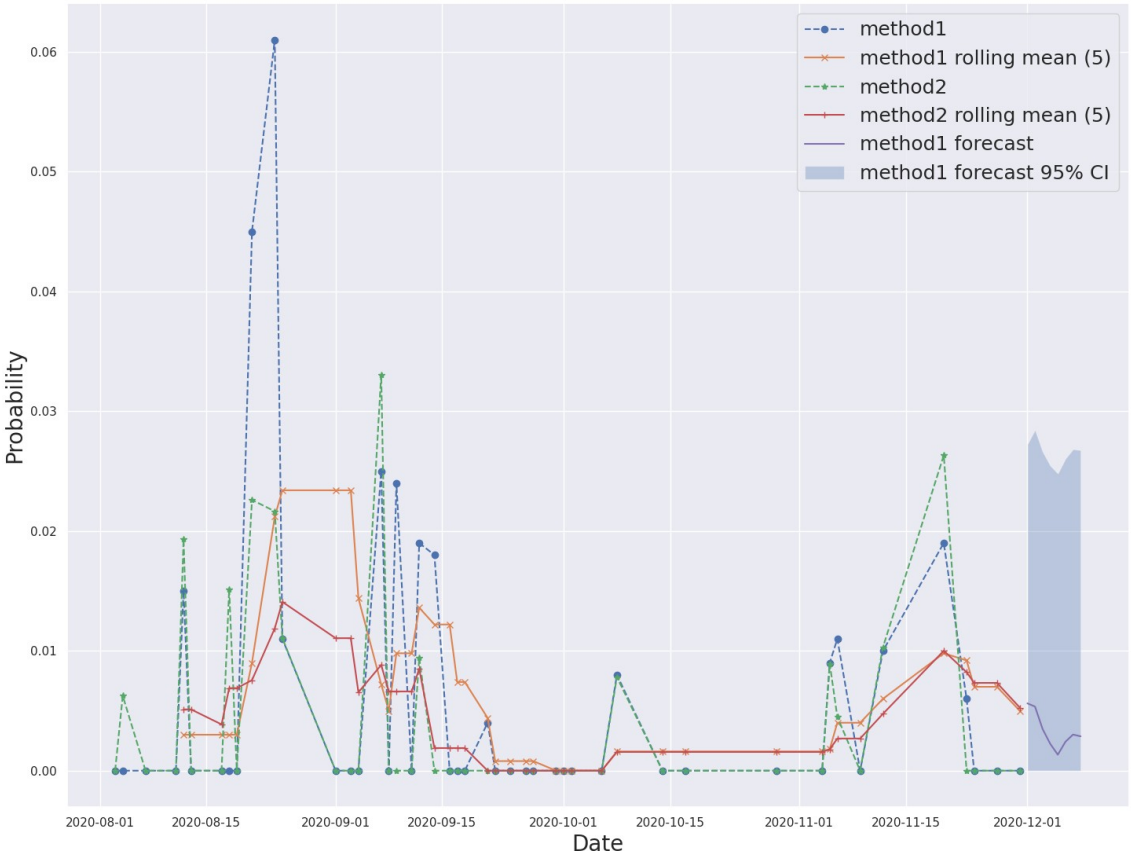


Figure 33. Fault probability comparison of method1 and method2 based on collected driving data.

Figure 34 shows a slowly increasing air leak based on P_4 probability where simulated data was used to train and test an ARIMA model (p=5; i=1; q=1). MSE was used to compare the fit of ARIMA model to rolling moving average of five previous observations and obtained scores were similar. ARIMA model should be recreated when new observations become available to make new forecasts (creating a rolling model).

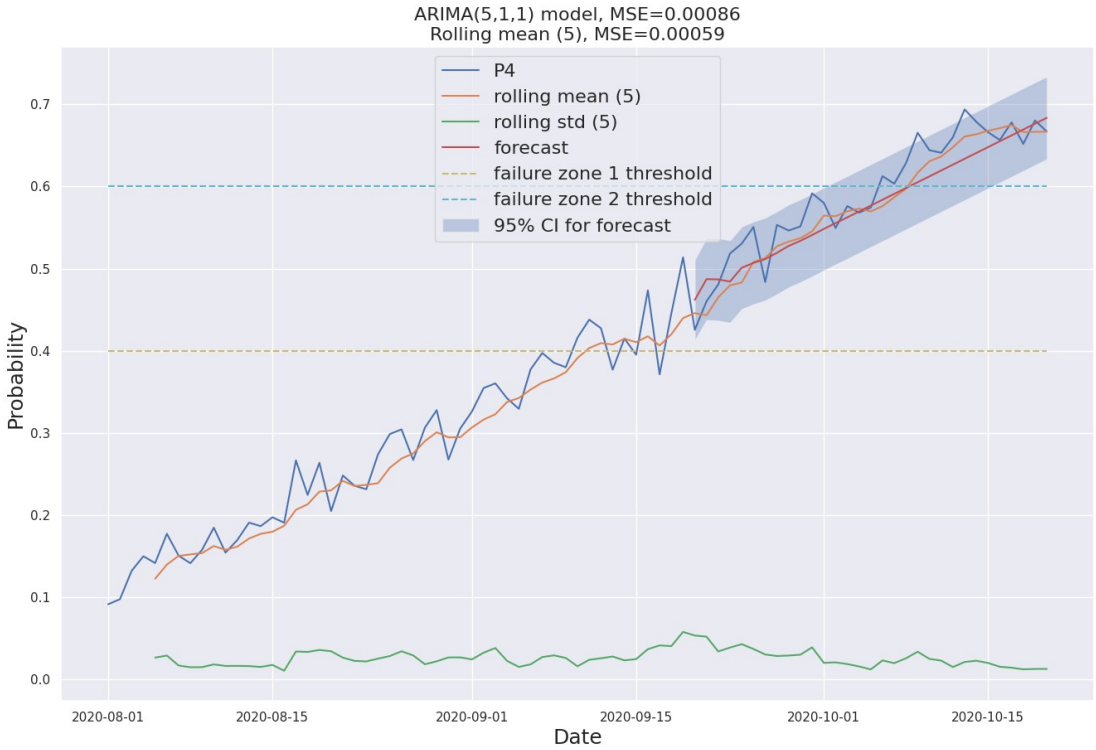


Figure 34. Air leak forecasting on simulated data using ARIMA(5,1,1) model.

4.2.3 Conclusion from experiment

Due to the way the engine works, air leaks have most impact under low engine loads (idling), therefore most significant data samples for analysis can be obtained from standstill. Data for this experiment was collected with our configured device and uploaded to cloud server. Various degrees of air leaks were created in the engine and their influence was observed on parameters such as STFT, LTFT and TA which were used for creating two models for fault detection. The first model used CDF obtained from total fuel trim histograms to calculate probabilities indicating normal and abnormal states. The second model used RF classification which resulted in 98% accuracy. Based on an ARIMA model created from driving and simulated data it is

shown how the used methods for fault detection could be used for time series forecasting. Alternatively, the same approach could be used to detect a problem with air-fuel mixture getting increasingly rich (indicated by negative fuel trim values) which could be caused by leaking fuel injectors or faulty fuel pressure regulator.

4.3 Key performance indicator evaluation

In addition to data from OBD-II also ping latency and NB-IoT radio signal parameters were collected and uploaded to cloud server while driving as depicted in Figure 35.

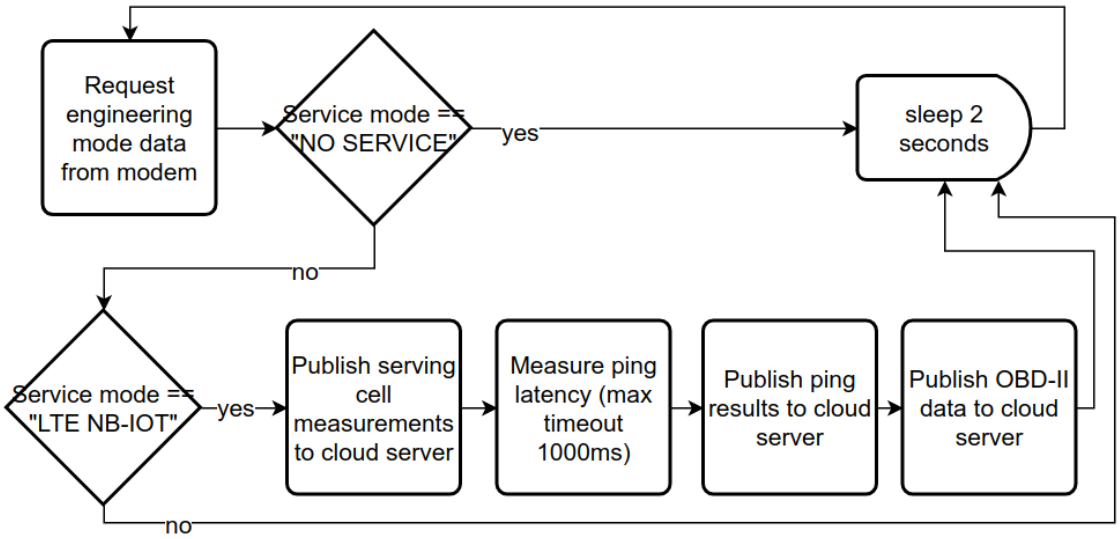


Figure 35. Flowchart for publishing ping latency, NB-IoT serving cell and OBD-II data to cloud server using SIM7070G modem.

Ping latency was measured from our cloud server (hosted physically in United Kingdom) by sending out one ping request at a time with packet size of **16** bytes and maximum delay of **1000**ms. Following serving cell NB-IoT parameters were collected: earfcn, pci, rsrp, rssi, rsrq, sinr, tac, cellid, mcc, mnc, tx power. Ping requests and attempts to publish data to cloud server were made only when service mode was “LTE NB-IOT”. Timestamps (in milliseconds) were added to collected data on the NB-IoT device, saved locally on microSD card and compared to data received on the cloud server to calculate packet loss. Data, formatted as JSON string, was published to cloud

server using MQTT protocol. Table 11 lists the parameters used for estimating the performance of NB-IoT and in Table 12 the results are presented for four scenarios:

1. IoT device is not moving and inside building (no OBD-II related data is sent)
2. IoT device is not moving and inside car
3. IoT device is inside car and driven in the city at moderate speeds
4. IoT device is inside car and driven at highway speeds

Table 11. Parameters for estimating the performance of NB-IoT.

Parameter	**Description**
Duration (sec)	Total duration (in seconds) of the measurement scenario
$Ping_{TX\_count}$, $Ping_{RX\_count}$	Total number of published/received ping packets
$Ping_{lost\_count}$	Total number of lost ping packets
$Ping_{timeout\_count}$	Total number of received ping packets with ping delay greater than 1000ms
$Ping_{timeout\_rate}$	$Ping_{timeout\_count}/Ping_{RX\_count}$
$Ping_{avg\_delay}$ (ms)	Average ping delay (in milliseconds) for received packets ($Ping_{RX\_count}$ - $Ping_{timeout\_count}$) where ping delay is less or equal to 1000ms
$Ping_{avg\_size}$ (bytes)	Average content length (bytes) of published ping packets
$Ping_{Tot\_TX}$, $Ping_{Tot\_RX}$ (bytes)	Total content length (bytes) of published/received ping packets
$Signal_{TX\_count}$, $Signal_{RX\_count}$	Total number of published/received NB-IoT signal packets
$Signal_{lost\_count}$	Total number of lost NB-IoT signal packets
$Signal_{avg\_size}$ (bytes)	Average content length (bytes) of transmitted NB-IoT signal packets
$Signal_{Tot\_TX}$, $Signal_{Tot\_RX}$ (bytes)	Total content length (bytes) of published/received NB-IoT signal packets
OBD_{TX_count}, OBD_{RX_count}	Total number of published/received OBD-II packets
OBD_{lost_count}	Total number of lost OBD-II packets
OBD_{avg_size} (bytes)	Average content length (bytes) of published OBD-II packets

Parameter	Description
OBD_{Tot_TX}, OBD_{Tot_RX} (bytes)	Total content length (bytes) of published/received OBD-II packets
T_{diff_max} (ms)	Maximum time difference (in milliseconds) between consecutive service mode states (“NO SERVICE” or “LTE NB-IOT”)
$Count_{no\_service}$, $Count_{in\_service}$	Total count of service mode status for NB-IoT (“NO SERVICE” or “LTE NB-IOT”)
Tot_{TX_count}, Tot_{RX_count}	Total number of published/received packets
Tot_{TX}, Tot_{RX} (bytes)	Total content length (bytes) of published/received packets
$Tot_{pkt\_loss\_rate}$	Total packet loss rate: (Tot_{TX_count} - Tot_{RX_count})/Tot_{TX_count}
$T_{no\_service\_rate}$	(2*$Count_{no\_service}$)/Duration
$T_{no\_service\_max}$ (sec)	Maximum duration (in seconds) during which modem reported “NO SERVICE”

Table 12. Obtained metrics for various driving conditions with NB-IoT device.

	Inside building (no OBD data)	Inside car (no movement)	City driving	Highway driving
Duration (sec)	1795	1765	1596	955
$Ping_{TX\_count}$	702	653	530	285
$Ping_{RX\_count}$	702	653	529	285
$Ping_{lost\_count}$	0	0	1	0
$Ping_{timeout\_count}$	0	97	221	147
$Ping_{timeout\_rate}$	0%	14.854%	41.776%	51.578%
$Ping_{avg\_delay}$ (ms)	470	428	453	547
$Ping_{avg\_size}$ (bytes)	42	42	43	43
$Ping_{Tot\_TX}$ (bytes)	29484	27620	22705	12264
$Ping_{Tot\_RX}$ (bytes)	29484	27620	22661	12264
$Signal_{TX\_count}$	702	639	488	263
$Signal_{RX\_count}$	702	639	487	263

	Inside building (no OBD data)	Inside car (no movement)	City driving	Highway driving
$Signal_{lost\_count}$	0	0	1	0
$Signal_{avg\_size}$ (bytes)	164	163	163	163
$Signal_{Tot\_TX}$ (bytes)	114818	104240	79614	42940
$Signal_{Tot\_RX}$ (bytes)	114818	104240	79451	42940
OBD_{TX_count}	-	653	522	285
OBD_{RX_count}	-	653	522	285
OBD_{lost_count}	-	0	0	0
OBD_{avg_size} (bytes)	-	354	357	356
OBD_{Tot_TX} (bytes)	-	362967	186450	101574
OBD_{Tot_RX} (bytes)	-	362967	186450	101574
T_{diff_max} (ms)	2838	3221	3486	3318
$Count_{in\_service}$	702	653	530	285
$Count_{no\_service}$	0	9	45	59
Tot_{TX_count}	1404	1945	1540	833
Tot_{RX_count}	1404	1945	1538	833
Tot_{TX} (bytes)	144302	362967	288769	156778
Tot_{RX} (bytes)	144302	362967	288562	156778
$Tot_{pkt\_loss\_rate}$	0%	0%	0.129%	0%
$T_{no\_service\_max}$ (sec)	0	12	28	72
$T_{no\_service\_rate}$	0%	1.019%	5.639%	12.356%

From results it is clear that in this implementation NB-IoT and MQTT ensure reliable data transmissions when the IoT device is in service as the maximum obtained packet loss rate was 0,129% with packet payload sizes ranging from 42 to 357 bytes. Higher driving speeds affected ping delay negatively – at highway speeds 51% of received ping requests had a delay greater than 1000ms while for city driving it was 41% and during engine idling 14% (Figure 36). In Figure 37 we can observe NB-IoT reference signal

received power (RSRP) during different conditions which shows that with a strong signal there is no need for cell reselection (device inside building) and service is not lost once while during engine idling serving cell was reselected three times and as a result there was no service for 18 seconds.

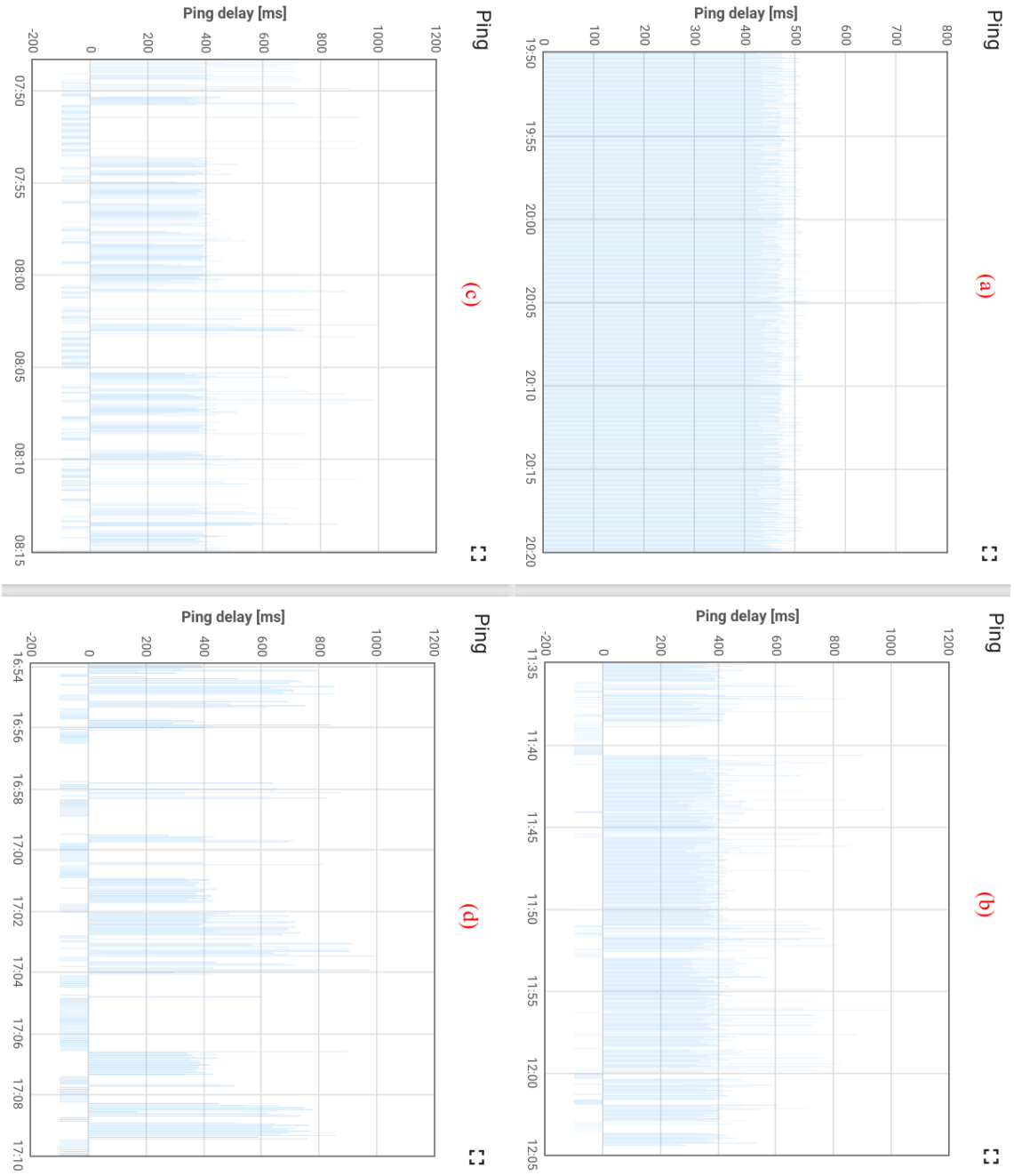


Figure 36. Ping delay: (a) inside building, (b) inside car at standstill, (c) city driving, (d) highway driving. Negative ping values denote that timeout (1000ms) was reached.

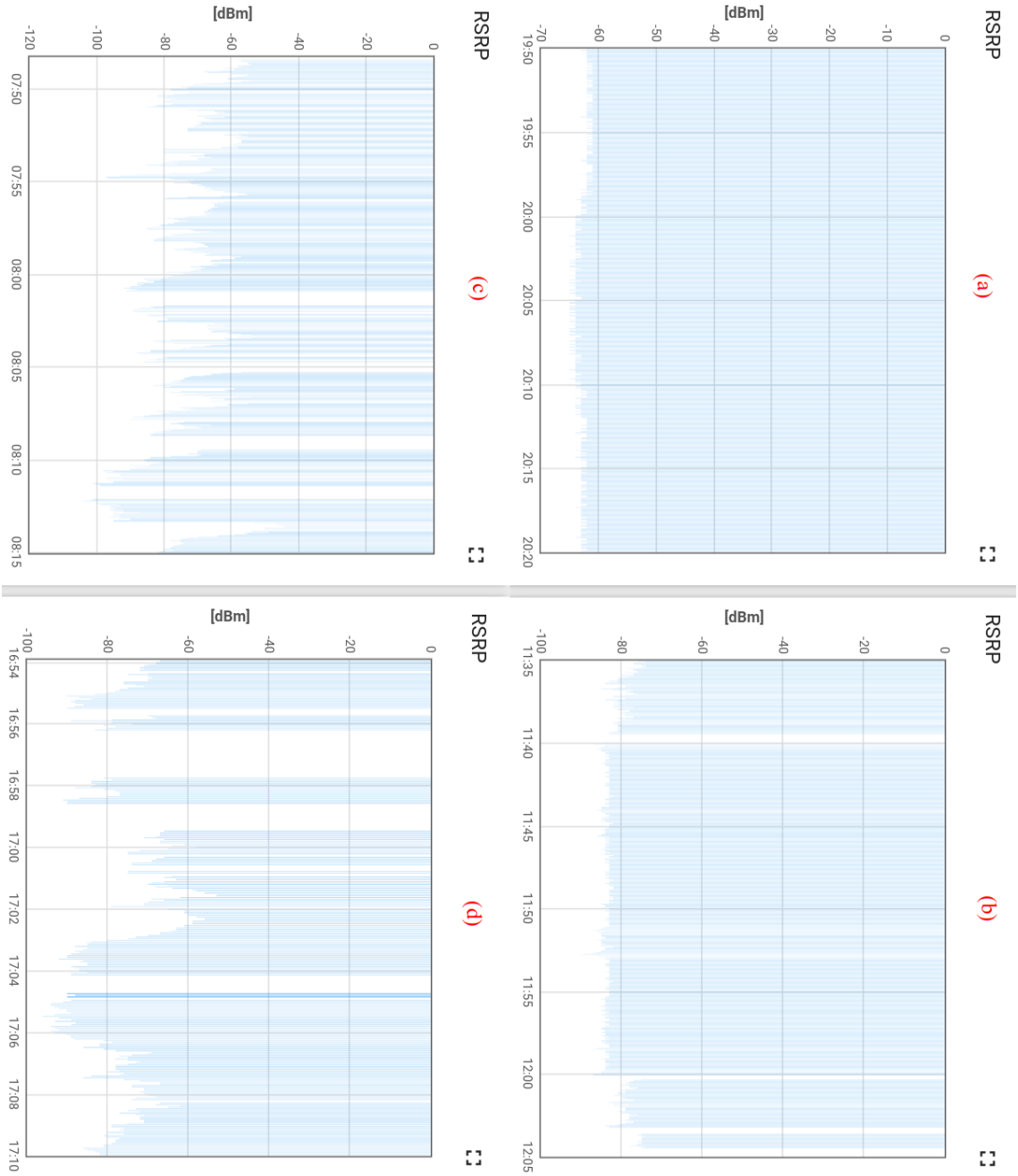


Figure 37. NB-IoT Reference Signal Received Power (RSRP): (a) inside building, (b) inside car at standstill, (c) city driving, (d) highway driving.

The impact of limited mobility management in used SIM7070G modem (based on 3GPP release 14) could be noted at higher driving speeds – the IoT device had no service for 5% of the total travel time during driving in the city at moderate speeds (on the highway it was 12%). A possible solution to overcome mobility limitations for vehicular applications with NB-IoT, which need not be monitored in real-time, could be

to transmit data when the journey has ended and vehicle has been parked or it has been detected that the vehicle has not been in motion for a while. This approach would require data to be stored locally until it has been forwarded to the network. In this implementation, however data was intended to be gathered over a longer time period, thus continuous network availability and low latency were not seen as a requirement.

5 Conclusion and future works

In this thesis an implementation for predictive car maintenance has been developed which is based on open solutions and uses NB-IoT for data transmissions between cloud server and vehicle. In chapter 1 an overview was given for the reader about relevant protocols and technologies which enable to integrate data streams from vehicles to cloud services. In chapter 2 state of the art machine learning algorithms, maintenance policies and techniques for estimating remaining useful life were introduced along with data collection devices and related works in predictive maintenance.

Data acquisition has an integral part in PdM, therefore the architecture of the system which must be able to store, transfer and process the data needs to be carefully considered. While data loggers and telematics units which interface with OBD-bus and communicate with cloud services is nothing new, the author of this thesis could not find an already existing solution which:

- is easily configured and customized
- is easily integrated to any cloud platform (uses open standards which are easy to process and interpret for such matter)
- uses NB-IoT for cloud server communications
- does not require the existence of a smartphone.

The experience gained from integrating one commercially available telematics unit was a learning point which gave the direction of developing a prototype which takes all of the above into account. The architecture and implementation of the configured predictive maintenance system is introduced in chapter 3. The use of open and inexpensive solutions makes it possible to further extend or customize this work as desired.

With embedded data acquisition systems storage space and processing power are limited, therefore a compromise needs to be made with the resources that are available – common practice thus far has been to store and process data on a cloud server instance or store aggregated data on the device itself. The benefits of server side processing are clear as the processing requirements are shifted away from the end-user device, however when monitoring systems which generate hundreds of signals frequently it would be impractical and not even possible to transmit them in real-time. Aggregating data can be an option in such cases but related works have confirmed that information for fault detection could be lost as a result. For a PdM system it would suffice if data could be analysed on the end-user device itself and only prediction results from deployed models be returned to the server to evaluate the need for maintenance actions. PdM is applied to systems where a gradual deviation or degradation process can be observed and identified, therefore real-time feedback from end-user device is not a necessity.

From the outset of this work it was not clear which parameters should be collected, how frequently and where they should be stored – either on cloud server or on the data collection device itself. It was decided to take the cloud server approach because this allowed to evaluate the performance of NB-IoT in a mobile environment and at the same time learn how sensors in vehicle react to various driving conditions. Having access to all available data on cloud server proved to be invaluable for identifying communication issues and validating the stored data. After having resolved all major communication and data related issues, two experiments were conducted with the authors own vehicle (chapter 4):

- airflow restriction
- engine air leak.

With those experiments it is shown how machine learning (using regression and classification analysis) and statistical methods (using histograms and autoregressive models) can be applied to data collected from vehicle CAN-bus to build models for predictive maintenance. In the first experiment highest prediction accuracy (97%) was obtained with RF and SVR models while in the second experiment RF obtained 98%.

All ML models created during the experiments had higher prediction accuracy than models with simple linear regression based on single input variable.

Future work is needed to deploy the resulting models on the end-user device itself to accommodate storage and processing requirements as for car manufacturers it would be desirable to integrate such systems. With electric and especially autonomous vehicles in mind the processing power of traditional ECUs as well as local vehicle-bus network throughput need to be increased. Most OBD-II PIDs collected in this work are relevant for vehicles with internal combustion engine as with electric vehicles the focus is more on battery health and optimization management, also electric vehicles are not required to be compliant with OBD-II.

Although the setup used in this work had limited mobility management (NB-IoT modem was based on 3GPP release 14), we have shown (in chapter 4) that packet loss in this implementation is not an issue even at higher driving speeds when the IoT device is in service. Potential loss of data would occur if it is not saved locally on the device and can not be sent to the server while there is no service – as a result in this work an indicator describing the rate of network unavailability was used for various mobility scenarios (standstill, city driving, highway driving). The rate of network unavailability increased with driving speeds – for city and highway driving the obtained results were 5% and 12% respectively while at standstill it was 1%. Ping delay measurements followed the same trend – for city and highway driving respectively 41% and 51% of total ping packets had a delay greater than 1000ms while at standstill it was up to 14%. Data rates provided by NB-IoT are suitable for applications with relatively small packet sizes (few hundred bytes) and medium to low latency requirements (ping delay of 500ms and more). NB-IoT is a preferred solution where communications with cloud server need to be established infrequently (in case some anomaly is detected) or to report some other key indicator of the system obtained over a longer time period, making it possible to serve thousands of devices within limited radio resources. Combined with advancements in machine learning, the widespread adoption of PdM should be on the rise in the future.

References

[1] Hatch, S. V. Computerized engine controls (9th Edition). Cengage Learning, 2012.

[2] OBD (VAG) vehicle diagnostic [WWW] https://allpinouts.org/pinouts/connectors/car/obd-vag-vehicle-diagnostic/ (27.10.2020)

[3] Johansson, K. H., Törngren, M. & Nielsen, L. Vehicle applications of controller area network. In *Handbook of networked and embedded control systems, Birkhäuser Boston*, 2005, 741-765.

[4] Watterson, C. Controller Area Network (CAN) Implementation Guide [WWW] https://www.analog.com/media/en/technical-documentation/application-notes/AN-1123.pdf (05.12.2020)

[5] Wu, Y. J. & Chung, J. G. Efficient controller area network data compression for automobile applications. *Frontiers of Information Technology & Electronic Engineering*, 2015, 16(1), 70-78.

[6] Talbot, S. C. & Ren, S. Comparision of fieldbus systems CAN, TTCAN, FlexRay and LIN in passenger vehicles. In *2009 29th IEEE International Conference on Distributed Computing Systems Workshops, IEEE*, 2009, 26-31.

[7] NB-IoT Deployment Guide to Basic Feature set Requirements [WWW] https://www.gsma.com/iot/wp-content/uploads/2019/07/201906-GSMA-NB-IoT-Deployment-Guide-v3.pdf (16.11.2020)

[8] Larmo, A., Ratilainen, A. & Saarinen, J. Impact of COAP and MQTT on NB-IoT system performance. *Sensors*, 2019, 19(1), 7.

[9] Mell, P. & Grance, T. The NIST definition of cloud computing. 2011.

[10] Sinha, R. S., Wei, Y., & Hwang, S. H. A survey on LPWA technology: LoRa and NB-IoT. *Ict Express*, 2017, 3(1), 14-21.

[11] The 5G Evolution: 3GPP Releases 16-17 [WWW] https://www.5gamericas.org/wp-content/uploads/2020/01/5G-Evolution-3GPP-R16-R17-FINAL.pdf (16.11.2020)

[12] Narrowband Internet of Things [WWW] https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_application/application_notes/1ma266/1MA266_0e_NB_IoT.pdf (27.10.2020)

[13] Andres-Maldonado, P., Ameigeiras, P., Prados-Garzon, J., Navarro-Ortiz, J. & Lopez-Soler, J. M. Narrowband IoT data transmission procedures for massive machine-type communications. *Ieee Network*, 2017, 31(6), 8-15.

[14] Mwakwata, C. B., Malik, H., Mahtab Alam, M., Le Moullec, Y., Parand, S., & Mumtaz, S. Narrowband Internet of Things (NB-IoT): From physical (PHY) and media access control (MAC) layers perspectives. *Sensors*, 2019, 19(11), 2613.

[15] MQTT Version 5.0 [WWW] https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html (16.11.2020)

[16] Mitchell, T. M. Machine learning. McGraw-Hill, 1997.

[17] Ignatow, G. & Mihalcea, R. Supervised Learning. In *Text Mining: A Guidebook for the Social Sciences, Thousand Oaks: SAGE Publications*, 2017, 62.

[18] Pao, Y. H. & Sobajic, D. J. Combined use of unsupervised and supervised learning for dynamic security assessment. *IEEE Transactions on Power Systems*, 1992, 7(2), 878-884.

[19] Carvalho, T. P., Soares, F. A., Vita, R., Francisco, R. D. P., Basto, J. P. & Alcalá, S. G. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 2019, 137, 106024.

[20] El-Rewini, Z., Sadatsharan, K., Selvaraj, D. F., Plathottam, S. J. & Ranganathan, P. Cybersecurity challenges in vehicular communications. *Vehicular Communications*, 2020, 23, 100214.

[21] Belgiu, M. & Drăguț, L. Random forest in remote sensing: A review of applications and future directions. *ISPRS Journal of Photogrammetry and Remote Sensing*, 2016, 114, 24-31.

[22] Guenther, N. & Schonlau, M. Support vector machines. *The Stata Journal*, 2016, 16(4), 917-937.

[23] Support Vector Machines [WWW] https://scikit-learn.org/stable/modules/svm.html# (16.11.2020)

[24] Hardesty, L. Explained: Neural networks [WWW] https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (16.11.2020)

[25] Amini, A. Introduction to Deep Learning [WWW] http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf (16.11.2020)

[26] Caggiano, A., Angelone, R., Napolitano, F., Nele, L. & Teti, R. Dimensionality reduction of sensorial features by principal component analysis for ANN machine learning in tool condition monitoring of CFRP drilling. *Procedia CIRP*, 2018, 78, 307-312.

[27] Schmidt, B. & Wang, L. Cloud-enhanced predictive maintenance. *The International Journal of Advanced Manufacturing Technology*, 2018, 99(1-4), 5-13.

[28] Krupitzer, C., Wagenhals, T., Züfle, M., Lesch, V., Schäfer, D., Mozaffarin, A., ... & Kounev, S. A Survey on Predictive Maintenance for Industry 4.0. *arXiv preprint arXiv*:2002.08224, 2020.

[29] Li, Z., Wang, K. & He, Y. Industry 4.0 - Potentials for Predictive Maintenance. *Advances in Economics, Business and Management Research,* 2016.

[30] Sikorska, J. Z., Hodkiewicz, M. & Ma, L. Prognostic modelling options for remaining useful life estimation by industry. *Mechanical systems and signal processing*, 2011, 25(5), 1803-1836.

[31] Weibull minimum continuous random variable [WWW] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.weibull_min.html#scipy.stats.weibull_min (16.11.2020)

[32] The ELM327 data sheet [WWW] https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf (16.11.2020)

[33] Wolf, P., Mrowca, A., Nguyen, T. T., Bäker, B. & Günnemann, S. Pre-ignition detection using deep neural networks: A step towards data-driven automotive diagnostics. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC), IEEE,* 2018, 176-183.

[34] Sass, A. U., Esatbeyoglu, E. & Iwwerks, T. Signal Pre-Selection for Monitoring and Prediction of Vehicle Powertrain Component Aging. *Nauka I Tehnika*, 2019, 18(6), 519-524.

[35] Kowalik, B. & Szpyrka, M. Architecture of on-line data acquisition system for car on-board diagnostics. In *MATEC Web of Conferences, EDP Sciences*, 2019, 252, 02003.

[36] Sezer, E., Romero, D., Guedea, F., Macchi, M. & Emmanouilidis, C. An industry 4.0-enabled low cost predictive maintenance approach for smes. In *2018 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC), IEEE*, 2018, 1-8.

[37] Gurung, R. B. *Random Forest for Histogram Data: An application in data-driven prognostic models for heavy-duty trucks* (Doctoral dissertation, Department of Computer and Systems Sciences, Stockholm University). 2020.

[38] Prytz, R. *Machine learning methods for vehicle predictive maintenance using off-board and on-board data* (Doctoral dissertation, Halmstad University Press). 2014.

[39] Zero2Go Omini User Manual [WWW] http://www.uugear.com/doc/Zero2Go_Omini_UserManual.pdf (16.11.2020)

[40] SIM7080 Series AT Command Manual V1.02 [WWW] https://www.waveshare.com/w/upload/3/39/SIM7080_Series_AT_Command_Manual_V1.02.pdf (16.11.2020)

[41] AS3249 Telia Eesti AS [WWW] https://ipinfo.io/AS3249 (16.11.2020)

[42] Ashok, B., Ashok, S. D. & Kumar, C. R. Trends and future perspectives of electronic throttle control system in a spark ignition engine. *Annual Reviews in Control*, 2017, 44, 97-115.

[43] Global OBD Vehicle Communication Software Manual [WWW] https://www.snapon.com/Files/Diagnostics/UserManuals/GlobalOBDVehicleCommunicationSoftwareManual_EAZ0025B43.pdf (16.11.2020)

[44] Mass Air Flow Sensor (G70): Implausible Signal [WWW] http://wiki.ross-tech.com/wiki/index.php/16485/P0101/000257 (16.11.2020)

[45] Audi Engine management Systems [WWW] http://www.vaglinks.com/vaglinks_com/Docs/SSP/VWUSA.COM_SSP_941002_Engine_Management_Level1.pdf (16.11.2020)

[46] Szabo, B. Understanding Ignition Timing: Making Maximum Power Means Knowing the Science [WWW] https://www.enginebuildermag.com/2017/09/understanding-ignition-timing-making-maximum-power-means-knowing-science/ (07.11.2020)

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Priit Kullerkupp

1 Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "5G NB-IoT Implementation for Predictive Car Maintenance", supervised by Muhammad Mahtab Alam

1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2 I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3 I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

15.12.2020

[1] The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 - Disabling unused kernel modules and system services

Onboard Wifi, Bluetooth and audio modules can be disabled by editing a text file in **/boot/config.txt** (as depicted in Figure 38). Additionally splash screen and boot delay were disabled to improve system startup speed.

```
dtoverlay=disable-bt
dtoverlay=disable-wifi
dtparam=audio=off
disable_splash=1
boot_delay=0
```

Figure 38. Disable hardware in /boot/config.txt.

Unused kernel modules were blacklisted from loading in **/etc/modprobe.d/rpi-blacklist.conf** (Figure 39).

```
blacklist snd_bcm2835
blacklist brcmfmac
blacklist brcmutil
blacklist btbcm
blacklist hci_uart
blacklist bcm2835_codec
blacklist v4l2_common
blacklist videobuf2_common
blacklist bcm2835_v4l2
blacklist v4l2_mem2mem
blacklist videobuf2_v4l2
```

Figure 39. Blaclisted kernel modules.

Unused system services were disabled using **systemctl** utility (Figure 40).

```
systemctl disable rpi-eeprom-update.service
systemctl disable triggerhappy.service
systemctl disable ssh.service
systemctl disable alsa-restore.service
systemctl mask alsa-restore.service
systemctl disable networking.service
systemctl disable raspi-config.service
systemctl disable dhcpcd.service
systemctl disable avahi-daemon
systemctl disable systemd-timesyncd.service
systemctl disable wpa_supplicant.service
systemctl disable hciuart.service
systemctl disable bluetooth.service
```

Figure 40. Disabled system services.

Appendix 3 – Cloud server information

General information about IP address and Linux distribution of cloud server is shown in Figure 41.

```
ubuntu@vps-5f16802c:~$ ifconfig
ens3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 51.89.167.146  netmask 255.255.255.255  broadcast 0.0.0.0
        ether fa:16:3e:28:08:a5  txqueuelen 1000  (Ethernet)
        RX packets 224152  bytes 27373673 (27.3 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 159206  bytes 23796498 (23.7 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 476682  bytes 147904180 (147.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 476682  bytes 147904180 (147.9 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ubuntu@vps-5f16802c:~$ curl ipinfo.io/51.89.167.146
{
  "ip": "51.89.167.146",
  "hostname": "vps-5f16802c.vps.ovh.net",
  "city": "Bexley",
  "region": "England",
  "country": "GB",
  "loc": "51.4416,0.1487",
  "org": "AS16276 OVH SAS",
  "postal": "DA15",
  "timezone": "Europe/London",
  "readme": "https://ipinfo.io/missingauth"
}ubuntu@vps-5f16802c:~$lsb_release -a
No LSB modules are available.
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.5 LTS
Release:        18.04
Codename:       bionic
ubuntu@vps-5f16802c:~$
```

Figure 41. Cloud server information showing public IP address, location and Linux distribution.

Appendix 4 – System service watchdog

The code in Figure 42 and Figure 43 is responsible for monitoring the contents of a text file every 30 seconds and performing a complete reset of main service when needed.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import os
import time
import datetime

reset_required_conf = "/home/pi/reset_required.conf"
check_interval = 30
reset_script = "/home/pi/resetomnipiservice.sh"
reset_log = "/home/pi/reset.log"

def _ts():
      return datetime.datetime.now().strftime("%Y-%m-%d_%H:%M:%S")

while 1:
      with open(reset_required_conf, 'r') as f:
            status = int(f.read(1))
            print(status)
            if status:
                  ts = _ts()
                  print(f"{ts} resetting service")
                  # log reset event
                  with open(reset_log, 'a') as r:
                        r.write(f"{ts}\n")
                  # run reset script
                  os.system(reset_script)
            else:
                  print(f"{_ts()} nothing to do")
      time.sleep(check_interval)
```

Figure 42. Code for monitoring system service.

```
#!/bin/bash

sudo systemctl stop omnipi.service
echo "omnipi.service stopped"
sleep 5
echo "0" > /home/pi/reset_required.conf
sudo systemctl start omnipi.service
echo "omnipi.service started"
```

Figure 43. Code for executing system service reset.