THE UNIVERSITY OF WESTERN AUSTRALIA

Final Year Project

Renewable Energy Vehicle Instrumentation: Graphical User Interface and Black Box

Author:

Daksh VARMA

Supervisor:

Prof. Thomas BRÄUNL

October 2009

Abstract

The current energy conservation and climate change discourse is no longer about lip service to a cause out there in the future. It is a here and now issue focused on alternative energy sources for road transport, fuelled in part by depleting fossil fuels, its increasing costs and the need to control vehicular emissions. One of the most viable solutions available is the Electric Car.

A Hyundai Getz has already been converted into a fully electric car as a part of the Renewable Energy project and a conversion is currently being undertaken on a Lotus Elise. The Getz is road legal, and the Elise will soon be. Both cars have performances similar to their petrol-powered counterparts, but with much better fuel economy and with zero emissions.

This thesis focuses on designing Graphical User Interfaces (GUIs) and developing code for the onboard controllers for both the Getz and Elise. The code is designed to read various sensors, process the data and display it though the GUI. In addition, there is a "Black Box" implemented in software to record relevant data gathered from the sensors. The onboard controller for the Getz is the Eyebot Mark 6, which runs Busy Box Linux and the controller for the Elise, is a PC running Windows XP embedded. The code for the Eyebot is written entirely in C, while the PC uses a Flash frontend with a Visual C++ backend. The project also focuses on interfacing the sensor hardware with the onboard controllers.

Letter of Transmittal

Daksh Varma
214 South Terrace
Como 6152
Western Australia

The Dean
Faculty of Engineering, Computing and Mathematics
The University of Western Australia
35 Stirling Highway
Crawley WA 6009

Dear Sir,

I submit to you this dissertation entitled "Renewable Energy Vehicle Instrumentation: Graphical User Interface and Black Box" in partial fulfillment of the requirement of the award of Bachelor of Engineering.

Yours faithfully,

Daksh Varma

Nomenclature

ASCII American Standard Code for Information Interchange

BMS Battery Management System

CAN Controller Area Network

CRC Cyclic Redundancy Check

CSV Comma Separated Values

DHCP Dynamic Host Configuration Protocol

DSP Digital Signal Processor

GPS Global Positioning System

GUI Graphical User Interface

IDC International Data Corporation

IMU Inertial Measurement Unit

IO Input/Output

IP Internet Protocol

LCD Liquid Crystal Display

LED Light Emitting Diode

MB Mega Byte

NMEA National Marine Electronics Association

PCB Printed Circuit Board

PIN Personal Identification Number

PSD Position Sensitive Device

REV Renewable Energy Vehicle

SWF Shock Wave Format

USB Universal Serial Bus

UTC Universal Time Coordinated

XML Extensible Markup Language

Acknowledgments

I would like to thank my supervisor Prof. Thomas Bräunl for his support, guidance and encouragement on this project. I would also like to thank all the members of the REV team I worked with, in particular: Daniel Kingdom, Colin Dickie, Franz Viertler, Jonathan Wan, Cameron Watts, Anne Flinchbaugh and Jurek Malarecki. This project would not have been possible without their support. Special thanks also go to the Workshop, especially Ken Fogden for his help in mounting components in the Lotus. In addition, I would like to thank my family for their patience and support in what has been a time of loss for us.

Contents

1 Introduction **13**

1.1 Description of the problem 14

1.2 Hardware Used 15

1.2.1 Eyebot M6 15

1.2.2 Car PC 16

1.2.3 GPS 17

1.2.4 Accelerometer 18

1.2.5 Motor Controller 19

2 Literature Review **20**

2.1 Usability 20

2.2 Existing Designs 20

2.2.1 Mitsubishi Lancer Evolution 21

2.2.2 BMW Connected Drive 21

2.2.3 iDrive 22

2.2.4 Tesla Roadster 23

2.2.5 Audi Q7 24

2.2.6 Cadillac Hybrid Escalade 25

2.3 Performance Monitoring 26

2.4 GPS 27

I Hyundai Getz **29**

3 Getz: Objectives **30**

4 Black Box and GUI **34**

4.1 Reading Digital Sensors 34

4.2 Writing to File 34

4.3 Code Integration 39

5 Breakout Box **41**

6 General Tasks **46**

6.1 Startup Scripts 46

6.2 Rewiring 47

6.2.1 Switches 47

II Lotus Elise **50**

7 Elise: Objectives **51**

7.1 Design 52

8 GUI **56**

8.1 Main Tab 56

8.2 GPS Tab 58

8.3 Stats Tab 60

8.4 Alerts Tab 61

8.5 Speed Tab 62

8.6 Trip Tab 63

8.7 Music Tab 64

8.8 G-Force Tab 65

9 Program Structure **66**

9.1 Backend 66

9.1.1 Main 67

9.1.2 GPS Thread 67

9.1.3 Accelerometer Thread 70

9.1.4 Hardware Black Box Thread 71

9.1.5 Flash Thread 72

9.1.6 Logging Thread 72

9.2 Frontend 74

9.2.1 Main 74

9.2.2 Event Handlers 75

9.2.3 Update Thread 76

10 Installation **78**

11 Conclusion **79**

11.1 Future Prospects 80

Appendices **84**

List of Figures

1.1 Eyebot Mark 6 15

1.2 Car PC 16

1.3 GPS 17

2.1 Mitsubishi Lancer Evo 21

2.2 BMW Connected Drive 22

2.3 iDrive 23

2.4 Tesla Roadster 24

2.5 Audi Q7 25

2.6 Cadillac Hybrid Escalade 25

2.7 GPS Satellites 27

3.1 Main Screen 30

3.2 Main Screen with Warnings 30

3.3 GPS Screen 31

3.4 Stats Screen 31

3.5 Warnings Screen 31

3.6 Speed Screen 32

4.1 Threads 39

5.1 Eyebot IDC headers 41

5.2 Connectors on Eyebot 42

5.3 Breakout board schematic (24 pin connector) 44

5.4 Breakout board schematic (20 pin connector) 44

5.5 Breakout board layout 45

6.1 Switch circuit 48

8.1 Main Tab 56

8.2 GPS Tab 58

8.3 Area covered by maps 59

8.4 Stats Tab 60

8.5 Alerts Tab 62

8.6 Speed Tab 63

8.7 Trip Tab 64

8.8 Music Tab 64

8.9 G-Force Tab 65

9.1 Backend - Main 67

9.2 Backend - GPS thread 68

9.3 Backend - Accelerometer thread 70

9.4 Backend - Hardware Black Box thread 71

9.5 Backend - Flash thread 72

9.6 Backend - Logging thread 73

9.7 Frontend - Main 74

9.8 Frontend - Setting up event handlers 75

9.9 Frontend - Main button event handler 76

9.10 Frontend - Update thread 77

1 Motor Controller Breakout - Schematic 85

2 Motor Controller Breakout - Layout 86

Chapter1

Introduction

This project was completed as part of the Renewable Energy Vehicle (REV) project. The goal of the REV project is to *"make electric cars that can be charged at ordinary plug points, and produce no pollution. It is planned to produce cars, which are both commercially viable and performance driven."*[1] Currently, both a Hyundai Getz and a Lotus Elise are being worked on as part of this goal. The REV project also aims at combating increasing fuel costs. [1]

The first part of the final year project is an extension on the author's third year project involving designing a Graphical User Interface for an Eyebot Mark 6, which is the onboard controller in the electric Hyundai Getz. The project this year involved integrating Black Box code with the existing GUI and interfacing with various sensors. The second part of the project dealt with the instrumentation for the Lotus Elise. The onboard controller for the Lotus was a PC running Windows XP. The coding here was done partially in Visual C++ in the .NET environment and partially in Action Scripting 3 using Adobe Flash. The author was the team leader of the instrumentation team and was also responsible for allocating tasks to the rest of the team.

This project is particularly relevant in the context of the energy conservation debate and the discourse around climate change. There is an increased urgency the world over to meet the growing threats of depleting fossil fuels, energy security and global warming. There are numerous efforts around the globe to combat this challenge. This project is one small step towards a better, safer, cleaner and a greener planet.

1.1 Description of the problem

Modern electric cars feature a lot of new and unconventional technologies. The REV's Hyundai Getz and Lotus Elise are no exceptions. There is use of Lithium Iron Phosphate batteries, a battery management system, voltage converters, an electric motor, a controller for the motor and various safety circuitry. The use of these technologies means there was a need to monitor their state for:

1. Checking that they are performing as they should be and alerting the driver about any errors present.
2. Using the data gathered to make further improvements.

In particular, the driver needs to be aware of the state of charge of the batteries so he knows how much longer he may drive.

All the information gathered needs to be displayed graphically to the driver as well as logged. The GUI needs to be easy to use while driving and at the same time present a multitude of data.

The use of an embedded controller in the Getz and a PC in the Elise meant that there was an opportunity to provide the driver with several extra features. Though not absolutely essential for driving, these were handy as they provided the driver with a one stop solution for all his needs.

1.2 Hardware Used

1.2.1 Eyebot M6

The Eyebot Mark 6 (shown in Figure 1.1) is an embedded controller developed and built at UWA. It is used in the Hyundai Getz to display and log data.

Figure 1.1: Eyebot Mark 6

Image from [2]

It has a colour LCD touch screen, two serial ports, three USB ports, a speaker, a microphone. Additionally, it can interface with fourteen servos, four motors and six Position Sensing Devices(PSDs). It also has sixteen digital IOs and three analogue inputs.

The Eyebot uses an ARM9 processor and a Xilinx Field Programmable Gate Array (FPGA). The FPGA handles low level tasks while the processor handles higher level tasks [3]. The Eyebot runs a version of BusyBox Linux as its operating system and makes use of Robot Basic Input/Output System (RoBIOS) libraries which were developed at UWA for control of attached hardware.

The Eyebot is a general purpose controller for use in embedded systems and the range of hardware with which it can interface makes it truly versatile.

1.2.2 Car PC

A PC specifically designed for cars was chosen for the Lotus Elise (shown in Figure 1.2). The PC used is the ZOTAC ION ITX D-E. It has a Intel Atom 330 2x1.6Ghz processor with 2GB of DDR2 RAM and a 160GB hard drive. There are 10 USB ports available, these allow interfacing with all the hardware in the car. It has a Ethernet port and built in WLAN. The PC runs Windows XP as this is an architecture independent platform.

Figure 1.2: Car PC

The PC is powered directly by the 12V battery in the car. There is also a 12V digital input for turning the PC on/off. The 12V ignition signal is connected to this input. When the input goes high the PC is turned on after 30 seconds. The signal going low causes the PC to perform a normal shutdown, provided the permanent 12V power is connected. This means the PC turns on when the car is on and turns off when the car is off.

A Xenarc 7 inch LCD touch screen with 800 × 480 pixel resolution has been used for display and interface with the driver.

1.2.3 GPS

The GPS module used in both the Hyundai Getz and the Lotus Elise is the GlobalSat BU-353 (shown in Figure 1.3). It is connected to the car PC/ Eyebot via USB. It is a serial device outputting ASCII strings and has a built in Prolific USB-Serial converter.

Figure 1.3: GPS

Like most GPS units the output strings follow the NMEA0183 standard. Data is transfered at 4800 baud, with 8 data bits, no parity, 1 stop bit and no flow control.[4]. The module outputs three strings terminated by a new line character every second. The format of the string used to extract all the GPS information is:

"$GPRMC,hhmmss,A,llll.ll,N,yyyyy.yy,W,kk.k,tt.t,ddmmyy,mm.m,*CS"[5]
Where:

- "hhmmss is the UTC time
- A is the status: data valid (A) or receiver warning (V)
- llll.ll is latitude (degrees, minutes.m - ddmm.mm)
- N is North or South
- yyyyy.yy is longitude (degrees, minutes.m - dddmm.mm)
- W is West or East
- kk.k is speed over ground in knots
- tt.t is track made good, degrees true
- ddmmyy is the date
- mm.m is magnetic variation, degrees
- *CS is the check sum"[5]

1.2.4 Accelerometer

The Accelerometer used is the SparkFun Electronics SerAccel v5 triple-axis serial Accelerometer. It is a serial device connected though a USB to serial converter. It outputs a string containing the acceleration along the X, Y and Z axis. Data flow is at 9600 Baud, with 8 data bits, no parity, 1 stop bit and no flow control.

1.2.5 Motor Controller

The Motor Controller used for the Elise is the UQM PowerPhase 75 Traction System. The Motor Controller has inputs for throttle, brake, direction control, and it also has a serial and a CAN interface. It provides information about the state of the motor and any errors that may be present though both CAN and Serial. A breakout board was designed to connect to the Motor Controller, the schematic and layout are shown in Appendix A and the pin out of the connector on the Motor Controller is shown in Appendix B.

Chapter2

Literature Review

2.1 Usability

Background research was carried out for existing car Graphical User Interfaces as well as general Usability design principles.

According to Jakob Nielsen [6] Usability is defined by five quality components:-

Learnability: How easy is it for users to learn how to do tasks the first time they use the design?[6]

Efficiency: How quickly and in how many steps can users get tasks done once they have learned how to use to design?[6]

Memorability: After not using the design for a length of time, how easy is it for the user to re-lean how to use it?[6]

Errors: How many errors come about in normal operation, and how easy is it to recover from these errors?[6]

Satisfaction: Is the design pleasing to use?[6]

The endeavor was to incorporate all these aspects of Usability into the design.

2.2 Existing Designs

Various display systems on commercially available cars were looked at for design ideas. These included:-

2.2.1 Mitsubishi Lancer Evolution

Figure 2.1: Mitsubishi Lancer Evo
Image from [7]

As seen in Figure 2.1, the battery remaining is shown graphically and a similar design has been used for the Eyebot GUI. However, it is better to also display the battery remaining as a percentage for increased precision. In addition, it is important to display the mileage and speed.[7]

2.2.2 BMW Connected Drive

BMW is developing a new system - the Connected Drive (shown in Figure 2.2). They plan to provide full Internet access. As a safety measure, the system only operates in the front seats while the car's engine is off, but rear-seat passengers can

surf the Internet even while the car is in motion.[8]

Figure 2.2: BMW Connected Drive
Image from [8]

2.2.3 iDrive

BMW was a pioneer in developing its unified interface - the iDrive (shown in Figure 2.3). This system has been heavily criticized as it was hard to use. It was a lesson in what not to do in the design for this project. In early versions of iDrive, all duplicate controls controlls were removed, this forced the driver to use a single push-button/ joystick/ knob to control the on screen software interface.[9]

While the hardware part of the iDrive worked well, its main problem was the software interface. It lacked usability because of its deficiency in the learnability aspect. *"Users claimed it was often difficult to figure out if you should be turning the knob, moving it like a joystick, or pushing it like a button to select on-screen menu items"*[9]. Because of this problem, many people got frustrated with the iDrive and just didn't want to use it.[9]

As a result of such criticism, BMW later restored some controls that people are used to and added a back button to the iDrive.[9]

Figure 2.3: iDrive
Image from [9]

However, the iDrive design was quite efficient. Once you gained some expertise in using it, it was easy to set destinations in the navigation or make calls through the Bluetooth interface quickly.[9]

2.2.4 Tesla Roadster

Figure 2.4 shows a picture of the user interface on the Tesla Roadster - an electric car based on the Lotus Elise. This screen of the user interface displays the battery remaining, time, distance remaining and a graphic of the car.[10]

The Tesla has a very basic interface. There are different screens for different information about the car - including fuel saved, service history, energy history and charge history. Other screens allow you to change charge settings and change the time on the clock. There is also the possibility to monitor tyre pressure and 'lock' the car using a Personal Identification Number (PIN).[10]

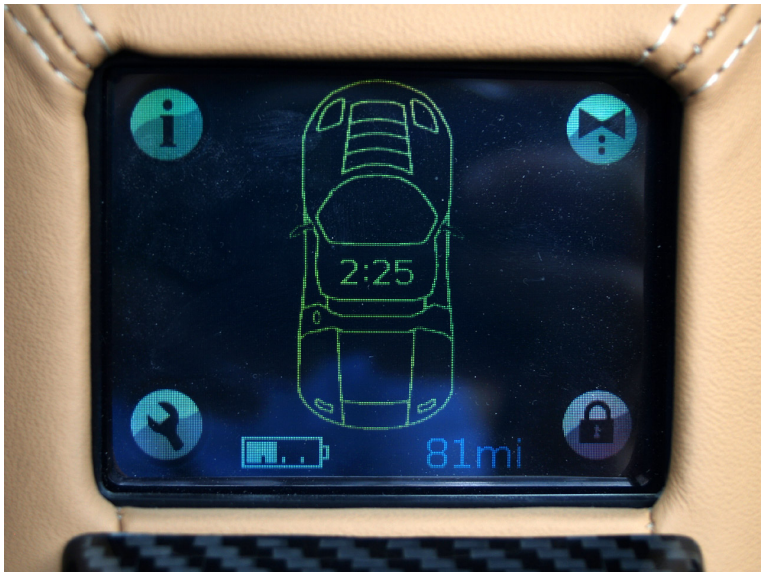

Figure 2.4: Tesla Roadster
Image from [10]

While the Tesla features greater control over hardware than the interfaces on the other cars studied, the interface is clunky and not aesthetically appealing.

2.2.5 Audi Q7

The Audi Q7 GUI (shown in Figure 2.5) provides camera assistance for parking and also has an interface for using mobile phones (connected though Bluetooth).[11]

Figure 2.5: Audi Q7

Image from [11]

2.2.6 Cadillac Hybrid Escalade

Figure 2.6: Cadillac Hybrid Escalade

Image from [12]

The touch button system used in the Escalade (shown in Figure 2.6) is very similar to the one used in the design of this project. While the Escalade put its buttons at the top, the decision was to go for buttons at the bottom of the screen in this design. However, it must be said that the two interfaces provide different functionality.[12]

2.3 Performance Monitoring

"A team from the University of Missouri-Rolla (UMR) have developed a solar powered car that competes in the World Solar Challenge, a 10 day event, in which contenders drive from the North to the South of Australia".[13] A large part of the team's effort goes towards performance monitoring for the car i.e. monitoring voltage and current from the batteries and temperature of various critical components. The data gathered is used for both making improvements and in making strategy decisions pertaining to an on-going race.[13]

There is a trade-off for the team between obtaining highly reliable data, which can help with fault detection, and saving weight and power. As a compromise, they use a low powered Texas Instruments Digital Signal Processor (DSP).[13]

The DSP monitors the electrical systems and computes power consumption. It also sends the data out through a radio modem to a computer in the support vehicle. This computer has more processing power and does the bulk of the number crunching.[13]

The DSP has two eight-channel, ten-bit Analogue to Digital to read the hall-effect sensor to measure current. Additionally, four eight-bit Digital to Analogue converters set the speed of the motor and run the driver's LCD speedometer display.[13]

The issues experienced by the UMR team were very similar to the issues dealt with by the Instrumentation team for the REV project.

2.4 GPS

GPS or Global Positioning System is a U.S. government-owned service. It has three parts: the space deployed part which consists of 24 satellites which transmit one way information about their location, a control system, which controls the position of the satellites and the data that they send out, and lastly a receiver section which uses the signals received from the satellites to triangulate its own position.[14]

GPS receivers as well as GPS satellites, both have clocks which keep track of time. The GPS receiver sends the time of transmission in the data it transmits. Thus, when the receiver gets the data from a satellite it can calculate the travel time of the data as it knows the time the data was received and when it was transmitted. Since the signals are simply electromagnetic waves, the distance from the satellite can be calculated as[15]:

$$\text{d} = \text{c} \times \text{travel time}$$

Now if the distance from a single satellite is known, the position of a receiver could be anywhere on the sphere having radius d with the satellite as the centre. If the distance from two satellites is known, the position of the receiver could be anywhere on the circle that lies on the spheres surrounding two satellites (see Figure 2.7).[16]

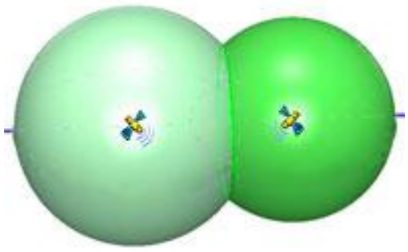

Figure 2.7: GPS Satellites
Image from [16]

If the distance from three satellites is known, then the three spheres intersect at two points. If the distance from four satellites is known, there is only one point of intersection. Thus, a receiver must get a signal from at least four satellites to be able to calculate its position. Further, the greater the number of satellites the receiver gets a signal from, the greater the confidence the receiver has of its calculated position.[16]

Part I

Hyundai Getz

Chapter3

Getz: Objectives

As part of the author's third year project, which was carried out in 2008, a GUI was developed for the Eyebot M6 to display all relevant information about the state of the car to the driver. Some screen shots from this program are shown below:-

Figure 3.1: Main Screen

Figure 3.2: Main Screen with Warnings

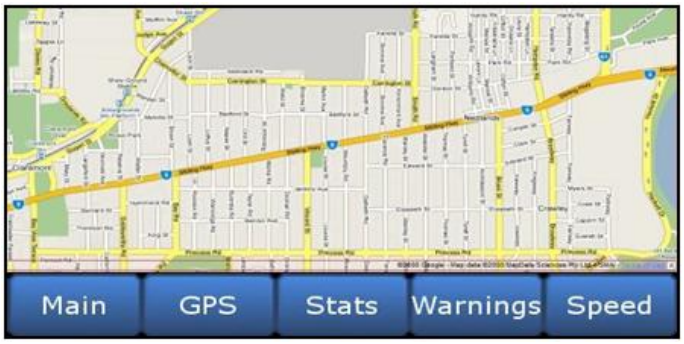

Figure 3.3: GPS Screen

Figure 3.4: Stats Screen

Figure 3.5: Warnings Screen

Figure 3.6: Speed Screen

In order to display all this information, the following data needed to be gathered from the car:

1. Analogue signals
   (a) Voltage of battery
   (b) Current being drawn from battery
   (c) State of charge of battery
2. GPS data
   (a) Exact position of car
   (b) Speed of car
   (c) Heading of the car i.e. exact direction car was moving in
3. Digital signals
   (a) Ignition signal
   (b) Inertia sensor signal
   (c) Door sensor signal
   (d) Seatbelt sensor signal
   (e) Fuel cap sensor signal

(f) Throttle signal

(g) Brake signal

There was “Black Box” code that was written last year by another member of the REV team, which read the digital and GPS signals and logged them to file. However, due to hardware and software changes on the Eyebot, the code to read the digital inputs no longer worked. Thus, there was a need to extend this code.

This Black Box code then needed to be integrated with the GUI, so that the data read from the sensors could be used by the GUI and also be written to file.

Additionally, all the sensors connected to the Eyebot were connected though 9 pin connectors placed behind the Eyebot in the centre console of the car. A system needed to be developed for a more convenient and flexible method of connecting sensors so that they could be easily added, removed and debugged.

Chapter4

Black Box and GUI

4.1 Reading Digital Sensors

Due to hardware changes with the Eyebot, the code to read from the digital sensors no longer worked. Thus, new code to read from the sensors was written.

The steps involved in reading from the sensors were:

1. Initialise latches
2. Initialise each digital IO bank
3. Initialise each digital IO

These steps are illustrated in Program 1 (on the following page).

4.2 Writing to File

The original Black Box software read from the sensors, wrote all the information to standard output, and printed all information onto the LCD screen in an infinite loop, which executed as fast as the hardware would allow. While this was a simple implementation it had the disadvantage that it would fill up the 64 MB of memory on the Eyebot very quickly.

With 76 characters per line in the output CSV file, with 1 Byte per ASCII character, the number of sets of readings was:

$$\frac{64 \times 1024 \times 1024}{76 \times 1} \approx 883011$$

Program 1 Setting up digital IOs

```
OSLatchInit();//initialise all latches
if ( (i=OSLatchBankSetup(IOBANK0, IN)) != 0 ){//set up bank 0
        printf("Set bank 0 error %d\n", i);
        return 0;
}
if ( (i=OSLatchBankSetup(IOBANK1, IN)) != 0 ){//set up bank 1
        printf("Set bank 1 error %d\n", i);
        return 0;
}
for (latch_n=LATCH0; latch_n<=LATCH15; latch_n++){//Each IO
        if(OSLatchSetup(LATCH1, IN)!=0){
                printf("Error with %d",latch_n);
                return 0;
        }
}
```

Although this was a large number of data sets, the Eyebot was writing to file as fast as possible and it was observed that the memory would fill up in about 4.5 hours. This meant that the logs would have to removed from the Eyebot after every 4.5 hours of Eyebot run time, else the system would crash. Ideally, the Eyebot should have had a logging capacity of at least a week.

The solution for this was three-fold:

1. Mount a flash drive for writing log files (see Section 6.1).
2. Do not write to file as fast as possible but write once every second (see Section 4.3 on Code Integration).
3. Log data only when car is driving.

To log data only when the car was running, the ignition signal was used to indicate if the car was on or not. Only when the ignition signal was active was the data written to file. Program 2 highlights this.

Program 2 Writing data to file

```
void writeData()
{
        if(curdat.ignition == 1){
                printf("Writing data\n");
                bb_write();
        }
}
```

It was also found that the original Black Box code would crash after running for several hours. The following problems were found and rectified:

1. The log file was being opened and closed every time sensors were read - this was changed so that the file was opened only when the ignition signal became active, and the file was closed only when the ignition signal became inactive.
2. The buffer on the serial port for the GPS would overflow if the GPS was not read once every second - this was fixed by reinitialising the GPS periodically.
3. Clearing the screen in an infinite loop caused a crash. This was because an instance of LCDClear() could not be called while another instance of LCDClear() was running - this was fixed by only clearing the screen when required.

Program 3 illustrates the above points.

Program 3 Black Box code

```
//if the ignition is on and the gps and not been initialised
if(1 == curdat.ignition && 0 == isGPSOn){
        isGPSOn = 1;//reinitialise the gps
        startGPS(&GPShndl);
}
//stop reading gps if ignition off
else if(0 == curdat.ignition && 1 == isGPSOn){
        isGPSOn = 0;
        close(usbFile);
        SetGPSDefaultValues();
}
else{
        readData();
        if(curdat.ignition == 0 && screenBlank == 0){
                //so second LCDClear() not called while one is running
                sleep(3);
                LCDClear();
                screenBlank = 1;
        }
        else if(curdat.ignition == 1) {screenBlank = 0;}
}
```

4.3 Code Integration

The need to integrate the GUI and Black Box stemmed from the fact that they needed the same inputs - i.e. read from the same sensors. To have two different processes reading from the same digital and analogue sensors, although possible, would be inefficient. Moreover, the GPS could only be read by one process as a serial port can not be opened by multiple processes.

A design decision was taken to write to file once every second. A timer was used to call the write function every second. Timers were also used for the reading and display operations. Thus, there were three different "threads" performing three different functions, but using the same variables in code. The function OSAttachTimer() in the Robios library allowed a function to automatically be called at a specified interval. Three threads and their functions are illustrated in Figure 4.1.

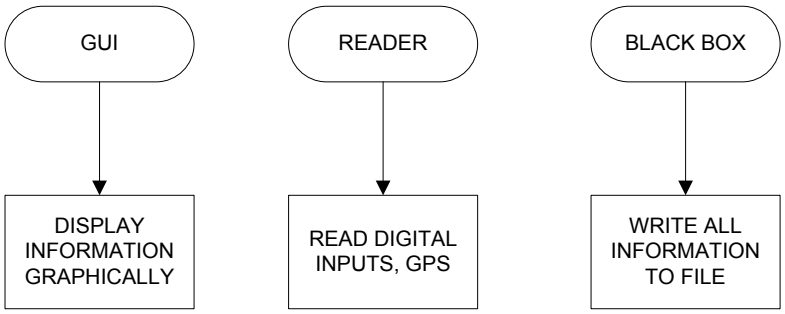

Figure 4.1: Threads

As the three "threads" were operating on the same variables some protection needed to be put into place to prevent variables being read from and written to at the same time. Semaphores were used for this purpose.

With semaphores when one thread is accessing a protected variable or group of variables the variable(s) in use are "locked" and can not be accessed by a different thread. This prevented data from being corrupted. This was especially important as many of the variables used were encapsulated into a structure. The use of semaphores ensured that there was never a situation when some elements of the structure were changed and others were not.

Chapter5

Breakout Box

The Eyebot was mounted in the centre panel of the car. Previously all the inputs to the Eyebot were connected directly to it through 9-Pin connectors. This made it very difficult to check the state of any of the inputs as it involved removing the entire dashboard of the car. It was not possible to change/add/remove any of the inputs as they were permanently soldered onto the 9 pin connectors.

Thus to improve flexibility and accessibility a Breakout Box was designed for the Eyebot and placed next to the steering column in the car.

Figure 5.1: Eyebot IDC headers
Image from [2]

The Eyebot board had several IDC header strips for digital IOs, the analogue to digital converters, Motors, Servos and PSDs as shown in Figure 5.1.

These header strips are connected to the header strip on the PCB via ribbon cables as shown in Figure 5.2

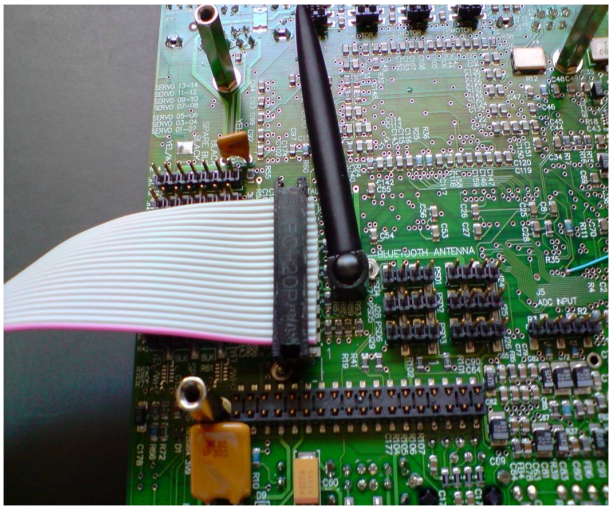

Figure 5.2: Connectors on Eyebot

The header strips for two servos, two motors and the analogue to digital converters, were connected to a 24 pin header on the breakout board, via a 24 pin ribbon cable, with a 7 pin connector for the servos, a 5 pin connector for the Analogue to digital converter and two 6 pin connectors for the motors. The functions of the pins on the 24 pin connector on the breakout PCB are tabulated overleaf.

Further, the 20 pin header with all the 16 digital IOs on the Eyebot is directly connected to the 20 pin header on the breakout PCB. All these inputs to the header strips were broken out into screw terminals, so that inputs could be added/removed or changed as required. The schematic (separated into two parts) and board layout are shown in figures 5.3, 5.4 and 5.5.

Pin	Function
1	5 V
2	Analogue Input 1
3	Analogue Input 2
4	Analogue Input 3
5	Analogue Ground
6	Servo 1 Signal
7	Servo 1 5V
8	Servo 1 Ground
9	Not used
10	Servo 2 Ground
11	Servo 2 5V
12	Servo 2 Signal
13	Motor 1 M- signal
14	Motor 1 M+ signal
15	Motor1 Ground
16	Motor1 5V
17	Motor1 Encoder B signal
18	Motor 1 Encoder A signal
19	Motor 2 M- signal
20	Motor 2 M+ signal
21	Motor2 Ground
22	Motor2 5V
23	Motor2 Encoder B signal
24	Motor2 Encoder B signal

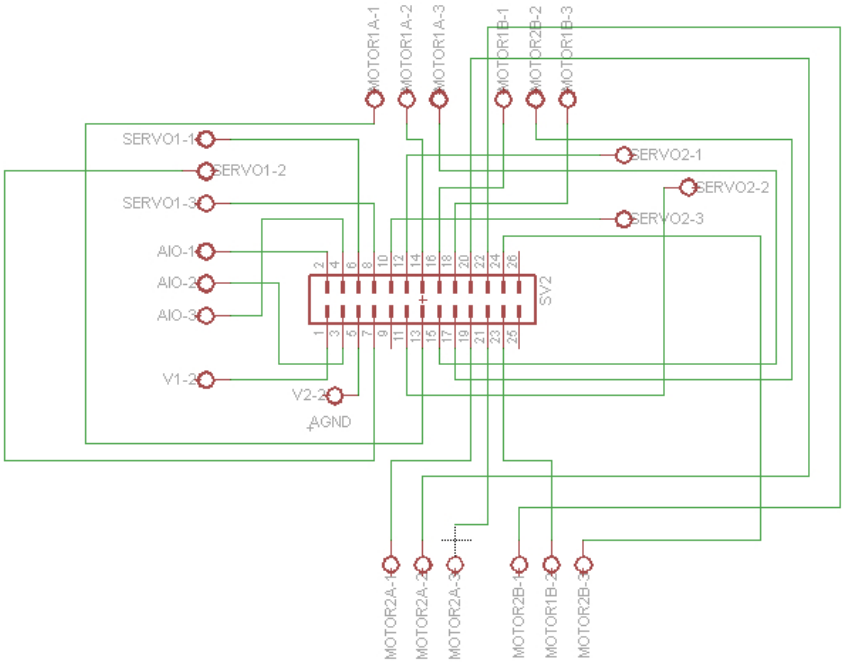

Figure 5.3: Breakout board schematic (24 pin connector)

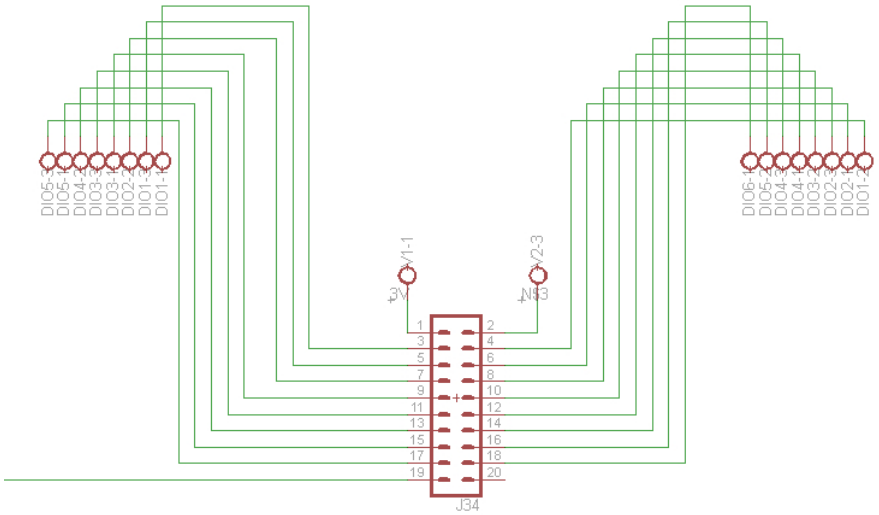

Figure 5.4: Breakout board schematic (20 pin connector)

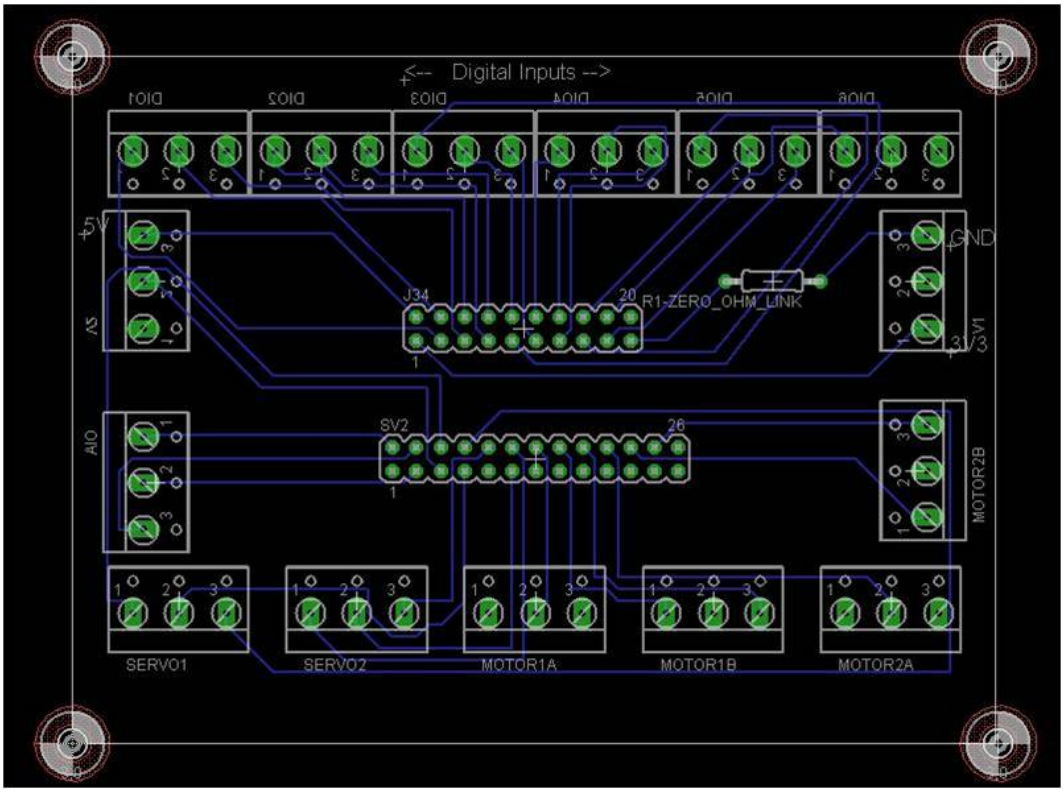

Figure 5.5: Breakout board layout

Chapter6

General Tasks

6.1 Startup Scripts

There are shell scripts on the Eyebot which run at startup. These scripts are named S10Eyebot - S90Eyebot. They are located at /etc/init.d. These scripts were modified to:-

1. Run GUI at startup. This is illustrated in Program 4.

Program 4 Selecting GUI

```
echo "*Starting GUI*"

# start GUI program!

[ -x /root/demo/gui ] && /root/demo/gui&
```

2. Set up a static IP address for the Eyebot. This is illustrated in Program 5.

Program 5 Setting static IP

```
#setup network

ifconfig eth0 192.168.1.101 netmask 255.255.255.0

hw ether 00:11:22:33:44:59 Up

#udhcpc -i eth0 &
```

As seen, the Dynamic Host Configuration Protocol (DHCP) is disabled.

3. Mount a flash drive for used by the Black Box code. This is illustrated by Program 6.

Program 6 Mounting flash drive

```
echo "mounting usb stick"

mount /dev/sda1 /media/bb
```

The Black Box code can write to the flash drive by writing to /media/bb as it would to any other folder.

6.2 Rewiring

The wiring behind the centre panel of the car was redone. Multi core cables were used instead of solid core cables as they are less prone to breaking due to the vibration experienced in a moving vehicle. A cable was run to connect all the outputs from the relay board in the engine compartment to the Eyebot Breakout Board.

Additionally, the 12V wiring to power the Eyebot, wireless router and Inertial Measurement Unit (IMU) was redone.

6.2.1 Switches

There were two switches present next to the steering wheel which turned the stereo and the heater on and off. These switches had built in LEDs that came on when the stereo/heater was switched on. Although the switches were wired correctly to operate the stereo and the heater, the LEDs did not come on. This problem was resolved as shown in Figure 6.1.

The 12V ignition signal is high (12 V) when the ignition is on and low (0 V) when the ignition is off. The 12V line is used as a control signal for turning the heater and stereo on and off .

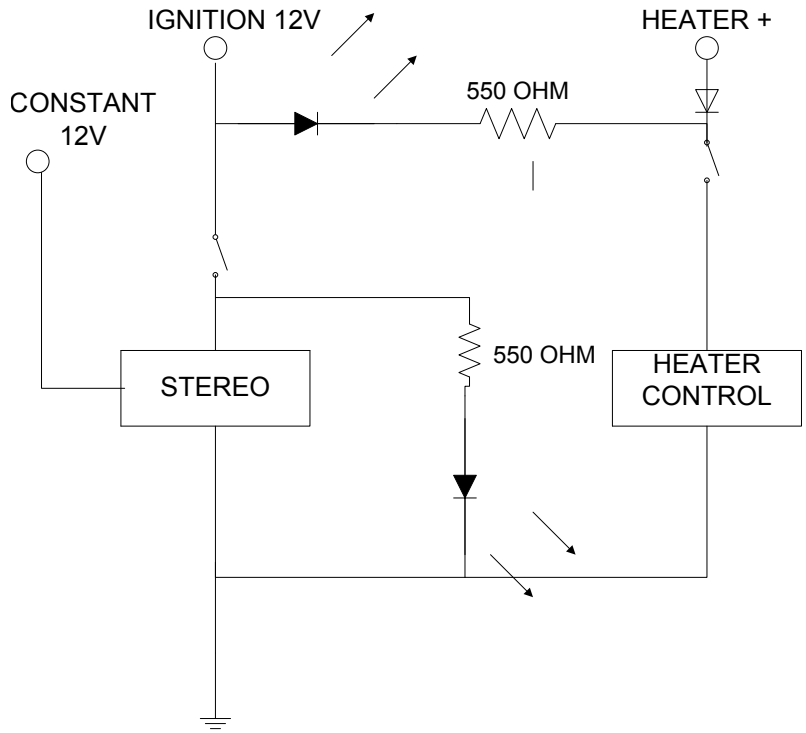

Figure 6.1: Switch circuit

The heater+ signal is at 12 V when the heater heater switch is open, but is pulled down to 0 V when the switch is closed. See [17] for further details on the heater control system.

The LEDs both go from 12V to ground with a series resister in between. The current rating for the LEDs used is 20mA and the voltage drop across them when forward biased is about 2V. Thus, the series resister needed to be:

$$R = \frac{V}{i}$$

$$= \frac{12 - 2}{.02}$$

$$= 500\Omega$$

To limit the current to slightly below 20mA, the resistance used was 550 Ω.

Closing the stereo switch forward biased the stereo LED, thereby turning it on. Closing the heater switch brought the heater+ line to ground, thereby forward biasing and turning on the heater LED. It was observed that disconnecting the heater had the effect of bringing the heater+ line to ground. Thus, to prevent the heater LED being forward biased when the heater switch was open and the heater was disconnected, a protective diode was put into the heater+ line.

Part II

Lotus Elise

Chapter7

Elise: Objectives

The idea for the Elise was to have a one stop solution for all driver assistive technologies. There was also a need to record all information about the state of the car to file for performance analysis. The information that needed to be recorded included:-

1. Battery information
   (a) Voltage
   (b) Current being drawn
   (c) State of charge
   (d) Temperature of batteries
2. Digital inputs from
   (a) Headlamps
   (b) Handbrake
   (c) Door
   (d) Indicators
3. All information about state of Motor Controller
4. GPS information
5. Acceleration in three dimensions

A subset of this information was required to be displayed to the driver in a way, which was both intuitive and efficient i.e. took minimum number of steps to perform. This idea is expanded in Section 7.1.

7.1 Design

The five Usability design principles mentioned in Section 2.1 were kept in mind while designing the GUI for the Lotus. The design followed on from the GUI developed for the Getz last year.

To avoid a compromise between functionality and aesthetics, a combination of two programming languages were used. There was a backend which did all the interfacing with sensors, manipulation on data and writing to file, and a frontend which graphically displayed all the data to the driver. While the backend was written in Visual C++, using Microsoft Visual Studio 2008, the frontend was written in Action Scripting 3 using Adobe Flash CS4. Flash was chosen for its aesthetic appeal while the core functionality was handled in VC++.

Flash was convenient for the frontend as it allowed use of basic building blocks like buttons, images, text boxes and labels without compromising on flexibility. The different screens, which were accessed by pressing different buttons could be kept at different "depths", with the current screen being at the highest depth. Action Script 3 provided high level functionality to dynamically change the display.

A Flash Shockwave file was created for the frontend, which was embedded as an Active X control in the VC++ project. The VC++ code was able to modify variables in the loader for the Flash Active X control. An example of this is shown in Program 7.

Program 7 Changing loader variables from VC++

```
paramString = "bestCornering=";
paramString = paramString + acc.getBestCornering();
this->gui->FlashVars = paramString;
```

These variables could then be read in Flash as illustrated in Program 8.

Program 8 Reading loader variables in Flash

```
cornerG.text = root.loaderInfo.parameters.bestCornering;
```

However, these loader variables could not be modified in Flash, hence for the backend to respond to a user input in the frontend an XML file was passed from Flash to the backend. An example of this is shown in Program 9.

Program 9 Sending XML file from Flash

```
private function forward(event:MouseEvent):void
{
ExternalInterface.call("ResizePlayer", 1, 0);
}
```

This sends an XML file to VC++ which can be read as shown in Program 10.

Program 10 Reading XML file in VC++

```
XmlDocument^ document = gcnew XmlDocument();
document->LoadXml(e->request);
XmlNodeList^ list = document->GetElementsByTagName("arguments");
int playlist = Convert::ToInt32(list[0]->ChildNodes[1]->InnerText);
int song = Convert::ToInt32(list[0]->FirstChild->InnerText);
```

Another benefit of separating the front and backend is that the Flash frontend can be modified or changed without changing the VC++ code. This effectively makes the application "skin-able" i.e. the look and feel can be changed without modifying the backend code. Any future frontends need only access the variables changed in the ActiveX loader by the backend.

These variables are listed below:

1. curSpeed: The current speed of the car (string).
2. distanceRemaining: The distance the car can travel on the remaining charge (integer).
3. batteryPercentage: The percentage of usable change remaining in the battery (integer).
4. voltageValue: The instantaneous voltage of the the battery pack (integer).
5. currentValue: The instantaneous current being drawn from the battery pack (integer).
6. powerValue: The instantaneous power being supplied by the battery pack (integer).
7. mapX: The X co-ordinate of where origin of map should be placed (integer).
8. mapY: The Y co-ordinate of where origin of map should be placed (integer).
9. songFile: The name of the song currently being played (string).
10. albumArtFilename: The path to the album art file to be displayed (string).
11. playlist: The name of the play list (folder) the current song belongs to (string).
12. heading: The rotation (in degrees) the GPS marker needs to undergo to indicate the direction the car is heading in (integer).
13. totalDistance: The total distance that has been traveled by the car (float).
14. distanceSinceCharge: The distance the car has traveled since the last charge (float).
15. moneySaved: The dollar value of money saved by driving an electric car (float).
16. best60: Best 0-60 km time in seconds (integer).
17. best100: Best 0-100 km time in seconds (integer).
18. averageSpeed: Average speed of car(in km/h) for the current trip (float).
19. x: The instantaneous sideways acceleration of the car (float).
20. y: The instantaneous lateral acceleration of the car (float).

21. bestForward: The greatest forwards acceleration of the car in the current trip (float).
22. bestBraking: The greatest braking acceleration of the car in the current trip (float).
23. bestCornering The greatest cornering acceleration of the car in the current trip (float).

Any future versions of the frontend must also send variables to the backend via an XML file. These variables are listed below:

1. nextSong: 1 if song should be changed to next song, -1 for previous song and 0 otherwise.
2. nextPlaylist: 1 if current play list should be changed to the next play list, -1 for previous play list and 0 otherwise.

Chapter8

GUI

The Flash frontend consisted of a number of different "Tabs" which grouped related sets of information. The tab could be changed using the menu buttons at the bottom of the screen. The native resolution of the Flash SWF was 800 × 600 pixels, which was scaled down to 800 × 480 pixels by the VC++ application it was embedded in, so that it was exactly the same resolution as the screen used.

8.1 Main Tab

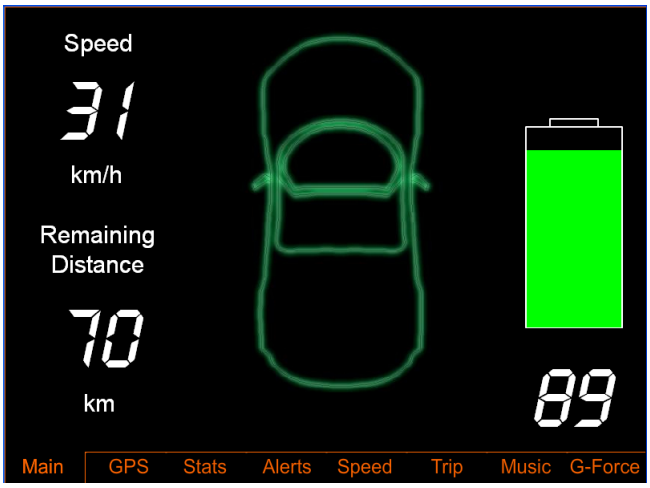

Figure 8.1: Main Tab

The Main Tab showed a graphic of the car, which looks like (Figure 8.1) when no alerts are present. Alert signage appears on the car graphic to communicate potential hazards to the driver. The driver may then navigate to the Alerts Tab to gain further information about the warning.

There is also a graphic indicating the percentage of the usable charge remaining in the battery and text fields with the exact battery percentage, the speed of the car and the distance the car can drive on the remaining charge in the battery.

The battery information - percentage, charge remaining, voltage and current is gathered from the Battery Management System which is first serialized by the Hardware Black Box system. The speed comes from the serial GPS (see Section 1.2.3). The speed obtained from the GPS is inaccurate when the car is stationary, so the speed displayed is 0 when the speed indicated by the GPS is less than 5km/h. To calculate the distance remaining, a running mileage is maintained:

$$\text{mileage}(km/Ah) = \frac{\text{Distance traveled}(km)}{\text{Charge depleted}(Ah)}$$

Both the distance traveled and charge depleted are reset when the battery is recharged. The distance remaining is calculated as follows:

$$\text{Driveable distance} = \text{mileage} \times \text{usable charge remaining in battery}$$

The usable charge differs from the actual remaining charge in the battery as a minimum charge of 30% or 18 Ah is always maintained in the battery. Maintaining this minimum charge extends the life of a Lithium Iron Phosphate battery to about 3000 charge-discharge cycles.

8.2 GPS Tab

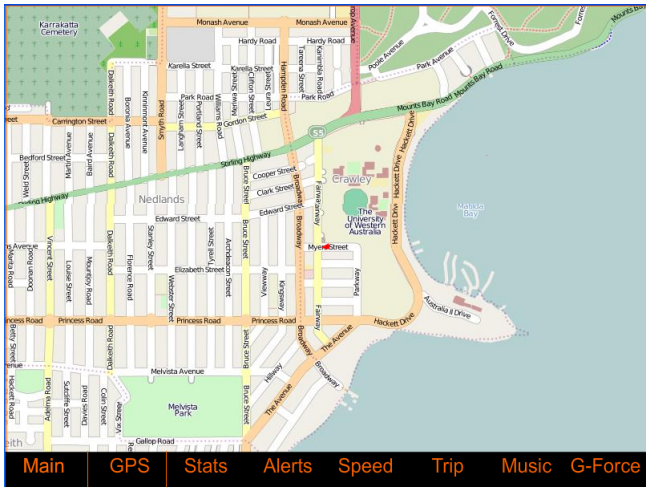

Figure 8.2: GPS Tab

The GPS Tab (Figure 8.2) contains a single large map image of 5108 × 6130 pixels. This image was created by combining several maps which were downloaded from Open Street Maps[18], an on-line repository of free to use maps. The area covered by this map spans from the coast in the West to Midland in the East, and from Joondalup in the North to Jandakot Airport in the South as show in Figure 8.3.

Only 800 × 555 pixels of the map are visible at any one time. This is because the resolution of the Flash movie is 800 × 600 pixels and the bottom 45 pixels are used by the buttons. The map moves around in the background to enable the current GPS location to lie in the centre of the screen where the pose indicator is present.

The co-ordinates for the boundaries of the map in GPS co-ordinates are:

North 31.823° S

South 32.085° S

West 115.743° E

East 116° E

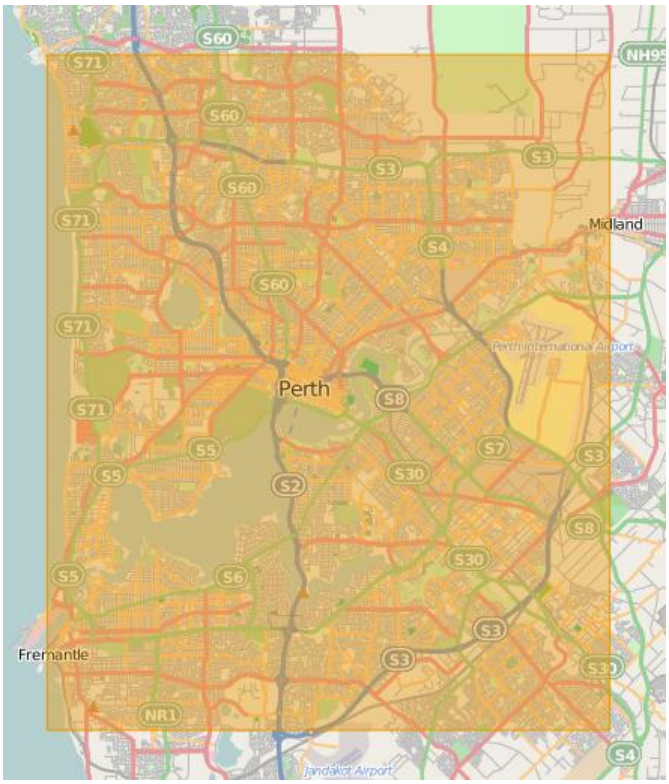

Figure 8.3: Area covered by maps

Image from [18]

As a convention, North and East are positive and South and West are negative. This means we have a range in latitude from -32.085 to -31.823 and a range of longitude from 115.743 to 116. Because Open Street Maps approximates the earth at a cylinder, there is a linear one-to-one correspondence between GPS and pixel co-ordinates on the image. The pixel co-ordinates can be calculated as follows:

$$\text{x} = \text{Image width} \times \frac{\text{Longitude - West boundary}}{\text{East boundary - West boundary}}$$

$$\text{y} = \text{Image height - Image height} \times \frac{\text{Latitude - South boundary}}{\text{North boundary - South boundary}}$$

There is an extra step of subtracting from the image height for the y-position because y should be the pixel distance from the top of the map, rather than from the bottom.

There is a pose (position and orientation) indicator in the centre of the screen which tells the driver where in the map he is currently located. This indicator rotates to indicate which direction the car is heading (the heading is obtained from the GPS). Since this heading information is only accurate when the car is moving, the rotation of the position indicator is only changed when the speed is greater that 5km/h. This means the pose indicator still provides an accurate approximation of the direction the car is facing even when the car stops.

8.3 Stats Tab

The Stats Tab (Figure 8.4) has a graphical display of the instantaneous voltage, power and current drawn from the battery. This is used to provide a visual indication of the effects of accelerating and regenerative braking on the battery.

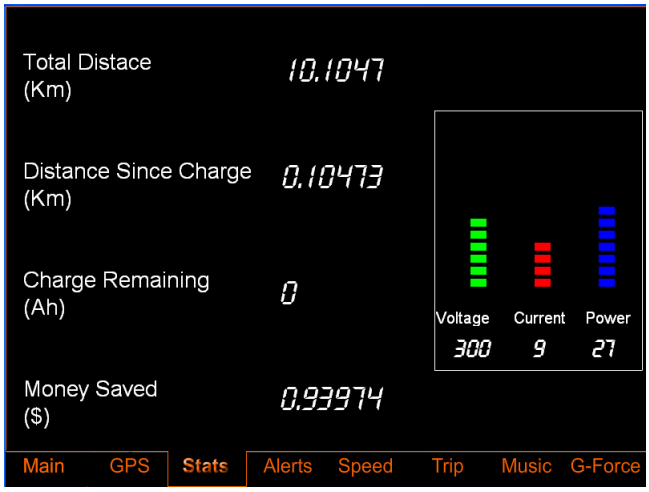

Figure 8.4: Stats Tab

Additionally, there are text fields to indicate the total distance traveled by the car, the distance traveled in the current trip, charge remaining in the battery and the amount of money saved by driving an electric car. The charge remaining is gathered from the battery management system, while the distance traveled is calculated form the GPS information. The GPS sends a serial string to the PC once a second. This string contains the instantaneous speed. The distance traveled in the last second can then be approximated by multiplying the speed by one second. The distance in the current trip and the total distance can then be incremented by this amount. The total distance traveled is written to file once a second and is read by the program at startup. A "trip" is defined as having started at either program startup or when the car moves after being stationary for 10 minutes. When a new trip starts, the trip distance is reset.

The combined urban and extra-urban mileage of a Lotus Elise is 8.3 l/100 km or 12.04 km/l [19]. The money saved is based upon the cost of running the Elise on electric power and the cost of running the car on petrol. The price per litre of fuel is input from a text file at program startup. As an example - if the price of fuel as given by the text file is AU$1.50/l the price of running the car for 100 km (distance that can be traveled on a full charge) is AU$12.45 which equates to AU$0.1245/km. On electric power, the expected cost would be AU$2.5 per 100 km which amounts to AU$0.025/km. This gives a saving of AU$0.0995/km. This figure can then multiplied by the total distance traveled to obtain the money saved.

8.4 Alerts Tab

The Alerts Tab (Figure 8.5) provides addition textual information about any potential hazards that a driver may need to know about. The driver will be alerted about the following events:

1. Battery charge low
2. Particular cell in battery is unbalanced - i.e its voltage is too high or too low
3. Temperature of battery cage too high
4. Door open
5. Light on when ignition off

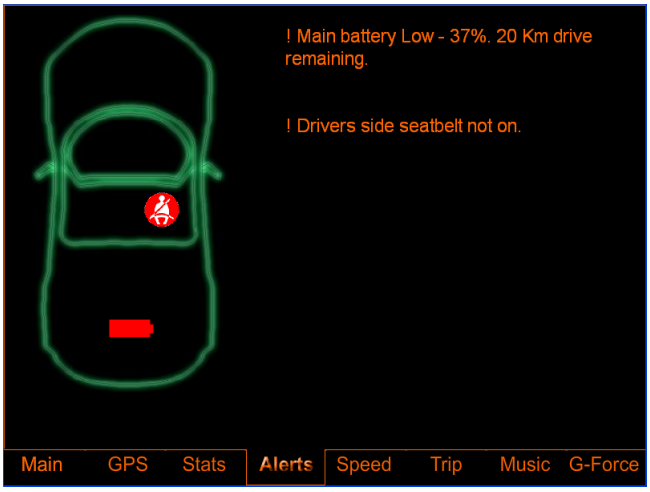

Figure 8.5: Alerts Tab

8.5 Speed Tab

The Speed Tab (Figure 8.6) is an additional safety feature when the driver goes over the speed limit. There are some commonly used presets on the left, and there is the flexibility provided to increase or decrease the speed limit by one. The system beeps when the set speed limit is exceeded. It is also possible to turn this feature off if one is driving around a racetrack.

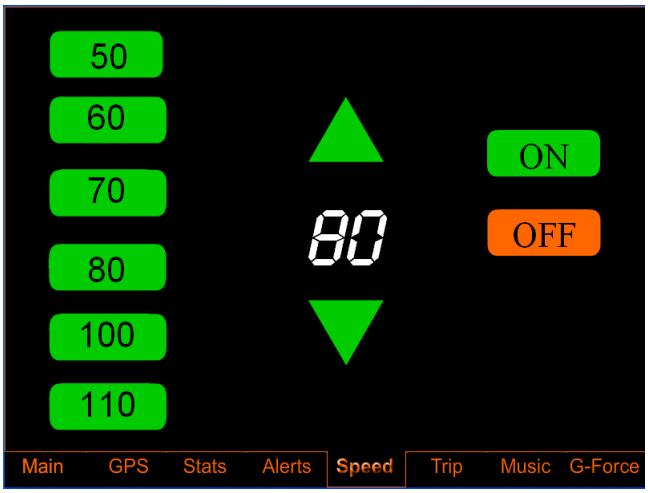

Figure 8.6: Speed Tab

8.6 Trip Tab

The Trip Tab (Figure 8.7) provides all information relevant to the current trip (see Section 8.3 for definition of a trip). The best times for 0-50 km/h and 0-100 km/h are measured in seconds and are calculated from a standing start. A time stamp of the last time the car was at 0 is maintained and the time period to get to 50/100 km/h is recalculated every time the car is at 50/100 km/h. The displayed values are updated when a time period less than the displayed time period is achieved. There is a linear interpolation performed to calculate the exact period as the GPS is only updated once a second (see Section 9.1.2).

The average trip speed is calculated as the total distance traveled in the current trip divided by the time since the start of the trip. There is also a manual stopwatch with the standard start/stop, split and reset functionality.

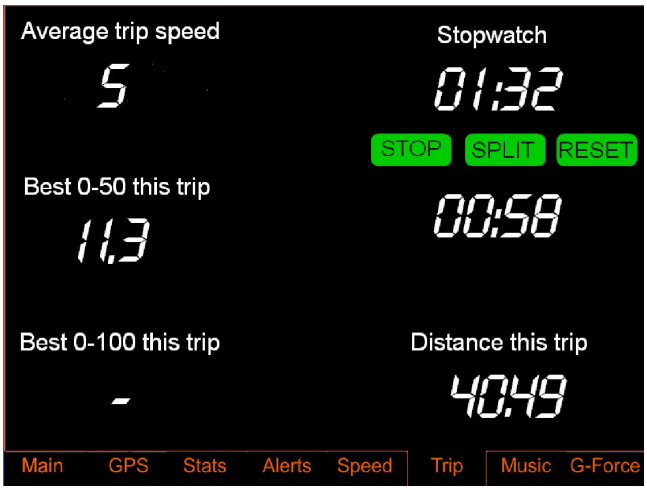

Figure 8.7: Trip Tab

8.7 Music Tab

The Music Tab (Figure 8.8) allows a user to play mp3 files by plugging in a USB flash drive. The song details - name, artist, album is extracted from the mp3 tag.

Figure 8.8: Music Tab

Album art is displayed on the left of the screen. A folder with a group of songs in it is read by the system as a play list. The play list name (folder name) is also displayed.

There are standard start, stop, pause, song forward and song back buttons provided. The fast forward and rewind buttons go forward and backwards between play lists.

8.8 G-Force Tab

The G-Force Tab (Figure 8.9) indicates the acceleration of car. It is measured in multiples of g (acceleration due to gravity). There is a graph which gives a visual indication of current acceleration, and three text fields indicate the best forward, braking and cornering accelerations. These three fields are reset to zero at program startup or at the beginning of a new GPS trip.

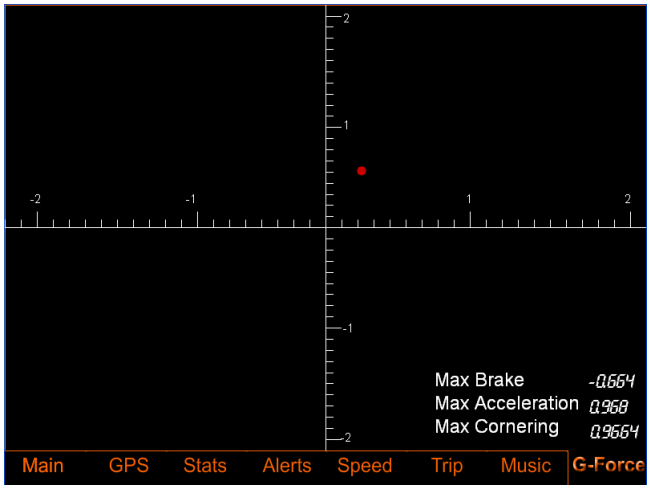

Figure 8.9: G-Force Tab

Chapter9

Program Structure

This chapter details the structure of both the frontend and the backend. With over 2500 lines of code the structure needed to be modular and upgradeable. The flowcharts in this chapter are a guide to how the code was laid out.

9.1 Backend

The backend as detailed by Figure 9.1 has different threads for the GPS, Accelerometer, Hardware Black Box, Motor Controller, Flash updater and Data logger. While the GPS, Accelerometer and Hardware Black Box threads are called when there is serial data present, the Flash updater and Data logger are called by timers to update the variables to be read by Flash and to write all data to the log file.

This multi-threaded interrupt driven approach means that the serial ports are not constantly polled.

9.1.1 Main

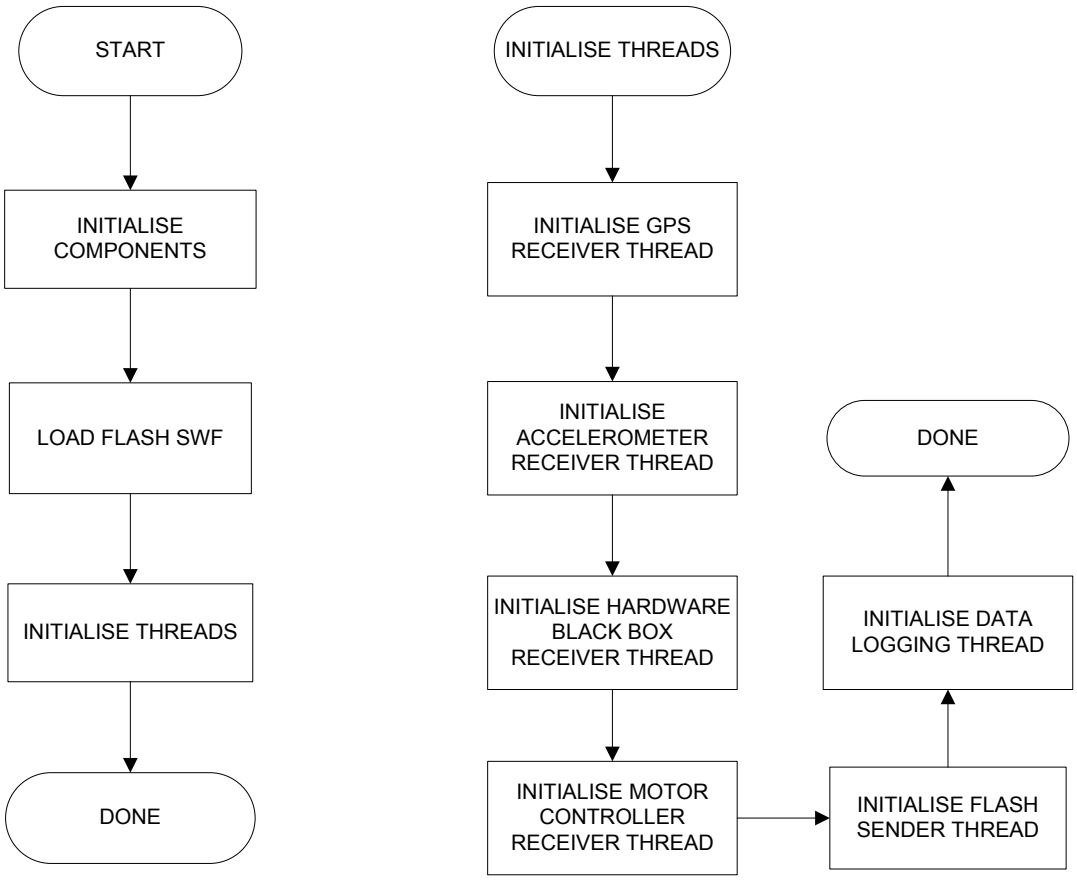

Figure 9.1: Backend - Main

9.1.2 GPS Thread

The GPS thread inputs the GPS data from the GPS module provided the data is valid. Note that the GPS module must have a signal from at least four satellites to have a lock. Further, there is a check sum calculation to check that data was not corrupted during the serial transmission.

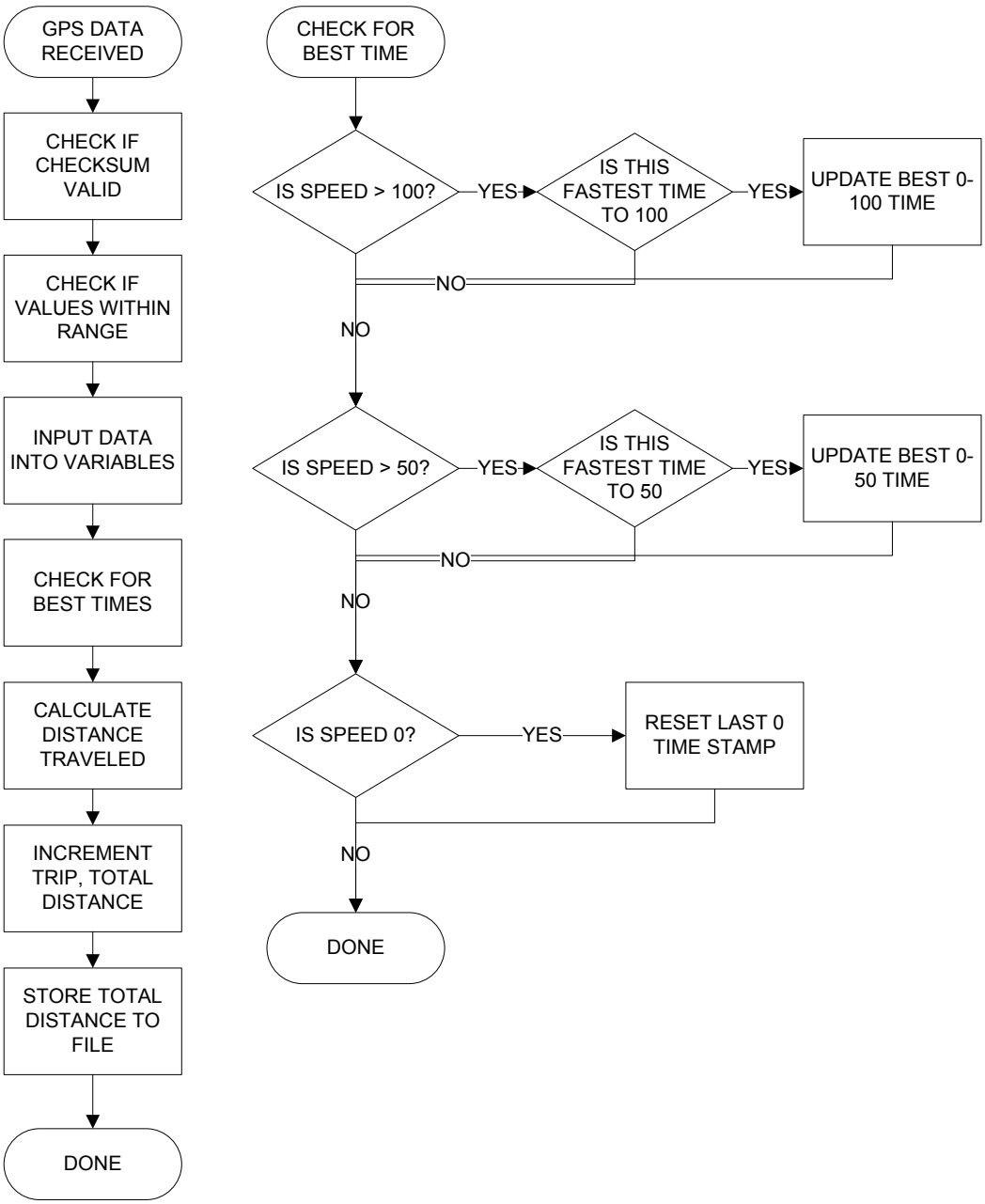

Figure 9.2: Backend - GPS thread

There is also a check made on each field, to check if it is within expected bounds. For example, the hour field must be between 0 and 23, the second field must be between 0 and 59, the latitude should be between -90 and 90 and the longitude should be between -180 and 180, etc.

Because data is received from the GPS module only once a second there needs to be a linear interpolation for approximating when the car starts moving and when it crosses 50/100 km/h to determine the best 0-50 and 0-100 times. Once again, the car is only said to be moving once a speed of 5 km/h or higher is detected by the GPS. If:

$\mathbf{t}_1$ was the first instant the car was in motion i.e. was traveling at over 5km/h (measured in seconds)

$\mathbf{s}_1$ was the speed at time t_1(measured in km/h)

$\mathbf{t}_2$ was the first instant the car was traveling at over 50km/h (measured in seconds)

$\mathbf{s}_3$ was the speed at time t_2(measured in km/h)

$\mathbf{s}_2$ was the speed at time t_2 - 1(measured in km/h)

Note that t_1 - 1 was the last instant the car was at rest and t_2 - 1 was the last instant the car was traveling at under 50km/h (this is because the resolution of the GPS is one second). Interpolating from these values, we can approximate the time the car was at 5km/h (t_5) and the time the car was at 50km/h (t_{50}) and hence the 0-50 time (t_{0-50}) as follows:

$$t_5 = t_1 - 1 + \frac{1}{s_1 - 0} \times (5 - 0)$$

$$t_{50} = t_2 - 1 + \frac{1}{s_3 - s_2} \times (50 - s_2)$$

$$t_{0-50} = t_{50} - t_5$$

A similar calculation can be made for calculating the best 0-100 time.

9.1.3 Accelerometer Thread

The Accelerometer thread reads from the serial Accelerometer connected via a USB-to-serial adapter. The best forwards, braking and lateral acceleration is maintained. This data is sent to the frontend by the Flash thread.

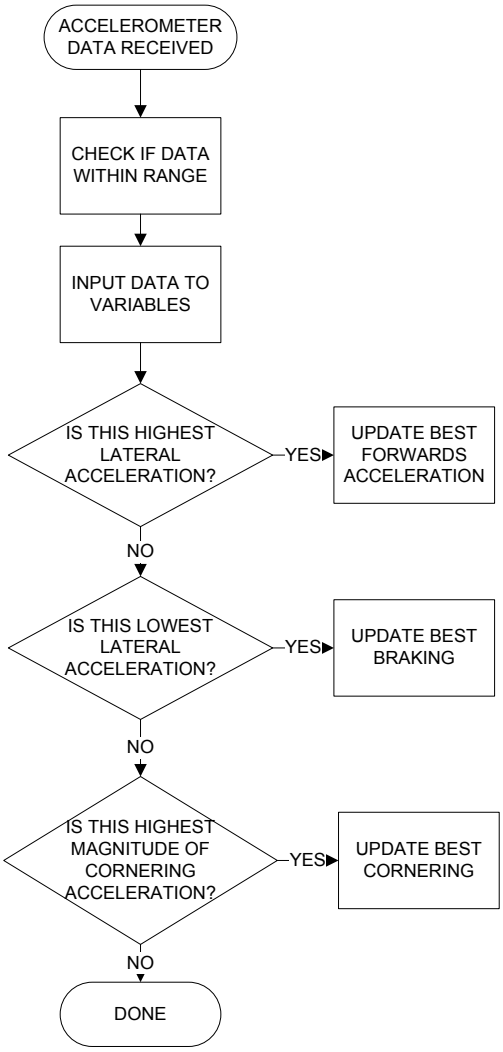

Figure 9.3: Backend - Accelerometer thread

9.1.4 Hardware Black Box Thread

The Hardware Black Box system was developed by Daniel Kingdom. It has a USB interface with FTDI drivers for an emulated serial port. It provides serialised information from the BMS - battery pack voltage, battery pack current and individual cell voltages and currents. It also transmits information about the state of the digital inputs and analogue inputs - temperature of the battery packs, headlights, door sensor and seatbelt sensor via serial.

All this data is sent via two ASCII strings. The Hardware Black Box thread tokenises these strings and extracts all the information.

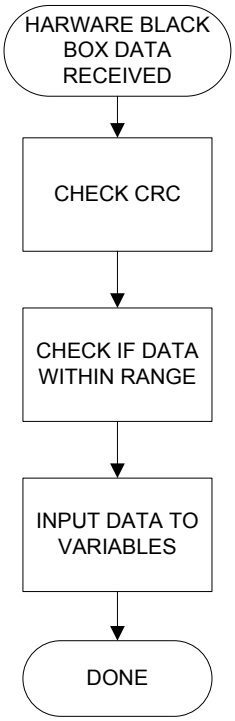

Figure 9.4: Backend - Hardware Black Box thread

9.1.5 Flash Thread

The Flash thread transmits all the variables mentioned in Section 7.1 to the frontend.

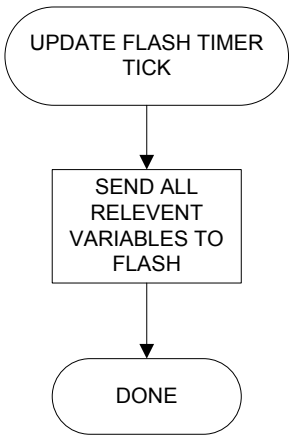

Figure 9.5: Backend - Flash thread

9.1.6 Logging Thread

Logs are written to the output CSV files once every 10 seconds, two new files are created at program startup and are named according to the date and time of file creation. The main log is named as ddMMyyyyhhmmMain.csv, while the battery log is named ddMMyyyyhhmmBattery.csv where dd is the day, MM is the month, yyyy is the year, hh is the hour and mm is the minute. The main log file stores the GPS time, GPS co-ordinates, heading, speed, distance traveled and Accelerometer data. The header for the main file is:

day, month, year, hour, minute, latitude, longitude, heading, speed, distance, accX, accY, accZ

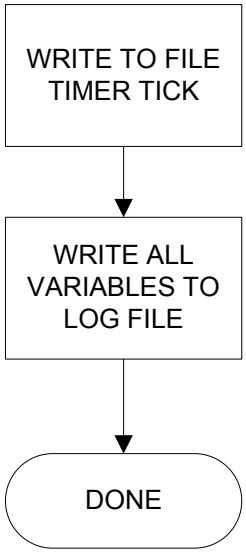

Figure 9.6: Backend - Logging thread

The battery log file stores all the information about the overall battery pack voltage, current, state of charge, and percentage of usable charge remaining. Additionally, there is information about each of the the 99 cells in the battery pack. The individual voltages, the maximum and minimum voltages and the bypass current of each cell is also logged. The sampling interval is the interval between successive polls of the BMS system by the Hardware Black Box. The bypass current is the current being drawn by a BMS module to discharge a particular cell during charging of the overall battery pack, if the voltage of the cell is too high. The header for the battery log is:

voltage, current, charge, percentage, cell1Voltage, cell1MaxVoltage, cell1MinVoltage, cell1BypassCurrent, ..., cell99Voltage, cell99MaxVoltage, cell99MinVoltage, cell99BypassCurrent

9.2 Frontend

The user interface in Flash is provided by eight buttons at the bottom of the screen which allow the user to change the tab they are currently in. There are also additional buttons on some of the screens for functionality specific to the tab: such as the music control buttons in the Music tab and the Stopwatch control buttons in the trip tab.

9.2.1 Main

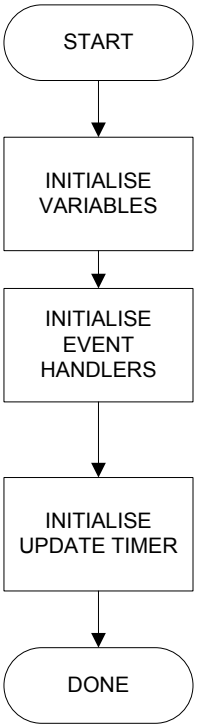

Figure 9.7: Frontend - Main

The main thread initialises the event handlers for the eight menu buttons, the stopwatch buttons and the buttons for music player buttons. An event listener calls the

appropriate event handler when a button is pressed. Further the update timer is initialised, this sets up a timer which updates the display ten times a second.

9.2.2 Event Handlers

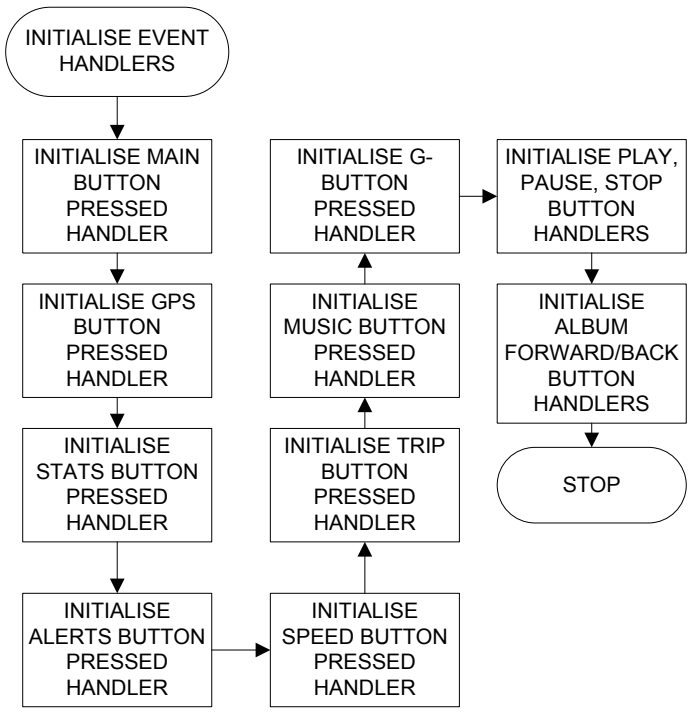

Figure 9.8: Frontend - Setting up event handlers

Figure 9.8 shows all the event handlers for all the buttons in the GUI being set up.

Figure 9.9 shows the event handler for the Main button. Such event handlers also exist for each of the other menu buttons. Flash allows components to be displayed at different "depths". This means that while all components are active they will not be visible if there are other components at the same position which are at a higher depth. When a menu button is pressed, all the components of the relevant tab are

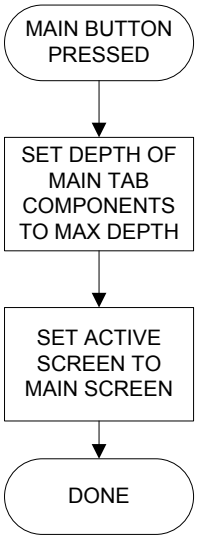

Figure 9.9: Frontend - Main button event handler

set to the highest depth, this makes all components in other tabs invisible.

There are also event handlers for the stopwatch buttons, and for the music control buttons. The music control buttons simply update variables in the backend indicating the song/play list needs to be changed. The frontend then sends the path of the song to be played.

9.2.3 Update Thread

This update thread is called ten times a second. The update thread checks what tab the GUI is currently displaying and updates all the items in that tab.

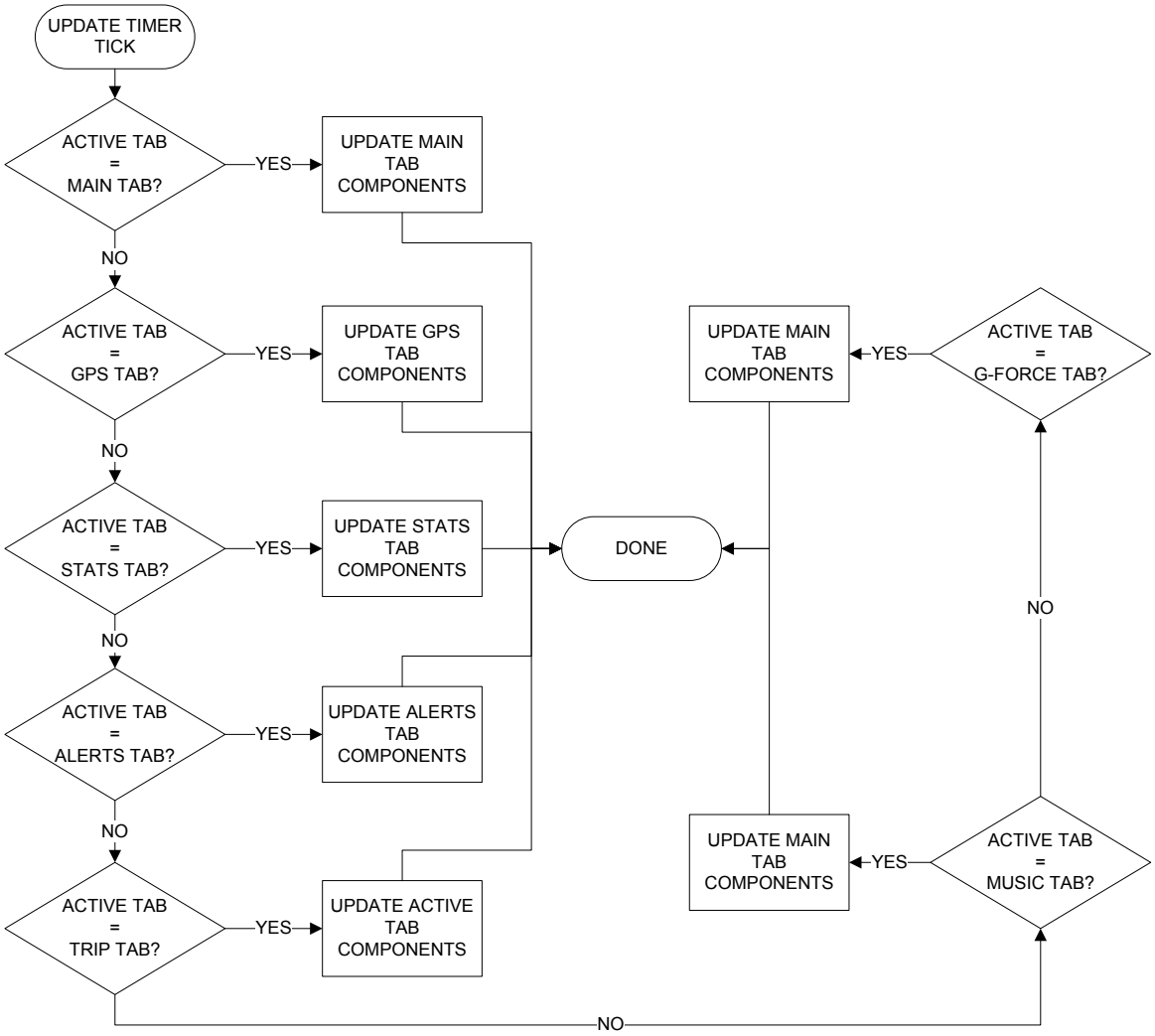

Figure 9.10: Frontend - Update thread

Chapter10

Installation

The Eyebot code was successfully loaded and run on the Eyebot in the Getz. The Breakout Board was mounted to the side of the steering column. This meant that it was easily accessible and also took clutter away from the cramped space behind the Eyebot. Ribbon cables with IDC crimp connectors were used to connect the board to the Eyebot and a multi-channel, multi-core cable was used to connect the outputs of the Motor Controller to the Breakout Box.

For the Lotus, it was decided to mount the car PC in the passenger cabin rather than in the boot to preserve signal integrity of the VGA cable running to the screen which had to be mounted where the driver could access it. The PC was mounted behind the passenger footrest. An Aluminium mounting plate was designed for this purpose. There was a permanent 12V power supply from the 12V battery. The 12V signal from the ignition was used to turn it on and off. This meant that turning the car on turns the PC on and turning the car off causes the PC to perform a normal shutdown. The screen was mounted just above the car PC where the stereo was originally located. This placement made the screen accessible to both the driver and passenger. Further, the dashboard provided shade to the screen which meant there was reduced glare when the car was being driven in bright sunlight. A mount was designed which slotted into the gap left by removing the stereo. The screen could be slid on and off this mount.

Chapter11

Conclusion

Working on the REV project proved to be challenging, stimulating and hugely rewarding. With over 2500 lines of code, this was the single largest software application developed by the author. In working on this project the author was able to extend his knowledge of electronics, specifically PCB design. He also learnt extensively about working with various sensors and integrating data from these sensors.

The project proved to be successful with the onboard driver assistive system working well on both the Getz and the Elise. Both these cars now have a whole host of capabilities, some of which are not available even on high end commercial cars. The dual objectives of providing the driver with all the information he requires as well as logging data for analysing and improving performance have been achieved. The logs obtained from the software can be used for performance benchmarking as well as for tweaking the Motor Controller or the Battery Management System. The project was also successful in providing the driver with extra features such as a mp3 player, an Accelerometer, GPS navigation and a stopwatch. The interface in particular is user-friendly and visually appealing.

As the hardware makes use of mostly standard off the shelf equipment and the software is written in widely used languages, this system can easily be ported to run on other cars, whether they are powered by electricity or by more conventional means.

The REV project as a whole has been very successful in performing conversions on both performance and economy cars.It has also been an excellent learning experience for all the students working on the project.

This project is at the forefront of onboard driver assistive systems and can lead on to further innovation.

11.1 Future Prospects

This project builds a base for further innovation in driver assistive technology. Some ways in which the project could be extended in subsequent years are:

1. Development of a full-fledged navigation system, where the driver can input destinations.
2. Extension of the area covered by the maps to all of Australia, or even have worldwide coverage.
3. Use of voice communication with the system while driving.
4. Integration of bluetooth for use of mobile phones.
5. Improvement upon design of the GUI.

Bibliography

[1] "About rev." http://www.therevproject.com, March 2008.

[2] J. Ward, "Embedded libraries," tech. rep., University of Western Australia, 2008.

[3] E. McLeod, "Eyebot m6 controlled sensor package in a renewable energy vehicle - hyundai getz ," tech. rep., University of Western Australia, 2008.

[4] GlobalSat, *USB GPS Users Guide - WIN*.

[5] W. S. Ltd, "Extracting information from the gps's string of data." http://www.windmill.co.uk/gps.html, 2001.

[6] J. Nielsen, "Usability 101: Introduction to usability." http://www.useit.com/alertbox/20030825.html, August 2003.

[7] M. Trend, "135i vs evo 2008 mitstubishi lancer evolution mr dash display." http://www.motortrend.com/photo_gallery/112_0806_135i_vs_evo_photo_gallery/photo_10.htı 2008.

[8] B. Blog, "More info on bmw car access." http://www.bmwblog.com/2008/03/05/more-info-on-the-bmw-full-in-car-web-access, 2007.

[9] W. Cunningham, "Driving it: Car interfaces and usability." http://reviews.cnet.com/4520-10895_7-6744922-1.html, June 2007.

[10] AutoBlogGreen, "First drive: Tesla roadster." http://green.autoblog.com/gallery/first-drive-tesla-roadster-sport-2, October 2009.

[11] Engadget, "2007 audi q7 in-dash system leaked." ,http://www.engadget.com/2006/02/20/2007-audi-q7-in-dash-system-leaked, February 2006.

[12] R. Menezes, "Cadillac announces pricing for the 2009 escalade hybrid." http://blog.pricewheels.com, August 2008.

[13] A. R. C. H. W. Louis McCarthy, Josh Pieper, "Performance monitoring in umrs solar car," *IEEE Explore*, vol. 1, p. 5, September 2000.

[14] N. National Space-Based Positioning and T. C. Office, "The global positioning system." http://www.gps.gov/.

[15] "How does a gps receiver figure its distance from a gps satellite?." http://www.how-gps-works.com/faq/q0111.shtml, 2006.

[16] "How does gps triangulation work?." http://www.how-gps-works.com/faq/q0110.shtml, 2006.

[17] R. Matthew, "Electric vehicle design concepts and project management," tech. rep., University of Western Australia, 2008.

[18] www.opensteetmap.org.

[19] "Technical specifications of 2006 lotus elise." http://www.carfolio.com/specifications/models/car/?car=144782, 2007.

[20] UQM TECHNOLOGIES, INC., *User and Installation Manual for PowerPhase 75 Traction System.*

Appendices

Appendix A

Motor Controller Breakout

This board was a joint effort of Cameron Watts, William Price and the author.

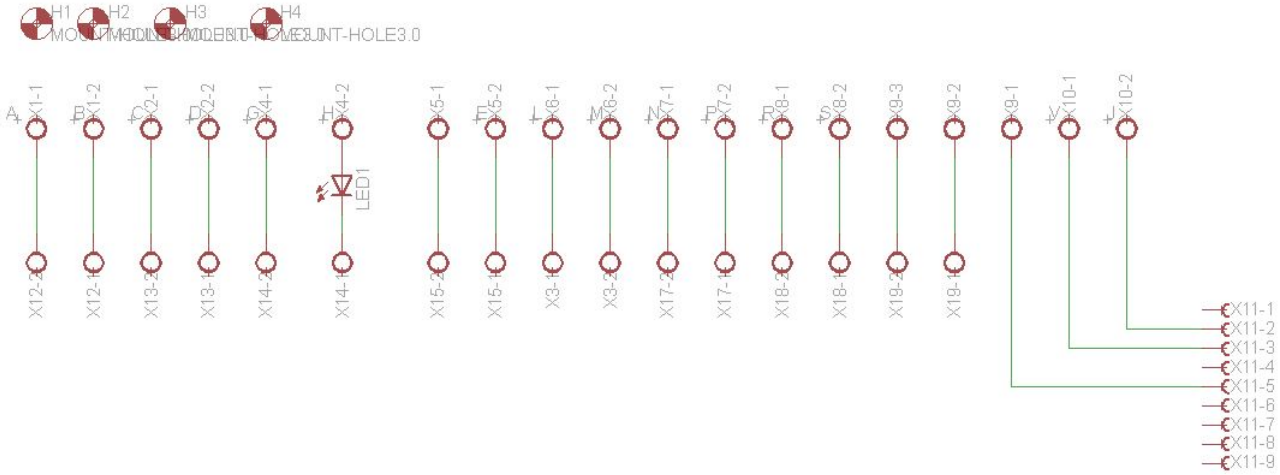

Figure 1: Motor Controller Breakout - Schematic

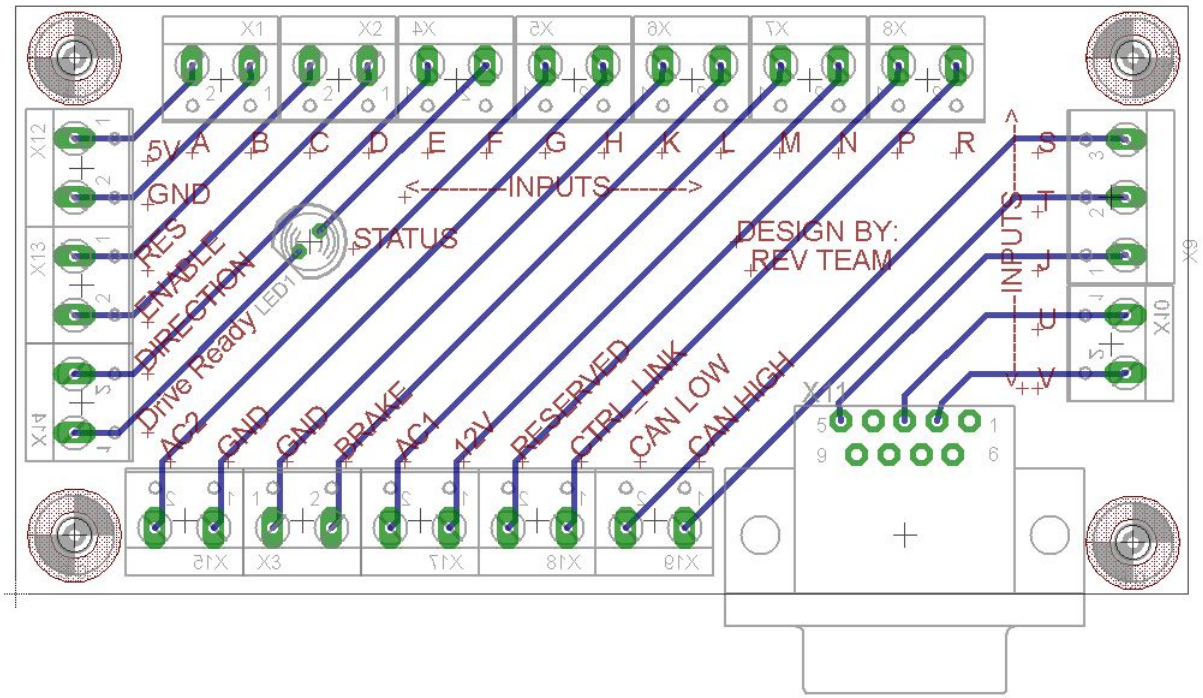

Figure 2: Motor Controller Breakout - Layout

Appendix B

Motor Controller Pinouts

Pin	Signal
A	Vcc user
B	GND user
C	Reserved
D	Enable
E	Direction
F	Drive Ready
G	ACCEL 2
H	GND user
J	GND serial
K	GND user
L	Brake
M	Accelerator
N	+12 VDC
P	Reserved
R	CTRL IN
S	CAN L
T	CAN H
U	Serial transmit
V	Serial receive

This data was obtained from the Motor Controller manual [20]