Automobile Driver Evaluation System

K.A.D.L Anuruddha

108552G

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfillment of the requirements of the Degree of Master in Information Technology.

May 2015

Declaration

I certify that this report does not incorporate, without acknowledgement, any material previously submitted for a degree and to the best of my knowledge and it does not contain any material previously published or written by another person or myself except where due reference is made in text. I also hereby give consent for my dissertation, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organizations.

Abstract

In this project I use Onboard Diagnostic Port of vehicles to monitor the driving styles of motor vehicle drivers. By using the values given by the onboard computer for monitoring the driving style of drivers, it is possible to get reliable information without depending on any human interaction. A Raspberry Pi Model B computer runs the onboard modules of the system. This processes the data collected from the onboard vehicle computer, analyze and uploads to centralized system after enriching with GPS coordinates and the timestamp.

Users can access the web based system which connects to the database and allow them to view near real time/ historical data of vehicles.

Table of Contents

Table of Figures

Chapter 1

Introduction

1.1. Background and Motivation

Driving style of motor vehicle drivers varies from driver to driver. However when a taxi company owns few hundreds to thousands of vehicles in their fleet and these vehicles being driven by hired drivers, it is very difficult to find out their driving styles. Driving style has a direct relationship to fuel efficiency, wear and tear of vehicle parts and ultimately the passenger comfort level and safety. If the vehicle is used to transport goods, reckless driving can damage the transporting goods resulting financial loses. Even though owners publish their telephone numbers with "How is my driving?" sign on the back of their vehicles, not many people used to call this number whenever they notice reckless driving. On the other hand owners cannot rely on the calls coming from strangers and evaluate their drivers. Furthermore the same issue can be affect to parents where parents are keen to evaluate their young kids driving styles to make sure they do not meet with accidents due to reckless driving.

1.2. Aim and Objectives

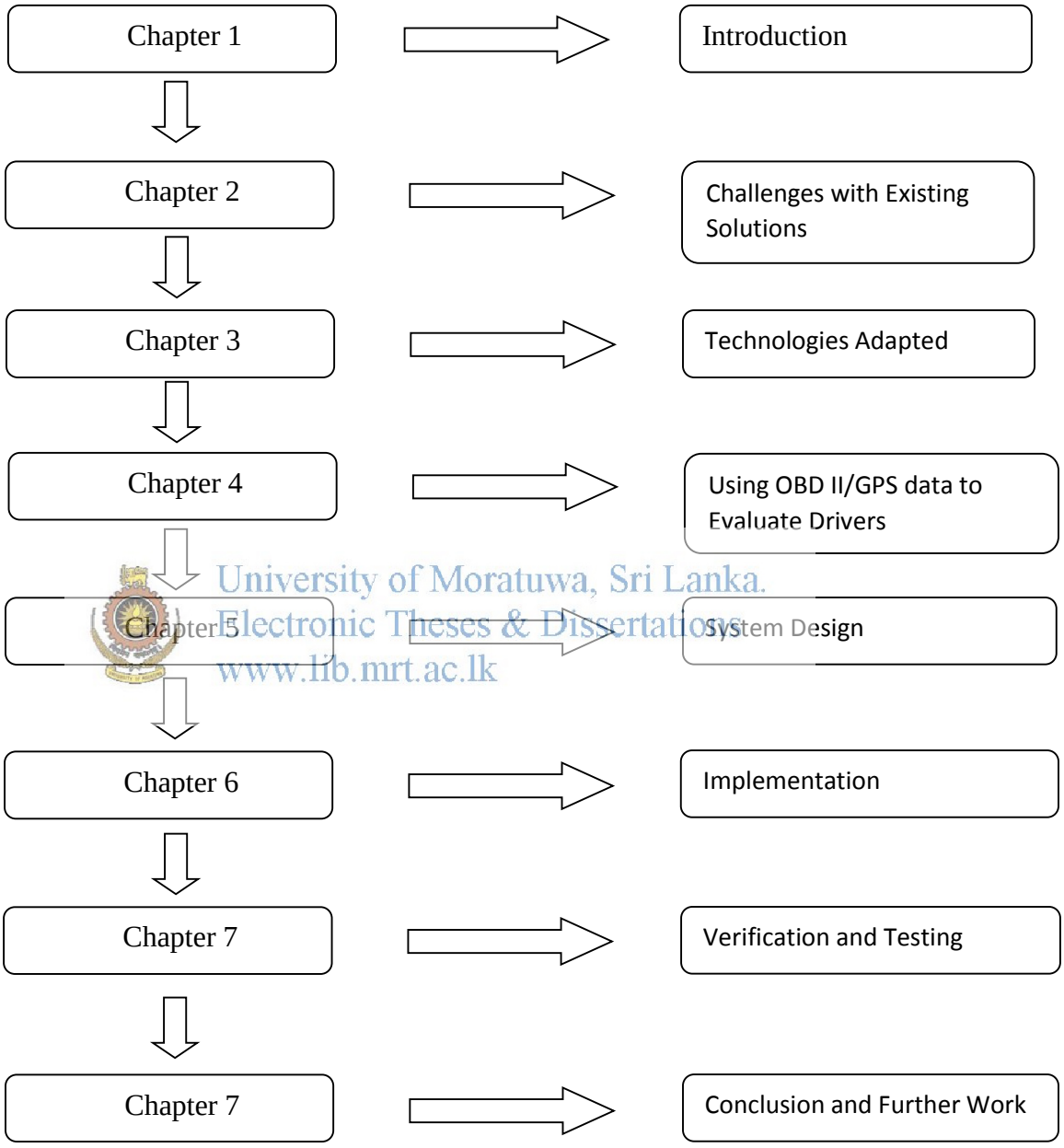
Aim of this project is to design and develop a system capable of monitoring driving patterns of automobile drivers with use of OBD and GPS technology. By evaluating drivers using the system can improve the quality and safety of the trip, resulting the taxi company attracts more customers. This project can reduce road accidents, fuel costs, vehicle maintenance cost and environmental pollution (increase of fuel efficiency results reduction of carbon emission). This massively benefits to the transport companies, society, country, and to the environment. Automobile Driver Evaluation System can be also used to evaluate performance of racing drivers and help them to improve their racing skills.

Study on OBD interface to retrieve data from vehicle Engine Control Unit effectively was one of the main objective needed to achieve in order to complete the project. Since there

are significant research and development attached with this part of the implementation, I developed a proof of concept (POC) type prototype for onboard part of the system before submitting the project proposal.

On-board System uses Vehicle OBD port to acquire real time vehicle performance indicators such as Engine RPM, Vehicle Speed Mass Airflow Rate. A GPS Dongle attached to the on-board system provides the GPS location which will be saved along with the data collected from the vehicle's ECU. OBD is an automotive term referring to a vehicle's self-diagnostic and reporting capability. OBD systems are capable of providing status of the various vehicle sub-systems in real time. The amount of diagnostic information available via OBD has varied widely since its introduction in the early 1980s' versions of on-board vehicle computers. Early versions of OBD would simply illuminate a MIL if a problem was detected but would not provide any information as to the nature of the problem. Modern OBD implementations use a standardized digital communications port to provide real-time data in addition to a standardized series of diagnostic trouble codes, or DTCs, which allow one to rapidly identify and remedy malfunctions within the vehicle. A Raspberry Pi computer runs the on-board application to get the data via ODB Port and GPS data via GPS dongle attached to it. Evaluator uses central system to view driver performances graphically in order to evaluate them. Central system presents the information to evaluator by processing data uploaded from onboard systems. By viewing the information evaluator will be able to identify when a particular driver exceeds the legal speed limit or when the driver over accelerates the vehicle. Alerting mechanism will be implemented in the central system to alert evaluators when drivers exceed certain thresholds of performance indicators. Central system will be developed as a web based system hence evaluators can immediately login and monitor the driving pattern upon receiving alerts from the system. Additionally system will allow evaluator to check any MIL codes saved in ECU.

1.3. Structure of the Report



Chapter 2

Challenges with Existing Solutions

2.1. Introduction

Currently there is no software in the market to track driving behaviors by using the data retrieved from Vehicles Electronic Control Unit (ECU). Most of the systems available in the market allow users to track vehicle position/speed using GPS. GPS only allows tracking the position and speed. When using data from ECU it is possible get all the required data from vehicle itself to determine the driving style. For an example Engine RPM (Revolution per Minute) is very useful data to evaluate driving style. Existing software in market used for fleet management in various transport services to track their vehicles.

Chapter 3

Technologies Adapted

3.1. Introduction

The main technologies used in the project are describes in this chapter. A brief description of the technology along with how the technology is used in the project mentioned here. Since the project consists of two sub systems which run in totally different platforms there are quite a bit of range of technologies adapted in to the project.

3.2. On-board diagnostics (OBD)

On-board diagnostics (OBD) [1] is an automotive term referring to a vehicle's self-diagnostic and reporting capability which has been improved over the last few decades. OBD systems give the vehicle owner or repair technician access to the status of the various vehicle sub-systems. The amount of diagnostic information available via OBD has varied widely since its introduction in the early 1980s' versions of on-board vehicle computers. Early versions of OBD would simply illuminate a malfunction indicator light or "idiot light" if a problem was detected but would not provide any information as to the nature of the problem. Modern OBD implementations use a standardized digital communications port to provide real-time data in addition to a standardized series of diagnostic trouble codes, or DTCs, which allow one to rapidly identify and remedy malfunctions within the vehicle. There were few versions of OBD has been introduced over the last few decades and the improvement introduced enables this project implementation. The latest version available is OBD II which is improved from OBD I in term of capability and standardization. OBD II has a diagnostic connecter interface which is again standardized (female 16-pin (2x8) J1962 connector). This connector port should be located within 2 feet from steering wheel as per the OBD II standard. Onboard part of the Automobile Driver Evaluation System connects to vehicles OBD II port and extract required data.

3.3. ELM327 OBDII USB Interface

Figure 1: ELM327 OBDII to USB device used in the project

This module consists of Integrated Circuit ELM327 which is designed to act as a bridge between OBD II and Universal Serial (RS232) Interface [2]. One end of this interface connects to vehicles OBD II connector port and other end connects to Raspberry Pi computer which is used to run the onboard part of the Automobile Driver Evaluation System. There are many benefits using this interface as a bridge between Raspberry Pi computer and Vehicle OBD II.

- ELM327 IC automatically search for protocols
- Universal serial (RS232) interface
- Fully configurable with AT commands
- Very low power consumption (In this project, it is powered by Raspberry Pi via USB)
- Cost per unit is very low

Almost all the vehicles manufactured today are required, by law, to provide a diagnostic interface which can be used for diagnostic test equipment to get data from the ECU. There are several standards for transferring data in these interfaces. However none of these standards are directly usable with PCs or Smart devices (mobile devices). ELM327 has been designed act as a bridge between OBD (On-Board Diagnostics) Ports and RS232 standard serial interface.

Apart from being able to detect and interpret many OBD protocols aromatically, ELM327 also capable of facilitate high speed communications. It uses very low power in sleep mode and it also support the bus and truck standard (J1939).

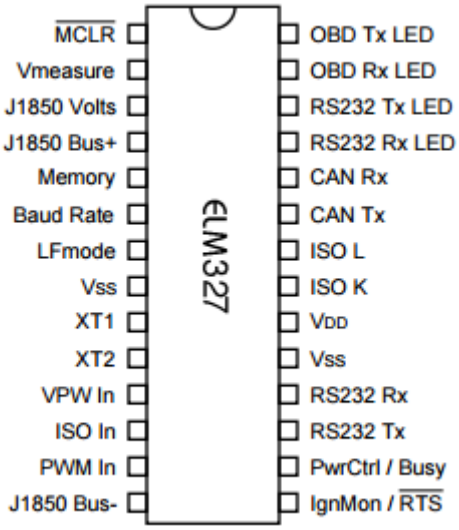


Figure 2: Connection Diagram PDIP and SOIC (top view)

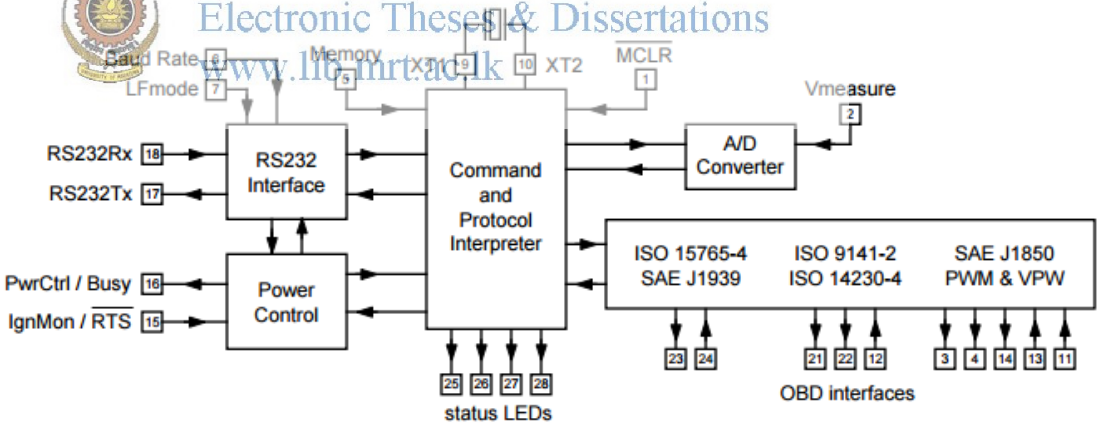


Figure 3: Block Diagram of ELM327

3.4. Communicating with ELM327

The ELM327 expects to communicate with a PC through an RS232 serial connection. Although modern computers do not usually provide a serial connection such as this, there are several ways in which a 'virtual serial port' can be created. The most common devices

are USB to RS232 adapters, but there are several others such as PC cards, Ethernet devices, or Bluetooth to serial adapters.

No matter how you physically connect to the ELM327, you will need a way to send and receive data. The simplest method is to use one of the many 'terminal' programs that are available (HyperTerminal, ZTerm, etc.), to allow typing the characters directly from your keyboard.

To use a terminal program, you will need to adjust several settings. First, ensure that your software is set to use the proper 'COM' port, and that you have chosen the proper data rate - this will be either 9600 baud (if pin 6 = 0V at power up), or 38400 baud (if PP 0C has not been changed). If you select the wrong 'COM' port, you will not be able to send or receive any data. If you select the wrong data rate, the information that you send and receive will be all garbled, and unreadable by you or the ELM327. Don't forget to also set your connection for 8 data bits, no parity bits, and 1 stop bit, and to set it for the proper 'line end' mode. All of the responses from the ELM327 are terminated with a single carriage return character and, optionally, a linefeed character (depending on your settings).

Properly connected and powered, the ELM327 will energize the four LED outputs in sequence (as a lamp test) and will then send the message:

ELM327 v2.1

>

In addition to identifying the version of this IC, receiving this string is a good way to confirm that the computer connections and terminal software settings are correct (however, at this point no communications have taken place with the vehicle, so the state of that connection is still unknown).

The '>' character that is shown on the second line is the ELM327's prompt character. It indicates that the device is in the idle state, ready to receive characters on the RS232 port. If you did not see the identification string, you might try resetting the IC again with the AT Z (reset) command. Simply type the letters A T and Z (spaces are optional), then press the return key:

>AT Z

That should cause the leds to flash again, and the identification string to be printed. If you see strange looking characters, then check your baud rate - you have likely set it incorrectly.

Characters sent from the computer can either be intended for the ELM327's internal use, or for reformatting and passing on to the vehicle. The ELM327 can quickly determine where the received characters are to be directed by monitoring the contents of the message. Commands that are intended for the ELM327's internal use will begin with the characters 'AT', while OBD commands for the vehicle are only allowed to contain the ASCII codes for hexadecimal digits (0 to 9 and A to F).

Whether it is an 'AT' type internal command or a hex string for the OBD bus, all messages to the ELM327 must be terminated with a carriage return character (hex '0D') before it will be acted upon. The one exception is when an incomplete string is sent and no carriage return appears. In this case, an internal timer will automatically abort the incomplete message after about 20 seconds, and the ELM327 will print a single question mark ('?') to show that the input was not understood (and was not acted upon).

Messages that are not understood by the ELM327 (syntax errors) will always be signaled by a single question mark. These include incomplete messages, incorrect AT commands, or invalid hexadecimal digit strings, but are not an indication of whether or not the message was understood by the vehicle. One must keep in mind that the ELM327 is a protocol interpreter that makes no attempt to assess the OBD messages that you send for validity – it only ensures that hexadecimal digits were received,
Combined into bytes, then sent out the OBD port, and it does not know if a message sent to the vehicle was in error.

While processing OBD commands, the ELM327 will continually monitor for either an active RTS input, or an RS232 character received. Either one will interrupt the IC, quickly returning control to the user, while possibly aborting any initiation, etc. that was in

progress. After generating a signal to interrupt the ELM327, software should always wait for either the prompt character ('>' or hex 3E), or a low level on the Busy output before beginning to send the next command.

Finally, it should be noted that the ELM327 is not case-sensitive, so the commands 'ATZ', 'atz', and 'AtZ' are all exactly the same to the ELM327. All commands may be entered as you prefer, as no one method is faster or better. The ELM327 also ignores space characters and all control characters (tab, etc.), so they can be inserted anywhere in the input if that improves readability.

One other feature of the ELM327 is the ability to repeat any command (AT or OBD) when only a single carriage return character is received. If you have sent a command (for example, 01 0C to obtain the rpm), you do not have to resend the entire command in order to resend the request to the vehicle - simply send a carriage return character, and the ELM327 will repeat the command for you. The memory buffer only remembers one command though, and there is no provision in the current ELM327 to provide storage for any more.

3.5. Raspberry Pi (Model B)

Figure 4: Raspberry Pi Model B

The Raspberry Pi is a credit-card sized computer which can be used in electronics projects, and for many of the things that your desktop PC does. Advantages of using a Raspberry Pi to run the onboard part of the project are,

- **Size:** It is very small, hence can be placed somewhere on the vehicle dash
- **Low power consumption:** Raspberry Pi is powered by Mini USB port. Without any peripherals attached, it uses less than 1W power.
- **SD card hard disk:** Raspberry Pi uses a SD card as its secondary memory, hence it is safer to use in a vehicle when compared to other mini computers which uses hard disk as the secondary memory.
- **Cheap:** Raspberry Pi computer is only costs USD 35.00 (without the SD card), hence cost for the implementation is low.

Specification of Raspberry Pi model B is as below,

- **Core Architecture:** ARM
- **Core Sub-Architecture:** ARM11
- **Features:** Credit card size video computer, HDMI, Ethernet & 2 USB ports, 512MB RAM
- **Kit Contents:** Evaluation Board
- **No. of Bits:** 32bit
- **Silicon Core Number:** BCM2835
- **Silicon Family Name:** BCM2xxx
- **Silicon Manufacturer:** Broadcom

3.6. Raspbian (Debian distribution for Raspberry Pi)

Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. However, Raspbian provides more than a pure OS: it comes with over 35,000 packages, pre-compiled software bundled in a nice format for easy installation on your Raspberry Pi.

The initial build of over 35,000 Raspbian packages, optimized for best performance on the Raspberry Pi, was completed in June of 2012. However, Raspbian is still under active development with an emphasis on improving the stability and performance of as many Debian packages as possible.

3.7. Perl Language

Perl language will be used to implement the onboard part of the system which will connected to vehicle's OBD II interface over ELM327 IC. Perl is a powerful scripting language which borrowed features from other programming languages such as C. There is a Perl module available as "Device::ELM327" to ease the communication with ELM327. This library will be used in this project to communicate with ELM327 IC to extract required data from ECU.

3.8. GPS

GPS data will be enriched to data gathered from the vehicle ECU to allow users view the exact location when the data was gathered. This has also helped to evaluate driver's styles in various road conditions. GPS data is crucial to details analysis of a driving behavior. This will need to be conducted manually as there will be various factors can affect to the driving pattern. Using GPS data along with the other data from the vehicle, it will be possible to analyze driving behavior in many road conditions. There were few options available to get GPS data to the system and I have chosen the quickest and cheapest solution for this research.

- Attaching a GPS Module to Raspberry Pi

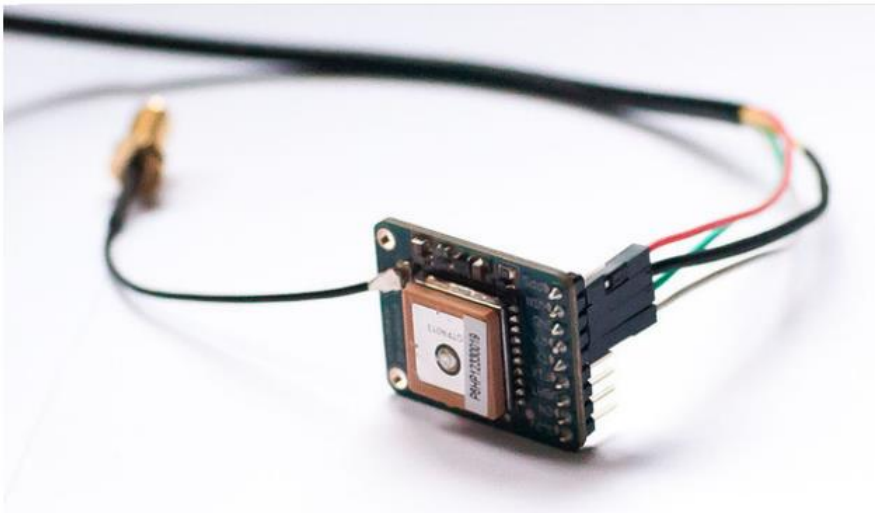


Figure 5: Adafruit Ultimate GPS on the Raspberry Pi

This module will be the ideal solution as this can be directly plugged into one of the USB ports on /Raspberry PI. This is cost effective and power consumption is very low hence it will get all the power from Raspberry Pi USB port. However due to the practical issues such as, custom clearance for GPS devices I had to choose an alternative approach to get GPS data.

- GPS Over BT

Getting GPS from GPS enabled Smart Phone (Android). I have used my Google Nexus 5 phone to send GPS data to Raspberry Pi via Bluetooth connection. In order to achieve this I used a USB micro Bluetooth dongle and “GPS over BT” tool which is available freely for Android phones. Implementation in the Raspberry Pi has not much difference in both options hence there will be only minor changes to implement the option 1 if required.

3.9. PHP/HTML Languages

PHP and HTML have been used to develop web portal to view processed data saved in database. Please note that the website has minimal functionality as this research is focused more on analyzing data taken from vehicle ECUs and test driver behavior.

3.10. Summary

It was pleasure to try many technologies and get them adapted to this research project. By adapting these technologies, I have gained a huge knowledge on various technologies which will be definitely helpful for the next research project.

Chapter 4

Approach to use ECU data to Evaluate Drivers

4.1. Introduction

My approach for successful completion of this project was, performing feasibility study for all the critical and risky functionalities at the beginning of the project. In this chapter I have described the approach I have chosen and the the main tasks I have done in order to achieve the completion of the project.

4.2. Research and Development

There was a significant amount of effort needed to be put on research and development part of this project. Firstly a Perl script for extracting real-time data such as Engine RPM, Speed, Mass air flow written which runs in the Raspberry Pi and it should extract data from vehicles OBD II interface. Without getting this done the total effort of the project could have wasted. Since this was a vital part of the project I have done a Proof of Concept (POC) prototype which extract data from ECU through ELM327 and save to CSV file. This has been presented to project supervisor alone with the project proposal.

4.3. Connecting Raspberry Pi to Internet in a moving vehicle

Once vehicle/GPS data collected it should be saved in the database in the centralized system. An internet connection is required for each unit used in the vehicle. It is possible to use a 3G dongle for this purpose and raspberry pi will be configured to connect the dongle to internet at boot-up to make sure internet connection is automatically enabled at the startup. Since Raspberry Pi has only two USB ports and 3G dongle uses considerable amount of power, a powered USB hub cab be used.

If the real time data upload is not required and owner wants to reduce the maintenance cost of the solution, Wifi dongle can be used instead of 3G internet dongle. In this case vehicle data will be saved to Raspberry Pi SD card and will be automatically uploaded to the central

system once the vehicle comes back to the yard. In order to achieve this Wifi network should be available in the vehicle yard. However this is a more cost effective method; for clients who do not need real-time data access. I have implemented this for the research to simplify the implementation as this option is cost effective and more practical. Since micro Wifi dongles which use very low power are available at a cheap price in the market this would be the ideal solution for a company with huge fleet of vehicles.

4.4. Centralized System with Web Interface

Web GUI is implemented for users to view data gathered from vehicles. Graphical representation of collected data can be viewed through this interface. Web Interface connects to database where all the data is saved in order to full fill user's requests. I have developed the web application simple as possible as the core goal of the project was not building the best eye catching web interface with all the fancy features. XAMP server is used to host the web interface as it comes with all the required tools and technologies to host this web application.

4.5. Many Sub Systems

Dividing the implementation into many sub systems has paid off while to project was progress. Since all these sub systems are independent to each other, I was able to prioritize development of sub systems according to the external issues arise in the middle of the project. It was also helped a lot while I was testing the system as I was able to test sub systems separately without depending on the other sub systems.

4.6. Summary

The approach I have followed has helped a lot for the completion of this project. Identifying risks in the beginning and breaking the system into multiple sub systems made the successful outcome of this research project.

Chapter 5

Analysis and System Design

5.1. Introduction

This chapter includes system design model and other relevant object and data diagrams graphically showing how the system stores and communicate with each component of the system. Analysing the problem was bit hard to perform as there are not many solutions which address the problem which is address by this research project.

To analyse further more into subject I used the methods PEST analysis. I also have added subjective weighting for each of them depends on their impact and also indicate that impact is positive or negative, if it's a positive impact indicate by plus (+) sign and if it's a negative impact indicate by minus (-) sign.

Political	**Economical**
Environmental friendliness and Energy efficacies (+10) Reducing reckless driving (+7) Legal aspects of monitoring public transport such as Taxis (-5) Overall subjecting weighting (+12)	Fuel Economy (+10) General wear and tear (+10) More customer attraction (+5) Reducing shipment damages (+3) Extra cost for each vehicle (-5) Extra cost for system maintenance (-4) Overall subjecting weighting (+12)
Social Reduce road accidents (+8) Privacy issues (-4) Overall subjecting weighting (+4)	**Technological** Reliable technology to get information for the company (+5) Overall subjecting weighting (+5)

5.2. System Component Integration

Figure 6 shows how the components of the system integrated each other in high level. This shows how the on-board systems collect data from GPS/OBD and upload it to centralized system running on the web server over the internet.

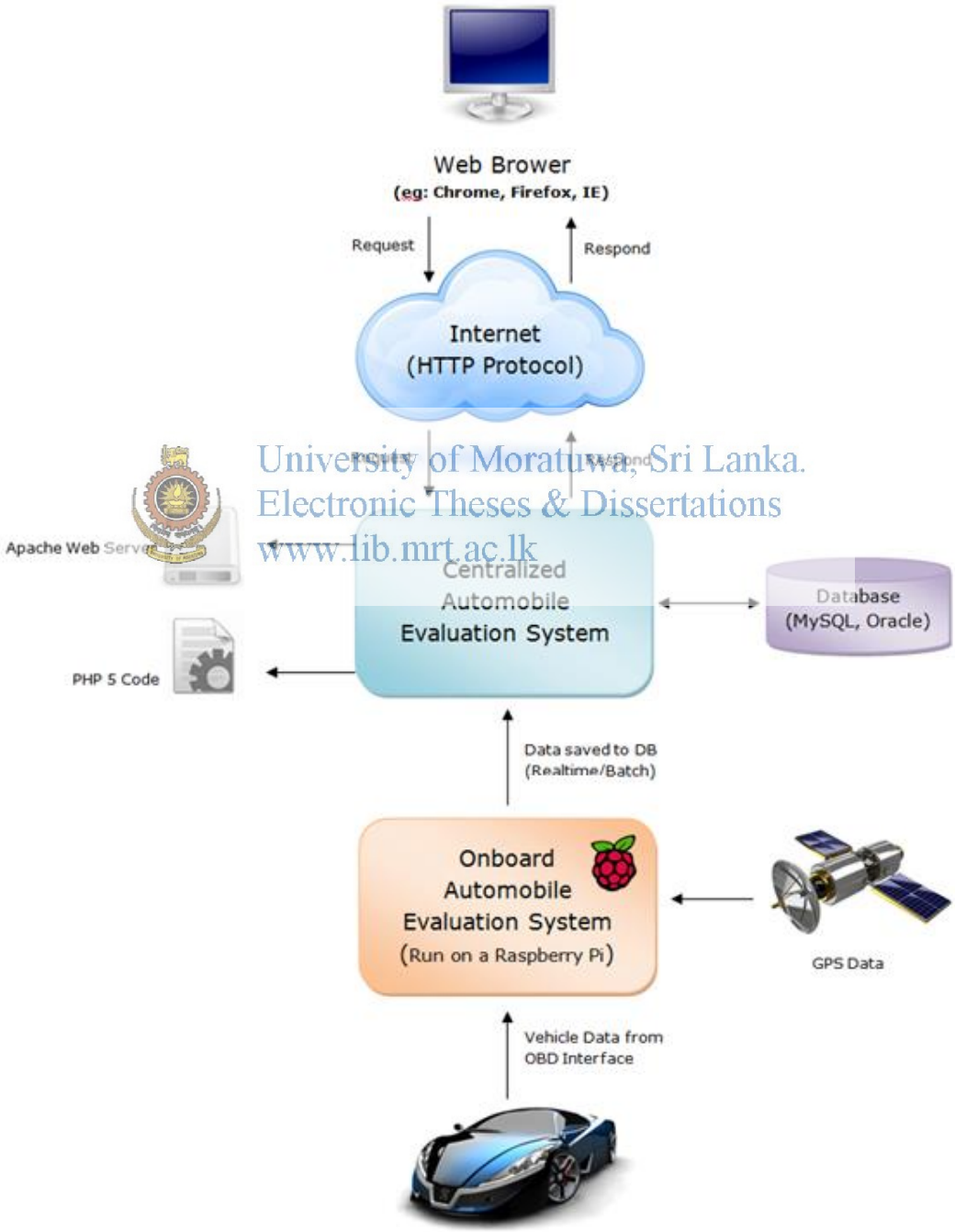


Figure 6: Top Level Architecture of the Proposed System

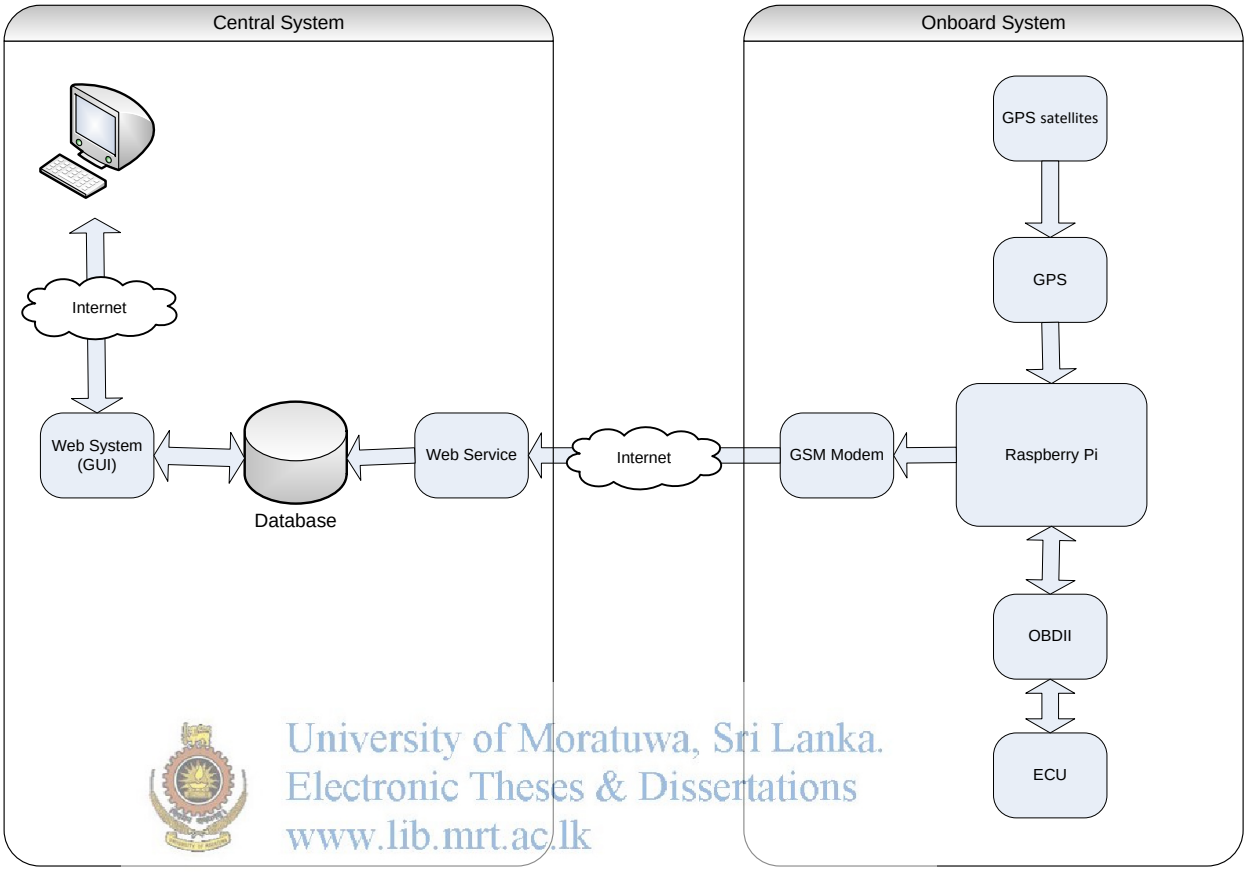


Figure 7: Architecture Design Diagram

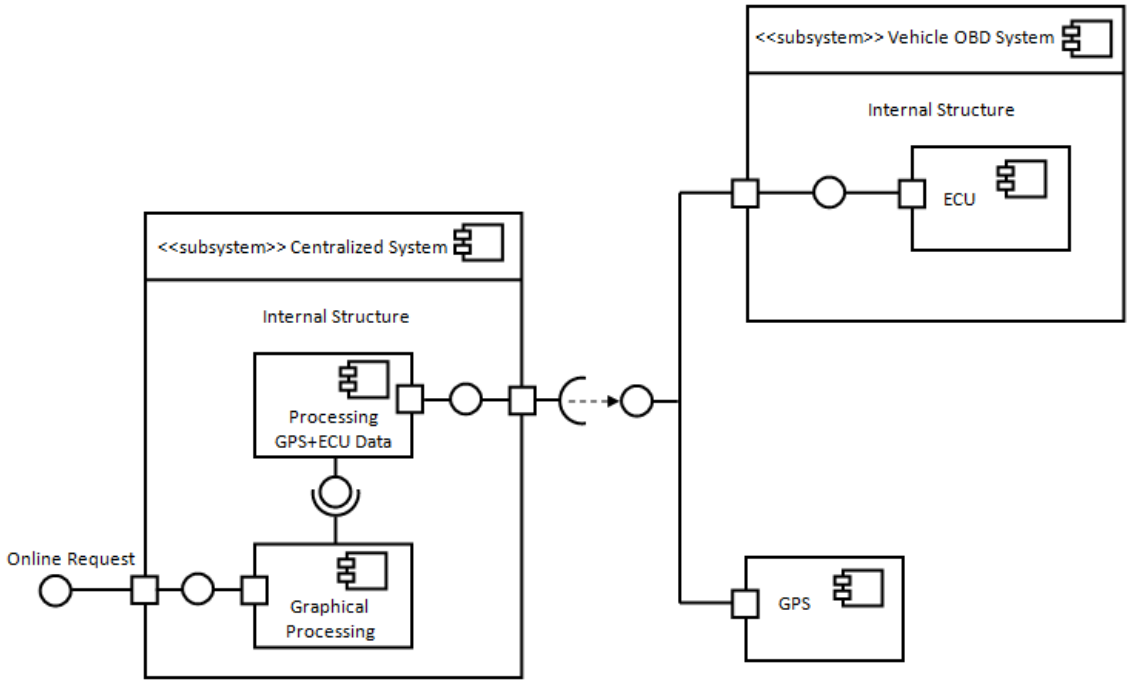


Figure 8: Component Diagram

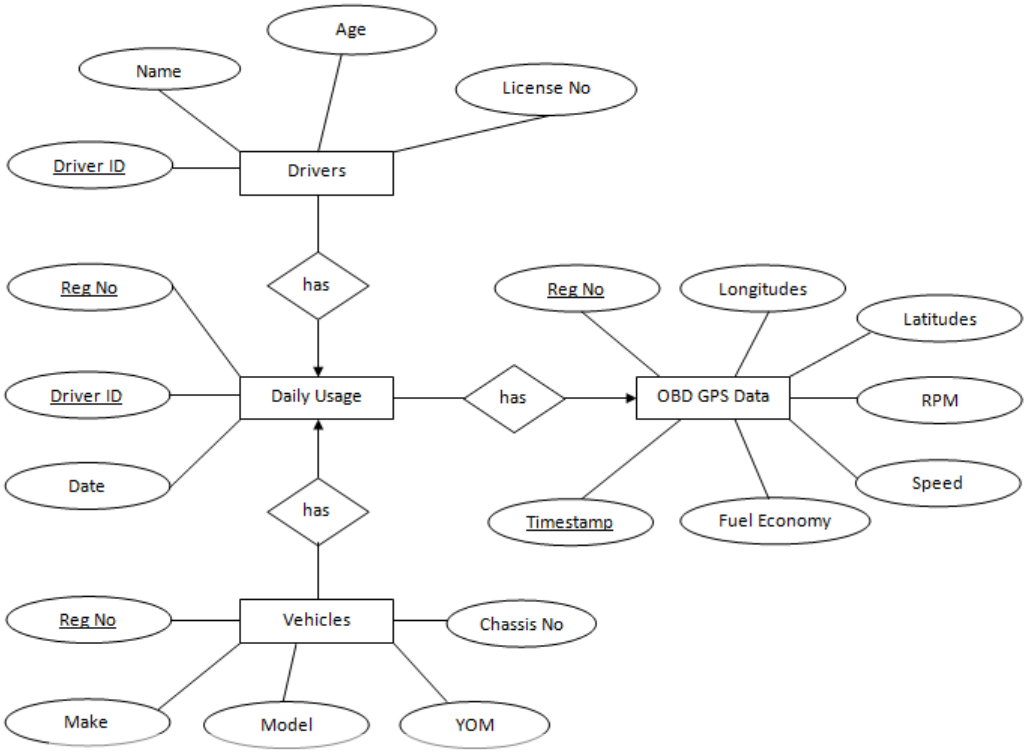


Figure 9: Entity Relationship Diagram

5.3. Summary

This system will be a reliable system to get real picture of driving styles of various drivers to users PC without rely on any other external party. This can be used as proof of how exactly driver drives car and compare driving behavior of various drivers.

Chapter 6

Implementation

6.1. Introduction

This chapter describes the implementation details of Automobile Driver Evaluation System. This consists of implementation details of perl scripts running in Raspberry Pi to fetch date from ECU, how the data send to centralized system and how the web application is implemented for users to evaluate drivers.

6.2. Get ECU data from OBD II interface

This was the most challenging part of the project which could lead to a project failure. As I have identified this area as a high risk area when the project proposal was submitted, I have done a "Proof of Concept" (POC) type prototype to make sure extracting the real-time vehicle data is feasible with technologies specified in the project proposal. After some research done in the internet, I have been able to gather knowledge about extracting data from vehicles' ECU.

Perl script "obd.pl" has been written to get data from OBD II interface to raspberry Pi running on the vehicle. Below is the partial code extract from the "obd.pl" script. This particular script calculate instantaneous fuel economy by using the Air Mass Flow sensor and Vehicle Speed Sensor values. In newer models of the vehicles driver can see the instantaneous fuel economy on a digital display. However this value is not published by the ECU. Instrument panel calculate the fuel efficiency same way I use to calculate the instantaneous fuel efficiency in obd.pl script. Logic behind calculating fuel economy is as follows,

All the vehicle manufactures are bound to use ratio of Air Mass to Fuel Mass as 14.7:1. This uses when fuel mixes with air. The purpose of Air Mass Flow sensor is to calculate the mixing air and let the ECU to decide the amount of fuel needed to be release for mixing. By getting the instantaneous value for Air Mass Flow and instantaneous value of Vehicle Speed Sensor, we can calculate instantaneous fuel efficiency.

```
#!/usr/bin/perl

use strict;
use warnings;
use Device::ELM327;
use Time::HiRes qw(usleep nanosleep);
use Fcntl ':flock';

sleep(30);

my $file_path = '/home/pi/perlobd';
my $log_file = $file_path . "/obd_logger.csv";
&log_rotate;
my $obd = Device::ELM327->new('/dev/ttyUSB0');

sub main()
{

    my $n = 0;
    my $rpm = 0;
    my $speed = 0;
    my $maf = 0;
    my $kml = 0;
    my $time = 0;
    $rpm = &process_read("Engine RPM");
    if (&trim($rpm) > 0)
    {
        &log_writer("RPM","Speed","Air Flow Rate","Fuel Economy");
        while(&trim($rpm) > 0)
        {
            usleep(600000);
            $rpm = &process_read("Engine RPM");

                usleep(200000);
```


Figure 10: A code segment from obd.pl

6.3. Get GPS data to Raspberry Pi

I have implemented getting GPS data to raspberry Pi independent to the extraction of OBD data. This allows any other program to get real-time GPS data as it is available in continuously update file in Raspbian OS. This method is used in Linux for publishing many other statistics such as memory, server load average, etc. "obd.pl" script refer this GPS file and get the value for every snapshot it write to the .csv file. A code segment of "read_gps.pl" is as below.

```
#!/usr/bin/perl
use File::Tail;
use Fcntl ':flock';

my $gps_read_file = '/home/pi/perlobd/gpsoverbt.log';
my $gps_write_file = '/home/pi/perlobd/gps';
my @aGPS_DATA;
my $lat;
my $lat_mins;
my $lat_side;
my $lon;
my $lon_mins;
my $lon_side;
my $altitude;


sub main()
{
    system ("rfcomm release 0");
    sleep 5;
    system ("rfcomm connect 0 &");
    sleep 10;
    system ("cat /dev/rfcomm0 > /home/pi/perlobd/gpsoverbt.log &");

    $file=File::Tail->new(name=>$gps_read_file,tail=>13);
    while (defined($line=$file->read))
    {
        if ($line =~ m/^\$GPGGA/)
        {
            @aGPS_DATA = process_comma_delimeter(&trim($line));
            $lat = $aGPS_DATA[2];
            $lat_side = $aGPS_DATA[3];
            $lon = $aGPS_DATA[4];
            $lon_side = $aGPS_DATA[5];
```


Figure 11: A code segment from read_gps.pl

6.4. Upload data to Centralized System

Perl script “pitodb.pl” has been written to extract data from CSV files generated by “obd.pl” and upload to the centralized system. CSV files generated by “obd.pl” is renamed with timestamp for each driving cycle, however “pitodb.pl” is written to access them in the same order. A code segment extracted from “pitodb.pl” is as below.

```
1  #!/usr/bin/perl
2
3  use warnings;
4  use Fcntl ':flock';
5
6
7  my $file_path = '/home/pi/perlobd/';
8  my $log_file = $file_path . &conf_reader("LOG_FILE");
9  my $data_file = $file_path . &conf_reader("DATA_FILE");
10 my $reg_num = &conf_reader("REG_NUM");
11 my $server = &conf_reader("SERVER");
12 my $port = &conf_reader("PORT");
13 my $last_timestamp = 0;
14 &log_rotate;
15
16
17 sub main()
18 {
19     my @aData;
20     my $time;
21     my $rpm;
22     my $speed;
23     my $maf;
24     my $kml;
25     my $gps;
26     my $altitude;
27     my @aFILE_LIST;
28     my @aSORTED_FILE_LIST;
29     my $insert_status;
30
31     opendir(DIR, $file_path) or die $!;
32     while (my $file = readdir(DIR))
```


Figure 12: A code segment from pitodb.pl

6.5. Present Driving Behavior Graphically

Web interface allows users to view Fuel Economy, RPM and Vehicle Speed Graphs for any driver for a selected date period. Some straight forward verifications are also check by the system and presented to user as a tabular report. This consists, Top speed driven, number of instances of breaching recommended speed value for each vehicle, highest RPM recorded, Recommended RPM breaching instances, and average Fuel Economy for selected date range. In addition to these graphs and statistics, system also generate a KML file which can be opened with Google Earth to study driving behaviors more closely with GPS data gathered. It is possible to draw a graph on the route which the vehicle has been driven. I use altitude to represent the value of Fuel Economy, RPM and Vehicle Speed for three paths in KML file.

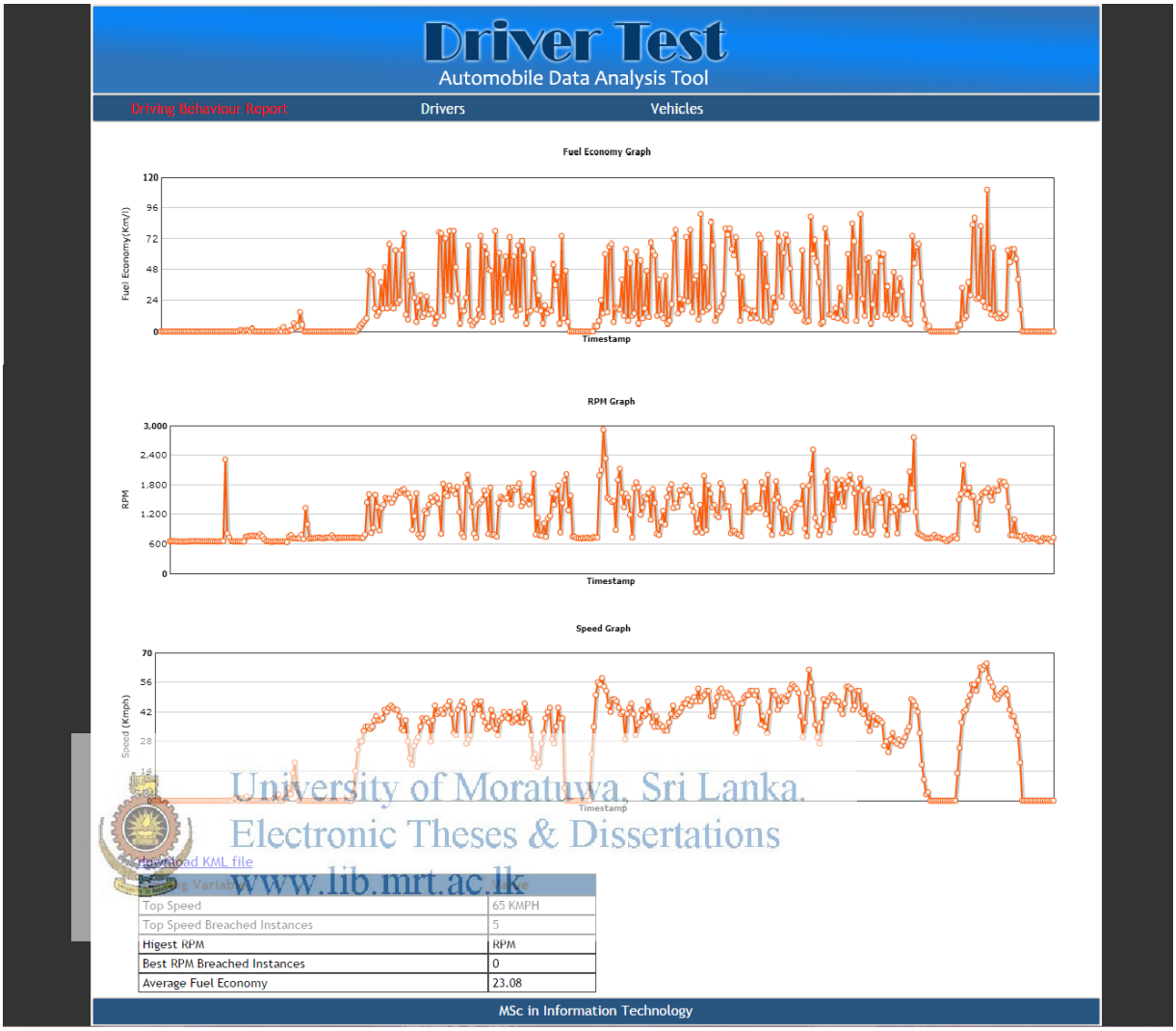


Figure 13: Driver Evaluation Page

Upon clicking in “download KML file” this page generates the KML file. Given that Google Earth is installed in the PC, user can open the KML file with Google Earth and perform analysis on the driving pattern more closely.

Figure 14: Trip from Independence Arcade to "Pannipitiya"

Figure 15: Zoomed-in picture of Fuel Efficiency values at "Thunmulla Junction".

Figure 16: Zoomed-in picture of RPM values at “Thunmulla Junction”.

Figure 17: Zoomed-in picture of Speed values at “Thunmulla Junction”.

Figure 18: Zoomed-in picture when Fuel Efficiency, RPM and Speed all selected

6.6. Summary

It is very interesting to analyze the graph closely and see how fuel efficiency drastically change with the vehicle RPM. The process is assisted further by analyzing the graphs in Google Earth as user can see the geographical factors which can explain driver's actions. For an example if the vehicle is driven uphill high RPM is acceptable.

Chapter 7

Verification and Testing

7.1. Introduction

The goal of all testing is to demonstrate that the program under inspection for bugs. Testing can never prove that a code is error free but no matter what testing technique is being used testing should be an activity considered thoroughly.

Since the system has few sub systems I have tested each sub systems separately when they were being developed. This was kind of a unit testing. As part of this project I had to rely on black box and white box testing as the application testing method. While developing this project mostly I had to run on debug mode to find out the issues which exist with the system.

7.2. Testing Methods

It is crucial to select the best method of testing and there are number of testing methods can be used to validate the overall functional testing and non-functional tests such as performance testing.

7.3. Functional Testing

Unit testing has been carried out after development of each subsystems as full system testing is more complex for this particular research project. Initially the on-board system testing was carried out and taken data to carry out centralized system.

7.4. Non-Functional Testing

Non-functional testing was carried out for on-board system as the on-board system is powered by a Raspberry Pi Model B where increasing the server resources were not an option for the Raspberry Pi. Model B was the highest configuration Raspberry Pi available when the project was implemented. Recently Raspberry Pi 2 was release with much higher hardware configuration. I wanted to test whether 512 MB of memory and single core CPU device can handle all the load from on-board sub system. I also monitored the memory usage for period of time to make sure there are no memory utilization issues which can crash the program after some time. Shutdown functionality of Raspberry Pi after ignition off of the vehicle was also tested.

7.5. Summary

Due to the fact that system is consist of many sub systems. I could cover lot of areas in unit testing. However a full system test was also carried out after integration of all the sub systems. It was not possible to cover all the functionalities in full system testing which were already covered in unit testing due to various constrains. However I have found few bugs in full system testing and they have been resolved and retested.

Chapter 8

Conclusion and Further Work

8.1. Introduction

I had to face numerous obstacles while doing this research project. On the other hand I also noticed that there are some possible new developments which can be included to the scope of the project. This chapter is for the conclusion of the project after the completion. This also specifies the nice to have features which can be added in future releases.

8.2. Limitations of the project

There are some limitations with the project due to the limitations and the behavior of the technologies I used in this project. They can be listed as,

Automatic shutdown of Raspberry PI after Ignition Off:

Raspberry Pi is a computer runs Debian Linus hens it should be properly shut down to prevent any data losses and file corruptions. Since I used the power outlet on the vehicle dashboard, whenever the driver switch off the vehicle, ungraceful shutdown occurs in Raspberry Pi. To prevent that I should use the fulltime dedicated power from battery where vehicle audio video players are connected to retain configurations and date even after ignition off. However, using this wire will also bring another issues. Since Raspberry PI uses some power even after shutting down, it can draw power from battery and if the vehicle stays not started for longer time. Vehicle battery can be drained by Raspberry Pi. I have implemented something simple to prevent this, but drivers will have to follow this whenever they switch off the vehicle. I have added a functionality to the system to shut down the Raspberry Pi whenever it notice the RPM is 0 after any positive value. This to work effectively, drivers should wait for approximately one minute after ignition off to complete switch off for taking the key out.

Few seconds gap between data sets from ECU:

I use the query mode of the OBD II protocol where the on-board system query for RPM, Speed and Mass Air Flow sequentially. There is a throttling control mechanism setup in ECU to control the query load coming through OBD interface. There is a 100 ms delay added for all the queries. This has been done to make sure ECU processing power is mainly used for vehicle tasks not for queries coming from outside. However there is another way of getting all the data simultaneously without adding an additional burden to ECU. It is possible to subscribe to multiple PIDs (RPM, Speed, Mass Air Flow, Engine Load, etc.) and OBD II device will get almost continuous data stream which will be more accurate when calculating fuel efficiency. However the ELM327 chip has very limited buffer size. I have tried this way of getting data but it can only handle one PID without getting the buffer filed in the middle. There are alternative chips available with larger buffer sizes, however devices made with those are expensive compared to ELM327

8.3. Next version of the system

As I could not find any other projects to monitor taxi/rent-a-car drivers driving styles using ECU data real time, it is not possible to compare this solution with other solutions. However there are many new features can be added if a new version of the system is decided to develop. Some of them are,

- System warns drivers when the system notices reckless driving
- By maintaining the speed limits for all the roads, system can inform the driver whenever driver exceed the speed limit for a particular section of the road. However this functionality is already available in GPS navigation equipments.
- By using the camera module developed for raspberry pi, the same system can use as a dash camera to start recording whenever system notices reckless driving.

8.4. Obstacles

It was not easy tasks to get everything working as expected due to the amount of research and development attached to this project. In some situations I had to try many things to get what was expected. There were some issues with the technologies I used where I had to find alternative solutions. For an example I was using an OBDII to Bluetooth device to get date from the vehicle initially. I have paired the device with Raspberry Pi successfully but every time I make the connections Raspbian OS frozen. Later I found that there is a bug with some versions of Bluetooth devices causing Raspbian OS to freeze. I had to order an OBDII to USB device to get over this issue and the delivery of the device took one month which could have delayed the project. Since the development of web application is totally independent from development of on-board system. I have prioritize the web application development until I get the new device from USA.

8.5. Conclusion

Successful completion of this research project showed very positives points towards the improvement of driving styles of the drivers. When I was analyzing the real data collected from the vehicles for testing purposes, I notice that the changing the driving style can save both money and the environment. As this can be used to improve the driving skills of racing drivers, the next trend should be a rally race where not the fastest driver wins but the most fuel economical driver wins. Racing team engineers should be asked to build the most fuel economical car instead of fastest car. The effort I have put on this research project has paid off by the final result and it was also fun to play with vehicle ECU and Raspberry Pi. By the time of the completion of the project a newer version of Raspberry Pi with higher configuration has been releases, yet I could not find a software product which can evaluate the motor vehicle drivers to save fuel and maintenance cost. Technology is keep evolving and still there are areas which can be benefited a lot using new technologies.

References

[1] http://en.wikipedia.org/w/index.php?title=On-board_diagnostics&oldid=622067427

[2] http://elmelectronics.com/DSheets/ELM327DSH.pdf

[2] https://developers.google.com/kml/documentation/kml_tut

[3] http://www.windmill.co.uk/obdii.pdf

[4] https://www.apachefriends.org/index.html

[5] http://dev.mysql.com/doc/

[6] http://www.raspberrypi.org/

Appendix A

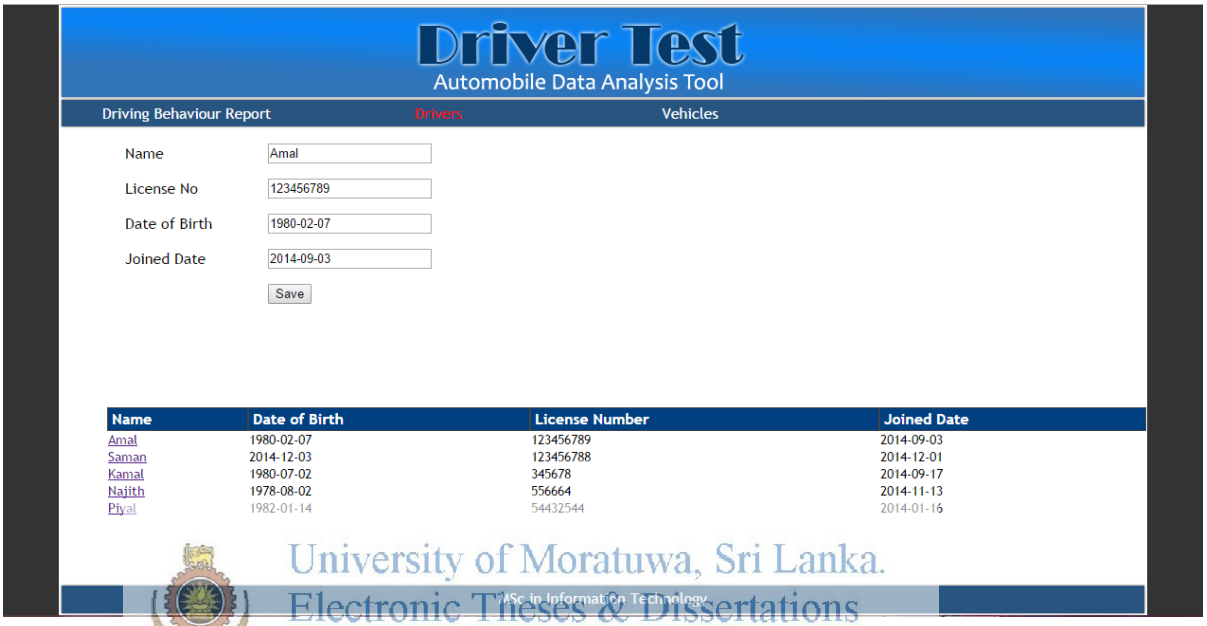
Automobile Driver Evaluation System

User Manual

March 2015

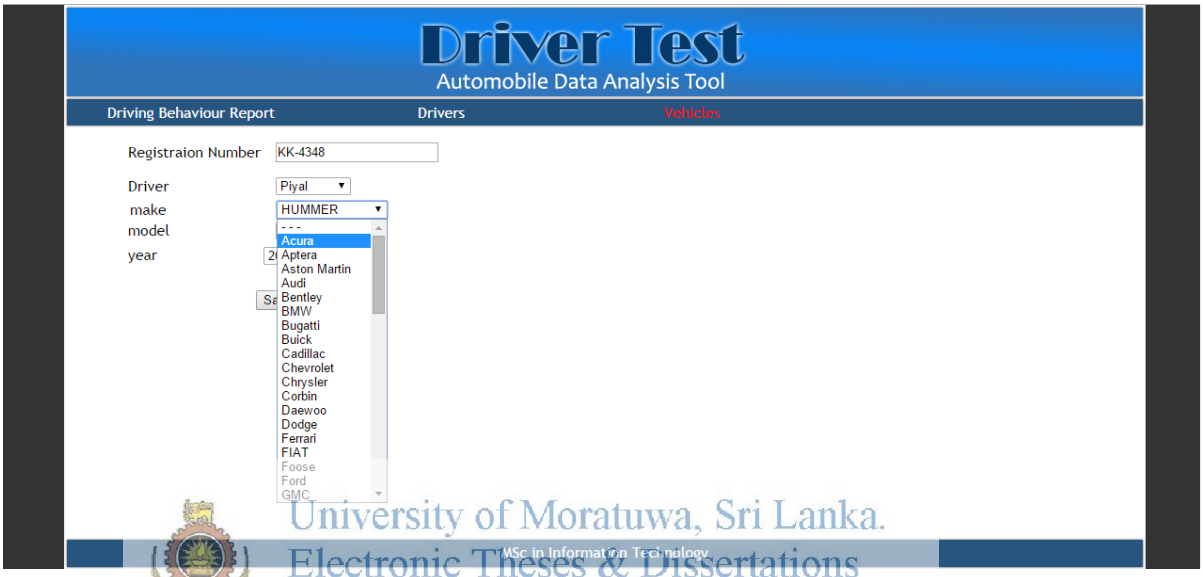
Add/Update Drivers

Users can be added and updated from Drivers page. Basic information of drivers saved to the system within this page.



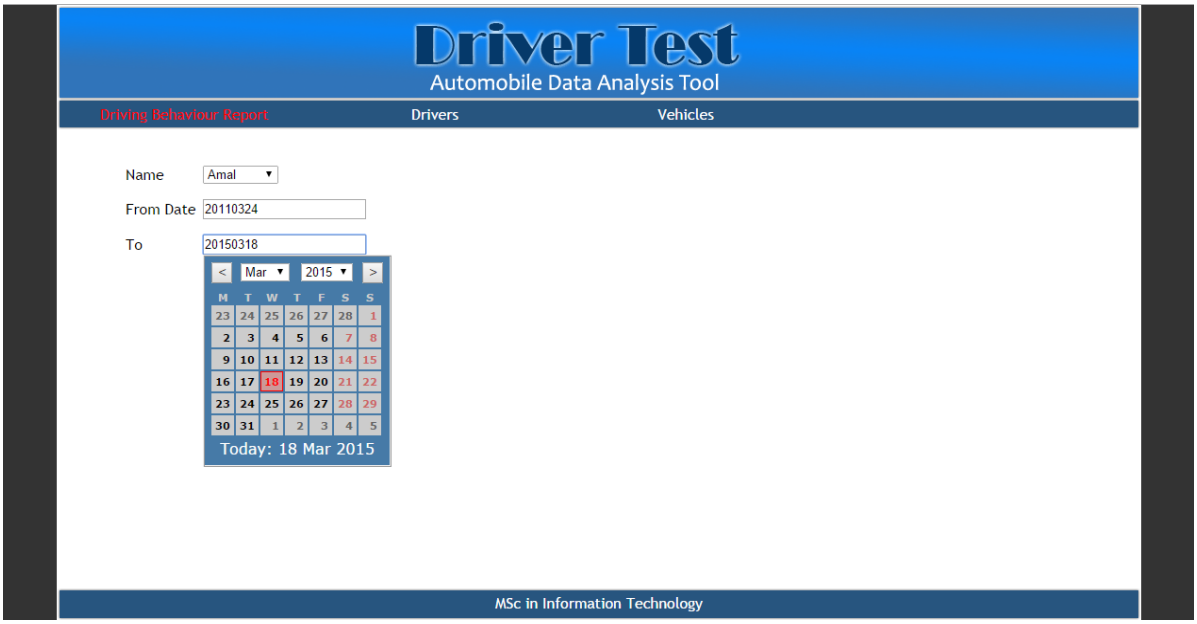
Add Vehicles

New vehicles can be added to the system with Vehicles page. There are 4188 models of vehicles are pre-loaded to database for users to select in this page. This covers almost all the models manufactured on or after year 2000.

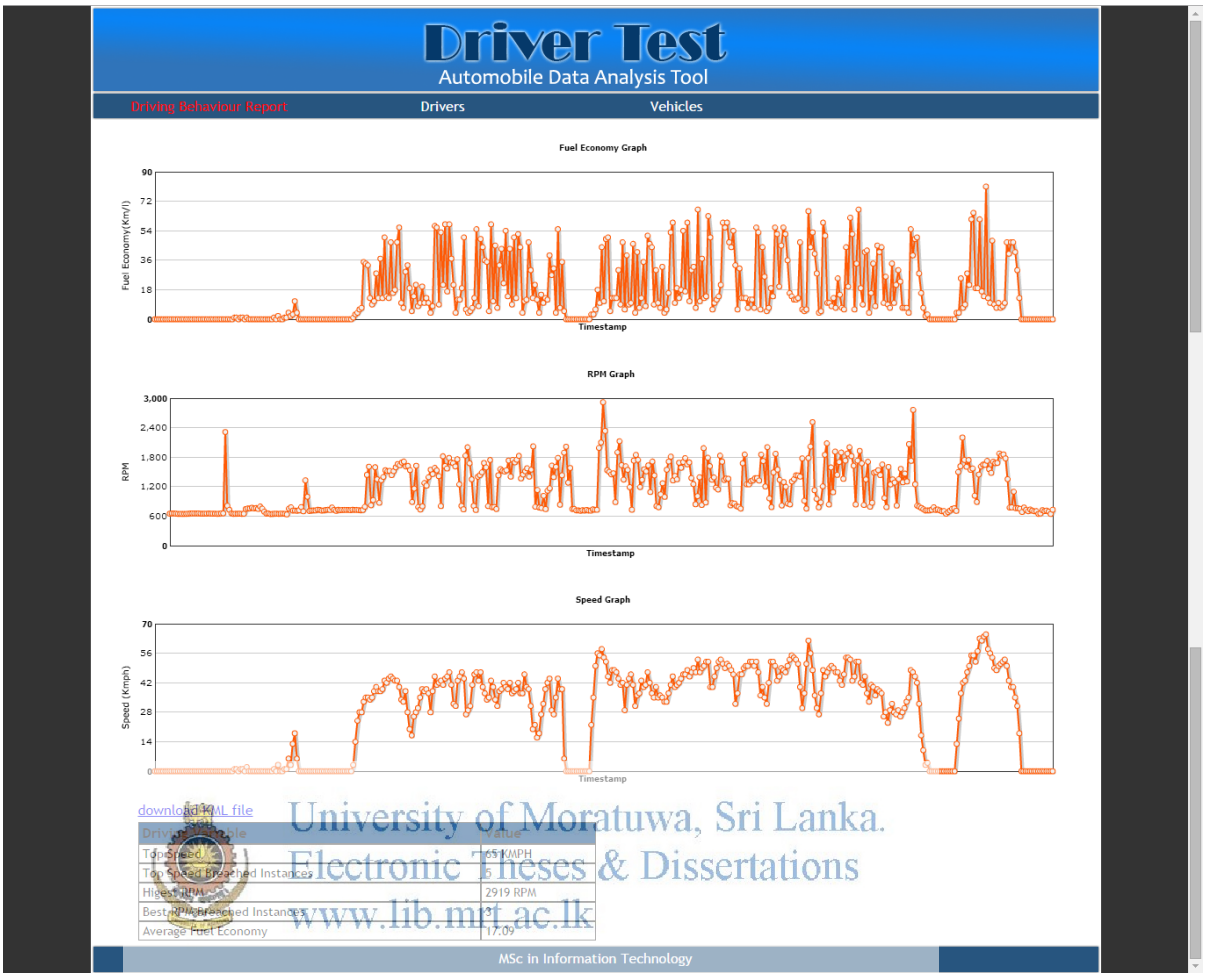


Analyze Driving Styles

User can generate report for a particular driver to analyze driving style in this page. Select driver name from "Name" drop down list and select two days to generate report. Data available for selected date period will used for the report.



Once select all the data click “Go” to continue. This will generate the report similar to below,



This report can be used to analyze how vehicle RPM and speed affect the fuel economy. Table printed bottom of the page shows some statistics for the selected date period.

Top Speed: The top speed which driver has driven the vehicle within date limit.

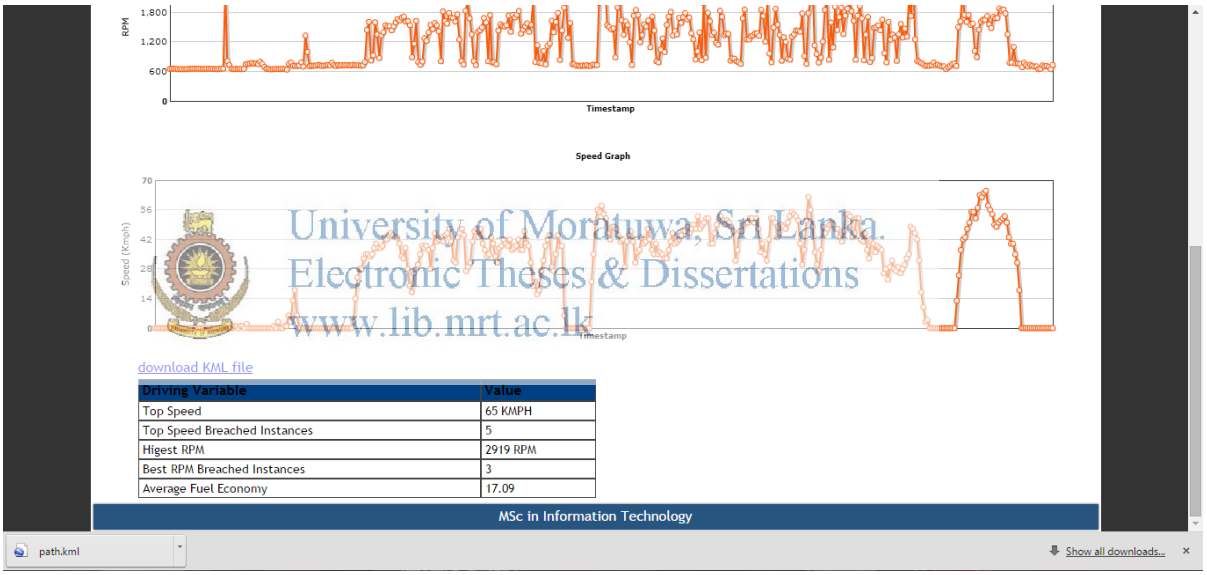
Top Speed Breached Instances: For each and every vehicle model there is a recommended speed which vehicle achieves its best fuel efficiency. This is setup in the system and value shown here is the number of instances the driver breached this recommended value. However due to the restrictions of highway rules, drivers are bound to maintain a minimum speed requirement. This can be validated when analyzing the KML file in Google earth.

Highest RPM: The top RPM value recorded by the driver for the selected date period

Best RPM Breached Instances: There is an optimum value for RPM for each vehicle model to achieve its best fuel efficiency. This is also set in the system and drivers are informed about this. Not like vehicle speed there are no restrictions setup in low to maintain minimum RPM by law. This value denotes how many times the driver has exceeded the recommended RPM value.

Average Fuel Economy: Average value of all the instantaneous fuel economy values saved to the system is shown here.

User can also click "download KML file" link and this will generate a KML file and downloaded to PC which can be opened in Google Earth.



Once opened with Google Earth, user can visible three paths Fuel Efficiency, RPM and Speed under Temporary Places. By selecting one or more paths user can view the Efficiency, RPM and Speed value graphs alone the vehicle traveled path. User can use functionalities in Google Earth such as zooming, move around and street view to analyze what was the speed, RPM and the fuel efficiency of the any sections or a special places such as bends of the traveled path.