Contents

[1 Abstract 2](#_Toc402274973)

[2 Idea 3](#_Toc402274974)

[3 Main Components 3](#_Toc402274975)

[3.1 OBDII 3](#_Toc402274976)

[3.2 Raspberry Pi 3](#_Toc402274977)

[3.2.1 Raspberry Pi Digital Display Monitor 4](#_Toc402274978)

[3.3 GPS Receiver 4](#_Toc402274979)

[3.4 Battery 4](#_Toc402274980)

[4 Core Functionality 5](#_Toc402274981)

[4.1 Real-time information 5](#_Toc402274982)

[4.2 GPS Record Journeys 5](#_Toc402274983)

[4.3 Dashboard Online Service 6](#_Toc402274984)

[4.4 Economic Driving 6](#_Toc402274985)

[4.5 Compared Results 6](#_Toc402274986)

[5 Software Design 7](#_Toc402274987)

[5.1 Web Application Backend 7](#_Toc402274988)

[5.2 GPS & OBD Extractor 7](#_Toc402274989)

[5.3 File Format 7](#_Toc402274990)

[5.4 Render Data to Screen 8](#_Toc402274991)

[5.5 Development Cycle 8](#_Toc402274992)

[6 References 9](#_Toc402274993)

Table of Figures

[Figure 1: ELM327 - OBDII device 3](#_Toc402274994)

[Figure 2: Raspberry Pi b+ 4](#_Toc402274995)

[Figure 3: Example route colouring - Nike Running App 5](#_Toc402274996)

[Figure 4: JSON file structure 8](#_Toc402274997)

# Abstract

This document has been produced as part of Phase 1 of ACE1 for CS413 – Embedded Systems. It describes the idea and assess the capabilities and functionality of the device we have decided to produce. It identifies all of the components we require to build the device and the cost of each individual component. In addition, the document includes a detailed hardware and software design of the device. Finally, it outlines a plan of which team member(s) will take on each task and the time assigned to each task involved in building the hardware and software.

# Idea

We have decided to create a device which we have named Progressive Improvement in Motoring Proficiency for Economic Driving (PIMPED). PIMPED will allow a user to connect to their cars on-board system and extract real-time data while they are driving. Most extracted data will be stored and the rest of the data - real-time data such as current miles per hour, current miles per gallon - will be displayed on a digital screen viewable to the user. A full description of the data which will be displayed to the user while driving is described in [LINK]. When the user completes their journey they will be able to remove the PIMPED from their car and upload the data extracted from their journey onto a web server. Each user of PIMPED will be able to create their own unique user account where they can upload their journey data to and have access to all the journey data they have ever uploaded to the web server. This allows each user of PIMPED to track their driving over time. The main benefit of this is that a user will be able to assess with ease, whether they are driving more efficiently.

# Main Components

## OBDII

An ELM327 device will be an essential component of the PIMPED device. On-board Diagnostics II (OBDII) will be used to extract any data from the on-board system of the car that the device is running on. OBDII is a that can connect to most cars produced after 1996 and can access data from the Engine Control Unit (ECU). [REF] shows the OBDII device that we will be using.

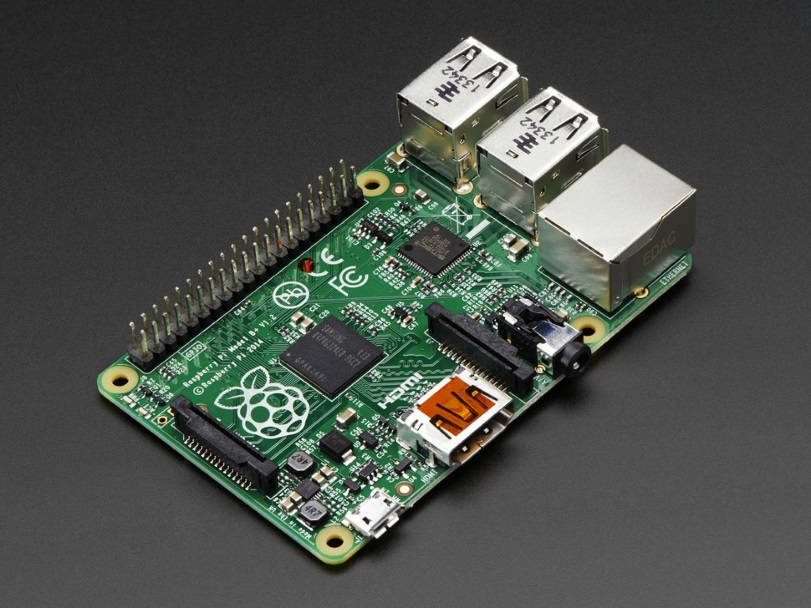


Figure : ELM327 - OBDII device

The full range of the data which is extractable by OBDII can be found on Wikipedia [REF].

## Raspberry Pi

A Raspberry Pi b+ will be a key component of the PIMPED device. A Raspberry Pi is a small, low-cost computer which has the standard capabilities of most desktop PC’s.



GPIO Headers

512 MB RAM

HDMI port

Ethernet port

USB ports

Figure : Raspberry Pi b+

Data will be passed back from the OBDII device to the Raspberry Pi. Some of this data, all of which is listed in section [LINK] will be displayed on a Digital LCD Display, detailed in section [LINK]. The rest of the data will be stored as a [FILETYPE] file on the Raspberry Pi. The extracted data file will be transferrable from the Raspberry Pi via USB onto the user’s computer once they have completed their journey. Once the extracted data file is on the user’s computer they will be able to upload the file onto the PIMPED web server into their own unique account. Here the user will be able to view all previously uploaded data and the data from the journey they have just completed. A full list of the data viewable to them from the web server can be found in section [LINK].

### Raspberry Pi Digital Display Monitor

A 3.2-inch Raspberry Pi LCD Digital Display will be used to display real-time data when the device is being used. The OBDII device returns data every one second, so the LCD display will be updated every one second.

## GPS Receiver

PIMPED will use journey tracking. In order to allow for this, it is crucial that a GPS receiver is installed into the PIMPED device. An Adafruit Ultimate GPS Breakout chip [REF] will be installed directly onto the Raspberry Pi.

## Battery

TBD

# Core Functionality

The core functionality of the device includes:

1. Display of real-time information;
2. Record journeys in car with GPS;
3. Dashboard online service that allows user to upload data to web server;
4. Scoring economic driving;
5. Compare friends results;

## Real-time information

We want to show real-time information as you are driving. Information should be shown on an LED display detailing:

1. Current miles per hour (MPH);
2. Current miles per gallon (MPG);
3. Miles driven in current drive;
4. GPS co-ordinates of the car.

This information should be scrolled across the LED screen in sequence. This information should not be distracting.

## GPS Record Journeys

We intend to have a GPS receiver component as part of our device that will be able to feed co-ordinates so that a map of the route taken can be drawn. The GPS receiver will start as soon as the device is connected to the car.

This will allow us to pinpoint certain events on a map. The colour of the route will gradually change depending on how economically you were driving. This is a similar idea to the route colouring technique used in the Nike Running app:



Figure : Example route colouring - Nike Running App

[REF] http://fitfeat.com/blog/wp-content/uploads/2013/08/nike-app.png

This would easily show where the speed was greatest or where you were driving most efficiently.  
We plan to use a digital display for the real-time information so the actual map of route taken would have to be displayed on our web interface as the digital display would not be able to handle this.

## Dashboard Online Service

The idea is that when drives have been recorded, the device can be connected to the Dashboard Online Service web service and upload all the information gathered. As a user you will be able to log into the web service and view the analysis of your drive or all of your drives.

You will be able to view statistics for individual drives and the route for each will be plotted onto a map. There will also be a screen of general statistics and averages of all your drives.

## Economic Driving

One of the main themes of the project is economic driving and we want to be able to tell the user how economic their driving is.

In our opinion there are two main factors in uneconomic driving. The first is in braking too harshly; this would be represented as a sudden decrease of speed from the data collected during a drive. The idea is that speed change should be gradual and planned. This is also the case for the second factor which is whether you are over-revving or under-revving. Over-revving and under-revving can be detected by checking the rev counter value is within the range of 800rpm-3300rpm.

We will use these factors to provide feedback to the driver on how they could drive more efficiently or tell them if they are already driving at optimum efficiency.

An Economic Driving Score will be calculated based on all the drives by a single driver.

## Compared Results

There will be a table on the Online Dashboard Service that allows you to compare your general statistics and your economic driving score to others who have used the device.

# Software Design

In order to make sure our software is as modular and therefore maintainable as possible, we will split out our code into separate applications, with each script dealing with one specific job. In this section we will detail how our scripts will carry out each of their jobs and how they will communicate with one another. In our planning stage, we identified four jobs that need to be performed in order for our device to function.

## Web Application Backend

This application is responsible for receiving data from the device and displaying in a clear and attractive way to the end user of our device. The application will require the user to login so that we can save their driving data specifically to their account online. When the user logs in, the user will be prompted to connect their device to their computer via USB, where the file stored on the device will be accessible. The file uploaded to the web application will contain a combination of the data about the car extracted via OBD and the GPX file generated from the devices GPS module. When uploaded, the data must then be stored. As all the processing of the data will be done on the device and saved in JSON format, we are going to use MongoDB [REF] to store it. MongoDB is a schema-less database and stores documents in a JSON format. This keeps data in the same format across our entire stack and makes it easy to query.

To handle the visualisation of data, we will need to make use of the Open Street Maps API v0.6 [REF] to display the GPS data for the journey. We will also need to develop a JavaScript library to allow us to render basic charts in the browser, for visualising feedback and comparing the users driving to that of other users on the system. The web application itself will be built using Ruby, HTML, JavaScript and CSS and will be hosted on a server running Ubuntu Server 14.04.1 LTS. DigitalOcean [REF], a cloud hosting company, kindly offer free credit to students, so we will be using them as our cloud service provider.

## GPS & OBD Extractor

As we are using a USB GPS unit on our device, we will need a script to extract this data and store it in a file that can be uploaded to the web later. Because Python is recommended as the language of choice for Raspberry Pi, that is the language we will be using. One concern here is power, so we do not want to have a script running constantly on the device. Our plan is that every § seconds, our Python script will run, get the latitude and longitude from the GPS unit, store this data in a file with a timestamp and then sleep, until it wakes and repeats. We will do the same thing for the OBD unit, polling it every second to extract data from the car. Both of these data points are combined by the device and stored on the internal SD card.

## File Format

We will use a JSON format for storing each journey and the events associated with it. This makes it easy to store in MongoDB and allows us to keep data for each journey in one document, rather than distributed across a relational database. Figure 4 shows the JSON file structure that will be used.



Figure : JSON file structure

## Render Data to Screen

We will be using a 7 Inch LCD screen, detailed in section 3.2.1, so we will need to design a GUI that displays the information that has been captured by the device. We are going to use Python to do this as well, using Python's built in GUI framework, TKinter [REF].

## Development Cycle

The following describes the order that each major part of the software will be implemented during development:

* Connect Raspberry Pi with GPS adapter and OBD cable and enable the Pi to extract data from both these devices.
* Store data from both USB devices to a file on the Pi. We will need to implement the file format detailed above.
* Build web application to upload file and show data in a meaningful way in the browser.
* Implement showing the live data from the storage file on a small screen.

# References

ELM327 OBDII Device Image:  
http://make.larsi.org/electronics/ELM327/ELM327v13a\_box.jpg

Full list of OBDII extractable data:  
<http://en.wikipedia.org/wiki/OBD-II_PIDs>

Raspberry Pi website:  
<http://www.raspberrypi.org/>

Adafruit Ultimate GPS Breakout:  
<http://thepihut.com/products/adafruit-ultimate-gps-breakout>

MondoDB website:  
<http://www.mongodb.org/>

Open Street Map API v0.6:  
<http://wiki.openstreetmap.org/wiki/API_v0.6>

DigitalOcean website:  
<https://www.digitalocean.com/>

TKinter:  
<https://docs.python.org/2/library/tkinter.html>