

Embedded System for Electric Vehicle

EGH456 Embedded Systems

Group Assignment

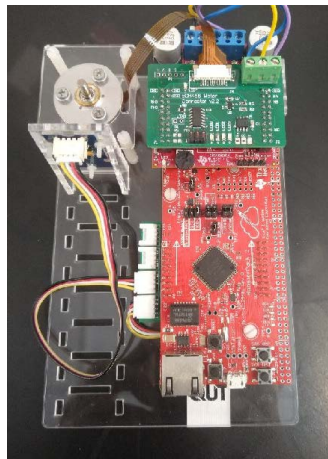
The specifications in this document are subject to minor changes if further clarification is needed

1. Problem Description

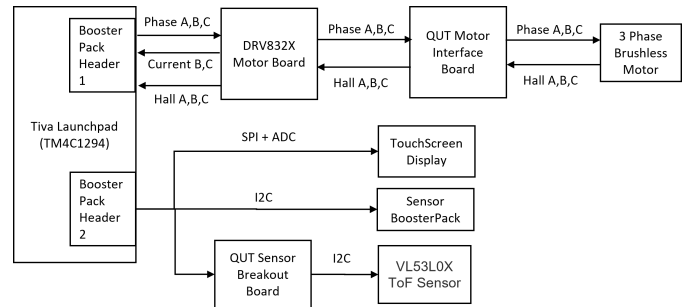
Battery-electric and hybrid-electric vehicles are gaining popularity due to their increasing affordability, fuel efficiency, and environmental benefits. The automotive industry is shifting towards electric vehicle design, with recent advancements in autonomous electric vehicles further fueling this transition. As an embedded system engineer, you have been contracted by an automotive manufacturer transitioning to battery-electric technology. Your task is to design the real-time sensing, motor control, and user interface software for the embedded system of an electric vehicle.

Your primary objective is to develop embedded software that ensures the safe monitoring and control of the electric vehicle, including the sensing and actuation of the 3-phase Brushless DC (BLDC) motor. This will involve monitoring the motor's state, including its rotational velocity, power and managing its start-up, braking, and emergency procedures. To accomplish this, your system design must be capable of handling multiple real-time tasks, such as sensor acquisition and filtering, motor fault handling, and the display of critical system information. The development kit for the Tiva TM4C1294NCPDT microcontroller and a motor testing kit, which includes the motor driver and sensor boards, will be provided for this assignment. Figure 1c illustrates the setup interface of the testing station. The inputs and outputs of the sensors and per-phase connections are compatible with the microcontroller ports, and an electrical interface will be used to achieve this compatibility.

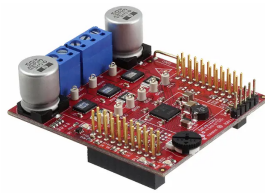
A separate reference document will provide additional information relevant to this task, including the mapping of the microcontroller pins (GPIO, I2C, ADC, UART) and the motor driver inputs and outputs. Additionally, you will receive resources such as a motor operation description, a custom motor driver library (MotorLib), and a motor kit setup guide to help you complete the task successfully.



(a)



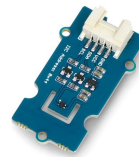
(b)



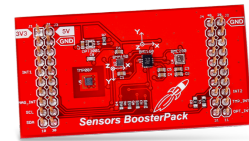
(c)



(d)



(e)



(f)

Figure 1: (a) Simulated Electric Vehicle Sensor and Motor Testing Station. The system includes a microcontroller development kit, a motor driver and sensor board, brushless motor with hall effect sensors for position and velocity feedback, IMU, distance and current sensors for fault monitoring and environment monitoring including Light and Temperature. (b) a simplified block diagram of the hardware used within this assignment. (c) BOOSTXL-DRV8323RH Motor Control Launchpad Boosterpack. (d) Digital I2C Time of Flight distance sensor. (e) Digital I2C Ambient temperature and Humidity sensor. (f) BOOSTXL-Sensor breakout board including an IMU (accelerometer and gyroscope), Magnetometer (compass) and Light Sensor. Please see the additional assignment reference document for a more detailed pinout of the Boosterpack headers

2. Requirements

The assignment can be divided into three primary sections: Motor Control, Sensing, and User Interface, each with its unique set of requirements, which are described below. Your task is to design the system using a modular software driver model for each subsystem and utilise a Real-Time Operating System (RTOS), such as FreeRTOS, to coordinate and manage real-time events in the vehicle. The RTOS should be used to schedule tasks, handle concurrency, and ensure responsiveness in time-critical functions such as motor control, sensor data acquisition, and user interface updates. The final software will integrate each subsystem's driver functionalities into one coherent solution for demonstration. To accomplish this, your team must collaborate to design the software system, managing events, shared resources, and CPU utilisation. It's critical to prioritise each subsystem and its tasks while doing so.

2.1 Motor Control

Your task for this section is to develop a device driver for the DRV832x motor driver board, along with the additional custom motor interface and protection board. Note that the additional protection board has been included to shut off power to the brushless motor if too much current is being supplied to the motor. This is indicated by a solid red LED on the board and you must power off the system to turn off the protection mechanism. See the reference documentation for more details on this protection board. The goal of the motor control subsystem is to create an Application Programming Interface (API) that provides the following functionalities:

Your API for the DRV832x motor driver board should include the following functionalities:

1. Start and safely control the speed of the motor, handling any fault conditions, by using the provided state transition diagram (see reference document). The API should control the phases of the motor, as described in the supporting documents using the provided MotorLib library.
2. Control the motor's speed in Revolutions Per Minute (RPM), based on user input.
3. Safely start and accelerate the motor to the desired speed (in RPM), following the motor acceleration specifications provided below. Your API should ensure that the motor accelerates safely.
4. Safely decelerate the motor, adhering to the motor deceleration specifications provided below.

Your API should ensure that the motor slows down and stops safely.

To emulate the driver of a vehicle pressing the foot pedal, the motor must be controlled to match a user-modified desired speed or RPM. This requires measuring the motor's speed and adjusting the power to the motor (PWM duty cycle) to match the desired speed. However, the actual motor speed may fluctuate depending on the motor load, and is not constant for a constant PWM duty value. Therefore, it's essential to measure the motor's speed to control its speed accurately (i.e., using a closed-loop controller). We do not expect you to perfectly tune this controller for this assignment and a control error (difference between desired and actual motor speed) that fluctuates between 0-20% is acceptable for this assignment (not likely in the real system)

For safe vehicle (motor) starting and braking, the system must regulate the motor's speed according to acceleration and deceleration limits. Please refer to the separate reference document for the speed control diagram. Additionally, you will need to explore how to start a 3-phase brushless motor correctly. To help set up the GPIO pins and switch the brushless motor's phases correctly, a third-party motor driver library (MotorLib) will be provided.

The system will also require different deceleration limits based on whether an emergency condition has occurred or the user has inputted a slower desired speed. The acceleration and deceleration limits are defined as follows:

1. Motor Acceleration $\leq 500RPM/s$
2. Motor Deceleration (setting slower speed) $\leq 500RPM/s$
3. Motor Deceleration (emergency stop aka e-stop) $= 1000RPM/s$

Please ensure you handle units correctly, as acceleration and deceleration values will be tested. Note, the deceleration for e-stop should be equal to (not greater or less than) the desired value as we would like the vehicle to come to a complete stop quickly but safely.

2.1.1 E-stop Conditions

The system should respond appropriately when certain sensor thresholds indicate an emergency condition for the vehicle. The following conditions should trigger an emergency stop (e-stop) for the motor control, causing the motor to safely brake using the previously indicated deceleration limit. Please note that you should only implement vehicle body acceleration, and distance conditions if you have selected the sensor in the relevant section 2.2.

1. Total motor current (from all three phases) has exceeded a user-defined threshold, indicating the motor has a fault
2. Absolute vehicle body acceleration measured from the IMU (not motor acceleration) has exceeded a user-defined threshold indicating an impact has occurred (see section 2.2).
3. Distance has gone below a user-defined threshold (the vehicle is about to hit an obstacle) measured via the Time of Flight distance sensor (see section 2.2).

2.1.2 Motor Debugging Tool - Serial Plot

As part of your motor control subsystem you will need to send debugging information via UART to a serial plotting tool of your choice such as [SerialPlot](#). SerialPlot is a free, open-source serial plotting tool that allows you to view real-time signals over UART. This is to visualise and validate performance of your motor control, such as maintaining deceleration and acceleration limits.

You are required to send actual vs desired RPM and the current PWM duty cycle applied to the motor via UART to the serial plotting tool. Additional signals can also be sent if these improve the validation of your motor control subsystem. Your implementation must output these motor control parameters at a regular interval at a rate fast enough to clearly show the timing of your motor control system (i.e acceleration of motor is $\leq 500RPM/s$ etc...). you should also use this tool to identify control issues like overshoot or steady state error in your controller and should be used to provide evidence of your working motor control subsystem for your final report.

2.1.3 Motor Library and Motor Kits

To help your group easily and safely control the phases of the motor, a custom motor library will be provided. You must use this library to minimise the risk of damaging the motor kits supplied with this assignment. Please refer to the separate instructions provided on the unit website to install and use the library.

However, using the custom motor library does not guarantee that the motors will not be damaged. Therefore, we recommend monitoring the motor kits during operation and, when debugging your software while operating the motors, either power down the motor kits or monitor them to prevent significant heating. The teaching team has added some protection mechanisms, such as the motor interface board to minimise the risk of damage, but it's almost impossible to protect against all possible failures with electrical and mechanical hardware.

The QUT motor interface board sitting on top of the motor driver board is monitoring the motor power continuously and will disable the use of the motor driver by indicating a solid red light after a few seconds of high power detection. If this is the case, you will be required to power off the motor driver to reset the protection mechanism.

2.2 Sensing

To monitor critical information about the state of the vehicle, you'll need to develop device drivers for the various sensors available on the vehicle. Two essential sensors are required, namely Power Sensing and Speed Sensing. These sensor drivers should provide an Application Programming Interface (API) that can perform the following functionalities:

1. **Motor Power:** The Power Sensing driver is responsible for accurately reading and filtering the two motor phase currents using the analogue signals provided by the current sensors on the DRV8323 board. It's worth noting that only two out of the three phases are available for ADC measurement, so you will need to estimate the total power consumption based on the two measured phases. To calculate the power usage of the motor, you should use the measured motor phase currents and nominal motor voltage of 24V. It is crucial to filter the current readings to eliminate any noise and ensure accurate power measurements. You might need to investigate timing of your current measurements and synchronising them with motor phase changes (motor update calls to the library) to improve filtering performance (as voltage spikes can occur at these times).
2. **Speed Sensing:** The Speed Sensing driver should measure and filter the motor's speed in Revolutions Per Minute (RPM) for the system to match a desired speed/RPM modified by the user interface. Speed-sensing can be achieved by measuring the time between the edges of the motor's hall effect lines (Hall A, B & C). **Note:** As speed sensing is measured using hall effect sensors (i.e. GPIO interrupts) the rate of sensing can be dependent on the speed of the motor. Therefore, it is recommended to utilize a separate timer that is monitoring the hall effect sensor counts to achieve consistent speed sensing.
3. **Light Level (OPT3001):** The Light Level sensor driver should be able to read and filter the light level data from the OPT3001 ambient light sensor over an I2C connection. The sensor measures the illuminance in lux and provides a digital output. This sensor will provide input to detect day or night light levels. **Note: When using OPT3001 and Touchscreen**

on the same boosterpack header the OPT INT gpio pin cannot be used.

Choose 1 out of the 3 following sensors to add to your project:

1. **Vehicle body acceleration (BMI160 on Sensor Boosterpack):** The vehicle body acceleration sensor driver should be designed to read and filter acceleration data on all three axes of the vehicle. This information can be obtained from the BMI160 sensor. The driver should then calculate the average of the absolute acceleration, which is the sum of the absolute values of the acceleration data in each of the three axes. This functionality is crucial in detecting a sudden crash event.
2. **Car Temperature and Humidity Environment Sensor (SHT31):** The temperature sensor provides ambient temperature and humidity readings of the cars cabin. The temperature and humidity sensor driver should be designed to read and filter the car cabin temperature and humidity from the SHT31 sensor (connected to the QUT breakout board). It is required to report the car cabin temperature in celsius and humidity in Relative Humidity (RH).
3. **Distance (VL53L0X):** Note this can be a challenging sensor to integrate due to some of its initialisation requirements. The distance sensor driver should be designed to read and filter distance measurements from the Time of Flight (ToF) distance sensor through an I2C connection. The ToF distance sensor measures the distance to an object by emitting a laser pulse and measuring the time it takes for the pulse to bounce back to the sensor. The device driver should include appropriate filtering to ensure accurate distance measurements to an object. This information will determine whether an obstacle is too close to the vehicle. Note that this sensor can be challenging to interface with without utilising other 3rd party libraries which are allowed to be used within this assignment.

2.2.1 Sensor Filtering

Sensor data is often noisy and requires appropriate filtering to be used reliably for control and safety decisions. Therefore a requirement of the sensor subsystem is to implement appropriate noise filters for all sensors implemented. All filtering must occur in threads, not ISRs. You are required to select and implement an appropriate digital filter type (e.g., low-pass, band-pass, or moving average) that reduces measurement noise and improves signal reliability. Your filter design

should be tailored to the expected signal characteristics of each sensor type and must be justified in your report.

Once selected, apply the chosen filter method to the raw sensor data ensuring the filtered signal responds in real time and avoids aliasing or excessive lag. Filter implementation must be performed outside of sensor ISR (e.g., within RTOS threads).

Minimum sampling specs as follows:

- Power (DRV8323) – 150 Hz or higher.
- Speed (Hall Sensors) – 100 Hz or higher
- Light (OPT3001) – rate of 2 Hz or higher.
- Accelerometer (BMI160) - the data from each axis must be sampled at a rate of 100 Hz or higher.
- Temperature and Humidity (SHT31) – rate of 1 Hz or higher.
- Distance (VL53L0X) – 20 Hz or higher.

It is important to note that the filtering of sensor data must not be performed in an Interrupt Service Routine (ISR). For instance, when capturing current samples, it is advisable to take one sample every 6.67ms (150Hz) via the ISR and maintain a sample buffer. You should then filter the buffer outside of the ISR using a sensing thread. There are various filtering techniques available such as a low pass, sliding window or exponential weighting method (examples can be found at [sliding window](#)). If you determine that a higher sampling frequency would enhance the performance of the filter, then it is acceptable to increase these specifications.

2.2.2 Sensor Debugging Tool - Serial Plot

To support development and validation of your sensor subsystem, you must transmit real-time sensor data via UART to a serial plotting tool such as [SerialPlot](#). SerialPlot is a free, open-source utility that enables you to view streaming data in real time, helping to verify sensor behavior and filtering effectiveness.

You must send raw and filtered values for each core sensor (motor speed, motor power and light), and for any optional sensors your team selects. Data must be output at a rate that closely matches

the rate captured by the sensors in order to allow for meaningful visualization. This will be used to validate filter performance (raw vs filtered), check sensor timing, and support debugging of threshold-based conditions like e-stop. Include example plots or screenshots in your final report to demonstrate and reflect on your results.

2.3 User Interface

For this section of the project, you have been tasked with developing a Graphical User Interface (GUI) that will allow users to control and monitor the electric vehicle's operation. The GUI must be intuitive, easy to navigate, and provide clear visual feedback. Your design should prioritise ease of use and clarity of information while operating safely with the real-time motor control and sensing subsystems. Apply best practices in user interface design such as consistent layout, readable text, and responsive controls. Include a layout diagram in your final report.

The following features must be implemented in the GUI:

1. Start and stop buttons for motor control (touch or push button integration).
2. LED indicator showing motor status (running, stopped, or e-stop).
3. User input field to adjust desired motor RPM (i.e. slider bar).
4. A real-time clock displaying the current date and time. The date should default to the demonstration day and time.
5. Fields for setting upper and lower threshold values for critical parameters (e.g., max current, vehicle acceleration, temperature). Exceeding these values must trigger an e-stop.
6. A system status panel that clearly displays the current mode (Running, Braking, E-stop) and any other status or warnings.
7. Implement an e-stop acknowledgement button. After an e-stop condition, the GUI must require the user to acknowledge/clear the event before restarting motor operation.
8. Classify and display whether it is day or night (nighttime is defined as ambient light levels below 5 lux). An LED should be turned on when the time of day is classified as nighttime to simulate auto headlights.

Depending on the one selected sensor from the sensing section, include the following additional GUI features:

1. Car Temperature and Humidity: Show current temperature in Celsius. Add limits for heating/cooling with status indication (e.g., “Cooling On”).
2. Vehicle Acceleration levels: Set an upper limit for the vehicle’s acceleration (i.e. vehicle has crashed). If this limit is exceeded, the system should trigger an e-stop and shut the motor down safely.
3. Distance level: Set a lower limit for the distance between the vehicle and any object in front of it. If this limit is exceeded, the system should trigger an e-stop and shut down the motor safely.

2.3.1 GUI Sensor Plots

A second page or tab in the GUI should plot the filtered sensor data from the vehicle over time. The time window on the graph should be greater than 5 seconds. The plots should have the ability to switch between different sensors or toggle them off and on. Units for the sensor data can be of different magnitudes and can be scaled or adjusted for display purposes. The design and structure of this page can be decided by you, as long as it displays the following information in a user-friendly manner. Include a visible legend and appropriate units. Consider including zoom or reset controls. The plot should implement:

1. Motor speed (in rpm).
2. Power usage of the motor (in watts) using phase current measurements.
3. Light levels: Ambient light level (in lux).

Include the additional sensor selected from section 2.2:

1. Car Temperature and Humidity: Current temperature in degrees Celsius and humidity in RH
2. Vehicle Body Acceleration: Average absolute acceleration from the IMU (in meters per second squared).
3. Distance: Distance to the closest object in front of the vehicle (in meters).

2.4 Advanced Features

Students are encouraged to explore and implement advanced features that go beyond the core requirements. This could include integrating more than one of the optional sensors from Section 2.2, implementing advanced control algorithms, additional useful functionalities or developing additional GUI functionality. Any additional features must be documented and demonstrated during the final presentation. While not mandatory, these extensions can contribute to a higher overall grade if the core system requirements have been met to a satisfactory standard.

3. Demonstration

As a group (max group of 4), you will present and demonstrate your work to the teaching team. During the demonstration, you will showcase your work and provide an explanation of your system. You should be prepared to answer questions from the teaching team throughout your presentation. It's worth noting that additional advanced features will only improve your mark if the system design requirements have been met.

Your presentation should aim to showcase your final design to the client, which is the automotive manufacturing company (i.e., the teaching team). You should provide an overview of your design, demonstrate how you have met the specifications, and highlight all relevant details.

Please note that the presentation duration is strictly three minutes per student, so come prepared. Additionally, the demonstration should not exceed 13 minutes. Each student should prepare one PowerPoint slide to accompany their presentation.

4. Suggested Format of the Report

The report should contain the following items. The content and style are flexible if these are separately identifiable. Approximate page guidelines are given in parentheses.

- **Cover page** – names, student ID numbers, course, and unit information. (1 page)
- **Table of Contents** – Section headings, list of figures, list of tables. (1 to 2 pages)
- **Introduction** – Problem statement, context, requirements, and statements on the individual contributions by each team member. Provide a useful introduction and overview and not a clone of the assignment task sheet (1 pages)

- ***Design and Implementation*** – Approach to design, important issues and choices and their relationships to theoretical concepts and the hardware and software platforms (2 pages per section - Motor, Sensors and GUI). Check if you have included:
 - A diagram and description of your RTOS task structure, inter-task communication and synchronisation (e.g., semaphores, queues).
 - Which drivers and modules did you use and how?
 - Did you use external libraries such as the graphics library?
- ***Results*** – Summary of evidence of functional requirements that were demonstrated and explanation of failures as learning outcomes in terms of what could have been done differently. Examples of how you tested your software on the real hardware. (1 page per section - Motor, Sensors and GUI)
- ***References*** – Including a list of vendor-supplied technical documents with a table that links each document to sub-sections of your report where they are relevant with entries: The reference number, the sub-section number, topic keyword or keywords, the pages that were found useful. Include references or links to libraries used within your project. (2 pages)