ELEC3330

Getz Graphical User Interface

The Renewable Energy Vehicle Project

Jurek Tadek Malarecki 20263575

22/10/2009

Acknowledgements

I gratefully acknowledge my supervisor and all those who helped me complete my third year project; Daksh Varma, Colin Dickie, Jon Wan, Daniel Kingdom and any others who offered helpful feedback.

Table of Contents

1 Introduction

The Renewable Energy Vehicle (REV) Team aims to highlight the viability of creating and using all-electric vehicles in day-to-day transit [1]. This third year engineering project revolves around completing the graphical user interface (GUI) in one of these vehicles by ensuring that the user will be able to effortlessly access all information regarding the vehicles present and past usage. The GUI itself will be operated via touch control, on an Eyebot M6 embedded system, which has been developed at UWA. It is dashboard mounted into the REV economy vehicle, a Hyundai Getz.

The REV project is future focused in that by emphasising the viability of electric vehicles will help to ensure environment sustainability. By creating fully functional, street registered vehicles, the project can demonstrate the advantages of all electric, renewable energy vehicles to all members of society [2]. Ultimately, this should make the use of renewable energy in vehicles more promising to the community.

The platform on which the GUI will function is the Eyebot M6 embedded system. It functions with a Linux based operating system [3], Busy-Box, and will be able to interface with the vehicle in order to present the user with all critical information. This report will discuss how the program was designed and the purpose of each of its various modes.

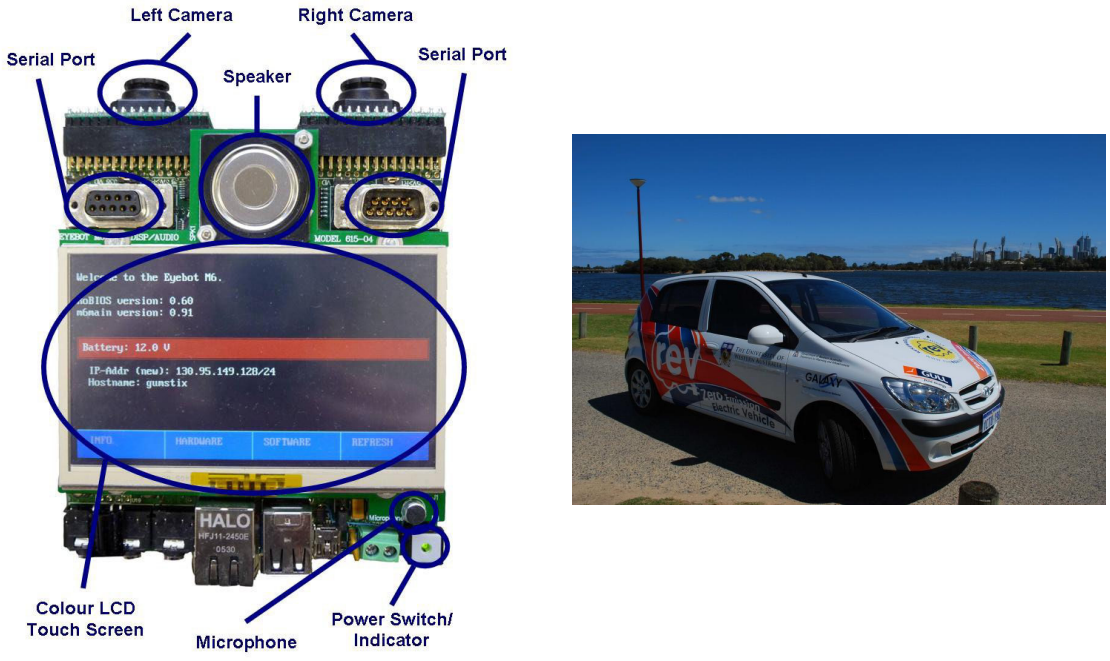

Figure 1-1 The Eyebot M6 embedded system [3] (left), and the converted Hyundai Getz [4] (right)

2 Graphical User Interface Design

2.1 Objective

The objective of this project is to extend and develop a user interface that conveys vital information regarding current and past operations of the car to the user, to enhance the driving experience. The interface itself, which will run on an Eyebot M6 embedded system, should make important information easily accessible and have an intuitive design.

As this project is based on the work of a previous project, the correct operation of the user interface needs to be ensured, and any deficiencies or problems need to be addressed. This especially includes the interfacing problems, which resulted in most information being filled with static placeholders as inputs could not be read, and problems with the GPS and black box code existed[5]. Furthermore, new features needed to be added including information regarding the current trip, which is defined as any period of activity following the vehicle having been stationary for at least 10 minutes. This incorporates determining current, average and maximum speed; distance travelled since trip start; best 0 – 50 kmh^{-1} time since trip start; and best 0 – 100 kmh^{-1} time since trip start. A stopwatch, which has both manual controls and the option to run based on whether the vehicle is stationary or not, needed to be developed. This involved incorporating a large digit display and the option to change between manual and automatic mode.

Additionally, an acceleration screen, graphically displaying the data read from an accelerometer; a telemetry information screen, displaying the current status of the remote wireless 3G connections; and settings menu were required. A consequence of these new features is that a new menu system needed to be implemented to account for the additional information that would be presented.

2.2 Prior Work

When this project had commenced, the Hyundai Getz had already been converted to an all-electric vehicle. The GUI had been started; however a few issues remained, for example only static dummy variables were being displayed [5]. Furthermore, as this aspect of the vehicle had already been started, the basic design had already been completed, and the style and navigation method had already been defined as having a menu bar with multiple screens or windows for accessing information. This was based on making the system easy to use, by making it easy to learn, easy to navigate and aesthetically pleasing [5].

In terms of the Eyebot, programs and files can be accessed and run on a USB flash drive, which acts as another storage medium for the Eyebot. Moreover, the Eyebot has been connected to a wireless router that is connected to a wireless 3G modem that simplifies transferring and executing programs on the Eyebot and allows for remote data logging, namely telemetry. The embedded system interfaces with many of the systems and sensors via USB, RS232 and potentially its own inputs [3]. Finally, predefined library functions located in the RoBios library are Eyebot specific functions that increase functionality of the system [3], for example by allowing graphics to be displayed, giving immediate access to the input and outputs, and so on.

2.3 Purpose

The purpose of this project is to tie in all the support systems that allow the user to access the current status of the car. Moreover, it gives access to some enhanced vehicle features that will be useful to the driver, for instance a GPS system, and emphasises the present state of technology that can be made available to those in the market for an economy vehicle. Finally, by completion of the project, it represents the production of an all-electric economy vehicle and hence highlights the viability of such vehicles in the market.

2.4 Method

The GUI has been developed on an Eyebot M6 embedded system. While the embedded system is quite capable of processing the tasks at hand, the manner at which graphics must be displayed is through the use of simple functions from the RoBios library[3]. These include drawing lines or filling areas with colour, or displaying 24 bit per pixel (bpp) RGB images. Displaying images is a useful feature, however it was challenging to lay them over one another whilst ensuring the correct image was displayed on top, to guarantee the various screens functioned and displayed information correctly. Hence, the software had to be designed with a layered approach to displaying information in mind.

Another challenge faced was the nature of reading key presses. The problem encountered was that of multiple key presses being recorded for every touch event. This was overcome through the use of a global Boolean variable as a test for whether or not to act on a key input. Namely, every action that occurs as a result of a key press was wrapped such that the action will only take place if the Boolean variable is false and whenever an action takes place, set this variable to true. A small function on a timer runs every 50 milliseconds, to check if that variable has been true for 250 milliseconds. If so, reset it to false. Hence, this effectively ignores any key presses for the next 250 milliseconds, which results in a responsive interface design.

Furthermore, given the greater number of windows from the base design, a menu bar handler function has been created that allows the GUI to know which menu it should currently show, in case the menu needs to be reloaded due to the layered image nature of the Eyebot's graphical functions. This function also handles which window is currently being displayed, and presents it in a different colour such that it is highlighted on the menu.

Given the problems reading data from sensors positioned around the car, rather than utilising the digital input and output on the Eyebot itself, a gateway board has been connected to the Eyebot via a serial to USB connector and all sensor outputs, including the battery data will be connected to the gateway board. This includes a brake vacuum pressure sensor, seatbelt sensors, boot/door/fuel lid open sensors, ignition position information and the battery voltage and current information. Note that there is also an Inertial Measurement Unit (IMU) that contains a GPS sensor and accelerometer, connected via USB, for which no issues were encountered. This data is stored in the black box data structure shown below in *Figure 2-1*.

```
typedef struct {
      int datestamp;
      int timestamp;
//from sensors
      int batt_low;
      int fuel_door;
      int inertia;
      int ignition;
      int speedo;
      int door_warn;
      int boot_warn;
      int seatbelts;
      int brake_vp;
      int throttle;
      int check_engine;
      float throttleAnalogue;
//from battery management module
      int mb_percent;
      float main_batt_v; //Analogue
      float main_batt_i; //Analogue
      int distanceRemaining;
//from IMU
      GPSdata_t gpsd; //GPS
      ACCdata_t accd; //Accelerometer
} bb_data;
```

Figure 2-1 The data structure for recording all input information

Another point of interest was the method for eliminating noise in the data inputs. In the case of the GPS, this was done by taking the last three readings, and finding the median value. This was chosen as the median value offers protection against large outliers and small variations in readings, as opposed to the mean, which is less tolerant to outliers.

The GUI employs numerous background timers to correctly handle the updating of vehicle information. These are discussed in *3 Design and Results*. These include timers to read data from the input devices, namely the IMU and gateway board and check for critical errors that require a full screen warning message. Also, there are timers to update odometer and speed information, and finally check for trip end.

As a result of the aforementioned challenges and changes, the new program structure can be represented in *Figure 2-2* below. Essentially, changing of screens occurs through the main function as a result of the menu handler. However, the critical warning supervisor timer has the ability to take over an active screen in order to display a full screen error message. Finally, the background timers, which are somewhat independent of the main program, run silently updating any variables with new data read from the inputs, such as sensors or IMU.

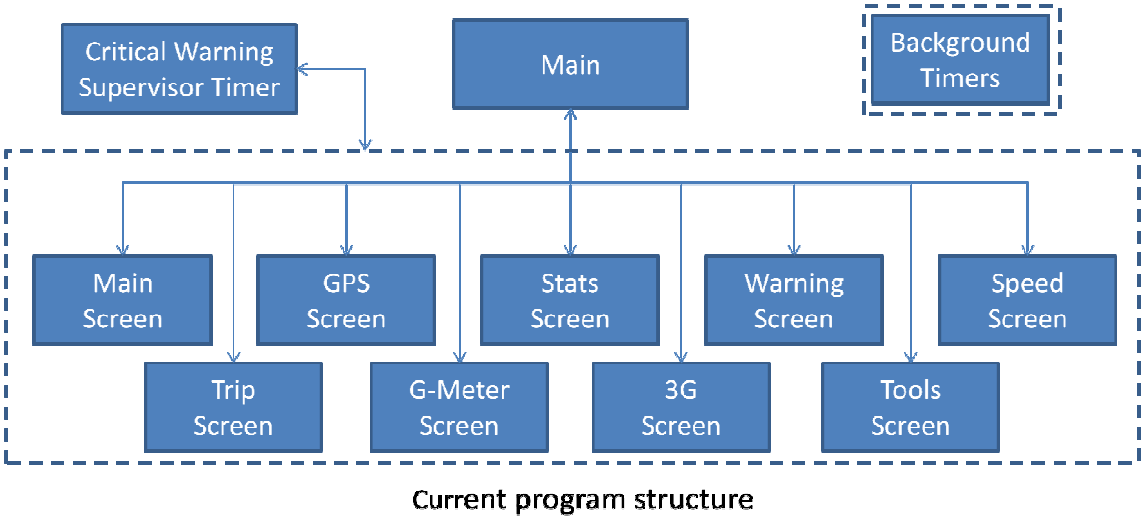

Figure 2-2 Main program structure

Furthermore, the general design for each screen, shown in *Figure 2-3*, involves setting the background image, initialising timers to update the information on the screen periodically, which in most cases is once per second to match the polling rate from the sensors, and then waiting until a key is pressed. If a key is pressed which relates to an operation on the current screen, then that action is completed; if it is a menu operation, close all open timers relating to the screen's operation, then return control back to the main function.

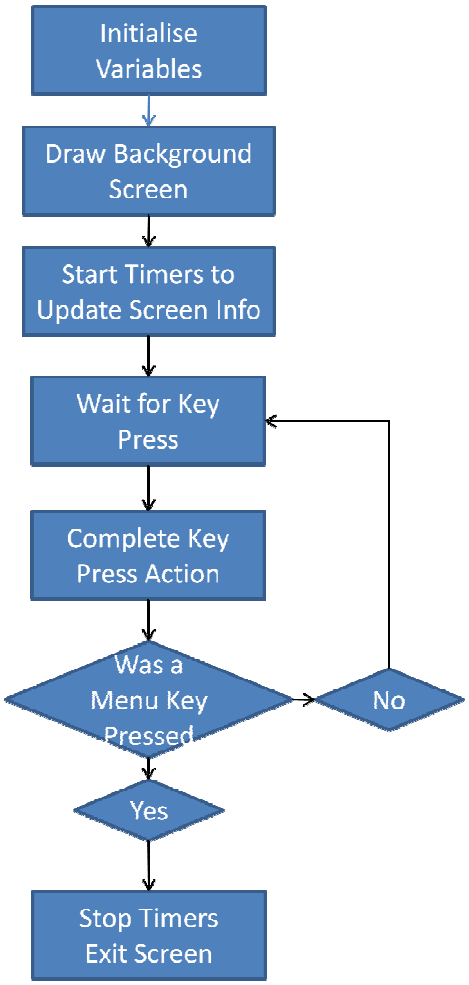

Figure 2-3 General screen operation

3 Design and Results

The final result of designing and programming each element of the graphical user interface has produced ten separate screens to display some facet of the vehicle's present state. These individual screens include, a main/home area; GPS navigation; statistical information, warnings; trip statistics; accelerometer data; speed warnings; 3G communication data; a tools menu; and, finally, an exit screen. Each screen, presented below, clearly displays the relevant information to the user in a simple to understand layout.

3.1 Main

Figure 3-1 Main Screen, displaying no warnings (left), all warnings (right)

The main screen, shown in *Figure 3-1*, is designed to present the most important information to the user, in one place. It displays the current speed, taken as the median from the last 3 GPS readings; battery status, read from the gateway board; remaining distance, calculated from the battery percentage and the range; and any warnings read from the sensors that are factory standard with the vehicle and those which were installed to ensure running of critical systems. For more information refer to *3.3 Vehicle Statistics* and *3.4 Vehicle Alerts* below.

3.2 GPS Navigation

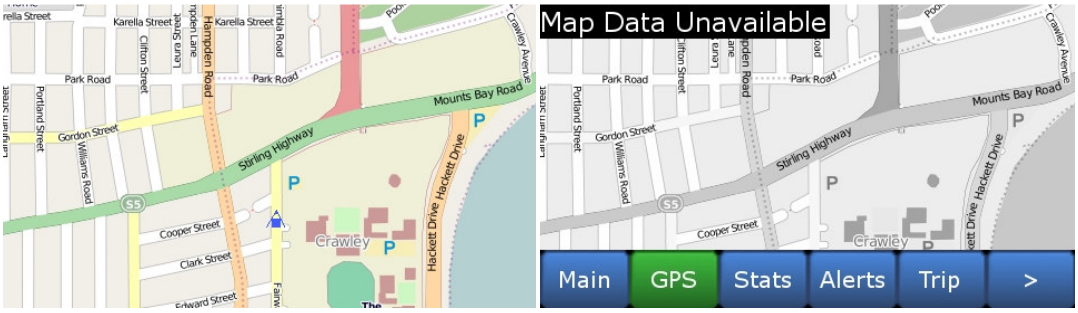

Figure 3-2 GPS navigation screen, active (left), inactive with error message (right)

The GPS navigation system operates by comparing the current latitude and longitude reading from the GPS to the boundaries associated with small images of street maps in order to present the current location of the vehicle to the user. Due to the layered nature of displaying

images on the Eyebot the most appropriate approach is to load these small images, stored on the Eyebot's memory, into RAM such that they can quickly be redrawn onto the screen as the car position is updated. Each small image is based on a larger street map taken from OpenStreetMaps.org, and tiled into separate PPM (portable pixel map) files using a small command line C program that divides up a larger PPM and logs the filename and boundary of each tile. Additionally, the small maps overlap to allow the interchange between maps to be easier to follow.

A small square can be drawn to represent the position of the car and is calculate by converting the distance between the current GPS latitude and longitude and each boundary latitude and longitude of the map as a percentage. As the size of the screen is known to be 480x272 pixels in addition to the known size of overlap between maps, the current pixel position can be accurately determined and a square drawn around it using the LCDArea Eyebot function. A small triangle depicts the direction of travel, if the speed is non-zero, based on the heading (measured clockwise in degrees from North) supplied by the GPS. The direction of the arrow on the screen can be calculated by using simple trigonometry. Namely, the x position add sine of the heading gives the x coordinate of the tip of the triangle and the y position subtract the cosine of the heading gives the y coordinate of the tip of the triangle. Likewise other points of the triangle can be calculated in a similar fashion by adding a particular angle to heading. That is, 120° for an equilateral triangle, or less or greater than 120° for an isosceles triangle.

Another feature of the GPS is that when the GUI loads, the last logged position is loaded as a greyscale image such that it can be displayed when the GPS screen is opened and while the true map and current position is being determined. This is completed at start-up as it takes three to four seconds to load the 480×272 pixels into RAM. When the car's GPS position corresponds to a new map then it loads both the greyscale and colour map into memory. The greyscale map will only shown if the connection is lost, in which case it also displays 'GPS Connection Lost Re-establishing', or the maps have moved out of range, and displays 'Map Data Unavailable', otherwise the colour map will be used to represent correct operation. Furthermore, if a different feature of the GUI is selected, the last loaded greyscale map is used as a starting point next time the GPS screen is loaded. To create a greyscale map using a full colour PPM file, a formula for converting the red, green and blue components of a pixel into a brightness value [6] was used and is given below in *Equation 1 Brightness from RGB values*.

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

Equation 1 Brightness from RGB values

As the screen is relatively small, it is best to show a full map rather than having 20% of the screen taken by the menu, hence to control this screen, when a touch event occurs the menu buttons appear for 5 seconds and functions as normal. An alternative functionality and approach that was suggested was to press centre to show menu, or press edges to move map around then press centre to return to the current vehicle position. However, due to the long loading times of images, namely three to four seconds, this was not implemented in favour of responsiveness.

3.3 Vehicle Statistics

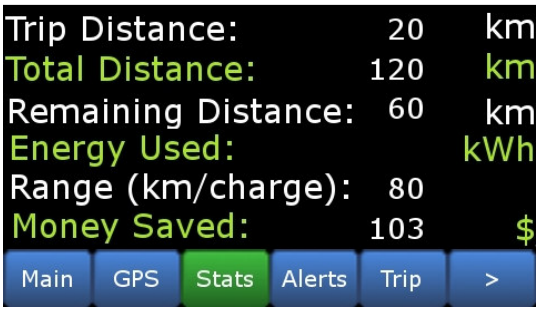

Figure 3-3 Vehicle statistics screen

Vehicle statistics presents more information pertaining to the driving information of the car. Trip distance, as described in section *2.4 Method*, is the summation of the median speed reading multiplied by one second to give the distance the car travelled in a particular second, refer to *Equation 2 Trip distance*, recalculated every second. This resets to zero when the end of a trip has been detected, that is when the car has been stationary for 10 minutes, or the Eyebot is switched off. Conversely, total distance does not reset to zero, and the long running odometer reading is logged in memory.

$$tripDistance_{new} = tripDistance_{old} + speed * time \quad km$$

$$= tripDistance_{old} + currentSpeed \times \frac{1}{3600} \quad km$$

Equation 2 Trip distance

Remaining distance is the approximate distance the vehicle may travel before the energy remaining in the battery is insufficient to power the motor. It is determined by the range, see *Equation 4 Range formula* below, multiplied by the remaining battery percentage. Energy used makes use of the voltage and current information from the battery management system and the simple relationship below.

$$Energy = V \times I \times t \quad (J)$$

$$Energy_{new} = Energy_{old} + V \times I \times 1 \times \frac{1}{3600 \times 1000} \quad (kWh)$$

Equation 3 Energy usage

Finally, range, representing the distance the car can travel on a full charge, is based on previous travel distances and power consumptions and can be calculated as shown by *Equation 4 Range formula*. This value can then be averaged over every trip that is taken by the vehicle, in order to calculate an estimate based on vehicle usage habits. In addition, the savings based on fuel saving can is determined as shown in Equation 4*Equation 5 Money saved formula*.

$$Range = Trip\ Distance \times (1 - Remaining\ Battery\ Proportion)$$

Equation 4 Range formula

$$Money\ saved = \frac{Total\ Distance}{Cost\ of\ fuel\ per\ km} \times Cost\ of\ pertrol\ per\ L - Energy \times Cost\ of\ electricity(kWh)$$

Equation 5 Money saved formula

3.4 Vehicle Alerts

Figure 3-4 Vehicle alerts screen, no warnings (left), all warnings (right)

The vehicle alert screen checks if any sensor is in the error state, namely, emitting a signal that warrants concern. If those signals are detected or the battery is low; this screen will display the relevant warning, shown above in *Figure 3-4*. For more information regarding the sensors, refer to section *2.4 Method*.

3.5 Trip Information

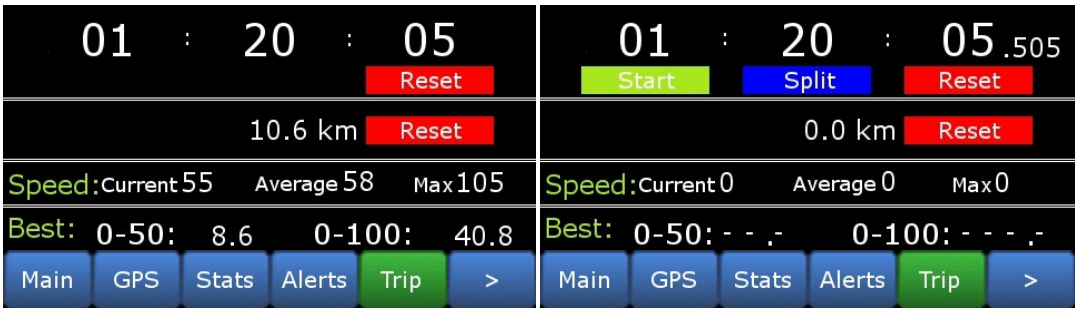

Figure 3-5 Trip information screen, running with automatic timer (left), reset with manual timer (right)

The current trip information screen, *Figure 3-5*, displays the current speed statistics, a resettable odometer and either a manual or automatic timer, that can be chosen in the tools screen. The current speed is simply the median GPS reading, the average speed is the long running average of the median speed calculated every second, and the maximum speed is determined by checking every second, whether or not the current speed is great than the recorded maximum speed. The fastest 0 to 50 kmh^{-1} and 0 to 100 kmh^{-1} are calculated by checking every second whether or not the speed is zero. If not, record the current time and then start checking if the speed is greater than 50 kmh^{-1}, or greater than 100 kmh^{-1}, respectively. If the speed is greater than 50 kmh^{-1}, use the formula defined in *Equation 6 Best time calculation*, in order to estimate the time when the vehicle reaches 50 kmh^{-1}. This is because the GPS can be polled at a maximum of once per second, hence the best time needs to be interpolated.

$$"Best\ 0-50" = \frac{50}{Current\ Speed} \times time$$

Equation 6 Best time calculation

The odometer reading is the same as current trip distance however is stored as a separate variable and can be reset without affecting other calculations.

Finally, the timer, when in manual mode, has a start/stop, split-lap and reset button and operates as a regular stopwatch. When in automatic mode, the timer automatically starts counting when the vehicle speed is not zero, namely, the car is in motion, and stops when the vehicle is stationary. In both cases, the time in milliseconds is shown only when the timer is

stopped, or split has been pressed. As an accurate time reading in milliseconds can be supplied by the operating system, based on the hardware clock, however due to the slow screen refresh rate, and as a potential distraction to the driver, it is not shown whilst in normal operation.

3.6 Accelerometer Information

Figure 3-6 Accelerometer screen

The accelerometer screen displays the reading taken from the accelerometer in the IMU, and displays in a graphical format. The vertical axis represents accelerations towards the front or rear of the car; depending on how fast the vehicle is accelerating or braking respectively, and the horizontal axis represents accelerations caused by turning the vehicle.

3.7 Speed Warning

Figure 3-7 Speed warning screen

The speed warning screen allows the user to select a speed such that if it is exceeded, the Eyebot will alert the driver in some manner. It does this by enabling the critical warnings timer to also check for this event, and if the set speed is being exceeded the Eyebot will beep. As the critical warning timer checks periodically for any warning event, see section *2.4 Method*, the sound will occur periodically. However, at the present time, an error occurs whilst trying to beep, hence as a temporary solution a warning message is displayed, however this is not the most desirable solution as it is potentially dangerous for the driver to examine the Eyebot screen whilst driving.

3.8 3G Communication

Figure 3-8 3G communication debug screen

The 3G communication, or telemetry system, allows the GPS coordinates of the vehicle to be sent to a remote server, such that the driving patterns and energy usages can be examined quickly and easily by anyone in the world. This was a separate final year project and interfaces directly with the black box code, or background tasks, within the GUI. Some of the present information based on the current GPS reading is submitted to a server via a 3G modem connected to the router underneath the passenger seat. The present status of the wireless internet connection, namely whether or not the connection is active, and a sample string of what is being sent is displayed on the screen. Whilst this information is mainly for debugging purposes, it can inform the user whether or not their current trip is being logged remotely.

3.9 Tools

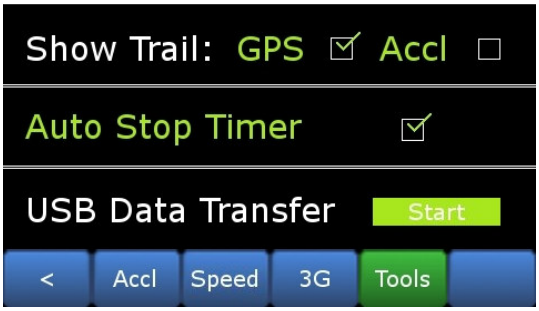

Figure 3-9 Tools screen

The tools or setup screen allows several options to be set by the user by modifying global variable. These include showing a GPS trail or acceleration trail, which modifies the operation of the respective screens such that the map, or area, is not cleared before the next position is drawn. 'Auto Stop Timer' controls whether or not the timer should automatically start and stop when the vehicle is either in motion or stationary, as described in section *3.5 Trip Information*. Finally, the 'USB Data Transfer' button allows the user to copy the logged car information by inserting a USB stick into the USB port mounted next to the Eyebot screen. The 'Start' button will either become 'Done' if the transfer was successful or an error message in the event that no device is found or a transfer error occurred, for example if there is not enough free space on the drive. If an error occurs, a timer is started that redisplays the start button after 5 seconds, allowing the user to try again.

3.10 Exit/Keypad

Figure 3-10 Exit/keypad screen

Finally, a blank button fills the last position in the menu bar which allows the program to be terminated. This is used only for debugging and allowing a modified or updated GUI to be copied over the old executable file, as a start-up script automatically starts the GUI when the Eyebot is turned on. To prevent a user accidentally exiting the GUI, a 4 digit pass code is required to terminate the program (easily obtained be examining the source code) and hence a keypad screen has been constructed. The benefit of this screen is that the keypad can be utilised elsewhere in program, for example, allowing the user to enter current electricity and fuel costs to more accurately calculate the fuel savings of the vehicle, as described in section *3.3 Vehicle Statistics*.

4 Conclusions

In conclusion, the graphical user interface now displays all relevant information to the user in a highly functional, intuitive system. The completion of the project represents the state of technologies that can be made available to consumers and demonstrates the potential for electric vehicles constructed from off the shelf components; epitomised by the fully functioning, all-electric Hyundai Getz.

4.1 Limitations and Recommendations for Further Work

The only limitation was the degree to which the GPS has been completed. It would have enhanced the experience if more control could be made in selecting which map is presently being displayed, refer to section *3.2 GPS Navigation*.

Potential improvements for the graphical user interface include finding a method of interfacing with the odometer and speedometer rather than using GPS for a more accurate reading, however this is not essential; and, attempting to fix the speed warning screen by making the Eyebot beep.

5 References

[1] The REV Project. (2009) The REV Project. [Online]. http://therevproject.com/ (accessed August 3, 2009)

[2] The REV Project. (2009) Research and development. [Online]. http://therevproject.com/vehicle-technology/research/45-vehicle-technology/48-research-dev (accessed August 3, 2009)

[3] T. Braunl. (2009) EyeBot Online Documentation. [Online]. http://robotics.ee.uwa.edu.au/eyeM6/ (accessed August 3, 2009)

[4] The Australian Electric Vehicle Asn. (2009) UWA REV project gets GULL support. [Online]. http://www.aeva.asn.au/forums/forum_posts.asp?TID=1097 (accessed October 17, 2009)

[5] D. Varma, "Eyebot Graphical User Interface," UWA, Perth, ELEC3330 Project Report, 2008. (accessed August 3, 2009)

[6] P. Bourke. (1994, Feb.) Converting between RGB and CMY, YIQ, YUV. [Online]. http://local.wasp.uwa.edu.au/~pbourke/texture_colour/convert/ (accessed October 17, 2009)