Renewable Energy Vehicles' User Interface:

Implementation of a Raspberry Pi

&

Engine Audio Replication System

Gabriel Feng
20253518
School of Mechanical and Chemical Engineering
University of Western Australia

Supervisor: Prof. Thomas Bräunl
School of Electrical, Electronic and Computer Engineering
University of Western Australia

Final Year Project Thesis
School of Mechanical and Chemical Engineering
University of Western Australia

Submitted: 28 October 2013

ABSTRACT

In this time when vehicles are getting more and more sophisticated, and information is abundant, there are many sources of distraction for a driver. A vehicle's user interface can be a major contributor if poorly designed. Time taken to complete a task on the user interface and the ergonomics of the action are two factors to consider in the design.

In this project the existing system in both Renewable Energy Vehicles are moved to a new platform and improved upon to provide a safer and more functional user interface. A comparison will also be done between the touchscreen interface in the Lotus Elise, and a one dimensional scroll wheel in the Hyundai Getz. The previous Engine Audio Replicating System will also be re-implemented.

Gabriel Feng
63 First Avenue
Rossmoyne, WA, 6148

28th October 2013

Winthrop Professor John Dell
Dean
Faculty of Engineering, Computing and Mathematics
University of Western Australia
35 Stirling Highway
Crawley, WA, 6009

Dear Professor Dell

I am pleased to submit this thesis, entitled "**Renewable Energy Vehicles User Interface**", as part of the requirement for the degree of Bachelor of Engineering.

Yours Sincerely

Gabriel Feng
20253518

ACKNOWLEDGEMENTS

The completion of this project would not have been possible without a number of people. I would like to thank Professor Thomas Bräunl for his continual help and guidance, without whom I would not have the opportunity to undertake this project. I would also like to extend my thanks to the guys in the robotics lab, the conversations and discussions helped with new ideas and stress relief. Rémi Keat who was undertaking a project as an exchange student, helped greatly with code debugging. Thanks to the REV team for their support.

I would like also to thank my friends and family for their support through the writing of this thesis. Finally, I would like to thank my girlfriend, Justine, for being patient while I spent many hours in the lab and for her support in times of difficulty.

TABLE OF CONTENTS

1. INTRODUCTION

The current fleet of vehicles in the University of Western Australia's (UWA) Renewable Energy Vehicle (REV) Project includes the 2013 Electric Self-Driving SAE Race Car, 2013 Electric Hub Motor SAE Race Car, 2013 Electric Jet Ski, BMW Drive by Wire, 2008 REV Eco (Hyundai Getz) and 2009 REV Racer (Lotus Elise). The REV Eco and REV Racer, a five-seater commuter vehicle and two-seater performance vehicle respectively, are currently the only cars in the fleet that are road registered. One of the goals of the REV Project is to "Create a series of vehicles appropriate for the personal transport needs of the Australian public … "(Braunl n.d.). Since this process is aimed towards the Australian public, a key task would be to optimize and implement a safe and easy to use Human Machine Interface (HMI).

Over the last few years several students have developed and improved the Graphical User Interfaces (GUIs) for both the REV Eco and REV Racer, as shown in Figure 1.1 to Figure 1.5.

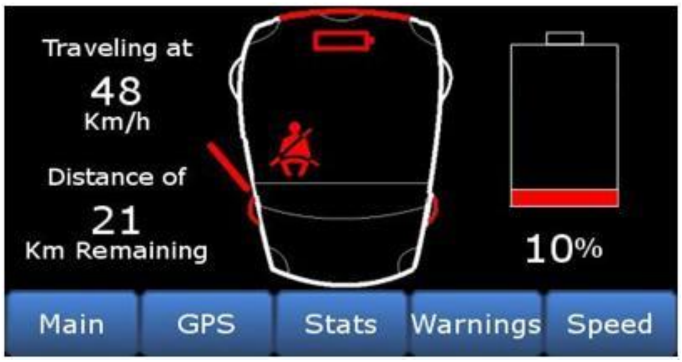


Figure 1.1 GUI for Getz (Varma 2009, p. 30)

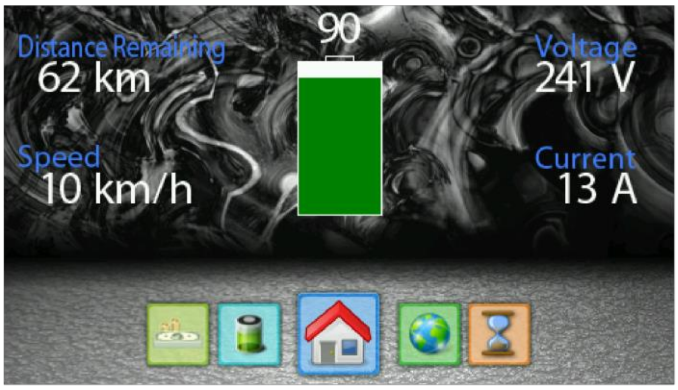


Figure 1.2 GUI for Getz (Trepp 2011, p. 73)

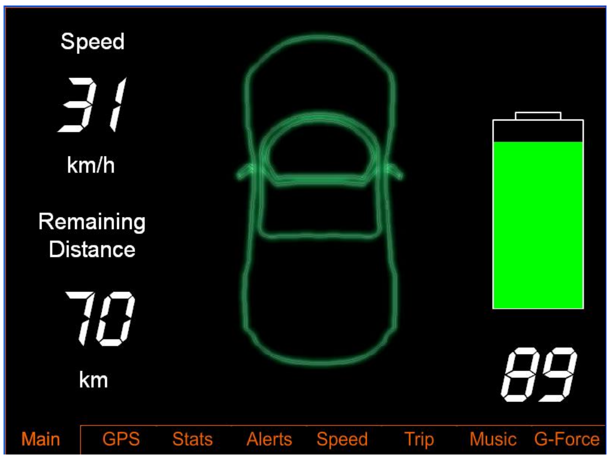


Figure 1.3 GUI for Lotus (Varma 2009, p. 56)

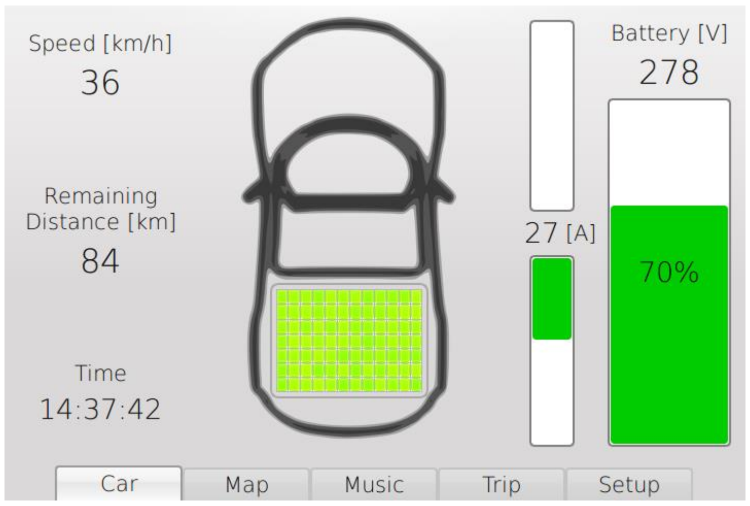


Figure 1.4 GUI for Lotus (Walter 2010, p. 61)

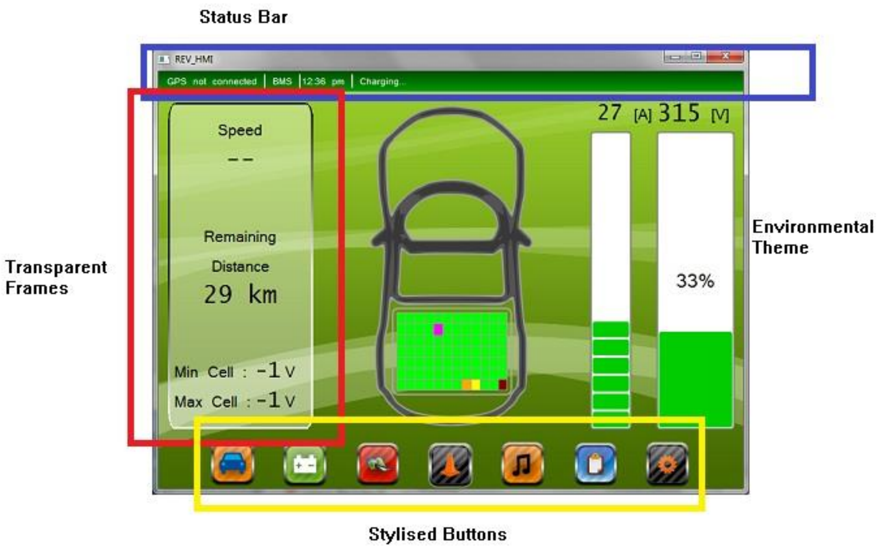


Figure 1.5 GUI for Lotus (Tyler 2011, p. 11)

It can be clearly shown that the aesthetics of the GUIs have improved and a wealth of information has been provided, however the layout and quality of information have remained similar through the variations. The downside of the amount of information available is that it could lead to a decrease in the driver's ability to concentrate primarily on the driving task. (Klaus Bengler et al. 2012, p.106)

The quality of interaction between the user and interface, such as time taken (Regan et al. 2009, p. 201) and ergonomics (Niedermaier et al. 2009, p. 444), is crucial; especially considering the context the interface will be used in. In terms of ergonomics, Western Australia has in place *Road Traffic (Vehicle Standards) Rules 2002* (they differ from state to state) that attempt to prevent electronic devices from being a source of hazard in a vehicle. This disallows visual display units from being visible to the driver, with the exception of a driver's aid (which is not explicitly specified). A driver's aid however is constrained to not obscuring a driver's view of the road and not impeding the movement of a person in the vehicle.

There are a number of guidelines and standards (Regan et al. 2009, pp. 450-451, 454-455) that designers can use when developing a HMI to minimise distraction. These include details ranging from where the device should be located relative to the driver to a text's character font size. BMW's use of ergonomics has resulted in an organised structure of dividing the display's and controls. Key displays are placed close to the forward view of the driver to minimise time taken to glance between the display and the road. Whereas controls and information that are critical to the driving task are placed in front of the driver while comfort displays and controls are placed in the middle of the vehicle (Niedermaier et al. 2009). The controls are also placed in close proximity to the driver such that the movement to the controls does not require the shoulder to be lifted off the seat.

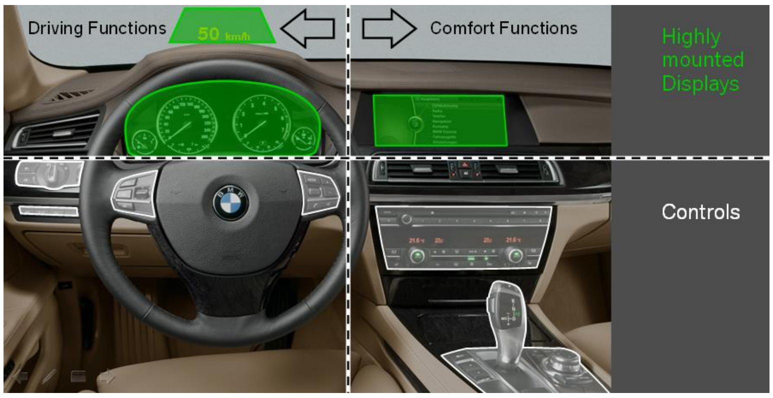


Figure 1.6 Division of displays and controls (Niedermaier et al. 2009, p. 445)

It is possible to measure objectively the amount of driver distraction a device causes. Japan Automobile Manufacturer's Association (JAMA) conducted tests that measured the vehicle's lateral displacement, speed, steering angle, accelerator operation, headway distance, lateral acceleration, total glance time and a subjective evaluation of driver stress (Regan et al. 2009, p.437). The tests were performed over four different road conditions using four different input methods (touch screen, joystick, remote control and rotary knob) to operate a navigation system. It was concluded that the maximum permissible total glance time, which does not cause stress or affect vehicle control, is 8.0 seconds. Glance time can also be measured in a laboratory environment using the occlusion technique, this consists of a user's vision being blocked and unblocked for certain time intervals to simulate a driver glancing between the road and display unit.

More care needs to be taken into designing the HMI for the REV Eco and REV Racer, taking into consideration the target audience and use of the vehicle. Further tests also need to be conducted to verify the HMI developed follows certain guidelines and standards chosen. An iterative process is required to further perfect the system although realistically there is no one solution for all conditions.

Another issue with regards to safety of an Electric Vehicle (EV) would be the minimal engine noise from an electric motor. Audio cues are used to sense surrounding traffic before visual cues, even more so for the visually impaired person. The visually impaired are taught to use the sound of vehicles as a reference, they use the audio cues to sense oncoming traffic and the direction of traffic (Deverell 2009, p. 199). More importantly they rely on audio gaps as a sign for a gap in traffic. Mistaking a lack of noise from an EV as an audio gap could lead to disastrous results.

When the noise level between the REV Eco and a petrol-driven Getz was compared it was found that the petrol-driven Getz measured 6.0dB and 14.5dB higher than the REV Eco while stationary and accelerating from stationary respectively (Braunl 2012). It was thus concluded that engine sound would be beneficial at speeds less than 25 km/h and while starting or reversing. In fact, there is a proposed rule in the United States for a minimum sound level for EVs (Federal Motor Vehicle Safety Standards; Minimum Sound Requirements for Hybrid and Electric Vehicles, C.F.R, pt 571).

In another article written in the Acoustics Australia Journal (Sandberg 2012) it is agreed that EVs are quieter than Internal Combustion Engine (ICE) vehicles in certain conditions. However, it disputes adding audio simulation or some kind of alerting system to EVs as a valid solution. Sandberg recommends that background ambient noise, which masks whatever minimal noise an EV might have, should be reduced through the replacement of heavy ICE vehicles with electrically driven vehicles. Since both EVs and ICE vehicles have the same approximate noise levels above 20 km/h due to tyre noise, Sandberg also argues that adding additional sound for speeds less than 20 km/h diminishes the noise reduction qualities of an EV. Sandberg concludes that non-acoustical solutions should be applied such as autonomous emergency braking systems or a soft horn used by GM Volt, where the driver initiates the sound on suspicion of a potential danger ahead, as a compromise.

Further review needs to be done into how drivers and pedestrians can be alerted to potential collisions, simply adding noise to a slow moving EV is merely a temporary solution. Such as, in what situations would the system need to alert the user, does it alert the pedestrian or the driver, or is the system based on a pedestrian looking out for a vehicle or vice versa? Imagine a congested street in the Central Business District of a busy city, where all vehicles are electric and all are emitting some kind of acoustical alerting system. Contrast it with the same street where the vehicles use a non-acoustical collision prevention system.

A Raspberry Pi is replacing the systems in both the REV Eco and REV Racer. The Raspberry Pi (Raspberry Pi Foundation n.d.) is a relatively tiny and low power consuming computer that is capable of doing many different tasks from playing high definition video to acting as a web server. The aim of the device was to introduce a relatively cheap computer to the educational market and encourage students to explore, experiment and get interested in programming without fear of damaging an expensive computer (Raspberry Pi Foundation n.d.). The Raspberry Pi is fitted with a Broadcom BCM2835 chip, 512MB RAM, and a Videocore 4 GPU. It also has 2 USB ports, an Ethernet port, an HDMI port, RCA Video output, 3.5mm audio output, several GPIO pins and an SD card slot from which it boots and runs. Figure 1.7 shows the components of the Raspberry Pi.

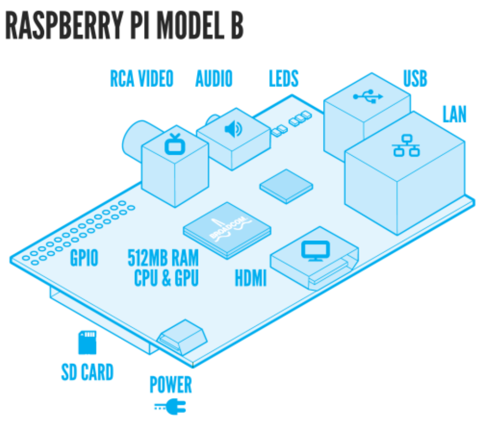


Figure 1.7 Components of Raspberry Pi (Raspberry Pi Foundation n.d.)

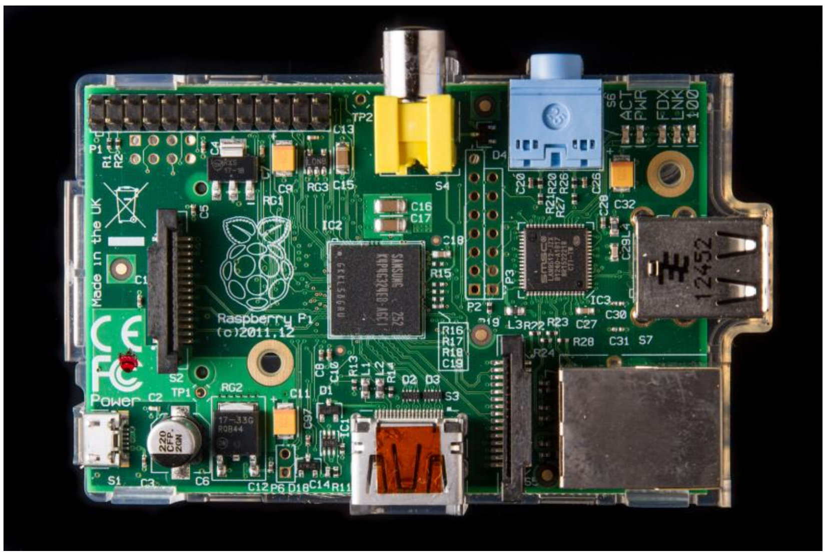


Figure 1.8 Raspberry Pi (Gabriel Feng 2013)

2. OBJECTIVES

The project will improve upon the existing HMI systems in the REV Eco and REV Racer, by modifying the GUI to improve visibility and interaction, implementing similar systems in both vehicles.

Apart from updating the GUI, both vehicles' current layout will be analysed for their ergonomics with regards to HMI. Recommendations will be made to improve the usability of the GUI and vehicular safety through ergonomics.

The specific tasks of the project are as follows:

1. To modify the GUI in both REVs to improve visibility and logging;
2. To re-arrange wiring within the REV Eco centre console for easier work and safety;
3. To program the EyebotM6 to automatically power down when ignition is turned off;
4. To port current REV Racer's GUI onto a Raspberry Pi;
5. To modify REV Racer's GUI to be suitable for REV Eco;
6. To improve quality of interaction for both vehicle's HMI;
7. To implement Raspberry Pi into both EVs;
8. To re-implement a synthetic engine sound system
9. To get tachometer in REVs working independent of GUI

Through re-implementation of a synthetic engine sound system, improvements to the GUI and improvements of the HMI with ergonomics this project aims to improve the overall safety of the vehicles.

3. PREVIOUS DESIGNS

Initially the objective of the project was to replace the PC in the REV Racer to a Raspberry Pi (RPi) and leave the EyebotM6 in the REV Eco. However, the scope was later changed to allow a more uniform system across the vehicles. As such objective 3 becomes obsolete and Section 3.4 will cover the changes made to the EyebotM6.

3.1. PREVIOUS DESIGN – REV ECO

The REV Eco previously used the EyebotM6 embedded controller, which is fitted with a Gumstix 400Mhz PXA255 processor and connects to a Samsung LTE430WQF0 touchscreen (Trepp 2011). The programs on the controller consist of the main GUI, logger, Global Positioning System (GPS), and Battery Management System (BMS) daemons. The daemons used the ZeroMQ library as a messaging system to transfer information from the serial output of the GPS and BMS to the GUI. The programs were refined from previous versions by Beau Trepp in 2011 for his Final Year Project. The BMS used is a TBS E-Xpert pro battery monitor that communicates to the EyebotM6 via a RS232 communication kit (TBS Electronics n.d.).

The program, developed by Beau Trepp, allows the user to view the details about the battery, trip, money saved, map of current position etc. on different pages. A logger also logs the GPS details and battery information onto separate CSV files. This provides the user with data to analyse for each trip.

3.2. PREVIOUS DESIGN – REV RACER

The REV Racer utilised a Dual Core Atom 1.6 GHz PC mounted within an enclosure specified for work in automotive environments (Tyler 2011). It is connected to a Xenarc 700TSV touchscreen monitor (Xenarc Technologies n.d.) via a USB interface. The BMS, designed by Ivan Neubronner, allows details of each specific battery cell to be transmitted to the PC.

The system was developed on the Qt C++ framework (Qt Project n.d.) by Thomas Walter in 2010 (Walter 2010) and modified by Matthew Tyler in 2011 (Tyler 2011). The program allows the user to view details about each battery cell's condition, map of the vehicle's current position, trip details, play music, and change the language if a translation is available. Furthermore the program was designed to implement the Engine Audio Replicating System (E.A.R.S.).

The E.A.R.S. takes the engine speed data from the GUI and opens up a pre-recorded sound file of an engine at that corresponding speed rounded to the nearest 100 revolutions per minute. The system was written in C# separate from the GUI program by Chris Hellsten and improved upon by Matthew Tyler (Tyler 2011, pp.17–22), the code utilised Microsoft's DirectSound libraries which provides great audio playback.

3.3. PROBLEMS WITH PREVIOUS DESIGN

At the beginning of the project the EyebotM6 controller in the REV Eco had the only working copy of the software. The controller also did not shut down automatically with the ignition key of the vehicle; the controller was usually powered down by cutting the power upon turning off the ignition power. This repeated action would sometimes corrupt the system image, requiring a re-flash of the EyeBot.

The colour scheme (blue on black) for the GUI was satisfactory in a laboratory condition but was not suitable in a vehicle in certain lighting conditions. The button navigation was also inefficient. The default page on startup and arguably the most important page is the battery status and distance remaining page. However the home button does not lead to this page, rather it leads to a page with buttons that lead to the other pages.

One of the reasons for replacing the EyebotM6 with the RPi was due to the fact that any physical hardware work that needed to be done on the unit required the center fascia to be removed, which was time consuming and repeated removals made the clips holding the center fascia in prone to breaking. The wiring layout in the vehicle behind the centre fascia was also very messy and potentially dangerous, with some wires having exposed copper, some wires being cut (which caused the LEDs in switches to not turn on with the switch), or even unused cables. Removing the EyebotM6 also required removing all the connectors and disconnecting the main unit from the screen connected by a fragile ribbon connector, risking damage to the pins or the connector.

The REV Racer was in better shape, with only the E.A.R.S. not currently working. Although the wiring in the Racer was neater than the Eco, the design of the Lotus Elise and placement of the PC enclosure makes working in the vehicle ergonomically unsound.

3.4. Changes To The EyebotM6

Since the EyebotM6 controller is now removed from the REV Eco this section is all but obsolete. Still, the source code for the daemon that was used to shut down the EyebotM6 automatically can be found in Appendix A.

With regards to logging, all the important data from the vehicle was already being logged, such as position data, speed and battery state. However, optional logging was added to the EyebotM6; originally data logging was done automatically at the start of the program. However, to increase performance of the GUI the user is given the option as to when to start and stop the logging process. A button was added, using the windowing toolkit provided, to the *Trip Panel* that would start the logging daemon and kill the logging daemon on each press.

The exit process was also modified to call the main program associated with the EyebotM6, written by Azman Yusof, if the user decides to exit the GUI. The associated daemons are also closed on exit. Previously upon exiting the GUI program, the user will just be taken to a blank screen.

As mentioned earlier the previous design of the Eyebot M6's GUI was difficult to read at various sun angles. The colour scheme was changed to a dark green coloured background with bright yellow and white coloured text, providing a greater contrast between the text and background. Figure 3.1shows the new colour scheme, the colours chosen also reflect the scheme of the REV logo.

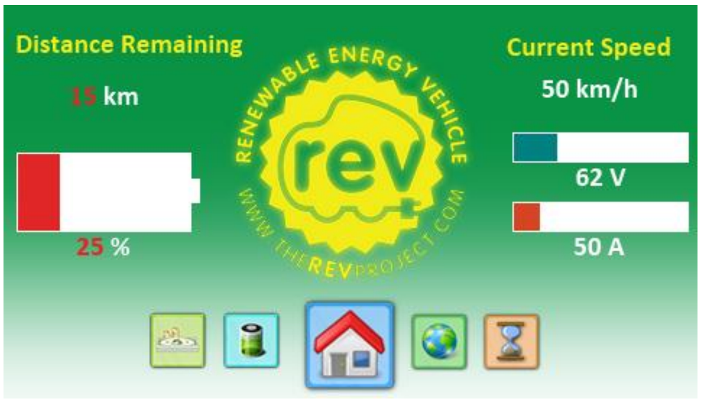


Figure 3.1 New colour scheme

4. VEHICLE ERGONOMICS

Originally both vehicles used a touch screen to interface with the GUI. Now the REV Eco is fitted with a Griffin Powermate input device, shown in Figure 4.1. The advantage of using a touch screen is that the user is not limited to interacting with the program in a certain order. The Powermate, due to its rotational wheel unlike a mouse, requires the user to highlight each option before moving on to the next, until they reach their desired option. Obviously this takes a longer time for the user, with a touch screen an option on a certain page can be selected within one click, but with the Powermate it is dependent on the number of options on that page. On the other hand the precision required by the user of a touch screen is dependent on the area of the specific input, i.e. a larger button requires less precision by the user.

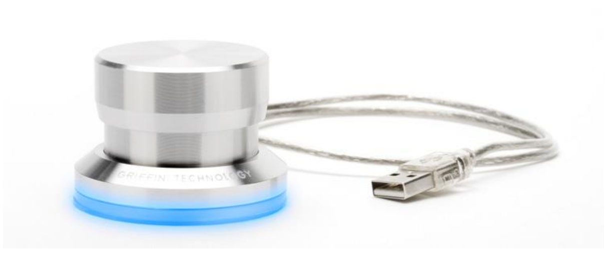


Figure 4.1 Grffin Powermate (Griffin n.d.)

Unlike physical buttons that provide some form of tactile feedback, the user is required to look at the screen in order to know what button is being pressed. This act while driving, like looking at a mobile device, is distracting and not safe. Utilising the Powermate limits the user to 6 different input actions to the software but allows the user shorter glance time of the screen. Since the Powermate has to progress through each option in a page in order, the number of options per page has to be reduced. Otherwise the time the user takes to interact with the device increases, which is also a distraction and reduces safety. Fortunately the GUI is responsive and the user is able to roll through options relatively quickly. There are also not more than 10 options per page, which means that the user is able to select an option on a page within 5 rotational steps.

An issue with this device is that the rotation of the knob is physically stepless, without the stepped tactile feedback the user cannot know for sure without looking at screen how many “buttons” have passed. To address this, a “click” sound effect was linked to each virtual step to provide an audio feedback to the user. There was no feedback at all previously, and since it is “one of the most important factors by which a coherent understanding of the system state can be provided” (Heiner Bubb 2012, p.52) it is implemented for the touchscreen alternative as well. The ALSA library (ALSA n.d.), which is discussed later, was used to create the click effect.

Optimal placement of both the visual display and the input device aids the quality of interaction and safety of the vehicle, their placement also either contributes to or subtracts from the overall ergonomics of the vehicle. Following BMW’s footsteps in Figure 1.6 would place the visual display in the REV Eco as shown highlighted in Figure 4.2.

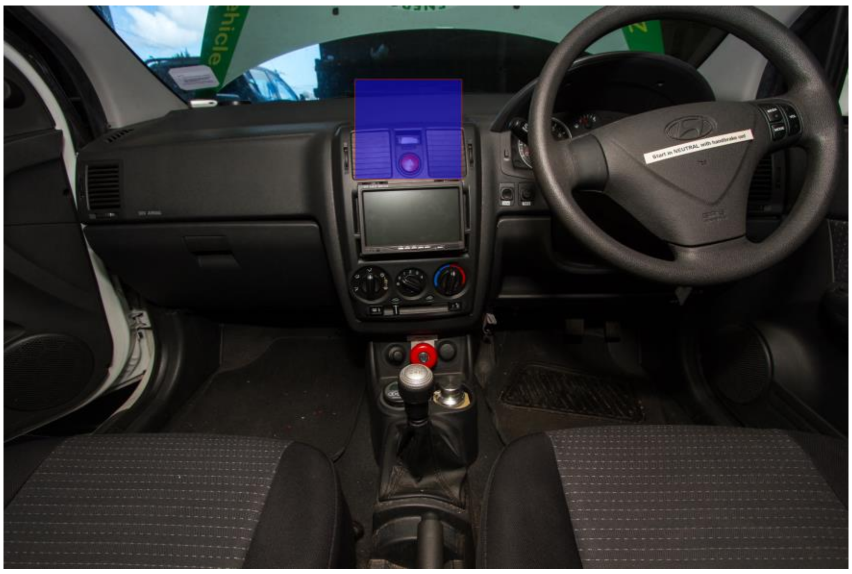


Figure 4.2 REV Eco Display-Optimal Location

However, putting the display at the optimal location would be a retrofit and as the vehicle was not specifically designed for the visual display being in this location, depending on the size of the display and the height of the driver, this could result in the display obstructing part of the driver’s view of the road. This violates the *Road Traffic (Vehicle Standards) Rules 2002* mentioned earlier in the introduction. Placing the display here would also require major work on the interior, which is costly.

The Powermate should be placed in the location shown highlighted in Figure 4.3, as it would be close to the driver without having the driver lift their shoulder from the seat. However, this would interfere with the operation of the hand brake. This shows that retrofitting the vehicle with an interactive visual display, essential to the electric conversion, has some compromise on optimality unless the interior of the vehicle can be redesigned.

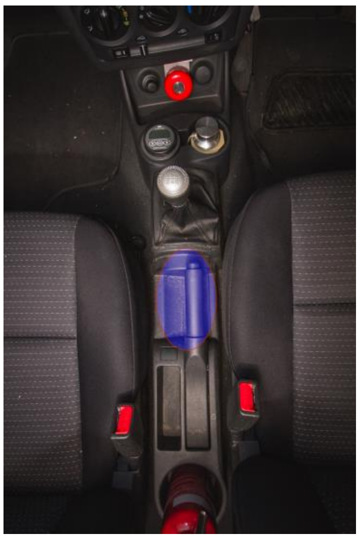


Figure 4.3 REV Eco Powermate-Optimal Location

The next best location for the display, without modifying the instrument cluster on the dashboard, would be where the previous display was located; this is not the first choice since it requires the driver to look further down and away from the road. We can compare the difference between the optimal location and the current location by looking at the line of sights. In Figure 4.4 the white line shows the line of sight of the driver looking straight ahead at the road, the red line shows the line of sight for the current location and the green line shows the line of sight for the optimal location. We can see that the difference in the glancing angle between the line of sights for the two locations and the line of sight for the road is approximately halved. We can speculate that the reduction in this angle for the optimal location would result in less time needed to glance between the display and the road.

One could argue that the driver need not or should not be allowed to look at the GUI while the vehicle is in motion. This can be possible if the driver does not require the map or media playing functionality. Further research will need to be done to determine whether this concept is viable to market and if interacting with navigation and media software significantly distracts a driver to warrant a safety standard that disallows GUIs from being displayed while the vehicle is in motion.

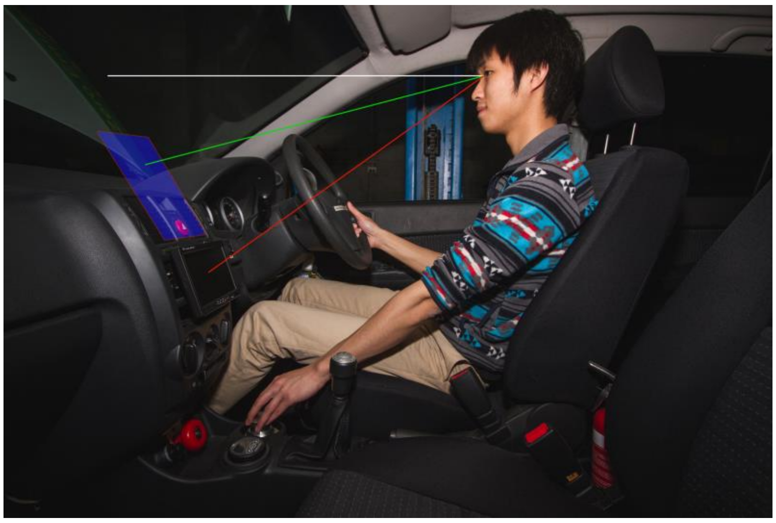


Figure 4.4 REV Eco-Comparing Line of Sights

The next best location for the Powermate was in the cup holder behind the gear selector. This is not completely ideal for two reasons; firstly, some drivers might have to lift their shoulder off the seat to reach the device, for most drivers however the device should be within reach. Secondly, while the device is within reach the posture of the driver is not in a natural position, leading to fatigue over an extended period of time. Again without re-designing the interior of the vehicle this location is a compromise.

Similarly, having to retrofit the displays for the REV Racer results in compromising between optimality and rework needed for the interior; additionally the REV Racer's interior makes it very difficult for the display to be placed in a satisfactory location. Figure 4.5 shows two ideal locations for the display. In the first optimal location, highlighted and labeled '1' in the figure, since the display is close to the windscreen, it would minimise the time the driver looks away from the road. However, in this vehicle due to the low seat height and depending on the height of the driver the display could obstruct part of the windscreen. Again this violates the *Road Traffic (Vehicle Standards) Rules 2002*, and can be considered hazardous. The second ideal location, highlighted and labeled '2' in the figure, does not obstruct the windscreen but the area already houses the vehicle's climate controls. Placing the display in that location would require major work on the interior of the vehicle to move the controls further to the left.

Despite these difficulties, the current location of the display is not recommended as it is requires the driver to lift their back from the seat in order to reach the display. This is not ideal especially while the vehicle is in motion. Figure 4.6 shows the distance between the driver and the display.

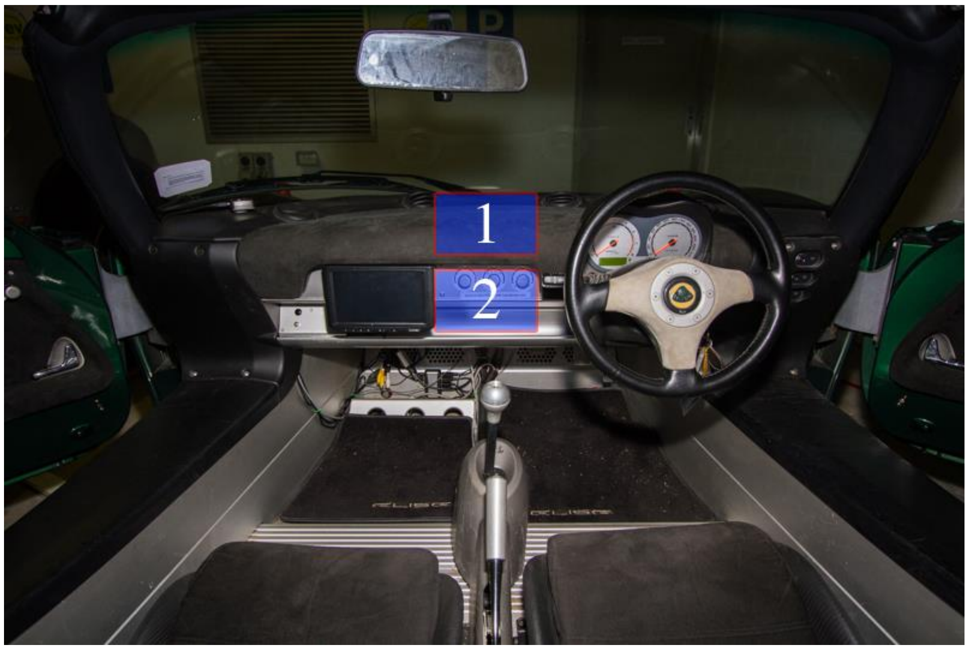


Figure 4.5 Ideal locations for display

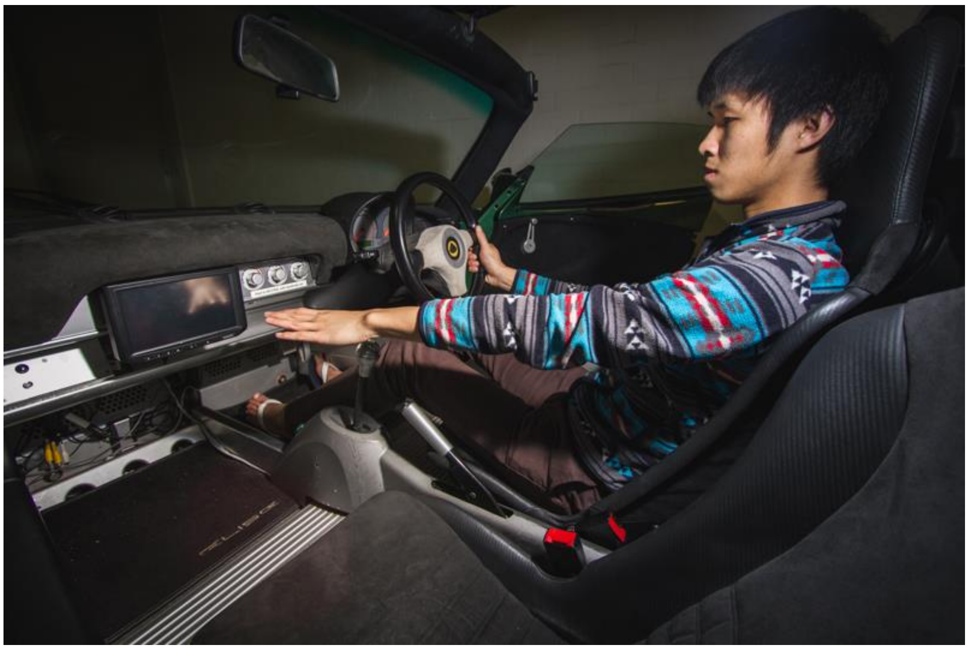


Figure 4.6 Driver reaching for the display

5. Revised Design

An important note for working with the RPi; since a user will likely be removing the SD card regularly the user should take care that the power is disconnected from the RPi before removing or inserting the SD card or risk the card being corrupted.

5.1. Improving Visibility

Due to the rotary nature and single navigational dimension of the Powermate device, the layout of the GUI is changed from a fully tabbed layout to a reduced tabbed layout, shown in Figure 1.5 and Figure 5.1. The main screen displays the key information of the vehicle such as the battery status, error icons and condensed information of each page. As the user rotates the device, the condensed information at the center of the circle changes its information corresponding with the highlighted "button". Figure 5.2 shows four "buttons" being highlighted and their respective information within the circle. This design minimizes the actions required by the user to pull key information from the GUI. If the user requires more information they can enter the page by clicking the device.

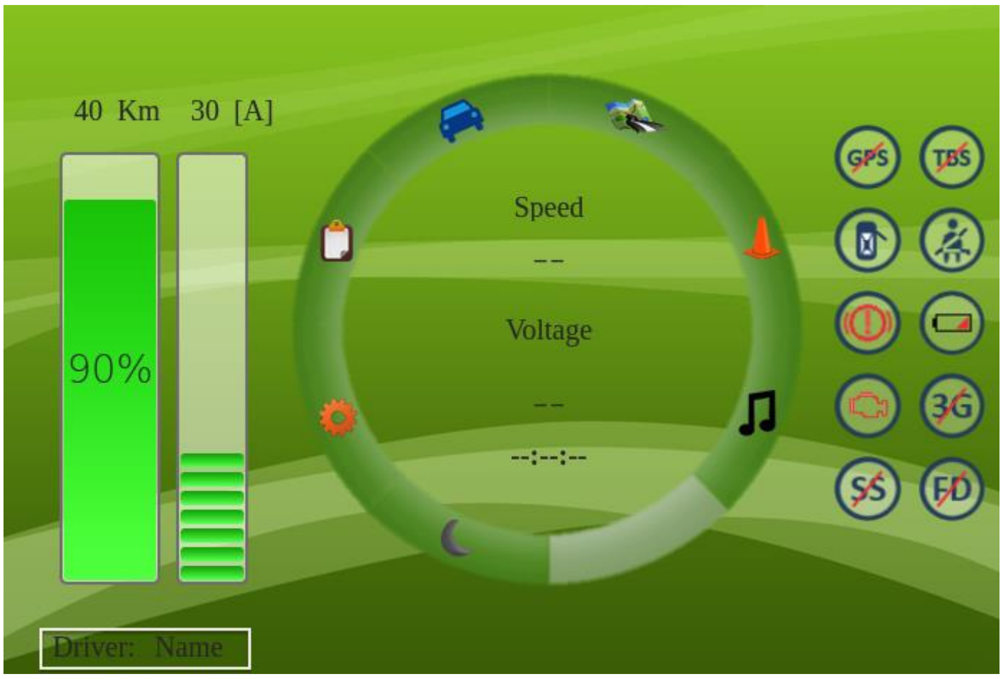


Figure 5.1 New layout of GUI

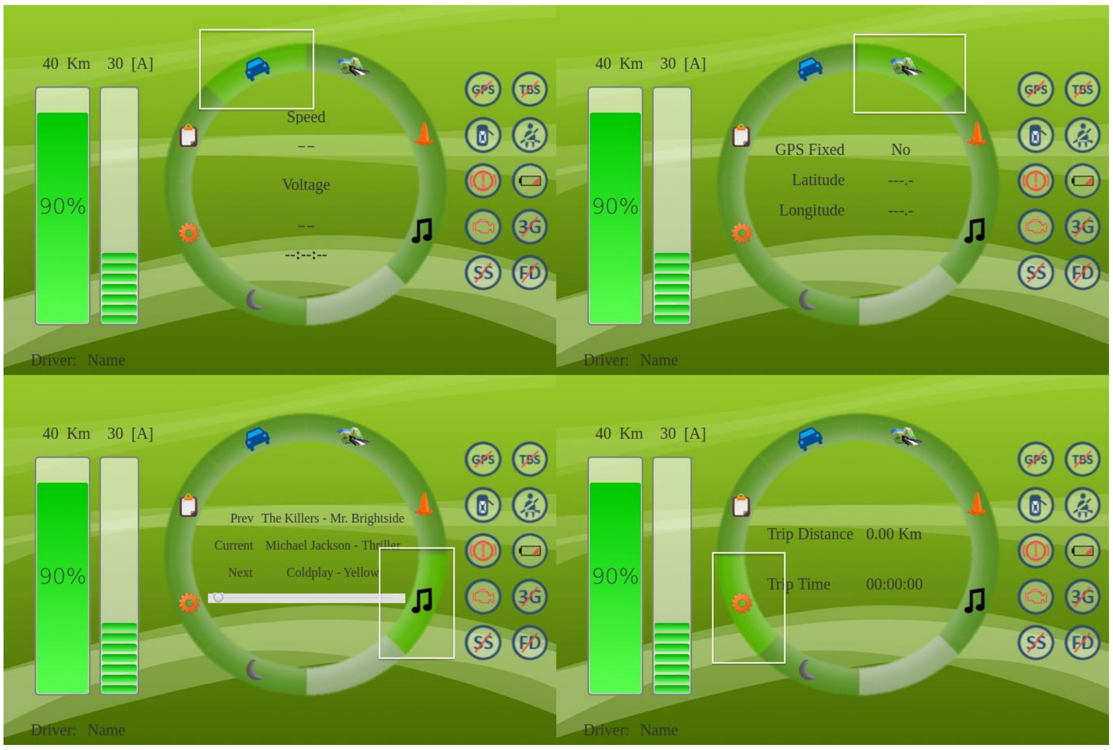


Figure 5.2 Highlighted menu items

The colour for the battery percentage bar was set to change to yellow when the battery's state of charge (SOC) goes below 50% and to red when it drops below 20%, shown in Figure 5.3 and Figure 5.4 respectively. This enables the driver to quickly recognize the urgent need to find a charging station without having to specifically read the numbers and thus reducing glance time. Furthermore, an alert sound effect is programmed to play when the SOC is below 20% in the REV Eco or when the voltage of any battery cell in the REV Racer falls below a threshold. The formula, (5.1), used to calculate the distance remaining from the battery's SOC was also modified to reflect the vehicle's current maximum range, given a safety margin of 10%.

$$\boldsymbol{Distance\ remaining = SOC * Max\ range * 0.9} \tag{5.1}$$

Since the maximum range of the vehicle decreases as the batteries get older, a new addition for the GUI is that the user is now allowed, after entering a password, to modify the maximum range of the vehicle in the Trip page. This will help ensure an accurate calculation of the distance remaining.

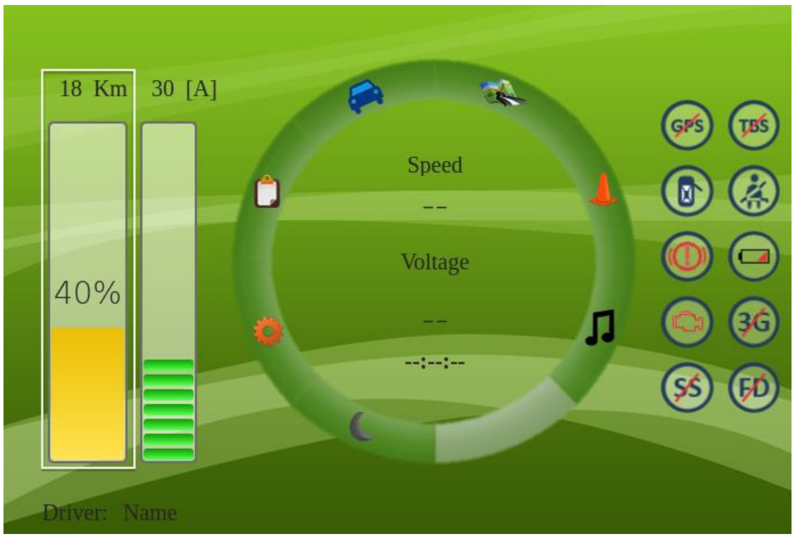


Figure 5.3 Battery SOC less than 50%

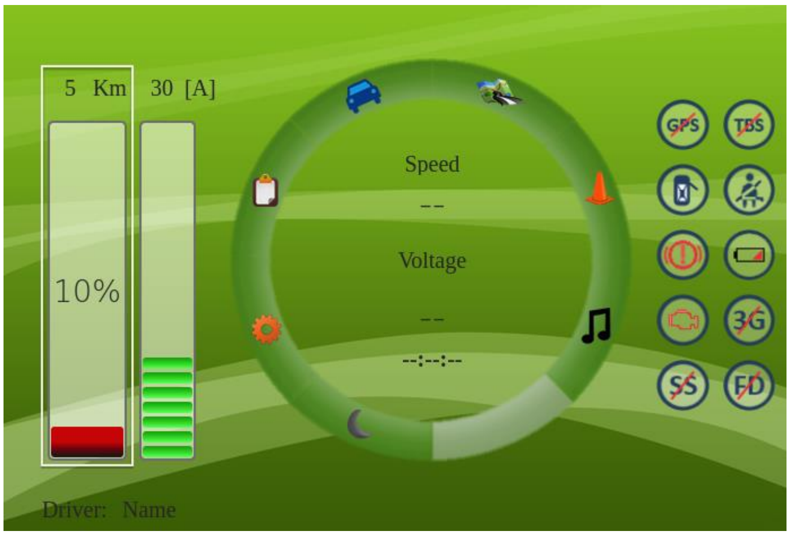


Figure 5.4 Battery SOC less than 20%

A potential problem found with the previous setup of the GUI is that under low lighting conditions such as during the night the user glances between a bright screen in the vehicle to the dimmer exterior of the vehicle or if the bright display is within the driver's field of view. This can be hazardous if the contrast is too great or if the driver's night vision is compromised. The map widget already has a feature to invert the generally light colours of the map to darker colours. The GUI is now similarly programmed to switch its background to a darker tint after a specified time; the fonts are also changed to a lighter colour for contrast, increasing visibility. (Figure 5.5) In addition to this, the user is allowed to display a completely black screen to further dim the interior of the vehicle. If a light sensor is installed in the vehicles, the "night mode" can be easily changed to be activated automatically.

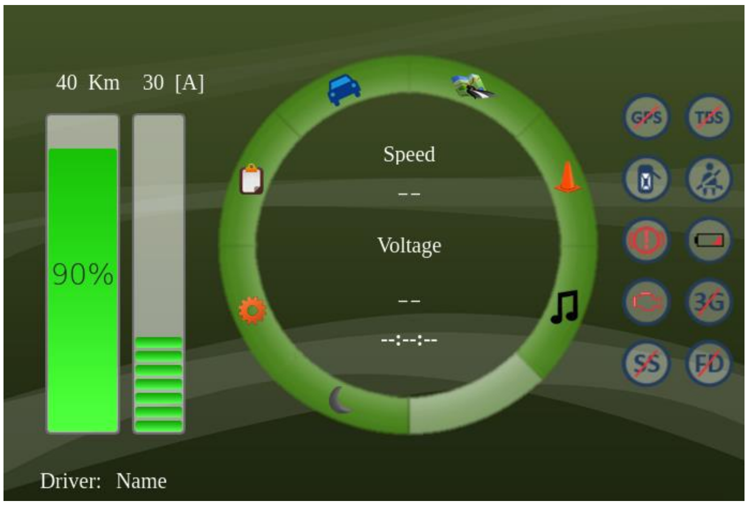


Figure 5.5 Night view of GUI

Figure 5.6 compares the old and new design of the maps page. The patterned area shows the space given to the map widget. In the new implementation the whole screen area is given to the map widget, this allows the user a larger view of their surroundings. This is especially useful in the REV Eco since it uses the Powermate as the drawback of using the Powermate is that the user cannot manipulate the slippy map widget programmed by Thomas Walter (Walter 2010, pp.51–54), which requires a click and drag action. With the new implementation the user can hide the controls for a clearer view of the map with the "Hide Controls" button, and by clicking the Powermate again the control panel reappears. The figure also shows that the currently highlighted button's text changes to red to show the user clearly which button is highlighted by the Powermate.

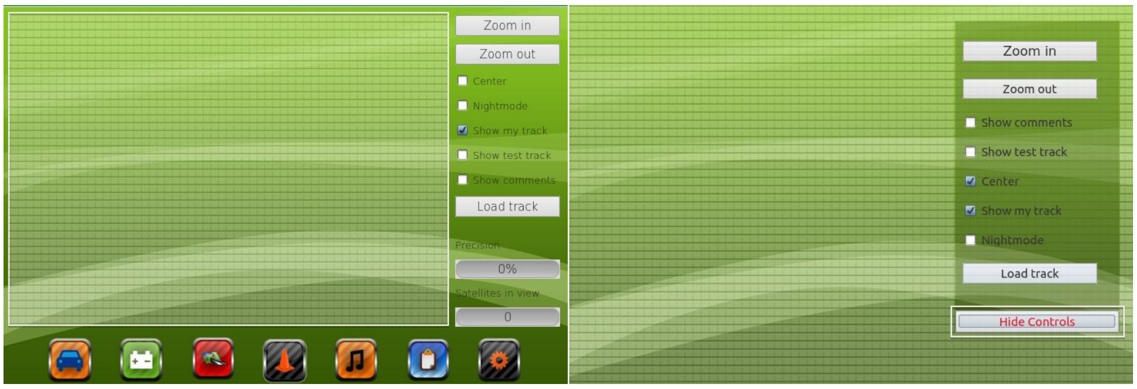


Figure 5.6 Old vs New Maps Page

On the right of the main screen are two columns of icons. These provide the user with a quick identification of the system's warnings or errors; some are replicated errors/warnings of the dash instrument panel. Table 5.a shows the icons and their meanings.

GPS	Invalid GPS message / Not connected	TBS	Invalid BMS message/ Not connected
	Car door open		Seat belt unbuckled
	Emergency brake engaged		Battery voltage low (<20%)
	Check engine light	3G	Internet (3G) connection error
SS	Safety switch engaged	FD	Fuel door open

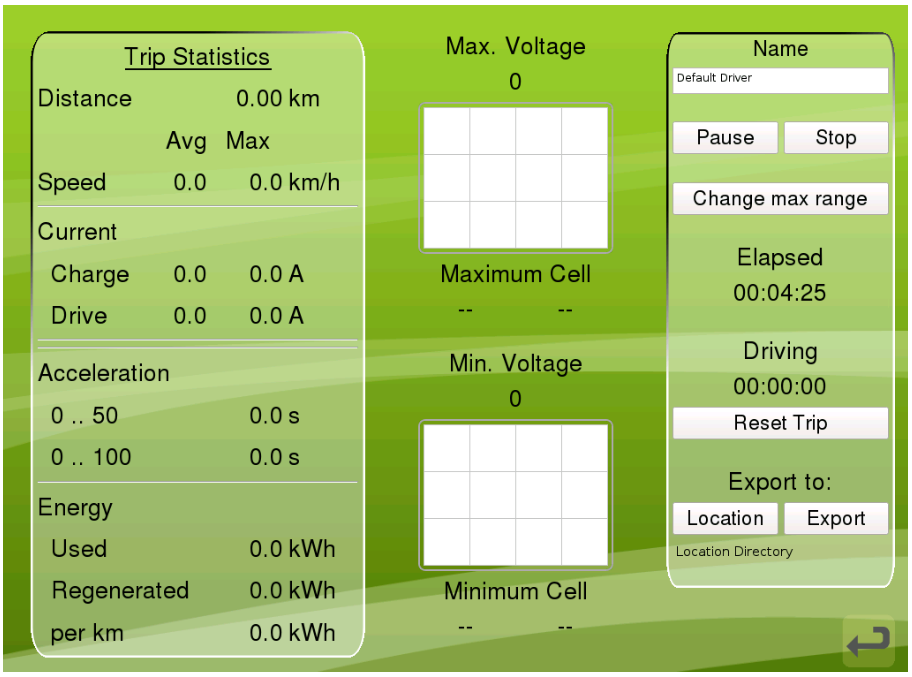
Table 5.a Icon labels

A quick but logical addition to the “Trip Computer” was the ability to reset the trip values. Previously the trip values were reset every time the program restarted. As compared to any other vehicle, the trip data remains until the user resets everything to zeros. An additional “Reset Trip” button was added to the “Trip Computer” page; it is also complemented with a message box asking the user to confirm the action before proceeding. This can be seen in Figure 5.7.

5.2. Improving Logging

The REV Racer’s GUI already allows the user to stop, pause and continue logging at their specification. However, in order to allow for better separation of the data for analysis, the name of the driver is also added into the log files. The user also has the ability to change the name of the driver in the GUI. The GUI also allows the user to export the log files to a specified location. Figure 5.7 shows these features, it also shows the previously mentioned button that allows the user to modify the maximum range of the vehicle.

Previously, the logger would attempt to create log files without any checks. Since the RPi uses an SD Card where space is an issue the logger will now check the logging directory and display a warning if the size of the directory is close to the specified size constraint or disallow logging if the directory is larger than the specified size constraint. Figure 5.8 shows the Debug screen with the warning and error.


Trip Statistics
Distance 0.00 km
Avg Max
Speed 0.0 0.0 km/h
Current
Charge 0.0 0.0 A
Drive 0.0 0.0 A
Acceleration
0 .. 50 0.0 s
0 .. 100 0.0 s
Energy
Used 0.0 kWh
Regenerated 0.0 kWh
per km 0.0 kWh

Max. Voltage
0
Maximum Cell
-- --
Min. Voltage
0
Minimum Cell
-- --

Name
Default Driver
Elapsed
00:04:25
Driving
00:00:00
Export to:
Location Directory

Figure 5.7 Trip Computer

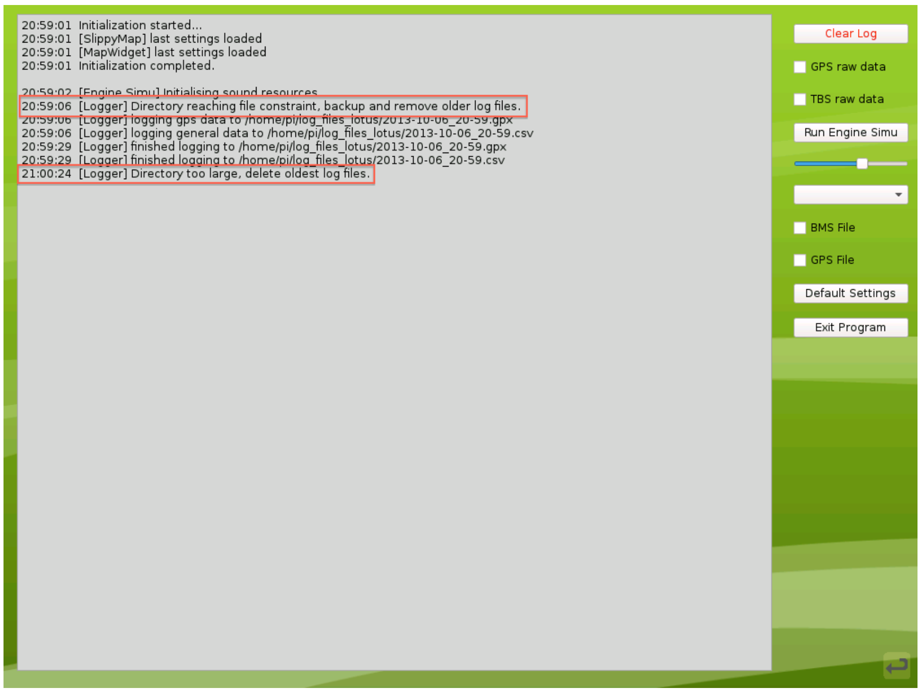

```
20:59:01  Initialization started...
20:59:01  [SlippyMap] last settings loaded
20:59:01  [MapWidget] last settings loaded
20:59:01  Initialization completed.

20:59:02  [Engine Simu] Initialising sound resources
20:59:06  [Logger] Directory reaching file constraint, backup and remove older log files.
20:59:06  [Logger] logging gps data to /home/pi/log_files_lotus/2013-10-06_20-59.gpx
20:59:06  [Logger] logging general data to /home/pi/log_files_lotus/2013-10-06_20-59.csv
20:59:29  [Logger] finished logging to /home/pi/log_files_lotus/2013-10-06_20-59.gpx
20:59:29  [Logger] finished logging to /home/pi/log_files_lotus/2013-10-06_20-59.csv
21:00:24  [Logger] Directory too large, delete oldest log files.
```

Figure 5.8 Logger warnings and errors

To check the directory size the function uses the `<sys/stat.h>` library; it loops through all the regular files in the directory and totals up their file sizes. If the total exceeds the specified size constraint then logging is not allowed and a debug message is printed to notify the user. Figure 5.9 shows the flow chart of the logger. Due to the addition of Input/Output (I/O) functionality the additional information should be logged as well. Appendix B shows the snippet of code used to check the log directory's size.

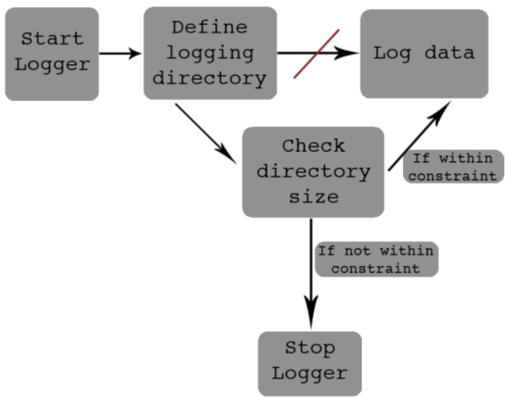


Figure 5.9 Flow diagram of Logger

5.3. Tachometer Design

One of the major concerns with the REVs is that both vehicles are designed to use a manual gearbox but with the clutch removed. Although both vehicles have enough torque to be driven in one gear most of the time, it is not recommended for the preservation of the gears or when the vehicle is used on the freeway a higher gear might be needed. Currently neither of the vehicles have a working tachometer; this means that the driver has to guess what engine speed they are currently at, how long to leave the gear in neutral before trying to slip into a higher gear or revving the engine by an unknown amount and hoping that the revs will match for a lower gear. Rev matching, matching the engine speed to the speed of the rotating gears while the vehicle is still in motion, is difficult for some and if not done correctly it can damage the gears. The electric motor also takes a longer time for the engine speed to decrease compared to a normal internal combustion engine, this difference requires more attention from the driver in order to make a smooth gear change, the lack of a tachometer makes it that much harder.

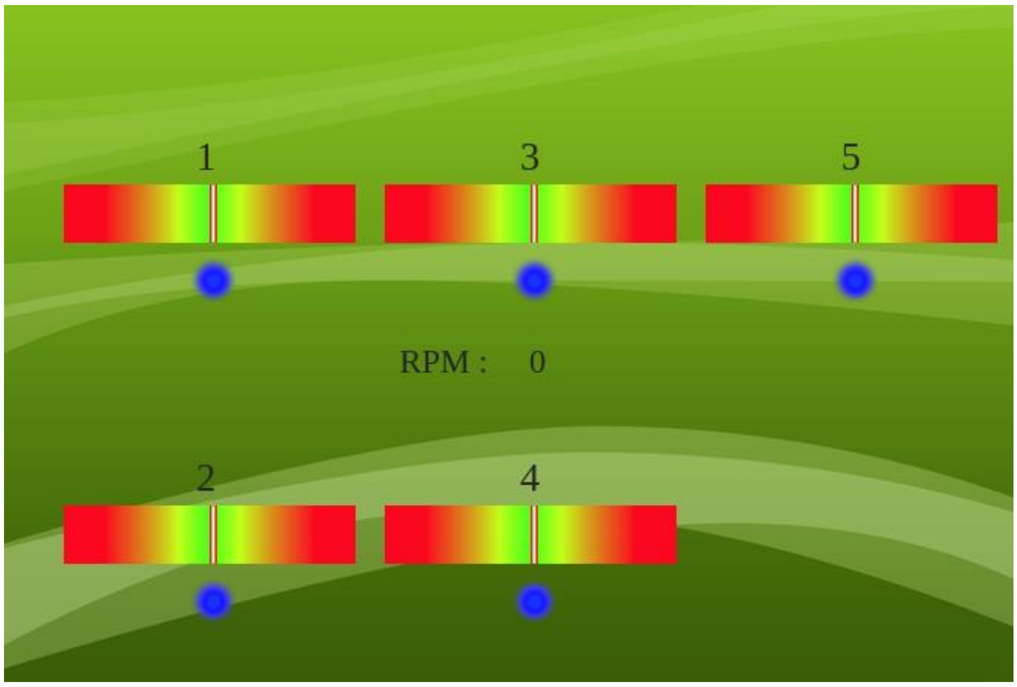


Figure 5.10 Gears Page

Figure 5.10 shows the layout of how the information is displayed to the user. Each gear is designated a bar and the current engine speed is displayed. Within each bar is a line that shows the relationship between the current engine speed and the engine speed required for that specific gear at the current vehicle's speed. The smaller the difference between the required and current engine speed the closer the line is to the center of the bar, which graduates in colour from green in the middle to red on either end. When the difference is within 5% a blue circle is shown next to the gear to prompt the driver that that gear is available. This was designed such that detailed numerical information is not given to the driver, which is not necessary for gear changing, to reduce distractions and glance time while still aiding the driver in gear changes.

6. IMPLEMENTATION

6.1. POWERMATE DEVICE

In order to use the Powermate device with the GUI, a driver, Gizmo Daemon (Gizmo Daemon n.d.), is used to map the Powermate's inputs to keyboard key presses. The driver utilises python scripts (stored in `$HOME/.gizmod/modules.d`) that can be customized as required. An example of part of a script is found in Appendix C. Within the script's initialisation function it checks if the application it is interested in, the GUI in this case, is running, otherwise the script has no effect in the mapping of the Powermate's inputs.

The main mapping occurs in the function `onDeviceEvent`, it checks the `Event.Type` to determine if the Powermate was clicked or rotated, `Event.Value` specifies the direction of rotation. It can also be determined if the device was rotated while held down by retrieving the current state of the device. Table 6.a shows the mappings for each type of input. The function `onDeviceEventButtonTimeout` is called if the user clicks and holds down the device for more than the specified duration. The specified duration needed for the device to be held down to timeout is specified in `.gizmod/GizmoDeviceStrings.py`.

Powermate Input	Keyboard Mapping	GUI Action
CCW Rotation	Q and Shift+Tab	Focus on previous item
CW Rotation	W and Tab	Focus on next item
CCW Rotation while held down	Comma	Decrease volume
CW Rotation while held down	Period	Increase volume
Clicked	E and Space	Enter page or click button
Clicked until timeout	Esc	Leave page

Table 6.a Powermate mappings

Within the Qt source code for the GUI, shortcut functions are assigned to those specific keys. The characters 'Q', 'W' and 'E' are used because the "buttons" on the home screen are actually images rather than push buttons, and the style sheet is written such that when the next item is focused the image changes colour. This is a workaround for the fact that if they were normal buttons and the focus was set on it there would be a rectangular highlight over that image, a purely aesthetical reason. Another reason is that the assigned shortcut functions are specifically programmed to emit a signal for a click sound effect discussed later. If a shortcut function were assigned to the “Tab” key, its original function of putting focus on the next button would be lost and not able to be used due to it being a private function call.

6.2. Input/Output

The `io.cpp` source file is designed to read comma-separated value (CSV) messages from a serial port. The serial port would be connected to another device that reads I/O signals from the vehicle and stream an asynchronous serial stream with the ASCII message format “`%Identifier, %Value\n`” explained in Table 6.b.

A limitation of the RPi is that it only has 8 GPIO pins with no over-voltage protection (eLinux n.d.). Therefore it is recommended to use an external board to deal with the multiple digital outputs and analog outputs available from the REV Eco. This is an area of consideration in future.

The digital and analog signals mentioned here already exist in the REV Eco and were connected to the previous controller but not used. Originally, trip time, trip distance, current speed, current altitude, current voltage and current are logged as a CSV file. The I/O information is simply concatenated onto the end of the line following the message format it came in.

Identifier	Value	
CD: Car door	1: Open	0: Closed
B: Seat belt	1: Unbuckled	0: Buckled
HB: Hand brake	1: Engaged	0: Not Engaged
CEL: Check Engine	1: Fault	0: OK
G: Internet	1: Fault	0: OK
SS: Safety switch	1: Engaged	0: Disengaged
FD: Fuel door	1: Open	0: Closed
RPM: Motor RPM	Actual value of motor RPM	
TPS: Throttle position sensor	Percentage of throttle position	
AUX: 12V Battery	Voltage of 12V Battery	

Table 6.b Message format of I/O

6.3. REV ECO CLEAN UP

Hidden behind the center fascia was a mess of wires as mentioned before. This was potentially hazardous to the previous controller or the vehicle and its occupants as it could accidentally cause a short circuit. An unused USB cable and USB splitter were even found within the mess of wires. Clean up of the wires were thus carried out. The switches for the Radio and Heater have LEDs to indicate their status that were not connected; the wires to power the LEDs were found and reconnected. Power to the Internet module under the passenger seat was also restored such that when the ignition is switched off, power to the module is also disconnected. Connecting the module's power line to the vehicle's ignition line, which is 12V when the key is turned to ACC, did this.

Upon removing the EyebotM6 a fair amount of space is freed, in future the I/O device mentioned earlier could be placed here if it is predicted that it wouldn't require regular access. In the place of the EyebotM6's display, a plate of aluminum was mounted with the 7-inch display attached to it.

6.4. Porting the GUI program for the Raspberry Pi

Fortunately it is relatively easy to cross-compile Qt for an ARM device. By mounting an image of the SD card that is to be used, Qt can be configured and compiled onto the card following certain instructions (Qt Project n.d.). Originally the GUI had the functionality of playing music through a media player. However the newer version of Qt that is currently being used removed the phonon library that was used previously to write the code for the media player. At time of writing there has been no suitable solutions for a decent media player on the Raspberry Pi with Qt. Due to this the media player page shows a media table but is non-functional.

The default for Qt is to use the OpenGL ES application programming interface however the touchscreen driver requires the X system to work. The solution to this, which only applies to the REV Racer, is to run the GUI with X. Running the program with the flag "`-platform xcb`" utilises the X system, enabling the touch screen to interact with the GUI. However this can only be done after `startx` has been run.

6.5. Modifying the GUI for the REV Eco

Although one of the reasons for having the same GUI for both vehicles was to have a standard interface, there are some differences between the hardware of the vehicles that require modification to the GUI for the REV Eco. The main differences that affect the GUI are; the resolution of the display screen, the touch screen, the battery management system, regenerative braking and the fuel door sensor.

The display screen in the REV Eco has a native resolution of 668 by 454 pixels and is connected through the RCA video output, this was found through experimentation, as the screen did not come with a descriptive specification sheet. This is significantly lower than the screen in the REV Racer, which requires the size of the GUI dimensions to be changed to suit. The current design for the GUI was designed such that even with a lower resolution the user is able to obtain as much information as quickly as possible without having to reduce the size of fonts or images significantly.

The REV Racer uses the BMS developed by Ivan Neubronner, which allows the GUI to log and display the state of each cell. However, the REV Eco uses the TBS battery monitoring system. The two systems use different ways of communicating the state of the batteries. The REV Racer's BMS sends asynchronous ASCII messages, separated by new line characters, over a serial line using the message format shown in Appendix D, while the TBS sends an asynchronous stream of hexadecimal bytes (TBS Electronics n.d.).

The TBS distinguishes the start and end of each string in the stream by assigning the Most Significant Bit (MSB) of the header and end of transmission (e.o.t.) trailer bytes to '1'. The first two bytes of the section determine the source and the device ID. The third byte represents the type of data for this string, i.e. voltage or current etc. The next three bytes state the value of the data. Since the MSB is reserved for identifying the header and e.o.t. bytes the value is reduced from a possible 24-bit number to a 21-bit number. An example of a string is shown in Table 6.c.

Header	**Source**	**Device ID**	**Type**	**Value**	**Value**	**Value**	**e.o.t**
`0x80`	`0x00`	`0x20`	`0x60`	`0xu1`	`0xu2`	`0xu3`	`0xFF`

Table 6.c Example of a TBS hexadecimal string

The first 6 bits of u1, u2 and u3 are merged to form the 21-bit number, with u1 being the more significant bits and u3 being the lesser. To do this u1 simply needs to be shifted left 14 bits and u2 shifted left 7 bits. This is done using the bitwise shift left operator, "<<".

Bits of u1							**Bits of u2**							**Bits of u3**						
6	5	4	3	2	1	0	6	5	4	3	2	1	0	6	5	4	3	2	1	0

Table 6.d 21-bit value

Some of the types of data use negative numbers; the value is not stored in 2's complement but rather bit<20> determines whether the value is positive, bit<20> is 0, or negative. The value now ranges from 0 to 1048575 only, using bit<0> to bit<15>. The code for translating the three hexadecimal bytes to their proper values is in Appendix E. The logger for the REV Eco is also modified respectively to correspond to the different data from the batteries.

The REV Racer also has the ability of regenerative braking where the REV Eco does not. A simple change here is required, since the current can only be negative for the REV Eco the colour for the current bar on the main screen is limited to a single colour whereas the REV Racer has two, blue for positive current (i.e. regenerative braking is occurring) and green for negative current. The lack of a fuel door on the REV Racer also only requires a simple removal of the fuel door icon from the REV Racer's main screen and removing logging for that status.

6.6. Tachometer for the REVs

While a working tachometer will make the task easier there is still an element of guesswork involved. The data calculations shown in this section is provided to the tachometer page in the GUI. Given the speed of the vehicle from a GPS signal and the radius of the tyre, the wheel's rotational velocity can be determined. Knowing the vehicle's gear ratios the engine's rotational velocity can also be determined. Angular velocity (ω), in rad/min, is related to the linear velocity of the center of the wheel (v), in km/h, and its radius (r), in m, by:

$$\omega = \frac{v * 16.66}{r} \tag{6.6.1}$$

Angular velocity can then be converted from rad/min to revolutions/min, and the engine speed can be calculated for the current gear or the desired x^{th} gear with equation 6.6.2 using Table 6.e (Lotus Cars Ltd n.d.; Hyundai n.d.).

	REV Racer	**REV Eco**
1st Gear	2.92	3.615
2nd Gear	1.75	1.950
3rd Gear	1.31	1.286
4th Gear	1.03	1.061
5th Gear	0.85	0.838
Final Gear	4.2	3.842

Table 6.e REVs' Gear ratios

$$Engine\ speed = \frac{\omega}{2\pi} * x^{th}\ gear * final\ gear \tag{6.6.2}$$

Now that the engine speed at specific gears is known they can be compared to the current engine speed to facilitate rev matching.

The function `update_ui_gears(IOData)` is called when the serial message from the I/O device, mentioned earlier, is parsed. A timer is set in `io.cpp` to parse an incoming message at a rate of 10Hz. IOData is a `struct` that consists the current engine speed as a `double`, and other variables. Using equation 6.6.2 for each gear and the current engine speed a percentage error is found.

```
int errX = (int) (((ioData.RPM – (wspeed * ratioX)) /
        (wspeed*ratioX))*100);
```

The code snippet above calculates the percentage error for gear X, and is done for all gears. As we can see from the code if `wspeed`, the wheel speed based on the vehicle's velocity, is 0 then errX will be infinite which would be useless. If the vehicle's velocity is 0 and the engine speed is 0, the vehicle can technically be put into any gear. This `if` statement simply hides the line in this specific case.

```
if (wspeed <=0) {
        ui->RPMlineX->hide();
}
```

Since the line object is specified to be a child of the bar object, the line co-ordinates are referenced from the bar. Figure 6.1 shows the (0,0) co-ordinate for the line, the bar is specified to be 200 wide and 40 tall and the line is specified to be 5 wide. The line will be at either end of the bar when the percentage error is ±100%. If the error is more or less than that amount the line is hidden. The code snippet below shows the implementation. Thus when the current engine speed matches the required speed of a specific gear the line will be in the middle of the bar corresponding to that gear.

```
if (errX > 100 || errX < -98)
        ui->RPMlineX->hide();
else {
        ui->RPMlineX->show();
        ui->RPMlineX->setGeometry(errX+98,0,5,40);
 }
```

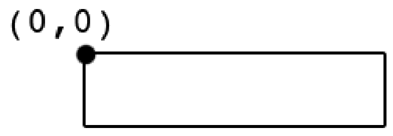


Figure 6.1 (0,0) coordinate of gear bar

6.7. Night Mode

Within the `update_ui_battery(BatteryData)` function, which is called every 5 seconds if a valid battery message is parsed, is an `if` statement which compares the current time to a specified time. The `nightMode` function changes the stylesheet of the objects within the user interface file. In future instead of comparing the time the nightMode function will be called based on the state of a light sensor.

```
void REV_HMI::nightMode(bool night) {
      if(night) {
            ui->centralWidget->setStyleSheet("… …");
            ui->checkBox_nightMode->setChecked(true);
            isNight=true;
            settings.setValue("car/night",true);
      } else {
            ui->centralWidget->setStyleSheet("… …");
            ui->checkBox_nightMode->setChecked(false);
            isNight= true;
            settings.setValue("car/night",false);
   }
}
```

6.8. Implementing the RPi in both vehicles

The RPi is currently attached to the back of the glove box of the REV Eco; this allows relatively quick access to the SD card. Since the RPi requires a 5V power supply it is connected to the 12V ignition line via a 12V-5V DC-DC converter with a 1A inline fuse. By connecting to the ignition line the RPi only powers up when the key is turned to ACC. The REV Racer follows the same set up except the RPi is placed in the passenger foot well. However, it is possible for the RPi to get kicked out of place, although unlikely, as it is currently only held in place with a strip of Velcro and only just out of reach. Ideally, it should be held more securely to the vehicle using the mounting slots provided on the rear of the RPi and the audio cable re-routed to allow the RPi to be placed at a better location.

The conundrum is that the RPi should be placed in a location that provides ease of access while also being out of the occupants' way. This is rather difficult in the REV Racer due to the minimalist interior design of vehicle and the length of the audio cable that was already in. Additionally, the design of the interior makes it very difficult to do any physical work in the foot wells, the person working on the vehicle would have to adopt very weird body positions that could be harmful over an extended period of time.

Figure 6.2 and Figure 6.3 show the placement of the RPi in both the REV Racer and REV Eco. In Figure 6.2 the larger highlighted area shows where the PC was previously. By replacing the PC with the RPi the noise within the cabin was greatly reduced, the noise source originated from the fans cooling down the PC, since the RPi does not require fans the noise source is removed.

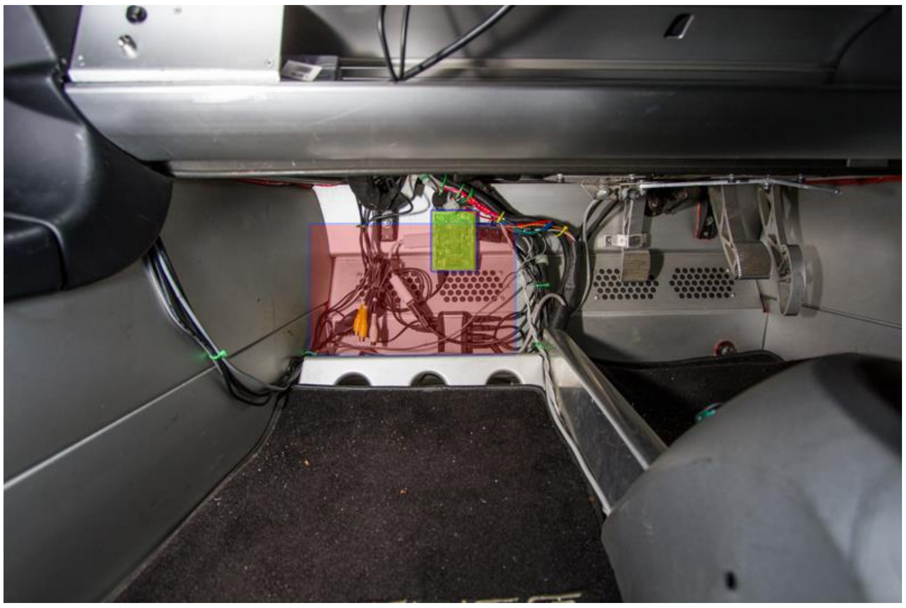


Figure 6.2 Placement of RPi (Racer)

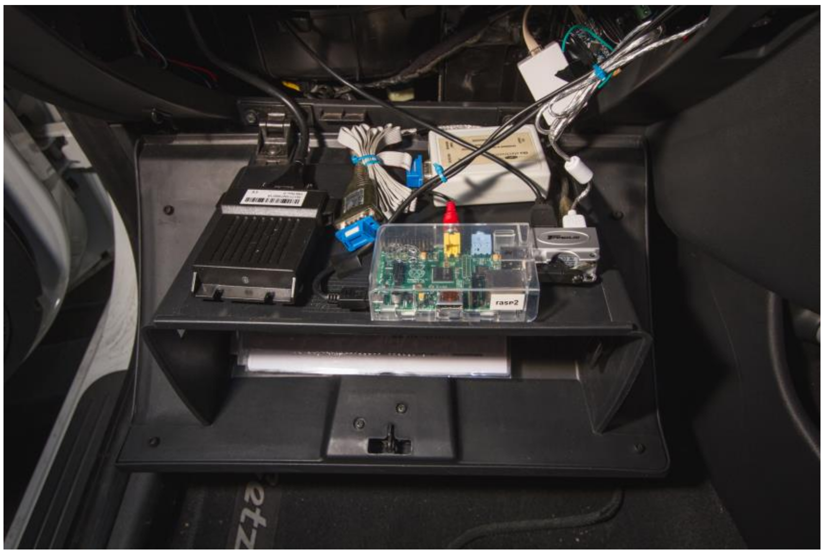


Figure 6.3 Placement of RPi (Eco)

The Powermate is placed in the cup holder in front of the gear selector stick. As mentioned before this location was chosen deliberately. Its USB cable is fed through the hole the TBS cables currently use. Figure 6.4 shows the placement of the Powermate and its cable.

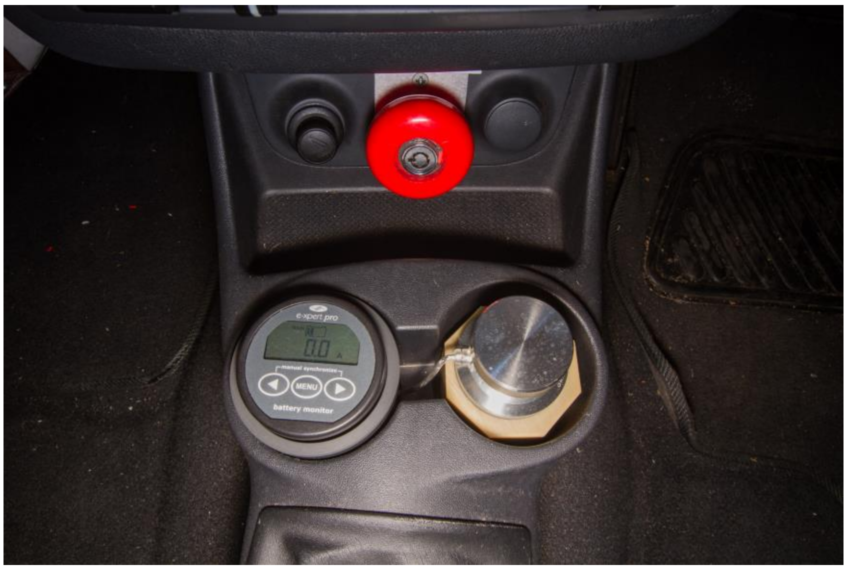


Figure 6.4 Placement of Powermate (Eco)

6.9. <u>RE-IMPLEMENTING THE SYNTHETIC ENGINE SOUND SYSTEM</u>

The synthetic engine sound system, called E.A.R.S in past projects, tries to mimic the noise of an engine by playing tracks of an engine that was pre-recorded at specific frequencies. The aim is to enhance the presence of an electric vehicle that might otherwise be overlooked by pedestrians due to its minimal noise at low speeds. DirectSound however cannot be ported to Linux, and new code had to be written for the RPi. The RPi has a 3.5mm audio output jack that is connected to an amplifier in the REV Racer.

The new code utilises the ALSA library (ALSA n.d.) and is coded in C++, also instead of a separate program and communicating over Transmission Control Protocol like the previous implementation, this will be coded into the main GUI as a child object of the REV_HMI class. Within the class, called `engine_port`, the main loop that plays the sounds is an infinite loop. So as not to degrade the quality of interaction with the GUI the class is run as a separate thread. Within this class will also house methods to play the click and volume check sound effects.

Upon starting up the GUI the `engine_port` class is initialized. During initialisation a Pulse Code Modulation (PCM) device is opened for playback and allocated with default parameters. Some parameters are then specifically set; in this case it is set for 2 channels and a sample rate of 44100Hz. Using `snd_pcm_hw_params_get_period_size()` to get the number of frames for a period, the size of a buffer required can be found, which is an important variable for reading the audio file.

```
buff_size = frames * channels * 2 /* 2 -> sample size */
```

When the user clicks the “Engine Simu” button located in the Log page, the function `runSimu()` is called, which consists mainly of an infinite loop. At the start of the loop it checks if the volume has changed since the last iteration and sets the PCM device to the corresponding volume level. Next, the current engine speed of the vehicle is queried and checked if it has changed since the last iteration and the corresponding audio file is opened using the Qt library functions provided. The audio file is then read and a buffer of `buff_size` is stored into the variable `buff`. A tally is kept to check if the audio file being read is near its end, it is “rewound” if it is and the tally reset. Using `snd_pcm_writei()`, `buff` is written to the PCM device with checks for write failures. At the end of the iteration a check is run to see if the user has clicked the “Stop Engine Simu” button and breaks out of the loop if so. Upon leaving the loop the memory allocated to the buffers are freed and the audio files are closed. Figure 6.5 shows an overview flow diagram.

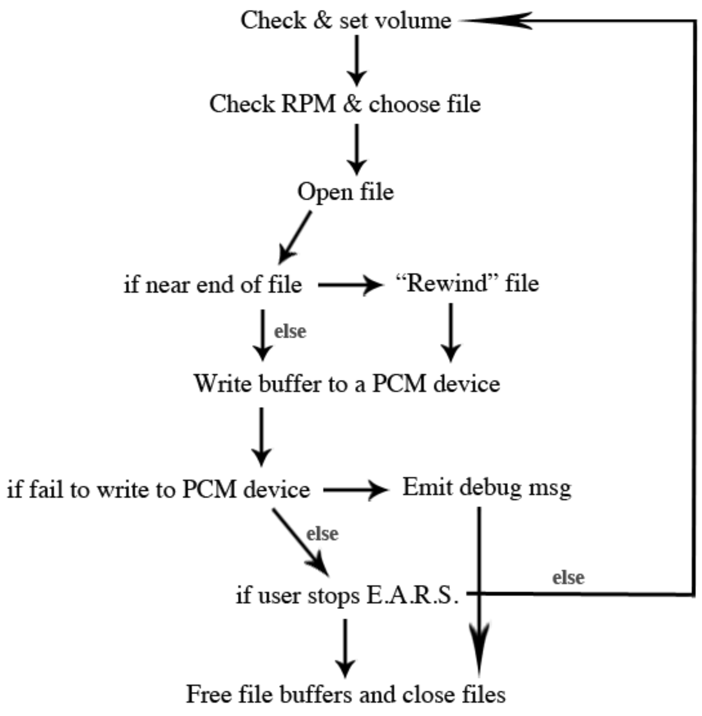


Figure 6.5 Flow diagram of simulation playback

At the moment since there is no device for reading the actual engine speed from the motor controller the engine speed is estimated from the current draw from the batteries. A relatively linear relationship between current draw and the engine speed is assumed. The maximum engine speed, while the current is below 130 amps, is 3500RPM at 150 amps, whereas the maximum engine speed, while the current is above 130 amps is 8000RPM at 200 amps. The reason for this is that the REV Racer is configured to draw a maximum of 200 amps (Tyler 2011, p.20). The first linear relationship below 130 amps assumes normal acceleration, while the second linear relationship above 150 amps assumes heavy acceleration and thus weights the gradient as such. Once the actual engine speed information is available this estimation would not be required.

There was a major drawback while converting the original code from the DirectSound library to the ALSA library. Mainly, it greatly simplifies the feature of the system. Previously the system would shift the frequency of the audio file being played in order to mimic a smooth increase in engine speed, this is required since the audio files are in intervals of 100 RPM. That means a single 3000 RPM audio file could be used between the range of 2950 to 3049 RPM. In this version the system only plays the audio files as is without modifying the frequency. Being very discontinuous and not at all realistic greatly degrades the system.

The click effect was done by calling `emit click()` at the functions `setInfo()` and `enterPage()`, which are called in `rev_hmi.cpp`, when the device is rotated and clicked respectively. Signals and Slots are used in Qt to communicate between objects (Qt Project n.d.). The signal `emit click()` is connected to a slot function in `engine_port.cpp`, the function opens a `click.wav` file located in the `Sounds` folder and plays it using the ALSA library; the method for playing the sound is similar to the engine sound simulation, except it does not require checks for volume since it does not run in an infinite loop. Similarly when the device is held down and rotated to change the volume a signal will be emitted which is connected to a slot that opens `vol.wav` and an effect is played back for the user to judge the sound level.

7. FUTURE WORK

Moving the GUI of both vehicles to a new platform requires a lot of work, not all can be completed within this project. Listed here will be some tasks that should be considered in future.

The removal of the EyebotM6 removed the extensive I/O functionality attached to that system. A new I/O device will be needed to communicate with the RPi, following the message format stated above in Section 6.2. Additionally the device could be used to control servos or motors to control settings such as the vehicle's air-conditioning. Testing can be done on objectively measuring the time it takes for the user to interact with the current system in both vehicles to determine if more changes needs to be made in order to increase safety. Research can also be done to determine if it is viable to remove the driver's interaction with any GUI in the vehicle while in motion, whether it will contribute significantly to safety or whether it is not viable from a marketing standpoint.

With the change to a new version of Qt the media player capabilities of the GUI was removed, in future there may be patches to the library that allows the media player to be re-added to the functionality. The synthetic engine sound system also requires a smooth transition between RPM changes; frequency shifting using the ALSA library is crucial for this effect. Currently the RPi's power is cut when the driver removes the key from the iginition, having no shutdown procedure can be harmful to the SD card. The I/O device mentioned before can detect when ignition line goes to 0V and notify the RPi to shutdown. However, a large enough capacitor is still needed to ensure that the RPi has enough power to finish the shutdown procedure. This should be implemented in future.

Lastly the display screen in the REV Racer needs to be moved to the second ideal location discussed in Section 4. This requires creating new holes in the front panel and swapping the current position of the display and the fan controls. This would allow the driver to use the touch screen without having to lift their shoulder from the seat.

8. CONCLUSION

Over the year various work has been done on both vehicles to update and improve their systems and ensuring they are ready to be used on the roads. The groundwork has been laid for future studies and work to be completed on the new system. New features have been implemented and older ones improved.The GUI system was successfully shifted to a new system, the Raspberry Pi, and the GUI from the REV Racer has been modified to suit the differences of the REV Eco and its input device, the Powermate. The Powermate was implemented in a way that feedback is provided to the user by using sound effects. Changes have also been made to improve the visibility of the interface in the day and at night.

Unfortunately the manufacturers of the motor controller used in the REV Racer would not provide the circuit diagram, which is required in order to attain the engine speed from the motor's Hall effect sensors. Also access to the REV Eco was not available as it underwent repairs. Thus the tachometer in the instrument clusters still does not work. As a result engine speed is only estimated, which affects the new functionality such as a driver's aide for changing gears without a clutch.

The receiving end of an Input/Output pair has been implemented within the GUI, and now only requires a device to measure the states of the vehicle and to send that data to the GUI. The engine sound system has also been moved to the new system with some drawbacks associated, in which more work needs to be done to make it fully functional.

Finally, the RPi was installed into both vehicles and configured such that both vehicles are able to utilise the GUI as they were designed. The current layout of the instruments within the interior of the vehicles has been studied for their ergonomics, safety and ease of use. Recommendations have been made on the placement of the display and for the mounting of the RPi for the REV Racer.

Overall, nearly all the objectives have been successfully met, with the exception of the synthetic engine sound system and independently working tachometer. However, with a little more work the HMI would be ready for new features to be added by future students.

9. Bibliography

ALSA, ALSA Library Reference. Available at: http://www.alsa-project.org/alsa-doc/alsa-lib/index.html [Accessed October 3, 2013a].

ALSA, ALSA Project Homepage. Available at: www.alsa-project.org [Accessed October 3, 2013b].

Braunl, T., 2012. Synthetic engine noise generation for improving electric vehicle safety. *International Journal of Vehicle Safety*, 6(1), pp.1–8.

Braunl, T., The REV Project. *The REV Project*. Available at: http://therevproject.com/ [Accessed May 18, 2013].

Department of Transportation, 2013. Federal Motor Vehicle Safety Standards; Minimum Sound Requirements for Hybrid and Electric Vehicles. *Federal Register*, 78(9).

Deverell, L., 2009. *Orientation and mobility methods: techniques for independent travel*, Kew, Vic.: Guide Dogs Victoria.

eLinux, RPi GPIO Wiki. Available at: http://elinux.org/RPi_Tutorial_Easy_GPIO_Hardware_%26_Software [Accessed October 3, 2013].

Gabriel Feng, 2013. *Raspberry Pi*, Available at: http://www.flickr.com/photos/gabfeng/10111299414/.

Gizmo Daemon, Gizmo Daemon. Available at: gizmod.sourceforge.net/index.html [Accessed October 3, 2013].

Griffin, Powermate. *Powermate*. Available at: store.griffintechnology.com/powermate [Accessed October 3, 2013].

Heiner Bubb, 2012. Information Ergonomics. In Michael Stein & Peter Sandl, eds. *Information Ergonomics: A theoretical approach and practical experience in transportation*. Springer, p. 266.

Hyundai, Hyundai Getz Shop Manual.

Klaus Bengler et al., 2012. Automotive. In Michael Stein & Peter Sandl, eds. *Information Ergonomics: A theoretical approach and practical experience in transportation*. Springer, p. 266.

Lotus Cars Ltd, Service Notes Elise 2001 Model Year Onwards.

Niedermaier, B. et al., 2009. The New BMW iDrive – Applied Processes and Methods to Assure High Usability. In V. G. Duffy, ed. *Digital Human Modeling*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 443–452.

Qt Project, Beginner’s guide to cross-compile Qt5 on RaspberryPi. Available at: http://qt-project.org/wiki/RaspberryPi_Beginners_guide [Accessed October 3, 2013a].

Qt Project, Download Qt, the cross-platform application framework | Qt Project. Available at: http://qt-project.org/downloads [Accessed May 19, 2013b].

Qt Project, QtCore 5.0: Signals & Slots. Available at: http://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html [Accessed October 3, 2013c].

Raspberry Pi Foundation, About Raspberry Pi. *Raspberry Pi*. Available at: www.raspberrypi.org/about [Accessed October 3, 2013a].

Raspberry Pi Foundation, Raspberry Pi. *Home Page*. Available at: http://www.raspberrypi.org/ [Accessed October 3, 2013b].

Raspberry Pi Foundation, Raspberry Pi FAQs. *Raspberry Pi FAQs*. Available at: http://www.raspberrypi.org/faqs [Accessed October 3, 2013c].

Regan, M.A., Lee, J.D. & Young, K.L. eds., 2009. *Driver distraction: theory, effects, and mitigation*, Boca Raton: CRC Press/Taylor & Francis Group.

Sandberg, U., 2012. Adding noise to quiet electric and hybrid vehicles: An electric issue. *Acoustics Australia*, 40(3), pp.211–220.

TBS Electronics, E-Xpert Accessory Kits. Available at: www.tbs-electronics.nl/products_batmons_acc.htm [Accessed May 19, 2013a].

TBS Electronics, E-xpert Pro Communication Interface Specification.

Trepp, B., 2011. *Design of an embedded data acquisition and display system using a modular approach*. Thesis. Perth, Western Australia: University Of Western Australia.

Tyler, M., 2011. *REV Performance Vehicle Instrumentation*. Thesis. Perth, Western Australia: University Of Western Australia.

Varma, D., 2009. *Renewable Energy Vehicle Instrumentation: Graphical User Interface and Black Box*. Thesis. Perth, Western Australia: University Of Western Australia.

Walter, T., 2010. *Development of a User Interface for Electric Cars*. Thesis. Perth, Western Australia: University Of Western Australia.

Xenarc Technologies, Xenarc Technologies - 700TSV - 7" Touchscreen TFT LCD Monitor with VGA and AV inputs. Available at: http://www.xenarc.com/product/700ts.html [Accessed May 19, 2013].

10. Appendix

10.1. Appendix A

```
#include <unistd.h> /* access()? */
#include <stdlib.h> /* malloc() */
#include <stdio.h> /* sprintf() */
#include <stdbool.h>
#include "../libM6adc/libM6adc.h"
#include "../libM6hdt/libM6hdt.h"


bool current FALSE;
bool prev FALSE;
int adcsample;

void turnOff() {
    if(system("poweroff") == -1)
        printf("Poweroff failed!\n");
    prev = FALSE;
}

int main(int argc, char *argv[])
{
    ADCHandle ig_handle;

    ig_handle = OSInitADC("ADC0"); //Should be for 144V battery
    adcsample = OSGetADC(ig_handle);

    while () {
        if (!current && prev) {
            turnOff();
        }

        adcsample = OSGetADC(ig_handle);

        if(adcsample == 0 && prev) {
            OSWait(100); //Is it still off after 1 sec?
            adcsample = OSGetADC(ig_handle);
            if(adcsample == 0)
                current = FALSE;
        }

        if(adcsample != 0) {
            prev = TRUE;
            current = TRUE;
        }

        OSWait(100);
    }

    OSADCRelease(ig_handle);
    return 0;
}
```

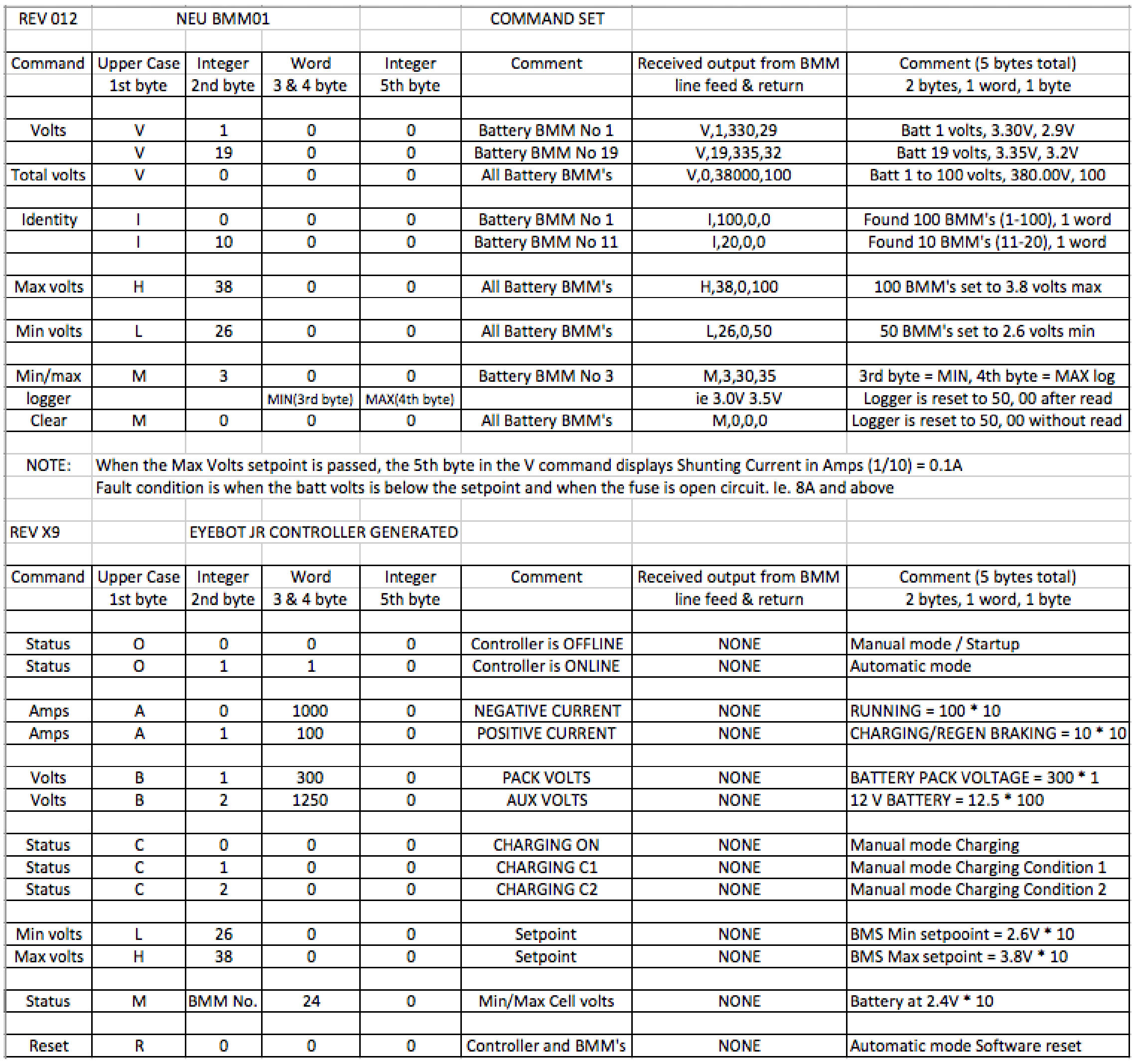
10.2. APPENDIX B

```
bool Logger::checkDirSize(char* logDir) {
  strcat (logDir,"/");
  DIR *dir;
  struct dirent* ent;
  struct stat st;
  char buf[PATH_MAX];
  off_t total = 0LL;

  if((dir = opendir(logDir)) == NULL) {
        emit debugMsg(QtWarningMsg, objName, "Checking directory
        size failed, couldn't open dir.");
        return false;
  }
  while((ent = readdir(dir) != NULL) {
        if(!strcmp(ent->d_name, ".") || !strcmp(ent->d_name,".."))
              continue;
        sprint(buf,"%s%s",logDir,ent->d_name);
        if(stat(buf,&st) == -1) {
              emit debugMsg(QtWarningMsg, objName, "Error checking
              file stat");
              printf("Couldn't
              stat %s: %s\n",buf,strerror(errno));
              continue;
        }
        if(S_ISREG(st.st_mode)) {
              total+=st.st_size;
              if(total/1024 > 100000) {
                    emit      debugMsg(QtWarningMsg,      objName,
                    "Directory  too  large,  delete  oldest  log
                    files.");
                    return false;
              }
        }
  }
  if(total/1024 > 90000) {
        emit debugMsg(QtWarningMsg, objName, "Directory reaching
        file constraint, backup and remove older files.");
  }
  return true;
}
```

10.3. Appendix C

```
def onDeviceEvent(self, Event, Gizmo = None):
        #Ensure Powermate is connected
        if len(Gizmod.Powermates) < 1:
                print "No Powermate connected"
                return False
        if Event.Type == GizmoEventType.EV_REL:
                if not Gizmo.getKeyState(GizmoKey.BTN_0):
                        if Event.Value < 0:
                                Gizmod.Keyboards[0].createEventPress
                                        (GizmoEventType.EV_KEY, GizmoKey.KEY_Q)
                                Gizmod.Keyboards[0].createEventPress
                                        (GizmoEventType.EV_KEY,
                                        GizmoKey.KEY_BACKSPACE)
                                Gizmod.Keyboards[0].createEventRaw
                                        (GizmoEventType.EV_KEY,
                                GizmoKey.KEY_LEFTSHIFT, 1)
                                Gizmod.Keyboards[0].createEventPress
                                        (GizmoEventType.EV_KEY, GizmoKey.KEY_TAB)
                                Gizmod.Keyboards[0].createEventRaw
                                        (GizmoEventType.EV_KEY,
                                GizmoKey.KEY_LEFTSHIFT, 0)
                        else:
                                Gizmod.Keyboards[0].createEventPress
                                        (GizmoEventType.EV_KEY, GizmoKey.KEY_W)
                                Gizmod.Keyboards[0].createEventPress
                                        (GizmoEventType.EV_KEY,
                                GizmoKey.KEY_BACKSPACE)
                                Gizmod.Keyboards[0].createEventPress
                                        (GizmoEventType.EV_KEY, GizmoKey.KEY_TAB)
                else:
                        if Event.Value < 0:
                                Gizmod.Keyboards[0].createEventPress
                                        (GizmoEventType.EV_KEY, GizmoKey.KEY_COMMA)
                                Gizmod.Keyboards[0].createEventPress
                                        (GizmoEventType.EV_KEY,
                                GizmoKey.KEY_BACKSPACE)
                        else:
                                Gizmod.Keyboards[0].createEventPress
                                        (GizmoEventType.EV_KEY, GizmoKey.KEY_DOT)
                                Gizmod.Keyboards[0].createEventPress
                                        (GizmoEventType.EV_KEY,GizmoKey.KEY_BACKSPACE
                                )
                        return True
        elif Event.Type == GizmoEventType.EV_KEY:
                if Event.Value == 0 and not Gizmo.Rotated:
                        Gizmod.Keyboards[0].createEventPress
                                (GizmoEventType.EV_KEY, GizmoKey.KEY_E)
                        Gizmod.Keyboards[0].createEventPress
                                (GizmoEventType.EV_KEY, GizmoKey.KEY_BACKSPACE)
                        Gizmod.Keyboards[0].createEventPress
                                (GizmoEventType.EV_KEY, GizmoKey.KEY_SPACE)
                        Gizmod.Keyboards[0].createEventPress
                                (GizmoEventType.EV_KEY, GizmoKey.KEY_BACKSPACE)
                        return True
def onDeviceEventButtonTimeout(self, Gizmo):
"""
Called when a Powermate's button times out

This is generated from 200-Powermate-ButtonTimeout.py
"""
        #Ensure Powermate is connected
        if len(Gizmod.Powermates) < 1:
                print "No Powermate connected"
                return False
        Gizmod.Keyboards[0].createEventPress(GizmoEventType.EV_KEY,
GizmoKey.KEY_ESC)
        return True
```

10.4. Appendix D



REV 012	NEU BMM01				COMMAND SET		
Command	Upper Case 1st byte	Integer 2nd byte	Word 3 & 4 byte	Integer 5th byte	Comment	Received output from BMM line feed & return	Comment (5 bytes total) 2 bytes, 1 word, 1 byte
Volts	V	1	0	0	Battery BMM No 1	V,1,330,29	Batt 1 volts, 3.30V, 2.9V
	V	19	0	0	Battery BMM No 19	V,19,335,32	Batt 19 volts, 3.35V, 3.2V
Total volts	V	0	0	0	All Battery BMM's	V,0,38000,100	Batt 1 to 100 volts, 380.00V, 100
Identity	I	0	0	0	Battery BMM No 1	I,100,0,0	Found 100 BMM's (1-100), 1 word
	I	10	0	0	Battery BMM No 11	I,20,0,0	Found 10 BMM's (11-20), 1 word
Max volts	H	38	0	0	All Battery BMM's	H,38,0,100	100 BMM's set to 3.8 volts max
Min volts	L	26	0	0	All Battery BMM's	L,26,0,50	50 BMM's set to 2.6 volts min
Min/max	M	3	0	0	Battery BMM No 3	M,3,30,35	3rd byte = MIN, 4th byte = MAX log
logger			MIN(3rd byte)	MAX(4th byte)		ie 3.0V 3.5V	Logger is reset to 50, 00 after read
Clear	M	0	0	0	All Battery BMM's	M,0,0,0	Logger is reset to 50, 00 without read
NOTE:	When the Max Volts setpoint is passed, the 5th byte in the V command displays Shunting Current in Amps (1/10) = 0.1A						
	Fault condition is when the batt volts is below the setpoint and when the fuse is open circuit. Ie. 8A and above						
REV X9		EYEBOT JR CONTROLLER GENERATED					
Command	Upper Case 1st byte	Integer 2nd byte	Word 3 & 4 byte	Integer 5th byte	Comment	Received output from BMM line feed & return	Comment (5 bytes total) 2 bytes, 1 word, 1 byte
Status	O	0	0	0	Controller is OFFLINE	NONE	Manual mode / Startup
Status	O	1	1	0	Controller is ONLINE	NONE	Automatic mode
Amps	A	0	1000	0	NEGATIVE CURRENT	NONE	RUNNING = 100 * 10
Amps	A	1	100	0	POSITIVE CURRENT	NONE	CHARGING/REGEN BRAKING = 10 * 10
Volts	B	1	300	0	PACK VOLTS	NONE	BATTERY PACK VOLTAGE = 300 * 1
Volts	B	2	1250	0	AUX VOLTS	NONE	12 V BATTERY = 12.5 * 100
Status	C	0	0	0	CHARGING ON	NONE	Manual mode Charging
Status	C	1	0	0	CHARGING C1	NONE	Manual mode Charging Condition 1
Status	C	2	0	0	CHARGING C2	NONE	Manual mode Charging Condition 2
Min volts	L	26	0	0	Setpoint	NONE	BMS Min setpooint = 2.6V * 10
Max volts	H	38	0	0	Setpoint	NONE	BMS Max setpoint = 3.8V * 10
Status	M	BMM No.	24	0	Min/Max Cell volts	NONE	Battery at 2.4V * 10
Reset	R	0	0	0	Controller and BMM's	NONE	Automatic mode Software reset

(Tyler 2011, p.60)

10.5. APPENDIX E

```
bool TBS::parseMsg(QByteArray tbsArray) {
    bool ok;
    QByteArray hexArray = tbsArray.toHex();
    if(rawLogging)
        emit debugMsg(QtDebugMsg, devName, hexArray.data());

    while(!hexArray.isEmpty()) {
        hexArray.remove(0,6); // remove first three
        //msg.remove(0,2); // Get the data type
        QByteArray type = hexArray.mid(0,2);
        hexArray.remove(0,2);
        QByteArray data = hexArray.mid(0,6); // get the data
        hexArray.remove(0,6);
        hexArray.remove(0,2); // remove trailing byte and validate msg
        if(hexArray.isEmpty()) {
            //emit debugMsg(QtDebugMsg, devName, "Valid TBS Message");
            //return false;
        }

        switch (type.toInt(&ok,16)){
        case 0x60: { // Main Pack Voltage
            tbsData.packVoltage =  rawToFloat(data,0);
            } break;
        case 0x61: { // Main Pack Current
            tbsData.packCurrent = rawToFloat(data,1);
            } break;
        case 0x62: { // Amp Hours
           tbsData.ampHours = rawToFloat(data,2);
            } break;
        case 0x64: { // State of Charge
            tbsData.soc = rawToFloat(data,4);
            } break;
        case 0x65: { // Time Remaining
            tbsData.timeRemaining = rawToFloat(data,5);
            } break;
        case 0x66: { // Temperature
            tbsData.temperature = rawToFloat(data,6);
            } break;
        }
    }

    return true;
}
```