FINAL YEAR PROJECT

20142809

REV Performance Vehicle Instrumentation

Author:
Matthew TYLER

Supervisor:
Prof. Thomas BRÄUNL

Thesis submitted as part of the B.E. degree in the School of Electrical, Electronic and Computer Engineering, University of Western Australia

Abstract

The REV Project is a multidisciplinary effort to design, build and evaluate electric vehicles with the goal of demonstrating the viability of renewable energy vehicles for personal transport. The following project outlines changes and additions to the REV Lotus instrumentation systems for the 2011 period.

Changes have been made to the core GPS & BMS programs, allowing for more robust operation. The telemetry module has been rewritten to cooperate with the now-running electric vehicle trial. A more informative battery monitoring panel has been added, in line with a need for more detailed battery analysis. The Engine Audio Replication System (EARS) has been reviewed and implemented into the user interface. A new panel has been implemented using the twitter API for relaying traffic data from Main Roads.

A new CAN-SPY device is introduced, which lays groundwork for implementing CAN based sensors and actuators, offering a more flexible and responsive platform for sending and receiving information across the vehicle. This simultaneously offers an expansive and high speed platform for future work. This first device is used to obtain feedback from the motor controller.

The problems with estimating battery capacity for the REV Lotus are introduced. Current methods of estimation both in production and in research are discussed. A new method is tested offering performance with an average error of approximately 10% through use of a combined neural network, coulomb count and Kalman filter approach.

Acknowledgements

The author would like to acknowledge several people to whom this project would not have been possible without their input and support.

I would like to thank Thomas Bräunl for providing this project and his leadership of the REV Project. Furthermore I would like to thank Ian Hooper and Ivan Neubronner for all their expertise and assistance throughout the year. I would also like to extend thanks to Beau Trepp; our spirited early morning discussions helped generate new ideas and forced me re-evaluate several approaches.

I would finally like to thank friends, family and work colleagues, all of whom have accommodated with my erratic schedule whilst this dissertation was completed.

Glossary of Terms

API	Application Programming Interface
ANN	Artificial Neural Network
BMS	Battery Management System
CAN	Controller Area Network
CARB	California Air Resources Board
ECU	Engine Control Unit
EKF	Extended Kalman Filter
FUDS	Federal Urban Drive Scheme
GA	Genetic Algorithms
GPS	Global Positioning System
GUI	Graphical User Interface
ICE	Internal Combustion Engine
IO	Input/Output
KF	Kalman Filter
OBD	On-Board Diagnostics
PWM	Pulse Width Modulation
REV	Renewable Energy Vehicle
RPM	Revolutions Per Minute
SA	Simulated Annealing
SAE	Society of Automotive Engineers
SoC	State of Charge
SoH	State of Health

List of Figures

List of Tables

Contents

1 Introduction

1.1 The REV Project

The Renewable Energy Vehicle (REV) project [2] is a multidisciplinary effort to construct and evaluate electric passenger vehicles for the private market. It began as a restart of earlier research into hydrogen fuelled vehicles, which had been discarded primarily due to costs. Since it's inception in 2008, the REV Project has successfully converted a number of conventional Internal Combustion Engine (ICE) vehicles to full electric systems. The vehicles converted include a Hyundai Getz, a Lotus Elise, and a formula SAE Vehicle. An extra conversion has been done for the Electric Vehicle Trial for a fleet of Ford Focus' but the physical conversion was undertaken by a third party, with members of the REV project in consulting positions. Each of these vehicles represents a different aspect of the goals of the project.

The converted Hyundai Getz (codenamed Rev Eco) was the first conversion attempted by the REV team. The conversion started and completed in 2008 and underwent significant alteration to install an electric drive motor, battery packs and suspension modification to handle the extra weight of the batteries. As the first conversion to be attempted by the groups, a lot of the monitoring and Battery Management Systems (BMS) are drop-in commercial models, as opposed to later conversions, which use entirely student designed systems. The goal of the Getz conversion was to prove that electric vehicles are a sustainable and valid option for typical commuting distances.

Figure 1: REV Eco : The Converted Hyundai Getz. Source: [http://therevproject.com/]

The logging and user interface (UI) systems on the Getz are handled entirely by an Eyebot M6 running a version of Linux designed for the Gumstix chip that powers the Eyebot. This system has undergone several modifications over the last three years of the project. The most recent update

Figure 2: REV Racer : The Converted Lotus Elise. Source: [http://therevproject.com/]

was performed this year by Beau Trepp, and resulted in a complete rewrite of the system to fix many significant stability and speed issues that existed in the original design.

The Lotus Elise conversion began in 2009 as the second conversion to be completed by the REV Team. This two seater performance vehicle is a much more powerful vehicle than the Getz, with the goal of proving that electric vehicles can meet the high performance demands imposed on sports cars. As a later conversion, it includes a BMS system that was designed at UWA, unlike the Getz. It also uses a much more powerful three phase AC motor instead of a series wound DC motor.

The Lotus includes a fully featured in-car PC operating a 1.6 GHz atom processor running Microsoft Windows XP. As such, it does not have as many limitations as the Gumstix-powered Eyebot M6. Functionality is driven by a program designed in the QT C++ framework.

The final vehicle to be converted is the REV Formula SAE vehicle. This donor vehicle for this conversion project was the UWA Motorsport race car from 2001. The end goal of this conversion is to produce a new SAE Formulae vehicle purpose built for the electric section of the Formulae SAE competition. Currently this project is on-going and slated to compete in Melbourne in December 2011, and then in Germany mid-2012.

The REV team has overseen the conversions of the Ford Focus' fleet for the WA Electric Vehicle Trial [3]. In addition to this capacity, telemetric and logging software built by members of the REV team is currently being used to evaluate electric vehicle transport in the metropolitan area. As these vehicles are driven far more often then the current fleet of REV-built cars, the team hopes to collect far more useful data for performance evaluation.

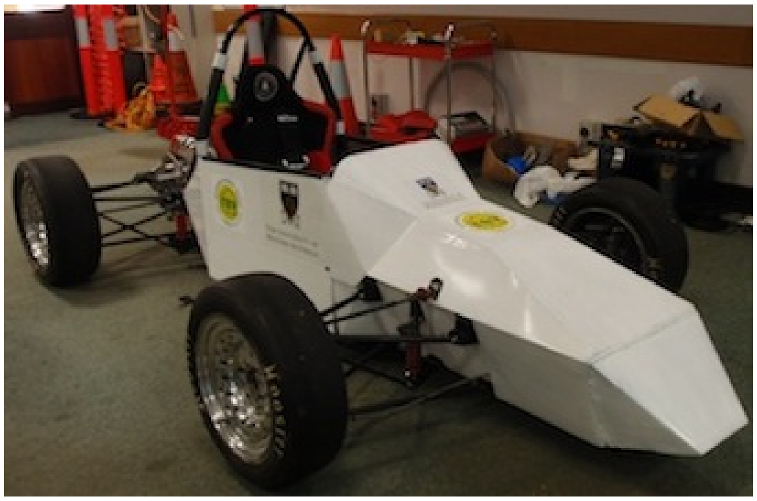


Figure 3: 2010 REV Formulae SAE-Electric. Source: [http://therevproject.com/]

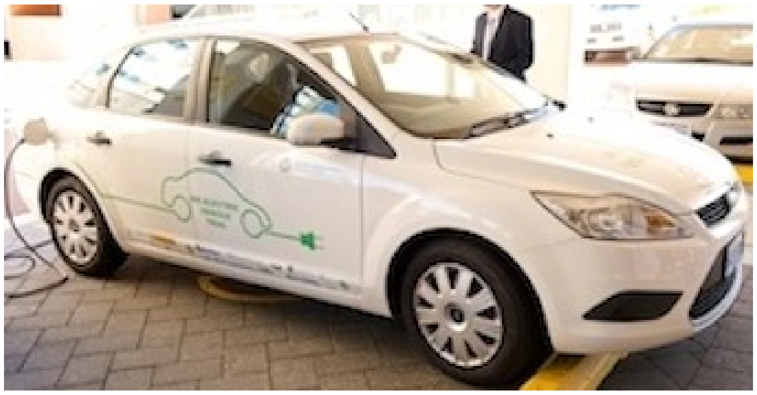


Figure 4: 2011 EV Works Ford Focus for Electric Vehicle Trial. Source: [http://therevproject.com/]

1.2 Electric Vehicles

Electric vehicles (EVs) are slowly coming to prominence as an alternative to conventional ICE vehicles. Ever depleting reserves of petroleum is going to force an uptake of a different propulsion system within the next century. In addition, pressure from the effects of global warming to reduce emissions means that cleaner technology fuels will need to be found for the transport industry, which currently produces 23% [4] of CO2 emissions worldwide. EVs are positioned well to respond to these factors, due to their efficient drive-systems and ability to use multiple power sources.

Most modern EVs use some form of battery storage to drive an electric motor. In the past, lead-acid batteries were used, but new designs have focused mainly on Lithium Phosphate cells (LiPO4). Lithium cells have a flat voltage response over most of the capacity, and have a reasonable capacity compared to their lead-acid counterparts. These cells are those used in plug-in EVs, although hydrogen fuel sources could potentially be used to power EVs in the future, which would still use similar drive mechanics as current EVs. DC motors are most commonly used, although more expensive, powerful AC motors are starting to be employed in EV's.

In terms of emissions, plug-in EVs do not produce any when in use. Their footprint is instead calculated by the energy source used to recharge the vehicle's batteries. The REV vehicles are powered by a solar grid, and therefore are zero-emission vehicles. However, vehicles that are charged off a typical mains grid in Australia would be supplied from burning coal. This would likely still be cleaner then burning of petroleum fuel sources.

EVs are not new. EVs were reasonably common during the early period of the 20th century, particularly in cities where range limitations had few consequences. Part of their appeal was the fact they did not require any manual effort to start at the time, and thus, they were commonly stigmatized as ”women's cars”, partially due to aggressive marketing as being more suitable for female drivers. However, with the invention of the starter motor by Charlie Kettering (itself an adoption of technology used to automatically open cash registers made by NCR) gasoline vehicles became far easier to start, the construction of large highways, and the discovery of plentiful oil in Texas, Oklahoma and California, caused EV technology to stall for almost a century.

EVs enjoyed a small resurgence in the early 1990's as auto manufacturers invested in cleaner technology. This was partly in response to the California Air Resources Board which pushed for lower emission vehicles. This resulted in the controversy that is detailed in ”Who Killed the Electric Car?” [5], whereby car companies produced EVs to be allowed to participate in the lucrative

California market, but did not advertise them and/or put them on unpopular lease terms, refusing to sell outright to the consumers and repossessing the vehicle at the end of the term. Those critical of the petroleum industry point to the lobbying by oil heavyweights as large reason for this behaviour by manufacturers. Unfortunately, despite the research into EVs during this time, low oil prices meant that consumers favoured large sports utility vehicles.

The economic crisis and the global warming debate has fuelled recent interest in EVs, as oil prices rise and make owning less-efficient vehicles less affordable. In 2004 Tesla Motors, a small up-start California car company, began work on the Roadster, which was eventually available in 2008. The Roadster went in an area where electric cars had not been before; use as a high performance sports car. This vehicle caught the attention of the media and the big car companies and renewed their interest in EVs. Following this, GM introduced the Chevrolet Volt, and Nissan produced the LEAF. The future of EVs looks to be an exciting time, as the existing car companies fight to gain a competitive advantage in a lucrative new market.

1.3 Automotive Instrumentation

Automotive instrumentation is a slowly evolving field, and has remained almost unchanged for most of the last half-century. Car dashboards were originally modelled after aircraft panels, and provided information regarding speed, temperature, battery and fluid levels in the car. The first implementations of such sensors were mechanical in nature, but with the advent of modern electronics, have mostly been upgraded to electrical sensors. As technology advances, complex communication systems and digital display panels have become more ubiquitous. Below in Table I is a brief time line of advances in vehicle instrumentation.

CAN communication networks have been arguably the largest advance in vehicle instrumentation. Introduced in 1986, the CAN system was proposed as a solution to the ever increasing amount of electronics being produced in the modern car. The system reduces the wiring of the vehicle, making it easier to add new electronic systems to the car. In it's inception, the CAN bus was mainly used as an output of the on-board diagnostic (OBD) connector and provided information on faults from the Engine Control Unit (ECU). However, in modern vehicles it is used for any number of functions, including driving instrument panels and controlling motors. It is not uncommon for vehicles to have multiple CAN buses controlling different elements of the vehicle.

1.4 REV Instrumentation & Document Structure

There are essentially two main goals of instrumentation with regards to the REV Project

1888	First electric speedometer invented by Croatian scientist Josip Belusic [6]
1901	Speedometer first featured in Oldsmobile [7]
1925	Combination Speedometer/Odometer invented by Arthur Warner [8]
1940	Padded Dashboards advocated by car safety pioneer Claire L. Straith [9]
1959	Volvo becomes first manufacturer to include front lap-shoulder seat belts as standard [7]
1969	Volkswagon includes first on-board computer system for fuel injection [10]
1975	Real-time fuel injection systems become common
1980	ALDL created by General Motors as first diagnostic system for testing Engines on assembly line [11]
1985	First commercially available Sat Nav produced by Steve Lebbezoo
1988	SAE recommend a standardised connector for diagnostics [12]
1990	GPS based satellite navigation begins to appear in market
1991	California Air Resources Board requires new vehicles sold in California to require ODB-I [12]
1994	ODB-II specification required by CARB [12]
1995	Airbags become a common feature
1996	ODB-II required now required for all new cars sold in the United States [12]
2008	New cars sold in United States required to implement ISO-15765-4 CAN standard [12]

Table I: Brief History of Automotive Advances

1. Improve, increase and communicate the information collected from the car in real-time in an effort to inform the driver of various conditions to aid in driver decision making.

2. Collect and collate data from sensory input to be used in evaluating the performance of the vehicle.

Any system added into the REV Lotus instrumentation systems should aid in completing either one or both of these goals.

The following sections of this document outline and evaluate systems developed for the Lotus, and provide background for design decisions. Whilst a lot of the systems are interrelated, the document has been structured to review each system in it's own section, as to ensure that there is significant detail and focus on each individual component and to guide future work in each area. The document is ideally read in the order that sections appear, as occasionally information will appear later that is dependent on a prior section.

As a secondary (yet equally important) goal, it is the hope of the author that this document provide a reasonable of amount of information and background for future students that may be required to work with the Lotus.

2 Overall System Design

2.1 Background

2.1.1 Software

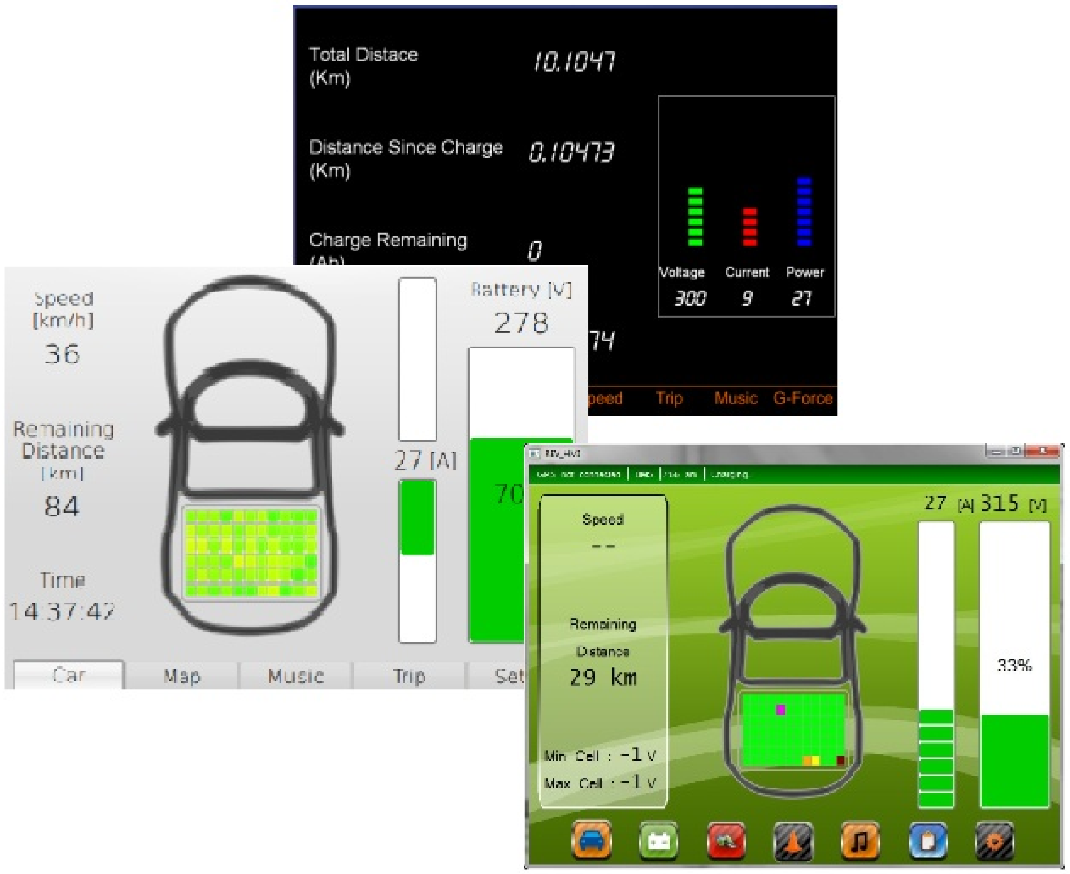


Figure 5: REV Lotus UI 2009 [Source: Varma], 2010 [Source: Walter] and 2011

The first attempt at unifying all aspects of the system was completed by Daksh Varma in 2009 [13]. This system was composed of Visual C++ backend and an Adobe Flash front-end. Data was passed between the two programs by writing to XML files. This was largely a rather inflexible platform, and a new system began development just six months later. This system implemented the core functionalities that would be present in each subsequent system; BMS communication, GPS tracking, maps, a music player and some logging functionality.

The second attempt at a unified system began in mid-year 2010 by Thomas Walter [14]. The QT C++ framework [15] was selected this time around, and has proven to be a reliable choice. QT allows for fast development of user interface elements and their associated back ends, and abstracts away from thread locking and other concurrency problems. The signals and slots system allows for easy communication between system elements. The end product is a developer-friendly, easy to use system that exhibits good extensibility.

Currently, further developments are being made to the original code written by Walter, rather than rewriting for a different platform. In future, it may be necessary to move off this platform, as the instability of it's parent company, Nokia, has left it's future in doubt. The availability of purpose-built tablet-based operating systems, such as Android and the upcoming Windows 8, could be viable alternatives in the future.

2.1.2 Hardware

The Lotus consists of centralised system around the in-car PC. The in-car PC consists of a Dual Core Atom 1.6 GHz Intel processor with 2 GB of RAM, running the Microsoft Windows XP operating system. This is mounted inside a VoomPC-2 enclosure specified for work in automotive environments and powered by an automotive DC-DC Car PC power supply. Furthermore, it is connected to a 7" resistive touch screen.

The BMS and GPS are then attached to the PC by way of serial-to-USB converters. The GPS is QStarz GPS 818X GPS receiver and outputs data based on the NMEA protocol. The BMS is a custom design by Ivan Neubronner and allows various details about the batteries to be sent to the in-car PC. For information about the message protocols used in these systems please see Appendices A1 and A2.

2.2 System Updates 2011

The system, as described by Walter follows in Figure 6.

Additions to the REV System result in Figure 7.

Following now is a discussion of changes made to the system, excluding Engine Sounds and CAN bus, which will be covered separately.

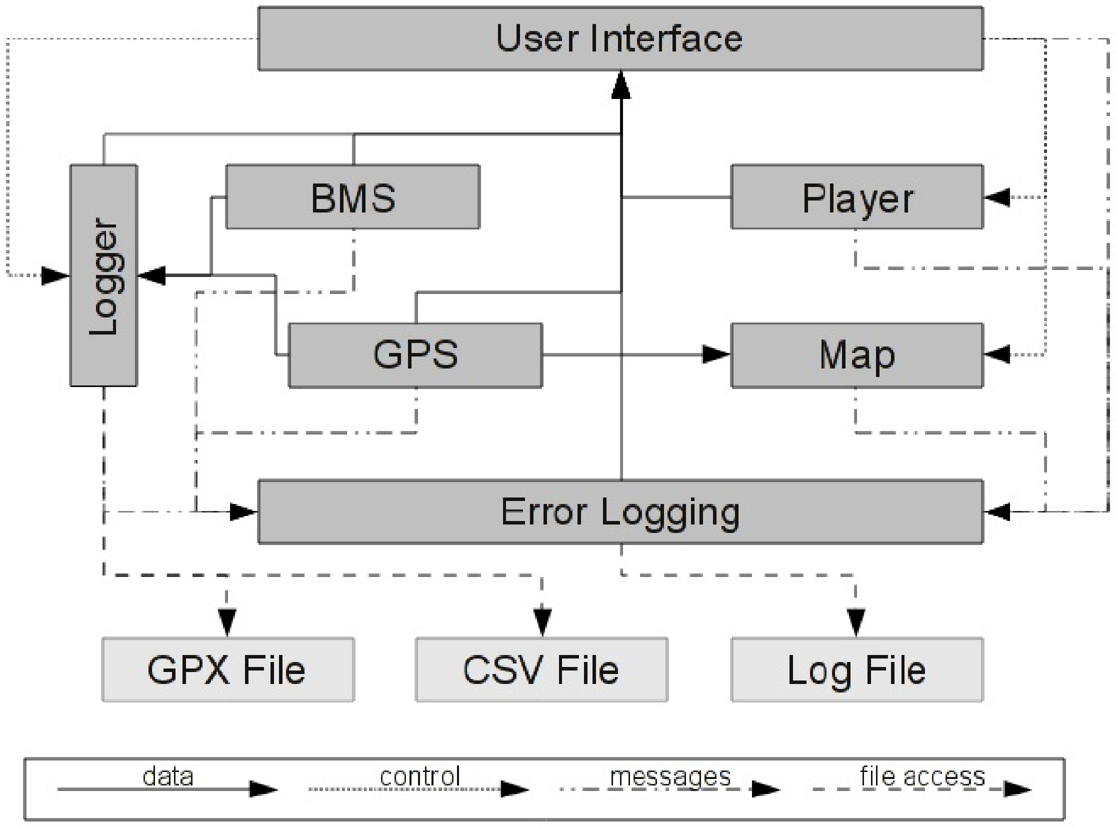


Figure 6: REV System 2010 [Source: Walter]

2.2.1 UI Design Changes

The UI has been given a few small cosmetic changes in order to increase it's usability. The addition of the status bar is one such element. It displays relevant conditions, such as BMS and GPS connectivity, to be reported on every page of the Graphical User Interface (GUI). This is important, and it reduces the need to flick to the system log to check common conditions. The next change is re-skinning of the GUI to make it more appealing. A green background fitting the environmental theme of the vehicle was selected, and a set of buttons replaces the grey tabs at the bottom of the screen. This has generally been well received by members of the team. Lastly, a transparent frame class has been added to the code. The object-orientated nature of the QT GUI forces all UI elements to inherit off a base-class 'frame'. These however, can be promoted to the transparent frame class, resulting in curve edged transparent frame seen in Figure 8.

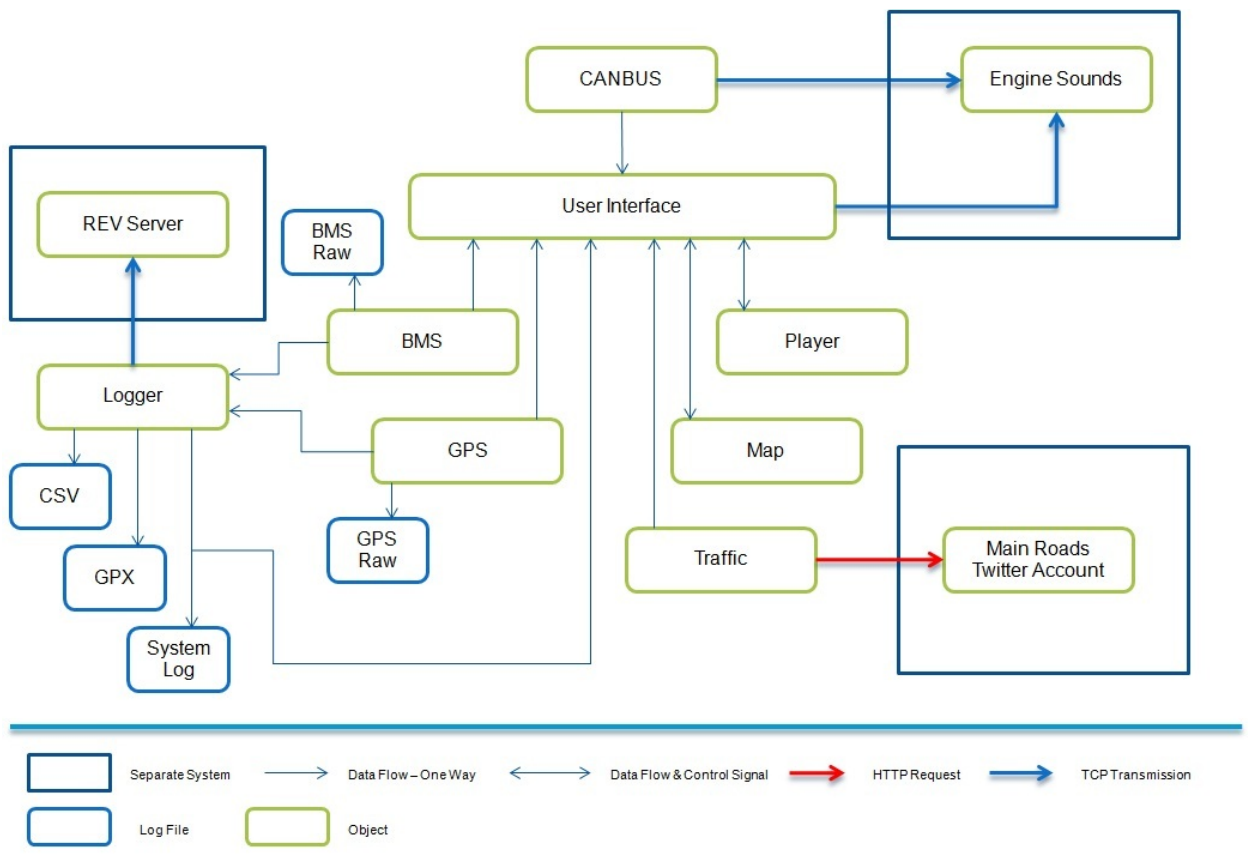


Figure 7: REV System 2011

2.2.2 Battery Status Panel

A limitation of previous design was the lack of information present with regards to the status of the cells in the battery pack. Previously, battery status was indicated by various colours, but this is difficult for a human operator to interpret with reasonable accuracy. As such, an extra panel was added that details the last recorded value for a particular cell from the BMS, with readings updated approximately every 20 seconds. Included in this are the values for the minimum, maximum and average cell voltage in the last BMS data cycle. Included is also an extra feature that constructs a data file that can be used for training offline neural networks, a feature discussed later in this paper.

2.2.3 Enhanced Logging Functionality

Whilst the current logging functionality produces incredibly detailed files for the purposes of evaluating the car, it does not log data from the BMS or GPS in the format that it is passed to the core program. Having such functionality would be useful, as the car PC software could be more rigorously tested by feeding it real world data. As such, two new logging functions have been added, ”log raw GPS” and ”log raw BMS”, which create files based on incoming GPS/BMS data.

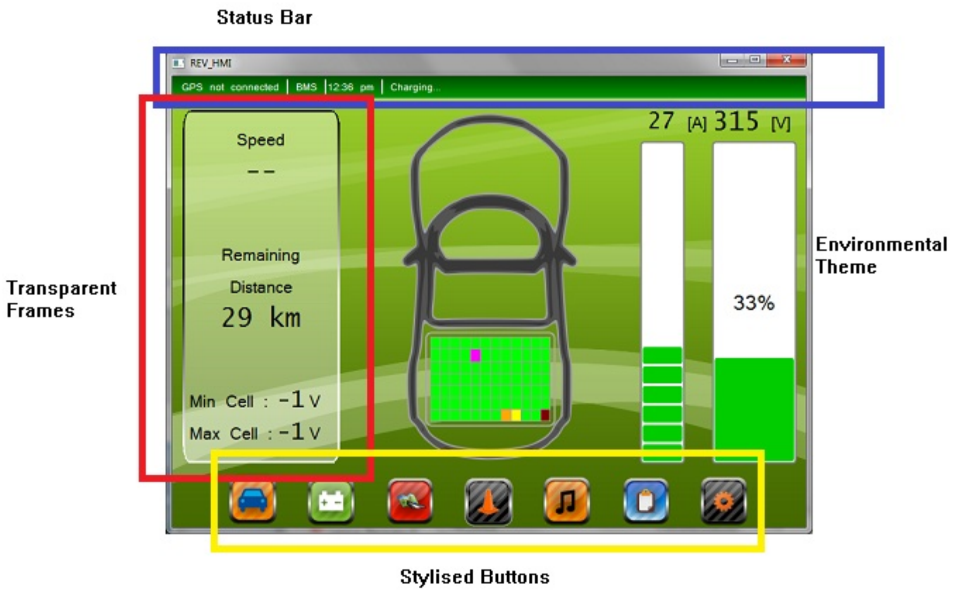


Figure 8: GUI changes 2011

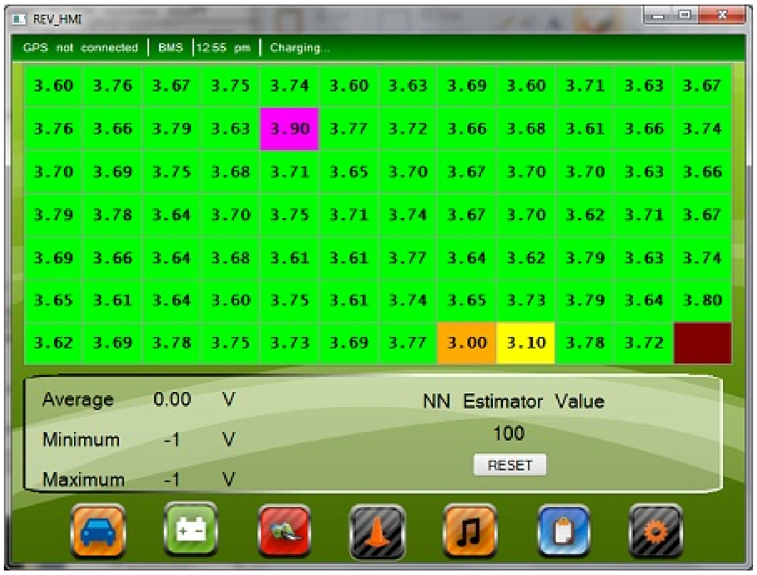
This new feature allows a debug version of the software, running on a separate PC, to pass in data obtained in the field, which in effect emulates the drive the data was obtained from. This is useful for modules such as Engine Sounds which produce data based on BMS data; as tuning the system to play the correct sounds for different actions is impossible to do in the online system.

2.2.4 GPS/BMS Core Modifications

The GPS and BMS classes initially exhibited poor performance in situations where their serial connections would fail. This was due to a design error in the original code that assumed that a connection could never be closed by their connected module. This disconnect can happen due to driver errors, but is most commonly exhibited in situations where the hardware undergoes a reset or power cycle. This is a particularly common occurrence with the BMS, as it will often reset itself after a successful charge cycle and occasionally when the vehicle is plugged or unplugged from an outlet. As such, the programs were redesigned to attempt to reconnect upon a disconnection. The old logic, compared with the new logic, is displayed in Figure 10.

2.2.5 Telemetry Core Modifications

The original telemetry module was included in the 2010 version of the software but has undergone some simplification and bug-fixing to get it in a working state.



GPS not connected | BMS | 12:55 pm | Charging...

3.60	3.76	3.67	3.75	3.74	3.60	3.63	3.69	3.60	3.71	3.63	3.67
3.76	3.66	3.79	3.63	3.90	3.77	3.72	3.66	3.68	3.61	3.66	3.74
3.70	3.69	3.75	3.68	3.71	3.65	3.70	3.67	3.70	3.70	3.63	3.66
3.79	3.78	3.64	3.70	3.75	3.71	3.74	3.67	3.70	3.62	3.71	3.67
3.69	3.66	3.64	3.68	3.61	3.61	3.77	3.64	3.62	3.79	3.63	3.74
3.65	3.61	3.64	3.60	3.75	3.61	3.74	3.65	3.73	3.79	3.64	3.80
3.62	3.69	3.78	3.75	3.73	3.69	3.77	3.00	3.10	3.78	3.72	🟥

Average 0.00 V
Minimum -1 V
Maximum -1 V

NN Estimator Value
100

Figure 9: Screenshot of battery cells page

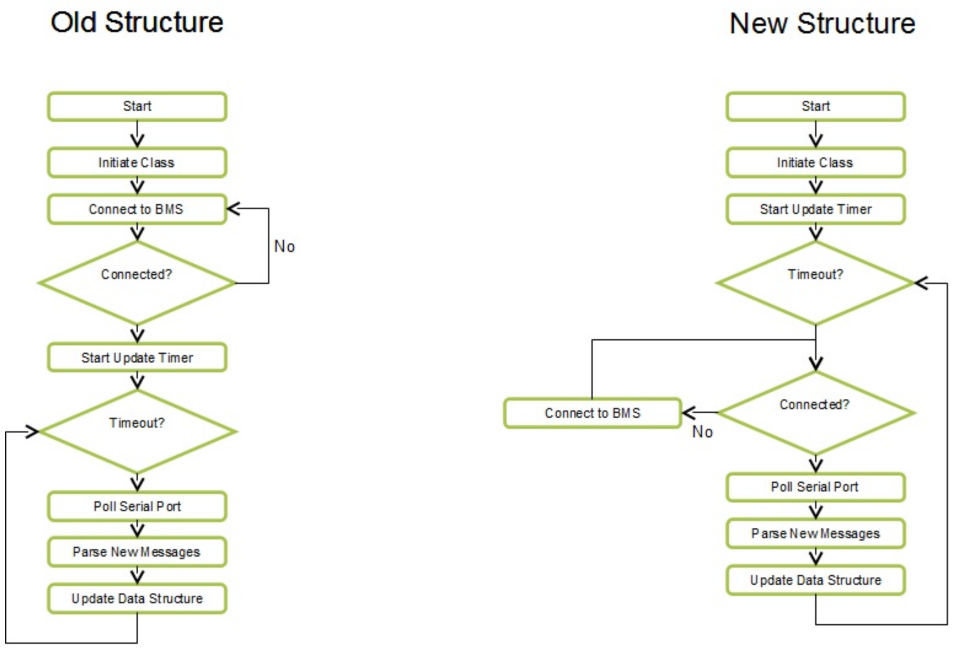


Figure 10: Comparison of BMS/GPS logic

Originally, John Pearce based the protocol on that used by the AT telemetry units [16]. A few elements of this protocol are not relevant to the REV vehicles or are not tracked by the server telemetry statistics, and therefore can be omitted or implement only a basic state. One such element is the reason codes, for which only the journey start, finish and time interval update codes are implemented. Appendix A3 includes the telemetry protocol by Pearce.

The structure of the original program was incredibly complicated for the function and has undergone some significant simplification to ease maintenance of the telemetry function. The 2010 version worked by building a buffer of messages and attempting to send these to the server. If the server did not acknowledge a message, it would clog the buffer up until it could be sent and removed from the queue. This has been greatly simplified by running a separate timer in Logger class, which constructs and sends a message every x seconds, where x is the timer interval. Messages that are not received by the server are destroyed.

2.2.6 Main Roads Twitter Panel

Another minor addition to the code is the addition of a panel displaying information from Main Roads Twitter account [17]. In effect, this allows the car to display information about traffic incidents in Perth that are posted to Twitter, forewarning the user of road closures, accidents, traffic light malfunctions and other road issues that can occur in the Perth metropolitan area. It operates by sending HTTP requests that are specified through the Twitter Application Programming Interface (API), which can then respond with an XML response. This response can then be parsed to format the data. The HTTP requests are sent using a C++ library called Twitcurl, which includes several wrappers for HTTP requests that are sent using cURL. Figure 11 shows a graphical outline of this system, whilst Figure 12 shows the Twitter API panel in operation. A decision was made to allow the user to refresh the information on command as opposed to by a timer, in an effort to reduce data costs incurred by the vehicle's 3G connection.

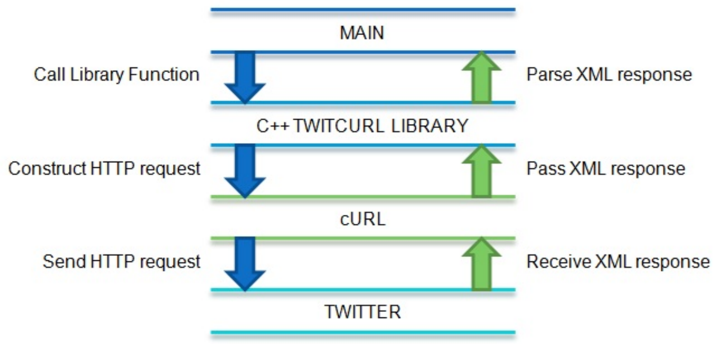
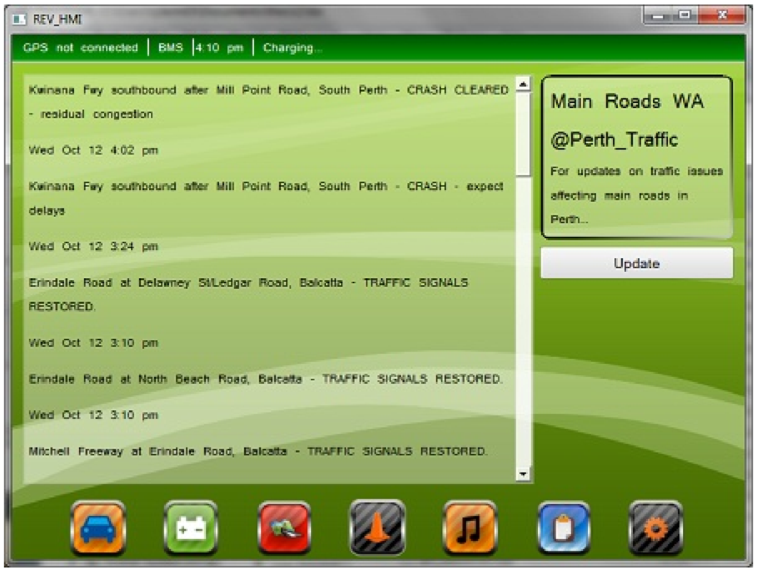


Figure 11: Twitter Program Structure



Kwinana Fwy southbound after Mill Point Road, South Perth - CRASH CLEARED - residual congestion

Wed Oct 12 4:02 pm

Kwinana Fwy southbound after Mill Point Road, South Perth - CRASH - expect delays

Wed Oct 12 3:24 pm

Erindale Road at Delawney St/Ledgar Road, Balcatta - TRAFFIC SIGNALS RESTORED.

Wed Oct 12 3:10 pm

Erindale Road at North Beach Road, Balcatta - TRAFFIC SIGNALS RESTORED.

Wed Oct 12 3:10 pm

Mitchell Freeway at Erindale Road, Balcatta - TRAFFIC SIGNALS RESTORED.

Main Roads WA

@Perth_Traffic

For updates on traffic issues affecting main roads in Perth...

Figure 12: Lotus Twitter Feed Display

2.3 Summary

Several changes were made to the system in 2011 in order to improve robustness, along with the addition of several new features. The BMS, GPS and telemetry systems now operate with added reliability and are capable of recovering from system errors. The BMS and GPS systems also received enhanced logging functionality for use in testing and emulation. The UI has gained an improved look along with features that will increase the usability of the system. Finally, the addition of the Twitter module enables traffic information to be relayed to the driver.

3 Engine Sounds

There is one element of EVs that proves to be both a blessing and a curse, and that is the amount of noise that they produce. Electric and hybrid vehicles tend to produce far less noise than their ICE counterparts. Whilst most proponents of EVs tend to use the reduced noise pollution as an argument for widespread adoption of EVs, the reality is that this can be a negative element of EVs.

Recent studies done with hybrid vehicles have indicated that there is a significant risk factor associated with low vehicle noise. A study by the National Highway Traffic Safety Administration within the U.S. Department of Transportation indicated that hybrid electric vehicles were two times more likely to hit cyclists and pedestrians at low speeds than conventional ICE vehicles [18]. Similar studies have done to evaluated the risk to blind pedestrians and have found similarly that

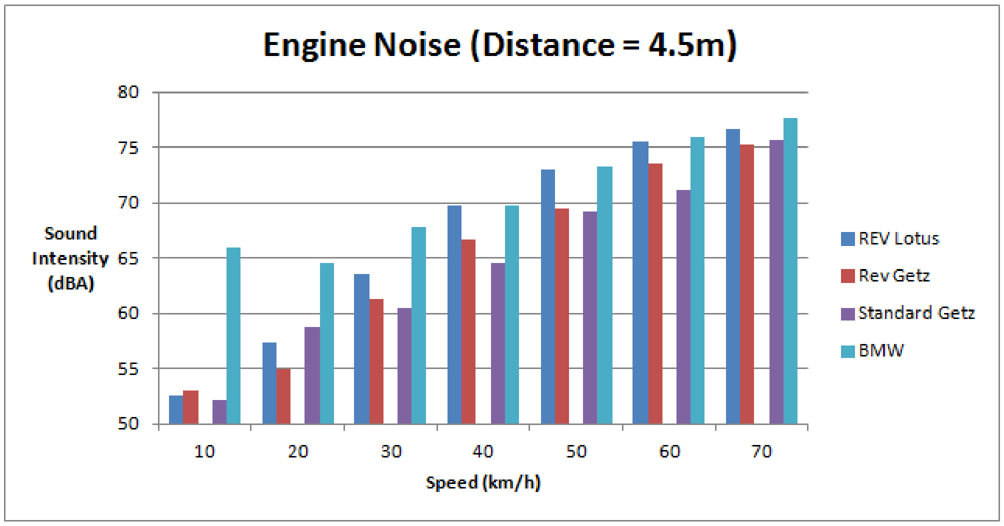


Figure 13: Comparison of Electric and Internal Combustion Engine Vehicle Noise at Different Speeds

EVs pose a greater safety risk as they are difficult to audibly detect [19] [20]. This is a large enough concern that US congress made moves to legislate the minimum amount of noise vehicles are expected to make in 2009 [21]. Earlier in 2011, President Obama signed the Pedestrian Safety Enhancement Act of 2010 requiring EVs to emit a certain amount of noise [22].

Tests done by the REV team indicate that EVs do indeed produce less noise than their petrol counterparts. Performed at the foreshore with a sound meter, it has been confirmed that on average an EV produces 3 dB less noise than ICE vehicles travelling at the same speed, which is roughly half the sound intensity. This tends to hold true for speeds under 30km/hr, after which noise caused due to wind resistance and tyre-road friction tend to take over. As an aside, any team member who has driven the Lotus from the undercover car park to the REV laboratory can attest to the fact that pedestrians tend to not notice the vehicle moving towards them if approaching from behind. Although the problem with EV noise only exists for low speeds typically under 30km/hr, it is important to remember that this is still a speed that can cause injury.

Of course, replicating engine noise is not without it's critics. The U.S. Federation for the Blind, whilst praising Nissan for it's pedestrian warning system, has made it known they are displeased that driver can turn the system off [23]. Similarly, anti-noise pollution proponents have argued that mandatory noise requirements will increase noise pollution in the environment [24]. Lastly, there has been criticism at the singling out of EVs in the Pedestrian Safety Enhancement Act of 2010. There exists several gasoline powered cars that are almost as quiet as electric and hybrid vehicles, yet are exempt from restrictions posed by the act. Those that raise this argument state that pedestrian safety is not a hybrid/electric vehicle problem, but a quiet car problem.

The following section presents background on solutions that have been undertaken in effort to manage the silent nature of EVs, before building on and implementing the solutions developed in previous years.

3.1 Background

Since EVs have been criticised as a possible safety risk at low speeds, a few different noise-making solutions have been proposed, and even implemented as is the case with a few commercially available EVs. Solutions tend to mainly take the form of two variants. The first, is to make the car emit certain warning sounds in particular high risk situations, akin to the truck-reversing sounds that most would be familiar with. The second is to emulate the sounds associated with an ICE vehicle. The latter is quite often the most popular solution, as it also tends to feed the public's association of high performance with loud engine noise.

Emulation of vehicle noise, particularly that of which is associated with engine revolutions, is not a particularly new concept. The video game industry has been working on technology to reproduce vehicle noise for the last two decades to produce realistic racing simulators. This commonly involves the use of dynometers and purpose-built microphones for capturing engine vehicle noise. In-game systems commonly use a set of samples and then frequency shifting to create appropriate engine noise.

The most recent addition to the world of engine noise systems is that of Nissan's Vehicle Sounds for Pedestrians (V.S.P) [25]. This takes the form of warning sounds, as opposed to that of replicating engine noise. In this system, the car produces a sweeping sine wave 600 Hz to 2.5 kHz. The rationale behind this is that most age groups will be able to hear a noise in this range and be able to respond to the vehicle appropriately. This system can be shut off by the driver, or shuts off automatically once speed increases beyond 20km/hr. V.S.P is currently employed on the Nissan LEAF.

The HaloSonic system is a noise making solution produced by a joint venture between Lotus Engineering and Harman automotive [26]. Unlike Nissan's solution, this is a representative of the engine noise replication camp. This system is currently configured to produce the sounds associated with V6 and V12 engines. Whilst this system has been around since 2009, it is still in development and has only demonstrated in a Toyota Prius, and the Lotus Evora 414E concept car. It is currently unknown whether the HaloSonic system will be bought to market.

There is currently a large focus on producing systems that only create noise in appropriate situations. Currently, most noise generation systems will switch off after they exceed certain speeds, but this is the extent of adaptive noise making. Directional sound equipment is being employed by ECTunes [27], whereby audio sounds are focused in the travelling direction of the vehicle, so that those outside of the cars path are not disturbed the noise of the vehicle. Similarly, General Motors is looking into technology that can sense whether pedestrians are in the vicinity of the vehicle, as opposed to other vehicles, and therefore emit appropriate warning sounds.

The REV project's first attempt at an engine sound reproduction program began in 2009 and was produced by Chris Hellsten [28]. This research primarily looked at generating intermediate sounds for a specific RPM from a set of engine sound samples, as well as software methods of emulating engine noise. The completion of this paper resulted in the Ferrari-on-a-Stick program, which formed the basis of all engine sound emulation work by the REV project.

Subsequently, the system was further refined by Karri Harper-Meredith in 2010 [29]. Whilst Hellsten was primarily concerned with the design of the software, Harper-Meredith's focus was primarily on implementation into the car. As such, extra code was added into the Ferrari-on-a-Stick platform that interfaced it with the Labjack I/O board and re-badged as the Engine Audio Replication System (E.A.R.S). However, this version of the software is tightly bound to the Labjack system and is not ideal for implementation into the Lotus, and is based on obtaining data from sensors that simply don't exist on the Lotus. Further work is mentioned on an embedded version of the software, but it does not appear that a physical implementation eventuated.

3.2 Design & Implementation

There were two completed versions of engine sounds supplied to be implemented in the Lotus. The first, completed by Chris Hellsten and improved upon by Karri Harper-Meredith, was written in C in 2010. The second is a C# implementation by Chris Hellsten that was written after he completed his thesis. Both of these programs were originally developed upon the assumption that there would be access to either a foot-pedal or tachometer, and that the speed of the car would be known. However, this was not the case initially, and it was eventually decided that the current would be a suitable method of approximating an appropriate RPM value. Thus either program would need to be modified. Each version has several advantages and disadvantages, although ultimately the C# version was selected.

The original C code has the advantage of being directly portable to the C++ system. As such, linking to the functions in E.A.R.S is a trivial task. It also includes an emulated gearbox system, which the C# code did originally have. Unfortunately, it does have several drawbacks that meant it was passed up on. The code is tightly bound to Labjack, meaning that a rewrite of the code would be necessary. Sound quality is much poorer, and it produces very audible popping noises from sound wave discontinuity as different sound files are shuffled in and out. It also lacks any form of volume control, and as most of the sound processing functions are not from any well-known libraries, would be be difficult to implement. Adding to all of this, is that the code was very poorly documented, with barely any comments and no accompanying explanation for various functions in the program.

The C# version was far more usable. It is cleanly commented and logically structured. Whilst it lacks the gear transmission emulation functions, these are easily ported from the C version of the source code. By far the greatest advantage is in its use of Microsoft's DirectSound libraries, and as such audio quality is much improved. Fortunately, there are DirectSound ports on Linux, and mono enables C# code to be runnable on Linux, so the choice of C# does not impact our choice of platform too greatly. The major issue with the selection of C# is it's inability to be linked from C++. Whilst both languages are similar, QT C++ is a type of unmanaged C++ code and as a result, the compiler tends to mangle the names of linked functions. This of course, is not compatible with the managed nature of C#. This can be gotten around through the use of the COM interop library, although it adds a significant overhead to each function call. As this is to run in a real-time, with many calls made each second, this was not going to be an appropriate option. As such, the decision was made to run each the Lotus UI system and Engine Sounds as separate processes and allow them to communicate over TCP.

To implement the system so that is controllable through GUI, the program is executed as QProcess object, which creates a running process on the machine and allows it to run in the background. Engine Sounds acts a TCP server on the local host address of the machine using port 5000. From here, linking the programs is enabled by connecting to the server and transferring commands across. These commands determine whether Engine Sounds should start, stop, change volume or play a different RPM sound file. These commands are detailed in Table II. Commands are both transmitted and read as ASCII.

The final system operates according to Figure 14. There are currently two types of cars that the system can currently emulate; a Ferrari and an ICE Lotus. These are user selectable on the system screen.

Command	Argument	Action
STAR	None	Initialise Engine Sound buffers
V###	Volume (Percentage of system volume 0 to 100)	Change volume to ###
####	RPM (0 - 9999 RPM)	Load and play sound according to ####
QUIT	None	Close engine sounds

Table II: Engine Sounds TCP Commands

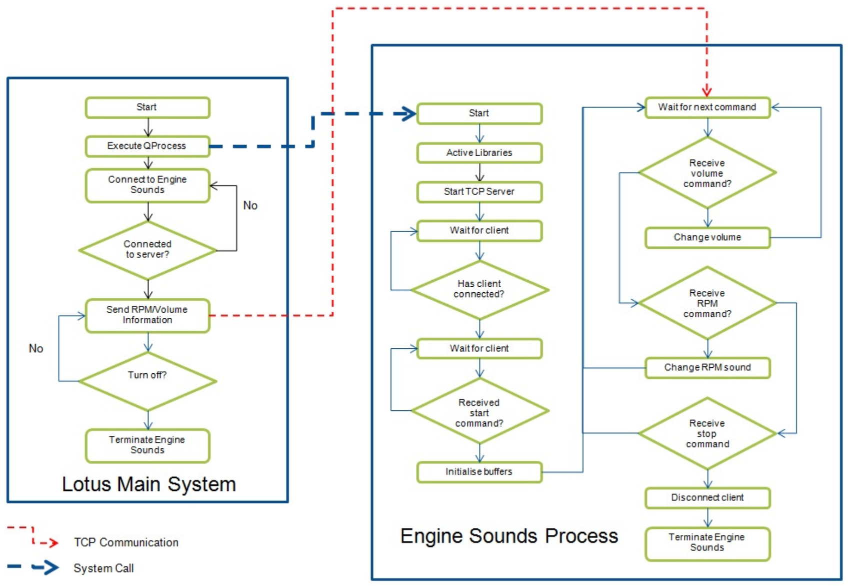


Figure 14: Engine Sounds Program Logic

3.3 Evaluation & System Tuning

Following now is a discussion of tuning the system to get a reasonable performance. In defining what a 'reasonable performance' is, we set up a brief list of criteria. Firstly, it should respond quickly to user input. If the driver puts the their foot down on their accelerator, it should be expected to be produced an increasing RPM sound in well under one second. Secondly, we ideally want a low amount of jitter. This means that if the accelerator is held constant, sound tracks should not flip back and forth, creating wave discontinuities. Thirdly, track swapping should be as inaudible as possible. Finally, the sounds played should be appropriate for the specific driving conditions and emulate an ICE vehicle as closely as possible. To realise the consequences of this is to know that EVs often use fewer gears than ICE vehicles, and as such RPM is often much higher. However, running the car at the reported RPM value of the electric motor will get annoying, as for a speed of around 50 km/hr would result in a sound of around 5000 RPM. This is much larger than the cruising RPM of most ICE vehicles, but wouldn't be altogether unusual in an EV.

Current to RPM translation is reasonably simple. However, to understand it requires an understanding of how much current the car passes in certain scenarios. Firstly, although the motor is technically designed to pass 400 amps, the most it is configured to draw currently is 200 amps. This is done by putting your foot to the floor. During a regular suburban drive, not exceeding 60km/hr and accelerating reasonably, the driver will be unlikely to exceed 70 amps for heavy acceleration. 30 amps is passed when driving at around 50km/hr. As such, we could assume that at 30 amps, we would playback at 1800 RPM, and linearly increase sounds to 7000 RPM at 200 amps. This works fairly well, but is strange in practice because gear shifts are not emulated, which was the next task.

Fortunately, a gearbox emulation design was present in the C version that was easily portable to the C# version. This code adjusts the RPM value that engine sounds should play based on the speed of the vehicle, throttle, and whether the car is exceeding a certain minimum and maximum RPM threshold. This appears to be partly based on transmission information obtained by Frans Ho [30]. This does create several new issues. The speed of the car is not always available due to the intermittent nature of GPS, and there is no throttle sensor on the car. A cursory glance at the gearbox code reveals that it's RPM tracks are essentially based on speed, so the value for current is instead used to indicate throttle in this scenario, thereby avoiding the need for current-RPM conversion. The speed is a much larger problem, as if the GPS cuts out it will result in the car sounding like it is stuck in one gear. The solution for this is not perfect at the moment. At present, we can obtain the motor RPM from the CAN bus, and as gear ratios are known, we could use these to get an estimate of speed to operate independently of the GPS. However, there is no sensor currently installed in the car to obtain what gear the vehicle is in, nor is it particularly necessary as the car is driven solely in first gear around 90% of the time. So a reasonable estimate for speed can be made solely from the RPM value of the motor delivered by the CAN bus line, and as such, can act as a fail-safe in the event the GPS is off-line.

As sensory data is not available currently for a perfect implementation of the Geared Engine Sounds, two switch-able modes are implemented, one that bases the value for RPM with a current translation, and one that uses speed and current as RPM and throttle respectively. As the Lotus is refined with more sensors, the program should be refined to have automatic and manual transmission modes, and operate in only the 'geared' version of the code. The author predicts that the current-RPM version will be phased out, although it is currently the most reliable conversion method.

Responding swiftly to user input is possibly the most difficult element and one to which a perfect solution has not been implemented as of current. Quite often, attaining a swift response is antagonistic with our second goal, which is to attain a low amount of jitter. Any sort of processing that correct for jitter, such as weighted-average-means and kalman filters quite often add some processing cost which reduces response times. However, the biggest factor by far is the speed of which we obtain our inputs. Chiefly, the suggested mechanism is to use current to determine RPM. This is not a bad idea, as current is related to the speed and throttle of the vehicle. Where this creates issues is the current speed at which current values are obtained, which is specified in the BMS at only around every 500 ms. In addition, because the Lotus PC system runs on timers and not threads,just because a timer goes off does not mean that the information is processed immediately, so there is an additional overhead. Although this overhead is quite small and indistinguishable currently, if the program gets more complicated (and it likely will) it will increase. Currently, because data sent from the BMS is used to drive engine sounds, the response is quite poor. The use of the RPM value from the CAN bus which is described in the CAN spy section will improve this, as by it's nature it operates approximately 10-100 times faster than the BMS serial line.

To understand how pop effects occur is to understand how the program operates it's sound buffers. The program has access to a set of sound files based on RPM values, at intervals of 100 RPM, from 2000 RPM to 7000 RPM. As an example, if accelerating from 2000 to 2100 RPM continuously, the system will frequency shift the 2000 value to 2049, and at the mid point will switch tracks to a 2100 RPM sound, frequency shifted to 2050 RPM. At the point that a buffer switches tracks, a small popping noise can be heard caused by sound wave discontinuity. The technique employed to reduce this is to utilise multiple sound buffers with varying amplitudes.

The last RPM track sent by the program is the the loudest, and sound tracks that were previously played are slowly reduced in volume so that upon switching sound tracks the popping effect is indistinguishable. This leaves us with one major design decision to make; how many buffers do we use? Using too many buffers, even with a reasonably large decay on older buffers, results in a large echo effect and in use sounds like a jar of bees. Similarly, speed of input data to Engine Sounds is important; if intermediate, continuous data is not supplied to Engine Sounds, an RPM jump from say, 2000 to 2500 RPM, will result in both 2000 and 2500 being played simultaneously. As the fundamental frequencies in these are far enough apart, it can result in a musical effect and sounds similar to a musical chord. As a result of these two effects, experimentation revealed that a smaller number of buffers of around 2-3, with a large volume cut on older buffers gave the best sound quality.

In addition to pop effects caused by changing sound buffers, they can also be induced by dramatically changing the volume. In order to reduce this effect, volume changes are implemented by decaying the sound down to the target volume level. Simply put, the volume is modified from the source volume to the target volume in degrees of 10% until the current volume reaches the target volume. This adequately reduces popping effects from volume changes.

Jitter is an interesting effect that only occurs in one particular scenario and is slightly related to the pop and response effects. It is reasonably obvious when a track switch occurs if you are listening for it; any other time it is difficult to distinguish. However, if the RPM is fluctuating between the track switching break point it becomes extremely obvious as the frequency shift is not perfect and there is a slight variation in pitch between the up-shifted old track and the down-shifted new track. This effect is most present in current-RPM translation of the system, as by it's definition the geared version of the program tends to favour certain RPM values and as such, prevents fluctuation. Stability can be attained by several methods; moving averages and kalman filtering (although reduces response time), and/or adding code to detect when this occurs. Detecting this situation is the ideal method as it does not negatively impact on the response time in any meaningful manner and is simple to implement. All that is required is to reject any changes to the value of RPM that increase it below or beyond a midpoint threshold if the last two values for RPM are within a certain percentage of each other. In this case, the tolerance was set to 5%, and this reduced fluctuations to an acceptable level.

3.4 Summary

The Engine Sounds system successfully builds upon the work undertaken in previous years and introduces a working system to the current Lotus UI system. The C# code has been improved to reduce audio disruptions from track-switching discontinuities that occur during system use. The addition of emulated gear system has allowed the system to better approximate the sound output of a conventional petrol vehicle.

4 CAN Bus SPY

The instrument cluster in the Lotus consists of a speedometer, tachometer and various other assorted indicators. In 2009, when the Lotus was shipped, the speedometer was replaced with one indicating km/hr, as the previous one indicated miles per hour. As a requirement of the Australian Design Rules [31], this was replaced. In addition, during the conversion the Engine Control Unit (ECU) providing all signals to the instrument had to be removed and rewired in accordance with its new drive system. Whilst the speedometer was reconnected in line with Australian requirements, a tachometer is not such a requirement, and as the ECU providing this signal was removed, it was not attached.

Fortunately, the UQM Powerphase 75 installed in the Lotus has the ability to output information about the engine over a connection from the motor controller [32]. This includes engine speed, torque, voltage, current and temperature. It can do this through two methods; one using a serial connection, and the other a CAN bus connection. As UQM does not provide information about the protocol used to send data over serial this could not be used. It does however, provide the protocol for CAN bus connections [33], based on the SAE J1939-21 standard [34], [35]. As such, it was decided to implement a system using the CAN bus. In addition to taking data from the CAN bus to create an RPM signal, it would be useful to be able to also relay this information to the in-car PC. As such, we define the following set of requirements:

1. Take input data from the CAN Bus
2. Output this over the serial line
3. Alter a Pulse Width Modulated (PWM) signal to drive the tachometer

Following on now is background information about the CAN bus. Afterwards the system design in both hardware and software is discussed, finishing with the evaluation of the system.

4.1 Background

4.1.1 CAN Bus

In 1986, Bosch [36] introduced the CAN bus to the world. It was designed to allow several different car components to communicate with either and to support messaging priority. Motors, motor controllers, instrument panels, etc are often designed to send and receive standard CAN messages. Most cars being produced today include a CAN bus, and it is common interface to complicated fleet management systems used in the transport industry. Although originally designed for passenger and commercial vehicles, it has seen uptake in the agricultural and manufacturing industries.

The CAN bus layer is based on several layers of abstraction. In it, following layers are defined; Application, Object, Transfer, Physical layers and Data Link Layers. These are based on the following ISO specifications:

ISO 11898-1:2003	Data Link Layer & Physical Layer
ISO 11898-2:2003	High Speed CAN Bus
ISO 11898-3:2006	Low Speed Fault Tolerant CAN Bus
ISO 11898-4:2004	Time Triggered Communication Protocol
ISO 11898-5:2007	High Speed CAN Bus Extension

Typically, the physical layer is only referred to in abstract terms and not directly specified. The most common implementation of the physical layer is through a two pair twisted wire, terminated with two 120 ohm resistors, as shown in Figure 15. This gives the CAN bus very high noise immunity through a low level of differential voltage. The twisted pair arrangement also helps to ensure that RF interference is kept to a minimum. Connector types and pin outs continue to lack formal treatment and auto manufacturers tend to have their own specific connectors. The most common connectors tend to be in the form of DE-9 and ODB-ii. It is often the case that second pair of wires runs parallel to the CAN-L and CAN-H wires, carrying a voltage supply rail and grounding rail, in order to provide power to CAN nodes.

The transfer layer represents the bulk of the CAN standard, and as such is too detailed to cover completely here. It passes valid messages to and from the object layer, and is responsible for bit timing, synchronization, message farming, arbitration, acknowledgement, error detection, error signalling and fault confinement. Perhaps the most important aspect is message arbitration. Message arbitration occurs in priority based arbitration system. All messages transmitted by a CAN

Table III: Popular CAN high level protocols and their common usage

Protocol	Use
DeviceNet	Factory Automation
CANopen	Medical Equipment, Vehicles, Automation, Rail, Electronics etc.
NMEA 2000	Marine Applications
SAE-J1939	Trucks, Bus, Fleet Management & Passenger Vehicles

system contain a priority. Those transmitting a lower priory will detect if a higher priority is emitting a message, and will back off until the higher priority message has finished receiving. 0 bits are dominant on the bus, so if a node emits 1 but reads a 0, it will know that a higher priority message is being transmitted and go into the waiting state.

The object layer handles message filtering and message handling. Under this layer, messages a particular can node is not interested in are filtered out, to reduce processing time. Finally, the messages are constructed in the object layer to be sent by the transfer layer.

The data link layer specifies the main workings for the CAN standard. It provides the following attributes of the CAN bus.

- Multi-master capability: Any CAN node may send a message, if the bus is idle.
- Broadcast communication: All messages transmitted are received in all nodes. All receiving nodes decide if they like to accept this message. This guarantees data consistency as all nodes in the system use the same information.
- Sophisticated error detecting mechanisms and re-transmission of faulty messages: This guarantees network-wide data consistency.
- Non-destructive bus arbitration: If two or more CAN nodes request simultaneously a message transmission, the protocol guarantees that the message with the highest priority gets bus access immediately.

The application layer is the final element of the CAN system. The original CAN standard does not include the tasks of the application layer, and so is left up to vendor implementation. Such tasks include flow control, device addressing, and transportation of data blocks larger than one message. These implementations outside the standard are often referred to as higher layer protocols. Some of the more popular protocols include DeviceNet [37], CANopen [38], NMEA 2000 [39] and J1939. Table III shows a list of such protocols and their common uses in various industries.

Table IV: Bit rates for specific bus lengths. Source: [1]

Bit Rate(kbit/s)	Bus Length (m)	Bit-Time (us)
1000	30	1
800	50	1.25
500	100	2
250	250	4
125	125	8
62.5	1000	20
20	2500	50
10	5000	100

CAN is designed to operate in two different modes: high speed and fault tolerant. Fault tolerant mode is typically defined at speed 40-100k transmission rates, whilst high speed mode operates at above 100k, up to 1Mbit/s. High speed CAN is the most popular standard for the physical layer and is utilised in the DeviceNet and openCAN standards. High speed is terminated with 120 ohms on each end of the network. Low speed differs in that each device has it's own termination and can continue operating in case of a wiring failure, and is less common. Typically the speed of the CAN bus is related to the length of the transmission line. The below Table IV illustrates the recommended transmission length lines max limit for a particular transmission speed. Figure 15 is of a typical high speed CAN network physical layer.

There are two main disadvantages when dealing with the CAN bus. The first is that of bus utilisation. It is possible for higher priority messages to hog the CAN bus. This can can occur if a high priority node is constantly transmitting. This can be the case for motor control systems. To get around this problem, most vehicles will have more than one CAN bus, with one usually dedicated for ECU signals, as is the case with the Nissan LEAF [40]. The second is that of a limit on the number of CAN nodes due to the address limit in the specification. Currently, most CAN networks are limited to 110 to 255 nodes. This can be gotten around by interconnecting CAN networks, however in practice it is rare that this situation occurs in automotive applications.

CAN Message frames are specified in CAN 2.0 Part A and CAN 2.0 Part B [41]. The former describes the base CAN format, whilst the latter describes both base and extended formats. In order for CAN compatibility a message must match a description in either part A or part B. Figure 16 describes the format of an extended frame CAN Message.

In operation, typically only the Identifier A, Identifier B and Data are specified. All other fields are constructed by software and hardware routines. These include the Start-of-Frame (SOF), Extension Identifier Bit (IDE), Remote Transmission Request (RTR), Reserve Bit (r0), Data Length

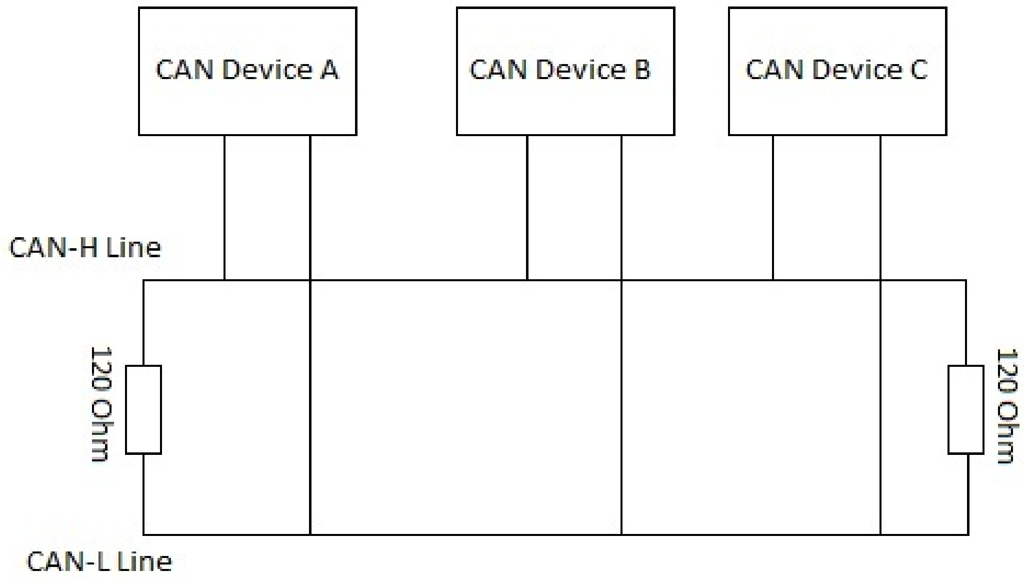


Figure 15: High Speed CAN Network

SOF	Identifier A	IDE	Identifier B	RTR	r0	DLC	Data	CRC	ACK	EOF

Figure 16: CAN Extended Frame

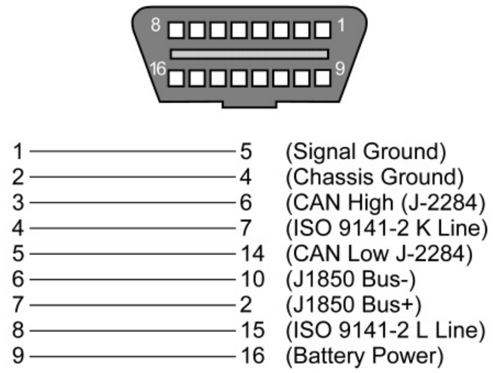


Figure 17: OBDII Pinout Diagram Source: [http://www.pinout.net]

Code (DLC), CRC (Cyclic Redundancy Check), Acknowledgement Bits (ACK) and End-of-Frame (EOF).

OBDII is the current diagnostic standard for vehicles. All cars sold in the United States past 2008 are required to implement ISO 15765-4, a variant of the CAN network bus. Typically it is a connector underneath the dashboard that may implement a variety of different network standards. Figure 17 is a typical diagnostic connector pin out diagram.

4.1.2 Tachometer

Tachometer implementation varies between year to year. Today, CAN bus driven tachometers confirming to a messaging standard are reasonably common. However, the Lotus was constructed in 2002. It wasn't until 2005 that Lotus started to drive its instrument clusters with CAN signals, and did not meet the CAN standard until 2008. Amusingly, this has led to a large market of post-2008 instrument clusters being retrofitted for 2005 era Lotus'. Because of this, and by looking through the Lotus' 2001 Service Manual [42], it can be deduced that the tachometer is likely to be a driven by a 12 volt signal emanating from a hall effect sensor on the crankshaft/camshaft. The Getz runs a similar setup, that was investigated by Daniel Kingdom in 2009 [43]. As such, we can use Table V in order to design a CAN-to-Analog Tachometer converter, that Kingdom formulated after probing the Getz.

This is roughly equivalent to the following equation:

$$f = \frac{f_{rpm}}{30} * 2 \tag{1}$$

PWM Frequency (Hz)	RPM (On Cluster)
0	0
32	1000
65	2000
102	3000
134	4000
167	5000
200	6000
232	7000
268	8000

Table V: Frequency vs Tachometer Value

4.2 Design

4.2.1 Hardware

The CAN signal is provided from the UQM Powerphase motor controller over the CAN-L and CAN-H lines [32]. In 2009, Cameron Watts [44] designed a breakout board from the Amphenol 19 connector on the controller, in order to easily access the motor control lines. Fortunately the foresight was had to break out all the signals provided from the controller, allowing easy access to these CAN lines. From here on, a compatible micro-controller is required to receive the CAN signals and relay them through serial, whilst also generating a PWM signal.

Olimex provides the AVR-CAN development board [45], of which uses an AT90CAN128 chip from ATMEL as well as the MCP2551 chip, a popular CAN transceiver. This board provides a DE9 CANBUS connector, as well as a DE9 serial connector. In addition it provides several input/output pins and 4 PWM channels. As such it is perfect for our requirements. Summary data sheets for the AVR-CAN, AT90CAN128 and MCP2551 can be find in Appendices.

The AVR-CAN board includes two 34 pin IDC plugs to access various pins. In order to access these lines, a breakout board is constructed in order to access these pins through screw terminal connections. A schematic of this breakout board is included in the appendix. Finally, the board supplies PWM signals at 5 volts. This signal must be amplified to between 7-12 volts. A schematic is included in the appendix.

Figure 18 illustrates the overall system design.

Table VI lists the various connectors.

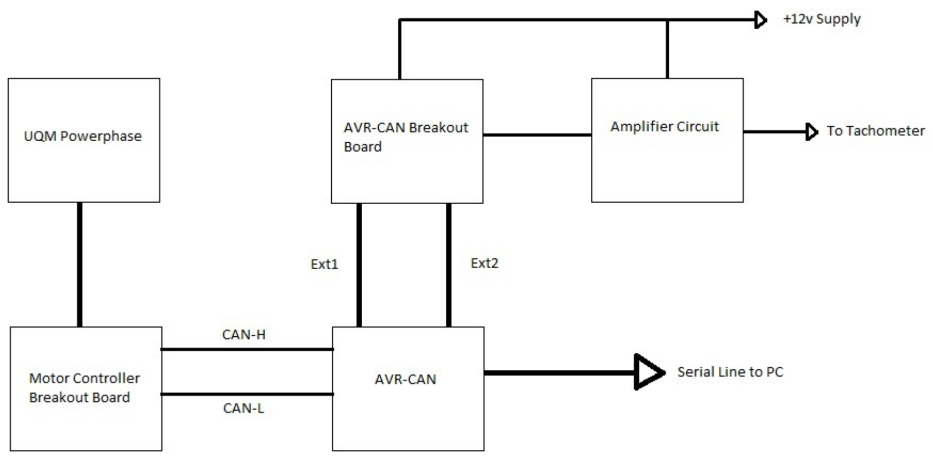


Figure 18: CAN-SPY System Design

Line	Connectors
Motor Controller - Breakout Board (MC)	Am19 to Am19
Breakout Board (MC) - AVR-CAN (AVR)	Stripped to DE-9
AVR-CAN - Breakout Board (AVR)	2 x 34 Pin IDC
Breakout Board (AVR) - Amplifier	Stripped to Stripped
AVR-CAN - PC	DE-9 to DE-9
AVR-CAN - Tachometer	Stripped to Stripped
+12v Supply Lines	Stripped to Stripped

Table VI: Connector specifications in system

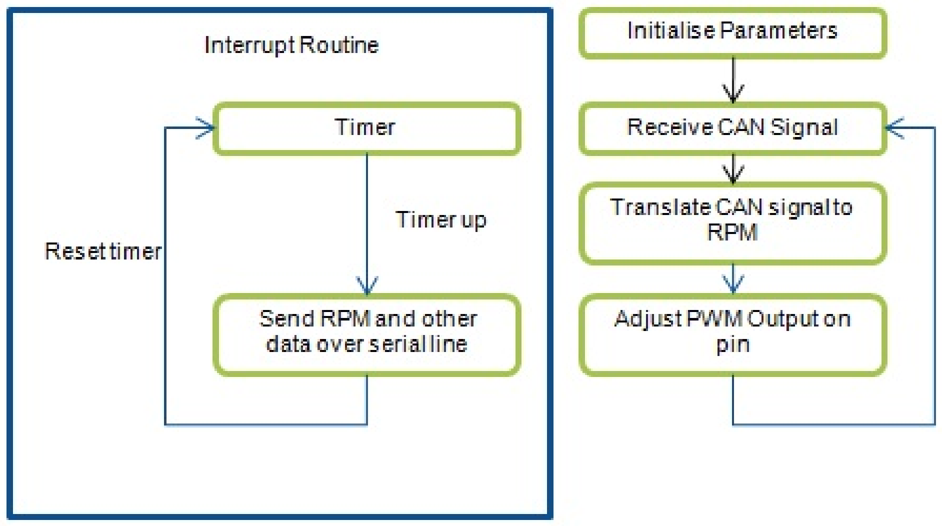


Figure 19: CAN-SPY Program Flow

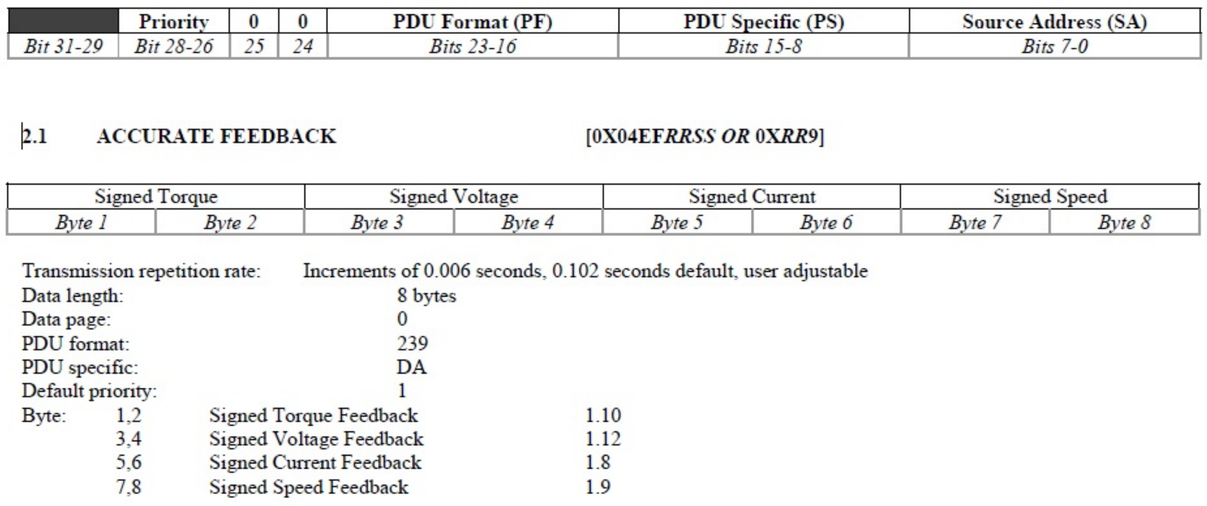
4.2.2 Software

The program design is quite simple and is done entirely in C. It must be compiled with Atmel Studio Version 4, as the newer version 5 does not currently support all Atmel chips, of which the AT90CAN128 is one. A suitable CAN Library was found, although it was used for the AT-DKV90CAN1 development board from Atmel, and not Olimex. As such, there are subtle differences in the library, although most code unessential to the boards functions have been removed.

From here, the program design is quite simple, and illustrated in Figure 19.

The CAN line on the Lotus is to be run at 500 kbit/s. This will result in a message received in 0.1 milliseconds (2 microseconds/bit, 50 bits per CAN frame). By far the bottleneck of the system is the serial line; 10 kbit/s results in a time of 100 microseconds to transmit one bit. As such tuning the timing of messages so that the tachometer can reasonably respond to changes whilst emitting serial messages is the most challenging element. Setting the timer to 100 microseconds should be all that is required to transmit messages and drive the tachometer.

Most important is of course capturing the relevant message. Fortunately most CAN controllers off the ability to filter for specific messages by specifying a message and a mask. In doing so, only the selected message or range of messages are written to the CAN controllers buffers. The particular message that is to be filtered for is displaying in Figure 20.



	Priority	0	0	PDU Format (PF)	PDU Specific (PS)	Source Address (SA)
Bit 31-29	*Bit 28-26*	*25*	*24*	*Bits 23-16*	*Bits 15-8*	*Bits 7-0*

2.1 ACCURATE FEEDBACK [0X04EF*RRSS OR* 0X*RR*9]

Signed Torque		Signed Voltage		Signed Current		Signed Speed	
Byte 1	*Byte 2*	*Byte 3*	*Byte 4*	*Byte 5*	*Byte 6*	*Byte 7*	*Byte 8*

Transmission repetition rate:		Increments of 0.006 seconds, 0.102 seconds default, user adjustable	
Data length:		8 bytes	
Data page:		0	
PDU format:		239	
PDU specific:		DA	
Default priority:		1	
Byte:	1,2	Signed Torque Feedback	1.10
	3,4	Signed Voltage Feedback	1.12
	5,6	Signed Current Feedback	1.8
	7,8	Signed Speed Feedback	1.9

Figure 20: Accurate Feedback CAN Message [Source: UQM CAN Communication Summary]

4.2.3 Alternate Hardware Design

Cost is an important consideration of the design, and one that wasn't looked at too closely in the preceding hardware design section. The platform selected was the AT90CAN128 chip, and more specifically the AVR-CAN development board from Olimex. It should be noted that whilst the one-off costs of a development board are acceptable, use in a production environment is prohibitively expensive. In the case of building the CANSPY, it was an extremely fast way to develop the functionality required by the project. While the use of this particular development board is acceptable as a once-off proof of concept, it is not a sustainable design decision in the long-term, and more purpose built solutions should be used in future.

Such a solution could be to purchase AT90CAN128 chips and produce purpose-built boards. This is a good idea, provided that they utilise all the chips features well, as at around $11 per chip it is quite expensive. Downgraded versions of this chip exist, most notably the AT90CAN32, whose only difference is a reduction from 128 kB of flash memory to 32 kB, at a reduced price.

There are of course, other chips in use. The SJA1000T from NXT Semiconducter (originally Philips) is a stand-alone CAN controller. This chip is found in many commercial CAN products, including the PCICAN from Kvaser. It is probably the most popular CAN controller on the market currently and has a low cost of approximately $7 per chip. This chip, unlike the AT90CAN128, does not include multiple I/O channels and PWM, so it must be adapted for use with another micro-controller. Though this adds to the cost, it is a more versatile solution. This means that smaller chips such as the ATiny series of Atmel controllers could be used to drastically reduce the

Chip	Purpose	Price (1)	Price (25+)	Notes
AT90CAN128	uC CAN Controller	$10.52	$9.38	Requires Transceiver, Has I/O & PWM
AT90CAN32	uC CAN Controller	$9.41	$5.90	Same as above
SJA1000T	CAN Controller	$7.68	$6.20	Requires uC, Separate transceiver optional
TJA1054	CAN Transceiver	$3.40	$2.73	Bare-bones transceiver w/ Fault Tolerant Mode
MCP2551	CAN Transceiver	$1.50	$1.15	Bare-bones transceiver w/ High Speed Mode

Table VII: Costs of Various Chips [Source: Mouser]

footprint of designs and the number of unused pins. Unlike the AT90CAN128, it does not require an external transceiver, although it can make use of one if the engineer so wishes.

Finally, a discussion on alternate design would not be complete without discussing the role of the CAN transceiver. The role of transceiver is to convert the digital I/O of the CAN controller to that of the system used by the bus as defined in the CAN standard. Whilst some chips like SJA1000T include in-built transceiver functionality, it is more common to use a separate transceiver. There are several transceivers available, but the most common are the TJA1053, TJA1054, PCA82C252 and MCP2551. The choice of transceiver is usually not greatly important provided they conform to the standard and specification that the designer is employing. As most high layer protocols implement the basic CAN standard, just about any generic CAN transceiver will perform. Some do have extra features, such as low power and sleep modes, which should be considered. Table VII displays various costs of different chips as listed on Mouser.

4.3 Evaluation

Due to damage done to the REV Racer and it's subsequent absence for repairs, testing and implementation could not be done directly on the car. Instead, individual elements were tested and confirmed to function as per Table VIII.

If all tests pass, it is likely implementation into the Lotus will succeed. It must be noted that the Lotus motor controller must be connected to a PC and be configured to send CAN signals using the supplied software that came with the UQM Powerphase 75. The final system is to be housed in a project box and installed in the Lotus.

Serial	Test with loop back to confirm that UART Library is operating correctly.
CAN	Connected to CAN compatible vehicle, output checked with serial to PC connection.
PWM Signal	Program designed to output 0Hz to 350Hz, increasing by 50Hz ever 3 seconds until upper limit. Confirm with probe.
Tachometer	Test frequencies based on Table V

Table VIII: CAN-SPY Testing List

4.4 Summary

This section outlined the basic workings of the CAN protocol, providing a reference for future and a base for CAN-based developments. Several different CAN controllers have been reviewed and their strengths discussed. A CAN based approach has been developed for attaining information from a CAN enabled motor controller, and has been used to drive an analogue signal. In addition, a systems has been described for passing CAN message to a PC.

5 Battery Estimation

5.1 Background

Battery estimation is a large issue currently facing EVs. An important feature of any vehicle is being able to accurately estimate the amount of fuel remaining, so as to make decisions regarding use of the vehicle. It also a particular important parameter with respect to evaluating the range of the vehicle, and it's behaviour at differing battery levels. The battery cells typically used in EVs are Lithium Iron Phosphate ($LiFePO_4$ cells) which have an extremely flat voltage-to-charge ratio, so common terminal voltage estimation methods do not work. Thus, research and development of more accurate methods is required. Suffice to say, a vehicle without the ability to determine it's fuel level would not be successful in the market. It is important to note though that fuel gauges in cars are notoriously unreliable, and tend to have a large margin of error. As such, any estimator that could achieve +- 10% of the actual fuel level would be appropriate for implementation.

The battery cells employed in the Lotus are $LiFePO_4$ cells. This chemical make-up was discovered at the University of Texas in 1997 [46]. They are reasonably low cost, non-toxic and exhibit high thermal stability. A flat voltage-charge ratio essentially means that these cells can deliver close to constant power for reasonably low C-rates, which is ideal for EVs. A graph displaying discharge capacity vs terminal voltage for differing C rates is displayed in Figure 22. There are some drawbacks with $LiFePO_4$ over conventional lithium cells. They have a lower energy density

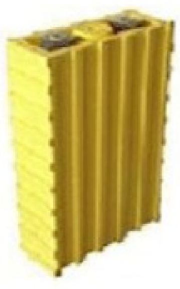


Figure 21: Thundersky $LiFePO_4$ Cell [Source: Thundersky Pty Ltd.]

and discharge rates are also reduced. In both cases, this is usually solved by using a larger battery, but research is being undertaken to improve both aspects. These advantages are also slightly offset by $LiFePO_4$'s longer life span in contrast to conventional lithium-ion cells. $LiFePO_4$ cells are currently the most popular cells used in hobby electric vehicle conversions, and in a few commercial vehicles. Whilst Tesla employed $LiFePO_4$ cells in the Roadster it is speculated that the new model S uses a $LiNiO_2$ battery pack jointly developed between Tesla and Panasonic that offers an improved capacity, based on Panasonic's 18650 cell technology [47][48].

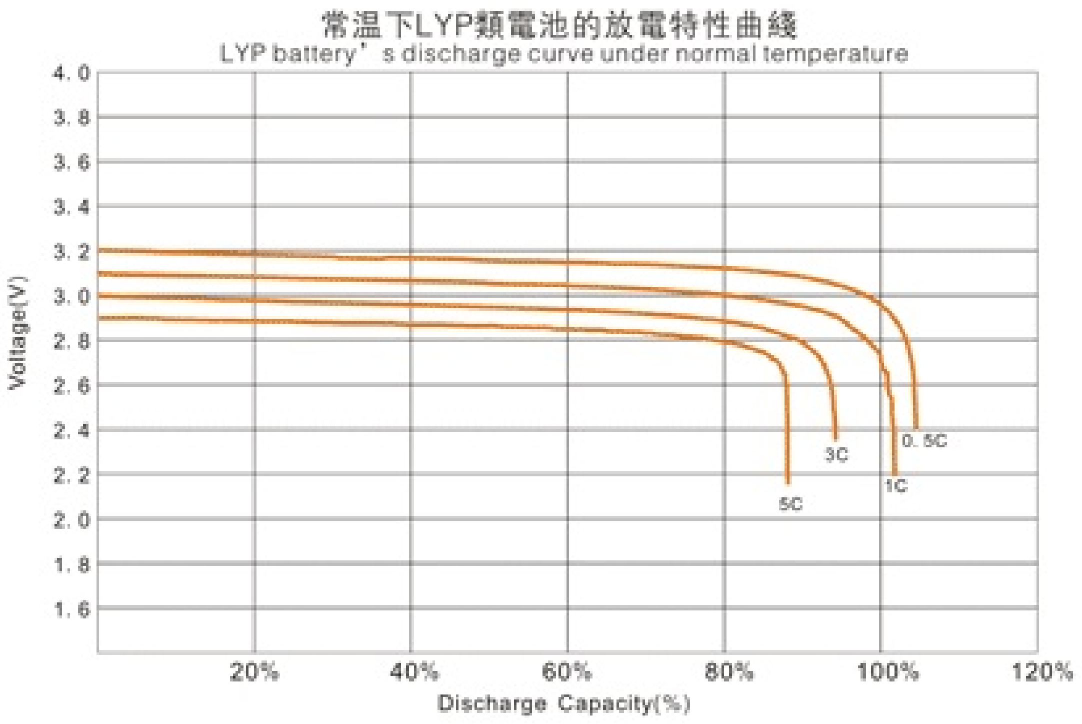


Figure 22: $LiFePO_4$ Discharge Characteristics [Source: Thundersky Pty Ltd.]

In estimating the capacity of a battery cell, there are a few main elements to be considered. The first is of State of Charge (SoC). Anyone who has used a car with a fuel gauge will be familiar with this measurement, which is essentially an estimate of the remaining that can be used to drive the vehicle with respect to a reference point of 100% capacity. The second is State of Health (SoH). SoH is typically calculated as a ratio of total capacity of the pack currently, compared to what it was when the pack was new. Battery packs tend to lose some charge as they age, so this is essentially what SoH is measuring. This section's main focus is on determining SoC, and so SoH is mostly ignored.

In using a battery pack as opposed to just a single cell, we suddenly introduce a third element of capacity, pack balance. Balance is essentially a measure of how close all terminal voltages in the pack are to the mean voltage. In a perfectly balanced pack, not only is terminal voltage equal across every cell, but so is the remaining capacity of each battery. Pack balancing is the responsibility of the BMS, and is by far it's most important aspect. It does not matter if 99% of the cells in a pack are currently at 100% of their rated capacity; if even one cell is at 0%, the vehicle cannot be driven or else that cell will be damaged and require a replacement. In summary, this means that a battery pack's usable capacity is limited by the capacity in the most depleted cell.

Before discussing the most popular methods for estimating SoC, it is important to outline some criteria that must be met in order to estimate SoC on the Lotus. The Lotus has some special requirements that are not present on commercial EVs due to the way in which varying electrical elements of the car are designed to operate. The car PC, BMS, battery pack and car electrics are all operating on different power circuits [44]. As such, it it possible for the BMS and the car PC to be turned off whilst the rest of the car is on. A consequence of this is that the BMS may not be able track every event that is occurring to the pack, and as such, tracking the SoC by knowledge of it's previous state alone will not accurately determine SoC. Likewise, if the car PC is switched off, it obviously will not receive data from the BMS. Most importantly, this means SoC must be determined from solely from parameters at a particular instant in time, or at the very least converge to an accurate estimate after some period of time. Secondly, online computation of SoC should be kept to a minimum. Ideally the driver wants accurate estimates of remaining capacity as often as possible. Most research papers regarding battery estimation methods for electric vehicles perform tests on one cell and suggest creating estimators for every cell in a pack. A lot of estimation methods use integrals, derivatives, multiplication and division, which by their nature are computationally expensive for a computer or a micro controller to perform. In the case of the Lotus, which uses an 83 cell battery back, this suddenly takes a lot of processing time. It is also likely to be inaccurate as the Lotus only attains new data for a particular cell roughly every 15 to 20 seconds. As such, it is important that any estimation methods extends itself to being easily generalised to the entire pack SoC.

In the following subsections are discussions of various battery estimation methods. Another good review of battery estimation methods is present in a series of papers by Gregory L. Plett, although he does not detail models based on learning algorithms [49][50][51].

5.1.1 Coulomb Counting

Coulomb counting is by far the most presently used technique is estimating capacity of battery cells that do not have a strong relationship between capacity and terminal voltage. Battery capacities are typically rated in amp-hours, so the problem is well conditioned for calculating the depletion of the battery by monitoring the current being drawn from it. However, it does suffer from several before mentioned caveats. Particularly, it's dependence on knowledge of the previous state. Additional drawbacks include inaccuracies in current sensor readings. However, these issues can be accounted for with more sophisticated implementations.

The most significant drawback with coulomb counting is it requires knowledge of the initial state and must monitor the current at all times. The simple requirement has one drawback that is not often considered; having the electrics 'on' all the time, creates an added risk where if the vehicle is not correctly put on charge the potential arises for the battery pack to be drained out overnight much faster than it would be if it were disconnected. In addition, there is always some power leakage occurring, and thus the naive solution of saving the last SoC of the battery before switching all power off is not likely to be accurate.

Another factor is the concept of measurement drift, where inaccuracies in readings taken over a period of time can add up and contribute to significant error. Coulomb counting is particularly prone to this, as it ideally requires attaining values for instantaneous current as often as feasibly possible. For this reason, coulomb counting is often paired with a way to reject accumulated errors. The two most used methods are recalibrating the measurement to some set point, usually empty or full capacity [52], and/or the use of a Kalman filtering technique [53].

In spite of the drawbacks of coulomb counting, it remains a popular solution. Utilising it with a recalibration is simple to implement, and has lead to it being utilised in a few commercial solutions. The most familiar implementation to members of the REV Team is the eXpert-Pro from TBS electronics (Figure 23). This module is the one currently used in the REV Getz to estimate SoC and is reasonably popular with EV enthusiasts in Australia. Coulomb counting will form part of the solution for estimating the remaining state of charge in the REV Lotus.

Figure 23: TBS eXpert Pro: A Coulomb Counting State of Charge Indicator [Source: TBS Electronics]

5.1.2 Electrochemical Modelling

Electrochemical modelling is another method of estimation. This technique attempts to model a cell at a molecular level and determine the SoC therein. Due to the nature of chemical reactions, it employs significantly more complex mathematics than coulomb counting [54]. Such models offer great accuracy in estimating the remaining capacity of a cell, although their complexity is often what makes them ill conditioned for direct implementation into vehicles [50].

The many parameters used in a cell model must be obtainable from the vehicle. This often includes variables that are not usually obtained from most battery management systems, such as individual cell temperature. The biggest drawback of such a system is the extra number of sensory data that would be required, increasing cost and complexity of systems in a vehicle. As such, it is unlikely that a commercial vehicle would implement such a model, especially as other methods of SoC determination are cheaper to implement.

Most importantly, to build a proper model of a battery, there are certain parameters of the batteries make-up that must be known. Between different manufacturers, various factors can differ between chemically similar batteries, such as anode/cathode surface area. Such differences can lead to differing performance. Unfortunately such detailed information is not available as most manufacturers are hesitant to make public their designs. The only option then is to reverse-engineer and probe the battery for it's various attributes, but this is obviously expensive and most of all dangerous as most batteries are toxic.

The complexity and expense of chemical methods for determining SoC rule out their use in the REV project. Other methods are far cheaper and more suitable for the needs of the project.

5.1.3 Artificial Neural Networks

Over the last ten years Artificial Neural Networks (ANN) have found increasing use in battery estimation problems [55] [56] [57] [58] [59]. ANNs are mathematical models whose structure is partly inspired by biological neurons. The greatest strength of these networks is their ability to approximate non-linear functions through machine learning, and have been described as universal approximators [60]. Given a good set of training data, it is possible to train an ANN to approximate battery dynamics. Neural networks have several advantages and disadvantages that will be detailed. For the purposes of brevity, information in this section is restricted to feed-forward back-propagating neural networks.

Selection of network structure is important. Within Back Propagating (BP) networks, there are two main structures, feed forward and recurrent networks. Feed forward networks are as the name suggest, and simply feed data through to the output through a number of layers. Recurrent networks include recurrent layers, which are usually a connection to and from the hidden layer. Such a connection allows the network to contain memory of previous inputs. Adding a recurrent layer does add some extra computational cost to the system. Some typical neural network structures are presented in Figure 24.

The next element of network structure is the selection of an appropriate number of input nodes, hidden layers, hidden nodes and output nodes. The selection of input nodes and output nodes is defined by the problem. It has been proven that only one hidden layer is necessary to estimate non-linear functions. The only major design choice left is the number of nodes to use in the hidden layer. There are many 'rules-of-thumb' that have been created, although many in the community disagree as whether they have any merit [61]. Wanas recommends that the number of hidden nodes be selected based on the logarithm of the number of training samples [62]. The only consistent opinion is that only through experimentation can one derive a reasonable number of nodes to use. It is important to note that increasing the number of nodes increases the time it takes to forward and back propagate exponentially, so if execution time is a major concern the hidden node count should be kept as low as feasibly possible.

The choice of activation function is also important in defining an neural network. It is required that the activation function used is differentiable. Furthermore, it is required that the activation function at the hidden layer is non-linear, as this is what enables the neural network to act as a

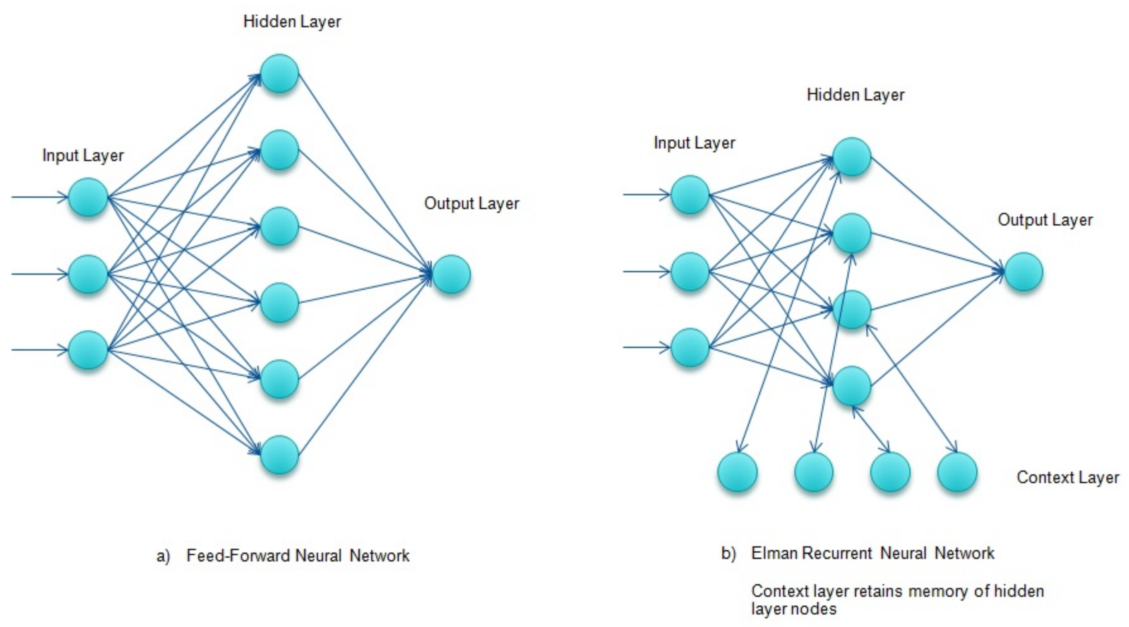

Hidden Layer
Input Layer
Output Layer
a) Feed-Forward Neural Network
Hidden Layer
Input Layer
Output Layer
Context Layer
b) Elman Recurrent Neural Network
Context layer retains memory of hidden layer nodes

Figure 24: Feed Forward and Elman Recurrent Neural Network Structure

non-linear estimator. The four most popular functions are Sigmoid, Gaussian and Radial-Bias functions. Functions selected at the input and output layers are usually chosen to mimic the data that should be input and output. To use the battery estimation problem as an example, the sigmoid function would be a good choice because output is clamped between 0 and 1 i.e.: output is restricted between 0 and 100%. Usually this does not have too much effect on the overall function approximation. The activation function at the hidden layer is far more important. Sigmoid and tansig functions are popular as they are easily differentiable by computers, and thus do not take a lot of time to calculate.

Ideally, inputs to neural networks should be normalised. A common goal is to ensure inputs are kept between -1 to 1. This ensures that when summed and input to an activation function that reasonable output is produced. If the input values are too high, it will generally cause the activation function to lock the neuron to a value that is close to either zero or one. Essentially, the neuron becomes 'always on' or 'always off'. As such, normalising inputs is extremely important.

ANNs can learn by three methods; supervised, unsupervised and reinforcement learning. In the former, input is fed to the neural network and the output compared to some expected data. Unsupervised learning is more complex and relies on the machine to learn to categorise it's outputs by itself. Reinforcement learning is more applicable in AI systems where input data is produced by the agent acting on it's environment. For our purposes, supervised learning is more appropriate

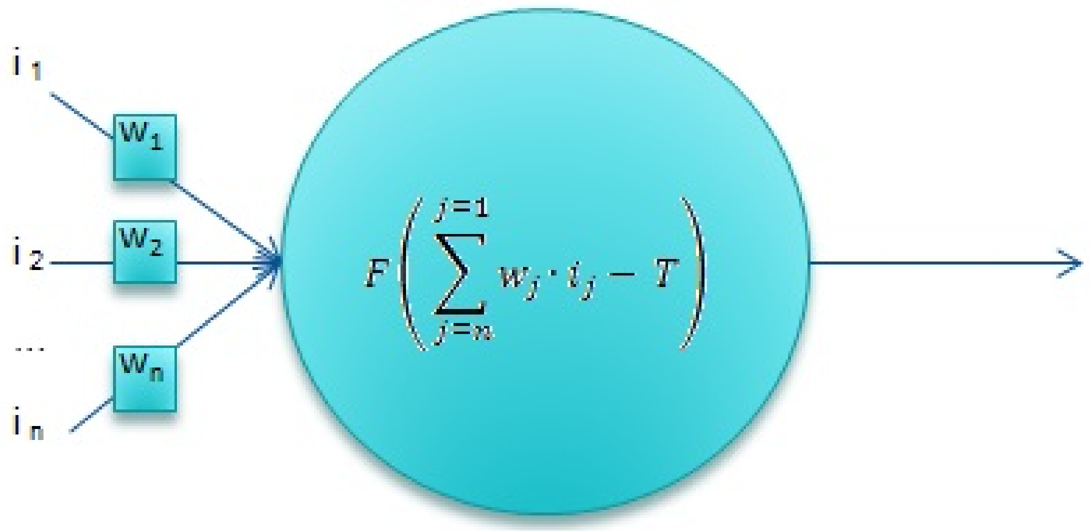


Figure 25: Forward Propagation Mechanism at the Node

as we are attempting to shape the network to approximate a certain response from a set of input data. The learning mechanism is described mathematically in Figure 26.

$$e_k = d_k - y_k \quad Output\,Error \tag{2}$$

$$\delta_k = \frac{dF_k}{dy_k}.e_k \quad Output\,Layer\,Error\,Gradient \tag{3}$$

$$\delta_j = \frac{dF_j}{dy_j}.\sum w_{jk_t}.\delta_k \quad Hidden\,Layer\,Error\,Gradient \tag{4}$$

$$w_{ij_{t+1}} = w_{ij_t} + \Delta w_{ij} \quad Input - Hidden\,Layer\,Weight\,Adjustment \tag{5}$$

$$w_{jk_{t+1}} = w_{jk_t} + \Delta w_{jk} \quad Hidden - Output\,Layer\,Weight\,Adjustment \tag{6}$$

$$\Delta w_{ij} = \alpha.w_{ij_t}.\delta_j \tag{7}$$

$$\Delta w_{jk} = \alpha.w_{jk_t}.\delta_k$$

$$where\,\alpha\,is\,the\,learning\,rate$$

Figure 26: Neural Network Back-Propagation Equations for a Three Layer Network Structure

The most difficult aspect of ANNs is training. It is difficult to ensure that the training data supplied to an ANN is going to be appropriate to approximate a particular function. The problems of overtraining and undertraining are relevant to this discussion. As we are only looking to estimate a curve, and not solve a characterisation problem, overtraining of the ANN is unlikely to affect our estimation. Undertraining is a huge concern. The largest set of any training data produced by the

REV Lotus is going to be the battery placed under a 30 ampere load or less. From the battery dynamic figure presented earlier, this is not a load that is particularly good for estimation. As such, it is important to ensure that training is weighted towards situations where the battery is under an increased load, lest it tend to favour data produced under low-load conditions. In addition, it should be obvious that the closer the relation of the input data to the output data, the more accurate the neural network will be.

It is important to note that although the ANN can approximates non-linear function, it does not infer that the internal structure and weights have an specific meaning. It is very difficult to infer meaning from layer weights, and it not normally possible to create a model of battery dynamics from understanding the network structure. As such, neural networks are best viewed as black boxes.

Due to the ease of implementation and versatility of neural networks, an implementation is used for charge estimation in this project.

5.1.4 Kalman Filter

The Kalman filter [63] is one of the most famous developments in control systems theory. It is a mathematical model that allows measurements to be taken and computed to form a more accurate reading and reject noise. It does this by taking measurements, comparing them to a model of the system and computing a more accurate reading based on the noise of the model and the actual measurements. The Kalman filter is simple to implement, and provided an appropriate model can be constructed, tends to produce stable and accurate readings. Although there are many variants of the Kalman filter, namely unscented and extended variants for systems that are non-linear functions of the system states, this discussion will restrict itself to the standard Kalman filter.

The Kalman filter can be thought of as a two step process, the first of which is commonly referred to as the 'predict' state. This system is displayed graphically in Figure 27. During this step a prediction of the next state is made using the associated model of the actual system. An update of estimate covariance is also made. The second step is the 'update' state. The measurement residual is found by subtracting the predicted next value from an actual measurement of the system output. From here, the covariance, optimal Kalman gain and improved measurements are obtained.

There are two important values to be considered in the Kalman filter, aside from the actual model and obtaining the relevant measurement. These are the values for process covariance noise (commonly denoted as Q) and the measurement covariance noise (commonly denoted as R). A larger

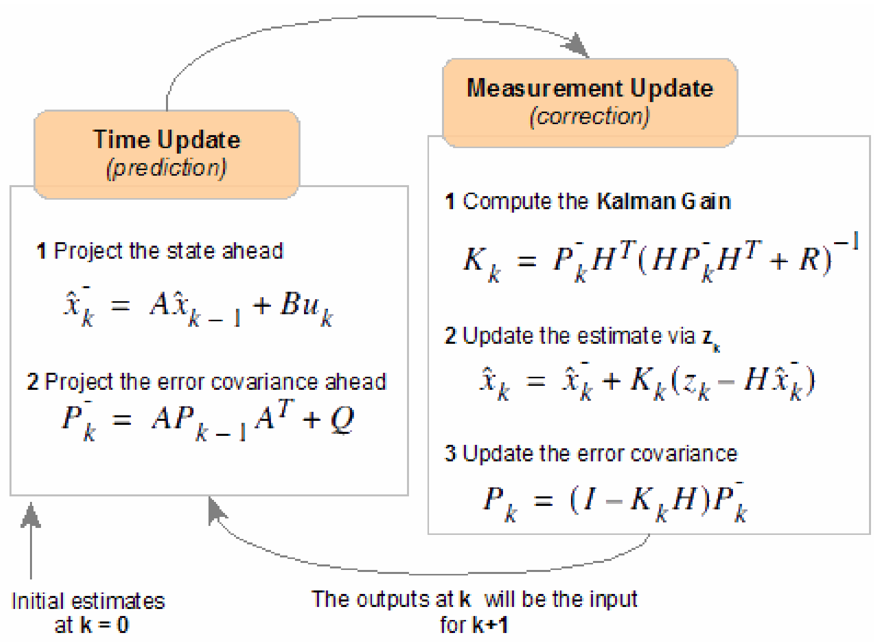

Time Update
(prediction)
1 Project the state ahead
$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k$
2 Project the error covariance ahead
$P_k^- = AP_{k-1}A^T + Q$
Measurement Update
(correction)
1 Compute the Kalman Gain
$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$
2 Update the estimate via z_k
$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$
3 Update the error covariance
$P_k = (I - K_k H)P_k^-$
Initial estimates at k = 0
The outputs at k will be the input for k+1

Figure 27: The Kalman Filter [Source: Bilgin's Blog]

value for either of these indicates less accuracy in the respective system. As such, using a correct ratio can weight the system towards 'trusting' measurements more or less. As it is often not known what the exact values for covariance noise are, they commonly derived from experimentation and tuning.

The Kalman filter forms the final piece of the battery estimation algorithm, and will be used alongside a neural network and coulomb counting to form an estimate for SoC. This is not unusual, as the Kalman filter in one form or another is implemented in many battery estimation algorithms.

Although not used in the final method, a review of battery estimation techniques would not be completed without briefly discussing genetic algorithms, simulated annealing and fuzzy logic.

5.1.5 Genetic Algorithms & Simulated Annealing

Genetic Algorithms (GA) and Simulated Annealing (SA) are two methods that are employed to find optimum solutions in a search space, though both approach this in a different manner. GA operates similarly to evolution and normally utilises strings which act as an analogue to DNA that represent attributes of the estimation function, and uses mutations, crossover, and inheritance to formulate optimal solutions. SA is based on the metallurgic concept of annealing, whereby metals are heated to cause 'atoms' to be come unstuck, and are then cooled so they settle in an optimum position. The main point of difference between the two methods is that at any one time SA only has memory of discarded solutions and the most optimum solution, where as GAs maintain a pool of solutions from which to spawn improved estimators. Both suffer from similar problems, the most important of which is the threat of the optimising on local minima and getting 'stuck', although there are methods to reduce this.

In SA each point x of the search space represents the state of some system, and the function E(x) represents the internal energy of the system. From this, the goal is to reduce the function E(x) to it's minimum value. As an example, this value would be the mean square error (or some other measure of fitness) of the battery estimation function when applied to some testing set of data. New function estimators are generated that neighbour the current function in the search space, and if fitter than the current selection they replace it. This can continue until it reaches some pre-set error tolerance or allocated computation time is exhausted.

GAs take a different approach to finding the optimal solution in a search space. A set of strings are used to encode a random set of solutions. In every generation, a random set of individuals are selected, evaluated according to some marker of fitness, and modified (either combined or randomly mutated) to form a new population. This new population then undergoes the process again, and it continues until a candidate meets the fitness criteria or computation time has expired.

Either of these methods can form part of a valuable contribution to battery state estimation. Neural networks in particular can be sensitive to their starting weights. Use of an SA or a GA approach could aide in producing a more accurate estimator by finding the optimal starting weights and/or hidden layer nodes [64] [65].

5.1.6 Fuzzy Logic

Fuzzy logic is the final method to be discussed. Fuzzy logic is a form of many valued logic, and can be used to define logic values between 0 and 1. The fuzzy system has been proved to be a universal approximator and is capable of fitting any non-linear function. The membership functions of the set need to be generated by an expert or some other system. This could be derived from battery characteristic models or from an unsupervised neural network. In operation, there is a mapping between the inputs to some fuzzy set, which is later transformed to give a 'crisp' point that is more certain of the measurement being made.

Popular models for fuzzy logic are often defined from impedance measurements and frequency response of the target cell. Whilst research on using fuzzy systems alone in state of charge determination is available, more recent research tends to focus on adapting it with other methods, such as neural networks, to form neural-fuzzy networks. More recent research focuses on using fuzzy logic for BMS functions in addition SoC estimation, in order to maintain health of the battery pack [66] [67].

5.2 Design & Evaluation

Many different networks were constructed, trained and evaluated in creating the final estimator. Initially, an entirely neural network approach was intended, based on previous work that was available in the literature. However, these papers had access to more training data and tended to only operate on single cells and not battery pack. Their tests were also far simpler, usually involving standard battery discharges tests such as the Federal Urban Drive Scheme (FUDS) and Manhatten power cycles, which generally place higher loads on batteries than those in practice. Limitations on data obtained from Lotus BMS, in comparison to a bench test focusing on one cell, also leads to discrepancies as cell balancing issues obscure the true capacity of the pack. In the end, a multi-step approach was required to obtain a reasonable estimate for battery capacity.

Before continuing on with the design, a few notes on the training data must be made. The data consists of a single set of approximately 55000 data points. This was obtained from a test drive of the Lotus, performed on 5/08/2011 for approximately 110 minutes on a day with a max temperature of 23 degrees Celsius. During this drive, the battery pack could only discharge 30% of it's maximum value, as two cells were well below the pack average and further driving would have caused damage to the cells. Whilst it was planned to obtain further data, damage to the Lotus incurred just days later had meant access to the Lotus was suspended, and as such, no more training data could be obtained. There is a second training set, that was taken on 29/07/2011 , but as there was a scaling issue with the amperage of the coulomb count the data had to be discarded. Table IX illustrates what data was obtained, as well as statistical information of the training set.

Parameter	Mean	Median	Minimum	Maximum	Std Deviation
Min. Cell Voltage (per BMS cycle)	2.99	3.04	2.67	2.26	0.16
Ave. Cell Voltage (per BMS cycle)	3.26	3.26	3.19	3.34	0.026
Current	-10.1	-4.4	-128.4	28.4	13.99
SOC	0.81	0.8	0.71	1.0	0.01

Table IX: Statistical Data of Training Set

The parameters were sampled every 100 ms. The average cell voltage and minimum voltage per BMS cycle were selected for the model as this is roughly equivalent to the model whereby a load on the battery will produce a voltage drop. The minimum cell in the pack at any instant in time, if the pack is balanced well enough, is the cell that is sampled when drive current is at its maximum. The difference between the average cell voltage and minimum cell voltage is not precisely the voltage of any one cell under a particular load, but it is equivalent.

Finally, the system was designed and evaluated using the python programming language. Extensive use of the PyBrain [68] and NumPy [69] packages has been employed.

5.2.1 Artificial Neural Network

The first step is to build and evaluate neural networks in an effort to determine what a reasonable network structure for our purposes will be. In doing this, several different neural networks were constructed and trained on a set of data acquired from the a test-drive of the Lotus. Furthermore, this data is then passed through the neural network and the outputs are compared to the output of the training data. From this comparisons can be drawn on how well the trained networks match the output data. Two basic networks were tested with hidden layer composed with a 5-15 hidden layers. These were tested on a standard feed forward network (network structure in Figure 24), and an Elman-type recurrent network, the results of which are illustrated in Figure 28. In training the networks, batch-learning was applied with a training time of 30 epochs before termination, and every set of data in the training set was utilised. Initial weights are randomly generated between -2 and 2.

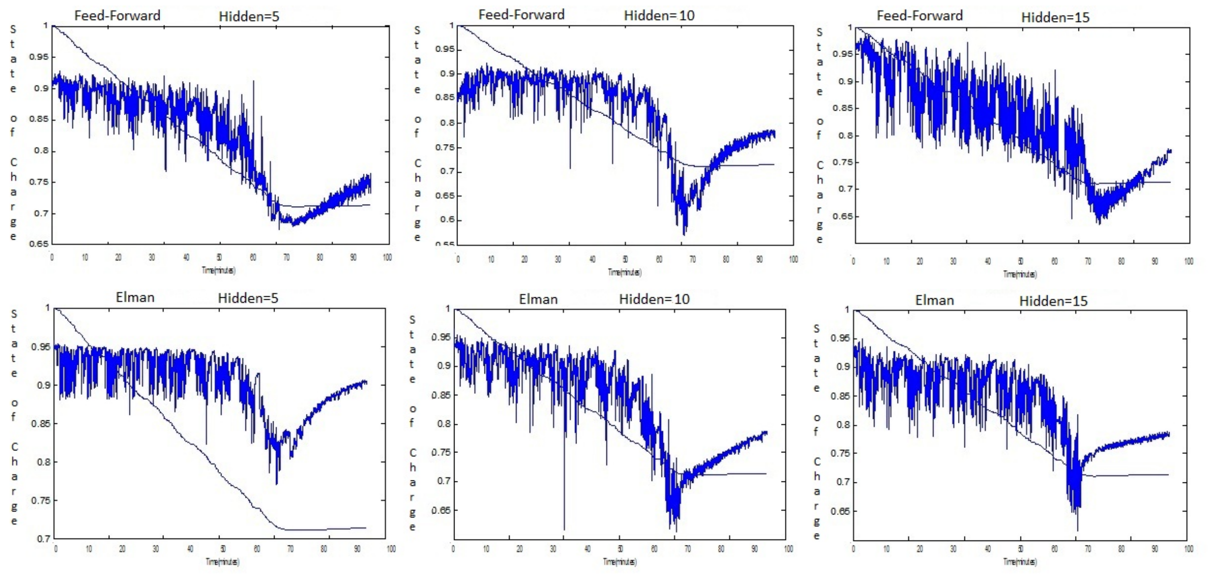


Figure 28: ANN Testing with Full Training Set, Variable Hidden Nodes and Network Structure

From the results, we can determine that the network structure and number of hidden nodes has little effect on the overall fitting. A second set of tests was done, this time only training the neural network on data points that had a current member that exceeded -30 and then -40 amperes. In doing so, this removes low load current values that do not drastically affect the internal resistance of the battery, regardless of the battery's discharge level. However, this does mean that estimations from the neural network do not occur as frequently. A positive of this approach is that this significantly reduces the amount of training the neural network is required to do, reducing the training set from the full amount of around 55,000 data points to around 3000. As a result, the network can undergo far more training, and therefore can create a better fit with less noise. These results are displayed in graphical form in Figure 29. From these, we can determine that using a reduced training set for estimation results in a better network, albeit one that can only be used for larger drive currents. This is still no ready for implementation into the vehicle, and will require the use of a Kalman filter to improve the estimation.

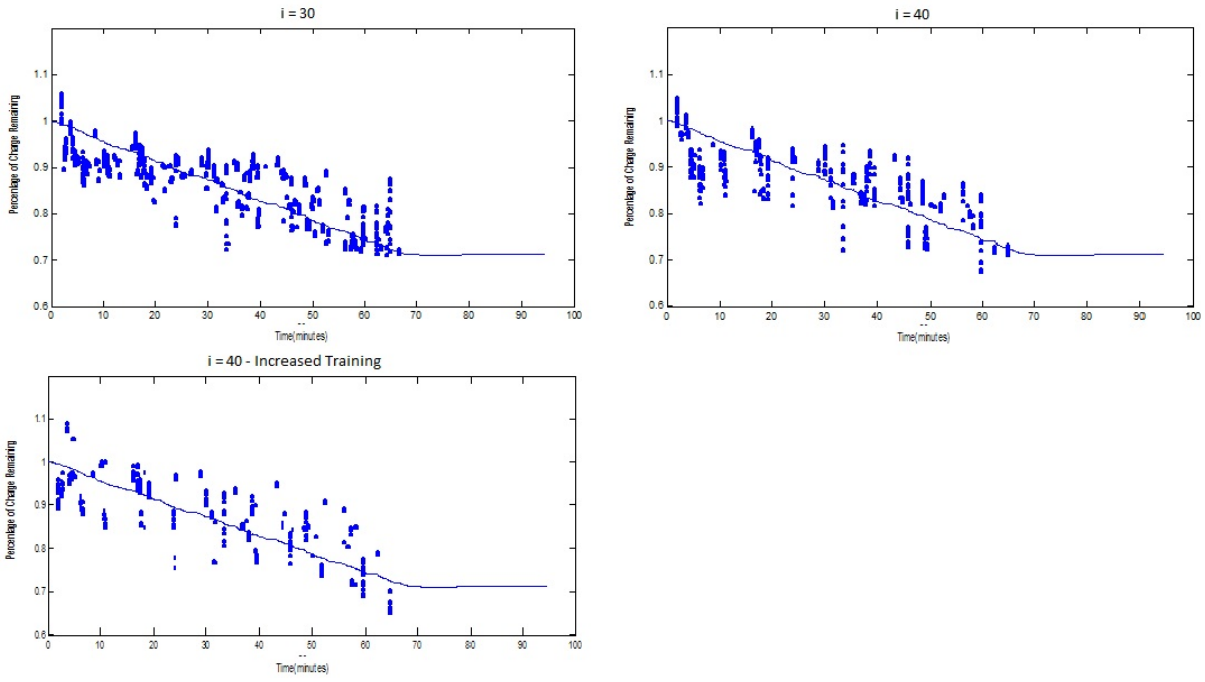


Figure 29: Neural Network Output when Ignoring Current Values Below a Threshold Value

5.2.2 Kalman Filter

As the neural network performs poorly on it's own another method is required to ensure a more accurate estimation. The method chosen to do this is the implementation of Kalman filter. In doing this, it is also expected that much of the noise will be removed from the filter. Figures 30 and 31 illustrate the mathematical models used in the development of the filter.

$$\hat{x}_k^- = \hat{x}_k - \frac{i_k.\Delta t}{60} \tag{8}$$

$$P_k^- = P_{k-1} + Q \tag{9}$$

Figure 30: Kalman Filter Prior Equations for State of Charge

$$K_k = \frac{P_k^-}{P_k^- + R} \tag{10}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(N_k - \hat{x}_k^-) \tag{11}$$

$$where\, N_k\, is\, the\, neural\, network\, estimate\, at\, time\, instant\, k \tag{12}$$

$$P_k = (I - K_k)P_k^- \tag{13}$$

Figure 31: Kalman Filter Posterior Equations for State of Charge

Testing of the filter is initially done using a neural network trained with the full set of training data. This results in the graph displayed in Figure 32. Comparing this side by side with the sole neural network method, we can see that we have successfully removed most of the noise from the estimator. The onus is then on matching the data correctly to actual SoC of the battery system. Improving this estimation is dependent on tuning Q and R parameters such that the system associates low load conditions with a greater degree of uncertainty than high load conditions. In effect, ideally the Kalman filter should discard estimations if the load drive current is below a certain minimum. However, it is important to remember that neural network estimation is poor if used over the entire training set. Instead then, the neural network that is partially trained using only high load current sets is utilised. When a low drive current is applied to the estimator, the system instead uses the coulomb count as it's measurement update, effectively skipping the measurement step. In doing so, the system essentially acts as a coulomb counting system that utilises a neural network to recalibrate the SoC when the load on the battery pack exceeds a pre-set limit.

Correct tuning of the Q and R parameters is necessary to form an accurate estimate. As coulomb counting is used in our model for SoC, selection of a suitable value is reasonably easy. A load of the average drive current per time sampling period results in a reduction of around 1e-4. Fluctuations in the current sensor generally occur at 1e-6, so we can safely assume this is approximately an amount of error suitable for the value of Q. Measurement noise R is harder to determine and is best found through experimentation.

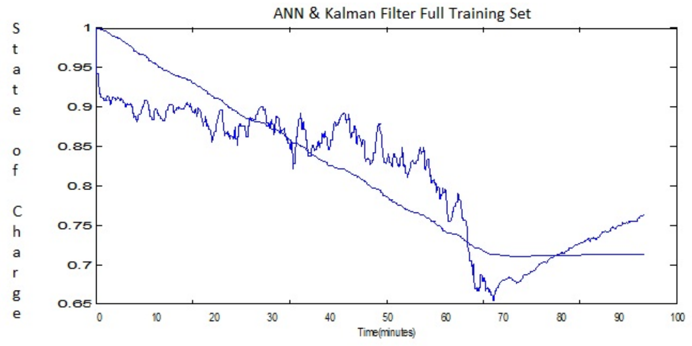


Figure 32: ANN & Kalman Filter Testing with Full Training Set

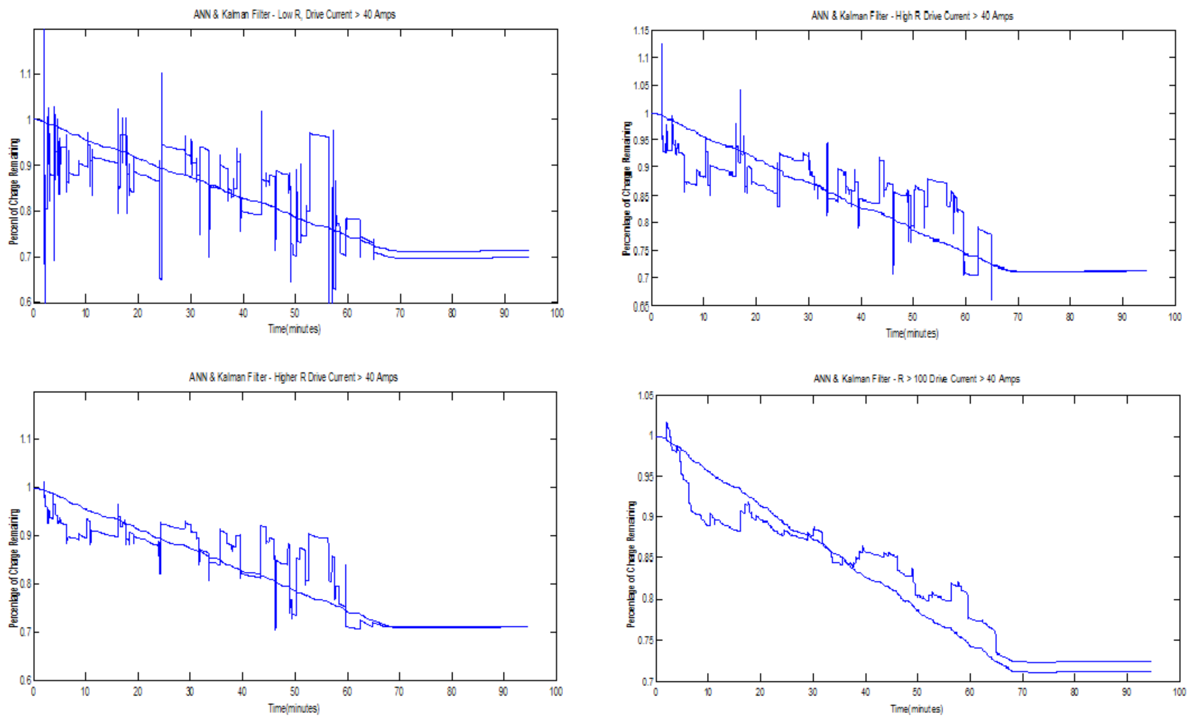


Figure 33: ANN & Kalman Filter with Variable R Values

As shown in Figure 33, generally better estimators are formed when the values for the measurement noise R are several magnitudes higher than that of the value being estimated. Whilst SoC is treated as a value between 0 and 1, keeping the measurement noise matrix R at between 10-100 seems to result in the best estimate function. Keeping the value for R high means that the system must attain several neural network estimates before it begins to affect the value. In doing so, the response of the estimator is reasonably smooth and outputs a reasonably close estimate of the actual SoC, with roughly 10% error at worst. Further testing is done to ensure that if the initial estimate is far below that of the actual SoC, that the system will converge. Figure 34 displays that this is indeed the case. This result is appropriate for implementation into the vehicle. A diagram of the final system is present in Figure 35.

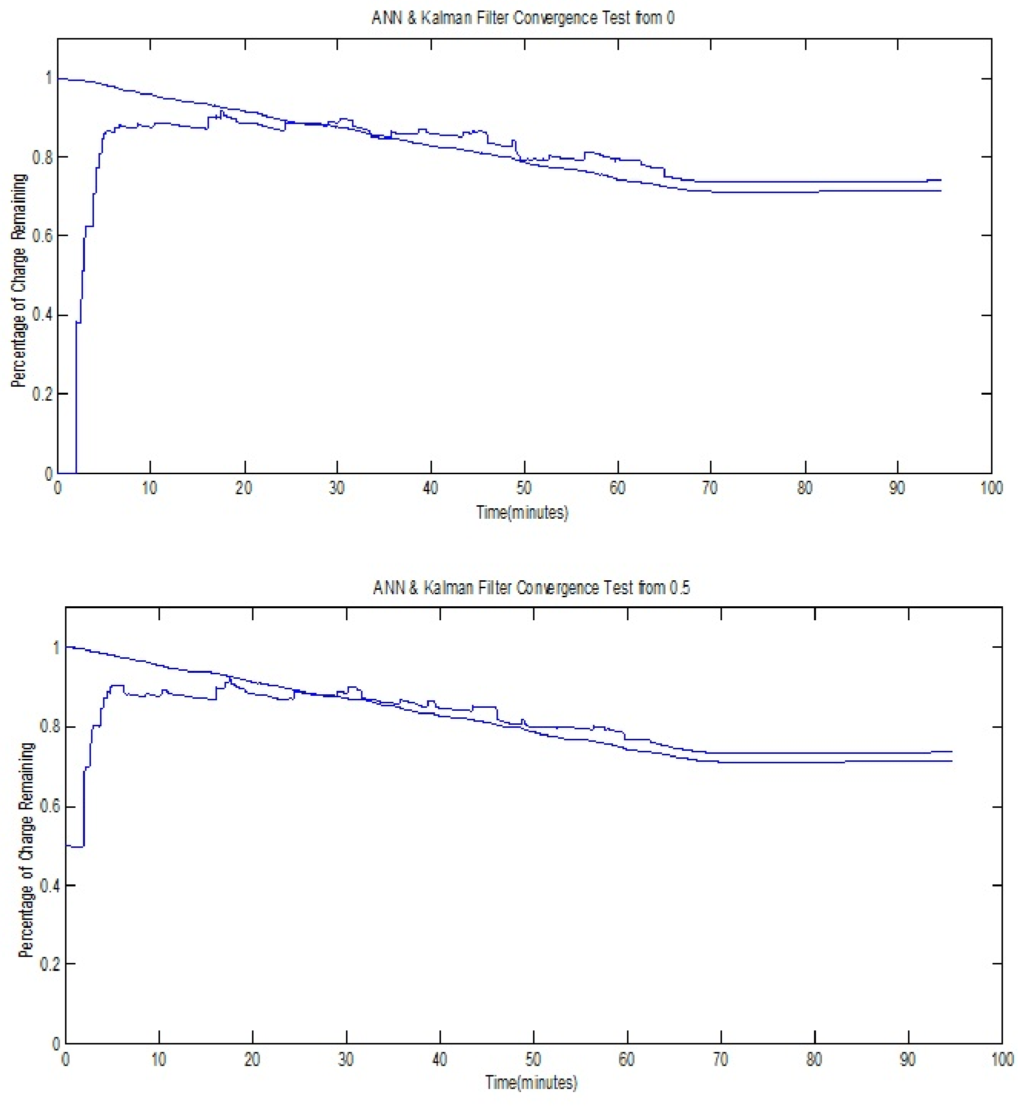


Figure 34: ANN & Kalman Filter System Convergence Tests

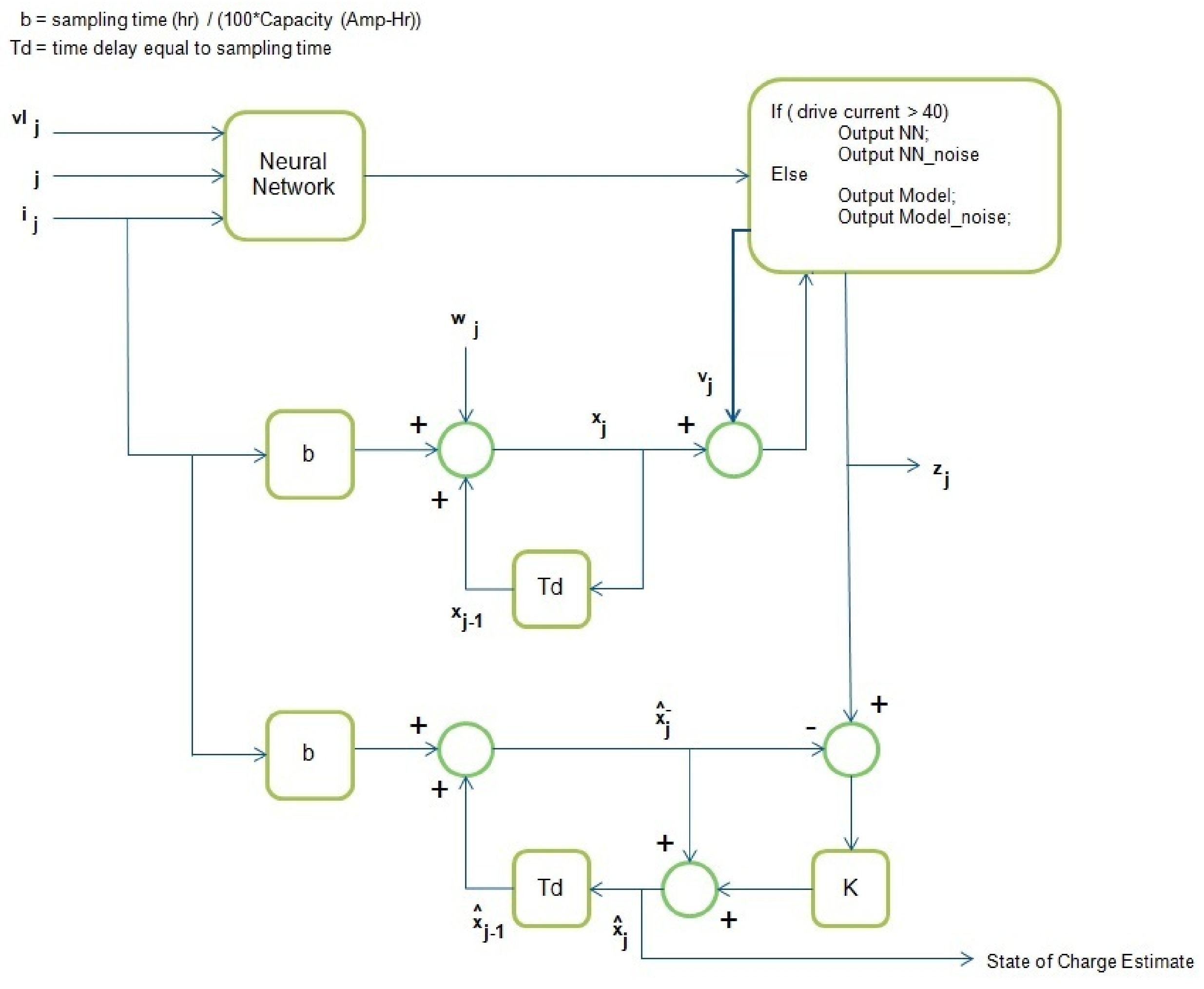


Figure 35: Block Diagram of Full ANN & Kalman Filter System

5.3 Summary

The section detailed various methods used recently in battery estimation problems and then proceeded to implement a neural network, Kalman filter and coulomb counting combined approach. Various designs were evaluated, including modifying the the neural network structure, and reducing the set of data used for neural network estimation. The Kalman filter was modified by trialling different values for estimate measurement noise, in order to gain the most accurate estimate for SoC. The result is a SoC estimator than can produce estimates within 10% of the actual value. This estimator still needs to be evaluated in the vehicle, and ideally needs additional training data to span the entire capacity of the battery.

6 Conclusions

This concludes the work done to further develop the systems for the Lotus in 2011. Various systems have been implemented and updated in order to improve the Lotus and prepare it for subsequent studies to be undertaken. Improvements have taken form in the implementation of Engine Sounds, a new method to determine battery capacity, research and development of CAN bus equipment, and updates the Lotus PC UI systems.

Engine Sounds was finally implemented into the vehicle, after a year and a half of development of the code. The final system emulates the noise of an internal combustion engine and will prove useful in future studies of pedestrian safety and driver acceptance. The Engine Sounds system has been improved in order to reject as many unwanted side effects of audio discontinuity as possible and provide an accurate reproduction of ICE noise.

The UI system underwent many changes to improve the backend and frontend of the system. Core changes to the telemetry, BMS and GPS backend systems have improved stability of the system and enable each component to be able to recover in case of error. The look and feel of the UI has been improved and should increase the appeal of using the system. Logging functions have been expanded to enable the system to be emulated off-line. Finally, the Twitter system was introduced to enable the the car to be able to receive traffic data from Main Roads.

CAN bus systems were reviewed in depth in order to create a better platform for attaching sensors in order to send and receive data better in the car. This does the groundwork necessary for distributed I/O in the car, and also provides information necessary to build a CAN-based motor controller. The first CAN system in the car was designed utilising an automotive motor controller, and is able to pass data from the CAN bus to the PC, in addition to driving a standard analogue tachometer.

A review of battery estimation methods was undertaken culminating in the implementation of a new estimation system for state of charge. The shortcomings of various battery techniques were discussed, as well as the issues with implementing a state of charge estimator in the lotus. An ANN - coulomb count - Kalman filter approach was designed and evaluated resulting in an estimator that can estimate state of charge within 10% of the actual value.

Several improvements were made to the Lotus in 2011, to differing elements of the car. This creates a solid foundation for branching out into separate sub-systems of the car and provides much potential for focused development of different areas. It is the hope of the author that a solid guide to the REV Lotus' systems has been provided, both for the present, and for the future.

7 Future Work

There are several future developments that could be undertaken to improve the current system:

- Development of a map server with routing engine for supplying navigation data to the REV Vehicles.
- Improve the graphics used in the Lotus user-interface and implement new classes for constructing user-interface elements.
- Development of touch screen hardware with embedded CAN controller.
- Development of a CAN-based motor controller system, that can respond to user requests on a touch screen.
- Development of CAN-based I/O board for interfacing vehicle sensors.
- Development of an Android based operating system specifically for car touch screen use.
- Development of an embedded engine sounds solution using a chip with on-board DSP and swappable SD cards containing car audio information.
- Evaluation of ICE audio emulation with respect to pedestrian safety.
- Further study and evaluation of battery estimation models.

- Development of hardware state of charge estimation solution for integration into Battery Management Systems.
- Development of a State of Health indicator for Battery Management Systems.

8 References

References

[1] "Can bus practical bus length," Softing, 2011. [Online]. Available: http://www.softing.com/home/en/industrial-automation/products/can-bus/more-can-bus/bit-timing/practical-bus-length.php?navanchor=3010538

[2] "The rev project," The REV Project, University of Western Australia, 2011. [Online]. Available: http://therevproject.com/

[3] "Wa electric vehicle trial," WA Electric Vehicle Trial, 2011. [Online]. Available: http://www.waevtrial.com.au/

[4] L. Schipper, H. Fabian, and J. Leather, "Transport and carbon dioxide emissions: Forecasts, options analysis, and evaluation," *Asian Development Bank*, 2009.

[5] E. E. (Firm) and C. C. L. of Congress), *Who Killed the Electric Car?* Sony Pictures Classics, 2006.

[6] D. Zubrinic, "History of croatian science," 1995. [Online]. Available: http://www.croatianhistory.net/etf/et22a1.html

[7] "Milestones in automotive history," AA1 Car Auto Diagnosis & Auto Repair Help, 2011. [Online]. Available: http://www.aa1car.com/library/timeline.htm

[8] "History of warner electric," Warner Electric, 2011. [Online]. Available: http://www.warnernet.com/history.asp

[9] "America on the move," National Museum of American History, 2011. [Online]. Available: http://americanhistory.si.edu/onthemove/themes/story_86_10.html

[10] "Cracking ferrari's engima code," Ferarri Online: Article appeared in Sports Car Market January 2011, 2011. [Online]. Available: http://www.ferraris-online.com/pages/article.php?reqart=SCM_201101_SS

[11] T. Boynton, “General motors computerized vehicle control systems: A short history,” 2011. [Online]. Available: http://tomboynton.com/GMnetworks.pdf

[12] “Eobd - a detailed history,” Omitech, 2011. [Online]. Available: http://www.omitec.com/en/support/technology-briefs/detailed-history-of-eobd/

[13] D. Varma, “Renewable energy vehicle instrumentation: Graphical user interface and black box,” School of Electrical and Electronic Engineering, 2009.

[14] T. Walter, “Development of a user interface for electric cars,” School of Electrical and Electronic Engineering, 2010.

[15] “Qt - a cross-platform application and ui framework,” Nokia, 2011. [Online]. Available: http://qt.nokia.com/products/

[16] J. Pearce, “Electric vehicle telemetry,” School of Electrical and Electronic Engineering, 2010.

[17] “Main roads wa on twitter,” Main Roads Western Australia, 2011. [Online]. Available: http://twitter.com/#!/perth_traffic

[18] R. Hanna, “Incidence of pedestrian and bicyclist crashes by hybrid electric passenger vehicles,” U.S. Department of Transport, Tech. Rep., 2009.

[19] L. Garay-Vega, J. Pollard, C. Guthy, and A. Hastings, “Auditory detectability of hybrid electric vehicles by pedestrians who are blind,” 2011.

[20] R. Wall Emerson, K. Naghshineh, J. Hapeman, and W. Wiener, “A pilot study of pedestrians with visual impairments detecting traffic gaps and surges containing hybrid vehicles,” *Transportation Research Part F: Traffic Psychology and Behaviour*, 2010.

[21] “H.r.734 - pedestrian safety enhancement act of 2009,” Open Congress, 2009. [Online]. Available: http://www.opencongress.org/bill/111-h734/show

[22] “Follow-up report: President obama signs pedestrian safety enhancement act into law,” CNET, 2009. [Online]. Available: http://reviews.cnet.com/8301-13746_7-20027830-48.html

[23] “Blind advocates disappointed in nissan e.v. sounds for pedestrians,” New York Times, 2010. [Online]. Available: http://wheels.blogs.nytimes.com/2010/06/17/blind-advocates-disappointed-in-nissan-e-v-sounds-for-pedestrians/

[24] “Electric vehicle noise pollution,” NoiseOff! The Coalition Against Noise Pollution, 2011. [Online]. Available: http://noiseoff.org/evs.php

[25] T. Tabata, H. Konet, and T. Kanuma, “Development of nissan approaching vehicle sound for pedestrians: How to solve the trade off between quietness and pedestrian safty of the electric vehicles?” 2011.

[26] “Halosonic noise management solutions,” Harman Automotive, 2011. [Online]. Available: http://www.halosonic.co.uk/

[27] “Ectunes adds sound to silent evs, but only where and when you need it,” Engadget, 2010. [Online]. Available: http://www.engadget.com/2010/09/09/ectunes-adds-sound-to-silent-evs-but-only-where-and-when-you-ne/

[28] C. Hellsten, “Ferrari on a stick - a system for emulating engine sounds,” School of Electrical and Electronic Engineering.

[29] K. Harper-Meredith, “Ears and rev,” School of Electrical and Electronic Engineering, 2010.

[30] F. Ho, “Drive system design for lotus elise electric car,” School of Electrical and Electronic Engineering, 2009.

[31] “Australian design rules,” Department of Infrastructure and Transport, 2011. [Online]. Available: http://www.infrastructure.gov.au/roads/motor/design/index.aspx

[32] *UQM Motor Systems User Manual*, UQM Technologies Inc., 2011.

[33] *CAN Communications Summary*, UQM Technologies Inc., 2011.

[34] “J1939 data link layer,” SAE, 2011. [Online]. Available: http://standards.sae.org/j1939/21_201012/

[35] “Introduction to sae j1939,” Kvaser Advanced CAN Solutions, 2011. [Online]. Available: http://www.kvaser.com/en/about-can/higher-layer-protocols/36.html

[36] “Controller area network,” Bosch, 2011. [Online]. Available: http://www.semiconductors.bosch.de/en/ipmodules/can/can.asp

[37] “Devicenet,” Open DeviceNet Vendors Association, 2011. [Online]. Available: http://www.odva.org/Home/ODVATECHNOLOGIES/DeviceNet/tabid/66/lng/en-US/language/en-US/Default.aspx

[38] “Canopen,” CAN in Automation, 2011. [Online]. Available: http://www.can-cia.org/index.php?id=canopen

[39] "Nmea 2000(r) edition 2.0," National Marine Electronics Association, 2011. [Online]. Available: http://www.nmea.org/content/nmea_standards/nmea_2000_ed2_20.asp

[40] "Leaf canbus decoding," MyNissanLeaf.com, 2011. [Online]. Available: http://www.mynissanleaf.com/viewtopic.php?f=44&t=4131

[41] "Can protocol specification," CAN in Automation, 2011. [Online]. Available: http://www.can-cia.de/index.php?id=164

[42] *Lotus Elise Service Manual 2001 Model Year Onward*, Lotus Cars LTD., 2001.

[43] D. Kingdom, "2009 rev management and on-board embedded systems," School of Electrical and Electronic Engineering, 2009.

[44] C. Watts, "Electrical designs for the renewable energy vehicles within the rev project," School of Electrical and Electronic Engineering, 2009.

[45] *AVR-CAN Development Board Users Manual*, Olimex Ltd., 2011.

[46] A. Padhi, K. Nanjundaswamy, and J. Goodenough, "Phospho-olivines as positive-electrode materials for rechargeable lithium batteries," *Journal of the Electrochemical Society*, vol. 144, p. 1188, 1997.

[47] "Tesla model s: The battery pack," CNET, 2010. [Online]. Available: http://reviews.cnet.com/8301-13746_7-20018836-48.html

[48] "Panasonic launches type 18650 li-ion batteries with a peak capacity of 3.1 a," Panasonic, 2010. [Online]. Available: https://industrial.panasonic.com/eu/news/nr201005IE002/nr201005IE002/Press_Release_Li-Ion_NNP_E.pdf

[49] G. Plett, "Extended kalman filtering for battery management systems of lipb-based hev battery packs:: Part 1. background," *Journal of Power sources*, vol. 134, no. 2, pp. 252–261, 2004.

[50] ——, "Extended kalman filtering for battery management systems of lipb-based hev battery packs:: Part 2. modeling and identification," *Journal of power sources*, vol. 134, no. 2, pp. 262–276, 2004.

[51] ——, "Extended kalman filtering for battery management systems of lipb-based hev battery packs-part 3. state and parameter estimation," *Journal of Power Sources*, vol. 134, no. 2, pp. 277–292, 2004.

[52] K. Ng, C. Moo, Y. Chen, and Y. Hsieh, "Enhanced coulomb counting method for estimating state-of-charge and state-of-health of lithium-ion batteries," *Applied energy*, vol. 86, no. 9, pp. 1506–1511, 2009.

[53] J. Wang, B. Cao, Q. Chen, and F. Wang, "Combined state of charge estimator for electric vehicle battery pack," *Control Engineering Practice*, vol. 15, no. 12, pp. 1569–1576, 2007.

[54] G. Sikha, R. White, and B. Popov, "A mathematical model for a lithium-ion battery/electrochemical capacitor hybrid system," *Journal of the Electrochemical Society*, vol. 152, p. A1682, 2005.

[55] J. Wang, L. Xu, J. Guo, and L. Ding, "Modelling of a battery pack for electric vehicles using a stochastic fuzzy neural network," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 223, no. 1, pp. 27–35, 2009.

[56] C. Bo, B. Zhifeng, and C. Binggang, "State of charge estimation based on evolutionary neural network," *Energy conversion and management*, vol. 49, no. 10, pp. 2788–2794, 2008.

[57] N. Cui, C. Zhang, Q. Kong, and Q. Shi, "A combined method of battery soc estimation for electric vehicles," in *Industrial Electronics and Applications (ICIEA), 2010 the 5th IEEE Conference on*. IEEE, pp. 1147–1151.

[58] S. Qingsheng, Z. Chenghui, C. Naxin, and Z. Xiaoping, "Battery state-of-charge estimation in electric vehicle using elman neural network method," in *Control Conference (CCC), 2010 29th Chinese*. IEEE, pp. 5999–6003.

[59] M. Charkhgard and M. Farrokhi, "State-of-charge estimation for lithium-ion batteries using neural networks and ekf," *Industrial Electronics, IEEE Transactions on*, vol. 57, no. 12, pp. 4178–4187, 2010.

[60] K. Hornik Maxwell and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[61] "How many hidden units should i use?" comp.ai.neural-nets, 2011. [Online]. Available: http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html

[62] N. Wanas, G. Auda, M. Kamel, and F. Karray, "On the optimal number of hidden nodes in a neural network," in *Electrical and Computer Engineering, 1998. IEEE Canadian Conference on*, vol. 2. IEEE, 1998, pp. 918–921.

[63] R. Kalman *et al.*, "A new approach to linear filtering and prediction problems," *Journal of basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960.

[64] Y. Zhou, J. Sun, and X. Wang, “Power battery charging state-of-charge prediction based on genetic neural network,” in *Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference on*. IEEE, pp. 1–4.

[65] Y. Lee, W. Wang, and T. Kuo, “Soft computing for battery state-of-charge (bsoc) estimation in battery string systems,” *Industrial Electronics, IEEE Transactions on*, vol. 55, no. 1, pp. 229–239, 2008.

[66] A. Salkind, C. Fennie, P. Singh, T. Atwater, and D. Reisner, “Determination of state-of-charge and state-of-health of batteries by fuzzy logic methodology,” *Journal of Power Sources*, vol. 80, no. 1-2, pp. 293–300, 1999.

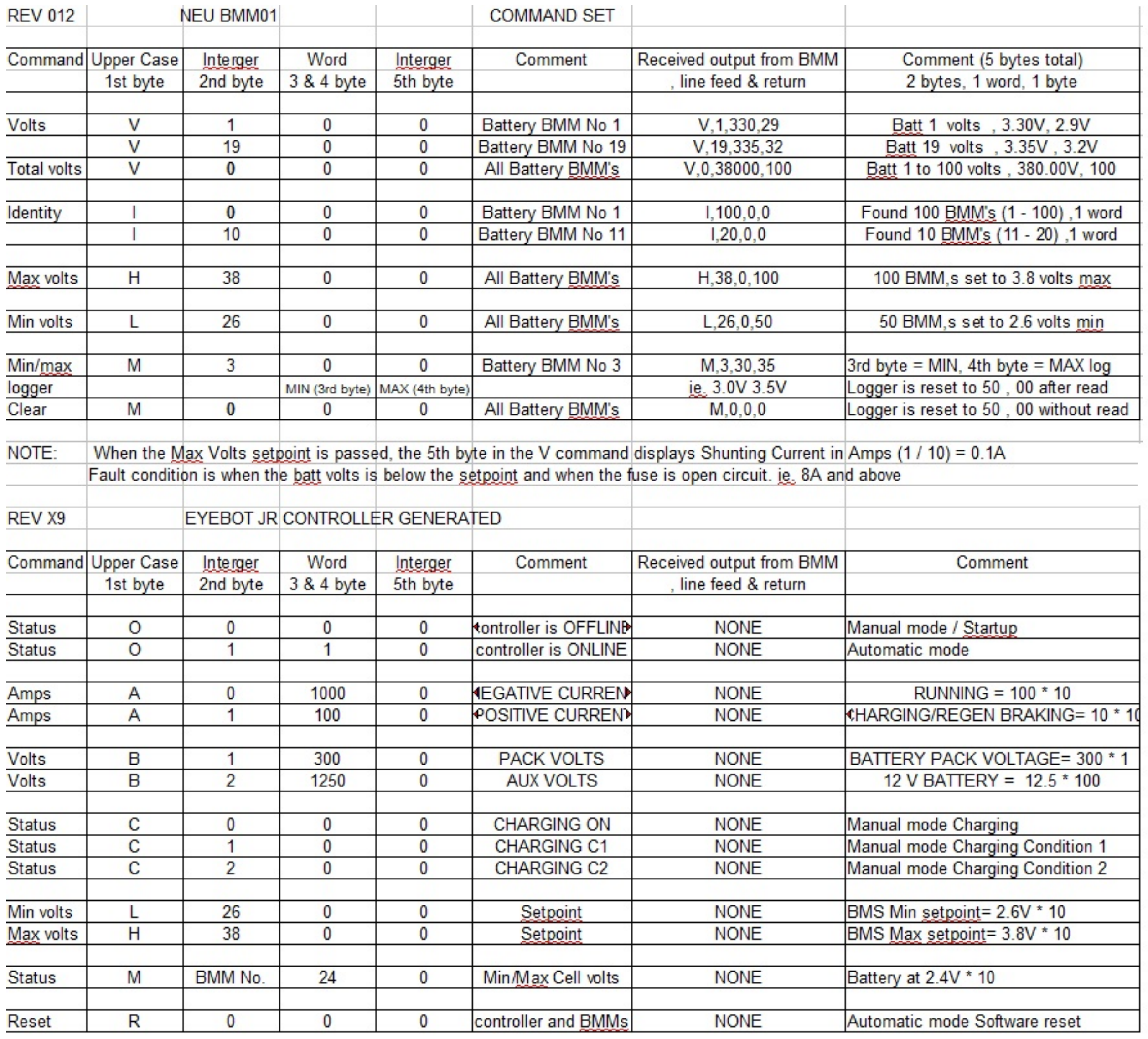
[67] K. Chau, K. Wu, and C. Chan, “A new battery capacity indicator for lithium-ion battery powered electric vehicles using adaptive neuro-fuzzy inference system,” *Energy conversion and management*, vol. 45, no. 11-12, pp. 1681–1692, 2004.

[68] “Pybrain: The python machine learning library,” PyBrain, 2011. [Online]. Available: http://wwww.pybrain.org/

[69] “Numpy,” NumPy, 2011. [Online]. Available: http://numpy.scipy.org/

A Appendix

A.1 BMS Protocol - Source: Ivan Neubronner



REV 012 NEU BMM01 COMMAND SET

Command	Upper Case 1st byte	Interger 2nd byte	Word 3 & 4 byte	Interger 5th byte	Comment	Received output from BMM , line feed & return	Comment (5 bytes total) 2 bytes, 1 word, 1 byte
Volts	V	1	0	0	Battery BMM No 1	V,1,330,29	Batt 1 volts , 3.30V, 2.9V
	V	19	0	0	Battery BMM No 19	V,19,335,32	Batt 19 volts , 3.35V , 3.2V
Total volts	V	**0**	0	0	All Battery BMM's	V,0,38000,100	Batt 1 to 100 volts , 380.00V, 100
Identity	I	**0**	0	0	Battery BMM No 1	I,100,0,0	Found 100 BMM's (1 - 100) ,1 word
	I	10	0	0	Battery BMM No 11	I,20,0,0	Found 10 BMM's (11 - 20) ,1 word
Max volts	H	38	0	0	All Battery BMM's	H,38,0,100	100 BMM,s set to 3.8 volts max
Min volts	L	26	0	0	All Battery BMM's	L,26,0,50	50 BMM,s set to 2.6 volts min
Min/max	M	3	0	0	Battery BMM No 3	M,3,30,35	3rd byte = MIN, 4th byte = MAX log
logger			MIN (3rd byte)	MAX (4th byte)		ie. 3.0V 3.5V	Logger is reset to 50 , 00 after read
Clear	M	**0**	0	0	All Battery BMM's	M,0,0,0	Logger is reset to 50 , 00 without read

NOTE: When the Max Volts setpoint is passed, the 5th byte in the V command displays Shunting Current in Amps (1 / 10) = 0.1A
Fault condition is when the batt volts is below the setpoint and when the fuse is open circuit. ie. 8A and above

REV X9 EYEBOT JR CONTROLLER GENERATED

Command	Upper Case 1st byte	Interger 2nd byte	Word 3 & 4 byte	Interger 5th byte	Comment	Received output from BMM , line feed & return	Comment
Status	O	0	0	0	ontroller is OFFLIN	NONE	Manual mode / Startup
Status	O	1	1	0	controller is ONLINE	NONE	Automatic mode
Amps	A	0	1000	0	EGATIVE CURREN	NONE	RUNNING = 100 * 10
Amps	A	1	100	0	OSITIVE CURREN	NONE	HARGING/REGEN BRAKING= 10 * 1
Volts	B	1	300	0	PACK VOLTS	NONE	BATTERY PACK VOLTAGE= 300 * 1
Volts	B	2	1250	0	AUX VOLTS	NONE	12 V BATTERY = 12.5 * 100
Status	C	0	0	0	CHARGING ON	NONE	Manual mode Charging
Status	C	1	0	0	CHARGING C1	NONE	Manual mode Charging Condition 1
Status	C	2	0	0	CHARGING C2	NONE	Manual mode Charging Condition 2
Min volts	L	26	0	0	Setpoint	NONE	BMS Min setpoint= 2.6V * 10
Max volts	H	38	0	0	Setpoint	NONE	BMS Max setpoint= 3.8V * 10
Status	M	BMM No.	24	0	Min/Max Cell volts	NONE	Battery at 2.4V * 10
Reset	R	0	0	0	controller and BMMs	NONE	Automatic mode Software reset

A.2 GPS Protocol - Source: NMEA

NMEA 0183 is the GPS protocol most widely used and consists of a set of messages.

$GPRMC,220516,A,5133.82,N,00042.24,W,173.8,231.8,130694,004.2,W,*70

Purpose: Recommended minimum specific GPS/Transit data

$ GPMRC	Identifier
220516	Time Stamp (UTC hhmmss)
A	Validity - A-ok, V-invalid
5133.82	Current Latitude
N	North/South
00042.24	Current Longitude
W	East/West
173.8	Speed in Knots
231.8	True Course
130694	Date Stamp (ddmmyy)
004.2	Variation
W	East/West
*70	Checksum

$GPGGA,170834,4124.89,N,8151.68,W,1,05,1.5,280.2,M,-34,M,1,2,*75

Purpose: GPS System Fix Data

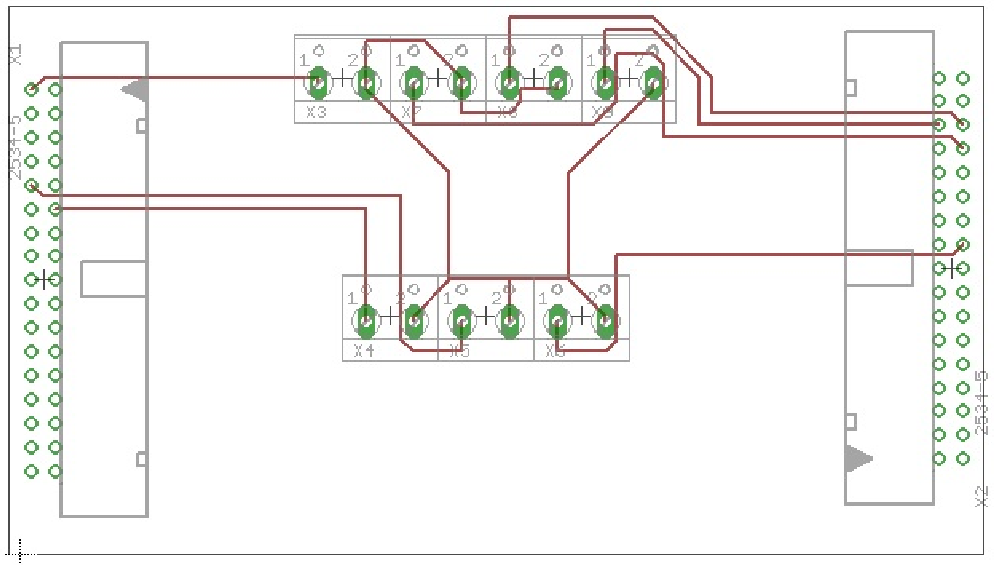
$GPGGA	Identifier
170834	Time Stamp (UTC hhmmss)
4124.89	Latitude
N	North/South
8151.68	Longitude
W	East/West
1	GPS Quality Index (0 = Invalid, 1 = GPD Fix)
05	Number of Satellites
1.5	Horizontal Dilution of Precision
280.2	Antenna Altitude Above Sea Level
M	Units of Antenna Altitude
-34	Geoidal Separation
M	Units of Geoidal Separation
1	Age of Differential GPS data (seconds)
2	Station Reference ID
*70	Checksum

A.3 Telemetry Protocol - Source: John Pearce

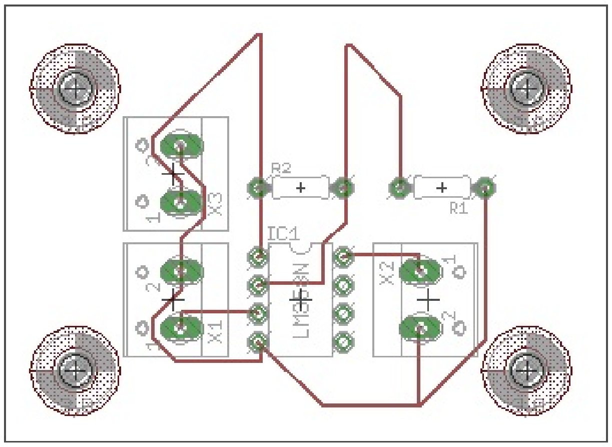
Detailed here is the Telemetry Protocol as designed by John Pearce. Minimum length of message is 33 bytes.

Field	Bytes	Comments
Protocol ID	1	’R’ for REV Vehicles
Packet Length	2	Length of whole message
IMEI	7	3G/GPRS Device IMEI
Latitude	4	
Longitude	4	
GPS Time	4	Julian Time
Speed	1	km/h
Heading	1	Heading divided by 2
Altitude	1	Metres divided by 20
Reason Code	2	Always 0 (time interval) for Lotus
Status Code	1	Always 0 for Lotus
DI Count	1	Number of Digital Inputs (Current 0 for Lotus
DI	1*(DI Count)/8 + 1	8 Digital Inputs Per Byte
ADC Count	2	Number of Analogue Inputs
ADC	2*ADC Count	Each Analogue Input is Scaled to 2 Bytes
Battery Level	1	
Checksum	2	Modbus CRC-16

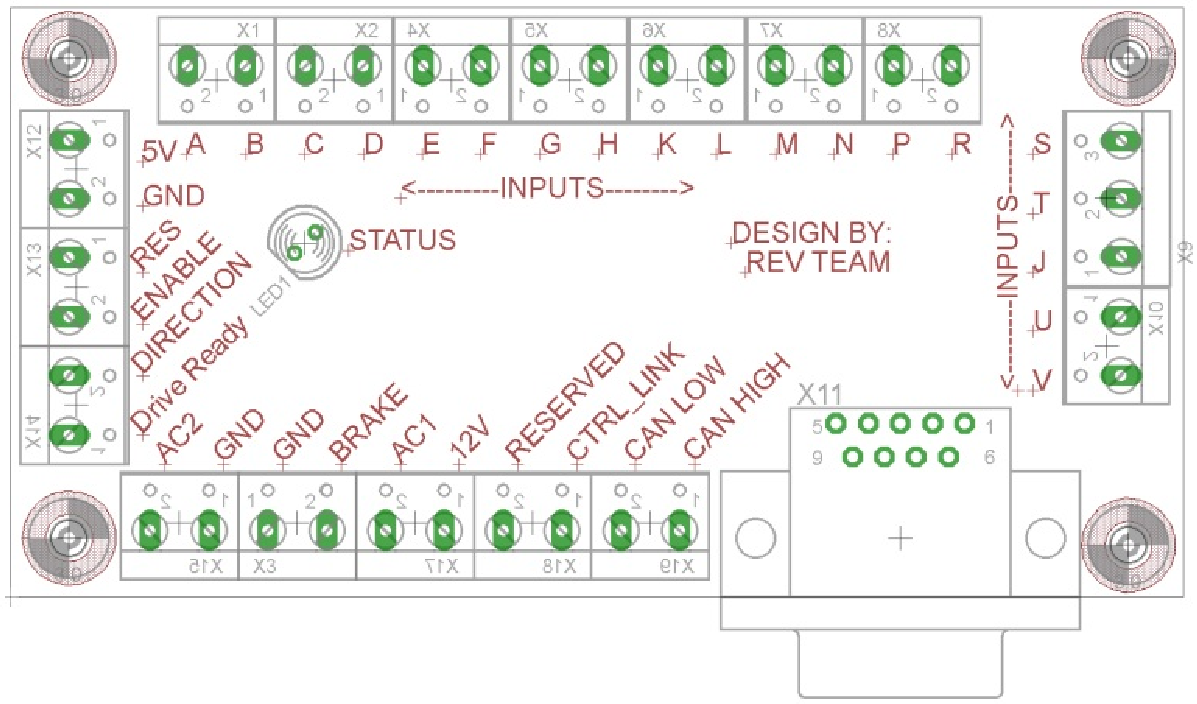
A.4 CAN-SPY Breakout Board



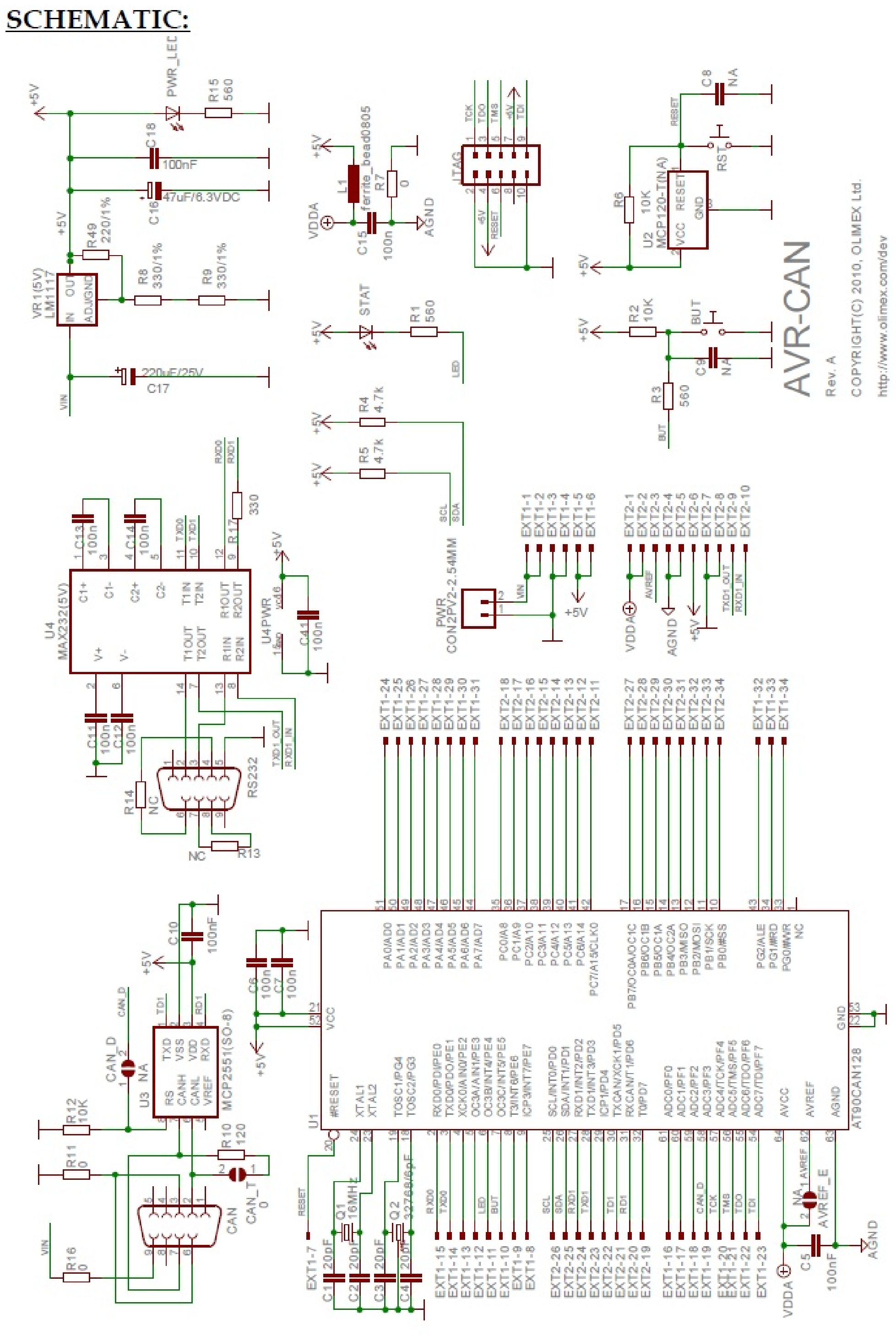
A.5 Tachometer Amplifier Board



A.6 Motor Controller Breakout Board - Source: Watts



A.7 AVR-CAN Schematic - Source: Olimex



A.8 Computer Listings

- AVR-CAN Datasheet
- AT90CAN128 Datasheet
- Lotus 2001 Service Manuals
- UQM Powerphase CAN Communication Summary
- UQM Powerphase 75 Manual
- Breakout Board Eagle Files
- Amplifier Eagle Files
- Source Code for AVR-CAN-SPY
- Source Code for Lotus REV
- Source Code for Engine Sounds
- Source Code and other files for Neural Network Estimation