FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mining Geographic Data for Fuel Consumption Estimation

Vitor Daniel Ferreira da Cunha Ribeiro

Master in Informatics and Computing Engineering

Supervisor: Ana Cristina Costa Aguiar (Prof.)

Co-Supervisor: João Rodrigues (PhD Student)

31st January, 2013

Mining Geographic Data for Fuel Consumption Estimation

Vitor Daniel Ferreira da Cunha Ribeiro

Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Ricardo Morla (Prof.)
External Examiner: Pedro Abreu (Prof.)
Supervisor: Ana Aguiar (Prof.)

18st February, 2013

Abstract

In today's world, society highly depends on mobility. Modern cities possess large and complex transportation networks, whose operating conditions sometimes preclude the minimization of human mobility associated costs, and contribute to an unsustainable growth of carbon emissions. Even in this technological era people live in, with space and mobile technology, the advantages or consequences of each mobility option, is not always direct or available to a user in real time. In this scenario, sometimes a user is imbued with finding the best option with limited resources.

In an effort to reduce both emissions and, mostly important, costs, since money is a global motivator, an innovative solution for fuel consumption estimation is proposed, taking advantage of the opportunities generated by the growing market of mobile devices.

The proposed solution involves calculating fuel consumption with a GPS only, present in every modern smartphone, taking advantage of its ubiquitous availability. In order to achieve this goal, a large dataset of GPS and fuel consumption data was gathered by various volunteers, using an Android mobile application as a gathering unit. The application logs the smartphone's embedded sensor data, and uses an external device known as an On-Board Diagnostics device, to gather vehicle data.

In the end, a solution oriented to emission maps is proposed to enhance the current urban monitor.

Sumário

É possivel afirmar, nos dias correntes, que a sociedade depende largamente da mobilidade. Nas cidades modernas que possuem sistemas de transporte complexos, por vezes, a falta de interoperabilidade entre estes serviços, limita a minimização dos custos associados com a mobilidade, e contribui para um crescimento insustentável de emissões de carbono. Mesmo na era tecnológica em que vivemos, com acesso à tecnologia espacial e móvel, as vantagens ou consequências de cada opção de mobilidade não estão sempre disponíveis em tempo real. Neste cenário, recai por vezes à pessoa a tarefa de encontrar a melhor opção pelos seus próprios recursos.

Num esforço para reduzir estas emissões e, principalmente, custos, dado que o dinheiro é um motivador global, é proposta uma solução inovadora para estimar o consumo de combustível, tirando partido das oportunidades geradas pelo crescimento notório do mercado móvel.

A solução envolve o cálculo de consumo de combustivel através de um sensor GPS, presente em qualquer smartphone moderno, tirando assim partido da disponibilidade ubíqua desta tecnologia. Para alcançar este objectivo, foi feita uma recolha de dados por parte de vários voluntários, usando uma aplicação móvel para Android. Esta aplicação grava os dados recolhidos pelos sensores embebidos no smartphone, e usa um dispositivo externo, conhecido como OBD, para recolher dados de veículos.

A solução encontrada pode ser usada para mapas de emissões.

Acknowledgements

First and foremost, i dedicate this work to the kind volunteers that agreed to spend their fuel in the name of this cause. I would like then to thank Professor Carlos Tavares de Pinho and Carlos Valentim for their time and patience. I thank Professor Jaime Villate for his friendship and Professor António Augusto de Sousa for his kindness and availability, for they are both valuable pillars in the MIEIC structure. I also thank their capability of reducing the initial awkwardness towards the teaching class.

A special thanks goes to my supervisor Ana Aguiar for all the help and guidance provided, and for putting up with me and my constant confusion, and never loosing the will to give one more push. I also thank my co-supervisor and dear friend João Rodrigues for the constant intellectual bullying, the bags of popcorn for the late hours, for teaching me about sugar turning into caramel, and being a light at times of despair more times than a sane person could (although i suspect he has already lost it). This work would have not been possible without my godly supervisors.

Finally, before the music starts, i thank my loved ones, family, friends, my dogs, Google, and all the people who supported me through my academic life, all the workers at Instituto de Telecomunicações for their effort in maintaining a fun and healthy work environment, and all the unnamed people involved in this thesis.

Thank you also my imaginary friend for reappearing 15 years later and not letting me talk alone to my bedroom wall.

"Mathematics is the art of the possible"

Paraphrased from the quote
"Politics is the art of the possible"
by Otto von Bismark.

Contents

List of Figures

List of Tables

Abbreviations

FEUP	Faculdade de Engenharia da Universidade do Porto
IT	Instituto de Telecomunicações
OBD	On-Board Diagnostics
UI	User Interface
EPA	Environmental Protection Agency
CARB	California Air Resources Board
ECU	Engine Control Unit
PID	Parameter ID
AFR	Air-to-Fuel Ratio
MDD	MyDrivingDroid
VSP	Vehicle Specific Power
VANET	Vehicular Ad-hoc NETwork
GPS	Global Positioning System
AP	Access Point
ANN	Artificial Neural Network
SVM	Support Vector Machine
API	Application Programming Interface
OLS	Ordinary Least Squares
EM	Errors-in-Model
SAE	Society of Automotive Engineers
CAN	Controller Area Network
NMEA	National Marine Electronics Association
SDP	Service Discovery Protocol
CoD	Class of Device

Chapter 1

Introduction

Mobility has always been an important factor in human history. Considering the rising and continuous change of user requirements, where people increasingly demand everything everywhere [GT08], mobility can be considered one of the foundations of economics and human development [Haa09].

In a world where the evolution of technology has widened the possibilities of innovation, providing new user experiences with mobile platforms like smartphones, tablets, PDAs, and other mobile devices that defined the mobile space, giving the power of mobility at the tip of a finger, this platform has become a hot research topic. Furthermore, mobile devices have become even more powerful with the constant improvement and addition of motion, position and environmental sensors. Harnessing the power of mobile platforms, it is possible to further exploit human mobility by continually improving traffic patterns and driving styles.

Parallel to this situation, is the constant improvement of space technology and wireless communications, which aims to provide ubiquitous wireless access [PvHD06].

1.1 Context

In modern cities where infrastructures provide various means of transportation, including private vehicles, buses, taxis, railways and other forms of public transit, a user is sometimes imbued with the task of planing and organizing his travel routes.

Because these systems can be large complex networks that usually operate independently instead of collaborating with each other in a holistic fashion [Lit03], the task of coordinating them can prove to be a difficult one as travellers are typically unaware of the current conditions of the various transportation networks, as of the exact time, monetary costs and environmental impact that a certain mobility option incurs.

Since time, money and environmental impact can be directly associated with a route option, this uncoordinated setting limits the minimization of these costs and contributes to an inefficient exploitation of the transportation services as to an unsustainable growth of carbon emissions associated with human mobility [RKBG07].

1.2 Motivation

Considering this scenario, it is difficult to know, for example, exactly how much time and fuel is spent in our travels, what is the most economic route between point A and point B at a certain time, or if there is someone who consumes less fuel or takes less time in the same route and vehicle class and how those differences can be explained.

On the other hand, the continuous rise of fuel prices has stimulated the search for alternatives, not only in engine types or driving behaviour [LLL+12], but has also boosted the search for new methods of intelligent transportation [Li12]. There are also on-going researches that aim towards giving vehicle users better fuel efficiency feedback [CQSK11], and lead some to seek driving techniques that maximize fuel economy, also know as *hypermiling*.

The wide availability of smartphones due to its increasingly accessible price has created a potential tool to use in the process of solving these problems. Mobile devices today, besides the obvious portability, possess a high processing power and are equipped with a wide range of sensors like an accelerometer, gyroscope, magnetometer, GPS, Wi-Fi and Bluetooth devices. Using the device's embedded capabilities, the implementation in these mobile platforms of algorithms that extract information about mobility patterns and ecological efficiency, offer a unique opportunity to leverage information to empower citizens to make informed decisions improving the quality of life of people in a perceivable way [LML10].

1.3 Goals

The general purpose of this thesis is to provide an innovative solution by joining the previous scenarios and present algorithms that enable estimation of fuel consumption from GPS data from a mobile phone that can be placed anywhere. The reasons for choosing only the GPS, besides the ubiquitous availability of the Global Navigation Satellite System (GNSS) in open spaces, is explained in detail on Chapter 5.

Among the current proposals, some models provide a solution for generic classes of vehicles with data gathered from an On-Board Diagnostics device (OBD). This will be further addressed in Chapter 2. But since the user must own an OBD device, and these

devices are not widely available due to price and required knowledge of OBD technology, the development of algorithms that can estimate the fuel consumption from mobile sensors enables anyone with only a smartphone to know more about his driving behaviour.

A mobile application for Android named MyDrivingDroid (MDD) was developed at Instituto de Telecomunicações that gathers data from a mobile device's sensors. The first goal of this thesis was to extend that application to be able to gather vehicle data from an On-Board Diagnostics device through a Bluetooth connection. The implementation details of this module are discussed in Chapter 5. A mobile-adapted version of the existing OBD models was implemented, used and improved in the same application to provide automotive data feedback in real time, like fuel consumption.

Through the constant use of the extended mobile application in multiple vehicles from various volunteers, a large dataset was collected from the available smartphone's sensors and external OBD sensor, and fuel efficiency obtained from the OBD dataset was validated. The validation process is explained in Chapter 5. Through constant improvement of the algorithms, this work's solution can contribute to an enhancement of the existing urban monitor systems.

Thus, with the proposed algorithm that estimates fuel consumption, a possibility is presented to provide each user with their own driving statistics, and, in a collaborative fashion, compare it with others and, with this data, recommend alternative mobility solutions, with the help of the mobile platforms' resources.

1.4 Outline

Besides the introduction, this document will detail the related work of the issues described previously regarding engine mechanics and some of the OBD standard's background, fuel consumption and emission models, among other topics, that were used to provide the proposed solution.

Furthermore, chapter 5 discusses the implementation details, including the OBD protocols, the chosen Android sensors, the fuel consumption calculation from vehicle data and the OBD Module for MDD, in order to enable the gathering of sensor data. Also in this chapter, the data collection methodology and processing are discussed, and results are presented, along with the methods used to analyse, calibrate and process data. It is also explained how some GPS values were derived and they were used in the obtained fuel consumption algorithms, followed by observations and conclusions.

The last chapter discusses regression and some machine learning approaches, and details experimented models and the formula obtained from the data.

The introduction has stated the problem which drove this research and provided an insight of what is expected. This, of course, will be further developed in the next chapters.

Chapter 2

State Of The Art

To develop an algorithm that enables fuel consumption estimation from GPS data from a mobile phone while driving a vehicle, it is necessary to get involved in various fields. In this section, the current state of the art related to this work is discussed. This primarily includes the study and analysis of current Fuel Consumption Models and Mobile and Collaborative Sensing Platforms.

The diagram 2.1 not only represents an overview of the related work, but also the solution's approach. The implementation details are referred in Chapter 5.

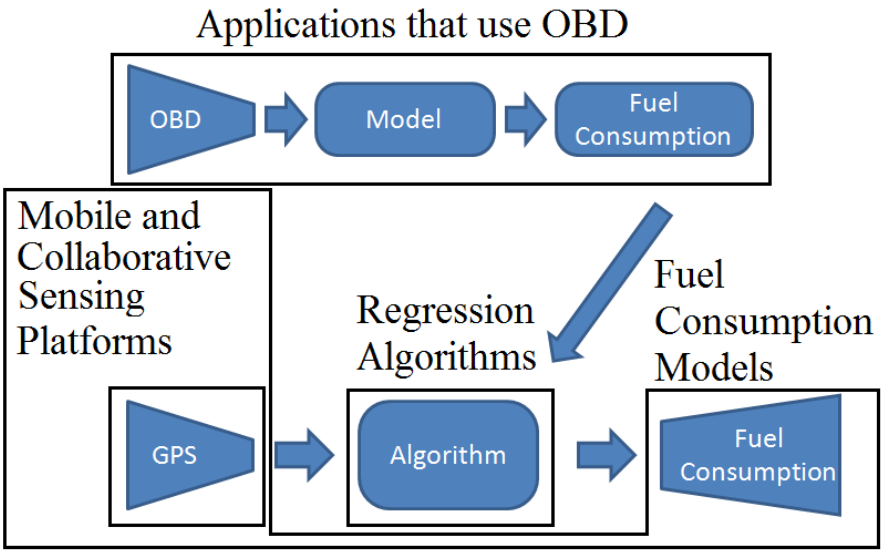


Figure 2.1: Approach and Fields of Study

However, applications that use On-Board Diagnostics devices are firstly addressed as some already provide information related to the previous fields and also because they

constitute an invaluable part of the solution by providing means to obtain tangible consumption data that can be used for result comparison and analysis or to train an algorithm.

Furthermore, it is also necessary to comprehend how OBD systems work as well as vehicle engines and what distinguishes them in means of consumption, and also what are the best User Interface patterns that can be used while driving a vehicle, optimizing user interaction and minimizing the risks associated with this action.

Over the years, vehicle engines have evolved by becoming more powerful and less fuel consuming, especially diesel engines that have largely improved primarily due to advances in the common rail direct injection systems (see Figure 2.2 [1]). Because of this, it has been largely debated what is most environment friendly engine [KGG07].

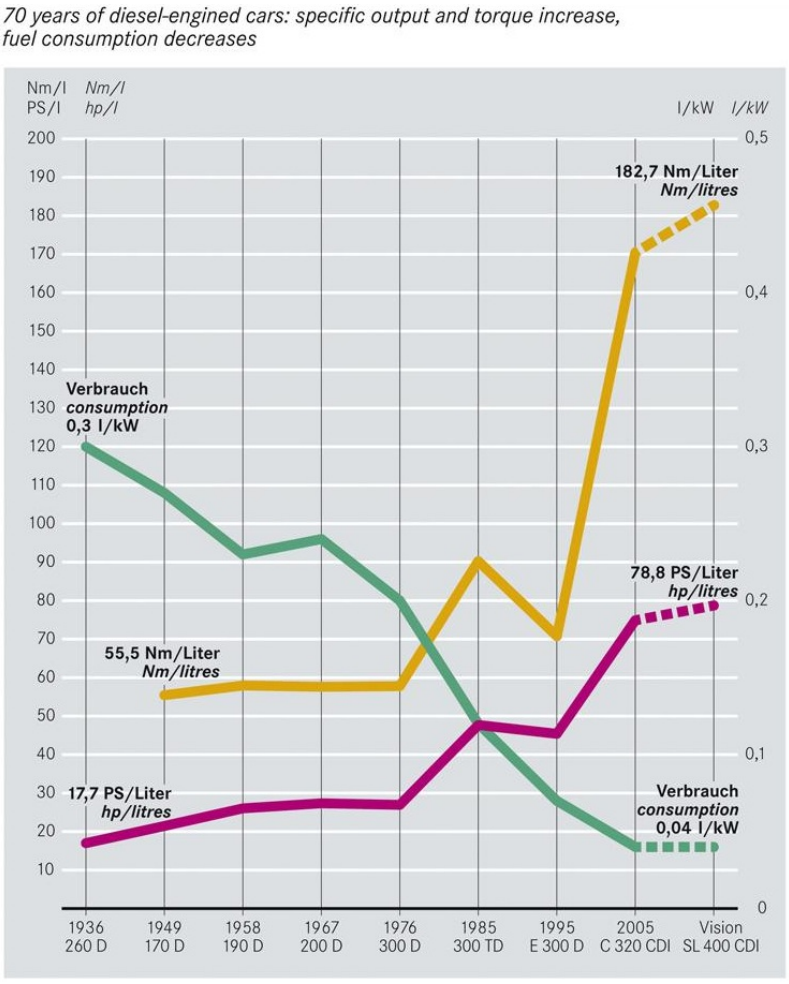


Figure 2.2: Engine Evolution

Although the basic designs of petrol and diesel engines are similar, the mechanics are different as demonstrated in Figure 2.3. A gasoline engine compresses its fuel and air charge and then initiates combustion by the use of a spark plug. A diesel engine just

[1] http://autospeed.com/

compresses air until the combustion chamber reaches a temperature for self-ignition to occur.

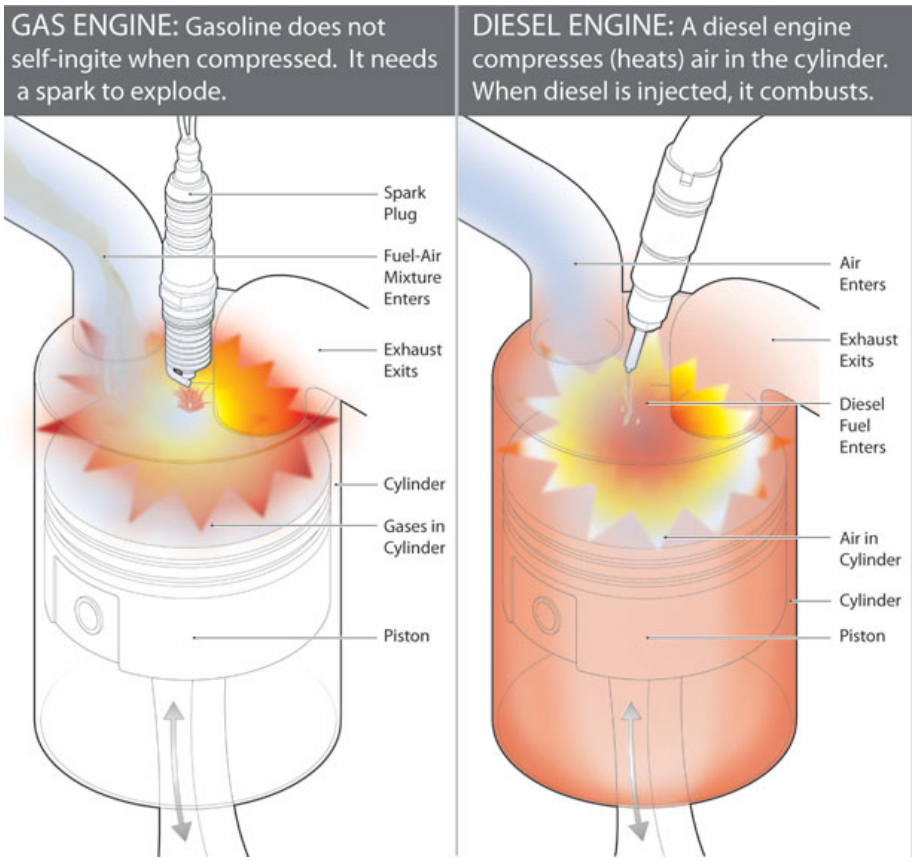


Figure 2.3: Gasoline VS Diesel

Because of this mechanical behavioural differences, there was a need to develop different fuel consumption formulas. To comprehend how the fuel consumption formulas were designed, the engine's mechanics topic is deepened in Chapter 5.

2.1 On-Board Diagnostics

Because of the rising concern of carbon emissions and its impact, Environmental Protection Agency (EPA) [2] and California Air Resources Board (CARB) [3] sought to ensure that new vehicles were running as cleanly and efficiently as they could by equipping vehicles with more-sophisticated emission equipment and better diagnostics systems to monitor that equipment.

However, since normally each manufacturer has its own engine design and diagnostics equipment, it would be very difficult to impose a certain type of equipment. On the

[2] http://www.epa.gov/
[3] http://www.arb.ca.gov/

other hand, purchasing diagnostics equipment for each of the manufacturers' proprietary vehicle information systems, is not only impracticable but also prohibited for third-party garages and testing centres as not all system information is public.

So the necessity of a standard emerged. In the year 1996, the On-Board Diagnostics standard was born as its subsequent OBDII revision which is a collection of connection and protocol standards [4].

2.1.1 The Protocol

Cars with an OBDII system have a 16-pin female interface connector located in the vehicle's cabin within approximately half a metre of the steering wheel. For most cars, it is located below the steering wheel, sometimes hidden within a compartment.

By connecting an OBDII standard compliant hardware interface, it is possible to monitor a vehicle's emissions system, like fuel system status, engine and vehicle speed and also the status of some of the vehicle's sensors, among others. It is also possible to reprogram emission related ECUs.

An Engine Control Unit (ECU) is an electronic unit that constantly interprets data from the engine bay's sensors, making the correspondent adjustments to ensure the optimal functioning of the internal combustion engine.

Data sent over an OBD connection is accessible via standardized Parameter ID (PID) codes. Some PIDs are reserved. Among the reserved PIDs, there are those proprietary of a manufacturer sometimes not publicly available. There are also other reserved PIDs for common issues that a vehicle's engine and emissions equipment may encounter. These special PIDs are known as *trouble codes*. For example, if there is some sort of engine related error or if a value falls outside of a predetermined range, the vehicle will illuminate its *Check Engine Light* and send the corresponding PID to the OBD interface. There are about 900 trouble codes in the OBDII standard.

2.2 Applications That Use OBD

OBDII is mainly used for emissions testing, however, since its birth, it has been used in a wide variety of ways, mostly by vehicle enthusiasts, since it is possible with the public PIDs to calculate various results. Fuel economy metres' results are the most appropriate for this thesis. Nonetheless, OBD can also be used as a Performance Tester or a Simple Code Reader and Data Logger.

Usually a Data Logger is meant to stay with the vehicle as it is driven about, recording all of the information provided from the On-Board Diagnostics device for further analysis.

[4] http://www.arb.ca.gov/msprog/obdprog/obdregs.htm

From that data, since vehicle and engine speed and steering angle are publicly provided, among other parameters, it is possible to measure the performance of a vehicle on a granular level. Performance testers can estimate horsepower, torque, the time from 0 to 100 Km/h and rotation speed, like a tachometer or RPM Gauge. Some manufacturers provide non-standard PIDs that can be used to more easily derive boost pressure or engine load.

Fuel economy metres normally use more complex models to calculate fuel consumption or emission rates since the standard PID set does not explicitly include a fuel economy parameter. OBD standards, however, have evolved in a way that there is now enough information in recent vehicles about the engine and fuel systems to extrapolate litres per distance fairly accurately, as seen in the validation section of Chapter 5. Some older vehicles, unfortunately, may supply insufficient or not enough data, resulting in some readings with poor accuracy or no accuracy at all.

The following subsections document some features and characteristics of commercial and non-commercial applications that use OBD devices, by discussing some of its capabilities and results. Among the various products, the Android applications most of the times resort to a Bluetooth OBDII device and transform the data read from the OBD port into some sort of visual interface. Other applications involve a computer software or a custom device that users must pay for but normally include more sophisticated models.

2.2.1 ecoRoute HD

Garmin is a company that manufacturers hand-held portable and fixed-mount products for navigation and communication. Garmin ecoRoute HD is an engine diagnostics tool and monitors vehicle performance. It also provides fuel efficiency and sensor and gauge data.

The ecoRoute HD [5] module consists of an OBDII connector that transfers data via Bluetooth to an Android smartphone with the Garmin Mechanic application installed to process data or a Bluetooth-compatible nüvi, which is a Garmin GPS navigation system.

Garmin applications allow a user to create vehicle profiles to improve the outcome's accuracy. The Garmin Mechanic application for Android is free, however the connector or the nüvi navigation system is not. Being a closed source commercial product, the model used is not public.

2.2.2 ScanGauge

Linear Logic is a company with a product line called ScanGauge [6]. A ScanGauge device is a trip computer with an OBDII connector that logs data and functions as an

[5] http://www.garmin.com/pt/
[6] http://www.scangauge.com/

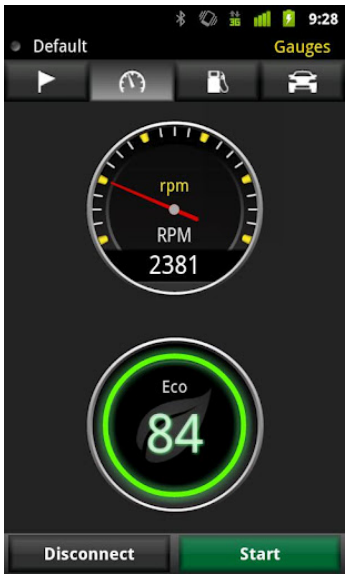


Figure 2.4: Garmin Mechanic Interface

error-code scanner for *Check Engine Lights* issues. The product is closed source and it is not free, although the manuals are and provide some tips for *hypermiling*.

2.2.3 ScanXL and DashCommand

Palmer Performance Engineering is an industry dedicated in manufacturing products for vehicle diagnostic and communications software, notoriously ScanXL and DashCommand. Their products rely on OBDII devices to provide vehicle data and deliver generic diagnostic data in real time [7]. It is also possible to view vehicle Diagnostic Trouble Codes.

The products are closed source and not free, but the manuals are publicly available at their website. This is a great source of information as the manuals provide instructions on how their technology, DashXL, works in detail.

Among the information, it is explained in a model how fuel efficiency is obtained. Since this thesis required a fuel consumption model, these manuals provided very useful information on the OBD model implementation.

2.2.4 Torque Pro

Torque [8] is an Android application made by Ian Hawkins that connects to an OBDII compliant device via Bluetooth serving as a vehicle performance and diagnostic tool. It allows the user to create vehicle profiles to help improve the accuracy of the outcome.

[7] http://www.palmerperformance.com
[8] http://torque-bhp.com/

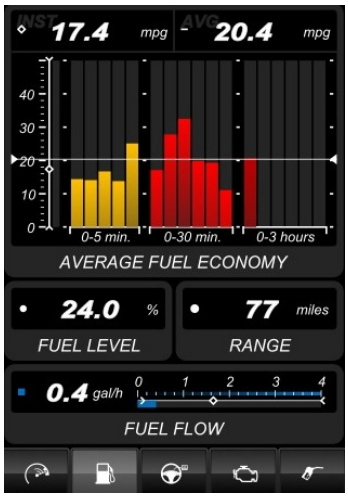


Figure 2.5: Dash Command Interface

It is a commercial product that shows graphs and virtual gauges with readings from the OBD port data, like horsepower, torque, acceleration, etc. Also uses Android sensors like the GPS. Unfortunately, since it is not open source and the documentation in the website is limited, it is not fully known how the data is used. The only open source component is a plugin for developers to improve and upgrade the application, but it does not provide useful information for this project.

2.2.5 alOBD ScanGenPro

Alexandre Beloussov has developed some Android Scantool applications [9] that provide basic automotive diagnostic by gathering data from an OBDII device via Bluetooth. The products are priced as well as some extensions designed for specific vehicles.

It is not as intuitive as Torque as, according to the author, is intended for an audience familiar with the operation of an engine control computer and vehicle sensors. It is also closed source.

2.2.6 OBDroid

OBDroid is a very simple diagnostics and vehicle interface tool that uses Bluetooth OBDII devices. It reads data directly from the OBD PIDs and shows it in some form of visual interface. Although is it not free and it does not possess a fuel consumption model, a part of the source code is hosted at github [10] and provides useful information on how to read data from a Bluetooth On-Board Diagnostics with an Android.

[9] http://www.alobdscanner.com/
[10] https://github.com/syntelos/obdroid

Figure 2.6: alOBD Interface

2.2.7 OBD II Reader

This is a free and simple open source Android application [11] that also uses a Bluetooth OBDII device to read PIDs and present simple data. It is also an interesting reference on how to code OBD data handling.

2.2.8 VoyagerDash

VoyagerDash from GTOSoft [12] is an engine trouble code diagnostics and engine monitor. It reads data from an OBD device via Bluetooth and shows it in form of gauges and graphs. It is not free and it is closed source.

2.2.9 DashDyno SPD

Auterra DashDyno SPD [13] is a device for instant and average fuel economy measures. It logs data from internal and external sensors. The internal being engine sensors and external are add-ons like GPS or O2 sensors for Air-to-Fuel Ratio measures. From speed and RPM it estimates horsepower and torque.

It also serves as a diagnostics tool that warns the user about issues related to the *Check Engine Light*. Comes with a software for Windows for travel analysis and simulation resorting to Google Earth. It is a commercial product and no source code or model is publicly available.

[11] http://code.google.com/p/android-obd-reader
[12] http://www.gtosoft.com/
[13] http://www.auterraweb.com/dashdynoseries.html

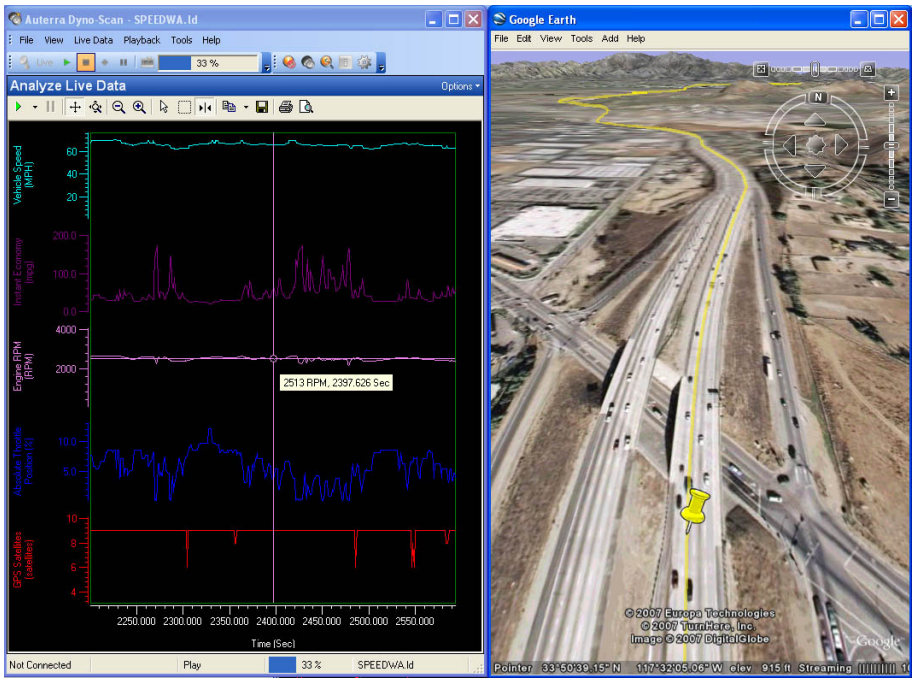


Figure 2.7: DashDyno Interface

2.2.10 Applications Overview

Since the source code for most of the applications is not public, it is difficult to comprehend their inner workings in detail. However, it is known that these applications all heavily depend on the available PID's from an OBD connector to calculate results.

Though some applications provide information on how to code OBD handling and how to use the retrieved data to construct a fuel consumption model, it is necessary to further the knowledge about this type of models. The next section will address that issue.

2.3 Fuel Consumption Models

Some of the previous mobile applications employ fuel consumption models, but they usually tend to generalize some values like vehicle specifications as not all vehicle related values can be obtained just through an OBD. The vehicle's structure, engine and parameters can have a great impact on the outcome, and simple models either use constants for generic types of vehicles, or prompt the user about that information.

Some applications prompt the user to insert vehicle profiles, which can encompass values like engine displacement, fuel type, tank level and capacity, car size and weight, RPM redline, among others, or even more detailed engine information. Unfortunately, for users not familiar with vehicle specifications, filling that information can be confusing and time-consuming.

Furthermore, there are more complex models that not only encompass more detailed vehicle parameters, but are also designed for specific vehicle classes [Fen07] or engines [Ono04], as it is expectable to obtain differences in results between, for example, a gasoline passenger car and a diesel heavy duty truck using just generic models, resulting in low accuracy.

There are some models that also use very different approaches, ranging from speed and acceleration [ARTV02] to vehicle specific power [HDY+05] [McC99], and some use external sensors, like a emission sensor on the tailpipe.

This section addresses an overview of model types and discusses existing models, pointing some of the advantages and disadvantages between them.

Environmental laws and rules from the past decades to the present date have been established due to environmental concerns. In an effort to mitigate this concerns and raise environmental awareness, researches have been and are currently being made related to this topic. So the majority of the existing models are primarily emission related. However, since emissions are directly related to fuel consumption [CCN02], it is plausible to use these models or just specific modules from them.

2.3.1 Emission Model Types

Models can be broadly categorized into macroscopic, mesoscopic, and microscopic models [FD12] where, as the name suggests, each is related to a certain scale.

Macroscopic are often related to large scale emissions, like in a city. Mesoscopic are medium-scale, such as roads or an individual vehicle's specific trips. Microscopic are the most detailed and vehicle-oriented and usually take into account more specific vehicle or engine parameters (see Figure 2.8 [14]).

Some of this models use vehicle variables that can be categorized into vehicle parameters and traffic or road parameters.

Vehicle parameters are vehicle class, mass and length, fuel type, tank size, engine displacement, accessory use, like air conditioning, among others. Traffic or road parameters are road related characteristics like road grade.

Macroscopic models are mainly based on the average speed of the traffic flow. So they do not account for individual operation conditions or speed fluctuations caused by individual vehicles. These models tend to simplify calculations to estimate fuel consumption and pollutant emissions, which finally leads to a reduced accuracy for individual vehicle dynamics.

Mesoscopic models use the average or instantaneous speed and acceleration, sometimes to construct virtual driving cycles, since they are more trip oriented. This means that these models usually use traffic or road parameters.

[14] http://www.cert.ucr.edu/cmem/

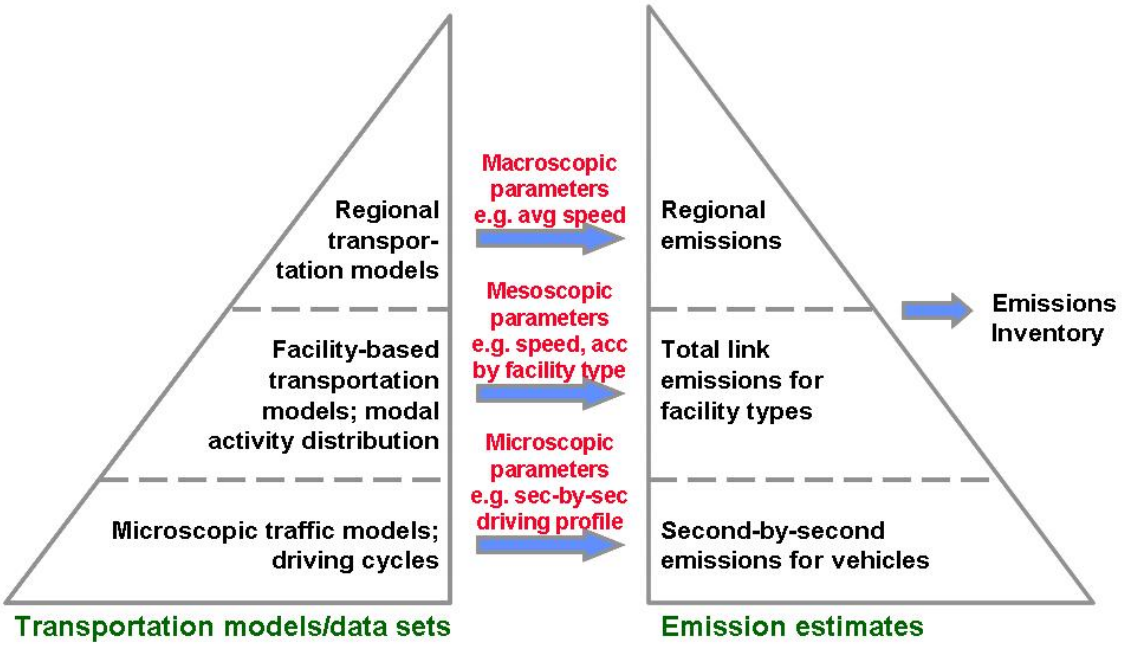


Figure 2.8: Transportation Pyramid

Microscopic emission models are based on instantaneous individual vehicle variables and can overcome some of the limitations of the macroscopic models since they take into account more detailed individual vehicle parameters and have higher temporal precision.

Microscopic models can be classified into three subtypes [CCN02]:

- **Emission Maps** normally based on speed/acceleration lookup tables;
- **Purely Statistical** normally regression-based;
- **Load-based** that use engine load.

Emission maps are matrices that contain the average emission rates for every combination of speed and acceleration. Although generally easy to generate and use, they are usually insensitive to traffic or road parameters.

Statistical regression-based models predict fuel consumption and emission rates by typically using regression methods on a large number of instantaneous vehicle speed and acceleration values.

Load-based models are primarily based on fuel consumption rate which is a surrogate for engine power demand, also referred to as engine load. These models try to simulate the physical phenomena that generate emissions. They are usually more detailed and flexible and can use a lot of vehicle variables. This also means that these models can be more complex and can consume a lot of computational power.

Most of the developed models are mesoscopic and microscopic that mostly use speed and acceleration as inputs, as they are closely related to emission outputs [IBL06]. Though

there is some discussion on whom is the most accurate and what is the best consumption calculation method [Yue08] [RA03] [DR02] [ZN01].

Huanyu Yue argues that most mesoscopic models use simplified mathematical expressions and ignore transient changes of a vehicle's speed and acceleration. On the other hand, microscopic models can be very costly and time consuming [Yue08].

While in an article comparing various models, some authors favour mesoscopic model VT-Micro (Section 2.3.4) [RA03] that relies on speed and acceleration, while in another article it is argued that speed values are insufficient [DR02].

The following subsections detail some of the existing models.

2.3.2 EMIT (EMIssions from Traffic)

EMIT is statistical model for instantaneous tailpipe emissions and fuel consumption derived from a regression-based and load-based emission modelling approaches [CCN02], composed by two modules as shown in Figure 2.9.

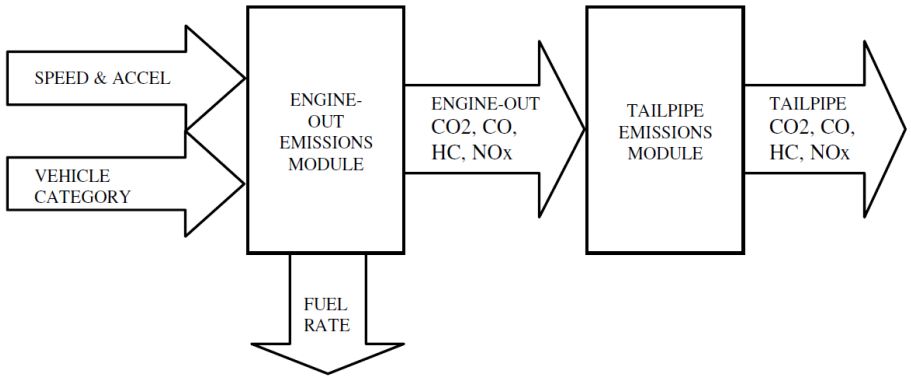


Figure 2.9: EMIT Model Structure [CCN02]

This model depends on second-by-second speed and acceleration and a vehicle category to predict the corresponding second-by-second fuel consumption. And since fuel consumption is related to emissions as previously stated, with these values the model also predicts second-by-second tailpipe emission rates.

This model has been widely referenced in other model proposals as it gives reasonable results with acceptable accuracy for fuel consumption. EMIT presents good results for carbon dioxide emissions, reasonable accuracy for carbon monoxide and nitrogen oxides, but less desirable accuracy for hydrocarbons [CCN02].

The greatest disadvantage of this model is that besides requiring calibration from a large database, it is limited for light-duty vehicles (like a passenger car for common use) and is only suitable for hot-stabilized conditions with zero road grade.

However, the fuel consumption module is greatly detailed and provides some useful formulas for fuel rate and allows to comprehend how it is related to emission rate.

$$EO = EI * FR \tag{2.1}$$

$$FR = \left(\frac{CO_2}{MWCO_2} + \frac{CO}{MWCO}\right) * (MWC + MWH * NM) + HC \tag{2.2}$$

EO - engine-out emission rate (g/s)
EI - emission index (mass of emission per mass unit of fuel consumed)
FR - fuel consumption rate (g/s)
CO_2 - measured Carbon dioxide engine-out emission rate
$MWCO_2$ - carbon dioxide molecular weight constant (44)
CO - measured Carbon monoxide engine-out emission rate
MWCO - carbon monoxide molecular weight constant (28)
MWC - carbon molecular weight constant (44)
MWH - hydrogen molecular weight constant (1)
NM - approximate number of moles of hydrogen per mole of carbon in the fuel (1.85)
HC - measured hydrocarbon engine-out emission rate

Equation 2.1 supports the statement about emissions being closely related to fuel consumption.

2.3.3 aaSIDRA and aaMOTION

These instantaneous emission models [AB03] were proposed in 2003 and later in 2007 were integrated in a closed source paid software and subsequently named SIDRA TRIP [15]. The model uses microscopic GPS or trip data representing a standard driving cycle and produces vehicle trip assessment data like distance, speed, operating costs, emissions, noise and fuel consumption.

If using a GPS, data is easily gathered and it produces reliable results. However, it depends on various vehicle variables like engine, traffic and road parameters. It is also possible to input fuel prices, for cost assessment. SIDRA TRIP provides default vehicle profiles, that can be configured or created, generalizing vehicle classes to facilitate user input, as some parameters can be complex for normal users.

2.3.4 VT-Micro

Virginia Tech Micro [AR03] [RA04] is a mesoscopic model for normal and high emitting vehicles that predicts the instantaneous fuel consumption and emission rates of individual vehicles.

[15] http://www.sidrasolutions.com/

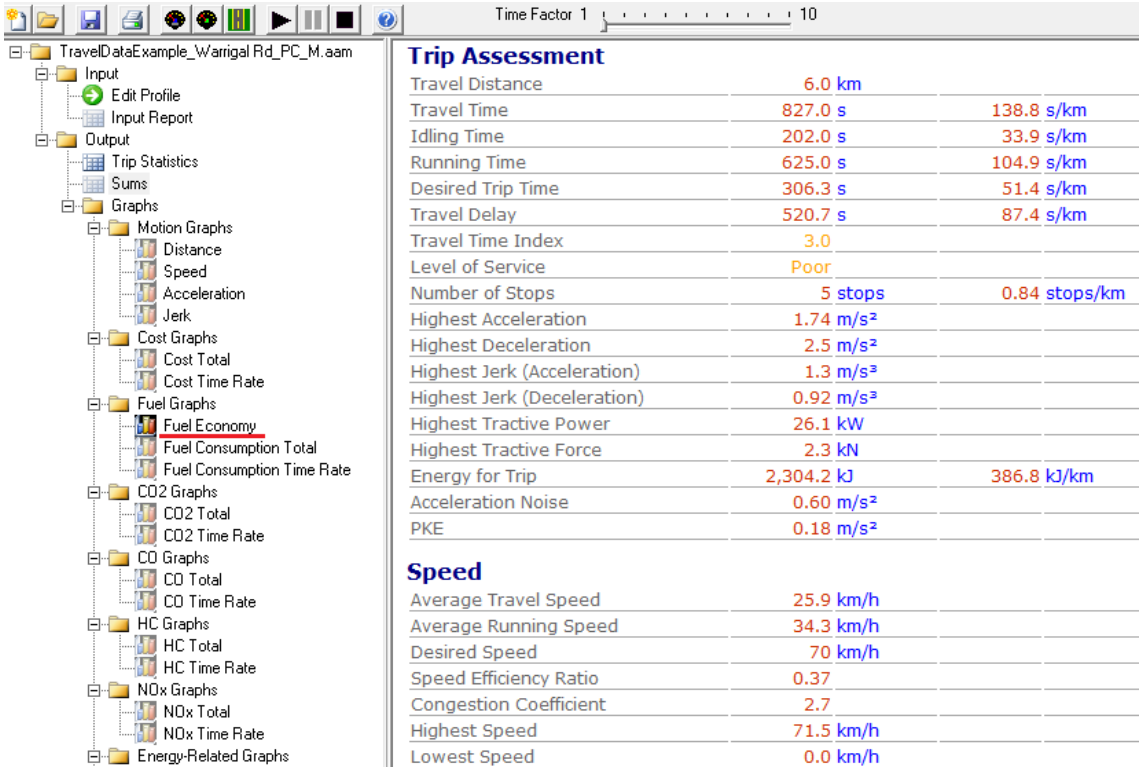


Figure 2.10: Trip Statistics

It constructs a synthetic drive cycle based on instantaneous speed and acceleration. For each drive cycle, the model estimates the proportion of time that a vehicle typically spends cruising, decelerating, idling and accelerating.

$$MOE_e = \begin{cases} e^{\sum_{i=0}^{3}\sum_{j=0}^{3}(L^e_{ij} * u^i * a^j)} & \text{for } a \geq 0 \\ e^{\sum_{i=0}^{3}\sum_{j=0}^{3}(M^e_{ij} * u^i * a^j)} & \text{for } a < 0 \end{cases} \tag{2.3}$$

MOE_e - instantaneous emission rate (mg/s)

$L^e{}_{ij}$ - model regression coefficient for MOE 'e' at speed power 'i' and acceleration power 'j'

$M^e{}_{ij}$ - model regression coefficient for MOE 'e' at speed power 'i' and acceleration power 'j'

u - instantaneous vehicle speed (km/h)

a - instantaneous vehicle acceleration (m/s^2)

The model can provide accurate estimates of distance-based average vehicle fuel consumption and emission rates, but because of the non-linear relationship between vehicle

emissions and vehicle speed, the model is also prone to inaccuracies [RA04] [Yue08].

2.3.5 CMEM (Comprehensive Modal Emission Model)

The core of this microscopic emission model is the fuel rate calculation which is a function of engine speed and engine load (power demand) [16]. This model consists of several modules as observed in Figure 2.11

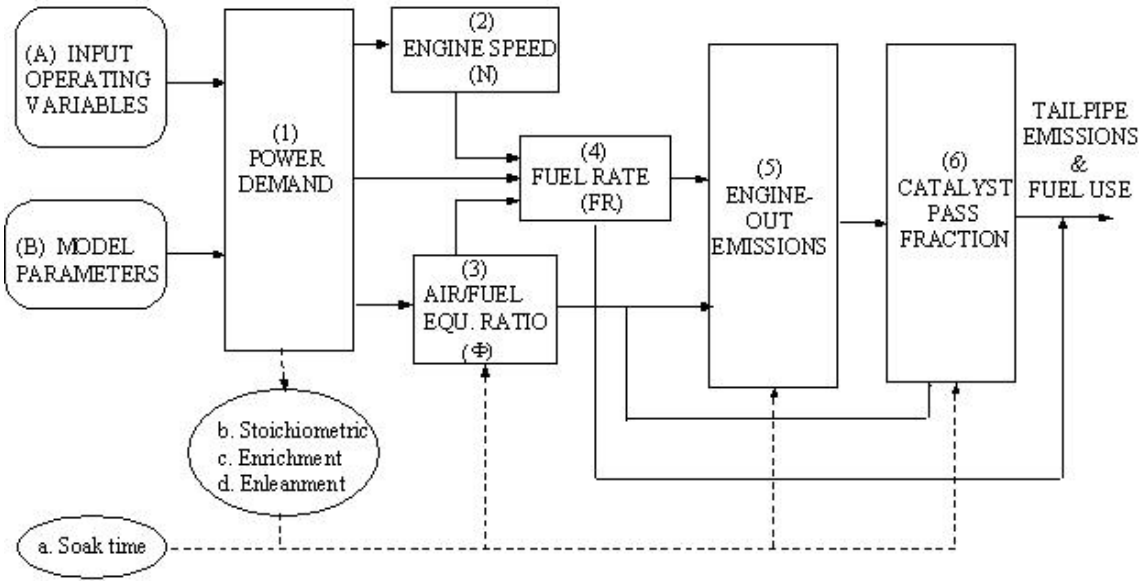


Figure 2.11: CMEM Model Architecture

Engine speed is determined based on vehicle velocity, gear shift schedule and power demand. The vehicle power demand is determined based on specific vehicle parameters, road grade and second-by-second vehicle speed, from which acceleration is derived. It can also account for accessory use like air conditioning. The model can use a total of 35 parameters to estimate vehicle tailpipe emissions and fuel use.

The greatest advantage of CMEM is that it is a public-domain model and it is claimed to be one of the most detailed and best tested models for estimating hot-stabilized vehicle exhaust emissions, and has been used in a variety of projects, like for air quality evaluation in Southern California [Bor07].

For the purpose of this thesis, it not necessary to study all modules of the model. The most appropriate is the Fuel Rate Module on page 48 of the CMEM User's Guide [BAY+06].

[16] http://www.cert.ucr.edu/cmem/

$$FR = \left(k*N*V+\frac{P}{\eta}\right)*\frac{1}{lhv}*\left(1+b_1*(N-N_0)^2\right) \quad (2.4)$$

$$K = K_0*(1+C*(N-N_0)) \quad (2.5)$$

$$N_0 = 30*\sqrt{\frac{30}{V}} \quad (2.6)$$

FR - fuel use rate in grams/second
P - engine power output in kW
K - engine friction factor
N - engine speed (revolutions per second)
V - engine displacement (liter)
η - measure of indicated efficiency for diesel engines (approximately 0.45)
b1 - coefficient of approximately 10^{-4}
C - coefficient of approximately 0.00125
lhv - the lower heating value of a typical diesel fuel (approximately 43.2 kJ/g)

2.3.6 MOVES (MOtor Vehicle Emission Simulator)

MOVES is EPA's latest motor vehicle emission model for estimating emissions from highway vehicles. This modal emission regression-based model [17] substituted its predecessor MOBILE6 [HD09] in 2010. It uses on-board emissions data and Vehicle Specific Power (VSP) as a variable on which emission rates can be based, and allows multiple scale analysis. The regression model requires a lot of vehicle and road parameters as explanatory variables, and assumes vehicle speed has no headwind. The VSP calculation and model details are published on section 3-10 of the document *Transit Bus Load-Based Modal Emission Rate Model Development* [Fen07]. VSP is a function of speed, acceleration, road grade, and other factors as shown in the Formula 2.7.

$$VSP = v*(a*(1*\varepsilon)+g*grade+g*C_R)+0.5*\rho*C_D*A*v^3/m \quad (2.7)$$

v - vehicle speed (assuming no headwind) (m/s)
a - vehicle acceleration (m/s^2)
ε - mass factor constant accounting for the rotational masses (approximately 0.1)
g - acceleration due to gravity (m/s^2)
grade - road grade (ratio of rise to run)
C_R - rolling resistance (approximately 0.0135)
μ - air density (1.2)

[17] http://www.epa.gov/otaq/models/moves/index.htm

C_D - aerodynamic drag coefficient (dimensionless)
A - the frontal area (m^2)
m - vehicle mass (metric tons)

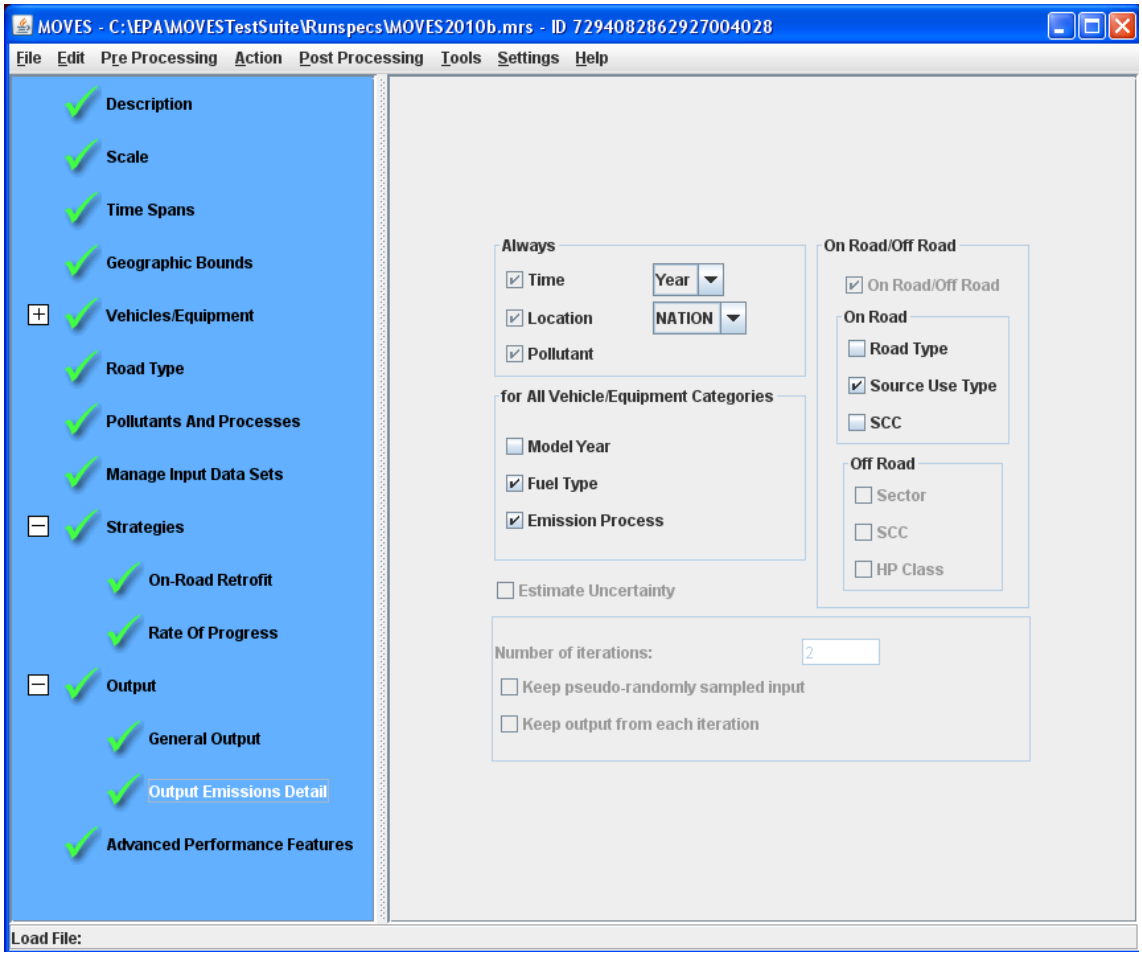


Figure 2.12: MOVES Interface

This model is implemented in a software publicly available at EPA's website (see Figure 2.12). The manuals are also freely available and provide a good description on the model's approach. However, it is not open source and so not all the formulas are fully disclosed.

Nevertheless, this model shows that a different approach is possible on vehicle emissions, namely through the Vehicle Specific Power function.

2.3.7 Models Overview

It is possible to conclude that most of the models use speed and vehicle parameters to estimate emissions and fuel consumption. They mostly require user input, especially

regarding vehicle variables and some are specific to certain types of vehicles or engines. Although, in most part, they provide reliable estimations, the complete models can be complex, but still some of their modules can be exploited and provide a firm basis used in the development of the proposed solution.

2.4 Mobile and Collaborative Sensing Platforms

Because the proposed solution involves mobile sensing and is to be preferably used in a collaborative environment, it is necessary to analyse current mobile and collaborative sensing platforms, some of them involving intelligent transportation systems, and most importantly comprehend its impact on mobility, which is a highlight topic of this thesis.

2.4.1 Urban Monitors

Real-time transit tracking is gaining popularity, but unfortunately many transit agencies still lack either the funding or initiative to provide tracking services [TBG10]. To overcome these limitations and improve the rider experience, reduce mobility associated costs and advance the transportation infrastructures, mobile sensing architectures for massive urban scanning using smartphones have been researched and proposed [RAV+11] like GeoServ [ALM11], or mobile sensor computing systems like CarTel [HBZC06], as well as vehicular ad-hoc networks (VANET) which aim to reduce fuel consumption, emissions and travel time [SB11], thus enhancing the current urban monitor.

In the city of Rome, Italy, an actual case study, which aimed to optimize urban dynamics, was performed, and marked an unprecedented monitoring of a large urban area in real-time using cell phones and a variety of sensing systems [CCL10]. The system used the LocHNESs (Localizing and Handling Network Event Systems) platform developed by Telecom Italia [18] that is a software platform for real-time evaluation of statistics, such as real-time road traffic estimation, based on the anonymous monitoring of mobile cellular networks. Most of the city of Rome was covered, and the study showed how modern digital technologies could empower citizens to make more informed decisions, by giving them a deeper knowledge of urban dynamics and more control over their environment.

2.4.2 Mobile Applications

The advancements in modern technology have permitted that today, smartphones come out of the factory with more and more sensors and different kinds of interfaces [RAV+11] at increasingly lower prices. Using these devices embedded capabilities and sensors, it is possible to use them in diversified ways, like as a mobile sensor for example.

[18] http://www.telecomitalia.com

In a time where mobility is widely discussed and has become a beacon of innovative proposals and research, a new market has been created that offers new business opportunities and means to improve the quality of life.

Currently, there are a wide variety of mobile applications that use various sensors for different purposes. Like, for example, the Portable Open Source Information Tool (POSIT) [19] for disaster management, or ecorio [20] which is an application designed to track a user's carbon footprint. Or just simply GPS navigation applications like Google Maps Navigation [21] which is a well known GPS navigation system with voice orientation that connects to the internet using Google services like Google Maps. There are also frameworks for the creation of data-driven sensor applications like Humansense [22].

2.4.2.1 POSIT

POSIT Software consists of two complementary client-server components: the POSIT-mobile client (Android/Java) and the POSITweb Server (PHP/MySQL). The goal of this application is to create a portable communication tool to aid people that can take advantage of a collaborative mapping platform [23].

This open source portable tool records information about 'Finds'. A 'Find' can be anything. From the moment a user adds a 'Find', it gives it a name and an image obtained from the camera. Then it can transmit the 'Finds' to a central server with image and location information from the GPS. It is possible to share 'Finds' with other users and communicate with them, creating a collaborative environment.

2.4.2.2 ecorio

Ecorio is the 2008 Google Android Developer Challenge winner application for Android smartphones that records GPS data to track a user's mobile carbon footprint. The objective is to track car trips and carbon outputs automatically, discover carpooling and public transit alternatives with the help of Google Transit, and ultimately share this information.

This project started in 2007 and is still in development, but since 2010 it has not seen any updates. There is currently no information about it on Google Play, what exists is a website talking about the project [24].

[19] http://code.google.com/p/posit-mobile/
[20] http://www.ecorio.org/
[21] https://play.google.com/store/apps/details?id=com.google.android.apps.maps
[22] http://jwf.github.com/Humansense-Android-App/
[23] http://posit.hfoss.org/
[24] http://www.ecorio.org/

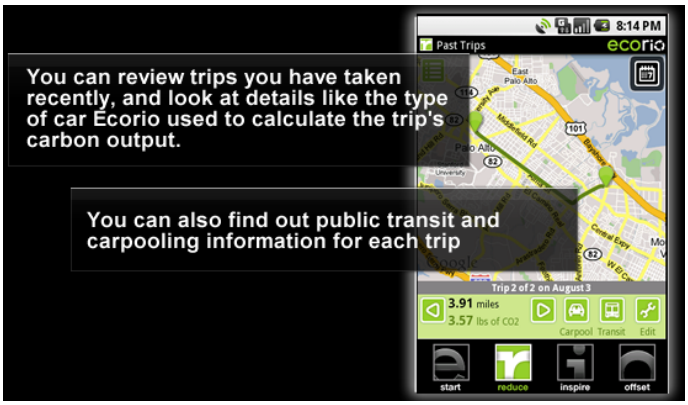


Figure 2.13: ecorio instructions

2.4.2.3 Humansense

Humansense is a data collection platform for Android that logs sensor information, like Wifi, GSM and GPS. It has the possibility of expansion via plugins that anyone can develop since it is an open source project [25]. The currently implemented plugins involve activity and location recognition. The first plugin builds models for physical activities like walking, running or jumping, to then later recognize which activity a user is peforming.

The second plugin is used to recognize places where a user spends time, based on building fingerprints from observable wifi base stations. It is a work in progress, but shows how Android smartphone's sensors can provide a multitude of uses.

2.4.2.4 greenMeter

The greenMeter application [26] is designed for the iPhone and iPod Touch and computes a vehicle's power and fuel usage characteristics in real time using the device's accelerometer. The objective is to evaluate the user's driving to increase efficiency, reduce fuel consumption and cost, and lower the environmental impact.

This application has six different graphs and three different eco-driving efficiency displays. The six available graphs include engine horsepower, fuel economy, fuel consumption, fuel cost, carbon footprint, and energy impact. The three available eco-driving efficiency displays include a gauge, leaf and a bar display and use the previous parameters to calculate the minimum and average efficiency.

The disadvantage of this product is that it requires a lot of user input and pre-calibration because it depends solely on the accelerometer, so the device must also be in a fixed position. The input includes ambient temperature to compute aerodynamic drag, fuel cost and

[25] http://jwf.github.com/Humansense-Android-App/more-info.html
[26] http://hunter.pairsite.com/greenmeter/

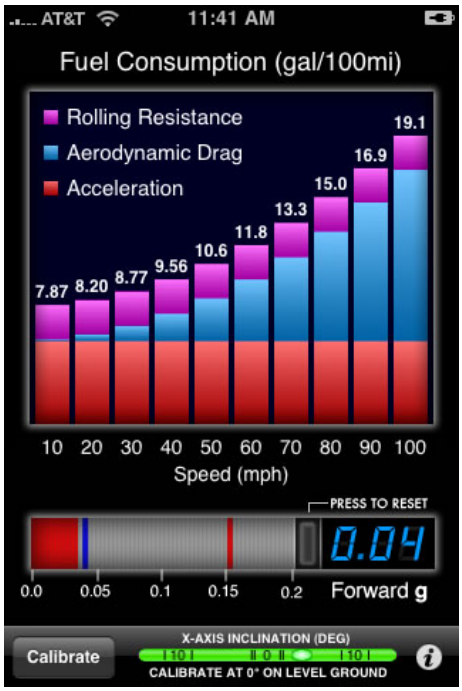


Figure 2.14: greenMeter Interface

vehicle characteristics. These in turn include gross weight, drivetrain loss, rolling resistance, drag coefficient, frontal area, engine efficiency, fuel type and a accelerometer pitch angle correction. Afterwards calibration is required to correct any accelerometer offsets in the device.

2.4.2.5 MyDrivingDroid

Currently at Instituto de Telecomunicações there is a platform named MyDrivingDroid (MDD) that consists of a mobile application for Android, a server and a website as a front-end.

The main purpose of the application is data gathering using the various sensors, like the accelerometer to measure proper acceleration, the magnetometer to measure the ambient geomagnetic field, the gyroscope to measure the rotation rate, GPS to provide location and satellite information, Wifi to log Access Points, Bluetooth to log Bluetooth devices information, and now to connect to an OBD to provide On-Board Diagnostics information. The architecture used is illustrated in Figure 2.15.

There are two main modules: the acquisition unit and the server.

The acquisition unit consists of an Android smartphone and is responsible for collecting data from the embedded sensors or from external devices, such as an OBD, storing it in a SQLite database. It is also responsible for the transparent aggregation and transmission of data to the server.

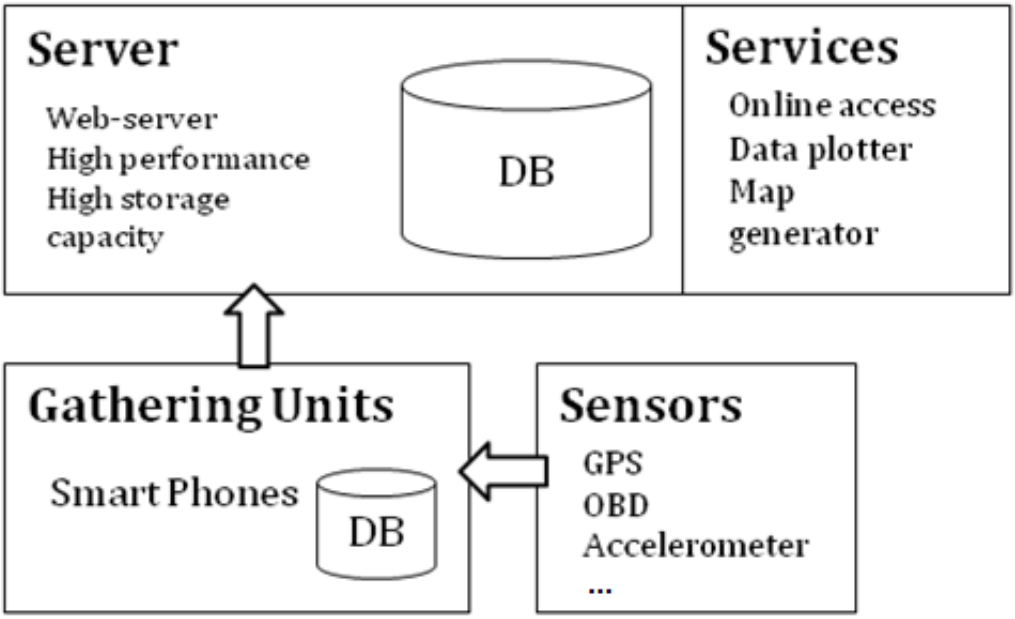


Figure 2.15: MDD Architecture

The server is responsible for verifying the integrity of the incoming data. Then it stores that data in a MySQL database and makes it available for analysis and visualization in a website as shown in Figure 2.16.

Figure 2.16: MDD Website

The goal of this project is to provide trip information, such as travel duration, the overall energy costs and carbon emissions, the areas of excessive traffic, among others. It is also intended that users share this information using this platform in a collaborative environment.

As referred in the introduction in Section 1.3, one of the goals of this thesis was to extend MDD to be able to gather vehicle data from a Bluetooth OBDII compliant device. To do so, the previously discussed OBD applications on Section 2.2 and models on Section 2.3 served as important references. This is further addressed in Chapter 5.

2.4.3 Overview

The advent of ubiquitous connectivity and powerful, affordable, user-friendly mobile devices offers a unique opportunity to leverage information and communication technologies to improve quality of life.

While there are several Android projects that use sensing, none were found that use only the GPS sensor to estimate fuel consumption. Furthermore, MDD offers a great opportunity for expansion since each sensor runs on an individual service. This means that as technology evolves, adding a new sensor is as simply as adding a new service (see Chapter 5 for the MDD architecture). Also, the MDD project can be used in a single or in a collaborative way exploiting the advantages that collaborative platforms offer.

2.5 UI Patterns

Anyone developing mobile applications intended to function while a user is driving, should be extremely cautious about user interfaces. Not only it is dangerous to operate a phone while driving [Gom12], it is also illegal in some countries, like Portugal [27]. That is why some vehicles provide embedded phones or an optional phone dock.

Taking those facts in account, 'Car Mode' mobile applications are available, that use simple interfaces mostly destined to be used while the smartphone is in a dock. They should abide some standard UI patterns [28] for quick interaction.

When developing the solution, this will be accounted for.

2.6 Conclusions

In this chapter, it is possible to learn the differences between vehicle engines and the reasons behind the appearance of OBD standards, and draw some conclusions about the

[27] http://www.ansr.pt/
[28] http://developer.android.com/design/index.html

necessity of emission and fuel models, not only for the environmental impact but also because of the impact they can have on a person's wallet.

There are some On-Board Diagnostics applications that effectively in the long-term can save money, though they possess the handicap that not all users want to or know how to use OBDs. Furthermore, they are focused on individuals and not on a collaborative perspective, so they are mostly for informative than comparative purposes, which is something the proposed solution can eventually provide using the power of mobile platforms.

Chapter 3

Problem

In Chapter 1 it is referred some of the problems caused by mobility that a user faces. One of those is the difficulty of knowing how much fuel a mobility option requires so a person can translate it to costs. So if a person wants to know his fuel consumption, either he owns and understands the OBD technology, and has the availability and knowledge to install an OBD interface in his vehicle, or he uses a software, like the ones presented in Chapter 2, that normally require user input. So, in the end, most of the offers require user intervention. On the other hand, the OBD technology has the limitation that not all vehicles support it. Also it is not as widespread as the mobile technology in the present day. In modern societies it is increasingly more common a person owning a smartphone than owning an OBD device, as this last technology is not usually well known by the average user. So the OBD technology can serve an important, but nevertheless, less popular purpose in our society than the mobile technology. Alo, no fuel consumption parameter is given by the OBD, so most of these softwares use undisclosed fuel consumption formulas, which is something this thesis provides in Chapter 5.

It is also important to note that *time* in a fuel consumption calculation is a very important variable, and it is known that not all vehicles have the same response time. This is also validated in Chapter 5. This means that not all vehicles are capable of providing all the variables necessary to calculate fuel consumption in a one second window.

So in order to better control mobility costs, very few free options are available. As referred in Chapter 2, the current solutions are either paid and/or depend on the OBD technology. Also not all vehicles posses an interface with trip assessment information, and even if they do, a user has little chance of using that information in a collaborative fashion. The uploaded aggregated data from different vehicles creates lots of opportunities. Besides contributing to an enhancement of the urban monitor, the usage of this data can eventually help the individual user, as suggested in Chapter 7.

Problem

Chapter 4

Proposed Solution

The proposed solution is an algorithm that estimates fuel consumption from GPS data. The GPS technology, besides its ubiquitous availability, is widely available in many devices, like the modern smartphones. In order to achieve this goal, and know how the GPS data relates to fuel consumption, a dataset of second by second fuel consumption from vehicles is needed along with the corresponding second by second GPS data. Thinking in mathematical terms, what is needed is a statistical technique to estimating the relationships among the dependent variable, fuel consumption, and the independent variables from GPS data.

The first step was to understand the GPS technology and what values could be derived from it. Next it was needed to know how fuel consumption could be obtained, and this is where the OBD technology came in. An OBD device connected to a vehicle transmits engine information in real time through a Bluetooth connection. However, as stated in Chapter 2, since the OBD protocol does not offer a specific fuel consumption parameter, it was also necessary to obtain a fuel consumption formula from the available OBD data. So the OBD protocol was also studied in order to know what values a vehicle can provide and how can they be fetched. With those values, emission and fuel consumption models were also studied, so to become possible to know what formula to use.

Furthermore, a gathering unit was needed and developed to log OBD and GPS data at the same time. The MDD application for Android was chosen because it is a data gathering application with the opportunity of expansion. To expand the OBD module, research into the Bluetooth communications protocols was also required. By using this data gathering unit, it becomes possible to construct the required dataset. But in order to obtain a dataset large enough to extract conclusions, some volunteers were needed. The more data, and more volunteers, and more vehicles the better. Unfortunately, this was a difficult task because, as stated, not all vehicles support the OBD protocol. As data became available, so did the goal of creating an algorithm that estimates fuel consumption from GPS data through regression analysis, became possible.

So the implemented algorithm consists of two modules that estimate fuel consumption. One that uses gathered OBD data, and another that uses only embedded GPS sensor data. The OBD dataset is used for the calibration of the other module. The approach is presented with the help of the Figures 4.1 and 4.2.

Figure 4.1: Model

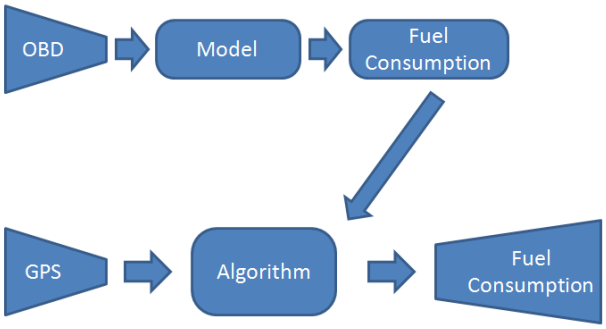


Figure 4.2: Detailed Model

Since the data gathering step had to include the broadest public possible, MDD was required to run on a wide range of smartphones. This topic, the process of gathering volunteers, along with the GPS and OBD technology and the module implemented in MyDrivingDroid are detailed in Chapter 5.

Chapter 5

Obtaining Data

This chapter details the implementation, data collection methodologies and the data processing steps. The beginning of each of these three sections summarizes the respective topic.

5.1 Implementation

In this section the choice over the Android's embedded sensors is discussed and the OBD communication protocols are detailed, as well as the values used from the OBD standards to calculate fuel consumption. The approach used in developing the fuel consumption formulas is also explained in detail, and a background in vehicle mechanics is presented to support the explanation. The integration and implementation details of the module added to MyDrivingDroid to gather OBD data via Bluetooth is also detailed. Finally, an overview of the implementation step is presented.

5.1.1 Android Sensors

As mentioned before, the goal of this thesis was to develop an algorithm that estimates fuel consumption from sensor data. Particularly from the GPS sensor present in today's mobile phones. To do so, it is also important to understand what results are possible to deduct from a sensor or a group of sensors.

Most Android-powered devices have built-in sensors capable of providing raw data with high precision and accuracy. While a device can have more than one sensor of a given type, very few have every type of supported sensors embedded. Nevertheless, most of the Android devices have a built-in accelerometer, magnetometer and a GPS [1].

[1] http://developer.android.com/guide/topics/sensors/sensors_overview.html

The Android platform supports motion, position and environmental sensors. While most of these sensors are hardware-based, some are software-based. The hardware-based sensors are physical components that directly measure specific environmental properties to derive data. Software-based are virtual sensors that mimic hardware-based sensors by deriving data from one or more of the hardware-based sensors.

The motion sensors include accelerometers, gyroscopes and virtual gravity and rotational vector sensors to pinpoint and measure acceleration and rotational forces along the three axes. The position sensors include GPS, magnetometers and virtual orientation sensors to measure the physical position of a device. The environmental sensors include barometers, photometers and thermometers to measure illumination, humidity, ambient air temperature and pressure.

The following subsections discuss how and what is possible to obtain from some of the Android's motion and position sensors, and why only the GPS sensor was chosen. Some very useful formulas and documentation on how to handle sensor data on an Android can be consulted at the Android Developers webpage [2]. For this project, the most useful ones are referenced below.

5.1.1.1 GPS

The Global Positioning System (GPS) [3] is a space-based satellite navigation system that provides location, time and satellite information anywhere on Earth. It is freely accessible by anyone with a GPS receiver, which is built-in in any present Android smartphone. With the GPS sensor with is possible to know a user's location information about latitude and longitude in degrees, altitude in meters and bearing in degrees to the north, also known as azimuth, speed in meters per second and a corresponding GPS timestamp.

The GPS system consists of three modules. The satellites that transmit the position information, the receiver that collects data from the satellites and computes its location based on that data, and the ground stations that control and monitor the satellites and update their information guaranteeing their health and atomic clock's accuracy. To compute a location, the GPS receiver synchronizes with the available satellites and downloads the navigation information, consisting of the satellite's atomic clock information, ionosphere data, and orbit (ephemeris) data.

To maintain a fix, the GPS receiver continuously recalculates the information from the moving satellites. Once it has a fix from sufficient satellites, it is possible to derive much more information than just location or time data, like travel direction (compass heading), distance and speed.

The GPS mechanism is very complex. The book *Understanding GPS* by Kaplan and Hegarty [KH97] is advised to deepen the knowledge of this technological marvel. It

[2] http://developer.android.com/reference/android/hardware/SensorEvent.html
[3] http://www.af.mil/information/factsheets/factsheet.asp?id=119

explains that location and velocity are obtained differently, and velocity is actually more precise as it is obtained by the Doppler effect [ZZGD06]. So velocity provides better accuracy for acceleration calculations (see Section 5.3.4 for details) than position.

There are various methods and algorithms to compute user location, and they are already implemented in the Android OS. Browsing the Android's Location Manager source code it is possible to discover an implementation of the Inverse Formula [Vin75] to compute horizontal distance and bearing between latitude and longitude points.

However, there are also some other methods implemented by specific manufacturers and for specific ROMs. This lead to some encounters with firmware bugs. Although the position and velocity information on the tested devices presented no problems, on some Android smartphones the GPS timestamp showed a one day difference on leap years, like the year 2012. Since there are no guarantees about the smartphone's system timestamp reliability (as shown on Section 5.3.3), to correct this bug, a NMEA Listener was also implemented to receive raw NMEA sentences from the GPS to correct time. NMEA is a standard for communicating with marine electronic devices and is a common method for receiving data from a GPS over a serial port [NME02]. So if the GPS timestamp has a bigger difference then a one day offset plus the smartphone's timestamp and a one day offset plus the NMEA timestamp, it is corrected by subtracting a day's equivalent time.

Satellites have an atomic clock to keep the time very precisely that is used in the location calculations. As previously referred, one of the applications of GPS technology is also to provide the correct time. Still, these calculations are likely to have an error margin. An accuracy error estimative is reported by the Android OS that helps to filter undesired data points. The most common error is receiver clock bias, which affects pseudorange measurements. This can be compensated by additional satellites. Also another common errors are the ionospheric delay due to the sun's radiation, receiver noise and multipath that occurs when the original GPS signal is reflected before it reaches the receiver [Wan09].

Still, the GPS is the most accurate sensor on position and velocity information, but it only works outdoors and it takes time to obtain a fix. In Android devices it also quickly consumes battery power, so there is a possibility to determine user location using cell tower and Wi-Fi signals, providing location information in a way that works both indoors and outdoors, responds faster, and uses less battery power.

However, the Network Location Provider cannot continually pinpoint the exact location and speed that is crucial for this work. So only GPS based locations are used. Although, most of the time spent driving is in an outdoor environment, this means that no data can be provided when the driver is, for example, in a tunnel. In this case, another approach is needed. But because there are always at least four GPS satellites available at any given time and position, since July of 1995, the outdoor data can always be gathered and proved to be sufficient.

5.1.1.2 Accelerometer, Magnetometer And Gyroscope

To obtain data when there is no GPS available, other approaches were though of, like combining the motion and position Android sensors, namely the accelerometer, magnetometer and gyroscope.

In Android smartphones the coordinate-system is defined relative to the centre of the screen of the phone in its default orientation (see Figure 5.1).

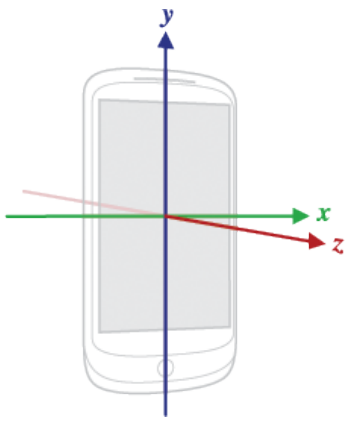


Figure 5.1: Coordinate System

In Android smartphones the accelerometer measures the physical acceleration applied to the device's accelerometer sensor as seen in Formula 5.1. Since the force of gravity is always influencing the measured acceleration applied to the device, this acceleration can also be viewed as Formula 5.2.

$$Ad = -\sum Fs/mass \tag{5.1}$$

$$Ad = -g - \sum F/mass \tag{5.2}$$

Ad - acceleration applied to the device
Fs - forces applied to the sensor itself
g - standard value of gravitational acceleration (9.81 m/s^2)

So when the device is stationary on a table it will have an acceleration value of g. In order to measure the real acceleration of the device, the contribution of the force of gravity must be eliminated by applying a high-pass filter. Conversely, a low-pass filter can be used to isolate the force of gravity.

The gyroscope measures the rate or rotation in rad/s around a device's axis, providing raw rotational data which means that noise and drift will introduce errors that need to be compensated with other sensors, such as an accelerometer.

The magnetometer monitors changes in the magnetic fields around the user, providing raw magnetic field strength data for each of the three coordinate axes. In conjunction with motion sensors it is possible to obtain azimuth and geomagnetic inclination data.

One of the main challenges in mobile devices is making the most out of battery life, so these sensors are a good alternative since they use much less power than the GPS, especially the accelerometer [4].

However, some Android smartphones shut down their sensors to conserve power, which presented a challenge in the development and is unfortunate for this work's scope. *Wake Locks* exist to prevent that event. This mechanism is used in Android to indicate that the application needs to have the device on. However there is a problem. Depending on the ROM installed in the smartphone, because power management is a sensitive operation normally managed by the system, sometimes a wake lock cannot override those system instructions, even with the proper permissions. So, it is possible to implement wake locks, but there are no guarantee it will work on every Android. As long as the screen is on or the smartphone is active, the sensors are also active. But when the smartphone or the user turns off the screen and the system goes to sleep, usually the system orders the sensors to shut down. So, being that the only guarantee is when the screen is on, which consumes a lot of power, using only these sensors becomes the saying *'cheap comes out expensive'*. Also, the device should be fixed and calibrated to obtain data usable for movement and position estimation from these sensors, as the device moving about inside the vehicle would provide very noisy, ergo irrelevant data, and calibration is needed from a starting point or other points of reference so it is possible to calculate the relationship between points, and this is not user friendly at all. And even after these steps, there can be a lot of noise in the sensor data [ABH12] (bumps in the road, for example). With these restrictions and being that user experience is important for the Android component to be used, the best choice was the GPS only.

5.1.2 The OBD Device And Protocol

The ELM327 OBD To RS232 Interpreter is an OBD device designed to act as a bridge between the OBD ports and a standard RS232 interface [ELM11]. Since its launch into the market, a number of clones have surfaced, which are cheaper, although some work better than others. These clone devices (Figure 5.2) were used to gather automotive data by interpreting the OBD protocols. Various clones from different sellers were acquired and tested, also to ensure maximum compatibility, which led to some challenges with the vehicle's communication that are discussed below.

To connect the OBD device to a vehicle the first step is to locate the OBD socket. For most cars, it is located below the steering wheel, sometimes hidden within a compartment

[4] http://developer.android.com/guide/topics/sensors/sensors_overview.html

Figure 5.2: ELM327 clones

(see Figure 5.3). An OBDII connector is mandatory on vehicles sold in the United States since 1996 and in Europe since 2001 for petrol engines and 2004 for the diesel counterpart. Some manufacturers started implementing it earlier, however, not all early OBDII compliant vehicles are fully compatible with the modern standards.

Figure 5.3: Mounting an OBD device to a Peugeot 207

There are five OBD communication protocols. Four non-CAN and one CAN. CAN is a vehicle bus standard. The four non-CAN are SAE J1850 PWM, SAE J1850 VPW, ISO 9141-2, ISO 14230-4 KWP, and the CAN is ISO 15765-4/SAE J2480. The last protocol that uses the CAN standard is the most modern and it is used since 2003. So most modern vehicles use it (in the United States is actually mandatory since 2008). The OBD protocol SAE J1979 and all the previous standards can be consulted at the Society of Automotive Engineers (SAE) International website [5], which is a standards setting organization for technical engineering standards. As a general rule, it is possible to determine the protocol used by a vehicle by looking at the pinout of the OBDII connector. Each protocol has its variants, although not very relevant for the scope of this work, it is important to note that there are slight differences mainly between the CAN and non-CAN protocols.

On-Board Diagnostics systems are designed to be very flexible, providing a means for several devices to communicate with one another [ELM11]. The main difference in message format between CAN and non-CAN standards are the header bytes and an extra PCI byte in the data bytes in the CAN OBD messages, like shown in Figures 5.4 and 5.5. Another difference is the multiline response.

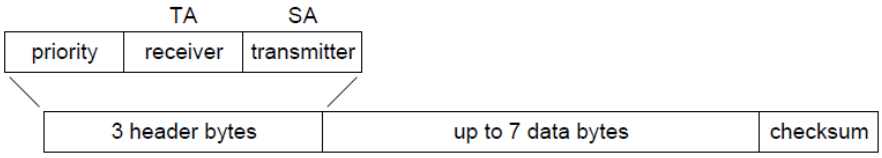


Figure 5.4: OBD Message Format

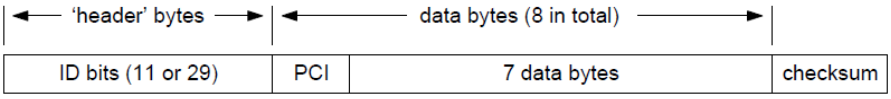


Figure 5.5: CAN OBD Message Format

The OBD data bytes are encapsulated within a message, with header bytes at the beginning, data in the middle and a checksum at the end. The non-CAN SAE J1850, ISO 9141-2 and ISO 14230-4 protocols all use essentially the same structure. A typical OBD message structure uses 3 header bytes to provide details concerning the priority, the Target Address (TA) and the Source Address (SA), a maximum of seven data bytes and one checksum byte. The ISO 15765-4 (CAN) protocol uses a very similar structure except the header bytes are called *ID bits* and the early CAN standards use 11, while the more recent allow for either 11 or 29 bit IDs. To detect errors in the transmission the various

[5] *http://www.sae.org/*

protocols all provide some form of check on the received data like Cyclic Redundancy Check (CRC).

The ELM327 devices and tested clones support all of the above protocols and also automatically fill the header bytes and perform the checksum calculation, unless it is stated otherwise and defined by the user. So the user usually only has to worry about the data bytes which are the commands. There are two types of commands, the ones intended for the ELM327's internal use that begin with the characters *AT*, and the OBD commands to exchange messages with a vehicle, that are only allowed to contain the ASCII codes for hexadecimal digits (0 to 9 and A to F). The first two bytes of an OBD command refer to the modes of operation. There are ten standard modes. Vehicles are not required to support all of them and they may also define their own additional modes, provided that none of the additional modes has the same hex value as the default ones. The modes used to gather relevant data for this work were *mode 01* to log engine data in real time, and *mode 09* to request vehicle information like fuel type (unfortunately not available on all tested cars), and the Vehicle Identification Number (VIN), which is the vehicle's unique serial number. Following the mode hex value, a Parameter ID (PID) hex value indicates the information a user wants to obtain, like a specific sensor data or its availability. Like the modes, vehicles are not required to support all possible PIDs, that is why within each mode, the PID 00 is reserved to show which of the mode's PIDs are supported. Because of the disparity of available PIDs between vehicles, multiple formulas for the calculation of fuel consumption were devised that use different variables.

5.1.2.1 The OBD Integration

The full standard PIDs are defined by SAE J1979 [SAE] and ISO 15031-5 [ISO], as well as the description and expected response for each PID. It also explains how to translate the response into meaningful data, since a response from a specific request is not always a direct decimal conversion. For example, if the user issues the command 010C (mode 01, PID 0C) to ask for the current engine RPM, the expected response returned is composed of two bytes. Assuming the letters A and B represent the decimal equivalent of the first and second bytes of data correspondingly, the real decimal value is given by the formula ((A*256)+B)/4 for this specific PID. Each PID has its specific expression. And the unit in which the value is returned must always be taken into account (for temperature is Celsius, Fuel Pressure is in Kilopascal, etc.) for calculation purposes.

The ELM327 manual states that to indicate the end of a command, it must be terminated with a carriage return character (\r or hex 0D). However, some clones detect an incomplete string in the OBD commands unless a linefeed character (\n or hex 0A) is added after the carriage return. When this happens, either an internal timer automatically aborts the incomplete message after some time, and a single question mark ('?') is

printed, or in the case of a continuous input of commands, the buffer overflows and prints an 'ERROR' or '7F' message. So at the end of all commands the characters '\r\n' were always added to resolve this issue.

After the response, the ELM327's prompt character ('>' or hex 3E) is shown. This indicates that the device is in the idle state and ready to receive new commands. In the tested clones, however, after a continuous input of commands, the responses in some vehicles were not always ordered. Adjusting the baud rate of the clones in an attempt to solve this issue resulted in error, as some clones do not allow this operation. The answer in resolving this problem is to look at the full response. Unless omitted by the user with the *AT* commands, the first bytes in the response indicate the requested mode and PID. Using the previous example, by issuing the command 010C for RPM data, the hex response can be 41 0C 2E E0 3E. The first byte 41 indicates mode 01, as the responses for the modes start with 4 (42 for mode 02, 43 for mode 03, and so on), the second byte indicates the PID 0C for engine RPM and the two following bytes indicate the response. In this case, hex 2E is decimal 46 and hex E0 is 224. So ((46*256)+224)/4 equals 3000 RPM. The last byte (hex 3E) is the prompt character which indicates the end of the response. By knowing which sensor is responding, the response order then becomes irrelevant.

As noted in the previous chapter, time is a very important issue on this specific data mining process, and not all vehicles have the same response time. So not all vehicles are capable of providing all the variables necessary in a one second window with the default settings. In an attempt to accelerate the response, the OBD devices were configured to cut some bits in the message format, leaving only the essential. This is done with the *AT* commands. Using again the example for the 010C command, the longest response can be something like 01 0C 48 6B 10 41 0C 2E E0 3E 0A. The first thing to do is to reset the device since its previous state is unknown. Then set the device to cut the echo responses, which are first 2 bytes that indicate the command submitted by the user, then the headers, which are the next 3 bytes that include the ECU that responded to the request, then cut the line feed character (hex 0A), and finally cut the spaces. So now the device responds the hex 410C2EE03E for the command 010C, which is shorter and faster. Another speed improvement, was changing the commands schedule. As soon as a response is received, the next queued query command is sent, and only then the received command is processed. This way, the response parsing does not slow down the OBD communication. Nevertheless, it is not guaranteed that all the required data is available in all seconds. In this case interpolation is used. This is discussed in Section 5.3.2. The figure below shows speed of response on the tested vehicles after these optimizations (Figure 5.6).

Unfortunately, not all clones recognize all AT commands, which means that the response parser has to be ready for every possible response. This is achieved using regular expressions. Also, another difference in the CAN and non-CAN standards is the multiline

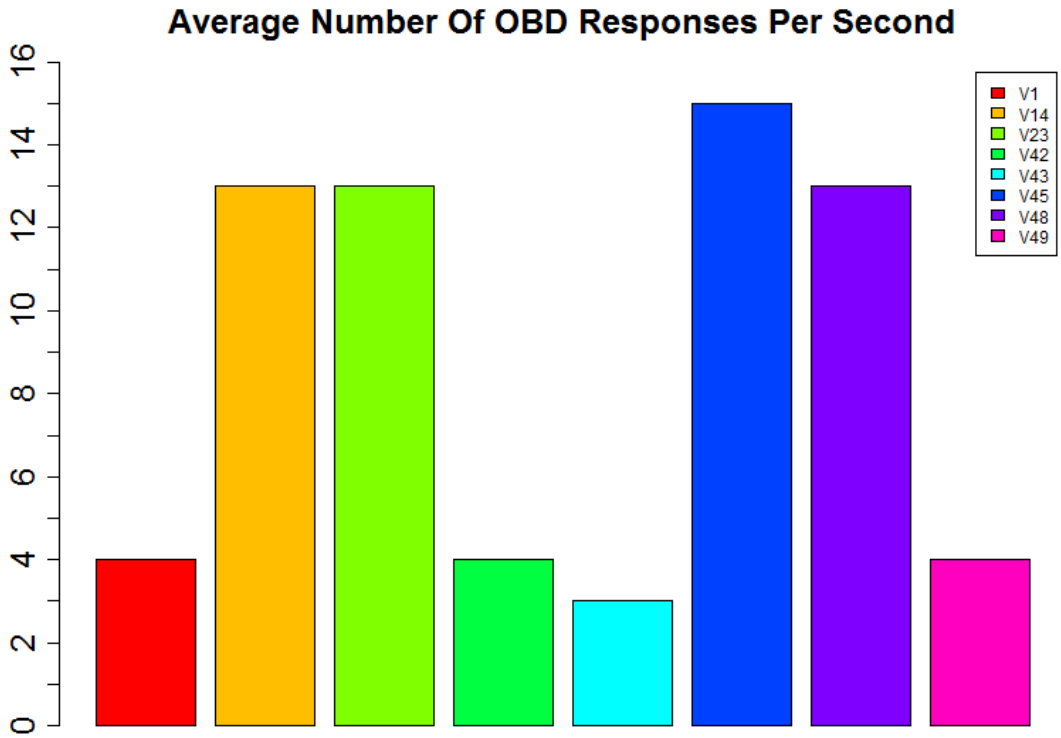


Figure 5.6: Average Number Of OBD Responses Per Second In Various Vehicles

responses as stated before. The details can be seen on the 'Multiline Responses' chapter in the ELM327 documentation [ELM11]. This was also resolved with regular expressions.

5.1.3 Fuel Consumption Calculation

Before expanding MyDrivingDroid with a service to gather OBD data, it is important to understand how engines and the OBD protocol work, and what is the data needed to calculate fuel consumption, since there are hundreds of codes on the OBD standard. This was done using the information obtained from the state of the art research as a guideline reference.

The purpose of an engine is to convert fuel into motion. In this case so that the vehicle moves. There are two kinds of combustion engines, external (like steam engines) and internal like petrol or diesel engines. Currently, most of the vehicles have these two types of internal combustion engines that use a four-stroke combustion cycle to convert fuel into motion. The four-stroke approach is also known as the Otto cycle, and correspond in order of execution to the intake stroke, compression stroke, combustion stroke and exhaust stroke (see Figure 5.7). The algorithm is aimed at this type of engines.

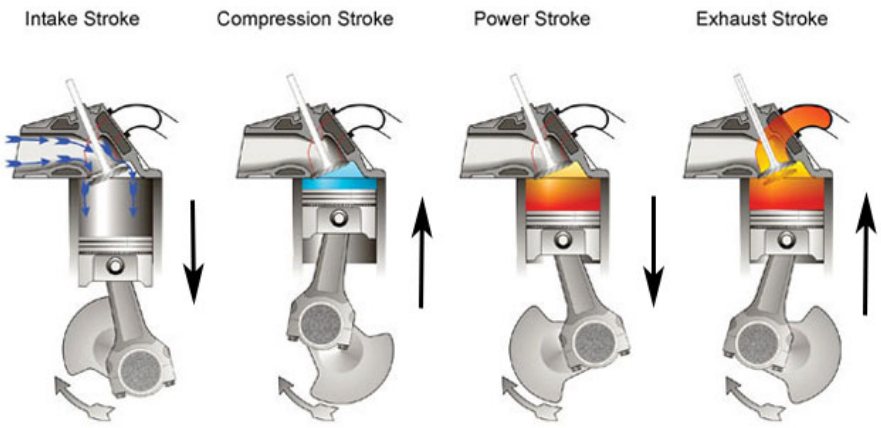


Figure 5.7: Four-Stroke Cycle

Fuel consumption, also known as fuel efficiency, is a ratio of fuel consumed per distance travelled. In most European vehicles it is normally represented as litres per 100 kilometres. The fuel consumed, also known as fuel flow or fuel rate, means how many litres are consumed per hour. So, at a given speed in kilometres per hour, to calculate fuel consumption the expression below is used.

$$FuelConsumption(l/100km) = \frac{FuelFlow(l/h)}{Speed(km/h)} * 100 \quad (5.3)$$

As stated before, the OBD standard does not explicitly include a fuel economy parameter, but includes PIDs for engine fuel flow (code 015E) and speed (code 010D). However, while speed is mandatorily available, fuel flow is not. Either because the manufacturer chooses not to make it available, or because there is no sensor inserted in the fuel line between the fuel tank and the carburettor of the engine to measure litres per hour. In most of the cases this information is not available, as was the unfortunate case of all the tested cars. So a fuel flow expression is needed.

The principle behind an internal combustion engine is that by putting fuel in a small enclosed space and igniting it, an amount of energy is released in the form of expanding gas. In order for combustion to occur, however, a fuel and an oxidant are needed to be burnt for this chemical reactions to take place. The difference between diesel and gasoline engines, besides the type of fuel, is that with gasoline engines, fuel is mixed

with air, compressed by pistons and ignited by a spark plug. In a diesel engine the air is compressed first, which causes it to heat up, and then when the fuel is injected it ignites.

It is important to know that there is a ratio for how much a certain type of fuel can be burnt with a certain amount of oxygen. So the air-fuel ratio in the cylinder is a limiting factor. If there is excess oxygen, it is called lean combustion. When there is an excess of fuel, it is a rich combustion. In an incomplete combustion not all the excess fuel is burnt and it will be pushed out through the exhaust valve. The 'perfect' ratio for both the fuel and the oxygen in the air to be completely consumed is called the stoichiometric air-fuel ratio. In order to be able to judge if an air-fuel mixture has the correct ratio of air and fuel, the composition of a fuel has also to be known. So when there is no information about the amount of fuel entering the combustion chamber (fuel flow in litres per hour), it can be determined by using the instantaneous rate of air entering the combustion chamber, if the air to fuel ratio and fuel density is known.

In the OBD standards there is a PID for Mass Air Flow or MAF (code 0110) that gives the mass of air read by the sensor placed just before the intake manifold in grams per second. So it is possible to use the formula below to determine fuel flow.

$$\frac{MAF * 3600}{FF} = AFR_A * FD <=> FF = \frac{MAF * 3600}{AFR_A * FD} \tag{5.4}$$

FF - fuel flow (l/h)
MAF - mass air flow (g/s)
AFR_A - actual air to fuel ratio
FD - fuel density (g/l)

Since diesel engines operate at a higher compression ratio, the mass air flow has to be adjusted. Both spark and compression ignition engines support Calculated Engine Load (code 0104) that is the relative cylinder charge. This value is the airflow divided by the peak air flow at wide open throttle in standard temperature (25 ^{o}C) and pressure (101.325 kPa) as a function of RPM. For diesel engines, it is the current output torque divided by peak output torque at current RPM. Engine Load is linearly correlated with engine vacuum. So there is linearity between load, airflow and torque. So for diesel engines MAF becomes:

$$MAF_e = MAF * LOAD_{CALC} \tag{5.5}$$

MAF_e - adjusted amount of air that is actually used to burn fuel (g/s)
MAF - mass air flow (g/s)
$LOAD_{CALC}$ - calculated load

Fuel density and the stoichiometric air to fuel ratio are constants that depend on fuel type (see Table 5.1 for stoich ratios). Although, as seen before, the actual air to fuel ratio in the engine is not always stoichiometric. Sometimes the are lean or rich mixtures. To correct this ratio, a lambda value is used based on corrections obtained from the Fuel Trim, which is generally calculated by using a wide set of data values. These include oxygen (O2) sensors, intake air temperature and pressure, coolant temperature, anti-knock sensors, engine load, changes in throttle position, and even battery voltage.

$$AFR_A = AFR_S + (AFR_S * (Lambda/100)) \tag{5.6}$$

AFR_A - actual air to fuel ratio

AFR_S - stoichiometric air to fuel ratio

Lambda - correction value

Table 5.1: Stoichiometric Ratios For Common Fuel Types

Fuel Type	**Stoich Ratio**
Gasoline	14.7
Diesel	14.6
Propane	15.5
Ethanol	9.0
Methanol	6.4

Lambda is a tricky value to obtain, and its calculation is quite different between engine types [AFO]. There are two modes of operation in an engine, that are *Open Loop* and *Closed Loop*. When the engine is first started, the system goes into *Open Loop* operation. It stays in this mode until the engine 'warms up', which means until the coolant sensor detects a specified temperature has been reached, or a specific amount of time has elapsed after the engine was started. After that, it enters in *Closed Loop* operation.

There are PIDs in the OBD standard to obtain the lambda correction for each specific group of cylinders (also known as banks) in a specific mode of operation. The *Long Term Fuel Trim* PID (codes 0107 for bank 1 and 0109 for bank 2) indicates the correction being used by the fuel control system in both open and closed loop modes of operation. The *Short Term Fuel Trim* PID (codes 0106 for bank 1 and 0108 for bank 2) indicates the correction being used by the closed loop fuel algorithm. In open loop it is more difficult to obtain correct readings. The lambda value obtained ranges between -100 and 100. If the value is 0, the actual air to fuel ratio is stoichiometric, less than zero means a rich condition, bigger than zero is a lean condition.

The OBD protocol has also a PID to obtain the fuel system status (code 0103) to know the system's mode of operation, so it is possible to choose the lambda value from the corresponding Fuel Trim. Unfortunately, as with many PIDs, this one is not always available. If that is the case, the workaround can be a time solution. About five minutes of operation is an acceptable time for engines to enter *Closed Loop*. When the lambda correction value is also not available, the stoichiometric air to fuel ratio is used for calculation (lambda is zero).

So the main PID now becomes the MAF value. But as with the fuel flow measurement, the manufacturer can choose not to make it available, or there is no sensor placed before the intake manifold, and so it is not possible to measure the instantaneous rate of air entering the combustion chamber. So if MAF is not available, another formula to calculate the mass of air is needed.

By looking at the Ideal Gas Law, a relation is shown with the amount of air to its pressure, volume and temperature. An ideal gas obeys the relationship below.

$$P * V = n * R * T \tag{5.7}$$

Where P is pressure in Pascal, V is volume in m^3, n is the number of moles, R is the ideal gas constant (J/K*mol) and T is the absolute temperature in Kelvin. The number of moles can also be translated by the following formula.

$$n = \frac{m}{MM} \tag{5.8}$$

Where n is the number of moles, m the mass of the gas in grams and MM the molar mass in grams per mol. To obtain the mass of air, the formula can be rearranged as shown below.

$$m_{air} = \frac{P * V}{R * T} * MM_{air} \tag{5.9}$$

Where m_{air} is the mass of the air in grams and MM_{air} is the molar mass of air in grams/mol.

As stated before in this section, the algorithm is aimed at four-stroke engine types. The first piston stroke is the intake stroke were a mixture of fuel and air, in the case of petrol engines, and just air in a diesel engine (as fuel is injected afterwards), is forced by pressure into the cylinder through the intake port. The intake valve then closes and the compression stroke takes place. The volume of air and fuel mixture in the cylinder relative to the maximum volume of the cylinder is called the engine's volumetric efficiency.

In the combustion chamber system, the pressure (P) inside the cylinder is a product of the pressure in the manifold and volumetric efficiency. The intake manifold absolute pressure (code 010B) and the intake air temperature (code 010F) are available in the OBD protocol.

$$P = MAP * VE \tag{5.10}$$

Where MAP is the intake manifold absolute pressure and VE the volumetric efficiency.

Considering that the volume (V) is known from the displacement of the engine, it is now possible to calculate the mass of air (m) in the cylinder proportional to the number of moles of air.

$$m_{air} = \frac{(MAP * VE) * ED}{R * (IAT + 273.15)} * MM_{air} \tag{5.11}$$

The manifold absolute pressure (MAP) is in kilopascal, the intake air temperature (IAT) is in Celsius, and ED is the cylinder displacement in litres.

Knowing the mass of the air, the next step is to determine the rate at which it enters the combustion chamber. The OBD standards states that the air mass per intake stroke is a function of revolutions per minute (RPM), and the maximum air mass is a constant for a given cylinder swept volume.

$$AM = \frac{AM_T}{(RPM/60) * (S * C)} \tag{5.12}$$

$$AM_T = \rho * ED \tag{5.13}$$

Where AM is the air mass in grams per intake stroke, AM_T is the total engine air mass in grams per second, RPM is revolutions per minute, S is the strokes per revolution and C the number of cylinders. The ρ symbol represents the density of air in grams per intake stroke.

The air mass is used to calculate the Absolute Load (code 0143), present in the OBD standard, that is the normalised value of air mass per intake stroke displayed as a percent. This translates to the current air mass and pressure divided by the maximum air mass at standard temperature (25 ^{o}C) and atmospheric pressure (101.325 kPa), and at 100% volumetric efficiency at wide open throttle. Absolute Load can then be calculated with the formula below.

$$LOAD_{ABS} = \frac{AM}{AM_T} \tag{5.14}$$

Where $LOAD_{ABS}$ is the absolute load.

Absolute Load is an indicator of the pumping efficiency of the engine, and at the peak value correlates with volumetric efficiency. A four-stroke engine has 2 revolutions per intake stroke. Looking again at the Ideal Gas Law, the rate of air considering the maximum air mass at 100% volumetric efficiency and at standard temperature and pressure can then be calculated.

$$MAF = \frac{P * ED}{R * (T + 273.15)} * MM_{air} * LOAD_{ABS} * (RPM/60)/2 \tag{5.15}$$

Where MAF is mass air flow grams per second, P is the standard atmospheric pressure in kilopascal and T is the standard air temperature converted to Celsius.

Spark ignition engines are required to support absolute load, while compression ignition engines (like diesel) are not required to do so. So compression engines will not be able to use this formula. If that is the case, it is possible to use the same formula considering the intake manifold absolute pressure (MAP) and the intake air temperature (IAT), but volumetric efficiency (VE) becomes an incognito.

$$MAF = \frac{(MAP * VE) * ED}{R * (IAT + 273.15)} * MM_{air} * (RPM/60)/2 \tag{5.16}$$

Volumetric efficiency is the measurement of how close the actual volumetric flow rate is to the theoretical volumetric flow rate. It is very difficult for an engine to use the full volume of a cylinder since there can be friction losses or leaks, or other factors [PDP+12]. The theoretical volumetric flow rate is a function of RPM and engine displacement at full volumetric efficiency (100%). The actual volumetric flow rate is a function of the mass flow rate and the density of the intake air. Even knowing that volumetric efficiency depends on RPM, it is not guaranteed to be the same among different vehicles. So since not enough data is supplied for the calculations, the approach used was to set volumetric efficiency at a value of 75% when unknown, which is a reasonable value for most common engines [Iri10].

There is a PID for fuel type (code 0151) but it is rarely available, so only engine displacement, and fuel type if not available, must be obtained from the user, while all the other values are extracted by the OBD.

So to calculate fuel consumption, a series of conditional steps are performed. First the

algorithm tests if speed is available, since with no speed it is not possible to obtain litres per 100 kilometres. Then checks if fuel flow is available. If it is, then the formula is direct (Formula 5.3), if not, it is calculated through Formula 5.4. In this case the algorithm first searches for a Lambda value to correct the stoichiometric air to fuel ratio constant. If no lambda is available then the stoichiometric value is used. Next the algorithm searches for a MAF value. If it is available then the fuel flow can be calculated (Formula 5.4), but first, it checks if the fuel type is diesel to correct the mass air flow with Formula 5.5. If no MAF is provided by the OBD, it has to be calculated. So now there are two ways to calculate MAF: Equation 5.15 and 5.16 and they both need RPM. So in this step, if RPM is not available, then fuel consumption cannot be calculated. If it is available, then the algorithm checks if Absolute Load is available to calculate MAF using Formula 5.15. If Absolute Load is not available, then Equation 5.16 is used with MAP and IAT if both are available. Otherwise, fuel consumption cannot be calculated. If MAF can be calculated with one of these equations, the algorithm checks again if the vehicle is a diesel type to adjust MAF with Formula 5.5. So now it is possible to calculate fuel flow with Formula 5.4 and finally fuel consumption (Formula 5.3).

5.1.4 MyDrivingDroid

The MyDrivingDroid project developed at Instituto de Telecomunicações is aimed at data gathering taking advantage of an Android Smartphone's embedded sensors. In order to gain acceptance by a wide audience in the general public, MDD was required to run on a wide range of smartphones, as mentioned in Chapter 4. Also, the interface was designed to be minimal, intuitive and easy. Enhancing user experience and validating a stable implementation on multiple smartphones constituted a significant time effort. The tested smartphones can be seen on Table 5.2.

In respect to the interface (see Figure 5.8), in the beginning of each trip a user presses the play button to start logging and the stop button at the end. While the application is logging data, it is possible to close it and use the smartphone for other daily purposes, as it runs in the background. As seen in Chapter 2, MDD is also constituted of a server that receives and processes data. When the user chooses to synchronize with the server, the data is sent to it and then deleted from the smartphone.

This section explains how the OBD data gathering module was implemented by explaining first the project's architecture and the problems encountered, as well as the improvements made in user experience. Since the latter was very important to convince people to use the application as referred, there was an interest in perfecting the interaction and automating repetitive functions, which also turned out to be a time consuming effort.

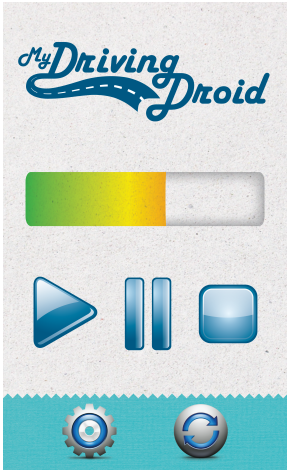


Figure 5.8: MDD Interface

5.1.4.1 Architecture

The project architecture was designed thinking in expansionism. This is explained with the aid of Figure 5.9.

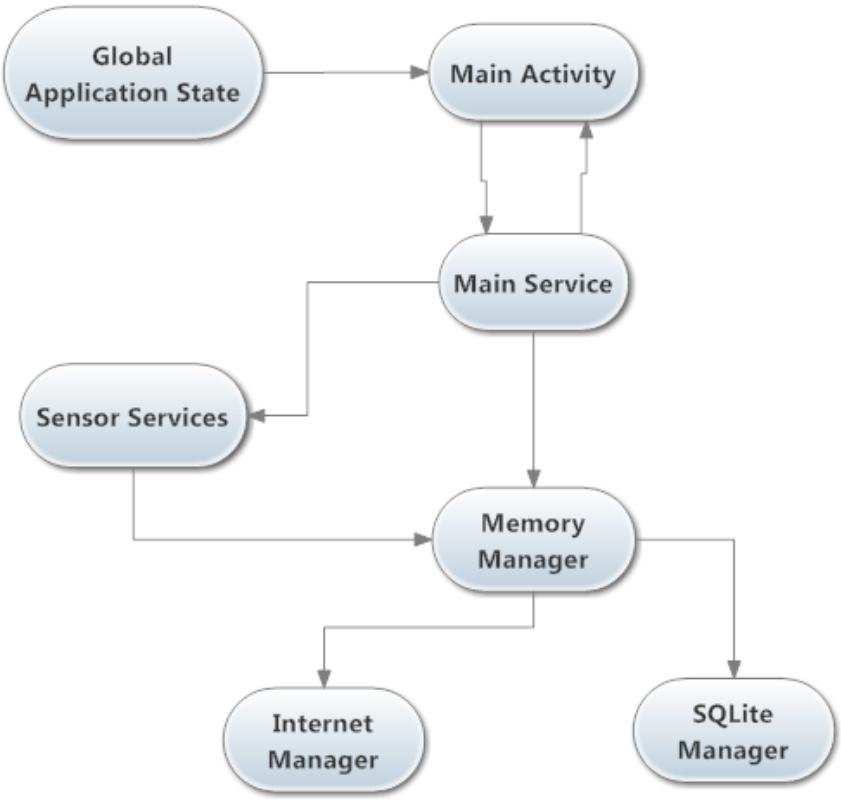


Figure 5.9: MDD Architecture

The application consists of the Main Activity and Main Service for interface and background work respectively, sensor managers for each sensor, a Memory Manager for data handling, an Internet Manager for server communication and a SQLite Manager for SQLite database handling. Each manager has its handler associated with its thread message queue so that all requests are serialized and no data is lost.

When the application runs, the first thing it executes is a base class that extends the Android Application super class whose function is to maintain a global application state. Then the Main Activity is invoked showing the interface and, in turn, creates and binds with the Main Service. The reason for this is because the Main Activity is designed to process only the user interface, leaving the background processing to other services, so it does not interfere with the user interaction. The Main Service is always running, so even when the interface is hidden or destroyed, this service can still do its work in the background and hold onto other services. The Main Activity and the Main Service communicate between them so that they remain synchronized and updated.

The Main Service is responsible for the Memory Manager. This class is initialized as soon as the service is created. It runs on a separate thread with a handler that receives requests, enqueueing and processing them. When sensor data is available, it is sent to the Memory Manager handler queue, and after a while or when the buffer is full, the Memory Manager flushes this data to the SQLite Manager and Internet Manager.

Each sensor has its own service and each runs on a different thread. A sensor service is only responsible for gathering data and sending it to the Memory Manager. So each sensor service can be killed manually at runtime, without harming the application.

SQLite Manager is a different thread that receives data, converts it into a single insert statement, and writes it to the SQLite database.

Internet Manager is also a different thread that receives data and sends it to the server.

The main data objects used are *JSONObjects* and *JSONArrays*. When a sensor reads data it creates a new *JSONObject* and puts the sensor data in it.

So in order to expand MDD with the OBD Module, all that is needed is a new service and a new table in the SQLite Manager and the corresponding *JSON* structure in the Memory Manager.

The new table and *JSON* structure added include the trip id, composed by some digits from the smartphone's IMEI and the timestamp of the trip's starting time, imported from the trip table that holds the various trips performed, an OBD command code with the mode of operation and the parameter id, the corresponding response value and a timestamp from when the response was recorded. The timestamps are obtained from the smartphones clock in the UTC standard.

MyDrivingDroid was tested in various devices as stated previously. The varying behaviour in different smartphones lead to constant adjustments. One of the most significant

addition was the pipeline threads to serialize requests. When multiple threads made requests, there was some loss of data because of either low processing speed or the system was too busy. One other alteration was that in cheaper smartphones with low processing speed and memory, the data had to be divided in chunks when exporting, because of the risk of overflowing memory, taking more time to complete this operation. This *trade-off* between speed and memory usage had to be tested. Another problem was to deal with sensors shutting down, as discussed in Section 5.1.1.2.

5.1.4.2 Bluetooth OBD Module Integration

To gather automotive data, a Bluetooth OBD interface is plugged in a vehicle, and data is exchanged wirelessly over short distances using short wavelength radio transmissions. Using the Android Bluetooth APIs, an Android application can scan for other Bluetooth devices, query the local Bluetooth adapter for paired Bluetooth devices, establish Radio Frequency Communication (RFCOMM) channels and connect to other devices, manage multiple connections and transfer data to or from other devices using the Bluetooth standard.

In the MDD application, Bluetooth is primarily used to log discovered Bluetooth devices and to connect to an OBD interface. Unfortunately, Bluetooth devices have different behaviour among different smartphones due to different chips and this subsection will also address the solution to that issue.

Working with Bluetooth in Android is similar to working with sockets. There is a Bluetooth socket with its input and output streams, and a connection can be made either as a server or a client. However, in Android, besides needing the right permissions on the manifest, a listener as to be registered for Bluetooth actions. When registering, it is possible to indicate the processing thread's handler and the action referring to what it listens (like when there is a new connection or the discovery process is finished), and a listener object which defines what is called for execution when the action happens. The instructions associated with an action are triggered when a broadcast informing that action has happened is received.

The connection step proved to be challenging. To connect with a Bluetooth device its MAC address is needed. The target device can be obtained from user input, device discovery or from querying paired devices. The last two were made automatic in this module's implementation. If there is no OBD device selected, MDD will attempt to find one within its radius and connect to it. If the connection is successful, it saves the device's MAC address. If there is an OBD interface selected or saved when attempting to make a connection but it is not within reachable radius, MDD will attempt to discover a closer device.

Most Bluetooth devices, when scanning, first search for all Bluetooth devices in their radius and find out their 'class' (CoD). Then they use the Service Discovery Protocol (SDP) to check the type of service the devices offer. The Class of Device (CoD) number can be used to identify the Service and Device Class. For example, if it is a computer, or a phone, or an OBD interface. That is how MDD filters the results to only accept devices with the CoD 7936 for OBD interfaces.

According to the Android API, there are two ways to create a valid RFCOMM Bluetooth socket. Either by a secure or insecure outgoing connection to the device using SDP lookup of a given UUID (immutable representation of a 128-bit universally unique identifier), to determine which channel to connect to. However, a secure link is not always possible (mostly on pre Bluetooth 2.1 devices), as some do not have that capability. Moreover, some RFCOMM chips do not use SDP correctly, which resulted in a connection block on some of the tested smartphones. To overcome this handicap, an insecure connection is possible albeit open to man-in-the-middle attacks. Unfortunately, insecure connections are only available as of Android 2.3.3 (API 10), which creates an hindrance for the lower versions and legacy devices. Creating a secure connection is a good practice. So, the alternative is to set the RFCOMM channel manually, instead of using SDP.

Digging in the Android source code, it is possible to find that some classes and methods are hidden from the API. This is not unusual. To create a secure connection using a given UUID, the method *createRfcommSocketToServiceRecord(UUID)* is used. This method uses SDP to obtain a channel from the UUID, then invokes the method *createRfcommSocket(channel)* that starts the secure connection on the given channel. Unfortunately, this last method is hidden. Although, a workaround was found through the *Java Method* [6] class. This class provides access to methods on a class. This enabled the use of the *createRfcommSocket(channel)* hidden method, manually giving a channel from 1 to 30. Relevant portions of the source code can be consulted in the Appendix A.

When the socket is created, it is only known if it is valid when the *connect()* method of the socket is called. After a successful connection, when attempting to transfer data on slower smartphones, an error can occur because the connect action was broadcast but the connection is still not ready. Bluetooth takes some time to establish a connection. Since it is not known how much time it is needed for that action, a Java latch can be used to prevent this error. The *CountDownLatch* [7] is a very useful synchronization aid used throughout MDD (being a multi-threaded application) that allows threads to wait for other threads to complete a set of operations. When the connection thread is ready, the latch is released and the data transfer management thread starts its processing. If the connection blocks, the latch suffers a 'timeout' and is also released, but because there is a connection error,

[6] *http://docs.oracle.com/javase/6/docs/api/java/lang/reflect/Method.html*
[7] *http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/CountDownLatch.html*

there won't be any data to transfer, so the data function signals that a new connection is needed.

Before a connection is established, the Bluetooth device must be paired. Pairing a new device is done once (unless the user 'unpairs' it) and involves user interaction, like the input of a passcode, to confirm the identity of the device. In some smartphones, an error happens when pairing devices programmatically. When attempting to connect, if the device is not paired, a pairing request is automatically executed by the *connect()* method, but sometimes the device is nevertheless unable to connect. However, unpairing and pairing the device again manually in the definitions resolves the issue. This error has already been reported several times in various question-and-answer computer programming websites. After a successful connection, MDD and the OBD interface are ready to exchange data.

When the device already holds a connection with another device, it is not recommended to use Bluetooth to perform OBD readings and device discovery at the same time. Android Developers site states that: "Performing device discovery is a heavy procedure for the Bluetooth adapter and will consume a lot of its resources. Once you have found a device to connect, be certain that you always stop discovery with *cancelDiscovery()* before attempting a connection. Also, if you already hold a connection with a device, then performing discovery can significantly reduce the bandwidth available for the connection, so you should not perform discovery while connected" [8].

However, in some smartphones it seems mandatory to cancel device discovery permanently when an OBD is connected. If this does not happen, the connection can be refused and the application may freeze. So the user is warned about the risk of using the same Bluetooth sensor for separate operations.

When the OBD connection is made, two more threads are created. One is designed to send commands and update the command list. The other is always listening to the socket, since the *read()* method is a blocking call, and sends the bytes to a buffer. Once the ELM327's prompt character ('>' or hex 3E) is read, it means a response was given and so it is parsed and sent to Memory Manager message queue.

As noted before, some vehicles take longer to respond and do not always have the same OBD parameters available. There are commands available to obtain supported PIDs. A supported PIDs request returns 32 bits of data, giving binary information about which of the next 32 PIDs are supported (0 means not supported, 1 means supported). The first PID for various modes is usually a supported PIDs request. So for mode 01, the PID 00 gives availability information from PIDs 01 to 20, PID 20 gives for PIDs 21 to 40, and so on (remember the PIDs are in hexadecimal, which gives a 32 PID interval). These commands, however, give information about the sensors' availability by the time they are called. Some vehicle sensors vary in availability. For example, while the vehicle is in

[8] *http://developer.android.com/guide/topics/connectivity/bluetooth.html*

Open Loop or *Closed Loop*, the *lambdas* availability change. So to determine which PIDs are available in a vehicle, to be able to choose the proper fuel consumption formula, the solution was to request all the possible values and analyse if the response was an error.

There are various types of responses by the OBD interface. When an AT command is sent, if it is successful, the response is either the string 'OK' or 'ELM327' plus the interface version (like 'ELM327 v1.4b'). When a PIDs request is sent, the string 'SEARCHING...' is printed then the response.

For the erroneous responses, they can indicate if there is a connection error ('UNABLE TO CONNECT' message), if the vehicle's engine is stopped ('STOPPED' message), if the sensor is recognized but there is no data available ('NO DATA' message), or if the command or sensor is not recognized. In this last case, each mode as an error message that starts with the hex value 7F, then the requested mode, then the hex 12. So, for example, if the command 0901 for VIN Message Count is not recognized, the response can be hex 7F0912 or it can also be the string 'ERROR'.

In respect to the order of the commands, it is important to first send the AT commands to reset the OBD interface configurations and enable the optimizations referred in Section 5.1.2. Then the supported PIDs requests are sent. Even if this information is not directly used, the response indicates if the engine is on. If it is, then the next step is to request vehicle information regarding fuel type and the VIN (mode 09). After that, it enters a loop requesting the sensor data (mode 01). If a sensor is constantly failing, it is removed from the loop. But because there might be a change in sensor availability like in *Open Loop* and *Closed Loop* as referred, if the fuel system status PID is not available, the loop reboots with all codes in each 5 minute interval.

5.1.4.3 Improved User Experience

With user experience in mind, to facilitate access to the application, a Widget was also created. Widgets are miniature application views that can be placed in the home screen and receive periodic updates. They work in a different manner than services or activities. Since it is intended to facilitate the access of the application, the MDD widget has only three buttons and the MDD icon. The icon opens the main application, and the three control buttons send requests to the Main Service through the service messenger just like the Main Activity.

The Main Service controls the application status because, as it was referred, the Main Activity is just for user interaction, and so is the widget. There is a big difference tough. When the user opens the application normally, the Main Activity binds with the service and, in doing so, the Main Service returns the reference to its messenger object so that they can communicate. The widget cannot bind with a service as it is independent, and

there are no guarantees that the Main Service is running since it is destroyed when the application has no more processes running and exits.

So the solution is to request, in the widget, the service to start on play and exit on stop. Looking at the Service Lifecycle diagram (Figure 5.10) it is possible to devise a solution.

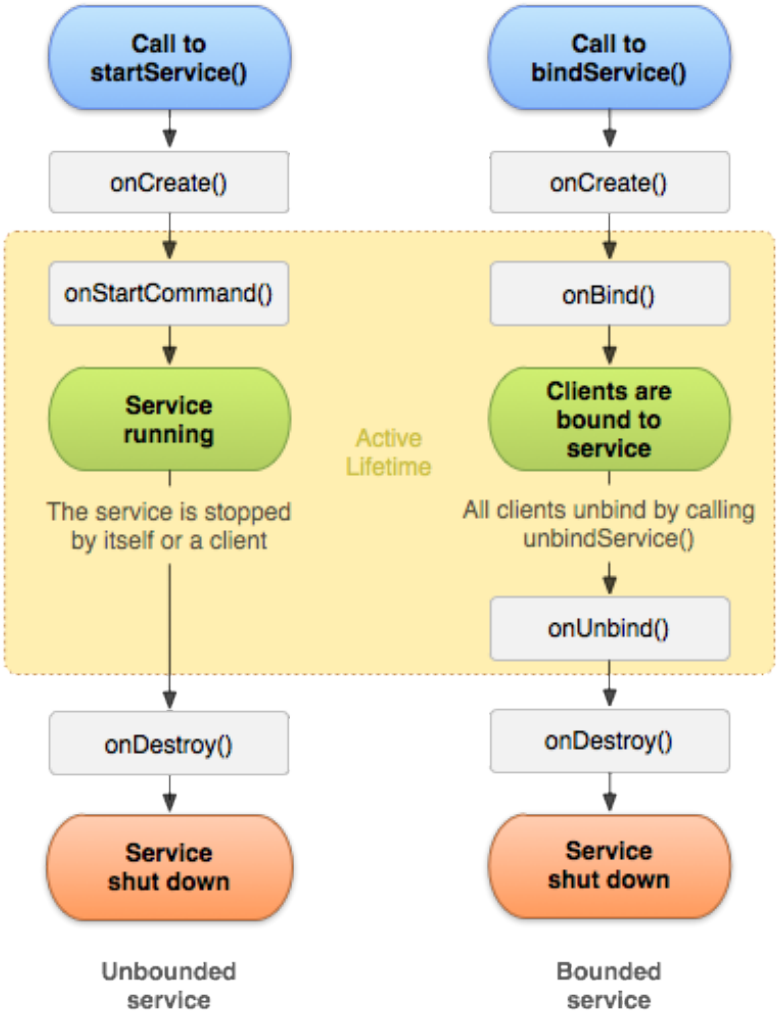


Figure 5.10: Service Lifecycle

The service launches the *onCreate()* method only once even if there is a call to start or bind multiple times, but each time a call is made, the methods inside the Activity Lifetime, like the *onStartCommand()* and the *bindService()* functions, are called accordingly and receive the intent of the caller.

So when a request is made, the Main Service knows through the intent who sent it. If it is the Main Activity it binds with it returning the messenger object so the Main Activity initializes it, if it is the widget it automatically calls play and initializes its messenger by itself.

5.1.5 Overview

The ELM327 documentation [ELM11], the SAE J1979 [SAE] and ISO 15031-5 [ISO] provide everything needed to understand the OBD communication protocols. However, some ELM327 clones do not obey or recognize some of the device's internal commands, and field tests were required.

The Fuel Consumption Algorithm involved a lot of logic reasoning and research into vehicle mechanics, physics and thermodynamic, but proved to be worthwhile as a whole and as a mean to understand the better approach on regression formulas, since understanding the way engines consume fuel is indispensable. In order to validate the formula, some tests were made with users recording the vehicle's on-board computer values and the ones given by the formula. The calibration process is described in section 5.3.4.

The development of the Bluetooth OBD service was filled with mishaps. The varying behaviour of the different ELM327 clones, allied with the varying behaviour of the different vehicles and the varying behaviour of the different smartphones, together with some Android bugs, and in which the only solution was *trial and error*, took an immense amount of time. Also, some smartphones did not even recognize OBD interfaces, as was the case of the Samsung Galaxy Gio GT-S5660, and this unfortunate event caused some users to be unable to contribute with OBD data.

In total, 10 OBD interfaces were tested on 12 different vehicles and the MyDrivingDroid application was tested in more than 15 devices (see Tables 5.2). So, for the application to be usable, it was constantly perfected. The tables below show some of the smartphones and vehicles this project was tested on.

Table 5.2: Tested Smartphones and Vehicles

Smartphone
Samsung Nexus S
Samsung Galaxy 550 GT-I5500
Samsung Galaxy Ace
Samsung Galaxy Gio GT-S5660
Samsung Galaxy S
Samsung Galaxy Tab GT-P7510
LG Maximo Black P970
LG Maximo 2X P990
Optimus Boston
Huawei IDEOS

Vehicle
Peugeot 207
Audi A4
Renault Clio
Fiat Punto
Volkswagen Golf 6
BMW Z3
BMW 320d
BMW 525d
Chevrolet Spark
Clio Storia
Renault Twingo

Even with the time consuming problems encountered, the development was successful due to the numerous extent of devices and vehicles tested in which the application proved to be stable. Aside from the Samsung Galaxy Gio GT-S5660 that cannot pair

Bluetooth OBD devices, in the rest of the smartphones and vehicles presented in Table 5.2 the application is perfectly functional.

5.2 Data Collection Methodology

The data gathering process involved various collaborators with different smartphones and different vehicles. This section will address how the collaborators were gathered and chosen and the reasons behind the vehicle's selection process.

5.2.1 Managing Collaborators

To convince and manage collaborators, a website was firstly created [9]. The purpose of this virtual space is to provide instructions, background and detail the advantages of the OBD technology, provide a manual for the MDD application as well as a download link, and discuss the thesis in overall. There is also a section reserved for collaborators. Anyone can register provided that an email and some vehicle specifications are submitted. The minimum vehicle specifications are engine size and fuel.

Resorting to the dynamic e-mail service of FEUP, an e-mail was sent to the academic community asking for volunteers to contribute with vehicle and sensor data. From the 23 people interested in collaborating, very few had the requirements for this project. After registering, an interview was conducted to known more about the volunteer's trip time and to conduct a vehicle inspection in order to find if it was OBD compatible. There was a challenge in finding the OBD socket in each vehicle. As referred before, an OBDII connector as to be located in the vehicle's cabin within approximately half a metre of the steering wheel. But its exact location varied with each vehicle model. Most of the vehicles of the volunteers were diesel type engines, but unfortunately were pre 2004. In the United States, the OBDII standard exists in all vehicles since 1996, but for the European diesel engines, it was only implemented in 2004. Still, at first, there were not enough OBD devices available for the number of drivers. So each driver gathered data until it reached at least 5 hours of total trip time, then the OBD switched to another driver. As new OBD devices became available, the drivers kept the OBDs.

Each driver signed an informed consent to allow the use of their gathered data for investigation purposes.

5.2.2 Data Selection

Even after the gathering process, some people had very short trips. In these cases the amount of data gathered resulted in vague scatter plots. Figure 5.11 represents the total number of points contributed by each vehicle where fuel consumption calculation

[9] http://www.fce.pt.vu/

was possible, and the vehicle identification is shown on Table 5.3 along with its engine displacement and fuel type.

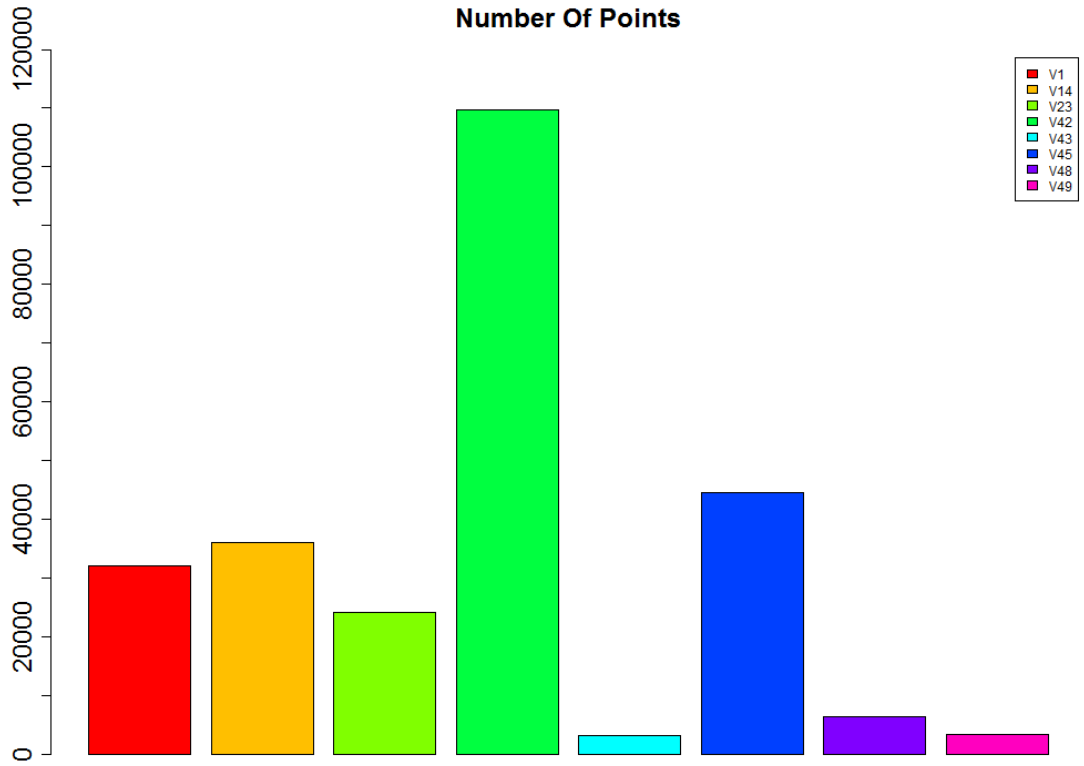


Figure 5.11: Number Of Points For Each Vehicle

Table 5.3: Vehicle Identification

Vehicle Id	**Model**	**Displacement**	**Fuel Type**
V1	Audi A4	1896 cm^3	Diesel
V14	Peugeot 207	1560 cm^3	Diesel
V23	Volkswagen Golf 6	1598 cm^3	Diesel
V42	Renault Clio	1461 cm^3	Diesel
V43	Renault Twingo	1461 cm^3	Diesel
V45	Fiat Punto	1248 cm^3	Diesel
V48	BMW 525d	1995 cm^3	Diesel
V49	BMW 320d	1995 cm^3	Diesel

In data analysis, it is important to recognize that there are a number of inherent limitations on what is possible to learn. One of these limitations is related to the finite amount of data gathered. One of the main practical consequence is the necessity of imposing some working assumptions in the analysis. From the vehicles shown in Table 5.3, only the ones with 30000 or more consumption points proved to have enough data for analysis. So only

the vehicles V1, V14, V23, V42 and V45 were used. Appendix A shows an evolution of the aggregated mean of speed versus fuel consumption (Figures A.1, A.2, A.3, A.4, A.5) showing a decreasing dispersion of points as expected.

5.3 Processing Data

This section discusses the methods used to derive values from the GPS, how those values were used and why an interpolation was applied in the OBD data. Also, it discusses the gathered data, how it was validated and the methods used to analyse and filter it. It also shows that, by means of an exploratory analysis, it was possible to find a structure in the dataset. This chapter also shows a comparative analysis of different results provided by different vehicles, a curious result found between system time and GPS time, and a justification why only system time is used to relate the obtained values.

5.3.1 Deriving Acceleration And Steepness

As referred is Chapter 2, with a GPS receiver it is possible to obtain time, latitude, longitude, altitude and azimuth. Through these values and the Doppler effect, using navigation equations, it is possible to know the location, travelled distance and speed. Using speed and time, acceleration can be derived, and using altitude and distance travelled, steepness can be also derived. The information is gathered at a sample second.

To obtain both acceleration and steepness a least square error method was used. For a given set of data, the least squares method obtains the best-fit curve that produces the minimum sum of the squared deviations. Since velocity obtained from the GPS has a very low error, as proved in plots 5.14 and 5.15, the acceleration in each second was derived from a least square error curve between just the 3 surrounding speeds.

To derive steepness, distance travelled and altitude is needed. Instead of using locations to calculate distance (like the Inverse formula [Vin75] used by Android), speed is used because of its superior precision [KH97] [Cha09]. As the speed is in metres per second and the data is obtained at a sample of one second, the sum of speeds gives distance travelled. While the acceleration curve is time limited, the inclination curve is spatially limited. This means that the number of points used on the inclination least square algorithm varies, contrary to the 3 fixed points in acceleration. From a certain point, to calculate inclination, the number of points used satisfies a minimum radius from the original position and a minimum of 5 points required. This means that the radius gradually increases if the number of obtained points does not meet the required minimum. The Figure 5.12 represents speed versus number of steepness points to help illustrate this point.

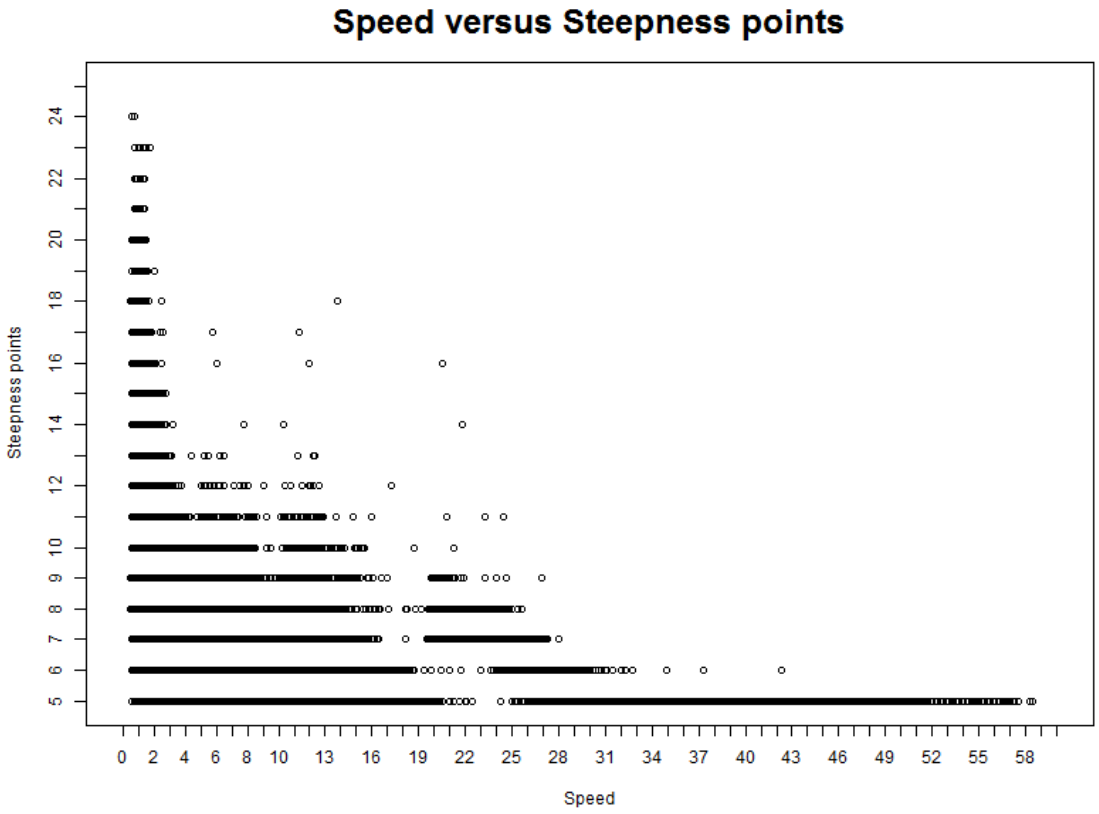


Figure 5.12: Speed versus Steepness points

So at lower speeds the radius is extended to capture at least 5 points. In lower speeds, the radius is less extended than in highways. At higher speeds, the driver is most likely in a highway and the steepness variations are more easily captured with less points.

In the Appendix A is the source code used to derive acceleration and steepness.

5.3.2 OBD Data Interpolation

As referred before, vehicles have different response rates, which means that for a given set of OBD commands it is not guaranteed that all the responses are available in all seconds. Also there is some delay in vehicle response time and OBD response time. For example, a vehicle with an average response rate of 4 codes per second (like vehicle V1), sends a request at millisecond 900. The vehicle takes more than 200 milliseconds to respond which means the smartphone will receive the response on the next second. In order to also prevent seconds in the middle of the data with no fuel consumption values, a simple solution was implemented which consisted in linearly interpolating the OBD response values with the GPS timestamps. Values were only interpolated in empty seconds if they were not more than 2 seconds apart from a valid data point. This means that large gaps

between data, where no information was available, were not used, since interpolating this gaps possibly resulted in bad data accuracy.

5.3.3 System Time And GPS Time

In the state of the art (Chapter 2) it is referred that there are no guarantees about the smartphone's system timestamp reliability as proved in Figure 5.13. If it is either a cheap crystal oscillator or a firmware bug cannot be stated for sure, and it also falls out of the scope of this work. However, some smartphones have the possibility of synchronizing with the operator's clock. Still, in the interval between synchronizations, the clock either suffers a delay or hastens.

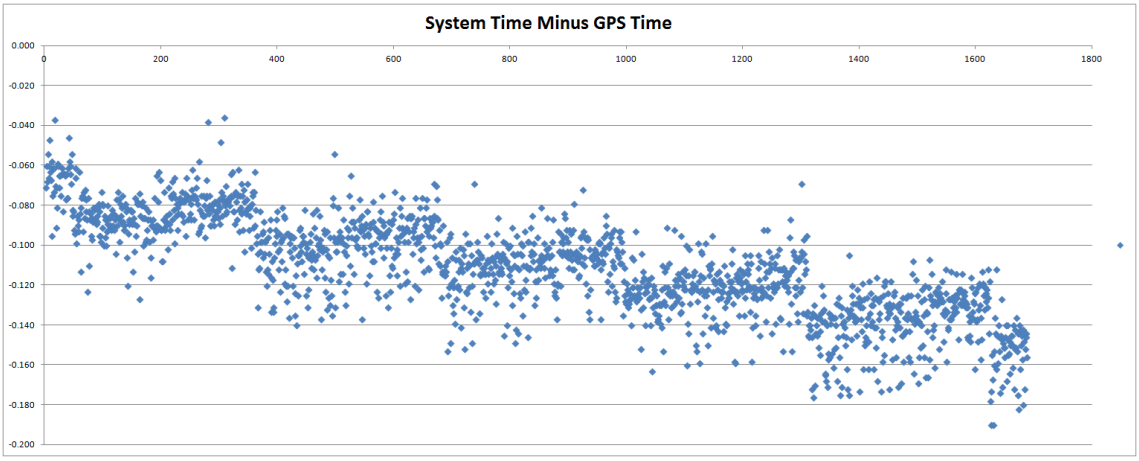


Figure 5.13: Delay suffered in System Time minus GPS Time

Some Android smartphones also use the GPS atomic clocks, which are incredibly accurate, to adjust the time. Although occasionally a leap second adjustment is applied to the Coordinated Universal Time (UTC) in order to keep its time close to the mean solar time, and although the GPS signal includes the offset difference between GPS time and UTC, some Android smartphones do not compensate this offset, and so this leads to some discrepancy on their GPS time. This lead to the workaround with the raw NMEA messages [NME02]. This bug has been debated for some time now in some question-and-answer computer programming or project hosting websites like the Android project page [10]. Although not directly related to this thesis proposition, this was an interesting finding.

So in order to relate the data gathered between various sensors, namely the OBD and GPS data, the UTC timestamp was used to relate between the various sources. But if at some point an accurate time for the data is needed, that data timestamp can be related to the correct GPS given time. Namely in the OBD and GPS synchronization for this work,

[10] $http://code.google.com/p/android/issues/detail?id = 5485$

the OBD data was interpolated onto the GPS timestamp, resulting in a more precisely synchronized dataset.

5.3.4 Calibration

It is important to validate the GPS values and the fuel consumption formula. To validate the GPS values, the GPS speed was plotted against the vehicle speed read by the OBD. The Figure 5.14 with all the gathered GPS versus OBD velocity points and plot 5.15 with the aggregated mean, show the relationship between these variables obeys a linear model. It also shows that the error decreases as velocity increases. The disperse points in plot 5.14 are usually caused by abrupt variations in speed, like quickly braking.

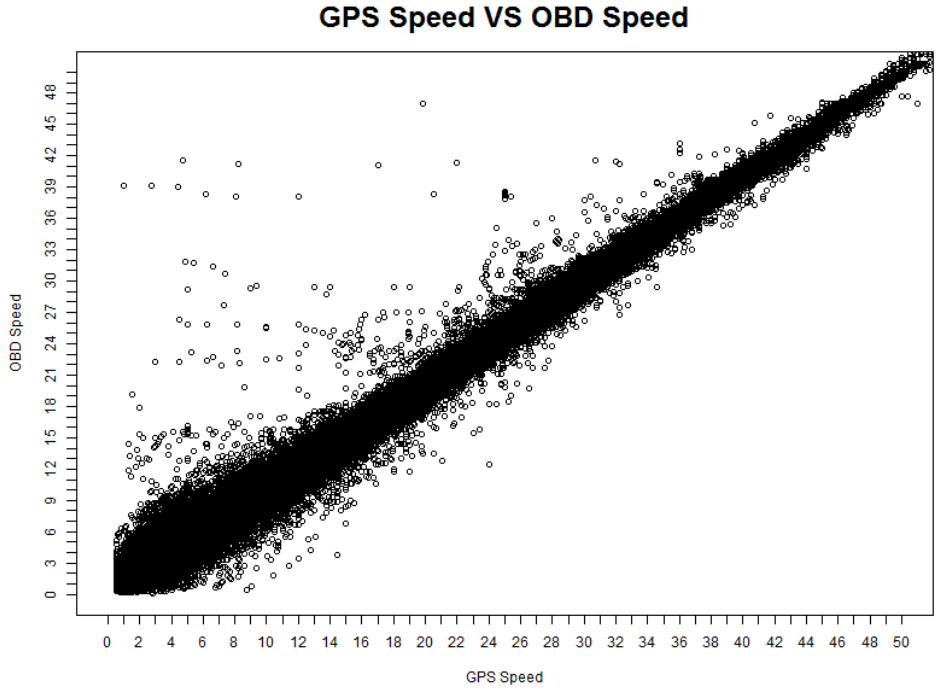


Figure 5.14: GPS Speed versus OBD Speed in metres per second

In order to validate the fuel consumption formula, some vehicles (like V1 and V14) offered a visual interface with fuel consumption estimative values that were used to compare with the calculated values from the formula. In order to do this, an event trigger was also implemented in MyDrivingDroid to record an event description and timestamp. So at the beginning of a trip a user defines an event, like 'Consumption at 6l/100km'. While driving, every time the vehicle's interface showed 6 litres per 100 kilometres the driver quickly tapped the smartphone's screen and an event was recorded. So after recording

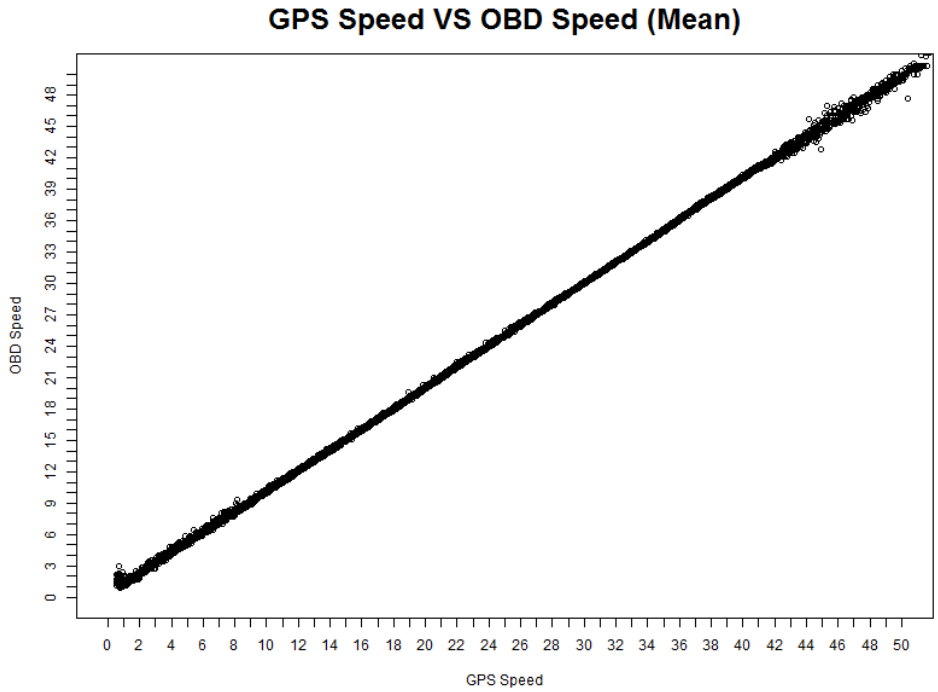


Figure 5.15: GPS Speed versus OBD Speed (Mean) in metres per second

some events, the vehicle's on-board computer values and calculated values from the formula can be compared using the timestamps from the event and the timestamps from the calculated value. Some of the results of the calibration process can be seen on Table 5.4.

Table 5.4: Calculated Versus Shown Values Sample

Calculated	**Shown**
8.9	8.9
9.6	8.9
8.7	8.9
8.6	8.9
8.9	8.9
8.5	8.9
Average	**Std. Dev.**
8.9	0.4

Calculated	**Shown**
7.4	7.5
7.4	7.5
7.3	7.5
7.4	7.5
7.4	7.5
7.7	7.5
Average	**Std. Dev.**
7.4	0.0

Calculated	**Shown**
5.3	6.0
6.3	6.0
5.1	6.0
5.4	6.0
5.9	6.0
5.4	6.0
Average	**Std. Dev.**
5.6	0.5

5.3.5 Data Relationships

The units used for the variables are litres per 100 kilometres for fuel consumption, metres per second for speed, metres per second squared for acceleration and degrees for inclination. In order to comprehend the relationship of fuel consumption with speed, acceleration and inclination, each was individually plotted. The results, considering all

points, are presented below (see Figures 5.16, 5.19, 5.22). In each relation, the aggregated mean was also calculated and is shown below (see Figures 5.17, 5.20, 5.23), along with the corresponding histogram for all points (see Figures 5.18, 5.21, 5.24).

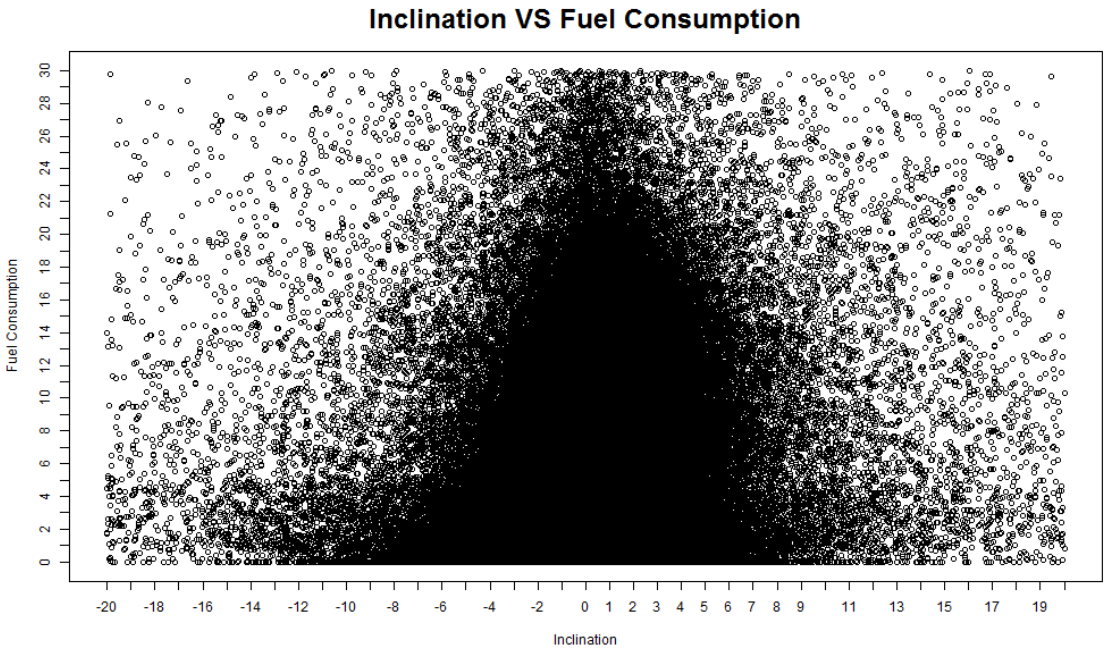


Figure 5.16: Inclination versus Fuel Consumption

The plotted means helped in detecting where dispersion occurs. In the inclination plot there are scattered points (see Figures 5.17 and 5.18) below -3 and above 3 degrees, in the acceleration plot (see Figures 5.20 and 5.21) below -1 and above 1, and in the speed plot (see Figures 5.23 and 5.24) a dispersion is detected above 40 metres per second. The reason behind this, is because most of the roads where data was gathered do not have an inclination above 3 degrees, vehicles commonly do not suffer an acceleration bigger than 1 metre per second squared, and there are very few points above 40 metres per second (>140 Km/h). So in overall there are very few points in these areas and they represent less than 10% of the data. This is shown by the histograms and Table 5.5.

Table 5.5: Percentile of Data Points

Data Points	**Percent**
Inclination above 3°	9.61%
Inclination below -3°	8.81%
Acceleration above 1m/s^2	2.39%
Acceleration below -1m/s^2	3.86%

These undesired points are considered outliers. An outlier is an anomalous data point inconsistent with most of the rest of the data's behaviour. Boxplots and histograms are a

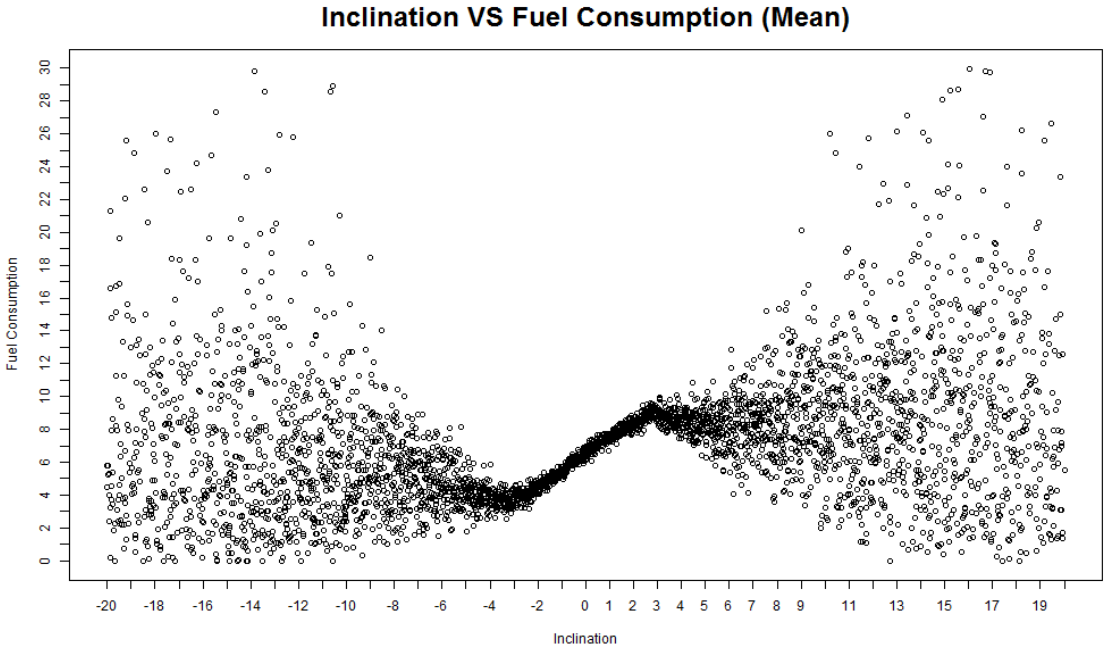


Figure 5.17: Inclination versus Fuel Consumption (Mean)

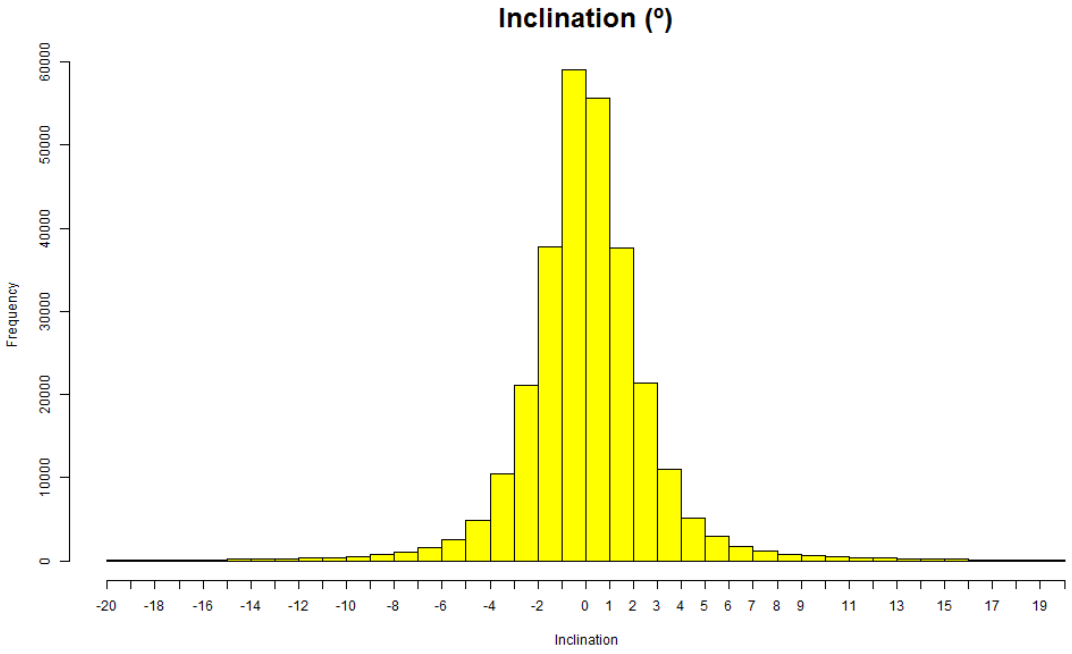


Figure 5.18: Inclination Histogram

helpful technique to detect outliers, or as Ronald Pearson calls them in his book *Exploring Data*, 'distributional monsters that lurk in the data' [Pea11]. He also states that the problem of finding relationships between variables is that, in general, no 'correct answer'

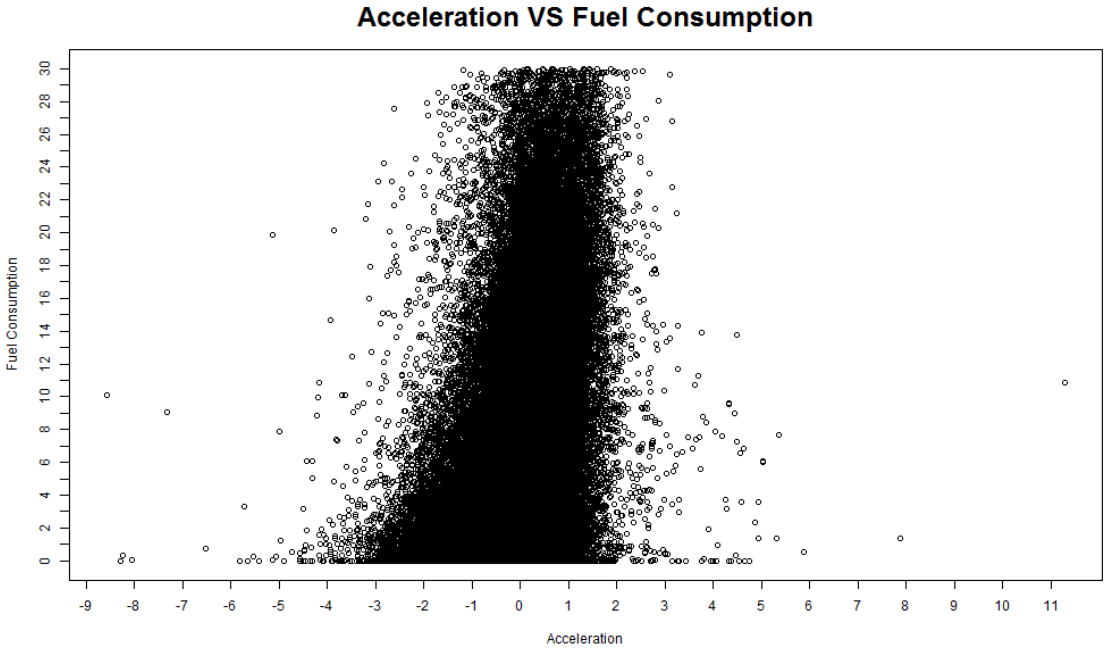


Figure 5.19: Acceleration versus Fuel Consumption

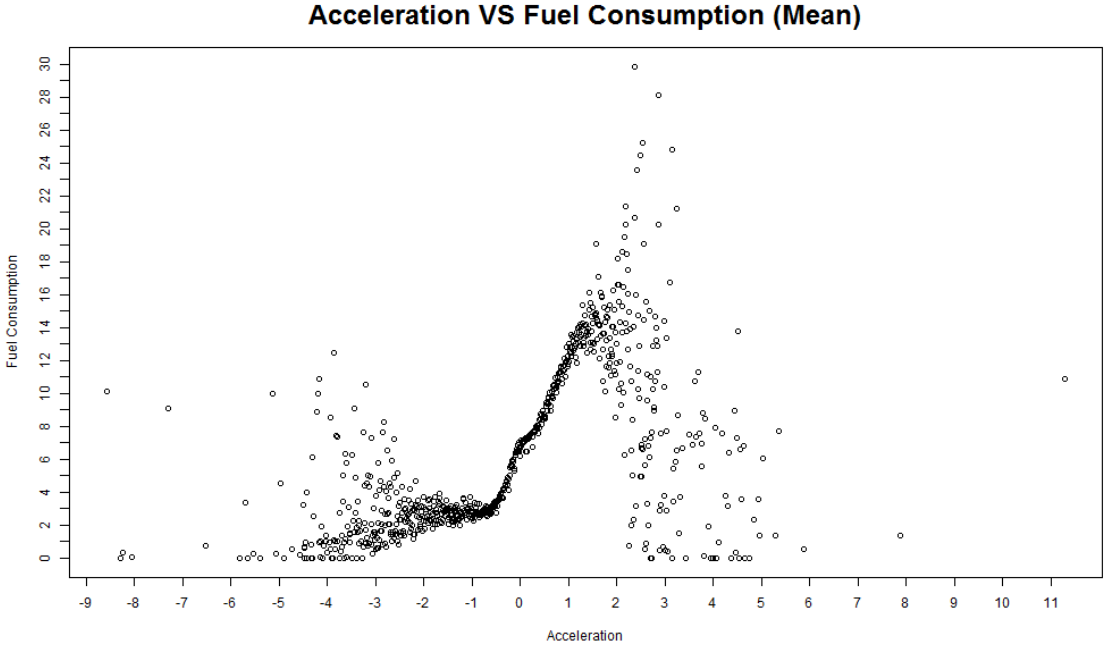


Figure 5.20: Acceleration versus Fuel Consumption (Mean)

exists to the questions we pose, as they commonly are approximations instead of a direct and precise, unambiguous mathematical solutions. Looking at these plots, it is shown that the dependence of fuel consumption on the independent variables speed, acceleration

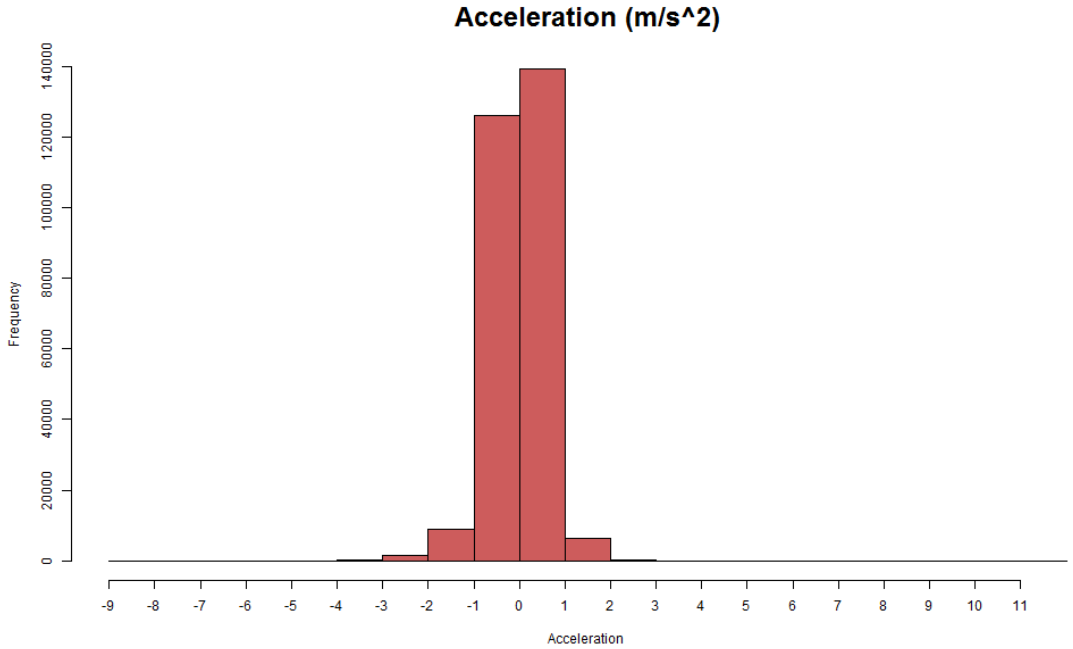


Figure 5.21: Acceleration Histogram

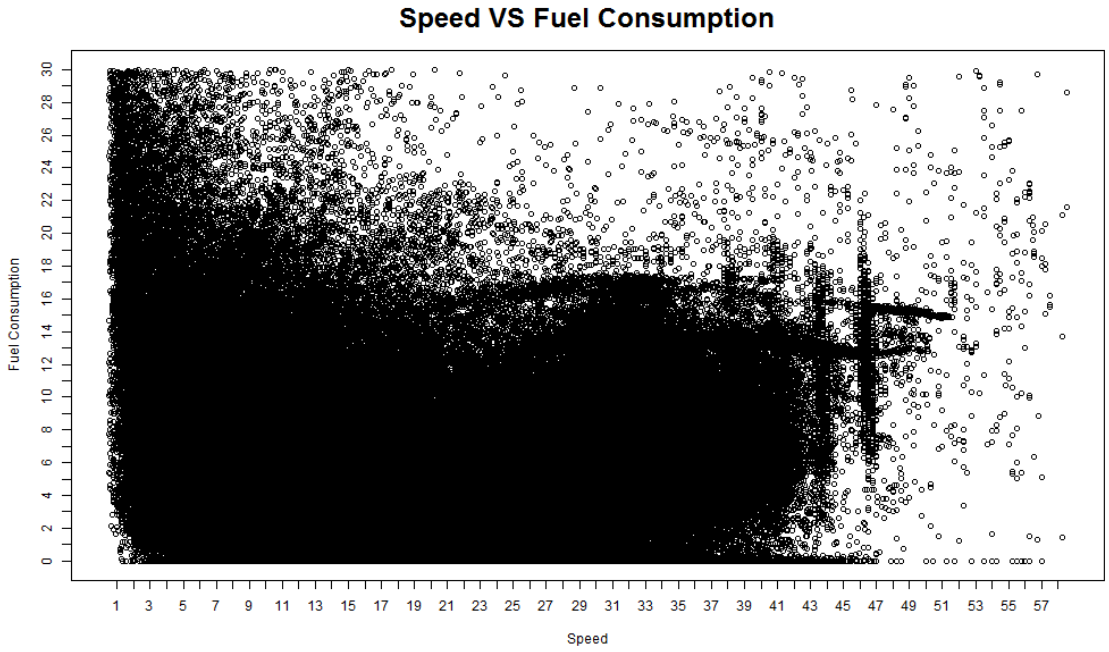


Figure 5.22: Speed versus Fuel Consumption

and inclination is not linear. This is further discussed in Chapter 6. The correlation and covariance between each variable using all gathered vehicle data without these outliers is presented in Table 5.6.

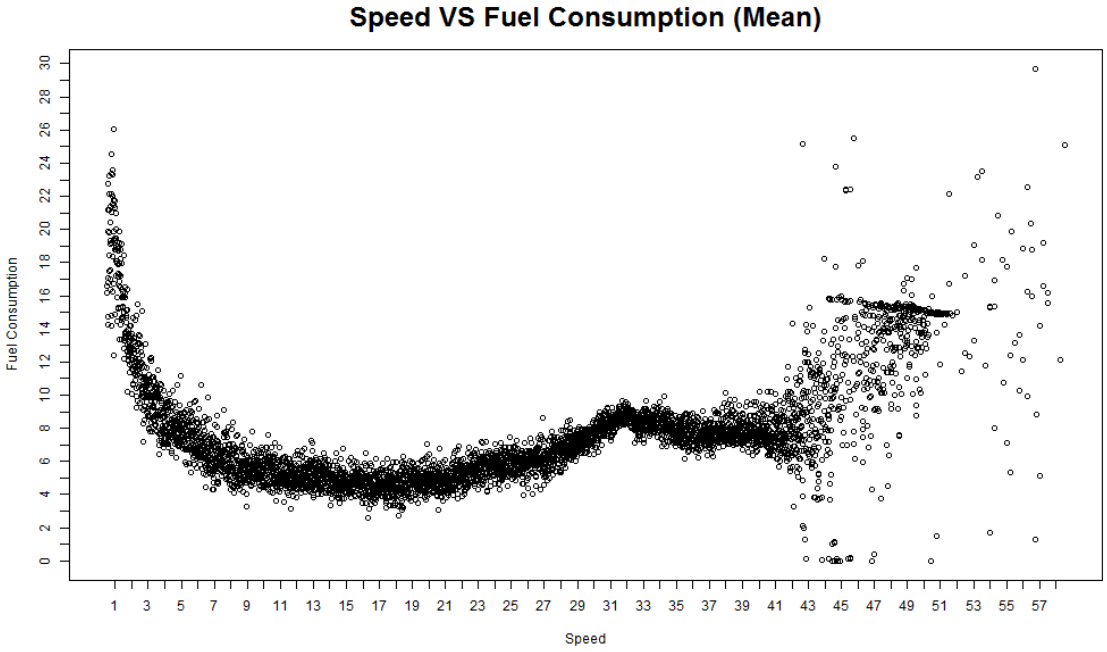


Figure 5.23: Speed versus Fuel Consumption (Mean)

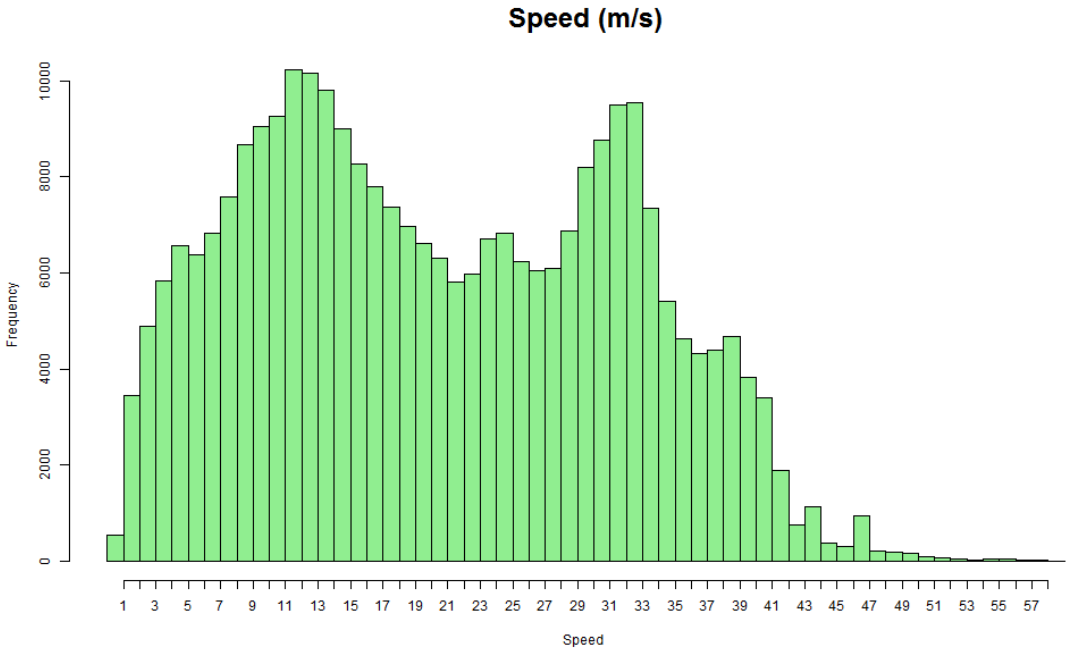


Figure 5.24: Speed Histogram

The covariance is a measure of the relationship between variables. The correlation coefficient serves as a mathematical measure of how much a variable influences another. The correlation is closely related to the covariance, as the last is used to calculate the first.

Table 5.6: Correlation And Covariance between variables

	Correlation	Covariance
Fuel Consumption - Speed	0.166431	7.868484
Fuel Consumption - Acceleration	0.3828733	0.5709074
Fuel Consumption - Inclination	0.3163376	2.026884
Speed - Acceleration	-0.04939249	-0.1674316
Speed - Inclination	-0.005352935	-0.07797148
Acceleration - Inclination	-0.03607315	-0.01657225

A correlation coefficient of 1 and -1 means that the two variables are perfectly correlated or inversely correlated respectively, while a correlation coefficient of zero means that they are not related. Being that the correlation between all the variables is low, it is possible to say they satisfy a mathematical condition of probabilistic independence. However, looking at the mean plots, the mean acceleration and inclination without outliers (Figures 5.25 and 5.26) become almost linearly correlated with fuel consumption, as shown in Table 5.7. So the removed outliers had a large residual, which had a great impact on correlation.

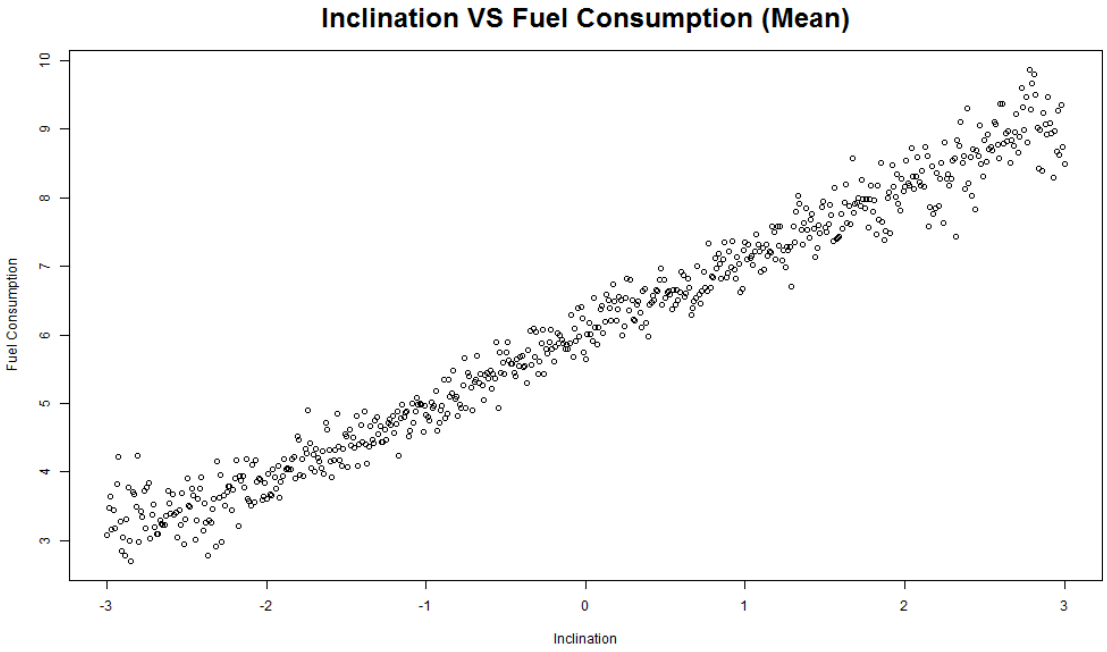


Figure 5.25: Inclination versus Fuel Consumption (Mean) without outliers

Another relation observed with fuel consumption and acceleration or inclination is that if the last two are positive, the fuel consumption plot is basically the same as if the two are negative, but with higher values. Figure 5.27 illustrates this point. The red dots is fuel consumption with only positive inclination, the black for using only negative.

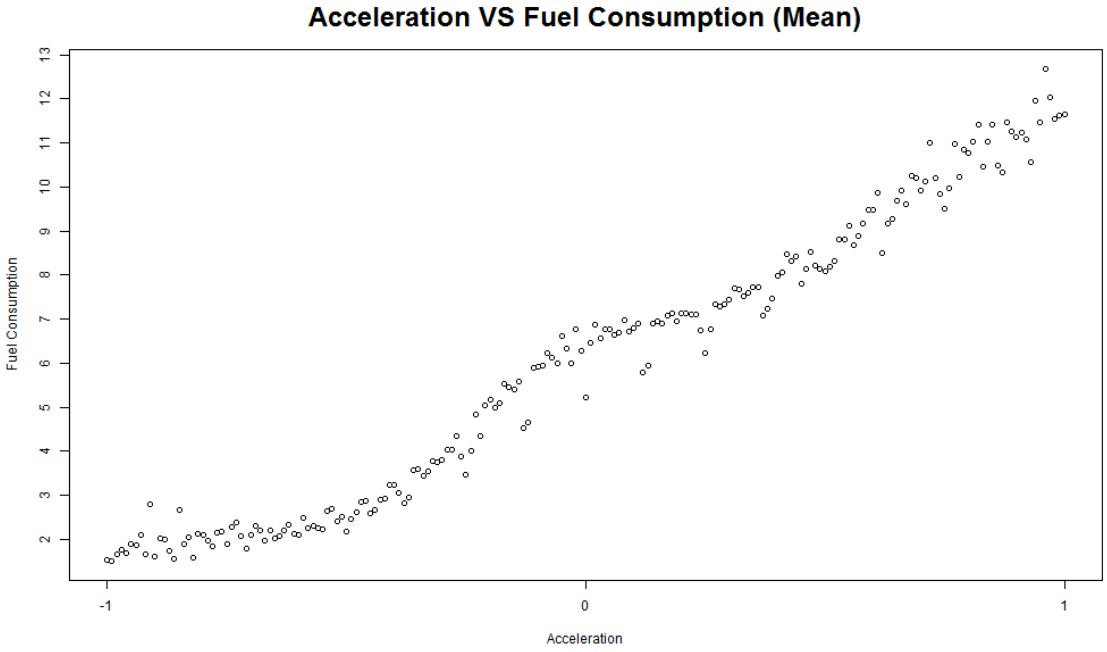


Figure 5.26: Acceleration versus Fuel Consumption (Mean) without outliers

Table 5.7: Correlation And Covariance between variables using aggregated mean and without outliers

	Correlation	**Covariance**
Fuel Consumption - Speed	-0.08209885	-2.662947
Fuel Consumption - Acceleration	0.9820617	1.784946
Fuel Consumption - Inclination	0.9864629	3.062958
Speed - Acceleration	-0.1170565	-0.27888
Speed - Inclination	-0.04122106	-0.1047755
Acceleration - Inclination	-0.5677758	-0.02401655

Note that positive inclination means the vehicle is going upwards. The blue dots is fuel consumption using only positive acceleration and the green for negative.

Comparing the mean consumption between different vehicles with different engine displacements, it is possible to observe that they closely follow the same pattern, as seen in Figures 5.28, 5.29 and 5.30. The vehicle with higher displacement (vehicle V1 corresponding to the black line), as seen in Table 5.3, deviates more from the pattern, and has higher consumption at lower speeds and less consumption at higher speeds in relation to the others as is normally expected from a high engine sized vehicle. Also, it is shown an optimal velocity for a minimum of fuel consumption.

Figure 5.31 enables another conclusion.

When the vehicle is in a descent (inclination is negative) it is normal that acceleration is positive, and when ascending (inclination is positive) it is normal that acceleration

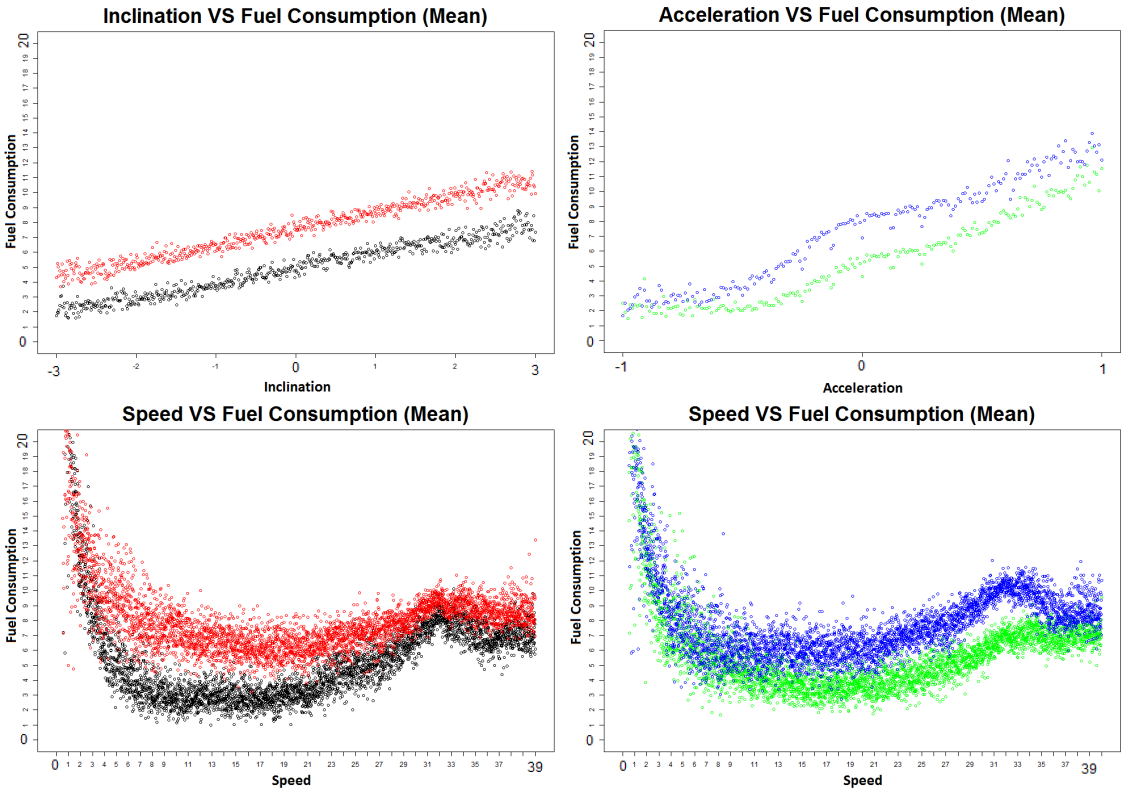


Figure 5.27: Inclination versus Fuel Consumption (Mean) for positive (Red) and negative (Black) values and Acceleration versus Fuel Consumption (Mean) for positive (Blue) and negative (Green) values

lowers, because of the effect of gravity.

5.3.6 Data Concentration

Another finding in the speed versus fuel consumption plots was the concentration of points in certain areas at certain speed intervals. Figure 5.32 illustrate this point.

This most likely happens because the driver is *'shifting gears'*. In motor vehicles the transmission adapts the output of the internal combustion engine to the wheels, so there is a speed-torque ratio in each engaged shift. For a five-speed gearbox, as is the case of the analysed vehicle, each ellipsis indicates the shift the driver engaged. So there is a rise in consumption each time a shift is engaged, then when changed, consumption lowers, then the next shift is engaged and consumption rises again. It is important to note that as the engaged shift increases, the consumption slope on the ellipsis area decreases as a consequence of the speed-torque ratio change.

Also another conclusion was found in the fuel consumption histogram of all vehicles (Figure 5.33). Most of the consumption points are below one litre per 100 kilometres of consumption. This is expected in modern diesel engines at idle speed. When the engine

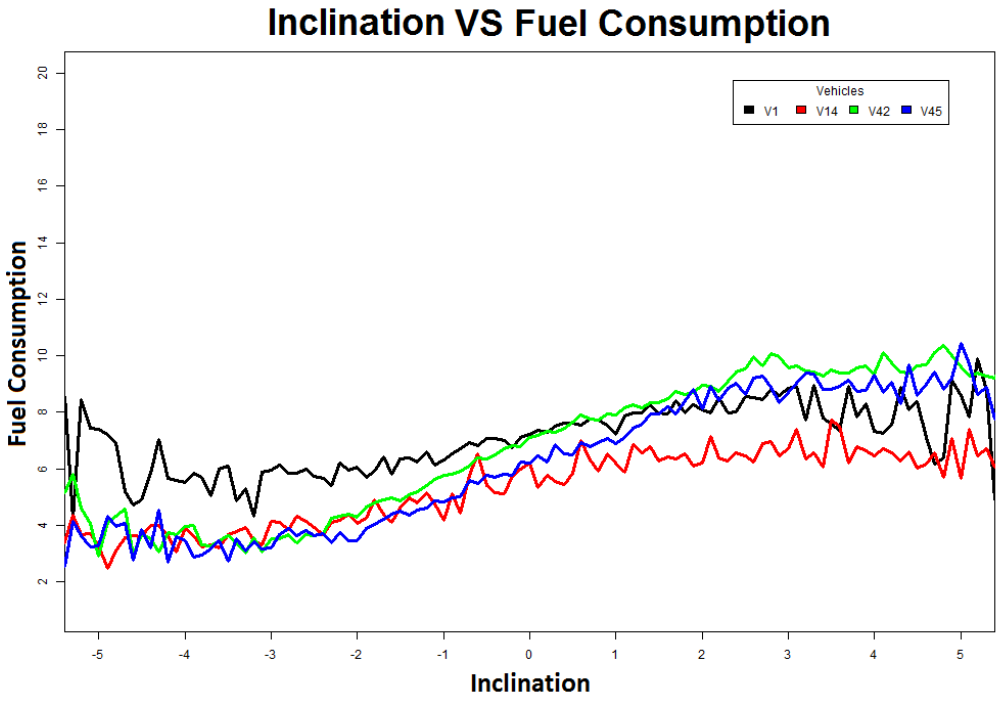


Figure 5.28: Inclination versus Fuel Consumption (Mean) for different vehicles

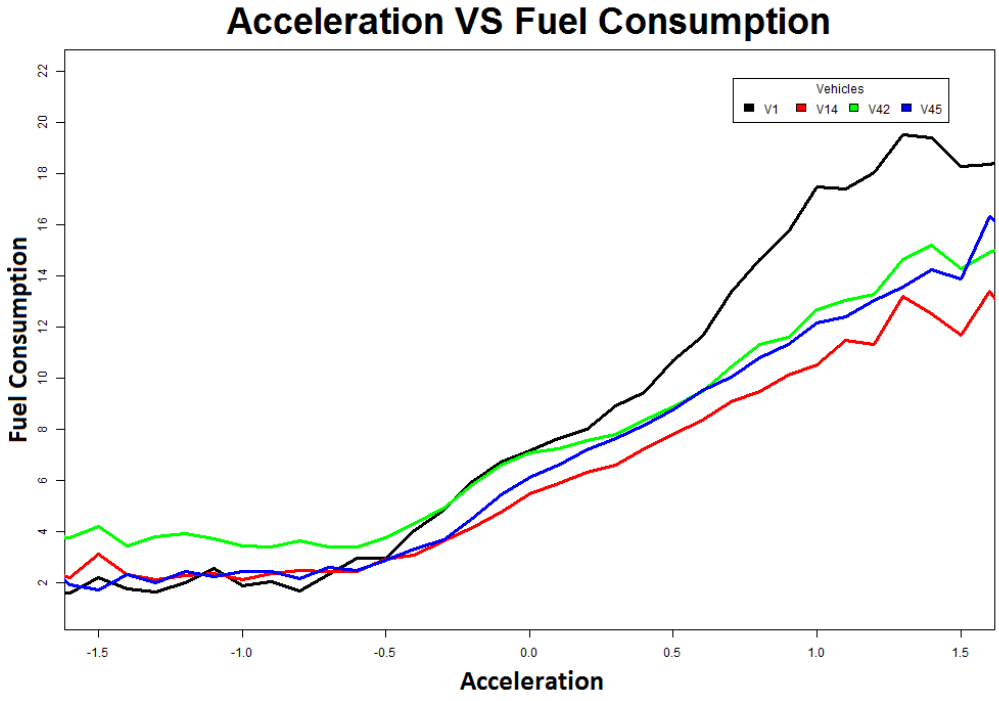


Figure 5.29: Acceleration versus Fuel Consumption (Mean) for different vehicles

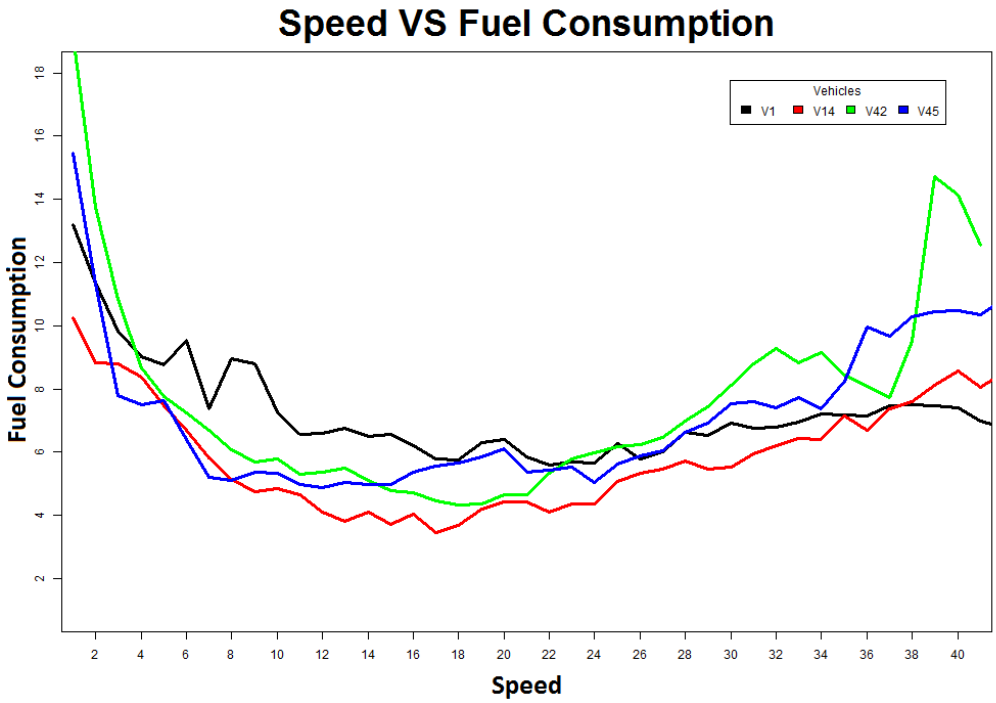


Figure 5.30: Speed versus Fuel Consumption (Mean) for different vehicles

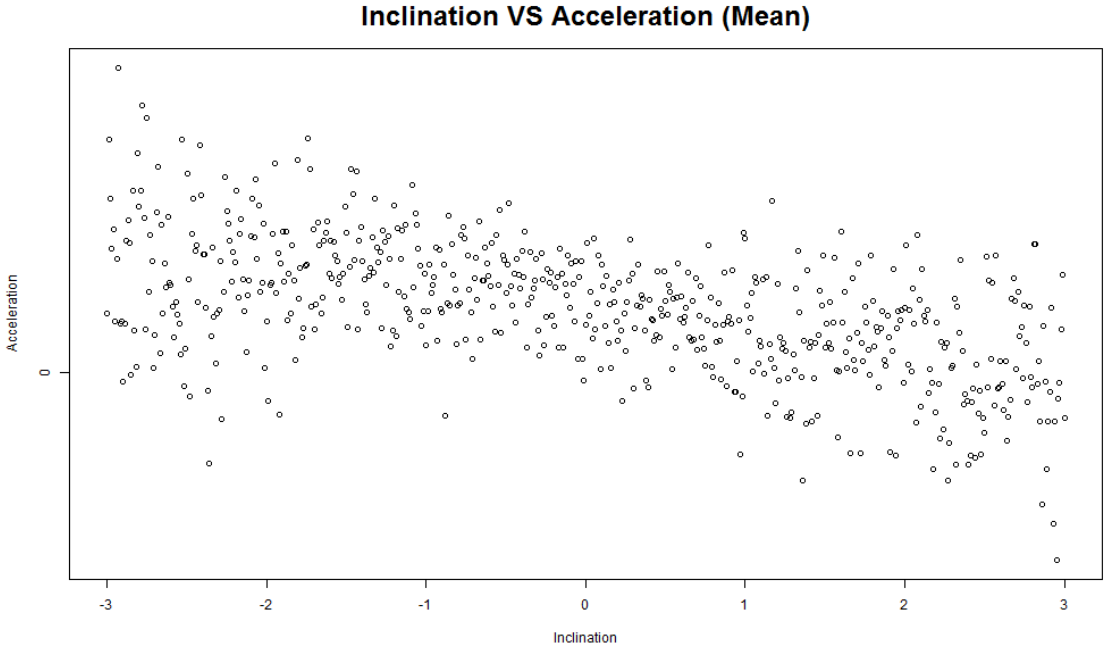


Figure 5.31: Inclination versus Acceleration (Mean)

is uncoupled to the powertrain (no shift is engaged) or when the throttle pedal is not pressed, the vehicle is at idle speed. In this scenario, the engine is capable of generating

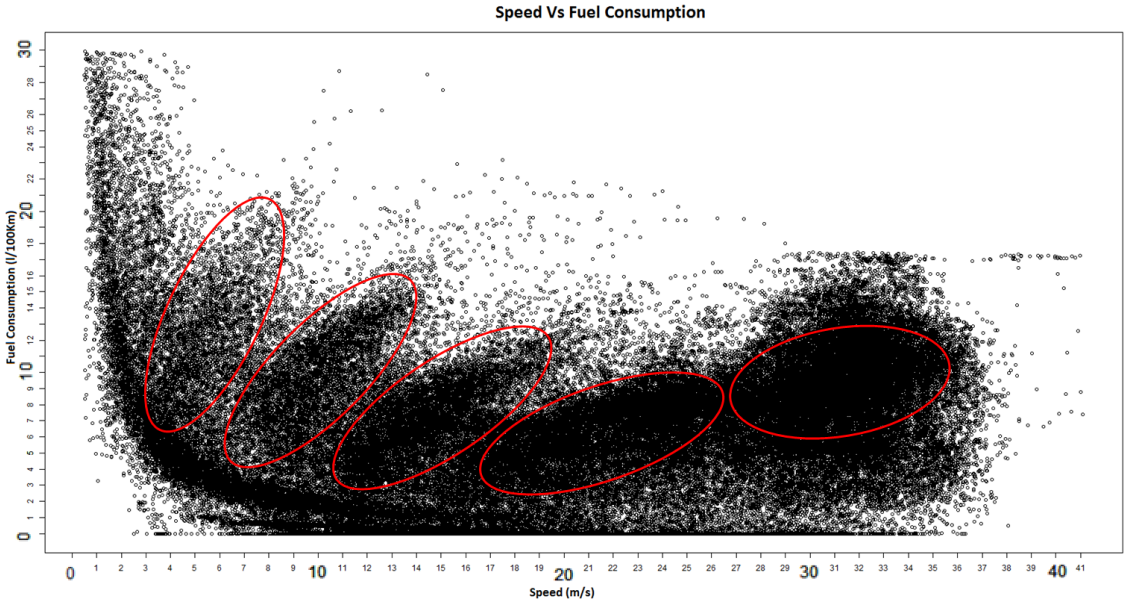


Figure 5.32: Speed versus Fuel Consumption from Vehicle V42

enough power to run, but not enough to increase rotation speed. So when acceleration is below zero in diesel engines, if the vehicle is not braking, it is probably idling and the consumption effectively approximates zero.

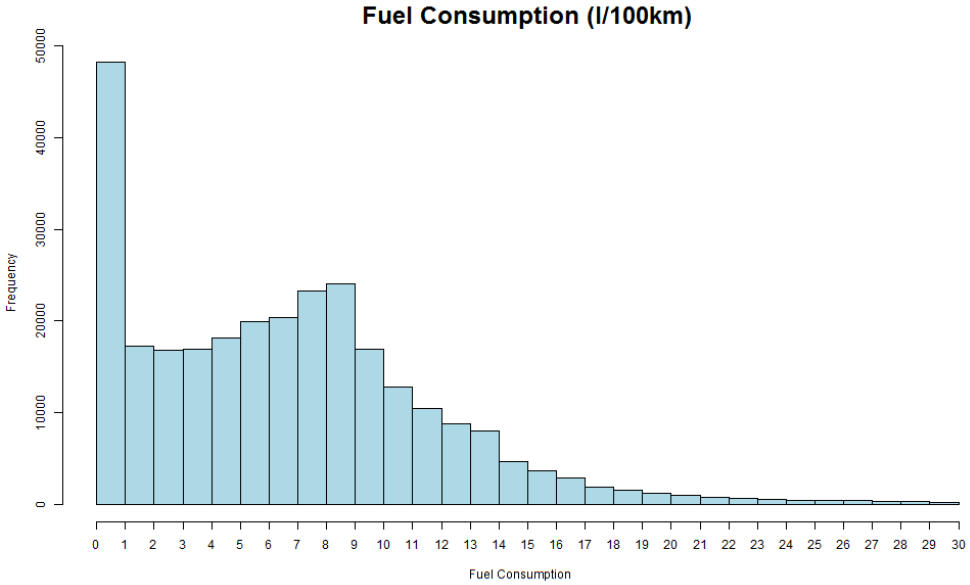


Figure 5.33: Fuel Consumption Histogram

Most of the data from vehicle V42 was gathered in a highway scenario as speed points concentration is at higher velocities as seen in Figure 5.34. It also shows a large con-

centration of fuel consumption points below 1 litre per 100 kilometres supporting the previous statement. The large concentration of red points in highway velocities are also an indicator of the average consumption at those speeds.

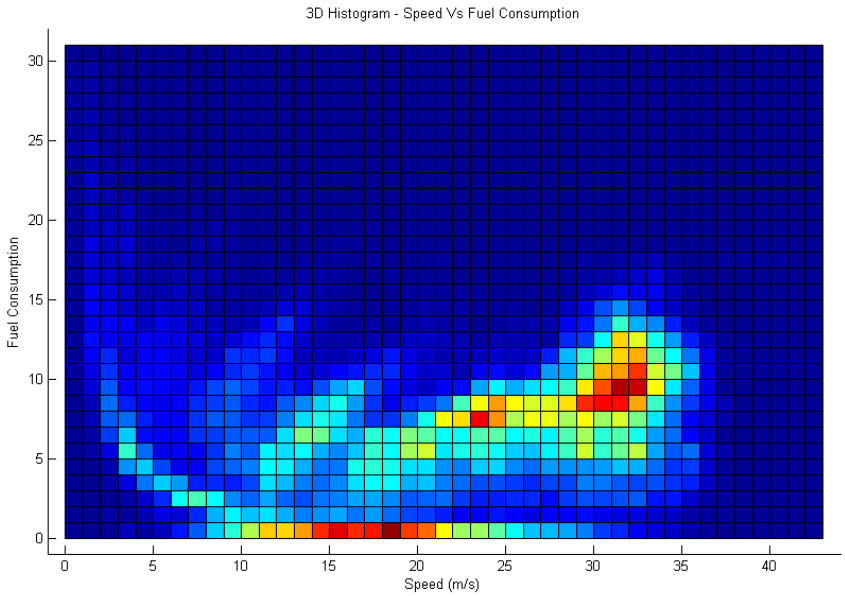


Figure 5.34: 3D Histogram

5.4 Conclusions

In conclusion, while the data gathering stage had some setbacks related to applications bugs referred in Section 5.1.4, and there was a low availability of vehicles that met the requirements, a large amount of data was still gathered from a total of 8 vehicles with a total of 347769 consumption points.

With this gathered data available, then the previously discussed formulas were used to derive acceleration and inclination and interpolate the OBD data. Some relevant code snippets about these methods can be found in the Appendix A. The data was then validated as it is explained in Section 5.3.4.

The outliers proved to have a high influence on correlation. In the case of the aggregated mean acceleration and inclination versus fuel consumption, by removing the outliers, they proved to have a high correlation. While the acceleration and inclination versus fuel consumption can be linearly fitted, in the case of speed versus fuel consumption it was found that it shares a non-linear relationship and can be modelled as a polynomial function as seen in the next chapter (Chapter 6). It is also hinted in this chapter about the effect the gearbox can have on fuel consumption.

Chapter 6

Results

This chapter describes the techniques and methods used to fit models to the gathered dataset, and provide a comparative analysis of said models. It also discusses in the first section why regression was chosen instead of other available methods.

6.1 Algorithms for Fuel Consumption Estimation

There are various approaches to construct the proposed solution. What was needed was to discover the relationship between OBD and GPS data, so that the OBD device could be discarded.

The first approach thought was a Machine Learning Algorithm, so this topic was researched. However, due to its simplicity relatively to Machine Learning algorithms, after analysing the dataset, the conclusion was that methods like regression models would also be worthwhile to explore. Some Machine Learning algorithms use regression techniques, so this part of the research was helpful in taking the first steps into regression analysis.

Machine Learning does not necessarily mean a complex algorithm. Where there are large data sets, learning algorithms can be a good approach as they can more easily understand and deal with those sets.

The two main types of learning algorithms are supervised learning and unsupervised learning. Since it was intended that the proposed algorithm uses the OBD dataset as a learning example, supervised learning was the first though solution. One of the approaches on supervised learning is regression. With regression it is possible to estimate the relationship among variables.

To better understand how regression works, it is shown below an example applied to this work's problem. As it was detailed previously, a fuel consumption model can have a lot of inputs, like vehicle parameters, velocity, acceleration and inclination. In this

example, the graph represented in Figure 6.1 plots only the vehicle speed obtained from a GPS against the average fuel-consumption obtained from the OBD, obtained from a diesel vehicle in a trip. A plot showing the same behaviour is presented in the paper *A Mobile Sensing Architecture for Massive Urban Scanning* [RAV+11].

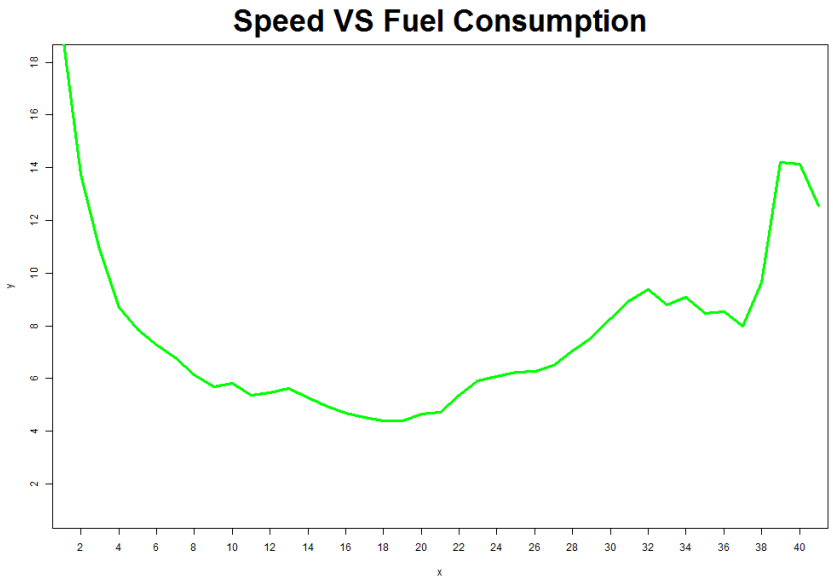


Figure 6.1: Fuel Efficiency

Now lets say through the use of only an Android and its sensors, new velocity values are given. This is where regression appears. If a straight line is put through the data, it is possible to locate where the new values fit in that line as in Figure 6.2.

However, this might not be the best option. So instead of sending a straight line to the data, a polynomial function might be a better fit as it is possible to observe in Figure 6.3.

The regression method is used to predict a continuous value output. With regression it is possible to use very complex polynomial functions, as a high enough degree polynomial can fit any dataset. However, using high degree polynomial functions does not mean better results. Where simple linear functions may cause 'underfitting', high degree polynomial functions may cause 'overfitting'. With 'overfitting', the learned hypothesis may fit the training set very well, but will fail to generalize to new examples.

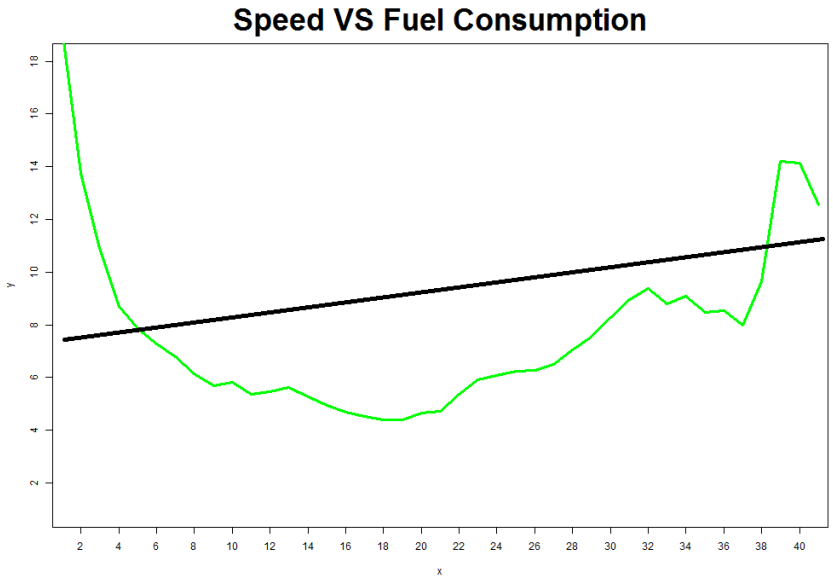


Figure 6.2: Fuel Efficiency with Simple Linear Regression

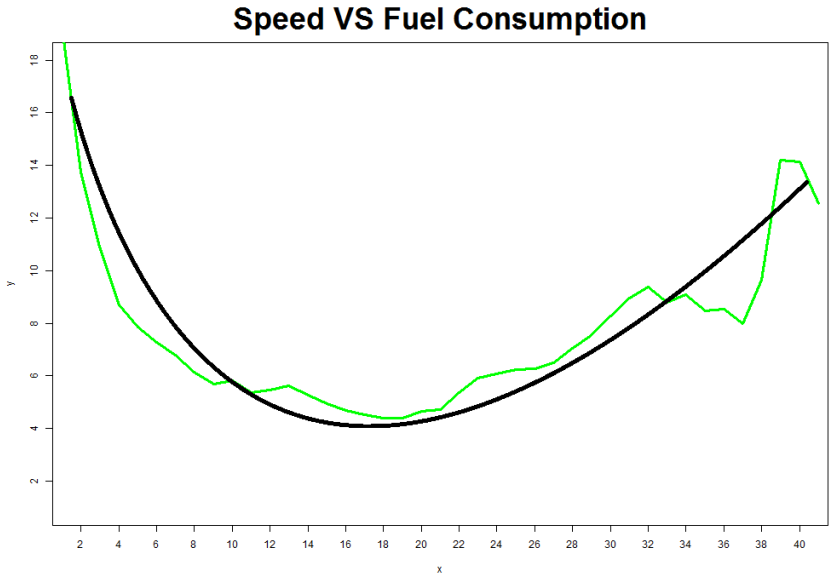


Figure 6.3: Fuel Efficiency with Polynomial Regression

6.2 Regression

Exploratory data analysis has to be dealt with carefully at the risk of leading to the wrong conclusions, as with the killer potato problem based on historical data [CGH+06] described by Ronald Pearson [Pea11]. For a careful analysis Velleman and Hoaglin offer a guide constituted of four fundamental steps known as *'the four **R**'s of exploratory analysis'*: **R**evelation, **R**esiduals, **R**eexpression and **R**esistance [VHM91]. The first **R**

was approached in the previous chapter (Chapter 5) as it consists of seeing what the data reveals and identifying outliers. **R**esiduals are the difference between the observed values and the predictions obtained from a model. They present an useful aid in identifying a model's fitness. The third **R** suggests another approach in revealing a structure in a dataset by transforming it (with a logarithmic function for example). And the last **R**, **R**esistance, refers to robustness and the impact that changes in the dataset can have on a model.

Since the goal is to obtain fuel consumption from GPS values, in the dataset obtained, fuel consumption serves as a dependent or response variable, and GPS velocity, acceleration and inclination as independent or stimulus variables.

A linear model is the simplest of models and widely used as it yields simple functional forms. Many important mathematical functions are analytic, which favour linear constitutive relations [Pea11]. There are various methods of fitting a line in a dataset. The most common assumption is an *errors-in-model* formulation (EM) in which the ordinary least squares (OLS) method is based.

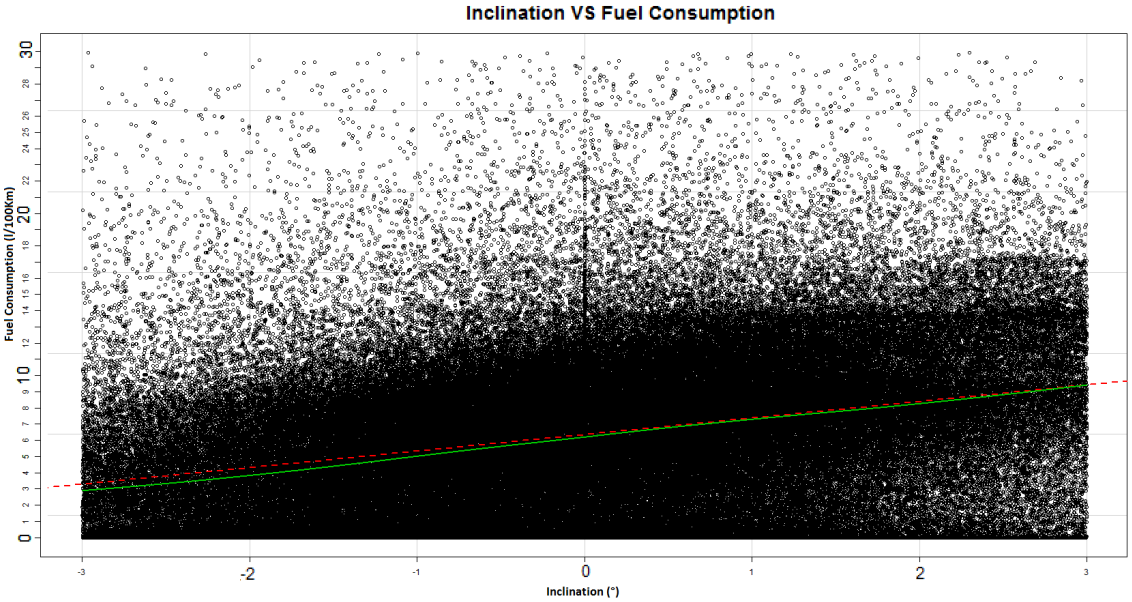
From the scatter plots shown in Chapter 5, it is hinted that not all relationships are linear. Although due to their inherent simplicity, linear models were fitted, along with their partial residual plots, that provide a good indication of the nature of the relationship between each variable. So instead of fitting the whole model, the partial regression plots are firstly generated.

Because the independent variables are statistically independent, in a linear model, the following formulation for this work's case can be viewed as the following:

$$FuelConsumption \approx a * Speed + b * Acceleration + c * Inclination + d \tag{6.1}$$

Figures 6.4, 6.6 and 6.8 represent the component residual plots, which is the partial residual plot plus the fitted line calculated with a linear model. These plots are helpful to evaluate non-linearity. The fitted lines were calculated using the Ordinary Least Squares method. The residuals and the coefficients of each model are also present in these figures, along with the histogram of the studentized residuals to test non-normality. The normality tests are used to determine whether a dataset follows a normal distribution. Considering that the inclination and acceleration versus fuel consumption residual histograms are close to a normal distribution, the predicted to the observed residual error is predominantly low.

So linearity is more notorious in the inclination and acceleration versus fuel consumption plots compared with the speed versus fuel consumption. The formulated hypothesis with all the three independent variables also follows a normal distribution of studentized residuals (see Figure 6.10). Still, it is perceptible, with the provided results, that a linear




```
Residuals:
    Min      1Q  Median      3Q     Max
-9.461 -3.262  -0.038   2.612  26.655

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 6.384223   0.009306     686   <2e-16 ***
inc         1.026751   0.006623     155   <2e-16 ***
Residual standard error: 4.326 on 216120 degrees of freedom
Multiple R-squared: 0.1001, Adjusted R-squared: 0.1001
F-statistic: 2.403e+04 on 1 and 216120 DF,  p-value: < 2.2e-16
```

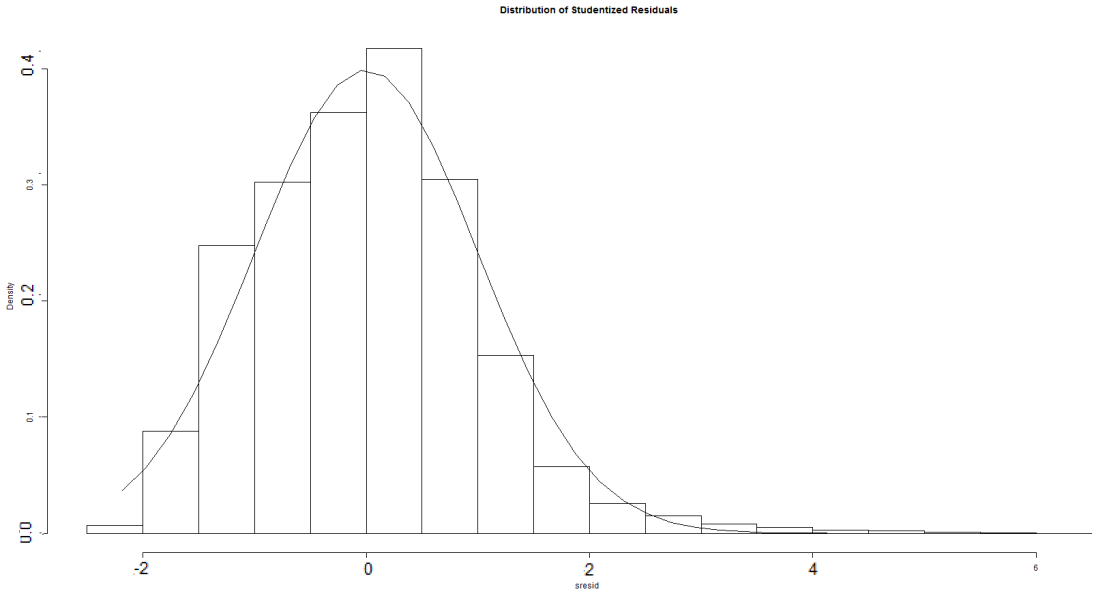
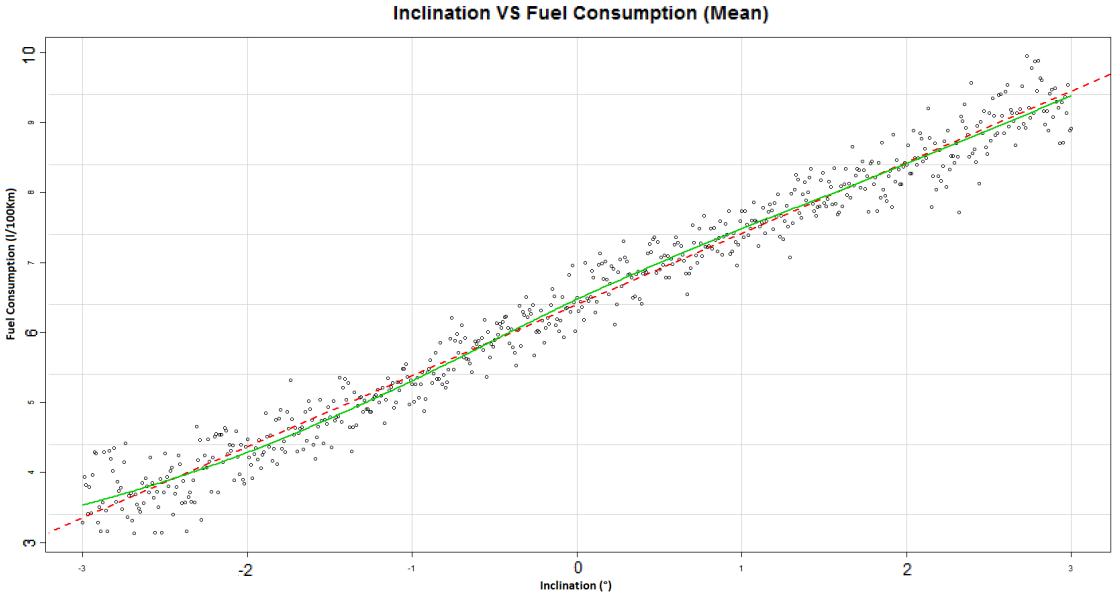


Figure 6.4: Component+Residual Plots, Linear Model Summary And Studentized Residuals for Inclination Versus Fuel Consumption




```
Residuals:
     Min       1Q   Median       3Q      Max
-1.04423 -0.18127 -0.00618  0.18728  0.86230

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.37215    0.01197   532.3   <2e-16 ***
inc          1.01590    0.00690   147.2   <2e-16 ***
Residual standard error: 0.2935 on 599 degrees of freedom
Multiple R-squared: 0.9731, Adjusted R-squared: 0.9731
F-statistic: 2.168e+04 on 1 and 599 DF,  p-value: < 2.2e-16
```

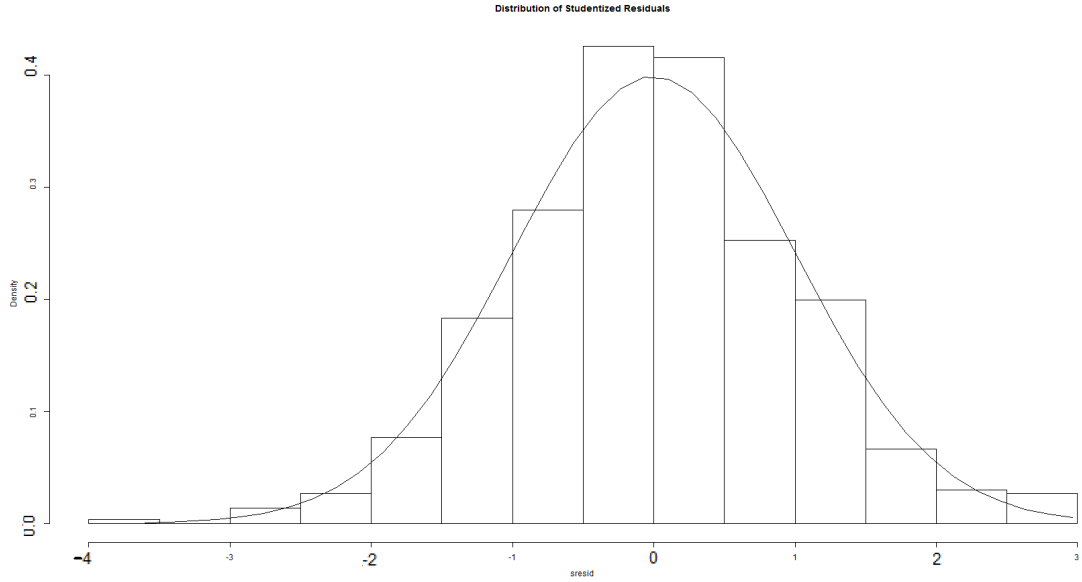
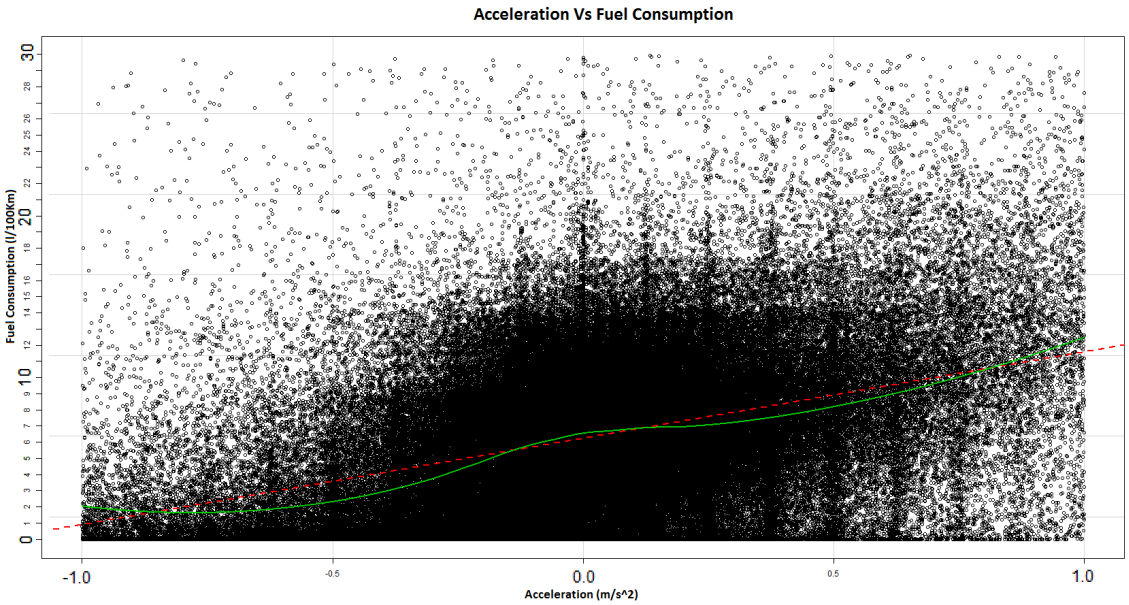


Figure 6.5: Component+Residual Plots, Linear Model Summary And Studentized Residuals for Inclination Versus Fuel Consumption (Mean)

model might not be the best fit, mostly because of the failure of fitting a straight line with speed as an independent variable.




```
Residuals:
     Min       1Q   Median       3Q      Max
-11.5753  -3.0783  -0.1754   2.5161  27.6905

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 6.272856   0.009082   690.7   <2e-16 ***
accel       5.340239   0.027715   192.7   <2e-16 ***
Residual standard error: 4.213 on 216120 degrees of freedom
Multiple R-squared: 0.1466, Adjusted R-squared: 0.1466
F-statistic: 3.713e+04 on 1 and 216120 DF,  p-value: < 2.2e-16
```

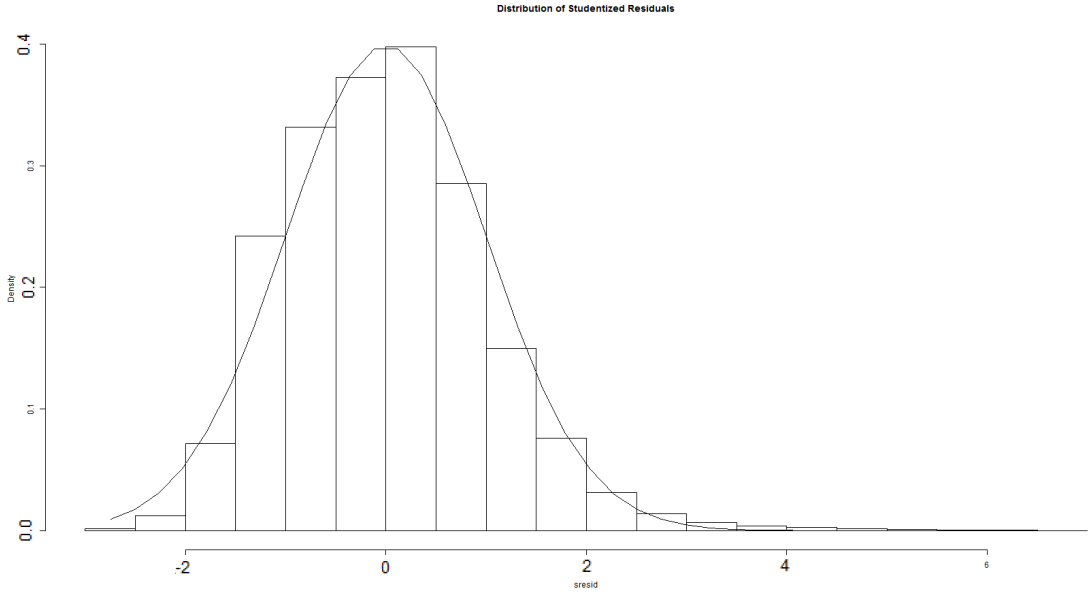
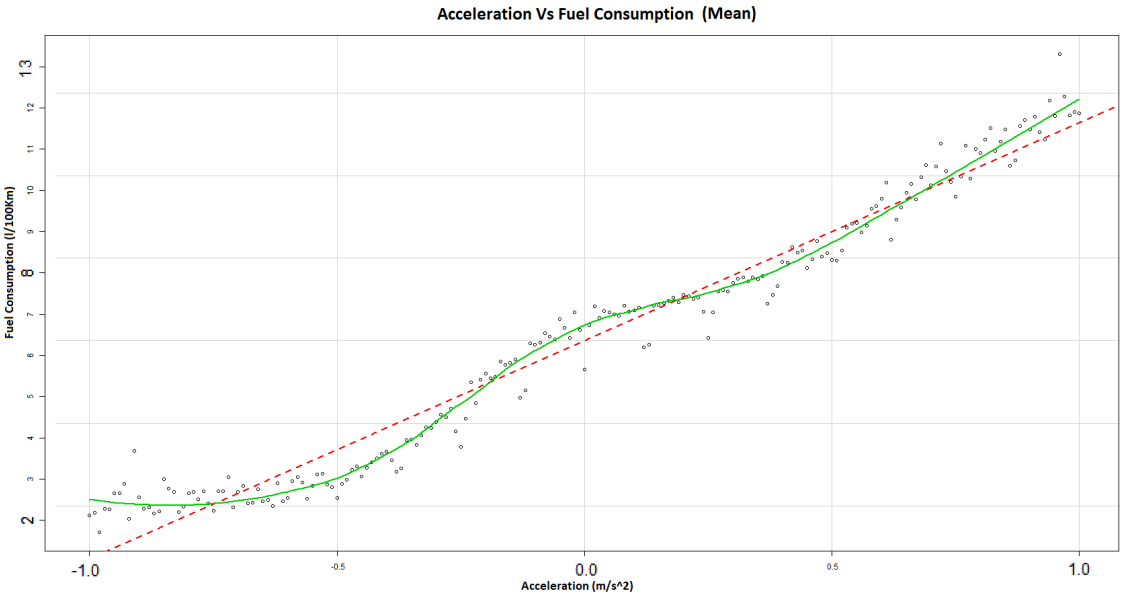


Figure 6.6: Component+Residual Plots, Linear Model Summary And Studentized Residuals for Acceleration Versus Fuel Consumption




```
Residuals:
     Min       1Q   Median       3Q      Max
-1.24608 -0.45216 -0.01456  0.39651  2.13289

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  6.36222    0.04166  152.71   <2e-16 ***
accel        5.27544    0.07180   73.47   <2e-16 ***
Residual standard error: 0.5907 on 199 degrees of freedom
Multiple R-squared: 0.9644, Adjusted R-squared: 0.9643
F-statistic:  5398 on 1 and 199 DF,  p-value: < 2.2e-16
```

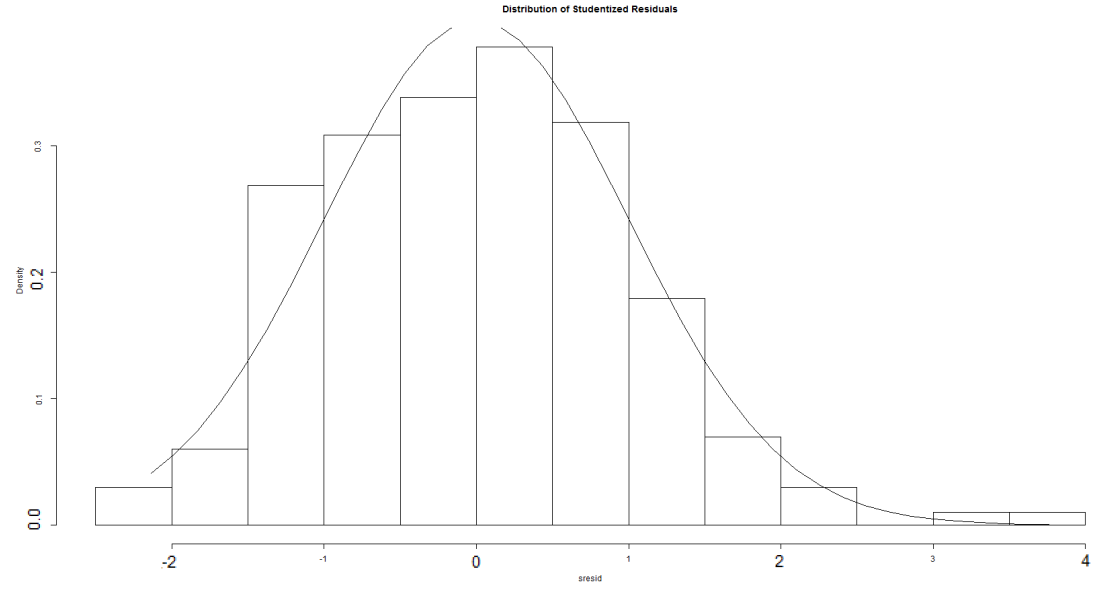
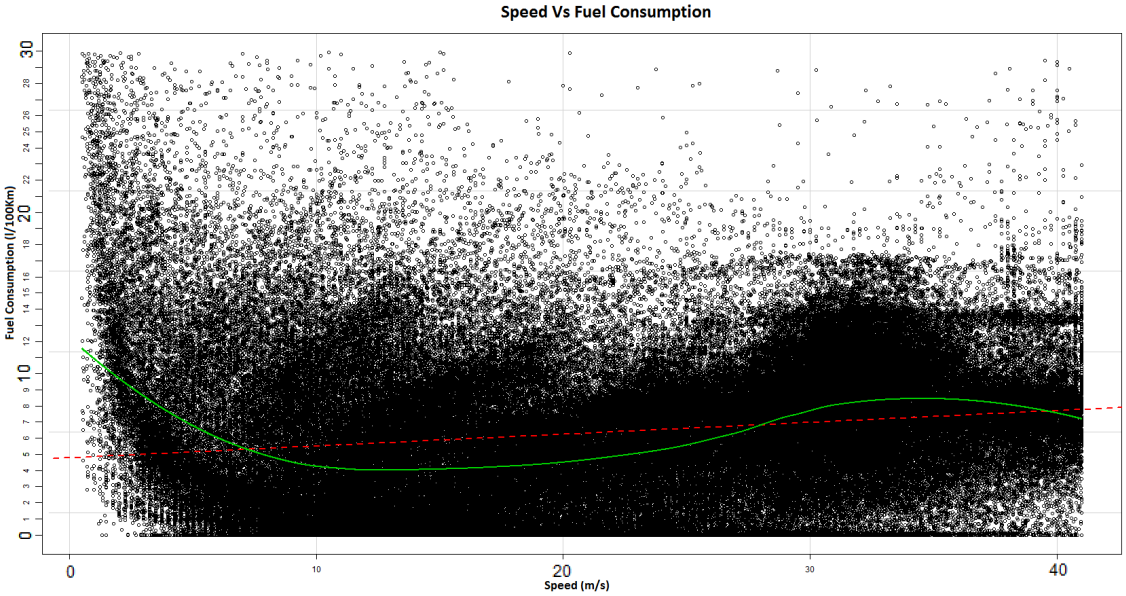


Figure 6.7: Component+Residual Plots, Linear Model Summary And Studentized Residuals for Acceleration Versus Fuel Consumption (Mean)




```
Residuals:
    Min      1Q  Median      3Q     Max
-7.7886 -3.4490 -0.1612  2.4517 25.0921

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 4.787083   0.022586  211.95   <2e-16 ***
speed       0.073208   0.000933   78.46   <2e-16 ***
Residual standard error: 4.497 on 216120 degrees of freedom
Multiple R-squared: 0.0277, Adjusted R-squared: 0.02769
F-statistic:  6157 on 1 and 216120 DF,  p-value: < 2.2e-16
```

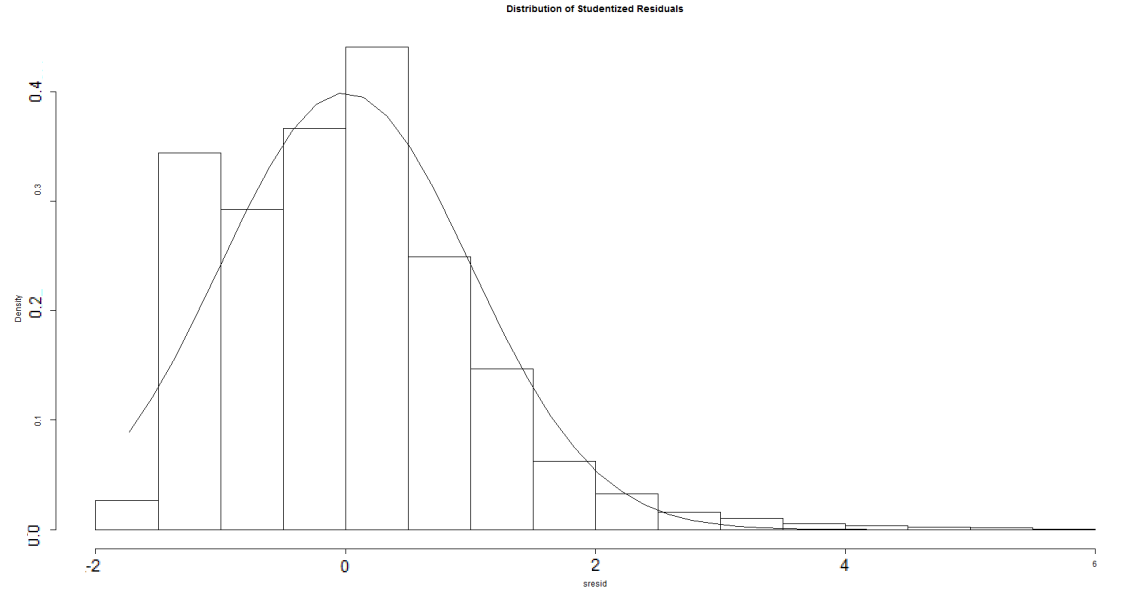
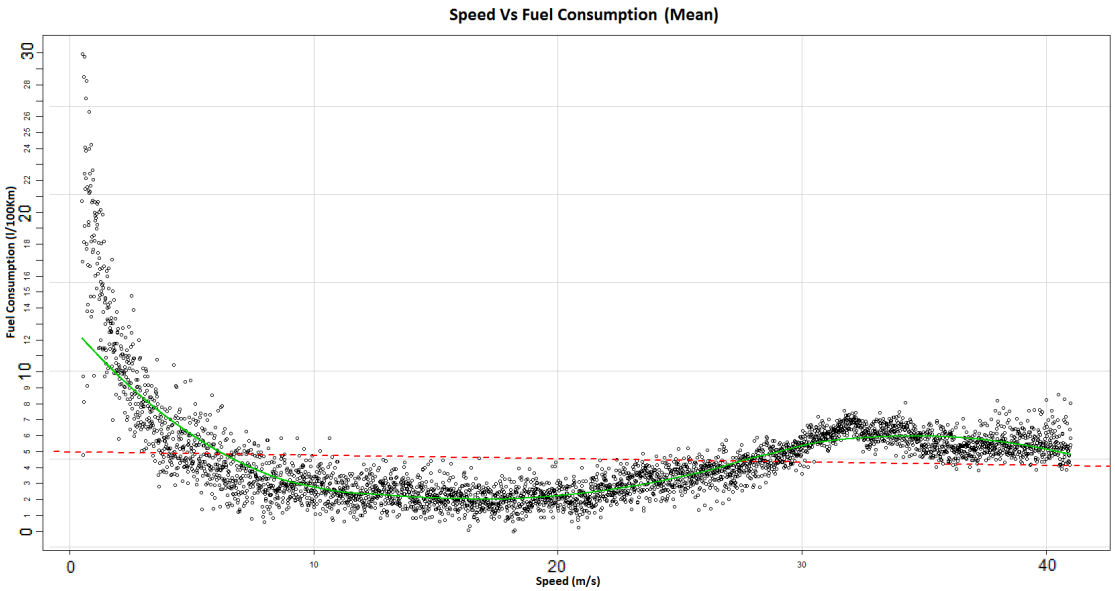


Figure 6.8: Component+Residual Plots, Linear Model Summary And Studentized Residuals for Speed Versus Fuel Consumption




```
Residuals:
    Min      1Q  Median      3Q     Max
-4.1539 -1.8894 -0.4811  1.2062 22.5734

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  7.356889   0.088591   83.04  < 2e-16 ***
speed  -0.019487   0.003718   -5.24 1.68e-07 ***
Residual standard error: 2.766 on 4047 degrees of freedom
Multiple R-squared: 0.00674,    Adjusted R-squared: 0.006495
F-statistic: 27.46 on 1 and 4047 DF,  p-value: 1.683e-07
```

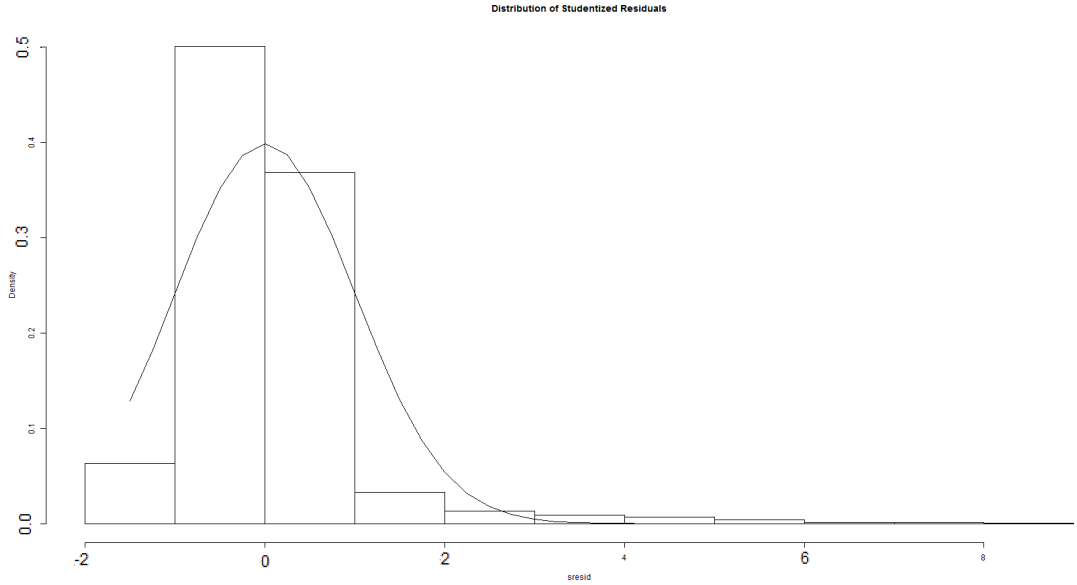


Figure 6.9: Component+Residual Plots, Linear Model Summary And Studentized Residuals for Speed Versus Fuel Consumption (Mean)

So the next step was to look at the third **R**, **R**eexpression.

The first thought when looking at the mean fuel consumption versus speed plot was

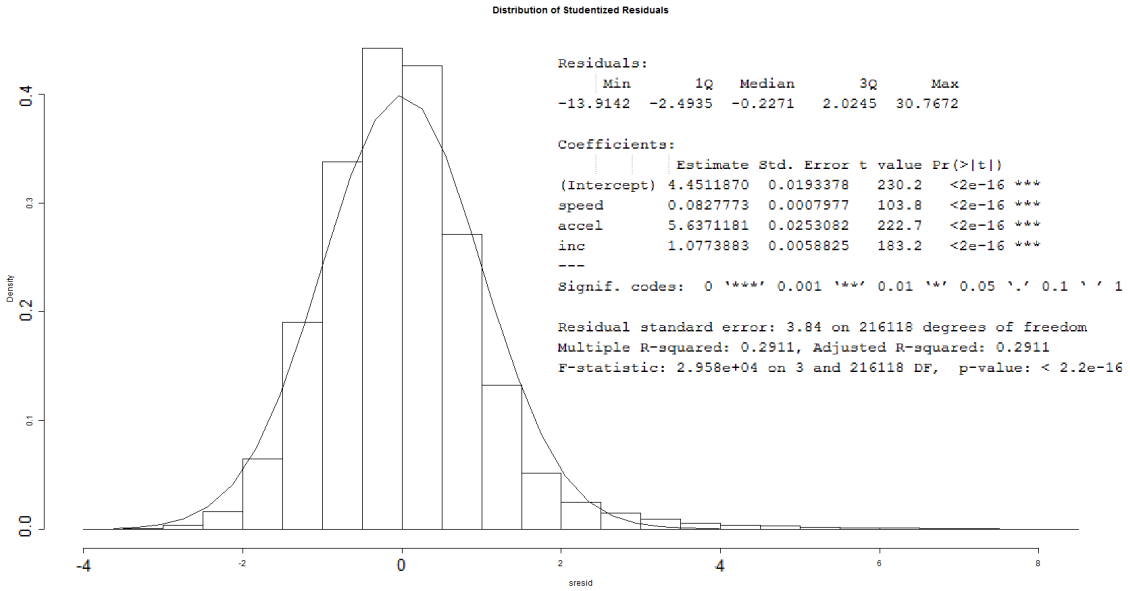


Figure 6.10: Linear Model Summary And Studentized Residuals for All Values Versus Fuel Consumption

that at lower speeds (below 20 metres per second) it seemed like an exponential curve. So a logarithmic transform was experimented (see Figure 6.11).

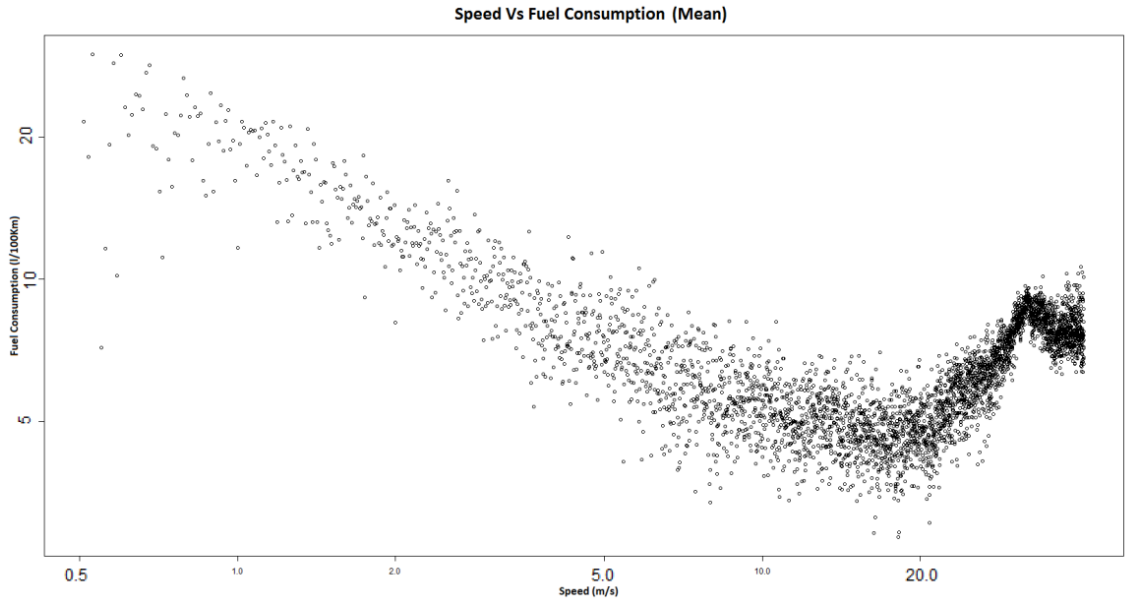


Figure 6.11: Logarithmic Transform on Speed and Fuel Consumption

However, while it seemed that city consumption becomes linear, as it shares a correlation of -0.9048564 when discarding the velocities above 20 metres per second, the plot

becomes very bowed at high speeds.

Even though there are many useful transformation techniques, like the Box-Cox Transformation [Sak92], looking at the component residual plot of speed versus fuel consumption it is suggested that a polynomial function might fit. It is known that a polynomial of sufficient order will fit any dataset. If the order of the polynomial is *n-1*, where *n* is the number of points to be fitted, the fit will be perfect. However, 'overfitting' a dataset causes the problem of failing to generalize new data. The computational power needed to fit a thousand points is also wasted if a lower order polynomial has the possibility of fitting. So a third order polynomial was experimented which yielded a better fit. Figure 6.12 represents speed vs fuel consumption fitted with a third (red) and fourth (green) order polynomial.

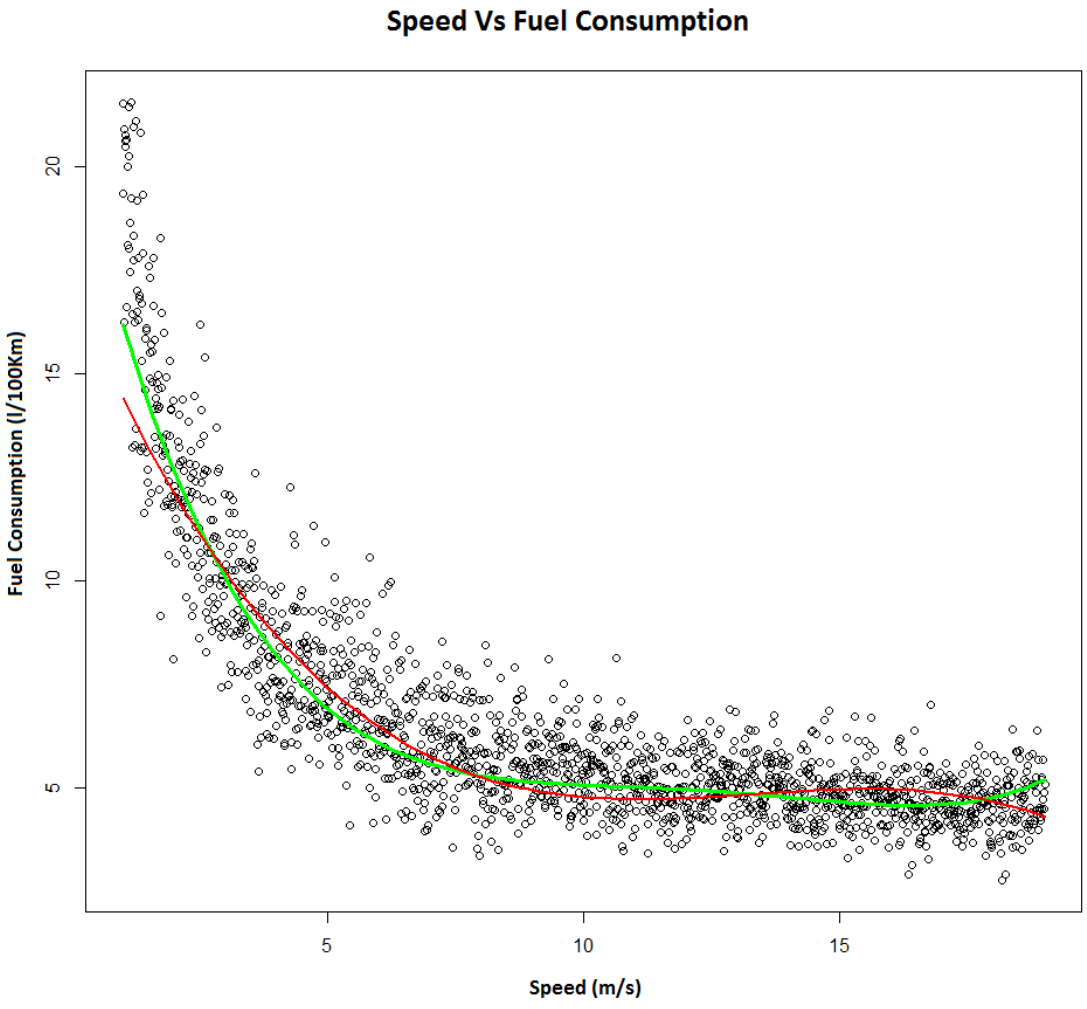


Figure 6.12: Speed vs Fuel Consumption (Mean) plot for lower speeds

The third order polynomial seems a good enough fit, and this is supported by the

difference of the residual standard error between the third and fourth order polynomial, which is approximately 0.005 in this case.

So the proposed formula became:

$$FuelConsumption \approx a + b * Acceleration + c * Inclination + d * Speed + e * Speed^2 + f * Speed^3$$

In order to evaluate if this model yielded a better fitting over the first linear model, both models were fitted in each vehicle's dataset and the dataset with all vehicles together, and the residuals were compared. Also for every tested fit, it was experimented with and without outliers to help measure the influence these 'distributional monsters' can have. The result was Table 6.1.

Table 6.1: Standard residual error for each model and some vehicles

Vehicle	LO	NLO	L	NL
All	3.668	3.506	4.626	4.312
14	3.881	3.803	4.847	4.590
42	2.967	2.662	3.908	3.317
45	4.269	4.222	5.269	5.052

The column names mean: **LO** - first model without outliers, **NLO** - second model without outliers, **L** - first model with outliers, **NL** - second model without outliers.

Analysing Table 6.1 it is seen that the second model presents less error than the first. Also, the outlier removal greatly reduces the standard residual error. However, the residuals between vehicles show a notorious difference. This could be a direct result of the driving style and aggressiveness. So while both models might not be the best solutions for individual vehicle dynamics, they can fit with emission maps, as with a large number of vehicles the excess residual balances with the default. Also, it is shown that the studentized residuals in both models (see Figures 6.10 and 6.13) follow a normal distribution.

So using the second model, fuel consumption can be approximately estimated with:

$$FuelConsumption \approx 10.1200 + 1.0980 * Inclination + 5.9010 * Acceleration - 0.8440 * Speed + 0.0354 * Speed^2 - 0.0003 * Speed^3$$

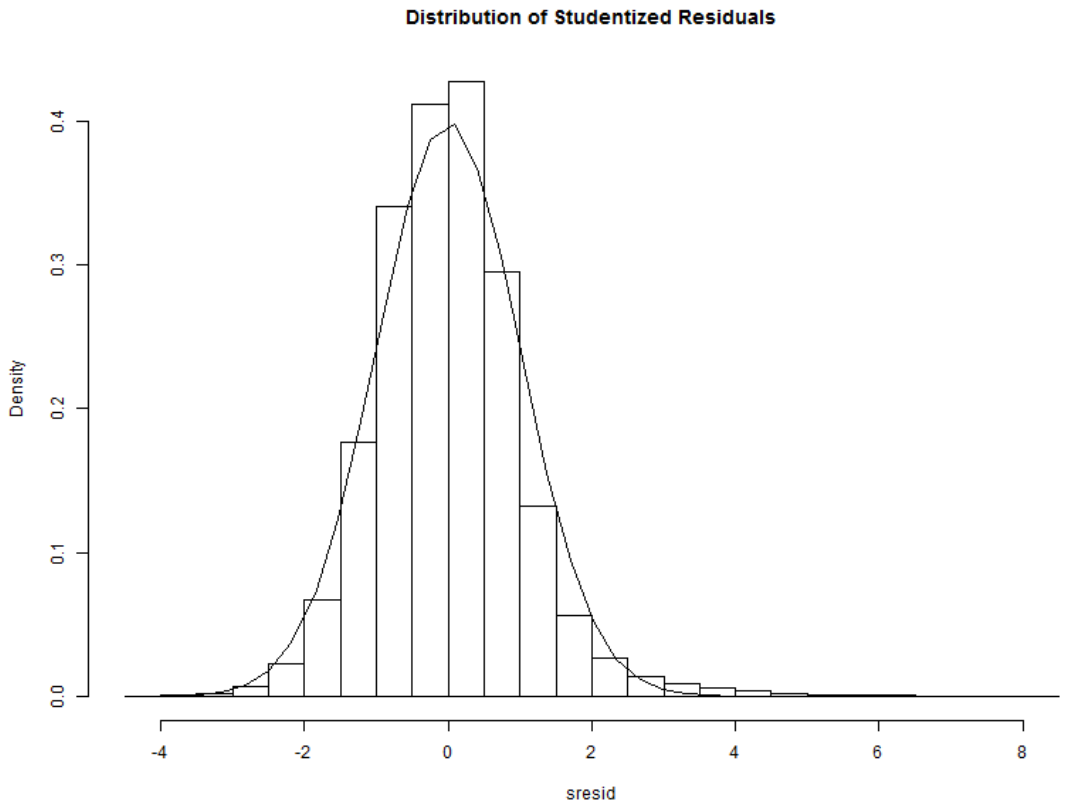


Figure 6.13: Studentized Residuals for All Values Versus Fuel Consumption

6.3 Conclusions

Various methods were tried in order to find the best fit for the presented dataset. The proposed solution shows a standard residual error of about 3 litres and a half for all vehicles but a normal distribution of studentized residuals. So, as referred, it is better fitted with emission maps than with individual vehicles. This solution is therefore a valid contribution to enhance the current urban monitor. Also, as seen in the previous chapter, the gearbox has an impact on consumption. So this formula is only valid when the engine is not in idle speed. The data was filtered by range, but using some techniques to discover the engaged shift, like a function of RPM and speed, could improve the outcome.

Chapter 7

Conclusions

In a society largely dependent on a finite amount of fossil fuels, which provokes a potential rise in price of this natural resource, an algorithm for fuel consumption estimation from GPS data provides an innovative solution that aims to supply a new way to inform citizens about their driving behaviour, reducing mobility related costs and improving quality of life.

This thesis encompassed a broad range of fields, since just the topic of mobility spawns a lot of research. Besides giving a background of vehicle mechanics and OBD history and usefulness, this document discussed fuel consumption models, applications that use OBD, mobile and collaborative sensing platforms, UI patterns, Machine Learning methods, and focused on the GPS and OBD technology and algorithms for fuel consumption estimation using GPS and OBD sensor data.

It was possible to conclude that some OBD applications provided useful information for OBD data handling and, alongside the analysed emission models, a valuable guideline for the implementation of the fuel consumption algorithm from OBD data. The ELM327 manual also proved fundamental for the implementation and comprehension of the OBD communication protocols.

The models also allowed a better understanding of emissions and vehicle dynamics, led to the learning of products and projects that use them, and allowed to understand their importance and impact on a society that highly depends on mobility.

The experience with the Android platform also proved to be challenging but fruitful, as the barriers that emerged resulted in research about the components like Bluetooth, and even different smartphones, and a closer look into the Android source code, which enabled an even bigger comprehension of this platform.

Since there is a lot of focus on the GPS, there was also a lot of research into the space technology and its navigation and localization methods.

The result was a regression model that provides fuel consumption in real time, that implemented in a data gathering mobile application like MyDrivingDroid, and used in a collaborative fashion, can generate even more data. The data mining phase resulted in a lot of stored data. In a global perspective, if users abide the previous proposition, on a wide scale, new researches can also be created because a large scale data mining generates a lot of information. And information is power. Information is a currency of today's world. And the sharing of information generates new opportunities as it is noted in the mobile and collaborative sensing platforms topic.

Not only can users compare their driving behaviour, but this can also be a potential tool in creating fuel friendly driving patterns, possibly recommend alternative mobility solutions, or even generate wide scale emission maps.

References

[AB03] Rahmi Akçelik and Mark Besley. Operating cost, fuel consumption, and emission models in aaSIDRA and aaMOTION. *25th Conference of Australian Institutes of Transport Research*, (December 2003):3–5, 2003.

[ABH12] Shahid Ayub, A Bahraminisaab, and B Honary. A Sensor Fusion Method for Smart phone Orientation Estimation. *cms.livjm.ac.uk*, 2012.

[AFO] Adriano Alessandrini, Francesco Filippi, and Fernando Ortenzi. Consumption calculation of vehicles using OBD data. *epa.gov*.

[ALM11] Jong Hoon Ahnn, Uichin Lee, and Hyun Jin Moon. GeoServ: A Distributed Urban Sensing Platform. *2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 164–173, May 2011.

[AR03] Kyoungho Ahn and H Rakha. VT-MICRO FRAMEWORK FOR MODELING OF HIGH EMITTING VEHICLES. *E-print Network*, (540), 2003.

[ARTV02] Kyoungho Ahn, Hesham Rakha, Antonio Trani, and Michel Van Aerde. Estimating Vehicle Fuel Consumption and Emissions based on Instantaneous Speed and Acceleration Levels. *Journal of Transportation Engineering*, 128(2):182–190, March 2002.

[BAY+06] M Barth, F An, T Younglove, G Scora, and C Levine. Comprehensive Modal Emissions Model (CMEM). *Center for Environmental Research & Technology*, (June), 2006.

[Bor07] Kanok Boriboonsomsin. Evaluating Air Quality Benefits of Freeway High-Occupancy Vehicle Lanes in Southern California. *Transportation Research Record*, (951):137–147, 2007.

[CCL10] Francesco Calabrese, Massimo Colonna, and Piero Lovisolo. Real-time urban monitoring using cell phones: A case study in Rome. *IEEE Transactions on Intelligent Transportation Systems*, 12(1):141–151, March 2010.

[CCN02] Alessandra Cappiello, I Chabini, and EK Nam. A statistical model of vehicle emissions and fuel consumption. *Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference*, (617):1–25, 2002.

[CGH+06] S. B. Carter, S. S. Gartner, M. R. Haines, A. L. Olmstead, R. Sutch, and G. Wright. *Historical Statistics of the United States: Earliest Times to the Present*. Historical Statistics of the United States: Earliest Times to the Present: Millennial Editio. Cambridge University Press, 2006.

[Cha09] Tom Chalko. Estimating Accuracy of GPS Doppler Speed Measurement using Speed Dilution of Precision (SDOP) parameter. 2009.

[CQSK11] T Camacho, F Quintal, Michelle Scott, and V Kostakos. Towards Egocentric Fuel Efficiency Feedback. *Proceedings of the workshop on PINC: Persuasion, Influence, Nudge & Coercion through mobile devices at ACM CHI 2011, Vancouver, Canada*, pages 6–8, 2011.

[DR02] Yonglian Ding and HESHAM RAKHA. Trip-based explanatory variables for estimating vehicle fuel consumption and emission rates. *Water, Air, & Soil Pollution: Focus*, 2(5-6):61–77, 2002.

[ELM11] ELM327 OBD to RS232 Interpreter. *Electronics*, pages 1–51, 2011.

[FD12] Michel Ferreira and Pedro M. D'Orey. On the Impact of Virtual Traffic Lights on Carbon Emissions Mitigation. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):284–295, March 2012.

[Fen07] Chunxia Feng. Transit bus load-based modal emission rate model development. *Georgia Institute of Technology*, (July), 2007.

[Gom12] Maria Gomez. Dangers of driving and using your cell phone. *Medical News Today*, 2012.

[GT08] Judith Gebauer and Y Tang. User requirements of mobile technology: results from a content analysis of user reviews. *Information-Knowledge-Systems Management - Enterprise Mobility: Applications, Technologes and Strategies*, 7(1,2):101–119, 2008.

[Haa09] Hein De Haas. Mobility and Human Development. *Oxford: International Migration Institute, University of Oxford.*, 2009.

[HBZC06] Bret Hull, Vladimir Bychkovsky, Yang Zhang, and Kevin Chen. CarTel: a distributed mobile sensor computing system. *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 125–138, 2006.

[HD09] H.L. Hwang and S.C. Davis. Off-Highway Gasoline Consumption Estimation Models Used in the Federal Highway Administration Attribution and Process. *ORNL/TM-2009/222*, 2009.

[HDY+05] Tao Huai, S Thomas D Durbin, Ted Younglove, George Scora, Matthew Barth, and Joseph M Norbeck. Vehicle specific power approach to estimating on-road NH3 emissions from light-duty vehicles. *Environmental science & technology*, 39(24):9595–600, December 2005.

[IBL06] Luc Int Panis, Steven Broekx, and Ronghui Liu. Modelling instantaneous traffic emission and the influence of traffic speed limits. *The Science of the total environment*, 371(1-3):270–85, December 2006.

[Iri10] Adrian Irimescu. Study of Volumetric Efficiency for Spark Ignition Engines Using Alternative Fuels. (2):149–154, 2010.

[ISO] ISO 15031-5. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=50816.

[KGG07] Ryan Keefe, James Griffin, and John D. Graham. The Benefits and Costs of New Fuels and Engines for Cars and Light Trucks. *Santa Monica, CA: RAND Corporation*, 2007.

[KH97] ED Kaplan and CJ Hegarty. Understanding GPS. *Artech House*, 1997.

[Li12] Chunxiao Li. An Open Traffic Light Control Model for Reducing Vehicles CO2 Emissions Based on ETC Vehicles. *Vehicular Technology, IEEE Transactions*, 61(1):97–110, 2012.

[Lit03] RG Little. Toward more robust infrastructure: observations on improving the resilience and reliability of critical systems. *System Sciences, 2003. Proceedings of the 36th*, 2003.

[LLL+12] Kun Li, M Lu, Fenglong Lu, Q Lv, and L Shang. Personalized Driving Behavior Monitoring and Analysis for Emerging Hybrid Vehicles. *Pervasive 2012: Proceedings of the 10th International Conference on Pervasive Computing*, 2012.

[LML10] ND Lane, Emiliano Miluzzo, and Hong Lu. A survey of mobile phone sensing. *Communications Magazine, IEEE*, (September):140–150, 2010.

[McC99] PM McClintock. Vehicle specific power: a useful parameter for remote sensing and emission studies. *Vehicle Emissions*, 1999.

[NME02] NMEA 0183 Standard For Interfacing Marine Electronic Devices. 2002.

[Ono04] Shigeru Onoda. PSIM-based modeling of automotive power systems: conventional, electric, and hybrid electric vehicles. *Vehicular Technology, IEEE Transactions*, 53(2):390–400, 2004.

[PDP+12] Radivoje B. PEŠIĆ, Aleksandar Lj. DAVINIĆ, Snežana D. PETKOVIĆ, Dragan S. TARANOVIĆ, and Danijela M. MILORADOVIĆ. Aspects of volumetric efficiency measurement for reciprocating engines. 4, 2012.

[Pea11] Ronald K Pearson. *Exploring data in engineering, the sciences and medicine*. New York ; Oxford : Oxford University Press, 2011. Formerly CIP.

[PvHD06] Adam Pollard, J von Hafen, and M Dottling. Winner-towards ubiquitous wireless access. *Vehicular Technology Conference, 2006. VTC 2006-Spring. IEEE 63rd*, 00(c):42–46, 2006.

[RA03] Hesham Rakha and Kyoungho Ahn. Comparison of MOBILE5a, MOBILE6, VT-MICRO, and CMEM models for estimating hot-stabilized light-duty gasoline vehicle emissions. *Canadian Journal of Civil Engineering*, 30(6):1010, 2003.

[RA04] Hesham Rakha and Kyoungho Ahn. Development of VT-Micro model for estimating hot stabilized light duty vehicle and truck emissions. *Transportation Research Part D: Transport and Environment,*, 9(0536):49–74(26), 2004.

[RAV+11] Joao G. P. Rodrigues, Ana Aguiar, Fausto Vieira, Joao Barros, and Joao P. Silva Cunha. A mobile sensing architecture for massive urban scanning. *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1132–1137, October 2011.

[RKBG07] S Kahn Ribeiro, S Kobayashi, M Beuthe, and J Gasca. Transport and its infrastructure. *Climate Change 2007: Mitigation. Contribution of Working Group III to the Fourth Assessment Report of the IPCC*, 2007.

[SAE] SAE J1979. http://standards.sae.org/j1979_201202/.

[Sak92] RM Sakia. The Box-Cox transformation technique: a review. *The statistician*, 41(2):169, 1992.

[SB11] Marcin Seredynski and Pascal Bouvry. A survey of vehicular-based cooperative traffic information systems. *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 163–168, October 2011.

[TBG10] Arvind Thiagarajan, James Biagioni, and T Gerlich. Cooperative transit tracking using smart-phones. *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, pages 85–98, 2010.

[VHM91] P. F. Velleman, D. C. Hoaglin, and D. S. Moore. *Perspectives on Contemporary Statistics*. Historical Statistics of the United States: Earliest Times to the Present: Millennial Edition. The Mathematical Association of America, 1991.

[Vin75] T. Vincenty. Direct and Inverse Solutions of Geodesics on the Ellipsoid with Application of Nested Equations. XXIII(176), 1975.

[Wan09] K Wanglund. Evaluation of GPS Velocity and Altitude Data used for Road Grade Estimation. (June), 2009.

[Yue08] H Yue. *Mesoscopic fuel consumption and emission modeling*. PhD thesis, Virginia Tech, 2008.

[ZN01] Y Zheng and D Niemeier. A Grid-Based Mobile Sources Emissions Inventory Model. *Institute of Transportation Studies*, 94274(18), 2001.

[ZZGD06] JAson ZHang, KEfei Zhang, RON GRenfell, and ROD DEakin. On the Relativistic Doppler Effect for Precise Velocity Determination using GPS. 80(DoD 1996):104–110, 2006.

Appendix A

Appendix

A.1 Relevant Figures

This section is reserved for some relevant figures.

A.1.1 Data Gathering Evolution

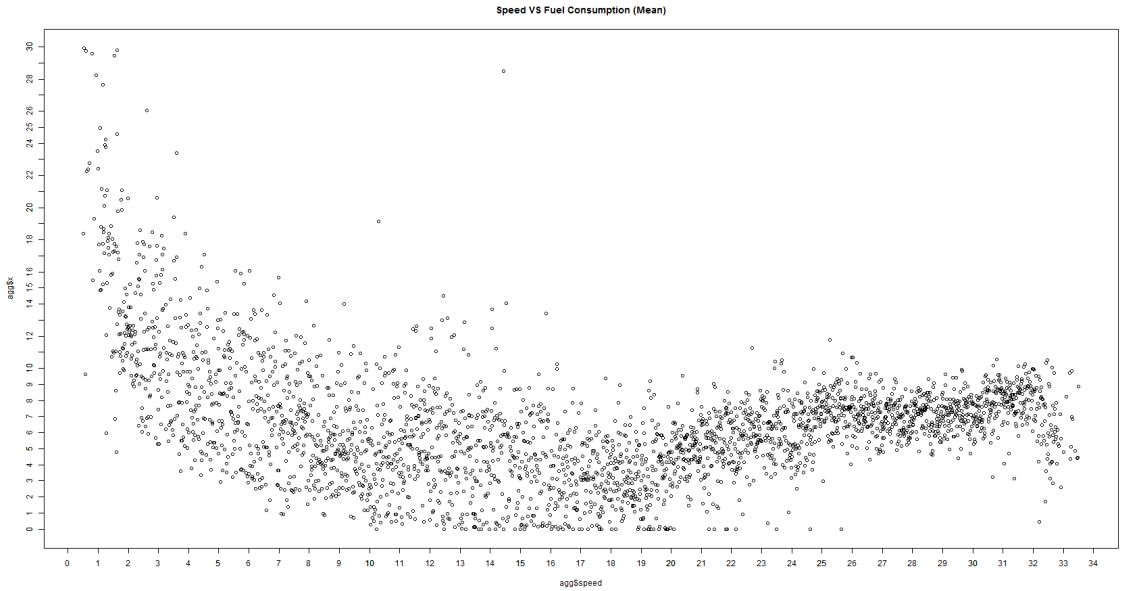


Figure A.1: Speed Vs Fuel Consumption (Mean) on 20 Trips

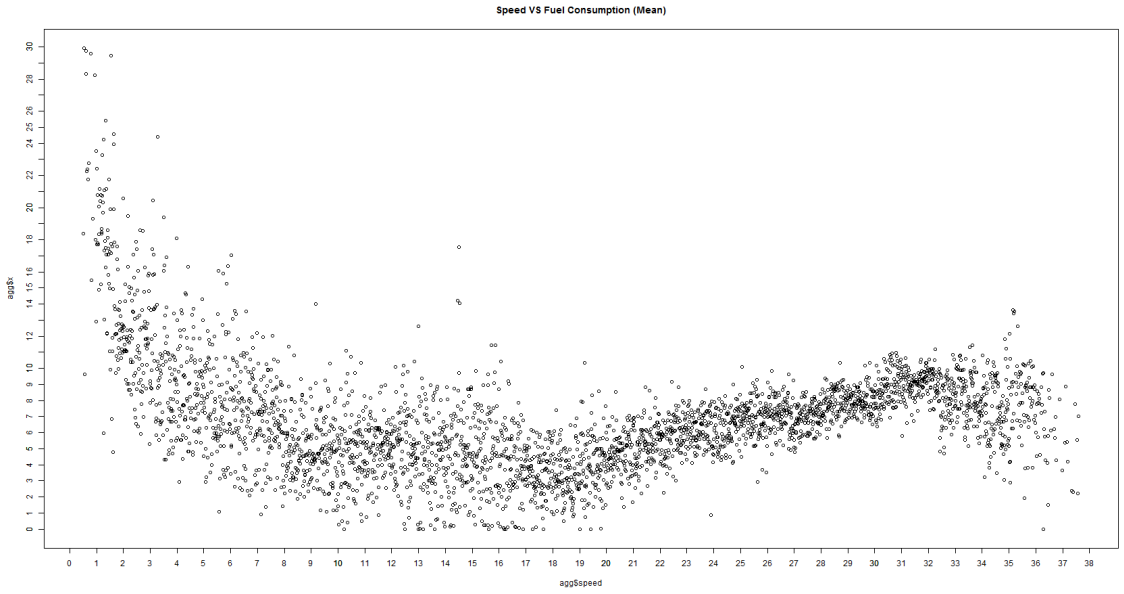


Figure A.2: Speed Vs Fuel Consumption (Mean) on 40 Trips

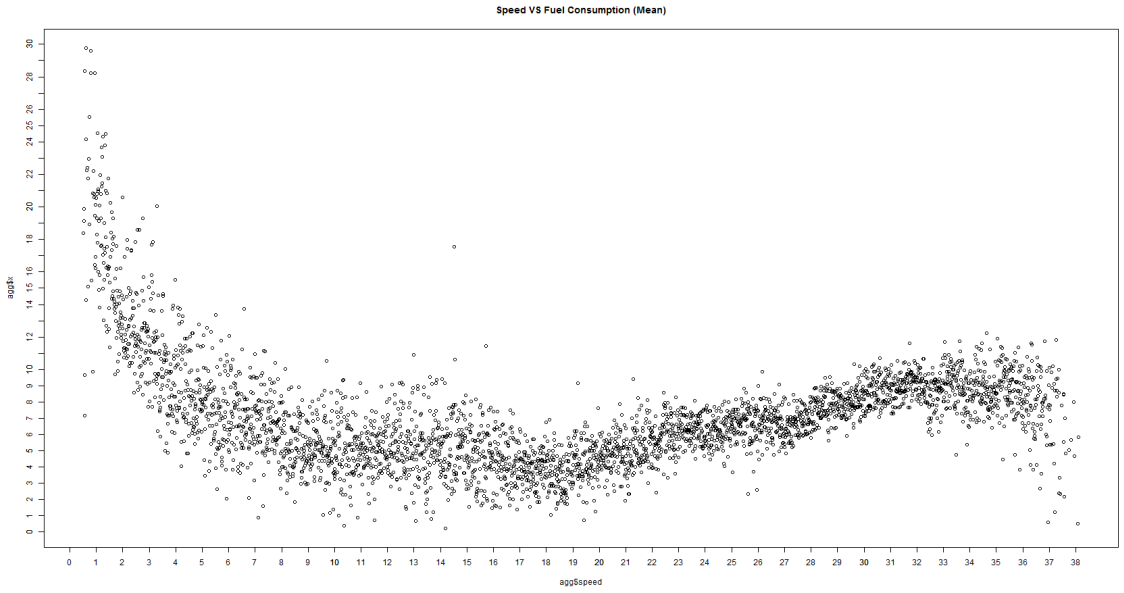


Figure A.3: Speed Vs Fuel Consumption (Mean) on 60 Trips

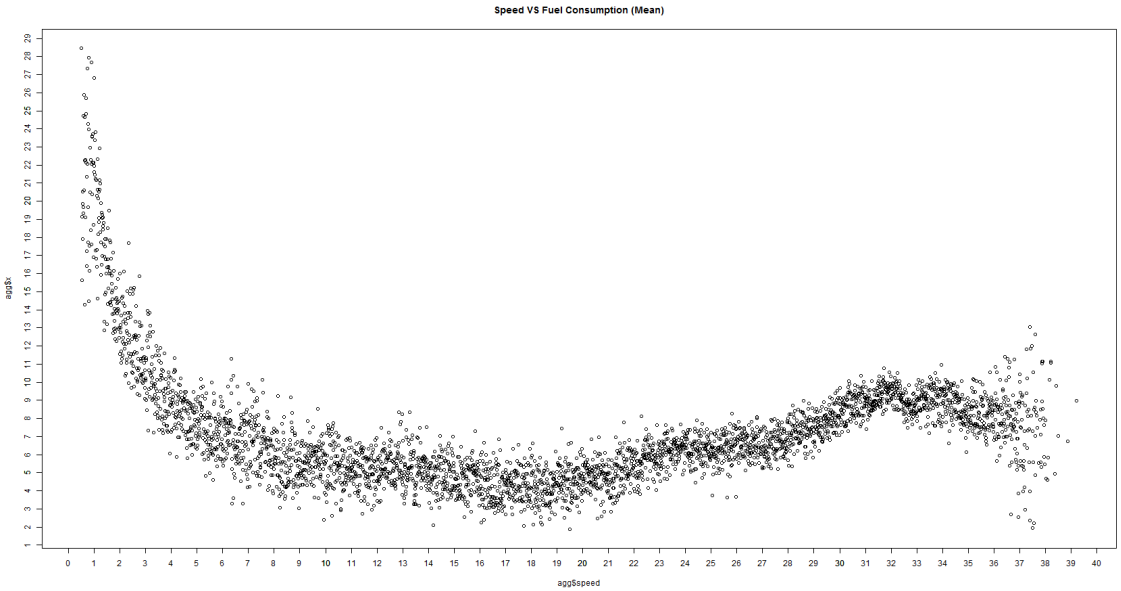


Figure A.4: Speed Vs Fuel Consumption (Mean) on 80 Trips

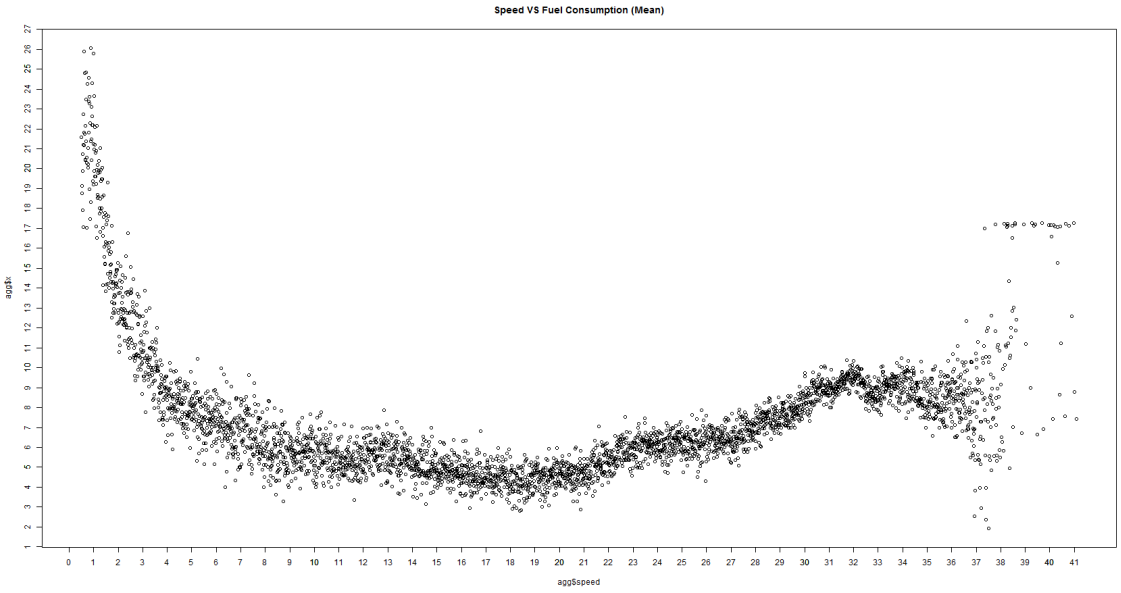


Figure A.5: Speed Vs Fuel Consumption (Mean) on 100 Trips

A.1.2 Formula Summary

```
Call:
lm(formula = fc ~ I(inc) + I(accel) + I(speed) + I(speed^2) +
    I(speed^3))

Residuals:
     Min       1Q   Median       3Q      Max
-16.5584  -2.2188  -0.0795   1.8983  29.6864

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.343e+01  5.047e-02   266.1   <2e-16 ***
I(inc)       1.065e+00  5.424e-03   196.4   <2e-16 ***
I(accel)     5.752e+00  2.334e-02   246.5   <2e-16 ***
I(speed)    -1.458e+00  8.881e-03  -164.1   <2e-16 ***
I(speed^2)   6.905e-02  4.565e-04   151.3   <2e-16 ***
I(speed^3)  -9.014e-04  7.010e-06  -128.6   <2e-16 ***

Residual standard error: 3.539 on 216116 degrees of freedom
Multiple R-squared: 0.3977,     Adjusted R-squared: 0.3977
F-statistic: 2.854e+04 on 5 and 216116 DF,  p-value: < 2.2e-16
```

Figure A.6: Summary of Final Formula

A.2 Relevant Code Snippets

This section is reserved for some relevant pseudo-code.

A.2.1 Connecting To An OBD Bluetooth Device And Sending, Receiving And Parsing Data

```
// Get BluetoothDevice
BluetoothDevice = BluetoothAdapter.getRemoteDevice(deviceMACAddress);

// Get a BluetoothSocket to connect with the BluetoothDevice
Method m = BluetoothDevice.getClass().
getMethod("createRfcommSocket", new Class[] {int.class});
BluetoothSocket = (BluetoothSocket) m.invoke
(BluetoothDevice, Integer.valueOf(channel));
BluetoothAdapter.cancelDiscovery();
BluetoothSocket.connect();

// Get Streams
InputStream = BluetoothSocket.getInputStream();
OutputStream = BluetoothSocket.getOutputStream();

// Send Command
```

```
OutputStream.write(CommandBytes));

// Receive Response
do {
bytes = InputStream.read();
ByteArrayOutputStream.write(bytes);
} while (((char) bytes) != '>' && bytes != -1);

// Parse Response
String = ByteArrayOutputStream.toUTFString();
// normalize newlines
String = String.replaceAll("\\r", "\n");
// delete empty lines
String = String.replaceAll("\\n\\n", "\n");
// delete spaces
String = String.replaceAll("[ \\t\\f]", "");
// delete 2nd+ headers on multiline responses
String = String.replaceAll("\\b49020[2-9]", "");
// delete line-count on multiline responses
String = String.replaceAll("(\\b4902)01", "$1");
// delete byte-count on multiline CAN responses
String = String.replaceAll("\\b[1-9]:", "");
// delete byte-count on multiline CAN responses
String = String.replaceAll("\\b[0-9]{3}\\n0:", "");
// delete newlines
String = String.replaceAll("\\n", "");
// put in the right format (MODE=PID=RESPONSE)
String = String.replaceAll
(".*?\\b4([159])([0-9a-fA-F]{2})((?:[0-9a-fA-F]{2})+)[><]", "0$1=$2=$3>")
String[] = String.split(">"); // split responses

// iterate responses
for (i=0;i<String[].length;i++) {
// divide response
String[] = String[i].split("=");
// test responses
if (OBDPID.internalCommand) {
...
} else if (OBDPID.supportedPIDs) {
...
} else if (OBDPID.vin) {
...
} else if (...) {
...
}

if (OBDPID.ERROR AND errorCounter > errorCounterMaximum) {
```

```
rebootOBD();
}
}
```

A.2.2 Calculating Distance And Bearing

This function computes the approximate distance in meters between two locations and the initial and final bearings of the shortest path between them. Distance and bearing are defined using the WGS84 ellipsoid. Based on the "Inverse Formula" (section 4), returns an array of doubles that holds the results. The computed distance is stored in result[0]. The initial bearing is stored in result[1]. The final bearing is stored in result[2].

```
public static final double[] calculateDistanceAndBearing
(double InitialLatitude, double InitialLongitude,
double FinalLatitude, double FinalLongitude) {

int MAXITERS = 20;
// Convert lat/long to radians
InitialLatitude *= Math.PI / 180.0d;
FinalLatitude *= Math.PI / 180.0d;
InitialLongitude *= Math.PI / 180.0d;
FinalLongitude *= Math.PI / 180.0d;

double a = 6378137.0d; // WGS84 major axis
double b = 6356752.3142d; // WGS84 semi-major axis
double f = (a - b) / a;
double aSqMinusBSqOverBSq = (a * a - b * b) / (b * b);

double L = FinalLongitude - InitialLongitude;
double A = 0.0d;
double U1 = Math.atan((1.0 - f) * Math.tan(InitialLatitude));
double U2 = Math.atan((1.0 - f) * Math.tan(FinalLatitude));

double cosU1 = Math.cos(U1);
double cosU2 = Math.cos(U2);
double sinU1 = Math.sin(U1);
double sinU2 = Math.sin(U2);
double cosU1cosU2 = cosU1 * cosU2;
double sinU1sinU2 = sinU1 * sinU2;

double sigma = 0.0d;
double deltaSigma = 0.0d;
double cosSqAlpha = 0.0d;
double cos2SM = 0.0d;
double cosSigma = 0.0d;
double sinSigma = 0.0d;
double cosLambda = 0.0d;
double sinLambda = 0.0d;
```

```
double lambda = L; // initial guess
for (int iter = 0; iter < MAXITERS; iter++) {
double lambdaOrig = lambda;
cosLambda = Math.cos(lambda);
sinLambda = Math.sin(lambda);
double t1 = cosU2 * sinLambda;
double t2 = cosU1 * sinU2 - sinU1 * cosU2 * cosLambda;
double sinSqSigma = t1 * t1 + t2 * t2; // (14)
sinSigma = Math.sqrt(sinSqSigma);
cosSigma = sinU1sinU2 + cosU1cosU2 * cosLambda; // (15)
sigma = Math.atan2(sinSigma, cosSigma); // (16)
double sinAlpha = (sinSigma == 0) ? 0.0 :
cosU1cosU2 * sinLambda / sinSigma; // (17)
cosSqAlpha = 1.0 - sinAlpha * sinAlpha;
cos2SM = (cosSqAlpha == 0) ? 0.0 :
cosSigma - 2.0 * sinU1sinU2 / cosSqAlpha; // (18)

double uSquared = cosSqAlpha * aSqMinusBSqOverBSq; // defn
A = 1 + (uSquared / 16384.0d) * // (3)
(4096.0d + uSquared *
(-768d + uSquared * (320.0d - 175.0d * uSquared)));
double B = (uSquared / 1024.0d) * // (4)
(256.0d + uSquared *
(-128.0d + uSquared * (74.0d - 47.0d * uSquared)));
double C = (f / 16.0d) *
cosSqAlpha *
(4.0d + f * (4.0d - 3.0d * cosSqAlpha)); // (10)
double cos2SMSq = cos2SM * cos2SM;
deltaSigma = B * sinSigma * // (6)
(cos2SM + (B / 4.0d) *
(cosSigma * (-1.0d + 2.0d * cos2SMSq) -
(B / 6.0d) * cos2SM *
(-3.0d + 4.0d * sinSigma * sinSigma) *
(-3.0d + 4.0d * cos2SMSq)));

lambda = L +
(1.0d - C) * f * sinAlpha *
(sigma + C * sinSigma *
(cos2SM + C * cosSigma *
(-1.0d + 2.0d * cos2SM * cos2SM))); // (11)

double delta = (lambda - lambdaOrig) / lambda;
if (Math.abs(delta) < 1.0e-12) {
break;
}
}
```

```
double distance = (double) (b * A * (sigma - deltaSigma));
results[0] = distance;

double initialBearing = (double) Math.atan2(cosU2 * sinLambda,
cosU1 * sinU2 - sinU1 * cosU2 * cosLambda);
initialBearing *= 180.0d / Math.PI;
results[1] = initialBearing;

double finalBearing = (double) Math.atan2(cosU1 * sinLambda,
-sinU1 * cosU2 + cosU1 * sinU2 * cosLambda);
finalBearing *= 180.0d / Math.PI;
results[2] = finalBearing;

return results;
}
```

A.2.3 Calculating Acceleration And Inclination

Least Square Error method is used for both. The difference is that for acceleration the 'x' and 'y' is time and speed, and for inclination it is distance and altitude. Also the maximum distance and interval is different. For acceleration the interval is 3. For inclination the interval doubles if at least 5 points are not found.

```
public static final double[][] LeastSquares
(double[] x, double[] y, double maxDistance, double interval) {
if (x.length != y.length OR x.length < interval) {
return null;
}
int radius = (int) (interval/2);
double[][] yi = new double[x.length][2];
double ln, lx, ly, lxx, lxy;
for (int i = 0; i < x.length; i++) {
if (i < x.length - 1 && x[i + 1] < x[i]) {
return null;
}
ln = 0.0d;
lx = 0.0d;
ly = 0.0d;
lxx = 0.0d;
lxy = 0.0d;
for (int k = -radius; k <= radius; k++) {
if (i + k >= 0 && i + k < x.length && Math.abs(x[i + k] - x[i]) <= maxDistanc
ln++;
lx += x[i + k];
ly += y[i + k];
lxx += x[i + k] * x[i + k];
lxy += x[i + k] * y[i + k];
```

```
}
}
// for inclination only there is an additional conditional
if (ln <= 3) {
radius *= 2;
}
if (ln <= 1) {
// If there's only the middle point in the window, no slope can be found
yi[i] = Double.NaN;
} else {
yi[i] = (ln * lxy - lx * ly) / (ln * lxx - lx * lx);
}
}
return yi;
}
```

A.2.4 Interpolating OBD Data

Linear Interpolation is used on the OBD data. Given the desired x positions to be interpolate and the maximum allowed distance between neighbours, an array of the y values at the desired x positions is returned.

```
public static final double[] interpLinear
(double[] x, double[] y, double[] xi, double maxDistance) {

double[] dx = new double[x.length - 1];
double[] dy = new double[x.length - 1];
double[] slope = new double[x.length - 1];
double[] intercept = new double[x.length - 1];
// Calculate the line equation (i.e. slope and intercept) between each po
for (int i = 0; i < x.length - 1; i++) {
dx[i] = x[i + 1] - x[i];
dy[i] = y[i + 1] - y[i];
slope[i] = dy[i] / dx[i];
intercept[i] = y[i] - x[i] * slope[i];
}
// Perform the interpolation here
double[] yi = new double[xi.length];
for (int j = 0; j < xi.length; j++) {
if ((xi[j] > x[x.length - 1]) || (xi[j] < x[0])) {
yi[j] = Double.NaN;
} else {
int loc = Arrays.binarySearch(x, xi[j]);
if (loc < -1) {
loc = (-loc - 1) - 1;
if( loc == x.length-1 || (maxDistance != 0 && (Math.abs(x[loc+1] - x[loc]
yi[j] = Double.NaN;
} else {
```

```
yi[j] = slope[loc] * xi[j] + intercept[loc];
}
// System.out.println("Found between: " + loc + " and: " + (loc+1) + ", x[loc
}
else {
yi[j] = y[loc];
}
}
}
return yi;
}
```