Queensland University of Technology

A Project for
EGH456: Embedded Systems

ELECTRIC VEHICLE PROJECT

Daniel Padbury (n10813934)
Zachary Edwards (n10778209)
Harry Leach (n11039639)
Dawn Wilkinson (n10520660)

October 27, 2025

Contents

1 Introduction

1.1 Problem Statement

The Australian Automobile Association's (AAA) EV Index shows that sales of Battery Electric Vehicles (BEVs) in Queensland rose 177.05% from 2022 to 2023[7]. As illustrated in Figure 1, this trend is not limited to Queensland, by which approximately 14 million BEVs were registered globally in 2023, bringing their total number on the roads to 40 million[8]. The prominence of such BEVs is largely due to affordability, fuel efficiency, and environmental benefits. Failure of organisations to shift their focus towards BEVs risks the viability of their operations from an environmental and socio-economic lens.

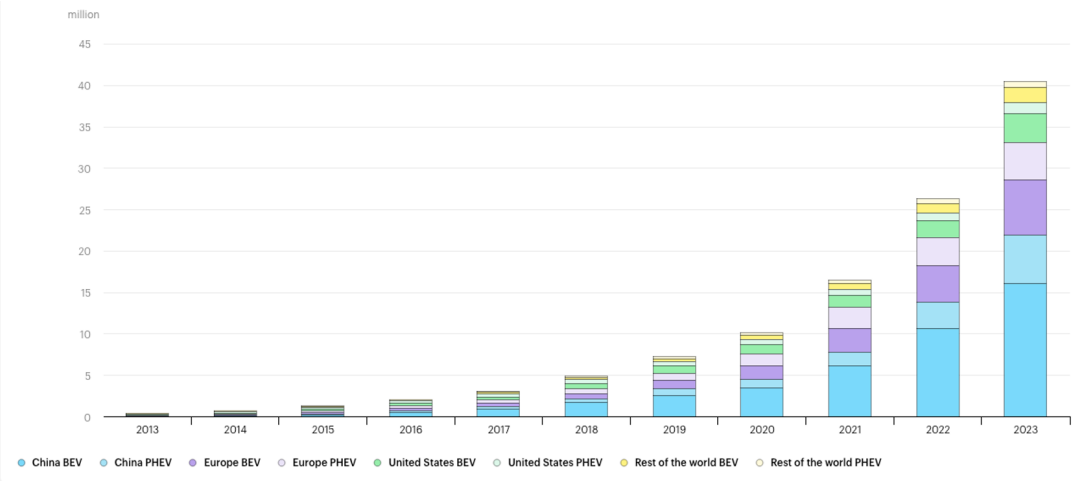


Figure 1: Shift to Electric Vehicles[8]

1.2 Context

To exploit the potential of BEVs, an important role is played by embedded systems. Responsible for the integration of hardware and software to monitor and control various aspects of the vehicle, a reliable embedded system is critical to ensure the safety, comfort and well-being of passengers and road users. Accordingly, the primary objective of this paper is to develop an embedded software system that ensures the safe monitoring and control of a simplified BEV, inclusive of the sensing and actuation of the 3-phase Brushless DC (BLDC) motor.

1.3 Requirements

The embedded system can be divided into three overarching sub-systems, namely, real-time sensing, motor control, and a graphical user interface (GUI), each with a range of unique requirements. Expanding on this, the motor control sub-system will be responsible for monitoring and managing the motor state, which includes its rotational velocity, acceleration and power. The motor control will also integrate start-up, braking, and emergency procedures. The real-time sensing integrates four sensors imperative for the safety of the vehicle, namely, speed, power, vehicle body acceleration and light levels. To ensure the reliability of the data, the embedded system will utilise a range of sampling and filtering techniques. Finally, the GUI has been developed to allow users to control and monitor the electric vehicle's operation, providing clear visual feedback of the vehicle's state.

To integrate the subsystems effectively, a real-time operating system is imperative. The kernel used for this system is FreeRTOS. FreeRTOS provides tasks, queues, and semaphore objects enabling various functionalities, and inter-thread communication protocols[5]. The embedded system is programmed on Texas Instrument's Tiva TM4C1294NCPDT microcontroller. It interfaces with a custom-designed motor testing kit, that allows engineers to simply call functions that control the motor.

1.4 System Architecture

To achieve the before-mentioned requirements, Figure 2 below illustrates the system architecture. The sub-components of the system architecture are documented in more detail throughout the paper and are organised as follows. Section 2 Outlines the Design and Implementation of each subsection. Section 3 outlines an analysis and evaluation of the effectiveness of each sub-component. Finally, the paper is concluded by a final remark in Section 4.

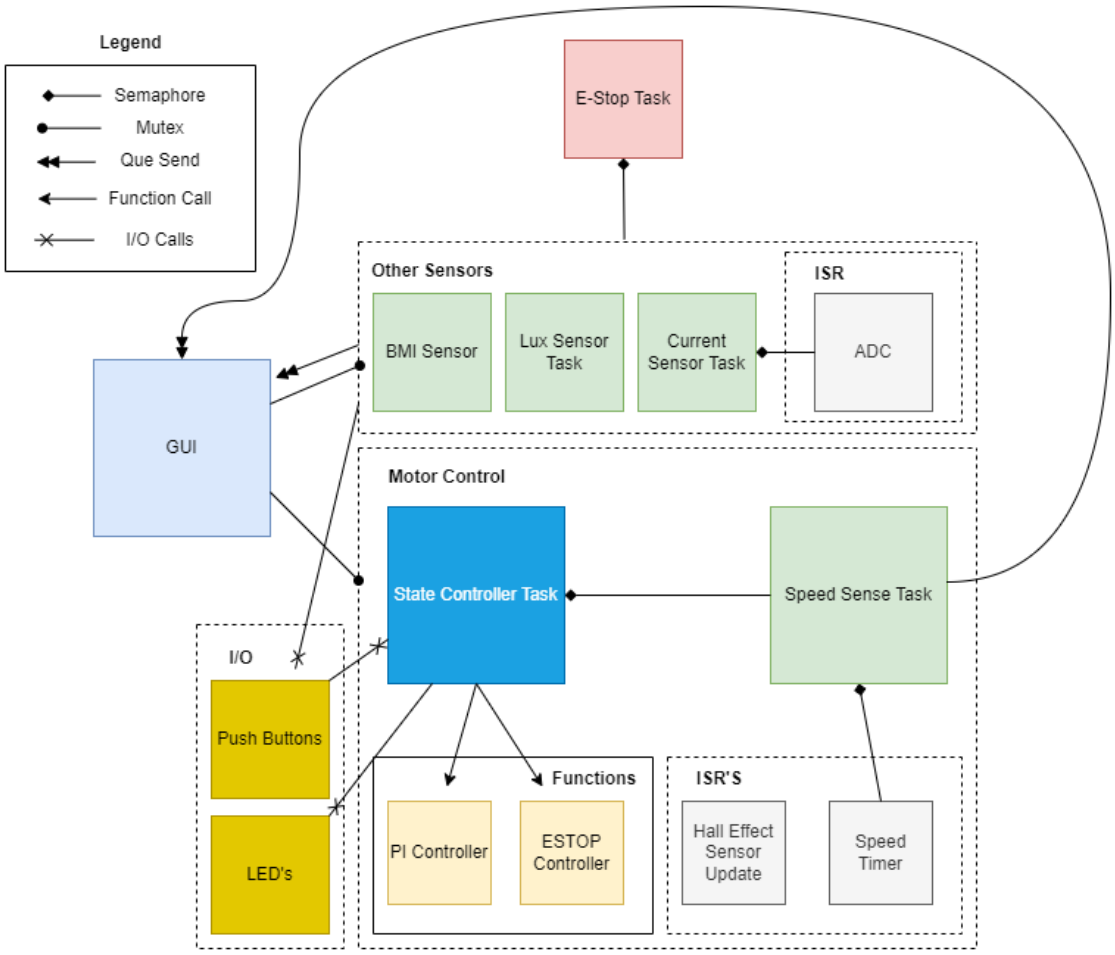


Figure 2: System Architecture Diagram

1.5 Statement of Contributions

We agree that all group members made a valuable contribution and therefore believe it is fair that each member receive the same grade for the assignment.

Student No.	Name	Contribution	Signature
n10813934	Daniel Padbury	Accelererometer (BMI160) Sensing, Light (OPT3001) Sensing, and GUI (plotting)	D.Padbury
n10778209	Zachary Edwards	Motor Control and Speed Sensing	Z.Edwards
n11039639	Harry Leach	Speed Sensing and GUI	H.Leach
n10520660	Dawn Wilkinson	Current Sensing, E-stop conditioning	D.Wilkinson

2 Design and Implementation

2.1 Motor Control

2.1.1 Overview

The embedded system featured a three-phase Maxon EC motor equipped with three Hall effect sensors, enabling precise phase state detection. Motor control was achieved using the pre-compiled motor driver library, motorlib. This driver handled reading the Hall effect sensors, defining the PWM period, and driving the motor through a PWM duty cycle. A FreeRTOS task was employed to manage the motor's state using a state machine. Figure (2) illustrates how this task communicated with various other tasks to gather essential sensor data, such as RPM, process the E-stop command, and receive speed commands from the GUI. A proportional integral controller was used to control the RPM of the motor while it was running and a separate other controller was used for E-stopping so that there was more control of the RPM limits for each state.

2.1.2 State Machine

The motor state machine consisted of four states: IDLE, STARTING, RUNNING and E-STOPPING. Figure (3) displays a flowchart of the transitions between each of the states.

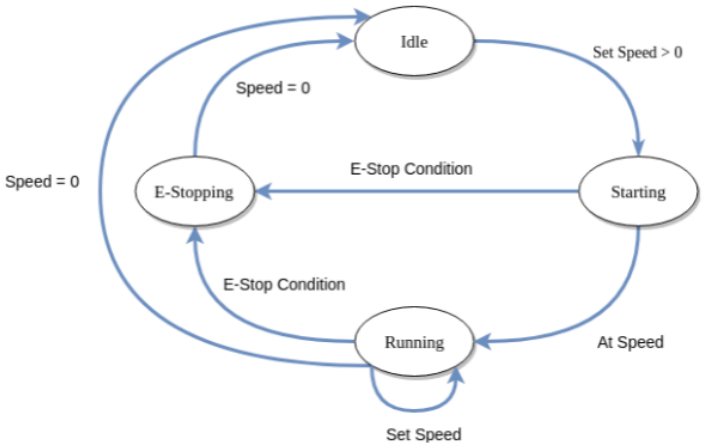


Figure 3: Motor State Machine Flowchart

Idle State

The idle state was the entry point of the task and exists to just wait on user input from push button one to start the motor at which point the state changes to STARTING.

Starting State

The starting state was responsible for initialising the motor with the hall effect sensors, enabling the motor and then bringing it up to a hard-coded safe RPM of 1400. Then the state would automatically be changed to running. The state could be changed to E-STOPPING should an E-stop be triggered by any of the sensors.

Running State

The running state managed the closed control loop, where the proportional-integral (PI) controller received the RPM input and output the next duty cycle value. Additionally, the running state handled motor stopping: if a stop command was issued via the push button, the task would use the PI controller to decelerate the motor to 1400 RPM before disabling it and transitioning to the IDLE state.

E-Stopping State

The E-stop state was triggered by a dedicated high priority E-stop task, which waited on the

semaphore, xESTOPSemaphore. This semaphore was signalled by any of the sensor tasks when the sensor value exceeded the user-defined E-stop threshold. At this point the motor task would activate the E-stop controller that slows the motor at the required rate of -1000 RPM/s.

2.1.3 Proportional-Integral Controller

Equation (1) describes the controller used by the motor task to control the RPM.

$$v = u + K_p e_p + K_i e_i \tag{1}$$

Where u is the current RPM, e_p is the error in RPM, e_i is the integral of the error in RPM, K_p is the proportional gain, K_i is the integral gain and v is the output RPM. The error was capped at 500 RPM and the integral error was capped at 100 RPM to stop the controller from exceeding the acceleration limit of 500 RPM/s.

The controller waited on a semaphore, xControllerSemaphore, from the speed task which allowed for the control loop to run. Otherwise, the motor task would idle and wait, this meant that the motor's control loop was dependent on the execution speed of the speed task.

2.1.4 Motor Characteristics

The specification required motor speed to be set using RPM, while the motor driver operated with PWM duty cycles. Therefore, a method to convert the desired RPM to the correct duty cycle was necessary. It was decided to have the PI controller operate solely on RPM, with the output subsequently converted to the appropriate duty cycle.

To establish the conversion equation, 10 tests were conducted at 10% duty cycle increments, measuring the corresponding RPM for each increment. The results were graphed, and a polynomial equation was derived to best fit the data. The resulting equation, shown in Figure (4), provided the polynomial relationship used to convert the PI controller's RPM output to the next duty cycle in the control loop.

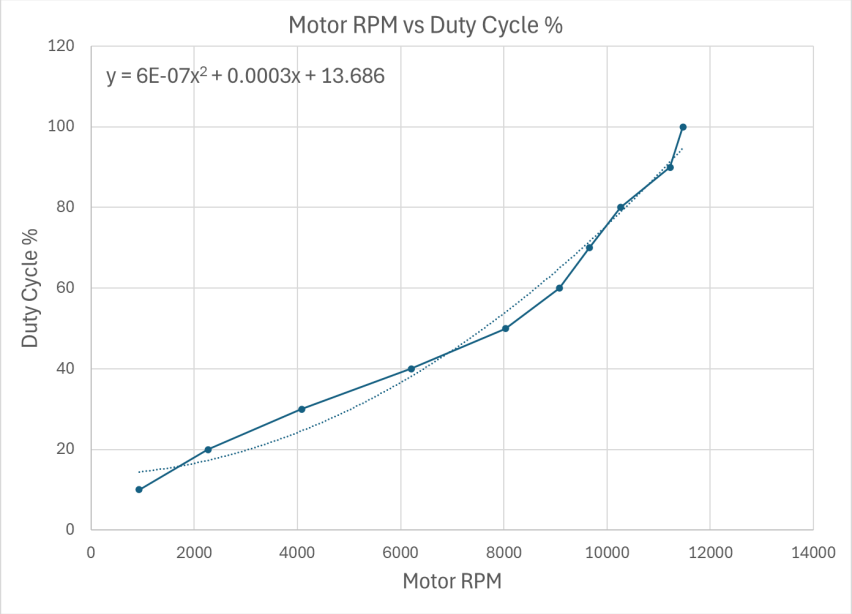


Figure 4: Motor Characteristic of RPM vs PWM Duty Cycle

2.1.5 E-Stop Controller

The E-Stop controller was defined as a simple function that decelerated the motor at a set rate of -1000 RPM/s. This meant that in each control loop the error was a fixed -1000 RPM, this continued until the duty dropped below the threshold of 14 at which point the motor was disabled.

2.2 Sensors

2.2.1 Overview

A sensor driver has been integrated into the embedded system to oversee critical information about the electric vehicle. As illustrated in Figure 5, the sensor driver is composed of four individual sensors, each tasked to measure critical components of the vehicle state, namely, vehicle body acceleration (m/s^2), light levels lux), speed (RPM) and motor power (W). Through a range of inter-thread communication protocols, the before-mentioned sensors interface with the embedded system to provide real-time data, enabling the system to make informed decisions and adjustments for optimal performance and safety.

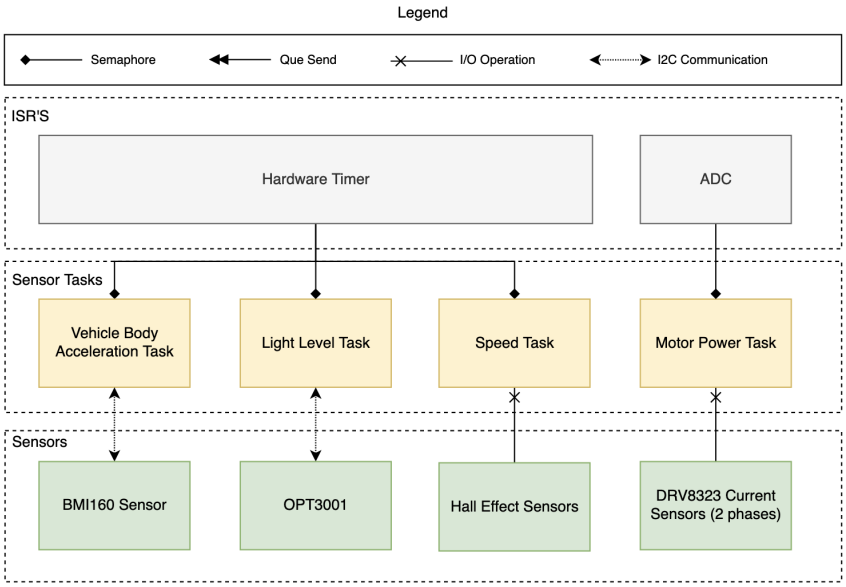


Figure 5: Electric Vehicle Sensor Driver Architecture

2.2.2 Sensor Software Structure

The software structure is standardised for each sensor driver to maintain a uniform sensor reading mechanism across the embedded system. Specifically, as illustrated in Figure 6, this consists of an ISR routine that activates the respective sensor tasks, prompting the associated task to perform a sensor reading, data filtration, E-Stop threshold check and data send via communication. The following subsections outline the sensor-specific software protocols in more depth.

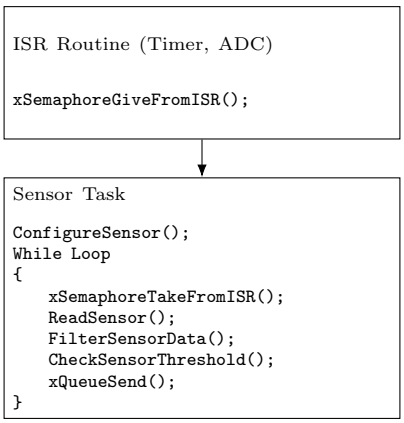


Figure 6: Overview of the general sensor software structure

2.2.3 Vehicle Body Acceleration (BMI160 Sensor)

The BMI160 sensor low noise 16-bit inertial measurement unit IMU is capable of measuring acceleration in 3-dimensional space (x, y, z. Accordingly, the Vehicle Body Acceleration task utilises the BMI160 sensor to determine the absolute acceleration acting on the vehicle, providing information critical to detecting a sudden crash event. This is achieved by calculating the magnitude of the acceleration vector, $|A_{xyz}|$:

$$|A_{xyz}| = \sqrt{\ddot{x}^2 + \ddot{y}^2 + \ddot{z}^2} \tag{2}$$

To obtain the acceleration sensor reading, two external software drivers were utilised, namely, TivaWare^{TM} Peripheral Driver Library[3] and Bosch Sensortec BMI160 Driver (revision 2.0.5). Specifically, the TivaWare's *i2c.c* Driver Library[3] was utilised to communicate with the Sensor via I2C, which involved initialising the I2C base to I2C2 coupled with handling the overhead of I2C read/write transactions. Conversely, the Bosch Sensortec driver was used to configure the BMI160, setting the output data rate to 400Hz, disabling the under-sampling flag, setting the acceleration range to 4G and filter bandwidth to normal mode. This configuration aligns with the recommended data processing settings from the BMI160 datasheet[1].

2.2.4 Light Level (OPT3001 Sensor)

The OPT3001 sensor is a single-chip lux meter, which measures the intensity of light as visible by the human eye. Similarly to the BMI160 Driver, to obtain the light level from the sensor, the TivaWare^{TM} Peripheral Driver Library[3] was used to support the I2C transaction between the TM4C1294NCPDT Microcontroller and the OPT3001 Sensor. Coupled with this, the Bosch Sensortec OPT3001 Driver (revision 2.0.5) was used to support the configuration, transaction, and conversion of data[2].

2.2.5 Speed (Hall Effects Sensors)

The DC motor's 3-phase design relied on three hall effect sensors, H1, H2, and H3. These sensors were connected to GPIO pins M3, H2, and N2 respectively. By calling an ISR after each pin read, the motor's state was updated, and a *motor state change* counter was incremented. With the assistance of the state diagram provided by the reference document, it was found that 12 state changes equated to one revolution. With this knowledge, speed was obtained by counting the number of ticks taken between the speed tasks, converting this into time (in seconds), and then calculating how many revolutions occurred by dividing the *motor state change* count by 12. By dividing the number of revolutions by the time taken, the speed was obtained in Revolutions Per Second (RPS). This was then multiplied by 60 to obtain Revolutions Per Minute (RPM). The *motor state change* counter was reset to 0 after the speed was calculated.

2.2.6 Power (DRV8323 Current Sensors)

The motor controller employed three analogue voltage sensors that each measured the voltage across some shunt resistors. These shunt resistors lie upon the voltage rail of each phase. Using the equation provided by TI[6] the current of a phase can be calculated for any given time.

$$I = \frac{\frac{V_{VREF}}{2} - V_{SOX}}{G_{SCA} * T_{SENSE}} \tag{3}$$

This can be used to derive the current of two of the motor's phases at any given time. Unfortunately, phase A cannot be measured. Due to the fact that on the BOOSTXL-DRV8323RH, only two of the three analogue voltage pins fall on an ADC channel. Figure 7 shows on row B1 that the ISENC, ISENB, ISENA are located on pins 6-8 respectively. These are the voltage pins for each phase. It can be seen that only phase C and B are connected to analogue channels, AIN0 an AIN4.

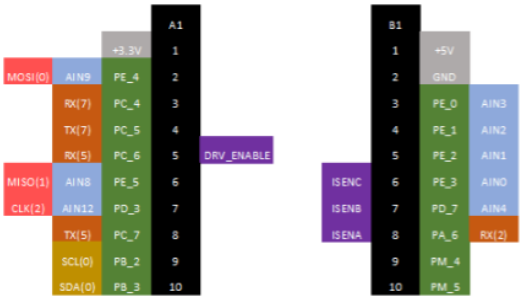


Figure 7: BOOSTXL-DRV8323RH pinout

To account for this, phase A must be estimated in code. To do this, a few properties of 3 phase systems, like this three-phase motor, can be used. Firstly, because each phase is a sin wave and they are all 120 degrees out of phase with one another, the sum of the voltage or current of all three phases at any point in time is zero

$$A(t) + B(t) + C(t) = 0$$

This can be re-arranged to give an expression in terms of phase A

$$A(t) = -(B(t) + C(t))$$

With this estimate of phase A voltage, and using equation 3, total current can be derived and so can power using the nominal voltage of 20v of the motor.

2.2.7 Data Filtration and Sampling

To ensure accurate fault detection and prevent false alarm and minimise the impacts of noise, accurate data filtering and sampling is imperative. Accordingly, in line with Client expectations, the following data filtration and sampling rates were used:

- Accelerometer (BMI160): Data sampled at 100Hz, with a buffer size 6.
- Light (OPT3001): Data sampled at 2Hz, with a buffer size 6.
- Current (DRV8323): Data sampled at 150Hz, with a buffer size 20.
- Speed (Hall Sensors): Data sampled at 100Hz, with a buffer size 6.

Current sensing was given a bigger filter size due to the fact that it used the potentially noisy inputs to estimate another input to derive the output, the noise would be redoubled.

2.3 Graphical User Interface

2.3.1 Overview

A Graphical User Interface (GUI) was developed that allowed users to control and monitor the electric vehicle's operation. The GUI was simple to use and provided operators with clear visual feedback on the vehicle's state. The TivaWare^{TM} graphics library, *grlib*[4], was crucial for the GUI's development. The library provided inbuilt functions for widgets, canvases, and sliders, as well as radio and on-screen buttons. These were used in a variety of ways to create the following GUI components. For example, on-screen buttons were used to navigate between page one and page two.

2.3.2 Initial Design

At the beginning of the project, a mockup design was envisioned and drafted. This provided a blueprint for the intended GUI design and was highly beneficial during development.

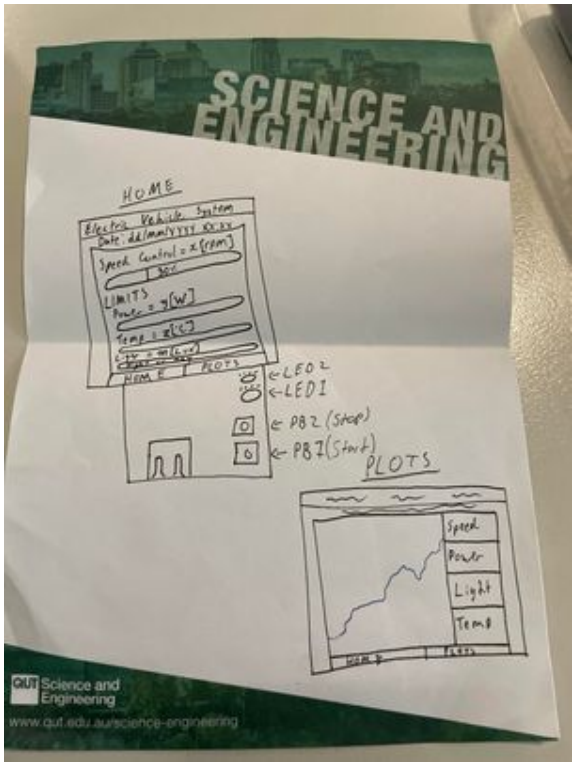


Figure 8: Initial GUI Design

2.3.3 Motor Control

The system used the inbuilt push button 1 to start the motor. This sent the state controller from IDLE to START, where the motor performed its first hall sensor update and then transitioned to the RUNNING state. In the RUNNING state, LED 1 is ON. Whilst RUNNING, the motor's speed was controlled using a slider which ranged from 0 to 12000 RPM. The speed control slider was initiated at 50% which was equivalent to 6000 RPM. By pressing push button 2, a stopping flag within the code would be set to true. As a result, the motor would be given a set speed of 0 and would return to the IDLE state. Consequently, LED 1 is set off. The push button inputs were debounced with a 200ms delay.

2.3.4 Date & Time

The GUI had a canvas widget at the heading of the first page to display the date and time. The time used a counter to track system ticks and would then re-display the canvas after a minute. The date was static. A genuine implementation was attempted which used the Real-Time Clock's (RTC) Calendar mode to display the date and time. Due to time constraints, this was unsuccessful.

2.3.5 Threshold Control

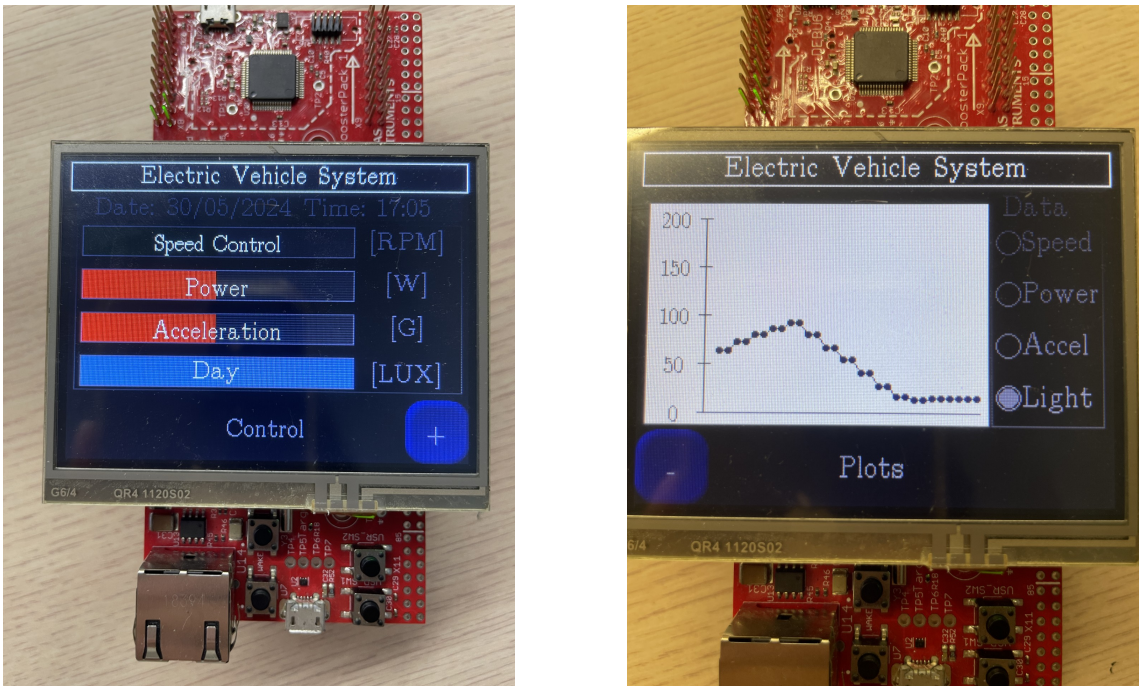
E-STOP conditions required that upper limit thresholds be set for the acceleration and power signals. Similar to the speed control, these thresholds were set using the TivaWare^{TM} slider object and started at 50%. Acceleration ranged from 0.0 to 2.0 G, whilst power ranged from 0 to 100 W. If the filtered data exceeded its respective limit, the system would trigger an E-STOP, shutting down the motor and returning it to an IDLE state.

2.3.6 Day/Night Indicator

The lux signal obtained by the light sensor was used to indicate whether or not the system was experiencing day or night. If the filtered lux data was above a set limit of 5, the system would consider this as *Day*. Otherwise, if the lux was below this, the system would experience *Night* and LED 2 was set to ON. A rectangular canvas object displayed this status. The colour choice (light blue for day, dark blue for night) was a conscious decision to assist with users' visual understanding of the system.

2.3.7 Plotting

The GUI's second page was dedicated to plotting. Upon navigating to page two, the user is given the option to select one of four radio buttons. These radio buttons were provided by the TivaWare^{TM} *grlib* library and represented which stream of data was used for plotting. The screen updated at a rate of 2 Hz to display data. Refer to Figure 9a and Figure 9b below an illustration of the GUI Control and Plotting functionality respectively.



(a) GUI Control

(b) GUI Plots

Figure 9: Visualisation of GUI Functionality

3 Results

3.1 Motor Control Results

The motor control component of the embedded system exhibited two significant issues but generally performed well. The proportional-integral (PI) controller successfully managed the motor, allowing the user to set its RPM to the desired speed. However, it encountered a problem with excessive acceleration.

As briefly discussed in section 2.1.3, the motor control task relied on a semaphore from the speed task to initiate the control loop. This reliance would not have been problematic if the control loop had included a time-step component. Due to the absence of a time step component, the acceleration could not be properly regulated, making the motor's acceleration almost entirely dependent on the sample rate of the speed task. This critical flaw prevented the motor from staying below the acceleration limit of 500 RPM/s and maintaining a constant emergency stop (E-stop) deceleration of 1000 RPM/s.

A potential solution would have been to incorporate a time component that recorded the duration between each control loop. This approach would have enabled acceleration control by restricting the RPM change to a maximum permissible value for each time step.

3.2 Sensor Results

3.2.1 Structure

The initial structure design stayed almost fully intact across development. Some minor adjustments were made to the queuing system, different queues for power and acceleration were used due to their data being floats instead of integers.

3.2.2 Vehicle Body Acceleration (BMI160)

The vehicle body acceleration sensor worked well. It successfully sensed acceleration in all three directions and averaged these into one value using equation 2. The final average acceleration was then converted into Gs, the earth's gravitational acceleration unit at sea level. This converted value was then filtered as per the filtering guidelines and the data was sent to the associated GUI queue.

The only problem encountered was the E-stop condition. Initially, it seemed the accelerometer threshold was never triggering the E-stop condition. It was found the threshold for power and acceleration were using the same variable so some structure of the E-stop condition had to be changed to account for the different thresholds of the power and acceleration measures. This resulted in a bug that caused the accelerometer to always trigger the E-stop condition, no matter how large the threshold was or how little acceleration was applied to the sensor. This problem was never resolved and as a result, the E-stop of the accelerate was never included in the final revision of the code.

3.2.3 Light Level (OPT3001)

Light sensing had few faults throughout development. One of the only problems encountered was when integrating with the plotting. Due to the slow 2Hz sample rate of the lux sensor, the plotting function was trying to access new data twice as fast as the lux sensor provided due to the filtering specifications chosen for the lux sensor. This was fixed on the GUI side and can be read about in section 3.3.4.

3.2.4 Speed (Hall Effect Sensors)

The speed sensing worked sufficiently. It was easily derived using the hall effect sensor data and the observations made in section 2.2.5. The raw data was successfully converted into RPM and was put through the filter as per the filtering specifications.

The speed sensing module also ran double duty calculating the motor acceleration to allow for proper motor control. This was not easily achieved. The end acceleration results were inaccurate and sporadic. This was never fully resolved. Improvements had been made by using filtering similar to speed sensing, but this did not eliminate the issue. This inaccurate acceleration data caused problems with the motor control, specifically in accelerating at the desired 500 RPM/s. This is explored in more depth in section 3.1.

3.2.5 Power (DRV8283 Current Sensors)

The current sensor had multiple points of potential error in its measurement. Firstly the ADC measurement and subsequent conversion to real voltage worked perfectly. Using a small test bed it could accurately measure voltage between 0 to 3.3 volts. The conversion from voltage to current and then from current to power could not be accurately tested on this test bed, instead the actual motor kit was required.

Upon testing it was noted that the two measured phases always gave a voltage of around 1.65 volts, this was slightly unexpected but did not change how anything operated or was calculated, yet. Further testing showed the motor sat at around 20 watts of power. This number slowly crept up and down a couple of watts due to possible noise despite the large filtering. This was eventually determined to be because of the three-phase estimation used. Another estimation type was tested, just using the average of the two other phases, this was possible due to the aforementioned constant voltages measured. This *seemed* to produce less noise and so was ultimately used in the final build.

When load was applied to the motor, it noticeably increased in power to about 30 watts before too much load was used and the motor stalled. When no in the running state the motor still reads some power, this is because equation 3 does not account for zero power going through the motor, so an additional check for if the state machine is in the RUNNING state was added which gave a linear ramp up and down of power when the state is changed from RUNNING to IDLE and vice versa.

Finally, the power E-stop condition worked as intended. When the e-stop power was set in the region of 25-30 and load was applied to the motor it triggered the E-stop condition to stop the motor. There was an initial bug where it would always hit the E-stop condition, but this was fixed with the introduction of the state machine and better filtering to reduce noise.

3.2.6 Temperature (MLX90615)

Initially, the MLX90615 temperature sensor was chosen as the additional sensor over the BMI160, but it was later found to not work due to its incompatibility with the FreeRTOS system. Accordingly, the I2C driver module was unable to communicate with the sensor device. Despite a range of testing via the bus module of the oscilloscope, which displayed the data transfer between the MCU (Master) and the sensor (Slave), it was apparent the Master was unable to send the requested by bytes to the Slave. As a result, a new sensor was required to meet the requirements of the Client. This had a range of adverse effects on the Embedded system due to the significant time and resources spent debugging the MLX90615 sensor to no avail (approximately 15 hours).

3.3 GUI Results

The GUI was capable of performing all specified actions, however, some system features did not meet the complete functionality requirements.

3.3.1 Motor Control

The motor control was able to receive the desired speed command from the user. This was proven by simply setting the speed and observing the motor adjust to meet this velocity.

3.3.2 Date & Time

The date and time were sufficiently displayed. In light of this, the RTC would be a powerful alternative for the system's next iteration. The hibernation feature would keep track of the system's time even after losing power.

3.3.3 Threshold Control

The threshold setting was also successful. The data sensing tasks received the threshold value, and if the condition was satisfied, the ESTOP event would occur. This was tested by setting the power threshold at a reasonably low mark and increasing the speed control. As a result, the motor demanded more power and surpassed its upper limit, sending the system into ESTOP mode.

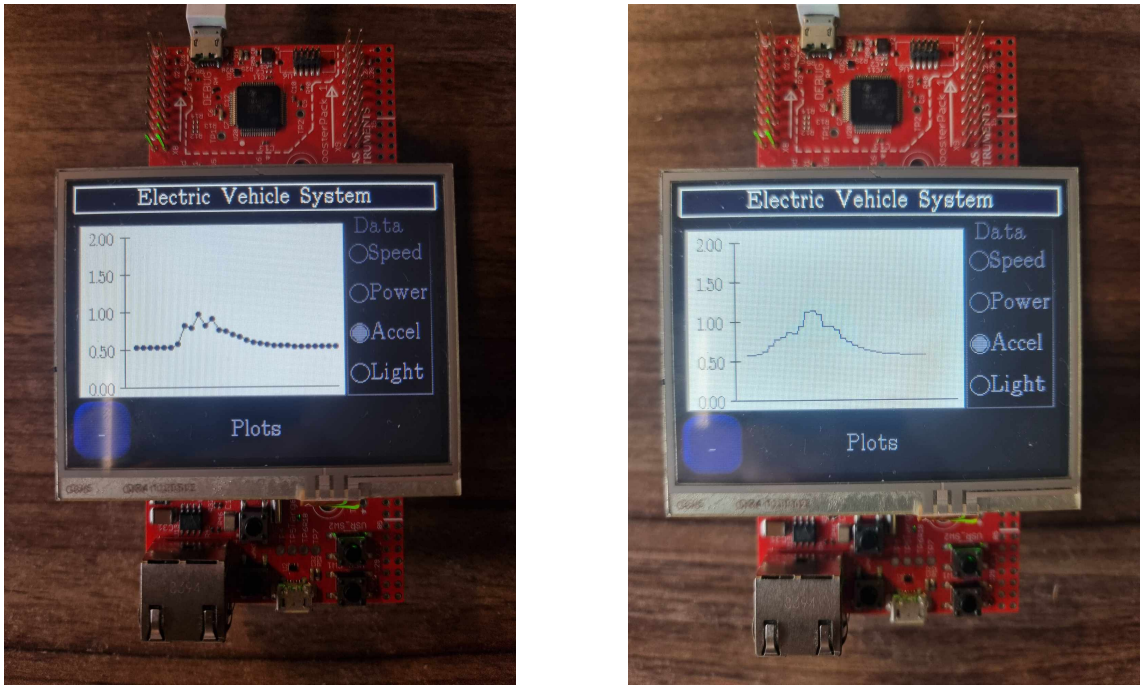
3.3.4 Day/Night Indicator

The Day/Night indicator met the specification requirements. This was assessed by shining a light on the OPT3001 sensor, for the lux readings to increase. As expected, LED 2 turned OFF and the GUI canvas changed to display *Day*, see Figure 9b. When the light was turned off, the lux readings decreased below the set limit of 5, resulting in the canvas displaying Night and LED 2 turning ON.

One challenge faced was seemingly random data displayed on the lux graph. This issue was caused by a bug where the GUI task updated twice as fast as the lux sensor task. As a result, when attempting to access previous values in the lux sensor's array, it retrieved incorrect data. The root cause was that the previous data array for the lux task wasn't shared between the two tasks, leading the display tasks to pull from random memory locations. This was resolved by declaring the array at a global scope, and including communication protocols to share the data safety between the two tasks. This solution resolved the issue, and the sensor worked as expected.

3.3.5 Plotting

The plotting aspect of the GUI managed to display all four data streams to provide sound motor status. Unfortunately, the system suffered from the 2Hz display rate (Figure 10a). This is because the screen's refresh rate wasn't fast enough to capture all the data sent by the sensors. Consequently, the plots lost a substantial amount of detail. To amend this, the refresh rate should've been set at the highest sensor rate - this was 150Hz (Figure 10b). By decreasing the x-axis time step, and removing the dots in place of lines, the GUI would plot the data with much finer detail. The impact of the choice of display rate is illustrated below in Figure 10 This is valuable learning that will greatly improve any future systems.



(a) Plot at 2Hz Display Rate

(b) Plot at 150Hz Display Rate

Figure 10: Impact of Display Rate on Data Plots

4 Conclusion

The project integrated real-time sensing, motor control, and a graphical user interface (GUI) to ensure safe and efficient operation of the 3-phase Brushless DC (BLDC) motor.

Despite challenges such as excessive motor acceleration due to the lack of a time step component in the control loop, the project met its primary objectives. The proportional-integral controller effectively managed the motor's RPM, and various sensors provided crucial data for system safety and reliability.

The GUI facilitated user interaction, offering clear feedback and control options. The use of FreeRTOS and the Texas Instrument's Tiva TM4C1294NCPDT microcontroller effectively managed the system's tasks and communication.

This project underscores the importance of embedded systems in electric vehicles, highlighting the need for robust hardware and software integration to enhance performance and safety. Future improvements could focus on refining the control loop to better regulate motor acceleration. Overall, the project sets a solid foundation for advancements in embedded systems for BEVs.

5 Reference Table

Reference Number	Document Name	Sub-Section Number	Topic Keywords	Pages Useful
1	BMI160 Datasheet	2.2.3	Accelerometer, BMI160	15, 18-20
2	OPT3001 Sensor Datasheet	2.2.4	Light, lux, OPT3001	10, 16-19
3	TivaWare™ Peripheral Driver Library	2	I2C, ADC	323-324 (I2C), 21 (ADC)
4	TivaWare™ Graphics Library User Guide	2.3	Widget, Canvas, Push & Radio Button, Slider	85-97, 199-237
5	FreeRTOS Operating System	2	FreeRTOS, Real Time, Kernel, Operating System, Multi-thread	22-155 (task), 157-206 (ques), 208-249 (semaphore), 253-296 (software timer), 298-320 (events)

References

[1] BOSCH. "BMI160. Small, low power inertial measurement unit." mouser.com. Accessed: May. 22, 2024. [Online.] Available: https://www.mouser.com/datasheet/2/783/BST-BMI160-DS000-1509569.pdf.

[2] Texas Instruments. "OPT3001 Ambient Light Sensor (ALS)." ti.com. Accessed: May. 17, 2024. [Online.] Available: https://www.ti.com/lit/ds/symlink/opt3001.pdf.

[3] Texas Instruments. "TivaWare™ Peripheral Driver Library." ti.com. Accessed: May. 01, 2024. [Online.] Available: https://www.ti.com/lit/ug/spmu298e/spmu298e.pdf.

[4] Texas Instruments. "TivaWare™ Graphics Library User Guide." ti.com. Accessed: May. 17, 2024. [Online.] Available: https://www.ti.com/lit/ug/spmu300e/spmu300e.pdf.

[5] Amazon Web Services. "FreeRTOS™." freertos.org. Accessed: May. 04, 2024. [Online.] Available: https://freertos.org/index.html.

[6] Texas Instruments. "DRV832x 6 to 60-V Three-Phase Smart Gate Driver" ti.com. Accessed: May. 17, 2024. [Online.] Available: https://www.ti.com/lit/ds/slvsdj3d/slvsdj3d.pdf.

[7] RACQ. "Shift to electric vehicles gathers pace" racq.com. Accessed: June. 01, 2024. [Online.] Available: https://www.racq.com.au/articles/evs/2024/1/shift-to-electric-vehicles-gathers-pace.

[8] International Energy Agency. "Trends in electric cars" iea.org. Accessed: June. 01, 2024. [Online.] Available: https://www.iea.org/reports/global-ev-outlook-2024/trends-in-electric-cars.