EyeBot M6 Controlled Sensor Package in a Renewable Energy Vehicle - Hyundai Getz

Ewan MacLeod

10329081

School of Mechanical Engineering, University of Western Australia

Associate Professor Thomas Bräunl

School of Electrical, Electronic and Computer Engineering,
University of Western Australia

3 November 2008

I. Abstract

The EyeBot M6 is an innovative new mobile robotics platform currently being developed at the University of Western Australia. Powered by a 400MHz ARM9 processor and SPARTAN 3E Field Programmable Gate Array, the EyeBot has a vast array of customisable sensor inputs and actuator outputs, a colour touch screen, and peripheral access to rival a computer.

The Renewable Energy Vehicle project hopes to revolutionise personal transport by building vehicles that produce no pollution, powered by a renewable energy source. The project is currently tasked with converting a five-door petrol Hyundai Getz into an electric car, by replacing existing components with batteries, an electric motor, and new instrumentation and sensors.

The EyeBot M6 construction and REV conversion projects are ongoing projects covered over several theses, with each student contributing to the development of the EyeBot or vehicle. This thesis focuses on the design and development of a Black Box data recorder and sensor package designed to run on an EyeBot installed in the REV Getz. The Black Box allows critical sensor information to be displayed on the EyeBot touch screen and saved to a removable disk for external data analysis.

II. Letter of Transmittal

Ewan MacLeod
8 Rhodes Place
MOSMAN PARK WA 6012

3 November 2008

Associate Professor Carolyn Oldham
Dean
Faculty of Engineering, Computing and Mathematics
University of Western Australia
35 Stirling Highway
CRAWLEY, WA 6009

Dear Associate Professor Oldham,

I am pleased to present this thesis, entitled *EyeBot M6 Controlled Sensor Package in a Renewable Energy Vehicle - Hyundai Getz* as part of the requirement for the Bachelor of Engineering (Mechatronics) component of the Bachelor of Computer Science / Bachelor of Engineering combined degree at the University of Western Australia.

Yours sincerely,

Ewan Angus MacLeod

10329081

III. ACKNOWLEDGEMENTS

I wish to extend special thanks to the many people who made this thesis and project possible. First, my supervisor, Associate Professor Thomas Bräunl, for his boundless advice and assistance throughout the project.

Secondly, my technical advisory panel, especially Gavin Hangchi, for his proof reading and technical assistance with everything, and Tracy Footitt, for her proof reading and grammatical skill. Without Aurora, none of this would have been possible.

To all of the people in the REV and EyeBot teams, Stephen Whitely, Michelle Ovens, and Rohan Mathew for their massive efforts with the car, David Churn, Justin Ward and Azman Yusof for their EyeBot developments, and Nicki Artman and Daksh Varma for their fantastic GUI.

Ivan Neubronner, for his electrical and EyeBot expertise, Ken Fogden, for his mechanical assistance with the car, Linda, for always being on the ball, and all the guys at work, for covering my shifts.

Most importantly, my support network for this very difficult and busy period of my life. My girlfriend, Sarah, for her love and patience. My friends for their patience and for keeping me sane. Finally, my family for their love and understanding of the late nights, early mornings and lots of skipped meals.

Thank you.

IV. Table of Contents

V. List of Figures and Tables

VI. NOMENCLATURE

AC	Alternating current
ADC	Analogue-to-digital converter
Black Box	The EyeBot sensor interface and data logging system
CPU	Central Processing Unit
DC	Direct current
DIO	Digital input/output
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
GUI	Graphical User Interface
PWM	Pulse-Width Modulated
REV	Renewable Energy Vehicle
REVBB	Renewable Energy Vehicle Black Box software
RoBIOS	Robot Basic Input/Output System
UART	Universal asynchronous receiver/transmitter
USB	Universal Serial Bus

1. INTRODUCTION

The electric car was among some of the earliest automobiles, predating Karl Benz's four-stroke internal combustion engine used in the first production automobiles. With the recent increase in environmental concern due to vehicles and the improvements in energy storage technology, electric vehicles are increasing in popularity.

Over 80 million barrels of oil are consumed worldwide every day, a figure expected to increase to over 100 million within the next 10 years (OPEC, 2008). At our current consumption, 32.2 million barrels are for road transportation alone. This demand is expected to grow even past the point of peak oil, where the worldwide oil production reaches a maximum and begins a terminal decline. This, combined with pollution issues and other factors, has called for alternative energy sources to replace oil. Renewable energy sources currently make up around 6% of world energy production, mostly from hydroelectricity and biomass fuels. This proportion is predicted to increase over the next 20 years (OPEC, 2008).

The Australian average unleaded petrol price has doubled in the nine years since October 1999 (FuelTrac, 2008). Despite this, large family cars and four-wheel-drives continue to be popular, even though many of these cars carry only the driver. There are over 11 million passenger vehicles in Australia, which each travel around 14,600km per year and use 25% of all energy consumed in Australia (ABS, 2008). A combination of these factors, the threat of peak oil, and ever present concern about carbon-dioxide pollution is creating a strong drive for more efficient petrol and alternative fuel vehicles, both in Australia and worldwide.

Very few vehicle manufacturers provide alternative fuel source options to consumers, but the strong uptake of hybrid vehicles such as the Toyota Prius has shown that there is a growing market. In recent years, more and more all-electric vehicles are becoming commercially available, such as the Tesla Roadster and the MINI E. Both of these production cars are electric conversions of existing models of the Lotus Elise and MINI Cooper respectively. While the vehicles look similar, the differences between the internal combustion engine and electric engine versions are extensive. The electric vehicles have sensors to monitor electric systems that replace the function of combustion engine components, such as battery state-of-charge as a new fuel gauge. In a production vehicle, these sensors are fully integrated into the vehicle design, monitored by the electronic control unit, and state information is displayed on the dashboard. However, on custom conversions these sensors need an external device to replicate the dashboard functions. Embedded systems provide the perfect solution.

This thesis focuses on the design and development of a Black Box data recorder and sensor package designed to run on an EyeBot M6 embedded system. The system will be integrated into the Hyundai Getz being converted to an electric vehicle by the University of Western Australia's Renewable Energy Vehicle (REV) project. The Black Box allows critical sensor information to be displayed on the EyeBot LCD touch screen and saved to a removable disk for external data analysis.

1.1 The EyeBot M6

The EyeBot M6 is a multipurpose embedded system developed by the EyeBot team within the Centre for Intelligent Information Processing Systems at the University of Western Australia.

Embedded systems are a common part of everyday life, from watches to washing machines, mp3 players to nuclear power plants. An embedded system is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function (Barr, 2008). Embedded processors span the range from simple 4-bit microcontrollers, like those at the heart of a greeting card or children's toy, to powerful custom 128-bit microprocessors and specialized digital signal and network processors.

The current generation of EyeBot controllers are being used in the LabBot, a differential-drive wheeled robot with a forward-mounted "EyeCam", in Electrical and Electronic Engineering laboratories at the University of Western Australia (Bräunl, 2006). These EyeBot controllers are also used to power a variety of wheeled robots (including soccer-playing robots, the OmniBot series and several model cars), walking robots (two- and six-legged), "balancing" robots (the BallyBot series), and both a UAV (Unmanned Aerial Vehicle, the FlyeBot) and AUV (Autonomous Underwater Vehicle, the Mako) (Bräunl, 2007).

The EyeBot M6 is being developed as a generic embedded controller adaptable for a range of uses. These include undergraduate teaching, undergraduate and postgraduate research, and project development. Ideally, the EyeBot M6 will replace the existing modules in the EyeBot robots, and allow for a more advanced design and simpler user interface for students and developers.

The EyeBot Mark-6 (M6) controller uses a powerful combination of an ARM9 processor single board computer, the Gumstix Connex, with a Xilinx Field Programmable Gate Array (FPGA). The FPGA handles sensor and actuator pre-processing and low-level image processing, freeing up the central processing unit (CPU) for high-level tasks. The M6 is equipped with stereo cameras, a colour touch screen, numerous sensor inputs and actuator outputs, and several peripheral connectors (Bräunl, 2008). The EyeBot development environment is familiar to Linux developers since it is based on a custom Linux build. EyeBot applications can be rapidly developed in the C programming language using the Robot Basic Input/Output System (RoBIOS) libraries on any system, and compiled using the EyeLin cross compiler. Programs can be copied to the EyeBot via the network or on a USB flash drive. Programs can be executed remotely through the network interface or using M6-Main, the default user interface and monitor program.

1.2 Renewable Energy Vehicle

The REV project aims to “revolutionise personal transport”, by building vehicles that produce no pollution, are powered by electricity from any plug point, and are viable to both the performance and commercial markets (Mathew, 2008).

The REV project started in mid-2004 under the supervision of Dr. Kamy Cheng and Dr. Lawrence Borle. The early REV projects involved building a solar and hydrogen fuel-cell vehicle with a custom designed chassis to replicate the Tamagawa University solar car project, which successfully drove a similar vehicle across Australia (Tamagawa, 2003). In 2008, under the supervision of Thomas Bräunl, our focus has been on converting a commercial petrol vehicle into an electric vehicle. Other members of the REV team have led research into developing a photovoltaic charging station to power the electric vehicles. While the vehicles are not charging, the charging station will contribute renewable energy to the power grid.

The first vehicle to be converted in the REV project is the Hyundai Getz. The 2008 S-class model is a 5-door, 5-speed manual transmission petrol vehicle. This economy car is available for around $15,000 new and claims to have 16 km/L fuel efficiency, which is average for a small car. By current prices, this relates to a cost of approximately $9.05 per 100km. The Getz won Australia’s Best Small Car in 2003 and 2005 (Australia's Best Car, 2003; Australia's Best Car, 2005). It was judged to have “above average” handling and performance and “average” safety features. Driver and front passenger airbags are a standard feature, as are the electronic windows and mirrors, remote central locking, and immobiliser.

Using an array of 45 Thunder Sky lithium iron phosphate batteries (LFP) connected in series, the vehicle is given a 144 V main voltage. These 90 Amp-hour batteries, together with a series-wound DC motor, and a Curtis 144 V 500 A controller combine to provide the car with a top speed of around 100 km/h and a range of 80-100 km. A full charge with the 144 V 15 A Zivan NG3 battery charger only takes a few hours and costs $0.22 at current mains power prices (Mathew, 2008). Once the photovoltaic charging station is installed, there is no cost for charging.

This conversion project used cheaply available off-the-shelf components. A home user can perform a similar conversion.

1.3 REV Black Box Project

This project focuses on the design and development of an EyeBot M6 controlled Black Box data recorder and sensor package for the REV Getz. The Black Box allows critical sensor information to be displayed on the EyeBot LCD touch screen and saved to a removable disk for external data analysis.

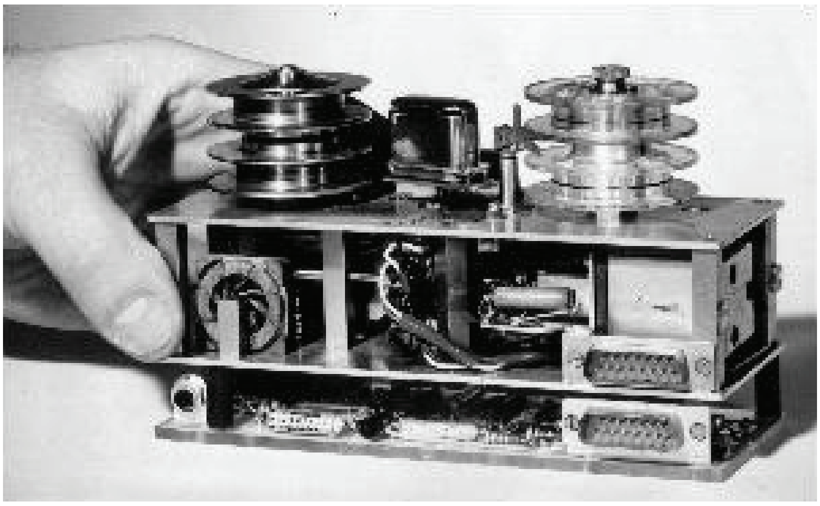

FIGURE 1.1: ARL FLIGHT MEMORY UNIT (THOMAS, 2007)

A black box is a device, system, or object viewed primarily in terms of its input and output characteristics. In common use, a black box refers to a flight data recorder (FDR), an Australian invention by Dr David Warren with the first prototype produced in 1957 (Fig. 1.1). Since then, FDRs and cockpit voice recorders (CVR) have become compulsory in all aircraft. Usually combined in one unit, the FDR and CVR record time-stamped flight telemetry and performance data in a crash-resistant box that can be recovered and used in aircraft mishap analysis. The FDR can also be used to study non-crash issues, such as engine performance, air safety, and material degradation. Most FDRs can record around 17-24 hours of data before overwriting existing data.

The Black Box project is inspired by aircraft FDRs but is not designed to be used in vehicle crash analysis, or even survive a severe crash. However, the REV Black Box is designed to record sensor data, vehicle telemetry, and performance data for analysis and further electric vehicle research.

2. Project Scope

This project aims to determine if it is possible to use an EyeBot M6 controller as a sensor package interface and black box-based data logger for a Renewable Energy Vehicle.

This project will interface the EyeBot M6 controller to the REV Getz as an embedded system controller for the new sensor packages installed during the conversion process. The Black Box package will be interfaced with a variety of sensors to record drive telemetry and monitor critical systems within the vehicle. Several key design requirements drove the development of the Black Box package. These requirements are:

- The package must be able to rapidly record the sensor values at high speed without incorrect readings
- The package must detect when any errors are registered by the sensors and display these errors to the driver
- The package must save the sensor readings taken over an extended period of time to a removable USB flash drive to allow for external analysis

Package portability must be maximised to enable installation in any future REV project vehicles, without requiring modification to the overall program algorithm and only minor changes to accommodate the differences in vehicle sensors.

Finally, the package must have a modular design allowing a graphical user interface (GUI) to display the current sensor readings to the driver without affecting the operation of the Black Box. The GUI will be simultaneously developed by another team within the REV group. Supervising the development is also a requirement of the Black Box project.

The Black Box package must be tested for completeness and correctness. Its performance must be evaluated to determine if the EyeBot M6, together with the Black Box software, is a suitable sensor logging package for a Renewable Energy Vehicle.

The Black Box package is the first application developed on the EyeBot. As we identified problems during development, we passed the details to the EyeBot development team. We also provided debugging and development assistance. Through this project we will demonstrate the EyeBot M6 as a successful general-purpose embedded system for a variety of situations.

Members of the REV group worked under one broad goal: get the Getz conversion project finished, licensed, and drivable on Australian roads. We have provided assistance installing sensors and other components, ensuring that the vehicle is safe to operate, and supervising other students in the REV group.

3. EyeBot M6 Touch screen upgrade

While researching the EyeBot hardware, a more suitable LCD touch screen was discovered: this new module (Samsung LTE430WQF0) has a larger (4.3”) widescreen display, higher resolution (480 × 272 pixels), maximum 24-bit colour, and improved brightness characteristics compared to the existing screen. The new LCD module is also available at a lower price and has existing documentation for interfacing to the Gumstix Connex board at the heart of the EyeBot M6 (Gumstix, 2008). The display dimensions (105.5 × 67.2 × 3.9mm) also match the 108mm EyeBot width more closely.

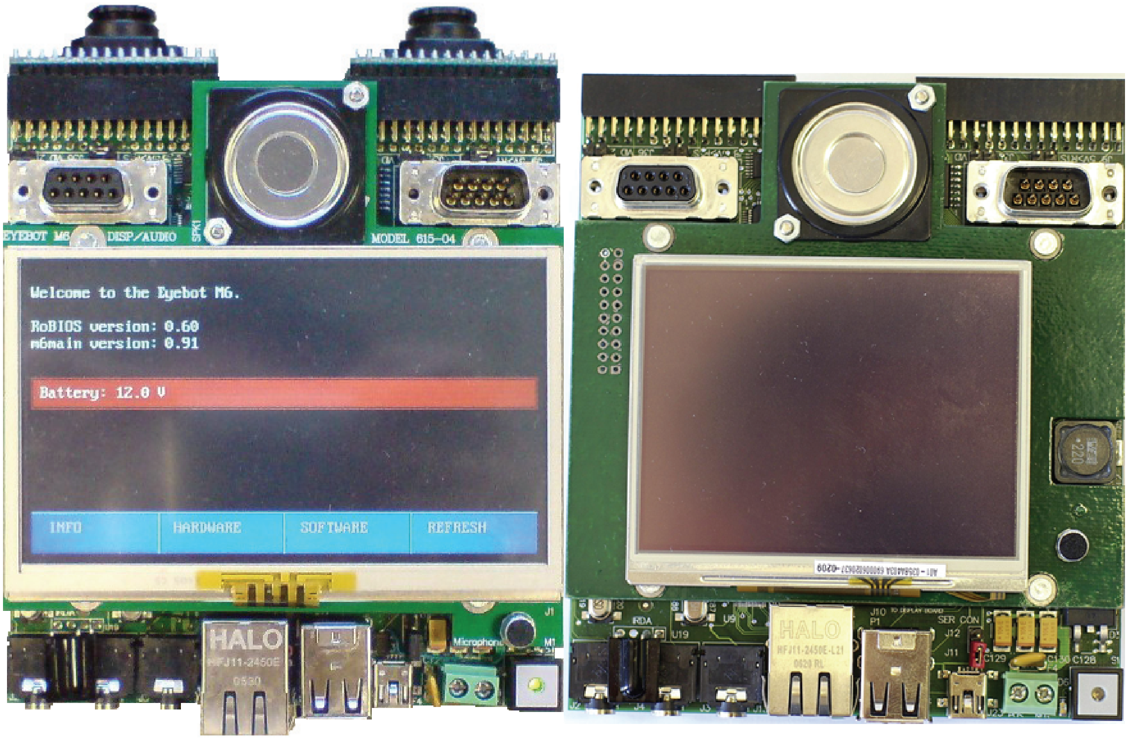

FIGURE 3.1: EYEBOT M6 WITH NEW TOP BOARD VS OLD TOP BOARD

The EyeBot design has a separate “top board” containing the touch screen, speaker modules, and a standardised connector. The EyeBot LCD controller is configurable for many different LCD modules, therefore restricting the EyeBot to one specific module is undesirable. On start-up, the automatic detection method identifies the top board device ID. The Linux kernel uses this ID to look up the configuration, and then configures the LCD and other devices on the top board.

In 2007, Sommer and Hintermann working from the Technische Universität München in Germany, developed the touch screen (Sommer, 2007) and LCD (Hintermann, 2007) routines for the existing module. We have since modified the LCD and touch screen routines to accommodate the new screen.

The `LCDInit()` routine was previously used to initialize the frame buffer device, allocating memory and storing the frame buffer settings to the memory address. This required users to perform an `LCDInit()` call in their program before any other call requiring the LCD module, else a segmentation fault would occur causing the program to crash with no error message displayed. The hardware LCD module and its associated parameters can be changed between each call by assigning the memory and frame buffer parameters on each `LCDInit()` call. `LCDInit()` also had a matching `LCDRelease()` function, to close the frame buffer and free the memory it required, and again was required during the termination stage of every user program.

It is extremely unlikely that a user would require a new LCD module to be installed during program execution, or even while the EyeBot is powered on. The functions of `LCDInit()` have been replicated in the boot routine, so the frame buffer is initialised once when the EyeBot is booted and released on power down. The `LCDInit()` and `LCDRelease()` functions have been removed.

4. SENSOR INTEGRATION

The primary objective of the Black Box integration is to monitor currently installed or possible future sensors. There are an almost unlimited number of sensors that could be installed into the vehicle. However, there are only a limited number of inputs on the EyeBot that we can monitor these sensors from. This section will cover the sensor selection and installation process, highlighting key issues faced.

4.1 SENSOR SELECTION

The EyeBot M6 was designed in 2006 (Blackham, 2006) and since then has only undergone minor hardware changes, such as the replacement top board detailed in Chapter 3. The controller was designed for robotics applications requiring a large number of actuator outputs. It was not designed for user input, so without using an external sensor interface, it is restricted in the number and types of inputs that can be read.

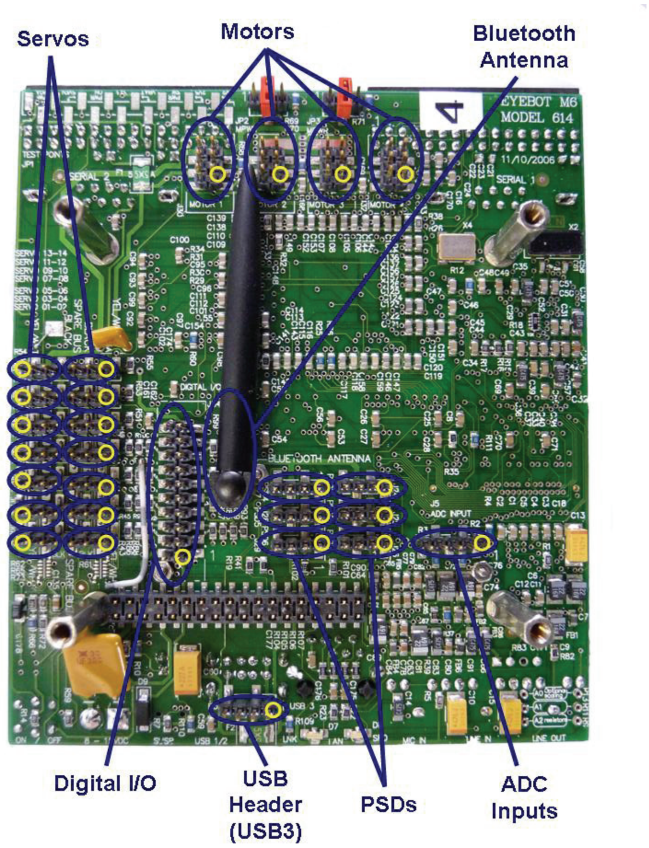

FIGURE 4.1: EYEBOT M6 - BOTTOM VIEW

The EyeBot has a wide range of sensor input and actuator outputs available to use, some of which are highlighted in Fig.4.1. Pin 1 for each connector is indicated by the yellow circle.

Most of the vehicle sensors are switches with binary states: these show if there is an error or if there is not. This type of sensor is perfectly suited to the two banks of eight digital inputs/outputs (DIO). Each bank can be independently configured to be in either input or output mode, setting all eight pins on that bank into that mode. This is achieved through the RoBIOS function call `OSLatchBankSetup(bank_number, direction)`. However, four pins on the second DIO bank, bank 1, are shared with the second EyeCam in output mode. The second camera is not currently being used, but in the future, both cameras will be installed, forcing DIO bank 1 to output mode and restricting the Black Box to the eight digital inputs provided by DIO bank 0.

DIO Pin	Sensor	Status
Bank 0 Pin 0	Main battery low (Throttle Retarder active)	Critical
Bank 0 Pin 1	Brake vacuum pump failure	Critical
Bank 0 Pin 2	Fuel cap switch active – charging cable attached	Critical
Bank 0 Pin 3	Inertia switch triggered	Critical
Bank 0 Pin 4	Ignition	Critical
Bank 0 Pin 5	Emergency Stop button pressed	Critical
Bank 0 Pin 6	Tachometer	
Bank 0 Pin 7	Speedometer	

TABLE 4.1: DIGITAL INPUT MAP

The REV Getz has been installed with a sensor relay box that does not allow the vehicle to drive unless all "critical system" sensors are in an error-free state. The relay box, developed by another team within the REV project, uses double-pole double-throw (DPDT) relays, with one pole to return the sensor status to the EyeBot for each relay sensor. The second pole of each relay is connected in series with the electric motor controller, so that if any relay detects an error state, the controller will not get the signal to drive and the motor will not move. This ensures that the vehicle is safe to operate before moving.

Table 4.1 shows the digital inputs monitored by the EyeBot. Six of the eight systems are "critical systems". These are key systems to display to the driver and integrate into the Black Box.

The two other digital inputs are used by the tachometer, which measures engine revolution speed, and the speedometer, measuring the vehicle speed in km/h. Each of these sensors output a digital pulse four times per revolution. Ideally, the digital input bank will continually read and count pulses to calculate the frequency. From this, we can derive the motor and wheel speed.

DIO Pin	Sensor	Status
Bank 1 Pin 0	"Check engine" light	
Bank 1 Pin 1	Unused	
Bank 1 Pin 2	Unused	
Bank 1 Pin 3	Unused	
Bank 1 Pin 4	*Shared with second camera*	Reserved
Bank 1 Pin 5	*Shared with second camera*	Reserved
Bank 1 Pin 6	*Shared with second camera*	Reserved
Bank 1 Pin 7	*Shared with second camera*	Reserved

TABLE 4.2: DIGITAL OUTPUT MAP

As shown in Table 4.2, digital output pins 4 to 7 are shared with the second digital EyeCam that will be installed in the future. These pins have been reserved to allow for this expansion. If they were used with a second camera installed, the image may become corrupted or the outputs may give false readings.

The REV Black Box project primarily deals with reading data from sensors, hence only one digital output has been identified, the "check engine" signal. This output is activated when one of the critical systems is detected to be in an error state. It will display the "check engine" light on the instrumentation cluster in front of the driver. Having an indicator within the visual range of the driver will allow a faster response time to detect errors.

ADC	Signal	Connection
ADC0	Main battery voltage (144 V)	User-defined
ADC1	Main line current (instantaneous)	User-defined
ADC2	Throttle position	User-defined
ADC3	Alternate battery voltage (12 V)	Pre-defined

TABLE 4.3: ANALOGUE-TO-DIGITAL CONVERTER MAP

The EyeBot M6 has an on-board 10-bit analogue-to-digital converter (ADC) that allows four analogue signals to be converted into 1024 discrete digital values. This is handled by the UCB1400 chip, a stereo audio codec with touch screen and power management interfaces. The EyeBot configuration only allows for three user-defined analogue inputs to the ADC. A fourth is hard-wired to the EyeBot power supply voltage, in our case, the 12 V alternate battery in the car.

The main battery voltage and main line current, along with several other statistics, are measured with a commercial battery monitor, the TBS Electronics E-Xpert Pro. A future goal of the Black Box project is to replicate all functions of the E-Xpert Pro battery monitor.

USB	Device	Mounting
USB1	USB Global Positioning System receiver	Internal
USB2	Unused	Internal
USB3	USB flash drive	On dash
USB Slave	Unused	Internal

TABLE 4.4: UNIVERSAL SERIAL BUS MAP

The EyeBot M6 has two USB host ports on board and a third USB host has been provided through USB header pins. Each of the USB 2.0 host ports can control a separate device allowing up to three USB devices, with appropriate pre-installed drivers, to be connected to the EyeBot at once. Over the course of this year, the USB flash drive and USB-serial drivers were ported to the EyeBot platform under other projects within the EyeBot development group. The EyeBot also offers a dedicated USB 1.1 slave port, which allows the controller to act as a USB device. Currently, this only allows an external USB controller, such as a computer, to power the EyeBot. With further operating system device drivers, it could allow full EyeBot controller interaction.

For the Black Box project, the EyeBot has been fitted with a GlobalSat BU-353 USB global positioning system (GPS) receiver. This module uses the NMEA 0183 protocol to provide ASCII serial communication to the EyeBot, including the global position, speed, and heading.

The USB header pins (USB3) have been connected to a USB port mounted within the centre fascia panel of the dashboard. This allows a USB flash drive to be inserted and removed easily.

Serial	Device	Status
Serial1	Debug console – 115,200bps	Available
Serial2	Three-axis accelerometer – 56,000bps	Available
	GSM modem – 19,200bps	Future project

TABLE 4.5: SERIAL (RS-232) PORT MAP

The EyeBot M6 also boasts two serial (RS-232) connectors, a "female" port for Serial 1 and the complementary "male" port for Serial 2. Under the current configuration, the EyeBot M6 will print all console information to the Serial1 port, which can be read by a terminal monitor configured with the correct parameters. This debug serial connection allows a user to monitor the boot procedures and, after a successful boot, log in and gain access to a console. The console can provide root access to the operating system for configuration when an Ethernet network is not available or incorrectly configured.

The second serial port is configurable for a wide range of devices. So far, only a SparkFun Electronics SerAccel v5 triple-axis serial accelerometer has been tested with the EyeBot. Due to the Linux kernel controlling serial interfaces, it is assumed that any device, with the correct drivers, will work as expected. In future, the installation of a GSM (Groupe Spécial Mobile or Global System for Mobile communications) modem will allow a remote uplink that provides two-way communication with the EyeBot while it is driving. The GPS position can be transmitted via the serial connection and mobile phone network to provide the real-time vehicle location to anyone watching on the Internet. The GSM modem was a late addition to the possible sensor network and has not been installed as part of the Black Box project.

To interface with the debug port, the receiver must use 115,200 bits per second (bps) in 8-bit bytes, no parity, zero stop bits (8N0), and no flow control. The serial accelerometer is configured for 56,000 bps and uses 8-bit bytes, no parity and one stop bit (8N1) with no flow control. The configuration for the Wavecom WMO2 GSM modem has not been tested, but the default configuration admits rates from 2,400 to 19,200 bps, uses 8-bit bytes, no parity, one stop bit (8N1), and hardware flow control.

Connection	Device	Status
Motor 1	Fuel dial – State-of-charge indication	Untested
Servo 1	Tachometer dial	Untested

TABLE 4.6: UNTESTED INSTRUMENTATION CLUSTER CONNECTIONS

Two further connections have been made between the EyeBot and the instrumentation cluster. At the time of installation, very little was known about the electrical configuration of the instrumentation cluster. Some assumptions were made based on the sensor signals and other instruments within the cluster to determine the best method of using the EyeBot to drive aspects of the cluster.

The fuel dial was originally used to show the amount of petrol in the fuel tank, measured by the fuel sender. The tachometer reading, a component of the internal combustion engine, was displayed on the tachometer dial in the instrumentation cluster. Since these have both been replaced with electrical counterparts, a new method of driving the instrument dials should be determined. The EyeBot monitors the battery state-of-charge and tachometer reading and, since the motor and servo controllers output a signal similar to the fuel sender and tachometer signals, these have been selected as trial candidates for driving the dials within the instrumentation cluster.

Normally, the LCD screen lies along the top of the EyeBot and is connected by a short zero insertion force cable. This layout was not suitable for installation into the Getz and so the electrical workshop modified the EyeBot to use a longer cable, allowing the LCD to be mounted perpendicularly to the EyeBot. This was installed in the centre fascia, replacing the normal stereo position.

4.2 SENSOR INTEGRATION AND ELECTRICAL CHARACTERISTICS

The sensor installation was primarily the responsibility of other members of the REV group; however, to design and construct the EyeBot-to-sensor connections, the team required the physical and electrical characteristics of the EyeBot sensor interface. At the commencement of this project, these had not been fully explored and documented. This section will cover the electrical wiring design of the Getz EyeBot interface and document the EyeBot electrical characteristics.

The EyeBot within the vehicle is powered by the 12 V car battery in parallel with most of the vehicle electronics. The 12 V battery has three different power supply states for devices, selectable by the ignition key:

- The “ignition” state will only provide power to devices when the key is in the “on” position (the normal position for driving); the brake vacuum pump and the power steering motor are powered by ignition power
- The “accessories” state will only provide power if the key is in the “accessories” or “on” position; the stereo and cigarette lighter socket are powered by accessories power
- The “permanent power” state provides a 12 V connection regardless of key position; the safety systems and the EyeBot have a permanent power connection

The EyeBot and many other systems have been installed into the dashboard and require connections to components within the engine bay. A hole was drilled in the chassis and a conduit installed to protect the cables running between the engine bay and the vehicle interior. The limited size of the conduit restricted the number of cables that could be installed. The EyeBot required 12V power, an eight-core cable to the sensor relay box, and wires to the speed-sensor, throttle position sensor, and tachometer. Each of these wires was labelled and documented by the sensors team to illustrate each function. Within the vehicle, the wires to the EyeBot were terminated using a nine-pin plug, allowing the EyeBot to be unplugged and removed for repairs or upgrades without affecting the installed cables. These two nine-pin plugs, visible in Fig.4.2, are hidden inside the dashboard behind the EyeBot and centre fascia panel.

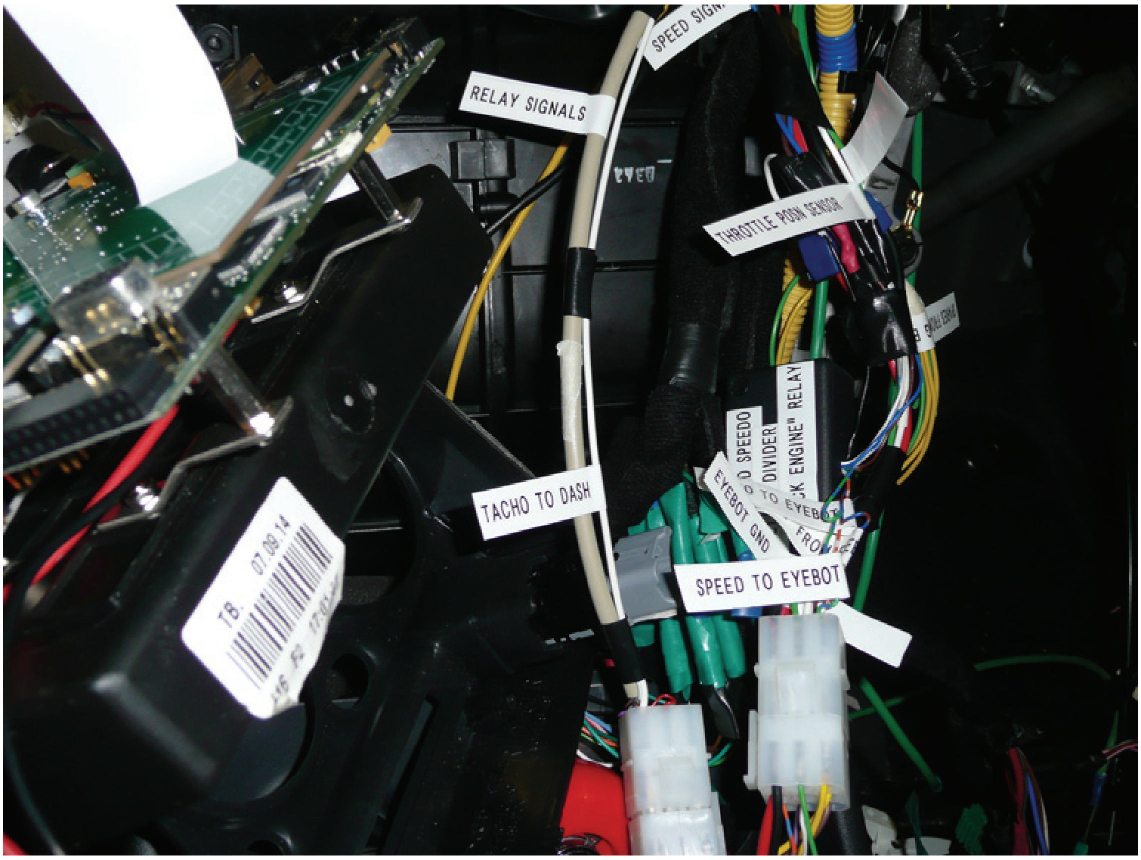

FIGURE 4.2: EYEBOT WITH LABELLED SENSOR WIRES

The EyeBot pin layout has been designed to use standard connectors for each sensor or actuator bank. The digital input/output pins are in two rows of ten pins each, spaced to accommodate a 20-pin insulation displacement connector (IDC) to connect to a 20-wire ribbon cable. The pin layout is shown in Table 4.7.

Digital Input/Output pin layout			
Ground	20	19	Ground
Digital IO Bank 1, Pin 7	18	17	Digital IO Bank 1, Pin 6
Digital IO Bank 1, Pin 5	16	15	Digital IO Bank 1, Pin 4
Digital IO Bank 1, Pin 3	14	13	Digital IO Bank 1, Pin 2
Digital IO Bank 1, Pin 1	12	11	Digital IO Bank 1, Pin 0
Digital IO Bank 0, Pin 7	10	9	Digital IO Bank 0, Pin 6
Digital IO Bank 0, Pin 5	8	7	Digital IO Bank 0, Pin 4
Digital IO Bank 0, Pin 3	6	5	Digital IO Bank 0, Pin 2
Digital IO Bank 0, Pin 1	4	3	Digital IO Bank 0, Pin 0
5 V signal supply	2	1	3.3 V signal supply

TABLE 4.7: DIGITAL INPUT/OUTPUT PIN LAYOUT

The digital input/output is handled within the EyeBot by a 74LPT16245 16-bit bidirectional transceiver microchip. Researching the specification for this chip, the electrical characteristics have been determined. Reading from an input pin, the chip will register a logical “high” for a voltage of between 2.2 V and 5.5 V and a logical “low” for a voltage between −0.5 V and 0.8 V. As an output, the chip will typically supply 3.0 V at output “high” and 0.2 V at output “low”.

From this, in conjunction with the sensor development team, we have determined that the EyeBot will use the 3.3 V supplied from pin 1 of the digital input/output pin layout (Table 4.7) in the sensor relay box to interface back to the input pins. The 3.3 V supplied is in the middle of the logic “high” range, so a small fluctuation would not affect the readings or damage the EyeBot.

Under normal conditions, the EyeBot will read the 3.3 V on each input pin and register a logic “high”, indicating no error. If the sensors indicate an error, 0 V would be read on the input pin and a logic “low” would be read, indicating this error to the user. Operating in this method provides a failsafe system – if a wire was severed, it would then register a logic “low” and indicate an error.

The speed-sensor and tachometer output 12 V pulses. These signals are each passed through a voltage divider (Fig.4.3) to reduce the maximum pulse voltage to within the logic “high” region.

Analogue-to-digital converter pin layout	
0	5 V signal supply
1	Analogue-to-digital converter 0
2	Analogue-to-digital converter 1
3	Analogue-to-digital converter 2
4	Analogue ground

TABLE 4.8: ANALOGUE-TO-DIGITAL CONVERTER PIN LAYOUT

The analogue-to-digital converter (ADC) has a five-pin horizontal layout with connections as shown in Table 4.8. The ADC is controlled by the UCB1400 microchip, which is an audio codec controller that also includes a touch screen controller and a power management monitor. The chip specification states that for each of the four 10-bit ADCs, a “typical voltage” of 7.5 V is expected. Tests performed show that the minimum recorded voltage is 0 V and corresponds to a reading of zero. A voltage of 8.3 V corresponds to the maximum reading of 1023. Each increase of 0.81 mV in the supply corresponds to an increase of 1 in the ADC reading.

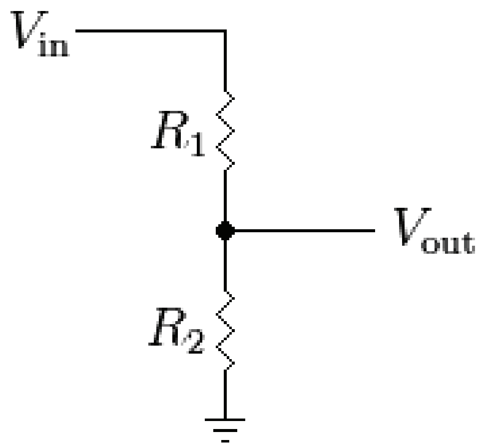

FIGURE 4.3: RESISTIVE VOLTAGE DIVIDER

The EyeBot has been constructed to use an independent voltage divider for each ADC input. This allows higher voltages to be measured, such as the 8-15 V EyeBot power. The resistive voltage divider, shown in Fig.4.3, divides the voltage input V_{in} between the two series resistances, R_1 and R_2. The output voltage, V_{out}, is a proportion of V_{in} defined by R_1 and R_2 according to the formula in Equation 4.1:

$$V_{out} = \frac{R_2}{R_1 + R_2} \times V_{in} \qquad (4.1)$$

For the three input ADC pin voltages, V_{in}, the EyeBot contains 10 kΩ resistors in the R_1 position and sockets to install an optional scaling resistor at R_2 on the underside of the bottom board. With the appropriate resistor installed, it allows a large range of voltages to be measured by the ADC without exceeding the maximum and potentially damaging the EyeBot. The supply voltage ADC has R_1 = 100 kΩ and R_2 = 27 kΩ as part of the construction, reading a 12 V supply voltage at 2.55 V (corresponding to an ADC reading of 315).

One of the input ADC pins measures the main battery voltage, nominally 144 V, significantly higher than the maximum ADC voltage. The main battery signal is passed through a 10 to 1 voltage divider, reducing the peak voltage to approximately 17 V and the nominal voltage to 14.4 V. Using Equation 4.1 above, a maximum voltage of 7 V can be obtained by installing 7 kΩ resistors for R_2 in the optional scaling resistor socket. The remaining ADC inputs use a 12 V signal but, to ensure high resolution, 7 kΩ resistors have been installed in the optional scaling resistor sockets.

Motor controller pin layout			
Encoder channel A	6	5	Encoder channel B
5 volt	4	3	Ground
Motor +	2	1	Motor −

TABLE 4.9: MOTOR CONTROLLER PIN LAYOUT

A late addition to the project was an attempt at using a motor controller to drive the fuel gauge within the instrumentation cluster. The EyeBot motor controllers use a six-pin connector in a 2×3 layout, with two pins to provide motor drive signal and two pins used for the onboard shaft encoders of the recommended EyeBot motor. The motors are powered using a pulse-width modulated (PWM) signal generated by the EyeBot FPGA with a selectable peak voltage. This is set by connecting the jumper pins (JP2 for motor controller 1 and 2, JP3 for motor controller 3 and 4) in one of two configurations, pins 1 and 2 to provide a peak PWM voltage of 5 V, and pins 2 and 3 to provide the EyeBot power voltage, 12 V.

The sensor team determined that the fuel gauge was powered by a 12 V PWM signal, suitable for connection to a motor controller with the jumper set to the 12 V supply voltage. The fuel gauge has been connected to motor controller 1 onboard the EyeBot. It has not had any tests performed to ensure operability and is left for further work and research into this project.

5. BLACK BOX PROGRAM

The Black Box program lies at the heart of the Black Box project. This user program has been designed to run continuously while the EyeBot is powered on. The program uses the Robot Basic Input/Output System (RoBIOS) to interface to the EyeBot sensors, actuators, and other features. This chapter covers all aspects of the Black Box software, including design, problems faced, and experimental benchmarking.

5.1 DEVELOPMENT ENVIRONMENT

The Renewable Energy Vehicle Black Box (REVBB) program has been written in C and uses the Robot Basic Input/Output System (RoBIOS) libraries, custom to the EyeBot. The RoBIOS libraries provide a set of high-level method calls to control low-level EyeBot functions reducing the technical detail required by users developing programs. This allows rapid user program development that can focus on algorithms and structure rather than low-level input/output manipulation. The libraries can be included by any user program with the pre-processor command `#include <eyebot.h>`.

The RoBIOS libraries are being developed by the EyeBot team within the Centre for Intelligent Information Processing Systems (CIIPS) within the school of Electrical, Electronic and Computer Engineering at The University of Western Australia, in conjunction with students at other universities, such as Sommer and Hintermann from Technische Universität München. The libraries have been incrementally developed, with each successive project adding new functions, fixing errors, and improving performance and user accessibility. The volatility of the RoBIOS libraries was a challenge for program development, requiring some features that were still being developed (Justin Ward's GPS functions) and with each RoBIOS update, fixing new problems that arose from the updated access methods for the routines and features. The RoBIOS libraries are well documented and the documentation is updated during each stage of work. As each function is developed internally, the documentation is changed to reflect the changes and released online before the libraries. Maintaining only the latest RoBIOS version's documentation compounded the difficulties faced by these rapid library changes during the development of the Black Box program.

The RoBIOS install utility for Windows installs the RoBIOS-Linux for ARM-based EyeBot Controllers or "EyeLin" development environment. This includes the ARM9-processor compilation utilities specific to the EyeBot, the RoBIOS libraries, sample demo programs, and settings for Dev-C++ (the integrated development environment used by some of the 2006 EyeBot team). The package also includes a key part of a Linux-like environment for Windows, called "Cygwin". The Cygwin Dynamic Link Library (DLL) included, `cygwin1.dll`, acts as a Linux API emulation layer providing substantial Linux API functionality (Cygwin, 2008).

This project used EyeLin and Dev-C++ to compile programs on a Windows Vista 64-bit operating system, marking the first use of EyeLin under either Vista or a 64-bit architecture. After EyeLin was installed, programs that compiled with no errors or warnings on other machines were not compiling on the Vista 64-bit machine. Researching the compiler error messages showed an incompatibility with Vista in an earlier version of Cygwin, fixed in a subsequent release. The `cygwin1.dll` included with each version of EyeLin was created in November 2006, before the release of Windows Vista and the common use of 64-bit architecture CPUs in personal systems. After updating the Cygwin DLL file, most of the errors subsided. The remainder were fixed by setting environment variables to their recommended settings. This updated `cygwin1.dll` file will be included in future EyeLin releases.

The EyeBot network interface is controlled from within the Linux kernel and configured during the EyeBot boot procedures. The Dynamic Host Configuration Protocol (DHCP) client running on the EyeBot allows for a dynamic IP-address allocation when connected to a DHCP-enabled network, such as the development network within the REV laboratory. Any computer that is connected to the network, with the appropriate address, can access the EyeBot through a secure shell (SSH) connection from a Linux or Mac terminal or through PuTTY on Windows.

Username	Password	Privileges
root	<no password>	Super-user, unlimited access
eyebot	eyebot	Default user account
rev	revrev	User account for REV team/development

TABLE 5.1: EYEBOT DEVELOPMENT USER ACCOUNTS

Connecting remotely to the EyeBot with username and password gives the user a terminal connection with the full privileges allowed by their account. User accounts can be created by a "root" account and given their own workspace, while restricting access to critical system files needed for EyeBot operation. User programs can be copied to the EyeBot workspaces through either a secure copy (SCP) or file-transfer protocol (FTP) client, such as WinSCP in Windows. Programs can be executed remotely through the SSH terminal or directly on the EyeBot through M6main.

5.2 BLACK BOX DEVELOPMENT

Several key aspects drove the Black Box program design. One design requirement was that the Renewable Energy Vehicle Black Box (REVBB) would run continually during EyeBot activation, to record sensor readings at the highest speed possible, and record these values to a removable USB flash drive in a human-readable method. Each record should be stored with the time and date that the reading was taken to allow for external analysis. The Black Box should also allow a graphical user interface to display current sensor readings to vehicle occupants, and be portable to allow for simple modification and installation into future REV vehicles.

REVBB is separated into two files, a header file `REVBB.h` and the program file `REVBB.c`. The header file contains definitions of the libraries included, the global variables required, and the data structures developed for the REVBB program. The header file can be given to other developers to concurrently develop their own program for integration; the graphical user interface (GUI) is one example.

The `bb_data` data structure contains variables to store readings from each sensor within the logging package. An array of `bb_data` structures is used to store previous sensor readings that have not yet been written to disk. Each entry also contains a `GPSdata_t` structure and `ACCdata_t` structure to store GPS and accelerometer values respectively. `bb_data` is defined as follows:

```
typedef struct {
    int datestamp;
    int timestamp;

    int batt_low;       //Critical - Error state is LOW (0)
    int brake_fail;     //Critical - Invalid -1
    int fuel_door;      //Critical - Normal HIGH (1)
    int inertia;        //Critical
    int ignition;       //Critical
    int e_stop;         //Critical
    int tacho;
    int speedo;

    int check_engine //Output

    float main_batt_v; //Analogue
    float main_batt_i; //Analogue
    float throttle;    //Analogue
    float alt_batt_v;  //Analogue

    float soc; //State-of-charge, percent

    GPSdata_t gpsd; //GPS
    ACCdata_t accd; //Accelerometer
} bb_data;
```

After initialisation, REVBB essentially runs as an infinite loop. The loop reads sensor information, displays it to the user, and, at a specified frequency, saves this data to disk. The simplicity of this design is shown by the main method (which runs on program execution):

```
int main (int argc, char *argv[]) {
    bb_init();
    while (1) {
        bb_read();
        bb_draw();
        if (BB_WRITE_STEPS == numdata) bb_write();
    }
    bb_close();
    return 0;
}
```

Initialisation is run once during program setup and performs all one-off operations. This includes initialising EyeBot inputs and outputs, starting the GPS and accelerometer, and creating a blank data file to save sensor data. The initialisation method is listed in the `bb_init()` method of `REVBB.c` included in Appendix A.

The SparkFun Electronics SerAccel v5 Triple-axis accelerometer installed in the REV uses a serial terminal to transmit data to the EyeBot. Linux serial interfacing requires the inclusion of the `<fcntl.h>`, `<termios.h>`, and `<unistd.h>` header files for file control, terminal attribute settings, and necessary miscellaneous functions respectively. The accelerometer port is opened and stored to the `accpt` handle by the command:

```
accpt = open(ACCPORT, O_RDWR|O_NOCTTY);
```

This opens the port defined by `ACCPORT`; for the EyeBot this was discovered to be `"/dev/ttyS2"`. The port is set to read and write mode (`O_RDWR`) and doesn't affect the process controlling terminal (`O_NOCTTY`, die.net, 2008). The `accpt` options are retrieved and the accelerometer baud rate is set to `B57600`, representing 56,000 bits per second. The options are then set for 8-bit words, no parity, and one stop bit (8N1) through the `c_cflag` structure. This structure is used to set the new options for the accelerometer.

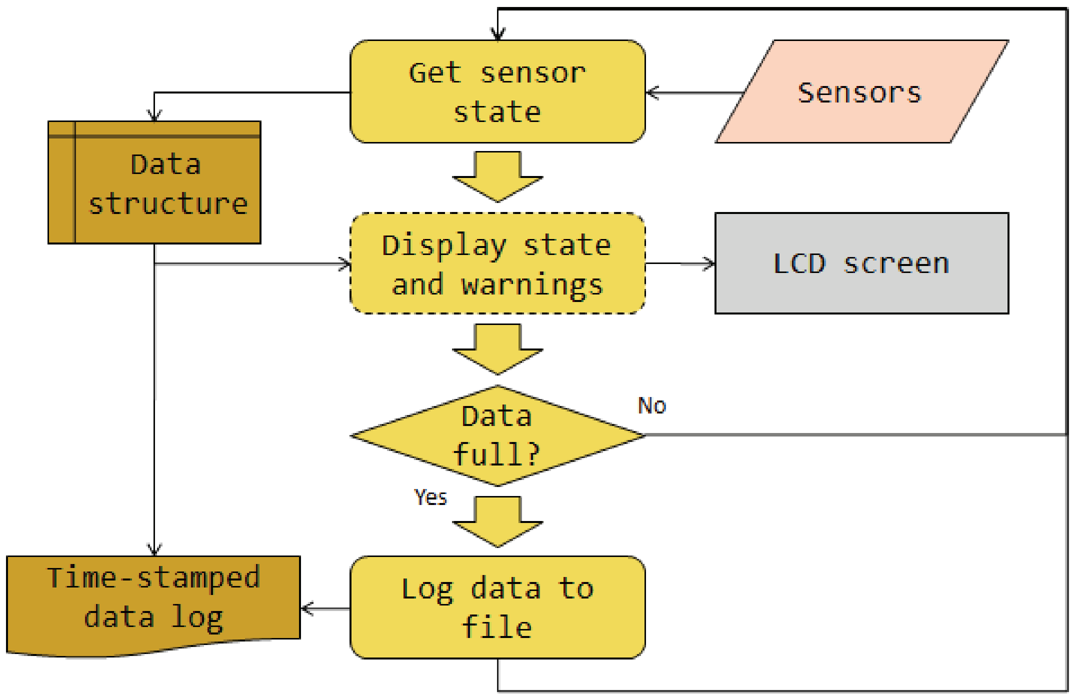

FIGURE 5.1: REVBB MAIN LOOP

The REVBB main loop follows the simple algorithm shown in Fig. 5.1 and the full REVBB source is included as Appendix A.

The first function, `bb_read()`, takes the current time and date as a year month day hour minutes seconds (YYMMDD HHMMSS) unique timestamp and attaches this to the current sensor reading `bb_data` structure, `curdat`. Next, it takes sequential readings from each of the digital inputs (latches zero through seven) and the analogue inputs (ADC zero through three) before requesting the current GPS reading (`returnGPSdata(GPShndl, curdat.gpsd)`) and accelerometer reading.

The accelerometer has several selectable operation modes. Currently it displays a plain text string of the calculated gravity values in three directions, x-, y-, and z-axis acceleration. Each acceleration triplet is displayed on a new line, for example:

```
X=-0.589 Y=-0.101 Z= 0.820
X=-0.600 Y=-0.104 Z= 0.812
```

These strings are parsed with some <strings.h> string manipulation methods, strstr() for searching and strncpy() to extract the six numerical characters. An atof() is then performed on the extracted characters to return the acceleration. These operations are performed three times, once for each axis.

The final stage of the bb_read() method copies the sensor readings into the prevdat data structure array for storage and, later, writing to disk.

The second method within the REVBB main loop is bb_draw(). Under the exact specification of the Black Box data recorder, this method is not required; however, it is useful to display the current sensor values to the LCD screen for debugging and to provide driver feedback. This project involved providing assistance to another team within the REV group that is developing the graphical user interface (GUI) to replace the bb_draw() method. The GUI was developed concurrently with REVBB and is able to display sensor readings stored in the bb_data structure curdat as they are read. The GUI is one of the first fully-developed EyeBot LCD and touch screen programs, and their requirements have driven change within both the REVBB program and the EyeBot development team members working with the touch screen routines.

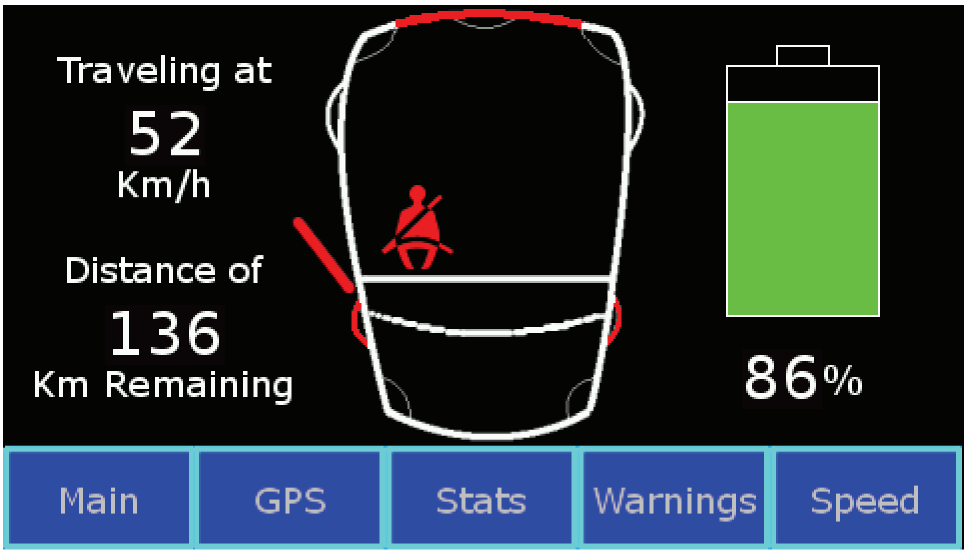

FIGURE 5.2: REVBB GUI MAIN SCREEN

The final aspect of the REVBB main loop is the bb_write() function. To increase performance, this routine is only called after a certain number of sensor readings, defined in `REVBB.h` by `BB_WRITE_STEPS` with a default value of 50. The number of loop iterations between logging effects performance as each write step takes longer than a read step. This performance requirement is balanced by the EyeBot's onboard 64 MB memory and the save frequency required. Recording data to disk once in every 50 loop iterations equates to approximately one save every five seconds – if the vehicle crashes or any other mishap occurs that disrupts the Black Box, a maximum of five seconds of data would be lost.

Before any data is written to disk, `bb_newfile()` creates a new comma-separated values (`.csv`) file. The file is named the time and date it was created, in the same manner that each record is time stamped. This allows the files to be sorted and categorised at a remote location for further analysis. The files are also populated with column headings explaining what each value refers to, allowing someone with minimal knowledge of the REVBB structure to analyse the data records.

After this heading row, the records stored in `prevdat` are printed in sequence with each record on a different line of the data file. The values within each record are printed in the predefined order separated by a comma. Using a `.csv` file permits maximum compatibility with a variety of spread sheet programs and allows for data manipulation without the need to convert the format. Each record is printed in the following order:

```
date YYMMDD, time HHMMSS, batt_low, brake_pump_fail,
fuel_door, inertia_switch, ignition, e_stop, tacho,
speed, main_batt_V, main_batt_I, throttle, alt_batt_V,
x, y, z, GPS, long_deg, long_min, long_secs, n_s,
lat_deg, lat_min, lat_secs, e_w, gps_speed, heading.
```

After each write cycle, the number of recorded entries is stored and the `prevdat` data structure results cleared from memory. The maximum number of entries that are to be recorded in a file is defined by `BB_ENTRIES_PER_FILE` in `REVBB.h`. If the total number of recorded entries is equal to or more than that maximum number, a new file is created using `bb_newfile()` and the number of recorded entries is reset, ready to take on more data. The `BB_ENTRIES_PER_FILE` has a suggested value of 10,000 entries, which is approximately 1 MB of data stored in each file. Again, a similar trade off is made for file size; repeatedly creating files takes longer but if a file is somehow corrupted, less data is lost.

REVBB also contains a `bb_close()` method, which under normal circumstances is never executed. This method performs a final `bb_write()`, releases the input and output handles, closes the accelerometer port and touch screen interface, and exits the program. REVBB is designed to be always operating while the EyeBot is powered on. The expected method of stopping the data logger is to power off the EyeBot. By the nature of this process, the EyeBot resources are made available upon reboot. The negative result of a hard power down is that all data is lost between the previous write step and the power down. This data loss is minimised by maintaining a small save frequency, as determined by `BB_WRITE_STEPS`; the suggested save period of approximately five seconds.

5.3 Experiments and Results

Several experiments have been performed on individual aspects of the REVBB code and the program as a whole. This section will detail some of the experiments performed, the results, and the problems highlighted by each test.

The REVBB program was developed in the early stages of the REV EyeBot integration project when little was known about the hardware or software configuration within the vehicle. The program was developed using RoBIOS version 0.4 which was missing most of the input and output controller functions, specifically the analogue-to-digital converter and GPS routines. Version 0.5 introduced the analogue-to-digital control routines and the updated touch screen routines covered in Chapter 3. The GPS routines have not yet been included into a release version of RoBIOS, currently at the third release of version 0.6.

There have been five EyeBot M6s developed. The first two, `gumstix` and `gumstok`, were developed with an older board design (M6.12) and `getz`, `elise` and `x5` (M6.14) designated for installation into each of the REV vehicles, the Hyundai Getz, Lotus Elise, and BMW X5 respectively. Shortly after production, the `getz` EyeBot was destroyed and decommissioned, and the `x5` EyeBot was tagged for REVBB development and testing. Not having a centralised EyeBot software repository has meant that each EyeBot has been configured differently by different people developing on it. One example of this is that the x5 did not contain the automatic USB flash drive loader possessed by gumstix and elise, which are used as development machines by the EyeBot team.

Device	Name	Function
/dev/ttyS0/	FFUART	Full Function UART – Debug console
/dev/ttyS1/	BTUART	Bluetooth UART
/dev/ttyS2/	STUART	Standard UART – Serial port interface
/dev/ttyS3/	HWUART	Hardware UART – Disabled

TABLE 5.2: EYEBOT M6 UART DEVICES AND PORTS

The serial accelerometer was a new addition to EyeBot development, using Linux serial interfacing. The accelerometer was connected to a serial terminal on the COM1 port on a Windows machine, which proved that the module worked as expected and was then configured. The EyeBot Linux serial terminals are on one of four devices, as listed in Table 5.2. I developed a program, acctest, to determine the device used as the accelerometer interface. The program reads from each of the four devices in turn. When acctest was executed on x5 it did not return any results, suggesting the accelerometer was not connected. Further research determined the name and function of each device (the second and third columns of Table 5.2) and that /dev/ttyS2/ is the expected serial device. A further test on x5 showed that the port was correct but was not receiving any data. Testing the same program on gumstix worked as expected, achieving results immediately. Extensive research into the software configuration revealed there was a hardware problem with the x5 serial port preventing data access on the standard serial port. Connections to the debug serial port worked as expected. To date, this has not been repaired.

Not all of the sensors required for logging have been installed into the vehicle and so we are unable to test the program under the full operating environment. However, the program can be tested using hardware simulated sensors to represent the critical systems. An array of eight single-pole, single-throw dip switches connected between the 3.3 V pin (pin 1) and input bank (pins 3-10) of the digital input/output pin array are used to simulate digital input sensors. When the switch is in the "on" position, the 3.3 V signal will be transmitted to the input pin, as in the vehicle when there is no error detected. The "off" position simulates the error state by disconnecting the 3.3 V supply from the input pin. The analogue sensors are simulated using a variable power supply, connected over a load, with a voltage probe running to each of the analogue-to-digital converter inputs. The accelerometer and GPS modules tested are the modules that have been installed into the vehicle.

Within the vehicle, each digital input is connected to a single pole within the double-pole, double-throw (DPDT) relays in the relay box. The analogue inputs measure the voltage of a probed node within a circuit. Both of these hardware simulations closely resemble the vehicle sensors and thus, we can assume that these simulation sensors operate in the same manner as the vehicle sensors.

Using these simulated test sensors on the `gumstix` EyeBot, allowed us to determine the operating parameters and test each aspect of the program for correctness. We tested each aspect of the Black Box program, including reading from each sensor, saving sensor values to memory, ensuring that each input is recording valid readings, creating and filling data files, and creating new data files after the previous is filled. The following results obtained are for an EyeBot operating under these simulated conditions:

REV Black Box operational parameters	
Sensor values per entry	28 (2 timestamp, 8 digital, 4 analogue, 3 accel., 11 GPS)
Sampling frequency	Approximately 8.75 Hz
Disk space per entry	0.10 KB
Entries per MB	Approximately 10,000
Time to record 1 MB	19.5 minutes
Time to record 1 GB	2 weeks

TABLE 5.3: REVBB OPERATIONAL PARAMETERS

The REVBB was operated for a test period of approximately five minutes, recording data and creating new data logs with a reduced save frequency and reduced maximum file size. Using the time stamps of the records, the sampling frequency was determined to be approximately 8.75 records saved per second with the M6main monitor program running as a background process. The 8.75 Hz measured is significantly less than the maximum data rate for the accelerometer, digital, and analogue inputs, of around 100 Hz. Running as the Black Box for the vehicle, M6main will be disabled, and the REVBB code will be executed on EyeBot power up, hopefully improving the sampling rate.

The 28 data points for each record are saved in plain text as a `.csv` file, with each record requiring 0.10 kB on the USB flash drive. At the calculated sampling frequency, it will take approximately 19.5 minutes to record the 10,000 entries stored in each 1 MB file. USB flash drives are available with varying capacity. USB flash drives with 1 GB flash memory are currently available for around $10 to $20 each. This lower capacity drive is sufficient for storing up to two weeks of continuous sensor readings. The vehicle is expected to be charged overnight for use during the following day. Since the recording capacity of the USB flash drive is greater than the operational period, the EyeBot Black Box can also be used to monitor battery voltages during a charge cycle without concern for reaching the maximum capacity.

The Black Box has also been designed to handle "hot swapping" of USB flash drives, where one drive is replaced with another during REVBB operation. When the EyeBot does not detect the presence of a USB flash drive at the beginning of a write cycle, the data is written to the EyeBot's onboard flash memory. However, the new data file stored on the EyeBot will continue to save sensor readings until a new file is created, even with the presence of a new USB flash drive. These 1 MB files can rapidly fill the 16 MB of onboard flash memory on the EyeBot. As a suggested future development, the GUI could display when a write cycle is performed, allowing users to change USB flash drives without logging any data to the EyeBot.

Examining the values recorded by the EyeBot in the sensor logs revealed a problem with the EyeBot digital input reading. The documentation suggests that only two possible values will be returned, '0' or '1'. Upon inspection, the sensor data logs are recording different values for the inputs. In one test, every digital input was read as "210996", regardless of state, while another trial recorded "171696" under the same test configuration. The values recorded are seemingly random and change each time the test is performed. This indicates a problem with the software configuration of the test EyeBots, possibly within the FPGA. The problem has been referred to the EyeBot development team and they have made no further progress on solving this problem.

Minimal testing has been performed since the EyeBot was installed in the centre facia panel and connected to the few sensors currently in the vehicle. There have been significant delays in the installation and configuration of the sensors, EyeBot, and other vehicle aspects in preparation to get the vehicle licensed for road travel. After the EyeBot is properly configured, the remaining sensors are installed, and the GUI is integrated into the REVBB program, there should not be any further issues with the sensor logging package for the REV Getz.

6. Completing the REV Getz and Vehicle Licensing

The later stages of this project have been focussed on helping with other areas of the vehicle, in preparation for licensing, despite not being in the scope of the original project. Getting the vehicle licensed by the Western Australian Department of Planning and Infrastructure (DPI) was the ultimate goal of the REV project. Having the vehicle licensed allows it to be legally driven on Western Australian roads, one of the first electric vehicles to do so. The vehicle underwent a successful inspection on Thursday 30 October 2008, and was able to travel on roads as of the next day. This chapter details some of the extra work performed on other systems and the preparation for the licensing process.

6.1 EyeBot and Sensor installation

Prior to reinstalling the dashboard, the EyeBot mounted within the centre fascia panel for the vehicle was damaged. This resulted in extensive repairs and most of the major components within the EyeBot being replaced. The zero insertion force cable connecting the EyeBot and the LCD module was not correctly re-installed and short-circuited the main power supply module, destroying the Gumstix control board and the FPGA, amongst other things. This caused extensive delays to the EyeBot installation and testing. The replacement Gumstix, FPGA, and LCD were taken from the `getz` EyeBot, destroyed shortly after production and configuration, in the middle of 2008. These replacement modules have retained the old configuration (for RoBIOS version 0.4) after reinstallation into `gumstok`, the EyeBot for the REV.

While waiting for this EyeBot to be repaired, cables were installed under the interior carpet between the dashboard and the floor under the passenger seat. To access the EyeBot peripheral connections after installation, cables were connected to the two serial ports and network port, allowing for the debug serial connection, a removable connection for the accelerometer, and the installation of a wireless network bridge for the EyeBot.

The wireless bridge requires 5 V power to operate, which is not available from the 12 V car battery. A 5 V power adaptor has been installed into a second cigarette lighter socket behind the centre console and wired under the carpet to the passenger seat. The wireless bridge configuration is documented in Appendix B.

After all of the wiring had been completed and the EyeBot repaired and returned, the dashboard was reinstalled over the course of several hours. The modular design of the dashboard with appropriate screws and bolts scattered around the work area, combined with the workshop manual from a much earlier model Getz, made this process very complicated. The final stage of dashboard installation was to install the EyeBot and centre fascia panel. The new wiring behind the fascia, combined with the plugs connecting into the EyeBot, made the panel impossible to install without a minor modification to the dashboard housing. After cutting a slot in the dashboard housing, the fascia module was installed, the EyeBot powered on, and the dashboard fully installed.

After the EyeBot was installed into the dashboard, the debug serial port was used to configure the EyeBot to work with the wireless bridge module. During this reconfiguration, it was discovered that the repairs had left an older software version installed on the EyeBot mounted within the vehicle. Some minor testing showed that the configuration must be updated before the REVBB program can be fully functional. The EyeBot development team have been informed but have not had access to the vehicle to update the software configuration. This should be completed shortly, so the vehicle and REVBB is operational for the upcoming display events and launch in November 2008.

6.2 Critical system repairs

After reinstalling the dashboard, several key vehicle systems were not working as expected. Some of these systems were required for the vehicle to be licensed, especially the horn and reversing lights. To fix each of these systems, wires were discovered and then traced in an attempt to find the fault.

Inside the steering wheel, the horn button acts as a switch, connecting the 12 V horn line to ground, through the steering column. We provided the horn directly with 12 V power, which made it activate; therefore the problem lay within the electrical systems of the vehicle. The 2003 Hyundai Getz workshop manual, used as a reference for the vehicle, identified a brown wire for the horn systems. Tests performed on this proved that it did not control the horn and more investigation was performed. After testing the horn relay within the fuse box, a white wire with an orange stripe was identified as the horn signal wire. Finding and then testing this wire under the dashboard showed that it was functional, supplied 12 V, and activated the horn when connected to ground. From here, the wire was traced up the steering column, under the steering column shroud to the back of the clock spring, and no faults were detected. The clock spring allows the steering wheel to be rotated while still maintaining an electrical connection to the airbag, horn, and stereo control buttons. Testing the electrical connections between each side of the clock spring determined that the internal wiring for six of the ten pins was broken, including the two airbag pins and the horn. After ordering and installing a new clock spring, the horn was repaired and acted as expected.

The reverse lights had a similar problem. The workshop manual stated that a plug connected to the trans-axle detected when the vehicle was in reverse, and after locating it, was shown to be error free. After tracing the wires from the rear lights back through the vehicle to the vehicle electronic control unit (ECU), the error was found to be between the ECU and engine bay. Investigating the unlabelled cables within the engine bay showed a matching connector for a cable connected to the transaxle. After reconnecting these two cables, the reversing lights acted as expected.

Ironically, neither of these key safety systems were tested during the licensing inspection.

7. Conclusion

The main aim of this project was to determine the possibility of using an EyeBot M6 controller as a sensor package interface and Black Box data logger for a Renewable Energy Vehicle.

The EyeBot M6 is a suitable embedded controller for installation as an electric vehicle Black Box sensor logger, due to the REVBB software developed in this project. The input restrictions imposed by the hardware design have limited the number and types of sensors able to be recorded by the sensor package; however, all of the critical systems and some additional systems are monitored.

This is the first full application developed for the EyeBot M6 and has proven the power and versatility of the general-purpose embedded system still under development at the University of Western Australia.

The REVBB program has been proven successful. It has a fully-functional GPS and accelerometer connection and inputs acting as expected using simulated sensors. Extensive testing has shown and resolved errors in the developing EyeBot functionality. The modular design of the program allows a graphical user interface module to display error signals to the driver as they occur. Errors are also recorded to the removable USB flash drive in a human readable format allowing further analysis on the recorded data.

Additionally, the program has minimised the changes required to install the REVBB system into another vehicle with a different sensor configuration.

Furthermore, the Hyundai Getz within the Renewable Energy Vehicle project has been successfully converted from a petrol vehicle to an electric vehicle and licensed for road travel. It is one of the first such vehicles in Western Australia.

This project has been completed successfully and is at the forefront of embedded system development, sensor recording, and display within electric vehicles worldwide. This new area of research and development will be continued in further projects and has virtually unlimited possibility for worldwide expansion.

7.1 FUTURE WORK

This project has created a successful first prototype for the REV Getz; however, there is still a lot of work that can be done in this new field. This section highlights some possible research and development projects for future students working on the EyeBot or REV development teams.

The highest priority project is to complete the sensor installation for the Getz. Some sensors, including the emergency stop button sensor and throttle position sensor, have not been installed yet and are monitored by the current REVBB software. This is a small but essential project. Testing should also be performed on the speedometer and tachometer reading and the fuel dial and tachometer outputs to the dashboard.

The calculations required to indicate the battery state-of-charge can be performed within the EyeBot code, after the battery readings have been calibrated for the current setup. The remaining charge can be indicated on the fuel dial within the instrumentation cluster using the motor controller signal.

After the sensors have all been installed, the GUI should be finalised for deployment. It should have the ability to control all required functions through the touch screen, and display GPS position on a larger map than is currently available. The EyeBot versatility allows for a variety of functions not listed here to be integrated into the user interface and controller.

The microphone installed within the car and connected to the EyeBot may allow a new user interface: voice commands. Providing auditory feedback and voice commands allows the driver to interact with the EyeBot without diverting their attention.

The installation of an RS-232 serial GSM modem will allow a user to connect to the car via mobile phone. Depending on the software and hardware implementation, it could track the vehicle in real time, display messages to the EyeBot screen, or even cut off power to the car completely.

In 2009, the new REV team will attempt to convert a Lotus Elise to a high performance electric vehicle. Expanding the REVBB project to encompass the new vehicles will be key to the continuation of this work.

Finally, after we collect enough sensor data from the Getz and Elise, a future project could analyse this data, compare it, and research the characteristics of electric vehicle use. This opens the possibility of further research into vehicle power sources, electrical systems, vehicle operation, and the feasibility of electric vehicles within the current and future economic and environmental state of the world.

REFERENCES

ABS 2008, *Year Book Australia, 2008*, Australian Burea of Statistics, [19 October 2008].

Australia's Best Car 2003, *Best Small Car*, Australia's Best Cars. Available from: http://www.australiasbestcars.com.au/2003/default.htm [30 September 2008].

Australia's Best Car 2005, *Best Small Car*, Australia's Best Cars. Available from: http://www.australiasbestcars.com.au/2005/default.htm [30 September 2008].

Barr, M 2008, *Embedded Systems Glossary*, Netrino. Available from: http://www.netrino.com/Embedded-Systems/Glossary [6 March 2008].

Blackham, B 2006, The Development of a Hardware Platform for Real-time Image Processing, The University of Western Australia.

Bräunl, T 2006, *Embedded Robotics: Mobile Robot Design and Applications With Embedded Systems*, Springer-Verlag Berlin Heidelberg, New York.

Bräunl, T 2007, *EyeBot-Robots Overview*. Available from: http://robotics.ee.uwa.edu.au/eyebot/ [30 September 2008].

Bräunl, T 2008, *EyeBot - Online Documentation*. Available from: http://robotics.ee.uwa.edu.au/eyeM6/ [30 September 2008].

Cygwin 2008, *Cygwin Information and Installation*. Available from: http://cygwin.com/ [28 October 2008].

die.net 2008, *Linux man pages*. Available from: http://linux.die.net/man/ [28 October 2008].

FuelTrac 2008, *Average monthly capital city unleaded petrol prices (cpl)*, Australian Automobile Association. Available from: http://www.aaa.asn.au/petrol/ULP.pdf [19 October 2008].

Gumstix 2008, *Gumstix Support Wiki*. Available from: http://docwiki.gumstix.org/index.php/Index.html [2008].

Hintermann, M 2007, Operating System Components for an Embedded Linux System, Technische Universität München.

Mathew, R 2008, *Welcome to the REV Project*. Available from: http://www.therevproject.com/ [30 September 2008].

OPEC 2008, *World Oil Outlook 2008*, Organization of the Petroleum Exporting Countries, [19 October 2008].

Sommer, T 2007, Application Program Interface for an Embedded Linux System, Technische Universität München.

Tamagawa 2003, *Tamagawa Solar Challenge Project*. Available from: http://tscp.tamagawa.jp/shc/ [30 September 2008].

Thomas, R 2007, *The Black Box Flight Recorder*. Available from: http://www.kidcyber.com.au/topics/blackbox.html [22 October 2008].

APPENDIX A – REVBB SOURCE CODE

REVBB.H

```
/** ******************************************
*
*       Renewable Energy Vehicle Black Box
*                    REV-BB
*
*                    HEADER
*
*       Developed by Ewan MacLeod within
*        the REV Team and EyeBot M6 Team
*     at the University of Western Australia
*                    in 2008
*
*********************************************/

#include <eyebot.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <string.h>
#include <time.h>

/*This structure is defined in the GPS headers
// it has been copied here for reference
typedef struct {

  int lat_degrees;
  int lat_mins;
  double lat_secs;
  int long_degrees;
```

```
  int long_mins;
  double long_secs;
  int hours;
  int mins;
  int secs;
  char n_s;
  char e_w;
  int date;
  double speed;
  double headding;
  double mag_variation;
  int flags[10];

} GPSdata_t;
*/

typedef struct{
        float x;
        float y;
        float z;
} ACCdata_t;

typedef struct {
//*******MEMCPY WILL FAIL for strings and pointers within
bb_data

    int datestamp;
    int timestamp;

    int batt_low;       //Critical - Error state is LOW (0)
    int brake_fail;     //Critical - Invalid -1
    int fuel_door;      //Critical - Normal HIGH (1)
    int inertia;        //Critical
    int ignition;       //Critical
    int e_stop;         //Critical
    int tacho;
```

```
    int speedo;

    int check_engine; //Output

    float main_batt_v; //Analogue
    float main_batt_i; //Analogue
    float throttle;    //Analogue
    float alt_batt_v;  //Analogue

    float soc; //State-of-charge, percent

    //GPSdata_t gpsd; //GPS
    ACCdata_t accd; //Accelerometer
} bb_data;

#define BB_GOOD_COLOUR LCD_BLUE
#define BB_BAD_COLOUR  LCD_RED

#define BB_ENTRIES_PER_FILE 10000
#define BB_WRITE_STEPS 50
bb_data curdat;
bb_data prevdat[BB_WRITE_STEPS];
int numdata;
int totdata;
int writdata;

#define ACCPORT "/dev/ttyS2"
#define ACCRATE B57600
int accpt; // Acc port handle
struct termios accopt;
int GPShndl;
ADCHandle AD_0,AD_1,AD_2,AD_3;

#define BB_LOC_SIZE 50
FILE *fp;
char current_file[BB_LOC_SIZE];
```

REVBB.c

```
/** ******************************************
*
*       Renewable Energy Vehicle Black Box
*                    REV-BB
*
*                 GETZ PROGRAM
*
*       Developed by Ewan MacLeod within
*        the REV Team and EyeBot M6 Team
*     at the University of Western Australia
*                    in 2008
*
********************************************/

#include "REVBB.h"

int timestamp(int *tme, int *dte){
    char ts[7];
    time_t ltime;
    struct tm *tmstr;

    ltime=time(NULL);
    tmstr=localtime(&ltime);
    strftime(ts,sizeof ts,"%H%M%S",tmstr);
    *tme = atoi(ts);
    strftime(ts,sizeof ts,"%y%m%d",tmstr);
    *dte = atoi(ts);

    return 0;
}

int bb_newfile() {
    char ts[13];
    int tm,dt;
```

```
    for (tm=0;tm<BB_LOC_SIZE;tm++){
        current_file[tm] = '\0';
    }

    strcpy(current_file,"/usbdisk/");

    timestamp(&tm,&dt);
    sprintf(ts,"%6d%6d.csv",dt,tm);
    strcat(current_file,ts);

    fp = fopen(current_file, "w");
    if (fp == NULL) {
            strcpy(current_file,"/home/rev/");
//            sprintf(ts,"%6d%6d.csv",dt,tm);
            strcat(current_file,ts);
            fp = fopen(current_file, "w");
            if (fp == NULL) {
                char * str;
                sprintf(str,"Creating File
%s",current_file);
                perror(str);
            }
    }
    fprintf(fp,"date YYMMDD,time
HHMMSS,batt_low,brake_pump_fail,fuel_door,inertia_switch,ig
nition");

fprintf(fp,"estop,tacho,speed,main_batt_V,main_batt_I,throt
tle,alt_batt_V,x,y,z,");

fprintf(fp,"GPS,long_deg,long_min,long_secs,n_s,lat_deg,lat
_min,lat_secs,e_w,gps_speed,heading\n");
    fclose(fp);
```

```
printf("**********New File -
%s*************\n",current_file);
    return 0;
}

//Method for writing current values to the screen
int bb_draw(){
    LCDPrintf("Digital : %d -
%d,%d,%d,%d,%d,%d,%d,%d\n",curdat.timestamp,curdat.batt_low
,

curdat.brake_fail,fuel_door,curdat.inertia,curdat.ignition,

curdat.e_stop,curdat.tacho,curdat.speedo);
    LCDPrintf("Analogue: %d - %f %f %f
%f\n",curdat.timestamp,curdat.main_batt_v,

curdat.main_batt_i,curdat.throttle, curdat.alt_batt_v);
    LCDPrintf("    Acc : %d - x %f, y %f, x
%f\n",curdat.timestamp,

curdat.accd.x,curdat.accd.y,curdat.accd.z);
}

//Method for reading data from sensors
int bb_read() {
LCDPrintf("reading %d\n",numdata);

    timestamp(&curdat.timestamp,&curdat.datestamp);
printf("T");
    //curdat.LATCH = OSLatchRead(LATCH0);
    curdat.batt_low = OSLatchRead(LATCH0);
    curdat.brake_fail = OSLatchRead(LATCH1);
    curdat.fuel_door = OSLatchRead(LATCH2);
    curdat.inertia = OSLatchRead(LATCH3);
    curdat.ignition = OSLatchRead(LATCH4);
```

```
    curdat.e_stop = OSLatchRead(LATCH5);
    curdat.tacho = OSLatchRead(LATCH6);
    curdat.speedo = OSLatchRead(LATCH7);
printf("L");
    curdat.main_batt_v = OSGetADC(AD_0);
    curdat.main_batt_i = OSGetADC(AD_1);
    curdat.throttle    = OSGetADC(AD_2);
    curdat.alt_batt_v  = OSGetADC(AD_3);
printf("A\n");


    returnGPSdata(GPShndl, curdat.gpsd);

    char accdata[29]; //Check to see this maximum
    int num_read = 0;
    char *start;
    char num[7];
        do {
        num_read = read(accpt,accdata,28);
        printf(":-");
        if (num_read < 0) perror("Acc error");
        }while (num_read < 28);
        //else printf("A read %d: '%s'--
\n",num_read,accdata);

        start = strstr(accdata, "X=");
        if (start == NULL) {printf("Error finding X=\n");}
        start += 2;
        strncpy(num,start,6); num[6] = '\0';
        curdat.accd.x = atof(num);
        start = strstr(start, "Y="); start += 2;
        strncpy(num,start,6); num[6] = '\0';
        curdat.accd.y = atof(num);
        start = strstr(start, "Z="); start += 2;
        strncpy(num,start,6); num[6] = '\0';
        curdat.accd.z = atof(num);
```

```
printf("memcpy(%d)\n",numdata);
    //*******MEMCPY WILL FAIL for strings and pointers
within bb_data
    memcpy(&prevdat[numdata],&curdat,sizeof(bb_data));

memcpy(&prevdat[numdata].accd,&curdat.accd,sizeof(ACCdata_t
));

memcpy(&prevdat[numdata].gpsd,&curdat.gpsd,sizeof(GPSdata_t
));

    numdata++; //issues with filled array!
    totdata++;
    return 0;
}

//Method for saving current/previous sensor data
int bb_write() {
    int i=0;
    bb_data bbw;
printf("fopen(%s)...",current_file);
    fp = fopen(current_file, "a");
    if (fp == NULL) {
       bb_newfile();
       writdata = 0;
    }
printf("Opened\n",current_file);
    for (i=0;i<numdata;i++) {
        printf("%d ",i);
        bbw = prevdat[i];

        //fprintf(fp,"date YYMMDD,time
HHMMSS,batt_low,brake_pump_fail,");
```

```
fprintf(fp,"%d,%d,%d,%d,",bbw.datestamp,bbw.timestamp,bbw.b
att_low,bbw.brake_fail);

//fprintf(fp,"fuel_door,inertia_switch,ignition,estop,");

fprintf(fp,"%d,%d,%d,%d,",bbw.fuel_door,bbw.inertia,bbw.ign
ition,bbw.e_stop);

//fprintf(fp,"tacho,speed,main_batt_V,main_batt_I,throttle,
");

fprintf(fp,"%d,%d,%f,%f,%f,",bbw.tacho,bbw.speedo,bbw.main_
batt_v,bbw.main_batt_i,bbw.throttle);
        //fprintf(fp,"alt_batt_V,x,y,z,");

fprintf(fp,"%f,%.3f,%.3f,%.3f,",bbw.alt_batt_v,bbw.accd.x,b
bw.accd.y,bbw.accd.z);

//fprintf(fp,"GPS,long_deg,long_min,long_secs,n_s,lat_deg,l
at_min,lat_secs,e_w,gps_speed,heading\n");
        fprintf(fp,"%c,",(bbw.gpsd.flags[1]==-1)?'V':'A');
//invalid/valid GPS lock
      if (bbw.gpsd.flags[4] == -1) fprintf(fp,"-,-,-,");
//no longitude
      else fprintf("%d,%d,%f,", bbw.gpsd.long_degrees,
bbw.gpsd.long_mins, bbw.gpsd.long_secs);
      if (bbw.gpsd.flags[5] == -1) fprintf(fp,"-,"); //no
long hemisphere (NS)
      else fprintf("%c,", bbw.gpsd.n_s);
      if (bbw.gpsd.flags[2] == -1) fprintf(fp,"-,-,-,");
//no lattitude
      else fprintf("%d,%d,%f,",bbw.gpsd.lat_degrees,
bbw.gpsd.lat_mins, bbw.gpsd.lat_secs);
      if (bbw.gpsd.flags[3] == -1) fprintf(fp,"-,"); //no
lat hemisphere (EW)
```

```
      else fprintf("%c,", bbw.gpsd.e_w);
      if (bbw.gpsd.flags[6] == -1) fprintf(fp,"-,"); //no
GPS speed
      else fprintf("%f,", bbw.gpsd.speed);
      if (bbw.gpsd.flags[7] == -1) fprintf(fp,"-\n"); //no
heading
      else fprintf(fp,"%f\n",bbw.gpsd.headding);

    }
    writdata += numdata;
    numdata = 0;
    fclose(fp);

printf("\n---------File Written - %d----------
\n",writdata);
    //issues with filled array!
    if (writdata >= BB_ENTRIES_PER_FILE) {
       bb_newfile();
       numdata = 0;
       writdata = 0;
printf("&&&&&&& totdata=%d, numdata = %d, writdata = %d
&&&&&&&\n",totdata,numdata,writdata);
    }
    return 0;
}

int bb_init() {

    KEYInit();
    OSLatchInit();
    OSLatchBankSetup(IOBANK0, IN);
    OSLatchBankSetup(IOBANK1, OUT);
    AD_0 = OSInitADC("ADC0");
    AD_1 = OSInitADC("ADC1");
    AD_2 = OSInitADC("ADC2");
    AD_3 = OSInitADC("ADC3");
```

```
    startGPS(&GPShndl);

    accpt = open(ACCPORT, O_RDWR|O_NOCTTY);
    if (accpt == -1) printf("%s connection
Error\n",ACCPORT);
    tcgetattr(accpt,&accopt);
    cfsetispeed(&accopt, ACCRATE);
    cfsetospeed(&accopt, ACCRATE);
    accopt.c_cflag &= ~(PARENB | CSIZE | CSTOPB);
    accopt.c_cflag |= CS8;
    int i;
    i = tcsetattr(accpt,TCSANOW,&accopt);
    if (i != 0) printf("Accelerometer Port Settings
Error\n");

    bb_newfile();

    numdata = 0;
    writdata = 0;

    return 0;
}

int bb_close() {

    bb_write();

    OSLatchCleanup();

    OSReleaseADC(AD_0);
    OSReleaseADC(AD_1);
    OSReleaseADC(AD_2);
    OSReleaseADC(AD_3);

    close(accpt);
    KEYRelease();
```

```
    exit(0);
}

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

    //Only use during development. You don't want to exit
while RUNNING the car!

LCDMenuI(3,"EXIT",BB_BAD_COLOUR,LCD_MENU_FGCOL,bb_close);

    while(1) {
        bb_read();
        bb_draw();
        if (BB_WRITE_STEPS == numdata) bb_write();
    //printf("Loop finished!!!!!!!!!!\n");
    }

    bb_close();
    return 0;
}
```

APPENDIX B – WIRELESS SETTINGS

The wireless module selected for installation into the vehicle is a LinkSys WET54G Wireless-G Ethernet Bridge. It has been configured to provide an ad-hoc network for connection to the EyeBot. The wireless bridge is only powered when the vehicle key is in the “accessories” position.

Wireless Bridge settings	
Service set identifier (SSID)	EyeBotAir
Passphrase	revrevrev
Encryption	WEP
Network infrastructure	Ad-Hoc

TABLE B.1: WIRELESS BRIDGE SETTINGS