

## [6. Software Design Description](#)

### [6.1. Introduction](#)

#### [6.1.1. System Objectives](#)

#### [6.1.2. Hardware, Software, and Human Interfaces](#)

- [Hardware Interfaces:](#)
- [Software Interfaces:](#)
- [Human Interfaces:](#)

### [6.2. Architecture Design](#)

#### [6.2.1. Major Software Components](#)

- [6.2.1.1. Database CSC:](#)
- [6.2.1.2. Input Observer CSC:](#)
- [6.2.1.3. Data Management CSC:](#)
- [6.2.1.4. Telemetry GUI CSC:](#)
- [6.2.1.5. Traction Control CSU:](#)
- [6.2.1.6. Virtual Differential CSU:](#)
- [6.2.1.7. Suspension Motor Controller CSC:](#)
- [6.2.1.8. Steering Wheel Unit Processor CSC:](#)
- [6.2.1.9. Safety Check CSC:](#)
- [6.2.1.10. Power Draw Monitor CSC:](#)
- [6.2.1.11. Graphics Interface CSC:](#)
- [6.2.1.12. Suspension Kinematics CSU:](#)

#### [6.2.2. Major Software Interactions](#)

- [Sensor Data Collection:](#)
- [Active Suspension, Steering, and Motor Control Communication:](#)
- [Wireless Steering Wheel Interface:](#)
- [Data Logging and Transmission to the Database:](#)

#### [6.2.3. Architectural Design Diagrams](#)

### [6.3. CSC and CSU Descriptions](#)

#### [6.3.1. Class Descriptions](#)

- [6.3.1.1. Detailed Class Description 1](#)
- [6.3.1.2. Detailed Class Description 2](#)
- [6.3.1.3. Detailed Class Description 3](#)
- [6.3.1.4. Detailed Class Description 4](#)
- [6.3.1.5. Detailed Class Description 5](#)
- [6.3.1.6. Detailed Class Description 6](#)
- [6.3.1.7. Detailed Class Description 7](#)

#### [6.3.2. Detailed Interface Descriptions](#)

- [6.3.2.4. Central Control Unit CSCI](#)

### 6.3.3. Detailed Data Structure Descriptions

#### 6.3.3.1 Database CSC:

#### 6.3.3.2 Input Observer CSC:

- This will have three floats for steering angle (positive is clockwise, negative is counterclockwise), throttle percentage, and brake percentage.

#### 6.3.3.3 Traction Control CSU:

#### 6.3.3.4 Virtual Differential CSU:

#### 6.3.3.5 Suspension Motor Controller CSC:

#### 6.3.3.6 Safety Check CSC:

#### 6.3.3.7 Power Draw Monitor CSC:

### 6.3.4. Detailed Design Diagrams

- FSAE\_More\_Detailed\_UML.png

### 6.4. Database Design and Description

#### 6.4.1. Database Design ER Diagram

#### 6.4.2. Database Access

#### 6.4.3. Database Security

## 6. Software Design Description

### 6.1. Introduction

This document presents the architecture and detailed design for the software of the FSAE Vehicle Control System project. The project aims to develop a comprehensive control system for a Formula SAE vehicle, encompassing data acquisition, processing, and real-time telemetry.

#### 6.1.1. System Objectives

The primary goal is to create a robust and efficient control system that enhances vehicle performance and safety. This includes developing modules for data management, input observation, telemetry, and various control units such as traction and suspension.

#### 6.1.2. Hardware, Software, and Human Interfaces

- Hardware Interfaces:
  - The project uses Raspberry Pi units to handle processing for various control systems, while Arduinos are used to gather data from sensors for inputs like steering and pedal positions. The system also includes brushless DC motors for suspension control, which help optimize the vehicle's handling. LCD displays are used in the cockpit to provide telemetry data to the driver.
- Software Interfaces:
  - The software runs on both a Windows laptop for the database and telemetry GUI and on the Raspberry Pi units in the car. The database, managed in PostgreSQL, stores telemetry data. The Telemetry GUI, developed in Go with the Fyne.io framework, allows users to view real-time data. Additionally, the system's core control functions are programmed in C++ to manage tasks like traction control and suspension adjustments.

- Human Interfaces:
  - The driver and team interact with the system through two primary interfaces. The telemetry GUI provides the race team with real-time performance and diagnostics data, allowing for strategic adjustments and troubleshooting. Meanwhile, the driver uses the steering wheel interface to receive immediate feedback on key metrics like speed, tire temperatures, and lap times, which is crucial for making quick decisions on the track.

## 6.2. Architecture Design

### 6.2.1. Major Software Components

#### 6.2.1.1. Database CSC:

- 6.2.1.1.1. Manages and stores telemetry data collected from the vehicle.

#### 6.2.1.2. Input Observer CSC:

- 6.2.1.2.1. Captures and processes input data from the vehicle's steering and pedals.

#### 6.2.1.3. Data Management CSC:

- 6.2.1.3.1. Controls data flow between various systems within the vehicle.

#### 6.2.1.4. Telemetry GUI CSC:

- 6.2.1.4.1. Provides a graphical interface for real-time data visualization.

#### 6.2.1.5. Traction Control CSU:

- 6.2.1.5.1. Manages wheel traction to prevent slippage.

#### 6.2.1.6. Virtual Differential CSU:

- 6.2.1.6.1. Controls power distribution between the vehicle's drive wheels.

#### 6.2.1.7. Suspension Motor Controller CSC:

- 6.2.1.7.1. Manages the vehicle's active suspension system.

#### 6.2.1.8. Steering Wheel Unit Processor CSC:

- 6.2.1.8.1. Processes driver inputs and displays data on the steering wheel interface.

#### 6.2.1.9. Safety Check CSC:

- 6.2.1.9.1. Monitors system health and triggers safety responses.

6.2.1.10. **Power Draw Monitor CSC:**

6.2.1.10.1. Tracks and monitors power consumption of various systems.

6.2.1.11. **Graphics Interface CSC:**

6.2.1.11.1. Displays crucial telemetry and system data to the driver.




6.2.1.12. **Suspension Kinematics CSU:**

6.2.1.12.1. Calculates optimal suspension dynamics based on driving conditions.

6.2.2. **Major Software Interactions**

- **Sensor Data Collection:**
  - Multiple sensors send analog data to the CCU through an I2C bus. The data collected includes inputs such as GPS, tire temperatures, and acceleration, all essential for vehicle control.
- **Active Suspension, Steering, and Motor Control Communication:**
  - The Active Suspension Controls, Steering/Pedal Unit, and Motor Control Units are directly connected to the CCU via cabled connections. This setup enables continuous, direct exchange of commands and data, which allows these control units to receive updates and send feedback to the CCU.
- **Wireless Steering Wheel Interface:**
  - The Steering Wheel Unit communicates with the CCU through a WiFi connection. This wireless link allows the driver's inputs from the steering wheel to be sent to the CCU, while telemetry and feedback data are transmitted back to the steering wheel display for real-time monitoring.
- **Data Logging and Transmission to the Database:**
  - The CCU sends processed telemetry data to an external database over WiFi. This data includes real-time metrics like tire pressure, engine metrics, and speed. The stored data can then be accessed by the Telemetry GUI for live viewing and further analysis by the race team.

### 6.2.3. Architectural Design Diagrams

-  FSAE\_Software\_UML\_Detailed\_Light.png
- <https://drive.google.com/file/d/1XJQdLfvUPEc-hL9Wth4m16wxI5Ti94fA/view?usp=sharing>
-  FSAE\_Software\_UML\_TopView.png
- <https://drive.google.com/file/d/1ZSpapb2ohz1bcl9hZ1t0WoXGupmw0adg/view?usp=sharing>
-  FSAE\_Software\_UML\_UseCase.png
- [https://drive.google.com/file/d/1I0SIbYij5DJVT3\\_ixCevE5P6Q\\_ZJQMkD/view?usp=sharing](https://drive.google.com/file/d/1I0SIbYij5DJVT3_ixCevE5P6Q_ZJQMkD/view?usp=sharing)

## 6.3. CSC and CSU Descriptions

### 6.3.1. Class Descriptions

#### 6.3.1.1. Data Management Class

Purpose: Acts as the main processing hub for data collection and management. The Data Management Class coordinates data flow across subsystems, ensuring real-time updates of car telemetry.

Fields:

- steeringAngle:float
- throttleInput:float
- brakeInput:float
- velocity:float
- tireAvgTemps:float[4]
- lapTime:float
- x\_acceleration:float
- y\_acceleration:float
- z\_acceleration:float
- activeSuspensionMotorSpeeds:float[4]
- activeSuspensionWheelToes:float[4]

Methods:

- + getSteeringWheelUnitData():map<String, float>
- + getTelemetryPacket():DataPacketStruct
- + getAccelerationData():float[3]
- + queryData():void

#### 6.3.1.2. Stability Control Class

Purpose: Manages driving systems such as ABS, Virtual Differential, and Suspension Kinematics.

Fields:

- float traction\_loss,
- float[2] abs\_throttle\_limiting,
- float[2] limited\_slip\_usage,
- float[4] suspensionKinematics

Methods:

- + detectLossOfTraction( motorPowerDraw:tuple<float, float>):  
tuple<float,float>
- + virtualDifferentialCalculator( steeringAngle:float,  
acceleration:float[3],velocity:float ): tuple<float, float>
- + activeSuspensionKinematicCalculator( acceleration:float[3] ):float[4]

#### 6.3.1.3. Safety Check Class

Purpose: Makes sure data flow is concurrent and uninterrupted. Detects if data flow has been interrupted.

Methods:

- + dataFlowCheck(sensorData: SensorPacket):void

#### 6.3.1.4. Power Draw Monitor Class

Purpose: Monitors power consumption of the main motors of the car as well as monitoring the overall power usage of the car.

Fields:

- totalPowerDraw: float
- activeSuspensionPowerDraw: float[4]
- motorPowerDraw: tuple<float, float>

Methods:

- + getTotalPowerDraw():float
- + getMotorPowerDraw():tuple<float, float>
- + getActiveSuspensionPowerDraw:float[4]

#### 6.3.1.5. TelemetryGUI Class

Purpose: Serves as the primary interface for displaying real-time and historical telemetry data, alerting the user to issues, and allowing for data visualization. It provides customizable data displays and supports real-time visualizations for immediate feedback.

Fields:

- window:FyneWindow
- currentData:Map

Methods

- + viewAllData():void

- + viewDayData( time:Date ):void
- + viewCurrentData():void
- + sortBy( metric:MetricEnum ):void
- + displayTireInfo():void
- + displayPowerInfo():void
- + displayPositionInfo():void
- + displayVehicleStabilityCSCInfo():void
- + startApp():void

#### 6.3.1.6. Telemetry Database Query Class

Purpose: The telemetry database query class serves to act as an access point where you can query the database for telemetry data.

Methods:

- + queryTable( time:Date ):Map
- + getTable():Map

#### 6.3.1.7. Telemetry Database Insertion Class

Purpose: The telemetry database insertion class is meant to safely receive telemetry data packets from the car and transfer that data into the database

Methods:

- + insertEntry( entry:DataPacketStruct ):void

#### 6.3.1.8. Motor Speed Controller Class

Purpose: Dynamically adjusts suspension characteristics, such as stiffness, to improve handling and stability based on real-time conditions and driver inputs.

Fields:

- motorPowerUsage:float
- motorSpeed:float
- wheelToe:float

Methods:

- + calculateMotorOutput( motorRotation:float, suspensionStrokeAcceleration:float, electroActuatorOutput:float, electroActuatorInternalInertiaForce:float):float
- + motorControllerAdjuster ( calculatedMotorOutput:float ):void
- + getMotorPowerUsage():float
- + getMotorSpeed():float
- + getWheelToe():float



#### 6.3.1.9. Input Observer Class

Purpose: Captures and processes driver inputs from the steering wheel and pedals, converting them into digital data for real-time vehicle control.

Fields:

- steeringAngle:float
- throttleInput:float
- brakeInput:float

Methods:

- + checkForFaults():void
- + getSteeringAngle():float
- + getThrottleInput():float
- + getBrakeInput():float

#### 6.3.1.10. Steering Wheel Display Class

Purpose: Displays relevant driver information on a small display located on the steering wheel of the car.

Fields:

- velocity:float
- tyreAvgTemps:float[4]
- throttleInput:float
- brakeInput:float
- lapTime:float
- window:GraphicsWindow

Methods:

- + startDisplay():void
- + displayData( velocity:float, tyreAvgTemps:float[4], throttleInput:float, brakeInput:float, lapTime:float ):void

### 6.3.2. Detailed Interface Descriptions

#### 6.3.2.1. The Active Suspension CSCI

6.3.2.1.1. The Suspension Motor Controller talks to a level sensor to receive information on the vehicle's ride height. The Suspension Motor Controller also talks to data management to get acceleration data.

6.3.2.1.2. Additionally, the Suspension Motor Controller talks to the Central Control Unit (CCU) to receive broader system data for integrated suspension control.

#### 6.3.2.2. The Input Observer

- 6.3.2.2.1. reads input and pressure data from the throttle and brake pedals and converts analog steering signals to digital data. It then sends processed steering and pedal data to the Central Control Unit (CCU) and monitors for any anomalies to ensure safe operation.
- 6.3.2.3. The Graphics Interface
  - 6.3.2.3.1. receives data from a Raspberry Pi and displays it to the driver. The Steering Wheel Unit Processor receives data from the Central Control Unit (CCU), sends driver inputs back to the CCU, condenses data into an interpretable graphic, and sends this graphic to the Graphics Interface for display.
- 6.3.2.4. Central Control Unit CSCI
  - 6.3.2.4.1. The Data Management CSC in the Central Control Unit manages data communication across all CSCs to maintain data integrity and timeliness. It receives data from an I2C Bus, including sensor, GPS, thermal camera, gyroscopic, and acceleration data.
  - 6.3.2.4.2. The Vehicle Stability CSC receives data from the accelerometer and electric drive motors to monitor wheel slippage and adjust vehicle stability. It limits throttle upon detecting traction loss, calculates the power distribution to electric motors for a limited-slip differential effect, and adjusts active suspension stiffness and power as needed. It also follows commands to modify behaviors and settings.
  - 6.3.2.4.3. The Safety Check CSC ensures a constant data flow from each sensor and will cut power to the motors if any fault connections are detected.
  - 6.3.2.4.4. The Power Draw Monitor CSC estimates overall vehicle power usage, tracks power consumption of each individual component, and sends telemetry data on component power usage to the database.
- 6.3.2.5. Telemetry App CSCI
  - 6.3.2.5.1. The Telemetry GUI CSC displays data with under 100ms latency for real-time visualization, supporting simultaneous display of up to 10 telemetry metrics to provide critical information to the driver and pit crew without lag.
  - 6.3.2.5.2. The Database CSC handles wireless data transmission with a latency of under 50ms, ensuring a real-time connection with the Telemetry GUI for timely telemetry updates.
  - 6.3.2.5.3. The Database CSC stores telemetry data at a rate of 10 samples per second per metric, providing high-resolution data for detailed post-race analysis.

#### 6.3.2.6. Steering/Pedal Unit CSCI

- 6.3.2.6.1. The Input Observer CSC in the Steering/Pedal Unit CSCI consistently reports steering and pedal inputs to the Central Control Unit (CCU) to ensure accurate and timely input data for vehicle control.
- 6.3.2.6.2. consistently displays all pertinent information to the driver through an intuitive and navigable UI. The Steering Wheel Unit Processor CSC maintains a stable Wi-Fi connection with the Central Control Unit (CCU), facilitating reliable communication, and consistently generates display data for the Graphics Interface.

#### 6.3.2.7. Central Control Unit CSCI

- 6.3.2.7.1. The Data Management CSC receives sensor data within 10ms to ensure rapid responsiveness, allowing the car to quickly react to changing situations.
- 6.3.2.7.2. The Vehicle Stability CSC detects and calculates traction loss within 10ms, enabling the car to react swiftly to minimize the impact of traction loss.
- 6.3.2.7.3. The Vehicle Stability CSC calculates motor power distribution within 50ms to effectively replicate the performance of a mechanical limited-slip differential.
- 6.3.2.7.4. The Vehicle Stability CSC calculates PID within 50ms to ensure the suspension responds quickly, maintaining tire contact with the ground for optimal stability.
- 6.3.2.7.5. The Safety Check CSC ensures all signals are within the expected range and free from errors or unexpected latency, preventing potential malfunctions with serious physical consequences.
- 6.3.2.7.6. The Power Draw Monitor CSC consistently reports the vehicle's total power usage, ensuring accurate power tracking to prevent issues across other systems.

### 6.3.3. Detailed Data Structure Descriptions

#### 6.3.3.1 Database CSC:

- This will be a table consisting of all pertinent telemetry data.

#### 6.3.3.2 Input Observer CSC:

- This will have three floats for steering angle (positive is clockwise, negative is counterclockwise), throttle percentage, and brake percentage.

#### 6.3.3.3 Traction Control CSU:

- This will have floats that measure all data that is referenced when determining if a tire is slipping.

#### 6.3.3.4 Virtual Differential CSU:

- This will have two floats that record the percentage of power going to the rear left and rear right wheel.

#### 6.3.3.5 Suspension Motor Controller CSC:

- This will have a float that will store how stiff the suspension of the vehicle is.


#### 6.3.3.6 Safety Check CSC:

- This will record multiple floats for the temperatures of vital systems. It will also have floats that ensure voltage is within expected range. Finally, it will have a boolean that is True if all data is within preset boundaries.

#### 6.3.3.7 Power Draw Monitor CSC:

- This will be multiple floats that monitor how much power all systems in the vehicle are using.

### 6.3.4. Detailed Design Diagrams

-  FSAE\_More\_Detailed\_UML.png
- <https://drive.google.com/file/d/1qaQqwqR6ZDseQPTMPin4T5ZZkbvT8Ng1/view?usp=sharing>

## 6.4. Database Design and Description

### 6.4.1. Database Design ER Diagram

-  FSAE Software Database UML Diagram.png

- [https://drive.google.com/file/d/1iLwwc9j8uE6WPFIO2mGloB-vx-0pm5\\_M/view?usp=sharing](https://drive.google.com/file/d/1iLwwc9j8uE6WPFIO2mGloB-vx-0pm5_M/view?usp=sharing)

#### 6.4.2. Database Access

- The database will be hosted in Postgresql on a windows operating system. The database will be connected to the Telemetry GUI on local ports on the same device. This is meant to reduce the time it takes to access and insert into the database since this is meant for usage by a small team. Accessing the database will involve executing SQL calls onto the Postgresql using a API Script in Go. This way the Telemetry GUI can easily include the API Script to query from the database. Inserting into the database will be a separate script in Go that will have to receive telemetry data packets and insert them into the database as a root user.
- The Go scripts that will access the database will be using the Go driver PGX. This is a popular open-source driver for accessing Postgresql databases with support for Postgresql's custom data types that are separate from SQL. The driver supports connection pools as well allowing for continuous connection with the database. This is planned so that we can have live telemetry reports to monitor the car's status.
  - PGX Repo: [github.com/jackc/pgx/v5](https://github.com/jackc/pgx/v5)

#### 6.4.3. Database Security

- The database will be solely used within a trusted team and thus security is not a huge concern for our database. The machine that will run the database will also be running the Telemetry GUI and thus there is no need for security checks as the database is not connected to the internet. There will be safety checks when removing entries from the database but that will be it.
- For receiving telemetryPackets there will be some security but a simple authentication hashCode should be enough as the Go Script to receive telemetryPackets only receives information from one source (the car).