University of Southern Queensland

Faculty of Health, Engineering & Sciences

# Condition Monitoring for Predictive Maintenance Purposes

A dissertation submitted by

M. Fisher

in fulfilment of the requirements of

**ENG4111 and ENG4112 Research Project**

towards the degree of

**Bachelor of Electrical & Electronic Engineering**

Submitted: October, 2019

# Abstract

Currently, a large amount of time and resources are invested in preventative and reactive maintenance, independent of industry. In the Aerospace sector we have seen large strides towards Health and Usage Monitoring (HUMS) and its associated predictive maintenance, which results in a large amount of labor and cost savings.

This research investigates whether an embedded device with peripheral sensors and modules can capture, store and transmit a number of physical variables leading to the successful prediction of a fault condition in a road vehicle. This would allow predictive maintenance to be carried out, reducing cost and down-time associated with preventative and reactive maintenance. Additionally, faults incurred in normal operations that spread damage beyond the faulty component may be prevented with the use of predictive practices.

Using a small but powerful micro-controller, suitable sensor arrays such as the MAX9814 microphone and LIS3DH Accelerometer were integrated to monitor engine audio and vibration. An OBD2 hardware/ software package was used to monitor real-time conditions during test phases. Additionally, an external GPS patch antenna was manufactured to mount on the skin of the vehicle and used in conjunction with a GPS module to log location data. Lastly, a Wi-Fi module was developed to communicate wirelessly when the vehicle/ module returned to a geo-located home-base.

Once the data had been successfully logged and sent back to the client computer at home-base, it was processed using a number of techniques to determine a fault or healthy condition. FFT and spectrogram plots were created to visualize the signal content of faulty and healthy data. Linear Predictive Coding was used in conjunction with a Mahalanobis Distance measure to test unknown audio samples against known fault/ healthy conditions.

The LPC/ Mahalanobis Distance techniques were able to detect a known condition (fault or healthy) with a high level of probability when tested against unknown audio samples of the same condition on the same vehicle. It is unknown if the algorithm would be able to detect similar conditions when tested against vehicles of the same make, model and year. It was not able to detect similar faults on vehicles of a different make, model or year.

It is the hope of this research that using the detection algorithms will lead to development of predictive maintenance models, where the Mahalanobis Distance measure will give indications of approaching faults.

University of Southern Queensland

Faculty of Health, Engineering & Sciences

**ENG4111/2 *Research Project***

## Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering & Sciences or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

**Dean**

Faculty of Health, Engineering & Sciences

# Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

M. Fisher

# Acknowledgments

These works would not have occurred without the support of my wife Britta-Rose and my two children, Calvin and Kaylee. I will be eternally grateful for the chance to go back to school at such a tender age.

I would also like to show my appreciation for my supervisor, Mark Phythian, who spent a large amount of time with me whether he was busy or not, and was always on hand to help out.

M. Fisher

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Chapter Overview

This chapter provides an introduction and background for the field of data logging. It will describe the project outline and research objectives which help to define the end goals being sought. Finally, it shall describe the necessary resources and timeline required to achieve the research objectives, as well as including a risk assessment for the activities undertaken in the course of the project.

## 1.2 Prologue

Since the beginning, man has fought a battle with machines. When the wheel was invented, man knew it wasn't perfect and he sought to make it so. Unfortunately, we haven't reached this point quite yet. While maintenance costs are decreasing due to an increase in vehicle quality (Antich 2014), there is still a significant amount of cost associated with operating a vehicle. A portion of this cost is preventative maintenance, things like periodic servicing, tire changes and cleaning. The remainder of this cost is reactive maintenance, maintenance that is incurred when a vehicle breaks unexpectedly.

Currently, a large amount of time and resources is invested in preventative maintenance, regardless of industry. As it is time-based, it can regularly inflate the cost involved, especially if the maintenance is done prematurely. Additionally, more spare parts are required, as it requires regular replacement of serviceable parts. Lastly, it is labor intensive, requiring trained maintainers for every service (Mehta 2018).

Common to aviation, heavy industry and the high performance racing industry is the practice of condition monitoring, the concept of continually monitoring a parameter, or parameters, in machinery in order to identify a significant change in operating conditions. This can lead to the diagnosis of faults and the prevention of costly and catastrophic failure (Freeman 2018). Condition monitoring is a component of Predictive Maintenance, where the maintenance relies on the condition of a machine rather than a set interval.

## 1.3 Background

The concepts and technology related to condition monitoring are not new. However, the application of this technology to road vehicles is a recently developing one. Due to the nature of aircraft, it became a safety mandated issue early on in their timeline to be able to prevent a fault occurring in flight. Alternatively, it was considered a minimum requirement of the monitoring system to be able to determine what had caused the aircraft to become faulty whilst in the air, for future prevention.

Road vehicles and plant do not suffer faults that lead to such catastrophic outcomes and loss of life. As such, the technology has been developed along different timelines. Only in

the last decade or so are we seeing advances in data logging applications for vehicles and plant.

Generally, only mechanics and auto-electricians have access to the software and hardware required to diagnose faults in a vehicle. Additionally, this information is very limited in scope and will usually only show faults after the fact.

The main driver for this research in vehicles is an economic one. If commercial or private owners are able to use a data logging system to aid in diagnosing faults, this will lead to an ability to schedule preventative maintenance to reduce costs and down time. There are a number of additional benefits as well, such as the ability to track by GPS in delivery services, view trend monitoring data for optimal routing and engine performance with different fuels or additives, and even to be able to review driver trends.

Upon an initial research foray into the field in the precursor subject, ENG4110 Engineering Research Methodology, it was determined that there was some scope for the development of a data logging system to help diagnose faults in private and public vehicles before catastrophic failure. There are a wide number of papers in the field of maintenance prevention, most of which describe some very specific applications of sensors in vehicles and the resulting analysis of the data in diagnosing faults. This does not cross over to long term data logging applications for the individual user however. It appears that the majority of data logging systems available to the public are limited in scope, do not provide fault diagnosis, or are sold at an exhaustive cost.

### 1.3.1 Health Monitoring in Aviation

An example of the condition monitoring systems in use today in aviation is that of the HUMS (Health and Usage Monitoring) system used on the new Australian Defence Force helicopter, the MRH-90 Taipan. The internally equipped system is able to provide information on the performance of the aircraft to aircrew, maintainers and fleet managers in order to aid in fault diagnosis and short term rectification of issues. It also collects trend data which is useful in long term structural monitoring (Fay 2009).

While the MRH-90 is one of the newest to enter service in the ADF, the concept of health monitoring is not new. Aircraft have employed 'black boxes', or flight recorders, since

the early 1960's in order to determine why aircraft would fail. Interestingly, the flight recorder was invented by an Australian scientist named David Warren working for the Aeronautical Research Laboratory in Melbourne, which was one of the precursors to DST (Defence Science and Technology), (DST 2018).

### 1.3.2 Fault Finding for the Automobile

Eventually, some of the monitoring applications in other industries lent themselves to the public transport industry. This was seen in the OnBoard Diagnostic (OBD) system, developed by the California Air Resources Board (ARB) in 1982 (Hanna 2017). Early on in OBD history, different adapters were required for different vehicles as well as having a different set of codes. This led on to the development of OBD2, which had a universal set of fault codes and adapters. ODB2 installation has been mandatory in Australian vehicles since 2005.

OBD2 systems conduct real-time system monitoring of the working condition of commercial vehicles. In addition to this, the OBD system has the ability to detect a malfunction within its monitoring scope and log a standardized Diagnostic Trouble Code (DTC) in memory. OBD scan tools can then access the DTC from the Engine Control Unit (ECU), leading to a reduction in troubleshooting time (Allam & Elhady 2018).

Until recently, the OBD diagnostics equipment was only found in local mechanical and auto-electrician workshops. This made access to the OBD data restricted and was often thought of as "specialist" knowledge. Nowadays, the process is as simple as plugging in the appropriate hardware modules and following along with the associated computer programs, or apps in the case of mobile phone versions. This opens up a new pathway for home enthusiasts to monitor the performance of their car in real-time.

Unfortunately, finding a fault in real-time is often too late. The ability to integrate OBD systems with a data logging capability would provide drivers with vehicle and tracking information that could prove invaluable in preventative maintenance and performance logging.

One area where data logging has been developed extensively is in the high performance racing industry, and more specifically, McLaren Applied Technologies. In 2007, the FIA

(Federation Internationale de l'Automobile), which is the regulatory body for most auto racing events, chose McLaren to be the sole supplier of ECUs in F1 vehicles. With the help of over 300 sensors, McLaren's F1 ECU deals with over 1000 input parameters and transmits more than 1.5 GB of live data during an average 300 km race. This is roughly 750 million data points over 2 hours (McLaren Applied Technologies 2016).

Similar systems with reduced capabilities are available to the public racing domains. They provide data from GPS modules, accelerometers, electrical sensors and the CAN bus to name a few. They prove high level processing and approximately 24 hours of logging time (Race-Technologies 2019). Unfortunately, they are designed more towards recording race conditions and split times rather than fault conditions.

### 1.3.3 Data Logging in Agriculture

Something which has found a niche in data logging is the agriculture industry. Farming equipment is commonly installed with systems similar to OBD2, where a large swath of data is available to the user. Development is accelerating in data logging applications which could be used to record sensor readings such as RPM, acres covered and maintenance data (Wehrspann 2016).

On the ground, agriculture business owners are installing data loggers to the CAN buses of the farming equipment to log and store data on the day to day operations. This gives the farmer a huge advantage, enabling them to understand hybrid performance by field, soil zone and population, with views to previous yield data.

Currently, data loggers in agriculture are limited by the array of sensors installed on the given plant/ equipment at manufacture. Given more development, additional sensors could be placed on planters to measure soil moisture, or on a disk to measure ground hardness, making the data collection of necessary farming data more automatic. This would lead to farmers making more informed operating decisions.

### 1.3.4 Maintenance Practices in Industry

Data loggers in manufacturing, mining, defence and other associated industries perform a wide variety of tasks, from monitoring accelerations on a train, to smoke and pressure de-

tection on a decommissioned warship in the throes of explosively sinking (ThermoFisher Scientific 2010). They are expected to be rugged and provide reliable collection of appropriate data.

Currently, preventative maintenance is the widely adopted strategy for conducting maintenance in industry. It means scheduling maintenance at regular intervals of time, based on the manufacturers recommendations. For the newly emerging technological industries, this is not cost effective. Each sector could benefit from looking towards predictive maintenance schedules based on condition monitoring systems (Mehta 2018). Transition to predictive maintenance schedules will provide a large opportunity to increase reliability and efficiency in production and operations.

### 1.3.5 Data Logging in Healthcare

Health care at first glance seems a surprising addition to a topical background of data logging and predictive maintenance applications. Interestingly, a heart rate monitor was one of the first data loggers invented for health monitoring, and had the ability to study heart rate in real time or record for later study. Since its invention in 1977 by Polar Electro for the Finnish National Cross Country Ski team, the heart rate monitor has pervaded fitness cultures around the world (Kite-Powell 2019).

Data loggers in the health care industry also provide functions such as autoclave validation (a device to log temperature and sterilize an environment), Vaccine monitoring and transportation, clean room monitoring, and pharmaceutical storage (MadgeTech 2018).

## 1.4 Project Outline

This research aims to determine whether an embedded device comprised of commercially available components can successfully predict fault conditions in a vehicle, which would allow predictive maintenance to be carried out, reducing costs, time and damages associated with preventative and reactive maintenance.

Testing and evaluation phases will determine a suitable configuration of the sensor array, controlling software, and data logging capabilities. The evaluated system will be used to

monitor a vehicle with a fault condition and the resulting data will be analyzed to confirm the diagnosis.

## 1.5 Research Objectives

The research objectives of this dissertation are derived from the project specification shown in Appendix A.

- Research the background information related to data logging micro-controllers that are used to monitor a similar variety of conditions in vehicles.
- Design and build an implementation of a real-time, condition monitoring micro-controller to monitor a set of specific conditions.
- Design and build an external antenna for integration with GPS receiver module, including any GPS signal filtering and amplifying.
- Perform sensor signal analysis and classify normal and abnormal behavior.
- Incorporate a Wi-Fi implementation for autonomous data transfer from the vehicle upon return to a geo-located base-station.
- Demonstrate the effective use of the condition monitoring module by diagnosing a vehicle fault.

If the capability exists for an extension of the project then the following will be considered:

- Design and build a Printed Circuit Board (PCB) and a mounting case to incorporate all of the modules in a neat, re-producible and install-able package.
- Incorporate additional sensors into the micro-controller to monitor alternative conditions and fault areas.

## 1.6 Methodology Summary

A methodology following the classical engineering approach was utilized in the undertaking of this project. The approach investigates a potential system and uses an iterative process to refine and achieve stated objectives. This approach establishes objectives, synthesis, analysis, construction, testing, and evaluation.

Research documentation, methods of analysis, quality assurance plans and a project breakdown are a prelude to the described hardware, software, and methods of data analysis which are used to achieve the project objectives.

## 1.7 Consequences of Applied Research

Historically, access to car diagnostic equipment has been in the domain of the mechanical workshops and tradesmen that operate them. This means that the individual car or vehicle owner had no low cost "plug and play" method of taking a look at the inner workings of their vehicle, if they would be so interested.

With the advent of our current technological age, it is now possible for the DIY operator to purchase electronic interfacing equipment to read DTC's directly out of the vehicle ECU and take a look at the real time operation of their vehicle. This project seeks to add to that capability by introducing a data logging and monitoring system, capable of recording more specific parameters of the vehicles operation. This data could then be transmitted, automatically via wireless or by manual methods, to a home terminal capable of analyzing and breaking down the vehicle usage over a set period.

Some of the impacts of this data collection method on condition monitoring trends are as follows (Plant Services 2016):

- Bench marking of performance across vehicles: Generally speaking, humans are interested in how their equipment compares to the neighbor or family member.
- Transparency for vehicle businesses: Giving the vehicle owner the ability to confirm vehicle specifications and run cost estimations from the manufacturer would be a huge boost to companies in the global market.

- Collaboration with enthusiasts: With a cloud based system, the DIY tradesman with the available technology would be able to consult with practitioners around the world. Sharing the collected information over the internet and/ or cloud would be a great way to troubleshoot.

## 1.8 Risk Assessment

In order to ensure that each task of this project was undertaken with an appropriate level of consideration for safety, a risk assessment was carried out to manage and identify hazards. Appendix B shows the respective assessment form.

Of the hazards that were identified in the risk assessment, there were a number of elements that warranted further investigation. This was felt to be a necessary step in order to better inform the project participants. These are described below:

Hazardous Substances - Solder (University Of Cambridge 2019):

- Solder contains lead, which can give rise to serious chronic health effect. Exposure is primarily through ingestion, whether this be skin or mouth/ nose. If it is deemed necessary, Personal Protective Equipment (PPE) is to be worn to mitigate exposure to lead ingestion.
- Rosin (colophony, ersin) is a resin contained in solder flux. Flux is what generates the visible fumes when soldering. Repeated exposure to solder flux can lead to eye, throat and lung irritations. Exposure to fumes should be managed in accordance with safe soldering guidelines, by soldering in a well ventilated place and utilizing fume extraction apparatus.

Shocks and Burns - Electrical connections, soldering and equipment mounting:

- Before using soldering irons, they must be inspected for any obvious damage to the body, cable or plug, and whether they have had an electrical safety test within the last 12 months. This also applies to any equipment used to manufacture the project equipment such as heat guns or laptop cables. Electrical connections within

the vehicle under test should also be considered, as the engine bay will contain electrical components that should not be bridged or touched.

- When the soldering iron is in use, case must be taken not to touch the shaft or tip of the iron, as it is kept at a constant high temperature.

- When conducting test and evaluation stages, care must be taken in the engine bay. While it may be cool initially, sections will heat up after periods of operation. If relocating components, ensure that heated engine parts are not touched, or project components rested upon them.

Road Safety - Test and Evaluation:

- After the initial design has been carried out, and throughout the project, there will be phases of testing on the vehicle. This will involve static and dynamic testing, where the vehicle may be tested in the garage or on the road to confirm the hardware is operational. Care must be taken when on the road, to ensure that the equipment is mounted securely and does not inhibit the ability of the driver to operate the vehicle and obey road safety laws.

## 1.9 Project Timeline

In order to manage the requirements of the project, a timeline is proposed in Appendix C. This is to ensure that goals are being met in accordance with a realistic time frame. Table 1.1 below summarizes the timeline in Appendix C, indicating proposed dates and the amount of time allocated to each task.

Table 1.1: Project Timeline

| *Task* | *Start Date* | *Time (Days)* | *End Date* |
|---|---|---|---|
| 1.1 - Financial viability | 01 January 2019 | 58 | 28 February 2019 |
| 1.2 - Research | 01 January 2019 | 94 | 19 April 2019 |
| 1.3 - Parts suitability | 01 January 2019 | 58 | 28 February 2019 |
| 2.1 - Supervisor Communications | 25 February 2019 | Tuesdays | 08 November 2019 |
| 2.2 - Resource allocation | 25 February 2019 | 21 | 28 February 2019 |
| 3.1 - Design schematics | 25 February 2019 | 25 | 22 March 2019 |
| 3.2 - Simulation | 18 March 2019 | 12 | 29 March 2019 |
| 4.1 - Sensor build and test | 18 March 2019 | 31 | 19 April 2019 |
| 4.2 - Combine Sensors | 01 April 2019 | 39 | 10 May 2019 |
| 4.3 - Data Logging | 01 April 2019 | 39 | 10 May 2019 |
| 5.1 - Design GPS antenna | 01 January 2019 | 58 | 28 February 2019 |
| 5.2 - Build GPS antenna | 01 January 2019 | 58 | 28 February 2019 |
| 6.1 - Analysis of results | 06 May 2019 | 98 | 12 August 2019 |
| 7.1 - Geo-location | 15 July 2019 | 21 | 05 August 2019 |
| 7.2 - Integrate WiFi | 15 July 2019 | 21 | 05 August 2019 |
| 8.1 - Confirm a fault diagnosis | 19 August 2019 | 22 | 11 September 2019 |
| 9.1 - Project Specification | 25 February 2019 | 31 | 19 April 2019 |
| 9.2 - Progress Report | 29 April 2019 | 35 | 21 June 2019 |
| 9.3 - Draft Submission | 22 June 2019 | 50 | 11 September 2019 |
| 9.4 - Final Submission | 11 September 2019 | 36 | 17 October 2019 |
| 9.5 - ENG4903 PP2 | 25 February 2019 | Sem 1 | 21 June 2019 |

## 1.10 Resource Requirements

A number of technical resources were required in the creation of this project, and can be seen below:

- Electronics Hardware
  - Sensor packages: LIS3DH Accelerometer, MPU-9250 IMU, MAX9814 microphone.
  - Microprocessor: Teensy 3.6
  - GPS receiver: Adafruit Ultimate GPS Breakout
  - GPS antenna 2x: A locally manufactured Passive antenna and one purchased GPS active antenna
  - GPS amplifier and connections: Nooelec SAW filter/ amplifier, uFL to SMA connector, coaxial cable w/ SMA
  - Wi-Fi: ATWINC1500 Wi-Fi Breakout
  - OBD2: OBD-II UART and FTDI breakout
- Software
  - Arduino IDE
  - MATLAB r2018
  - Audacity
  - TeXworks
  - MicroCap
  - OBD Auto Doctor
- Other
  - SD card, USB cables.
  - Laptop: i7, 2.9GHz, 8GB RAM, 64-bit, Windows 10.

The complete breakdown of the resource requirements can be seen in Appendix D.

## 1.11 Dissertation Overview

An overview of each subsequent chapter of the dissertation is provided below.

### 1.11.1 Chapter 1: Introduction

This chapter provides an introduction to the project, giving some contextual background in the field of data logging and condition monitoring. It outlines the project goals, the methodology used to achieve those goals, and gives some scope to the project resources, risk management outcomes, and desired time frames.

### 1.11.2 Chapter 2: Literature review

Chapter 2 provides an intensive literature review of related research in the field of data logging and condition monitoring. A comprehensive look at appropriate sensors and microprocessors will be undertaken, as well as the hardware/ software used to process, store and transmit this data.

### 1.11.3 Chapter 3: Methodology

This chapter introduces the methodology used to determine the suitability of the condition monitoring system as a fault diagnosis tool. This includes the design, test and subsequent analysis methodologies.

### 1.11.4 Chapter 4: Design

Chapter 4 delves into the design decisions that were made during the development of the project. It is broken down into the design of each separate sensor or breakout module, and then discusses the total system architecture and integration of the individual packages.

### 1.11.5 Chapter 5: Results and Discussion

Chapter 5 provides a selection of results from the testing phases, and discusses the implications of the data collected. Sensor data is managed appropriately, and analysis of the data is carried out and discussed.

### 1.11.6 Chapter 6: Conclusions and Further Work

Chapter 6 is a summary of the work carried out in the scope of this project and carries a number of conclusions made about the work done. Further work is a summary of limitations in the report that would require more work, as well as areas that could follow on from the project in the future.

# Chapter 2

# Literature Review

## 2.1 Chapter Overview

This chapter provides a comprehensive review of literature in the field of data logging. Related research in the field will be presented in a logical and clear manner to better illustrate the current challenges and progress thus far. The chapter will also show some typical sensor packages and micro-controllers used to achieve some of the related research objectives.

## 2.2 Introduction

In order to undertake any significant or further research in a particular field, namely data logging and its applications on vehicles, it is necessary to conduct a review of literature and research on related topics. This allows us to see if a space exists for our research to be completed and gives an idea of its significance in the field.

## 2.3 Significance of Condition Monitoring

Condition monitoring and maintenance management is a process based upon systems engineering. It encompasses economics, engineering and scientific disciplines, Information Technology, maintenance management, fault detection, diagnostics and even legal issues. The benefits of such a system are many.

"It has been clearly demonstrated that the use of appropriate condition monitoring and maintenance management techniques can give industries significant improvements in efficiency and directly enhance profitability" (Rao 1996).

Condition Monitoring Systems (CMS) or Health Monitoring Systems (HMS) play a hugely important role in establishing condition-based maintenance and repair schedule, which is more beneficial to the company than preventative maintenance (Hameed, Hong, Cho, Ahn & Song 2009).

The main point that all papers agree on is that the implementation of a condition monitoring system will significantly decrease the maintenance costs and down time. Another significant point to mention is the ability of the trend monitoring data to be able to prolong component life. If, for example, an engine was mandated by the manufacturer to be replaced at 10,000 running hours, and from the trend monitoring we deduced that we could extend this to 15,000 hours, we would have utilized our mechanical resources much more efficiently. From these arguments, we can begin to see that the implementation of such a system in commercial fleets will be of significant importance.

## 2.4 Sensor Packages

A wide range of sensor packages are available for measuring just about anything a project could require. Due to the large amount of data available, the following sections will focus on some very specific applications of sensor packages, related to the research objectives of this project.

### 2.4.1 Vibration

Vibration analysis tools have been around since the 1970's and are commonly used across many disciplines. Civil engineers use vibration sensors such as accelerometers and strain gauges to monitor bridge, building and road integrity (Carden & Fanning 2004). Mechanical and electrical engineers use vibration and wear debris analysis for gearbox machinery and electrical plant condition monitoring (Peng, Kessissoglou & Cox 2005). Monitoring of machine conditions has long been accepted as one of the most cost efficient approaches to avoid calamitous failure. Applying the technology to commercial vehicles will be a simple transfer of method, utilizing similar sensors to measure the vibration.

Studies have been developed in the field of less expensive accelerometers, called MEMS (Micro-Electro Mechanical Systems), which can be used in lieu of conventional, more expensive versions. The tests described in the text were carried out on a real CNC machine in a typical industrial workshop (Albarbar, Mekid, Starr & Pietruszkiewicz 2008).

MEMS accelerometers are manufactured using microelectronic fabrication techniques, and create microscopic sensing structures made of silicon. When they are integrated with electronic circuits, they can be used to measure a physical parameter like acceleration. An interesting characteristic of MEMS accelerometers is that they can be used to measure frequencies down to 0 Hz. Some examples of MEMS accelerometers are variable capacitive and piezoresistive, where capacitive accelerometers are typically used for structural monitoring and constant acceleration measurements because of their lower range, and piezoresistive accelerometers are used for shock and blast applications due to their higher range (PCB Piezotronics 2019).

In a study by Barelli, Bidini, Buratti & Mariani (2009), they proved that there is a correlation between vibration measurements acquired by an accelerometer placed on the

cylinder head, and fundamental parameters of the engine. In their study, they used a series of very specialized and expensive accelerometers purchased from PCB piezotronics, mounted unobtrusively and externally. These accelerometers were placed at four different points on the surface of the engine block near the cylinder heads. Concluding their research on vibration on the engine block, they were able to show using a Discrete Fourier Transform (DFT) that at three different load settings, most of the signal energy is in the frequency range within 2000 Hz. This confirms that most of the force transmitted at the cylinder head occurs at low frequencies and can be seen in Figure 2.1.

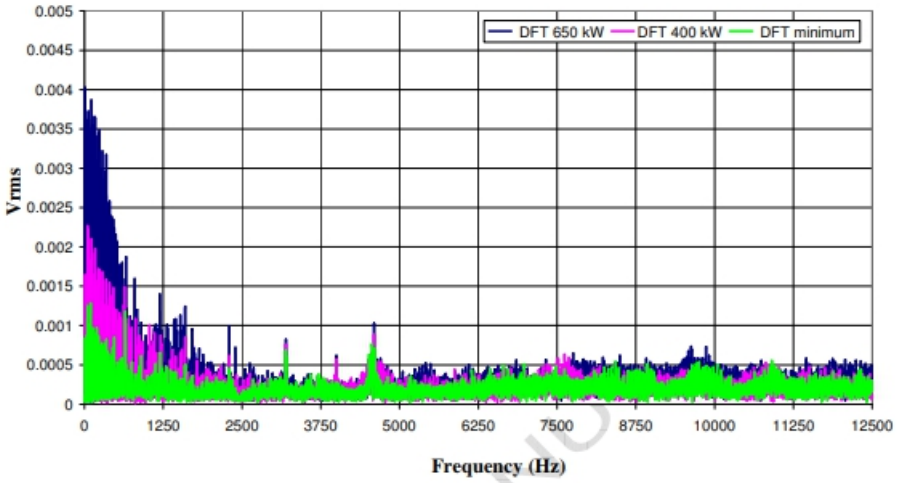



Figure 2.1: DFT of cylinder head accelerometer and corresponding frequency range (Barelli et al. 2009)

Figure 2.2 shows a closer view of the same graph. It should be noted that the graph shows a fundamental frequency of 12.5 Hz, which corresponds to the combustion frequency of the engine under test (with one cylinder). The rotational velocity of the engine under test was 1500 rpm. The fundamental frequency is calculated using Equation 2.1 where n is the order of the particular frequency. Mostly, we will be concerned with the first order frequency, where n = 1.

$$FundamentalFrequency = \frac{RPM}{120} \cdot NumberofCylinders \cdot n \tag{2.1}$$

The y axis in Figure 2.2, Vrms, also shows the magnitude of the signal. For the same engine, it can be seen that for different loads, the magnitude increases as we would expect.

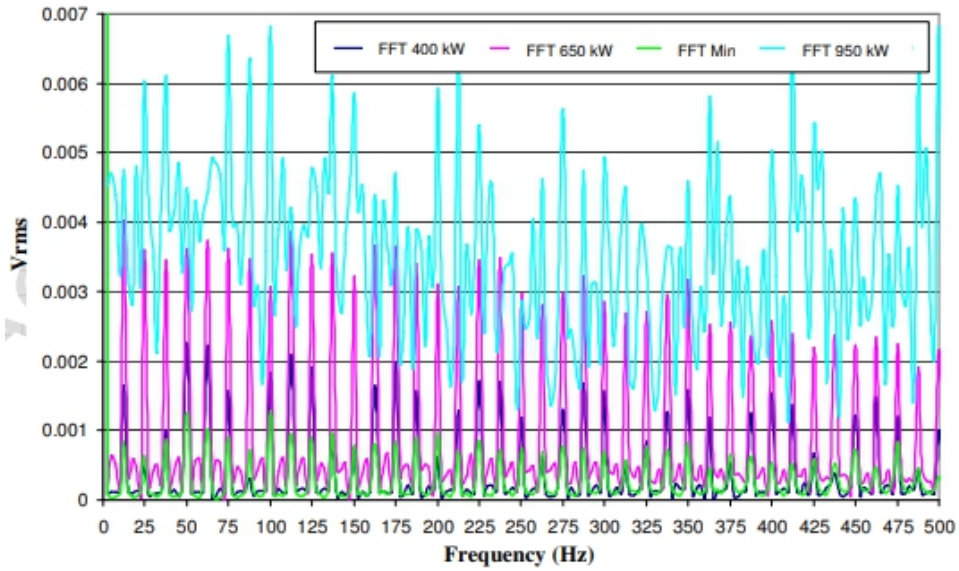



Figure 2.2: DFT of cylinder head accelerometer and corresponding frequency range (Barelli et al. 2009)

A range of economically viable accelerometers exist that could be used in the scope of this project, and all have similar technical characteristics. These include but are not limited to: Analog Devices ADXL343, Adafruit LIS3DH and Adafruit MMA8451. These are all packages manufactured by Adafruit and Analog Devices to breakout the MEMS chipsets for use in microcontroller applications.

The Adafruit MMA8451 has a maximum range of $\pm 8g$ @ 14 bit, whereas the LIS3DH and ADXL343 both have a range of $\pm$ 16g @ 14 bit. Between the LIS3DH and the ADXL343, the LIS3DH, seen below in Figure 2.3, is capable of a higher sampling rate at 5 kHz (albeit only @ 8 bit) which is more than the Nyquist frequency for the engine frequency range stated above. The Nyquist frequency is defined as the minimum rate at which a signal can be sampled without introducing errors, which is twice the highest frequency present in the signal (Oxford Dictionary 2019).

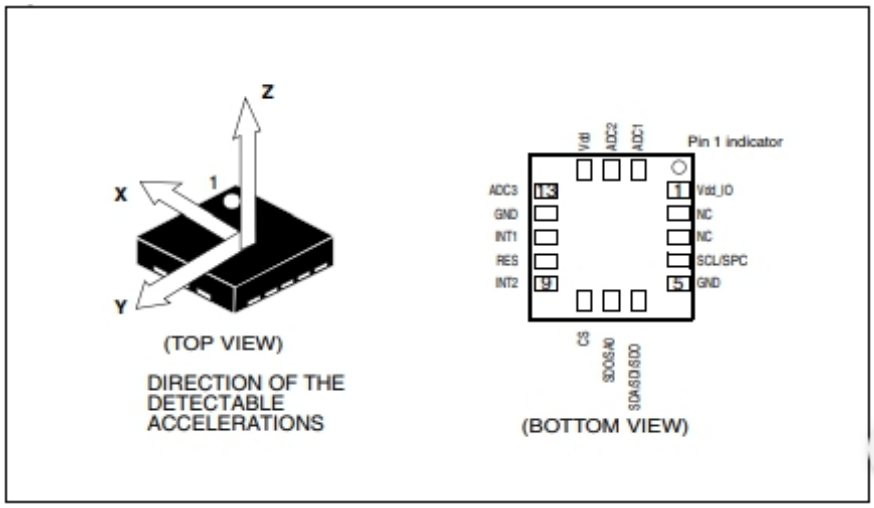



Figure 2.3: LIS3DH sensor package (Adafruit 2019*c*)

The chip block diagram can be seen below in Figure 2.4. When an acceleration is applied to the sensor, the mass displaces from its original position which causes an imbalance in the capacitive bridge. This imbalance is then measured and applied to the charge amplifier, converted to a digital value and transmitted via the chosen protocol.

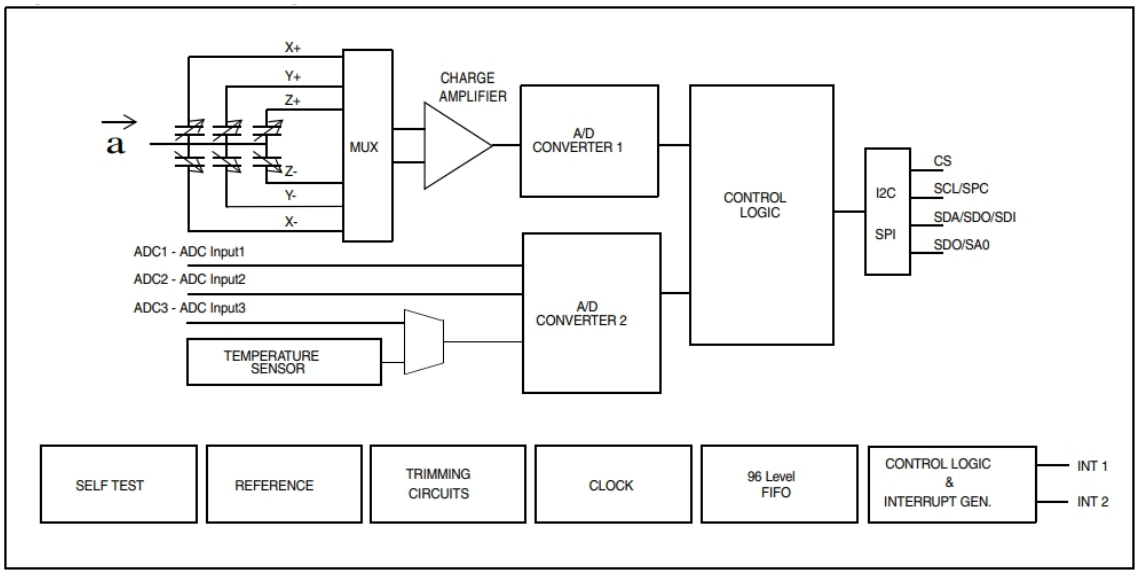



Figure 2.4: LIS3DH sensor schematic (Adafruit 2019*c*)

### 2.4.2 Audio

In concert with vibration data, audio signals are extremely important in measuring and analyzing mechanical systems to aid in fault diagnosis. A change in sound is usually the first symptom of a fault that could later lead to a breakdown. Because most of the sounds of a car come from its engine, recording the engine sound becomes paramount in fault diagnosis.

It has been reported that 99% of mechanical failures are preceded by noticeable indicators. We saw in the previous section that vibration analysis is very prevalent. More recently, acoustic signals are preferred over other types of signals because of their airborne properties. Most commonly, a microphone is used to record acoustic signals and converts sound waves into electrical signals (Yadav, Tyagi, Shah & Kalra 2011).

The main disadvantage to acoustic signals is its non-localization property. This means that while it may be possible to detect a fault or different operating sounds, we have no way of identifying specific components or events. It does however have an advantage over vibration sensor data in being able to pick up changes faster, where using vibration data can result in the remaining operational life of the component being degraded more.

In a study by Yadav et al. (2011), audio data was collected using PCB 130D20 piezoelectric microphones. Similar to the study we saw in Section 2.4.1, these sensors are expensive and high range, much more than required in the scope of this project. The maximum frequency for the engine state used in the study is 12 kHz, and a Nyquist criterion for the sampling frequency was chosen to be 50 kHz. The audio data collected was then filtered by a band pass filter in the range of 800 to 12500 Hz and down sampled to reduce the data load.

The frequency bands for specific faults in the study are listed below in Figure 2.5, where CCN is Cam Chain Noise, CHN is Cylinder Head Noise, MRN is Magneto Rotor Noise and PGD is Primary Gear Damage. It is important to note that these are not the only faults that can occur in an engine, and were simply the ones chosen to identify in the study.

| S. No. | Fault | Frequency Band |
|---|---|---|
| 1 | CHN | 3.5-4.5 kHz |
| 2 | PGD | 2-3 kHz |
| 3 | CCN& MRN | 0.5-2 kHz |

Figure 2.5: Engine Fault Frequencies (Yadav et al. 2011)

A number of electrical microphone breakouts are commercially available, and are mostly based on the electret microphone which is a type of electrostatic capacitor microphone. The name comes a combination of *electr*ostatic and magn*et*, and draws its name from how magnetic fields are aligned. Electrets are made by melting a dielectric and allowing it to re-solidify in a powerful electrostatic field. As we can see below in Figure 2.6, a typical electret pre-amp circuit uses a FET (Field Effect Transistor) in a common source configuration. The FET is powered externally by the supply voltage and the resistor sets the gain and output impedance.

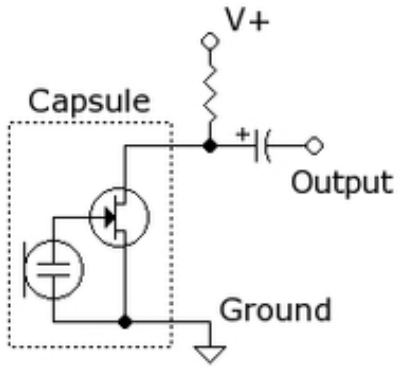



Figure 2.6: Electret Microphone circuit diagram (**?**)

The microphone polar pattern is very relevant when embarking to record engine sounds. Typically, microphones come in 3 different polar patterns and can make a large difference to the audio being taken. The three patterns can be seen below in Figure 2.7. The omni-directional pattern is equally sensitive to sound in all directions, the bi-directional is sensitive at the front and the back, and the cardioid is least sensitive at the back.

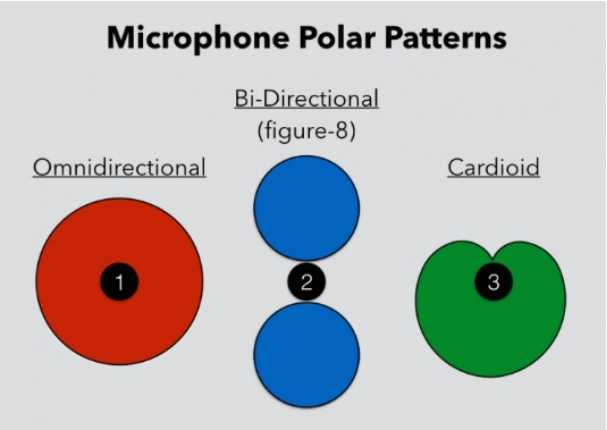



Figure 2.7: Microphone polar patterns (E-Home Recording Studio 2019)

Additionally, the microphone placement will play a large part in what kind of audio is captured. Depending on the microphone sensor package, gain can be automatic, adjustable, or fixed. Placing a microphone with large fixed gain right next to a large signal source will result in nothing but static and clipping of the recorded signal. Ideally, a microphone with an automatic gain feature should be used. Since the proposed project performs data logging on a moving vehicle, microphone placement should take into consideration the potential environmental sounds that would be encountered in a typical trip, as well as the requirement to be far enough away from the engine to reduce clipping.

An example of an audio sensor package is the Adafruit Electret Microphone amplifier, MAX9814 (Adafruit 2019*a*). This package contains Automatic Gain Control (AGC), which allows the loud sounds to be quietened and the soft sounds to be amplified. Unfortunately the microphone is omni-directional, and care will have to be taken when recording in an environment with external noises.

A simplified block diagram for the Adafruit MAX9814 microphone can be seen below in Figure 2.8. The LNA (Low Noise Amplifier) block has a fixed gain of 12 dB, while the VGA (Variable Gain Amplifier) is able to automatically adjust the gain from 20 dB to 0 dB. Finally, the output amplifier offers selectable gains to the user or 8dB, 18 dB and 29 dB.

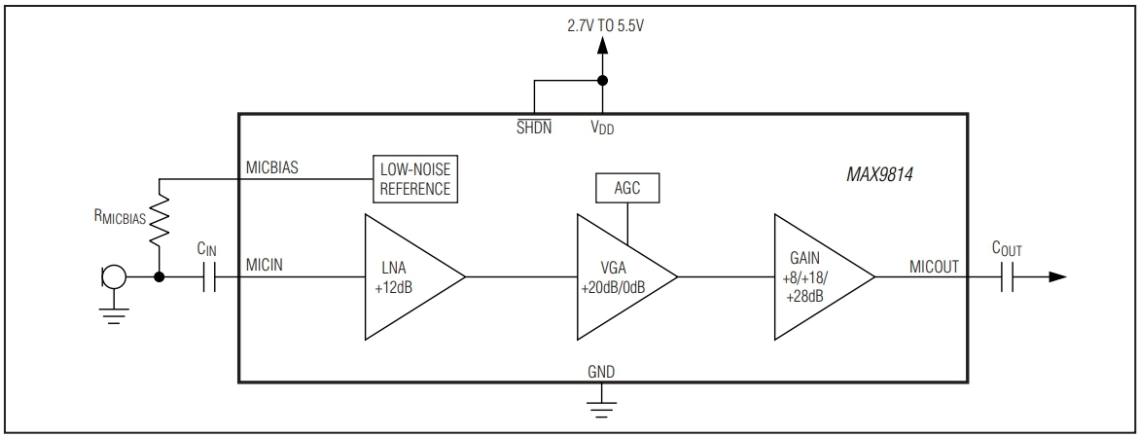



Figure 2.8: Adafruit MAX9814 block diagram (Adafruit 2019*a*)

### 2.4.3 Location and Distance

A tracking system employing GPS (Global Positioning Satellites) provides an accurate measure of position, velocity and time information for any system operating within a mobile communications system. Common systems employ a sensor, communications link, workstation, and a GPS reference receiver (Brown & Sturza 1993). The sensor is able to be operated autonomously following its initialization, and then continues to make time and frequency difference measurements. The raw data is then provided to the workstation via the communications link, and provides corrections in order to determine position and velocity.

In any condition monitoring system, velocity and operating time are going to be critical variables in trend monitoring. This data can also be used simultaneously for a delivery company who wishes the customer to be able to determine where their package is.

For optimal performance of a GPS system, there are a number of requirements for the antenna and supporting hardware and can be seen below (u-blox 2009):

- A low level of directivity.
- Good antenna visibility to the sky.
- Good matching between antenna and cable impedance.
- High Gain
- Filter

### GPS antenna

In order to implement location and distance tracking, it is necessary to include an antenna. Since most GPS receivers and external antennas in the micro-controller world use small but blocky ceramic antennas, it became a goal of the project to design and implement an external patch antenna with a different physical profile to supplement the GPS receiver, which could be mounted on the vehicle skin.

Generally speaking, there are two types of antenna, Active and Passive. Passive antennas contain only the radiating element and can occasionally contain a matching network to match the connection to a desired impedance. Active antennas contain a Low Noise Amplifier (LNA), which is helpful in a number of ways. The losses in the cable after the LNA will not affect the noise figure of the GPS receiver, and the LNA will help to reduce the overall noise figure in the system, resulting a better sensitivity (u-blox 2009).

Figure 2.9 below helps break down the major differences between an active and passive antenna which will be useful in deciding which path to take in design. Since the GPS antenna will be placed externally to the microprocessor, an active antenna must be used.

| Active antenna | Passive antenna |
|---|---|
| – Needs more power (10 – 60 mW) than a passive antenna | + Does not add anything to the power budget |
| + Is more tolerant to minor impedance miss-match or cable length than passive antenna (see section 5.3). | – Antenna must be connected with a carefully designed micro strip or strip line of maximum 10 cm to the GPS receiver to ensure good GPS performance. |
| + Helps to keep the receiver noise figure low. | – Jamming signals coupled into the micro-strip or strip line negatively affect the performance. |
| + Is less affected by jamming into the antenna cable than a passive antenna (if equipped with filter). | – RF design experience is required to properly design a passive antenna. |

Figure 2.9: Active versus Passive Antennas (u-blox 2009)

Antenna placement on the vehicle is also very important and can be seen below in Figure 2.10.

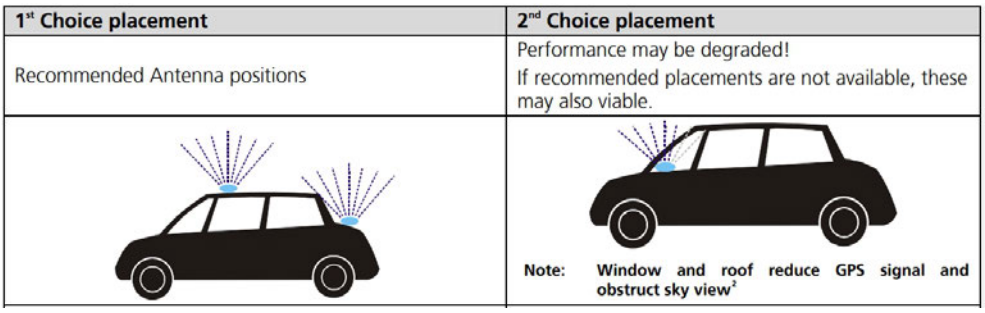



Figure 2.10: GPS Antenna placement (u-blox 2009)

Micro strip patch antennas are among the most common antenna types in use today, particularly in the frequency range 1 to 6 GHz. Its flat profile and reduced weight as compared to other parabolic antennas make it more attractive to airborne and space applications in their initial development, and then by extension to handsets, GPS receivers and other wireless applications (Banu, Prabhu & Sasikala 2015). They are also extremely versatile because of their ability to be printed on a circuit board.

In Figure 2.11 below we can see a microstrip antenna fed by a transmission line.

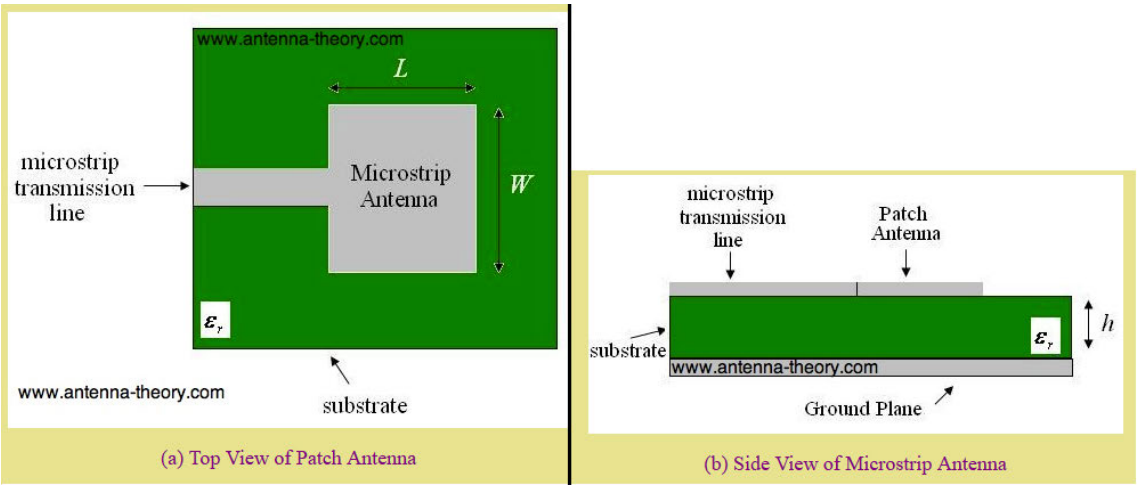



Figure 2.11: Microstrip Patch Theory 1 (Antenna-Theory.com 2011)

The patch, transmission line and ground plane are made from a high conductivity metal such as copper and the frequency of operation is determined by the following mathematical relationship seen in Equation 2.2, where L and $\epsilon$ can be seen in Figure 2.11 and c is the speed of light in a vacuum, 2999, 792, 458 m/s. The value W, or width, has an effect on the input impedance and for this type of feed method, the input impedance is typically

high. The thickness of the ground plane is not critical, but should be more than 1/40th of a wavelength.

$$f_c = \frac{c}{2L\sqrt{\epsilon_r}} \tag{2.2}$$

Next, seen in Figure 2.12, we can begin to see why the antenna radiates. The antenna has an open circuit end, which means that the current will be zero at the end, maximum in the middle of the patch, and ideally, zero at the start of the patch.

Since the antenna is an open circuit transmission line, the voltage reflection coefficient ($\Gamma$) will be 1. This also implies that the voltage and current will be out of phase. Therefore, at the end of the patch, current will be zero and voltage will be maximum. At the beginning of the patch, current will be zero again and voltage will be a minimum. This produces the fringing fields we can see in the figure. The fringing E-fields on the edges will now add up in phase and produce the radiation of the microstrip antenna.

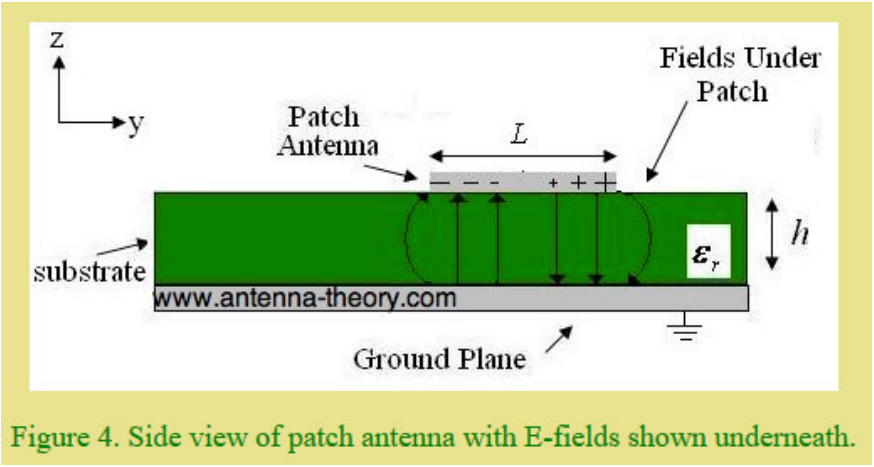



Figure 2.12: Microstrip Patch Theory 2 (Antenna-Theory.com 2011)

The directivity of the patch antennas is approximately 5-7 dB. The fields, as seen in Figure 2.13, are linearly polarized when viewed in the horizontal direction, which is ideal for a GPS antenna that is placed facing the sky.

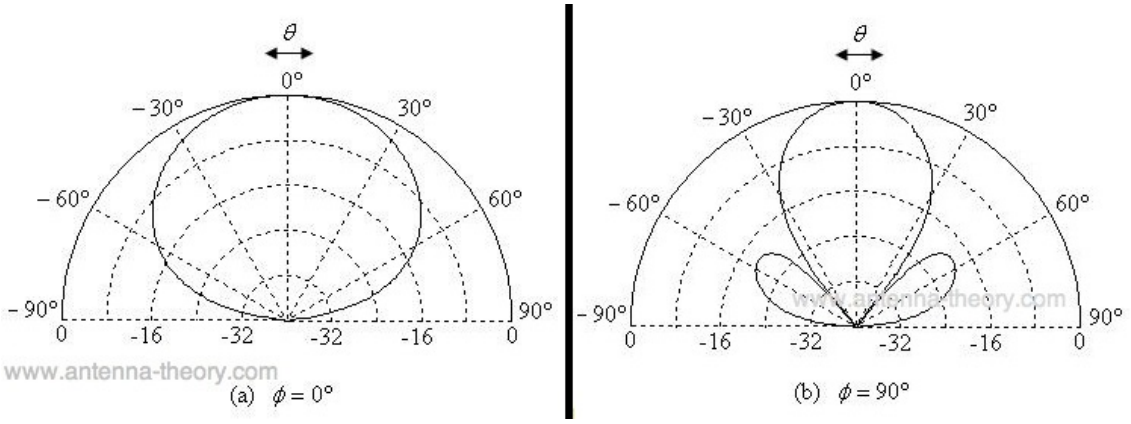



Figure 2.13: Patch Antenna Directivity (Antenna-Theory.com 2011)

It is also important to note at this point that patch antennas are very limited in bandwidth and will exhibit narrow band characteristics. Typically, the bandwidth of a rectangular microstrip antenna is 3% (Antenna-Theory.com 2011).

The previous feed method has a fixed high impedance, as seen in Figure 2.11. We would like to modify the feed and change the input impedance. We know from the previous discussion that the current is low at either end of the patch and increases in magnitude towards the middle. Therefore, we could reduce the input impedance if we simply moved the feed closer to the middle. This can be seen below in Figure 2.14.

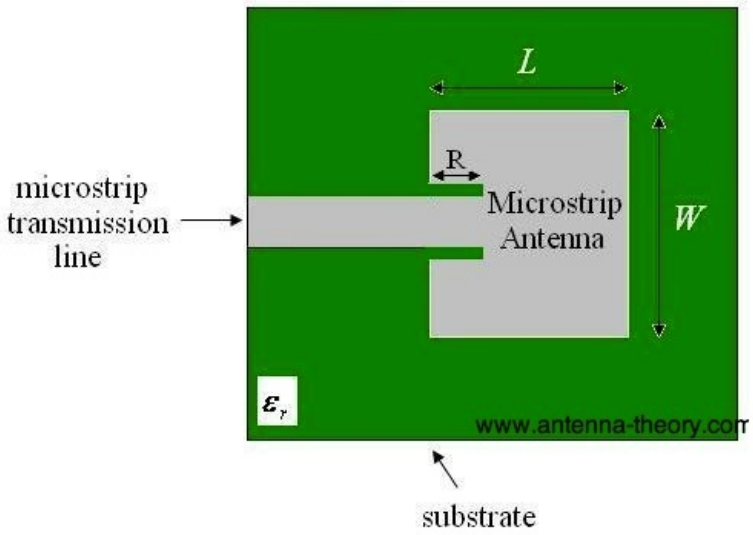



Figure 2.14: Microstrip Patch Theory 3 (Antenna-Theory.com 2011)

Using Equation 2.3 below we could theoretically calculate and manipulate the inset feed to produce the desired input impedance. $Z_{in}$ is the input impedance if the patch was fed

from the end, and R is the distance into the patch as seen in Figure 2.14.

$$Z_{in}(R) = cos^2(\frac{\pi R}{L})Z_{in}(0) \tag{2.3}$$

A study by Yegin (2007) researched the on-vehicle gain patterns for GPS antennas. While GPS is commonly used for vehicle operations, little is known on how the gain patterns change due to different mounting positions and elevations. The study investigated a number of elevation angles for the antenna to be mounted at, from 10° to 90°, and a number of different mounting positions on the vehicle chassis as seen below in Figure 2.15. Each position and elevation angle were then compared to each other.

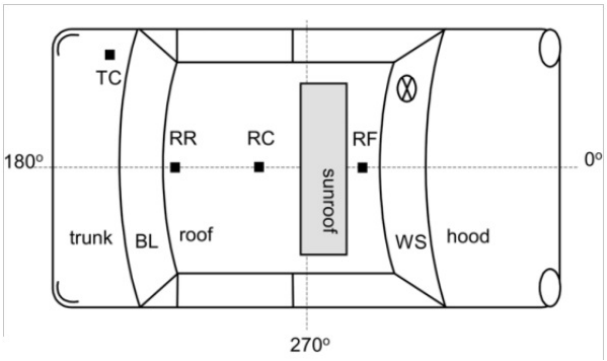



Figure 2.15: Antenna mounting positions on the vehicle (Yegin 2007)

In the study, it was determined that the roof center position was the optimum spot for any elevation angle above 60°, although the roof rear position was a very good compromise for any elevation angle. The trunk corner position was found to exhibit moderate gain at higher elevation angles, but deep nulls at lower angles.

### Low Noise Amplifier

In order to design an active antenna, a LNA must be included. Figure 2.16 below shows a typical GPS receiver setup with an active antenna. As we can see in the figure, a LNA is included directly after the antenna stage in order to achieve the best results.

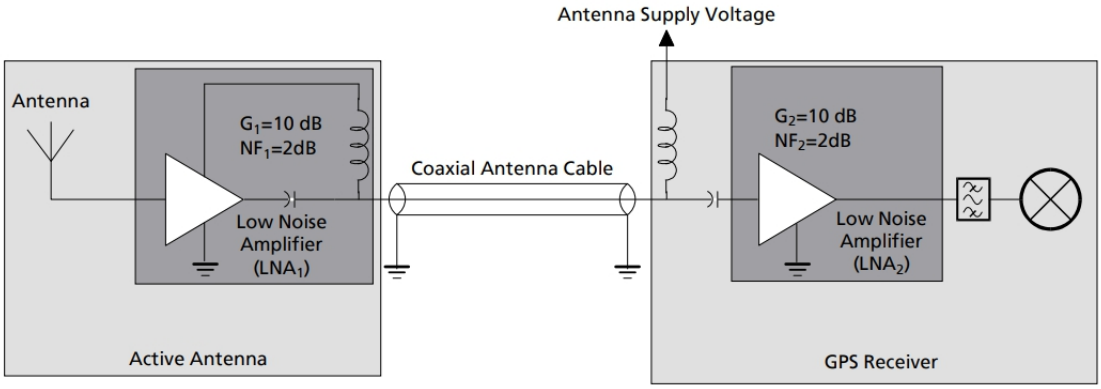



Figure 2.16: Typical Low Noise Amplifier connection (u-blox 2009)

A range of LNAs exist, from single chips such as the ALM-1612 GPS LNA-Filter Front-End module to the advanced Nooelec SAWbird iO Barebones, which combines all the necessary electronics on a PCB with SMA connectors for immediate connection. Both would be suitable to amplify a GPS patch antenna signal, but the ALM-1612 chip would require board manufacture, component soldering and test phases before connection. A suggested PCB Layout board for the ALM-1612 from Avago Technologies can be seen below in Figure 2.17. The data-sheet also contains links to all of the relevant Gerber files that would be required for PCB manufacture.

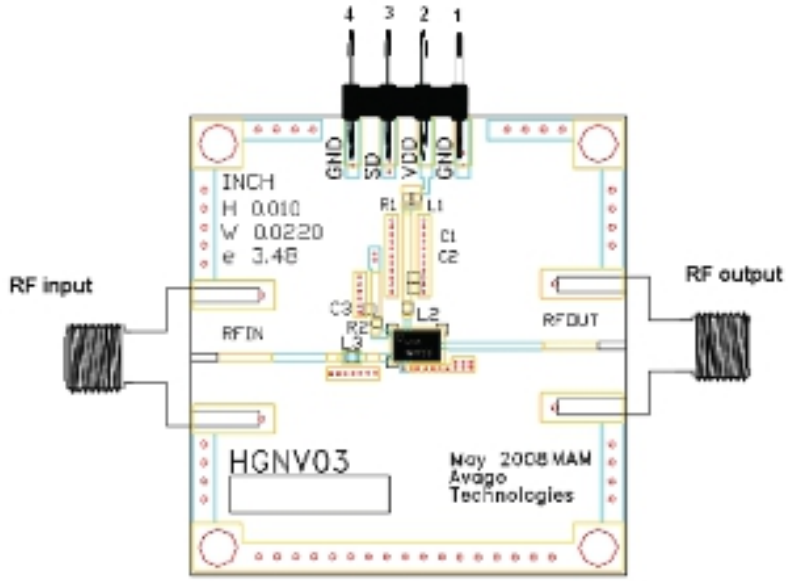



Figure 2.17: ALM-1612 Suggested board layout (Avago Technologies 2012)

### GPS receiver

GPS satellites circle the earth twice a day in a very precise orbit. Every GPS satellite transmits a signal and orbital parameters that allow GPS devices to determine the position of the satellite (Sharma 2017). GPS receivers then use this information to determine the users exact location. Basically, the GPS receiver measures the distance to each satellite it can see by the length of time it takes to receive the signal. A 2-D position (Longitude and Latitude) will require at least 3 satellites and a 3-D position (altitude) will require 4. After the position has been determined, a GPS unit is capable of calculating Speed, Bearing, Track, Trip Distance and Distance to destination.

The signals that the GPS satellites transmit are very low power but can travel in line of sight. This means they can pass through transparent objects but not solid ones. The signals contain 3 types of information (Sharma 2017):

- Pseudo-random code: an ID code to identify the satellite.
- Ephemeris data: determines the satellites position, current health and date/ time.
- Almanac data: tells the GPS receiver where each GPS satellite should be in the sky, and in relation to other satellites.

Micro-controller based GPS receivers are commonly in use today, and used widely across location-based research for testing and proof of concepts in similar projects. One such project was carried out by Ramani & Valarmathy (2013), which investigated a vehicle tracking and locking system based on GPS. The micro-controller used the project was based on the Atmel AT89C52, which is fairly basic in terms of memory and processing speeds, and utilized a commonly available GPS receiver. While the applications of the technology in this project are different to those being proposed currently, the operation and interactions between the GPS receiver and micro-controller are the same.

For the home user, there are a number of GPS receivers available for integration into a micro-controller based project. An example and standout receiver is the Adafruit Ultimate GPS Breakout. It has 66 channels with 10 Hz updates. It contains a passive ceramic patch antenna in the breakout board, but also has an external $\mu$.Fl connection for including external antennas.

We can see the schematic for the Adafruit Ultimate GPS breakout below in Figure 2.18. The receiver is able to detect the presence of an active antenna (provided this signal is greater than 4 mA), and provides an input path through a SPDT (Single Pole Double Throw) switch and SAW (Surface Acoustic Wave) filter.

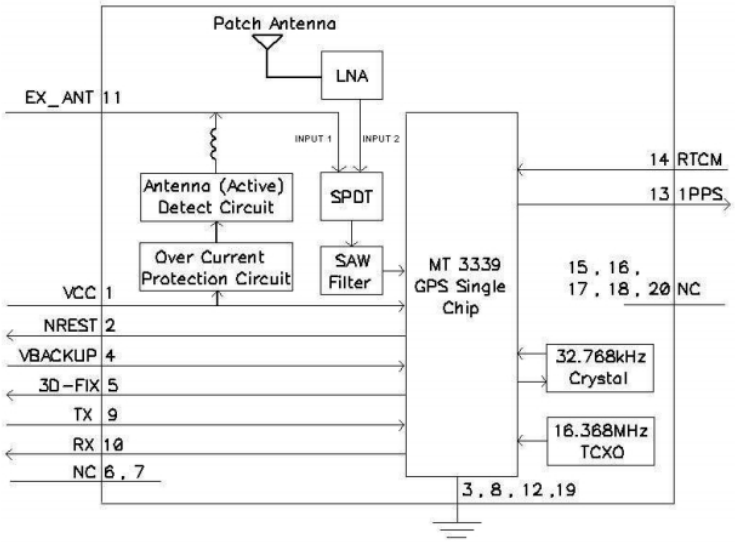



Figure 2.18: Adafruit Ultimate GPS breakout schematic (Adafruit 2019*b*)

### 2.4.4 CAN Bus and OBD2

The CAN (Controller Area Network) bus is used in automobile and aerospace industries to allow communication between ECUs (Electronic Control Units) and their sensor networks. It allows components to communicate on a single or dual wire network bus up to 1 Mbps. This eliminates the need for complex, dedicated wiring in between each ECU (CSS Electronics 2019). A simplified example of a CAN bus can be seen below in Figure 2.19.

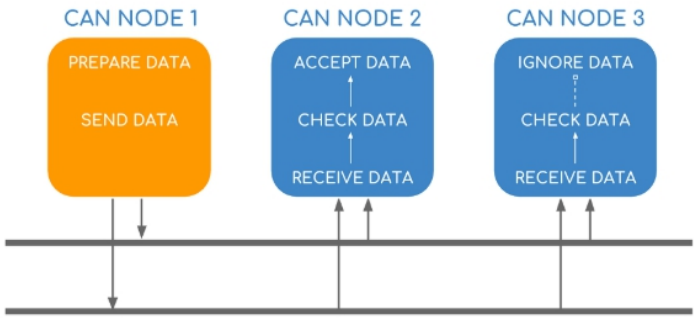



Figure 2.19: CAN Bus explained (CSS Electronics 2019)

The data sent over the CAN bus is broken down in Figure 2.20. This is an example of CAN 2.0B, the format used in J1939 protocols for heavy vehicles, and contains a 29 bit identifier. Smaller vehicles use an 11 bit identifier for OBD2 protocols, but this is the primary difference between the formats. OBD2 is a self-diagnostic and reporting capability that was discussed in Section 1.3.2.


Figure 2.20: CAN bus message breakdown (CSS Electronics 2019)

In order to extract the data from the CAN bus via OBD2 protocols, there are a wide range of available products and can be broken down into two main categories: OBD2 scan tools and OBD2 code readers. Scan tools are more expensive, but provide the ability to read manufacturer specific codes, see live data and record it, and provide advanced troubleshooting support. Code readers can read and clear generic codes, but this is limited in scope (Collins 2019).

One example of an OBD2 scanner commonly used for micro-controller projects is the OBD-II UART manufactured by sparkfun. Is is a development board that combines three different chips to extract OBD2 data from the vehicle CAN bus. These are the ELM327 OBD to RS232 interpreter, the MCP2551 High-speed CAN transceiver and the STN1110 multi-protocol OBD to UART interpreter. This allows great flexibility in design and connection options. A simple FTDI (Future Technology Devices International) Basic breakout allows serial communications to a Personal Computer.

A block diagram for the ELM327 chip can be seen below in Figure 2.21. The OBD interfaces directly with the vehicle, interprets the data and outputs via the RS232 interface.

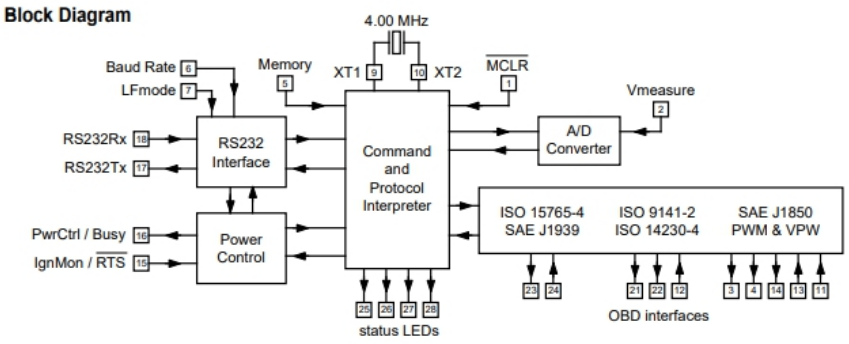



Figure 2.21: ELM327 Block Diagram (ELM Electronics 2019)

A wide variety of projects and research has been carried out in the field of vehicle health monitoring based on extraction of OBD2 data. These will be used to inform the project and its capabilities in the field of OBD2 recording. A study by Amarasinghe, Kottegoda, Arachchi, Muramudalige, Bandara & Azeez (2015) investigated a cloud-based vehicular data acquisition and analytic system for real-time driver behavior monitoring, trip analysis and vehicle diagnostics. The system consisted of an OBD2 port using Bluetooth to transmit data, a mobile app, and a cloud based back-end. The system was able to alert the driver via a smart phone about any anomalies in the engine operation such as coolant temperature, fuel leaks and impending sensor failures. This project is a good representation of the research in the field, and shows the depth of investigation and prediction available by utilizing OBD2 data.

## 2.5 Micro-controllers

Development boards are becoming increasingly popular with the maker communities, and each year brings more innovative solutions to the market. In order to breakdown the list, a comprehensive review was carried out and the suitability of each was compared to applicable criteria as follows:

- Physical Size
- Processor speed and type
- Memory size and flexibility
- GPIO (General Purpose Input Output)
- IDE (Integrated Development Environment)
- Cost

This does not list every development board that is available at the time of writing, and simply illustrates the most popular boards that could fit in the scope of the project.

Table 2.1: Micro-controller breakdown 1

| *Name* | *Processor* | *Memory* | *GPIO* |
|---|---|---|---|
| 1. Teensy 3.6 | ARM Cortex-M4 180 MHz | 1M Flash, 256KB RAM, 4K EEPROM | 62 |
| 2. BeagleBone Black | AM3358 1GHz | 4GB Flash, 512MB SDRAM | 94 |
| 3. Raspberry Pi 3 | Cortex-A53 1.4GHz | 1GB LPDDR2 SDRAM | 40 |
| 4. Thunderboard Sense 2 | ARM Cortex-M4 38.4MHz | 1MB Flash, 256KB RAM | 20 |
| 5. Arduino Uno R3 | ATMega328P 16 MHz | 32KB Flash, 2KB SRAM, 1KB EEPROM | 20 |
| 6. Particle Photon | ARM Cortex-M3 120 MHz | 1MB Flash, 128KB RAM | 18 |
| 7. Udoo Neo | ARM Cortex-A9 1GHz | 512MB DDR3 | 36 |

Table 2.2: Micro-controller breakdown 2

| *Name* | *IDE* | *Cost* | *Operating Voltage* | *Physical Size* |
|---|---|---|---|---|
| 1. Teensy 3.6 | Various C Compilers or Arduino Software (Teensyduino) | $56.32 | 3.3V | 2.4×0.7 inches |
| 2. BeagleBone Black | N/A: Debian Linux OS | $130.61 | 3.3V | 3.5×2.15 inches |
| 3. Raspberry Pi 3 | N/A: Any suitable OS | $54.96 | 3.3V | 3.34×2.2 inches |
| 4. Thunderboard Sense 2 | Simplicity Studio | US$36 | 3.3V | 1.77×1.18 inches |
| 5. Arduino Uno R3 | Arduino Software | $44.50 | 5V | 2.7×2.1 inches |
| 6. Particle Photon | Particle.io/ cloud-based IDE | $34.95 | 3.3V | 1.4×0.78 inches |
| 7. Udoo Neo | N/A: Android Lollipop and UDOObuntu 2 | $100.72 | 3.3V + 5V | 3.5×2.32 inches |

There are a few key standouts from Table 2.1 and Table 2.2. Since we will be processing audio, handling multiple sensor inputs, and writing this data to storage, processing speed will be important. For this reason, the Arduino Uno R3 and Thunderboard Sense 2 may struggle.

The BeagleBone Black, Raspberry Pi 3 and Udoo Neo all have high processing speeds, but this is because they are designed as standalone Operating Systems (using Linux or Android OS). For these to be used as data loggers, Python language is typically used to program sensor data collection.

Of notable mention are the Teensy 3.6 and Particle Photon. Both systems have reasonable processing speeds, memory, prices and size. Unfortunately, the Particle Photon falls down in the GPIO region, with only 18 I/O ports. Its cloud based IDE looks interesting however, and support for a local IDE is also provided. The Particle Photon is designed more as an IoT device and has an inbuilt Wi-Fi chip to enable a connection to the internet. The

Teensy 3.6 also has an SD (Secure Digital) card slot and supporting libraries which may make it very invaluable as a standalone data logger used in a vehicle.

## 2.6 Data storage

Most data loggers use a memory card or flash stick to save data. More advanced models have the potential to send the data automatically over an Ethernet connection or through a wireless network (Bluetooth, Wi-Fi, LoRa etc). The final application of the data loggers is the determining factor in deciding how to store data (CAS DataLoggers 2019).

For this project, Wi-Fi or Ethernet connections will not be available in the vehicle's transit period. This means that some form of local memory storage must be used until the module returns to base. MicroSD card breakouts are common in the micro-controller world, and provide a useful storage location for data logging projects.

In a similar research project developed by Baker (2014), an SD card and FAT32 file storage system is used to perform sensor measurements and historical data storage. The data is then sent to a larger, cloud-based website for further dissemination via an Ethernet connection.

When interacting with SD cards there are a number of common pitfalls in micro-controller design. SD cards are strictly 3.3V devices and the power draw is quite high compared to most of the other breakouts in the market (100 mA or more). The logic interfacing with the pins must be 3.3V, and must be very 'square'. Level shifters may be required in cases where the interfacing wires travel over a significant distance. Lastly, SD cards employ 'raw' storage, and have no set structure. In order to interface the card with an OS, FAT16 or FAT32 should be used (Adafruit 2019*d*).

## 2.7 Data Transmission

For a vehicle based data logging system to be autonomous, some form of data transmission should occur at end of a trip/s. This has a number of reasons: To ensure the data is not being overwritten, to remove the need for a manual transfer of data, and to provide a timely analysis of the data before any faults can compound.

A number of wireless transmission methods are available. They are generally divided into four broad categories and can be seen below (TechTarget 2007):

- **WPAN (Wireless Personal Area Network):** These are made to reach short distances, no greater than 10m. IrDA (Infrared Data Association) and Bluetooth and common WPAN examples. Zigbee and UWB (Ultra-Wide Band) are also emerging technologies in this area.
- **WLAN (Wireless Local Area Network):** The most widely deployed example, WLAN incorporates Wi-Fi and is the most popular protocol in this category. It can deliver up to 200 Mbps at distances up to 100m.
- **WMAN (Wireless Metropolitan Area Network):** This technology offers up to 75 Mbps over links that can reach a few kilometers, and is generally used for providing mobile broadband access across cities.
- **WWAN (Wireless Wide Area Network):** WWAN differs from WLAN technology by using mobile networks such as LTE (Long Term Evolution) or CDMA2000 to transfer data.

Of these, there are only few suitable technologies for this project. LoRa and LoRaWAN is becoming increasingly popular in maker community, and features low energy transmissions with greater noise immunity at distances up to 40 km. Unfortunately, gateways are expensive and data rates are low, in the order of 27 kbps . Bluetooth is a standard that millions of devices use to communicate with each other, and is found in mobile phones, computers, microcontrollers and more. Bluetooth modules are commercially available and at a reasonable cost. Wi-Fi is another very common method of connecting wirelessly, and is similar in price to Bluetooth modules. Wi-Fi has the added benefit of being able to connect to the internet via gateways found in most homes and workplaces. Lastly, Zigbee deserves a mention. Xbee modules use wireless communications using Zigbee protocols to create communications networks and have a large range (Core Electronics 2019).

One such system was developed by Choque, Davila, da Silva & da Rocha (2017). The goals of the project were to construct a Wireless Local Area Network (WLAN) multi-Data Acquisition System (DAS). This is very similar in scope to the proposed project idea, although it utilizes different sensors. In this case, three sensors monitoring light intensity, temperature and voltage in the form of an analogue signal were collected and

converted into digital data using code on the Arduino microprocessor. Following this, the data was then broadcast to the internet by use of the WLAN technology supported by an add-on to the Arduino board.

Many other systems exist in the form of 'Smart Homes', which monitor house conditions and provide optimal responses, as well as inter-connectivity through smart devices (Adriansyah & Dani 2014).

We can see an example of a Wi-Fi breakout below in Figure 2.22. This is a 802.11bgn capable module, and used for networking between devices. It uses SPI (Serial Peripheral Interface) to communicate with the micro-controller, and provides reliable packet streaming at up to 12 MHz.

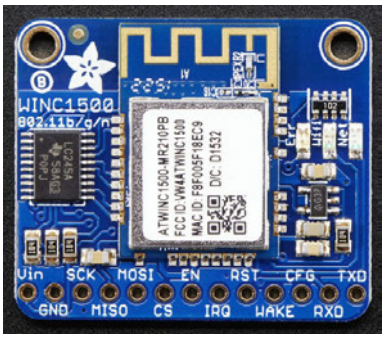



Figure 2.22: ATWINC1500 WiFi Breakout (Core Electronics 2019)

## 2.8 Data Analysis Software

A number of software packages are required to analyze and present the data arriving from the sensors/ micro-controller and will be discussed in the following sections.

### 2.8.1 MATLAB

MATLAB (Mathworks 2018a) is the industry standard for signal processing and analysis. It includes signal processing packages, techniques, and support in order to analyze time-series data. It also provides a way to implement DSP (Digital Signal Processing) algorithms on a PC and can acquire, measure, transform, filter and visualize signals without being an expert in the theory of signal processing (Mathworks 2019).

### 2.8.2 Audacity

Audacity was developed by a group of volunteers and is an open source program for Windows, Mac OS X and GNU/ Linux. It features a multi-track audio editor and recorder and is able to import Import/ Export many different types of sound files including RAW. It also has a spectrum window mode in order to perform detailed frequency analysis (Audacity 2019).

### 2.8.3 Google Earth

Google Earth is a program that creates a 3D image of Earth that is based mainly on satellite imagery. The program works by superimposing satellite and GIS data onto a globe. Users are able to explore by address, coordinates or by keyboard and mouse. The feature of google maps that makes it most relevant in this project is its ability to allow users to upload their own data and overlay this data on the maps. The data may be in the form of KML (Keyhole Markup Language), or can simply be a CSV (Comma Separated Value) file (Wikipedia 2019).

### 2.8.4 OBD2 scan tools

As alluded to in Section 2.4.4, OBD2 scan tools are useful in providing a larger amount of data on the vehicle under test, when compared to OBD2 code scanners. A number of OBD2 programs are available for purchase online, and can be used with the majority of OBD2 hardware. OBD Doctor is a leading car diagnostic software and has the ability to turn a computer into a highly capable automotive scanner (OBD Doctor 2019). This software, and many like it provide the ability to view engine data in real time, in both numerical and graphical formats.

## 2.9 Data Analysis

Utilizing the software mentioned above in Section 2.8, the raw data extracted from the sensor packages can be analyzed to extract meaningful information about the engine's operation. This will give us an indication of trending or changing operating conditions in the vehicle.

### 2.9.1 Fast Fourier Transform

The Fast Fourier Transform (FFT) is a speedier version of the Discrete Fourier Transform (DFT), and produces the same results but in a faster time frame. The DFT takes a discrete signal in the time domain and transforms it into a discrete frequency domain representation. This gives us the ability to examine the spectrum of a signal (University of Rhode Island 2019). The image in the Figure 2.23 below gives a good representation of how the basic FFT works.

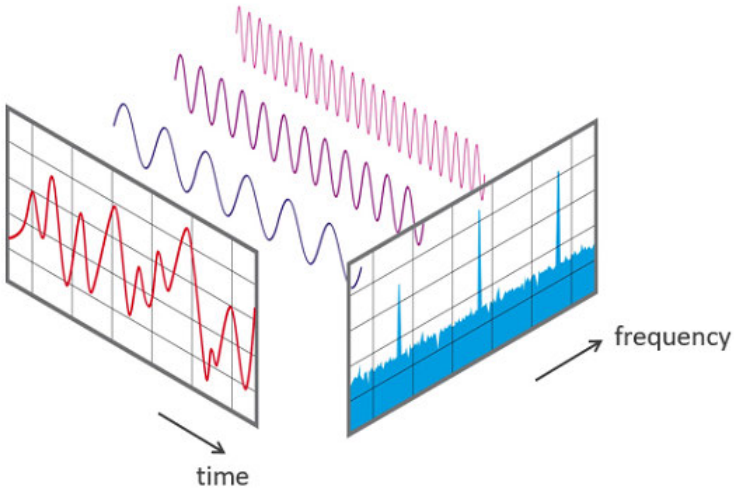



Figure 2.23: Fast Fourier Transform - Basic (NTI Audio 2019)

The defining FFT equation can be seen below in Equation 2.4, which transforms N time samples x(n) into N frequency samples X(k).

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-jn\omega_k} \qquad \omega_k = \frac{2\pi k}{N} \tag{2.4}$$

In a study by Sujono (2014), a method for detecting detonation (knocks) with microphone sensors and an active filter was investigated. After the raw data was collected, analysis and classification utilized FFT algorithms and Euclidean Distance measures to determine fault data. The FFT algorithm was used to show the difference in frequency bins between a healthy sample and a fault sample. It showed a clear difference, where the fault data FFT indicated the presence of high frequency components and the healthy data FFT did not.

The study develops the identification process further by measuring the Euclidean Distance between the regression pattern for a signal with a fault and that of the reference signal. The regression signal in this study can be defined as an approximation of the original signal, after being filtered with an active type Sallen-Key filter (which is a second order High Pass Filter (HPF)). This gives a determination of whether the input signal is a normal engine, or suffers from a knock fault.

While this study is limited to detonation faults, it shows that FFT analysis of a signal is still extremely relevant to engine faults and leaves much to be elaborated upon.

### 2.9.2 Linear Predictive Coding

Linear Predictive Coding (LPC) uses the concept that a linear, discrete-time system can be modeled by predicting the next sample in a time series as a product of past samples in that series, weighted according to position delay. The general concept is to sample the audio, process frames in a small window, and derive the parameters that best represent that data in the frequency domain (Leis 2018). Using this concept it becomes possible to represent a large number of discrete points with a smaller number of coefficients that is able to approximate the data and can be seen in Equation 2.5, where the prediction for each sample $\hat{x}$(n) is formed as a weighted sum. The true sample x(n) is the prediction plus an error term as seen in Equation 2.6.

$$\hat{x}(n) = \sum_{k=1}^{P} a_k x(n-k) \tag{2.5}$$

$$x(n) = \hat{x} + e(n) \tag{2.6}$$

LPC is a very powerful tool used in speaker recognition, based upon a linear model of speech production, and can provide an accurate estimation of speech parameters. To develop this concept further, a study by Monica Chamay, Se-do Oh and Young-Jin Kim (2013) was carried out to build a diagnostic system using LPC/ Cepstrum analysis in machine vibration. We know from this study that a lot of research on detection of faults using vibration data had been done, but it had not been possible to find an appropriate method of detecting faults in specific cases where frequency analysis falls short. The study showed that it was possible to apply LPC to extract an exact pattern based on the principal characteristics of the data being represented. This statement also introduces a possibility that LPC could be used in the context of this project, and apply to both audio and vibration data profiles.

On its own however, LPC does not classify data. It will simply represent the data in a more efficient manner, ready to be classified. In order to determine fault data, we must use some order of statistical measure between the healthy data and the fault data. This is explained in the following section.

### 2.9.3 Mahalanobis Distance

Mahalanobis Distance (MD) is the distance between two points in multivariate space (Statistics How To 2017). In a regular Euclidean space, variables are represented by axis at right angles where the distance between points can be measured with a ruler. For uncorrelated variables, Euclidean Distance is the same as Mahalanobis Distance. If the values are correlated, axes are not at right angles anymore and it become impossible to measure. MD solves this conundrum, and can measure distances between points, even for correlated values. It is a measure from the centroid, where the means of variables intersect, and can be seen in Figure 2.24. A larger MD means further away from the centroid.

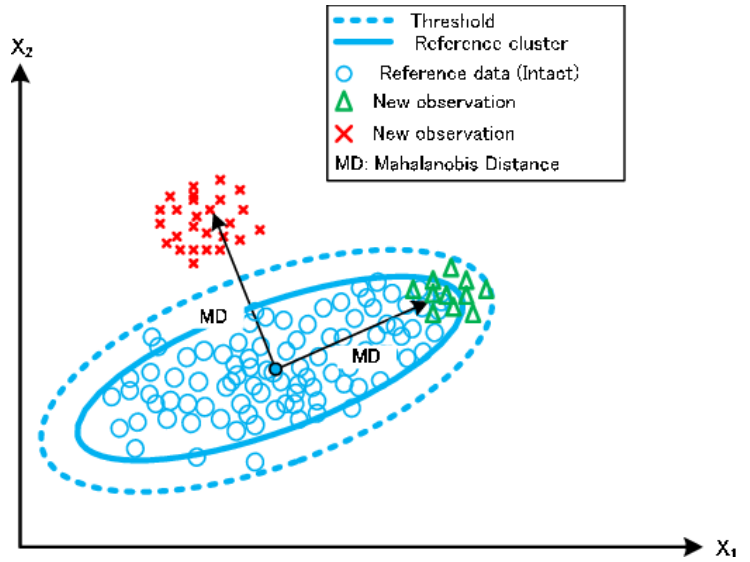



Figure 2.24: Mahalanobis Distance Graphic (Chul-Woo Kim 2013)

Following on from Section 2.9.2, the study by Monica Chamay, Se-do Oh and Young-Jin Kim (2013) showed that it was possible to classify engine vibration data with LPC coding. Further to this, they showed that in order to classify fault data they had to use some sort of statistical measure. This was in the form of MD. Euclidean Distance was discussed and discarded for two reasons: Euclidean Distance is extremely sensitive to scale of variables involved, and the Euclidean distance is blind to correlated values. The problems of scale and correlation are not an issue when using MD. Additionally, MD is more sensitive to small differences between variables and is useful in the detection of outliers.

They were able to extract behavior patterns from the healthy and fault data and compare them using Mahalanobis Distance. These comparisons showed a marked difference, and were used to successfully identify system failures with 100 percent accuracy as seen in Figure 2.25.

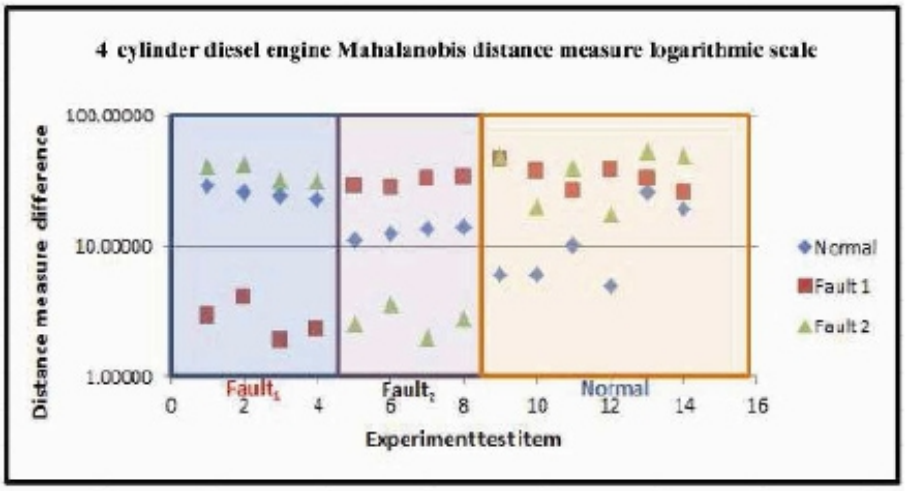



Figure 2.25: Mahalanobis Distance in Practice (Monica Chamay, Se-do Oh and Young-Jin Kim 2013)

## 2.10 Chapter Summary

This chapter provided an in-depth look into the literature surrounding vibration and audio research on diagnosing a fault in a combustion engine. It also discussed the current market availability and technology of hardware/ software that could be used to bring this project to fruition.

# Chapter 3

# Methodology

## 3.1 Chapter Overview

This chapter presents the methodology used to determine the viability of a low cost condition monitoring system that could be used for predictive maintenance. The methodology is further broken down into research, design and verification methods that were used to come to appropriate conclusions.

## 3.2 Introduction

A design methodology based upon the classical engineering approach was utilized in the undertaking of this research. The following quote explains the classical engineering process:

"The process of devising a system, component or process to meet desired needs, it is a decision making process (often iterative) in which the basic sciences, mathematics, and engineering sciences are applied to convert resources optimally to meet a stated objective. Among the fundamental elements of the design process are the establishment of objectives and criteria, synthesis, analysis, construction, testing and evaluation." (Accreditation Board for Engineering and Technology 1987).

The method used in this report is very similar and involved background research, an implementation method, and testing and realignment to ensure project goals were met appropriately. The primary objective of the project, previously identified within the project specification, was to implement a micro-controller based system capable of recording applicable data in a vehicles run-time, and perform analysis of this data to successfully predict a fault condition.

Existing research on data logging and fault diagnosis in vehicles involves the use of expensive sensor arrays, as shown previously in Section 2.4.1 and Section 2.4.2. Large companies use this technology to monitor their equipment and collect data on its usage in the field, as introduced in Chapter 1. It was identified in the course of this research that a suitable gap exists in the field of research for a small, micro-controller based system usable by an individual or small fleet owner. This system would not require large amounts of mechanical knowledge, and be available for a much smaller cost than the systems currently available.

## 3.3 Research Documentation

Due to the large scope of this project, it is extremely important to monitor and record relevant results for the reporting stages at the end of the project. This will include, but is not limited to: constant note taking in an appropriate medium, schematic draw-

ings, screenshots of software development, photos of circuit operation, data transmission reports, and reporting of the signal processing and trend monitoring elements. It was proposed that all reporting data be stored on a personal computer, with regular backups made to a cloud drive in order to preserve the information in the case of an accidental loss.

- Notes are made on paper during each design, build or test phase, with regular uploads and condensed versions added to Word documents.
- Schematics are made in MicroCap or a similar hardware design/ simulation package and updated at regular intervals after any configuration changes.
- Software development is conducted in an appropriate IDE (Integrated Development Environment) package. Screenshots can be taken of operational parameters, code added into word document as an appendices, and version control is enacted regularly.
- Circuit hardware and operation in testing phases showing configurations will be photographed in a neat manner and uploaded as appropriate.
- Data transmission is monitored and reported upon, for storage upload.
- Data analysis is carried out in MATLAB, Audacity, OBD2 scan tools and any other appropriate medium, and recorded appropriately.

## 3.4 Methods of Analysis

The ability of any good research project to produce and display its findings lie in the quality of the analysis.

“Monitoring and evaluation plans, needs assessments, baseline surveys and situational analyses are all located within a project cycle and require high-quality data to inform evidence-based decision-making and programmatic learning.” (The Open University 2017)

In order to realize this, we must investigate the differences between quantitative and qualitative data, and their benefits and limitations. Being able to identify appropriate research questions, and utilizing the best methodologies will also be extremely useful.

When investigating the nature of the research data, we must define two types of data, qualitative and quantitative. Generally, quantitative data is in numerical form, and qualitative is not. The former is data collected through measuring things, and analyzed through numerical comparisons, whereas the latter is usually collected through observations and analyzed by theme (Mcleod 2017)

Obviously, due to the nature of this project, a quantitative approach to the data analysis will be most beneficial. However, if the interests of the consumer are to be observed as well, then additional economic data might be collected and analyzed in a qualitative manner in order to determine personal preferences on what sensor data they would like to monitor.

Previously, we developed research objectives, which is a key step in the planning process. We have identified knowledge gaps in the field in order to inform our research project. Within the objectives, we developed a set of 'research questions' which will help us focus on the endgame of the project.

At the end of this lies the analysis of the data, interpreting the findings and making some conclusions. In order to do this, we must determine if our methodology and data supports answering the aims of the project by answering a series of questions:

- Did we answer the aims appropriately?
- Are there different, contradictory findings from the research?
- What kind of limitations was the project operating under?
- Are there areas of the project that require additional work or research?

In order to analyze our data appropriately, our method of analysis must turn the reporting data into useful information to help with decision making and future development. We should be able to summarize the findings of the project into what was useful, irrelevant, or needing further development. Descriptive summaries will also provide useful data collection points.

In presenting the analysis, the report must provide complete transparency. Stepping through from an introduction, to the methodology, to the findings and discussion should

be a seamless process designed to provide an accurate representation of the work undertaken and how the project objectives were met. If we were to try and misrepresent the data, then the project would become irrelevant at best, and illegal at worst.

## 3.5 Quality Assurance Plan

In order to maintain the project relevancy, timelines and integrity (i.e. meeting the aims), a series of checks are in place to touch base with USQ supervisors and assessing staff.

- Regular weekly meetings will be held with the project supervisor to discuss current progress, developing scope, and difficulties.
- Project specifications, progress reports and draft submissions are set up by ENG4111 and ENG4112 subjects in order to formally report on the status of the project.

Additionally, there are internal procedures in place to keep the project on track and of appropriate quality. These will include the following:

- Component assessment and integration into the project will provide a measure of assurance that the project is running smoothly, giving an indication at crucial stages that the circuitry is capable of operating and can provide the required information.
- Determining signal processing requirements and being able to convert the data in a manner showing appropriate trends will provide the beginning of an answer to the viability of the project.
- Reporting and analyzing the work undertaken shall be ruthless, ensuring a minimum of missed opportunity for the discussion and conclusion.
- Attempting to follow the project schedule as closely as possible, while maintaining an eye on the aims and scope of the project will be useful in staying on track and within time-frames.

## 3.6 Project Breakdown

This project was separated into phases that were completed at key times throughout the year in the course of completing the final year project.

- Phase 1: Additional research and determination of viable components, development boards, software and compilers to be used in the scope of the project.
- Phase 2: Collection and determination of available resources with which to begin work, as well as write up of the project specification to satisfy ENG4111 Research Project Part 1 requirements.
- Phase 3: Preliminary design.
- Phase 4: Build and test the sensors, microprocessor and data storage, with the end goal to have a working development model that could be installed in a vehicle.
- Phase 5: Design and manufacture a GPS patch antenna.
- Phase 6: Analysis of results, additional design and re-visitation of build and test if appropriate.
- Phase 7: Integrate WiFi and Geo-located downloads.
- Phase 8: Demonstration of results in successfully diagnosing a fault condition.
- Phase 9: Dissertation report, discussion and conclusions.

## 3.7 Hardware

A list of suitable components was developed based upon the requirements identified in the literature review. In essence, the two sensors (audio and vibration) would have to be capable of measuring the engine parameters without losing too much fidelity of the data. The micro-controller had to be capable of processing and storing the amount of data that would be flowing in from the sensors, which also included a GPS input and OBD2 breakout in addition to the vibration and audio sensors. A proposed system architecture was developed early in the proceedings and can be seen below in Figure 3.1.

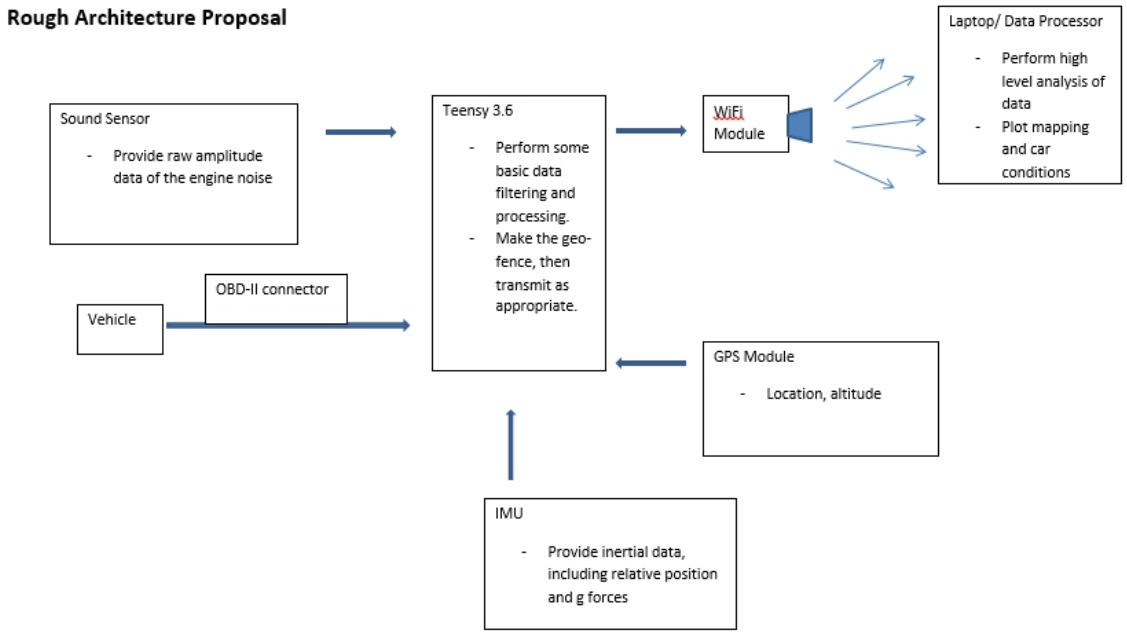



Figure 3.1: Proposed system architecture

### 3.7.1 Teensy 3.6 and IDE

The literature review identified a number of micro-controllers that may have been suitable for the project and its requirements. In the end, the Teensy 3.6 was decided upon. Its unique support by developer Paul Stoffregen has made it a stand out in the field. Due to its increased RAM and processing power, the Teensy 3.6 is capable of providing very flexible audio processing, and is still able to complete all other tasks required of the processor. It also holds a micro-SD slot, which is extremely important in a data logging project. The Teensy 3.6 can be seen below in Figure 3.2.

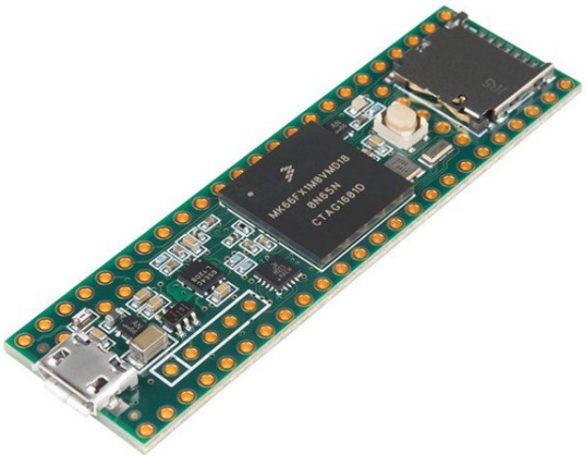


Figure 3.2: Teensy 3.6 (Core Electronics 2019)

An unfortunate drawback to the Teensy 3.6 is that it only has a 3.3V pin capability. Some other development boards in the field, such as the Arduino models, provide support for 5V protection which allows much greater flexibility in design. When compared to the Proton development board seen in the literature review, the Teensy 3.6 does not have a wireless capability. This is something that would have to be integrated into the project externally.

The Teensy models can be programmed using a bootloader and any C editor such as Visual Studio, but the most popular method is using the Arduino IDE. This has the benefit of providing support for any cross-compatible Arduino libraries that could be used in the scope of the project.

### 3.7.2 LIS3DH accelerometer

The LIS3DH was chosen as the vibration sensor for the project and can be seen below in Figure 3.3. The range of cheap MEMS accelerometers are fairly similar across the board, although this model does provide some additional benefits. Most accelerometers in the commercial range are used for low frequency applications and typical sampling rates are not much higher than 1-2 kHz. The LIS3DH model provides sampling at up to 5 kHz, albeit at a lower bit range. As we saw in the literature review, the accelerometer must be chosen with sample rates and the Nyquist frequency in mind. While accelerometers with higher sample rates are available, these are too expensive for the purposes of this project.


Figure 3.3: LIS3DH accelerometer (Core Electronics 2019)

### 3.7.3 MAX9814 microphone

The MAX9814 microphone with AGC can be seen in Figure 3.4. This microphone was chosen for its ability to control the gain. Engine sounds are often loud and will change in pitch regularly. In addition to the AGC the microphone provides, hardware and wiring solutions exist to further reduce or increase the gain in response to the engine. This microphone breakout also has support within the Teensy community, and a number of hardware and software designs exist to complement this microphone's abilities.


Figure 3.4: MAX9814 Microphone with AGC (Core Electronics 2019)

### 3.7.4 GPS

A number of GPS components were investigated for suitability in the project, and two pathways were selected for both the antenna and the LNA (Low Noise Amplifier). It was decided to first pursue manufacturing an antenna and LNA PCB. For comparison purposes, an additional antenna and LNA were purchased to support flexible testing arrangements.

#### Antenna

Figure 3.5 below shows an example of an active GPS external antenna. This was purchased as a secondary component to support the testing of a manufactured GPS antenna and associated LNA. Additionally, it would provide a method of testing the data logging system in transit, as the manufactured antenna and LNA would initially have no way of being attached to the external frame of the vehicle. This antenna, being an active antenna, has its LNA included in the package already and was a plug and play solution

to the GPS antenna problem.

Figure 3.5: GPS Antenna - External Active Antenna (Core Electronics 2019)

### LNA

In order to support a manufactured GPS antenna, a LNA had to be either purchased or manufactured. Initially, a PCB solution to the LNA problem was proposed and investigated. Unfortunately, the manufacture proved too difficult with the available tools and a LNA was purchased instead. Figure 3.6 shows the amplifier that was decided upon, the Nooelec Premium SAW filter and LNA. This is a specialty item, designed for use with Nooelec components that support Software Defined Radio (SDR), but could be easily adapted to amplify signals from the manufactured GPS antenna.

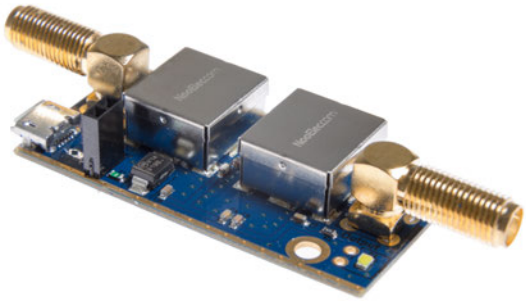


Figure 3.6: Nooelec Premium SAW filter and LNA (Nooelec 2019)

### Receiver

Aw with the accelerometers discussed above, the range of GPS receivers in the maker space are very similar in specification. This made it an easy decision to purchase the Adafruit Ultimate GPS breakout with 10 Hz updates as seen below in Figure 3.7. The

Adafruit libraries are well supported and provide a simpler method of processing and displaying GPS data.


Figure 3.7: Adafruit Ultimate GPS Breakout (Core Electronics 2019)

### 3.7.5 ATWINC1500 Wi-Fi

The ATWINC1500 Wi-Fi breakout was briefly discussed in the Literature review, and can be seen previously in Figure 2.22. This module was chosen to achieve the project wireless objectives and features SPI communication with the micro-controller. The module is 802.11bgn capable and comes with level shifting on the input pins, meaning 3V or 5V logic may be used.

### 3.7.6 OBD2 UART and FTDI Breakout

Last of the hardware is the OBD2 UART module and associated FTDI breakout. The OBD2 UART module can be see in Figure 3.8. It was decided to purchase a serial communication module, rather than a Bluetooth or Wi-Fi kit, as the goals of the project align better with a module that can communicate directly with the micro-controller. The Bluetooth and Wi-Fi kits are relatively cheap compared to this module, but do not provide much flexibility in choosing scan tool software. In addition to the OBD2 UART module, a FTDI breakout was purchased. This allows serial communication from the vehicle to USB using RS-232 protocols.

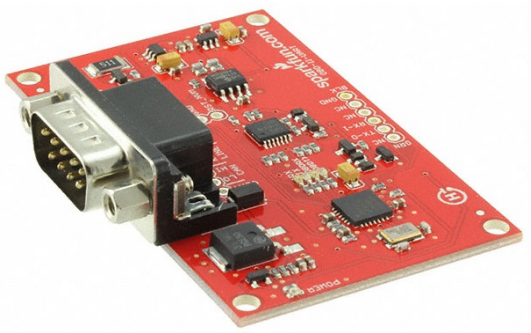


Figure 3.8: Sparkfun OBD2 UART (Core Electronics 2019)

## 3.8 Software

In order to program the hardware listed above and process the data it collected, it was necessary to outline a list of suitable software that could produce results in accordance with the literature review. The basic premise was to find an IDE (Integrated Development Environment) to program the hardware and collect data, then and a suite of tools to delve into and interpret the raw data.

### 3.8.1 MATLAB

The majority of the analysis of the raw data (audio and vibration) was carried out in MATLAB, written in MATLAB scripts. Most functions were derived from the basic package of MATLAB, without the use of tool packages in order to better understand what the functions are doing. These varied from audio processing functions such as the short Fourier Transform to specialized ones for Mahalanobis Distance and covariance.

### 3.8.2 Teensyduino IDE

The micro-controller code was written using the Arduino IDE for Teensy, Teensyduino, and in the embedded C language. A selection of libraries were used, available from a variety of sources, to develop the software necessary to operate the data logger and connected sensors. This included software to process and record the audio in a RAW format, vibration and GPS data in TXT files, SD card recording, Wi-Fi operations, and including the various setup routines and run-time operation of the system.

### 3.8.3 Audacity

Audio data was edited using Audacity, which provided an open source method of importing RAW audio data, and exporting WAV files to be processed within MATLAB. Audacity also provides a spectral analysis tool which was useful in quickly analyzing audio data before beginning the MATLAB process.

### 3.8.4 Google Earth

Raw GPS data was stored in the original TXT format from the Teensy module, and could be easily imported into Google Earth to show location data from the operation of the vehicle.

### 3.8.5 OBD2 scan tool

OBD Auto Doctor was used to process data from the vehicle CAN bus. This was an invaluable scan tool which provided in depth data from the Electronic Control Units in the vehicle.

## 3.9 Data Analysis

In order to properly analyze the raw data it becomes important to select the most efficient and accurate methods to supplement the software chosen above. The first stage is to break down the signal for further analysis. This should be done in accordance with the following steps.

- Create a method of viewing the signal as it operates over time.
- Break down the signal content into the frequency domain, to view significant frequencies.
- Conduct a spectrogram analysis to view how the frequency content changes over time, and its resonant frequencies.

Additionally, it would be useful to have a method for classifying faults. While we may be able to view the frequency content of a signal, in magnitude and over time, this does not tell us if a fault is present. The time, FFT and spectrogram plots will not be an automatic process, and neither can they be compared across faults to provide us with a "simple" answer. We have no baseline of operation. In order to achieve this classification stage, Linear Predictive Coding was deemed to be a logical jump, and has been used to classify engine faults as we saw in the literature review, Section 2.9.2. The steps are broken down as follows:

- Break down the audio recordings into LPC coefficients and average said coefficients. Use the first 75% of the audio file to create a baseline for the faulty/ healthy data.
- Compare the LPC coefficients from the previous step with the LPC coefficients of the entire audio file broken up into 25% segments. Use a Mahalanobis Distance algorithm to show the comparison closeness in a single figure.

## 3.10 Chapter Summary

This chapter has outlined the decisions and methodology that are required to achieve the project objectives described in Section 1.5. It has outlined the hardware, software and Data Analysis procedures that were used to validate the information presented in the literature review and provide a path towards the next chapter, 4 - Design. The Design chapter is based upon the methodology described here and the results from this are presented in Chapter 5, Results.

# Chapter 4

# Design

## 4.1 Chapter Overview

This chapter presents the hardware and software design considerations that were used to develop a working system model for the location based data logger. It shows an overall system architecture which identifies key components of the model, and delves into the work carried out on each module of the system.

## 4.2 Audio

The audio design section encompasses the MAX9814 microphone, Teensy 3.6, SD card and reader, and various passive devices used to interface the microphone with the micro-controller.

### 4.2.1 Hardware

The following explains the hardware design and connection paths for the MAX9814 microphone. Unfortunately, the MAX9814 module has a 1.25V offset when recording audio data, which is not compatible with the Teensy Audio library, which uses a 0.6V offset (0 - 1.2V range). In order to use the microphone module with the Teensy micro-controller, a hardware design solution was required to couple the input to the Teensy.

The circuit seen below in Figure 4.1 is able to effectively AC couple the input to the Teensy, and provides a method of setting the bias to 0.6V so that the recordings are properly offset. Additionally, the capacitors provide filtering from any fluctuations in the ground and power sources.

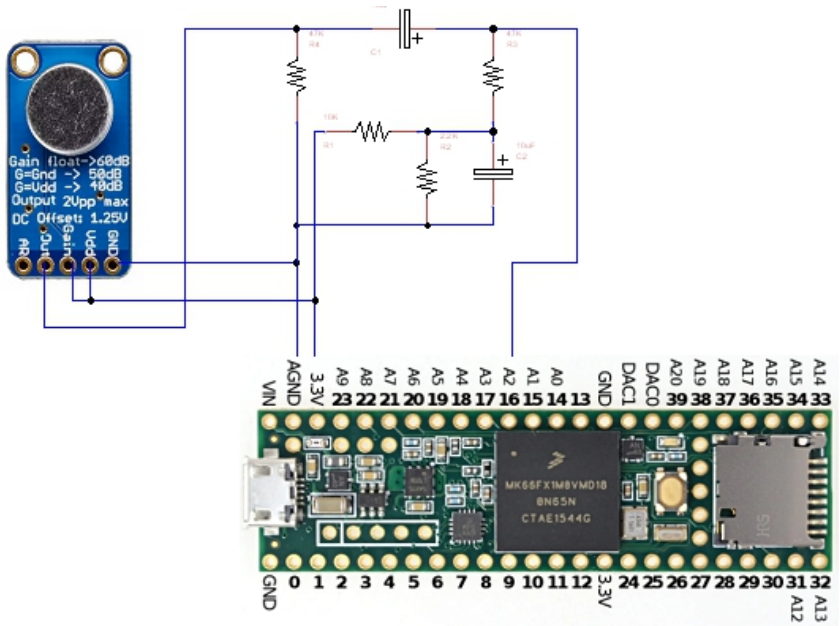



Figure 4.1: MAX9814 Microphone Design

### 4.2.2 Software

The software for the MAX9814 was developed using a number of libraries, such as “Audio.h”, “SD.h” and “Wire.h” within the Teensduino IDE. Additionally, there is Teensy GUI (Graphical User Interface) available online to automatically generate setup code. This is similar to the Node-RED flow based programming tool for the Internet of Things and can be seen below in Figure 4.2

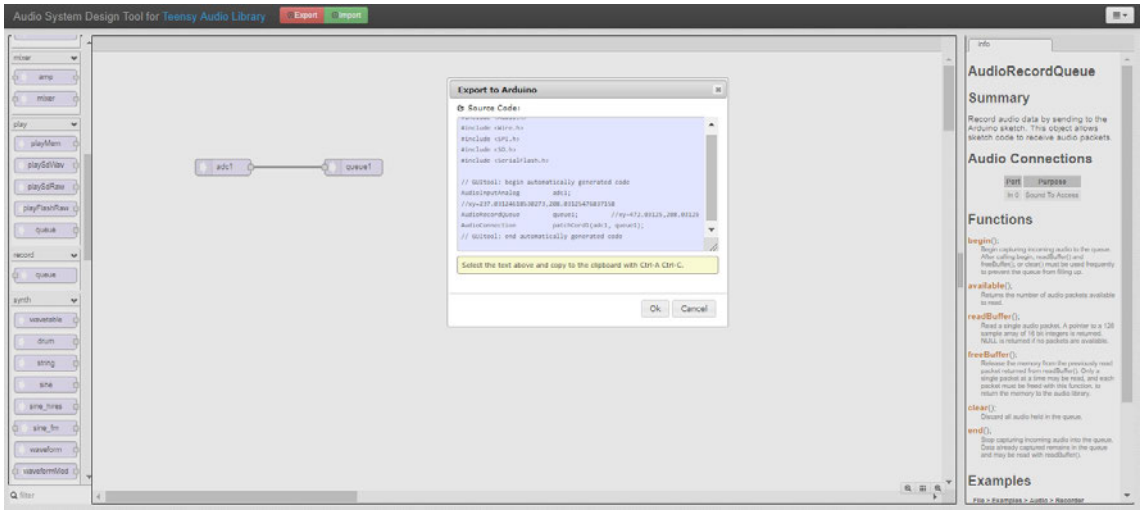



Figure 4.2: Teensy Audio GUI: Flow based programming

Because of the large size of audio data, it was prudent to develop the SD card software in this stage also. Development in the Arduino IDE is described by the following pseudo-code:

- Library #includes, GUItool automatically generated code and pin #definitions
- Variable declarations for modes (i.e. stopped, recording), and various variables, and File names for SD card recordings.
- Setup an appropriate amount of Audio Memory, input pin and start the timer the recorder is based upon.
- Create a filename function, where the old files will not be overwritten and new ones will be created upon Teensy startup.
- Start recording: Open file on the SD card and begin stacking the buffer with audio data.

- Continue Recording: Fetch 2 blocks from the audio library and place into 512 byte buffer. Use internal memory to copy out buffer values then write all 512 bytes to the SD card.
- Stop recording: Pause recording for a very short interval in order for other processes to operate. Finish writing any data left from the audio library into the buffer/ SD card then close the file on the SD card.

## 4.3 Vibration

The vibration design section encompasses the LIS3DH accelerometer, Teensy 3.6 and SD card reader.

### 4.3.1 Hardware

The hardware setup for the LIS3DH accelerometer can be seen below in Figure 4.3. The LIS3DH is capable of communicating in either I2C (Inter-Integrated Circuit) or SPI (Serial Peripheral Interface) modes. I2C requires less wiring, but is generally slower than SPI rates and for this reason it was decided to use SPI communications between the accelerometer and the micro-controller.

The SPI bus uses four logic signals which are as follows: SCLK (Serial Clock), MOSI (Master Output Slave Input - data output from master), MISO (Master Input Slave Output - data output from slave) and SS (Slave Select - often active low when only one slave module). In our design, Pin 14 is used as our SCLK, Pin 10 is the MOSI, Pin 11 is the MISO and Pin 12 is the SS (held low in this case).

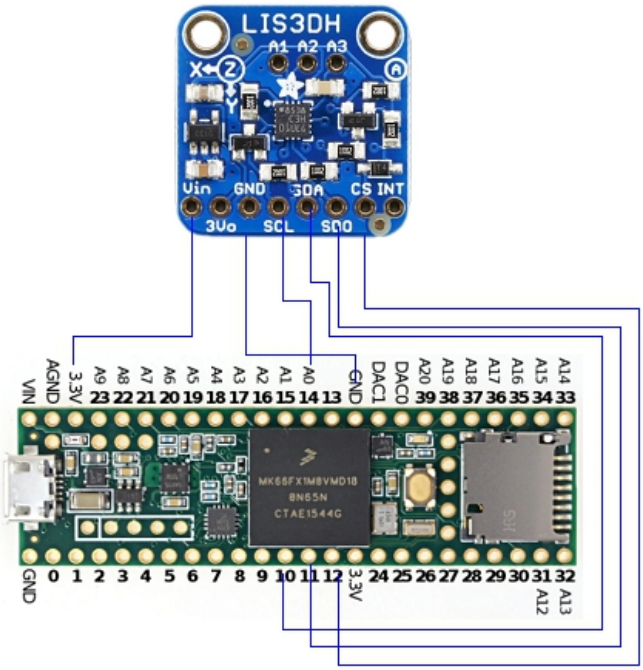



Figure 4.3: LIS3DH accelerometer setup

### 4.3.2 Software

A number of libraries are available to support the LIS3DH accelerometer, such as “Adafruit LIS3DH.h” and “Adafruit Sensor.h”. These libraries provide a layer of code that allows more efficient development time and functionality.

The LIS3DH sensor module provides accelerometer values in a raw value format, and will not provide a “meters per second per second” value. Fortunately, the library “Adafruit Sensor.h” provides a way of normalizing the values. It is able to do this by providing a sensor event to snapshot the data. Data is then able to be extracted from the sensor event by using object-based extraction from the event.

Development of the accelerometer software is described in the following pseudo-code:

- Library #includes and pin #definitions.
- Variable declarations for File names for SD card recordings.
- Set up software SPI communication definitions (CS, MOSI, MISO, CLK).

- Begin setup function calls: Disallow the Teensy to run if accelerometer not found, set up the range (2, 4, 8 or 16G), set up the data rates (5kHz, 1.6kHz, 400 Hz, etc).
- Begin main loop: Call SD writing function every iteration of main loop. Simply records current accelerometer values into TXT file on SD card.

## 4.4 Test Leads

A set of leads had to be manufactured in order to ensure that the results from the sensors within the engine bay could transmit data to the main body of the car and did not suffer from any distortion due to being un-shielded or having multiple connection points. In order to achieve this, lengths of shielded 6-core cable was modified with appropriate breadboard connections on either end as seen in Figure 4.4. Connection points on the sensor package were protected with PTFE tape for protection against moisture and dirt ingress, and electrical tape for structural protection.

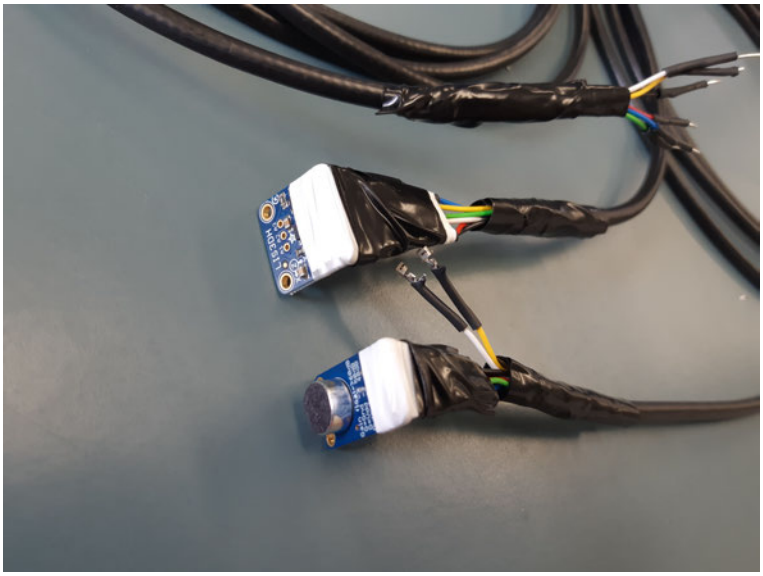


Figure 4.4: Test lead manufacture

## 4.5 GPS

The GPS design section encompasses the GPS receiver, Teensy 3.6, GPS patch antenna and Low Noise amplifier (LNA).

### 4.5.1 GPS Receiver Hardware

The connection hardware for the Adafruit Ultimate GPS receiver can be seen below in Figure 4.5. Connection is rather simple, and only requires four wires: Transmit (TX), receive (RX), Power In (Vin) and Ground.

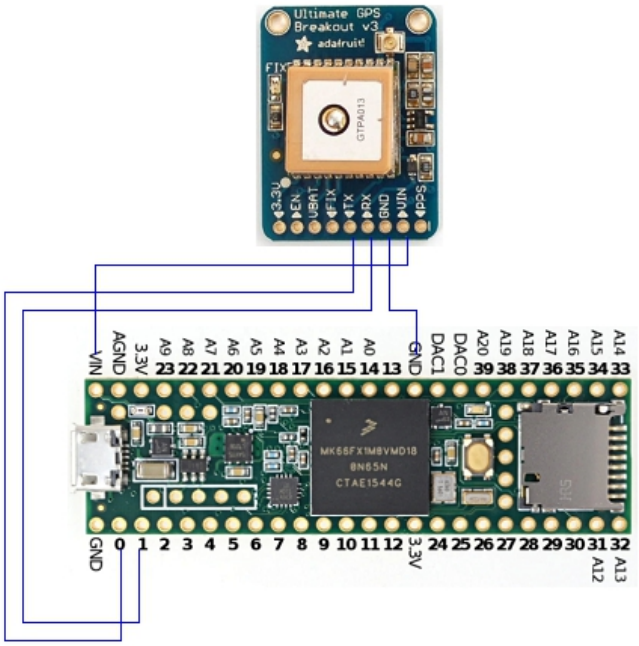



Figure 4.5: Adafruit Ultimate GPS setup

### 4.5.2 Software

This section contains two parts of software which were developed at different times. The general operation GPS code was written to utilize the GPS sensor data during data collection modes. The Geo-location code was written later and provides a geo-located fence around a designated home base in order to allocate when to begin WiFi operations. It is based upon the current GPS position, and incorporates a number of advanced functions to calculate the distance to the home base.

### General GPS operations

As with the other sensor modules, there are a number of libraries available to support GPS modules. In this case, "Adafruit GPS.h" was instrumental in developing the software. Additionally, one of the Teensy support libraries "TimeLib.h" was helpful in converting the GPS NMEA sequences into Australian relevant GMT times.

Development of the GPS software is described in the following pseudo-code:

- Library #includes and pin #definitions.
- Declare variables and filenames to be used for recording purposes.
- Begin setup functions: define NMEA output style and update rates, ask RX to send antenna status, ask for firmware version, start GPS timer
- Create filename convention, new file for each time recording is started, do not overwrite.
- Parse the GPS data appropriately, updating the NMEA receive flag to false if new sequence not received.
- Set the local clock on the micro-controller based on the GPS clock when there is a fix.
- Record the relevant GPS data on the SD card, TXT file format with separating commas.

### Geo-location

In addition to capturing the GPS data, it was also necessary to incorporate a geo-located fence. This allowed the WiFi to have a range of operation, where sensor data would stop recording and the Access Point server would begin to set up. This was achieved by utilizing the previously developed GPS code and manipulating the sensor data. This is best seen in the following psuedo-code:

- Declare variables, including degrees to radian conversion, earth's radius, GPS string (lat/ long) of home-base, and specified range from target.

- If the GPS has a fix of appropriate quality, begin the calculations to determine current position and range to home-base:
  - Convert current GPS position to string format and print to serial monitor.
  - Utilize haversine formula function to determine the distance between current position and home-base. The haversine formula determines the great circle distance between two points on a sphere given their longitudes and latitudes.
  - If the range to target is less than a specified amount, the module has reached home base.
- Once home-base has been reached, enter WiFi operations and set up the Access Point.

### 4.5.3 GPS antenna

Design of the GPS patch antenna was carried out in HFSS (High Frequency Structure Simulator), developed by ANSYS (Analysis Systems). The goal was to design then manufacture a GPS patch antenna with a feed gap as described in the Literature Review. Using Equation 2.2, we can roughly design for a center frequency of 1.575 GHz, which is the L1 band for GPS signals. A key feature of the HFSS software is its ability to implement antenna optimizations by manipulating variable properties, so we only have to approximate the variable lengths within the design environment.

The design environment and variable lengths can be seen in Figure 4.6. The software works by simulating environmental and radiation properties of the antenna and immediate surroundings to generate an approximation of how the antenna will behave in real-time after manufacture.

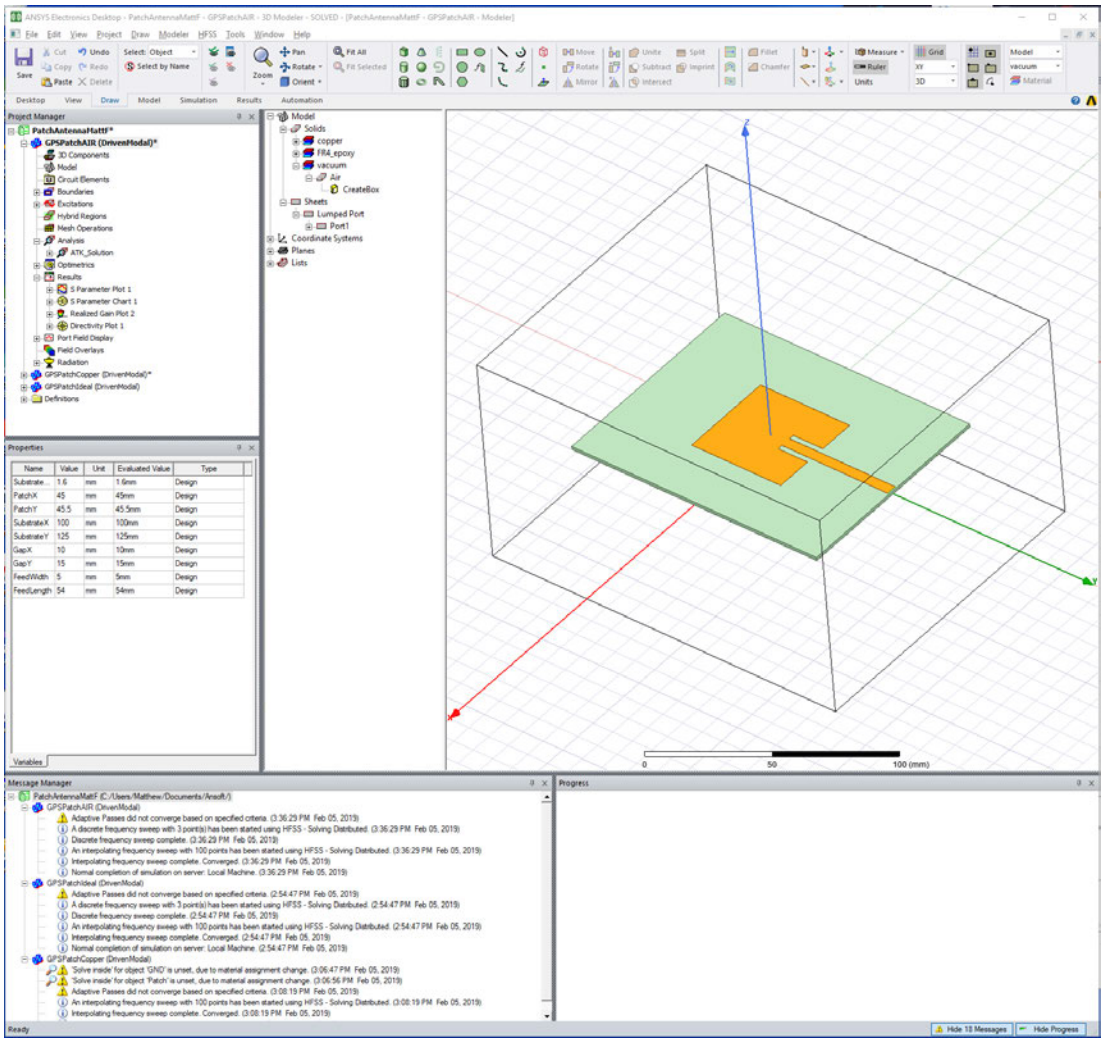


Figure 4.6: HFSS Design environment

The following steps were taken to build the simulation model within HFSS:

- Build substrate of the antenna, using FR4 epoxy 1.6mm
- Design the patch measurements out of the top copper layer, setting the physical measurements as variables so that they are able to be manipulated by the optimization profile function later on.
- Define input port as a lumped port boundary.
- Create air field environment, or radiation to cover the antenna. This has to be larger than 1/4 of a wavelength.
- Run optimization function, and analyze results. Repeat steps above as necessary to reach an optimal antenna.

As we can see below in Figure 4.7, the GPS patch antenna should have the following characteristics after manufacture if all of the design restraints we programmed are the same as in real life. Our smith chart on the left hand side of Figure 4.7 indicates that the impedance matching should be close to ideal at the GPS L1 band frequency. Our S parameter plot on the right hand side of Figure 4.7 shows the reflection coefficient of our antenna, and indicates that it will radiate best at 1.5758 GHz.

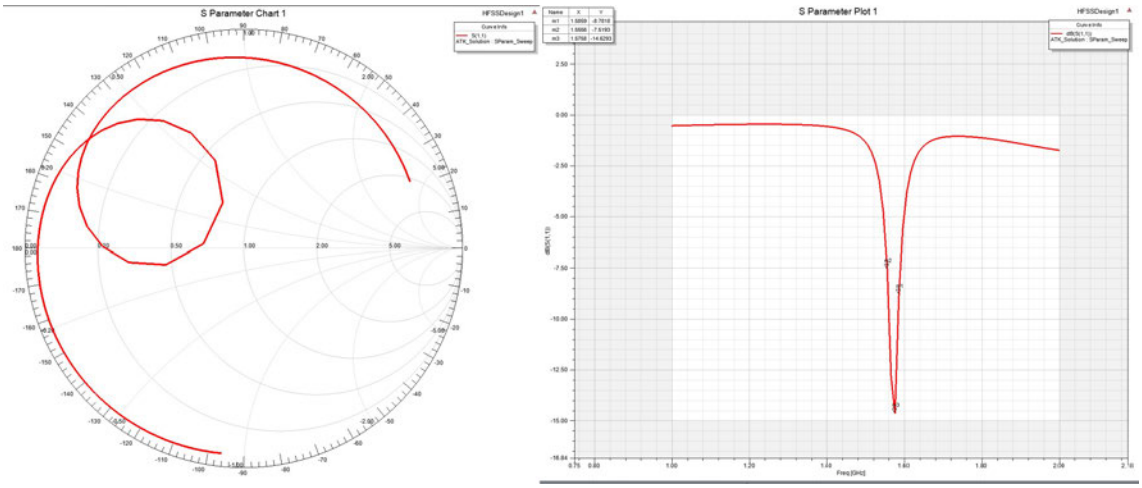



Figure 4.7: GPS Patch antenna results

The next step in the design process is to manufacture the antenna, to provide proof that our conceptual model from HFSS will work. This is done in two stages: To output the HFSS model using a DXF file format, and translate this into a gerber file format for the PCB milling machine to understand. This can be seen below in Figure 4.7. The output from HFSS plots the antenna model into XY coordinates, which are then used by the PCB milling machine to convert into drill sequences and removes the appropriate amount of copper from the top layer of substrate.

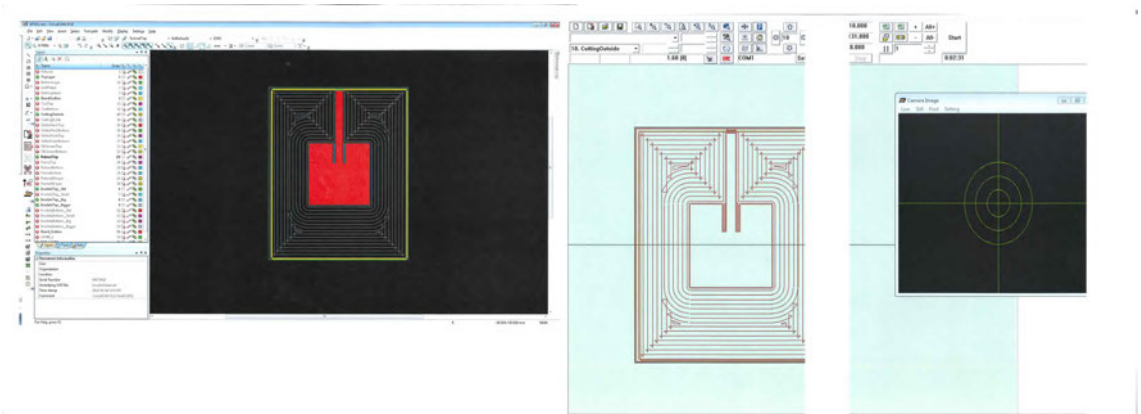


Figure 4.8: GPS Patch antenna manufacture

Following the manufacture of the antenna, the following results were obtained from the VNA (Vector Network Analyzer), as seen in Figure 4.9. The VNA provides an ability to save ".s1p" type files for further analysis, and the results seen in Figure 4.9 were viewed using an online viewer platform. As we can see, the manufactured antenna radiates best at 1.570 GHz, which is slightly off our desired value of 1.575 GHz but still falls within our range of 3% bandwidth. On the lighter side however, we can see from out smith chart that our antenna impedance is almost perfectly matched.

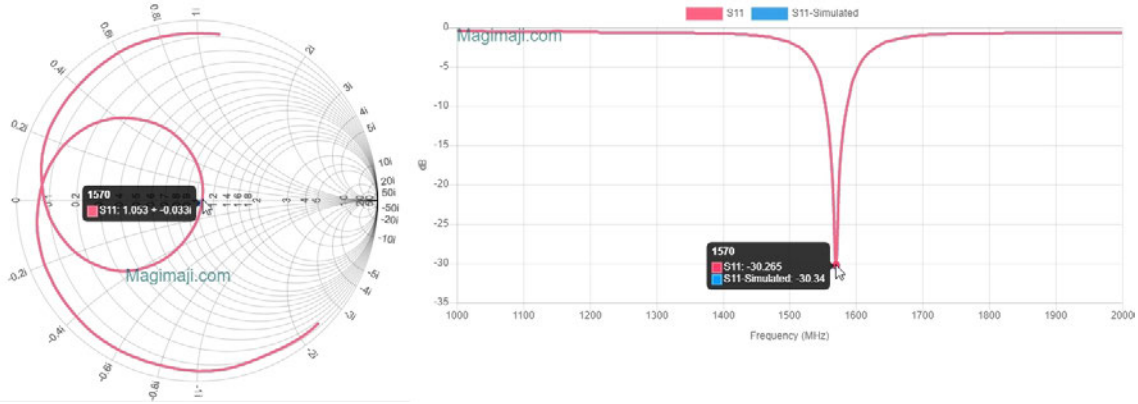



Figure 4.9: GPS Patch antenna measurement results after manufacture

The physical results can be seen below in Figure 4.10. A SMA (SubMiniature version A) connector has been soldered on to facilitate the previous measurement results from the VNA.

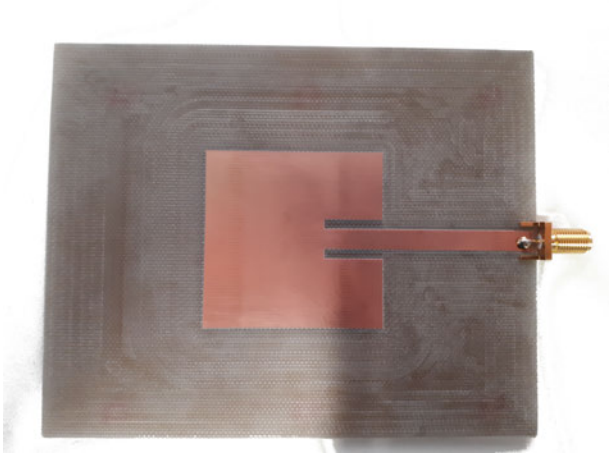


Figure 4.10: GPS Patch antenna manufacture results

### 4.5.4 LNA and connections

In order to incorporate an external passive antenna into the project, as designed above, it was necessary to build an LNA, effectively making the passive antenna active. The ALM-1612 GPS LNA-Filter Front-End Module was sourced as the most appropriate chipset to complete this requirement, and the circuit layout can be seen below in Figure 4.11. The ALM-1612 chipset current requirements were appropriate for the Adafruit GPS receiver, and could be powered through the coaxial cable via an appropriate RF choke.

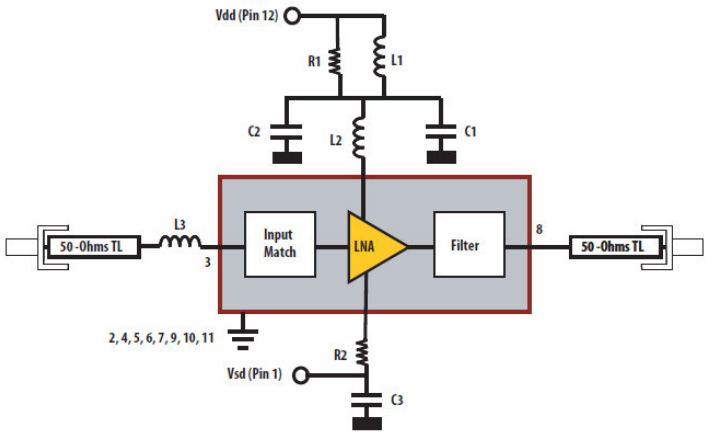



Figure 4.11: ALM-1612 GPS LNA

It was the original intent to manufacture an appropriate PCB board for the ALM-1612 chipset and solder on the appropriate components. The build results can be seen below in Figure 4.12.

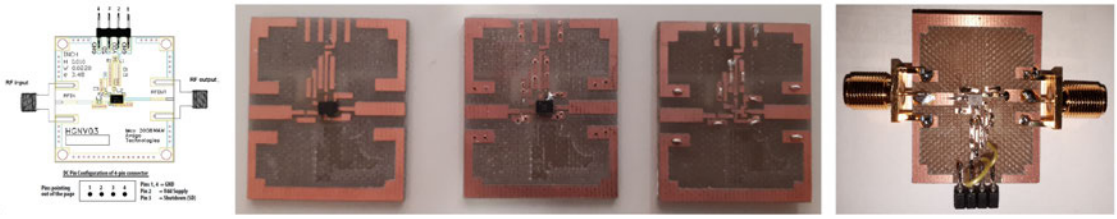


Figure 4.12: GPS LNA Build results

Unfortunately, the manufactured chip did not perform under test and had to be discarded after a period of fault-finding. This led to the development of another line of thinking, utilizing an out of the box solution in the form of the Nooelec Premium SAW filter and LNA that we saw in the Literature Review. This would require additional components such as an external power supply and RF block however, as the Nooelec LNA current

requirements were too high for the Adafruit Ultimate GPS receive and could not be powered through the coaxial cable.

In order to compensate for the inability of the GPS receiver to provide enough current to the Nooelec LNA, a new circuit was developed as seen in Figure 4.13. The Nooelec LNA could be powered externally by either 3-5V worth of battery, or via USB. This provided the LNA with enough current. We also had to trick the GPS receiver into thinking that there was an external antenna attached. This was provided by the resistor and inductor (RF Choke) between the coaxial center line and ground. The resistor was designed to sink enough current (more than 4 mA) from the GPS receiver, and had a value of 470 Ohms. The RF choke was designed to block the high frequency GPS signal, and was in the order of 10 mH. Additionally, now that the GPS receiver had been successfully tricked into thinking there was an external antenna, we had to block the DC output from the GPS receiver onto the central line of the coaxial cable that is usually used to power the GPS LNA. This was achieved using a DC blocking capacitor of value 15 pF, targeted directly at the GPS L1 band 1.575 GHz.

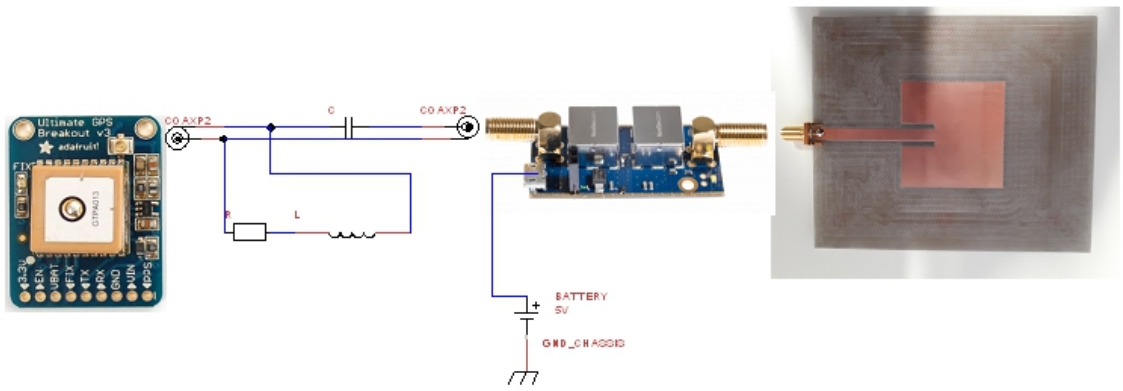



Figure 4.13: GPS Nooelec SAW Filter and LNA

## 4.6 OBD2

The OBD2 design section encompasses the OBD2 UART module, FTDI basic breakout and the scan tools used to monitor and extract vehicle data.

### 4.6.1 Hardware

Connection to the vehicle CAN bus through the OBD2 protocol is a straightforward matter as seen below in Figure 4.14. The OBD2 UART module uses a serial RS232 plug to interface with the OBD2 port located on the vehicle, in the drivers foot well. This data is then processed using the chips discussed in Section 2.4.4. Further to this, the FTDI basic breakout chip is used to convert the serial data to USB for communications with the micro-controller or computer. This allows communications in both directions, where OBD2 codes and data is available from the vehicle, and fault codes can be reset from the micro-controller/ computer.

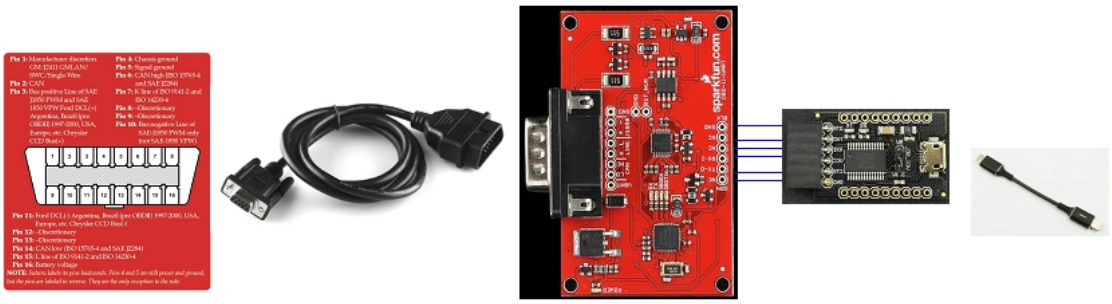


Figure 4.14: OBD2 connection hardware

### 4.6.2 Software

Some of the functionality of the OBD2 scan tool, OBD Auto Doctor, can be seen below in Figure 4.15. It provides an in depth view into the operation of the vehicle under test. Being such a complex analysis tool, it was prudent to develop a list of specific conditions to monitor in conjunction with the audio and vibration sensors in order to effectively fault-find.

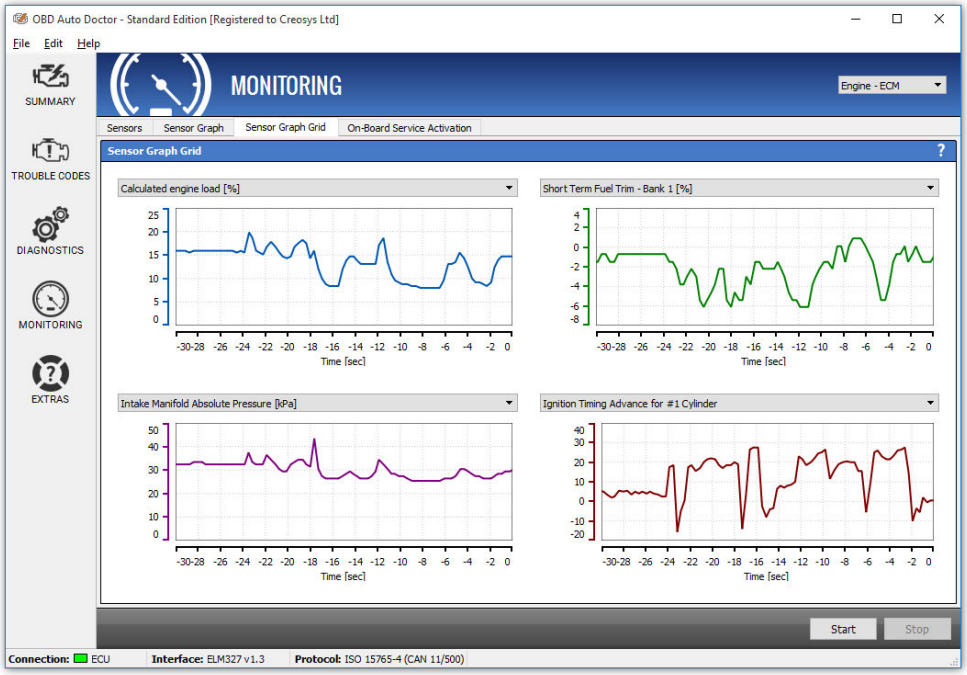



Figure 4.15: OBD Auto Doctor

## 4.7 Wi-Fi

The Wi-Fi section encompasses the ATWINC1500 Adafruit Wi-Fi module and its hardware connections, as well as the software development of the Teensy Access Point server and associated web pages.

### 4.7.1 Hardware

Hardware connection to the Teensy module utilizes SPI communications as alluded to in Figure 4.16 by the MOSI/ MISO pins. SPI mode was described previously in the LIS3DH hardware design section. The ATWINC1500 Wi-Fi module also features an enable pin (tied to High), IRQ (used for interrupt features) and RST (Module reset).

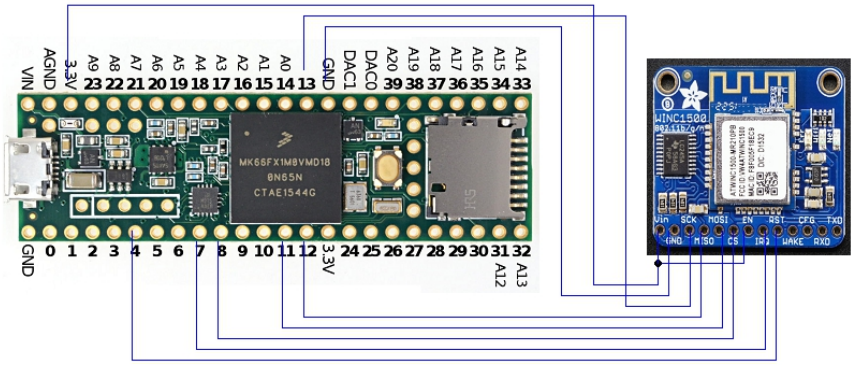



Figure 4.16: ATWINC1500 WiFi Setup

### 4.7.2 Software

The ATWINC1500 module from Adafruit utilizes a library called "WiFi101.h". It is supported by Arduino and is compatible with a number of different WiFi modules and platforms such as the Teensy 3.6 micro-controller.

The software was built in two sections: A server script, and a number of HTML scripts. The server script was written to set up and run the Access Point server and handle WiFi communications via the Teensy and ATWINC1500 module, and was developed in the Arduino IDE.

#### HTML

The web page scripts were developed in notepad and are in HTML format. They were saved on the Teensy SD card for later upload upon request by the client computer. The HTML scripts generate three different web pages to allow a client computer local access to download the data files stored from the Teensy SD card to their respective computer via WiFi communications. The web pages also provide a simple interface for the user to interact with complex data files.

An excerpt from the HTML index script can be seen below in Figure 4.17

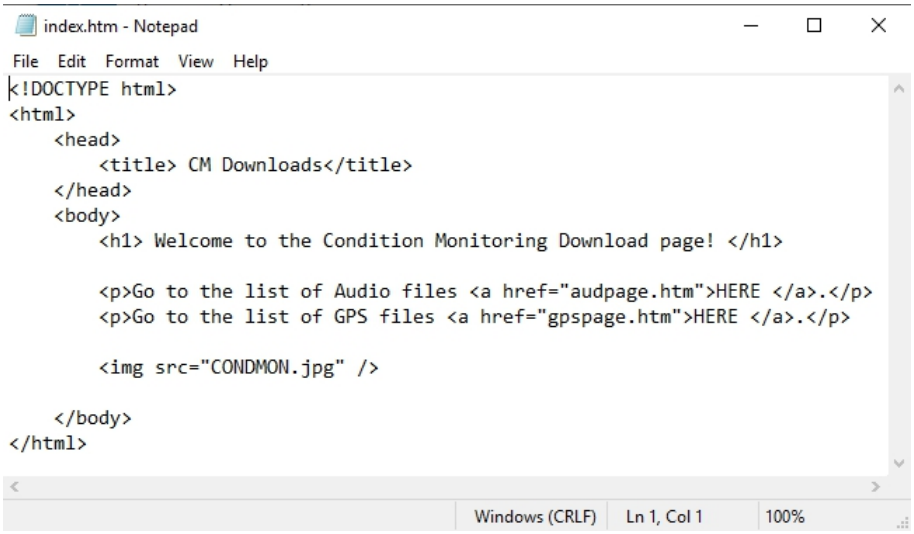



Figure 4.17: HTML script example

### Wi-Fi/ Server

The Wi-Fi and server software was developed in the Teensduino IDE for operation on the Teensy micro-controller. Development of the Wi-Fi/ Server software can be seen in the following pseudo-code:

- Library #includes and #Defines.
- Declare variables, buffer sizes, server ports and Boolean values to be used for Wi-Fi Access Point.
- Wait until the GPS Geo-locate has determined the vehicle/ module is within perimeter of home base, then set up WiFi conditions:
  - Set Wi-Fi pins.
  - Query WiFi module and determine correct hardware connections.
  - Create the open network and listen for incoming client requests.
  - Host the server and display the current Wi-Fi status.
- Once the setup has completed, commence run-time operations:
  - If a client connects, display MAC address of client.
  - Handle client requests using if/ else statements and display the appropriate web pages or upload requested data files.

  - Close the connection once the request is complete and display that the client is disconnected.

- Re-enter the listening phase and await further client instruction.

## 4.8 Data Analysis

Data analysis was primarily undertaken in MATLAB, making use of its signal processing capabilities. Vibration and audio data was entered into the MATLAB programming environment, and analyzed using in-built functions such as the Fast Fourier Transform (FFT) and self-coded functions to perform the Short Fourier Transform, Mahalanobis Distance and Covariance values.

Data analysis was undertaken in two stages. The initial stage, using the FFT function, some windowing and spectrogram analysis, was to extract the frequency components from the data to better understand what kind of effects the faults in an engine will have on the frequency profile of the data. This was performed on both the vibration and audio data collections. Unfortunately, this had some limitations in terms of fault classification, which will be discussed later.

The second stage was to develop the Linear Predictive Coding (LPC) coefficients, then use a statistical measure such as the Mahalanobis Distance to classify different faults.

### 4.8.1 FFT/ Spectrogram Software

MATLAB was the weapon of choice for analyzing the vibration and audio data. In order to extract meaningful information from the data, a number of steps were taken and can be seen in the following pseudo-code:

**FFT:**

- Input the CSV (vibration) or WAV (audio) file and use MATLAB functions to extract vector-based data.

- Determine the size of the data in terms of vector length and time.

- Determine sampling frequency: For WAV file this can be achieved with audioread() function and for CSV can be calculated based on the time step between the first two samples.
- Plot the data over time to get an idea of the waveform, using the actual data and an RMS plot for vibration data.
- Determine the FFT (Fast Fourier Transform) of the data and plot to show frequency analysis of the data.

The following code snippet from MATLAB, seen in Figure 4.18 is useful to visualize this pseudo-code.

```
[name,path] = uigetfile('.wav','Select WAV File to Load, Plot, Compute FFT');
addpath(path);

[Vec_0, Fs_0] = audioread(name);   % load an audio file
Vec_0 = Vec_0(:, 1);    %first channel only if applicable (should only be 1 channel anyway)

% Total number of sound samples
NumOfSamples = length(Vec_0);

sampleDuration = NumOfSamples/Fs_0;
timeDuration = 0:sampleDuration/(NumOfSamples-1):sampleDuration;

%Calculate Basic FFT for separate plot comparisons
FFT_0 = fft(Vec_0 / length(Vec_0));
freq_0 = (0:length(FFT_0)-1)* Fs_0 / length(FFT_0);
FFT_1 = FFT_0;
%remove second half of FFT calculations (rect window) for complex values
FFT_1((length(FFT_1)/2):end) = 0;
```

Figure 4.18: FFT code snippet from MATLAB

Additionally, a spectrogram analysis was developed in order to have a visual representation of the frequency over time. This is seen in the following pseudo-code below and in Figure 4.19

**Spectrogram:**

- Define window length, step size and the number of FFT points to use
- Apply the Hamming window to the length of the window
- Calculate the Short Fourier Transform values of Magnitude, Frequency and Time using original vector values and calculations from FFT function above.

- Amplify and scale the values of Magnitude.
- Use a surf plot to visualize the data.

```
%Compute spectrogram values for plot
window_length = 1024;
step_size = window_length / 4;
no_of_fft_points = 4096;

%apply the Hamming window function
window = hamming_window(window_length);

%function call to do Short FT
[Magnitude, Freq, Time] = short_FT(Vec_0, window, step_size, no_of_fft_points, Fs_0);

%Spectogram plot is Time, Freq, Magnitude in surf plot
Amp_wind = sum(window)/window_length;  %Amplification of the window
Magnitude = abs(Magnitude) / window_length / Amp_wind;  %scaling the amplitude
Magnitude = 20 * log10(Magnitude + 1e-6);   %put magnitude in dB
```

Figure 4.19: FFT code snippet from MATLAB

### 4.8.2 LPC/ Mahalanobis Software

Two scripts were written to develop the LPC coefficients, one to define fault/ normal operation baselines then write to a text file, and the second to input the baselines and compare to an unknown sample using the Mahalanobis function. Given some functional restraints, only the audio data was used for this analysis. The LPC baseline code is explained in the following pseudo-code:

**Create LPC Baseline profiles:**

- Using Audacity, create a WAV file which is the first 75% of the total sample file. This will become the baseline sample. This is done for each type of fault
- Input the WAV (audio) file and use MATLAB functions to extract vector-based data.
- Determine the size of the data in terms of vector length and time, and define frame sizes, order of LPC coefficients.
- In steps of the frame size (for loop), calculate LPC coefficients that approximate each frame. Loop the entire data set to determine a series of LPC coefficients for each frame.

- Take the average of the LPC coefficients to create a baseline over the whole sample.
- Remove NaN entries and write LPC coefficients to an external TXT file. This is now the baseline.

A section of the MATLAB code from the Baseline profiles script can be seen below in Figure 4.20. This is useful to visualize how the LPC coefficients are calculated frame by frame.

```
% %define the matrices before execution
parameter_matrix_full = [];
%
% %take it in steps of the frame size
for N = 1:frame_size:length_input
% %break the loop if we reach the end of the input data
    if (length_input - N) < frame_size
        break;
    end
    %Calculate LPC coefficents for current frame
    parameter_matrix = lpc(sampled_data(N:N+frame_size-1), nth_order);
    %Add coefficients to a matrix
    parameter_matrix_full = [parameter_matrix_full, parameter_matrix'];

end
```

Figure 4.20: LPC code snippet from MATLAB

The Mahalanobis Distance Pseudo-code can be seen below:

**Mahalanobis Distance:**

- Using Audacity, create WAV files which are the last 25% of the original sample file, as well as other samples from different recordings of the same fault. These will become the test samples.
- Input baseline LPC baseline files and place in vector variables within MATLAB.
- Input the unknown test WAV (audio) file and use MATLAB functions to extract vector-based data from the samples.
- Determine the size of the sample in terms of vector length and time, and define frame sizes, order of LPC coefficients.
- In steps of the frame size (for loop), calculate LPC coefficients for the unknown sample that approximate each frame. Loop the entire data set to determine a series

of LPC coefficients for each frame. Remove NaN entries.

- Using the Mahalanobis function, compare the baseline LPC coefficients (average) with the unknown sample LPC coefficients (for each frame). The Mahalanobis function also uses a covariance function internally in order to arrive at a single value of Mahalanobis Distance. This reflects how far away the unknown sample is from the baseline data. Smaller values indicate that the unknown sample is closer to the baseline.

A section of the MATLAB code from the Mahalanobis script can be seen below in Figure 4.21. This is useful to visualize how the Mahalanobis Distance is calculated, where the inputs are the baseline and the unknown data.

```
function MahalDist = MahalanobisDistance(sample, Baseline)
    [rowX, columnX] = size(sample);
    [rowY, columnY] = size(Baseline);

    sumN = rowX + rowY;

    if(columnX ~= columnY)
        disp('Columns in X must be same as in Y')
    else
        xDifference = mean(sample, 1) - mean(Baseline, 1);
        covarianceX = Covariance(sample);
        covarianceY = Covariance(Baseline);
        calcCov = (rowX / sumN * covarianceX + rowY / sumN * covarianceY);
        MahalDist = sqrt(xDifference * inv(calcCov) * xDifference');
    end
end

function covarianceValue = Covariance(X)
    [rows, ~] = size(X);
    shuffleCov = X -repmat(mean(X),rows,1);
    covarianceValue = shuffleCov' * shuffleCov / rows;
end
```

Figure 4.21: Mahalanobis code snippet from MATLAB

## 4.9 System architecture

The total architecture can be seen below in Figure 4.22. This is an approximation of how the system works together and shows some of the various attributes of each module as well as the communication protocols each module uses to communicate with the microcontroller.

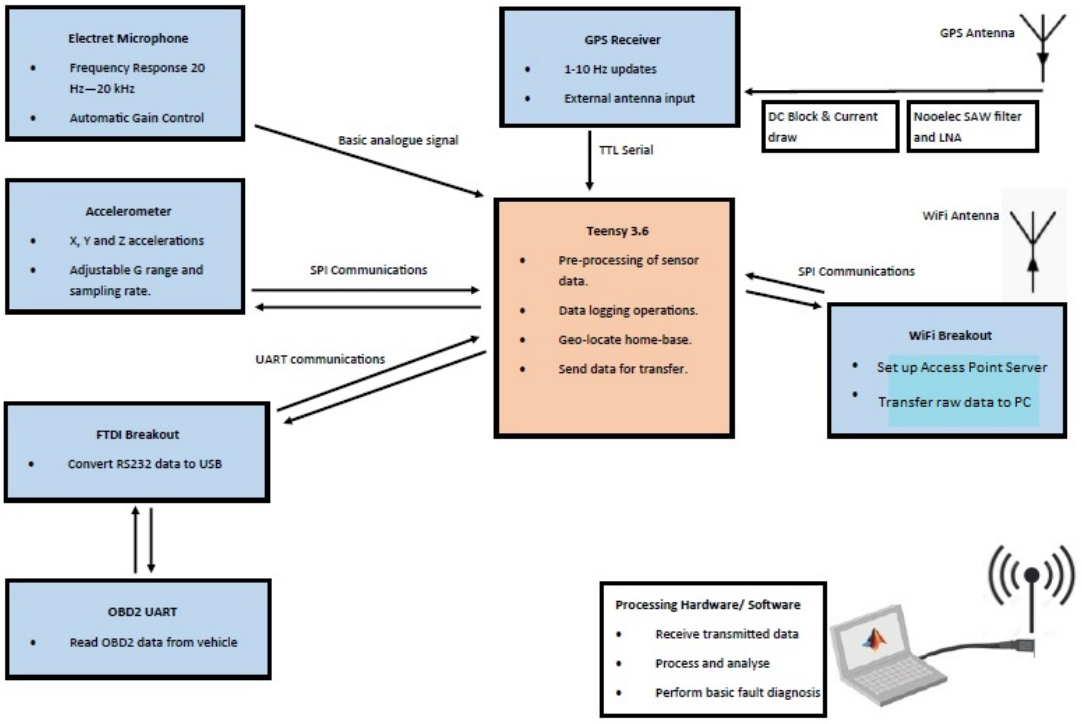



Figure 4.22: System Architecture

## 4.10 An Integrated System

In order to integrate a number of different sensor and breakout modules together, it was necessary to map out the connections to the Teensy board. For example, there were a number of sensors that communicate using SPI protocols, and the Teensy GPIO pins are not all compatible. Additionally, the GPS receiver had to have separate Ground and PWR connections, as the module drew too much power.

Once all the connections had been made, it was necessary to amalgamate the software. This came hand in hand with the hardware connections and transferring the pin allocations was a simple matter. The difficult part of this process was timing. The audio

recording component takes up a large part of the processing space. The challenge was to leave out enough time for every other sensor to perform a reading, record to the SD card, and bounce back to the audio with enough time left for the gap not to be noticed. This was partly solved by the large processing capability of the Teensy micro-controller, where the micro-controller could remain recording and buffer audio, leaving only small gaps in the order of micro-seconds where the additional sensor recordings from the accelerometer and GPS to the SD card could take place.

Integrating the Wi-Fi module was a complex hardware matter, as the new module also required SPI communications. The Teensy 3.6 board does provide three SPI connections. Since two SPI connection sets were already used in the previous construction, the third one was investigated. After some pad soldering and software tests it was found that the "WiFi101.h" library was not compatible with the third set of SPI pins. This led to the disconnection of the accelerometer sensor, and connection of the Wi-Fi module to the SPI0 set. The Wi-Fi/ server software was mostly run after sensor data collection and did not affect the operation of the previously developed code.

It may be possible to integrate the accelerometer using the pad mounts for SPI2, or alternatively edit the "WiFi101.h" library for use with SPI2, but this was not investigated due to time constraints.

The system was able to be fully integrated with the exception of the accelerometer sensor, and operated as advertised. It was able to collect sensor data without any significant lapses, then provide the data to a client via Wi-Fi communications upon return to a geo-located home base.

## 4.11 Chapter Summary

This chapter has provided the design components and overall architecture of the system that was developed from the methodology in Chapter 3. An in-depth description of how each component was designed and integrated into the system was formulated, along with any stumbles along the way.

# Chapter 5

# Results and Discussion

## 5.1 Chapter Overview

This chapter presents and discusses the results of the testing phases. Results are presented in a chronological manner as the work progressed. Components were tested individually to ensure they would meet the needs of the project, then amalgamated and used in a series of trials on two different vehicles. Additionally, a number of recordings of engine faults were sourced from a database online for use in the MATLAB analysis stages.

## 5.2 Preliminary Component Testing

A range of individual tests were carried out to verify the operation of each component, and its suitability in the project. This would help determine overall project compliance when each component could be seen to satisfy individual objectives.

### 5.2.1 LIS3DH Accelerometer

To validate the work carried out in the literature review, the accelerometer had to be tested using some local vibration source that could simulate an approximate frequency. To do this, it was decided to mount the accelerometer to a mobile phone with a vibration application installed, where the vibration profile could be edited. We know that the vibration frequency of the average mobile phone is approximately 150 Hz (Yim, Myung & Lee 2017), so all we had to do was measure this to prove that the accelerometer could pick up vibrations and could be used in testing on the engine profile. We also know that the fundamental frequency of the engine, using Equation 2.1, will be less than 75 Hz. As we can see in Figure 5.1, the accelerometer under test was able to successfully record the 150 Hz vibration coming from the mobile phone.

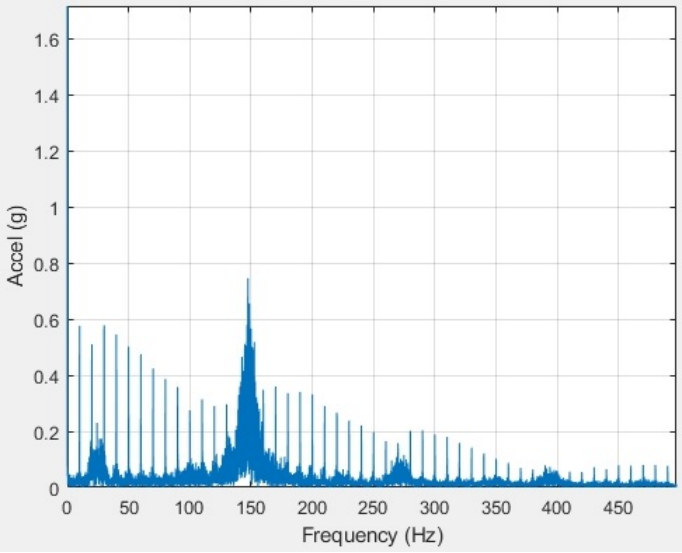



Figure 5.1: Phone Vibration testing

### 5.2.2 MAX9814 Microphone

A similar test was conducted on the microphone, using a 1 kHz test tone from a mobile phone. This test was conducted to ensure that the microphone was capable of picking up the sound without clipping, and with appropriate gain. The recording can be seen below in Figure 5.2, which was imported as a RAW file and edited with headers and informational data to become a WAV file.

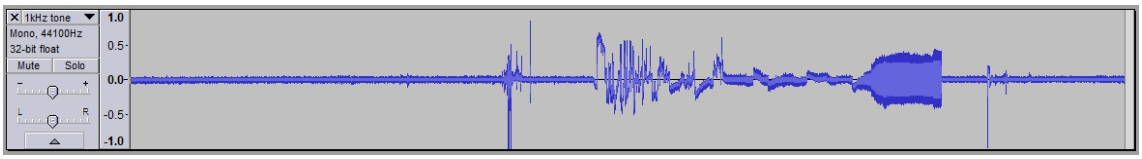



Figure 5.2: 1 kHz test recording

Figure 5.3 below shows the results of the audio analysis in Audacity, clearly showing a 1 kHz tone being picked up by the microphone.

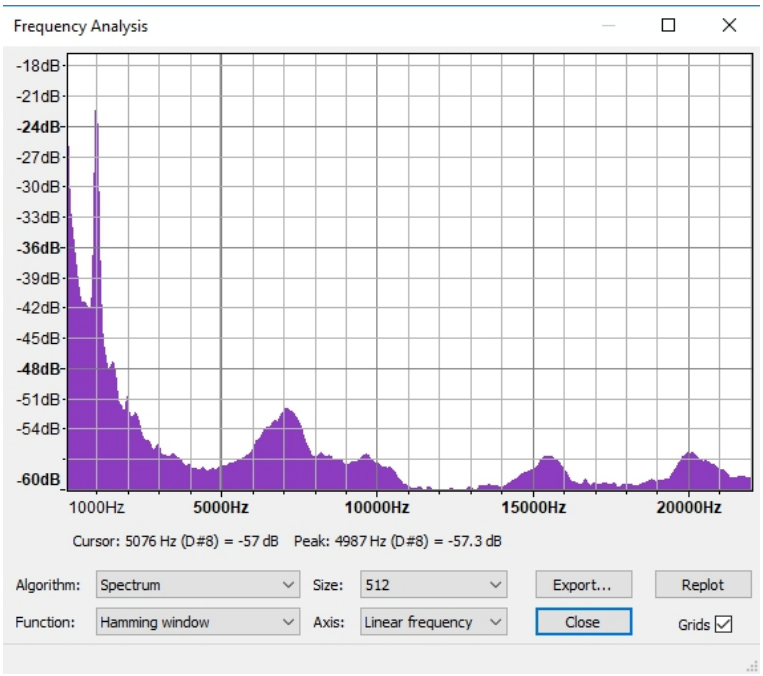



Figure 5.3: 1 kHz test recording

### 5.2.3 Data Storage and file formats

The sensor data has a need to be stored temporarily on the data logger until it can be transferred on return to home base for further processing and analysis. The local storage on the Teensy 3.6 module is insufficient for the masses of data being recorded, and the SD card was utilized. This was tested in conjunction with the audio and vibration tests carried out in the preceding sections.

In order to record to the SD card, an SD library was utilized under the auspices of the Arduino IDE. This library allowed recording of audio, vibration and GPS data to two different types of files, RAW and TXT, in the FAT32 file format. These files provided a low memory option of recording data and could then be quickly removed from the data logger for processing.

### 5.2.4 GPS sub-system

GPS testing was carried out in a number of steps, ensuring compliance of each component before proceeding. Initially, the manufactured antenna had to be tested for resonance at the correct frequency. This was carried out using the simulation software HFSS (High-Frequency Structure Simulator) created by ANSYS (Analysis System), for a proof of concept on the measurements, and then on a Vector Network Analyzer once manufacture was completed. From the design section, we had seen that the reflected power at the desired frequency was approximately -15dB in the HFSS simulation environment. The manufactured antenna was then tested using a Vector Network Analyzer to ensure this figure was reflected in the physical world.

Once the GPS antenna had been manufactured, the next process was to amplify the signal and successfully record a GPS reading to the SD card, through the GPS receiver. Using the purchased Nooelec SAW filter and LNA (Low Noise Amplifier) module, the GPS receiver, some coaxial cabling, various SMA connectors, a USB power supply, a locally manufactured DC block/ current sink and a u.FL to SMA connector, the GPS signal was successfully received through the external antenna and data was stored on the SD card. The testing setup yielded the results that can be seen below in Figure 5.4.

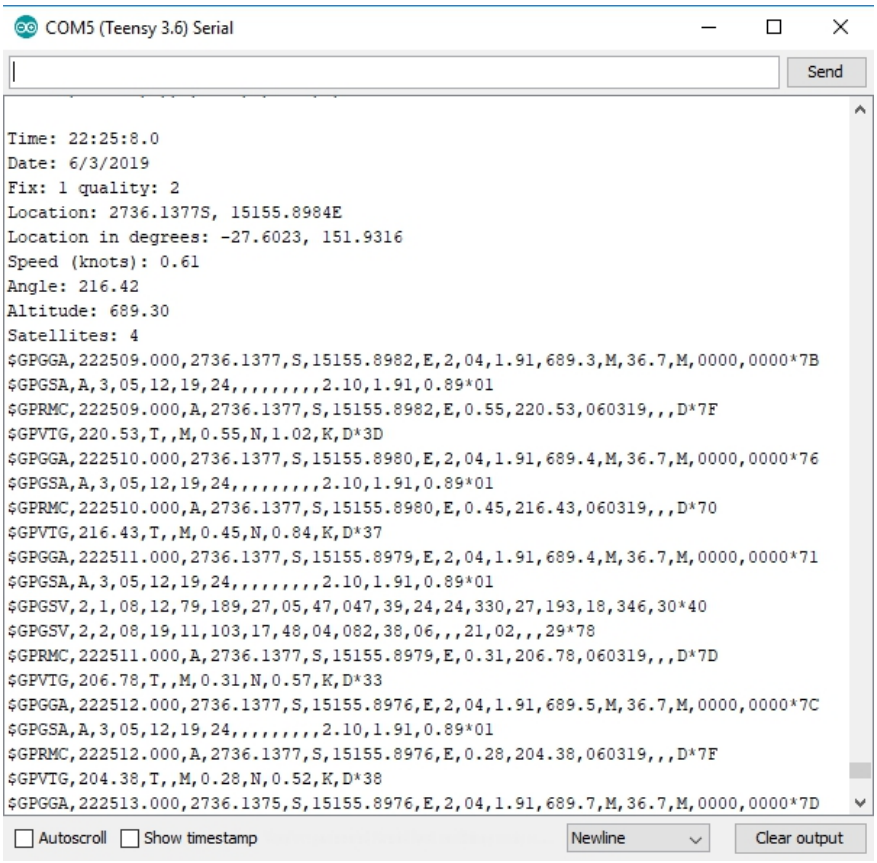



Figure 5.4: GPS test results

### 5.2.5 Wi-Fi and Access Point server

The output of the serial monitor connected to the micro-controller is shown below in Figure 5.5. This represents some of the stages of the Wi-Fi and server test setup. The GPS sub-system is polled until a suitable fix is made. The distance to home base is then calculated based upon an embedded location and the current location of the module. When the module is located inside range of home base, it finishes recording data from the sensor array and begins to setup the server. The server contains a label (which is seen from the Wi-Fi tab on most computers), an address to the web page, and the strength of the signal. Finally, the serial monitor shows when an external device is connected, namely the PC used to download the data.

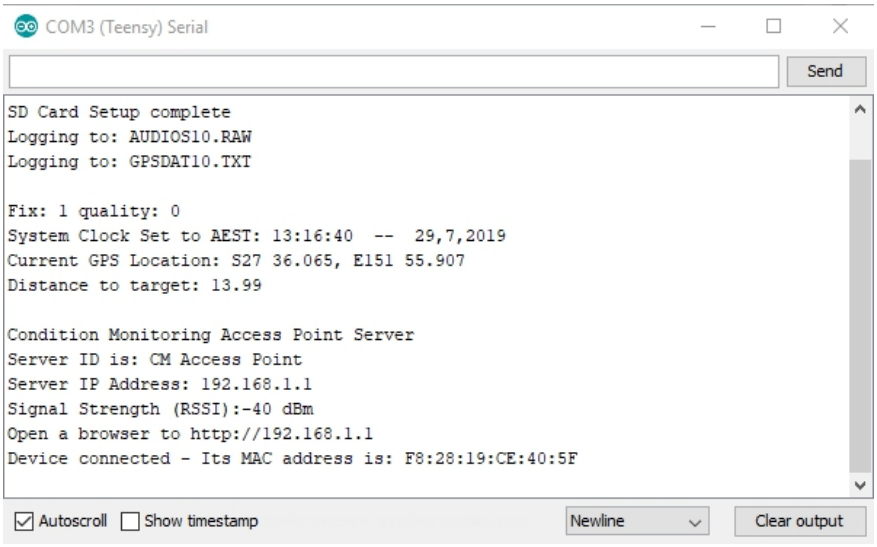



Figure 5.5: Preliminary Wi-Fi results

### 5.2.6 Final Configuration

The final setup can be seen below in Figure 5.6. The testing leads were omitted for the photograph, and the OBD2 connector is not shown connected to a vehicle. In practice, the accelerometer and microphone would be connected to ends of the test leads rather than on the breadboard as seen in the figure. Additionally, the GPS antenna would be mounted to the top of the vehicle. This does however approximate the configuration that was used to assess the suitability of the module in recording data and transferring it to the home computer.

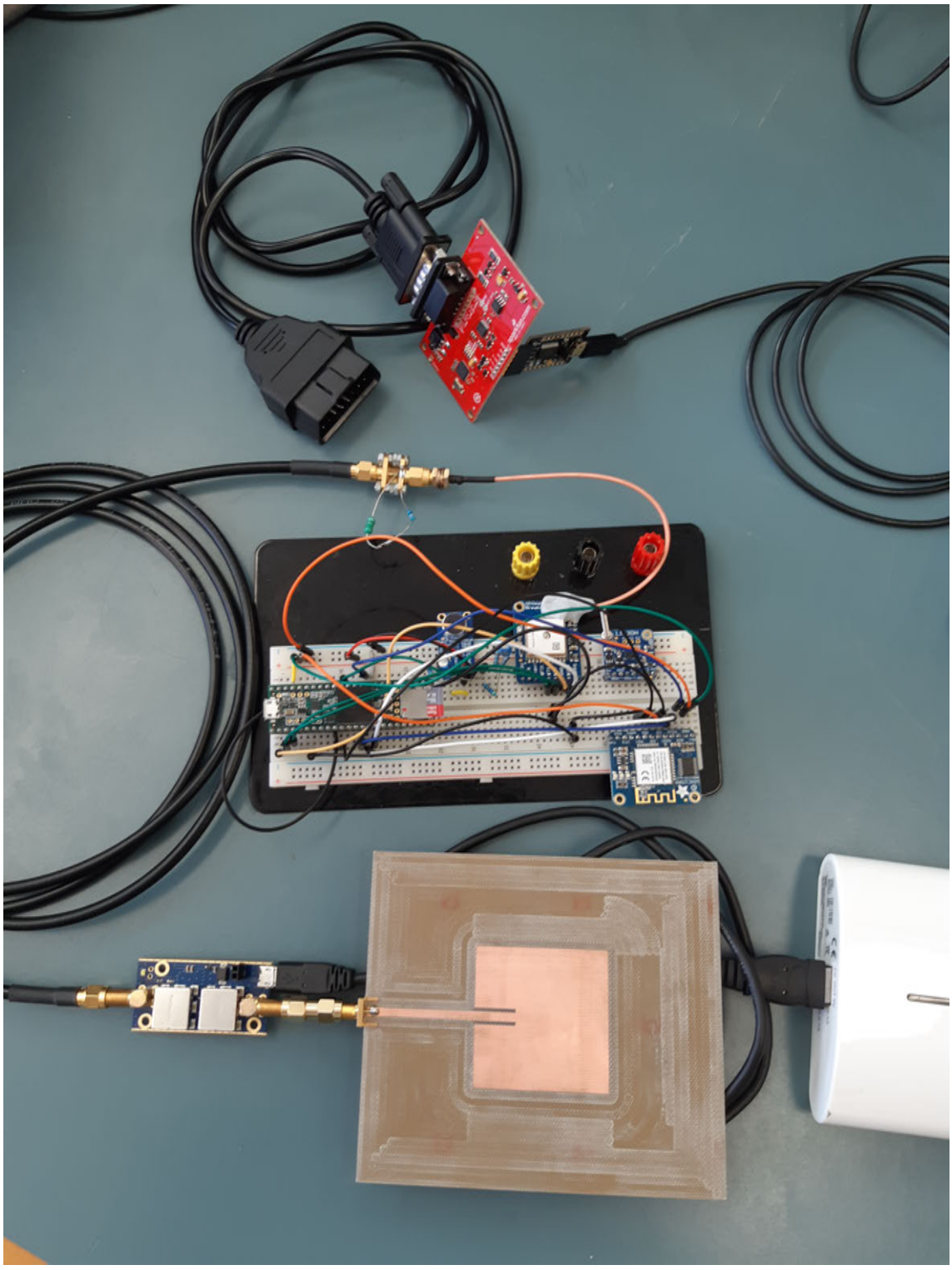


Figure 5.6: Final configuration

## 5.3 Trial Summaries

The results in the following sections were garnered during a series of trials on two different vehicles. There were three main trials in which results were gathered. In addition to the trials, further research was carried out on engine recordings taken from an online database. It should be noted at this point that not all results are presented as the amount of content would overwhelm the reader. It is the intent of the following structure that the most relevant results be presented. The trials can be summarized as follows:

- **Trial #1 Subaru Forester 21/03/2019:** This trial was a proof of operation test for the GPS and OBD2 data predominantly. It also incorporated an initial test on the accelerometer and microphone to ensure correct setup and position. It was discovered after this initial test that the accelerometer settings within the code were quite incorrect, and the placement of the microphone (in addition to gain settings) was leading to some significant clipping.

- **Trial #2 Subaru Forester 22/04/2019:** This trial was to confirm the correct operation of the accelerometer and microphone, including placement. OBD2 and GPS data was gathered again also.

- **Trial #3 Hyundai Getz 20/05/2019:** Trial #3 was the first trial with introduced faults. After correct operation was confirmed in the preceding trials, a baseline recording of operation was conducted on the Getz, and then two different faults were introduced into the Getz and recordings taken for each. The first fault introduced was a disconnected ignition plug, and the second "fault" was to change the fuel from normal unleaded to premium. It must be noted here that these tests were conducted with the engine at a constant idle speed, as opposed to the previous tests which show driving speeds and up to 2000 RPM.

- **Trial #4 Online database:** This was not a trial as such. It gathered a series of engine recordings from an online database with and without faults (both on the same engines) and attempted to conduct an accurate analysis using the MATLAB tools.

## 5.4 GPS Results

After the individual GPS system was tested, as described in the preceding section, it was necessary to ensure operation and accuracy on a moving vehicle.

### 5.4.1 Trial #1 Subaru Forester

As can be seen in Figure 5.7 below, the manufactured antenna was able to track and log the vehicle position accurately for the duration of the trial. The raw GPS data was stored in a TXT file, and contained latitude, longitude and time/ date information. This raw data was then imported into Google Earth using a simple import tool and displayed as below.

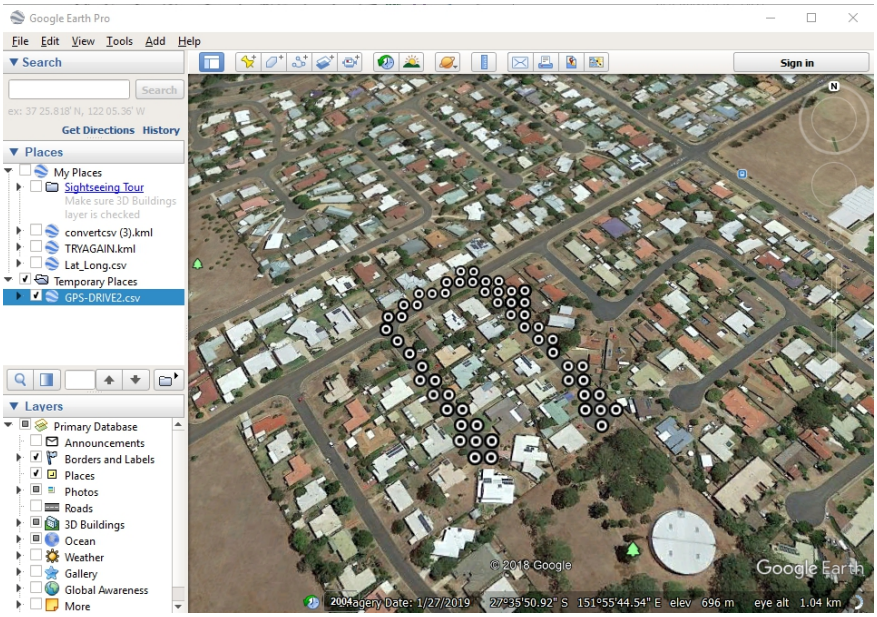



Figure 5.7: GPS plot from Trial #1

As we saw earlier in the individual test section, the GPS is also capable of recording many other conditions such as altitude, velocity, angle of direction, number of satellites used in the fix, and the quality of the signal received. Additionally, these conditions can be overlaid onto the GPS dot points if further trip information is required.

Another area that was investigated was the ability of the accelerometer to record driver conditions, rather than engine vibration. This used a much lower sampling rate and the results can be seen below in Figure 5.8. Each individual point on the map can contain

a plethora of information for the user. The figure below is showing accelerometer values (X,Y,Z), the time this point was recorded, and the latitude/ longitude at this point.

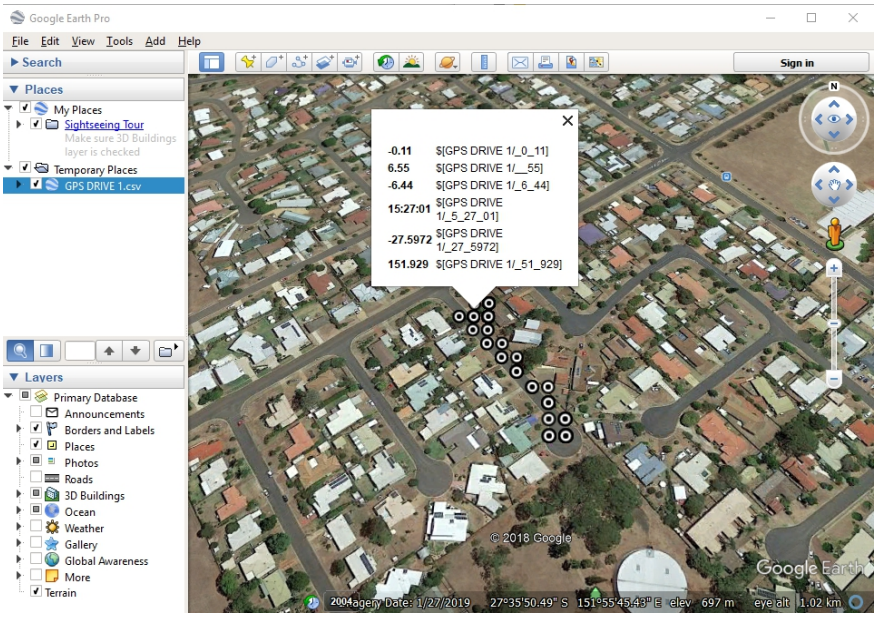



Figure 5.8: GPS plot from Trial #1

### 5.4.2 Geo-location

The geo-located home base can be seen below in Figure 5.9. The coordinates of home base and a fixed range value are embedded in the Teensy code. The location of the module is then compared to the location of home base and a distance to target calculated. If the distance to target is less than the range value, the module has entered the "fence" and server set up is commenced.

A minor issue for the GPS fence was the accuracy of the module. The tests were carried out in the same location as the embedded coordinates. A 30m fence around the home base worked every time for a stationary position of the module, but not so for 20m and 10m, showing that the module is more than 20m inaccurate occasionally.

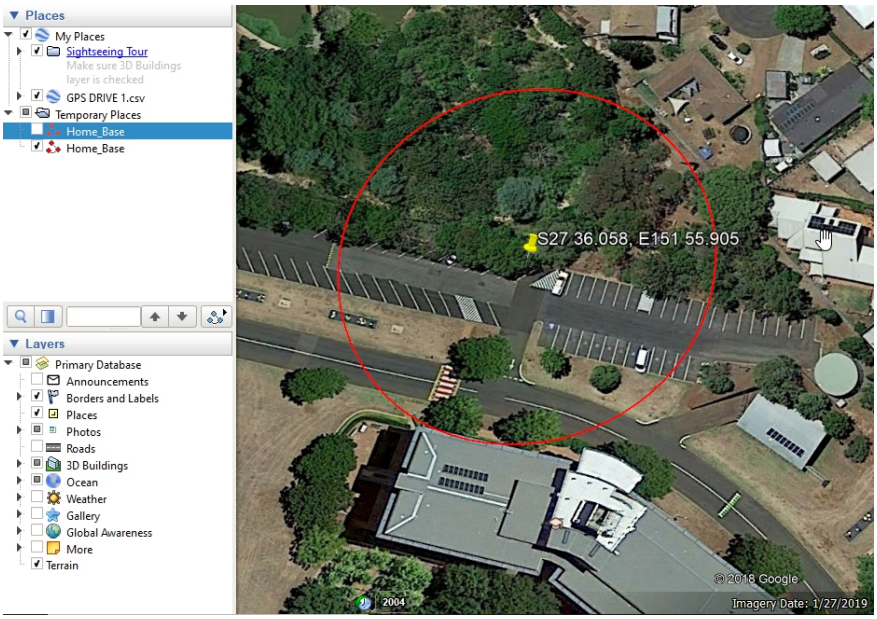



Figure 5.9: 30m Geo-located fence

## 5.5 OBD2 Results

The ODB2 system saw its first test in the trial, as it was unable to be tested individually without the vehicle.

### 5.5.1 Trial #1 Subaru Forester

The OBD2 results from Trial #1 can be seen below in Figure 5.10. This information was extracted from the CAN bus on the vehicle by the OBD2 protocol and the connection hardware. A scan tool software called OBD Auto Doctor then takes this information and displays the chosen aspects of the car operation on screen. We can see from the figure below that the drive was approximately 60 seconds, intake air and engine coolant temperatures were fairly regular, and the vehicle speed and RPM correspond to each other as we would expect. There is much more information available through the OBD2 protocol, this is simply an example to prove the operation of the hardware.

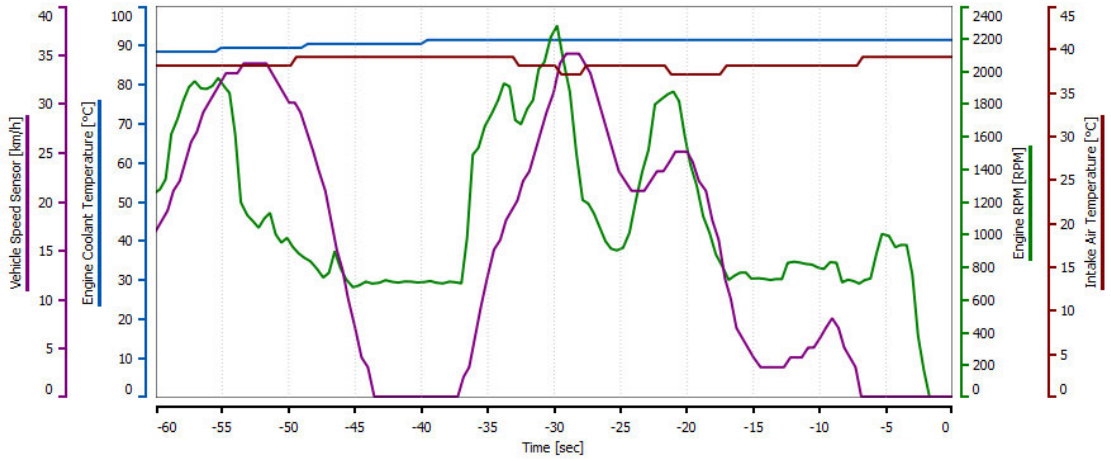



Figure 5.10: OBD2 plot from Trial #1

## 5.6 Wi-Fi and Server results

One of the web pages hosted by the micro-controller can be seen in Figure 5.11. This shows the index or home page for the site. Two other pages are also linked, which provide a list of files available for download from the Teensy SD card. These files will be either GPS, Audio or accelerometer data in a raw format. Once the files are downloaded by the client computer they are able to be processed using Audacity and then MATLAB to diagnose a fault.

The speed at which the client computer was able to access the data was quite slow unfortunately. The image that can be seen in the figure below had a size of approximately 40 kB, but would load in the "dial-up" fashion of the 90's internet. Audio and GPS files of a few seconds length were able to be downloaded with no significant delay, but this would increase with file sizes.

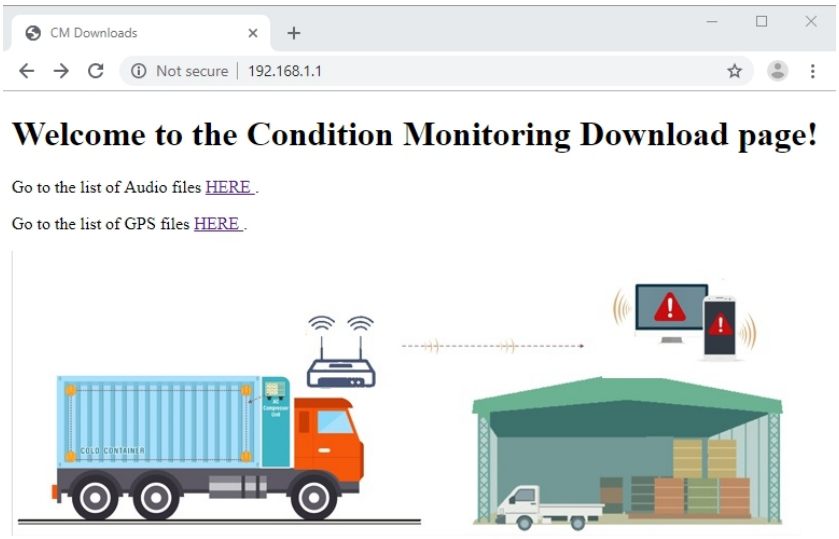



Figure 5.11: Condition Monitoring Web page

An example of a client request can be seen below in Figure 5.12. In this case, the client has requested the most recent audio file for download which is indicated at the top of the figure by "GET /Audio1 HTTP/1.1". The host IP address is also shown, as well as a swathe of HTTP information in the form of requests and communication information. User-Agent for example is a request string that allows peers to identify application type, operation system and software version.

The range of the Wi-Fi was not tested effectively, and tests were limited to the length of the laboratory room. There were no issues with connectivity in the scope of the tests that were undertaken.

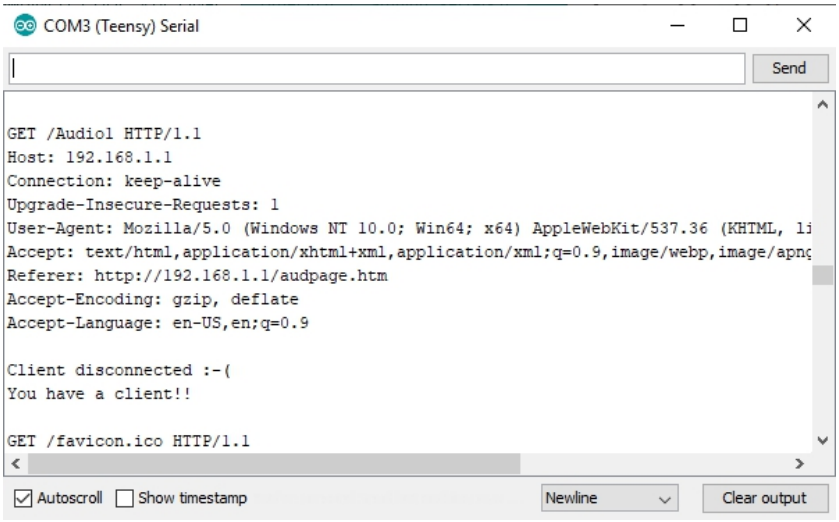



Figure 5.12: Wi-Fi communications

## 5.7 Vibration Results

This section presents the vibration data that was collected from the accelerometer for the duration of the Trials. The raw data has been manipulated for presentation and proof of the data collection operation. It is in the form of vibration profiles in the time and frequency domains.

### 5.7.1 FFT and Spectrogram Analysis

FFT and spectrogram plots were developed in this section to better analyze and visualize the raw data from the engine vibration profile. These were conducted on a healthy engine (Trial #2) and on an engine with two known faults introduced (Trial #3).

#### Trial #2 Subaru Forester

This trial involved a stationary test, increasing the RPM of the engine from idle to approximately 2000 RPM and measuring with an accelerometer sampling at 5 kHz. This can be seen in Figure 5.13. The graph shows the data being plotted over time, and clearly shows the starting of the engine, the idle period, and the increased vibration from RPM increase. Due to the sampling frequency used in by the accelerometer (5 kHz), the resolution of the measurements is quite rough.

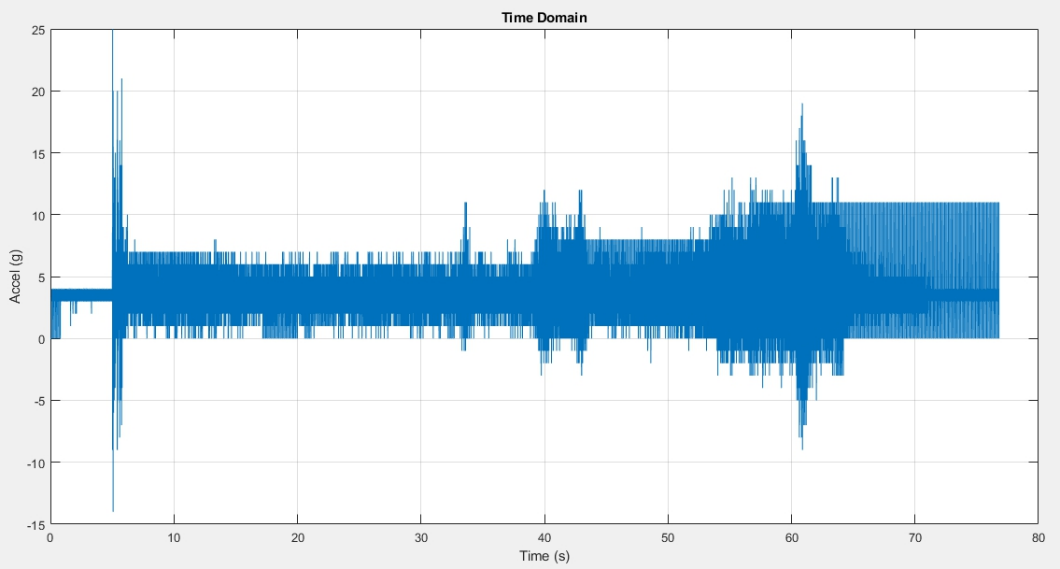



Figure 5.13: Time analysis in MATLAB using Vibration data

It was also useful to calculate a spectrogram analysis using a hamming window and short Fourier Transform Functions, then see this on a plot as shown in Figure 5.14. The spectrogram plot provides us with a visual representation of how the spectrum of frequencies of a signal vary over time. This also gives us an idea of where the resonant frequencies are located. Again, we can see the engine start period, and where the RPM was increased, including where the resonance from these events lies. This should be compared to the time plot above in Figure 5.15 and similarities noted.

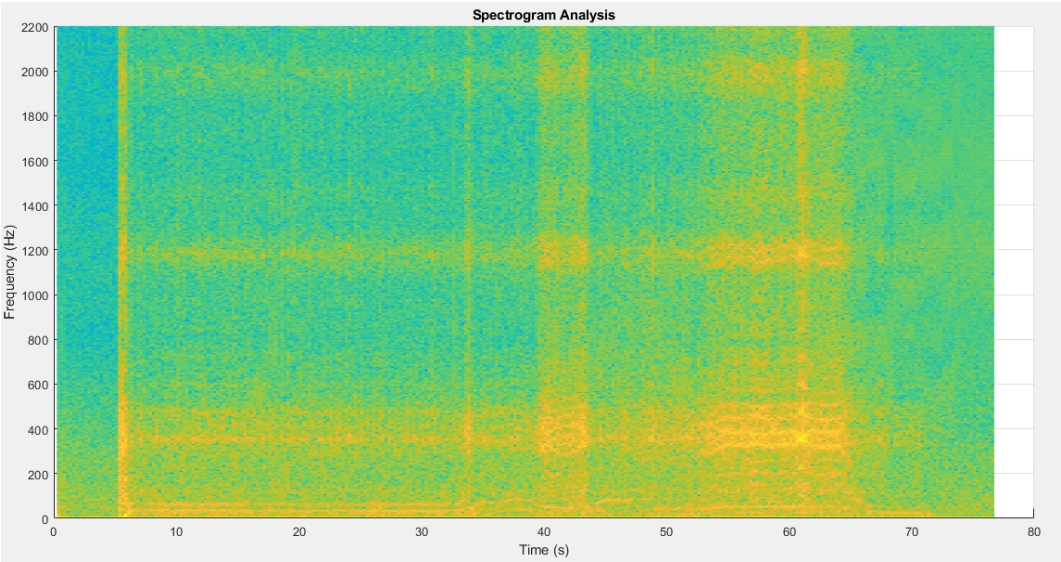



Figure 5.14: Spectrogram analysis in MATLAB using Vibration data

Lastly, we can extract frequency information from the FFT plot in Figure 5.15. There is a clear spike at approximately 32 Hz, which corresponds with the frequency of the engine as calculated from Equation 2.1, and to the large amount of time the engine spent at an idle RPM of 1000. The time spent at 2000 RPM is not as clearly identifiable (which would be approximately double the frequency - 64 Hz).

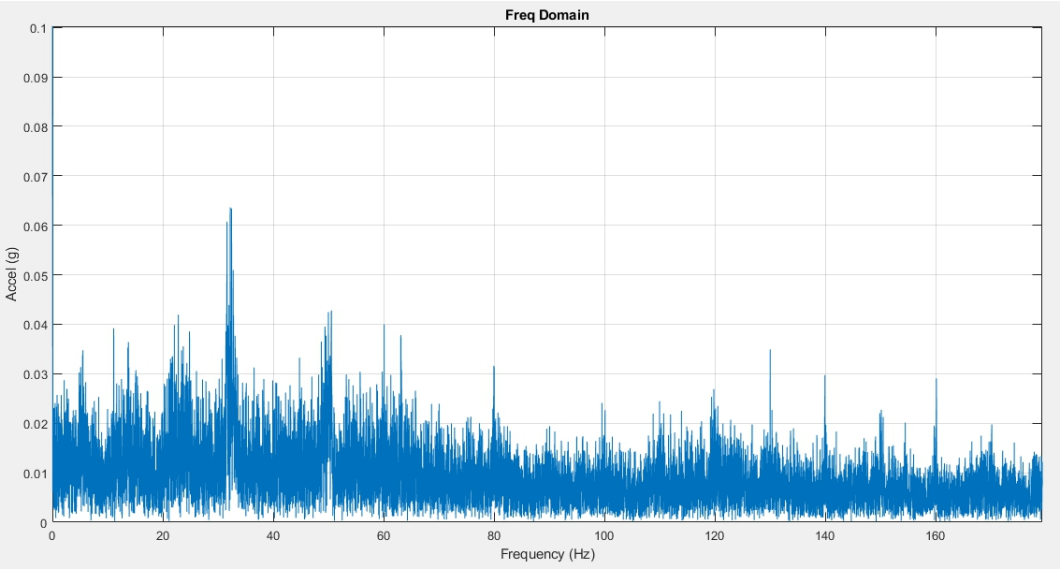



Figure 5.15: FFT analysis in MATLAB using Vibration data

### Trial #3 Hyundai Getz

The results seen in Figure 5.16 below show an overlay of the baseline/ healthy data with the fault data. It can be clearly seen that the magnitude of the fault data is much larger. This test was run at idling, and the ignition and the peaks at beginning/ end are the ignition/ turn-off points. The particular fault introduced in the Hyundai Getz, and seen in the figure, was a disconnected ignition plug as described in Section 5.3.

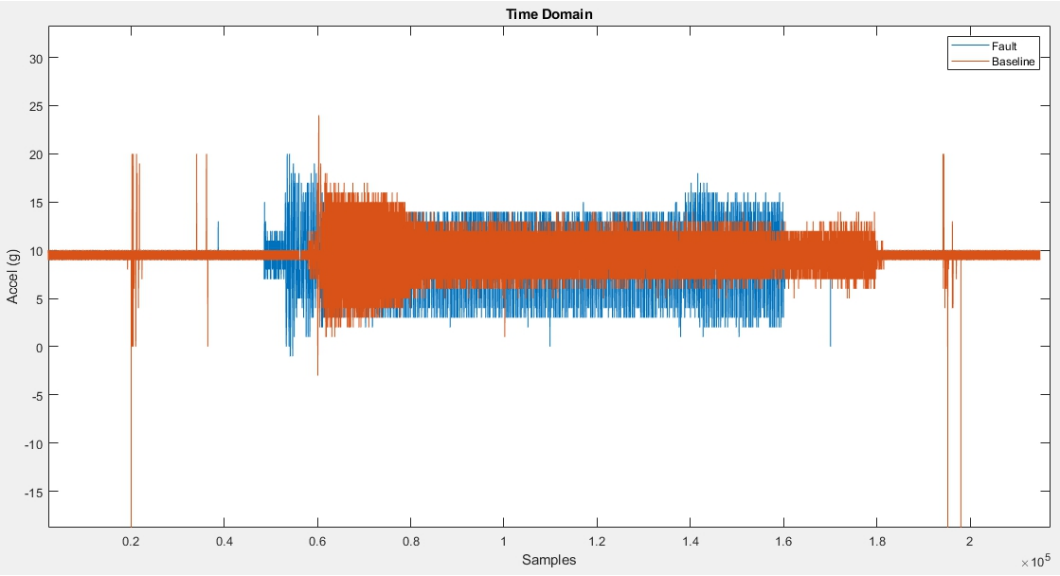



Figure 5.16: Vibration overlay for fault data on Hyundai Getz

In addition to knowing the magnitude of the response in the vibration profile, it is also useful to see the magnitude of particular frequencies. Although hard to see in Figure 5.17 when compared to the fault data, the baseline has a maximum at around 32-34 Hz, which is the frequency of the engine at idle. At approximately 16-17 Hz (and surrounding resonant frequencies), we can see a huge spike in the fault data. It should be noted at this stage that 17 Hz is one half of the maximum seen at the baseline data.

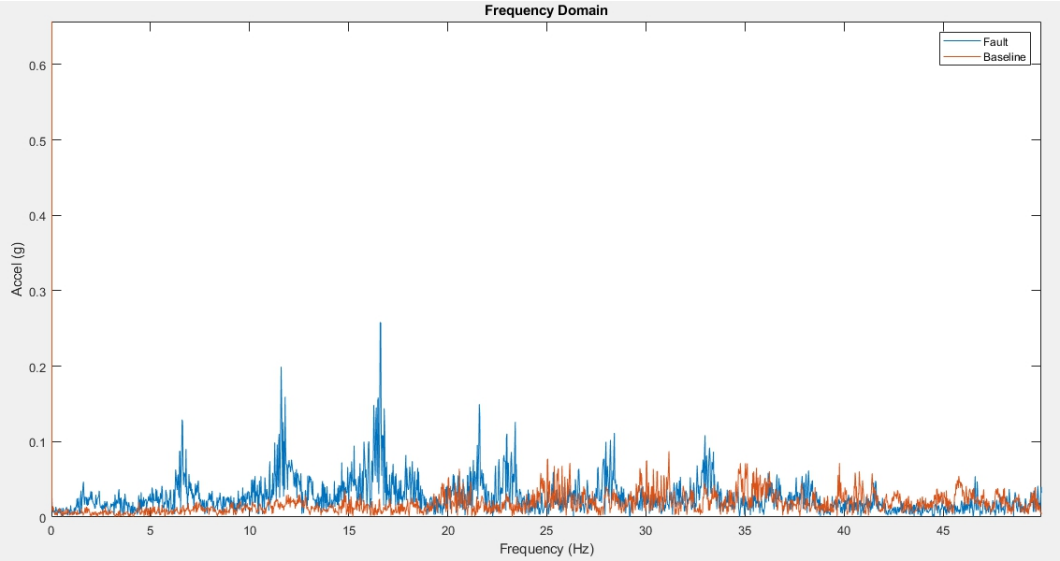



Figure 5.17: FFT domain overlay for fault data on Hyundai Getz

## 5.8 Audio Results

This section presents the audio data that was collected from the microphone for the duration of the Trials. The raw data has been manipulated for presentation and proof of the data collection operation. It is in the form of audio profiles in time and frequency domains.

### 5.8.1 FFT and Spectrogram Analysis

FFT and spectrogram plots were developed in this section to better analyze and visualize the raw data from the engine audio profile. These were conducted on a healthy engine (Trial #2) and on an engine with two known faults introduced (Trial #3).

### Trial #2 Subaru Forester

As alluded to in Section 5.7.1, this trial involved increasing the revolutions of the engine and recording its condition. This part of the trial was conducted concurrently to the accelerometer recording and was done with a microphone sampling at 44.1 kHz. Below in Figure 5.18 we can see a similar profile emerging, where the ignition or engine start is observed, and increasing revolutions noted. As compared to the time profile from the accelerometer graph, the audio data has a much higher resolution.

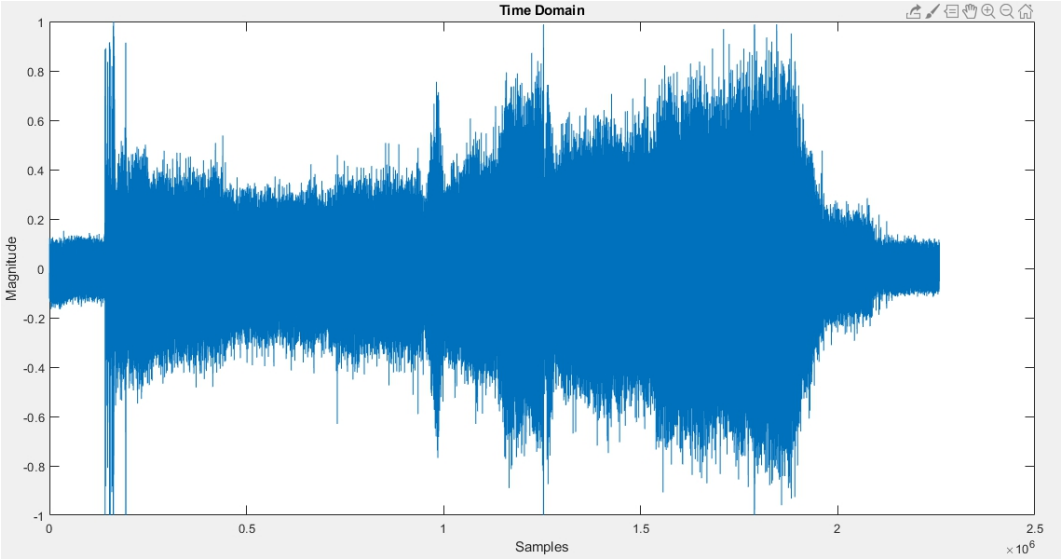



Figure 5.18: Time domain plot of Forester audio recording

Further to this, we can see the frequency content of the signal in Figure 5.19. When compared to the content in the vibration FFT seen previously in Figure 5.15, the specific engine states (1000 and 2000 RPM) are not as obvious. What we can see however is a strong indication that the engine was operating approximately between 35 Hz and 70 Hz, corresponding to 1050 and 2100 RPM. From this, it could be inferred that in order to better diagnose fault frequencies it will be necessary to record at one speed of the engine only.

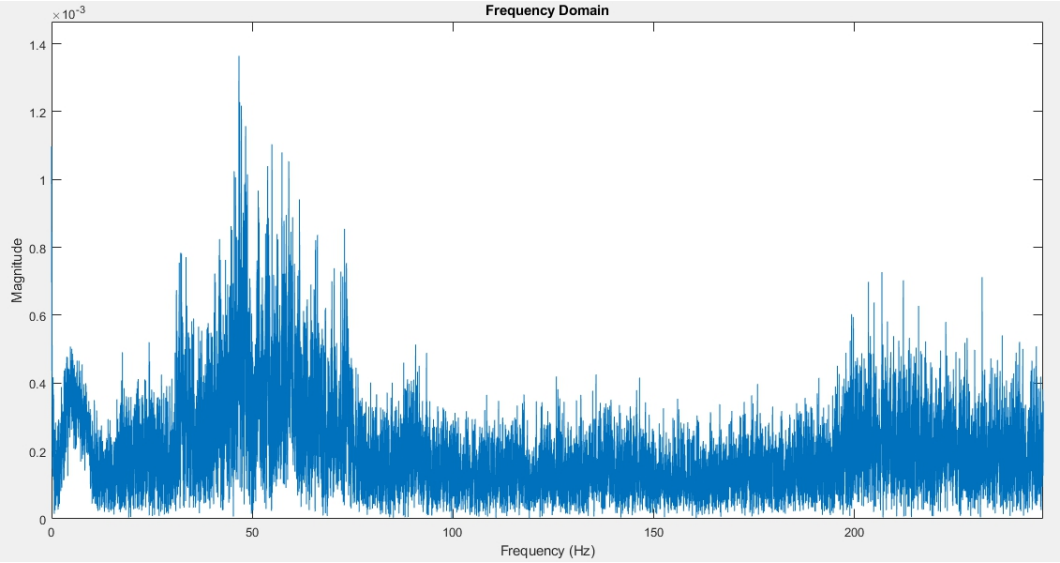



Figure 5.19: Frequency domain plot of Forester audio recording

Lastly, in Figure 5.20, we can see a visual representation of the frequency content over time. Again, it is important to note how we can see resonant frequency content over time and in relation to the original event. This should be compared to Figure 5.18, as with the vibration content above for similarities in the signature.

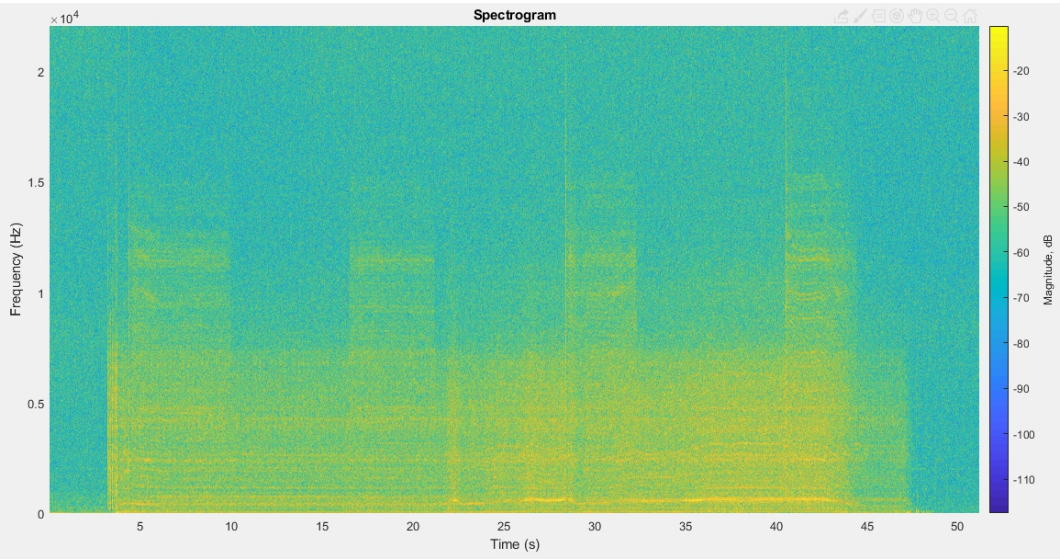



Figure 5.20: Spectrogram plot of Forester audio recording

### Trial #3 Hyundai Getz

This part of the trial was conducted concurrently to the vibration test seen in Figure 5.16. We can see that the results are somewhat similar, but the audio recording seen below in Figure 5.21 does not show as much of an increase in magnitude between the fault and baseline data when compared to the vibration data.

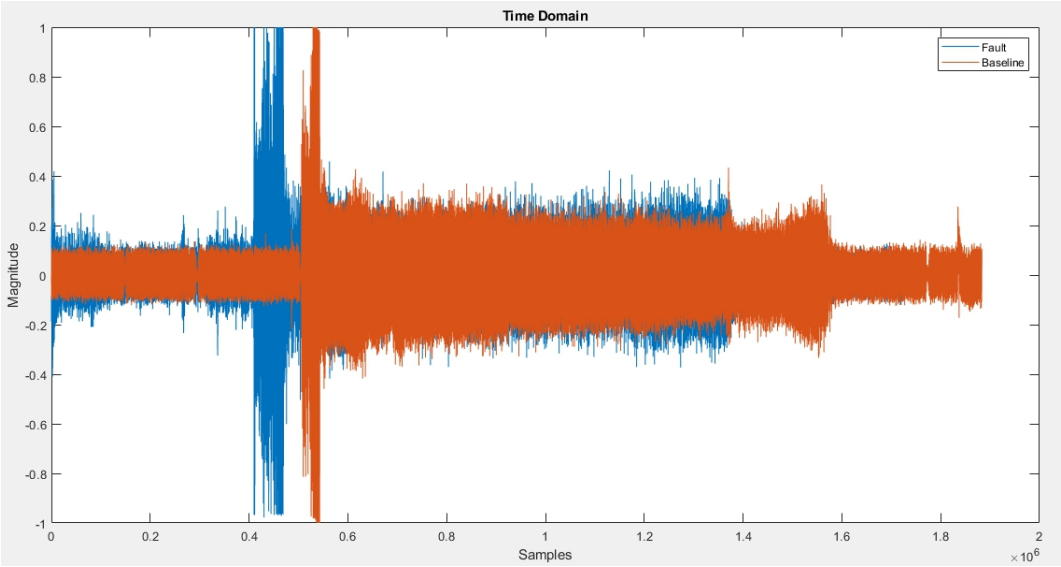



Figure 5.21: Audio overlay plot of Hyundai Getz recording

Interestingly, we can see some of the same spikes at 16-17 Hz and 32-34 Hz below in Figure 5.22, when compared to the FFT plot of the vibration data in Figure 5.17. What the vibration FFT does not show us however, is the resonances that appear at higher frequencies as a result of the fault introduced. These occur at multiples of the 17 Hz, and can be seen at 187 Hz, 378 Hz and 565 Hz.

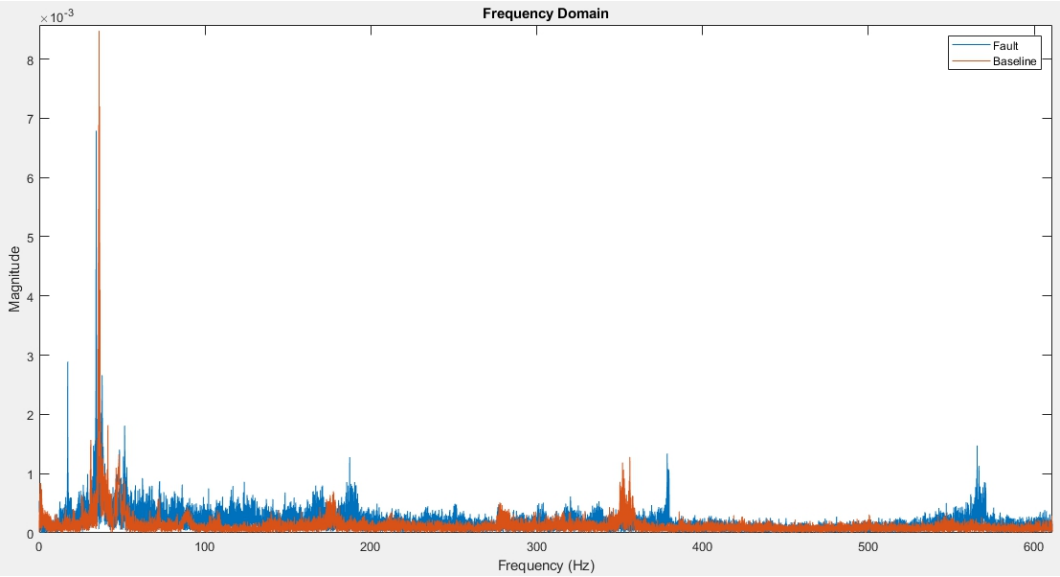



Figure 5.22: FFT overlay plot of Hyundai Getz audio recording

### 5.8.2 LPC and Mahalanobis Distance

We have seen from the time and frequency domain plots that the faults are evident. Unfortunately, there was no simple way to automatically identify the fault from that representation of the data. This led to the application of LPC coefficients and the Mahalanobis Distance measure.

Using the LPC technique, it became efficient to store a small number of float values in a text file as a baseline that represented particular faults and healthy engines.

Within a MATLAB script, LPC coefficients were calculated for a frame size of 3000 samples then averaged over the input sample time size. These averaged float values represented Nth degree polynomial coefficients, where N = 15 for the following results presented. The polynomial was taken over 75 percent of the audio data file, and represented approximately 15 - 20 seconds of normal or faulty engine run-time, sampling at 44.1 kHz. The separate float values, or 15th dimensional vector points, were then rounded off after the 8th decimal place.

The saved LPC coefficients could then be imported into a separate MATLAB script as a 15th dimensional vector. Within the separate script, we would then have a 15th dimensional vector for the first 25 percent of each recording (healthy and fault condtions) which represented our baseline data to use in future comparisons.

Similar to the process of creating baseline LPC coefficients described above, an analysis of an unknown audio sample was carried out. This resulted in a collection of Nth dimensional vectors, that clustered about a centroid specific to the unknown signal under test. It is this measure of how the LPC coefficients cluster about particular centroids that allows us to compare two different signals. The unknown audio LPC vectors (for each frame of 3000 samples) are compared to each baseline fault vector to show their "closeness" to the unknown data LPC centroids using the Mahalanobis Distance measure as described in Section 2.9.3.

### Audio data

For the LPC and Mahalanobis analysis, only the raw audio data was used (as opposed to the raw accelerometer data), as it showed greater depth of frequency as discussed above in the FFT analysis of audio and vibration data. Audio data was also easier to procure for cross platform analysis. Additionally, there were time constraints on the development of vibration analysis code using the LPC/ Mahalanobis classification method. It was decided to pursue audio analysis only from this point in the research.

### Test conditions

The baseline LPC parameters that are used to define the fault/ healthy condition were derived from an average of the first 75% of the audio files. This 75% block was taken out of the original file using tools in the editing software, Audacity. LPC coefficients were derived for each frame of 3000 samples within the 75% block, and then an average was taken using MATLAB functionality, resulting in a 15th dimensional vector (16 float values) for the entire 75% block of audio.

LPC parameters for each frame of the remaining 25% of the original audio file was then derived and compared to the LPC baseline data using the Mahalanobis function. Additional data to test the baseline was derived from other recordings of the same fault, but at a different time within the trial.

### Trial #3 Hyundai Getz

By being able to induce a fault in the Hyundai Getz, and having baseline data of its healthy operation as well, we were able to collect suitable audio samples in order to assess the LPC/ Mahalanobis classification method. The results seen below in Figure 5.23 are amongst the most significant findings presented in this report.

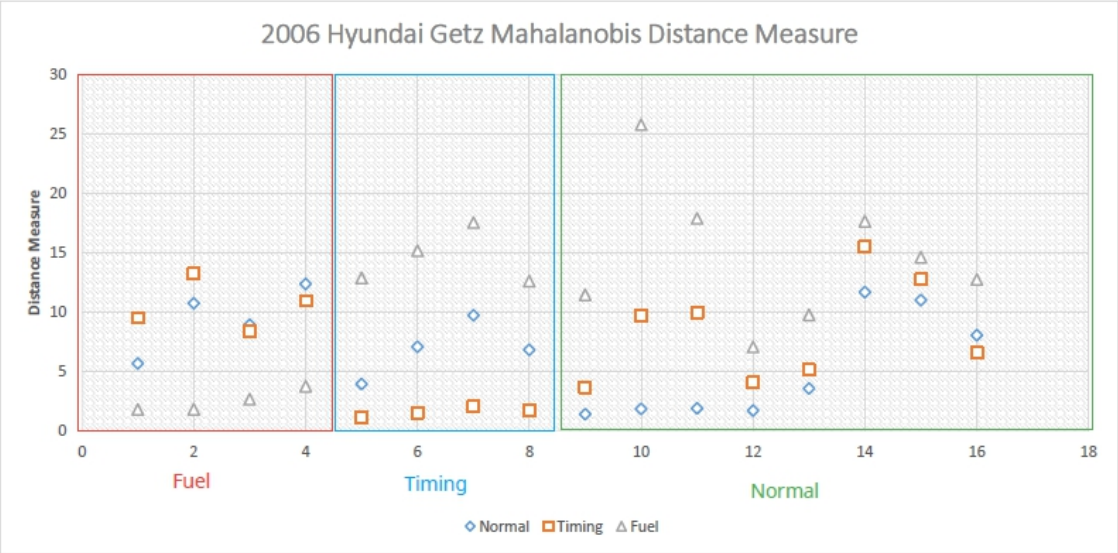



Figure 5.23: Mahalanobis Distance Results

In 15 out of 16 cases, or 93.75 % of the time, the LPC coefficients and the Mahalanobis Distance Algorithm were able to detect the correct condition, using an unknown input sample.

This can be seen in Figure 5.23 above, using the legend below the graph. For all of the first four "unknown" inputs on the left hand side of the figure, the algorithm is able to compare them with the stored baseline LPC vectors and determine that all four unknown input samples are fuel fault recordings, indicated by the lowest Mahalanobis Distance value. The samples are not actually unknown, they are just seen as such to the algorithm under test. This helps us accurately assess the effectiveness of the Mahalanobis Distance algorithm.

The Y value, or Mahalanobis Distance value, gives us a singular value for just how close the unknown audio LPC coefficients are to the baseline fuel fault LPC centroids (one for each vector value).

The trend follows across to the four timing fault recordings, and to the majority of the last eight normal engine operation recordings with one exception (the 16th test), where the lowest Mahalanobis Distance on the graph indicates which fault or normal condition is closest to the baseline LPC data.

## 5.9 Further research - Engine Database recordings

It was also prudent to investigate different fault conditions on a variety of car models. This was achieved by downloading a number of audio files from an online database which contained before and after (faulty and healthy) recordings of engine faults. These were then compared using similar methods of analysis as the sections above.

### 5.9.1 FFT analysis

As we can see below in Figure 5.24, this vehicle presented a magnitude difference with similar characteristics to the ignition plug fault in the the Hyundai Getz. The vehicle is question was stated as a BMW N47 from the database, and the fault was given as "timing". Listening to the sample also indicates that the microphone position was changed during the recording, which is different to the Getz, where the microphone was fixed. This shows in the figure as the magnitude of the fault and baseline samples increasing and decreasing.

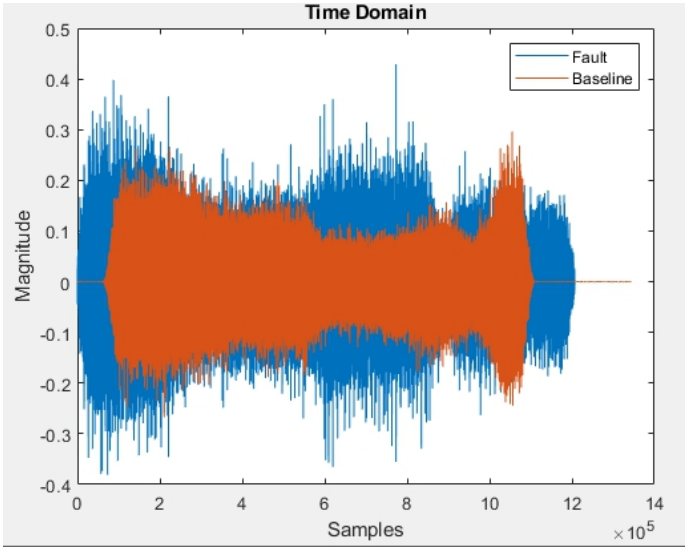



Figure 5.24: BMW audio overlay

With the Hyundai Getz fault in trial #3 we saw peaks at certain frequencies which we can see again in Figure 5.25. These frequencies differ, as the engine under test is different. The BMW engine is a four cylinder, like the Hyundai, but is a common rail diesel, which operates very differently to the Hyundai Getz.

Unfortunately, we have less information this time and cannot extrapolate on the RPM of the engine, and its resonant frequencies, as we do not know the particular idle RPM of the engine under test. We can see however, that the fault data exhibits larger peaks at higher frequencies, and is quite significant at 83 Hz, as compared to the baseline magnitude at the same frequency.

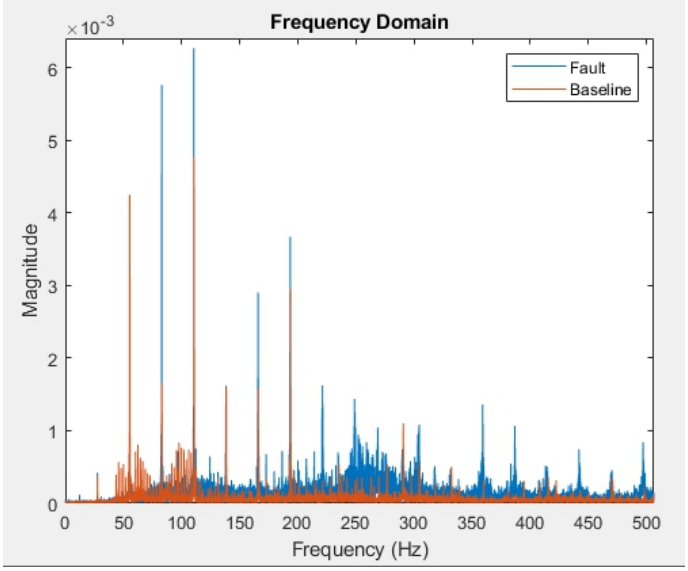



Figure 5.25: BMW FFT overlay

### 5.9.2 LPC and Mahalanobis Distance

The next stage was to try and use the classification method on actually unknown data (where before we knew the fault condition present). This can be seen below in Figure 5.26. The algorithm was able to detect when the engine has a timing fault (no further description), and when the engine is operating normally, with the exception of one test (7/8 tests or 87.5% accurate). Classification of the fault is indicated on the graph by the lowest Mahalanobis Distance value on the Y axis.

Upon further investigation, it appears that the last 25% of the recording (used as an unknown sample to test the baseline) had the microphone moved significantly further

away. The timing fault can be heard faintly, but is overshadowed by environmental noises and may explain why the algorithm was unable to capture the condition.

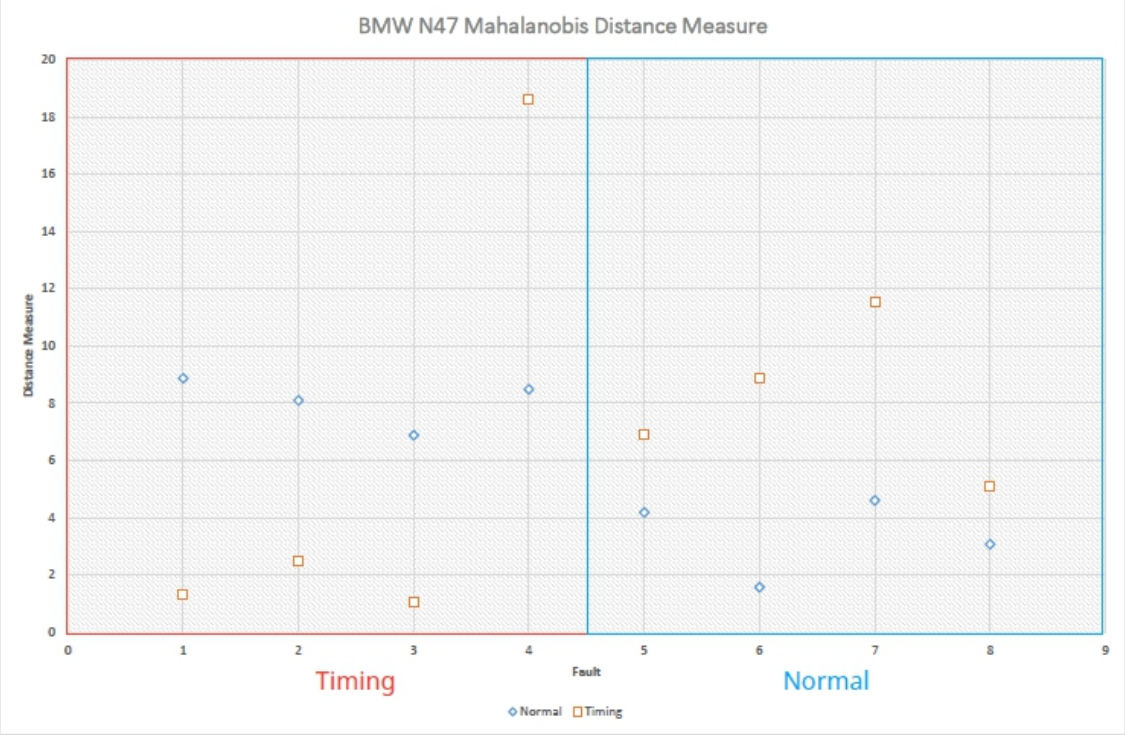



Figure 5.26: BMW Mahalanobis Distance Results

This does not give us a description of the fault however, only that a fault exists. What we do have however, is a baseline recording of a fault that could be used later on if the fault is diagnosed.

## 5.10 Cross platform analysis

So far, all tests have been conducted only on the models themselves. This means that recordings were taken and tested on their respective platforms only. This investigation would be remiss if it did not attempt to classify similar faults or healthy behavior across a variety of models. From the previous test phases there are a number of baseline files which could be used to try and find similar faults in unknown data on different models of car. These baselines were tested to see if the Mahalanobis Distance algorithm could detect any similarities in different models. The baseline files mentioned are as follows:

- Hyundai Getz Ignition Plug disconnected

- Hyundai Getz Premium Fuel
- Hyundai Getz Normal operation
- BMW N47 Unknown Timing Fault
- BMW N47 Normal operation
- Subaru Forester Normal operation

The baselines listed above were derived from the first 75% of the recording, and the remaining 25% was used to test against the baselines in seen in Figure 5.27. As we can see, the most significant result is that each sample has the most in common with itself, as we would expect.

Further to this however, it appears from Figure 5.27 that common faults and common healthy engine recordings do not have any correlation (i.e. we could not use a timing fault on the BMW N47 to diagnose a timing fault on a Hyundai Getz, or a healthy recording of any vehicle to ensure correct operation on another).

Lastly, there does appear to be a relationship between faulty and healthy engines of the same model. This can be seen where the BMW N47 healthy and faulty columns exhibit the lowest Mahalanobis Distance respectively. The Hyundai Getz also shows the same for its 3 conditions (timing, fuel and healthy).

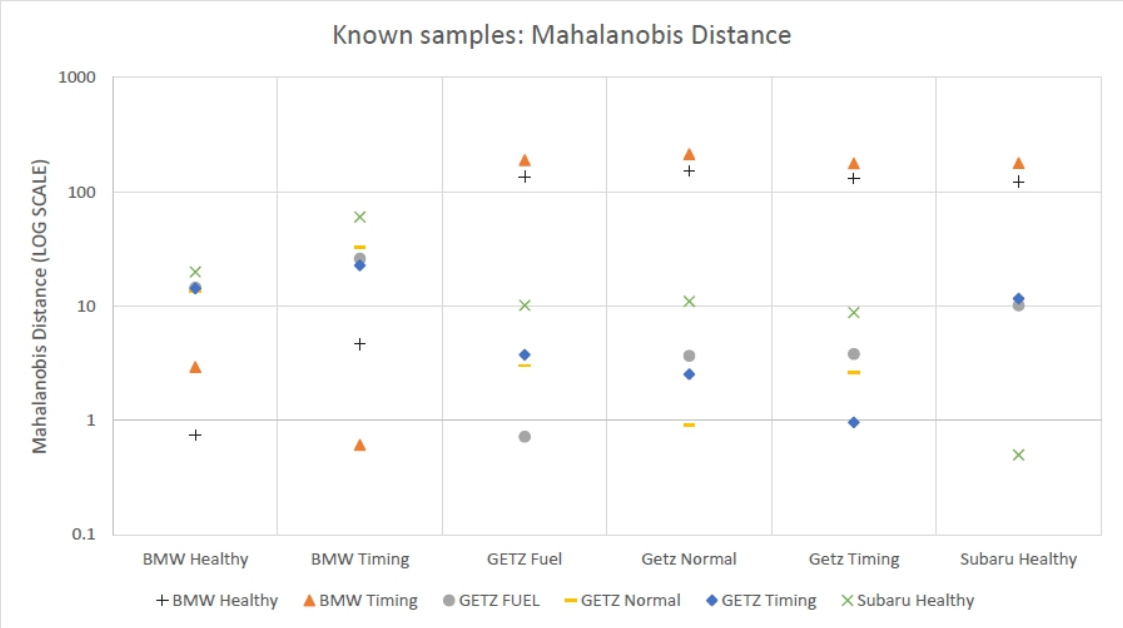



Figure 5.27: Known Samples: Mahalanobis Distance

In the last series of tests, a number of unknown and unrelated samples were tested against the baselines listed above. This was to further investigate whether there was any significant relationship between model conditions. The unknown samples introduced were a series of different knocking faults, a small number of singular faults such as a fan clutch ball-bearing, camshaft displacement, and engines pre-service, and a number of typical healthy engines from different models. The results can be seen below in Figure 5.28.

There does not appear to be any significant correlation between the input samples and the baselines. None of the healthy engine input samples have a low Mahalanobis Distance when compared to healthy engine baselines, or even a lower MD value than other baseline conditions. The BMW N47 recordings (faulty and healthy) seem to have the most in common with every unknown recording, which is not useful from a fault finding standpoint. Additionally, the Subaru Forester commonly has the largest MD distance when compared to most input samples.

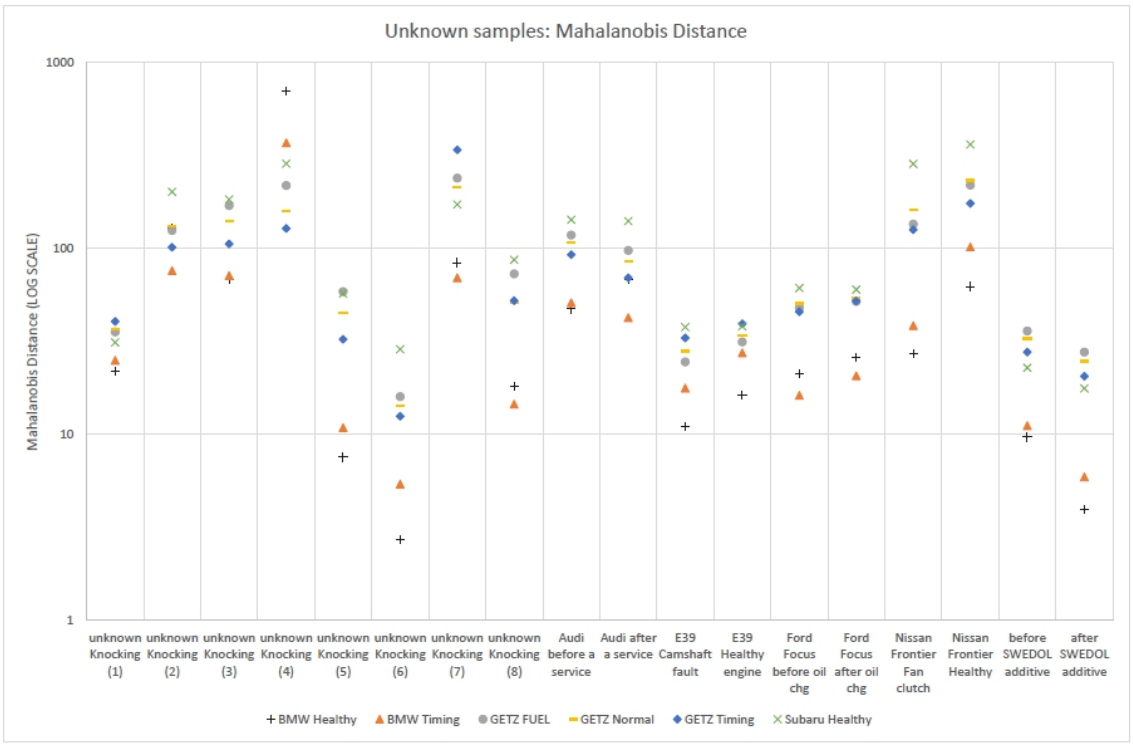



Figure 5.28: Unknown Samples: Mahalanobis Distance

## 5.11 Discussion of results

This section attempts to disseminate and discuss the results in the preceding sections.

### 5.11.1 Preliminary Component Testing

The individual component testing was a necessary step in confirming the correct operation of the sensor packages, and saved a large amount of time that would have otherwise been spent in the field. By delving into the operation of the sensors, it was possible to "see" into what was to be recorded and make some initial assumptions about the engine recordings phases, and tweak the sensor setup and code to suit.

Both the microphone and the accelerometer module selections proved to be suitable for purpose. In addition to this, the Teensy micro-controller platform was more than suitable for processing the large amounts of audio data present.

The GPS sub-system proved to be a challenge after the design phase, as the self-manufactured Low Noise Amplifier failed in the test phase, leading to the purchase of an out of the box solution. The manufactured PCB LNA was intended to be powered through the coaxial line from the GPS receiver and when the original plan failed, a bulky battery solution had to be added into the circuit to power the LNA externally. The manufactured patch antenna worked well however, and had appropriate gain considering the dielectric it was manufactured from (FR4).

The preliminary Wi-Fi tests showed the connection of a client computer to the micro-controller server over WiFi communications and proved the successful operation of the ATWINC1500 module.

### 5.11.2 Data Collection

A number of vehicle trials were conducted in order to assess the accurate usage of the sensor packages and data collection methods. During the trials, a number of significant issues were resolved and required mention as follows:

**Vibration:** After Trial #2, the accelerometer settings came into doubt. During the

preliminary testing and Trial #1 it was assumed that the sampling rate (400 Hz) would be sufficient to pick up relevant engine vibrations. After some initial data analysis and research, higher sampling rates were investigated (up to 5kHz - maximum sampling rate of the accelerometer package) in order to maximize the spread of data being collected from the engine. An unfortunate side effect of sampling at 5 kHz was that the data became quite granular, as the resolution at 5 kHz is reduced and enters a low power mode.

**Audio:** It was discovered during Trial #1 the placement and gain settings for the microphone were incorrect. The microphone was originally mounted in the engine bay with the bonnet closed, but this proved too much for the microphone and resulted in a large amount of clipping evident in the audio plots. Additionally, gain setting were reduced physically (using wire connections to the micro-controller and sensor package) to work in conjunction with the automatic gain levels. This proved to work well and a resulting audio plot can be seen in Trial #2, Figure 5.18.

**Data Storage:** A large issue that arose after Trial #2 was combining the data from the sensors onto the memory card of the micro-controller. Only one information source could write to the memory at a time, and led to latency issues and missing data until the issue was discovered. The audio recording was the most affected, as the it was being held while the accelerometer and GPS modules wrote to the card. This was fixed through software, where the audio was allowed to continue recording to a buffer in the background while the shorter processes wrote to the memory.

**GPS:** The manufacture and use of a patch antenna in the scope of this project was intended to show the suitability of an antenna which could be mounted to the skin or window of the vehicle. While the bulky battery and LNA solution was not ideal, the trials showed that a patch antenna could be mounted to the upper skin of the vehicle and provide accurate GPS readings for the duration of the trip. A variety of information is able to be added to the map depending on the requirements of the user. This could include any sensor measurement taken at the point of GPS location capture.

**OBD2:** The use of the OBD2 hardware and software was limited in the scope of this project, and was shown mainly as a supplemental aid to the sensor information being collected. The OBD2 trials showed that the vehicle under test was capable of producing a large amount of accurate information on the vehicles operation. This was limited to real-time information only however, or in the form of graphical plots over time rather

than the raw data itself.

**Wi-Fi:** The Wi-Fi and server application showed successful data transfers between the micro-controller and the client computer via WiFi communications, and upon return to a geo-located home base. GPS accuracy was a minor issue, and further investigation into Wi-Fi clocking speeds and bandwidth is warranted to transfer large data files.

### 5.11.3 FFT and Spectrogram Analysis

The time and frequency domain plots gave us the first glance at the informational content in the raw data. This was immensely useful in fault finding the sensor settings, as well as showing how the signal was behaving in time and its frequency content.

**Time:** Both the accelerometer and the microphone showed very similar profiles in the time domain, and it was clear to see engine start times, periods of acceleration and stop times. Additionally, the magnitude of the fault signals were clear to see. This was useful in beginning to understand how a fault may behave, where fault signals exhibit a larger magnitude.

**Frequency:** When we used the FFT function and moved into the frequency domain, we were able to see the magnitude of various frequencies present in the accelerometer and microphone data. This is where the content of the signal we were shown began to differ. Vibration FFT plots showed clear spikes at the lower ranges, between 0 and 100 Hz but did not show much response above that even when sampling rates were increased.The audio recordings on the other hand were able to pick up content between approximately 10 Hz and up to 20 kHz.

The frequency domain also showed us the first significant glance at the specific differences between faulty and healthy data. The first fault that was introduced on the Hyundai Getz, seen in Figure 5.22, was an ignition plug disconnected. This showed up very clearly on the FFT plot, where the fundamental frequency of the engine can be seen at 32-34 Hz and the fault frequency at half of this, 16-17 Hz. As half of the cylinders stopped firing when the plug was disconnected, this is evidence that the FFT reflects the engines frequency. Additionally, resonances were seen in the FFT plot as a result of the fault.

Further to this, we saw again in Figure 5.25, which was an unknown timing fault on a

BMW N47, that the FFT plot showed a significant relationship between the baseline data and fault data for the fundamental frequencies of an operating engine.

**Spectrogram:** The spectrogram plot gave us a visual representation of the frequencies as they varied with time, and incorporated a heat map to indicate the intensity of the content. This was used mostly to better understand the information in the signal to inform later investigations. The spectrogram plots showed a clearer picture of resonant frequencies, intensity, and the operation of the engine over time.

### 5.11.4 LPC and Mahalanobis Distance

**LPC:** Linear Predictive Coding provided a way to classify the large vectors of raw data with a much smaller number of polynomial coefficients that accurately represented the data in question. In many typical speech applications, an LPC10 coder is used (containing 10 coefficients) and a frame size of 10-20 ms is used. Based upon the study by Monica Chamay, Se-do Oh and Young-Jin Kim (2013) on LPC use for engine vibration analysis, which was elaborated upon in the literature review, a particular number of coefficients was used (15) and a frame size of 3000 samples was used (for sampling rate of 44.1 kHz), which is roughly 73 ms. Engine data is very different to speech patterns, and for this reason the frame size and number of coefficients used in this report is based heavily on the aforementioned study. Smaller frames sizes and less coefficients were briefly investigated, but the results are not presented in this report as they proved inconclusive and warranted further study. The LPC coefficients created in a separate script to the analysis script, then added to a TXT file through MATLAB. This provided an effective way of storing the baseline data for later analysis and comparison to the test samples.

**Mahalanobis Distance:**

The Mahalanobis Distance measure proved to be a strong method for comparing the LPC coefficients. It was able to show the similarities between different recordings of the same condition (i.e. 2 healthy recordings of the same vehicle). We were able to use its algorithm to successfully predict a fault or healthy condition, if we held the baseline data for that fault or healthy condition.

Extrapolating from this, it may also be possible to predict when a different car of the

same model and year presents with the same fault or healthy condition but this was not tested in the scope of this report.

The cross platform analysis did not prove anything conclusively, except for the indication that using Mahalanobis Distance measure and LPC coefficients to diagnose unknown vehicle faults would not be possible. In order to diagnose successfully, it is necessary that baseline data for that particular fault and model of vehicle is present in the database.

LPC coefficients and Mahalanobis Distance are not able to distinguish or separate specific faults from the surrounding noise of the vehicle and/ or environment. Age and model of the vehicle, where the vehicle has spent its operating life, and method of recording the vehicle sound are a number of factors that will inhibit the algorithms from accurately classifying a fault or healthy condition.

## 5.12 Cost Benefit Analysis

At the forefront of Health and Usage Monitoring Systems (HUMS) research is the Defence Science and Technology Organization (DSTO). Vehicle research carried out by the Land Division of DSTO describes some appropriate models and tools for the Cost Benefit Analysis (CBA) of Condition based maintenance (Gallasch 2016). Two of the more relevant tools to this project work will be described in the following sections:

**Qualitative - KT Box HUMS Cost Benefit Tool:** The HUMS cost benefit tool examines the relationship between the benefits of fitting HUMS to a vehicle fleet, and the added cost of doing so. The tool attempts to capture a wide range of considerations including the life-cycle of the equipment, spares and personnel. An example of the output from the HUMS cost benefit tool can be seen below in Figure 5.29. This gives us a qualitative view with no specific values, is very overstated when compared the research carried out in this project, and it would give us no clear picture on the value of this project. It does however strongly indicate the potential benefits of an installation of HUMS or related Condition Monitoring systems on a vehicle fleet.

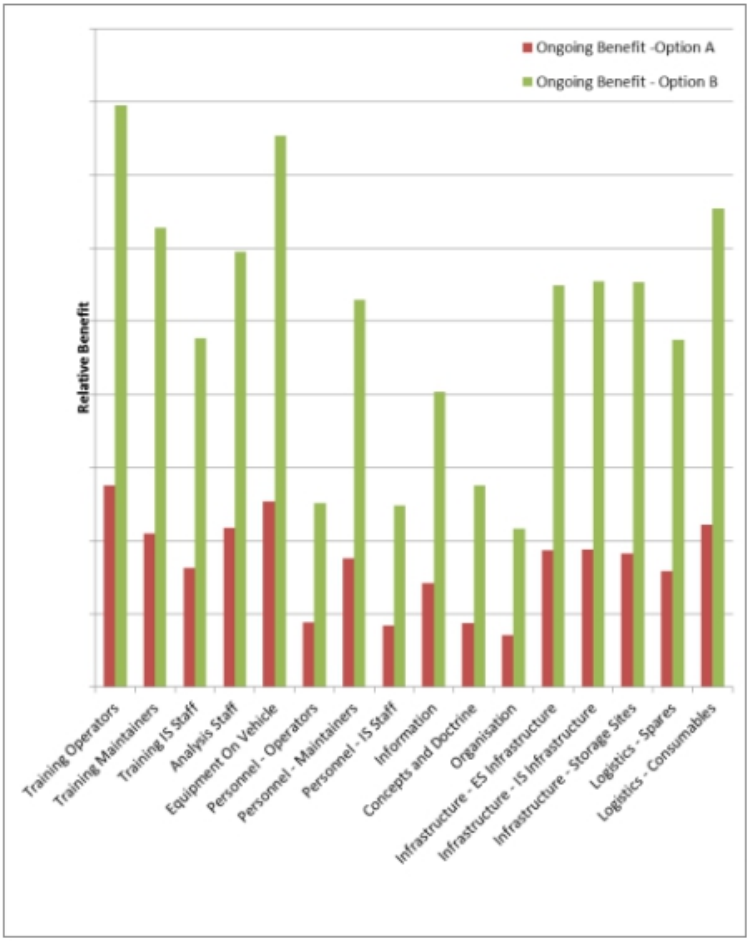



Figure 5.29: Example of HUMS Cost benefit tool

**Quantitative - EXAKT:** EXAKT is a decision support tool used for optimizing predictive maintenance. It is not a cost benefit tool in itself but supports cost benefit calculations if maintenance and condition data is available. EXAKT can predict equipment failure, estimate remaining life and define a mix of preventative replacement and run-to-failure in order to optimize costs and achieve a balance. The tool requires component parameters, records of failure and preventative replacements, and a history of condition data. Using this data, it can produce survivability models (based on the Weibull distributions) to characterize the probability of failure as seen below in Figure 5.30. The EXAKT tool relies heavily on historical data of inspections, maintenance costs, failures and operation. For this reason, it would be well suited to an established fleet of vehicles fitted with a condition monitoring system.

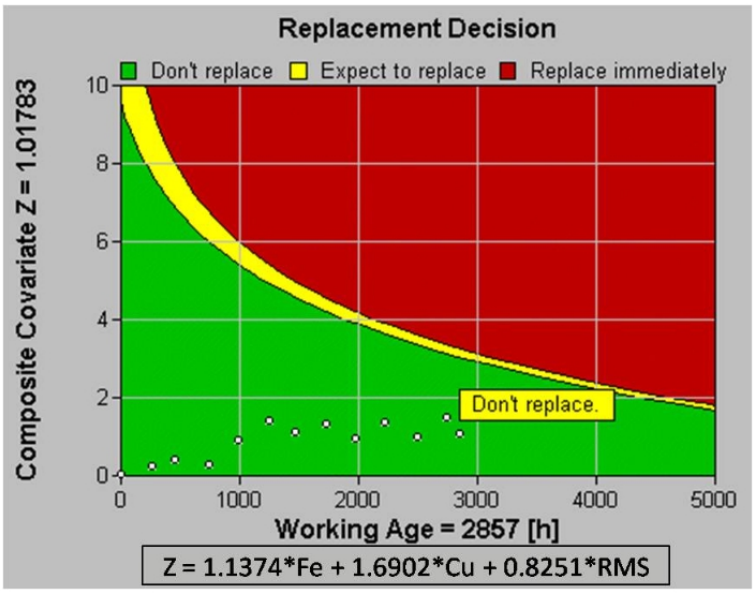



Figure 5.30: Example of EXAKT decision analysis

### Cost Benefit Analysis approximation

In an attempt to quantify the possible benefits or outcomes from this project work, a Cost Benefit Analysis based on the research above was carried out in this section and can be summarized as follows:

- **Cost baseline:** This estimate covers the approximate costs of operating a vehicle, including initial outlay, support costs, and disposal **without** using condition monitoring. The RACQ (RACQ 2019) has recently published a series of studies on the various operating costs for private vehicles and are current as of the 19th of March, 2019. The cost, over a 5 year period and including outlay, ranges from $29,738.26 for the Mitsubishi Mirage hatch to $127,519.33 for the Tesla Model X (electric vehicle).

- **Econometric considerations:** The study by the RACQ above takes into account the current conditions of the market. In addition to this, we must attempt to predict inflation rates, purchase discounts, fuel prices, and labor rates. This is beyond the scope of this report.

- **Operating Scenarios:** Vehicle fleets will be operated very differently to private vehicles and as such, should be analyzed separately if required. The RACQ study above uses an average distance traveled of 15,000 km every year to calculate fuel

costs and maintenance.

- **Condition Monitoring implementation costs:** The approximate cost of the hardware used in this project is $600, in addition to time invested (nominally free). In order to implement this hardware commercially however, there would have to be significant investments in time and resources to bring the product to market. It exists currently only as a proof of concept and is not able to be mounted or installed permanently. A PCB package and mounting case, antenna fixtures, and re-producible analysis software are among these requirements.

- **Ongoing costs:** Ongoing costs of operating the condition monitoring software/ hardware are difficult to estimate at this point in time. The hardware used may suffer from early failure as it is not built to withstand the rough engine conditions over an extended period. This may result in expensive sensor purchases in the future of the project.

- **Cost benefits:** Again, it is difficult to estimate the cost savings of operating the condition monitoring software/ hardware. In similar systems used in aircraft, HUMS was installed and resulted in savings in parts cost and operation in the field of $2.1 million for a fleet of AH-64 helicopters over 8 years(Honeywell Aerospace 2015). These savings were primarily attributed to a reduction in maintenance.

- **Non-monetary benefits:** In addition to saving money, there are a number of non-monetary benefits. These include an increase in vehicle data awareness for fleet management, operator safety and contributions to vehicle availability.

- **Relative importance of costs and benefits:** This section would require an assessment of the relative importance of the cost factors against the non-monetary benefits. If the system is not justified by cost savings alone, it may be further advantaged by discussing non-monetary benefits.

- **Re-assessment of value over time:** After the condition monitoring program has be running for a period of time it becomes necessary to re-visit and re-assess the viability of the system to ensure its continued worthiness.

While it is hard to place a numerical value on the benefits of a condition monitoring system in the scope of this project, it has been shown that the practice in other sectors

has been very beneficial in terms of maintenance hours spent, cost savings, and non-monetary benefits. It would extend that a condition monitoring system in vehicle fleets will follow this trend. It is also the hope that discussing these additional factors will strengthen the interest in condition monitoring research and analysis in vehicles.

## 5.13 Chapter Summary

This chapter has provided a selection of results from the testing phases, and discussed the implications of this data. It has been shown that the sensor data was collected and stored in a manner that was appropriate. Additionally, this data was successfully analyzed using MATLAB to show that LPC coefficients and Mahalanobis Distance measure could predict a fault or healthy condition of a vehicle given the baseline data was present. Finally, a Cost Benefit Analysis was roughly discussed and its implications on the project.

# Chapter 6

# Conclusions and Further Work

## 6.1 Chapter Overview

This chapter presents a number of conclusions that were made based upon the results and design work outlined in the preceding chapters. These conclusions are separated into sections to better disseminate the content and inform the reader succinctly.

## 6.2 Hardware and Software Conclusions

With some small exceptions, the majority of the hardware used in the course of this project was adapted to help achieve the objectives stated in the project specification document and is summarized as follows:

- **Microphone:** The MAX9814 microphone module was able to record the engine audio faithfully at 44.1 kHz, with some initial gain adjustment. The automatic gain feature of this module is an important reason for the success of this component.
- **Accelerometer:** The LIS3DH accelerometer was suitable to measure the vibration of the engine in testing, but the project could have benefited from an accelerometer with higher resolution at higher sampling frequencies.
- **GPS system:** The manufactured GPS antenna and supporting LNA and Adafruit GPS module worked without fault at critical periods, and was able to faithfully provide the GPS data to the micro-controller. Size of the LNA power source is an issue for future projects that require a mountable antenna.
- **OBD2:** The OBD2 interfacing hardware was not utilized much in the scope of this project. It was simple to set up and operate in order to view the CAN bus data via the OBD scan doctor software. This was useful to view actual RPM speeds and compare to audio/ vibration spikes in the FFT.
- **Wi-Fi:** The ATWINC1500 Wi-Fi module worked well to communicate information from the micro-controller to the client computer. The supporting library also provided well built functions for developing an Access Point server for local access only. One drawback to the library however is its inflexibility, and was not suited to micro-controllers not based on the Arduino model.
- **Teensy 3.6:** The Teensy 3.6 was the standout of the project. It provided high clocking speeds, large program storage, external memory in the form of micro SD. Most important of all was the support of the designer and associated libraries. The Teensy was able to process and store a high amount of audio, vibration and GPS data without fault.

## 6.3 Analysis Conclusions

Two main methods of analysis were investigated in the course of the project, FFT analysis and LPC/ Mahalanobis Distance analysis. The conclusions are summarized below:

- **FFT:** FFT (and spectrogram plots) were useful in the sense that it gave a visual representation of how the signal content changed when fault data was compared to healthy data. Large faults such as the missing ignition plug on the Hyundai Getz were immediately apparent. In addition to this, the time plots also provided an insight in the fault conditions, as the fault data was commonly larger in magnitude. Both audio and vibration data was able to be processed in MATLAB to show some basic correlation between the two types of data being collected.

  One drawback to the FFT analysis was the storage of data, where entire raw data sets for individual faults would have to be stored for later comparison to the unknown data. Another drawback of the FFT was the inability to classify just how different the two inputs were. We could see that there was indeed a visual difference in frequency content of the two input data sets, but this had no overall significance in being able to diagnose a specific fault on its own, and without prior knowledge of the fault.

- **LPC/ Mahalanobis:** In order to overcome some of the drawbacks to the FFT analysis methods stated above, Linear Predictive Coding and Mahalanobis Distance were investigated. LPC provided an efficient way for fault data to be stored, where raw fault data in the form of hundreds of thousands of values could be characterized with just a few polynomial coefficients. LPC coefficients were taken over a frame size of 3000 samples, and then averaged over the entire data set to reduce the raw data to just 15 values (based on a 15th order LPC).

  A Mahalanobis Distance function was then used to compare the "difference" between LPC coefficients of the unknown input data (for each frame) and the stored fault data coefficients. This resulted in a single value that represented how far away the unknown samples were from the stored data.

  In the scope of this project, the Mahalanobis Distance function was able to detect a known condition (fault or healthy) with a high level of probability when tested against a different audio samples of the same fault or healthy condition on the same vehicle.

It was not able to detect similar cross platform faults, where the vehicle was of a different make, model or year. As it stands from the results of this analysis, the algorithm requires the LPC coefficients for the appropriate condition on the same vehicle.

It may be possible to diagnose a condition across a vehicle with the same make, model and year, but this was not investigated. Operating conditions of the vehicle may also affect the diagnosis, but was also not investigated.

If vehicles of the same make, model and year that presented with common faults were able to be diagnosed using the Mahalanobis functions, it would be possible to build a database of LPC coefficients to compare recordings with. This database would be wide in scope, but since the LPC coefficients are not large, it would not be significant in storage space (it would also be much smaller than an FFT database).

## 6.4 Project Objectives

This project has evolved to meet a series of stated objectives that began with the project specification document. It is important to state when these were achieved and can be seen below:

1. The background information relating to data logging micro-controllers used to monitor vehicle conditions was thoroughly researched as seen in the Literature Review, Chapter 2.

2. An implementation of a real-time, condition monitoring micro-controller that would be used to monitor a set of specific conditions was designed in Chapter 4.The results can be seen in Chapter 5. The micro-controller package utilized an SD card to store the input data.

3. The design of hardware and software that developed audio and vibration sensor packages can also be seen in Chapter 4.

4. An external GPS patch antenna, with supporting hardware and amplification was designed and built in Chapter 4. This was integrated with the receiver module and logged data to the micro-controller accurately. Results can be seen in Chapter 5.

5. Sensor signal analysis in order to classify normal and abnormal behavior can be seen in Chapter 4 and 5. FFT analysis as a visual reference and LPC coefficients as a numerical application were used to classify the behavior for different conditions present in the test vehicles.

6. WiFi was used as a communications medium to transfer recorded data between the micro-controller and the client computer. Design and results can be seen in Chapter 4 and 5.

7. Built upon the LPC coefficients, the mahalanobis distance measure was used to calculate a difference figure that represented the diagnosis of a vehicle fault based upon audio recordings. Design of the algorithm and results from testing can be seen in Chapter 4 and 5 respectively.

## 6.5 Overall relevance

The hardware, software and analysis research gave an insight into the size and breadth of the capabilities of condition monitoring. A Cost Benefit Analysis was also briefly visited to better inform the reader of the potential and benefits of using Predictive Maintenance and Condition Monitoring practices. It is the hope of this research that using low cost sensor modules and the detection algorithms will lead to development of predictive maintenance models, where the Mahalanobis Distance measure will give indications of approaching faults.

## 6.6 Further Work

This research attempted to answer a number of questions, generally raised from the project objectives. Unfortunately, while it has answered the project objectives for the most part, it has raised more questions than it has answered.

There was a lack of time to investigate the following, and any and all are appropriate for future work.

- Amplitude normalization for the audio signals was not carried out prior to compar-

ing them using the LPC coefficients and Mahalnobis Distance measure. Doing this in future would allow the comparisons to eliminate any effects that the amplitude might have on the signal (i.e. placement of the microphone). Placing the microphone in the same location would not completely mitigate amplitude variations.

- The design and implementation of a suitable Printed Circuit Board (PCB) and mounting package would be a large step to commercializing this product. This would allow the package to be re-produced easily, travel well, and be stored appropriately.

- Adding further sensor arrays to the project would increase the importance and relevance of the project. These could include but are by no means limited to: Oil quality (acoustic or light sensors) or a more sensitive accelerometer.

- The external GPS antenna, while suited to mounting on the skin of vehicle, does not have supporting hardware that allows this. The LNA and its power source require a design that allows the LNA to be powered from the GPS module itself, via the coaxial cable.

- The OBD2 module and support software was not investigated sufficiently. It is possible to integrate the OBD2 module with the micro-controller and store/ process real time data on the SD card rather than via a computer. This could be an invaluable tool when comparing fault markers.

- The Wi-Fi transfer speeds on the ATWINC1500 module were quite slow, and increasing the bandwidth and/ or clocking speed of the module would be a logical step for transferring large data files.

- Only the raw audio data was used in LPC and Mahalanobis Distance comparisons. LPC is used most commonly on speech parameters and audio files. It may be useful to investigate processing the accelerometer data with the same algorithms.

- Investigate the effects of different frame size and number of LPC coefficients on the accuracy of the results.

- Investigate using the Mahalanobis Distance algorithms with Linear Predictive Cepstral Coefficients (LPCCs) rather than the LPC parameters, and its effects on accuracy of diagnosis in engine faults.

- Research the spectrum envelope of the LPC parameters as opposed to the FFT function, to prove that the LPC coefficients are a good representation of the FFT

and the content of the raw signal.

- Building a confidence model based upon further tests and recordings would be a simple mechanism to increase the relevance of this research. It would show that the conclusions made above are accurate.

## 6.7 Chapter Summary

This chapter has made a number of conclusions based upon the results presented in the preceding chapters. Suitability of hardware and analysis methods were summarized, and project objectives were re-visited to ensure compliance. Lastly, possible areas of future work were identified.

# REFERENCES


Accreditation Board for Engineering and Technology (1987), '"Criteria for accrediting programs in engineering in the United States', https://www.tandfonline.com/doi/pdf/10.1080/10408347308003631. [Online; accessed 20 September 2018].

Adafruit (2019*a*), 'Electret microphone amplifier - max9814 with auto gain control', https://core-electronics.com.au/electret-microphone-amplifier-max9814-with-auto-gain-control.html. [Online; accessed 23 April 2019].

Adafruit (2019*b*), 'Fgpmmopa6h gps standalone module data sheet', https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf. [Online; accessed 24 April 2019].

Adafruit (2019*c*), 'Mems digital output motion sensor ultra low-power high performance 3-axes "nano" accelerometer', https://cdn-shop.adafruit.com/datasheets/LIS3DH.pdf. [Online; accessed 22 April 2019].

Adafruit (2019*d*), 'Micro sd card breakout board', https://learn.adafruit.com/adafruit-micro-sd-breakout-board-card-tutorial/look-out. [Online; accessed 24 April 2019].

Adriansyah, A. & Dani, A. (2014), 'Design of small smart home system based on arduino', *Controls and Informatics Seminar (EECCIS)* pp. 121–5.

Albarbar, A., Mekid, S., Starr, A. & Pietruszkiewicz, R. (2008), 'Suitability of mems accelerometers for condition monitoring: An experimental study', *Sensors* **8**(2), 784–99.

Allam, S. & Elhady, U. (2018), 'On the development and implementation of the obd ii vehicle diagnosis system', *International Journal of Engineering Inventions* **7**(4), 19–27.

Amarasinghe, M., Kottegoda, S., Arachchi, A. L., Muramudalige, S., Bandara, H. M. N. D. & Azeez, A. (2015), Cloud-based driver monitoring and vehicle diagnostic with obd2 telematics, *in* '2015 Fifteenth International Conference on Advances in ICT for Emerging Regions (ICTer)', pp. 243–249.

Antenna-Theory.com (2011), 'Microstrip (patch) antennas', `http://www.antenna-theory.com/antennas/patches/antenna.php`. [Online; accessed 23 April 2019].

Antich, M. (2014), 'Commercial fleet maintenance costs remain flat in cy-2014', `https://www.automotive-fleet.com/155742/commercial-fleet-maintenance-costs-remain-flat-in-cy-2014`. [Online; accessed 14 September 2018].

Audacity (2019), 'Audacity', `https://manual.audacityteam.org/man/audacity_tour_guide.html`. [Online; accessed 25 April 2019].

Avago Technologies (2012), 'Alm-1612gps lna-filter front-end module', `https://datasheetspdf.com/pdf-file/657882/Avago/ALM-1612/1`. [Online; accessed 24 April 2019].

Baker, E. (2014), 'Open source data logger for low-cost environmental monitoring', *Biodiversity data journal* **2**, e1059.

Banu, N. M. S., Prabhu, M. R. & Sasikala, U. (2015), 'Design a square microstrip patch antenna for s-band application', *IOSR Journal of Electronics and Communication Engineering* **10**(2), 24–30.

Barelli, L., Bidini, G., Buratti, C. & Mariani, R. (2009), 'Diagnosis of internal combustion engine through vibration and acoustic pressure non-intrusive measurements', *Applied Thermal Engineering* **29**(8-9), 1707–1713.

Brown, A. & Sturza, M. (1993), Vehicle tracking system employing global positioning system (gps) satellites, Google Patents, Google Patents.

Carden, E. & Fanning, P. (2004), 'Vibration based condition monitoring: a review', *Structural health monitoring* **3**(4), 355–377.

CAS DataLoggers (2019), 'What is a data logger?', https://www.azom.com/article.aspx?ArticleID=16599. [Online; accessed 24 April 2019].

Choque, N., Davila, L., da Silva, W. & da Rocha, A. (2017), 'How construct a wlan multi-data acquisition system based on the integration of arduino and ni-labview platforms for educational applications', *DESAFIOS* **4**(4), 117–25.

Chul-Woo Kim (2013), 'Structural fault detection of bridges based on linear system parameter and MTS method', https://www.researchgate.net/figure/Concept-of-Mahalanobis-distance-MD_fig7_275701517. [Online; accessed 22 July 2019].

Collins, D. (2019), 'The best obd2 scanners (review and buying guide) in 2019', https://www.carbibles.com/best-obd2-bluetooth-scanners-reviewed/. [Online; accessed 24 April 2019].

Core Electronics (2019), 'Wireless', https://core-electronics.com.au/wireless.html. [Online; accessed 25 April 2019].

CSS Electronics (2019), 'Can bus explained - a simple intro (2019)', https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en. [Online; accessed 24 April 2019].

DST (2018), 'David warren - inventor of the black box flight recorder', https://www.dst.defence.gov.au/innovation/black-box-flight-recorder/david-warren-inventor-black-box-flight-recorder. [Online; accessed 14 September 2018].

E-Home Recording Studio (2019), 'A beginner's introduction to microphone polar patterns', https://ehomerecordingstudio.com/microphone-polar-patterns/. [Online; accessed 23 April 2019].

ELM Electronics (2019), 'Elm327 obd to rs232 interpreter', https://cdn.sparkfun.com/assets/learn_tutorials/8/3/ELM327DS.pdf. [Online; accessed 24 April 2019].

Fay, J. (2009), 'Thirteenth australian aeronautical conference'.

Freeman, T. (2018), 'Condition monitoring (cm)', https://www.corrosionpedia.com/definition/314/condition-monitoring-cm. [Online; accessed 14 September 2018].

Gallasch, G. (2016), 'Ninth dsto international conference on health & usage monitoring'.

Hameed, Z., Hong, Y., Cho, Y., Ahn, S. & Song, C. (2009), 'Condition monitoring and fault detection of wind turbines and related algorithms: A review', *Renewable and Sustainable Energy Reviews* **13**(7), 1–39.

Hanna (2017), 'Get to know the obd2: A brief history and explanation', https://engieapp.com/obd2-explained/. [Online; accessed 04 April 2019].

Honeywell Aerospace (2015), 'STUDY DOCUMENTS ECONOMIC BENEFITS OF HUMS', https://aerospace.honeywell.com/en/news-listing/2015/march/study-documents-economic-benefits-of-hums. [Online; accessed 30 July 2019].

Kite-Powell, J. (2019), 'Polar: The Original Fitness Tracker And Heart Rate Monitor', https://www.forbes.com/sites/jenniferhicks/2016/02/28/polar-the-original-fitness-tracker-and-heart-rate-monitor/. [Online; accessed 04 April 2019].

Learning about Electronics (2019), 'What is an Electret Microphone?', http://www.learningaboutelectronics.com/Articles/Electret-microphones. [Online; accessed 22 April 2019].

Leis, J. (2018), 'Communication Systems Principles Using MATLAB', JohnWiley & Sons, Inc.

MadgeTech (2018), 'Data loggers in healthcare — what are the top 5 uses?', https://www.news-medical.net/whitepaper/20181016/Data-Loggers-in-Healthcaree28094What-are-the-Top-5-Uses.aspx. [Online; accessed 04 April 2019].

Mathworks (2019), 'Digital signal processing (dsp)', https://au.mathworks.com/solutions/dsp.html. [Online; accessed 25 April 2019].

McLaren Applied Technologies (2016), 'The brain of an f1 car', lhttps://www.mclaren.com/appliedtechnologies/lab/brain-of-an-f1-car-mclaren-ecu/. [Online; accessed 04 April 2019].

Mcleod, S. (2017), 'Qualitative vs. Quantitative Research', https://www.simplypsychology.org/qualitative-quantitative.html. [Online; accessed 20 September 2018].

Mehta, Y. (2018), 'Condition monitoring crucial to industry 4.0', https://readwrite.com/2018/06/08/condition-monitoring-crucial-to-industry-4-0/. [Online; accessed 04 April 2019].

Monica Chamay, Se-do Oh and Young-Jin Kim (2013), 'Development of a diagnostic system using lpc/cepstrum analysis in machine vibration', *Journal of Mechanical Science and Technology* **27**(9).

Nooelec (2019), 'Nooelec SAWbird+ iO Barebones - Premium SAW Filter & Cascaded Ultra-Low Noise LNA Module for L-Band (Inmarsat AERO/STD-C) Applications. 1542MHz Center Frequency', https://www.nooelec.com/store/sawbird-plus-io.html. [Online; accessed 03 May 2019].

NTI Audio (2019), 'Fast Fourier Transformation FFT - Basics', https://www.nti-audio.com/en/support/know-how/fast-fourier-transform-fft. [Online; accessed 22 July 2019].

OBD Doctor (2019), 'Get to know your car better with OBD2 Diagnostic Software for PC, Mac and Mobile', https://www.obdautodoctor.com/. [Online; accessed 22 April 2019].

Oxford Dictionary (2019), 'Nyquist frequency', https://en.oxforddictionaries.com/definition/us/Nyquist_frequency. [Online; accessed 22 April 2019].

PCB Piezotronics (2019), 'Introduction to mems accelerometers', http://www.pcb.com/Resources/Technical-Information/mems-accelerometers. [Online; accessed 22 April 2019].

Peng, Z., Kessissoglou, N. & Cox, M. (2005), 'Study of the effect of contaminant particles in lubricants using wear debris and vibration condition monitoring techniques', *Wear* **258**(11), 1651–62.

Phythian, M. (1998), 'Speaker Identification for Forensic Applications', https://eprints.qut.edu.au/36079/3/__qut.edu.au_Documents_StaffHome_StaffGroupR24_rogersjm_Desktop_36079_Digitised20Thesis.pdf. [Online; accessed 15 July 2019].

Plant Services (2016), 'Top 5 impacts of big data on condition monitoring', https://www.plantservices.com/articles/2016/

pd-bd-ta-5-impacts-of-big-data-condition-monitoring/. [Online; accessed 05 April 2019].

Race-Technologies (2019), 'Data loggers', http://www.race-technology.com/au/racing/products/data-loggers. [Online; accessed 04 April 2019].

RACQ (2019), 'Car running costs', https://www.racq.com.au/cars-and-driving/cars/owning-and-maintaining-a-car/car-running-costs. [Online; accessed 30 July 2019].

Ramani, R. & Valarmathy, S. (2013), 'Vehicle tracking and locking system based on gsm and gps', *I.J. Intelligent Systems and Applications* **9**, 86–93.

Rao, B. (1996), 'Handbook of condition monitoring', Elsevier.

Sharma, R. (2017), 'How to interface gps module (neo-6m) with arduino', https://create.arduino.cc/projecthub/ruchir1674/how-to-interface-gps-module-neo-6m-with-arduino-8f90ad. [Online; accessed 24 April 2019].

Statistics How To (2017), 'Mahalanobis Distance: Simple Definition, Examples', https://www.statisticshowto.datasciencecentral.com/mahalanobis-distance/. [Online; accessed 15 July 2019].

Sujono, A. (2014), 'Utilization of microphone sensors and an active filter for the detection and identification of detonation (knock) in a petrol engine', *Modern Applied Science* **8**(6).

TechTarget (2007), 'Wireless protocols learning guide', https://searchnetworking.techtarget.com/tutorial/Wireless-protocols-learning-guide. [Online; accessed 24 April 2019].

The Open University (2017), 'Methods of data collection and analysis', www.open.edu/openlearncreate/mod/resource/view.php?id=52658. [Online; accessed 20 September 2018].

ThermoFisher Scientific (2010), 'Case study: Decommissioning a destroyer escort', http://www.thermofisher.com.au/Uploads/file/Environmental-Industrial/Process-Monitoring-Industrial-Instruments/

Data-Acquisition/Data-Loggers/DataTaker/CS-0021-B0-Decommissioning-of-a-Destroyer-Escort.pdf. [Online; accessed 04 April 2019].

u-blox (2009), 'Gps antennas rf design considerations for u-blox gps receivers', https://www.u-blox.com/sites/default/files/products/documents/GPS-Antenna_AppNote_%28GPS-X-08014%29.pdf. [Online; accessed 24 April 2019].

University Of Cambridge (2019), 'Soldering safety', https://safety.eng.cam.ac.uk/safe-working/copy_of_soldering-safety. [Online; accessed 05 April 2019].

University of Rhode Island (2019), 'FFT Tutorial', http://www.phys.nsu.ru/cherk/fft.pdf. [Online; accessed 15 July 2019].

Wehrspann, J. (2016), 'New tool logs field data', https://www.farmprogress.com/farm-recordkeeping/new-tool-logs-field-data/. [Online; accessed 04 April 2019].

Wikipedia (2019), 'Google Earth', https://en.wikipedia.org/wiki/Google_Earth. [Online; accessed 22 April 2019].

Yadav, S. K., Tyagi, K., Shah, B. & Kalra, P. K. (2011), 'Audio signature-based condition monitoring of internal combustion engine using fft and correlation approach', *IEEE Transactions on Instrumentation and Measurement* **60**(4), 1217–1226.

Yegin, K. (2007), 'On-vehicle gps antenna measurements', *IEEE Antennas and Wireless Propagation Letters* **6**, 488–491.

Yim, J., Myung, R. & Lee, B. (2017), 'The mobile phone's optimal vibration frequency in mobile environments', *Lecture Notes Computer Science* **4559**, 136–785.

# Appendix A

# Project Specification

ENG 4111/2

# Project Specification

For: **Matthew Fisher**

Title: Real Time Condition Monitoring for Preventative Maintenance Purposes

Major: Electrical and Electronic Engineering

Supervisors: Mark Phythian

Enrolment: ENG4111 - ONC S1, 2019
ENG4112 - ONC S2, 2019

Project Aim: To investigate whether an embedded device comprised of commercially available components can successfully predict fault conditions in a vehicle, which would allow preventative maintenance to be carried out, reducing costs, time and damages associated with reactive maintenance.

**Programme: Version 1, 27th February 2019**

1. Research the background information related to data logging microcontrollers that are used to monitor a similar variety of conditions in vehicles.
2. Design an implementation of a real-time, condition monitoring microcontroller to monitor a set of specific conditions.
3. Build hardware and software to develop sensor packages that are used in monitoring and recording vehicle conditions.
4. Design and build an external antenna for integration with GPS reciever module, including any GPS signal filtering and amplifying.
5. Perform sensor signal analysis and classify normal and abnormal behaviour.
6. Incorporate a Wi-Fi implementation for autonomous data transfer from the vehicle upon return to a geo-located base-station.
7. Demonstrate the effective use of the condition monitoring module by diagnosing a vehicle fault.

*As time and resources permit:*

1. Design and build a Printed Circuit Board (PCB) and a mounting case to incorporate all of the modules in a neat, re-producible and installable package.
2. Incorporate additional sensors into the microcontroller to monitor alternative conditions and fault areas.

Agreed:

Student Name: Matthew J.M. Fisher
Date:

Supervisor Name: Mark Phythian
Date:

## Appendix B

# Risk Assessment

University of Southern Queensland **Offline Version**

# USQ Safety Risk Management System

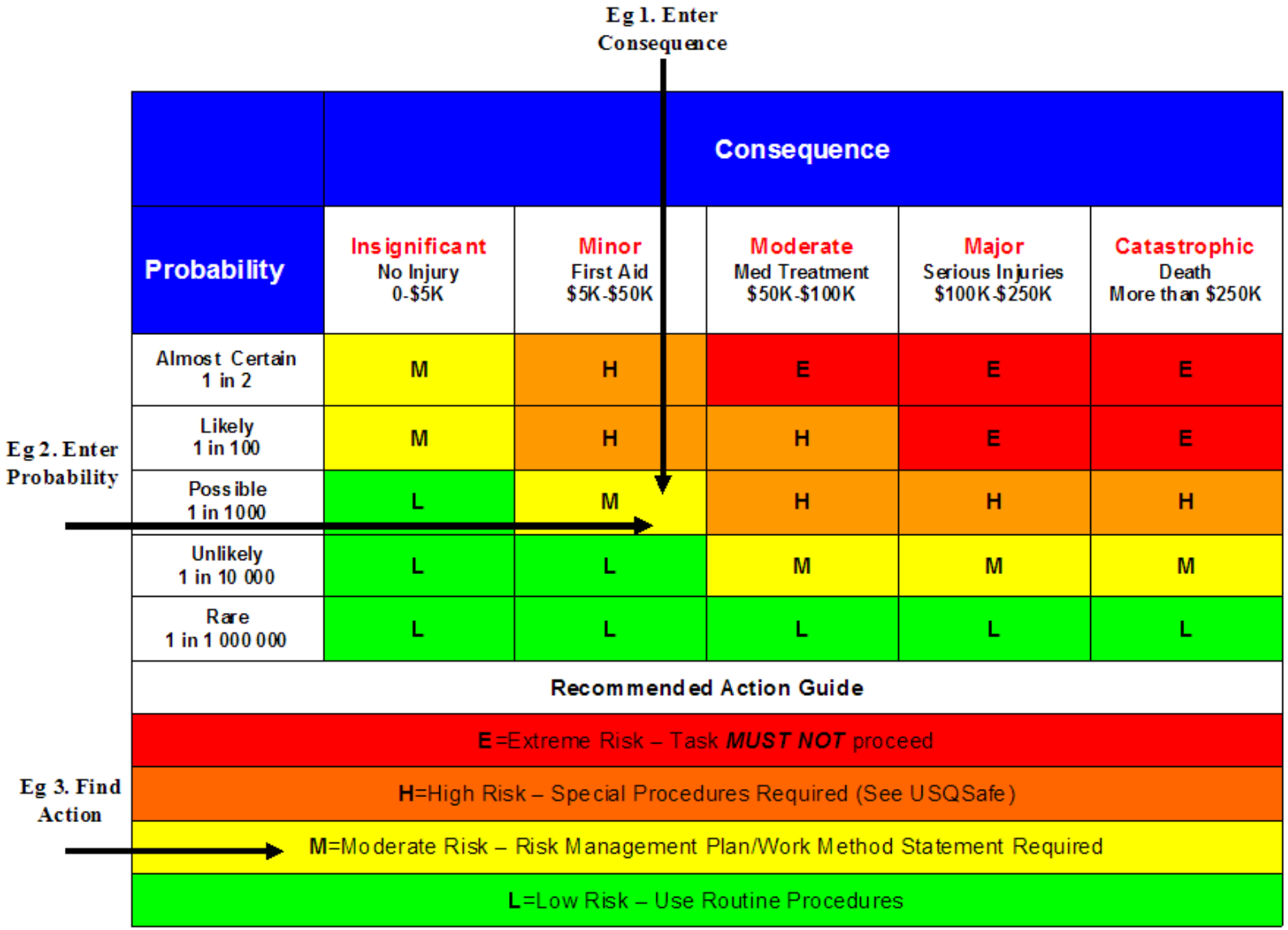
**Note**: This is the offline version of the Safety Risk Management System (SRMS) Risk Management Plan (RMP) and is only to be used for planning and drafting sessions, and when working in remote areas or on field activities. It must be transferred to the online SRMS at the first opportunity.

| Safety Risk Management Plan – Offline Version | | | |
|---|---|---|---|
| Assessment Title: | ERP2019 - ENG4111 & ENG4112 | Assessment Date: | 5/04/2019 |
| Workplace (Division/Faculty/Section): | Faculty of Health, Engineering and Sciences | Review Date:(5 Years Max) | 5/04/2020 |

| Context | | | |
|---|---|---|---|
| **Description:** | | | |
| What is the task/event/purchase/project/procedure? | Engineering Research project for Electrical undergraduate program | | |
| Why is it being conducted? | To fulfil the requirements of an undergraduate degree. | | |
| Where is it being conducted? | University of Southern Queensland Toowoomba Campus | | |
| Course code (if applicable) | ENG4111 & ENG4112 | Chemical name (if applicable) | |

| What other nominal conditions? | |
|---|---|
| Personnel involved | Matthew J.M. Fisher |
| Equipment | AC equipment (Soldering irons, heat guns), vehicles, electronics |
| Environment | USQ Laboratory, Mechanic workshops, home garage, public roads |
| Other | |
| Briefly explain the procedure/process | Design and manufacture a condition monitoring system for predictive maintenance purposes. |

| Assessment Team - who is conducting the assessment? | |
|---|---|
| Assessor(s) | Matthew J.M. Fisher |
| Others consulted: | Mark Phythian |


Eg 1. Enter Consequence
Eg 2. Enter Probability
Eg 3. Find Action
Consequence
Probability
Insignificant
No Injury
0-$5K
Minor
First Aid
$5K-$50K
Moderate
Med Treatment
$50K-$100K
Major
Serious Injuries
$100K-$250K
Catastrophic
Death
More than $250K
Almost Certain
1 in 2
M H E E E
Likely
1 in 100
M H H E E
Possible
1 in 1000
L M H H H
Unlikely
1 in 10 000
L L M M M
Rare
1 in 1 000 000
L L L L L
Recommended Action Guide
E=Extreme Risk – Task MUST NOT proceed
H=High Risk – Special Procedures Required (See USQSafe)
M=Moderate Risk – Risk Management Plan/Work Method Statement Required
L=Low Risk – Use Routine Procedures

| Step 1 (cont) | Step 2 | Step 2a | Step 3 | | | | Step 4 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| *Hazards*: From step 1 or more if identified | *The Risk:* What can happen if exposed to the hazard with existing controls in place? | *Existing Controls:* What are the existing controls that are already in place? | *Risk Assessment:* Consequence x Probability = Risk Level | | | | *Additional controls:* Enter additional controls if required to reduce the risk level | *Risk assessment with additional controls:* | | | |
| | | | Consequence | Probability | Risk Level | ALARP? Yes/no | | Consequence | Probability | Risk Level | ALARP? Yes/no |
| **Example** | | | | | | | | | | | |
| Working in temperatures over 35$^0$ C | Heat stress/heat stroke/exhaustion leading to serious personal injury/death | Regular breaks, chilled water available, loose clothing, fatigue management policy. | catastrophic | possible | high | No | temporary shade shelters, essential tasks only, close supervision, buddy system | catastrophic | unlikely | mod | Yes |
| Soldering Fumes | Exposure to lead and/ or solder flux leading to respiratory issues | PPE and adminstrative controls | Moderate | Rare | Low | Yes | additional self education to be carried out on the risks of soldering | Select a consequence | Select a probability | Select a Risk Level | Yes or No |
| Exposed/ unservicable equipment | Can lead to shocks, heart rate irregularites, falls | Regular electrical checks by maintainers | Major | Rare | Low | No | Identify equipment condition before use | Major | Rare | Low | Yes |
| Soldering Iron | Burns | Serviceable grips, awareness education | Moderate | Rare | Low | Yes | | Select a consequence | Select a probability | Select a Risk Level | Yes or No |
| Engine Bay | Burns | Self education on which components are likely to be heated after operation | Moderate | Rare | Low | No | PPE or admiinistrative controls surrounding the engine operation and test times. | Moderate | Rare | Low | Yes |
| Road Safety | Distraction while in testing phases, on the road. | Experience in driving, awareness of the possibility that components may dislodge from the vehicle. | Major | Rare | Low | Yes | | Select a consequence | Select a probability | Select a Risk Level | Yes or No |
| Manual handling | Back or muscle injuries | Education on correct lifting techniques | Moderate | Rare | Low | Yes | | Select a consequence | Select a probability | Select a Risk Level | Yes or No |
| Slips and trips | Contact or muscle injuries | Environmental awareness | Moderate | Rare | Low | Yes | | Select a consequence | Select a probability | Select a Risk Level | Yes or No |
| Fire | Electrical components may pose a fire hazard if connected poorly, incorrectly. | 3 years of study | Major | Rare | Low | Yes | | Select a consequence | Select a probability | Select a Risk Level | Yes or No |
| Eye strain | Working long hours | Family to rein in the overwork | Insignificant | Possible | Low | Yes | | Select a consequence | Select a probability | Select a Risk Level | Yes or No |
| Sun exposure | heat stroke, heat stress, sunburns | Living in Australia for 30 years | Minor | Unlikely | Low | Yes | | Select a consequence | Select a probability | Select a Risk Level | Yes or No |

| Step 5 - Action Plan (for controls not already in place) | | | |
|---|---|---|---|
| *Additional controls:* | *Resources:* | *Persons responsible:* | *Proposed implementation date:* |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |
| | | | Click here to enter a date. |

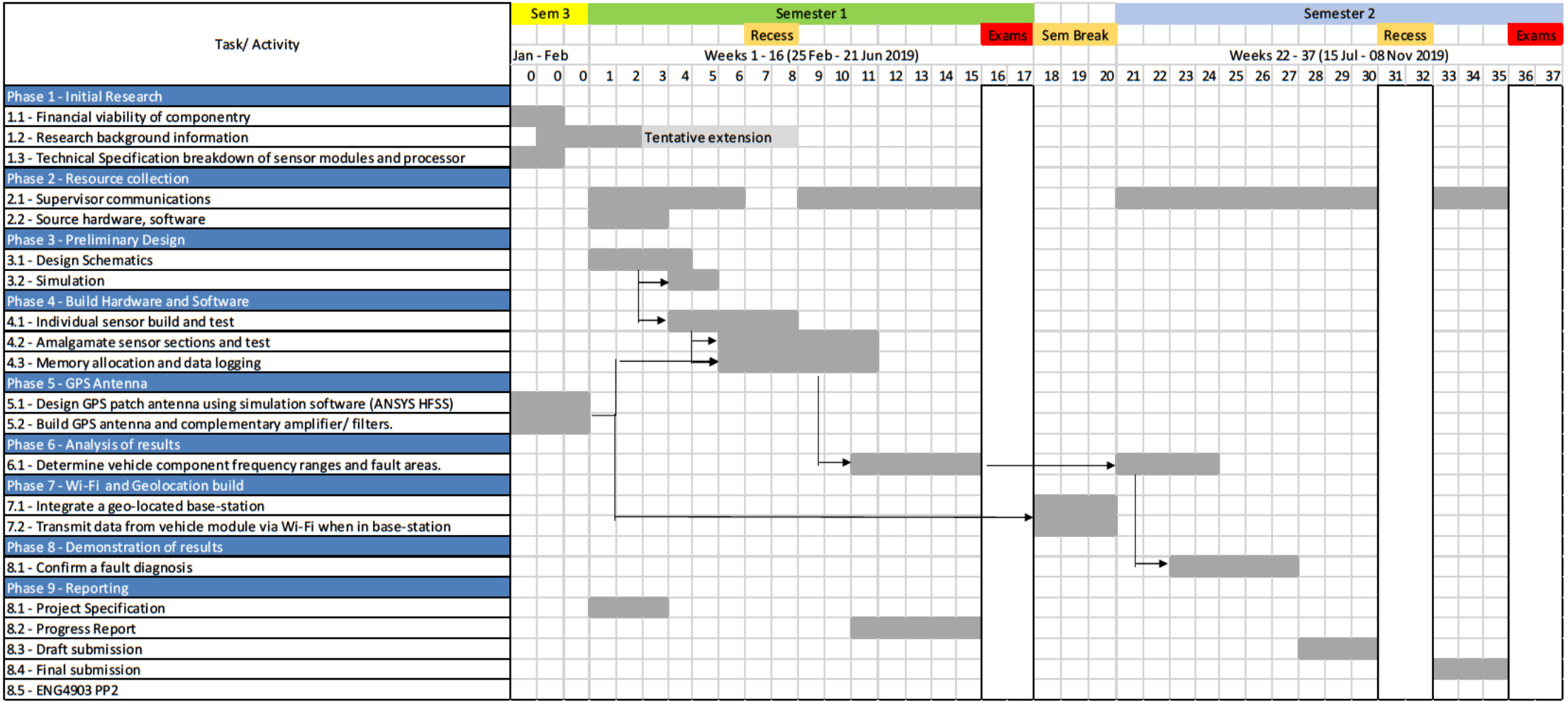
| Step 6 - Approval | | | |
|---|---|---|---|
| Drafter's name: | Matthew J.M. Fisher | Draft date: | 5/04/2019 |
| Drafter's comments: | Nil | | |
| Approver's name: | | Approver's title/position: | |
| Approver's comments: | | | |
| I am satisfied that the risks are as low as reasonably practicable and that the resources required will be provided. | | | |
| Approver's signature: | | Approval date: | Click here to enter a date. |

## Appendix C

# Project Timeline

**Title:** Project Plan

**Description:** Organisational chart for engineering research project undertaken in ENG4111 and ENG4112



# Appendix D

# Resource Requirements

**Purchasable Resources**

| Item | Number Required | Source | Cost per item |
|---|---|---|---|
| Sensor packages | 3 | Core Electronics - Student | $46.40 |
| Microprocessor/ development board | 1 | Core Electronics - Student | $62.87 |
| GPS Receiver | 1 | Core Electronics - Student | $75.89 |
| GPS antenna Backup | 1 | Core Electronics - Student | $35.90 |
| WiFi breakout | 1 | Core Electronics - Student | $47.40 |
| OBDII diagnostics kit | 1 | Core Electronics - Student | 104.47 |
| FTDI breakout | 1 | Core Electronics - Student | $21.07 |
| Manufactured GPS antenna, amplifier/ filter, SMA connectors | 1 | Defence Science and Technology – Student | Free |
| GPS SAW filter/ amplifier | 1 | Nooelec - Student | $41.95 |
| Connecting cables, breadboard | AR (As Required) | Student | $2 - 5 |
| Arduino Software IDE | 1 | Online | Free |
| Signal Processing (MATLAB) | 1 | USQ Bookshop | $130 |
| Word processor | 1 | Microsoft - Student | $20 |
| MicroCap or similar package | 1 | Online | Free |
| Personal Computer | 1 | Student | N/A |
| Circuit assembly tools and spares. This would include measurement, soldering and construction tools, as well as spares like solder, ties, tape, isopropyl, PPE, and wire. | AR | Student/ USQ Laboratory | Free - $50 |
| | | **Approximate cost** | **$600** |

**Borrowed Resources**

| Item | Number Required | Source | Cost per item |
|---|---|---|---|
| Vehicle | 1 - 5 | Student + External (mechanic) | Fuel Costs |
| Office/ laboratory space | 1 | USQ | Free |

**Potential Hurdles**

- There are not a lot of significant hurdles in resource allocation for this project. The office space is not a necessity, although welcome. All purchasable resources have been ordered and secured, pending an operational test to ensure functionality.
- The only significant hurdle will be testing on a faulty vehicle. This is contingent on organising a suitable vehicle and time at a mechanic somewhere in the Toowoomba surrounds. Baseline testing is able to be carried out on personal vehicles owned by the student, before transitioning onto a vehicle with a fault.

## Appendix E

# Mahalanobis Distance Raw Data

| Hyundai Getz 2006 | | | | |
|---|---|---|---|---|
| System status | Distance Measure | | | |
| | Normal | Timing | Fuel | Result |
| | 5.6424 | 9.4815 | 1.772 | Fuel |
| | 10.7187 | 13.1969 | 1.7851 | Fuel |
| | 8.9133 | 8.3374 | 2.635 | Fuel |
| Fuel | 12.3422 | 10.921 | 3.7192 | Fuel |
| | 3.9068 | 1.117 | 12.8334 | Timing |
| | 7.05 | 1.5202 | 15.1393 | Timing |
| | 9.6929 | 2.0164 | 17.5071 | Timing |
| Timing | 6.7932 | 1.6766 | 12.5828 | Timing |
| | 1.3818 | 3.6061 | 11.4271 | Normal |
| | 1.8068 | 9.6449 | 25.7465 | Normal |
| Normal | 1.8633 | 9.9587 | 17.8725 | Normal |
| 1 | 1.6852 | 4.0482 | 7.0273 | Normal |
| | 3.536 | 5.0997 | 9.7329 | Normal |
| | 11.6426 | 15.4747 | 17.6234 | Normal |
| Normal | 10.9801 | 12.7278 | 14.5752 | Normal |
| 2 | 8.0233 | 6.5527 | 12.7185 | Timing |

| BMW N47 | | | |
|---|---|---|---|
| System status | Distance Measure | | |
| | Normal | Timing | Result |
| | 8.8724 | 1.3478 | Timing |
| | 8.1028 | 2.4843 | Timing |
| | 6.8835 | 1.0713 | Timing |
| Timing | 8.4852 | 18.6259 | Normal |
| | 4.1871 | 6.89 | Normal |
| | 1.58 | 8.87 | Normal |
| | 4.61 | 11.55 | Normal |
| Normal 1 | 3.08 | 5.08 | Normal |

| All Baseline Data Tests | | | | | | |
|---|---|---|---|---|---|---|
| | Distance Measure | | | | | |
| System status | BMW Healthy | BMW Timing | GETZ FUEL | GETZ Normal | GETZ Timing | Subaru Healthy |
| BMW Healthy | 0.74 | 2.94 | 14.51 | 13.62 | 14.31 | 19.93 |
| BMW Timing | 4.65 | 0.61 | 25.99 | 32.67 | 22.69 | 60.2 |
| GETZ Fuel | 136.23 | 189.2 | 0.72 | 3.01 | 3.76 | 10.17 |
| Getz Normal | 153.58 | 213.85 | 3.67 | 0.92 | 2.54 | 10.99 |
| Getz Timing | 130.59 | 177.66 | 3.81 | 2.62 | 0.96 | 8.76 |
| Subaru Healthy | 123.06 | 178.02 | 10.14 | 11.63 | 11.65 | 0.5 |
| unknown Knocking (1) | 21.88 | 24.86 | 35.36 | 36.78 | 40.28 | 31.02 |
| unknown Knocking (2) | 127.24 | 75.66 | 124.45 | 131.79 | 100.98 | 200.95 |
| unknown Knocking (3) | 68.14 | 71.14 | 169.13 | 140.1 | 105.2 | 182.95 |
| unknown Knocking (4) | 698.26 | 368.73 | 216.91 | 158.86 | 127.62 | 284.47 |
| unknown Knocking (5) | 7.48 | 10.79 | 58.26 | 44.81 | 32.26 | 56.82 |
| unknown Knocking (6) | 2.69 | 5.37 | 15.88 | 14.07 | 12.44 | 28.51 |
| unknown Knocking (7) | 83.53 | 69.26 | 237.65 | 212.8 | 337.69 | 171.48 |
| unknown Knocking (8) | 17.95 | 14.46 | 72.6 | 50.63 | 52.08 | 86.43 |
| Audi before a service | 46.99 | 50.81 | 117.48 | 107.6 | 92.2 | 142.23 |
| Audi after a service | 67.76 | 42.22 | 97.16 | 84.73 | 69.23 | 139.5 |
| E39 Camshaft fault | 10.96 | 17.61 | 24.43 | 27.86 | 32.76 | 37.53 |
| E39 Healthy engine | 16.27 | 27.31 | 31.28 | 33.74 | 39.12 | 37.86 |
| Ford Focus before oil chg | 21.11 | 16.13 | 47.77 | 50.31 | 45.42 | 60.81 |
| Ford Focus after oil chg | 25.67 | 20.55 | 51.6 | 53.9 | 51.82 | 59.75 |
| Nissan Frontier Fan clutch | 26.93 | 38.28 | 134.54 | 160.18 | 125.67 | 283.72 |
| Nissan Frontier Healthy | 62.13 | 101.41 | 218.05 | 231.39 | 173.92 | 360.72 |
| before SWEDOL additive | 9.6 | 11.07 | 35.8 | 32.6 | 27.57 | 22.69 |
| after SWEDOL additive | 3.91 | 5.87 | 27.53 | 24.63 | 20.4 | 17.53 |

# Appendix F

# Embedded C: Teensy 3.6 Source code

```
/* Project: Condition Monitoring for Predictive Maintenance
   Purposes
 * Program Name: Teensy 3.6 Arduino Code
 * Author: Matthew J.M. Fisher
 * Date Created: 02 Feb 2019
 * Revision: 15
 * Purpose: To embed code in the Teensy 3.6 micro-controller and
 *            control a variety of sensor arrays.
 * Acknowledgements : Teensy Audio/ SD (open source) - Paul
    Stoffregen
 *                  : Reverse Geo-Locate (open source) - Kenton
    Harris
 *                  : WiFi (open source) - Starting Electronics
 */

//-----GPS includes and defines
    ----------------------------------------
#include <Adafruit_GPS.h>
#include <math.h>
#define mySerial Serial1
IntervalTimer GPSTimer;
Adafruit_GPS GPS(&mySerial);

// GPS echo should be true to listen to raw NMEA sequences
#define GPSECHO  true

//use time library in conjunction with GPS to set system clock
#include <TimeLib.h>
const int offset = +10;   // Brisbane time

const float degree2radian = 0.01745329251994;
const float radiusEarth = 6371000.0;
float dist_to_target = 3000;
String me;

//set GPS coordinates for the home base (USQ, Toowoomba)
String you = "S27␣36.058,␣E151␣55.905";
//specify the range from home base to begin to setup WiFi -
   circular range
float outerRange = 30;
```

```
//-----Audio / SD includes and defines
   ------------------------------------
#include <Audio.h>
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include <SerialFlash.h>

// GUItool: begin automatically generated code
AudioInputAnalog         adc1;
AudioRecordQueue         queue1;
AudioConnection          patchCord1(adc1, queue1);
// GUItool: end automatically generated code

// Use these with the Teensy 3.5 & 3.6 SD card
#define SD_CS    BUILTIN_SDCARD
#define SD_MOSI  11
#define SD_SCK   13

unsigned long currentMillis;
unsigned long startMillis;
const unsigned long period = 1000;
int count = 0;

// Remember which mode we are in
int mode = 0;  // 0 = stopped, 1 = recording, 2 = playing

//-----Accelerometer includes and defines
   ---------------------------------
//#include <Adafruit_LIS3DH.h>
//#include <Adafruit_Sensor.h>
//
//IntervalTimer ACCTimer;
//
//// Used for software SPI
//#define LIS_CLK 14
//#define LIS_MISO 12
//#define LIS_MOSI 11
//// Used for hardware & software SPI
//#define LIS_CS 10
//
////SPI
//Adafruit_LIS3DH lis = Adafruit_LIS3DH(LIS_CS, LIS_MOSI,
//                                      LIS_MISO, LIS_CLK);

//---------Filename includes and defines
   -----------------------------------

// Create the files where data is recorded
File frec;
//File accel;
File gps;
File WIFIFile;
//char filename[] = "ACCELE00.TXT";
char audioFile[] = "AUDIOS00.RAW";
char gpsFile[] = "GPSDAT00.TXT";

//int accelerometerXBuffer[1000];
//int accelerometerYBuffer[1000];
//int accelerometerZBuffer[1000];
//int i = 0;
//int j = 0;

  //-----WiFi includes and defines
     ---------------------------------------
#include <WiFi101.h>
#include "arduino_secrets.h"
```

```
#define buffer_size   20

char name_id[] = SECRET_NAME_ID;
char password[] = SECRET_PASSWORD;

int status = WL_IDLE_STATUS;
WiFiServer server(80);

boolean runalready = false;
boolean wifiValue = false;

char req_HTTP[buffer_size] = {0};
char req_index = 0;

//------------Function prototypes
   ------------------------------------------
//void readACC();
void readGPS();

void startRecording();
void continueRecording();
void stopRecording();
//void startAccRecording();
void recordGPS();

void digitalClockDisplay();

String gps2text (String lat_1, float lat_2,
                   String lon_1, float lon_2);
float haversine_formula (float lat_1, float lon_1,
                         float lat_2, float lon_2);
float text2long (String inString);
float text2lat (String inString);
String integer2FW (int in, int length_n);

void macAddress_1(byte mac_byte[]);
void WiFiStatus();
void clearString(char *stringptr, char length_1);
char stringContain(char *stringptr, char *sfind);

//-----------------SETUP function
   ------------------------------------------
void setup()
{
  //-----------------------Accelerometer setup
     ------------------------------
//  #ifndef ESP8266
//  // will pause Teensy until serial console opens -
   accelerometer DEFINE
//  while (!Serial);
//  #endif
//
//  Serial.begin(115200);
//  //Test sequence for Serial Monitor
//  Serial.println("Hello world!");
//
//  if (! lis.begin(0x18))
//    {
//      Serial.println("LIS3DH Couldnt start");
//      //get stuck on purpose when the teensy cant find the
   accelerometer
//      while (1);
//    }
//  Serial.println("Accelerometer found!");
//
//  lis.setRange(LIS3DH_RANGE_16_G);
//  // 2, 4, 8 or 16 G
//
//  lis.setDataRate(LIS3DH_DATARATE_LOWPOWER_5KHZ);
```

```
//  //LIS3DH_DATARATE_400_HZ  //LIS3DH_DATARATE_LOWPOWER_1K6HZ
//
//  Serial.print("Range: "); Serial.print(2 << lis.getRange());
//  Serial.println("G");
//  Serial.print("Data rate = "); Serial.println(lis.getDataRate
   ());

  //----------------GPS Setup
     ------------------------------------------------

  GPS.begin(9600);

  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
  // 1 Hz update rate
  GPS.sendCommand(PGCMD_ANTENNA);

  GPSTimer.begin(readGPS, 1000);
  //timer interrupt value in microseconds

  delay(100);
  Serial.print("GPS␣Firmware␣Version:␣␣");
  // Ask for firmware version
  Serial.println(PMTK_Q_RELEASE);
  Serial.println("GPS␣Setup␣complete␣");

  //----------------------AUDIO SETUP
     -------------------------------------
  // The record queue uses this memory to buffer incoming audio.
  AudioMemory(120);

  pinMode(A2, INPUT);
  delay(1000);
  startMillis = millis();

  //-----------------------SD Card setup
     ---------------------------------
  // Initialize the SD card
  SPI.setMOSI(SD_MOSI);
  SPI.setSCK(SD_SCK);

  if (!(SD.begin(SD_CS)))
  {
    Serial.println("Unable␣to␣access␣the␣SD␣card");
    while (1);
  }
  Serial.println("SD␣Card␣Setup␣complete␣");

  //-------------------Filename setup
     ---------------------------------------

  // create a new file for the Accelerometer
  //  for (uint8_t i = 0; i < 100; i++)
  //  {
  //    filename[6] = i/10 + '0';
  //    filename[7] = i%10 + '0';
  //    if (! SD.exists(filename))
  //    {
  //      // only open a new file if it doesn't exist
  //      accel = SD.open(filename, FILE_WRITE);
  //      break;  // leave the loop!
  //    }
  //  }

  // create a new file for the Audio
  for (uint8_t i = 0; i < 100; i++)
  {
    audioFile[6] = i/10 + '0';
    audioFile[7] = i%10 + '0';
    if (! SD.exists(audioFile))
```

```
    {
      // only open a new file if it doesn't exist
      frec = SD.open(audioFile, FILE_WRITE);
      break;  // leave the loop!
    }
  }

  // create a new file for the GPS
  for (uint8_t i = 0; i < 100; i++)
  {
    gpsFile[6] = i/10 + '0';
    gpsFile[7] = i%10 + '0';
    if (! SD.exists(gpsFile))
    {
      // only open a new file if it doesn't exist
      gps = SD.open(gpsFile, FILE_WRITE);
      break;  // leave the loop!
    }
  }

  //  if (! accel)
  //  {
  //     //if it cant find the accelerometer file
  //    Serial.println("couldnt create accelerometer file");
  //  }
  if (! frec)
  {
    //if it cant find the audio file
    Serial.println("couldnt create audio file");
  }
    if (! gps)
  {
      //if it cant find the GPS file
    Serial.println("couldnt create gps file");
  }

  //  Serial.print("Logging to: ");
  //  Serial.println(filename);

  Serial.print("Logging to: ");
  Serial.println(audioFile);

  Serial.print("Logging to: ");
  Serial.println(gpsFile);
}


//-------------------MAIN LOOP
   ----------------------------------------------

//put the current time in the timer variable.
uint32_t timer = millis();

// run over and over again
void loop()
{
  if(wifiValue == false)
  {
      // if a sentence is received, we can process it
      if (GPS.newNMEAreceived())
      {
        // this also sets the newNMEAreceived() flag to false
        if (!GPS.parse(GPS.lastNMEA()))
        return;
      }

      if(mode==0)
      {
        //call the start audio recording function
```

```
  startRecording();
}

if(mode==1)
{
  //if the mode is set properly, continue audio recording
  continueRecording();
  //start the accelerometer recording using function call
  //startAccRecording();

  //if the accelerometer buffer contains 500 values
  //    if(i > 499)
   //  {
    //stop audio for a small portion, close SD card file
    stopRecording();
    //start audio buffering (doesnt use SD card)
    startRecording();
    //write the Accelerometer values to SD
    //startAccSDWrite();
    //write the GPS values to SD
    recordGPS();
//  }
}

// reset timer if it wraps around
if (timer > millis())
{
  Serial.println("Timer␣reset");
  timer = millis();
}

// approximately every 1 second or so,
//    print out the current stats (debug)
if (millis() - timer > 1000)
{
  timer = millis();

  Serial.println();
  Serial.print("Fix:␣");
  Serial.print((int)GPS.fix);
  Serial.print("␣quality:␣");
  Serial.println((int)GPS.fixquality);

  setTime(GPS.hour, GPS.minute, GPS.seconds,
                GPS.day, GPS.month, GPS.year);

  adjustTime(offset * SECS_PER_HOUR);
  digitalClockDisplay();

  if (GPS.fix)
  {
    me = gps2text((String)GPS.lat, GPS.latitude,
              (String)GPS.lon, GPS.longitude);
    Serial.print("Current␣GPS␣Location:␣");
    Serial.println(me);
    dist_to_target = haversine_formula(text2lat(me),
                text2long(me), text2lat(you), text2long(
                   you));
    Serial.print("Distance␣to␣target:␣");
    Serial.println(dist_to_target);
    Serial.println();

    if(dist_to_target < outerRange)
    {
      wifiValue = true;
    }
  }
}
```

```
    }
  if(wifiValue == true)
  {
    if (runalready == false)
    {
      //------------------------WiFi setup
         ---------------------------------

      WiFi.setPins(8,7,4);

      Serial.begin(9600);
      Serial.println("Condition␣Monitoring␣Access␣Point␣Server")
         ;

      //ensure WiFI chip is connected
      if (WiFi.status() == WL_NO_SHIELD)
      {
        Serial.println("WiFi␣shield␣not␣present");
        while (true); //stop here
      }

      // default local IP address is 192.168.1.1
      // Change with - WiFi.config(IPAddress(10, 0, 0, 1));

      //Serial.print("Creating AP Server named: ");
      //Serial.println(name_id);

      // Create the open network.
      status = WiFi.beginAP(name_id);
      //If not listening
      if (status != WL_AP_LISTENING)
      {
        Serial.println("Creating␣access␣point␣failed");
        while (true);
      }

      // delay of 10 seconds for connection
      delay(10000);

      server.begin();
      WiFiStatus();
      runalready = true;

      //setup is complete - boolean runAlready to true
  }
  //------------------------ Server and WiFi UPLOAD
     -------------------------

  if (status != WiFi.status())
  {
    //update status if neccessary
    status = WiFi.status();

    //if the status is connected
    if (status == WL_AP_CONNECTED)
      {
        byte address_AP[6];
        Serial.print("Device␣connected␣-␣Its␣MAC␣address␣is:␣");
        WiFi.APClientMacAddress(address_AP);
        macAddress_1(address_AP);
      }
    else
      {
        //now disconnected, in listening mode
        Serial.println("Device␣disconnected␣from␣AP");
      }
  }

  //listen for incoming
```

```
WiFiClient client = server.available();

//if there is a client
if (client)
{
  boolean currentLine = true;
  Serial.println("You␣have␣a␣client!!");
  Serial.println();
  //create variable to hold incoming data
  //String currentLineIN = "";

  while (client.connected())
  {
    if (client.available())
    {
      char read_char = client.read();

      if (req_index < (buffer_size - 1))
      {
          req_HTTP[req_index] = read_char;
          req_index++;
      }

      //debugging - show char read
      Serial.write(read_char);

      if (read_char == '\n' && currentLine)
      {
             //Open the index page first!
             if (stringContain(req_HTTP, "GET␣/␣")
                           || stringContain(req_HTTP, "GET␣/
                              index.htm"))
             {
                  client.println("HTTP/1.1␣200␣OK");
                  client.println("Content-Type:␣text/html");
                  client.println("Connnection:␣close");
                  client.println();
                  WIFIFile = SD.open("index.htm");
              }
              //Open audio page if requested
              else if (stringContain(req_HTTP, "GET␣/audpage.
                 htm"))
              {
                  client.println("HTTP/1.1␣200␣OK");
                  client.println("Content-Type:␣text/html");
                  client.println("Connnection:␣close");
                  client.println();
                  WIFIFile = SD.open("audpage.htm");
              }
              //Open GPS page if requested
              else if (stringContain(req_HTTP, "GET␣/gpspage.
                 htm"))
              {
                  client.println("HTTP/1.1␣200␣OK");
                  client.println("Content-Type:␣text/html");
                  client.println("Connnection:␣close");
                  client.println();
                  WIFIFile = SD.open("gpspage.htm");
              }
              //Display Image on index page
              else if (stringContain(req_HTTP, "GET␣/CONDMON.
                 jpg"))
              {
                  WIFIFile = SD.open("CONDMON.jpg");
                  if (WIFIFile)
                  {
                      client.println("HTTP/1.1␣200␣OK");
```

```
            client.println();
        }
    }
    //Download the most recent audio file
    else if (stringContain(req_HTTP, "GET␣/Audio1"))
    {
        client.println("HTTP/1.1␣200␣OK");
        //use text-csv file format
        client.println("Content-Type:␣text/csv");
        client.println("Connnection:␣close");
        client.println();

        File recentAudio = SD.open(audioFile,
            FILE_READ);
        if (recentAudio)
        {
            byte cBuf[64];
            int cCount = 0;

            while(recentAudio.available())
            {
              cBuf[cCount] = recentAudio.read();
              cCount++;

              if(cCount > 63)
              {
                client.write(cBuf,64);
                cCount = 0;
              }
            }
            if(cCount > 0) client.write(cBuf, cCount
                );
            recentAudio.close();
        }
    }
    //Download the most recent GPS file
    else if (stringContain(req_HTTP, "GET␣/GPS1"))
    {
        client.println("HTTP/1.1␣200␣OK");
        //use text-csv file format
        client.println("Content-Type:␣text/csv");
        client.println("Connnection:␣close");
        client.println();

        File recentGPS = SD.open(gpsFile, FILE_READ)
            ;
        if (recentGPS)
        {
          byte cBuf[64];
          int cCount = 0;

          while(recentGPS.available())
          {
            cBuf[cCount] = recentGPS.read();
            cCount++;

            if(cCount > 63)
            {
              client.write(cBuf,64);
              cCount = 0;
            }
          }
          if(cCount > 0) client.write(cBuf, cCount);
          recentGPS.close();
        }
    }

    //if the WIFI file exists, send to client
    if (WIFIFile)
```

```
                {
                    while(WIFIFile.available())
                    {
                        client.write(WIFIFile.read());
                    }

                    WIFIFile.close();
                }

                //reset and clear the string value
                req_index = 0;
                clearString(req_HTTP, buffer_size);
                break;
            }

            if (read_char == '\n')
            {
                currentLine = true;
            }

            else if (read_char != '\r')
            {
                currentLine = false;
            }
          }
        }

        // close connection after a short delay
        delay(1);
        client.stop();
        Serial.println("Client disconnected :-( ");
      }
    }
}

//------------------------------FUNCTIONS
    ---------------------------------

void readGPS()
//function - Capture GPS values from Sensor
{
  GPS.read();

  /*Serial.print("Location: ");
  Serial.print(GPS.latitude, 4); Serial.print(GPS.lat);
  Serial.print(", ");
  Serial.print(GPS.longitude, 4); Serial.println(GPS.lon);
  Serial.print("Location in degrees: ");
  Serial.print(GPS.latitudeDegrees, 4);
  Serial.print(", ");
  Serial.println(GPS.longitudeDegrees, 4);

  Serial.print("Speed (knots): "); Serial.println(GPS.speed);
  Serial.print("Angle: "); Serial.println(GPS.angle);
  Serial.print("Altitude: "); Serial.println(GPS.altitude);
  Serial.print("Satellites: "); Serial.println((int)GPS.
     satellites);*/
}


//void startAccRecording()
//function - capture Accelerometer values from sensor
//{
//  lis.read();
//  sensors_event_t event;
//  lis.getEvent(&event);
//
//  accelerometerXBuffer[i] =  event.acceleration.x;
//  accelerometerYBuffer[i] =  event.acceleration.y;
```

```
//  accelerometerZBuffer[i] =  event.acceleration.z;
//  //Serial.println(accelerometerXBuffer[i]);
//  i++;
//}

//void startAccSDWrite()
//function - write Accelerometer values to SD
//{
//  accel = SD.open(filename, FILE_WRITE);
//  if (accel)
//  {
//    // write to file
//    Serial.println("writing to accelerometer txt file"); //
   debugging
//
//    for(int count = 0; count < i; count++)
//    {
//        accel.print(hour());
//        accel.print(":");
//        accel.print(minute());
//        accel.print(":");
//        accel.print(second());
//        accel.print(",");
//
//        accel.print(accelerometerXBuffer[count]);
//        accel.print(";");
//        accel.print(accelerometerYBuffer[count]);
//        accel.print(";");
//        accel.print(accelerometerZBuffer[count]);
//        accel.println();
//
//        if(count % 2 == 0 )
//        {
//            startAccRecording();
//        }
//
//    }
//
//    Serial.print("buffer value: ");
//    Serial.println(i);
//    Serial.println("Accel Done"); //debugging
//
//    i = 0;
//    //close the SD file
//    accel.close();
//
//
//  }
//  else
//  {
//    // if the file didn't open, print an error:
//    Serial.println("error opening test.txt");
//  }
//}

void recordGPS()
//function - record GPS values on SD
{
  gps = SD.open(gpsFile, FILE_WRITE);

  if (gps)
  {
    // write to file
    //Serial.println("writing to GPS txt file"); //debugging

    //lat and long
    gps.print(GPS.latitudeDegrees, 4);
```

```
    gps.print(",");
    gps.print(GPS.longitudeDegrees, 4);
    gps.print('','');

    //Serial.print(",");
    gps.print(day());
    gps.print(":");
    gps.print(month());
    gps.print(":");
    gps.print(year());
    gps.println();

    //Serial.println("GPS Done"); //debugging
    gps.close();
  }
}

void startRecording()
//function - prepare audio file and begin capturing audio
{
  //Serial.println("startRecording");
  frec = SD.open(audioFile, FILE_WRITE);

  //begin capturing incoming audio in the buffer
  queue1.begin();
  mode = 1;

}

void continueRecording()
//function - keep buffering and recording audio to SD
{
  if(!frec)
  {
     frec = SD.open(audioFile, FILE_WRITE);
  }

  if (queue1.available() >= 2)
  {
    byte buffer[512];
    //copy two bytes into the buffer
    memcpy(buffer, queue1.readBuffer(), 256);

    //free the buffer, audio library GUI
    queue1.freeBuffer();
    memcpy(buffer+256, queue1.readBuffer(), 256);
    queue1.freeBuffer();

    //write to the SD card
    frec.write(buffer, 512);
  }
}

void stopRecording()
//function - free audio buffer and close audio file
{
  //Serial.println("stopRecording");
  queue1.end();

  while (queue1.available() > 0)
  {
    //write everything in the buffer to the SD
    frec.write((byte*)queue1.readBuffer(), 256);

    //Free the buffer
    queue1.freeBuffer();
  }
  //close the file!!
  frec.close();
```

```
}

void digitalClockDisplay()
// digital clock fucntion - display of the time
{
  Serial.print("System␣Clock␣Set␣to␣AEST:␣");
  Serial.print(hour());
  Serial.print(":");
  Serial.print(minute());
  Serial.print(":");
  Serial.print(second());
  Serial.print("␣␣--␣␣");

  Serial.print(day());
  Serial.print(",");
  Serial.print(month());
  Serial.print(",");
  Serial.print(year());
  Serial.println();
}

String integer2FW (int in, int length_n)
//function for defining fixed width
{
  String out = (String) in;
  while (out.length() < length_n)
  {
    out = "0" + out;
  }
  return out;
}

String gps2text (String lat_1, float lat_2, String lon_1, float
    lon_2)
//function for converting lat/long values into a string
{
  int degree = (int) lat_2 / 100;
  int dec_minutes = (int) lat_2 % 100;
  int dec_minutes_minutes = (int) round(1000 * (lat_2 - floor(
     lat_2)));
  String gps2latitude = lat_1 + integer2FW(degree, 2)
                      + "␣" + integer2FW(dec_minutes, 2)
                      + "." + integer2FW(dec_minutes_minutes, 3)
                        ;

  degree = (int) lon_2/100;
  dec_minutes = (int) lon_2 % 100;
  dec_minutes_minutes = (int) round(1000 * (lon_2 - floor(lon_2)
     ));
  String gps2longitude = lon_1 + integer2FW(degree, 3)
                      + "␣" + integer2FW(dec_minutes, 2)
                      + "." + integer2FW(dec_minutes_minutes, 3)
                        ;

  String exitString = gps2latitude + ",␣" + gps2longitude;

  return exitString;
};


float text2lat (String inString)
//function for converting latitude string into float values
{
  float latitude = ((inString.charAt(1) - '0') * 10.00)
                   + (inString.charAt(2) - '0') * 1.00
                   + ((inString.charAt(4) - '0') / 6.00)
                   + ((inString.charAt(5) - '0') / 60.00)
                   + ((inString.charAt(7) - '0') / 600.00)
```

```
                    + ((inString.charAt(8) - '0') / 6000.00)
                    + ((inString.charAt(9) - '0') / 60000.00);

  latitude *= degree2radian;

  if (inString.charAt(0) == 'S')
    latitude *= -1;

  return latitude;
};

float text2long (String inString)
//function for converting longitude string into float values
{
  float longitude = ((inString.charAt(13) - '0') * 100.00)
                    + ((inString.charAt(14) - '0') * 10.00)
                    + (inString.charAt(15) - '0') * 1.00
                    + ((inString.charAt(17) - '0') / 6.00)
                    + ((inString.charAt(18) - '0') / 60.00)
                    + ((inString.charAt(20) - '0') / 600.00)
                    + ((inString.charAt(21) - '0') / 6000.00)
                    + ((inString.charAt(22) - '0') / 60000.00);

  longitude *= degree2radian;

  if (inString.charAt(12) == 'W')
    longitude *= -1;

  return longitude;
};


float haversine_formula (float lat_1, float lon_1, float lat_2,
   float lon_2)
//haversine function - determines the great-circle distance
   between
//  two points on a sphere given their longitudes and latitudes
{
  float calc_h = sq( (sin((lat_1 - lat_2) / 2.00)))
                + ( cos(lat_1) * cos(lat_2)
                * sq( (sin((lon_1 - lon_2) / 2.00))));

  float calc_out = 2.00 * radiusEarth * asin(sqrt(calc_h));

  return calc_out;
};

void WiFiStatus()
//function for printing WiFi status
{
  Serial.print("Server␣ID␣is:␣");
  Serial.println(WiFi.SSID());

  IPAddress ip_address = WiFi.localIP();
  Serial.print("Server␣IP␣Address:␣");
  Serial.println(ip_address);

  long strength = WiFi.RSSI();
  Serial.print("Signal␣Strength␣(RSSI):");
  Serial.print(strength);
  Serial.println("␣dBm");

  Serial.print("Open␣a␣browser␣to␣http://");
  Serial.println(ip_address);

}

void macAddress_1(byte mac_byte[])
//function for printing connected MAC address
{
```

```
  for (int count = 5; count >= 0; count--)
  {
    if (mac_byte[count] < 16)
    {
      Serial.print("0");
    }

    Serial.print(mac_byte[count], HEX);
    if (count > 0)
    {
      Serial.print(":");
    }
  }
  Serial.println();
}

char stringContain(char *stringptr, char *findstring)
//function for searching string values
{
    char foundMe = 0;
    char index = 0;
    char length_1;

    length_1 = strlen(stringptr);

    if (strlen(findstring) > length_1)
    {
        return 0;
    }
    while (index < length_1)
    {
        if (stringptr[index] == findstring[foundMe])
        {
            foundMe++;
            if (strlen(findstring) == foundMe)
            {
                return 1;
            }
        }
        else
        {
            foundMe = 0;
        }
        index++;
    }

    return 0;
}

void clearString(char *stringptr, char length_1)
//function for clearing the string values (WiFi)
{
    for (int count = 0; count < length_1; count++)
    {
        stringptr[count] = 0;
    }
}
```

# Appendix G

# MATLAB: Audio FFT and Spectrogram analysis

Listing G.1: A MATLAB function for analyzing the frequency content of raw audio data.

```
%-------------------------------------------------------------------------

clear all
clc
close all
%-------------------------------------------------------------------------

[name, path] = uigetfile('.wav','Select WAV File to Load');
addpath(path);

% load an audio file
[Vec_0, Fs_0] = audioread(name);
%first channel only if applicable (should only be 1 channel anyway)
Vec_0 = Vec_0(:, 1);

% Total number of sound samples
NumOfSamples = length(Vec_0);

sampleDuration = NumOfSamples/Fs_0;
timeDuration = 0:sampleDuration/(NumOfSamples-1):sampleDuration;

%Calculate Basic FFT for separate plot comparisons
FFT_0 = fft(Vec_0 / length(Vec_0));
freq_0 = (0:length(FFT_0)-1)* Fs_0 / length(FFT_0);
FFT_1 = FFT_0;

%remove second half of FFT calculations (rect window) for complex values
FFT_1((length(FFT_1)/2):end) = 0;
```

```
%Compute spectrogram values for plot
window_length = 1024;
step_size = window_length / 4;
no_of_fft_points = 4096;

%apply the Hamming window function
window = hamming_window(window_length);

%function call to do Short FT
[magnitude, freq, time] = ...
    short_FT(Vec_0, window, step_size, no_of_fft_points, Fs_0);

%Spectogram plot is Time, Freq, Magnitude in surf plot
%Begin with amplification of the window
Amplify_wind = sum(window)/window_length;
%scaling the amplitude
magnitude = abs(magnitude) / window_length / Amplify_wind;
%put magnitude in dB
magnitude = 20 * log10(magnitude + 1e-6);

%Plot the samples
figure(1)
plot(Vec_0);
title('Time Domain');
xlabel('Samples');
ylabel('Magnitude');

%plot the freq
figure(2)
plot(freq_0,abs(FFT_1))
title('Frequency Domain')
xlabel('Frequency (Hz)');
ylabel('Magnitude');

%spectrogram plot
figure(3)
surf(time, freq, magnitude);
shading interp;
axis tight;
view(0, 90);
xlabel('Time (s)');
ylabel('Frequency (Hz)');
title('Spectrogram');
hcol = colorbar;
ylabel(hcol, 'Magnitude, dB');


function [matrix_results, frequency, time] = ...
    short_FT(raw_data, window, step_size, no_of_fft_points, Fs)

    raw_data = raw_data(:);
    %signal length
```

```
    data_length = length(raw_data);
    %window length
    window_length = length(window);

    unique_points = ceil((1 + no_of_fft_points) / 2);
    %number of frames
    No_of_frames = 1 + fix((data_length-window_length) / step_size);

    %preallocate the matrix
    matrix_results = zeros(unique_points, No_of_frames);

    for i = 0 : No_of_frames - 1
        %windowing function
        current_window = ...
            raw_data(1 + i * step_size : window_length + i * step_size) ...
            .* window;
        %FFT function
        current_FFT = fft(current_window, no_of_fft_points);
        %draw out the results
        matrix_results(: , 1 + i) = current_FFT(1 : unique_points);
    end

    time = ...
        (window_length / 2 : step_size : ...
        window_length / 2 + (No_of_frames-1) * step_size) / Fs;
    frequency = (0 : unique_points - 1) * Fs / no_of_fft_points;
end


function output = hamming_window(input)
    %Hamming window function - see vibration script
    locations = pi / input *(( 1 - input): 2: 0)';
    first_wind = 0.54 + 0.46 * cos(locations);
    output = [first_wind; first_wind(floor(input / 2): -1 : 1)];
end
```

# Appendix H

# MATLAB: Vibration FFT and Spectrogram analysis

Listing H.1: A MATLAB function for analyzing the frequency content of raw accelerometer data.

```
%-------------------------------------------------------------------------
close all
clear all
clc
%-------------------------------------------------------------------------

[name, path] = uigetfile('.csv','Select CSV File to Load');
values = csvread([path name]);

[rows, columns] = size(values);

%first column contains time values
time = values(:,1);
%second column contains data values
data = values(:,2);
%sampling frequency
Fs_0 = 1/(time(2)-time(1));

sampleDuration = time(end);

%Plot Data over time (samples)
figure(1)
plot(time, data)
xlabel('Time (s)');
ylabel('Accel (g)');
title('Time Domain');
grid on;

%-------------------------------------------------------------------------
```

```
%Determine freq values
frequency_values = 0 : Fs_0 / length(data) : Fs_0 / 2;
%Find FFT values
fft_values = fft(data);
%Normalize the FFT values
fft_values = 1 / length(data) .* fft_values;
fft_values(2:end-1) = 2 * fft_values(2:end-1);

%Plot FFT values
figure(2)
plot(frequency_values, abs(fft_values(1 : floor(rows/2) + 1)))
xlabel('Frequency (Hz)');
ylabel('Accel (g)');
title('Freq Domain');
grid on;

%-----------------------------------------------------------------------

%Spectrogram Plot
[x, y, z] = spectrogram(values, Fs_0, 4);

figure(3)
surf(x, y, log(z),'EdgeColor','none')
xlabel('Time (s)');
ylabel('Frequency (Hz)');
zlabel('Amplitude');
title('Spectrogram Analysis');
grid on
ylim([0 2200])
view(2)


function [x, y, z] = spectrogram(data, Fs, per_second_cuts)

    no_of_points = length(data(:,1));
    values = data(:,2);
    points_per_cut = floor(Fs / per_second_cuts);

    values = ...
        reshape(values([1 : floor(length(values) / points_per_cut) ...
        * points_per_cut]), points_per_cut, []);

    [rows, columns] = size(values);

    data_values = [0: rows - 1];
    slice_time = data(rows, 1) - data(1, 1);
    slice_time = slice_time + (slice_time / rows);
    %normalize the data values
    data_values = data_values .* (1 / slice_time);

    %Apply a hamming window
```

```
    hamming_window = [1: rows];
    window_values = ...
        (0.53836 - 0.46164 * cos((2 * pi * hamming_window(:))...
        ./ (length(hamming_window) - 1)));

    %preallocate the values
    abs_values = values;

    for j = [1: columns]
        %FFT function
        values(:,j) = fft(values(:,j) .* window_values(:));
        %Absolute values
        abs_values(:,j) = abs(values(:,j)) / (0.5 * length(values(:,j)));
        %remove DC values
        abs_values(1,j) = 0;
    end

    points_to_plot = points_per_cut / 2;
    start_point = 1;
    end_point = points_to_plot;

    x = [1: columns] / per_second_cuts;
    y = data_values([start_point + 1 : end_point + 1]);
    z = abs_values([start_point + 1 : end_point + 1], :);

end
```

# Appendix I

# MATLAB: Create LPC Baselines

Listing I.1: A MATLAB function used to create the baseline LPC coefficients for Mahalanobis Distance comparison.

```
%-------------------------------------------------------------------------

clear all;
clc;
close all;

%-------------------------------------------------------------------------

[input, path] = ...
    uigetfile('.wav','Select WAV File to Load');
addpath(path);

%extract the data from the WAV file
[sampled_data, sample_rate] = audioread(input);

sampled_data = sampled_data(:,1);
original_data = sampled_data(:,1);

%Length of the file in seconds
length_t = length(sampled_data)./ sample_rate;

%length of the file in values
length_input = length(sampled_data);

frame_size = 3000;
nth_order = 15;
sample_size_t = length_t/(length(sampled_data)/frame_size);

%LPC BASELINE DATA
%-------------------------------------------------------------------------
```

```
%%define the matrices before execution
parameter_matrix_full = [];

% take it in steps of the frame size
for N = 1:frame_size:length_input
%%break the loop if we reach the end of the input data
    if (length_input - N) < frame_size
        break;
    end
    %Calculate LPC coefficents for current frame
    parameter_matrix = lpc(sampled_data(N:N+frame_size-1), nth_order);
    %Add coefficients to a matrix
    parameter_matrix_full = [parameter_matrix_full, parameter_matrix'];

end

% for nan - columns
parameter_matrix_full = ...
    parameter_matrix_full(:,all(~isnan(parameter_matrix_full)));

%store the coefficients in a TXT file, average the results.
fileID = fopen('Knocking.txt','w');
fprintf(fileID,'%12.8f\r\n', mean(parameter_matrix_full, 2));
fclose(fileID);
```

# Appendix J

# MATLAB: Mahalanobis Distance

Listing J.1: A MATLAB function for computing the Mahalanobis Distance between baseline and input data.

```
%-------------------------------------------------------------------------

clear all;
clc;
close all;

%-------------------------------------------------------------------------

%input and define the baseline data - close the files afterwards!
fileID_FUEL = fopen('BASELINE_FUEL.txt','r');
fileID_HEALTHY = fopen('BASELINE_HEALTHY.txt','r');
fileID_TIMING = fopen('BASELINE_TIMING.txt','r');

formatSpec = '%f';
parameter_matrix_full_FUEL = fscanf(fileID_FUEL,formatSpec);
parameter_matrix_full_HEALTHY = fscanf(fileID_HEALTHY,formatSpec);
parameter_matrix_full_TIMING = fscanf(fileID_TIMING,formatSpec);

fclose('all');


%INPUT FAULT DATA
%-------------------------------------------------------------------------
frame_size = 3000;
nth_order = 15;

[input_1, path_1] = ...
    uigetfile('.wav','Select WAV File to Load');
addpath(path_1);

%extract the data from the WAV file
```

```
[sampled_data_1, sample_rate_1] = audioread(input_1);


sampled_data_1 = sampled_data_1(:,1);
original_data_1 = sampled_data_1(:,1);

%Length of the file in seconds
length_t_1 = length(sampled_data_1)./ sample_rate_1;

%length of the file in values
length_input_1 = length(sampled_data_1);

sample_size_t_1 = length_t_1/(length(sampled_data_1)/frame_size);

%LPC FAULT DATA
%-------------------------------------------------------------------------

% %define the matrices before execution
parameter_matrix_full_1 = [];

%take it in steps of the frame size
for N = 1:frame_size:length_input_1
%break the loop if we reach the end of the input data
    if (length_input_1 - N) < frame_size
        break;
    end
    parameter_matrix_1  = lpc(sampled_data_1(N:N+frame_size-1), nth_order);
    parameter_matrix_full_1 = [parameter_matrix_full_1, parameter_matrix_1'];

end

%remove columns with NaN entries
parameter_matrix_full_1 = ...
    parameter_matrix_full_1(:,all(~isnan(parameter_matrix_full_1)));

%Use mahalanobis function to calculate Distance value between baseline and
%input data
mahal_distance_total_FUEL = ...
    MahalanobisDistance(parameter_matrix_full_1', ...
    parameter_matrix_full_FUEL');

mahal_distance_total_HEALTHY = ...
    MahalanobisDistance(parameter_matrix_full_1', ...
    parameter_matrix_full_HEALTHY');

mahal_distance_total_TIMING = ...
    MahalanobisDistance(parameter_matrix_full_1', ...
    parameter_matrix_full_TIMING');


function MahalDist = MahalanobisDistance(sample, Baseline)
    %Determine row and column sizes
```

```
    [rowX, columnX] = size(sample);
    [rowY, columnY] = size(Baseline);

    %sum the rows of each together
    sumN = rowX + rowY;

    %Disclaimer for incorrect data input
    if(columnX ~= columnY)
        disp('Columns in X must be same as in Y')
    else
        xDifference = mean(sample, 1) - mean(Baseline, 1);

        %Covariance of the two input vectors
        covarianceX = Covariance(sample);
        covarianceY = Covariance(Baseline);

        %Mahanobis Distance algorithms
        calcCov = (rowX / sumN * covarianceX + rowY / sumN * covarianceY);
        MahalDist = sqrt(xDifference * inv(calcCov) * xDifference');
    end
end

function covarianceValue = Covariance(X)
    [rows, ~] = size(X);

    %reorganize input data, subtract from original
    shuffleCov = X - repmat(mean(X) , rows, 1);

    %covariance equation
    covarianceValue = shuffleCov' * shuffleCov / rows;
end
```