Zips Racing Electric - Vehicle Control Unit

Senior Project Final Report

Design Team 10

Tetra Engdahl

Brian Glen

Ryan Stoller

John Wozencraft

Faculty Advisor Dr. Farhina Haque

27 April 2025

Table of Contents

List of Figures (JW/RS)

List of Tables (JW/RS)

Abstract

Zips Racing Electric, a design team at the University of Akron, competes in Formula SAE electric car racing competitions. To improve performance, the team is developing a four-wheel drive system, but the current vehicle control unit (VCU) is unable to handle the added complexity. It lacks the necessary sensor inputs, computational power, and control systems. This project focuses on developing a new VCU to monitor the vehicle's real-time state, calculate motor torque and speed values for all four wheels, and manage regenerative braking. The new system will interface with the powertrain through the existing CAN bus and use sensor data, such as driver inputs and accelerometer readings, to control the four-wheel drive system. The project scope includes designing the hardware, building the embedded system, and developing a control system for torque vectoring and regenerative braking. A simulation model based on the Zips Racing vehicle will be used to evaluate and refine the system's performance.

Key Features:

- Reads sensor inputs like driver commands and accelerometer data.
- Calculates torque and speed for four independent electric motors.
- Incorporates regenerative braking.
- Interfaces with the powertrain via CAN bus.
- Evaluated through dynamic vehicle simulations.
- Controls several auxiliary vehicle systems.

Problem Statement

Need Statement (BG)

Zips Racing Electric is a design team at the University of Akron that builds and races electric cars at university competitions. Recently, the team has embarked on the design of a four-wheel drive system to increase vehicle performance[1].

However, the existing computer on the vehicle in its current state cannot adequately control the more complex powertrain; it lacks the sensor inputs, computing power, and control systems necessary to do so[2]. This results in the need for a new solution that can monitor the real-time state of the vehicle, host a control system, and control the four-wheel drive system.

Objective Statement (BG)

The project objective is to build a hardware device and an embedded control system that will serve as the main control unit of the electric vehicle's drive system.

This device will read in various sensors to determine the vehicle's state, such as driver inputs and an accelerometer. Then, a control system will calculate torque and speed values for each of the four electric motors, with the goal of using the electric motors to assist with turning the vehicle. This information will be sent to the powertrain via an existing CAN bus communication.

The scope of the project will involve designing and building an embedded device and designing / developing a control system. The device will interface through a communication bus to an existing powertrain system and motor controller. A new control system for torque vectoring and regenerative braking will be designed, and an existing vehicle simulation model representative of the Zips Racing vehicle will be used to evaluate its dynamic performance.

Design Team Background (JW)

This design team is being sponsored by The University of Akron's Formula SAE electric racing team, Zips Racing Electric, to design and build a new vehicle control unit (VCU). Currently, Zips Racing Electric is implementing a rear-wheel drive system but would like to transition to an in-wheel four-wheel drive system. The reason Zips Racing Electric would like to switch to this four-wheel drive system is because an in-wheel drive system does not need a driveshaft, differential, or transmission attached to the motors and wheels, which will consequently direct more of the motor output power into the wheel directly[3]. Therefore, this project will include redesigning and implementing a new VCU that will incorporate four-wheel drive with torque vectoring. In addition, to increase the power efficiency, a regenerative braking system will be designed and added to the VCU to recoup power used during braking. To do this, one must model the dynamic behavior of the vehicle as a system of input vectors that can be controlled and monitored by the VCU. The VCU will need to interface with Zips Racing Electric's vehicle sensor inputs to produce outputs for the existing power train in real-time. Thus, this VCU will be a real-time embedded control system, which will require research regarding how to properly implement an embedded control system.

Formula SAE Competition (JW)

Formula SAE is an international design competition hosted by the Society of Automotive Engineers (SAE) where student design teams design and prototype a small Formula-style racecar[4]. SAE created and hosted the first Formula SAE race in 1981 and created many spin-offs since then to allow students more ways to show off their creativity and engineering skills across the world. Formula SAE has many learning objectives meant to foster the next generation of engineers, such as developing and preparing technical documentation, team collaboration, project management, budgeting, etc[4]. After the prototype is designed and constructed, the design

team will participate in a Formula SAE event where their prototype is scored based on both static events and dynamic events. Static events are technical events such as the design event, the cost and manufacturing analysis event, and presentation event. While the dynamic events are hands-on racing events such as the acceleration event, skippad event, autocross event, fuel economy event, and endurance event[4]. Approximately half of the competition consists of the presentation, documentation, etc. while the other half of the competition consists of various human-driven races in the Formula car that teams designed. This introduces another element to the competition that requires a team to be well-rounded to win the overall competition[5]. This international design competition encourages students to creatively problem solve while also offering an opportunity to win sponsored awards and participate in networking because there are many large companies involved in the competitions.

Competition Rules (JW)

Formula SAE enforces an extensive set of competition and design rules that will impact the design of the control system. Section T.3 and Section T.4 of the Formula SAE Rules address the rules regarding the braking system of the car, which is within the scope of the project as the control system must supply inputs to the motors that accelerate and decelerate the car[5]. A notable braking rule is T.3.2.4 that specifies constraints on regenerative braking in electric vehicles, which will be integrated into the system. There are many more rule sections for electric vehicles that will influence the design such as EV3.3 stating that the maximum power must not exceed 80kW, voltage must not exceed 600V DC, and the powertrain cannot use regenerative braking when under 5 km/hr. More important sections are T11.8 and T 11.9 that describe the requirements of accelerator pedal position sensors (APPS) for measuring pedal travel and actuations[6].

Team Background (RS)

The University of Akron's Formula SAE electric racing team, Zips Racing Electric, are acting as customers that will be sponsoring/paying for this senior design project. They are requesting this team to build a new vehicle control unit for their Formula car that will include four-wheel drive and regenerative braking. The team has been around since 2018 and has members from several engineering disciplines, including mechanical and electrical students; however, no doctorate students are allowed to be on the team. When joining the team, the new members gain access to the knowledge and experience of other more experienced team members, faculty advisors, and team alumni[7]. This can assist the new members in developing the technical and practical engineering skills needed to succeed in the automotive industry[7]. The team must be able to coordinate with one another on several fronts, including budgeting, scheduling, purchasing equipment, race car design, and race car testing[8]. This experience of working together with a team prepares students for a real job in the automotive or any other industry. Also, team members obtain access to the design center, which allows them to practice implementing theory learned in their classes in a hands-on environment[7]. Every year the team competes against other Formula SAE electric teams across the globe[7]. These competitions are another great opportunity to network and build relationships with potential employers and the teams corporate sponsors[7].

Current Vehicle Platform (BG)

Figure 1: University of Akron ZRE24 at the Formula Hybrid+Electric Competition

The current vehicle design, named ZRE24 (Figure 1), is a rear-wheel drive electric racecar powered by a 374 V lithium-ion battery pack and an Emrax 228 MV permanent-magnet synchronous motor (PMSM). Power is delivered from the electric motor to the wheels through a motorcycle chain drive connected to a Drexler V3 limited slip differential. This powertrain layout is shown in Figure 2. The vehicle produces 107 horsepower and weighs 387 lbs.

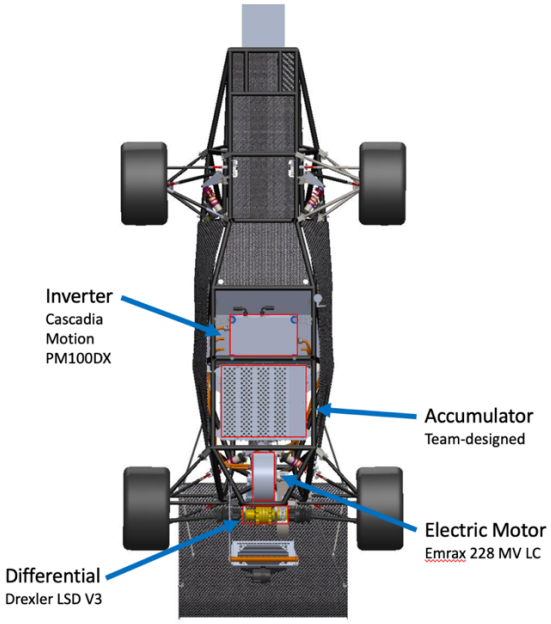

Figure 2: Powertrain layout for ZRE24

The vehicle is controlled by a student-developed vehicle control unit (VCU). The VCU contains a dsPIC33 microcontroller. The function of the VCU is to monitor the vehicle state and handle driver inputs relating to powertrain torque. When the vehicle is energized and in the 'ready-to-drive' state, the VCU reads in signals from the drive-by-wire throttle assembly and sends torque values to the traction inverter. The traction inverter commands the total torque produced by the motor to meet the requested torque value, and the limited slip differential distributes the torque between each rear wheel. Different combinations of ramp angles and preloads inside the LSD allow passive adjustment of this torque distribution[9].

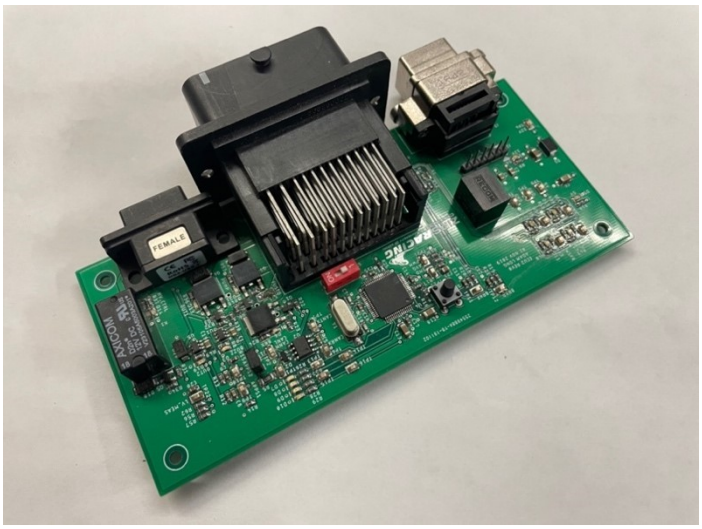

Figure 3: Vehicle Control Unit for ZRE24.

Communication from the VCU to the traction inverter is implemented using CAN bus.

Proposed Vehicle Topology (BG)

For the 2025 season, Zips Racing Electric will merge with its combustion engine-based sister team, Zips Racing, to build a single electric racecar that will compete in the Formula SAE competition in the United States and in Formula Student competitions in Europe. Named ZR25, the vehicle will be a combination of the technological advancements made by both teams. These include a carbon fiber monocoque chassis, active aerodynamic elements, and a powertrain consisting of four in-wheel electric motors. The electric motors are AMK-Motion permanent-magnet synchronous servo motors and are driven by a quad-output inverter (Figure 4), for a total power output of 120 kW[10]. The inverter can be controlled via CAN bus, EtherCAT, or through a direct I/O interface, and allows for the electric motors to either be driven or used as a generator for energy recuperation[10]. The accumulator utilizes a new pouch-type battery cell, which allows for higher charging rates, to take advantage of regenerative braking on all four wheels.

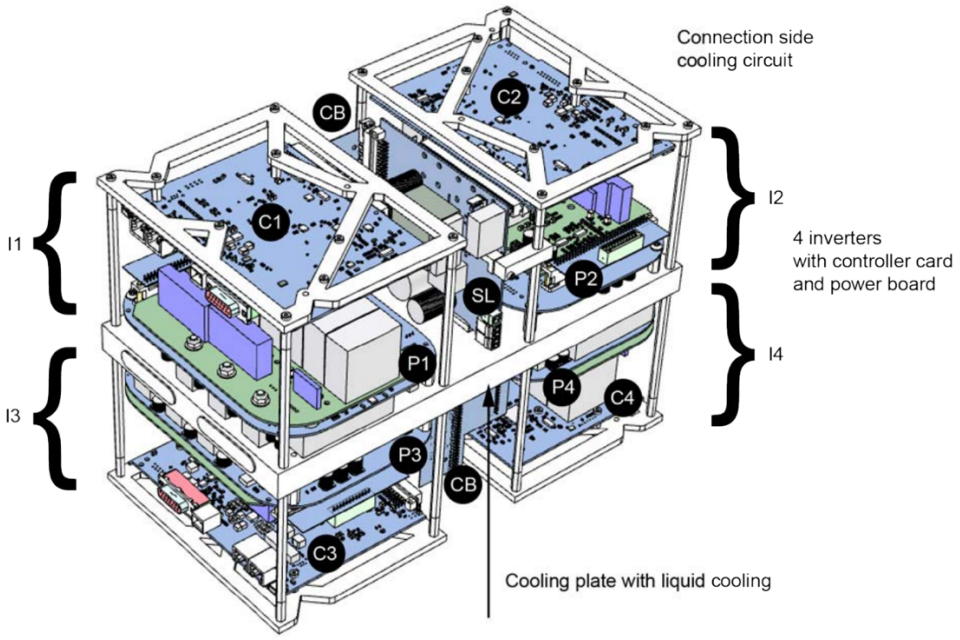

Figure 4: AMK-Motion quad-inverter assembly[11].

Embedded Control Systems (JW)

The root of this project is to design and develop a real-time embedded control system (a VCU) for the operation of the Formula SAE car that Zips Racing is building. Embedded systems are used in various fields from power generation and waste management to healthcare[11]. An embedded control system is “an embedded system dedicated to control in real-time a system or a sub-system"[12]. This means that it is most important to consider the reliability and safety of the control system due to the nature of real-time systems, the competition safety rules, and the fact that there will be human drivers operating the vehicles. While there are a multitude of safety considerations to be made while designing an embedded control system, some important ones for a vehicle would be related to proper power outputs to motors, providing proper grounding, and ensuring safe control of the vehicle[11]. Another important thing to consider when designing this control system is to identify the system that is being controlled. Some important identifications are the available power/voltage, the speed requirements, system input, etc.

Electric Four-Wheel Drive Systems (RS)

A traditional vehicle has one central motor that provides power to the wheels via a driveshaft, clutch, and other mechanical and electrical components. In these traditional vehicles, motor power is transferred through a gearbox, which reduces speed and increases torque[3]. By moving the motor to the wheel, the mechanical complexity of transmitting the motor power to the wheels is reduced (this is because there is no longer a driveshaft, differential, or transmission) and it is now possible to have the wheel angular velocity and motor angular velocity correlate one to one[3]. This is beneficial because it ensures the full output of the motor is directly connected to the wheel without frictional losses in the transmission[3]. As of now, Zips Racing Electric plans to have a planetary gearbox within the wheel; so, while there will be some loss of energy due to friction, it will still be less than a traditional system and transmitting the motor power will still be simpler. Another benefit is that each wheel can be controlled separately, which enables the possibility of torque vectoring. Torque vectoring is also commonly used in vehicles which use differentials to allow some wheels to spin faster than others; in electrical vehicles with in-wheel motors, the response time and efficiency is greater than that of traditional differentials due to the simplicity of the motor being directly attached to the wheel[3]. In-wheel motors also are more friendly to regenerative breaking, especially because during torque vectoring one wheel can brake while the other wheel(s) can continue to drive[3]. This allows for a vehicle's battery to last longer and increases vehicle range. In-wheel motors also lower the center of gravity and help with the distribution weight on the vehicle[3]. However, since these motors are mounted on the wheel, they are subject to additional exposure to vibrations (due to un-damped tire spring rate), salt, water, and other fluids that would not be present if the motor were in a central location like on a traditional vehicle. In-wheel motors are also more expensive

due to the increased complexity of the vehicle motor control and due to the cost of three additional motors[3].

Interfacing With Existing Systems (BG)

One objective of this project is to develop a system that can be integrated with the proposed vehicle. This includes working around the requirements of existing systems and developing interfaces for other features new for this season.

The grounded low voltage (GLV) power system is the chassis-referenced low voltage electronics of the vehicle. Previous Zips Racing vehicles implement this system with a separate low-voltage battery, where the bus voltage may vary between 14.4V and 9V, and the GLV systems including the vehicle control unit must draw as little power as possible so that the low voltage battery can last the entire endurance race. For 2025, this system may stay the same, change to a 24V bus, or be powered by an isolated DC-DC converter that uses the high voltage battery to supply power.

The next vehicle system is the CAN bus communication system. Previous vehicles implemented this using a single CAN2.0b bus operating at 1 Mbps. For 2025, the system will change to CAN2.0a which eliminated the option for extended 29-bit message identifiers, which was not a feature that was used. Baud-rate will stay the same. The vehicle control unit is not at the end of the bus; however, it should have the ability to enable termination resistors if this changes.

Additionally, with torque-vectoring systems relying on feedback from sensors to operate correctly, there is a possibility that multiply CAN buses may be implemented, where vehicle systems of high and low priority may be separated to improve system reliability. The vehicle control unit should have the ability to communicate on multiple buses.

Next is the high voltage battery system known as the accumulator. The accumulator provides all energy for the powertrain system. The accumulator is designed as a fully separate system with a custom battery management system that monitors pack voltage, current limits, cell temperatures, and sensor faults. This system is fully separate from the rest of the vehicle because often the accumulator is charged and monitored outside of the vehicle, where these safety systems still need to work. The vehicle control unit will need the ability to receive messages from the accumulator via CAN bus for power and state-of-charge information, as needed to control the torque vectoring system. For 2025, the battery cell type and layout will change to accommodate the new powertrain requirements, however the battery management and high voltage electronics will be iterations of existing systems.

Related to the accumulator is the shutdown loop. The shutdown loop is a series interlock safety system that runs throughout the vehicle. Inside the accumulator, there are two isolation contactors that when closed allow high voltage to exit the accumulator structure and enter the vehicle-side tractive-systems, which include the powertrain motor inverters, high-voltage disconnects, and discharge circuits. The purpose of the shutdown loop is to directly control the coil current of the isolation contactors, allowing high voltage to exist the accumulator. Although not required by competition rules, previous Zips Racing vehicles have included a relay in series in the shutdown loop on the vehicle control unit, requiring the system to boot and perform a self-test before allowing the vehicle to enter a high-voltage state. For 2025, the VCU should include the hardware to allow this function.

The last system is the physical wiring implementation of the vehicle. High quality wiring of the electrical systems is needed to allow efficient testing and debugging of the vehicle before competition. Zips Racing accomplishes this by requiring electrical systems to use a consistent set

of connector systems throughout the vehicle to reduce the cost of crimping tools. For non-waterproof wire-to-board and wire-to-wire connectors, Molex Micro-Fit+ connectors are used. For waterproof wire-to-wire connectors, Deutsch DTM connectors are used. For waterproof wire-to-wire bulkhead connectors, and critical wire-to-wire connectors, Deutsch Autosport connectors are used. Unlike many automotive connectors, these have high mating cycles which is required for a prototype vehicle undergoing a lot of testing and debugging. For 2025, only the non-water-proof Molex Micro-Fit+ connector series may change due to high insertion and extraction forces. For physical wiring of GLV-systems, a maximum 20 AWG wire size is enforced, and the wire must meet M22759/32 mil-spec standards. The physical board mounting, and enclosure of the vehicle control unit must be fully waterproof and must support the PCB where there are connector headers. No soldering or screw terminals are allowed anywhere in the harness, and PCBs must have a header for wire connections and not be direct-to-wire soldered.

Torque Vectoring Control Algorithms (BG)

Four-wheel drive powertrain systems that use independent motors to control each wheel offer three major benefits to a Formula SAE vehicle:

1. The ability to actively control the yaw response.
2. An increase in efficiency from an increase in regenerative braking capability.
3. Increased traction for longitudinal acceleration.

The most prominent benefit is the active yaw control of the vehicle. This is accomplished via a method called torque vectoring (TV). TV systems actively distribute torque between each wheel to obtain a desired dynamic behavior from the vehicle. One of the most common implementations of torque vectoring is to actively control the vehicle's yaw rate to match a desired yaw rate, typically determined by the driver's steering wheel angle input. This is

accomplished by designing a yaw rate controller[13]. Common control system approaches to yaw rate controllers include proportional integral derivative (PID), linear quadratic regulators, (LQR), or H-infinity methods. More advanced methods include sliding mode control and model predictive control[14].

The yaw rate controller's output is the additional yaw moment required to accelerate the vehicle around its vertical axis to achieve the target yaw rate. With this moment determined, the next objective is to determine how this moment should be generated. This is known as torque distribution optimization[13]. The longitudinal forces F_x for each tire shown in Figure 5 are forces that can be generated by either driving or electrically braking the electric motors at each wheel, or by using the mechanical brakes. The torque distributions must occur such that the desired extra moment is generated, while staying within the dynamic limits of the physical system (such as the traction limit of the tires), the electrical limits of the motors, the limits of the Formula SAE ruleset, and the limits imposed by the accelerator pedal position. The overall power generated by the vehicle must not exceed 80 kW and the system cannot supply more total torque to the wheels than what is requested by the driver at any one time[5].

Because of the complexity in designing these control systems, it is common to validate their performance using simulation software such as VI-Grade or IPG CarMaker. Using these software packages, a multi-body dynamic physics model of the vehicle can be defined. With software such as MATLAB Simulink, the control system can be developed, and then used as an input to the dynamic simulation to validate its control ability and quantify performance improvements.

Marketing Requirements (JW)

1.1.1. The control system must be able to monitor the real-time state of the vehicle.

1.1.2. The control system design must maximize vehicle performance.

1.1.3. The control system must be able to control a four-wheel drive vehicle.

1.1.4. The hardware and software must meet competition rules.

1.1.5. The vehicle must be able to operate safely.

1.1.6. The control system shall interface with all existing vehicle systems and be adequately documented for easy implementation by Zips Racing team members.

Engineering Analysis (RS, BG, JW, TE)

Several of the values discussed in Section 3, Engineering Requirements, needed to be researched to choose the correct value for safe and optimal operation of the vehicle. Additionally, vehicle data analysis and a vehicle model development need to be performed before the engineering design of the control systems could begin.

Electronics

The battery system in the vehicle will need to be kept at a reasonable temperature for both the safety of the driver and car as well as performance and battery durability. The battery is made of several cells arranged in series and parallel. These cells' voltage and temperature are monitored by the BMS. During regenerative braking, the temperature and voltage of these cells increases, thus a cut-off point needs to be set. While setting a temperature cut-off is a simple matter of consulting the rulebooks and data sheets, determining a voltage cut-off requires more analysis. The battery data sheet lists the maximum voltage at 4.28V and the BMS shuts the vehicle down at 4.15V. Thus, an upper voltage limit of 4.05V was set to allow for some overflow. After this limit is hit, the voltage must drop to 3.90V before turning on again to allow for hysteresis and to not toggle the regenerative braking on and off many times successively. Regenerative braking will be on at all other times.

One of the auxiliary systems is a brake light for driver safety. A safe value for the brake light to turn on was deemed to be 5% of the maximum pressure exerted on the brake sensor; this data was taken from the current vehicle. A pressure based on a calibration of maximum driver pedal brake pressure will determine the 5% setpoint.

Vehicle Dynamics (BG)

To implement a torque vectoring control system, the dynamic behavior of the vehicle must first be modeled as a plant. Once modeled, the system can then be analyzed for its response, and a controller developed. The yaw rate ψ of a vehicle is defined as the rotational velocity about the vertical axis of the vehicle, intersecting the vehicle's center of gravity (C.G.)[15].

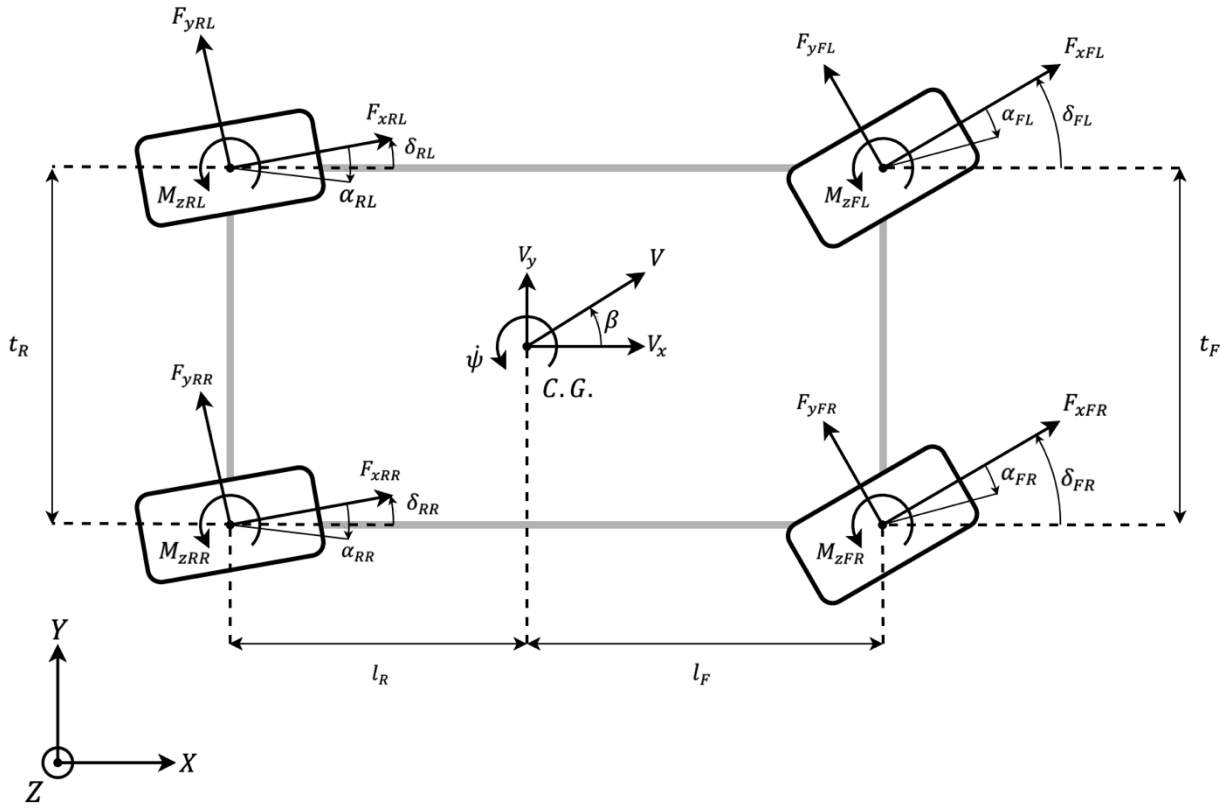

Figure 5: Vehicle forces contributing to the moment around the C.G.

This type of vehicle model, depicted in Figure 5, is known as the double track vehicle model. To change the yaw rate ψ, an angular acceleration must be produced from a moment. This yaw moment $\dot{\psi}I_{zz}$ is generated by the individual forces from each tire acting around the C.G, and from the self-aligning moments around each tire:

$$\ddot{\psi}I_{zz} = [\vec{d}_{FL} \times \vec{F}_{FL} + \vec{d}_{FR} \times \vec{F}_{FR} + \vec{d}_{RL} \times \vec{F}_{RL} + \vec{d}_{RR} \times \vec{F}_{RR}] + \vec{M}_{zFL} + \vec{M}_{zFR} + \vec{M}_{zRL} + \vec{M}_{zRR}$$

The distance vectors from the center of gravity to each tire are defined as:

$$\vec{d}_{FL} = \hat{x}l_F + \hat{y}\frac{t_F}{2}$$

$$\vec{d}_{FR} = \hat{x}l_F - \hat{y}\frac{t_F}{2}$$

$$\vec{d}_{RL} = -\hat{x}l_R + \hat{y}\frac{t_R}{2}$$

$$\vec{d}_{RR} = -\hat{x}l_R - \hat{y}\frac{t_R}{2}$$

The tire force vectors in the vehicle coordinate system are derived from the component forces in the local tire coordinate system:

$$\vec{F}_{FL} = \hat{x}(F_{yFL}\sin\delta_{FL} + F_{xFL}\cos\delta_{FL}) + \hat{y}(F_{yFL}\cos\delta_{FL} + F_{xFL}\sin\delta_{FL})$$

$$\vec{F}_{FR} = \hat{x}(F_{yFR}\sin\delta_{FR} + F_{xFR}\cos\delta_{FR}) + \hat{y}(F_{yFR}\cos\delta_{FR} + F_{xFR}\sin\delta_{FR})$$

$$\vec{F}_{RL} = \hat{x}(F_{yRL}\sin\delta_{RL} + F_{xRL}\cos\delta_{RL}) + \hat{y}(F_{yRL}\cos\delta_{RL} + F_{xRL}\sin\delta_{RL})$$

$$\vec{F}_{RR} = \hat{x}(F_{yRR}\sin\delta_{RR} + F_{xRR}\cos\delta_{RR}) + \hat{y}(F_{yRR}\cos\delta_{RR} + F_{xRR}\sin\delta_{RR})$$

This moment then produces an angular yaw acceleration $\dot{\psi}$ and a change in yaw rate:

$$M_z = \dot{\psi}I_{zz}$$

$$\psi(t) = \int \dot{\psi}(t)dt$$

With I_{zz} being the moment of inertia of the vehicle about the vertical axis. For each tire, the component forces, F_x and F_y and the self-aligning moment, M_z are calculated using a non-linear tire model. The tire models are created empirically by parameterizing test data captured for the specific tires developed for Formula SAE competition[16]. These test datasets are provided by the FSAE Tire Testing Consortium, of which Zips Racing is a member.

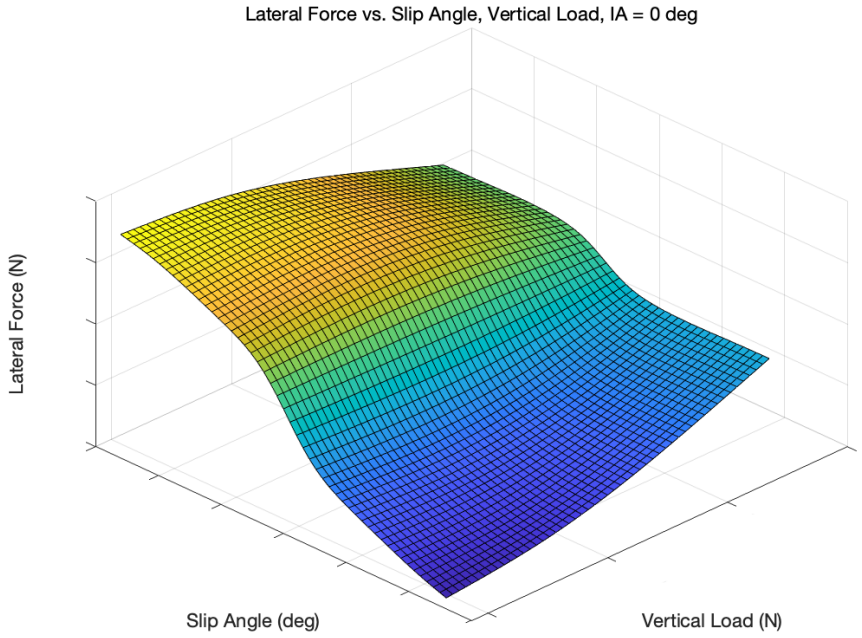

Figure 6: Example of a tire model fit to test data using cubic spline interpolation

The tire forces and moments are modeled as functions of slip angle, normal force, an inclination angle for each tire. To solve the dynamics of the double track model in Figure 5, the slip angles of each tire are calculated. The slip angle of a tire is the angle between the direction the tire is pointing and the direction the tire tread is traveling. It is a function of the steering angles, the vehicle side-slip angle, and the yaw rate of the vehicle:

$$V_y = V_x\beta$$

$$\alpha_{FL} = \frac{V_y+\psi\cdot l_F}{V_x-\frac{\psi\cdot t_F}{2}} - \delta_{FL}$$

$$\alpha_{FR} = \frac{V_y+\psi\cdot l_F}{V_x+\frac{\psi\cdot t_F}{2}} - \delta_{FR}$$

$$\alpha_{RL} = \frac{V_y-\psi\cdot l_R}{V_x-\frac{\psi\cdot t_R}{2}} - \delta_{RL}$$

$$\alpha_{RR} = \frac{V_y - \psi \cdot l_R}{V_x + \frac{\psi \cdot t_R}{2}} - \delta_{RR}$$

The normal force on each tire must also be calculated. A force balance must be solved, where the normal force on each tire determines that tire forces. The tire forces then produce vehicle accelerations, which affect the normal load on each tire (known as load transfer), affecting tire forces, until a steady-state balance is achieved.

Vehicle performance improvement via understeer correction (BG)

A vehicle's performance can be partially defined by its lateral acceleration capability. For the Formula SAE competition, the racetrack design emphasizes low speed cornering capability. In Figure 7, a simulation sweep of a simple vehicle model over a typical FSAE endurance track shows that an increase in the lateral acceleration capability is approximately 3 times more effective at reducing lap time compared to an increase in longitudinal acceleration capability.

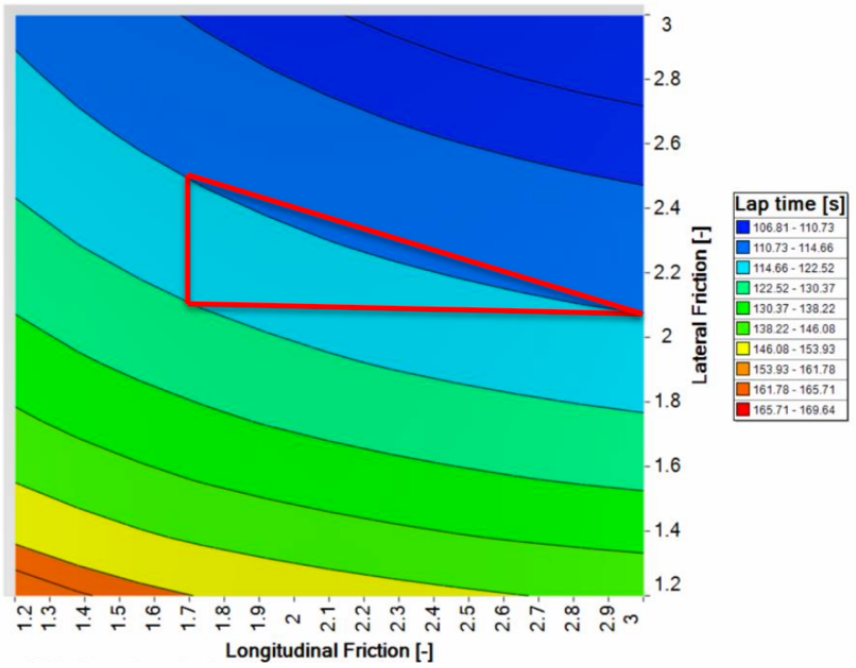

Figure 7: Effect of longitudinal and lateral vehicle acceleration on lap time.

A vehicle's lateral acceleration during the apex of a corner can be described as a quasi-steady-state condition equivalent to steady-state circular motion:

$$a_y = \frac{V^2}{R}$$

Using simplified linear single-track vehicle models, the steady-state cornering condition at a given velocity and radius can be calculated as a function of the steering angle:

$$\delta_{dyn} = \delta_{kyn} + K_u a_y$$

$$K_u = \left(\frac{l_r}{l}\left(\frac{m}{C_f F_{zf}}\right) - \frac{l_f}{l}\left(\frac{m}{C_r F_{zr}}\right)\right) = \frac{\partial \delta_{dyn}}{\partial a_y}$$

$$\delta_{kyn} = \tan^{-1}\frac{l}{R}$$

δ_{dyn} is the dynamic steering angle as a function of velocity for a constant turning radius. K_u is the understeer gradient and is commonly referred to as the change in dynamic steering wheel angle per change in lateral acceleration. The understeer gradient is a function of the tire cornering stiffnesses C_f and C_r, which determine the tire slip angles α_f and α_r. At higher speeds, the tire slip angles of both tires contribute significantly to the instant center location and thus turning radius of the vehicle, as shown in Figure 8.

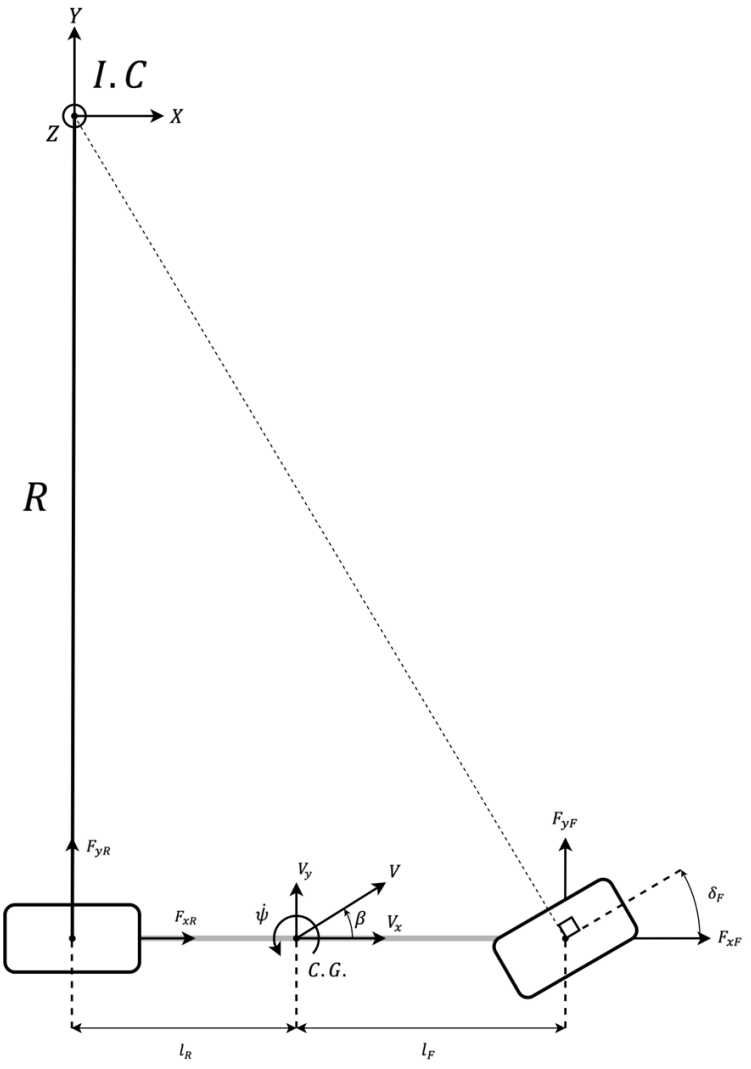

Figure 8: Single-track kinematic model. At low speeds, the rear tire slip angle is small and does not contribute to the instant center location.

δ_{kyn} is the kinematic steering angle and represents the steering angle at very low speeds as shown in Figure 9. Because the rear tire is not steered, at low speeds there is very little vehicle side-slip angle and yaw rate, which makes the rear tire slip angles very small. Therefore, the geometry of the vehicle dominates the steering response.

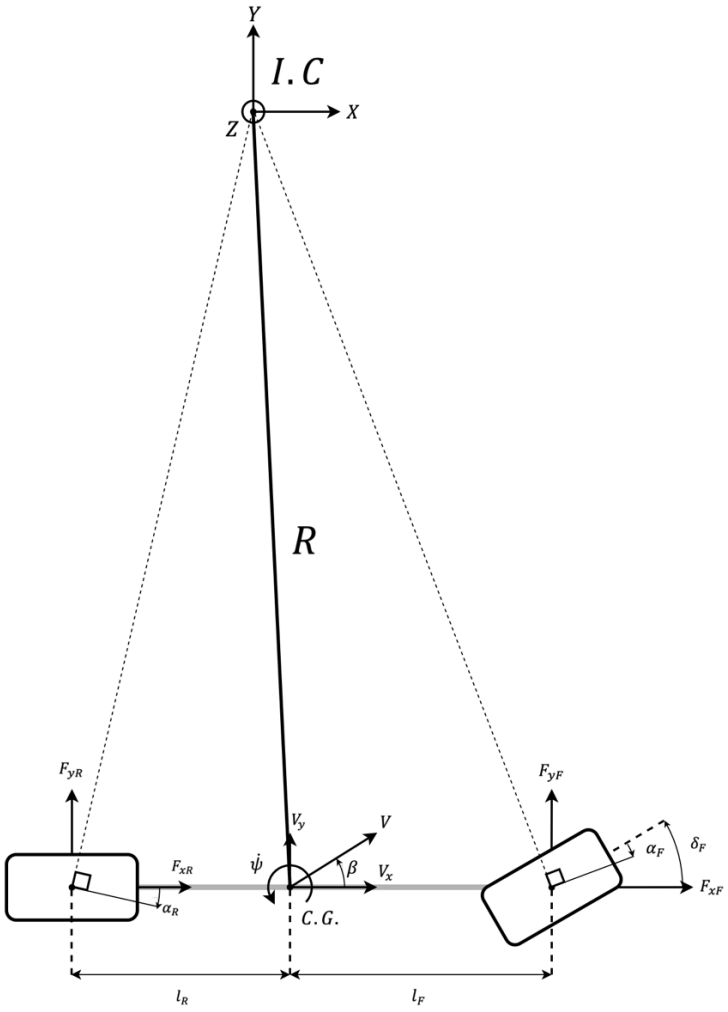

Figure 9: Single-track dynamic model. At higher speeds the rear tire slip angle is significant and contributes to the instant center location.

With the dynamic steering relationship of the linear vehicle model determined, the lateral acceleration of the vehicle for a fixed corner radius can be plotted as a function of the understeer gradient:

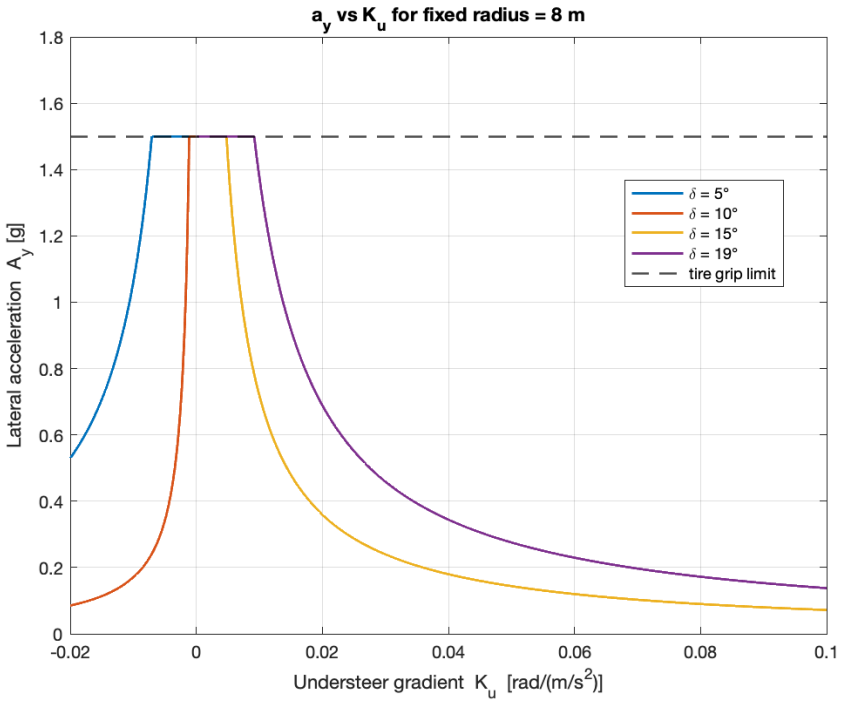

Figure 10: Lateral acceleration vs understeer gradient for a fixed radius, and for different steer angles up to the maximum tire steer angle for the vehicle, 19 degrees.

Designing a vehicle with a neutral understeering gradient will result in a vehicle with the highest lateral acceleration capability for any given corner radius. In Figure 10, it is shown that for the typical FSAE skidpad radius of 8 m, the vehicle needs to have an understeer gradient between -0.01 and 0.01 to achieve the maximum lateral acceleration, which is limited by the grip of the tires. However, the real vehicle deviates from the ideal single-track model with non-linearities in the tire behavior, kinematic compliance, and aerodynamic forces. A control system actively adjusting the vehicle behavior to achieve a target understeer gradient can overcome these non-linearities and improve the vehicle's cornering acceleration throughout the vehicle speed range.

Model Development (BG/RS)

A non-linear transient plant model of the vehicle's lateral dynamics was developed in MATLAB Simulink to simulate the yaw response of the vehicle. The plant model inputs are the driver inputs – steering wheel angle, accelerator pedal position, and brake pedal position. Initial conditions such as initial velocity are set, and the vehicle dynamic states in both the body and inertial reference frame are calculated as plant outputs.

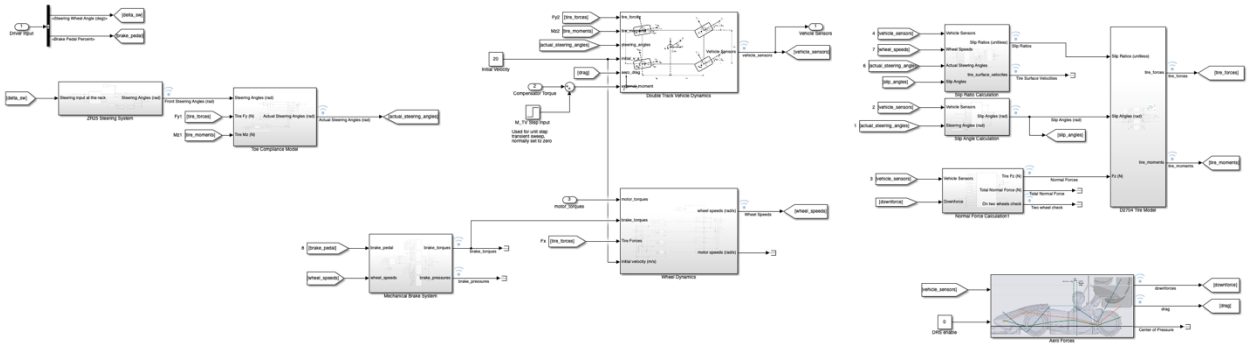

Figure 11: Top-level view of the vehicle plant model.

The lateral dynamics are modeling using the double track vehicle model. Vehicle force inputs calculate body frame reference accelerations and velocities, and then are converted to inertial reference frame values. Force inputs include tire forces and moments, as well as external forces such as aerodynamic downforce and drag.

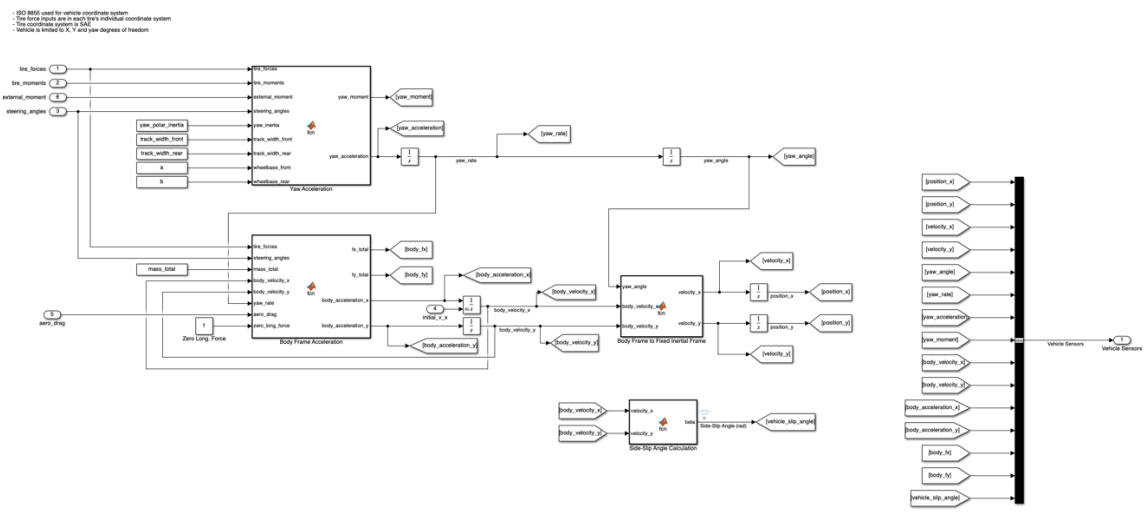

Figure 12: Double track vehicle dynamics block.

The steering system was also modeled as a separate block. This involved a lookup tables containing the individual steer angles of each tire as a function of steering wheel angle, which allows the Ackermann and kingpin angle geometry of the suspension kinematics to be included without directly modeling them. Additionally, provisions in the Simulink model were added for steering system compliances, specifically lateral-force and self-aligning moment-induced toe angle deflections, because these compliances affect the front and rear axles differently, and thus are a significant contributor to the balance of the vehicle and the understeer gradient.

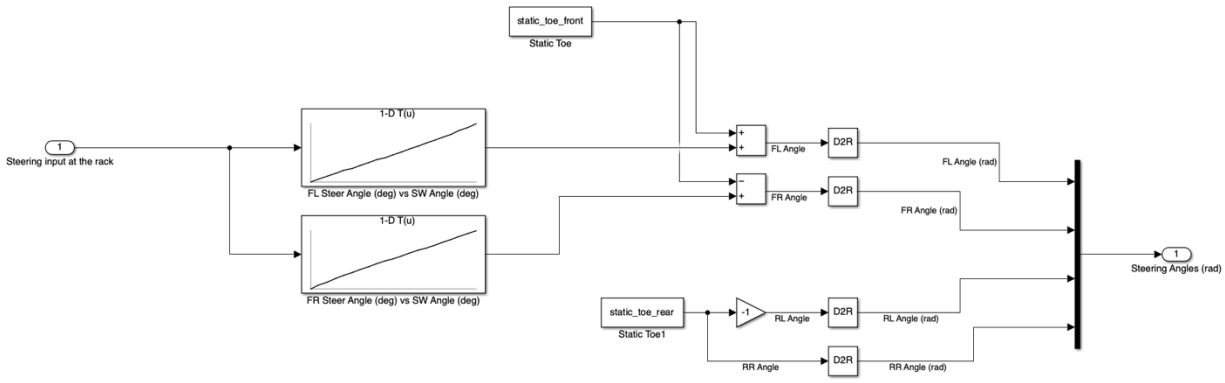

Figure 13: Steering system block before compliances are added.

The individual wheel dynamics were also modeled. This additional state was included because the relative slip between the wheel angular velocity and the tire surface velocity is a significant factor in determining the appropriate longitudinal tire forces.

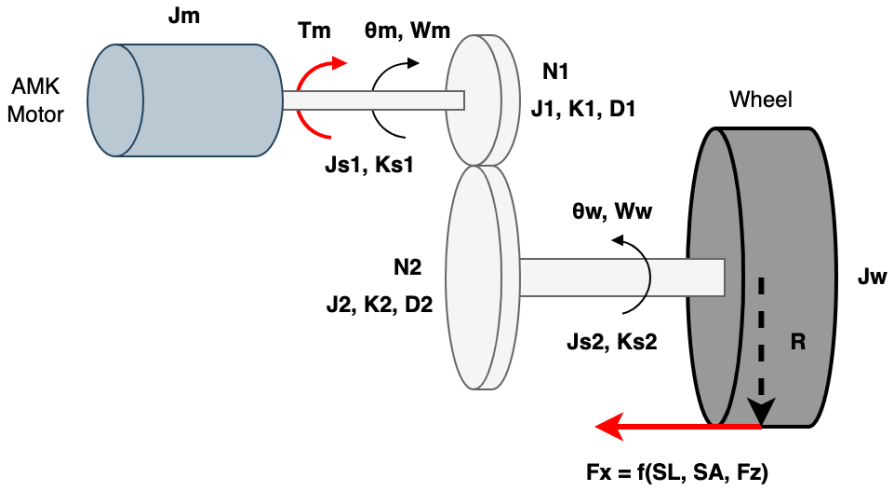

Figure 14: Equivalent model of the wheel dynamics. The actual vehicle uses a planetary gear system, so the equivalent inertia of the gearbox at the input was calculated in CAD and applied to the model at J1.

Tire Forces (BG)

The non-linear tire forces are very important to model as they have the largest effect on the vehicle behavior. Additionally, the objective of the vehicle as a race car is to develop the highest levels of acceleration, which require the operation of the tires within their non-linear, saturated regions. The tires selected for use on the vehicle were tested by the FSAE Tire Test Consortium, which pools funding from multiple contributing universities to conduct flat-trac testing at the Calspan Tire Test facility, as shown in Figure 15. The testing consisted of multiple test procedures analyzing the pure cornering performance of the tire as well as the longitudinal drive/brake performance, and combined conditions.

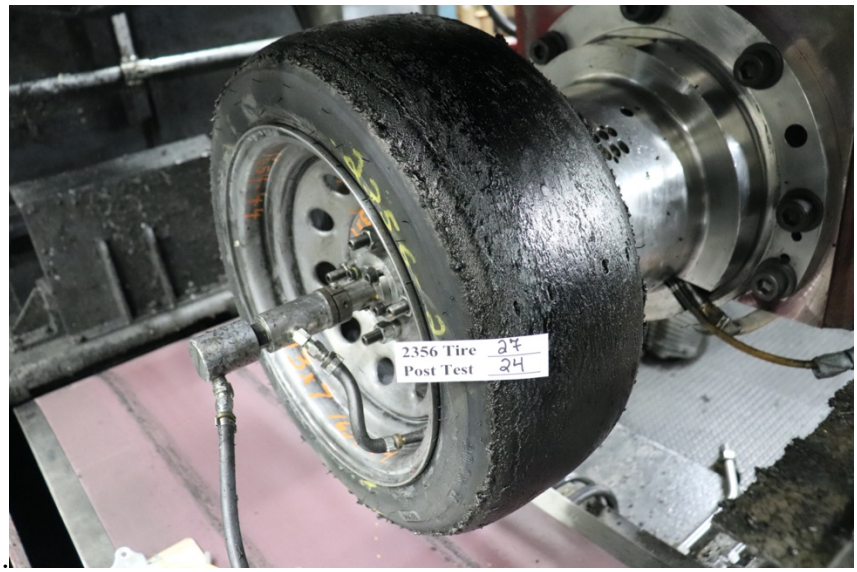

Figure 15: Example of flat-trac testing the Goodyear FSAE tire.

The raw data was imported into MATLAB, cleaned up and organized, and each run was fit with polynomial equations. Constant pressure and inclination angle data was selected to reduce the complexity of the tire model. This fitting allowed filtering and removal of hysteresis from multiple sweeps.

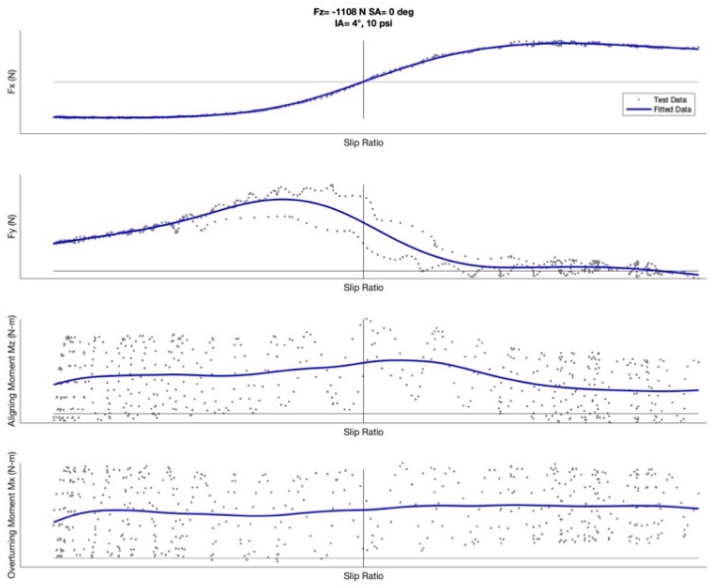

Figure 16: Raw data processing.

The cleaned-up data from the different test procedures was then combined in a multi-dimensional lookup table, with slip angle, slip ratio, and normal load as independent columns, and longitudinal force, lateral force, and self-aligning moment as dependent columns. Because the test procedures from pure cornering, drive/brake, and combined tests used different independent variable sampling ranges, a method for interpolating multi-dimensional scattered data was used to create the tire force curves. Interpolation using a radial basis function in MATLAB was performed, because this method can handle multi-dimensional, scattered data, and can also interpolate outside the convex hull of the test data. 3-D tire lookup tables were generated as shown in Figure 17:

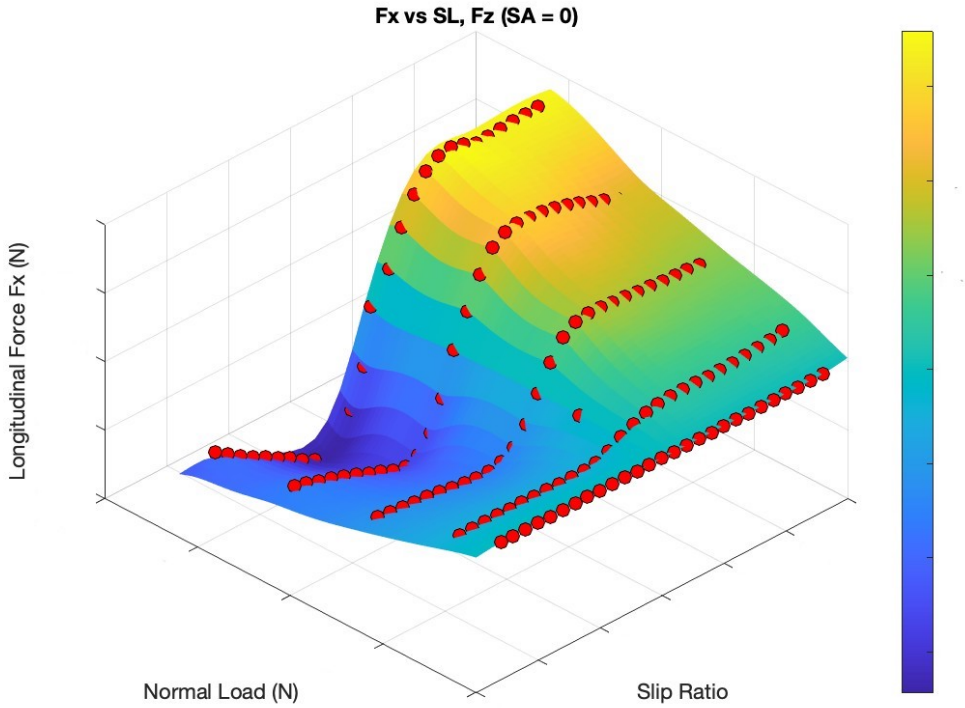

Figure 17: Final RBF-generated tire lookup model. A slice of longitudinal force data at a constant slip angle and varying normal load and slip ratios.

This method of a fully empirical tire model was chosen over more common semi-empirical tire models, such as the Pacejeka model, for several reasons, primarily because the empirical lookup table can be easily inverted and used in firmware to solve for tire slip ratios given a target longitudinal force. Other factors include a high level of slip angle asymmetry in the chosen tire, which gives the RBF interpolation a better fit to test data than the Pacejeka model.

The transient effects of the tires were also modeled from tire test data. This was approximated using a 1st order system and fitting an appropriate time-constant to find the tire relaxation length, which is the distance required for a tire to roll to achieve 67% of the final steady-state force. MATLAB's lsqcurvefit function was used for this process. In the Simulink model, the 1st-order system is in series after the steady-state lookup table block.

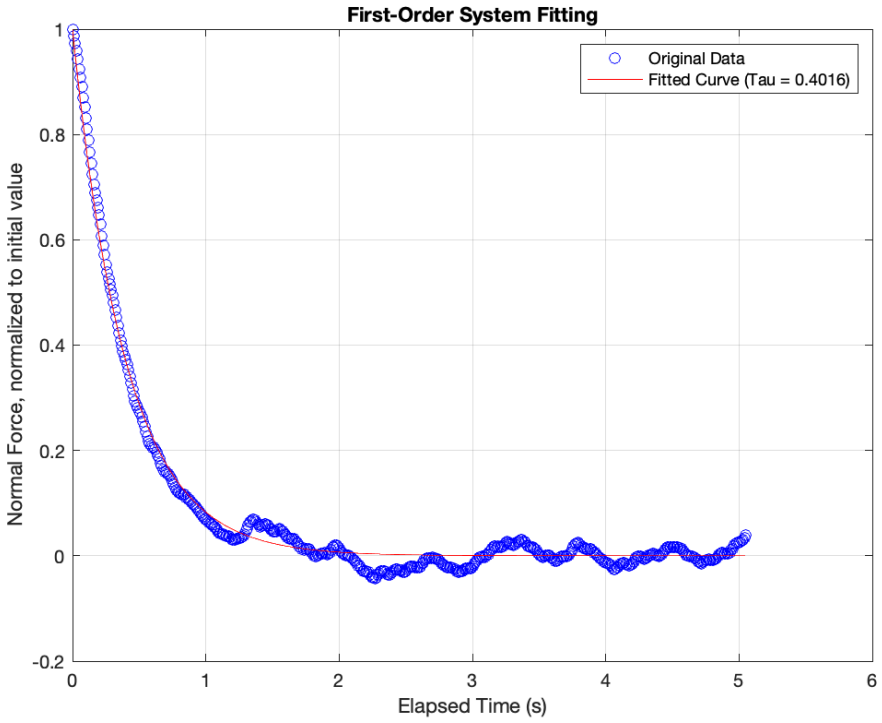

Figure 18: Fitting 1st order system to transient tire test data, for Fx and Fy

Control Algorithm Bandwidth

The system is designed to control the yaw-rate of the vehicle by actuating the electric motors. Since the system is implemented on an embedded device, and electric motor control is performed over CAN bus communication, the system output is updated at a fixed frequency. This output frequency must satisfy the following conditions:

- The update frequency must allow the control system to achieve the desired closed-loop bandwidth.
- The update frequency must be at or below the Nyquist frequencies of the input sensor sampling frequencies. Higher update frequencies require sensors that sample at higher frequencies which may not be obtainable.
- The system must be able to reliably compute the new output state of the controller within the time specified by the update frequency.
- The electric motors, which are actuated by the control system, must have a useful gain at the update frequency. This is further limited by the inertia of the drivetrain and wheels attached to the electric motor output shaft.
- The update frequency is limited by the maximum update rate of the motor inverters.
- The update frequency should be as low as possible to reject sensor noise while meeting all other conditions.

To determine the desired bandwidth of the control system, a simulation was performed with the vehicle driving around a track typical for the FSAE competition. Then, yaw-rate data of the simulation was recorded, and a frequency analysis was performed:

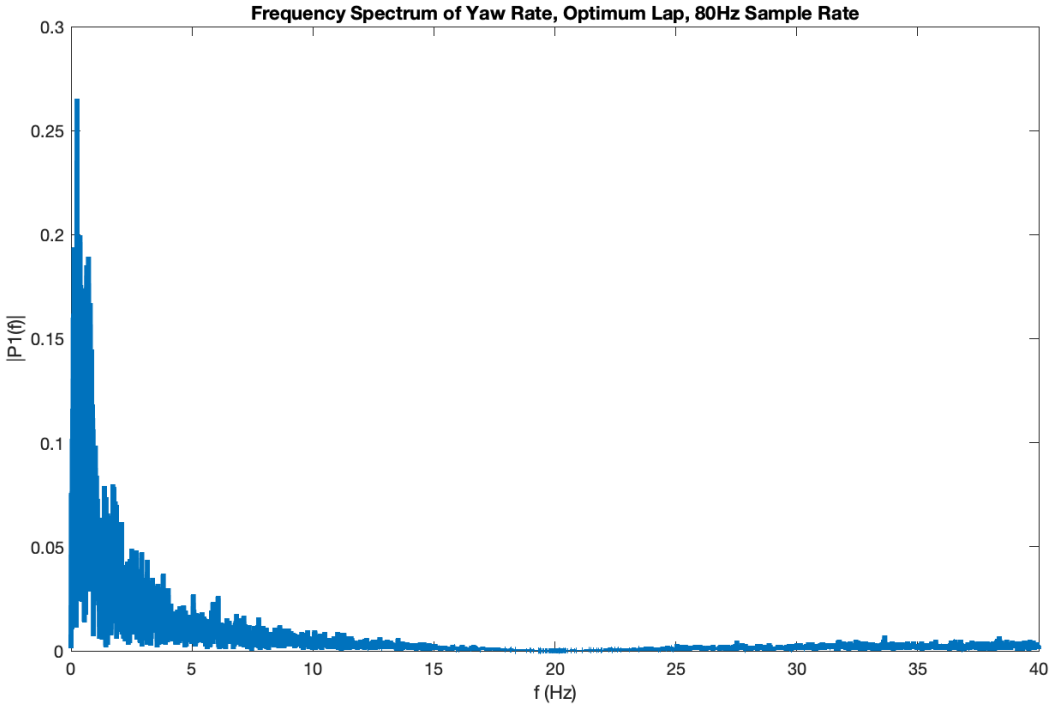

Figure 19: Frequency Spectrum of the yaw-rate, FSAE vehicle simulation

It's clear that most of the frequency content within the yaw-rate occurs at or below 5 Hz. From the other conditions listed above, an update rate of 25 Hz or higher was determined to be acceptable.

Engineering Requirements Specification (BG, RS, JW, TE)

The engineering requirements outlined in this section describe the design and implementation of the vehicle control unit, which will ensure the safe and efficient operation of the powertrain, within the boundaries of the competition rulesets. The system will provide methods for controlling torque and speed based on driver inputs, featuring various vehicle states for safety management. Key functionalities include interlocks, multiple control modes, and effective fault detection mechanisms. Additionally, the system must interface with the existing vehicle systems and comply with mechanical, electrical, software, and safety specifications. More detailed information on engineering and marketing requirements can be found in the appendices, which will aid in the design and implementation processes.

Functional

1.1.7. The system shall provide a method of controlling the torque and speed inputs to the powertrain motor controller(s) using driver and vehicle inputs and a control system.

1.1.8. The system shall control a set of “vehicle states” per FSAE 2025 rule EV.9, which ensures the safe operation of the vehicle and prevents the powertrain from operating unless in the desired state (See Table 33 in Appendices).

1.1.9. The system shall provide a method of switching between different “control modes” for the powertrain, which can be selected by the driver (See Table 34 in Appendices).

1.1.10. The system shall include the required interfacing capability for sensors necessary for powertrain control algorithms (See Table 36 in Appendices).

1.1.11. Per FSAE rule T.3.4 the system shall activate a brake light on the rear of the vehicle based on of the following conditions:

1.1.11.1. The driver presses the brake pedal and exceeds a setpoint of 5% of maximum pressure in the brake system.

1.1.11.2. When regenerative braking is occurring.

1.1.12. The system shall control water pump(s) and radiator fan(s) associated with the vehicle's cooling system in the following ways:

1.1.12.1. The system shall turn on the radiator fan(s) when the temperature exceeds 45 degrees C, and turn them off when the temperature drops below 30 degrees C.

1.1.12.2. The system shall turn on water pump(s) when in the Ready to Drive state.

1.1.13. The system shall not command the powertrain to draw more than 80 kW of power from the high voltage battery output terminals, per FSAE 2025 rule EV.3.3.1 & EV3.4.1.

1.1.14. The system shall monitor the switched states of the interlocks in the shutdown loop.

Electrical

1.1.15. Voltage

1.1.15.1. The system shall be powered and ground-referenced from the existing GLV system of the vehicle.

1.1.15.2. The system shall operate with a nominal input voltage of 12 V DC and an input range of 4.5 to 17 V DC.

1.1.16. Current

1.1.16.1. The maximum current draw of the system shall be 3 A from the GLV system.

1.1.17. Electrical Protection

1.1.17.1. The power supply input shall implement over-voltage protection, reverse-polarity protection, and limit the system to 3 A.

1.1.17.2. Power outputs to external devices (pumps, fans, sensors, etc.) shall be current limited to their respective maximum operating currents.

1.1.17.3. The device, when fully assembled and ready for installation into the vehicle, shall have adequate ESD protection for human handling.

Software

1.1.18. The system shall record and store its settings (tuning parameters, etc.) across power cycles.

1.1.19. The system shall include watchdog functionality to detect software faults resulting in unresponsive behavior.

1.1.20. Real-time Processes

1.1.20.1. The system must update the powertrain at a fixed rate of at least 25 Hz.

1.1.20.2. For every CAN Bus message relevant to the system, the system shall process (update its internal state) and/or respond within a specified process time.

System Integration

1.1.21. The system shall be able to interface with the existing low voltage system, CAN bus communication, motor inverters, battery management systems (BMS), cooling systems, and shutdown loop.

Environmental

1.1.22. The system shall be waterproof.

Safety

1.1.23. The system shall detect system faults and operate in a predictable way to protect the driver and the vehicle. This includes performing fault analysis and error checking (see Table 28 in Appendices).

Engineering Standards Specification (BG/TE)

Units

All units shall be in SI units. This ensures consistency in calculations and simplifies integration with sourced components as SI is the standard globally. SI compliance also helps with clear communication within the design team and through documentation. The transmission and storage of data in firmware may use multiples of SI units to achieve the precision required within data type size limitations.

Coordinate Systems

All control algorithms shall use the ISO 8855 standard for the global vehicle coordinate system.[22] This standard provides a consistent framework for defining vehicle dynamics and orientation and ensure consistent alignment between algorithms, simulations, and real-world testing.

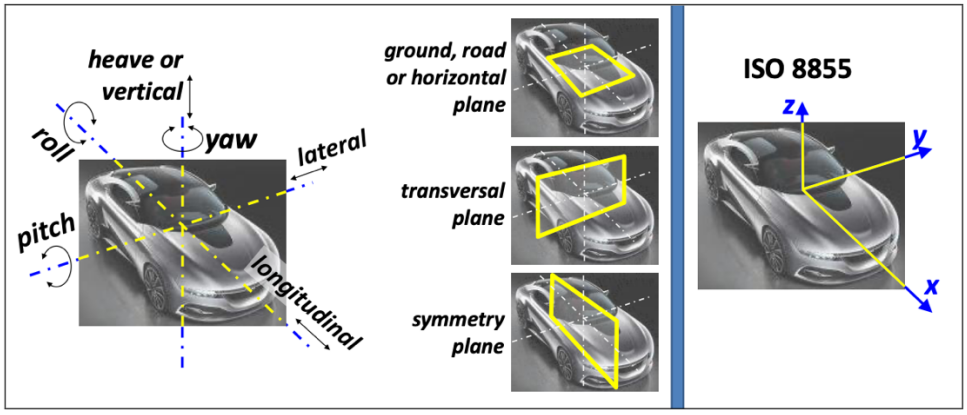

Figure 20: Mechanical description of ISO 8855 coordinate system

All tire models used by control algorithms shall be based on the SAE J670 Tire Coordinate System. This standard gives a consistent framework for defining tire forces and moments, which is critical for accurate vehicle dynamics modeling. Adhering to SAE J670 will ensure precise

torque vectoring, stability control, and improved vehicle handling in simulations and real-world testing.

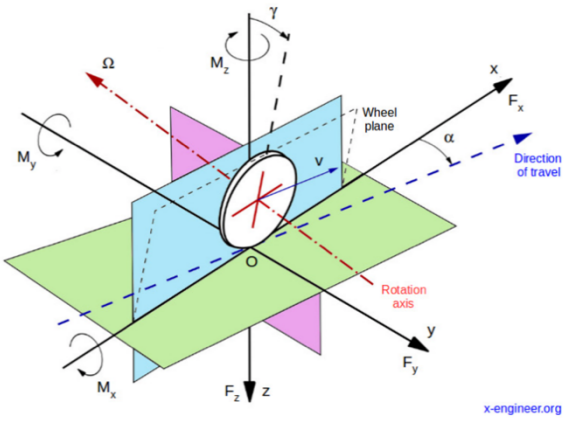

Figure 21: Tire axis system and terminology defined by SAE standard

Communication Bus

Components will communicate with each other using the CAN Bus communication physical layer standard ISO 11898.[21] This standard is widely used in the automotive industry for its reliability and efficiency in high-noise environments. The standard also ensures reliable communication between the VCU, sensors, and powertrain systems, and is commonly used in real-time systems.

Programming Languages

Microcontroller firmware will be written in the C programming language, standardized as ISO/IEC 9899.[20] C is the de-facto standard for real-time embedded systems due to its efficiency and low-level hardware access. Adhering to the ISO/IEC 9899 standard ensures the firmware is portable and compatible with many embedded frameworks, which will enable reliable and responsive control of the VCU.

Accepted Technical Design

Hardware (BG, JW, RS)

Level 0 Hardware Block Diagram

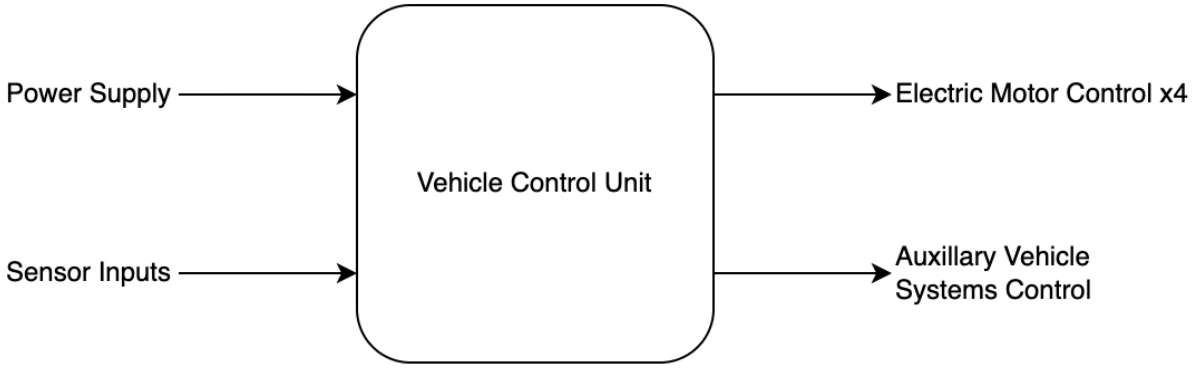

Figure 22: Level 0 hardware block diagram

At the highest level, the VCU will be given sensor inputs about the vehicle location and driver inputs. Using power from the GLV system on the vehicle, the VCU will control the motor outputs as well as several auxiliary systems.

Table 1: Functional requirement table for Level 0 vehicle control unit

Module	**Vehicle Control Unit (VCU)**
Designers	Tetra Engdahl, Brian Glen, Ryan Stoller, John Wozencraft
Inputs	Sensor Inputs – The driver's inputs, such as steering, brake, and accelerator positions, and other vehicle information from various external sensors, will be passed to the VCU for processing.
Outputs	Motor Control – After the VCU has processed the inputs, the VCU will output a signal to control the torque, speed, and direction of each of the four vehicle motors.
Description	Process the inputs from the vehicle and driver to determine necessary torque and direction values of the motors.

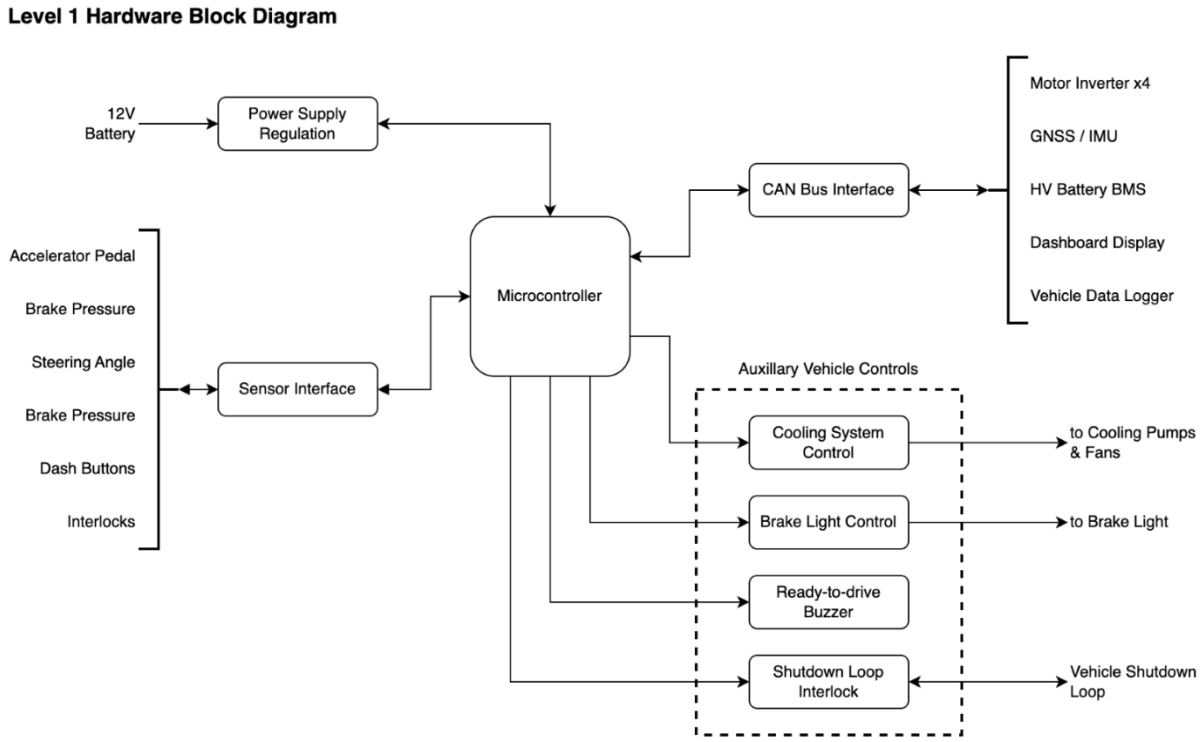

Figure 23: Level 1 hardware block diagram

The level 1 diagram shows the specific systems the VCU will be interfacing with. This includes the sensors it will be interacting with as well as the auxiliary systems it will be controlling. This level also shows some of the hardware devices the VCU will use to interact with the motor drives.

Table 2: Functional requirement table for Level 1 power supply regulation

Module	**Power Supply Regulation**
Designers	Ryan Stoller
Inputs	12V battery and enable signals from microcontroller.
Outputs	+3.3V rail, +5V rail, battery voltage monitoring signal
Description	Regulates the power delivered to the microcontroller to ensure proper functionality. Measures battery voltage to check for low-battery condition.

Table 3: Functional requirement table for Level 1 sensor interface

Module	Sensor Interface
Designers	Brian Glen
Inputs	Accelerator pedal, brake pressure, steering angle, dash buttons, interlocks
Outputs	Microcontroller-compatible sensor signals
Description	Interfaces with all sensors in the vehicle to condition the signals for the microcontroller to process.

Table 4: Functional requirement table for Level 1 shutdown loop interlock

Module	Shutdown Loop Interlock
Designers	Tetra Engdahl
Inputs	Vehicle shutdown signal, microcontroller shutdown signal
Outputs	Outgoing vehicle shutdown signal
Description	Opens the shutdown loop, deactivating the vehicle in case of a critical system fault. Also alerts the driver of a safety issue and can be used for debugging information.

Table 5: Functional requirement table for Level 1 ready-to-drive buzzer

Module	Ready-to-drive Buzzer
Designers	John Wozencraft
Inputs	Ready-to-drive signal from microcontroller.
Outputs	Audible Sound
Description	When the system enters the ready-to-drive state, an audible buzzer must be sounded to alert the driver.

Table 6: Functional requirement table for Level 1 CAN bus interface

Module	CAN Bus Interface
Designers	Tetra Engdahl
Inputs	CAN bus UART controller signal
Outputs	CAN bus differential signal
Description	The CAN bus is used to communicate with various sensors/systems through the vehicle and is needed for proper VCU processing.

Table 7: Functional requirement table for Level 1 microcontroller

Module	Microcontroller
Designers	Tetra Engdahl, John Wozencraft
Inputs	Regulated power supply, Vehicle sensor data
Outputs	Shutdown signal, Ready-to-drive signal, Brake light control signal, Cooling system control signal, CAN bus communication signal
Description	The microcontroller is the computer that implements the VCU, taking in all vehicle signals to control the motors and other internal vehicle control systems.

Table 8: Functional requirement table for Level 1 cooling system control

Module	Cooling System Control
Designers	John Wozencraft
Inputs	Cooling system control signal from microcontroller
Outputs	Fan and cooling pump power signals
Description	Vehicles cooling system controller to keep the vehicle within a safe and functional temperature range

Table 9: Functional requirement table for Level 1 brake light control

Module	Brake Light Control
Designers	Ryan Stoller
Inputs	Brake light control signal from microcontroller
Outputs	Brake light power signal
Description	Vehicles brake light control system to display the brake light during braking.

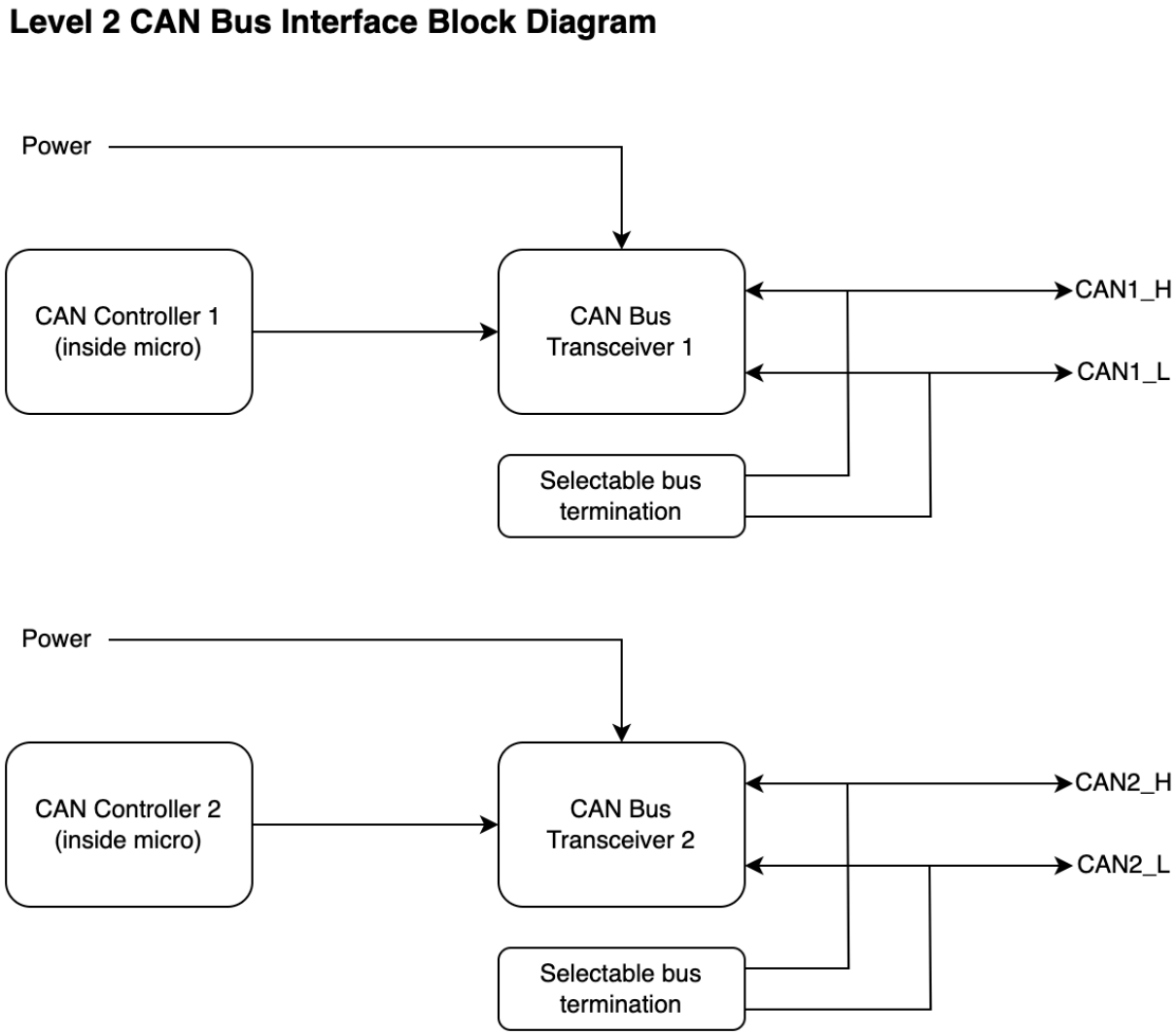

Figure 24: Level 2 CAN bus interface hardware block diagram

Two CAN bus transceivers are implemented on the VCU. This ensures that there is adequate bandwidth on the vehicle network(s) as sensors and peripherals are added to the vehicle. Additionally, it allows the separation of critical devices needed for powertrain control and non-critical devices, which increases vehicle reliability by keeping the number of physical bus connections on the critical device network to a minimum.

The CAN Bus interface was implemented in with the TCAN1042, an automotive rated, fault-protected, non-isolated CAN bus transceiver, and the two CAN controllers integrated into

the STM32F405. Bi-directional TVS diodes on both differential lines ensure adequate ESD protection. Selectable bus termination was implemented with a DPDT switch into a split termination with cut-off frequency of 1.1 Mhz, which is adequate for the target 1 MBps baud rate.

Table 10: Functional requirement table for Level 2 CAN controller 1

Module	**CAN Controller 1**
Designers	Brian Glen, Ryan Stoller, John Wozencraft, Tetra Engdahl
Inputs	Microcontroller internal CAN bus signals
Outputs	CAN bus UART signal
Description	CAN controller that is inside the microcontroller and controls outgoing CAN bus communication

Table 11: Functional requirement table for Level 2 CAN bus transceiver 1

Module	**CAN Bus Transceiver 1**
Designers	Brian Glen, Ryan Stoller, John Wozencraft, Tetra Engdahl
Inputs	Power, CAN bus transceiver signal
Outputs	CAN_H and CAN_L signals
Description	Outputs CAN_H and CAN_L, based on the transceiver signal, to outgoing CAN bus transmission lines.

Table 12: Functional requirement table for Level 2 selectable bus termination 1

Module	**Selectable bus termination 1**
Designers	Brian Glen, Ryan Stoller, John Wozencraft, Tetra Engdahl
Inputs	CAN_H and CAN_L signals
Outputs	Selectable bus termination signal
Description	Prevents rebound of CAN bus signals and allows for the location of CAN bus termination to be changed by the software.

Table 13: Functional requirement table for Level 2 CAN controller 2

Module	**CAN Controller 2**
Designers	Brian Glen, Ryan Stoller, John Wozencraft, Tetra Engdahl

Inputs	Microcontroller internal CAN bus signals
Outputs	CAN bus transceiver signal
Description	Redundancy CAN controller: CAN controller that is inside the microcontroller and controls outgoing CAN bus communication

Table 14: Functional requirement table for Level 2 CAN bus transceiver 2

Module	**CAN Bus Transceiver 2**
Designers	Brian Glen, Ryan Stoller, John Wozencraft, Tetra Engdahl
Inputs	Power, CAN bus transceiver signal
Outputs	CAN_H and CAN_L signals
Description	Redundancy CAN bus transceiver: Outputs CAN_H and CAN_L, based on the transceiver signal, to outgoing CAN bus transmission lines.

Table 15: Functional requirement table for Level 2 selectable bus termination 2

Module	**Selectable bus termination 2**
Designers	Brian Glen, Ryan Stoller, John Wozencraft, Tetra Engdahl
Inputs	CAN_H and CAN_L signals
Outputs	Selectable bus termination signal
Description	Redundancy selectable bus termination: Prevents rebound of CAN bus signals and allows for the location of CAN bus termination to be changed by the software.

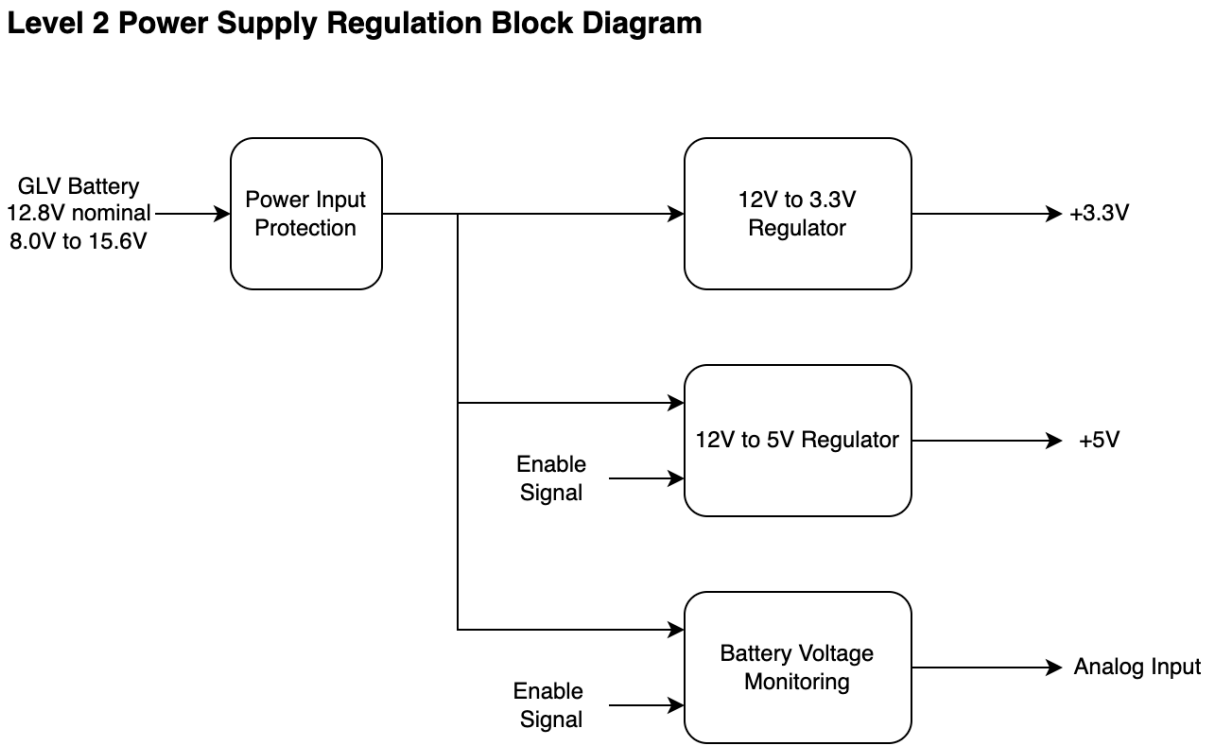

Figure 25: Level 2 power supply regulation hardware block diagram

The power supply will convert a 12 V DC voltage from the GLV system to the appropriate voltage for each device and will provide voltage and current protection of said devices. When choosing the 3.3 V and 5 V switching regulators, an analysis of the loads on each voltage rail was done, with the results shown in Tables 16 & 17. For the 3.3 V regulator, the Recom R-78K3.3-1.0 was chosen due to its high efficiency, load capability, and low ripple current. However, since a reference voltage of 3.3 V was needed, even a low ripple current is not acceptable. So, a low pass filter was added at the output of the 3.3 V converter to give a steady reference voltage.

The 5 V rail will not turn on immediately. This is because of the desired sequencing and the ensuing protection of components. Waiting to turn on the 5 V rail allows the microcontroller to check for any errors and irregularities in the system before turning on the system. It also allows the microcontroller to boot up before turning on the sensors and flooding the

microcontroller with signals. For the 5 V converter, the Texas Instruments “TPS562243DRLR” was chosen because it has an enable pin and its efficiency, small footprint, load capability, availability, and affordability.

The circuit schematics for both the 5 V and 3.3 V regulator are shown in Figure 29. The 3.3 V converter has two capacitors and an inductor (C7, C8, and L2) to help filter out any noise. The 5 V converter has input filter capacitors (C2, C3, and C4) as well as an output filter capacitor and inductor (L1 and C6). The IC chosen has a variable output so the output must be set using a resistive voltage divider (R1 and R3) while C1 is added to increase bandwidth and phase margin. A bootstrap capacitor (C5) is used to connect pins 2 & 4 for proper operation.

Table 16: Analysis of the 5 V loads required by the DC / DC converter

Device	Max Load (mA)	Max Load (mW)
Steering Angle Sensor	30	150
Push Road Force Sensor	34	160
Buzzer	80	400
Accelerator Sensor	50	250
CAN Transceiver	100 (x2)	500 (x2)
Total	314	1,570

Table 17: Analysis of the 3.3 V loads required by the DC / DC converter

Device	Max Load (mA)	Max Load (mW)
Brake Pressure Sensor	3.5	11.55
Microcontroller	240	792
Total	243.5	907.55

Table 18: Functional requirement table for Level 2 power input protection

Module	Power Input Protection
Designers	Brian Glen, Ryan Stoller
Inputs	GLV battery power
Outputs	Regulated battery power
Description	Prevents power faults and overloads to ensure safe power outputs.
Description	Converts the 12V input from the power source into the 3.3V microcontroller needs.

Table 19: Functional requirement table for Level 2 12V to 3.3V regulator

Module	12V to 3.3V Regulator
Designers	Brian Glen, Ryan Stoller
Inputs	GLV battery power
Outputs	+3.3V
Description	Converts the 12V input from the power source into the 3.3V microcontroller needs.

Table 20: Functional requirement table for Level 2 12V to 5V regulator

Module	12V to 5V Regulator
Designers	Brian Glen, Ryan Stoller
Inputs	Regulated battery power, Enable signal
Outputs	+5V
Description	Converts the 12V input from the power source into the 5V microcontroller needs.

Table 21: Functional requirement table for Level 2 battery voltage monitoring

Module	Battery Voltage Monitoring
Designers	Brian Glen, Ryan Stoller
Inputs	Regulated battery power, Enable signal
Outputs	Analog Input
Description	Provides voltage monitoring of the input power to check for battery state and voltage issues.

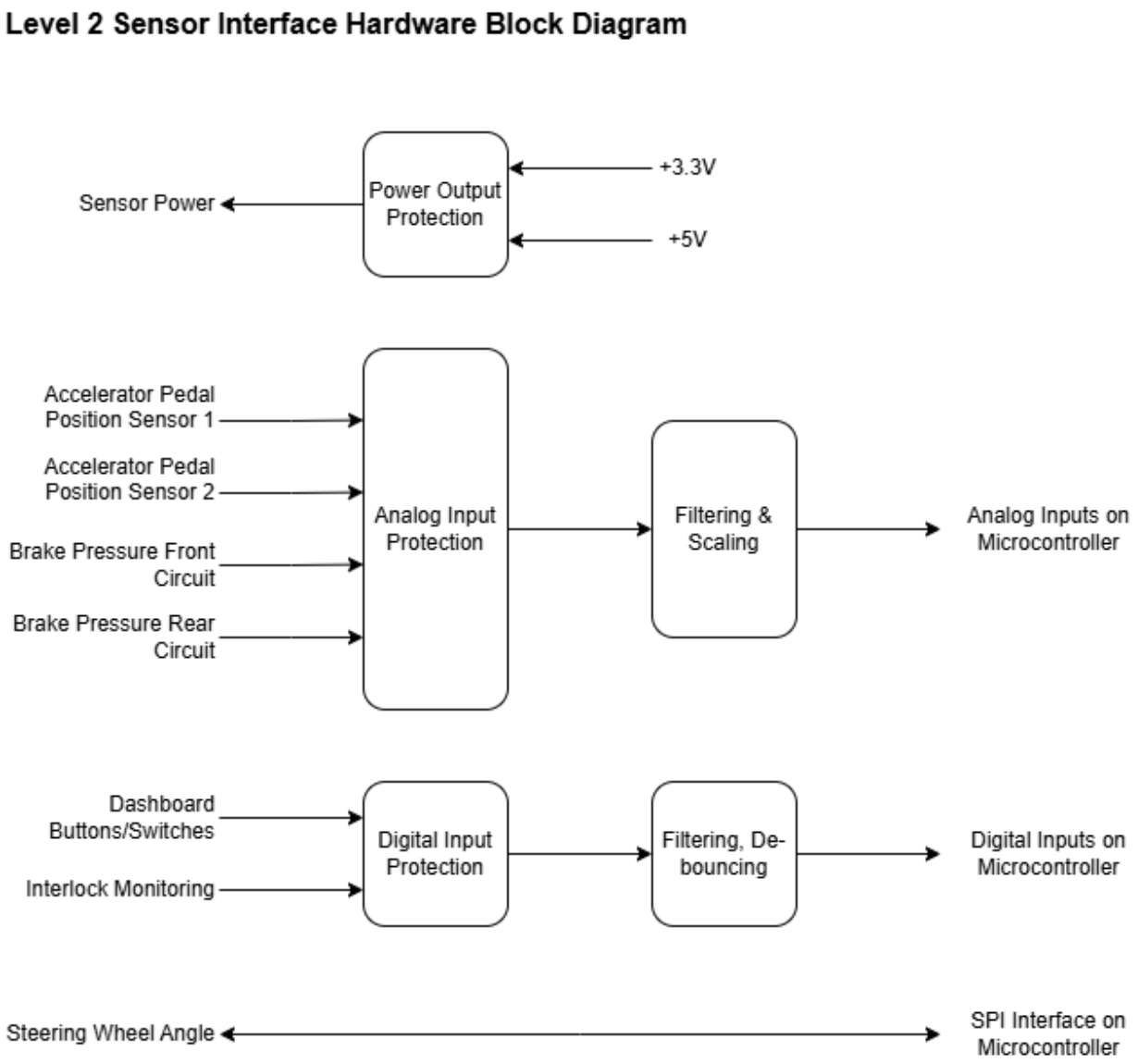

Figure 26: Level 2 sensor interface hardware block diagram

The sensor interface will process the signals from the input sensors and then will pass them to the microcontroller, which includes both analog and digital signals. The sensor interface is implemented in Figure 30.

First, 5V and 3V power output rails are implemented. Because the power regulator output caps provide enough capacitance to shunt a typical HBM ESD discharge within the voltage rail limits, the only additional output protection needed was to include 500 mA PTC resettable fuses that will trip if a sensor output rail is shorted to ground. This means that a sensor power short

trips the fuse instead of tripping the power regulator short-circuit protection, which would result in a microcontroller reset.

Next, the analog input circuitry is created with input protection and voltage scaling. A unidirectional TVS diode provides ESD protection. A voltage divider with a potentiometer serves to scale the voltage from 5V to 3.3V. The potentiometer and test point allows each divider to be accurately set to account for part tolerance. Additionally, the voltage divider serves as a pull-down resistor when the sensor input is floating, allowing the software to detect a floating input. An RC filter is implemented with a cut-off frequency of 5 kHz for noise filtering, and the series resistor of the filter also serves as a current limiting resistor during an ESD event.

The digital inputs also include the same unidirectional TVS diode for ESD protection. Additionally, a hardware de-bouncing circuit made with a pull-up resistor and a capacitor to ground serves to ensure that critical inputs from the driver (such as the start button or vehicle settings adjustments) do not register as multiple inputs due to the physical “bouncing” of the switch mechanisms. The RC time constant of the de-bounce filter is approximately 1 ms.

Table 22: Functional requirement table for Level 2 power output protection

Module	**Power Output Protection**
Designers	Brian Glen
Inputs	+3.3V, +5V
Outputs	Sensor power
Description	Provides the vehicle sensors with necessary protected power.

Table 23: Functional requirement table for Level 2 input protection

Module	Analog Input Protection
Designers	Brian Glen, Ryan Stoller
Inputs	Accelerator pedal position sensor 1, Accelerator pedal position sensor 2, Brake pressure front circuit, brake pressure rear circuit
Outputs	Protected analog input
Description	Prevents power faults and overloads to ensure safe analog inputs.

Table 24: Functional requirement table for Level 2 filtering and scaling

Module	Filtering & Scaling
Designers	Brian Glen, Ryan Stoller
Inputs	Protected analog input
Outputs	Analog inputs for microcontroller
Description	Removes noise and prevents aliasing as well as scaling the inputs to any unit needed for microcontroller inputs.

Table 25: Functional requirement table for Level 2 input protection

Module	Digital Input Protection
Designers	Brian Glen, Ryan Stoller
Inputs	Dashboard buttons/switches, Interlock monitoring
Outputs	Protected digital input
Description	Prevents power faults and overloads to ensure safe analog inputs.

Table 26: Functional requirement table for Level 2 filtering and de-bouncing

Module	Filtering, De-bouncing
Designers	Brian Glen, Ryan Stoller
Inputs	Protected digital input
Outputs	Digital inputs for microcontroller
Description	Removes unwanted noise and filters out short signals and increases reliability of digital input to the microcontroller.

Microcontroller (BG)

The STM32F405RGT6 microcontroller was chosen and implemented in Figure 32. The microcontroller met the needs of the project, with specific features such as two internal CAN

controllers, USB 2.0 FS, 1 Mbyte of flash memory, and 192+4kB RAM. The supporting hardware implementation includes switches for boot memory selection and external reset, an external high speed 8 Mhz oscillator needed for precise CAN and USB timing, Tag-Connect programming header pinned for SWO-trace debugging, and three debug LEDs and test points.

Auxiliary Vehicle Controls (BG, RS)

The auxiliary vehicle controls schematic in Figure 31 include control for a safety interlock relay on the vehicle shutdown loop, cooling system control, buzzer control, and brake-light control.

The shutdown circuitry is implemented with a DPDT relay. A fault signal from the microcontroller sinks coil current and enables the relay. If the microcontroller detects a safety fault, the fault line is released and a pull-up resistor opens the coil, opening the shutdown loop and turning the high voltage systems of the vehicle off.

The cooling system circuitry is comprised of high-side N-MOSFET switches that driver multiple cooling pumps and radiator fans. The VCU receives an external 12V power source that is fused at the vehicle's fuse box for cooling system control. Implementing a high-side switch is necessary in this case because the cooling pumps and fans ground out on the vehicle chassis outside of the PCB itself. This means that a simple low-side switch would expose +12V all the time and could be susceptible to shorting out. The N-MOSFET was chosen for its low series on resistance as the cooling pumps could draw up to 3 amps continuous. A temperature-rise of 9.6 deg C was estimated using the worst-case series on resistance and at an ambient temperature of 30 degrees C, which was deemed acceptable as the circuit would not require a heat sink.

The buzzer control circuit is used to give an audible warning to people when the vehicle enters the Ready-to-Drive state. It is implemented using a piezo-buzzer indicator with a built-in drive circuit that is external to the PCB. The high-side switch circuit from the cooling system is re-used.

The brake light circuit also re-used the high-side switch from the cooling system to minimize part count, despite having a much lower drive current requirement.

USB-C Communication (BG)

It was decided that USB communication was to be implemented on the VCU, in Figure 28. As the intended design of the vehicle involves interacting with the VCU setting via a dashboard on the CAN bus network, it was desired that a command-line interface accessible through a USB virtual COM port be implemented so the VCU can be easily bench-tested when outside of the vehicle. A USB-C 2.0 full-speed interface was added where the VCU acts as a USB device.

Hardware Implementation

The hardware design was created in Altium Designer and was implemented on a 4-layer PCB using primarily surface mount components, as shown in Figure 27. The hardware is mounted in a waterproof, 3D-printed enclosure, and an internal wire harness connects the PCB to waterproof panel-mount circular connectors.

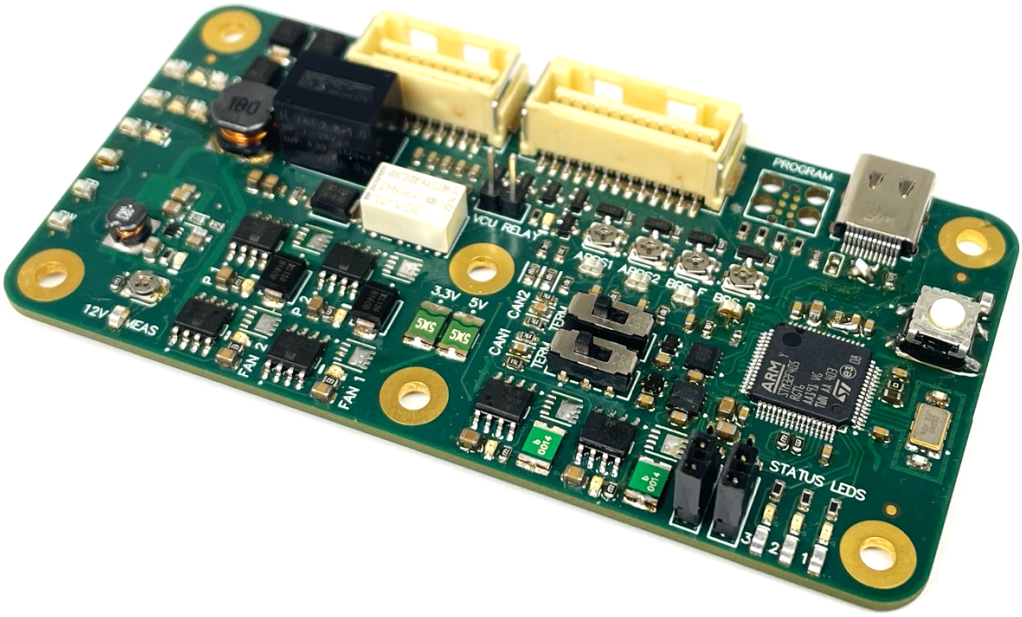

Figure 27: Final vehicle control unit hardware.

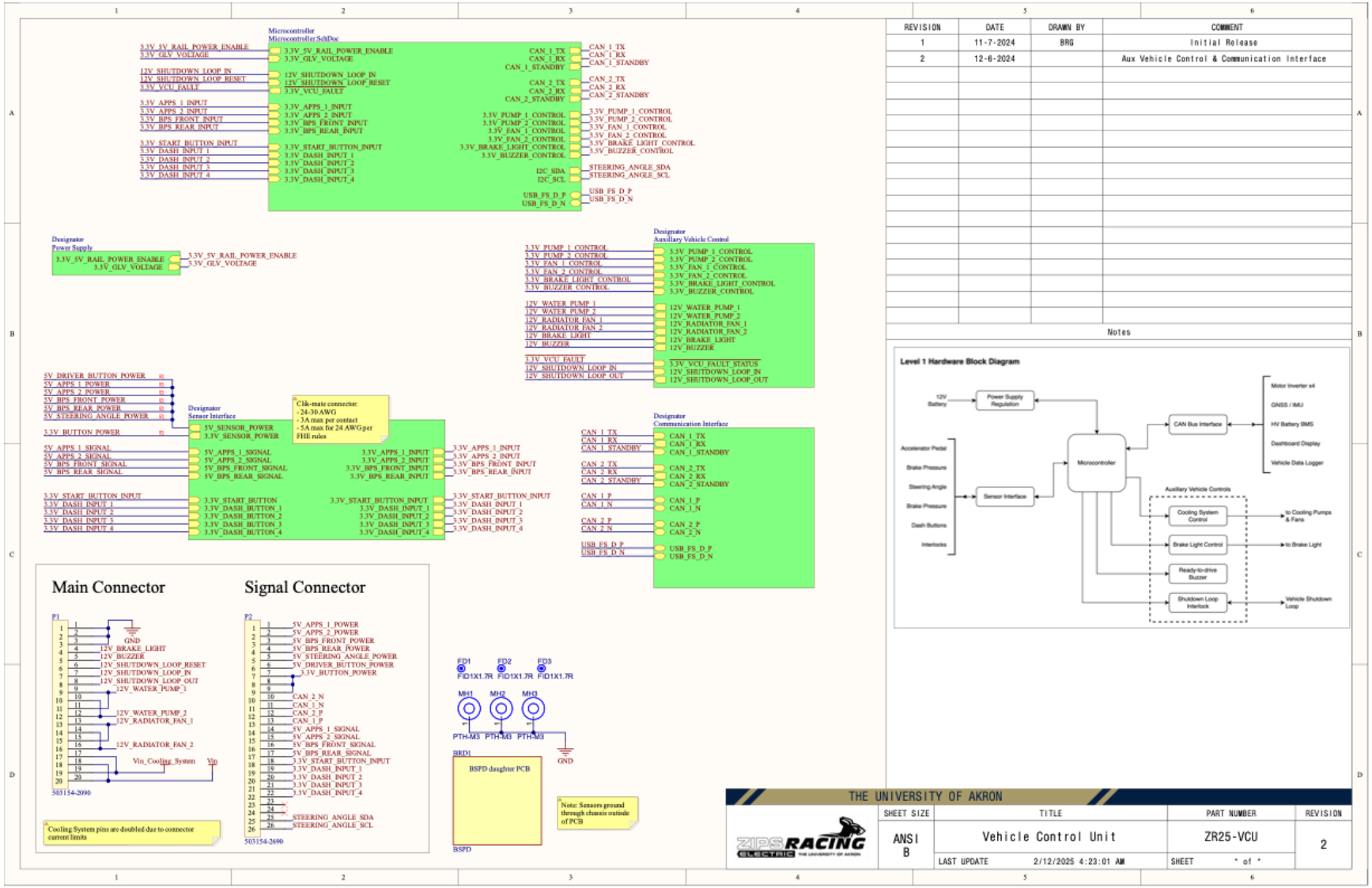

Figure 28: Vehicle Control Unit Schematic - Top Level

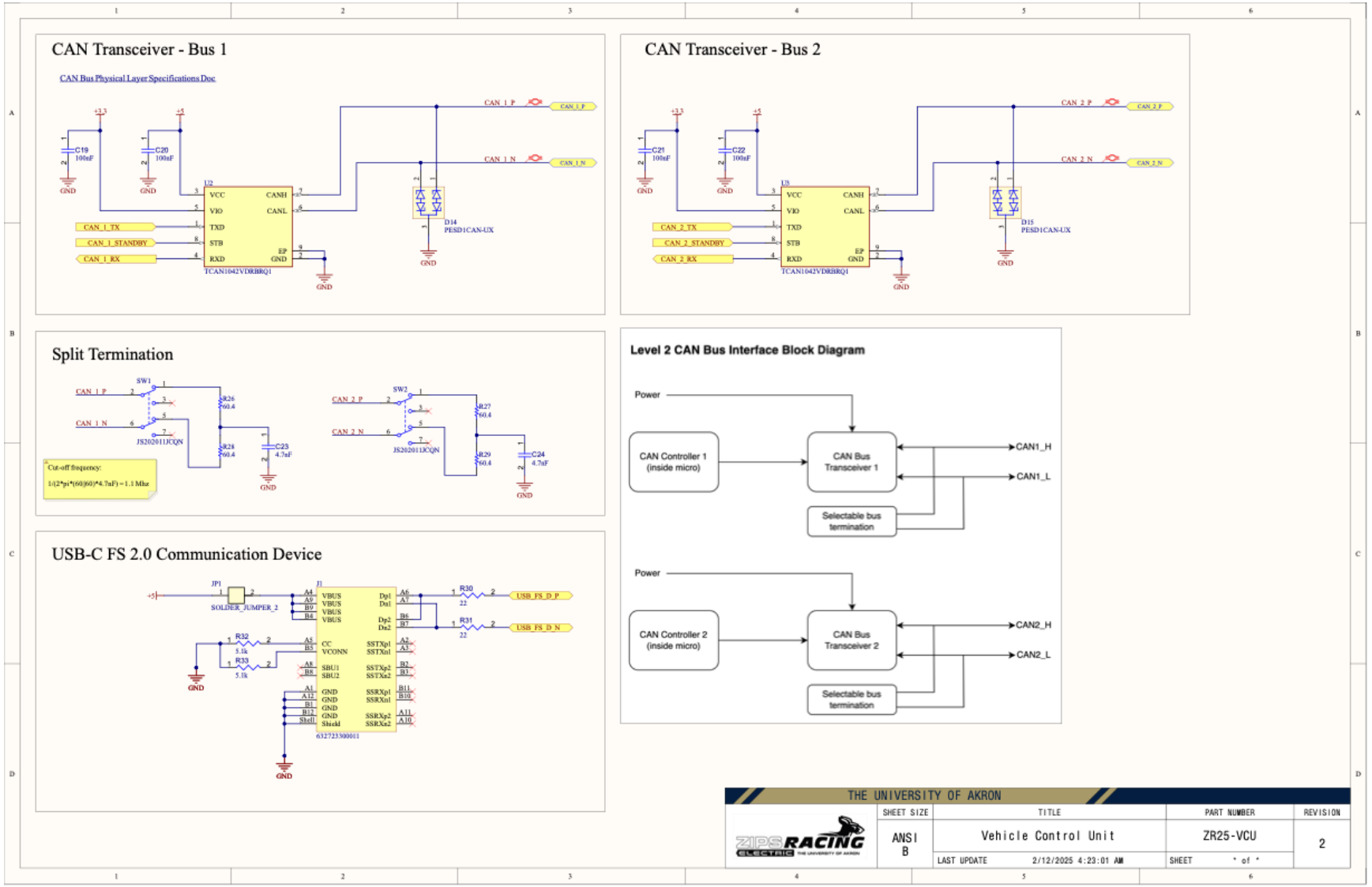

Figure 29: Communication Interface Schematic Sheet

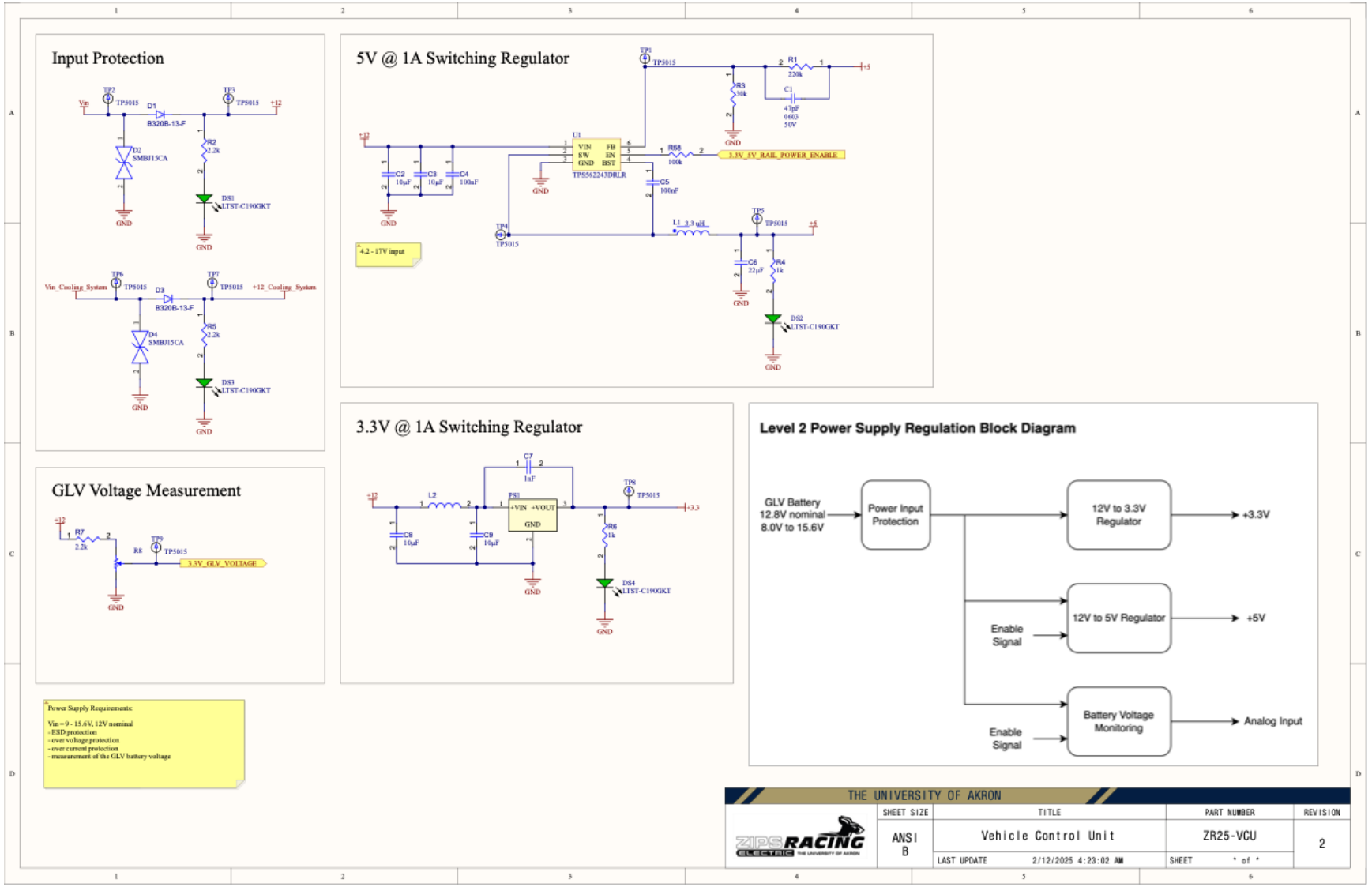

Figure 30: Power Supply Schematic Sheet

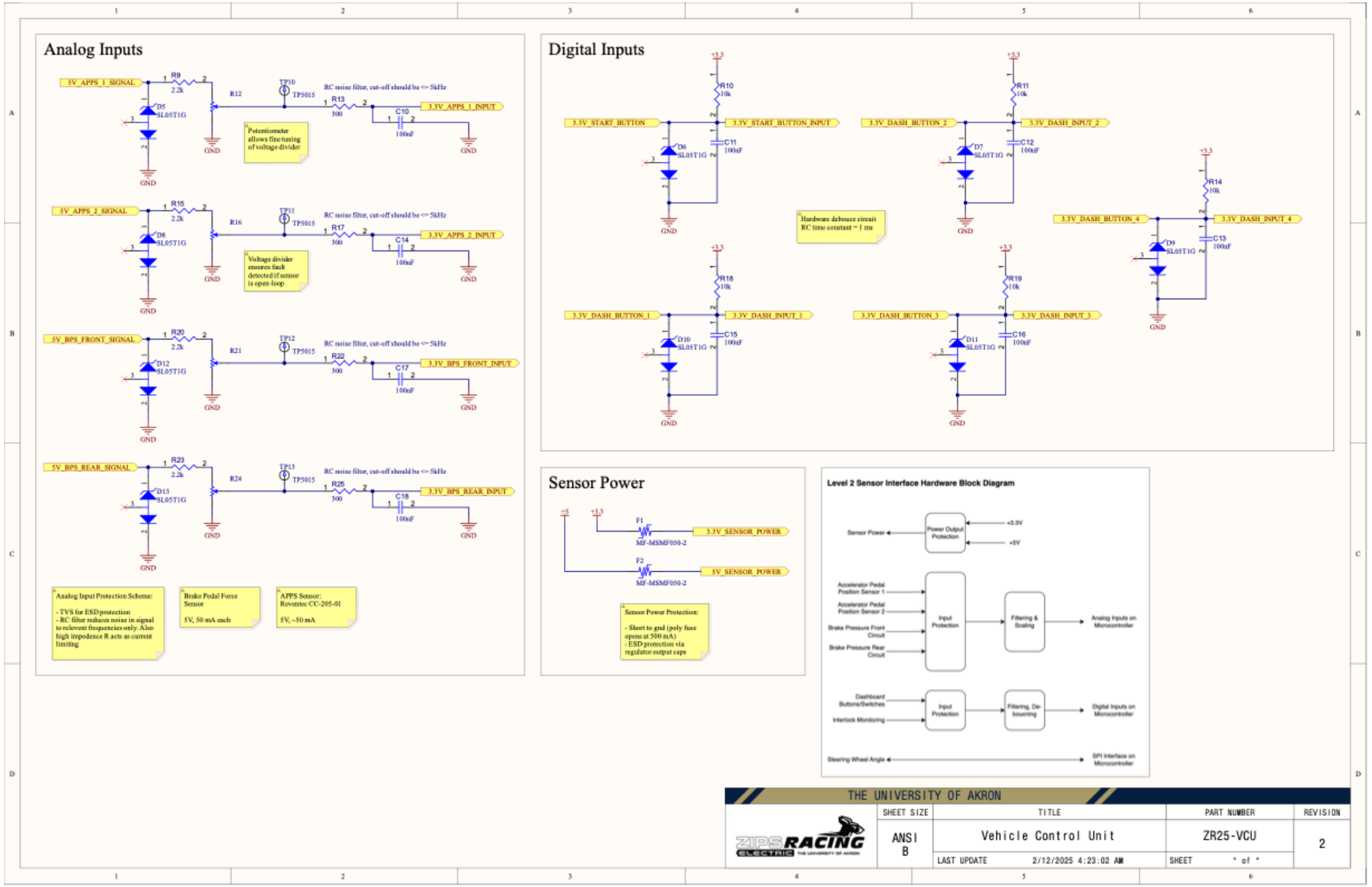

Figure 31: Sensor Interface Schematic Sheet

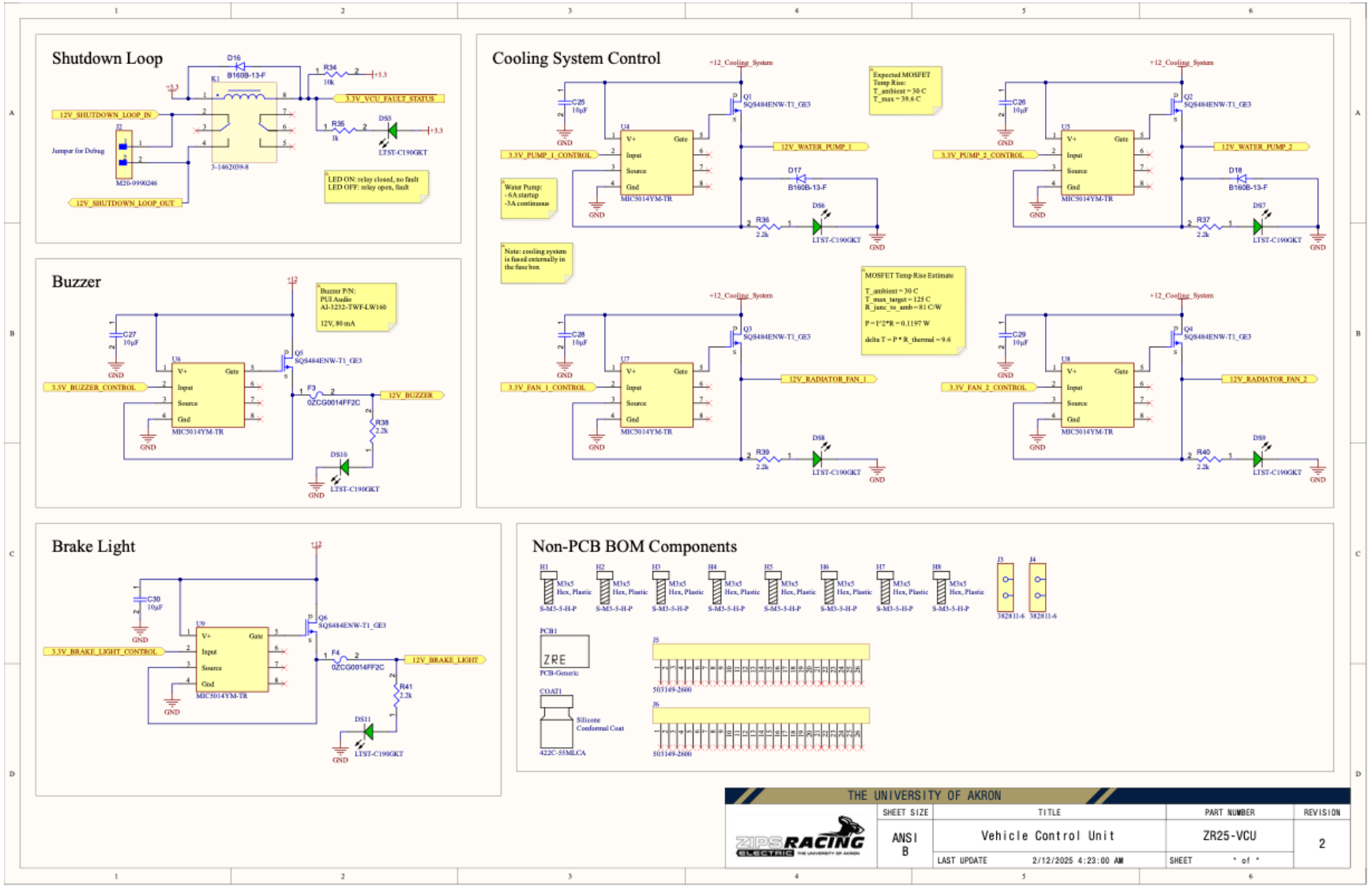

Figure 32: Auxiliary Vehicle Control Schematic Sheet

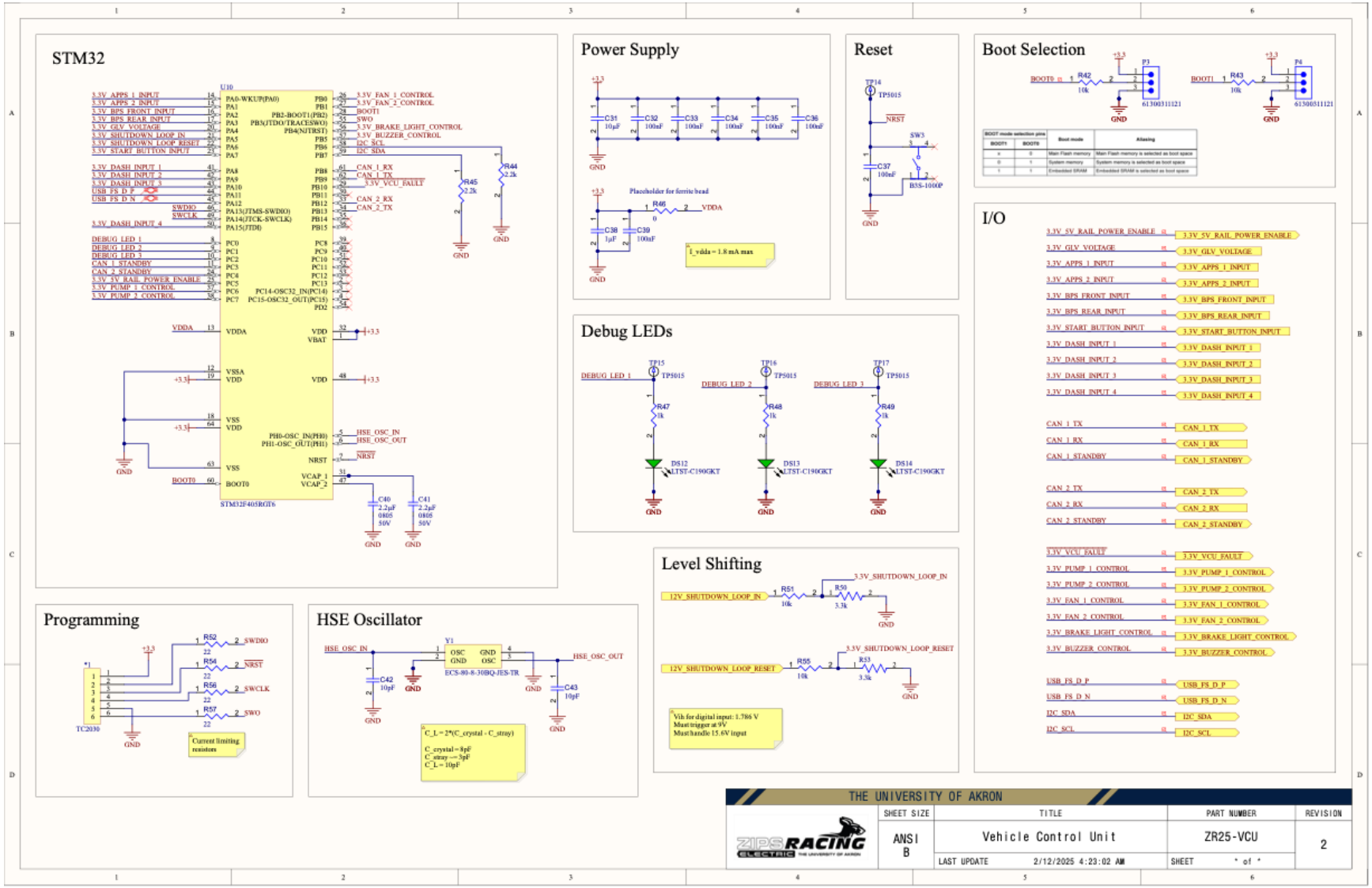

Figure 33: Microcontroller Schematic Sheet

Software (JW/TE/BG/RS)

Control System

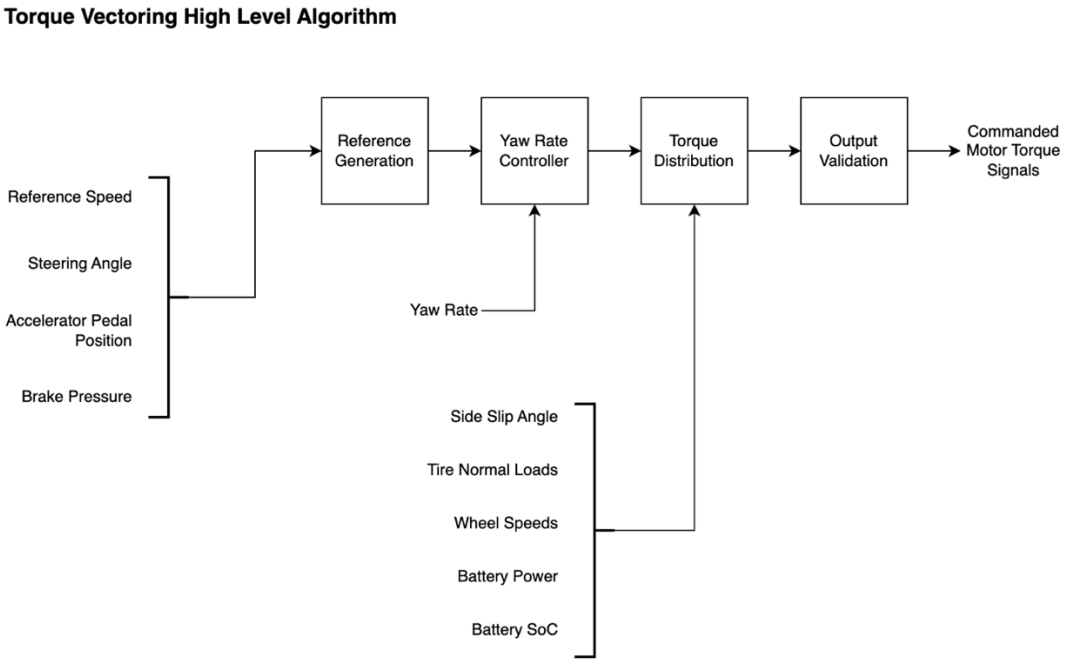

Figure 34: High level software flow diagram for torque vectoring algorithms

The diagram in Figure 34 describes the data flow used to implement the torque vectoring algorithm. The Reference Generation block computes the current state of the car using various sensors and determines what the target yaw-rate reference value should be. The Yaw Rate Controller, implemented in Simulink as shown in Figure 35, describes an algorithm to compute the torque vectoring (TV) yaw moment required to control the vehicle. The Torque Distribution step takes in the TV yaw moment as an input and runs an optimization algorithm to determine the best distribution of torques across the electric motors to achieve the TV yaw moment. The Output Validation step checks the desired motor torques, to verify the control algorithm is producing reasonable valid torque commands.

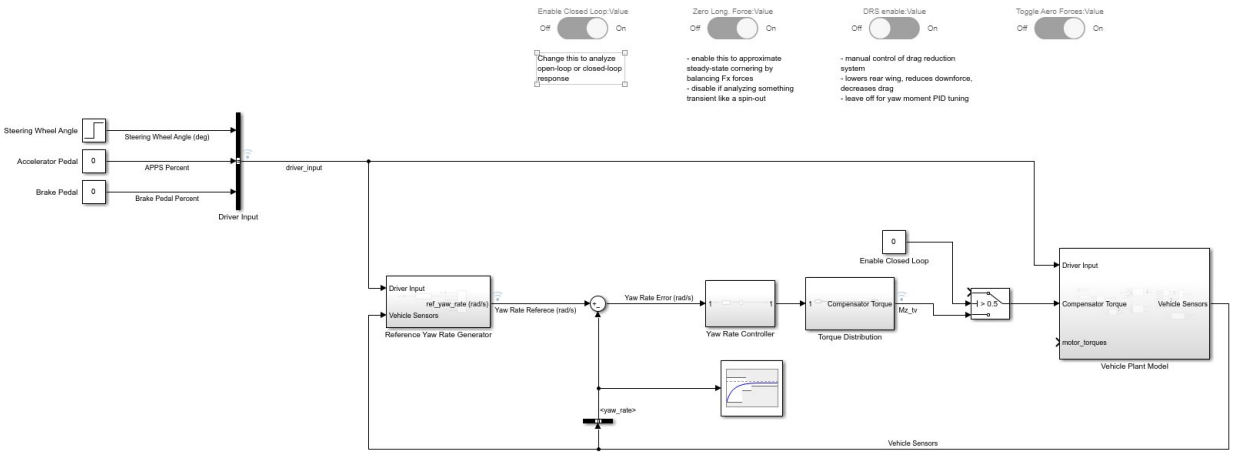

Figure 35: Simulink model of the controller and vehicle plant

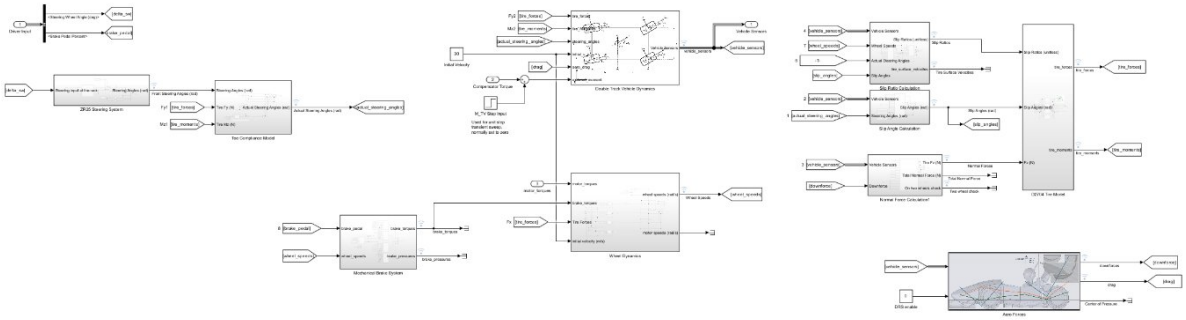

Figure 36: Simulink model of the vehicle plant

Yaw Controller (RS)

The yaw controller was designed using MATLAB and Simulink. First, the plant model was developed using the non-linear double track vehicle equations. The plant model was also supplemented with tire dynamics, wheel / motor dynamics, aero-drag, vehicle body stiffness, etc. A more detailed model of the plant is shown in Figure 36. Once the vehicle plant model was completed and the vehicle yaw rate could be modeled, a reference yaw rate must be calculated.

Using the ideal single track vehicle model described earlier, the reference yaw rate is calculated using the following equation:

$$\dot{\psi}_{ref} = \frac{V}{l + K_u V^2} + \delta_{dyn}$$

$$\dot{\psi}_{max} = \sigma \mu g$$

Where V is reference velocity, l is the vehicle wheelbase, K_u is the desired understeer gradient, and δ_{dyn} is the average dynamic steer angle of the front tires. This reference varies based on the operating point of the car and is always limited to $\dot{\psi}_{max}$, which represents the maximum physically attainable yaw rate, based on an estimate for the vehicle's maximum tire friction. The tunable parameter σ allows $\dot{\psi}_{max}$ to be adjusted for different grip conditions, such as racing in the rain. Figure 37 shows the reference yaw rate for several different operating points. The target yaw rate was chosen based on a neutral understeering gradient, when $K_u = 0$.

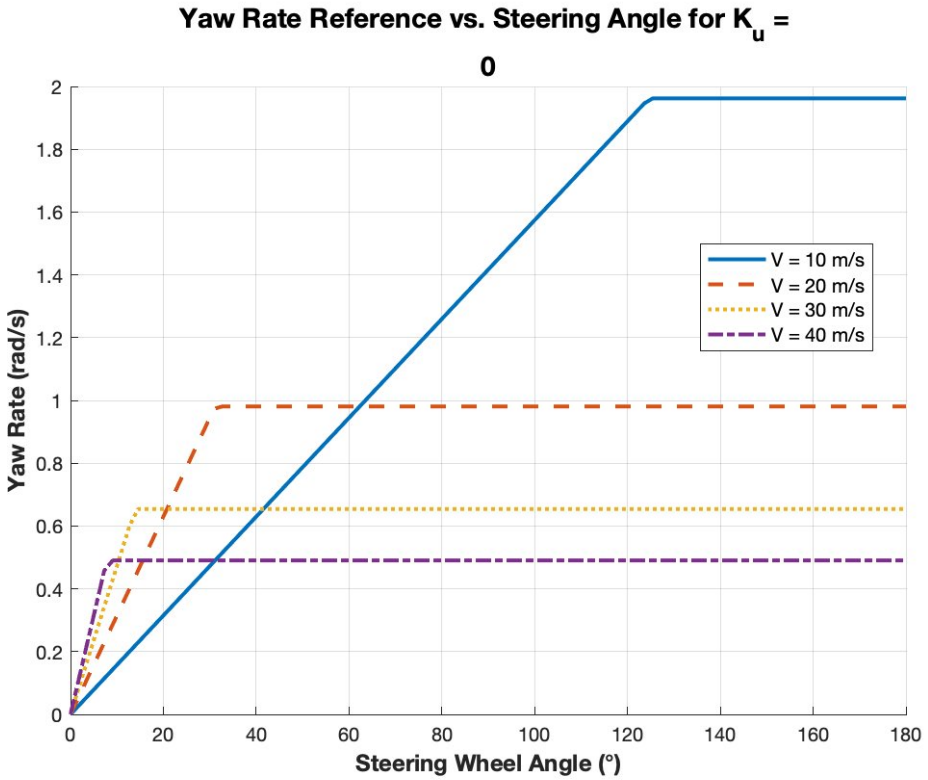

Figure 37: Chart showing the reference yaw rate at different operating points

This target yaw rate is then compared to the transient yaw rate response to calculate the yaw rate error. Because the non-linearity of plant model, a gain-scheduled PI controller was chosen as the control method.

Next, the yaw controller was tuned using the Simulink model. The scheduling parameters chosen were reference velocity and steering angle, which significantly affect the natural response of the vehicle to a step-steer input. Due to the complex nature of the model, with a transient response coming from the combination of feedforward steering input and the yaw controller, a gradient-descent approach for directly tuning the non-linear model was chosen, using MATLAB's Response Optimizer. Proportional and integral gains of the controller were tuned concurrently at each operating point until the desired step response was achieved.

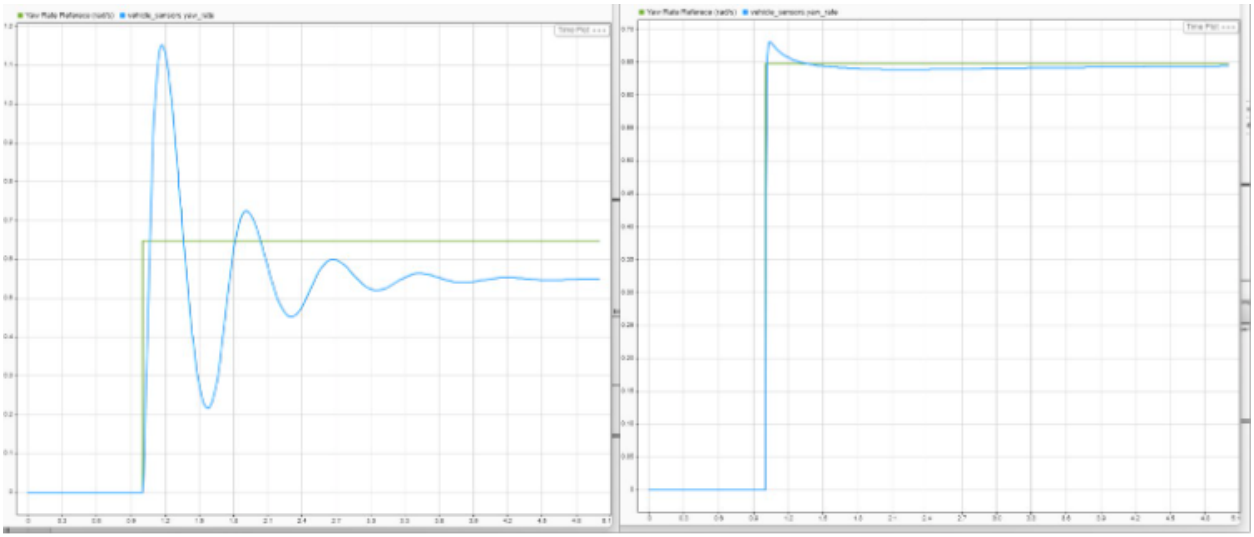

Figure 38: Graph showing the uncompensated step response (left) and the compensated response (right) with the blue being the yaw rate of the vehicle and the green being the reference

The step response was tuned to comply with the following parameters:

Rise Time (s)	1 s
Settling Time (s)	2 s
Percent Overshoot	10 %
Percent Rise	80 %
Percent Settling	1 %

Percent Undershoot	1 %

Table 27: Step response requirements.

As shown in Figure 38, the step response of the yaw rate is greatly improved once the closed loop PI controller is active. Thirty different operating points were chosen, and the gains tuned for each point. Once the gains for each point were tuned, MATLAB was used to interpolate the gains in between operating points. These gain maps are shown in Figures 39 and 40. This allows the PI controller gains to be continuously adjusted during vehicle operation. It should be noted that the gain-scheduled PI controller can only guarantee stability at the tuned operating points, and that proper control requires enough operating point resolution to uncover all the dynamics within the expected speed and steering angle range for the vehicle.

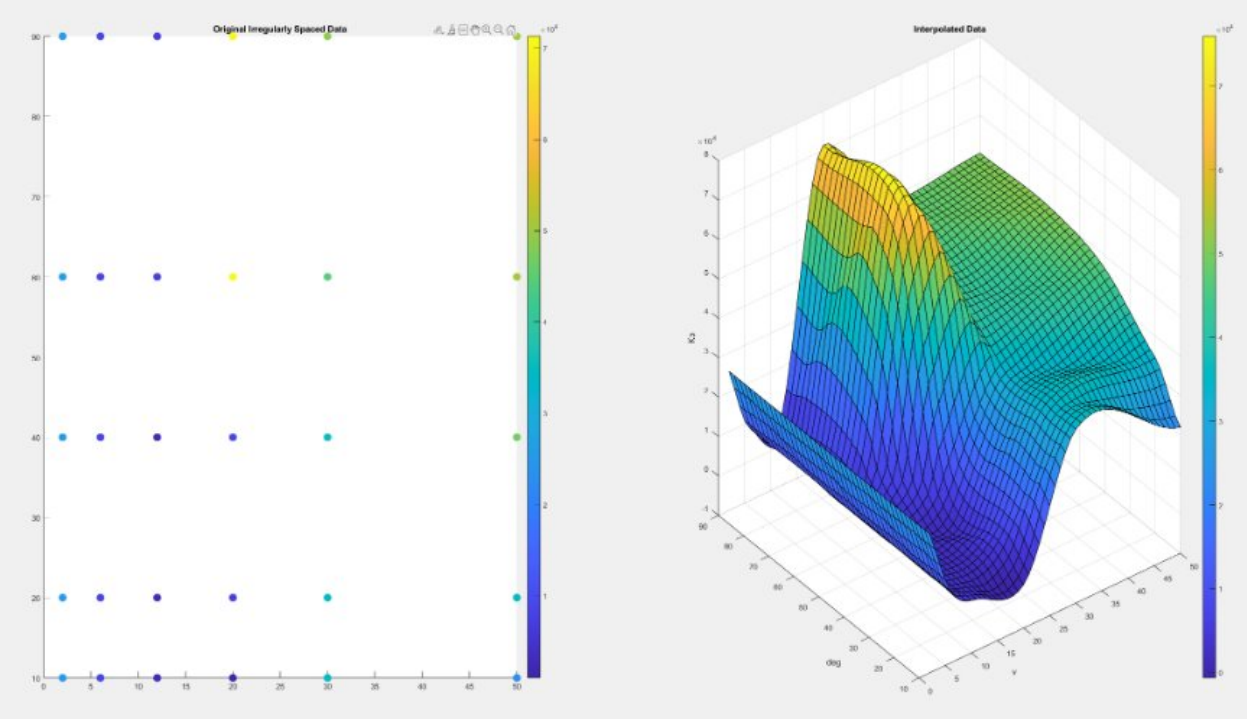

Figure 39: Proportional controller gains

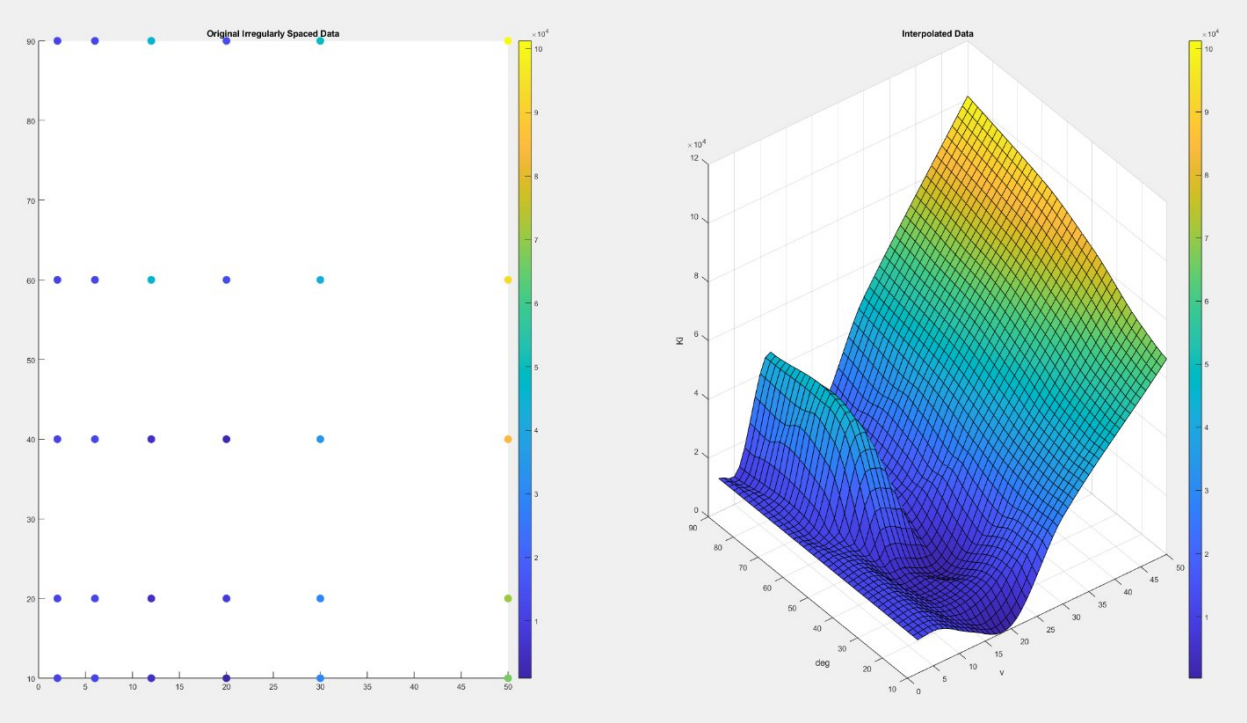

Figure 40: Integral controller gains

To implement the yaw rate controller on the hardware, a Tustin transformation was performed on the PI transfer function to bring the controller into the discrete domain. Using the discrete transfer function, a difference equation was implemented on the microcontroller. The output of the yaw controller gives a desired moment. When this moment is applied, the car will achieve the target yaw rate. To implement this properly, torque optimization algorithms were used.

Torque Optimization (BG)

With multi objectives to achieve, combined with an over-actuated vehicle platform and multiple physical and safety limitations, the torque command problem was structed as a multi-objective constrained optimization problem, and a quadratic programming (QP) solver was selected to solve the optimization problem in real-time on the microcontroller. These objectives can be described in a weighted-sum method as:

$$minimize \ \ J$$

$$subject\ to\ l_i \leq A_i(\tau) \leq u_i$$

where

$$J = w_1 \frac{f_1(\tau)}{a_1} + \ w_2 * \frac{f_2(\tau)}{a_2} + \cdots w_n * \frac{f_n(\tau)}{a_n}$$

J, the cost function, is made up of objective functions f, with w weights and a scaling factors. The weights sum to 100% so that they can be intuitively described as percentages. The scaling factors normalize each objective function's output value so the weight can take on a meaningful value. $w_n a_n$ is the true weight of each objective function. Each objective function is dependent on the set of decision variables, which are the motor torque commands $\tau \in \mathbb{R}^4$, where $\tau[n] = \ \tau_i$ for $\ i \ \in \{FL, FR, RL, RR\}$.

The objectives are described as follows:

Objective Function	Description
$f_1 = \left(M_{z,tv} - M_z\right)^2$	Achieve the yaw moment requirement from high level yaw controller.
$f_2 = \left(a_{x,ref} - a_x\right)^2$	Achieve the longitudinal acceleration requirement from APPS sensor.
$f_3 = SL_{FL}^2 + \ SL_{FR}^2 + \ SL_{RL}^2 + \ SL_{RR}^2$	Minimize the slip ratio magnitudes to avoid tire saturation and excess wear.

Table 28: Description of objective functions for torque optimization.

The constraints A_i are limited to a minimum l_i and maximum u_i and are described as follows:

Constraint	Description
$\tau_{min,i} \leq \tau_i \leq \tau_{max,i}$ for $i \in \{FL, FR, RL, RR\}$	Solution must be constrained to individual torque limits for each motor, which are found from the electric motor lookup tables, motor speed, motor temperature, and DC bus voltage. These are instantaneous limits, meaning they change every time step. Since each limit is unique for each motor, these are four separate constraints.
$P_{min} \leq \frac{1}{\eta_{batt} + \eta_{inverter}} \sum_{j=0}^{4} \frac{\tau_j \omega_j}{\eta_j} \leq P_{max}$	Solution must be constrained to a maximum and minimum power limit. This power limit is the power draw at the accumulator, so the solution's power draw is found from each motor's torque output and efficiency at that operating condition, motor speed, as well as battery and inverter efficiencies. Maximum and minimum power limits are set by the driver.
$\tau_{min} \leq \sum_{j=0}^{4} \tau_j \leq \tau_{max}$	The solution must be constrained to a maximum and minimum torque limit. These are set by the driver. This is mainly used in practice to limit forward acceleration without constraining torque used to yaw the vehicle.
$SL_{min} \leq SL_i \leq SL_{max}$ for $i \in \{FL, FR, RL, RR\}$	The solution must be constrained to maximum and minimum slip angles. This is done to avoid selecting a high slip angle (~

	> 0.3) as a solution to reducing tire longitudinal force. This ultimately limits the amount of wheel slip allowed.

Table 29: Description of torque optimization constraints.

Both the objective functions and the constraint equations must be described as functions of the motor torque commands. To simplify the optimization and solve for a solution in real-time, the vehicle dynamics are linearized around the current vehicle operating point. A local linearization technique is used. This involves:

1. Linearizing the system around the current operating point and computing the objective functions and constraints.
2. Solving for the new torque solution via a QP solver.
3. Solve for the new slip ratio operating points at the torque solution by using an inverse tire model lookup table.
4. Comparing the error between the expected objective outputs and the objective outputs at the new slip ratio operating point.
5. Repeat the optimization if the error is above a threshold, or if an optimization cycle limit has not yet been hit. If re-optimization is necessary, we start at the slip ratio operating point of the last solution.

This process may require multiple optimization cycles, but it allows the non-linearity of the vehicle dynamics to be reflected in the solution. Additionally, by linearizing the vehicle dynamics and keeping the objective function form to be $f = (AT - b)^2$, the optimization problem format will always be convex, which is a requirement for most of the QP solvers.

CVXGEN is the QP solver selected. It is an online tool which can be used to generate solver code in C and MATLAB for convex optimization problems, and is specifically intended for use on embedded systems. CVXGEN was used first for its simplicity. The code used to generate the solver is:

```
dimensions
  # aka size of arrays to represent problem with
  n = 4 # number of decision variables
  m = 4 # number of constraints
end

parameters
  # placeholders for problem data to be filled in with the generated solver
  P(n, n) symmetric psd # matrix which defines the quadratic part of the objective
functions
  q(n) # vector defining the linear part of the objective functions
  A(m, n) # matrix representing the linear constraints. Each row represents one
constraint
  l(m, 1) # column vector representing the lower bound for each constraint
  u(m, 1) # column vector representing the upper bound for each constraint
end

variables
  x(n) # vector of decision variables

end

minimize
  (1/2) * quad(x, P) + q'*x # typical QP form, also identical to OSQP if we want to
switch / compare solvers
subject to
  l <= A*x <= u
end
```

Figure 41: CVXGEN solver generation code.

To use the solver, the objective optimization problem is re-written in a quadratic form:

$$\begin{aligned} &minimize && \frac{1}{2}x^T Px + q^T x \\ &subject\ to && l \leq Ax \leq u \end{aligned}$$

Where P is a matrix defining the quadratic part of the objective function, and q vector defines the linear components of the objective function. A is a matrix with each row representing a

constraint, and x is a vector containing the decision variables, the four torque commands. MATLAB's symbolic toolbox was used to compute the matrices for the QP problem form.

Real-time Operating System (TE/JW)

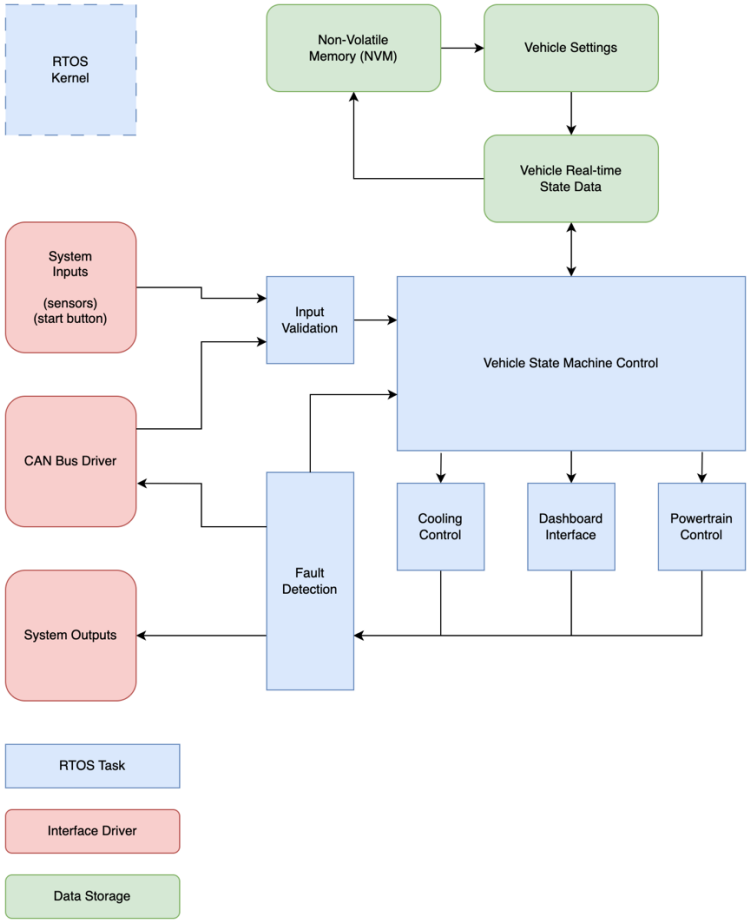

Figure 42: High level software task diagram for the microcontroller

The task diagram in Figure 42 describes the flow of information between individual software tasks, which are scheduled and timed by a Real-Time Operating System (RTOS). Blocks with rounded corners are external sensors and I/O, whereas blocks with sharp corners indicate tasks. Some key tasks include:

- Vehicle State Data Task, which maintains a record of all current variables needed to represent the state of the vehicle, such as velocity, sensor readings, etc., and signals to other tasks when a variable they are watching is changed.
- Vehicle State Machine, which implements the high-level state diagram described in Figure 43.
- Motor Control / Braking, which implements the torque control algorithm in Figure 34.
- Fault Detector, which watches for abnormal vehicle states, which could indicate a hardware fault, and opens the shutdown loop if necessary.

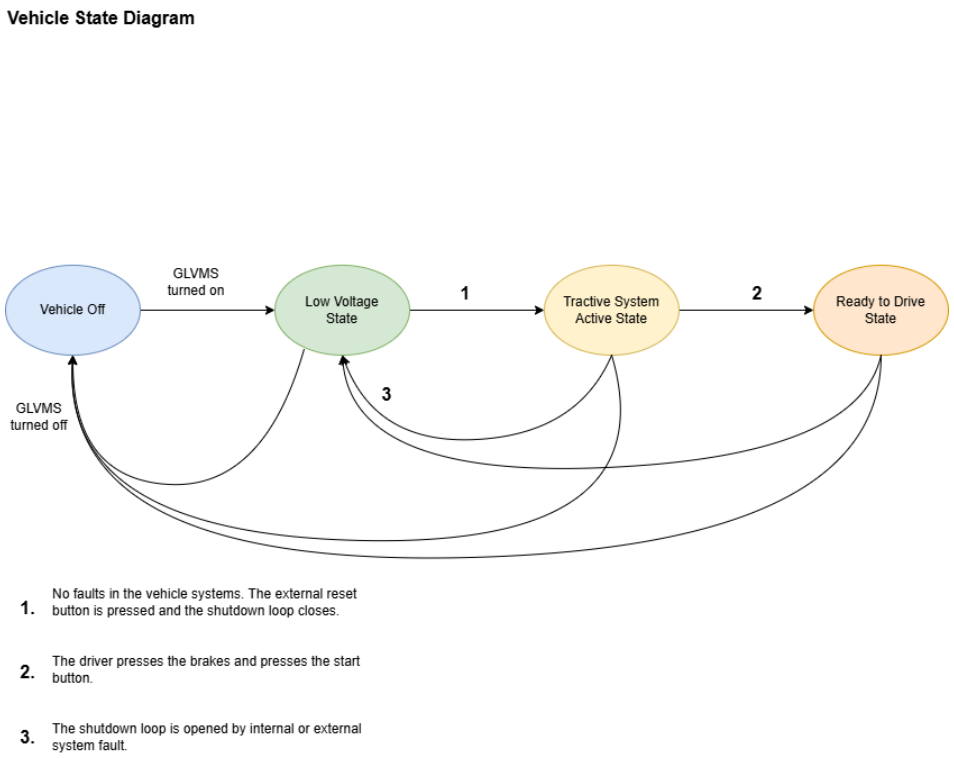

Figure 43: High level state transition diagram for the vehicle state machine

Power Sequencing and Startup Software

```c
#include "main.h"
#include "stm32f4xx_hal_adc.h"

// ADC thresholds for 5V and 3.3V rails
#define ADC_5V_THRESHOLD_LOW   3818  // ADC value for 4.8V
#define ADC_5V_THRESHOLD_HIGH  4095  // ADC value for 5.2V
#define ADC_3V3_THRESHOLD_LOW  3276  // ADC value for 2.8V
#define ADC_3V3_THRESHOLD_HIGH 4095  // ADC value for 3.5V

ADC_HandleTypeDef hadc1;

static void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC1_Init(void);

int main(void)
{
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_ADC1_Init();

    uint32_t adc_value_5V = 0;
    uint32_t adc_value_3V3 = 0;

    while (1)
    {
        // Read 5V signal on PA0
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
        adc_value_5V = HAL_ADC_GetValue(&hadc1);

        // Check if 5V signal within range
        if (adc_value_5V >= ADC_5V_THRESHOLD_LOW && adc_value_5V <= ADC_5V_THRESHOLD_HIGH) {
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_SET);
        } else {
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
        }

        // Change to PA1 for 3.3V signal
        ADC_ChannelConfTypeDef sConfig = {0};
        sConfig.Channel = ADC_CHANNEL_1;
        sConfig.Rank = 1;
        sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
        HAL_ADC_ConfigChannel(&hadc1, &sConfig);

        // Read 3.3V signal on PA1
        HAL_ADC_Start(&hadc1);
        HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
        adc_value_3V3 = HAL_ADC_GetValue(&hadc1);

        // Check if 3.3V signal within the range
        if (adc_value_3V3 >= ADC_3V3_THRESHOLD_LOW && adc_value_3V3 <= ADC_3V3_THRESHOLD_HIGH) {
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_SET);
        } else {
            HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
        }

        // Change to PA0 for next check
        sConfig.Channel = ADC_CHANNEL_0;
        sConfig.Rank = 1;
        sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
        HAL_ADC_ConfigChannel(&hadc1, &sConfig);

        HAL_ADC_Stop(&hadc1);
        HAL_Delay(100);
    }
}

static void MX_ADC1_Init(void)
{
    ADC_ChannelConfTypeDef sConfig = {0};

    hadc1.Instance = ADC1;
    hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
    hadc1.Init.Resolution = ADC_RESOLUTION_12B; // 4095
    hadc1.Init.ScanConvMode = DISABLE;
    hadc1.Init.ContinuousConvMode = DISABLE;
    hadc1.Init.DiscontinuousConvMode = DISABLE;
    hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
    hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
    hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
    hadc1.Init.NbrOfConversion = 1;
    hadc1.Init.DMAContinuousRequests = DISABLE;
    hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;

    if (HAL_ADC_Init(&hadc1) != HAL_OK) {
        Error_Handler();
    }

    // Configure ADC channel to PA0
    sConfig.Channel = ADC_CHANNEL_0;
    sConfig.Rank = 1;
    sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
    HAL_ADC_ConfigChannel(&hadc1, &sConfig);
}
```

Figure 44: Power Supply Regulation code part 1

```c
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};
    __HAL_RCC_GPIOD_CLK_ENABLE();

    GPIO_InitStruct.Pin = GPIO_PIN_12 | GPIO_PIN_14;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, GPIO_PIN_RESET);
}

static void SystemClock_Config(void)
{
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  RCC_OscInitTypeDef RCC_OscInitStruct;

  /* Enable Power Control clock */
  __HAL_RCC_PWR_CLK_ENABLE();

  /* The voltage scaling allows optimizing the power consumption when the device is
     clocked below the maximum system frequency, to update the voltage scaling value
     regarding system frequency refer to product datasheet.  */
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /* Enable HSE Oscillator and activate PLL with HSE as source */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  HAL_RCC_OscConfig(&RCC_OscInitStruct);

  /* Select PLL as system clock source and configure the HCLK, PCLK1 and PCLK2
     clocks dividers */
  RCC_ClkInitStruct.ClockType = (RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2);
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
  HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);

  /* STM32F405x/407x/415x/417x Revision Z and upper devices: prefetch is supported  */
  if (HAL_GetREVID() >= 0x1001)
  {
    /* Enable the Flash prefetch */
    __HAL_FLASH_PREFETCH_BUFFER_ENABLE();
  }
}

void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
```

Figure 45: Power Supply Regulation code part 2

The C code in Figures 44 and 45 measures the output of a power supply regulator's 3.3V and 5V rails. These outputs are scaled down using voltage dividers to approximately 2.5V to ensure compatibility with the STM32's ADC, which can only measure up to 3.3V. Scaling to 2.5V provides a margin for proper analysis and tolerances. An enable signal is then output if the measured rails are within safe voltage ranges; for demonstration purposes, this is indicated by LEDs lighting up or turning off.

The code begins by initializing various STM32 peripherals, including the Hardware Abstraction Layer (HAL), system clock, GPIO, and ADC. The main loop executes every 100ms, polling the ADC to read the 3.3V and 5V rails. This involves switching the ADC to the appropriate channel, polling it for a conversion, and comparing the result to predefined voltage thresholds. Each channel is processed sequentially.

The ADC initialization function configures parameters such as clock pre-scalers, resolution, and other settings, and sets the default channel to ADC channel 0. The GPIO initialization function sets up the STM32 LEDs so they can be toggled during the main loop. The system clock is configured using baseline parameters from STM32 demonstration code. Lastly, an empty error handler function is included, which will be further defined in the future.

CAN Bus Software

```
//------------------------------
// File generated by generate_can_code.py on: 2025-03-11 14:36:24
//------------------------------

#ifndef INC_CAN_MESSAGES_H_
#define INC_CAN_MESSAGES_H_

#include "can_db.h"

#ifdef __GNUC__
#define PACKED __attribute__((packed))
#else
#define PACKED
#endif

#define CAN_DB_VCU_STATUS_ID 256
typedef union {
    uint64_t as_u64;
    struct PACKED {
        uint64_t VEHICLE_STATE : 2;
        uint64_t TORQUE_PLAUSIBLE : 1;
        uint64_t PEDALS_PLAUSIBLE : 1;
        uint64_t TORQUE_DERATING : 1;
        uint64_t VCU_EEPROM_STATE : 2;
        uint64_t CAN_PLAUSIBLE : 1;
        uint64_t APPS_1_PLAUSIBLE : 1;
        uint64_t APPS_1_CONFIG_PLAUSIBLE : 1;
        uint64_t APPS_2_PLAUSIBLE : 1;
        uint64_t APPS_2_CONFIG_PLAUSIBLE : 1;
        uint64_t BSE_F_PLAUSIBLE : 1;
        uint64_t BSE_F_CONFIG_PLAUSIBLE : 1;
        uint64_t BSE_R_PLAUSIBLE : 1;
        uint64_t BSE_R_CONFIG_PLAUSIBLE : 1;
        uint64_t AMK_RL_VALID : 1;
        uint64_t AMK_RR_VALID : 1;
        uint64_t AMK_FL_VALID : 1;
        uint64_t AMK_FR_VALID : 1;
        uint64_t GPS_VALID : 1;
        uint64_t _reserved21 : 3;
        uint64_t GLV_BATTERY_VOLTAGE : 8;
    } fields;
} CANMessage_VCU_STATUS;
```

Figure 46: Generated code in can_messages.h

```c
// Each CANDatabaseMessage_t represents the contents of a message
// from/to a certain can_id. The signals are a list of bit fields
// which the message may be split into.

typedef struct {
        const char *name;
        int bitpos;
        uint64_t bitmask;
} CANDatabaseSignal_t;

typedef struct {
        const char *name;
        uint32_t can_id;
        int num_signals;
        CANDatabaseSignal_t *signals;
} CANDatabaseMessage_t;

typedef void (*CANCallback_t)(uint32_t can_id, uint64_t messageContents, void* custom);

typedef struct {
        int message_count;
        CANDatabaseMessage_t *messages; // sorted by can_id
        uint64_t *message_contents;
        bool *message_contents_valid;
        CANCallback_t *callbacks;
        void **callback_payloads;
} CANDatabase_t;

extern CANDatabase_t can_db;
extern CAN_HandleTypeDef hcan1;
extern CAN_HandleTypeDef hcan2;

void initCANDatabase();

typedef uint32_t CANDatabaseEntryId;
```

```c
void initCANDatabase();

typedef uint32_t CANDatabaseEntryId;

// Returns -1 if entry does not exist
CANDatabaseEntryId CANGetDbEntry(uint32_t can_id);

// Must acquire can_db.message_contents_lock first.
// Returns false if message not valid.
bool CANGetMessageContents(CANDatabaseEntryId entry_id, uint64_t *out);

//// Must acquire can_db.message_contents_lock first.
//void CANSetMessageContents(CANDatabaseEntryId entry_id, uint64_t contents);

// Returns true once message is successfully queued.
// Does not need the can_db lock to be acquired.
// Does not set the contents of the message in the db.
bool CANQueueMessageToSend(CANDatabaseEntryId entry_id, uint64_t contents, CAN_HandleTypeDef* hcan);

// Must acquire can_db.message_contents_lock first.
// Have a function be called whenever a message with a given id is received.
// Returns false if another callback has already been registered and was overwritten.
bool CANRegisterCallback(CANDatabaseEntryId entry_id, CANCallback_t callback, void *custom_argument);

// Returns true if message recognized by CAN DB
bool CANIRQRxHandler(CAN_RxHeaderTypeDef *header, uint8_t rx_data[8]);

const static osThreadAttr_t can_task_attrs = {
        .name = "CAN_DB_Task",
        .stack_size = 128 * 4,
    .priority = (osPriority_t) osPriorityNormal,
};

void StartCanDbTask(void* _argument);
```

Figure 47: CAN controller interface in can_db.h

The STM32F405 has two CAN communication interfaces, which are both managed by can_db.c. The file describes a data structure in which pre-defined CAN messages can be stored, tasks may queue outgoing messages, and tasks may subscribe to incoming messages by providing a callback function. At the root of the project folder there is a python script which may take a .DBC file (a standardized file format for describing CAN messages) as input and produce a C header file containing structs corresponding to CAN messages. The can_messages.h header file contains these generated type definitions and is used in controller code that may use the CAN interface to control peripherals.

Vehicle Data

```
extern osMutexId_t vdb_apps_lockHandle;
extern osMutexId_t vdb_bps_front_lockHandle;
extern osMutexId_t vdb_bps_rear_lockHandle;
extern osMutexId_t vdb_sas_lockHandle;
extern osMutexId_t vdb_inverter_lockHandle;
extern osMutexId_t vdb_fsm_state_lockHandle;
extern osMutexId_t vdb_powsup_lockHandle;
extern osMutexId_t vdb_cooling_lockHandle;
extern osMutexId_t vdb_dashboard_lockHandle;
extern osMutexId_t vdb_torquectrl_lockHandle;
extern osMutexId_t vdb_faulttask_lockHandle;
extern osMutexId_t vdb_gps_lockHandle;
extern osMutexId_t vdb_defaultTask_lockHandle;

typedef struct {
  APPSSensor_t apps;
  BPSSensor_t bps_front;
  BPSSensor_t bps_rear;
  SteeringAngleSensor_t sas;
  AMKState_t inverter;
  VCU_State_t fsm_state;
  PowSupData_t powsup;
  DashboardData_t dashboard;
  TorqueCtrlData_t torquectrl;
  GPSState_t gps;
  FaultType_t faultmgmt;
  StrainGaugeData_t strain_gauge;
  BMSData_t bms;
} VehicleData_t;

extern VehicleData_t VehicleData;

void initVehicleData();
```

Figure 48: Vehicle Data Header Code

The vehicle data task is crucial to the operation of the VCU, as it allows each task to receive and update all vehicle data relevant to the tasks' functionality. The VehicleData_t structure contains all data structures from the various tasks and initializes a global VehicleData variable as well as mutex locks for each task. This ensures that the tasks can update the vehicle data without race conditions.

Fault Management

```
void update_fault_management_data(FaultType_t fault){
    osMutexAcquire(vdb_faulttask_lockHandle, osWaitForever);
    VehicleData.faultmgmt = fault;
    osMutexRelease(vdb_faulttask_lockHandle);
}

void StartFaultTask(void *argument){
    thread_id = osThreadGetId();
    FaultType_t fault = {.faultBits = FAULTS_NONE};
    osThreadFlagsSet(thread_id, fault.faultInt);
    for(;;){
        fault.faultInt = osThreadFlagsGet();

        fault_check();
        fault_callback();
        fault_clear_flags();
        update_fault_management_data(fault);

        osDelay(FAULT_MGMT_TASK_PERIOD);
    }
}

void fault_callback(){
    FaultType_t fault = {.faultBits = FAULTS_NONE};
    fault.faultInt = osThreadFlagsGet();

    uint32_t criticalFaults = 0;
    uint32_t nonCriticalFaults = 0;

    for (uint8_t i = 0; i < NUM_FAULTS; i++) {
        if (fault.faultInt & (1 << i)) {
            if (fault_critical[i]) {
                criticalFaults |= (1 << i);
            } else {
                nonCriticalFaults |= (1 << i);
            }
        }
    }

    if(criticalFaults){
        fsm_flag_callback(FLAG_INDEX_FAULT_DETECTED, 1);
    }
    else {
        fsm_flag_callback(FLAG_INDEX_FAULT_DETECTED, 0);
    }

    if(nonCriticalFaults || criticalFaults){
        DashboardFaultCallback(1);
    }
    else {
        DashboardFaultCallback(0);
    }
}

void fault_check(){
    FaultType_t fault = {.faultBits = FAULTS_NONE};

    apps_bps_implausibility_check(&fault);
    sas_implausibility_check(&fault);
    gps_check(&fault);
    inverter_check(&fault);
    bms_check(&fault);
    vcu_check(&fault);
    strain_gauge_check(&fault);
```

Figure 49: Fault Management Code Part 1

```
void strain_gauge_check(FaultType_t *fault) {
    uint8_t strain_plausible = 1;
    VehicleData_t local_vehicle_data = {0};

    osMutexAcquire(vdb_defaultTask_lockHandle, osWaitForever);
    local_vehicle_data.bms = VehicleData.bms;
    osMutexRelease(vdb_defaultTask_lockHandle);

    if (local_vehicle_data.strain_gauge.fl_tire_load < STRAIN_FL_LOAD_MIN || local_vehicle_data.strain_gauge.fl_tire_load > STRAIN_FL_LOAD_MAX) {
        strain_plausible = 0;
    }

    if (local_vehicle_data.strain_gauge.fr_tire_load < STRAIN_FR_LOAD_MIN || local_vehicle_data.strain_gauge.fr_tire_load > STRAIN_FR_LOAD_MAX) {
        strain_plausible = 0;
    }

    if (local_vehicle_data.strain_gauge.rl_tire_load < STRAIN_RL_LOAD_MIN || local_vehicle_data.strain_gauge.rl_tire_load > STRAIN_RL_LOAD_MAX) {
        strain_plausible = 0;
    }

    if (local_vehicle_data.strain_gauge.rr_tire_load < STRAIN_RR_LOAD_MIN || local_vehicle_data.strain_gauge.rr_tire_load > STRAIN_RR_LOAD_MAX) {
        strain_plausible = 0;
    }

    local_vehicle_data.strain_gauge.plausible = strain_plausible;
    if (!strain_plausible) {
        fault->faultBits.Fault_strain_gauge = 1;
    }
    else {
        faultsToClear.faultInt |= (1 << FAULT_INDEX_STRAIN_GAUGE_FAILURE);
    }
}

void fault_clear_flags(){
    osThreadFlagsClear(faultsToClear.faultInt);
    faultsToClear.faultBits = FAULTS_NONE;
}

void fault_flag_callback(uint8_t fault, uint8_t value){
    FaultType_t faults = {.faultBits = FAULTS_NONE};
    if (value){
        if(fault == (FAULT_INDEX_BMS_COM_FAILURE | FAULT_INDEX_GPS_COM_FAILURE)){
            ControlMode_t ctrl_mode = 0;
            update_control_mode(ctrl_mode);
        }
    }
    else{
        faultsToClear.faultInt |= (1 << fault);
    }

    osThreadFlagsSet(thread_id, faults.faultInt);
}
```

Figure 50: Fault Management Code Part 2

The fault management task's overall flow, seen in Figure 49, is to check for each fault, update the dashboard and/or FSM on any faults that occur, clear any flags for faults that are no longer present, and update the fault management data in the VehicleData. An example of a fault check can be seen in the straing_gauge_check function in Figure 50; The relevant vehicle data is retrieved, then a set of conditionals are checked to determine if the sensor in question is in a fault state.

Driver Sensor Software

```c
void StartDriverSensorTask(
    void *void_args
){
    DriverSensorTaskArgs_t* args = (DriverSensorTaskArgs_t*)void_args;
    ADC_HandleTypeDef hadc1 = args->hadc1;
    while (1) {
        read_driver_input(&hadc1);
        print_driver_input();
        update_driver_sensor_data();
        fsm_sensor_callback();

        osDelay(50);
    }
}

/*
 * Initializes the driver input sensors
 */
void init_driver_input(I2C_HandleTypeDef *i2c)
{
    // Initialize the AM4096
    (void)am4096_init(&s_steering_angle.i2c_device, i2c);
}

/*
 * Reads in the raw sensor data and updates the sensor variables
 */
void read_driver_input(ADC_HandleTypeDef *adc)
{
    // Read Raw ADC Values
    s_apps.raw_value_1 = read_adc(adc, APPS_1_CHANNEL);
    s_apps.raw_value_2 = read_adc(adc, APPS_2_CHANNEL);
    s_bps_front.raw_value = read_adc(adc, BPS_FRONT_CHANNEL);
    s_bps_rear.raw_value = read_adc(adc, BPS_REAR_CHANNEL);

    // Get raw steering angle values
    (void)am4096_read_angle(&s_steering_angle.i2c_device);
    (void)am4096_read_angular_velocity(&s_steering_angle.i2c_device);

    // Convert Raw Values to Voltages
    s_apps.voltage_1 = adc_to_voltage(s_apps.raw_value_1);
    s_apps.voltage_2 = adc_to_voltage(s_apps.raw_value_2);
    s_bps_front.voltage = adc_to_voltage(s_bps_front.raw_value);
    s_bps_rear.voltage = adc_to_voltage(s_bps_rear.raw_value);

    // Convert Voltages to Physical Values
    (void)calc_apps_percent(&s_apps);
    (void)calc_bps_pressure(&s_bps_front);
    (void)calc_bps_pressure(&s_bps_rear);

    // Perform Plausibility Checking
    // Handling of plausibility is outside the scope of measuring driver input sensors
    s_apps.plausible = validate_apps(s_apps);
    s_bps_front.plausible = validate_bps(s_bps_front);
    s_bps_rear.plausible = validate_bps(s_bps_rear);
    s_steering_angle.plausible = validate_steering_angle(s_steering_angle);
}
```

Figure 51: Driver Sensor Code

The driver sensor task overall flow, seen in Figure 51, is to read all driver inputs, print those inputs the USB C port for debugging, update the driver sensor vehicle data, and perform any necessary callbacks to the FSM. To read the driver input, the raw values are first read directly from the ADC channels, as well as the steering angle through I2C. Then these raw values are

converted to voltages based on a minimum, maximum, and threshold voltage. The acceleration pedal travel percentage as well as the brake pressure is calculated, which is a factor in the validation of sensor plausibility.

Mechanical Sketch (BG)

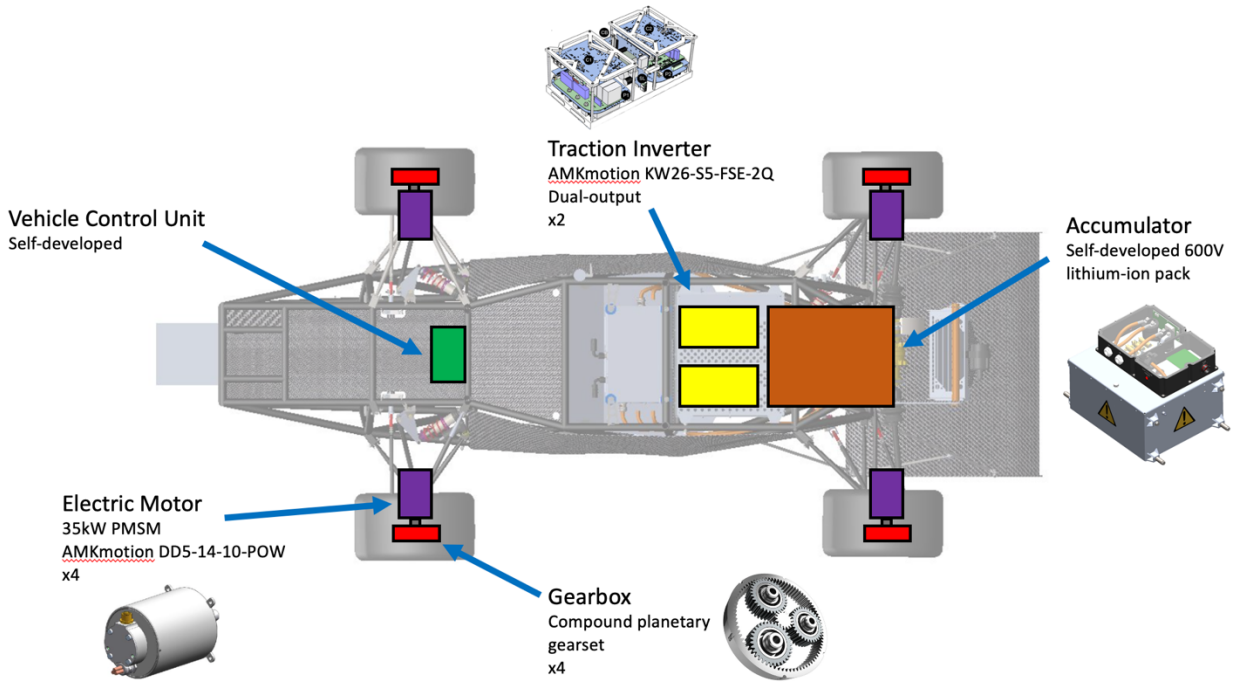

Figure 52: Mechanical sketch of overall vehicle and various subsystems

Team Information

Brian Glen, Electrical Engineering, has taken ESI Fall 2023

John Wozencraft, Computer Engineering, has taken ESI Fall 2022

Ryan Stoller, Electrical Engineering, has taken ESI Fall 2024

Tetra Engdahl, Computer Engineering, has taken ESI Fall 2023

Parts List

Table 30: Parts List

Qty.	Refdes	Part Num.	Description
2	-	TC2030-IDC	Programming Cable TC2030-IDC 6-Pin Tag-Connect Plug-of-Nails™ Spring-Pin Cable with Legs
2	-	ARM20-CTX	Programmer Adapter ARM20-CTX 20-Pin to TC2030-IDC Adapter for Cortex
6	C22, C23	C0603C472J5RACTU	CAP CER 4700PF 50V X7R 0603
66	C4, C5, C9, C10, C11, C12, C13, C14, C15, C16, C	KGM15BR71H104KT	CAP CER 0.1UF 50V X7R 0603
6	C41, C42	C0603C100C5GAC7867	CAP CER 10PF 50V C0G/NP0 0603
3	C37	0603YC105KAT2A	CAP CER 1UF 16V X7R 0603
30	C2, C3, C8, C24, C25, C26, C27, C28, C29, C30	GRM21BR61E106KA73L	CAP CER 10UF 25V X5R 0805
6	C39, C40	CL21A225KBFNNNE	CAP CER 2.2UF 50V X5R 0805
3	C7	885012205061	CAP CER 1000PF 50V X7R 0402
3	C6	CL21A226MOCLRNC	CAP CER 22UF 16V X5R 0805
3	COAT1	422C-55MLCA	CONFORMAL COATING - SILICONE WIT
9	D16, D17, D18	B160B-13-F	DIODE SCHOTTKY 60V 1A SMB
6	D14, D15	PESD1CAN-UX	TVS DIODE 24VWM 50VC SOT323
6	D2, D4	SMBJ15CA	TVS DIODE 15VWM 24.4VC DO214AA
6	D1, D3	B320B-13-F	DIODE SCHOTTKY 20V 3A SMB
42	DS1, DS2, DS3, DS4, DS5, DS6, DS7, DS8, DS9, DS1	LTST-C190GKT	LED GREEN CLEAR CHIP SMD
3	J2	M20-9990246	CONN HEADER VERT 2POS 2.54MM
30	J3, J4	382811-6	CONN SHUNT 2POS OPEN 2.54MM
3	K1	3-1462039-8	RELAY TELECOM DPDT 5A 12V
6	P1, P2	5031542690	CONN RCPT 26POS 0.059 TIN SMD
6	P3, P4	61300311121	CONN HEADER VERT 3POS 2.54MM
3	PS1	R-78E3.3-1.0	DC DC CONVERTER 3.3V 3W
18	Q1, Q2, Q3, Q4, Q5, Q6	SQS484ENW-T1_GE3	MOSFET N-CH 40V 16A PPAK1212-8
18	R4, R6, R35, R47, R48, R49	ERJ-3EKF1001V	RES SMD 1K OHM 1% 1/10W 0603

30	R10, R11, R14, R18, R19, R34, R42, R43, R51, R55	CRCW060310K0FKEA	RES SMD 10K OHM 1% 1/10W 0603
12	R26, R27, R28, R29	RC0603FR-0760R4L	RES 60.4 OHM 1% 1/10W 0603
18	R30, R31, R52, R54, R56, R57	ERJ-3EKF22R0V	RES SMD 22 OHM 1% 1/10W 0603
3	R46	RC0603JR-070RL	RES 0 OHM JUMPER 1/10W 0603
45	R2, R5, R7, R9, R15, R20, R23, R36, R37, R38, R3	RC0603FR-072K2L	RES 2.2K OHM 1% 1/10W 0603
6	SW1, SW2	JS202011JCQN	SWITCH SLIDE DPDT 300MA 6V
3	SW3	B3W-1000	SWITCH TACTILE SPST-NO 0.05A 24V
51	TP1, TP2, TP3, TP4, TP5, TP6, TP7, TP8, TP9, TP1	5015	PC TEST POINT MINIATURE
6	U2, U3	TCAN1042VDRBRQ1	IC TRANSCEIVER 1/1 8VSON
3	U10	STM32F405RGT6	IC MCU 32BIT 1MB FLASH 64LQFP
3	X*1	ECS-80-8-30BQ-JES-TR	CRYSTAL 8.0000MHZ 8PF SMD
27	D5, D6, D7, D8, D9, D10, D11, D12, D13	SL05T1G	TVS DIODE 5VWM 11VC SOT23-3
6	F1, F2	MF-MSMF050-2	PTC RESET FUSE 15V 500MA 1812
6	F3, F4	0ZCG0014FF2C	PTC RESET FUSE 60V 140MA 1812
3	J1	632723300011	CONN RCP USB3.1 TYPEC 24P SMD RA
15	R8, R12, R16, R21, R24	TC33X-2-103E	TRIMMER 10K OHM 0.15W J LEAD TOP
12	R13, R17, R22, R25	RC0603FR-07300RL	RES 300 OHM 1% 1/10W 0603
6	R32, R33	RC0603FR-075K1L	RES 5.1K OHM 1% 1/10W 0603
6	R50, R53	RC0603FR-073K3L	RES 3.3K OHM 1% 1/10W 0603
18	U4, U5, U6, U7, U8, U9	MIC5014YM	IC GATE DRVR HI/LOW SIDE 8SOIC
3	C1	C1608C0G1H470J080AA	CAP CER 47PF 50V C0G 0603
3	L1	744773033	FIXED IND 3.3UH 2A 86 MOHM SMD
3	L2	RLS-186	FIXED IND 18UH 1.89A 100MOHM SMD
3	R1	RC0603FR-07220KL	RES 220K OHM 1% 1/10W 0603
3	R3	AC0603FR-0730KL	RES SMD 30K OHM 1% 1/10W 0603
3	U1	TPS562243DRLR	IC REG BUCK ADJ 2A SOT563

Table 31: Material budget list

Qty.	Part Num.	Description	Unit Cost	Total Cost
2	TC2030-IDC	Programming Cable TC2030-IDC 6-Pin Tag-Connect Plug-of-Nails™ Spring-Pin Cable with Legs	$ 33.95	$ 67.90
2	ARM20-CTX	Programmer Adapter ARM20-CTX 20-Pin to TC2030-IDC Adapter for Cortex	$ 29.95	$ 59.90
6	C0603C472J5RACTU	CAP CER 4700PF 50V X7R 0603	$ 0.10	$ 0.60
66	KGM15BR71H104KT	CAP CER 0.1UF 50V X7R 0603	$ 0.02	$ 1.16
6	C0603C100C5GAC7867	CAP CER 10PF 50V C0G/NP0 0603	$ 0.35	$ 2.10
3	0603YC105KAT2A	CAP CER 1UF 16V X7R 0603	$ 0.18	$ 0.54
30	GRM21BR61E106KA73L	CAP CER 10UF 25V X5R 0805	$ 0.09	$ 2.58
6	CL21A225KBFNNNE	CAP CER 2.2UF 50V X5R 0805	$ 0.24	$ 1.44
3	885012205061	CAP CER 1000PF 50V X7R 0402	$ 0.10	$ 0.30
3	CL21A226MOCLRNC	CAP CER 22UF 16V X5R 0805	$ 0.31	$ 0.93
3	422C-55MLCA	CONFORMAL COATING - SILICONE WIT	$ 26.81	$ 80.43
9	B160B-13-F	DIODE SCHOTTKY 60V 1A SMB	$ 1.05	$ 9.45
6	PESD1CAN-UX	TVS DIODE 24VWM 50VC SOT323	$ 0.33	$ 1.98
6	SMBJ15CA	TVS DIODE 15VWM 24.4VC DO214AA	$ 0.35	$ 2.10
6	B320B-13-F	DIODE SCHOTTKY 20V 3A SMB	$ 0.66	$ 3.96
42	LTST-C190GKT	LED GREEN CLEAR CHIP SMD	$ 0.12	$ 4.87
3	M20-9990246	CONN HEADER VERT 2POS 2.54MM	$ 0.10	$ 0.30
30	382811-6	CONN SHUNT 2POS OPEN 2.54MM	$ 0.13	$ 3.93
3	3-1462039-8	RELAY TELECOM DPDT 5A 12V	$ 5.14	$ 15.42
6	5031542690	CONN RCPT 26POS 0.059 TIN SMD	$ 2.84	$ 17.04
6	61300311121	CONN HEADER VERT 3POS 2.54MM	$ 0.12	$ 0.72
3	R-78E3.3-1.0	DC DC CONVERTER 3.3V 3W	$ 3.92	$ 11.76
18	SQS484ENW-T1_GE3	MOSFET N-CH 40V 16A PPAK1212-8	$ 0.83	$ 15.01
18	ERJ-3EKF1001V	RES SMD 1K OHM 1% 1/10W 0603	$ 0.03	$ 0.56
30	CRCW060310K0FKEA	RES SMD 10K OHM 1% 1/10W 0603	$ 0.03	$ 0.81
12	RC0603FR-0760R4L	RES 60.4 OHM 1% 1/10W 0603	$ 0.01	$ 0.11
18	ERJ-3EKF22R0V	RES SMD 22 OHM 1% 1/10W 0603	$ 0.03	$ 0.56
3	RC0603JR-070RL	RES 0 OHM JUMPER 1/10W 0603	$ 0.10	$ 0.30

45	RC0603FR-072K2L	RES 2.2K OHM 1% 1/10W 0603	$ 0.01	$ 0.54
6	JS202011JCQN	SWITCH SLIDE DPDT 300MA 6V	$ 0.93	$ 5.58
3	B3W-1000	SWITCH TACTILE SPST-NO 0.05A 24V	$ 0.70	$ 2.10
51	5015	PC TEST POINT MINIATURE	$ 0.33	$ 16.71
6	TCAN1042VDRBRQ1	IC TRANSCEIVER 1/1 8VSON	$ 1.74	$ 10.44
3	STM32F405RGT6	IC MCU 32BIT 1MB FLASH 64LQFP	$ 11.67	$ 35.01
3	ECS-80-8-30BQ-JES-TR	CRYSTAL 8.0000MHZ 8PF SMD	$ 0.78	$ 2.34
27	SL05T1G	TVS DIODE 5VWM 11VC SOT23-3	$ 0.25	$ 6.80
6	MF-MSMF050-2	PTC RESET FUSE 15V 500MA 1812	$ 0.16	$ 0.95
6	0ZCG0014FF2C	PTC RESET FUSE 60V 140MA 1812	$ 0.16	$ 0.95
3	632723300011	CONN RCP USB3.1 TYPEC 24P SMD RA	$ 3.93	$ 11.79
15	TC33X-2-103E	TRIMMER 10K OHM 0.15W J LEAD TOP	$ 0.27	$ 4.01
12	RC0603FR-07300RL	RES 300 OHM 1% 1/10W 0603	$ 0.01	$ 0.14
6	RC0603FR-075K1L	RES 5.1K OHM 1% 1/10W 0603	$ 0.10	$ 0.60
6	RC0603FR-073K3L	RES 3.3K OHM 1% 1/10W 0603	$ 0.10	$ 0.60
18	MIC5014YM	IC GATE DRVR HI/LOW SIDE 8SOIC	$ 2.64	$ 47.52
3	C1608C0G1H470J080AA	CAP CER 47PF 50V C0G 0603	$ 0.10	$ 0.30
3	744773033	FIXED IND 3.3UH 2A 86 MOHM SMD	$ 1.62	$ 4.86
3	RLS-186	FIXED IND 18UH 1.89A 100MOHM SMD	$ 1.41	$ 4.23
3	RC0603FR-07220KL	RES 220K OHM 1% 1/10W 0603	$ 0.10	$ 0.30
3	AC0603FR-0730KL	RES SMD 30K OHM 1% 1/10W 0603	$ 0.10	$ 0.30
3	TPS562243DRLR	IC REG BUCK ADJ 2A SOT563	$ 0.22	$ 0.66
			Total	$ 463.49

Project Schedules (JW/RS)

TASK	ASSIGNED TO	PROGRESS	START	END
Revise Gantt Chart	BG / JW / RS / TE	100%	1/13/25	1/26/25
Hardware Implementation - Breadboard Components				
CAN Bus Interface	TE	100%	1/13/25	1/26/25
Power Supply Regulator	RS	100%	1/13/25	1/26/25
Sensor Interface	BG	100%	1/13/25	1/26/25
Cooling System Control	BG / RS	100%	1/13/25	1/26/25
Brake Light Control	BG / RS	100%	1/13/25	1/26/25
Ready-to-Drive Buzzer	BG / RS	100%	1/13/25	1/26/25
Shutdown Loop Interlock	BG / RS	100%	1/13/25	1/26/25
Hardware Implementation - Layout and Generate PCB(s)				
CAN Bus Interface	BG	100%	1/20/25	2/2/25
Power Supply Regulator	BG	100%	1/20/25	2/2/25
Sensor Interface	BG	100%	1/20/25	2/2/25
Cooling System Control	BG	100%	1/20/25	2/2/25
Brake Light Control	BG	100%	1/20/25	2/2/25
Ready-to-Drive Buzzer	BG	100%	1/20/25	2/2/25
Shutdown Loop Interlock	BG	100%	1/20/25	2/2/25
Hardware Implementation - Assemble Hardware				
CAN Bus Interface	TE	100%	2/18/25	2/25/25
Power Supply Regulator	RS	100%	2/18/25	2/25/25
Sensor Interface	BG	100%	2/18/25	2/25/25
Cooling System Control	BG / RS	100%	2/18/25	2/25/25
Brake Light Control	BG / RS	100%	2/18/25	2/25/25
Ready-to-Drive Buzzer	BG / RS	100%	2/18/25	2/25/25
Shutdown Loop Interlock	BG / RS	100%	2/18/25	2/25/25
Hardware Implementation - Test Hardware				
CAN Bus Interface	TE	100%	2/10/25	2/25/25
Power Supply Regulator	RS	100%	2/10/25	2/25/25
Sensor Interface	BG	100%	2/10/25	2/25/25
Cooling System Control	BG / RS	100%	2/10/25	2/25/25
Brake Light Control	BG / RS	100%	2/10/25	2/25/25
Ready-to-Drive Buzzer	BG / RS	100%	2/10/25	2/25/25
Shutdown Loop Interlock	BG / RS	100%	2/10/25	2/25/25
Hardware Implementation - Revise Hardware				
Finished Schematic	BG / RS	100%	2/17/25	2/23/25
CAN Bus Interface	TE	100%	2/17/25	2/23/25
Power Supply Regulator	RS	100%	2/17/25	2/23/25
Sensor Interface	BG	100%	2/17/25	2/23/25
Cooling System Control	BG / RS	100%	2/17/25	2/23/25
Brake Light Control	BG / RS	100%	2/17/25	2/23/25
Ready-to-Drive Buzzer	BG / RS	100%	2/17/25	2/23/25
Shutdown Loop Interlock	BG / RS	100%	2/17/25	2/23/25
Demonstrate Hardware Subsystems	BG / RS / TE	100%	2/25/25	2/25/25

Figure 53: Finalized Gantt Chart Part 1

Software Implementation - Develop Software				
CAN Bus Interface	TE	100%	1/13/25	2/16/25
Power Supply Regulator	JW	100%	1/13/25	2/16/25
Sensor Interface	BG	100%	1/13/25	2/16/25
Torque Vectoring Algorithm	BG	100%	1/13/25	2/16/25
Yaw Rate Controller	BG / RS	100%	1/13/25	2/16/25
VCU State Machine	TE / JW	100%	1/13/25	2/16/25
Dashboard Interface	TE / JW	100%	1/13/25	2/16/25
Cooling System	TE / JW	100%	1/13/25	2/16/25
Software Implementation - Test Software				
CAN Bus Interface	TE	100%	1/20/25	2/17/25
Power Supply Regulator	JW	100%	1/20/25	2/17/25
Sensor Interface	BG	100%	1/20/25	2/17/25
Torque Vectoring Algorithm	BG	100%	1/20/25	2/17/25
Yaw Rate Controller	BG / RS	100%	1/20/25	2/17/25
VCU State Machine	TE / JW	100%	1/20/25	2/17/25
Dashboard Interface	TE / JW	100%	1/20/25	2/17/25
Cooling System	TE / JW	100%	1/20/25	2/17/25
Software Implementation - Revise Software				
CAN Bus Interface	TE	100%	2/10/25	2/24/25
Power Supply Regulator	JW	100%	2/10/25	2/24/25
Sensor Interface	BG	100%	2/10/25	2/24/25
Torque Vectoring Algorithm	BG	100%	2/10/25	2/24/25
Yaw Rate Controller	BG / RS	100%	2/10/25	2/24/25
VCU State Machine	TE / JW	100%	2/10/25	2/24/25
Dashboard Interface	TE / JW	100%	2/10/25	2/24/25
Cooling System	TE / JW	100%	2/10/25	2/24/25
Demonstrate Software Subsystems	BG / JW / RS / TE	100%	2/25/25	2/25/25
System Integration - Assemble Complete System Integration				
Interface with Zips Racing Hardware	BG / JW / RS / TE	100%	2/26/25	3/11/25
Determine Testing Demonstration	BG / JW / RS / TE	100%	2/26/25	3/11/25
CAN Bus Interface	TE	100%	2/26/25	3/11/25
Power Supply Regulator	JW / RS	100%	2/26/25	3/11/25
Sensor Interface	BG	100%	2/26/25	3/11/25
Cooling System Control	BG / RS	100%	2/26/25	3/11/25
Brake Light Control	BG / RS	100%	2/26/25	3/11/25
Ready-to-Drive Buzzer	BG / RS	100%	2/26/25	3/11/25
Shutdown Loop Interlock	BG / RS	100%	2/26/25	3/11/25
Torque Vectoring Algorithm	BG / RS	100%	2/26/25	3/11/25
Yaw Rate Controller	BG	100%	2/26/25	3/11/25
VCU State Machine	TE / JW	100%	2/26/25	3/11/25
Dashboard Interface	TE / JW	100%	2/26/25	3/11/25
Cooling System	TE / JW	100%	2/26/25	3/11/25

Figure 54: Finalized Gantt Chart Part 2

System Integration - Test Complete System Integration				
CAN Bus Interface	TE	100%	3/12/25	3/18/25
Power Supply Regulator	JW / RS	100%	3/12/25	3/18/25
Sensor Interface	BG	100%	3/12/25	3/18/25
Cooling System Control	BG / RS	100%	3/12/25	3/18/25
Brake Light Control	BG / RS	100%	3/12/25	3/18/25
Ready-to-Drive Buzzer	BG / RS	100%	3/12/25	3/18/25
Shutdown Loop Interlock	BG / RS	100%	3/12/25	3/18/25
Torque Vectoring Algorithm	BG / RS	100%	3/12/25	3/18/25
Yaw Rate Controller	BG	100%	3/12/25	3/18/25
VCU State Machine	TE / JW	100%	3/12/25	3/18/25
Dashboard Interface	TE / JW	100%	3/12/25	3/18/25
Cooling System	TE / JW	100%	3/12/25	3/18/25
Preliminary Demonstration of Complete System (Hardware-in-the-loop done t	BG / JW / RS / TE	100%	4/1/25	4/1/25
System Integration - Revise Complete System Integration				
CAN Bus Interface	TE	100%	3/19/25	4/1/25
Power Supply Regulator	RS / JW	100%	3/19/25	4/1/25
Sensor Interface	BG	100%	3/19/25	4/1/25
Cooling System Control	BG / RS	100%	3/19/25	4/1/25
Brake Light Control	BG / RS	100%	3/19/25	4/1/25
Ready-to-Drive Buzzer	BG / RS	100%	3/19/25	4/1/25
Shutdown Loop Interlock	BG / RS	100%	3/19/25	4/1/25
Torque Vectoring Algorithm	BG / RS	100%	3/19/25	4/1/25
Yaw Rate Controller	BG	100%	3/19/25	4/1/25
VCU State Machine	TE / JW	100%	3/19/25	4/1/25
Dashboard Interface	TE / JW	100%	3/19/25	4/1/25
Cooling System	TE / JW	100%	3/19/25	4/1/25
Determine Demo-Day Demonstration	BG / JW / RS / TE	100%	3/19/25	4/10/25
Drop-Off Project	BG / JW / RS / TE	100%	4/10/25	4/10/25
Demo Day	BG / JW / RS / TE	100%	4/11/25	4/11/25
Develop Final Report				
Write Final Report	BG / JW / RS / TE	100%	1/13/25	4/27/25
Car Readiness Evaluation	BG / JW / RS / TE	100%	3/18/25	3/25/25
Submit Final Report	BG / JW / RS / TE	0%	4/11/25	4/27/25
Project Demonstration	BG / JW / RS / TE	100%	4/11/25	4/11/25
Project Presentation	BG / JW / RS / TE	100%	4/11/25	4/11/25

Figure 55: Finalized Gantt Chart Part 3

The overall flow of the Gantt Chart plan was carried out successfully. Most major deadlines were completed on time, with some subsections being delayed due to physical testing restrictions with

an incomplete vehicle build. There was also some delay in software functionality as testing was limited before the PCB was fabricated. This was due to the DEV boards used in the previous semester having a different MCU than what would be used in the PCB, which made much of the software design incompatible with the DEV boards for testing. After the PCB was created, all Gantt chart deliverables were met as close as possible to the expected deadlines. Overall system integration was not truly possible, as we could not integrate the VCU since the vehicle itself was incomplete. However, all systems that our design team was responsible for building were successfully integrated together.

Conclusions and Recommendations (JW)

Overall, this project aimed to create a VCU for Zips Racing Electric's 2025 racecar. Key functionalities included torque and speed control utilizing a four-wheel-drive system, which allows for enhanced traction and efficiency. Safety management was implemented through defined vehicle states, ensuring that the powertrain operates accordingly to unsafe conditions, with safety protocols in both hardware and software that include a vehicle state machine, safety interlock loop, and fault tables to handle any potential issues, while also voltage/current protecting inputs and outputs from the GLV system and other components. At the same time, there is large focus on interfacing with the current vehicle systems, such as the cooling system, battery management system, dashboard interface, etc. Over the last two semesters, we have translated these plans into a working vehicle control system that meets all the requirements of the Formula SAE competition and the Zips Racing Electric team. With the team finalizing the physical vehicle, we will be able to thoroughly test and verify the VCU functionality to prepare the team for the upcoming races this summer.

References

1.2. https://etd.ohiolink.edu/acprod/odb_etd/ws/send_file/send?accession=akron1689104916891684&disposition=inline

1.3. https://ideaexchange.uakron.edu/cgi/viewcontent.cgi?article=2386&context=honors_research_projects

1.4. https://www.youtube.com/watch?v=yS3w2IjkzxU&ab_channel=TheEngineersPost

1.5. https://www.sae.org/attend/student-events/about-formula

1.6. https://www.fsaeonline.com/cdsweb/gen/DownloadDocument.aspx?DocumentID=369d01c0-589d-4ebe-b8d4-b07544f4a52b

1.7. https://www.formulastudent.de/fileadmin/user_upload/all/2024/rules/FS-Rules_2024_v1.0.pdf

1.8. https://www.zipsracingelectric.org/

1.9. https://zipsracing.org/

1.10. Drexler LSD Assembly Instructions - https://autotech.com/pdf/183884

1.11. https://www.amk-motion.com/am21k-dokucd/dokucd/en/content/resources/pdf-dateien/fse/r25/pdk_205481_kw26-s5-fse-4q_en_2020-44_.pdf

1.12. https://patents.google.com/patent/US11477083B2/en?q=(embedded+control+systems)&oq=embedded+control+systems

1.13. https://ebooks-ohiolink-edu.ezproxy.uakron.edu:2443/viewer/57dac238-bd40-11ea-8c73-0a28bb48d135/1

1.14. https://upcommons.upc.edu/bitstream/handle/2117/334223/eduardmoreratorres-treballfidegrau.pdf?sequence=1

1.15. http://www.dem.ist.utl.pt/poliveira/Ensino/paper_2017c.pdf

1.16. https://www.researchgate.net/publication/348399436_Design_of_Integrated_Autonomous_Driving_Control_System_That_Incorporates_Chassis_Controllers_for_Improving_Path_Tracking_Performance_and_Vehicle_Stability

1.17. https://www.millikenresearch.com/fsaettc.html

1.18. https://patents.google.com/patent/US8360533B2/en

1.19. https://fordauthority.com/2023/02/ford-patent-filed-for-manual-torque-vectoring-system/#:~:text=Ford%20Motor%20Company%20has%20filed,and%20assigned%20serial%20number%2011584224.br

1.20. https://www.iso.org/standard/74528.html

1.21. https://www.iso.org/standard/86384.html

1.22. https://www.iso.org/standard/51180.html

Appendix

Extended Engineering Requirements Information (BG/JW/RS)

Figure 56: List of marketing requirements for the VCU

Marketing Requirements
1. The control system must be able to monitor the real-time state of the vehicle. 2. The control system design must maximize vehicle performance. 3. The control system must be able to control a four-wheel drive vehicle. 4. The hardware and software must meet competition rules. 5. The vehicle must be able to operate safely. 6. The control system shall interface with all existing vehicle systems and be adequately documented for easy implementation by Zips Racing team members.

Table 32: Marketing requirements, requirement number, and explanation for each engineering requirement

Marketing Requirements	**Engineering Requirements**	**Justification**
1, 2, 3	The system shall provide a method of controlling the torque and speed inputs to the powertrain motor controller(s) using driver inputs and a control system.	The car must react accordingly to driver inputs to operate the vehicle.
1, 4, 5	The system shall control a set of "vehicle states" (See Table 33).	Ensures the safe operation of the vehicle and prevents powertrain operation unless in the desired state, as per FSAE 2025 competition rule EV.9.
2, 3, 5	The system shall provide a method of switching between different "control modes" for the powertrain, which can be selected by the driver (See Table 34).	Gives greater control to the driver to enhance operating ability; allows iterative control algorithm development; allows a safety mode.
1, 2, 3, 6	The system shall include the required interfacing capability for sensors necessary for powertrain control algorithms (see Table 36)	The control system must have the necessary hardware interfaces to allow a variety of control algorithms to be used.
4, 5, 6	The system shall activate a brake light on the rear of the vehicle based on the following conditions:	Competition rules requirement: indicates to all other drivers that the car is braking, allowing safe

	1. The driver presses the brake pedal and exceeds a setpoint of 5 kPa in the brake system. 2. The vehicle is engaging in regenerative braking.	response to the vehicle's decrease in speed.
1, 2, 4, 5	The system shall control water pump(s) and radiator fan(s) associated with the vehicle's cooling system.	Powertrain motors, motor inverters, and battery cells are liquid cooled. Inadequate cooling could result in damage to the vehicle and unsafe conditions for the driver. Battery power must also be conserved so the vehicle can operate for the duration of the competition.
1, 4, 5	The system shall not command the powertrain to draw more than 80 kW of power from the high voltage battery. The BMS systems shall provide power and state-of-charge updates to the system to ensure this limit.	Required by FSAE 2025 rule EV.3.3.1 & EV3.4.1.
1, 5	The system shall monitor the status of the interlock switches in the vehicle shutdown loop. The system will broadcast the status of these interlock switches on the CAN bus.	These assist debugging of the vehicle systems and are used to notify the driver of a safety issue.
2, 5	The system shall be powered from the existing GLV system.	We must interface with various systems/sensors of the vehicle.
1, 2, 6	The system shall operate with a nominal input voltage of 12 V DC and a range of 4.5 to 17 V DC.	The system must operate from an unregulated low-voltage GLV referenced battery.
1, 2, 5	The maximum current draw of the system shall be 3 A from the GLV system.	GLV battery capacity and current capability is limited, and other systems must operate on the same voltage bus.
2, 5	The power supply input shall implement over-voltage protection, reverse-polarity protection, and limit the 5 V rail to 2 A and the 3.3 V rail to 1 A.	Prevents failure if the wrong voltage level is applied during system testing, or the battery is overcharged. Vehicle performance is compromised if the system fails.
2, 5	Power outputs to external devices (pumps, fans, sensors, etc.) shall be current limited to their respective maximum operating currents.	Prevents external loads from drawing levels of current that could damage the system.
2, 5	The device, when fully assembled and ready for installation into the vehicle,	Protects the electronics from electrostatic discharges during testing, installation, and

	shall have adequate ESD protection for human handling.	maintenance. Vehicle performance is compromised if the system fails.
1, 2	The system shall record and store its settings (tuning parameters, etc.) across power cycles.	This will allow the vehicle to be driven the same every time it starts. Often there is limited time to set up the vehicle before driving.
5	The system shall include watchdog functionality to detect software faults resulting in unresponsive behavior.	The system must put the vehicle into a safe state during software faults, such as infinite loops, hangs, incomplete ISRs, etc.
1, 2	The control system shall update the powertrain at a minimum rate of 25 Hz.	The control system needs to update fast enough to achieve the yaw rate bandwidth required by high vehicle performance. A fast update rate is also needed for the driver to operate the vehicle smoothly and safely.
1, 2, 6	For every CAN Bus message relevant to the system, the system shall process (update its internal state) and/or respond within a specified process time.	Fast communication is needed between the system and any other devices connected to it to operate effectively. Responding to all messages helps ensure predictable operation.
1, 2, 3, 4, 5, 6	The system shall be able to interface with the existing low voltage system, CAN bus communication, motor inverters, battery management systems (BMS), cooling systems, and shutdown loop.	The new VCU must be able to interface with the existing systems in order to control the vehicle.
2, 5	The system shall be waterproof.	Prevents water damage and fire risks. The vehicle must operate when raining and pass a “rain test” during competition.
1, 3, 4, 5, 6	The system shall detect system faults and operate in a predictable way to protect the driver and the vehicle. This includes performing fault analysis and error checking (see Table 35).	Ensures the safety of the driver / vehicle and predictable behavior of the vehicle during faults. Also ensures compliance with existing rules set by the racing organizations.

Table 33: Vehicle state diagram state definition and explanation

Vehicle State	**Explanation**
Low Voltage	The system is in this state when GLV power is initially turned on.

Tractive System State	The system is in this state when GLV power is turned on and the shutdown circuit is closed (high voltage from the battery is enabled). This state shall not enable the powertrain's motor controller or allow motor control.
Ready to drive	1. The system switches to this state when the following sequence occurs: the vehicle in the Tractive System Active state; the driver presses the brake pedal; the driver presses a start button. 2. If the system detects a safety-related fault, the interlock in the shutdown loop shall open, and the system shall enter the Lockout state. 3. The VCU shall enter the Low Voltage state if a driver presses the dashboard E-stop. 4. The system shall include a "start" button, used to enter the Ready to Drive state.
Lockout	The lockout state requires an external reset signal to be cycled before the vehicle can enter the Tractive System Active state. The lockout functionality must be implemented by the system using an interlock in series with the shutdown loop.

Table 34: Vehicle control modes definition and explanation

Control Mode	**Explanation**
Limp Mode	The vehicle drives using only the rear electric motors, using only the accelerator pedal position sensors and brake pressure sensors as control inputs. The vehicle shall not use regenerative braking while in limp mode.

Table 35: Error checking signal definitions and explanations

Error Checking Signal	**Explanation**
Analog Control Signals	This error checking shall detect open circuit, short to ground, short to sensor power, and short

	between signal lines of redundant sensors per FS rule T 11.8.6.
Digital Control Signals	This error checking shall detect failures from a loss of communication, and implausibility from out-of-range signals.
Digital Control Signals (via CAN bus)	This error checking shall detect failures from a loss of communication, data corruption, and message delay.
Acceleration Control Signal	This error checking shall command the powertrain to shut down power to the electric motors if an error in the acceleration control signal is detected for more than 100 ms, per FSAE rule T.4.2.5

Table 36: Torque Vectoring Input Signal definitions and explanations

Torque Vectoring Input Signals	**Explanation**
Steering Wheel Angle	Used to determine the yaw rate reference that the control system targets. Captured through a magnetic encoder mounted on the steering rack which communicates through SPI.
Accelerator Pedal Percentage	Used to determine the yaw rate reference that the control system targets. Redundant accelerator pedal position sensors (APPS) send analog signals to the system.
Brake Pressure	Used in the torque distribution portion of the control system to determine regenerative braking torques and avoid commanding driving torques that act against the mechanical brakes. Brake pressure is captured using two analog hydraulic pressure sensors on both the front and rear brake circuits.
Reference Speed	Used to determine the yaw rate reference that the control system targets. On a 4-wheel-drive vehicle, the reference speed is captured using a GNSS / IMU sensor communicating over CAN bus.
Wheel Speeds	Compared against the reference speed to determine the relative slip of the tire against the ground, for use in the torque distribution portion of the control system. Wheel speeds are measured with encoders on each electric motor. The inverters communicate this data over CAN bus to the system.

Vehicle Side Slip Angle	Used to determine the individual slip angles on each tire to calculate the tire forces. Side slip angle is captured by measuring the reference speed heading angle relative to the orientation of the vehicle, using the GNSS / IMU sensor, communicating over CAN bus.
Vehicle Yaw Rate	Used as feedback for the yaw-rate controller. Yaw-rate is calculated from the gyroscope data captured by the GNSS / IMU, communicated over CAN bus.
Tire Normal Forces	Used to determine the available grip on each tire in the torque distribution portion of the control system. Linear accelerations captured by GNSS / IMU sensor communicated over CAN bus, are fed into a load transfer model of the vehicle to determine normal forces.
Battery Instantaneous Power	Used by the torque distribution portion of the control system to limit total power draw to 80 kW. Voltage and current data are communicated over the CAN bus from the battery management system on the vehicle.
Battery State of Charge (SoC)	Used by the torque distribution portion of the control system to limit total power and regenerative braking to safe values for the battery design. Voltage and current data are communicated over the CAN bus from the battery management system on the vehicle.