I hereby declare that this work has not been submitted for any other degree/course at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original and entirely the result of my own work at the University of Strathclyde under the supervision of Dr Bruce Stephen.

Implementation of an Automotive Battery Health Management and Diagnostic System

Taylor Phillips - 201911043

Supervisor: Dr Bruce Stephen

Date: 31/03/2023

Abstract

This project lays the foundation of the software implementation of a Battery Management System (BMS) for the University of Strathclyde Motorsport (USM) team's high voltage battery in an electric racecar for competing at the international Formula Student (FS) competition. It outlines and goes into detail the processes involved in implementing the embedded software, interfacing with the devices in the battery, monitoring and controlling the battery state, balancing cells and more. How the graphical user interface was developed for configuration of the battery management system and comprehensive testing of the systems implemented is also discussed in detail. Furthermore a model of the battery was developed to test State of Charge (SOC) estimation algorithms and verify the Hybrid Pulse Power Characterisation (HPPC) process. The two systems: the embedded system and the companion utility Graphical User Interface (GUI), both work together to create a configurable battery management system with comprehensive monitoring and testing capabilities.

Contents

List of Abbreviations

BMS Battery Management System

USM University of Strathclyde Motorsport

FS Formula Student

SOC State of Charge

isoSPI Isolated Serial Peripheral Interface

CAN Controller Area Network

PWM Pulse Width Management

IC Integrated Circuit

GPIO General Purpose Input/Output

MUX Multiplexer

ADC Analogue to Digital Converter

GLSL OpenGL Shading Language

CPU Central Processing Unit

GPU Graphics Processing Unit

EV Electric Vehicle

RCB Relay Control Board

HV High Voltage

OCV Open Circuit Voltage

EKF Extended Kalman Filter

HPPC Hybrid Pulse Power Characterisation

ECU Electronic Control Unit

HTML Hyper-Text Markup Language

MISO Master-In Slave-Out

GUI Graphical User Interface

1 Introduction

With the world becoming increasingly dependant on energy storage with movements towards sustainability, energy systems are becoming more dependant on batteries. These batteries are increasing in power and capacity and the need for managing these systems are becoming more pertinent which is where a BMS comes in. These devices monitor cell temperatures, voltages and currents to keep track of the battery's SOC and make sure it is operating safely. It watches for any faults to occur during operation and acts accordingly, from warning the user of problems to shutting the system down.

This project focuses on the implementation of a BMS in the context of an electric vehicle with a High Voltage (HV) battery and a human driver, thus safety is a huge priority. On top of just monitoring the battery's state it must be able to shut the battery down if required.

In the previous year the USM team acquired an off-the-shelf BMS unit called the 'Orion 2'. This unit was sufficient for the team's first Electric Vehicle (EV) battery to reduce the complexity and amount of work required to implement the design and reducing the number of unknowns. This unit however had many limitations and drawbacks, some of which were clear before purchase and some discovered through its testing and implementation. A new custom BMS overcomes many drawbacks and limitations of the 'Orion 2' identified below:

Reduction in Size & Weight - The 'Orion 2' is a bulky device weighing over 2 kg. Size and weight are crucial to the success of a competitive racecar where it needs to be lightweight & aerodynamic. A custom system can be designed purposefully to match the needs and specifications of our battery rather than being a generic option with redundant features.

Open-ended Design - A custom system allows for constant innovation and development with plenty of scope for future work. Continuing to use the off-the-shelf unit requires using its limited and restrictive interface for configuration which has no room for development.

Interconnectivity - It is required to interface with the 'Orion 2' according to its design and specification therefore you cannot expand its interface to add interconnectivity between additional devices i.e the Relay Control Board (RCB). This allows for a more intelligent and interconnected system within the battery as well as in the car.

True HV Isolation - If each battery segment has been isolated from each other, they are not completely isolated internally inside the 'Orion 2' when the cell tap wiring harness is connected. This is potentially a danger to anybody operating on the battery as two connected segments could reach 120 V DC. The new system ensures segments are totally isolated when separated.

Cheaper Implementation - The 'Orion 2' is an expensive device with a price tag of $1,3445. A custom BMS is a significantly cheaper alternative and provides the groundwork for further innovation within the team, reducing the cost of future implementations as well as the barrier to entry for designing a new BMS in-house for a new battery's specifications.

The project's goal is to implement the software for a BMS that replaces the previous unit for the reasons listed above, whilst also adhering to rules specified by the FS body so the vehicle can compete. It must also be designed in a flexible and configurable way so the USM team can use it for future batteries.

2 System Overview

This BMS is a part of the USM team's electric racecar battery. The BMS is one of many systems within the battery let alone the car that all have to integrate together to produce a fully functioning and safe vehicle.

The BMS's main function is to monitor the voltage, current and temperature of the battery to ensure safe operation. However, it has other responsibilities due to it needing to integrate with other systems as well as needing to adhere to rules specified by the FS body.

Other systems the BMS must integrate with is the charger, the relay control board, the fans within the battery, the Electronic Control Unit (ECU) and the shutdown circuit. A diagram of the devices within the battery that the BMS has to interface with is shown in Figure 1.

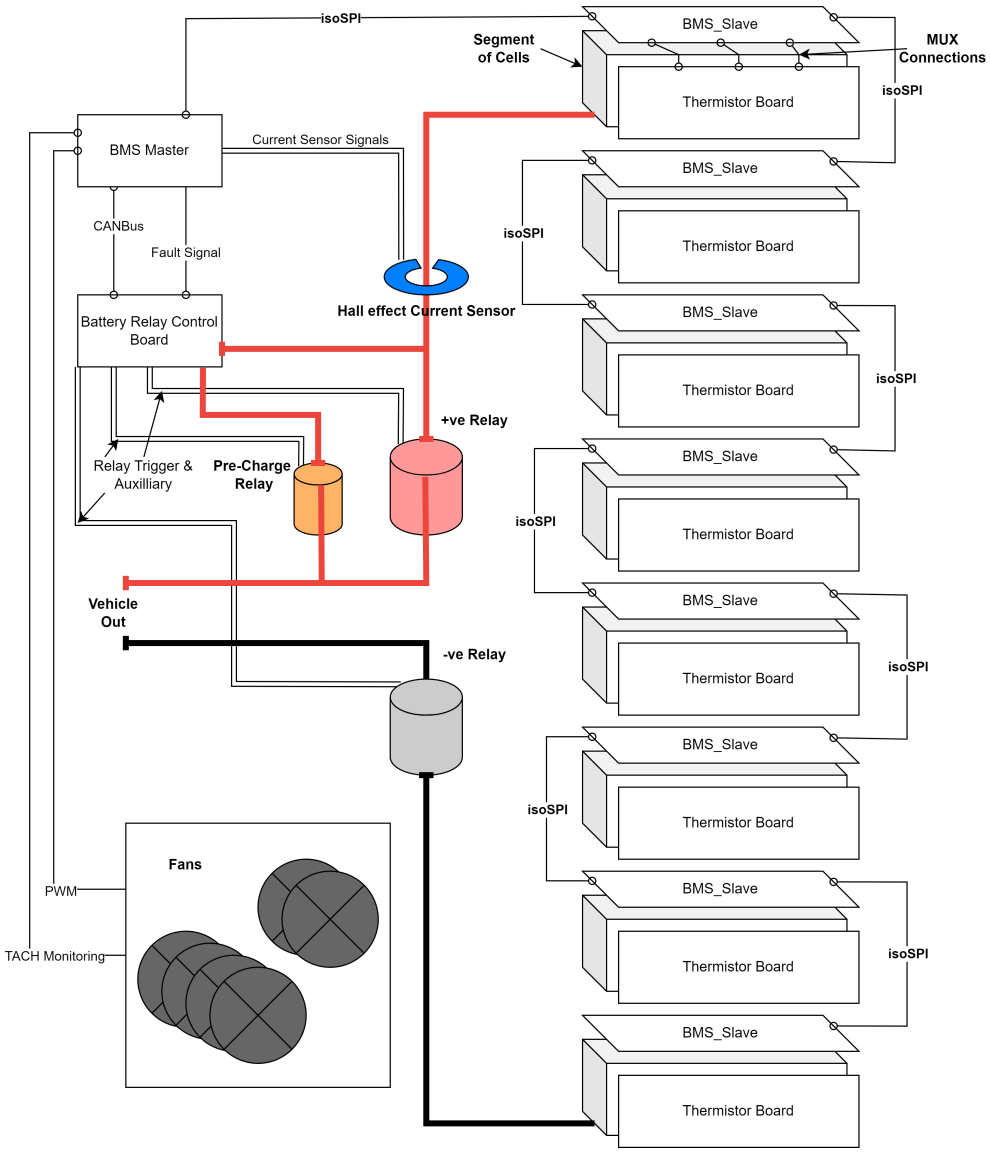

Figure 1: System Overview.

2.1 System Requirements

Here the system requirements for both the embedded system and the companion utility for the BMS will be laid out as a clear indication of all the systems and processes required to implement the BMS within the USM vehicle.

Embedded BMS

- Communicate with 16 LTC6840-1 BMS chips over Isolated Serial Peripheral Interface (isoSPI).
- Gather 112 voltages and 176 temperatures from BMS chips at least every 1.0 s.
- Gather pack current data from hall-effect current sensor (LEM DHAB S/134 [1]).
- Predict SOC from current and open-circuit voltage data.
- Detect faults within the battery and make decisions regarding safety and shutdowns.
- Control fan Pulse Width Management (PWM) and read fan tachometers.
- Communicate with charger unit to enable charging at a set voltage and current.
- Communicate with ECU to broadcast to the car that battery is operational or not.
- Integrate with shutdown circuit to shutdown battery if fault occurs.
- Be configurable via companion utility to enable testing in different scenarios.
- Transmit and receive information on the vehicle's CAN Bus.
- Output all relevant information onto CAN Bus for companion utility to interpret.
- If unexpected error occurs, battery should fault safely and shutdown.

Companion Utility

- Built-in serial monitor for receiving serial messages via CAN adapter.
- CAN bus monitor for parsing in serial messages to be relayed to necessary functions.
- Displaying of all necessary information for validation and testing.
- Display temperatures and voltages in an easy to interpret at-a-glance manner.
- Configuration and storage of various parameters for the embedded BMS.
- Retrieval & flashing of configuration profile of BMS.
- Storage of a documentation viewer for all systems in the utility.
- Configuration and storage of the settings for the utility itself.
- Log entire CAN bus to file for analysis of runtime.
- Capability of plotting any battery parameter over time.
- Set or reset balancing for individual cells for testing.
- Run on light-weight systems while maintaining performance and usability.

3 Embedded System Software Design

3.1 Overview

There are two parts to the hardware interface for the system's design: the Master and Slave. The Master being responsible for current monitoring, CAN Bus interface, fan tachometer monitoring and communicating with the Slaves over isoSPI. The slave is responsible for monitoring cell voltages, cell temperatures, open-wiring faults and balancing cell stacks while transmitting this information back to the Master for handling. The Master board is controlled by an STM32F446RE which is programmed with code written in the STM32CubeIDE and the slave boards are controlled by two LTC6804-1 BMS Integrated Circuit (IC)s which receive commands from the STM32 via isoSPI The BMS master is shown in Figure 2.

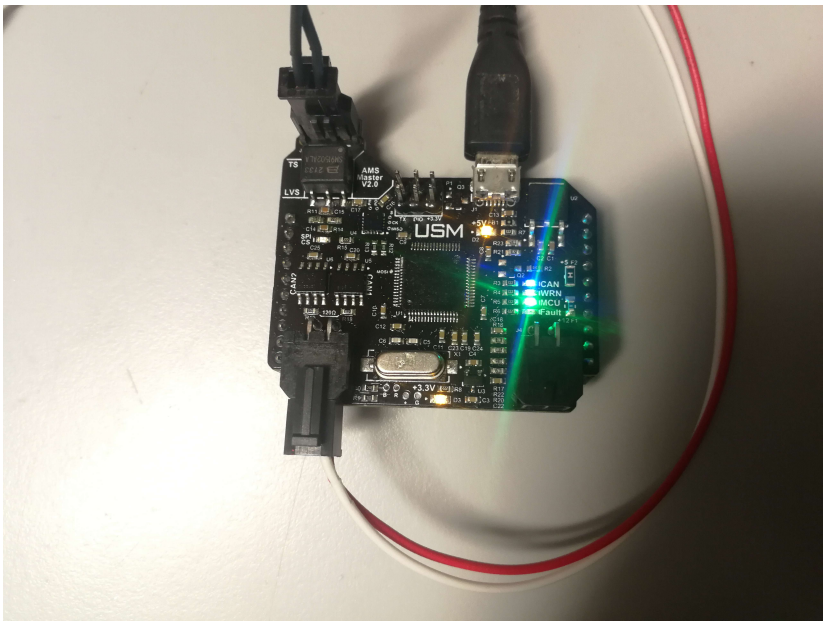

Figure 2: BMS Master PCB with isoSPI, CAN and USB.

3.2 Software Implementation

3.2.1 Communication with BMS ICs

The Master and all BMS ICs (LTC6804-1) communicate over an isoSPI daisy chain with various commands that can either work as a broadcast message that all ICs receive simultaneously or as a daisy chained message that propagates through the daisy chain, shifting all requested data to or from the Master.

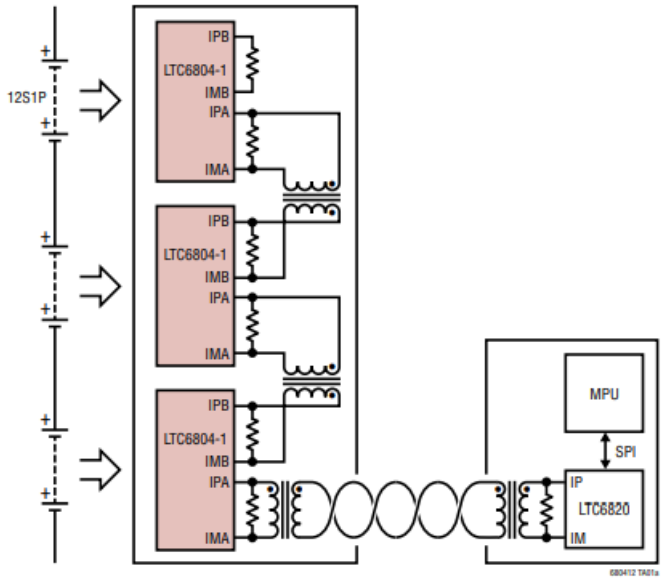

*Figure 3: Daisy Chaining Setup of LTC6804-1 Chips [**?**].*

'Analogue Devices' provide Arduino libraries for interfacing with the LTC6804-1 chips and can be found on Github [2]. Arduino libraries use a different version of C as well as as well as interface with their peripherals differently compared to STM32 libraries so they had to be rewritten and ported to be compatible with the STM32F446RE. The differing version of C proved to be less of a problem than initially expected with the only difference being utilised was the primitive data type 'bool' or boolean which is not a primitive data type in the version of C the project is written in. This was simple to remedy by defining a macro which replaces the words 'True' and 'False' with '1' and '0' respectively. Remedying the difference in interfacing required significantly more work and testing to get working as intended with the biggest change in peripheral interface between Arduino and STM32 is isoSPI communications.

The libraries were first verified to work by using the original Arduino libraries on an Arduino Uno with an isoSPI shield on top connected to an LTC6804-1 development board. This worked out of the box and didn't require any troubleshooting. The next logical step was to swap out the Arduino for an STM32 'bluebill' while keeping the Arduino isoSPI shield which was already verified to work. With some trial and error as well as research into implementing isoSPI communications on STM32s, a connection was established with the LTC6804-1 development board and voltages were successfully retrieved from the device, verifying the code works as well as the development board and Arduino shield with the setup shown in Figure 4.

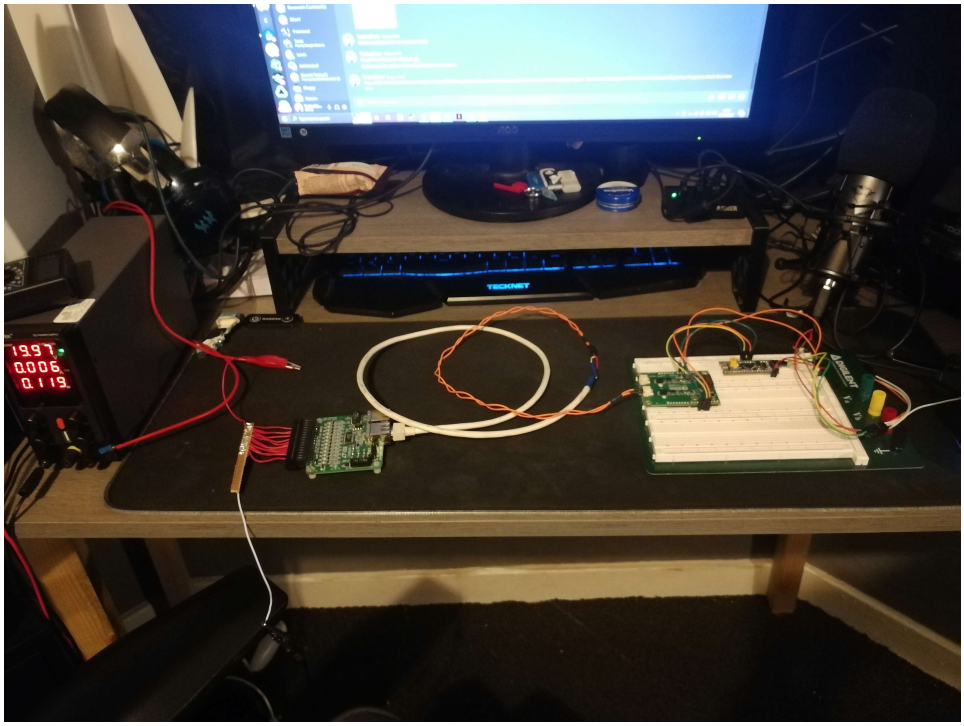

Figure 4: isoSPI Communications with STM32 & Arduino Shield Setup.

Then the master board had to be verified to work as the Arduino shield is only a means for testing other components as a control. At this point the code for isoSPI communications have been verified to work as well as the development board however, the BMS master board's isoSPI interface has not been verified with the removal of the shield. This part of development took longer than anticipated with the complication arising of isoSPI communications not working. After many days of testing, troubleshooting and comparing the differences between the hardware of the previous setup and the new setup, there was a very simple fix: the isoSPI shield had a pull-up resistor on the isoSPI Master-In Slave-Out (MISO) pin with the fix being to enable a pull-up resistor on that pin with software on the STM32. The working setup with the development board and BMS master is shown in Figure 5.

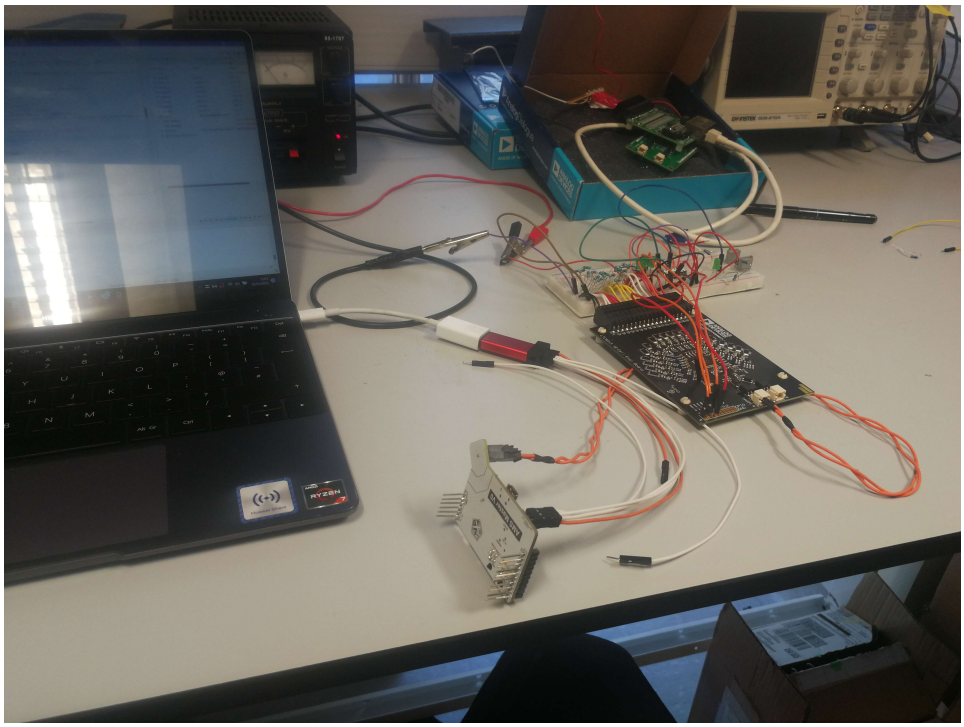

Figure 5: SPI Communications with STM32 & Development Board Setup.

Finally, the BMS slave board had to be verified to work as the LTC6804-1 development board was just for testing. The slave replaces it and will be used in the USM vehicle battery.

The LTC6804-1 BMS chips have configuration registers than can be controlled with isoSPI via the BMS master board. These registers are outline below in Table 3.1.

Table 3.1: LTC6804-1 Configuration Registers [3].

REGISTER	RD/WR	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
CFGR0	RD/WR	GPIO5	GPIO4	GPIO3	GPIO2	GPIO1	REFON	SWTRD	ADCOPT
CFGR1	RD/WR	VUV[7]	VUV[6]	VUV[5]	VUV[4]	VUV[3]	VUV[2]	VUV[1]	VUV[0]
CFGR2	RD/WR	VOV[3]	VOV[2]	VOV[1]	VOV[0]	VUV[11]	VUV[10]	VUV[9]	VUV[8]
CFGR3	RD/WR	VOV[11]	VOV[10]	VOV[9]	VOV[8]	VOV[7]	VOV[6]	VOV[5]	VOV[4]
CFGR4	RD/WR	DCC8	DCC7	DCC6	DCC5	DCC4	DCC3	DCC2	DCC1
CFGR5	RD/WR	DCTO[3]	DCTO[2]	DCTO[1]	DCTO[0]	DCC12	DCC11	DCC10	DCC9

Changing the bits in these registers allow for controlling the states of General Purpose Input/Output (GPIO) pins as well as cell balancing states. There are also various commands that can be sent to perform actions such as starting Analogue to Digital Converter (ADC) conversions.

3.2.2 Voltage Monitoring

The BMS chips connect to every cell stack terminal with one IC connecting to the first seven cells and the other IC connecting to the last seven cells. Each chip can only support twelve cells therefore two must be used as fourteen cells are needed.

To monitor the voltage of cell stacks, the master must tell the BMS ICs to begin the ADC conversion for cell readings. This is a broadcast message and thus all slaves will begin ADC cell conversions. ADC conversions are not instantaneous and therefore the master must wait before it can request the result. If the master requests the cell conversion results too early it will transmit the previous result causing duplicates of data and not give an accurate representation of the battery's voltages. There is no indication message for conversions finishing so the request for the cell conversion results are scheduled after a delay of $10ms$ which is significantly longer than the $1.1ms$ specified in the chip's data sheet.

3.2.3 Temperature Monitoring

To monitor the temperature, it is a more involved process than cell voltage monitoring as there is the extra component of multiplexing. Each slave monitors 21 temperatures with three multiplexers, each connected to seven thermistors. With each slave having two ICs, the lower IC is connected to two multiplexers, and the upper connected to one. Before starting a GPIO ADC conversion on the Multiplexer (MUX)'s outputs, it must first set GPIOs to select the desired thermistor from the MUX. This message must be daisy-chained through all BMS ICs, setting each GPIO pin to the desired selection. To keep consistency, all BMS ICs will be set to the same selection for multiplexing so they all receive the same GPIO configuration, making it simpler to read and sort the data coming in. Similar to how ADC conversions are not instantaneous and take some time, there must be a delay between receiving the message and changing the MUX selection, where again there is no indication that this process has completed. There is however a way to determine if the GPIO selection was correct at the time of ADC conversion completion; it can read in the GPIO states and compare them to the expected selection for the current multiplexer index, e.g. if selecting MUX #5, the selection should be 101. Therefore when reading in thermistor data, it must not only perform an ADC conversion on the MUX outputs GPIO, but also the MUX select GPIO. This means after every temperature reading it can detect if the correct thermistor was selected, if not a fault is thrown.

3.2.4 Current Monitoring

The system uses a hall-effect current sensor mounted around the battery's HV lines to monitor the current of the battery. The current sensor used in the system is the LEM DHAB S/134 [1] which has two separate channels with differing sensitivities. Each channel is more accurate than the other within their respective ranges, with Channel-1 being from $-50A$ to $+50A$ and Channel-2 being from $-200A$ to $+200A$. The two channels provide a voltage between $0V$ and $+5V$ with $0V$ at full-swing negative, $+5V$ at full-swing positive and $+2.5V$ at $0A$. The equation used to extract the current from both channels from their voltage measured with an ADC is shown below:

$$I = (\frac{5}{U} \times V_{out}) \times \frac{1}{G} \tag{1}$$

From the above equation U represents the measured supply voltage to the current sensor which is factored in to reduce errors in measurement if the supply is not exactly $+5V$. V_{vout} is the measured voltage from the channel and G is the sensitivity of the channel with a value of $40mV/A$ used for Channel-1 and a value of $10mV/A$ used for Channel-2. If the current is between $-50A$ and $+50A$, Channel-1 should be used with $G = 10$, otherwise if the current is between $-200A$ and $+200A$, Channel-2 should be used with $G = 40$. With a rated frequency of $50Hz$ it can achieve a sampling period of $20ms$.

3.2.5 Current Derate

The maximum current rating on cells is only valid if they are being used at their nominal operating conditions. As a cell's operating condition approaches an extreme, it is necessary to derate the maximum allowable current to prevent drawing more current than is safe. This maximum current will be broadcast to the ECU where it can then modulate the torque request to the motor-controller so that it does not exceed that current. This modulation means as the battery's temperature approaches its bounds, the torque request will be limited by the derate which in turn will also reduce the thermal output of the battery. Reducing the amount of heat the battery generates as the temperature increases means the driver won't have to worry about a thermal shutdown based on current draw and will instead simply be forced to drive slower with a reduced torque request.

Thermal Derate The parameters for the thermal derate are provided through the Configuration Utility with separate parameters for charging and discharging shown in Figure 6 and Figure 7 respectively.

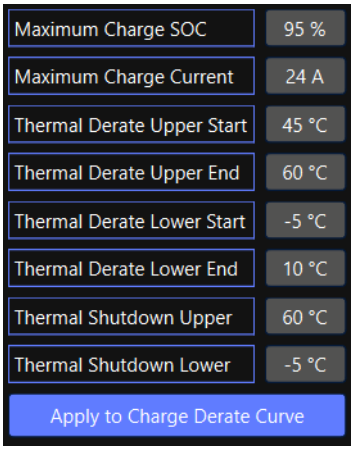

Figure 6: Charging Thermal Derate Settings.

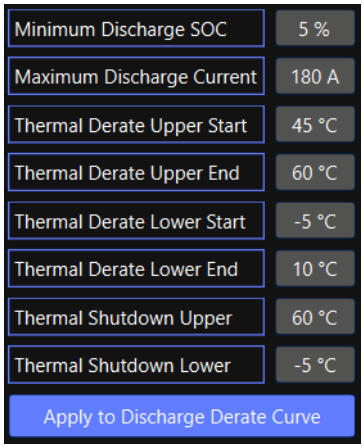

Figure 7: Discharging Thermal Derate Settings.

Using a linear derate with the above charging parameters the following derate plot is generated shown in Figure 8.

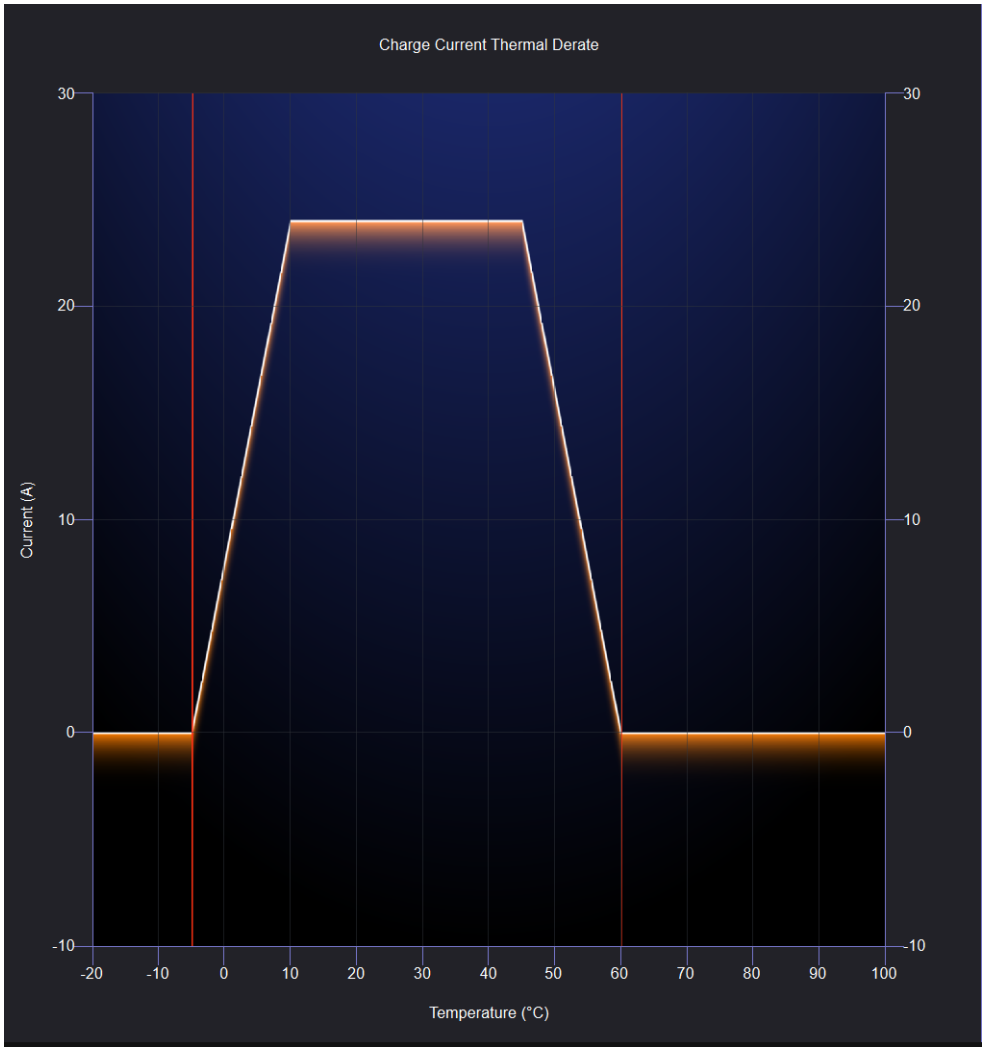

Figure 8: Linear Charging Current Thermal Derate Plot.

A linear derate however may not be the desired approach depending on the scenario and the battery involved. As such an Easing parameter has been added to allow for a derate curve instead of a derate line shown in Figure 9. This curve is created by an Ease-Out Exponential function which is defined below using a normalised input between 0 & 1:

$$A = (1 - 2^{-10x}) \times A_{max} \tag{2}$$

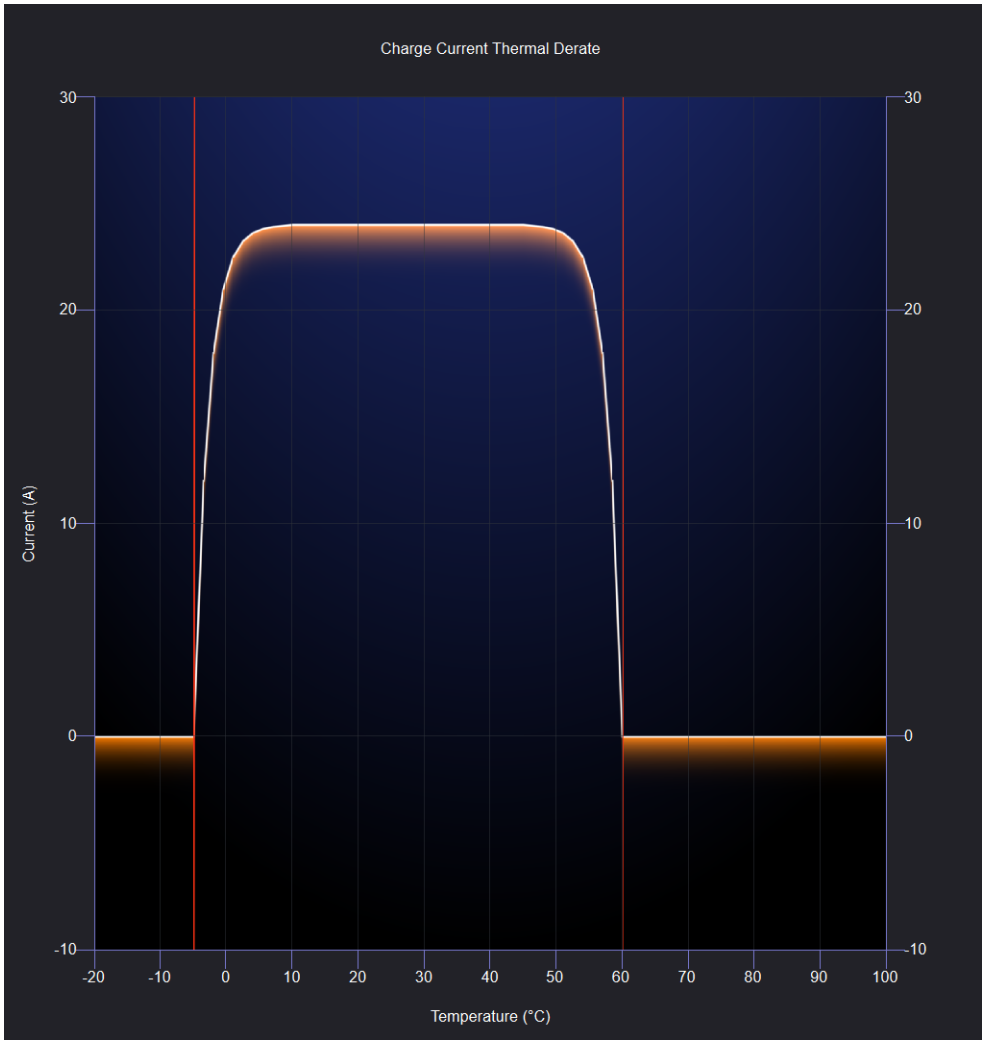

Figure 9: Ease-Out Exponential Charging Current Thermal Derate Plot.

SOC Derate Another important condition of the battery that affects its maximum current draw is its SOC.

3.2.6 State of Charge Estimation

Estimating the SOC of the battery accurately is important for making safety decisions and detecting faults as the battery is running. During runtime the voltage at the terminals of the battery is dependant on the current as there will be a voltage drop across the cell's internal impedance when current flows. This means when the car is running, the pack voltage will fluctuate with current and will be an unreliable indication of how much capacity is left in the battery. There are various methods of estimating SOC, these include but are not limited to:

Coulomb Counting - This is where the change in charge within the cell is estimated by integrating the current in & out of the cell. It has a low computational complexity, simple implementation and is reliable when the battery's runtime is short as any drift in error is corrected when current stops flowing at the end of use when the voltage at the battery's terminal is the same as its Open Circuit Voltage (OCV).

Equivalent Circuit Model - A model is created for the battery that is simulated at run-time [4]. Data for voltage, current and temperature are gathered during operation to feed the model, giving a simulated representation of the battery that SOC and internal resistance can be extracted from. This method is computationally complex and thus runs a risk of being too demanding on the embedded system at runtime, although has the potential for being more reliable.

Extended Kalman Filter - An Extended Kalman Filter (EKF) is a standard means of non-linear state estimation that isn't limited to batteries. EKFs are used in a variety of systems for correcting error and predicting the next state based on the previous state and parameters fed into it [5]. These filters are great at reducing noise and making an estimated state of a model more accurate.

The advantages for coulomb counting over others are: it is easy to implement on a C based embedded system, it has a reduced computational complexity freeing up processing power for its many other important tasks as well as the runtime being short for our battery so any drift by error will be corrected upon the end of the vehicle's drive. From the advantages above, for the purpose of this year's USM EV battery, coulomb counting has been chosen as the system's SOC estimation method.

SOC estimation is a very open-ended part of this design that has much potential to be improved in the future with various methods and complexities, however a simple, efficient estimation is more than sufficient for operating the battery in a safe and controlled manner. A more complex means of SOC estimation could predict the remaining life-time of the battery and individual cell stacks, have a more accurate estimation of SOC and estimate other parameters such as internal impedance within the cells.

The hall-effect current sensor is rated for taking measurements at $50Hz$, so that will be our current sample rate f_s. Discrete integration will then be performed at this rate to determine the accumulated change in charge of the cell with the following equation:

$$x[n] = x[n-1] + \Delta t * u \tag{3}$$

Where $x[n]$ is the current state, $x[n-1]$ is the previous state, Δt is the change in time which in our case will be $1/F_s$ so $20ms$ and u is the most recent current value.

This only gives us a change in SOC so an initial SOC is required to determine the actual SOC of the battery. The initial SOC can be mapped to the battery's OCV with a HPPC test. This is used to first determine the capacity of the battery by fully charging then fully discharging it. Then it is fully charged and fully discharged again in stages of SOC to build an SOC against OCV curve. This curve can be used to extract an initial SOC based on the current OCV of the battery on start-up, so a change in SOC reflects an accurate estimation of the remaining capacity of the battery.

3.2.7 Fan Control & Tachometer

The fan control for the system only uses a single PWM channel for controlling all four internal fans and two external fans. The PWM signal runs at $5kHz$ with a variable duty cycle. The duty cycle for the fans is determined by the temperature of the battery according to the configurable fan curve shown in Figure 36.

To ensure the fans are spinning as they're supposed to, tachometer readings are taken for each fan individually. The tachometer sends out a pulse every half rotation of the fan. The equation for extracting the fan's speed from the pulses is shown below:

$$rpm = \frac{30n}{T_n} \tag{4}$$

Where in the above equation n is the number of pulses recorded and T_n is the time taken between receiving the first and last pulse.

As a duty cycle doesn't directly correlate to a fan speed because the fan's steady state speed varies with air flow and power, as long as there is a minimum of $60rpm$ the fan will be considered working. If the fan has a duty cycle other than zero but the fan rpm is less than 60, a fault will be triggered. It is important that all fans are functioning as expected as during runtime in the vehicle the battery is expected to have a high thermal generation, with failing fans the temperature will cause a thermal shutdown which will have an impact in the FS competition.

3.2.8 Fault & Shutdown Handling

Knowing when to shutdown the battery is vital to ensuring the safety of the driver and everyone involved with the USM vehicle. The system has been allotted 64 different fault codes split up into four different categories: A B C & D; most of which will not be used in this project but allows room for development and innovation, of which may arise with new faults and errors to look out for. There are many possible points of failure within the battery ranging from loss of communications to open-wiring of the cell's measurement taps.

Points of Failure - At this point in time, there are many points of failure to keep track of that have been outlined below, with more sure to come as the project is handed down to the next year of the USM team.

- Cell voltage too high for charging.
- Cell voltage too low for discharging.
- Cell voltages too far apart.
- SOC too high for charging.
- SOC too low for discharging.
- Open-wiring of cell voltage measurement points.
- Temperature too high.
- Temperature too low.
- Communications over isoSPI to slaves lost.
- Communications over CAN bus to the vehicle lost.
- LTC6804-1 BMS chips not selecting correct MUX pin.
- Current draw too high.

- Fans not spinning as expected.

Not all of these faults will warrant shutting the battery down and may only issue a warning, especially during testing it is helpful to be able to ignore certain faults to test other ones are functioning as expected. Each fault is therefore assigned a flag via the configuration utility which sets whether or not the fault should be fatal thus killing the battery or is just a warning. Most of the limits for these faults are also configurable via the configuration utility to allow for testing under certain conditions and criteria that are not absolute limits for the battery. This means limits can be set at safe and nominal battery states but faults can still be tested without putting the battery at risk by testing to its extremes.

3.2.9 Configuration Utility

The configuration utility for the BMS is a crucial component for testing the device in different scenarios and calibrating it to fit in the untested and variable environment of the FS vehicle. The configuration is set up as a large array of 32-bit unsigned integers containing each parameter that needs to be configured and its associated index in the array. The configuration sequence is shown in Figure 10 below:

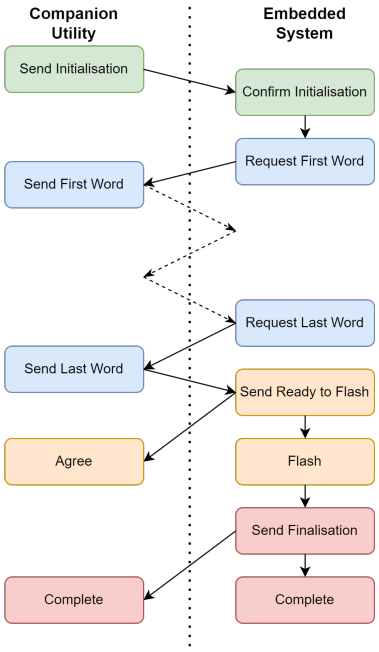

Figure 10: Flowchart of configuration sequence of BMS.

The configuration is comprised of two CAN messages, one from the companion utility and the other from the BMS itself. The first byte of the message dictates what is contained in the message: an initialisation, data, a finalisation or an error. During an initialisation or finalisation event, the data within the message can be ignored, it is only necessary to read past the first byte during an error or data event.

There is also the reverse process of receiving the configuration from the BMS to load into the companion utility. This is important to allow a different device running the companion utility to get the current profile of the BMS for diagnosing problems or modifying the configuration.

3.2.10 Cell Balancing

Cell balancing is when cells in a battery are charged or discharged to bring them all within a certain voltage range of each other. This is important for maximising the usable charge in the battery. As a battery charges, without balancing they will all have differing voltages as shown in Figure 11 because every cell has a slightly different chemical composition so charge and discharge at slightly different rates.

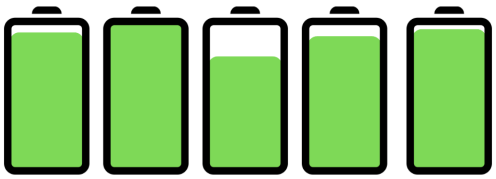

Figure 11: Cell charges without balancing - Charged.

At this point one cell is fully charged and thus no more charge can safely enter the battery so charging must stop. If the battery is then used in this state it can only be discharged until the lowest cell is fully discharged. This means there is charge left in the other cells shown in Figure 12 that cannot be accessed without over-discharging the lowest cell which will harm it. This deviation in voltage will only get worse after each charge and discharge cycle until the cells are balanced.

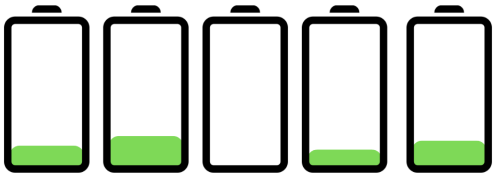

Figure 12: Cell charges without balancing - Discharged.

The BMS has been implemented with passive balancing to discharge cells that are too high a voltage during charging, letting all cells reach their maximum charge and thus less charge will be wasted after discharging the cells to depletion. The effects of voltage deviation during discharging will be fixed upon the next charging cycle instead of getting progressively worse after every discharge cycle. This gives us the discharge cycle shown in Figure 13 after being fully charged with passive cell balancing.

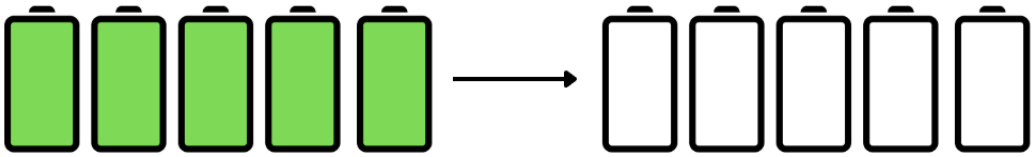

Figure 13: Cell charges with balancing.

The algorithm for deciding which cells should be balancing is based on a few parameters shown in the **Balancing Settings**. The process is shown in Figure 14 which brings all the cells in the battery down to the lowest cell voltage with a configurable threshold voltage range.

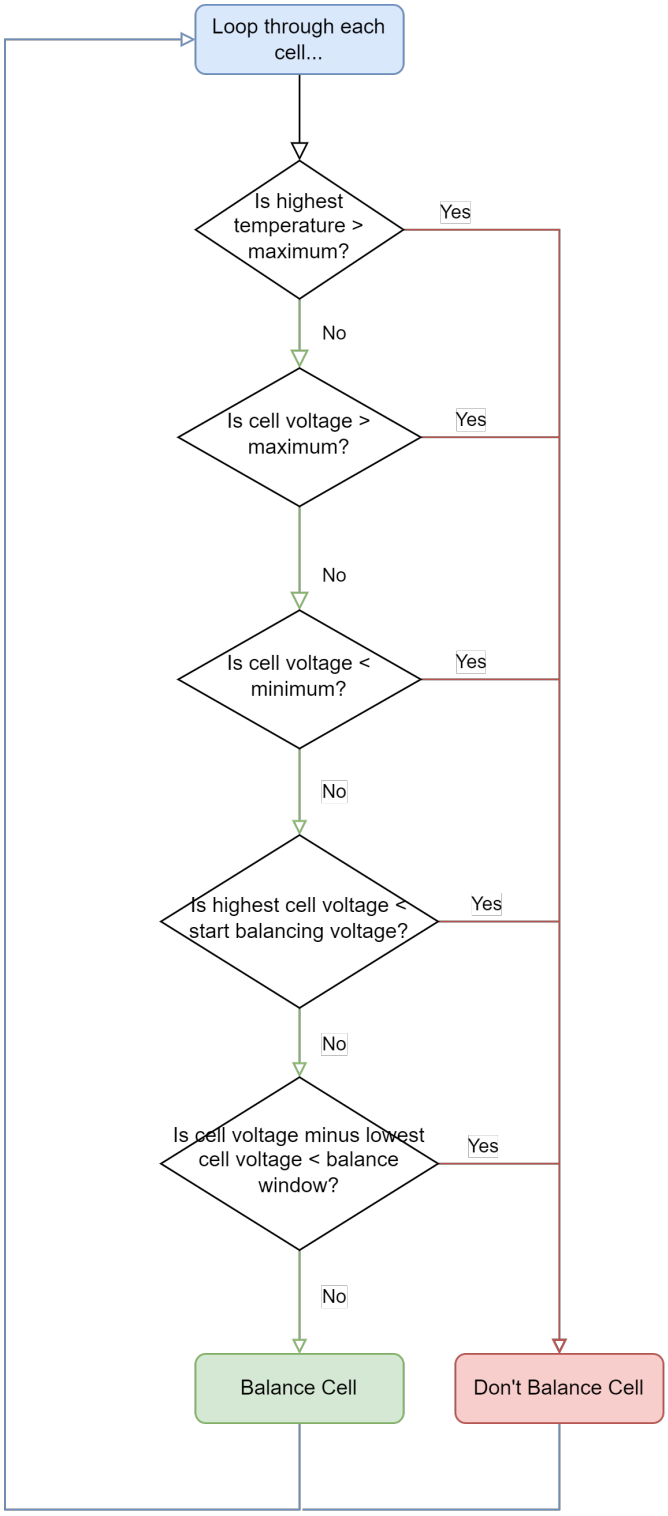

Figure 14: Flow diagram of cell balancing.

With this technique we can achieve cell balancing to converge all cells to within $10mV$ of each other. As passive balancing generates heat by dissipating excess charge over resistors, a FLIR thermal camera was used to monitor the temperature the balancing circuit was producing which can be seen in Figure 15.

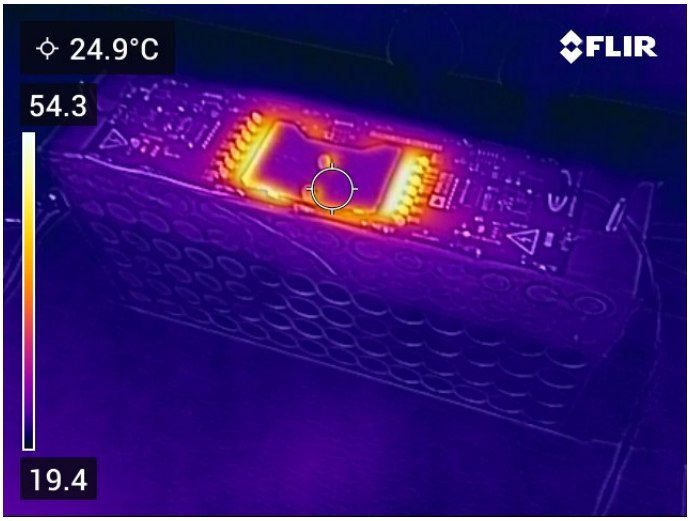

Figure 15: Thermal image of cell balancing in real time with FLIR camera.

4 Companion Utility Design

4.1 Overview

With a device that is in control of a high voltage battery it is crucial that every aspect can be thoroughly inspected and tested before it gets put into a live vehicle with a driver. This is where the companion utility comes in. It allows viewing of many different aspects of the BMS in real time including: serial messages, CAN messages, temperature, voltage, SOC, all fault states, tractive state, relays states and more. The companion utility has been written in C++ using Qt packages to create an interactive and user friendly GUI that is functional with comprehensive monitoring.

4.2 Software Implementation

4.2.1 Serial Interface

The serial monitor is the main interface between the BMS and the companion. It allows CAN messages to be sent from the BMS to the companion and vice versa via the serial to CAN adapter. The serial data is read in and parsed into the CAN bus utility if it is a valid CAN message, otherwise it can simply be used as a generic serial monitor.

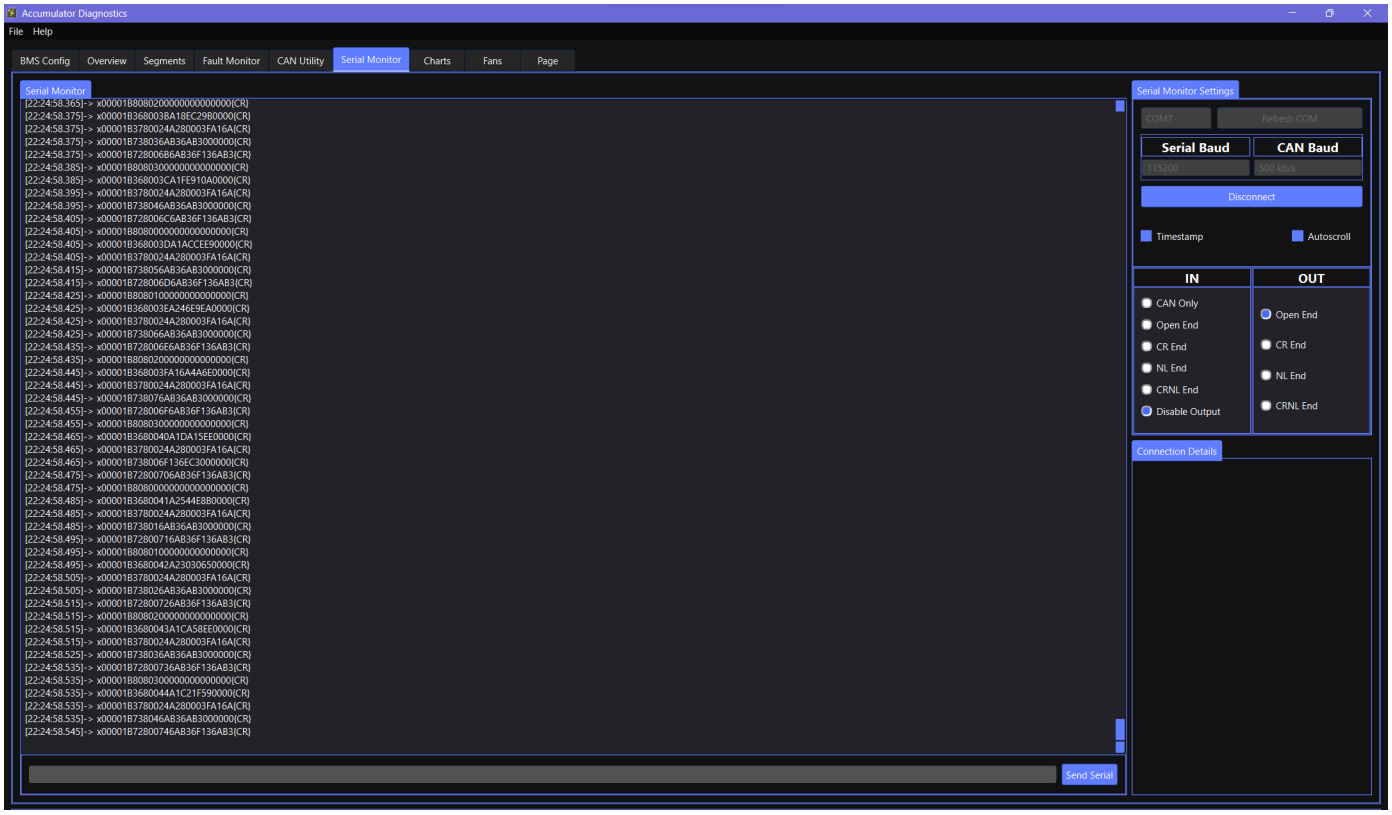

Figure 16: Serial Monitor built into Companion Utility.

This interface can operate at a configurable baud rate via the drop down menu in Figure 16 as well as send serial data by typing a message into the bar at the bottom and pressing "Send Serial". The interface can break up long strings of serial data based on a configurable end-of-line character whether that's a new-line character (ASCII 0x0A) or a carriage return (ASCII 0x0D) or both as well as append these onto a message the user would like to send.

The user is also able to select whether to add a timestamp to the parsed serial data and select whether

the feed will auto-scroll with each new line of data. On top of this the user can change the set baud rate the serial to CAN adapter uses on its first connection. All this configurability of the serial interface is necessary to facilitate any slight changes in future design without needing a complete redesign of the application and interface.

4.2.2 CAN Bus Utility

Viewing the CAN bus messages is an important part of testing and diagnosis of errors during development, whether it's an internal problem within the BMS, or an external problem elsewhere in the car. It has been useful for diagnosing issues with other partially tested systems such as the ECU and motor-controller communications, as such it is very useful to be able to view and monitor all messages on the CAN bus. The utility receives parsed serial messages with the message ID and data separated out for easy processing. The messages are filtered based on their ID and passed onto their appropriate functions for further processing, i.e. displaying a temperature or voltage measurement.

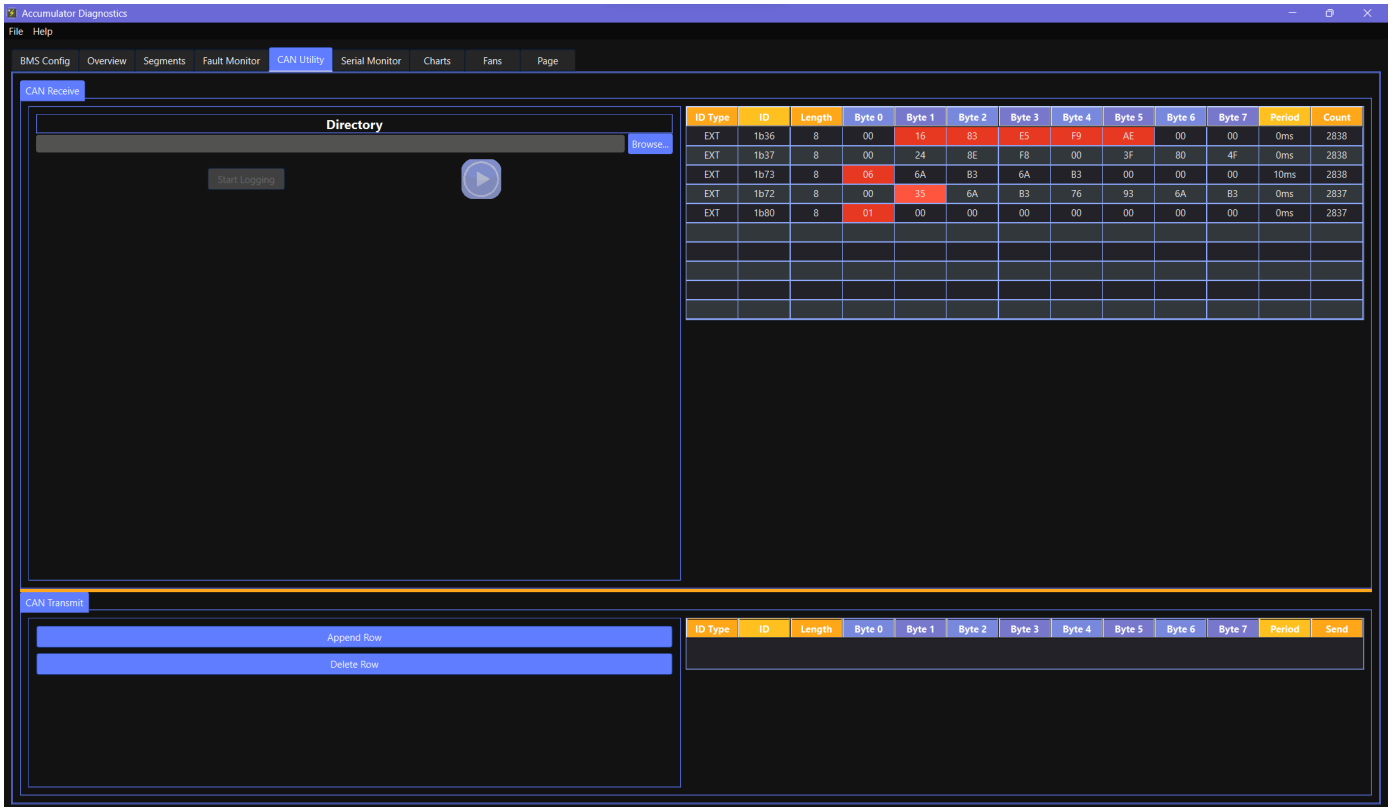

Figure 17: CAN Bus Monitor built into Companion Utility.

The CAN bus utility comes in two parts: CAN Receive & CAN Transmit.

CAN Receive The upper section of this page contains the receiving part of the CAN bus utility. Every unique CAN ID is added to the table and updates as a new message containing that ID comes in. If a byte is different to the previous byte in that message, it is highlighted red to indicate when data is changing, which is useful to know when diagnosing problems. It also keeps track of how often each message is coming in and a counter of how many of each message have been received. All received messages can also be logged to a file of the user's choosing and can 'play' and 'pause' the logging. The 'pause' is useful in-case there is a time between tests where any data would be useless and only take up storage space. The logging of the entire CAN bus is an incredibly useful feature as it provides a complete timeline of everything that happened in the car, not just the battery, which the USM team can analyse and reconstruct to gain a better understanding of how the vehicle is performing and for testing other systems in the car.

CAN Transmit The lower section contains the transmitting part of the CAN bus utility. The user can add or remove CAN messages with the ability to set the ID, number of bytes, the data contained within those bytes as well as the period of the message if it's to be recurring periodically. This feature is less critical but can help during testing parts of the car where messages aren't being sent as they should be from certain devices.

4.2.3 Temperature Visualisation

Viewing temperatures during charging is mandatory as per the FS rules specifications but is also a great way to test all temperatures are being read in correctly as well as being shown in their correct location. There were several iterations of how this information has been displayed, each iteration more clear and easy to interpret than the previous. The first method was a simple grid of colour-interpolated squares, each square representing a thermistor positioned as it is on its corresponding battery segment shown in Figure 18.

Figure 18: First iteration of heatmap with simple colour squares.

Although this method is simple and only requires an interpolation between set colour stamps, it is crude to look at and difficult to discern at first glance what is happening on the screen when this should really be as easy to understand and visualise as possible.

The next iteration was using circles with radial gradients to add a smoothing between each temperature value with the same colour-interpolation for the centre points of each temperature shown in Figure 19.

Figure 19: Second iteration of heatmap with radial gradients.

This method for visualisation was better than the first in terms of readability of temperature data at-a-glance but computing the smoothing of gradients for eight separate segments each with 21 temperatures is resource intensive and performance heavy, slowing the program down. The readability gained was not worth the performance loss so further methods were investigated.

This is where shaders come in. The problem with the previous iteration was it was computing all the interpolated colours for each pixel using the Central Processing Unit (CPU) whereas shaders are programs that run on Graphics Processing Unit (GPU)s so can take advantage of its parallel processing which is great for computing per-pixel. These shaders were written using OpenGL Shading Language (GLSL) on the OpenGL rendering pipeline. This method is very similar to the previous however its is built using a shader and runs on the GPU, massively increasing the performance of the program which is shown in Figure 20.

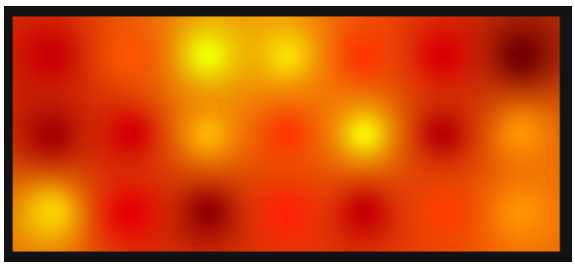

Figure 20: Third iteration of heatmap with radial gradients by writing shaders.

There is however an issue with this implementation. The computational complexity of the shader is proportional to the number of thermistors loaded into it as per-pixel, it loops through each thermistor to determine which ones it is closest to to determine what colours it should interpolate between. This performance was okay with the seven by three configuration but when extended to a larger set of temperatures there were visible performance hits. This did not even reach being fully implemented as it was clear early on how unacceptable the performance was.

The final iteration was true to the initial vision of how this should have looked, while retaining performance and ease of readability which can be seen in Figure 21.

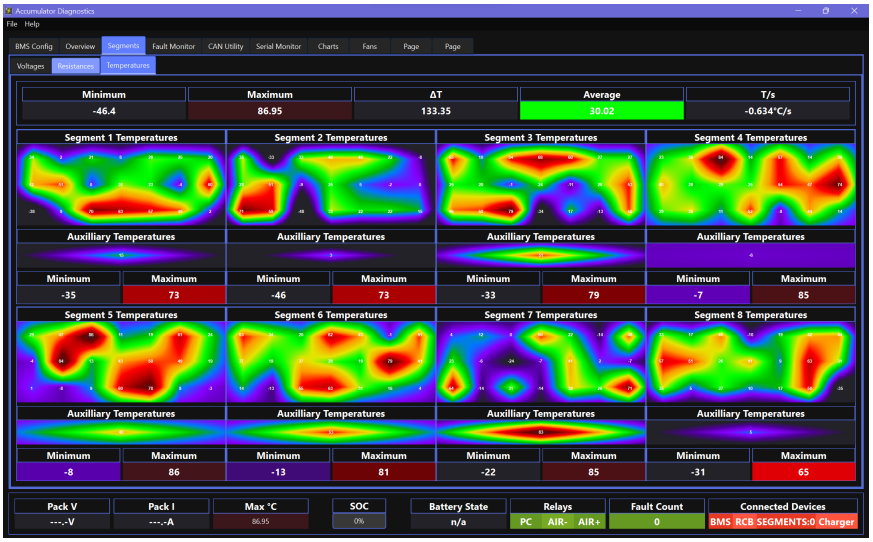

Figure 21: Fourth iteration of heatmap with bi-linear interpolation shader.

This method uses a bi-linear colour-interpolation algorithm to compute the colour of each pixel giving a very smooth and intuitive graph for understanding the overall temperature profile of the battery. It retains low computational complexity even with large amounts of temperatures by instead of looping through the position of each temperature to determine which one that pixel is closest to, because each

thermistor is in a predetermined location in the frame, it can just compute which ones are closest without the loop. This shader can easily run on low-end devices provided they have a GPU, freeing up computation for the more important tasks of the application i.e. reading in data from the CAN bus.

4.2.4 Voltage Visualisation

Voltage readings must also be displayed as per the FS rules specifications. These voltages must be accurate and current so a user or technician can monitor the individual cell voltages for general operation and for testing different scenarios making sure fault detections happen as expected.

The voltages will be displayed in two different forms: a simple grid of printed voltages organised in columns of their corresponding segment, and a chart of the battery voltages to aid in visualising the entire state of the battery at-a-glance. The two displays are shown below:

Figure 22: Voltage page in Companion Utility.

For precise reading of the cell's voltages the user can look at the grid, however this is difficult to read when trying to get a picture of the battery as a whole or when trying to identify any outliers or abnormal cell voltages. To get an easier to understand, holistic view of the battery, a shader was implemented to get a colour-indicated bar graph of all cell voltages. Each bar has a colour at the top to indicate its voltage level: red at its lowest and green at its highest. Each bar can also turn blue to indicate whether a cell is being balanced, turn pink if it has the highest voltage out of all cells, and purple if it has the lowest voltage out of all the cells.

In designing the above shader for visualising cell voltages there were several stages in the design before the final result, some with interesting and unintentional visuals. Some interesting interim shaders are shown below:

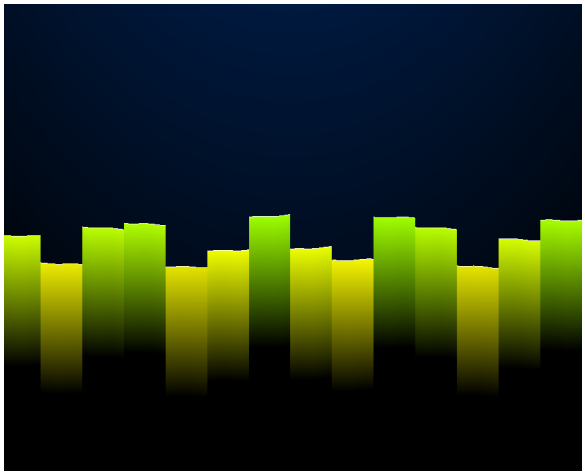

Figure 23: Voltage visualisation early stage 1.

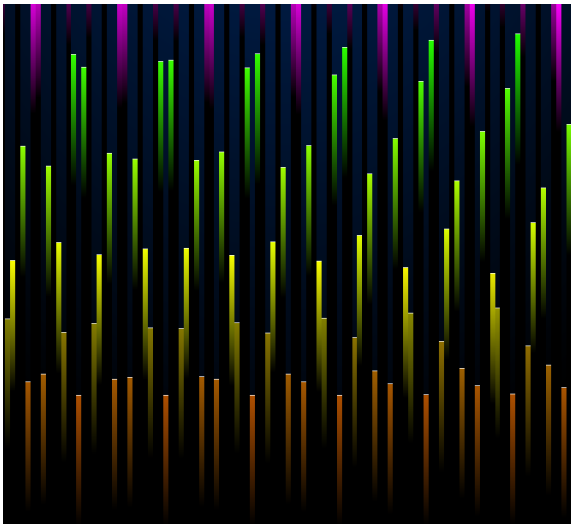

Figure 24: Voltage visualisation early stage 2.

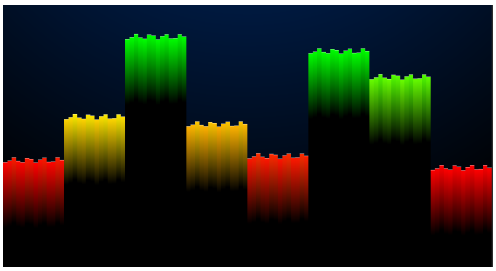

Figure 25: Voltage visualisation early stage 3.

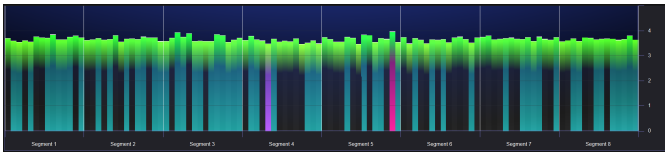

Figure 26: Voltage visualisation early stage 4.

4.2.5 Flexible Parameter Graphing

Being able to monitor battery measurements at a single instance in time provides useful information on the current state of the battery but doesn't give an understanding on how the battery is changing over time. Being able to plot a battery parameter over time can easily show the user how a temperature or voltage is changing which is useful for testing the features implemented in this project.

After implementing voltage and temperature visualisation shaders, a line graph shader was investigated as a means for plotting battery parameters over time. The stages in the design of the shader are detailed below.

Simple Sine The first part to tackle was drawing a simple line that changes across the graph. A sine wave was chosen as it is an easy function to define that changes based on an input. The sine wave was defined internally within the shader with a fixed window and axis shown in Figure 27.

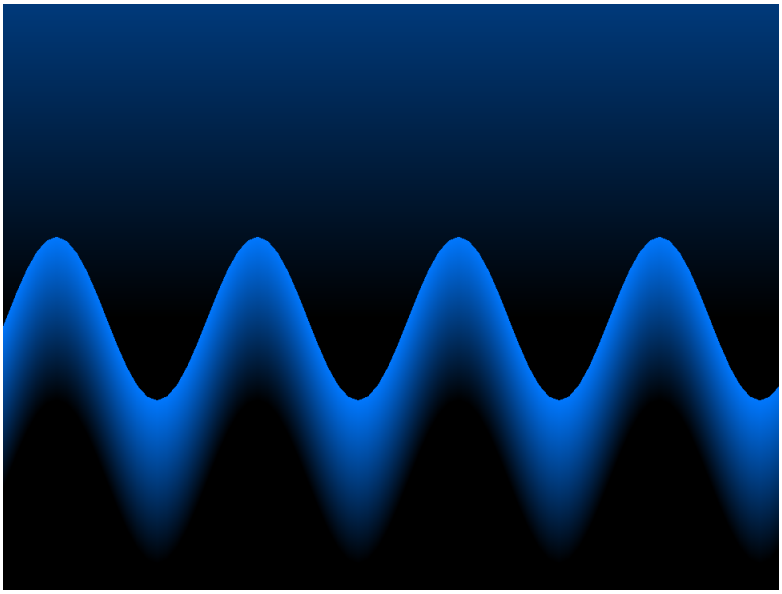

Figure 27: First iteration of line plotting shader.

Line Plot Next the graph needs to have a set of points as an input which can be displayed for the user. Each point must have a line drawn between them which is done by interpolating between each point [n] and their next point [n+1]. The line plot is shown in Figure 28.

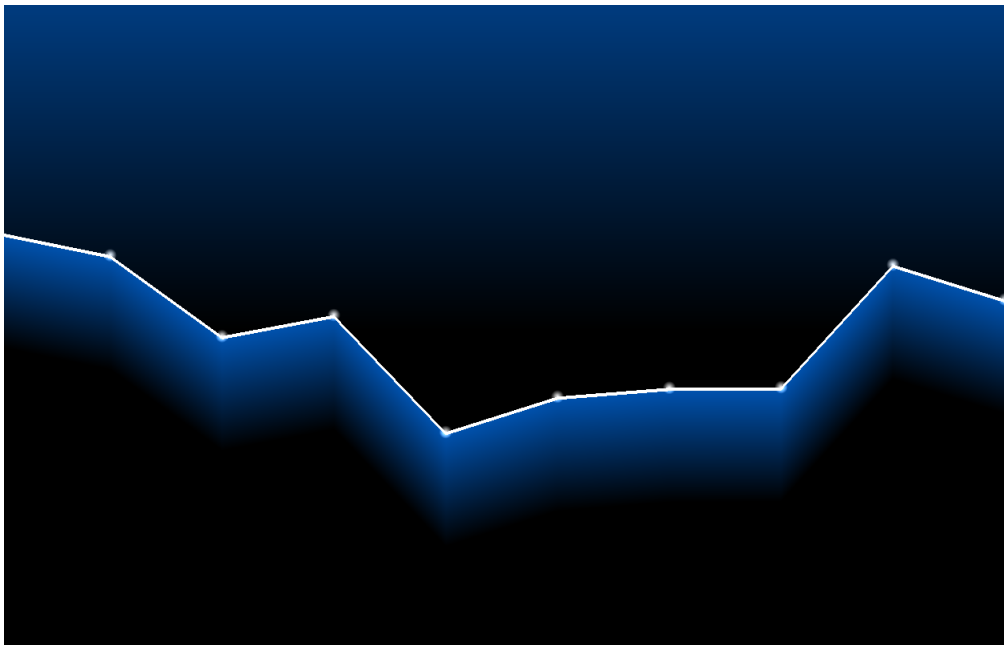

Figure 28: Second iteration of line plotting shader.

Configurable Plot Configurability is a recurring aspect of this project so it is important to keep it in mind when implementing new features, as such the graphs have been equipped with a range of parameters to alter how the graph is displayed. Some of these settings are background colour, foreground colour, line width and axis window sizes. Testing for the variety of parameters is shown in Figure 29.

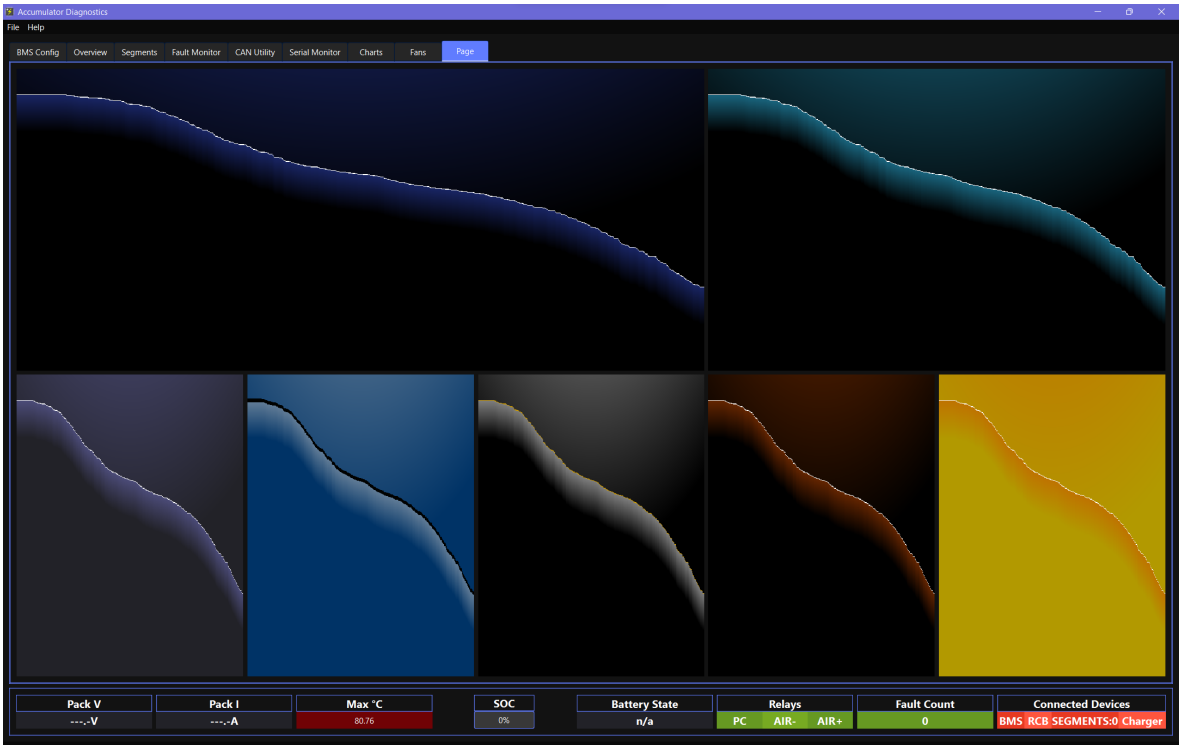

Figure 29: Third iteration of line plotting shader.

Flexible Measurement Plot Finally the point of these graphs is for the user to be able to select what to plot from many options. The options the user has been given are: any battery temperature, any battery voltage, any fan speed and the maximum & minimum temperature which is shown in Figure 30.

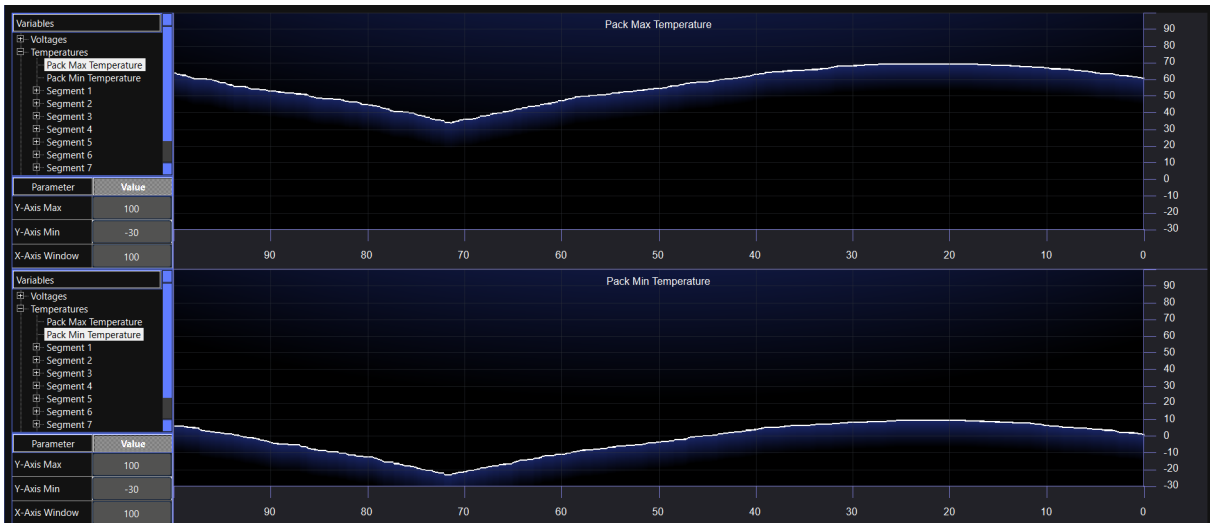

Figure 30: Final iteration of line plotting shader.

4.2.6 Utility Settings

The companion utility is designed with the future in mind. This means designing the program in a way that means it doesn't require a redesign when the USM team builds a new battery with different cells, cell configuration or thermistor grids. Part of this is to anticipate what factors about the battery may change in future builds and making these parameters configurable to the user. These parameters have been identified as follows:

- Number of fans.
- Number of segments.
- Number of cells per segment.
- Rows of Thermistors per segment.
- Columns of Thermistors per segment.
- Number of auxiliary thermistors per segment.
- Number of other temperatures in the battery.
- Number of other parameters to show in the utility, their name and units.
- Debug console verbosity.
- Number of stop-clocks and their name.

These parameters are written to a comma separated "settings.conf" file which is read in upon start-up to setup the application appropriately. If these parameters are changed while the program is running, all the required changes are made to the application accordingly to support the new interface. A page from the settings pop-up window is shown below:

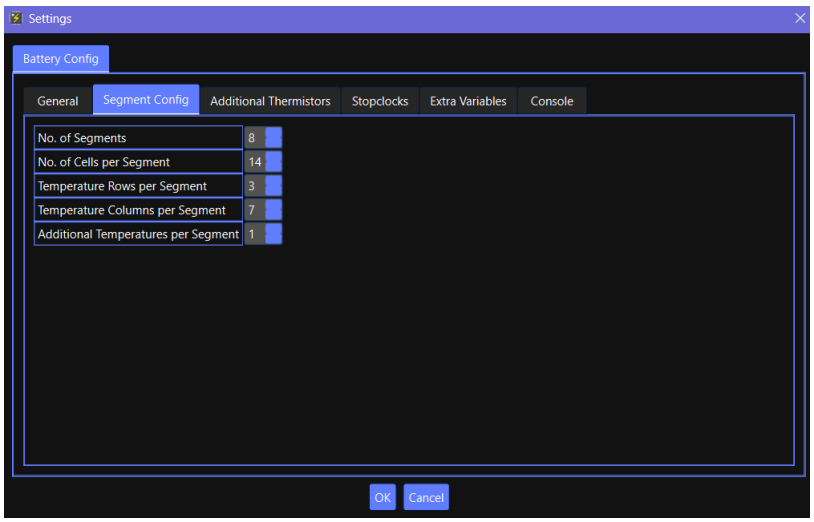

Figure 31: Settings page in Companion Utility.

4.2.7 BMS Configuration Utility

The companion application has been fitted with a section purely for setting parameters to be sent over to the BMS via the flashing sequence. This page has been equipped with a variety of settings pertaining to different components of the BMS with graphs to help visualise what parameters have been set to. The different sections and their respective settings are outlined below.

Voltage Settings

- Minimum Cell Voltage: The lowest voltage an individual cell can drop to before the shutdown is triggered.
- Maximum Cell Voltage: The highest voltage an individual cell can reach to before the shutdown is triggered.
- Maximum Cell Voltage Charging: The highest voltage an individual cell can charge to before shutdown is triggered.
- Maximum Cell Deviation: The maximum difference between the lowest voltage cell and highest voltage cell before a shutdown is triggered.

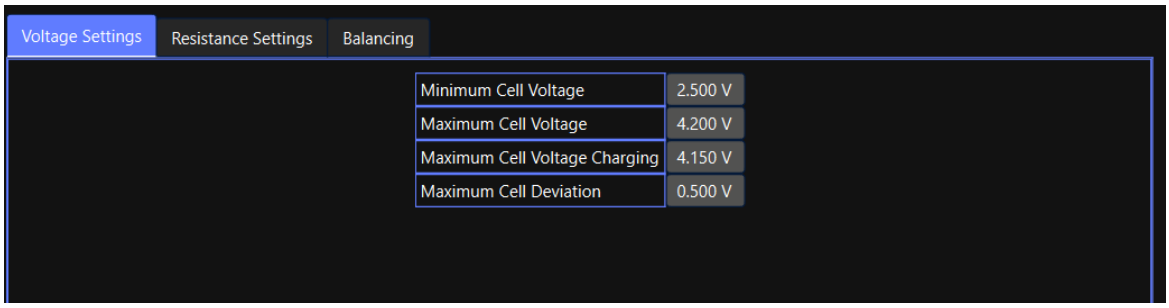

Figure 32: Voltage Configuration Page.

Balancing Settings

- Cell Balancing Enable: Enables automatic cell balancing based off cell voltages. When disabled, override is enabled via the voltage visualisation page.
- Minimum Cell Voltage while Balancing: Only balance cells that are above this voltage.
- Maximum Cell Voltage while Balancing: Only balance cells that are below this voltage.
- Start Balancing Voltage: Only enable balancing if at least one cell is above this threshold. If all cells are below, automatic balancing is disabled.
- Cell Balancing Window: The difference in voltage between the lowest cell and all other cells where balancing is enabled for the respective cell. If the difference in voltage is less than this window the cell is considered balanced and requires no further balancing.
- Balancing Period: The length of time balancing happens for before taking a wait period.
- Balancing Wait Period: The length of time balancing pauses for after having been balancing for the balancing period.

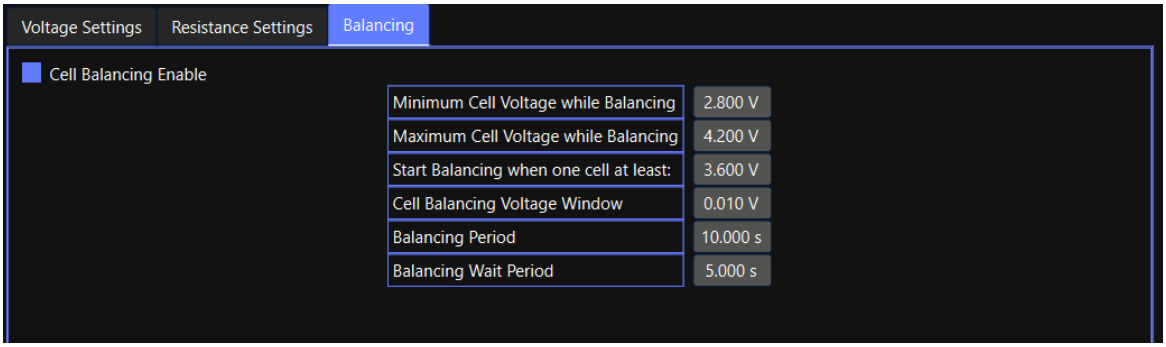

Figure 33: Balancing Configuration Page.

Charging Limits

- Maximum Charge SOC: The SOC that when reached during charging, disables charging.
- Maximum Charge Current: The maximum current guide transmitted to the charger unit to request a charge current.
- Thermal Derate Upper Start: The temperature at which the maximum charge current starts to derate as temperature increases during charging.
- Thermal Derate Upper End: The temperature at which the maximum charge current has finished derating to zero as temperature increases during charging.
- Thermal Derate Lower Start: The temperature at which the maximum charge current has finished derating to zero as temperature decreases during charging.
- Thermal Derate Lower End: The temperature as which the maximum current starts to derate as temperature decreases during charging.
- Thermal Shutdown Upper: The upper temperature at which a shutdown is triggered during charging.
- Thermal Shutdown Lower: The lower temperature at which a shutdown is triggered during charging.

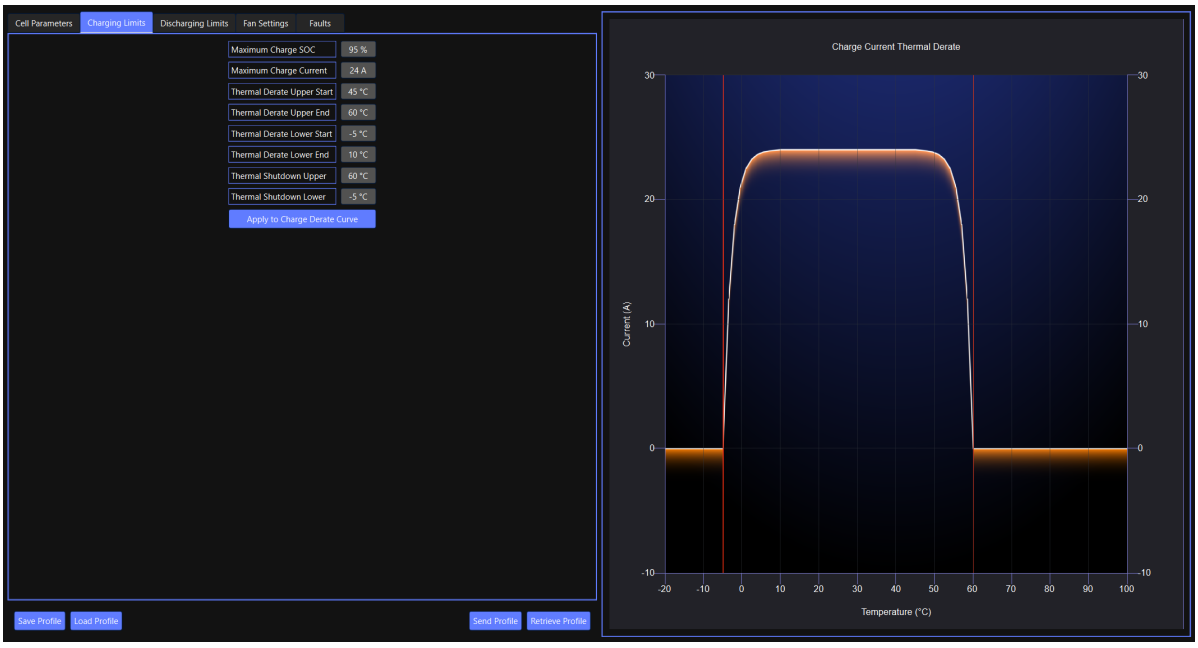

Figure 34: Charge Limit Configuration Page.

Discharging Limits

- Minimum Discharge SOC: The SOC that when reached during discharging triggers a shutdown.
- Maximum Discharge Current: The maximum current guide transmitted to the ECU for derating torque requests.
- Thermal Derate Upper Start: The temperature at which the maximum charge current starts to derate as temperature increases during discharging.
- Thermal Derate Upper End: The temperature at which the maximum charge current has finished derating to zero as temperature increases during discharging.
- Thermal Derate Lower Start: The temperature at which the maximum charge current has finished derating to zero as temperature decreases during discharging.
- Thermal Derate Lower End: The temperature as which the maximum current starts to derate as temperature decreases during discharging.
- Thermal Shutdown Upper: The upper temperature at which a shutdown is triggered during discharging.
- Thermal Shutdown Lower: The lower temperature at which a shutdown is triggered during discharging.

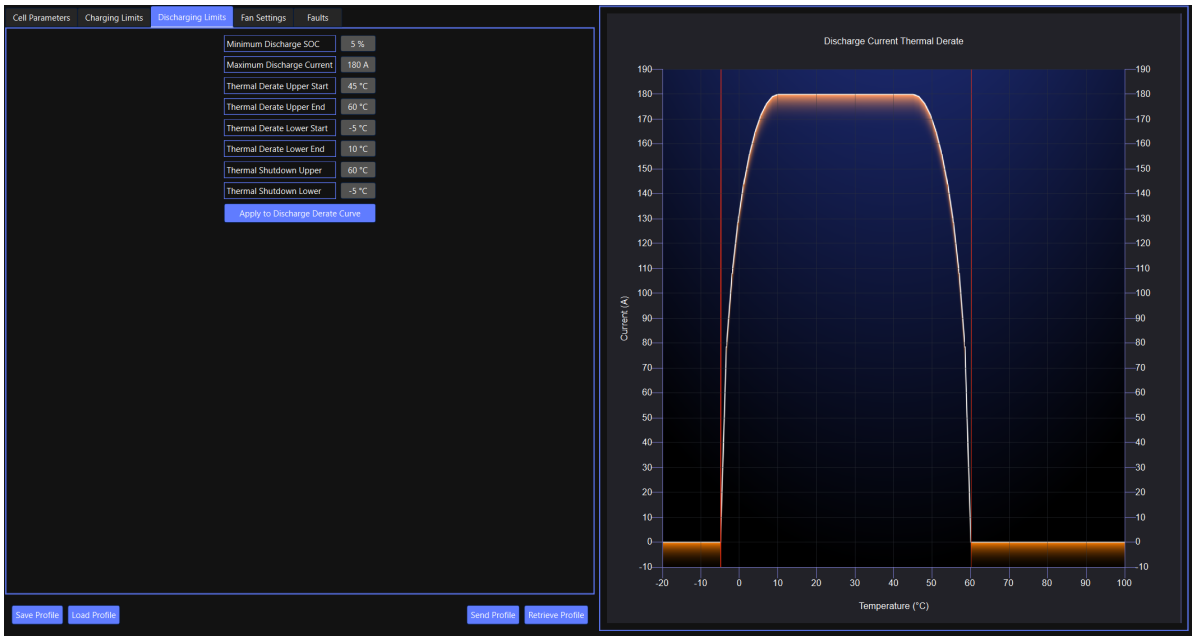

Figure 35: Discharge Limit Configuration Page.

Fan Settings

- Fan Duty Cycle 1: The temperature at which the duty cycle of the fans PWM is 0%.
- Fan Duty Cycle 2: The temperature at which the duty cycle of the fans PWM is 10%.
- Fan Duty Cycle 3: The temperature at which the duty cycle of the fans PWM is 20%.
- Fan Duty Cycle 4: The temperature at which the duty cycle of the fans PWM is 30%.
- Fan Duty Cycle 5: The temperature at which the duty cycle of the fans PWM is 40%.
- Fan Duty Cycle 6: The temperature at which the duty cycle of the fans PWM is 50%.
- Fan Duty Cycle 7: The temperature at which the duty cycle of the fans PWM is 60%.
- Fan Duty Cycle 8: The temperature at which the duty cycle of the fans PWM is 70%.

- Fan Duty Cycle 9: The temperature at which the duty cycle of the fans PWM is 80%.
- Fan Duty Cycle 10: The temperature at which the duty cycle of the fans PWM is 90%.
- Fan Duty Cycle 11: The temperature at which the duty cycle of the fans PWM is 100%.

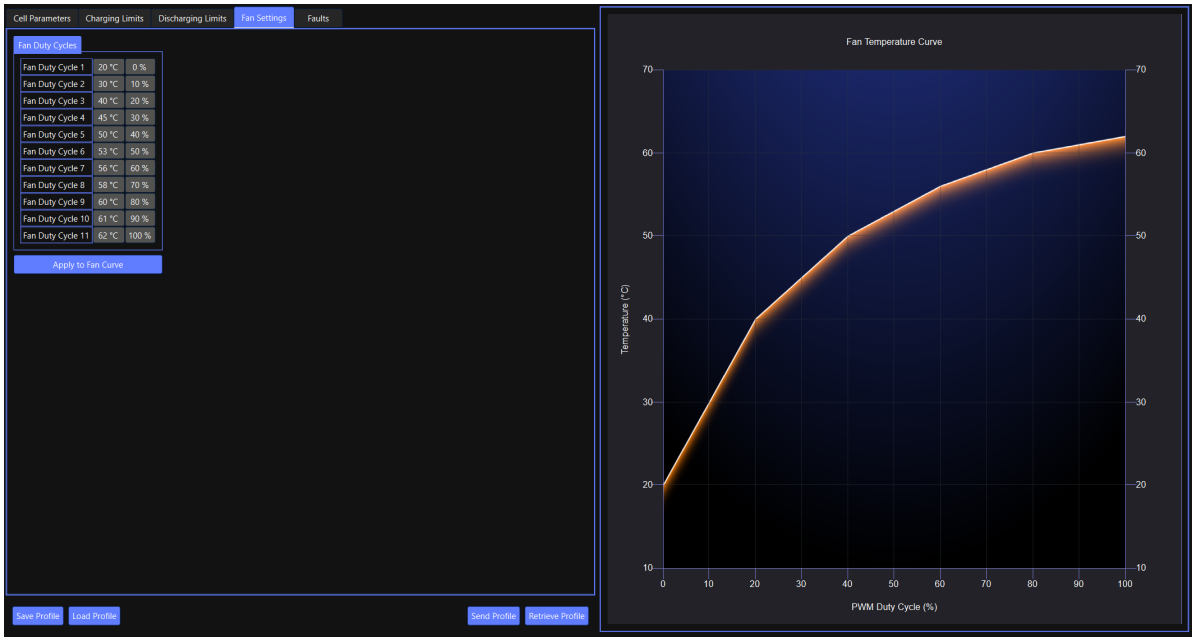

Figure 36: Fan Thermal Curve Configuration Page.

Fault Settings The fault settings are split up into four categories for organisation, each with sixteen faults available to fill, some may be obsolete for the time being until they are required. Simply click on the fault to set whether it is a fatal (red) or a warning (yellow).

Figure 37: Fault Configuration Page.

4.2.8 Documentation Viewer

As the program is going to be handed to the USM team to be used for the current battery as well as for future batteries, it is important that details about every aspect of the program and its interface with the battery is well documented and that this documentation is easy to navigate. The information contained within the documentation must be very specific in how parameters are stored, their word length, their endianness as well as specifics as to what CAN bus messages are transmitted and received and the information contained within them.

The documentation viewer has been implemented with Qt in the companion application as a Hyper-Text Markup Language (HTML) site in a web engine and can be accessed in the 'Help' option in the application's toolbar as shown in Figure 38.

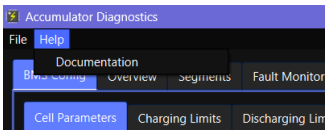

Figure 38: Documentation Viewer Help Toolbar.

The documentation viewer could have been implemented in a variety of ways:

External Website - Using an external website such as a wiki environment for documentation. An advantage of this method is there is no technical learning curve in implementing the documentation, although a disadvantage is that you must have access to the internet to view the website which is not ideal if out testing the USM vehicle on a track with no internet or mobile signal.

Word or PDF - Using Microsoft Word or PDF software is a great option for implementing the program's documentation with lots of flexibility in formatting with all information being contained within a single document. Hyperlinks can be also be added leading to datasheets and other areas in the document for easy navigation for future users. A problem with this is that the documentation is separate from the application.

Qt HTML Site - This is the decided upon method of implementing the documentation viewer. With it being a part of the companion utility it means there can be links within the program, such as help buttons, which bring up their respective page so users can easily find the information they're looking for. It is offline so does not require the internet to access and also retains the ability for hyperlinks to link between pages in the HTML site.

5 Battery Model

Battery models have many uses ranging from simulating battery behaviour to characterising existing batteries. There are many different kinds of implementations of battery models with varying complexity and computation required as well as either being a live model or an offline model.

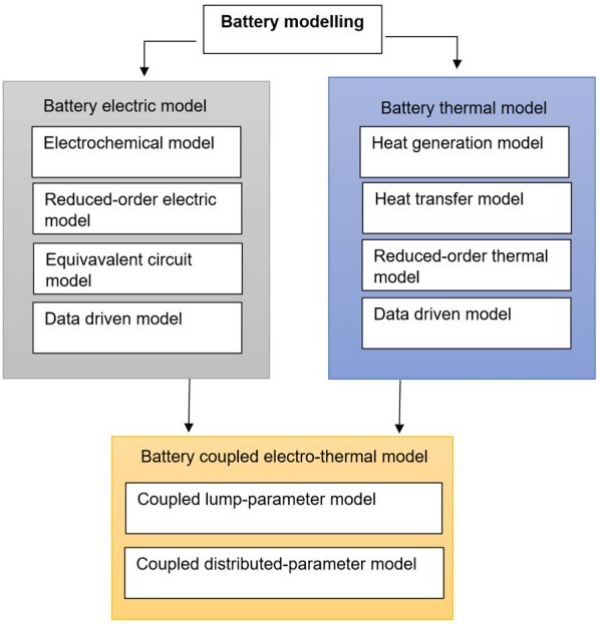

Figure 39: Classification of Different Battery Models [6].

A variety of model implementations were investigated ranging from Electrochemical Models [7] to Discretisation methods [8] to Equivalent Circuit Models [9]. Due to the model being a lesser component to the project, the simplicity and reduced computational complexity of the equivalent circuit model proved to be the best fit and thus was chosen for modelling the battery and testing SOC algorithms.

5.1 Equivalent Circuit Model

5.1.1 Simulink Model

The equivalent circuit model has been implemented in Simulink using the SimScape model package. It is compromised of a cell model in series with a resistor as the load and a switch to enable and disable discharge. The Simulink model is shown in Figure 40.

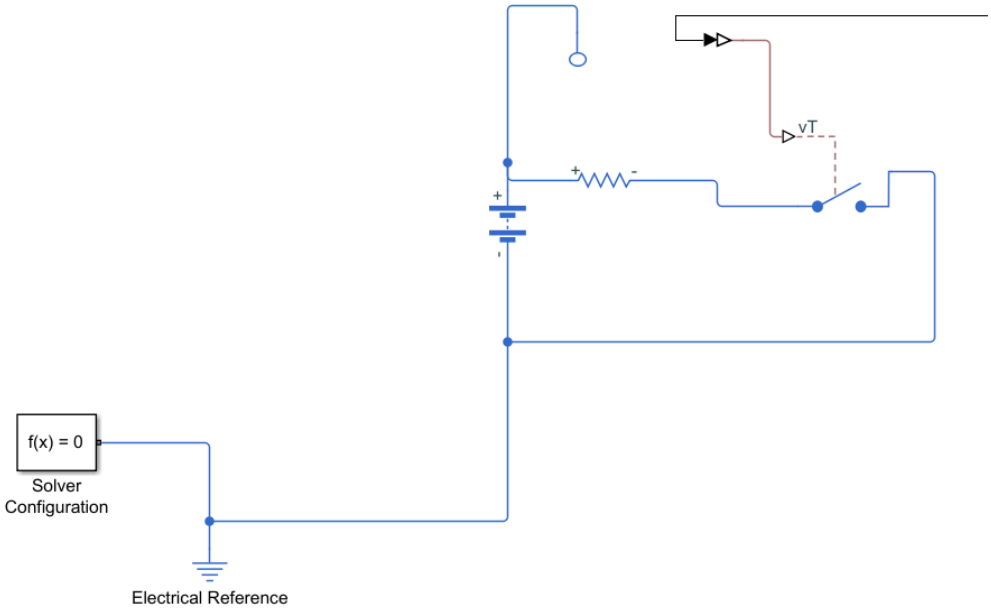

Figure 40: Battery Discharge Model Simulink.

This provides the model for simulating charge and discharge of the battery across a load while monitoring current and voltage. This current and voltage data can be exported to the MATLAB workspace to test out and validate SOC algorithms. Using a pulse generator on the input to the switch the model can simulate a periodic discharge of the battery giving a current curve that can be plugged into a Coulomb Counting algorithm. The voltage and current plots from the periodic discharge are shown in Figure 41.

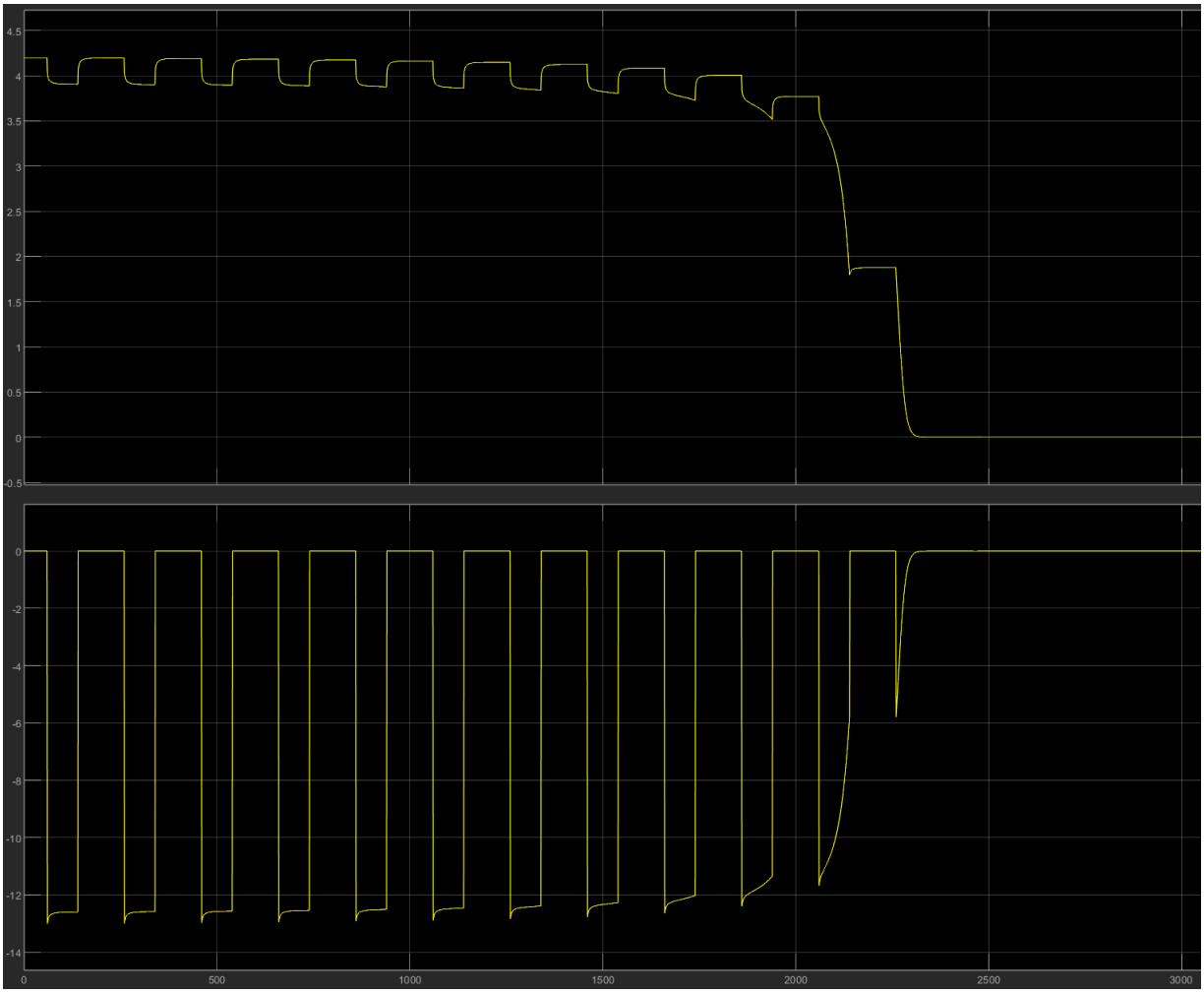

Figure 41: Plot of Voltage and Current from Simulated Battery Model.

The current data generated by the simulation from Figure 41 was plugged into the Coulomb Counting algorithm in MATLAB giving the following SOC plot shown in Figure 42.

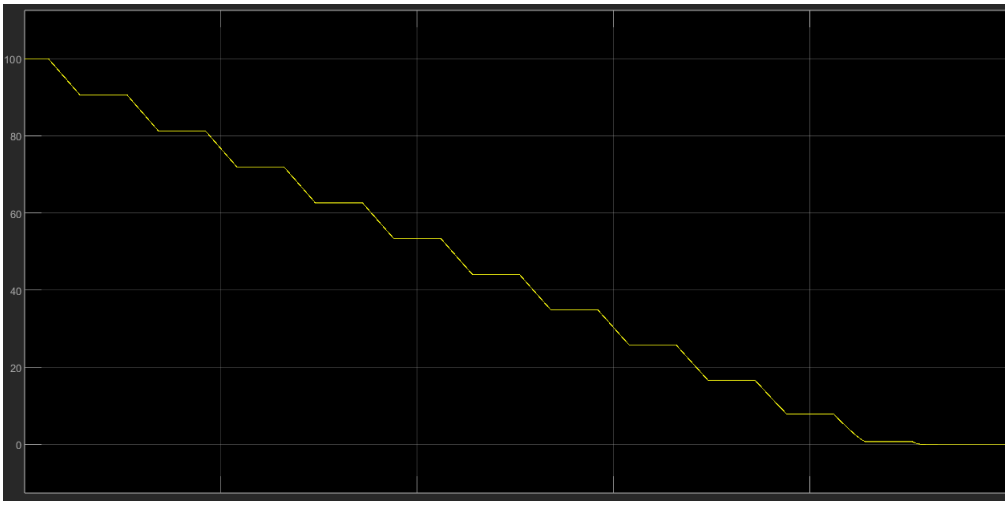

Figure 42: Plot of Coulomb Counting from Simulated Battery Model.

The above plot shows the SOC periodically decreasing as the battery discharges, however the decline in SOC is linear, giving an accurate representation of the amount of charge left in the battery. This linear change in SOC is desired as this shows the algorithm is working as expected from the current curve. With a validated SOC estimation algorithm, it can be implemented on the BMS embedded system on the USM team HV battery.

5.1.2 HPPC

Coulomb counting only computes a change in SOC so an initial SOC is required as an offset to determine the amount of charge left in the battery. This is where a HPPC test comes in, comprised of two parts: a full charge and full discharge of the battery, and a full charge and full discharge in stages of SOC.

Full Charge & Discharge - Charge the battery to full then discharge slowly while applying coulomb counting SOC estimation. Discharge until the minimum voltage has been reached and take the change in SOC as the total capacity of the battery. This provides calibration to the capacity of the specific battery segment or cell.

Discharge in SOC Stages - Charge the battery to full then discharge is stages of SOC, letting the voltage settle between discharge cycles. Each discharge cycle depletes the battery by the same percentage. After the voltage settles, collate the cell's OCV at that SOC to generate an SOC against OCV curve.

Using Simulink and the previous battery model, a HPPC test circuit was implemented to discharge the battery in stages of SOC with a wait period between discharge cycles. The simulation model is shown in Figure 43.

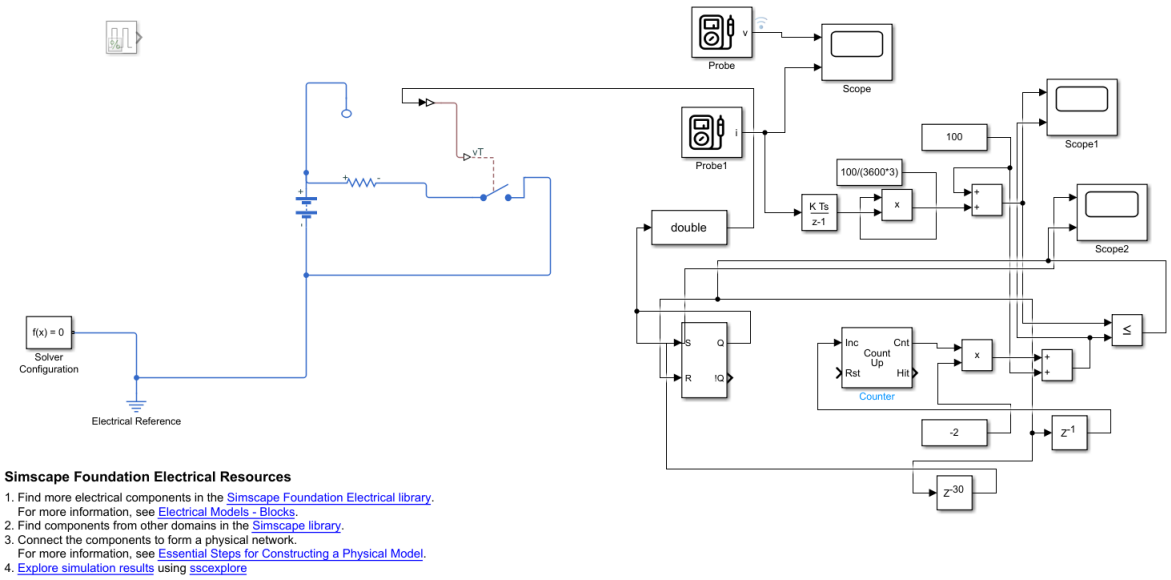

Figure 43: Simulated Battery Model HPPC Test Circuit.

This circuit generates the following voltage and SOC plots shown in Figures 44 & 45 respectively.

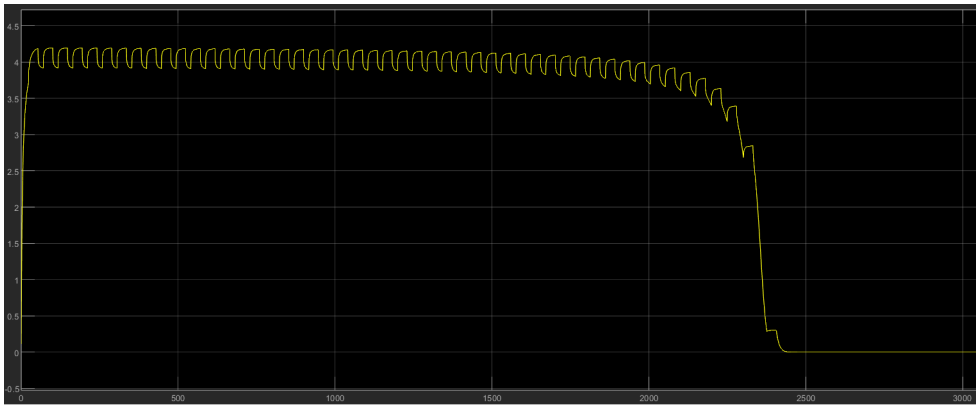

Figure 44: Simulated HPPC Voltage Curve.

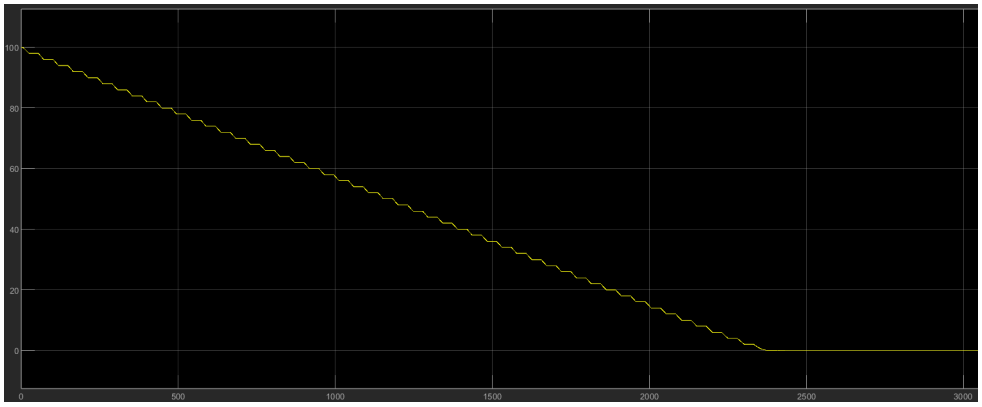

Figure 45: Simulated HPPC SOC Curve.

The voltage plot from Figure 44 was exported to the MATLAB workspace where every local maxima was collated together giving the OCV for every discharge cycle. These OCV are mapped to their respective SOC stage to generate the OCV against SOC curve shown in Figure 46.

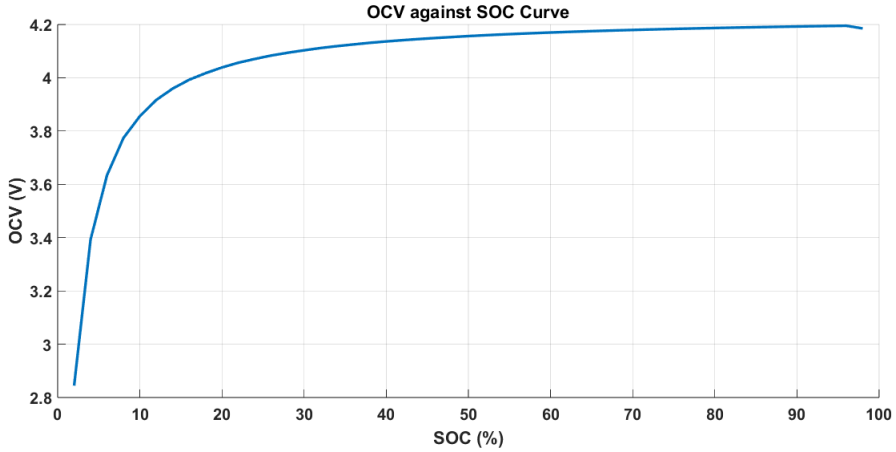

Figure 46: Open-Circuit Voltage vs State of Charge Curve.

The OCV against SOC curve shown in Figure 46 provides a map for determining the initial SOC for coulomb counting. This curve is the result of the HPPC test and validates the methodology for performing this test on the physical battery in the future to accurately determine the initial SOC of the USM team's HV battery.

6 Future Work

As this project lays the foundations for the team's custom battery management system, many systems had to be implemented and thus none could be specialised. This leaves plenty of room for innovation in the battery management system over various aspects that could use further development as well as additional features. Some aspects of the project that could be developed upon have been outlined below.

Active Cell Balancing - This project implements passive cell balancing to converge cell voltages after they've deviated during charge and discharge cycles which burns off the excess energy of cells that are too high a voltage. This is inefficient as it wastes charge and generates a significant amount of heat, both of which can be reduced by implementing active balancing in the battery management system. Active balancing is when excess charge is redistributed to other cells instead of wasted, reducing the amount of charge lost and the amount of heat generated when balancing cells.

State of Health - The BMS has no understanding of the health of the battery in terms of how degraded the cells have become over time and over charge & discharge cycles. Building a model that can monitor the health of cells in the battery can help to understand how degraded cells have become through use and can extrapolate the life span of each cell indicating when they should be replaced to retain optimum performance. However, as there is constant innovation within the USM team it is unlikely that the battery will keep the same cells for more than a few years so cell degradation will be limited.

Fault Detection - There is a fairly limited set of faults to diagnose that could certainly be expanded upon. These have been outlined previously with most of them being configurable via the companion utility. These faults are very obvious to detect in the battery with simple thresholds that if exceeded get triggered. However, there are less obvious things that could go wrong such as detecting sensor calibration issues and damaged components that could potentially be found by analysing historic voltage, current and temperature data [10].

Live Battery Model - The current model of the battery has been simulated externally from the battery, giving the BMS a static SOC to OCV curve for SOC estimation. A live model means having a model that runs on the BMS constantly updating and analysing sensor data to predict and estimate how the battery should behave. This would be more computationally complex but would mean for an SOC estimation that maintains accuracy as the battery degrades as well as being able to estimate cell parameters i.e. internal resistance.

HPPC of Battery - To gain an SOC to OCV curve that has been calibrated to the characteristics and behaviour of the HV battery, a HPPC test can be undertaken. This would require two complete charge and discharge cycles of the battery, at a low current to ensure accuracy, which would take a considerable amount of time. With a pack capacity of $18Ah$ it would take 18 hours to discharge the entire battery once at $1A$, and at over $400V$ this means at least $400W$ power dissipation the entire duration of the discharge cycles. This power dissipation as well as high currents and voltages would need constant monitoring the entire duration to ensure safe operation.

7 Reflection

Through the implementation of the BMS, most requirements from 2.1 System Requirements have been met. Most notably the requirements that haven't been met completely are:

Communicate with 16 LTC6804-1 Chips - Initially only a single slave was tested containing two LTC6804-1 BMS chips, however more slaves were manufactured nearing the end of the project and running two slaves with four chips total was successful. Further testing must happen to verify the system can be daisy-chained to all sixteen BMS chips.

Predict SOC from current & OCV data - The estimation of SOC has not been verified to work in the physical battery over charge and discharge cycles. However it has been verified using the MATLAB Simulink with generated current data.

Communicate with ECU As all testing has happened outside the vehicle, no communications have been established between the BMS and the vehicle's ECU. Although, the system has been designed in a way to allow for easy additions to interconnectivity between devices so, once the system is in place, it will be a quick and painless task.

The addition of shaders was unforeseen at the beginning of the project. These has a steep learning curve to understand how to implement them in the companion utility. However, the intuitive visualisations of temperature, voltage and charts make the program much easier to use and meets the at-a-glance understanding of many data points.

One of the biggest challenges for the project was building a serial monitor to parse all the CAN bus messages. This system had to be very high speed with a CAN bus of $1Mbit/s$ without halting the application meaning it had to run on a separate thread. It also had to very reliable and never miss a message or misinterpret the information contained within the CAN bus message.

Finally, another part of the project that has been implemented to a very high standard was the configuration protocol. This is the exchange of BMS parameters from the companion utility to the BMS and vice versa. This system is very robust, constantly making sure every piece of information is transmitted and received. It also has an indication when power or communications are lost during the sequence at various points to let the user know why and when something went wrong.

8 Conclusion

This project has gone through the entire development of the software for implementing a BMS on an EV's HV battery and lays the groundwork for future development of the system within the USM team. The BMS solves many of the problems that this project identified about the previously bought unit, the 'Orion 2', reducing the overall size and weight, reducing the costs of implementation, reducing the barrier to entry for designing a new BMS, allowing for more intelligent interconnectivity in the car as well as providing true HV isolation when the battery segments disconnected from each other. The software for the embedded system side of the BMS as well as the companion utility have been verified to work by testing them against each other ensuring communications between them are expected and using comprehensive monitoring of the state of the BMS with configurability to test certain aspects in particular scenarios. The BMS has been designed according to the FS specifications because the USM electric vehicle will go on to compete in the international FS competitions. All software has been designed in a flexible and configurable way to allow the software to be compatible with new revisions of the physical battery because the USM team will use this for years to come.

Bibliography

[1] "DHAB S/134 — DHAB V2 — Open loop Hall effect." https://www.lem.com/en/product-list/dhab-s134.

[2] "Linduino/LTC68041.cpp at master · analogdevicesinc/Linduino." https://github.com/analogdevicesinc/Linduino.

[3] A. Devices, "LTC6804-1/LTC6804-2 – Multicell Battery Monitors - Datasheet." https://www.analog.com/media/en/technical-documentation/data-sheets/680412fc.pdf.

[4] Z. Liu and X. Dang, "A New Method for State of Charge and Capacity Estimation of Lithium-Ion Battery Based on Dual Strong Tracking Adaptive H Infinity Filter," *Mathematical Problems in Engineering*, vol. 2018, p. e5218205, Sept. 2018.

[5] G. Plett, "Extended Kalman filtering for battery management systems of LiPB-based HEV battery packsPart 1. Background," *Journal of Power Sources*, June 2004.

[6] "Battery Management Systems – Part 1: Battery Modeling." https://www.engineering.com/story/battery-management-systems-part-1-battery-modeling.

[7] S. Kosch, Y. Zhao, J. Sturm, J. Schuster, G. Mulder, E. Ayerbe, and A. Jossen, "A Computationally Efficient Multi-Scale Model for Lithium-Ion Cells," *Journal of The Electrochemical Society*, vol. 165, p. A2374, Aug. 2018.

[8] Y. Shi, G. Prasad, Z. Shen, and C. D. Rahn, "Discretization methods for battery systems modeling," in *Proceedings of the 2011 American Control Conference*, pp. 356–361, June 2011.

[9] M.-K. Tran, M. Mathew, S. Janhunen, S. Panchal, K. Raahemifar, R. Fraser, and M. Fowler, "A comprehensive equivalent circuit model for lithium-ion batteries, incorporating the effects of state of health, state of charge, and temperature on model parameters," *Journal of Energy Storage*, vol. 43, p. 103252, Nov. 2021.

[10] M.-K. Tran and M. Fowler, "Sensor Fault Detection and Isolation for Degrading Lithium-Ion Batteries in Electric Vehicles Using Parameter Estimation with Recursive Least Squares," *Batteries*, vol. 6, p. 1, Dec. 2019.

A CAN Messages

A.1 Transmit

Table A.1: Cell Data Message: Contains information on each cell, looping through each cell ID every message.

ID	0x1B36
DLC	8
BYTE 1	Cell ID. uint16 Big Endian. MSB indicates cell balancing.
BYTE 2	
BYTE 3	Cell Voltage. uint16 Big Endian. Units are $100uV$.
BYTE 4	
BYTE 5	Cell Impedance. uint16 Big Endian. Units are $100u\Omega$.
BYTE 6	
BYTE 7	
BYTE 8	

Table A.2: Cell Bounds Message: Contains information on the extreme cells in the pack.

ID	0x1B37
DLC	8
BYTE 1	Highest Voltage Cell ID. uint16 Big Endian.
BYTE 2	
BYTE 3	Highest Cell Voltage. uint16 Big Endian. Units are $100uV$.
BYTE 4	
BYTE 5	Lowest Voltage Cell ID. uint16 Big Endian.
BYTE 6	
BYTE 7	Lowest Cell Voltage. uint16 Big Endian. Units are $100uV$.
BYTE 8	

Table A.3: Primary Temperature Message: Contains information on the individual temperatures on battery segments, looping through each temperature ID.

ID	0x1B72
DLC	8
BYTE 1	Thermistor ID. uint16 Big Endian.
BYTE 2	
BYTE 3	Thermistor Temperature. uint16 Big Endian. Units are $10mK$.
BYTE 4	
BYTE 5	Highest Battery Temperature. uint16 Big Endian. Units are $10mK$.
BYTE 6	
BYTE 7	Lowest Battery Temperature. uint16 Big Endian. Units are $10mK$
BYTE 8	

Table A.4: Temperature Segment Bounds Message: Contains information on the extreme temperatures per battery segments, looping through each segment number.

ID	0x1B73
DLC	8
BYTE 1	Segment Number. Zero indexing. uint8.
BYTE 2	Highest Temperature on Segment. uint16 Big Endian. Units are $10mK$.
BYTE 3	
BYTE 4	Lowest Temperature on Segment. uint16 Big Endian. Units are $10mK$.
BYTE 5	
BYTE 6	
BYTE 7	
BYTE 8	

Table A.5: Fan Speed Message: Contains information on each fan, looping through each fan ID.

ID	0x1B80
DLC	8
BYTE 1	Fan ID. Zero indexing. uint8.
BYTE 2	Fan RPM. uint16 Big Endian
BYTE 3	
BYTE 4	Fan Duty Cycle. uint8. Units are 1%
BYTE 5	
BYTE 6	
BYTE 7	
BYTE 8	

Table A.6: Keep-Alive Message: Sends periodically every 1s to indicate an established connection.

ID	0x1B00
DLC	8
BYTE 1	
BYTE 2	
BYTE 3	
BYTE 4	
BYTE 5	
BYTE 6	
BYTE 7	
BYTE 8	

Table A.7: Fault Codes A & B Message: Contains the fault flags and fatality of fault categories A & B.

ID	0x1B16
DLC	8
BYTE 1	Faults A. uint16 Big Endian.
BYTE 2	
BYTE 3	Faults A Fatality. uint16 Big Endian.
BYTE 4	
BYTE 5	Faults B. uint16 Big Endian.
BYTE 6	
BYTE 7	Faults B Fatality. uint16 Big Endian.
BYTE 8	

Table A.8: Fault Codes C & D Message: Contains the fault flags and fatality of fault categories C & D.

ID	0x1B17
DLC	8
BYTE 1	Faults C. uint16 Big Endian.
BYTE 2	
BYTE 3	Faults C Fatality. uint16 Big Endian.
BYTE 4	
BYTE 5	Faults D. uint16 Big Endian.
BYTE 6	
BYTE 7	Faults D Fatality. uint16 Big Endian.
BYTE 8	

Table A.9: Configuration Message: Contains the message sequence for configuring the BMS.

ID	0xA12
DLC	8
BYTE 1	Message type. uint8. '0' for ACK, '1' for Data, '2' for complete, '3' for error.
BYTE 2	Next message ID. uint16 Big Endian. '0xFF' if final byte has been received.
BYTE 3	
BYTE 4	
BYTE 5	
BYTE 6	
BYTE 7	
BYTE 8	

A.2 Receive

Table A.10: Auxiliary Battery Temperature Message: Contains information of extra temperatures within the battery.

ID	0xA12
DLC	8
BYTE 1	Temperature ID. uint8. Zero indexing.
BYTE 2	Temperature. uint16 Big Endian. Units are $10mK$.
BYTE 3	
BYTE 4	
BYTE 5	
BYTE 6	
BYTE 7	
BYTE 8	

Table A.11: Timer Message: Contains information on timers in the battery.

ID	0xA12
DLC	8
BYTE 1	Timer ID. uint8. Zero indexing.
BYTE 2	Timer time. uint32 Big Endian. Units are μS.
BYTE 3	
BYTE 4	
BYTE 5	
BYTE 6	Count. uint8.
BYTE 7	
BYTE 8	

Table A.12: Companion Utility Keep-Alive Message: Sends periodically every 1s to indicate an established connection.

ID	0x1C00
DLC	8
BYTE 1	
BYTE 2	
BYTE 3	
BYTE 4	
BYTE 5	
BYTE 6	
BYTE 7	
BYTE 8	

Table A.13: Companion Utility Configuration Message: Contains the message sequence for configuring the BMS.

ID	0xA12
DLC	8
BYTE 1	Message type. uint8. '0' for ACK, '1' for Data, '2' for complete, '3' for error.
BYTE 2	Next message ID. uint16 Big Endian. '0xFF' if final byte has been received.
BYTE 3	
BYTE 4	Data word length in bytes. uint8.
BYTE 5	Data. uint32 Big Endian.
BYTE 6	
BYTE 7	
BYTE 8	