Formula SAE Data Acquisition System

Final Report

Prepared By:

Camille Gowda
Kyle Rosenberg
Jillian Hornby
Javier Olmos
Diego Oliver
Elena Kim

Presented in Partial Fulfillment of the Requirements for

ECE 4951 Spring 2026

17 April 2026

Executive Summary

The Formula SAE Data Acquisition System project focused on designing a centralized and scalable data collection platform to enable the Vanderbilt Motorsports team to improve performance. The goal was to integrate data from multiple sources and protocols into one synchronized logging and visualization pipeline to enable the team to make data-driven decisions for vehicle performance.

Our team designed a “nervous system” architecture using Arduino-based gateway nodes, a CAN bus, and a Raspberry Pi server. The Raspberry Pi acts as the central logger, CAN receiver, synchronization controller, and file storage device. We validated communication between the Pi and Arduino nodes, implemented synchronization timing windows to timestamp data, and created a Python logger that decodes gateway data into visualization compatible CSV files. In parallel, the team developed wiring diagrams, PCB gateway designs, a battery distribution system, sensor documentation, and tools to support future installation and expansion.

We also developed a scalable Python-based visualization system that enabled real-time and post-run analysis of telemetry data. The system allowed users to dynamically select and filter sensor channels, organize plots across multiple pages and partitions, and view signals with independent scaling for clearer interpretation. It was designed to handle large datasets efficiently while maintaining responsiveness and included functionality to export processed data into structured formats for further analysis. This tool provided the Vanderbilt Motorsports team with an intuitive interface to quickly interpret vehicle performance data and identify areas for optimization.

Final vehicle-ready assembly was limited by the late availability of sponsor-procured electrical components ordered for the PCB and battery system. The team instead delivered a validated prototype with the full sensor to visualization system pipeline demonstrated, as well as implementation documentation for Motorsports to complete deployment.

Table of Contents

PAGE

Part 1. Project Execution

A. Background

The Formula SAE Data Acquisition System project is sponsored by the Vanderbilt Motorsports team. Each year, the team designs and builds a Formula SAE racecar that competes internationally against more than 120 university teams. In 2024, Vanderbilt placed 3rd in the Efficiency event and 27th overall, showing significant strength on the national and international stage. The Motorsports team members want to improve their understanding of how the car behaves and how critical subsystems, such as the engine, cooling system, and suspension perform in real time. The team has decided that data acquisition is a key area where improvements could enhance vehicle performance.

At this current time, the team lacks a reliable system to gather real-time data from the car's various subsystems. The need for accurate information has only grown as the vehicle's complexity has increased, particularly in areas like cooling and vehicle dynamics. The team wants to make data-informed decisions and avoid relying solely on driver feedback or manual inspection. Without measurements of engine behavior, temperatures, pressures, GPS data, and dynamic motion, it becomes difficult for team members to diagnose issues, validate new designs, or optimize the car's performance from year to year. This gap in data highlights the broader problem our project aims to address.

To support the team's engineering needs, we were tasked with developing an integrated, real-time data acquisition (DAQ) system capable of combining data streams from multiple sources. All data must be centrally logged on a Raspberry Pi running Python, enabling synchronized timestamps and access to vehicle information. The lack of such a system has limited the team's ability to analyze the car holistically, making this design project essential for improving their development.

In addition to supporting the Motorsports team, our project plays a role in assisting the Mechanical Engineering Cooling System Senior Design Team. Their work relies heavily on accurate radiator inlet and outlet temperatures, airflow, and pressure data that the current vehicle does not reliably collect. The DAQ system allows our team and the mechanical engineering team to collaborate more effectively by sharing a dataset that captures how cooling performance interacts with engine load and driving conditions.

The need for this design project comes from the team's multi-year effort to build a fully integrated data acquisition system. We aimed to improve how real-time vehicle data is collected, synchronized, and analyzed, letting the team make informed design decisions and validate changes more effectively. This project moves the team closer to a complete, long-term DAQ solution that will support future vehicle development and performance.

B. Scope of Work

The project focused on the design and implementation of a data acquisition system that can coordinate information from multiple sources while driving the Formula SAE car. This involved collecting, timestamping, and recording data from sensors mounted on the car. An important part of the work involved working closely with the Mechanical Engineering Senior Design team to capture the radiator inlet and outlet temperatures, airflow data, and pressure measurements needed to verify their thermal models. In addition to data acquisition, the project involved creating Python-based logging, processing, and visualization frameworks that will allow the team to perform post-run analysis, identify trends, and quickly resolve issues. Finally, the scope covered the creation of a secure and reliable storage format that supports long-term data acquisition and scalable analysis for FSAE testing and development.

The scope of the project does not include any mechanical fabrication or modification of vehicle components beyond what is necessary for sensor mounting, as those efforts are handled by the Mechanical Engineering team and the Vanderbilt Motorsports team. It also does not involve engine tuning, calibration changes, or any form of performance optimization related to the PE3 ECU or the RaceBox hardware. Redesigning or altering the embedded hardware, control systems, or commercial devices used for data collection is also excluded. Additionally, other activities such as full-scale aerodynamic modeling, chassis redesign, or structural analysis fall outside the boundaries of this project. The work is also not aimed at commercialization or external deployment beyond the internal needs of the Vanderbilt FSAE program. Instead, the focus remains on delivering a functional and well-integrated data acquisition pipeline that supports testing, engineering decision-making, and continued development within the team.

C. User & System Requirements, and System Specifications

The Vanderbilt Motorsports team wants to gather and visualize performance data to assist with developing their racecar for the 2026 FSAE Competition. Table 1 outlines their specific user requirements, along with the corresponding system requirements and system specifications. These user requirements ensure that the development of the DAQ system maps directly back to the project goals throughout the entire process.

Table 1: User and System Requirements Matrix

ID	User Requirement	System Requirement	System Specification
1.0	**Data Integration and Logging**		
1.1	The team needs to log data from the RaceBox Mini S.	The DAQ system must interface with the RaceBox Mini S and parse its data stream.	Data: GPS, acceleration, lap timing, IMU Protocol: USB/Serial
1.2	The team needs to log engine data from the Performance Electronics PE3 ECU.	The DAQ system must read and parse CAN bus messages broadcast by the PE3 ECU.	Data: Engine RPM, Throttle Position, Manifold Pressure, Engine Coolant

			Temperature, Inlet Air Temperature Protocol: CAN bus
1.3	The team needs synchronized data from all sources for analysis.	The central processor (Raspberry Pi) must timestamp all incoming data packets relative to a single system clock.	Format: Relative Time (sec)
1.4	The team needs to store data for post-run analysis.	The system must write data to storage in a unified format readable by standard tools.	Storage Media: MicroSD Card File Format: .csv
2.0	**Cooling System Integration**		
2.1	The team needs to measure coolant temperature drop across the radiator.	The system must measure coolant temperature at both the radiator inlet and outlet.	Input: 2x Analog Temp Range: 150 – 230 °F (while running)
2.2	The team needs to measure airflow pressure drop across the radiator.	The system must measure air pressure at the radiator inlet and outlet.	Input: Digital via SPI Airflow Range: 20 – 40 mph (75 mph max)
2.3	The team needs to integrate the cooling system sensors with the Mechanical Engineering Cooling System Senior Design Team.	The system must follow mechanical design constraints.	
3.0	**System Architecture and Hardware**		
3.1	The DAQ system needs to be integrated and run on a single-board computer.	The central processing DAQ hub will be a Raspberry Pi.	Hardware: Raspberry Pi 4 OS: Raspberry Pi OS Software: Python
3.2	The DAQ system must be robust enough for vehicle mounting.	The hardware must be housed properly and utilize secure connectors.	
3.3	The DAQ system must be extensible for future sensors.	The software and hardware architecture must allow for adding new CAN or analog channels without major redesign.	
4.0	**Data Visualization and Software**		

4.1	The team needs to visualize engine, vehicle, and cooling data.	The system must provide software tools to generate graphs and dashboards from log files.	Output: Python scripts, Matplotlib plots, dashboards
5.0	**Battery System Design**		
5.1	The battery must safely provide power to the DAQ and be swapped and recharged easily.	The system shall provide a stable 5 V DC output to the gateways and over-current protection.	Battery: DeWalt 20 V Output voltage: 5 V Fuse Rating: 5 A maximum

D. Constraints

ABET says that “Engineering design is a process of devising a system, component, or process to meet desired needs and specifications within constraints.” The most important constraint is the functional integration of multiple independent sources of data in real time. Each data stream operates on different communication protocols, sampling rates, formats, and timing behaviors, which forces the design to handle synchronization. Because all data must be recorded centrally on a Raspberry Pi, the entire system must be able to support multi-source capture.

Another key constraint is hardware and electrical feasibility. The wiring harnesses connecting the sensors, Pi, CAN interface, RaceBox, and power system must be designed to withstand vibration, heat, and the tight packaging of a FSAE race car. The routing of wires, connectors, and sensor mounting must work within the chassis while avoiding interference with steering, braking, and suspension components. Additionally, critical components near the engine and exhaust must tolerate extreme heat, with temperatures reaching up to 1200°F around the exhaust headers. This limits what kinds of sensors, connectors, and mounting places can be chosen, particularly in areas with limited space or high temperature.

The project is also constrained by compatibility. The system must be compatible with existing FSAE electronics and all custom sensors. At the same time, the final product must be usable by the Vanderbilt Motorsports team, meaning file formats, Python scripts, and logging must integrate well with the tools the team already uses for analysis and their competitions.

Sustainability and maintainability are also important aspects of the design. The system must log data in structured formats that future FSAE members can use and extend. Because new sensors and subsystems will be added in later years, the architecture needs to remain modular and well documented so that future teams can troubleshoot or change components without having to rebuild the entire system. Long term usability is almost as important as initial performance since the project, if done right, will serve multiple seasons

of cars and many hours of testing.

Usability constraints shape how Motorsports team members will interact with the finished system. Data must be easy to access, easy to visualize, and easy to interpret by students with varying levels of experience in Python, CAN protocols, and instrumentation. User friendly design also matters for the Mechanical Engineering cooling subsystem team, who rely on the system to evaluate radiator temperatures, airflow pressures, and cooling efficiency without needing to understand every technical detail of the DAQ system.

One important ABET outcome for electrical and computer engineering students is that they demonstrate “an ability to apply engineering design to produce solutions that meet specified need with consideration of public health, safety, and welfare, and global, cultural, social, environment, and economic factors.” This outcome is directly relevant to the Formula SAE data acquisition project, where we considered these responsibilities with each design decision.

Public health, safety, and welfare are critical responsibilities when working on any engineering project and have been weighed greatly when making design decisions for the DAQ system. Although the project is created within an academic organization, the DAQ system directly influences decisions that affect driver safety. Electrical systems also have innate hazards such as overheating or electric shock. Our senior design team utilized safe manufacturing and testing methods following our established safety plan and the rules of the Vanderbilt Motorsports Lab. Using the proper personal protective equipment and following safety guidelines was essential for the safety and welfare of our team members.

Formula SAE has a significant global, cultural, and social impact as part of the broader motorsports industry. Motorsports holds major cultural significance internationally, with professional leagues such as Formula 1 attracting millions of viewers and shaping engineering and performance standards across the automotive sector. While FSAE obviously operates on a much smaller scale, it participates in the same engineering culture of competition and innovation. Additionally, our project fosters social development by promoting teamwork, leadership, communication, and interdisciplinary collaboration among students who will eventually enter the engineering workforce. This project therefore serves both as a technical design exercise and as a professional training environment. By creating a system that is well-documented and accessible, we can ensure that future Vanderbilt team members who come from a variety of academic backgrounds can understand and expand the system.

Environmental and economic sustainability are considered throughout the design process. The team minimized waste and unnecessary spending by reusing existing components when possible. To reduce the project’s long-term environmental footprint, we documented all designs and procedures to enable future reuse of both hardware and software. Additionally, the design emphasizes modularity and scalability to allow components to be repurposed or upgraded instead of discarded, reducing material consumption across future design cycles. During the procurement process, the team conducted cost-benefit analyses of different models for the same parts to ensure that each component provided a clear performance value relative to its cost. Our design

choices prioritized functionality and durability while remaining within budget constraints.

E. Ethical and Professional Responsibilities

According to ABET, engineering graduates must demonstrate an “ability to recognize ethical and professional responsibilities in engineering situations and make informed judgements, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts.” While the FSAE DAQ System is developed within an academic organization, there are still ethical and professional responsibilities that apply to the project.

As engineers who were designing a vehicle system, our most important ethical responsibility was the safety of the driver and the people near the vehicle. No matter what, a failure in the DAQ system must not result in serious hazards such as loss of vehicle control or an electrical fire. Further, the design and fabrication should not pose any safety threats to team members. To address safety concerns, all design and fabrication decisions were made in consultation with our safety plan and the Vanderbilt Motorsports lab safety guide. Fire hazards were mitigated by ensuring wire harnesses are made with proper wire gauges and safe connections to prevent overheating or short circuits. The entire DAQ system is designed such that a failure will not interfere with the operation of the vehicle and its other components.

Any system that collects data is subject to ethical considerations, specifically data integrity and accessibility in this case. The processed data visualizations and dashboards must not hide issues in the data or exaggerate performance of the vehicle. Not only is this unethical, but it is also dangerous if the Vanderbilt Motorsports team tunes their vehicle components based on false data readings. To commit to truthful data, we stored raw data alongside processed data to be accessed by any team member. We will flag data quality issues in the log files so that the team knows when the data is unreliable. In a similar vein, engineering solutions should contribute to advancements and be accessible to others. To ensure future teams can benefit from the collected data, a unified storage format was created with documentation so that non-electrical members can easily access and interpret the data.

Economic decisions were made with the understanding that the Motorsports team operates on limited funds. It is our professional responsibility to respect their budget constraints and thoroughly research components before buying them. Before buying new parts, we took inventory of the lab to determine what electrical components we could use to save money. For missing parts, we performed cost-benefit analysis in collaboration with our advisors to select high-value components.

Engineers also have the responsibility to consider the lifecycle and environmental impact of their designs. One of the most important considerations when designing a system is what happens to it at the end of its lifecycle. The goal of our design was to minimize waste by developing an extensible system that will allow future teams to add new sensors without having to discard it and build a new one from scratch. This resulted in our “nervous system” design that is flexible for future expansions which will be explained further in Part 2.

Formula SAE is a global competition involving 120 teams from around the world. Our engineering solutions for the DAQ system directly affect the performance of the Vanderbilt Motorsports team on a global stage. By focusing on data-driven development, the vehicle's performance can be measured against international competitors and the global FSAE community standards. Our solution not only benefits the current team as they prepare the vehicle to be competition ready but will also assist future teams for years to come with a scalable system and stored data that can be used as a benchmark for future innovation. Because of the long-term impact of this project, following ethical and professional responsibilities is critical for a successful system.

F. Standards

In terms of engineering standards that were adhered to in our project, it is important to note that our DAQ system is only for observing data and does not commit safety critical commands to the car or influence vehicle behavior. However, as the system will lie physically in the vehicle, we must adhere to engineering standards to ensure safe electrical integration, data handling, compatibility, and reliable operation. This means adhering to proper bus communication protocols, low voltage automative wiring protocols, compatibility, and environmental and durability standards as outlined by ISO and IEEE, as well as also adhering to the rules in the FSAE 2026 Rulebook if the system is to be used in the competition.

Relevant Engineering Standards
ISO 11898-1: Defines CAN bus protocols such as timing, arbitration, error-handling, and message structure. This includes both the data link layer which specifies data transmission and the physical layer which specifies physical connection requirements.

SAE J1128: Defines performance requirements and insulation best practices for low voltage automotive primary cable in vehicle electrical systems.

SAE J1742: Standard for connector and terminal performance such as mechanical strength, corrosion resistance, and signal integrity under vibration and temperature cycles.

ISO 11452: Defines testing procedures for evaluating interference from off-vehicle electromagnetic disturbances to electrical components.

IEEE 829: Defines best practices for test documents, such as test plans, test design specifications, test case specifications, and logs.

FSAE 2026 Rulebook:
T.8 (Fasteners): Defines rules for any component mounted to chassis as well as acceptable fasteners.

T.9 (Electrical Equipment): Defines requirements for all low-voltage wiring components such as insulation, chassis mounting, protection from environment, routing, enclosures,

grounding and overcurrent protection. Also mentions that system must be shut down when master switch is open.

Other: Requires DAQ must not interfere with controls, steering, braking systems or vehicle function. Every electrical component must be represented in team wiring diagrams submitted for technical inspection. This focus on clarity in documentation will assist in expansion in future years as well.

We incorporated these engineering standards directly into both the hardware architecture and software implementation of the DAQ system to ensure safety and long-term usability. For communication standards, the CAN bus link follows ISO 11898-1 by adhering to the specified bit timing, message formatting, and error-handling conventions. This included configuring the CAN interface with validated baud rates, ensuring proper termination, and implementing software-level handlers for bus arbitration and fault states. All data parsing scripts was built around these standardized message structures so that logs remain consistent with Motorsports norms and compatible with future FSAE hardware. Similarly, wiring for the entire system complies with SAE J1128, meaning all low-voltage cables used for sensors and auxiliary electronics met the insulation, temperature, and vibration durability standards required for automotive environments.

Mechanical and electrical connections adhere to SAE J1742, which guide connector strength, corrosion resistance, and signal reliability. This requires using sealed automotive-grade connectors, proper crimping techniques, and strain-relieved harnesses routed according to best practices. These choices align with our safety plan, which emphasized trained use of crimping and soldering tools, proper PPE, grounding practices, and safe battery disconnection during installs. Because the system operates inside a competition vehicle, all wiring followed T.9 of the FSAE Rulebook, which mandates secure mounting, abrasion protection, accessible routing, and correct grounding. We documented every connector, harness, and sensor in our wiring diagrams for technical inspection to ensure full compliance.

To ensure reliability during testing and competition, the team adhered to ISO 11452 by minimizing electromagnetic interference in both layout and wiring. Shielded cables, careful separation of high-current and signal paths, and secure enclosure of the Raspberry Pi and CAN hardware helped reduce susceptibility to off-vehicle disturbances. On the software side, we followed the principles of IEEE 829 when generating test plans, maintaining logs, and validating multi-stream synchronization. This ensures that each subsystem (RaceBox, CAN, gateways, and custom sensors) is systematically verified and traceable throughout development.

G. Work Breakdown

The execution of this project followed a structured, role-driven workflow that aligns closely with our work breakdown structure and the technical demands of the FSAE environment. Our team began by determining project scope, expected deliverables, and system requirements, ensuring that everyone understood the constraints imposed by sensor capabilities, vehicle packaging, and the integration needs of Vanderbilt Motorsports. From there, work proceeded in parallel across hardware and software tracks, with each team

member assuming responsibilities that match their technical specialization.

Jillian and Camille led the embedded systems side, setting up the Raspberry Pi, configuring CAN and GPIO interfaces, and validating system communication. Their work established the foundation on which all sensor data streams will run. Diego focused on the mechanical and electrical assembly tasks, including cutting and preparing wiring harnesses, designing the gateway PCB, and physically mounting brackets, sensors, and protective sheathing on the chassis. His role ensured that the hardware withstands vibration, heat, and the tight tolerances of the FSAE car. Javier handled instrumentation-level tasks such as selecting sensors, calibrating them, verifying their outputs, and coordinating installation on the vehicle in collaboration with the mechanical cooling team. His work ensured that the system collects accurate and meaningful data across pressure, temperature, flow, and dynamics measurements. Kyle and Elena took the lead on software, data processing, and troubleshooting. Kyle built the Python-based data acquisition framework, developed logging formats, established data validation methods, and maintained the GitHub repository. Elena managed testing and debugging using tools such as oscilloscopes and multimeters, ensuring that signal integrity remains consistent across sensors.

To coordinate this work, the team met weekly with our advisors to report progress, review test results, and identify potential risks or roadblocks. These meetings followed a standard agenda and were held in-person at the Motorsports Lab. Our team also held many informal working sessions, often multiple times per week, during hardware assembly deadlines or whenever subsystem integration required multiple areas of expertise. All documentation, wiring diagrams, safety plans, and code were stored in the shared SharePoint and GitHub system to maintain record, transparency, and accountability. Regular communication with the FSAE Mechanical and Cooling teams was also built into the workflow to ensure compatibility with their subsystems and to coordinate installation and testing windows. Altogether, this structured execution plan ensured that tasks remain organized, and responsibilities were clearly defined for each team member.

H. Project Schedule

Our Gantt chart shows how our planning and setup phase during the first semester of the project enabled the longer integration phases that followed. The cooling system and sensor work extended across almost the entire design project timeline because each subsystem required hardware installation and software iteration. Performance testing tasks began later in the schedule once enough hardware was in place to support such tests. The visualization and documentation occurred in parallel throughout the design process to allow system visualization to occur throughout the testing process. Our Gantt chart previously included integration within the Motorsports car; however, this step was significantly delayed as described in Section J due to missing parts for physical mounting. Documentation was provided for this final step for the Motorsports team or future senior design teams to integrate in the future.

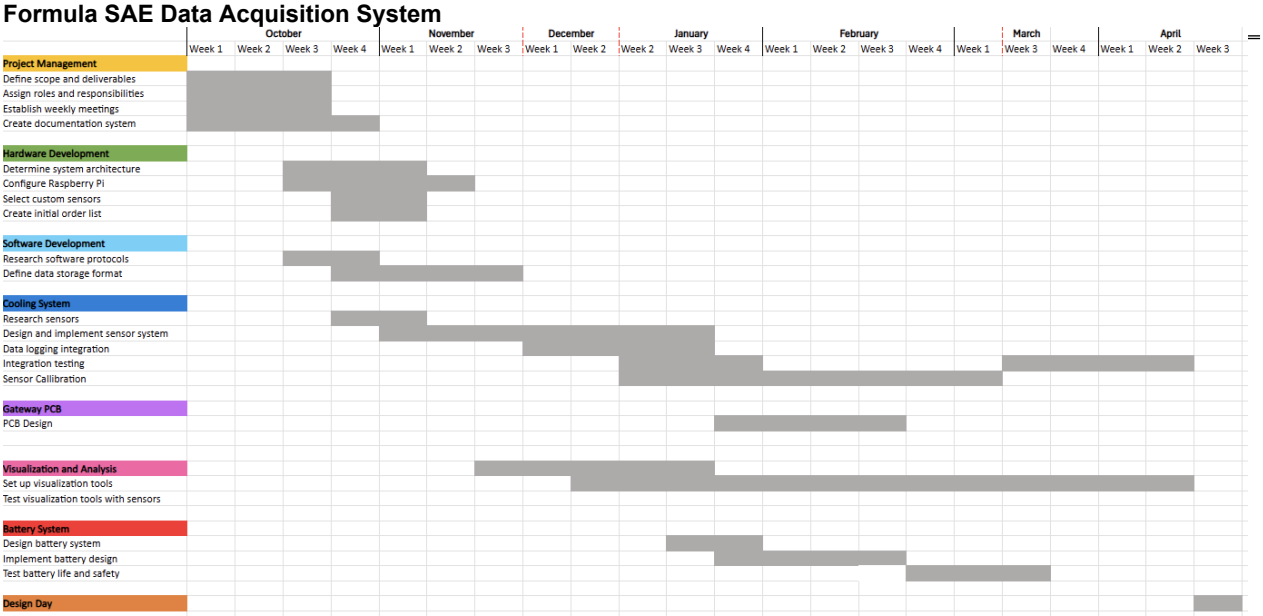


Figure 2. Gantt Chart showing the project's tasks, their durations, and how they overlap.

The biggest changes from our initial Gantt chart were the introduction of the battery system, removing physical integration with the Motorsports team, and starting the data visualization and analysis process earlier. Previous Gantt charts can be seen in Appendix A. We initially planned to connect our system to the car battery, however due to concerns about current draw we opted to create a separate battery system for the data acquisition system. We also decided to start the data visualization software earlier as mentioned above to allow system visualization earlier in the process. The most recent change was the removal of physical integration within the car. Due to logistical challenges as described in Section J, our parts for mounting our system within the car did not arrive until days before Design Day. Further, the Motorsports car currently remains unfinished, and many of our proposed integration steps could not be completed regardless.

I. Detailed Budget

Our project does not have a fixed budget, but instead we got purchases approved on an as-needed basis from Phil, the faculty head of Vanderbilt Motorsports. This is because our expenses are drawn from the Motorsports budget, which encapsulates the 120 students within the organization as well as the sponsorship of our project and the Mechanical Senior Design Team. The budget for Vanderbilt Motorsports General Supplies for 2025 - 2026 is $26,710, but a majority of this budget goes towards raw materials and parts for the car. Keeping this in mind, we wanted our spending to support the team's long-term goals and to minimize any extraneous purchases.

	General Supplies Breakdown		
2750	differential	**TOTAL**	26710
4000	raw materials (steel tubing, sheet metal, delrin, ...)		
4800	tires (~$300 * 4 tires * 4 sets = $4800)		
500	wheels		
200	hardware (bolts, washers, nuts) (probably a low ball estimate?)		
2000	engine rebuild parts (sprocket, extra parts in case something goes wrong)		
2000	senior design funds		
60	impact attenuator		
600	shocks		
2500	fittings (breather tubes, brakes, fuel, paint)		
5000	new engine		
1000	exhaust development		
1000	induction development		
300	shipping costs		

Figure 3. Motorsports General Supplies Breakdown budget.

According to Figure 3, we have $2,000 allocated to Senior Design projects. However, this includes funds for both our project and the Mechanical Senior Design Team, which is building a radiator for the car. The radiator subsystem is expensive and took up a lot of this budget with raw materials. Considering this, we made our best efforts to be as resourceful as possible and to justify any purchases of new components. We did this by reusing as much of the equipment from last year's Senior Design Project as we can, including our central Raspberry Pi, which we were able to wipe and reconfigure for our architecture.

We also leveraged the Wond'ry makerspaces as much as we could, which proved useful with prototyping and small parts. They also maintain an extensive collection of power supplies, oscilloscopes, resistors, wires, breadboards, and other lab tools that we have used for bench testing and prototyping. We found the use of Wond'ry equipment and department stock invaluable for reducing the need to purchase small parts and for testing proof of concepts before ordering dedicated equipment. We also borrowed test equipment, when possible, for tasks where we need a piece of equipment only for setup. For example, in our Raspberry Pi setup, we required a lot of cables and adapters which we were able to borrow from a department professor. Additionally, when we needed thermometers to perform our temperature sensor calibrations, we were able to borrow them from a professor in another department. In terms of parts we need to order, our advisor specified that we should simply send him order lists when we need parts.

A	B	C	D	E	F	G
Item	**Description**	**Supplier**	**Notes**	**Cost**	**Multiplier**	**Total**
Gateway Buck Converter	LM2596 DC to DC Buck Converter 3.0-40V to 1.5-35V Adjustable Voltage Regulator	Amazon	pack of 5	$ 7.99	1	$ 7.99
Server Buck Converter	DROK Buck Converter 12v to 5v, 5A USB Voltage Regulator DC 9V-36V Step Down	Amazon		$ 9.99	1	$ 9.99
Barometric Pressure Sensor	BMP388 Digital Temperature Detector Module Atmospheric Air Pressure Sensor Module 24Bit	Amazon	Pack of 2, SPI/I2C compatible	$ 15.95	1	$ 15.95
Pro Mini Microcontroller	HiLetgo Atmega328P 5V/16M Replace	Amazon	pack of 3	$ 13.99	1	$ 13.99
SPI to CAN converter	MCP2515 CAN Bus Module TJA1050 Receiver SPI Module	Amazon	pack of 3	$ 7.89	1	$ 7.89
ADC	ADS1115 16 Bit 16 Byte 4 Channel I2C IIC Analog-to-Digital ADC PGA Converter	Amazon	pack of 4	$ 11.88	1	$ 11.88
USB to Ethernet adapter	Plug & Play USB to Ethernet Adapter with PXE, MAC Address Clone Support	Amazon		$ 9.99	1	$ 9.99
					Total	$ 77.68

Figure 4. First purchase list for radiator sensors and associated hardware.

A	B	C	D	E	F	G
Item	**Description**	**Supplier**	**Notes**	**Cost**	**Multiplier**	**Total**
ProMini Programmer	HiLetgo FT232RL Mini USB to TTL Serial Converter Adapter Module 3.3V/5.5V	Amazon		$ 6.49	2	$ 12.98
RS485 CAN HAT	RS485 CAN HAT for Raspberry Pi 5/4B/3B+/3B/2B/B+/Zero/Zero W/WH/2W, Long-Distance Comm	Amazon		$ 16.31	1	$ 16.31
Pro Mini Microcontroller	HiLetgo Atmega328P 5V/16M Replace	Amazon	pack of 3	$ 13.99	1	$ 13.99
SPI to CAN converter	MCP2515 CAN Bus Module TJA1050 Receiver SPI Module	Amazon	pack of 3	$ 7.89	1	$ 7.89
USB A to MINI USB	3ft Mini USB cable, 2 Pack USB 2.0 to mini b charger cord USB type A male Fast Charging cable Co	Amazon	pack of 2	$ 7.99	1	$ 7.99
BMP388 Pressure Modules	BMP388 Digital Temperature Detector Module Atmospheric Air Pressure Sensor Module 24Bit	Amazon	Pack of 2	$ 15.95	1	$ 15.95
Ring Connector Terminal	TERM SCREW M3 4 PIN PCB	Digikey		$ 0.46	40	$ 18.40
Screw Terminals	TERM BLK 4POS SIDE ENTRY 5MM PCB	Digikey		$ 1.03	10	$ 10.25
Large Screw Terminal	TERM BLK 2POS SIDE ENTRY 5MM PCB	Digikey		$ 0.81	4	$ 3.24
M3 Rubber Mounts	uxcell Rubber Mounts 8pcs M3 Male/Female Vibration Isolator Shock Absorber, for Garage Motor A	Amazon	Pack of 8	$ 9.99	2	$ 19.98
10 A Fuse						
					Total	$126.98

Figure 5. Second purchase list for PCB, CAN HAT, and associated hardware.

Figures 4 and 5 show what we did for our first order, which included our microcontrollers, highest priority sensors and equipment for the radiator, and nervous system architecture components. For this order, we identified multiple sensor options at different price points and capability levels and discussed these with our advisor to balance performance and cost. We ordered in batches and prioritized components while scaling incrementally. Because our architecture was designed modularly, our scalable framework was well suited to ordering in batches because once we had our “nervous system” setup, integrating and ordering future parts did not affect the functionality of our existing system.

J. Sponsor Interaction

We worked closely with our sponsors to support the needs of the team and to align priorities. We had weekly meetings with both of our sponsors (Phil Davis and Zack Martin) to update them on our progress and to set immediate goals. Sometimes, only one sponsor could attend the meeting each week, but we typically updated Phil (faculty head of Motorsports) and Elizabeth (Motorsports President and member of the Mechanical Senior Design Team) throughout the week. During these meetings we provided structured updates, shared results, approved our components orders, and defined short term goals. With Phil, our conversations usually centered around ensuring our work aligns with the constraints of the competition, ordering parts, as well as getting his input on class assignments relevant to the project. We worked with Elizabeth to coordinate on timelines, and to make sure our work was integrated with the efforts of the Mechanical Senior Design team. Our discussions with Zack centered around more technical considerations such as communication protocols and sensor selection as he was more familiar with the electrical side of Motorsports and worked on a similar project during his undergrad years.

Earlier in the first semester, our sponsor interactions were less efficient. Initially, our meetings with our sponsors were planned by coordinating every team member's schedule with both of our advisors at the same time. With eight busy schedules to accommodate, this approach led to gaps without meetings. This was not a productive approach, and we struggled initially with getting clear deliverables and priorities from our advisors. However, after receiving feedback, we instead opted to meet with our advisors at the same time every week, and if a team member could not make the meeting or only one advisor could show up, we would update them on progress. We also opted to create a brief slide of our updates each week, summarizing what we accomplished, issues and questions, and

concrete goals for the next week. This also allowed us to keep meetings more focused and for our sponsors to understand our progress easily and provide feedback. We also discussed meeting in smaller groups with just Zack when more technical details needed to be discussed, as Phil deals more with coordinating the larger Motorsports team effort, and Zack has more directly relevant expertise. Implementing these changes resulted in much more productive communication with our sponsors.

Entering the second semester, our advisor engagement shifted primarily toward Zack during the weekly meetings, as Phil indicated a preference for fewer routine updates. We continued to meet consistently with Zack to share progress, address questions, and work through technical challenges. In February, we submitted a second order request to Phil for key components, including our PCB, mounting materials, and other electrical parts. Unfortunately, due to a series of logistical issues, including a package being lost on campus and a subsequent reorder also being misplaced, these materials were not received until less than a week prior to Design Day. In parallel, delays in the construction of the vehicle limited our ability to integrate and test the system on the car, even once the components arrived. We maintained communication with Zack throughout this period to ensure visibility into these constraints. Given these circumstances, we coordinated with the course instructor to adjust our approach, focusing on validating as much of the system as possible through a functional prototype and providing documentation to support future integration and testing by the Motorsports team ahead of competition.

Part 2. Technical

A. Technical Approach

The technical approach that was used for the Formula SAE data acquisition system revolved around developing a time-synchronized, distributed, and scalable sensing architecture that is analogous to a nervous system embedded throughout the vehicle. From high-frequency engine measurements to cooling system pressures and inertial vehicle-dynamics data, the various data sources needed to come together in a single, timestamped dataset that supports engineering decisions and performance optimizations. To develop this, the approach had to incorporate embedded systems engineering, digital communication protocols, and the mathematical principles of time-synchronization and sampling. The foundational concept for the “nervous system” was a series of "clients," "gateways," and a central "server" distributed across the chassis, reflecting a deliberately designed modular system for robustness, extensibility, and reuse across future FSAE seasons.

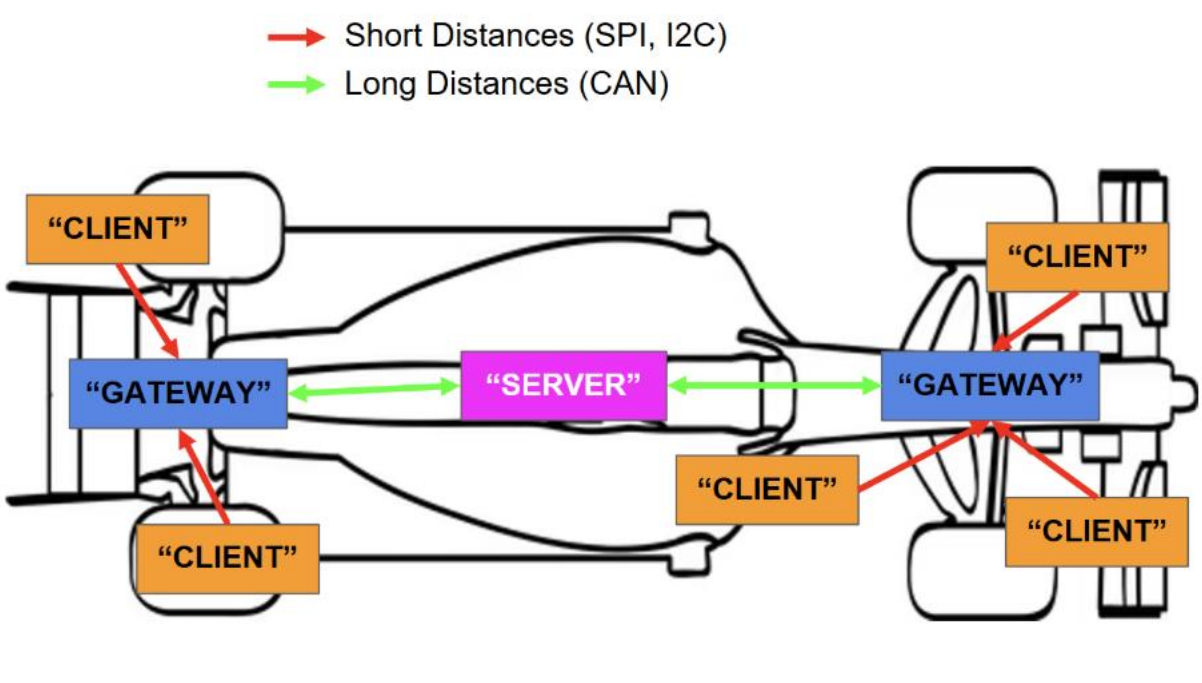


Figure 6. “Nervous System” Diagram

At the heart of the system is the Raspberry Pi, which acts as a server, data hub, and global clock. This unit needs to process simultaneous input from the RaceBox Mini S, the Performance Electronics PE3 ECU, and a suite of custom sensors for wheel speed, brake pressure, and radiator pressure and temperature channels.

Each subsystem is sampled at an appropriate rate as determined by its underlying dynamics. Wheel-speed sensors exhibiting rotational kinematics normally necessitate sampling in the hundreds of Hertz to capture transient changes in acceleration. Thermal

sensors in the cooling system change much more slowly, making sampling rates on the order of 1–10 Hz potentially suitable. This implies that the architecture must support different sampling frequencies without sacrificing timestamp alignment across streams. The system resolves this challenge by using centralized timestamping: whether the data is captured via SPI, I²C, analog-to-digital conversion, serial, or CAN, it gets a uniform timestamp at the Raspberry Pi. The intent is to ensure that subsequent merging of RaceBox IMU data, ECU telemetry, and sensor feedback allows meaningful quantitative analysis.

The engineering innovation for scalability and simplification of integration is the introduction of intermediary gateways. Each gateway combines a subset of sensors, converting short-range protocols like SPI or I²C into more robust, vehicle-wide formats such as CAN. This is rooted in the principles of networked embedded systems: short-range buses degrade over longer wire runs due to signal attenuation, electromagnetic interference, and clock skew, particularly in the racecar environment, which has constant stressors like vibration and heat. CAN, by contrast, is internally differentially signaled, as well as providing inherent arbitration and fault tolerance for multi-node communication across the length of the chassis. Gateways isolate sensor-side complexity such that team members can design client modules without needing to consider high-level networking, while also ensuring that the central hub receives data in a unified and robust format.

Sensor engineering has its own set of technical considerations. Most custom sensors require signal conditioning, such as voltage dividers for analog brake-pressure transducers, filtering for noisy temperature probes, or thresholding for Hall-effect wheel-speed sensors. The proposed approach involves collecting calibration datasets and fitting models for accurate conversion from raw voltages into physical units.

On the server side, software design ties the mathematical and engineering foundations of the system together. It runs a Python-based logging framework on the Raspberry Pi that parses CAN frames and merges all data streams into structured storage formats for post-run analysis. This requires developing asynchronous data-handling routines capable of managing multiple interfaces simultaneously. Engineering error-checking principles, such as CRC validation on CAN frames and time-drift correction using periodic synchronization messages, help maintain system reliability.

Once data is synchronized and stored, mathematical and software tools are used to obtain performance insights. For the cooling system development, as an example, analysis of temperature drop across the radiator involves application of conservation of energy and simple thermodynamic relations, with expected temperature gradients being determined by coolant mass flow rate and heat capacity. Kinematic equations can fuse IMU and GPS data to reconstruct vehicle trajectory, and wheel-speed differences are analyzed through rotational dynamics to infer slip ratios during corner entry. Each data stream has a unified timestamp, so multi-sensor correlations, such as measuring the impact of engine load on cooling airflow, are accomplished without any need for synchronization.

The system is thus a hierarchical engineering solution: clients capture raw physical data;

gateways transform, buffer, and standardize that data; and the server performs synchronized logging, storage, and analysis. This modular architecture directly satisfies different goals including synchronized multi-source capture, unified storage, integration with the cooling system senior design team, and a scalable platform for future FSAE vehicles. By designing each component using appropriate scientific and mathematical principles, our approach ensures that the final system is functional and analytically rigorous. Finally, the nervous-system model supplies the team with a flexible, extensible backbone for high-quality engineering data, driving deeper insight into the car's behavior and providing future teams with a robust foundation for iterative performance enhancements.

A Failure Modes and Effects Analysis was performed to identify the highest risk failure points in the proposed architecture and to guide design decisions before full vehicle integration. Because the system combines distributed sensor nodes, a Raspberry Pi server, and a separate battery system, the most important risks were communication loss, corrupted data, synchronization failure, power instability, and unusable output for the Motorsports team.

Table 2. FMEA summary for the Formula SAE data acquisition system

Failure Mode	Effect on System	Likely Cause	S	O	D	RPN	Mitigation / Design Change
Arduino node misses startup frame	Node does not transmit	Pi sends startup before node is ready	7	5	4	140	Added boot-flag synchronization and programmed each node to wait for its assigned startup frame before transmitting
Node floods CAN bus during transmit window	Excess traffic, possible collisions, difficult logging	Repeated transmission inside allowed time window	7	4	5	140	Limited node transmission to once per cycle
Unstable or incomplete power distribution	Pi reset, node reset, data loss	Battery/buck/fuse hardware unavailable or unstable wiring	10	4	6	240	Designed fused 20 V to 5 V regulated architecture, documented final build for handoff, PCB power distribution design
Sensor calibration error	Logged values inaccurate or misleading	Poor calibration model or unverified scaling	8	4	5	160	Bench-calibrated temperature sensor and verified BMP pressure/temperature conversions
Wiring / connector failure under vehicle conditions	Loss of data or intermittent operation	Vibration, heat, poor strain relief, poor routing	8	5	6	240	Selected modular gateway architecture, specified wire gauges, connector routing, and protected harness layout
Logged output incompatibl	Data unusable by team	Raw data format not aligned with visualization	6	4	2	48	Standardized logger output to sensor_id,time_since_startup,value

e with analysis tool		pipeline					
Loss of traceability for future teams	System hard to extend or troubleshoot	Poor documentation or inconsistent interfaces	6	5	5	150	Created wiring diagrams, protocol definitions, YAML config, code documentation, and implementation handoff

Severity (S), Occurrence (O), and Detectability (D) were scored on a 1–10 scale, with Risk Priority Number (RPN) calculated as RPN=S x O x D. The FMEA identified power distribution and wiring robustness as the highest-risk areas, followed by calibration accuracy, startup synchronization, and long-term maintainability. This analysis directly informed several design choices, including the use of boot-flag synchronization, once-per-cycle CAN transmission, a fused 20 V to 5 V power architecture and power architecture implementation for the gateway PCB, standardized CSV output for visualization compatibility, and a modular documented gateway structure for future Motorsports extension.

B. Results / Final Design Details

The final design approach involved a distributed data acquisition architecture using the Arduino Pro Mini microcontrollers as gateways to send sensor data in neat packages to the Raspberry Pi server via a noise resistant CAN bus. This design was chosen due to the requirement for the sensors to be physically distributed across the chassis and radiator while still creating one synchronized dataset for visualization. In our final design, the Raspberry Pi functions as the DAQ hub, CAN receiver, synchronization controller, timestamp reference, and file logger. The Arduino-based gateway nodes act as local sensor interfaces that read individual sensors via a variety of protocols, package sensor data into CAN frames, and transmit to the server in assigned time windows. This can then be viewed with ease by the Motorsports team through the visualization tool.

Raspberry Pi and Core Bring Up

We began our project with an unorganized assortment of electrical parts that we inherited from the previous ECE senior design team. Most of the parts were no longer usable because they were damaged; however, we wanted to reuse as much as possible in consideration of our economic and environmental impact. We were able to repurpose a Raspberry Pi 4 that served as our central data logging hub, as well as some cables, one usable analog-to-digital converter (ADC), and a set of Xbee modules for wireless RF communication. We started our work by gathering specs for these components and determining what parts we were missing.

Because the Raspberry Pi is the central hub to our DAQ system, a significant amount of time was spent setting up the Raspberry Pi to be used throughout the entire design process. This involved working with Vanderbilt ECE professor Dr. Andrew Sternberg to ensure the Raspberry Pi set up would work within our system constraints. We had to acquire hardware such as microSD cards and readers, a 5 V power supply, HDMI displays, a keyboard and mouse, and an ethernet cable. We set up the Pi's operating system using Raspberry Pi Imager, and we were able to successfully boot. We changed

the boot mode to text console for lower overhead and easier headless operation. We then set up the device on the vuDevices wireless network so we could enable SSH and connect to the Pi without a dedicated monitor.

Once we were able to wirelessly connect to the Raspberry Pi, we connected our Xbee RF modules and confirmed that the USB serial devices could be connected via the Port. We wrote a Python dummy script to send data from the Pi to our local laptops via the modules, which resulted in successfully logging data from the Raspberry Pi. We also wrote scripts to save data to .csv files on the Pi and then pulled those files to our local computer to access them headlessly, confirming that our serial communication worked. This was a critical progress marker for the DAQ system development because the RaceBox uses the same protocol and provided us with a proof of concept that data can be easily recorded into files on the Pi's SD. This dual system allows us to both stream data wirelessly and to have a backup to upload in case of noisy transmission. The Motorsports team ultimately decided that they would prefer post-run telemetry data rather than live transmission, but the Xbee modules allow this to occur if future teams would prefer real-time access.

After configuring our central server and establishing our gateway and hub system with the Arduino Pro Mini microcontrollers, we were able to wire and test our connection using a breadboard prototype. As the Raspberry Pi GPIOs are 3.3 V and the Arduinos transmit messages at 5 V, we configured a voltage divider to power the Pro Mini and successfully ran a blinking light test as an indicator of correct configuration. We installed Arduino CLI on the Raspberry Pi which allows us to compile and update the Pro Mini firmware directly from the Pi. We then ran a simple echo test program on the Pro Mini to observe bidirectional communication with the Pi as seen in figure 6, which verified the functionality of our equipment and transfer of data between our gateway and server, which is the core infrastructure that we used to synchronize our timestamps and integrate different data types.

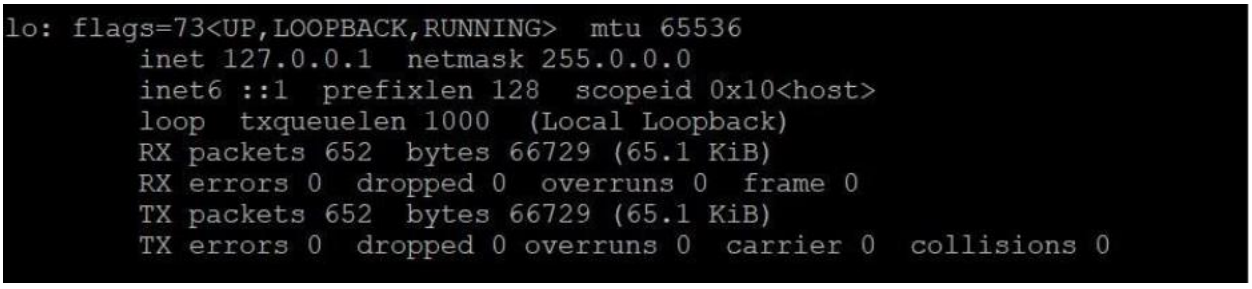


Figure 7. Raspberry Pi terminal messages showing transmission (TX) and receiving (RX) of data packets to and from the Arduinos without errors.

Verifying this connection was essential as it allows us to synchronize the different sensor sources and allows the software team to write programs to send the data between the sources.

Power Distribution and Gateway PCB Design

The battery system uses a fused 20 V input from a DeWalt battery adapter and a 4 A

blade fuse, which is stepped down to a regulated 5 V rail using a buck converter. The 5 V rail powers the Raspberry Pi and gateway boards, while common ground is maintained across the Pi, Arduino nodes, CAN HAT, MCP2515 modules, and sensors. The schematic includes the fuse, buck converter terminals, gateway PCB power terminals, status LEDs, and CAN_H/CAN_L routing between the Pi server and gateway nodes.

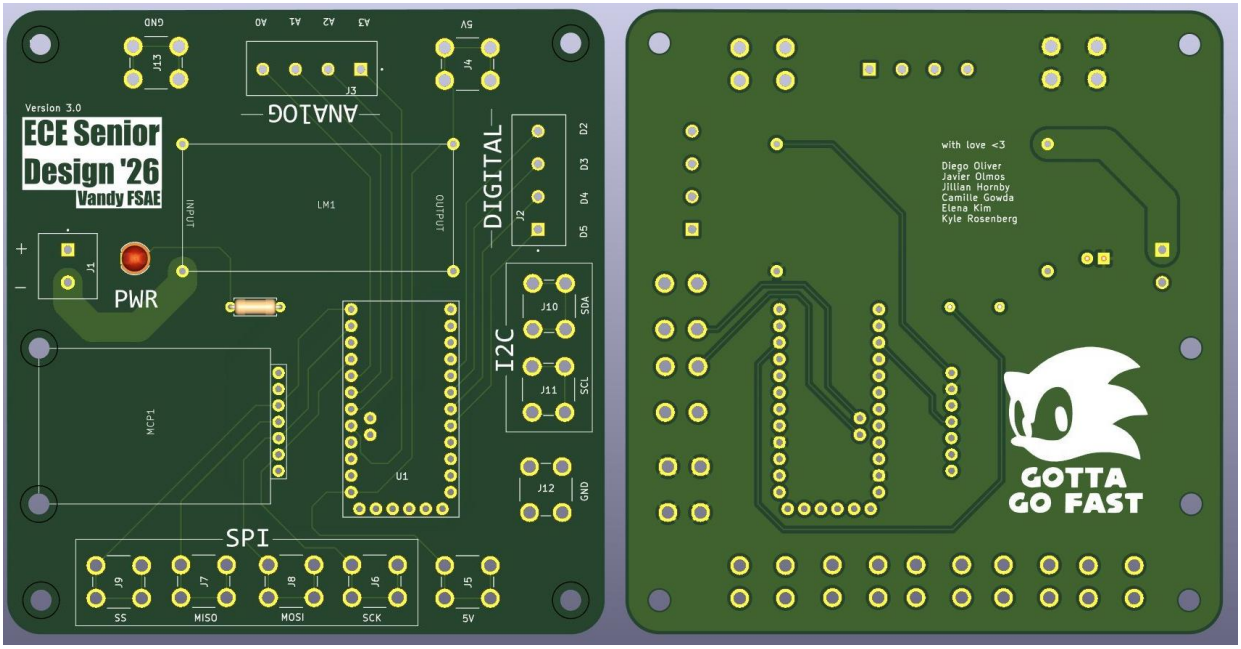


Figure 8. Front (left) and back (right) of the final iteration of the custom gateway PCB

The gateway PCBs were designed to be physically distributed near local sensor clusters to minimize wiring distance and improve signal integrity. Each board acts as a compact power and communication hub, collecting local sensor data and forwarding processed data packets onto the CAN bus. The PCB integrates three primary components: an Arduino Pro Mini, a buck (step-down) voltage converter, and an MCP2515 CAN interface module. Power is supplied from a DeWalt 20 V battery connected to an adapter through the buck converter to generate a regulated 5 V rail for the logic and communication components. The Arduino Pro Mini is then interconnected with the MCP2515 CAN module through dedicated signal traces on the PCB, enabling CAN frame transmission to the vehicle network. The CAN interface is routed to its own screw terminal labeled CAN_H and CAN_L for connection to the main CAN wiring harness.

Around these core modules, the PCB provides multiple standardized sensor interfaces to support mixed-signal data acquisition. Analog and digital sensors connect through labeled side-entry screw terminals that route directly to GPIO pins on the Arduino Pro Mini via controlled copper traces. Higher-bandwidth or protocol-based sensors, such as barometric pressure sensors, are connected using SPI and I2C interfaces that have grouped through-hole terminal blocks (including SCL, SDA, MOSI, MISO, and SCK connections). The board is also physically labeled and segmented (ANALOG, DIGITAL, I2C, SPI, PWR) to simplify wiring during installation.

Compared to earlier iterations as seen in Appendix B, two key PCB design improvements were implemented. First, a dedicated ground terminal was set up on the back side of the

board. This gave us the advantage of simplifying our wiring to common ground as well as preventing any signal degradation potentially caused by electromagnetic interference in the vehicle's environment. Second, the battery input trace width was significantly increased to safely accommodate higher current draw from the 20 V supply without excessive voltage drop or overheating. The overall PCB layout emphasizes mechanical robustness and has securely mounted screw terminals as well as modular placement of components to withstand vibration and real-world vehicle integration conditions.

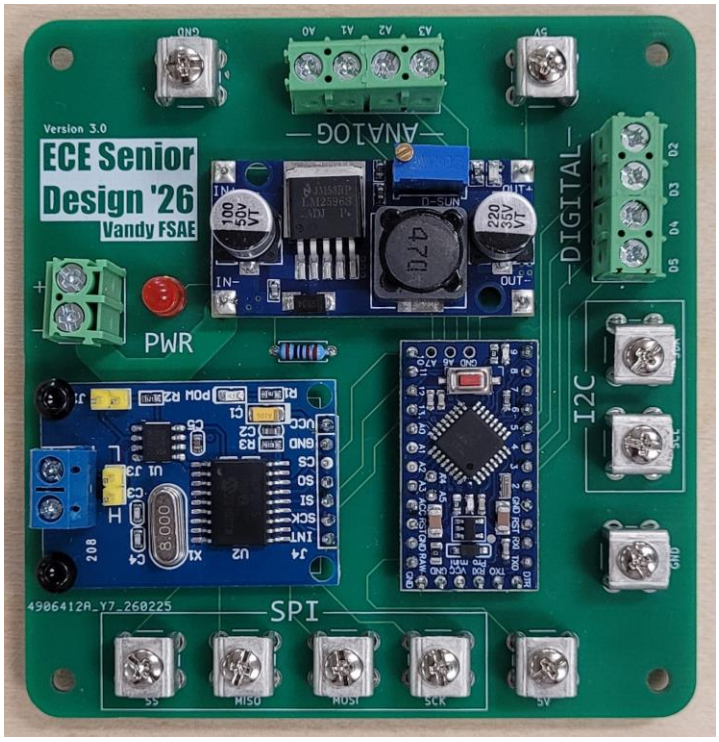


Figure 9. Fully assembled data acquisition board

Sensor Integration and Testing

The sensor selection process and the broader system design, driven by the performance requirements of the FSAE vehicle and the constraints of the cooling subsystem, were ultimately the guide in how we approached our hardware and software design choices. The team began by identifying the environmental and operational conditions each sensor must withstand, including coolant temperatures ranging from 150–230°F during operation, inlet air pressures corresponding to vehicle speeds between 20 and 75 mph, and the electrical requirements of a DeWalt 20V battery system. These parameters guided component selection and ensured that each sensing element would remain reliable under the thermal, mechanical, and electrical stresses present in the vehicle. Components such as the BMP388 barometric pressure sensor, MCP2515 CAN interface, ADS1115 precision ADC, LM2596 and DROK buck converters, and the Atmega328P Pro Mini microcontrollers form the foundation of the data acquisition architecture and have been selected based on their compatibility with the Raspberry Pi, CAN bus requirements, and the noise environment of the vehicle. These parts collectively support pressure measurements, analog conversions, microcontroller-level preprocessing, and stable

power regulation for each subsystem.

After setting up the Raspberry Pi and Arduino connection, the first sensor integration was performed with the radiator temperature sensing hardware. Analog NPT temperature sensors were obtained from our advisor that connect directly to the radiator inlet and outlet ports to measure the coolant temperature. Its body behaves like a resistor that changes its resistance in response to temperature changes. These temperature sensors were calibrated by submerging them in water at different temperatures ranging from 30 to 210 degrees Fahrenheit and collecting data points using two thermometers. After calibration, a simple telemetry test was run that showed that this sensor achieved an accuracy of +/- 1.25 degrees Fahrenheit, which was much better than our original goal of +/- 5 degrees Fahrenheit.

Our next sensor integration involved a BMP388 digital pressure module. This sensor measures the pressure within an environment and transmits data via I2C or SPI. The pressure module measurements were confirmed by performing a 10-minute telemetry test. The sensor was sealed inside of a box with hose lines attached used to increase or decrease the pressure with a digital pressure monitor. The recorded values were within +/- 0.005 psi of the pressure values read on the digital monitor. After the sensing hardware and conversion methods were validated, the next step was packaging those values into robust CAN messages for transport to the Raspberry Pi.

CAN Bus and Data Logging

To configure the Raspberry Pi to interface with the CAN network, we ordered the RS485/CAN HAT using the socket CAN interface. The CAN HAT allowed the Pi to use Python-based CAN libraries to transmit startup synchronization frames, receive data frames from the Arduino nodes, and write these decoded frames directly to the CSV files. The advantage of this approach is that it maintains higher level logging and data formatting on the Pi instead of on the local microcontrollers, making the system easier to scale as new sensors are added.

Each gateway node uses an Arduino Pro Mini connected to a MCP2515 CAN controller, which allows the Arduino to read local sensors and send compact 4-byte CAN messages. The MCP2515 modules were configured at 125 kbps using an 8 MHz oscillator setting. This was done explicitly to ensure the Arduino modules matched the oscillator rate of the software configuration. Without this, the gateway and server could not reliably decode each other's frames.

To configure our CAN communication protocol, we used a boot flag to achieve synchronization and reduce CAN message collisions. At startup, the Raspberry Pi sends boot flags to each Arduino node. Each boot flag contains a transmit offset and transmit window. Nodes and individual sensors on each gateway are assigned CAN IDs. The current timing cycle is 500 milliseconds, so once a node receives its boot flag, it begins transmitting only during its assigned time slot. This was implemented to prevent collisions on the bus. This message structure was chosen because it is compact and easy to decode.

The Python logger maps each CAN ID to a named sensor channel in the visualization executable. The first two bytes of each message are decoded into time since startup, and the final two bytes are decoded into the scaled sensor value. This structure can be seen in figure 10 with pseudo data being logged on the Raspberry Pi. This lets the DAQ system directly generate files that can be used by the visualization tool without requiring manual reformatting.

```
jilli@raspberrypi:~ $ python3 pi_logger_v3.py
CAN bus initialized on can0
Sending boot flags...
Boot flag sent -> ID=0x000 offset=0ms window=150ms
Boot flag sent -> ID=0x00F offset=250ms window=150ms
Logging to can_sensor_log_20260402_191623.csv
Listening... Press Ctrl+C to stop.
RAW ID=0x003 DLC=4 DATA=['0x5c', '0xc8', '0x8', '0x68']
bmp_temperature_f,2375.200,21.5200
RAW ID=0x004 DLC=4 DATA=['0x5c', '0xc8', '0x27', '0x4']
bmp_pressure_psi,2375.200,99.8800
RAW ID=0x003 DLC=4 DATA=['0x5c', '0xcd', '0x8', '0x68']
bmp_temperature_f,2375.700,21.5200
RAW ID=0x004 DLC=4 DATA=['0x5c', '0xcd', '0x27', '0x4']
bmp_pressure_psi,2375.700,99.8800
RAW ID=0x003 DLC=4 DATA=['0x5c', '0xd2', '0x8', '0x68']
bmp_temperature_f,2376.200,21.5200
RAW ID=0x004 DLC=4 DATA=['0x5c', '0xd2', '0x27', '0x4']
bmp_pressure_psi,2376.200,99.8800
RAW ID=0x003 DLC=4 DATA=['0x5c', '0xd7', '0x8', '0x68']
bmp_temperature_f,2376.700,21.5200
RAW ID=0x004 DLC=4 DATA=['0x5c', '0xd7', '0x27', '0x4']
```

Figure 10. Successful multi-node data logging with pseudo data via CAN bus

The design evolved from a two-Arduino proof-of-concept into a Raspberry Pi-centered architecture. Initially, two Arduinos were used to confirm that MCP2515-based CAN communication worked. After this was validated, one Arduino receiver was replaced with the Raspberry Pi and CAN HAT so that the Pi could serve as the central logger. Early tests showed that the Arduino nodes could miss startup messages, so a boot-flag synchronization process was added. The Arduino nodes now wait until they receive their assigned boot frame before transmitting. Another key iteration was explicitly configuring the Arduino MCP2515 oscillator as 8 MHz, which was required for reliable communication with the Pi. We then added another Arduino back in so that we could successfully log data from multiple nodes at one time.

Software and Data Visualization

The software system developed for this project consists of two components: a data logging pipeline running on a Raspberry Pi and a desktop-based visualization application designed for interactive analysis of collected telemetry data. Together, these components form a complete workflow that enables reliable data acquisition from the vehicle and efficient post-run analysis by the FSAE team. The system was designed with scalability as a core principle, allowing new sensors and data channels to be incorporated with minimal changes to the underlying codebase.

The data logging pipeline follows a layered architecture that separates hardware interaction, message decoding, and data storage. Sensor data is transmitted from the ECU and other onboard sensors through a gateway microcontroller over the CAN bus, received by the Raspberry Pi via a SPI interface, decoded using a configurable schema, and written to CSV files. Rather than hardcoding message formats, the system uses a YAML configuration file that defines each CAN message and its associated signals, including byte positions, scaling factors, and offsets. This design allows new sensors to be added simply by updating the configuration file, eliminating the need to modify Python source code. The decoding process extracts raw byte data from each CAN frame, converts it into meaningful physical values, and logs it alongside timestamps. The logging script ensures robustness by sending data to disk after every write and handling shutdowns to prevent data loss. Additional modules support analog sensor integration through an ADC interface and temperature conversion utilities, enabling the system to handle both digital and analog inputs. A serial communication module also provides the foundation for real-time wireless telemetry, allowing data to be transmitted while simultaneously maintaining a local backup on the Raspberry Pi.

```python
def decode_frame(frame: CanFrame, config: dict) -> Dict[str, float]:
    """Turn a CAN frame into named values using config.yaml."""
    msg_cfg = config.get("messages", {}).get(hex(frame.can_id), None)
    if msg_cfg is None:
        return {}

    values: Dict[str, float] = {}
    for name, field in msg_cfg.items():
        start = field["byte"]
        size = field["size"]
        scale = field.get("scale", 1.0)
        offset = field.get("offset", 0.0)

        raw_bytes = frame.data[start:start+size]
        raw = int.from_bytes(raw_bytes, "big", signed=False)
        values[name] = raw * scale + offset

    return values
```

```yaml
messages:
  "0x100":
    coolant_temp_C:
      byte: 0
      size: 2
      scale: 0.1
      offset: -40.0

    radiator_pressure_kPa:
      byte: 2
      size: 2
      scale: 0.1
      offset: 0.0
```

Figure 11. The YAML configuration file (right) defines the byte layout for each CAN signal, which decode_frame() (left) applies at runtime to produce named, physical-unit values.

The visualization application complements the logging pipeline by providing a structured and interactive environment for analyzing telemetry data. Built using PySide6 for the user interface and Matplotlib for plotting, the application separates data processing, state management, and UI logic into distinct layers to maintain modularity. At its core is a data model that represents each session as a combination of time-indexed and distance-indexed datasets, with the distance-indexed datasets being derived by integrating vehicle speed over time to align data with track position. The system automatically computes derived channels such as speed conversions and acceleration metrics, enabling deeper analysis without requiring additional work. The interface is organized into multiple pages, each containing vertically stacked partitions that allow different signals to be visualized with independent scaling. Users can dynamically assign channels to plots through drag-and-drop interactions, and a synchronized cursor provides real-time value readouts across all displayed signals. The application also includes live file monitoring, allowing it

to update automatically as new data is logged, and supports exporting selected data for use in external tools. These features collectively provide a flexible and efficient workflow for the FSAE team to analyze the telemetry data.

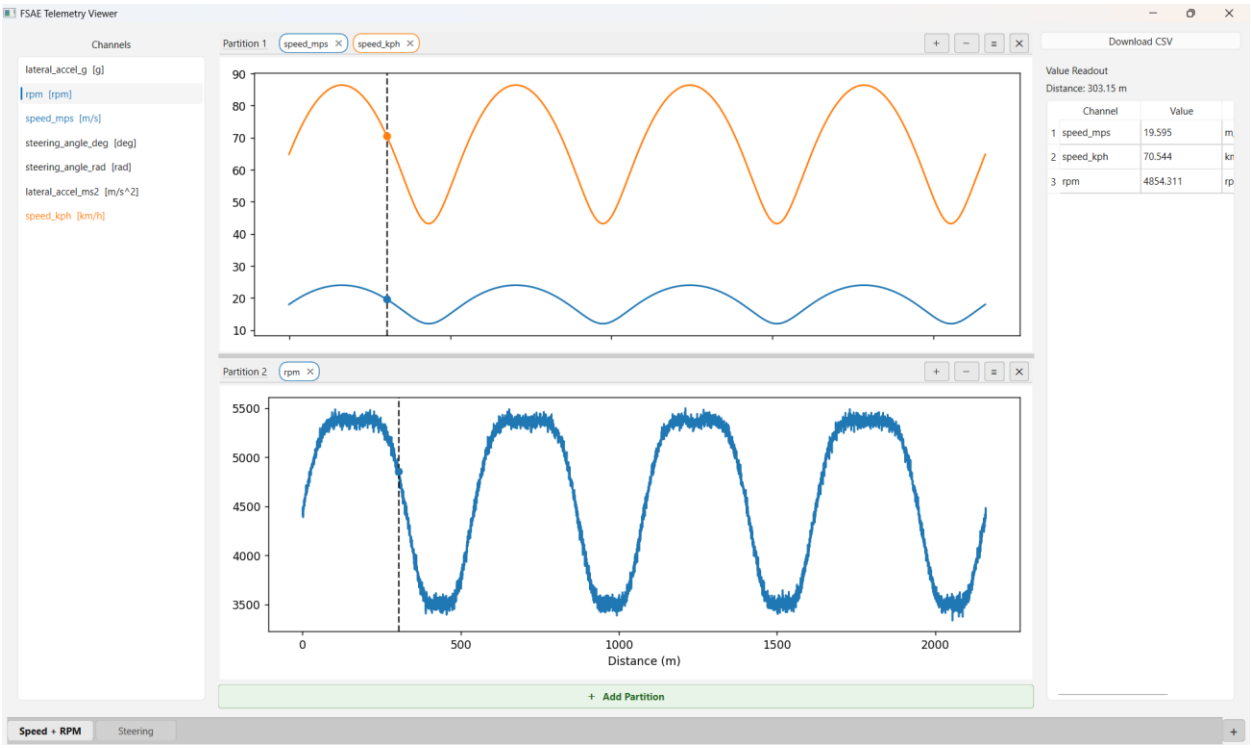


Figure 12. The FSAE Telemetry Viewer displaying a logged session across multiple plot partitions. A cursor shows channel values at a selected track distance in the readout panel on the right.

Overall, the system delivers an accurate and scalable solution for data acquisition and analysis, supporting the needs of the FSAE team while remaining adaptable for future expansion. The YAML-driven decoding pipeline means new sensors can be integrated by updating a single configuration file, with no changes to the logging code. On the analysis side, the visualization tool is channel-agnostic, meaning it automatically discovers and displays whatever signals are present in the loaded CSV, so the interface scales naturally as the team adds more sensors in the future.

Physical Integration

All subsystems of the data acquisition system were designed with physical integration into the Motorsports car in mind. Maintaining data integrity within a moving vehicle poses many challenges such as mechanical vibration, space constraints, vehicle temperature, and other physical constraints as described in Section D. Unfortunately, the physical integration of our subsystems with the physical car was unable to be executed due to several factors we outline in Part 3, including delays in supply parts as well as delays from the Motorsports team regarding the actual construction of the car. However, we were able to create a well-documented plan if members of the Motorsports team choose to

integrate with the car in the future.

The Raspberry Pi hub and battery system is to be placed on a tray at the front of the car that can be easily accessed to charge the battery and obtain logged data from the Raspberry Pi. A CAN bus must run from the Raspberry Pi hub through the car which provides a robust data transmission line that is shielded from electromagnetic resistance and maintains data integrity over long distances.

The sensors must also be securely mounted within the vehicle to prevent damage and ensure data collection is consistent. One of the biggest challenges with the cooling system sensors is the high temperatures within the radiator. Placing the pressure sensor within the radiator would prevent it from properly obtaining data and likely cause it to burn. To prevent this safety hazard and ensure data accuracy, a pneumatic line needs to be connected to a static port at the measurement site and runs to the sensor which is placed in a cooler zone of the car. To obtain inlet and outlet radiator pressures, a tube should be placed at each of these measurement sites. This design choice also ensures that the placement of the pressure sensors does not disrupt the airflow within the radiator since they are mounted remotely. The coolant temperature sensors are NPT sensors that are threaded to screw directly into radiator, which ensures a tight seal that resists effects from vibrations.

Wiring diagrams as seen in Figure 13 were provided to the Motorsports team to provide detailed instructions for connections and wiring sizes for physical integration and future changes.

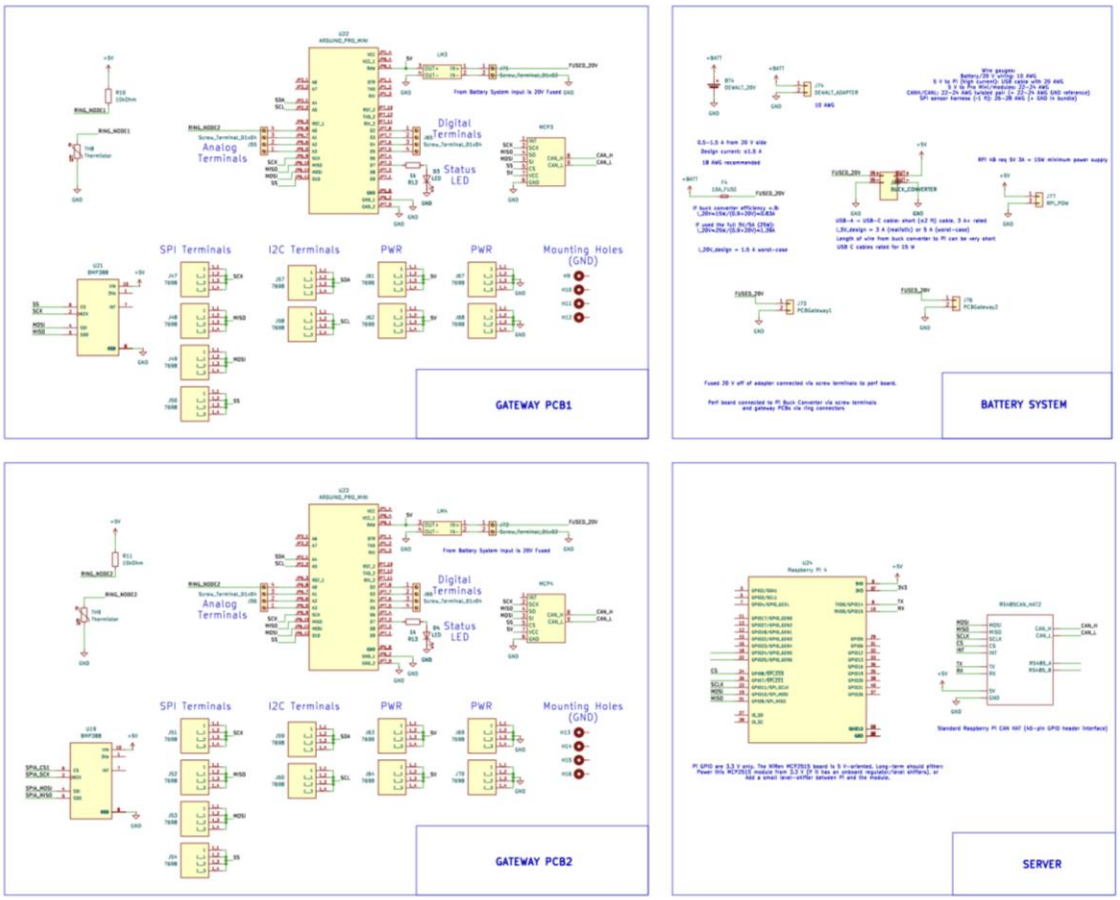


Figure 13. Wiring diagram for Gateway PCBs, Raspberry Pi, and Battery System

C. Design Validation

Our design was validated incrementally by isolating and testing subsystems and integrating towards our full data acquisition pipeline. The goal of validation in our system was to confirm our systems ability to gather accurate sensor data, synchronize Arduino nodes, receive sensor frames on the Pi, decode those measured values correctly, and sync to the software visualization tool.

Embedded Systems

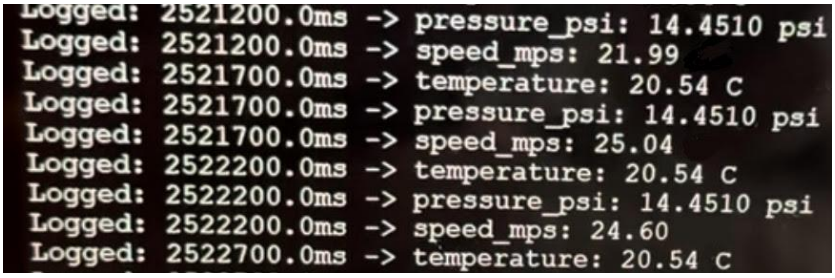
To calibrate and validate our sensors, we first connected them directly to the Pi for easier debugging and visualization. We then used the Pro Mini to ensure that the code to gather data was compatible and we could view it on the serial monitor.

To validate our CAN interface, we first confirmed that the Pi detected the MCP2515 CAN HAT, and verified that the socket CAN interface appeared. After enabling SPI and configuring the MCP2515 overlay, the Pi successfully initialized can0. The interface was then manually reset and brought up at 125 kbps. Successful initialization was confirmed

when it reported the interface in the UP state. This demonstrated that the Pi-side CAN hardware and Linux CAN driver were functioning properly before Arduino nodes were added.

Each Arduino node was validated first by monitoring its serial output. Before receiving a boot flag, each node printed a waiting message, confirming that it would not transmit prematurely. After the Pi sent the boot frame, the Arduino printed that it had received its offset and transmit window, then began sending CAN messages. This confirmed that the boot flag synchronization strategy worked.

The complete communication path was validated by replacing one Arduino receiver with the Raspberry Pi and confirming that the Pi could both transmit boot frames and receive data frames. The successful test showed that the Pi transmitted boot flags on CAN IDs 0x000 and 0x00F, after which the Arduino nodes transmitted frames such as 0x001, 0x002, 0x003, and 0x004. Receiving the expected IDs and payload patterns on the Pi confirmed that the physical CAN wiring, bitrate, oscillator configuration, and software packet structure were all compatible. After first validating dual node synchronization to the Pi, Node 2 was updated to transmit BMP pressure and temperature values instead of marker bytes. The Pi logger decoded these values into named sensor channels and the logged CSV was validated against the required input format for the visualization tool. The final logger writes one row per sensor measurement using the columns sensor ID, time since startup, and value. This long-format structure allows multiple sensors with different sampling rates to be stored in the same file without requiring a fixed-width table. It also allows the visualization tool to filter and plot values by sensor channel.



sensor_id	time_since_startup_ms	value
temperature	2517700	20.55
pressure_psi	2517700	14.451
speed_mps	2517700	24.51
temperature	2518200	20.55
pressure_psi	2518200	14.451
speed_mps	2518200	33.94
temperature	2518700	20.55
pressure_psi	2518700	14.451
speed_mps	2518700	32.88
temperature	2519200	20.55
pressure_psi	2519200	14.451
speed_mps	2519200	31.18
temperature	2519700	20.55
pressure_psi	2519700	14.451

Figure 14. Raspberry Pi log (left) and data logged in CSV format (right) obtained from a full-system test of two gateway nodes with multiple sensors connected

Sensors

At the hardware level, we conducted both component-specific and system-wide validation. Individual sensors were tested to confirm they operate within their correct temperature, pressure, or resistance ranges. We ensured the sensors respond smoothly during controlled bench tests, including the temperature and pressure tests as described in Section 2, Part B. These steps ensure that the physical architecture of the DAQ system works as intended and is durable before installation on the car.

During one of our early pressure tests, we noticed data dropouts as seen in Figure 15 that resulted in unrealistic values. To determine the root cause of this issue, we connected the system to a logic analyzer and read the hexadecimal values being sent. We found

that when the data dropouts were occurring, the sensor wasn't responding to requests to send data packages, which likely pointed to a loose connection preventing data from being sent. To resolve this issue, we checked the soldered joints and created new wire connections that were more robust. This resulted in the pressure sensor performing as expected with no data dropouts over a new 5-minute telemetry test.

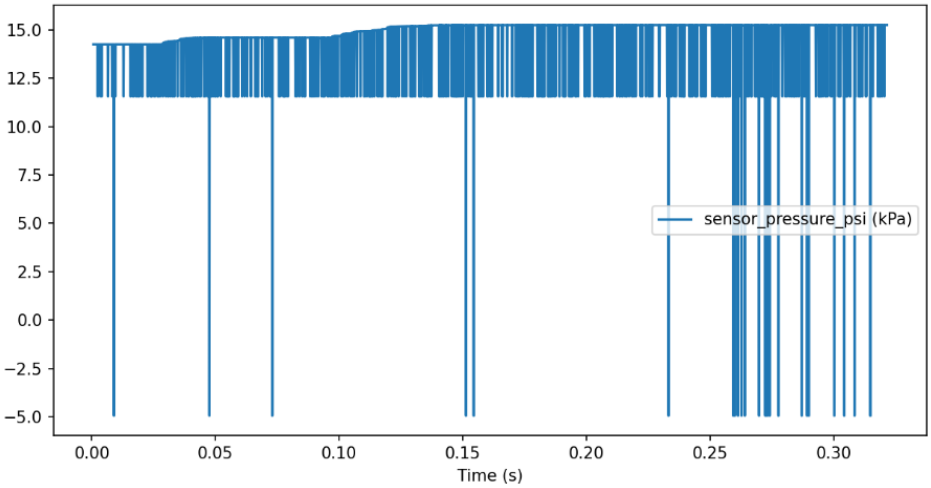


Figure 15. Pressure sensor test visualization showing data dropouts

While the Motorsports car is not currently ready for testing, the Motorsports team will perform full vehicle-level testing on the FSAE car to verify the system's readiness for competition. This includes validating radiator flow, coolant temperature, and pressure measurements in collaboration with the Mechanical Engineering Senior Design Team. Real driving tests will confirm that the wiring, sensors, and gateways remain reliable under vibration, heat, and motion. Results from these tests will guide design adjustments, such as tuning sampling rates, updating configuration files, or refining visualization tools. Our documentation for use of our system has been provided so that when the car construction is complete, the team will be able to implement the data acquisition system.

Determining whether the project meets all user requirements will involve continuous communication with major stakeholders. If the data supports their decisions, the technology integrates cleanly into their workflow, and tests show that it is accurate and maintainable, then the system will be successful.

Data Visualization

Our design validation process for the software component involved a combination of software unit tests and iterative improvement. We began by validating the software components of the data acquisition platform. On the visualization side, we used DevOps practices to incrementally test the software we developed. After completing an initial viable product, we created a mock data generation pipeline to verify various features, such as the formation of several plot graphs, zooming in and out, and selecting different data channels.

We also validated the system's ability to handle streaming data by simulating continuous sensor inputs and confirming that plots are updated correctly. Additional validation steps included verifying that axis scaling and labeling were correctly applied for different sensor types, testing the responsiveness of the interface when switching between channels, and confirming that the visualization remained stable when datasets were loaded. Additionally, we corresponded with our sponsor and advisor to ask for feedback on several versions of our data visualization system. These steps allowed us to confirm early on that the logging and visualization pipeline functions as intended and can handle the dynamic conditions of real FSAE testing.

Part 3. Summary & Reflection

Our team was tasked to create a data acquisition system for the Vanderbilt Motorsports team. By combining data from different sensor sources and visualizing the data in a centralized platform, we aimed to help the Motorsports team make more informed decisions about their car. To achieve this, we designed a distributed data acquisition architecture capable of future scaling for the team to gain insight into vehicle dynamics.

Over the course of the project, we accomplished the design and partial implementation of our system. We configured our central server, created Python-based logging scripts, added wireless communication capabilities, validated communication between our gateway nodes and our server using a noise-resistant CAN bus, coordinated multi-source data synchronization, and ensured that data could be written into a visualization compatible format. We also created a scalable visualization tool that allows the Motorsports team to view and interpret logged data, designed the gateway PCB and battery distribution architecture, created wiring diagrams, and documented the system so the team can implement this before the competition once the vehicle is ready.

We were able to meet many of the core sponsor requirements related to system architecture, logging, and visualization, although we could not complete the final vehicle installation. The largest barrier was not the design itself, but the delayed arrival of electrical parts required for the PCB and battery system. Additionally, the Motorsports team did not have the car ready to test for full system installation before Design Day. Due to these challenges, the remaining work is mostly physical mounting and verifying stability in the car environment ahead of the competition. We documented all of our work for our wiring diagrams, code, visualization tool, and implementation to support the integration of our system before the competition and will host our resources on the Vanderbilt Motorsports drive.

Through these challenges, our team learned the importance of thorough documentation. Taking detailed notes, making wiring diagrams to map physical connections, and using Github to ensure code was well annotated ensured that although our definition of success shifted, we were able to create a handoff package for the team. This experience proved that documentation is essential for a successful project.

Additionally, our sponsor interactions improved our effective communication abilities. Our shift during the first semester to making structured update slides and coming into regularly scheduled advisor meetings with a plan, specific action items, and communicating progress and roadblocks clearly led us to increasing our pace of progress. Ultimately, we found that transparent communication within the group, with our advisors, and with our course instructors is essential to success.

The project also enhanced our ability to attain the ABET student outcomes in terms of teamwork and acquisition of new skills. Our project covered a wide variety of skills across embedded systems, hardware design, and software. This required us to divide tasks based on strengths, and to quickly acquire and apply concepts we were previously unfamiliar with. This project required us to break down a complex problem into smaller subsystems, including sensor interfacing, CAN communication, power distribution, data

formatting, synchronization, and visualization. We also applied engineering design principles to produce a solution that considered factors such as safety, usability, maintainability, cost, and scalability.

Overall, while the final system was limited by our purchase order delays and vehicle readiness, the project produced a functional and validated foundation for Vanderbilt Motorsports' data acquisition system. We accomplished the core architecture, communication, logging, visualization, and documentation goals, and we provided a clear path for completing vehicle integration.

Part 4. Acknowledgments

We would like to thank our project advisors, Phil Davis and Zack Martin, for their continued guidance, expertise, and support throughout the development of our data acquisition system. We also extend our appreciation to the Vanderbilt Motorsports team and the Mechanical Engineering Senior Design team for their collaboration and assistance during this phase of the project. Their insights and feedback have been invaluable, and we look forward to continuing our work together in the second half of the project.

Part 5. References

[1] RaceBox Ltd., RaceBox Mini S User Documentation and Data Specifications, RaceBox, 2024. [Online]. Available: https://www.racebox.pro
[2] Raspberry Pi Foundation, Raspberry Pi 4 Model B Technical Documentation, Raspberry Pi Ltd., 2024. [Online]. Available: https://www.raspberrypi.com/documentation/
[3] Performance Electronics Ltd., PE3 Series Engine Control Unit Manual, Version 3.0, Performance Electronics Ltd., 2024.
[4] Vanderbilt University Motorsports, Team Shared Documents Repository, Vanderbilt University, 2025. [Online]. Available: https://vanderbilt365-my.sharepoint.com/sites/motorsports/Shared%20Documents
[5] International Organization for Standardization, ISO 11898-1: Road Vehicles — Controller Area Network (CAN) — Part 1: Data Link Layer and Physical Signalling, ISO, Geneva, Switzerland, 2015.
[6] SAE International, SAE J1128: Low Voltage Primary Cable, SAE, Warrendale, PA, USA, 2010.
[7] SAE International, SAE J1742: Connections for On-Board Road Vehicle Electrical Wiring Harnesses, SAE, Warrendale, PA, USA, 2010.
[8] International Organization for Standardization, ISO 11452: Road Vehicles — Component Test Methods for Electrical Disturbances from Narrowband Radiated Electromagnetic Energy, ISO, Geneva, Switzerland, 2015.
[9] IEEE, IEEE Std 829-2008: IEEE Standard for Software and System Test Documentation, IEEE, New York, NY, USA, 2008.
[10] SAE International, Formula SAE Rules 2026, SAE, Warrendale, PA, USA, 2025.

Part 6. Appendices

A. Prior Gantt Charts and Network Diagrams

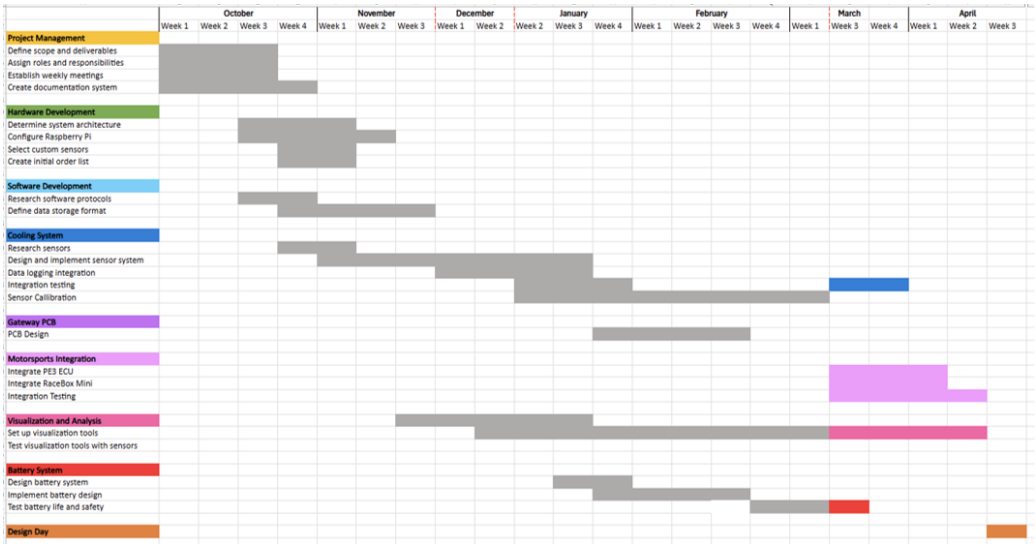


Figure 16. Gantt chart updated as of 3/17/2026

Figure 17. Gantt chart updated as of 2/3/2026

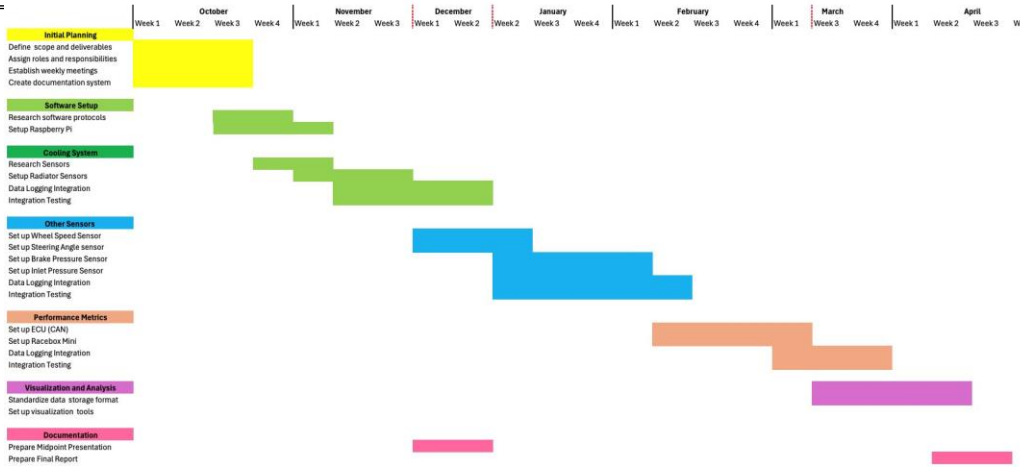


Figure 18. Gantt chart updated as of 10/30/2025

B. Supporting Information

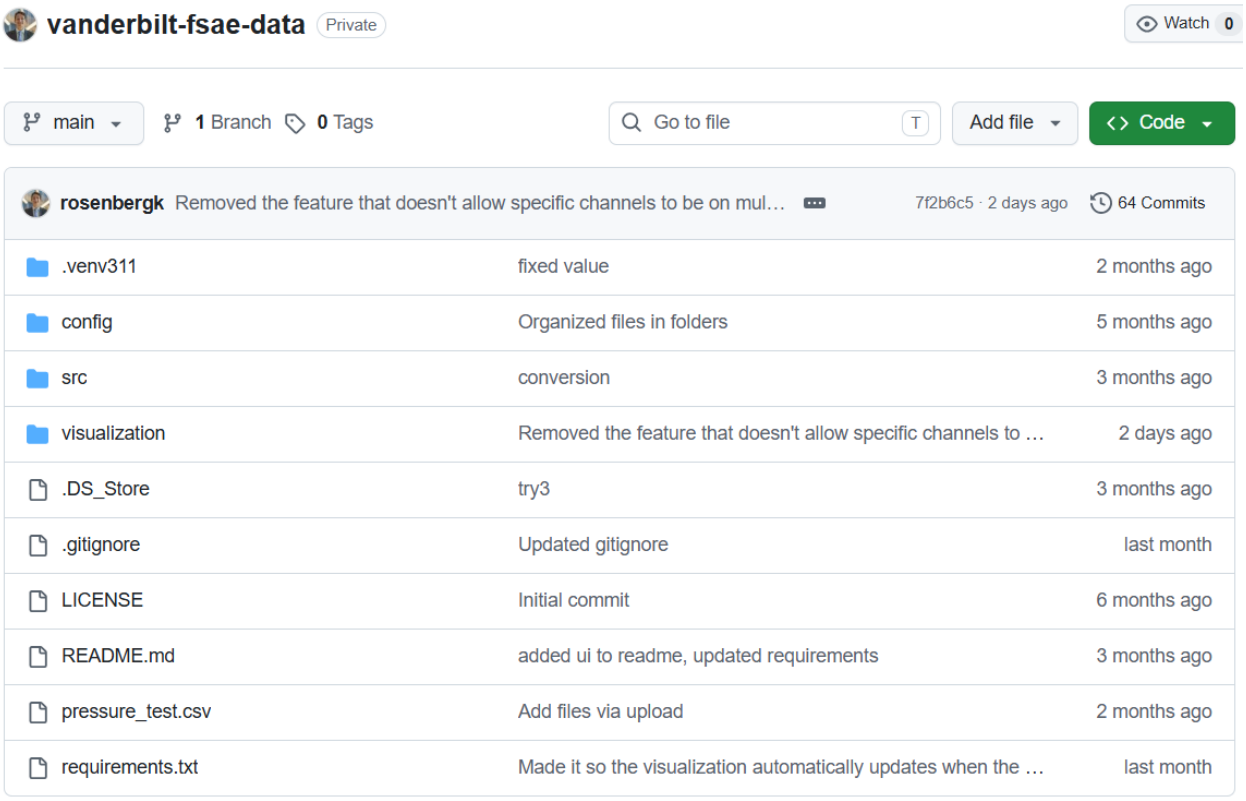


Figure 19. GitHub repository structure for the FSAE data acquisition and visualization system, showing organized modules for configuration, data processing, and visualization.

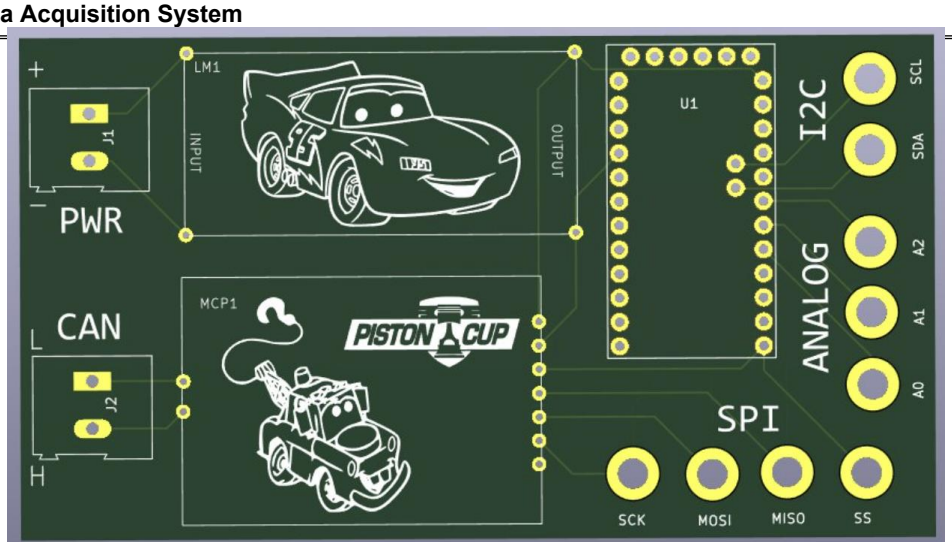


Figure 20. Gateway PCB version 1

C. Short Bios

Camille Gowda is a senior Electrical and Computer Engineering student at Vanderbilt University, with minors in Computer Science and Physics. Her primary areas of concentration are Embedded Computing & Cyber-Physical Systems and Signal, Image, Data & Medical Systems. On the Formula SAE Data Acquisition project, Camille focused on embedded systems development and integration. Her work emphasized practical system integration, signal handling, and software support for data acquisition. Outside of this project, Camille has experience in simulation and hardware design through her research in radiation effects on semiconductor devices and prior coursework in VLSI and embedded systems. She is interested in applying embedded and data-driven approaches to engineering systems.

Kyle Rosenberg is a senior Electrical and Computer Engineering student at Vanderbilt University, with minors in Computer Science and Engineering Management. His academic focus includes software systems, artificial intelligence, and data-driven engineering applications. On the Formula SAE Data Acquisition project, Kyle developed and integrated a real-time telemetry and visualization system, with an emphasis on scalable data pipelines and tools for analyzing vehicle performance data. Outside of this project, Kyle has experience in software engineering and AI development through internships, where he built AI-powered tools and worked on custom transformer models. He has also developed projects in machine learning, game development, and full-stack applications, including a Premier League analytics tool and a Unity-based typing game. Kyle is interested in applying software engineering and AI to real-world systems, particularly those involving real-time data processing and intelligent decision-making.

Jillian Hornby is a senior Electrical and Computer Engineering student at Vanderbilt University, with minors in Digital Fabrication and Computer Science. Her areas of concentration area Embedded Computer & Cyber-Physical Systems and Signal, Image, Data, and Medical Systems. On the Formula SAE Data Acquisition project, Jillian worked on embedded systems development and sensor testing and integration. Jillian's experience in embedded systems and data acquisition comes from previous research with the Vanderbilt Aerospace Design Lab and the Vanderbilt Photonics Lab. Outside of Vanderbilt, she has experience in materials science and software development through previous internships. Jillian is interested in applications within the intersection of materials science and electrical engineering including wearable technologies, flexible electronics, and semiconductor devices.

Javier Olmos is a senior Electrical and Computer Engineering student at Vanderbilt University with a minor in Engineering Management. His primary areas of focus include embedded systems and digital design, marked by an interest in hardware to software integration. On the Formula SAE Data Acquisition project, Javier worked on system architecture and embedded implementation, which involved integrating multiple sensor streams and communication protocols such as CAN and UART into a unified, real-time data system. Outside of this project, Javier has prior experience in embedded systems and digital hardware design through coursework and research projects, including a repurposed STM32-based commercial drone flight controller to demonstrate single-axis attitude control and designing a pipelined floating-point adder in Verilog. He also gained industry experience through two instrumentation internships at ExxonMobil, where he focused on improving safety and operational reliability in a chemical plant.

Diego Oliver is a senior Electrical and Computer Engineering student at Vanderbilt University with a minor in computer science, originally from Managua, Nicaragua. On the Formula SAE Data Acquisition project, Diego worked on the hardware and firmware development for the CAN pipeline, designing a custom PCB to host the various communication interfaces. Outside of Formula SAE, Diego has worked on a range of embedded and hardware projects through NASA Lunabotics, Vandy Amateur Radio, and various surgical robotics research labs on campus. He now focuses on creating projects at the intersection of technology and art, combining embedded systems, creative hardware, and interactive media to build experiences that feel both technically precise and deeply personal.

Elena Kim is a senior Electrical and Computer Engineering student at Vanderbilt University, with minors in Computer Science and Cinema and Media Arts. Her areas of concentration are Embedded Computing & Cyber-Physical Systems and Signal, Image, Data & Medical Systems. On the Formula SAE Data Acquisition project, Elena focused on front end development of the data visualization system. She applied Python Qt GUI frameworks and Matplotlib libraries in her work. Outside of this project, Elena has experience in AI engineering, LLM research, and user interface design. She is interested in the intersection of art, media, and technology.