"We pledge our honor that we have abided by the Stevens Honor System"

CS 347 Project Document

Alset IOT Hugs the Lanes

Section B, Team 14: The Seth Rogan Fan Club
04/11/2022

Jack Chen

Sev Napora

Jessica Noel

Sophia Roper

Table of Contents

Section 1: Introduction

Section 1.1: Problem Statement

Currently, there are 38,000 fatalities caused by automobile accidents per year in the United States (ASIRT). In addition, road crashes are the leading cause of death for people aged 1-54. This project will address this problem by presenting a mission critical real-time embedded system. This system relies on the Internet of Things as the most advanced architecture for the products presented in this document.

Section 1.2: Project Goals

Our goal is to assist the Alset “Hug the Lanes” project in its pursuit of making cars safer, less costly, and more efficient by leveraging Internet of Things technology. We will do this by implementing autonomous driving software into a line of fully electric, high performance vehicles. The result will be a driving experience that is more safe, efficient, and sustainable than the gas powered vehicles on the road today. The nature of the project is a mission critical real-time embedded system.

Section 1.3: Project Features and their Importance

The features we will be implementing will contribute to the safety, efficiency, and power of the driving experience, as well as the comfort and functionality of being a passenger.

The high level features we will be using include edge devices that can capture data from the car and environment and an analytic engine on board to process information and make decisions that are passed on to the Car Control System. Additionally, the driver will be able to enter commands, receive status reports, and download the latest rules and updates for operation from the Cloud into the IoT engine.

These features are all important because they contribute to the safety of both the driver and passengers. We want to ensure that our customers receive an affordable project that does not compromise on safety or efficiency. The features implemented in our project will reflect the values held by our customers.

Listed below are the technical specifications of our vehicle and autonomous driving software:

Vehicle

- Dual motors
- Battery
- Built in WiFi
- GPS

Sensors and Cameras

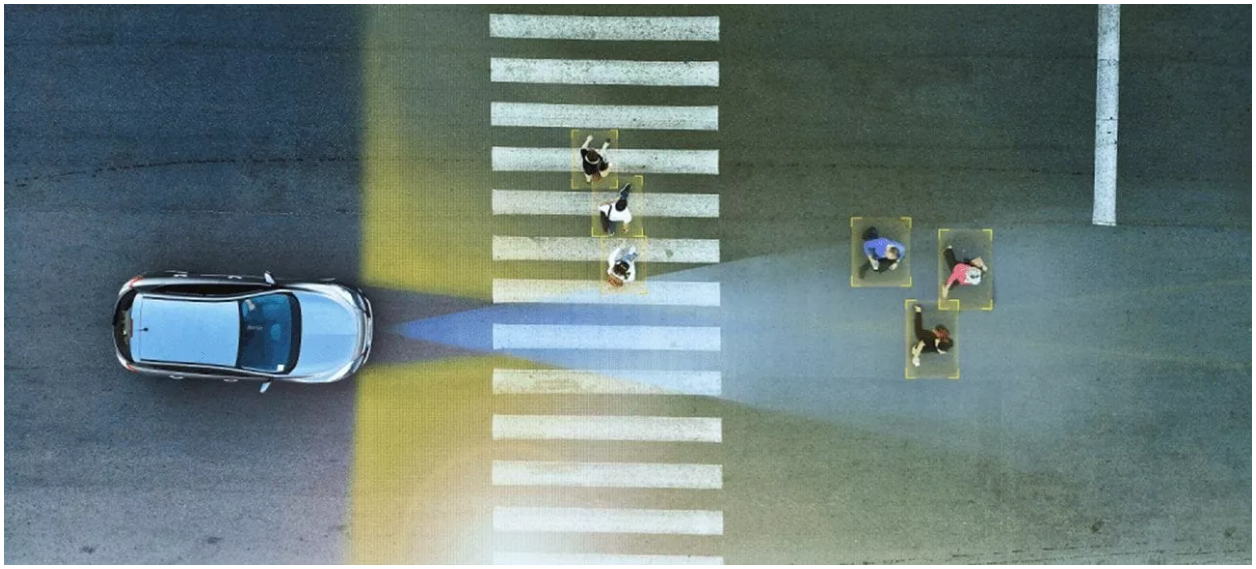
- Light Detection and Ranging (LIDAR) sensor
- Radar/Proximity sensors
- Ultrasound sensors
- Light sensors
- Color detection (cameras)
- Rear view camera
- Side cameras
- Fuel tank pressure sensors
- Voltage Sensor
- TPMS sensor (Tire pressure)
- Throttle Position Sensor
- Coolant Temperature Sensor
- Wheel Speed Sensor
- For cruise control
    - RPM sensor
    - Brake pedal sensor that signals to stop ACC (adaptive cruise control)

Interface

- Touch screen display
- Holographic HUD on windshield
- 15 teraflops of processing power
- Voice recognition and control
- Consistent software updates

Using these technologies, we will be able to achieve the following features:

- Fully electric
- Full autonomous driving in daylight or night time
- Assisted cruise control
- Automatic parallel and standard parking
- Collision prediction
- Environment identification
- Traction control
- Blind spot detection
- Automatic braking
- Supercharge capability
- 1000+ horsepower
- 400+ mile range
- 72 hours of idle time (including heat, internet, and light)



Section 1.4: Our Commitment to Quality

Our team is committed to manufacturing the software for Alset in the most appropriate and efficient manner. After speaking to the stakeholders we will create commitments that will go through a software development process that outlines the project's scope, timeline, risks and other formal requirements. Our team constantly seeks to find the best methods to deliver quality products which is why we provide our stakeholders with quality assessments and help product owners understand the risks.

Section 1.5: Development Process

This project will be incorporating the Unified Process Model, which includes five stages - inception, elaboration, construction, transition, and production. After communicating with stakeholders and planning potential ideas that address their concerns, the plan evolves and we start modeling, the elaboration phase. In this phase, we begin to develop early versions of our software, which will be modified to better incorporate details provided by stakeholders or improved processes. After the modeling process has ceased, we will move on to the construction process, where we develop the actual software that will be deployed to the stakeholders. After constructing a version of the software, we transition into the transition stage, where the software is fully deployed and we are able to receive feedback and make improvements.

Section 1.6: Team Members

This project will be managed by four undergraduate students from the Stevens Institute of Technology. All students have multiple years of coding experience.

Section 1.7: Time Frame

The development of this project began on February 1, 2022 and has an expected completion date of May 1, 2022.

Section 2: Functional Architecture

Section 2.1: Overview

The Importance of Architecture

For any complex system, there needs to exist an organized system to manage that complexity. Any form of self-driving vehicle would be extremely complex so this type of architecture would be crucial in this case. This architecture should further be standardized across the industry to speed up development and grow the cumulative safety knowledge of the industry at large.

State of Literature

Given that autonomous vehicles are a relatively new technology, literature on the architecture design aspects of Autonomous Vehicles (AVs) is hard to come by. This documentation will focus on research and experimentation of autonomous vehicle design, e.g. algorithms for object detection and trajectory planning, as well as system components and our competition’s

implementation; all of which have been lacking in existing AV design literature. The design, planning, components, and implementation will also cover Advanced Driver-Assistance Systems (ADAS), a key underlying technology seen in the emergence of AVs.

Discussion Organization

This document will be organizing the discussion on functional architecture around six different topics: three of which fall under *functional architecture*, two of which fall under *software architecture*, and the last is known as "computation platform," which is outside the scope of both. "Functional Architecture" is defined as "the logical decomposition of the system into components and subcomponents, as well as the data-flows between them" (refer to Fig 1).

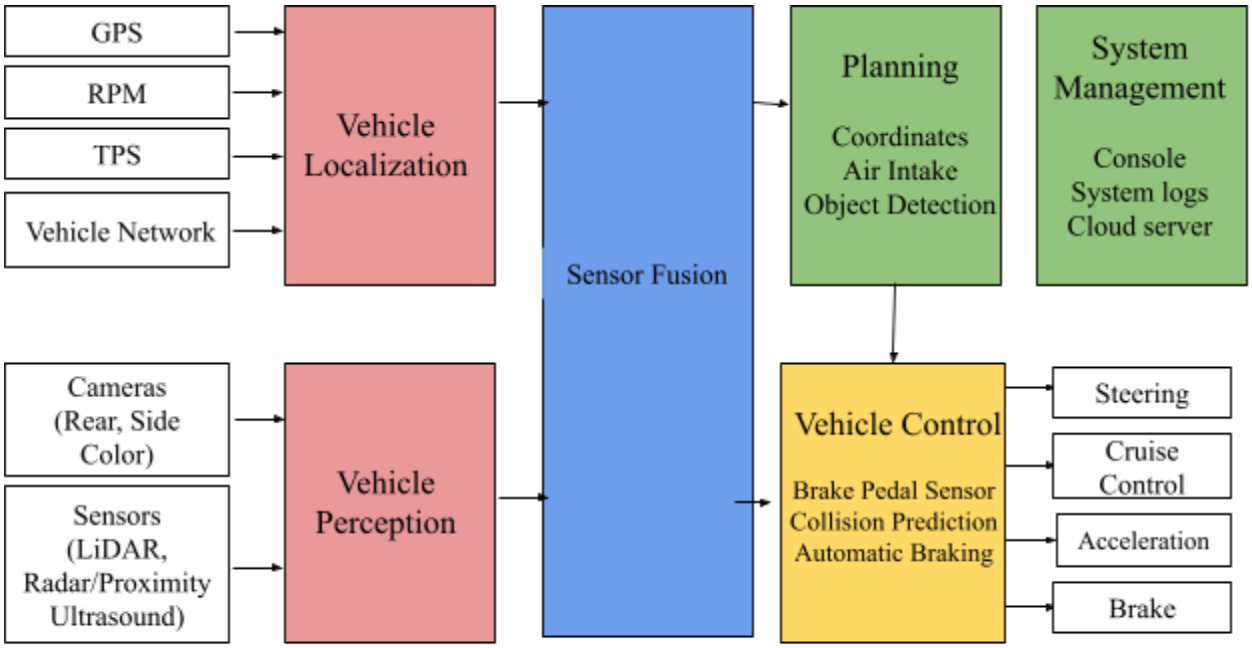


Fig. 1 Functional Architecture

Whereas "Software Architecture" defines the responsibilities and relationships between the layout of the infrastructure components. The issues that fall under *functional architecture* are "functional distribution," "data flow," and "fault management." "Software platform" and "data model/interprocess communication" fall under *software architecture.*

Analysis Framework

In order to analyze the quality of a piece of architecture design, we will be following six "quality attributes" which build our analysis framework. These attributes are from the book *Software Architecture in Practice* by Len Bass, Paul Clements and Rick Kazman at the Software Engineering Institute, and are as follows: Availability, Interoperability, Modifiability, Performance, Security, and Testability.

Section 2.2: Functional Distribution

Identification of Functional Components

Our working definition of functional architecture deals with three high-level components: perception, planning, and control. Perception takes in environmental data from cameras, GPS, LiDAR sensors, and integrated drive unit (IDU). Its submodules may include ego-vehicle localization, object detection, object tracking, semantic understanding, and world module. Planning makes decisions based on the overall goal and the environmental data. Its submodules may include mission planning, behavior planning, and trajectory generation. Control deals with executing the plan. Its submodules would include acceleration, braking, steering, and collision avoidance/emergency braking.

Separated Safety System

As seen in existing component designs [1,2], there has been an increase in the popularity of placing security into a specific and emphasized component rather than treating it as another element of other components. Separating safety from the traditional control module helps ensure that no component-specific restrictions have a negative impact on developing the safest system possible.

World Model

In order to properly determine the functionality and operations of an autonomous car, it is imperative to first create a strong world model of the whole system. Within the world model, external factors will be broken down into multiple categories to help properly define the bounds and limitations of the model. These sub-categories would be contained within a static model and include passive and active model components. Active components represent pieces that rely heavily on human interaction and feature higher variability (i.e. pedestrians, other vehicles, etc.); passive components on the other hand are more permanent fixtures within the environment featuring less human interaction and more predictably (i.e. street lights, railways, bodies of water, etc.). The world model is an integral part of the system as a whole because it provides crucial information to the system and enhances its decision making.

Driver Assistance

The IoT Hugs the Lanes vehicle is a fully autonomous vehicle with numerous levels of autonomy. The lower levels of autonomy are the Advanced Driver Assistance Systems, which helps the driver with specific tasks, but the vehicle is still ultimately manually controlled by the driver. These tasks include parallel parking, lane tracking, and automatic braking. The sensors listed in the architecture and described later in this document all work under the ADAS.

Self Driving

The upper levels of autonomy are full autonomous driving. This system seeks to replace the human driver, which will ultimately increase safety and efficiency in traffic. While this system also implements localization and perception sensors, there is a need for more advanced machine

intelligence and IoT technologies for the system to function. The architecture for this system is also included in this document.

System Administration

The aforementioned self-driving architecture's requirement, advanced machine intelligence, is often provided by neural network algorithms and information integration. This requires a high level of system administration provided by IoT Hugs the Lanes. System Administration and Management is described in detail further below.

Section 2.3: Mapping Functional Components to Computing Units

Centralization vs Distributed

Many in the industry disagree on whether these components should be implemented on a centralized or distributed architecture. Advantages of a centralized architecture include minimal network complexity, minimal delay in communication, greater flexibility to change and upgrade the system's algorithms, and more modular compatibility with off-the-shelf components. Advantages of a distributed architecture include more space to increase computational complexity, greater fault tolerance, cost and weight effectiveness with computing units being able to be placed closer to the sensors, and parallel development and testing.

Decoupling the Vehicle Platform

As briefly mentioned in the State of Literature section, ADAS systems have become more prominent in traditional vehicles and are a central component of AVs. ADAS systems are "primarily vision based (cameras), but light detection and ranging (LiDAR), radio detection and ranging (Radar), and other advanced-sensing technologies are also becoming popular" [3]. Many of these systems, including "lane keeping, automatic braking, and adaptive cruise control" are commonly seen in traditional vehicles [3]. Because ADAS systems are so fundamental in the creation of AVs, there is an overlap between the two, and the majority of software developed with AVs in mind will also be applicable to more un-autonomous Assistant Driving vehicles.

Localization Sensors

The IoT Hugs The Lanes vehicle is equipped with a number of sensors that help locate the car's position in real time. These sensors are Global Positioning System (GPS), Revolutions per Minute (RPM), Throttle Position Sensor (TPS), which all communicate through IoT with the Vehicle Network. These sensors are critical to the function of the Self-Driving architecture of the vehicle. The GPS system pairs accurate GPS updates (10 hz frequency) with inaccurate inertia measurements (200 hz frequency) to provide localization. This sensor is accurate most of the time, with the exception of obstructions such as tunnels.

Perception Sensors

The separation of perception sensors and localization sensors was done to create an instantiation of the "separated safety pattern." This allows for safer technician access and isolation of each component, as IoT Hugs the Lanes is focused on high quality system management. The vehicle is equipped with the following perception sensors: Light Detection and Ranging (LiDAR), Vision Cameras, and Ultrasound (for Radar and Positioning). These sensors provide information regarding distance between the vehicle and other objects with high accuracy. The LiDAR system offers high accuracy and can localize against high-definition maps, but is susceptible to noise caused by certain weather conditions. The second level of defense against obstacle collision is the vision cameras, placed strategically around the vehicle. The cameras provide color information, but perform worse in poor light conditions. The final sensor, Ultrasound, has superior performance in short distance detection but performs poorly in object detection.

Sensor Fusion

The sensor fusion is a cognitive intelligence module that computes the ideal allocation of information to different actuation modules. The data from localization sensors (vehicle positioning) and perception sensors (obstacle detection) are sent to the sensor fusion first, where it is then sent to the correct architecture module. This architecture design allows for greater software modification.

Planning

The planning module receives data allocated from the sensor fusion such as Coordinates and Air Intake to determine proper vehicle action. The component then combines the top-level goal and perceived environment to make decisions about behavior and trajectories. The submodules of Planning include mission planning (route from origin to destination), and semantic understanding (e.g. what the vehicle does in a school zone).

Vehicle Control

The control module refers to the ability for the vehicle to actually execute what is planned by the previous module. It does so by specifying the strength and direction of acceleration, and implementing the vehicle actuators: longitudinal control (propulsion and braking), lateral control (steering), and reactive control (object detection and emergency braking).

Section 2.4: System Management

Logs

All actions the AV takes during operation will be logged and stored locally in the system in addition to a cloud management system. These logs will allow for error handling and testing of components. Every action taken by a module of the function architecture is logged in System

Management. These logs are then converted into a human readable format, allowing for easy testing of individual modules. The methods of testing the system are as follows: “injecting faults,” “probing states,” and “running in the sandbox.”

Technician Access

All logs and system management files are accessible by qualified IoT Hugs the Lanes technicians. Also available to technicians are communications with the IoT cloud server, through the operating system built into the vehicle and accessible via the technician console.

Software Updates

System components and software are constantly improved, and updates are periodically sent to all IoT Hugs the Lanes vehicles. These updates will only be deployed when the vehicle is inactive. Updates are universally sent to all models, for consistency in vehicle behavior and ease of access. As a result, all logs will be in the same format, testing and maintenance can be performed by any technician, and users will have the highest performing software possible.

Section 3: Requirements

Section 3.1: Functional Requirements

This section details the functional requirements for the requested features (cruise control, assisted driving, self driving, etc). Essentially follows the functional architecture.

Functional Requirement Guideline:
Description
Preconditions
Numbered Requirements
Postconditions

Section 3.1.1: Setting the Cruise Control

Cruise control is a driver assisting feature that allows the vehicle to coast at a certain speed.

3.1.1.1 Preconditions:

3.1.1.1.1: The car must be in drive
3.1.1.1.2: The vehicle must be at a minimum speed of s >0
3.1.1.1.3: The speed sensor must confirm the current speed of the vehicle

3.1.1.2 Function:

3.1.1.2.1: Driver turns cruise control on by pressing the cruise control button, display indicates that the vehicle is in cruise control
3.1.1.2.1.1: On the drivers digital dashboard the words “CRUISE ON” will be displayed above the speedometer indicating that cruise control is active.
3.1.1.2.2: Speed data will enter sensor fusion
3.1.1.2.3: Sensor fusion sends speed request to IOT HTL for cruise control
3.1.1.2.4: IOT HTL continually sends speed requests to the Vehicle Control System
3.1.1.2.5: IOT HTL logs the cruise control event in System Management

3.1.1.3 Postconditions:

3.1.1.3.1: The vehicle continues to cruise at the selected speed, until brakes are applied

Section 3.1.2: Disengaging Cruise Control

3.1.2.1 Preconditions:

3.1.2.1.1: Vehicle has cruise control engaged.

3.1.2.2 Function:

3.1.2.2.1: Driver applies braking, brake sensor sends signal to IOT HTL sensor fusion
3.1.2.2.2: Sensor fusion sends request to IOT HTL
3.1.2.2.3: Vehicle control system turns off cruise control
3.1.2.2.4: Display turns off

3.1.2.3 Postconditions:

3.1.2.3.1: The cruise control is no longer active and driver is in control of speed

Section 3.1.3: Automated Parking

Automated parking is a driver assisting feature that allows the vehicle to park without any driver intervention.

3.1.3.1 Preconditions:

3.1.3.1.1: Vehicle is within ten feet of the parking space

3.1.3.2 Function:

3.1.3.2.1: Driver initiates automated parking by pressing the automated parking button, display indicates that automated parking is engaged
3.1.3.2.2: Data from cameras and sensors enter sension fusion
3.1.3.2.3: Sensor fusion sends parking request to IOT HTL
3.1.3.2.4: IOT HTL continuously sends parking requests to vehicle control system
3.1.3.2.5: IOT HTL logs the automated parking event in System Management

3.1.3.3 Postconditions:

3.1.3.3.1: Vehicle is safely parked in a space

Section 3.1.4: Collision Prediction

Collision prediction is a driver assisting feature that allows the vehicle to intervene in the case of an imminent collision when the driver is incapacitated or has a delayed response.

3.1.4.1 Preconditions:

3.1.4.1.1: Vehicle is in motion
3.1.4.1.2: Vehicle is approaching an obstacle or being approached by an obstacle
3.1.4.1.3: Driver has not reacted to the obstacle within a variable time (calculated internally using sensor data)
3.1.4.1.4: Vehicle has space to avoid the obstacle

3.1.4.2 Function:

3.1.4.2.1: Sensors send distances to nearest obstacle and current speed of the vehicle to sensor fusion
3.1.4.2.2: Sensor fusion calculates the time until the collision
3.1.4.2.3: Sensor fusion reports imminent collision to planning if the calculated time falls within the variable collision prediction time
3.1.4.2.4: Planning sends a trajectory change request and data to IOT HTL
3.1.4.2.5: IOT HTL continuously sends a trajectory change request to the vehicle control system
3.1.4.2.6: The vehicle control system applies an algorithm to determine whether the vehicle should slow down, stop, or steer in a different direction
3.1.4.2.7: IOT HTL logs the collision prevention event in System Management

3.1.4.3 Postconditions:
3.1.4.3.1: Vehicle has avoided the obstacle to the best of its ability

Section 3.1.5: Environment Identification

Environment identification is a feature that implements a vehicle's cameras and sensors to identify its surroundings, including other vehicles, pedestrians, and/or obstacles.

3.1.5.1 Preconditions:
3.1.5.1.1: N/A - environment identification is continuous

3.1.5.2 Function:
Section 3.1.5.2.1: All sensors continuously send data to sensor fusion
Section 3.1.5.2.2: Other functional requirements that utilize environment identification incorporate this data in their processes

3.1.5.3 Postconditions:
Section 3.1.5.3.1: N/S - environment identification is continuous.

Section 3.1.6: Traction Control

Traction control is a safety feature that prevents the vehicle’s wheels from losing traction on roads, especially during hazardous weather conditions.

3.1.6.1 Preconditions:
3.1.6.1.1: Sensor fusion senses a discrepancy between the vehicle’s speed and its wheels’ rotational speed

3.1.6.2 Function:

3.1.6.2.1: Sensor fusion determines if the vehicle speed matches the wheel rotational speed

3.1.6.2.2: If they do not match within a specific interval, sensor fusion sends a traction control request to IOT HTL

3.1.6.2.3: IOT HTL continuously sends continuously sends traction control requests to the vehicle control system, display indicates that traction control has been implemented

3.1.6.2.4: IOT HTL logs the traction control event in System Management

3.1.6.3 Postconditions:

3.1.6.3.1: The vehicle's wheels temporarily stop spinning until it regains traction

Section 3.1.7: Blind Spot Detection

Blind spot detection is a safety feature that monitors a vehicle's blind spot areas and alerts drivers if there are any potential hazards in them.

3.1.7.1 Preconditions:

3.1.7.1.1: Vehicle detects an approaching vehicle near its blind spot

3.1.7.2 Function:

3.1.7.2.1: Data from blind spot sensors are sent to sensor fusion

3.1.7.2.2: Sensor fusion determines if there is an obstacle

3.1.7.2.3: Sensor fusion sends blind spot detection request to IOT HTL

3.1.7.2.4: IOT HTL continuously sends an alert to the vehicle control system

3.1.7.2.5: IOT HTL sends an audible and tactile alert to the driver

3.1.7.2.6: IOT HTL logs the event in System Management

3.1.7.3 Postconditions:

3.1.7.3.1: Driver is alerted to the potential hazard

Section 3.1.8: Lane Mitigation and Tracking

Lane mitigation and tracking is a driver assisting feature that alerts the driver if they are either drifting out of their lane or off of the road.

3.1.8.1 Preconditions:

3.1.8.1.1: Vehicle is either beginning to drift out of its lane or off its road

3.1.8.2 Function:

3.1.8.2.1: Vehicle coordinate data is sent to sensor fusion
3.1.8.2.2: Sensor fusion determines if the vehicle is drifting
3.1.8.2.3: Sensor fusion sends late mitigation request to IOT HTL
3.1.8.2.4: IOT HTL continuously sends an alert to the vehicle control system
3.1.8.2.5: IOT HTL sends an audible, visible, and tactile alert to the driver
3.1.8.2.6: If the driver does not fix the vehicle position, the vehicle control system makes minor rectifications
3.1.8.2.6: IOT HTL logs the event in System Management

3.1.8.3 Postconditions:
3.1.8.3.: Driver is alerted to their drifting

Section 3.1.9: Emergency Braking

Emergency braking is a driver assisting feature that allows the vehicle to intervene if it is approaching an obstacle.

3.1.9.1 Preconditions:
3.1.9.1.1: Vehicle is in motion
3.1.9.1.2: Vehicle is approaching an obstacle
3.1.9.1.3: Driver has not reacted to the obstacle within a variable time (calculated internally using sensor data)

3.1.9.2 Function:
3.1.9.2.1: Sensors send distances to nearest obstacle and current speed of the vehicle to sensor fusion
3.1.9.2.2: Sensor fusion calculates the time until the vehicle collides with the obstacle
3.1.9.2.3: Sensor fusion reports imminent collision to planning if the calculated time falls within the variable collision prediction time
3.1.9.2.4: Planning sends a stop request and data to IOT HTL
3.1.9.2.5: IOT HTL continuously sends a stop request to the vehicle control system
3.1.9.2.6: IOT HTL logs the event in System Management

3.1.9.3 Postconditions:
3.1.9.3.1: Vehicle has stopped before reaching the obstacle

Section 3.1.10: Traffic Sign Recognition

Traffic sign recognition is a feature that allows the vehicle to recognize the traffic signs on its road.

3.1.10.1 Preconditions:

3.1.10.1.1: Vehicle is in motion

3.1.10.1.2: Vehicle is on a road with traffic signs

3.1.10.2 Function:

3.1.10.2.1: Sensors send data collected by front-facing cameras and GPS coordinates to sensor fusion

3.1.10.2.2: Sensor fusion determines if there are relevant signs to display

3.1.10.2.3: Sensor fusion sends a traffic sign recognition request to IOT HTL

3.1.10.2.4: IOT HTL sends the traffic sign information to the driver's digital dashboard

3.1.10.3 Postconditions:

3.1.10.3.1: Vehicle displays the traffic sign to the driver

Section 3.2: Non-Functional Requirements

NOTE: All tests will be performed by a tester car. All tester cars have documented diagnostic reports/sensor data that is sent to company neural networks to train the AI (car).
All testers will have a valid driver's license for at least 5 years with no prior accidents, there will be an equal number of men and women as well at least five testers from each age group starting at the age 20. (age group is divided by the tens. I.e 20-30, 30-40, 40-50…)

Section 3.2.1: Performance

3.2.1.1: Cruise control engages/disengage shall not take more than 100 nanoseconds

3.2.1.1.1: This test will be scripted. The test steps are as follows:

3.2.1.1.1.1: The tester will engage cruise control and simultaneously the timer will be triggered.

3.2.1.1.1.2: The tester shall wait until the sensor is active, showing that cruise control has been successfully engaged.

3.2.1.1.1.3: The tester will confirm or deny that the time to successfully engage cruise control less than 100 nanoseconds

3.2.1.2: IoT HtL shall be able to process 320 trillion requests / second

3.2.1.2.1: This test will be scripted. The test steps are as follows:

3.2.1.2.1.1: The tester will be given the amount of requests needed per activity. The activities to be performed by the tester are: engage in cruise control, break to disengage cruise control, passively have nearby object detection sensors activated.

Note: These activities will have needed requests equal to or greater than 320 trillion.

3.2.1.2.1.2: The tester will engage cruise control, break to disengage cruise control while passively having nearby object detection sensors active.

3.2.1.2.1.3: The tester will confirm that the activities from step 3.2.1.2.1.2 were able to be processed within 1 second.

Section 3.2.2: Reliability

3.2.2.1 Cruise control shall not fail more than once per 300,000 hours of operation

3.2.2.1.1: This test will be scripted. The test steps are as follows:

3.2.2.1.1.1: One hundred testers shall operate the car for 3,000 total miles in a State Approved Autonomous Driving Location.

3.2.2.1.1.1.1: Tester shall assert that the car does not go more or less than 60 miles per hour when cruise control is activated on a 60mph speed limit highway.

3.2.2.1.1.1.2: Tester shall assert that the car successfully engages and disengages cruise control 100 times.

3.2.2.1.1.1.3: Tester shall assert that the car is able to operate 3,000 miles without failing the above steps.

3.2.2.2 IoT HtL should be operable in temperatures ranging from -15 to 130 ℉

3.2.2.2.1: This test will be scripted. The test steps are as follows:

3.2.2.2.1.1: Tester will place the car in 5 negative degree temperatures lower than -15℉ and record if IoT HtL is able to fully operate.

3.2.2.2.1.2: Tester will place the car in 5 positive degree temperatures higher than 130 ℉ and record if IoT HtL is able to fully operate

3.2.2.2.1.3: The tester will confirm or deny that IoT HtL was able to successfully operate all 10 times in the prior steps.

3.2.2.3 IoT HtL shall have a battery backup that can power the car for at least (50 miles)

3.2.2.3.1: This test will be scripted. The test steps are as follows:

3.2.2.3.1.1: IoT HtL will have empty gas to start

3.2.2.3.1.2: Tester will confirm that IoT HtL can drive at least 50 miles prior to shut down.

Section 3.2.3: Security

3.2.3.1 Access to IoT HtL shall be secured by UserID and password

3.2.3.1.1: This test will be scripted. The test steps are as follows:

3.2.3.1.1.1: Tester will be given a used/real UserID and Password to a tester car.

3.2.3.1.1.1.1: Tester will attempt to log in and report an error if unsuccessful

3.2.3.1.1.2: Tester will be given a non real UserID and Password to a tester car

3.2.3.1.1.2.1: Tester will attempt to log in a report an error if successful

3.2.3.2 IoT HtL must differentiate between technician and operator access

3.2.3.2.1: This test will be scripted. The test steps are as follows:

3.2.3.2.1.1: Tester will be given a technician log in detail

3.2.3.2.1.1.1: Tester will use technician log in to log into IoT HtL's system. If unsuccessful the tester will report an error.

3.2.3.2.1.2: Tester will be given an operator's log in

3.2.3.2.1.1.2: Tester will use operator log in to log into IoT HtL's system. If unsuccessful the tester will report an error.

Section 3.2.4: Software Update

3.2.4.1 Software updates by company cloud which production software is uploaded to when approved by Quality.

3.2.4.1.1: This test is unscripted. Tester is required to execute based upon the test objectives and document summary of testing

3.2.4.1.2: Tester will provide screenshots of software located in company cloud with Quality Approval

3.2.4.2 Software updates secure by encryption

3.2.4.2.1: This test is unscripted. Tester is required to execute based upon the test objectives and document summary of testing

3.2.4.2.2: Tester will provide screenshots of encryption software being used for the software updates

3.2.4.3 "Engine" must be off while software updates occur

3.2.4.3.1: This test is scripted. The steps are as follows:

3.2.4.3.1.1: Tester will attempt to download a software update while the car engine is on.

3.2.4.3.1.1.1: Tester will report an error if the software is able to successfully download

3.2.4.3.1.2: Tester will attempt to download a software update while the car engine is off.

3.2.4.3.1.2.1: Tester will report an error if the software is not able to successfully download

3.2.4.4 Backups of previous data and configurations must be stored to the company cloud before updating

3.2.4.4.1: This test is unscripted. Tester is required to execute based upon the test objectives and document summary of testing

3.2.4.4.2: Tester will provide screenshots of the *upload of* software backups and the backups themselves present in the company cloud

3.2.4.5 Software updates must not conflict with other softwares, applications and processes

3.2.4.5.1: This test is unscripted. Tester is required to execute based upon the test objectives and document summary of testing

3.2.4.5.2: Tester is to use software while updating 10 times and report an error if the system is unable to simultaneously run.

3.2.4.6 User must provide valid authentication credentials to initiate a software update

3.2.4.6.1: This test is scripted. The steps are as follows:

3.2.4.6.1.1: Tester is given non real authentication credentials

3.4.6.1.1.1: Tester is to attempt to initiate a software update with the given credentials and will report an error if able to successfully initiate a software update

3.2.4.6.1.2: Tester is given real authentication credentials

3.2.4.6.1.2.1: Tester is to attempt to initiate a software update with the given credentials and will report an error if unable to successfully initiate a software update

<u>Section 3.2.5: User Interface (For technician)</u>

3.2.5.1 Must be able to read and access audit logs which are logged in the company cloud

3.2.5.1.1: Audit logs must be backed up in company cloud

3.2.5.1.1.1: This test is unscripted. Tester is required to execute based upon the test objectives and document summary of testing

3.2.5.1.1.2: Tester will provide screenshots of the *upload of* software backups and the backups themselves present in the company cloud

3**.2.5.1.2:** Audit Logs must be accessed with valid technician credentials (This feature **3.2.5.1.2** is an extension of **3.2.3.2**. Reference section **3.2.3.2** for testing strategy).

Section 3.2.6: Display (for driver and passengers)

3.2.6.1 Display will be in the center console so as to not distract the driver

3.2.6.1.1: This test is unscripted. Tester is required to execute based upon the test objectives and document summary of testing

3.2.6.1.2: Tester will provide photo evidence of the display being in the center console of the car

3.2.6.2 Display will show all active driver assist features

3.2.6.1.1: This test is unscripted. Tester is required to execute based upon the test objectives and document summary of testing

3.2.6.1.2: Tester will provide photo evidence of assisted features being shown on the drivers dashboard when activated.

3.2.6.3 Display will be a full touch screen equipped to control internal systems such as climate control, audio, and navigation

3.2.6.3.1: This test is unscripted. Tester is required to execute based upon the test objectives and document summary of testing

3.2.6.4 Adjusting settings may only be accessed when the car is parked, meaning the car is going at a speed, s, s = 0 mph.

3.2.4.6.1: This test is scripted. The steps are as follows:

3.2.4.6.1.1: The tester is to attempt to access settings while going at a speed, s>0. If the tester is successful at accessing the settings, the tester is to report an error.

3.2.4.6.1.2: The tester is to attempt to access settings while going at a speed, s<0. If the tester is unsuccessful at accessing the settings, the tester is to report an error.

Section 3.2.7: Data Collection

3.2.7.1 Autonomous test cars will send data to the Automated Vehicle Transparency and Engagement for Safe Testing Initiative for transparency of testing results

3.2.7.1.1: This test is unscripted. Tester is required to execute based upon the test objectives and document summary of testing.

3.2.7.1.2: Tester will provide screenshots that the data is being sent to the Automated Vehicle Transparency and Engagement for Safe Testing Initiative

3.2.7.2 The car will send environmental data to Cloudfactory (third party data solution) such as road rules, road signs, area wildlife, and weather conditions to provide a numerous data sets to train the AI

3.2.7.2.1: This test is unscripted. Tester is required to execute based upon the test objectives and document summary of testing.

3.2.7.2.2: Tester will provide screenshots of environmental data being uploaded to Cloudfactory

3.2.7.3 When the car fails it will send a report to our company which will be further investigated

3.2.7.3.1: A failing car means: Any part of the car that is supposed to be functional, is non-functional i.e the sensors are unable to transmit data, circuitry for lights, doors, windows are not working, battery dies, engine dies, cruise control malfunctions etc.

3.2.7.3.2: This test is scripted. The steps are as follows:

3.2.7.3.2.1: The tester is to have a test car which has a faulty proximity sensor.

3**.2.7.3.2.2:** The tester is to back up the car close to an object and see if the proximity sensor triggers, by either a beeping noise to show that the car is close to the object or the car triggering auto-brakes.

3.2.7.3.2.2.1: A failing car which does not fulfill the criteria for the proximity sensor above, will send a report to the company.

3.2.7.3.2.3: The tester will assess and document screenshots of the report being automatically sent to the company. If the report does not go through, the tester is to report an error in feature 3.2.7.3.

Section 4: Requirement Modeling

Section 4.1: Use Case Scenarios

Section 4.1.1: Use Case 1 - Enabling Cruise Control

Primary Actor: Driver

` **Goal in Context:** Driver enables cruise control

Preconditions:

- Vehicle is at a minimum speed s > 0
- Vehicle is in drive

Trigger: Driver pushes Cruise Control button

Scenario:

1. Driver sends Cruise Control activation request by pressing the Cruise Control button on the Digital Dashboard
2. Digital Dashboard displays the words “SETTING CRUISE” above the Speedometer with a yellow light
3. Planning begins to analyze the speed data continuously sent by Sensor Fusion

    a. If either or both exceptions are met, the Digital Dashboard displays the words “CRUISE UNAVAILABLE” above the Speedometer with a red light
4. Planning sends the speed request and data to the Vehicle Control System
5. Vehicle Control System maintains the Vehicle’s speed
6. Digital Dashboard displays the words “CRUISE ON” above the Speedometer with a green light
7. System Management logs the Cruise Control Event

Exceptions:

- Vehicle is not in motion (speed = 0)
- Vehicle is not in drive

Section 4.1.2: Use Case 2 - Collision Prediction

Primary Actor: Vehicle

` **Goal in Context:** Vehicle avoids a potential collision

Preconditions:

- Driver is operating the Vehicle
- Vehicle is approaching or being approached by an Obstacle

Trigger: Driver has not reacted to approaching Obstacle in a variable amount of time

Scenario:

1. Sensor Fusion continuously sends data collected by Proximity Sensors and Cameras to Planning
2. Planning calculates the time until a possible collision with an Obstacle
    a. If the Driver reacts to the obstacle within the variable time, Collision Prediction is not implemented
3. Planning applies an algorithm to the Sensor Fusion data to determine the Vehicle’s best course of action (stopping, changing lanes, slowing down, etc.)
4. Planning sends a Trajectory Change request to the Vehicle Control System
5. Vehicle Control System implements the action recommended by Planning
6. Vehicle emits an audible chirping sound through the Digital Dashboard and a tactile buzz through the Steering Wheel to alert the Driver that a Collision Prevention event occurred
7. System Management logs the Collision Prevention Event

Exceptions:

- Driver has reacted to the Obstacle within the variable time

Section 4.1.3: Use Case 3 - Traction Control

Primary Actor: Vehicle

` **Goal in Context:** Vehicle regains Traction

Preconditions:

- Driver is operating the Vehicle

- Traction Control is enabled (Default Setting)

Trigger: Vehicle has lost traction

Scenario:

1. Sensor Fusion continuously sends data from the RPM Sensor and the Wheel Speed Sensor to Planning
2. Planning finds a discrepancy between the Vehicle's speed and the rotation speed of the Vehicle's wheels
3. Planning sends a Traction Control request to the Vehicle Control System
4. Driver receives a visual and audible alert that Traction Control was engaged from the Digital Dashboard
5. Vehicle Control System temporarily stops the Vehicle wheels until traction is regained
6. System Management logs the Traction Control event

Exceptions:

- Traction Control is not enabled

Section 4.1.4: Use Case 4 - Lane Mitigation and Tracking

Primary Actor: Vehicle

` **Goal in Context:** Vehicle stays in lane

Preconditions:

- Driver is operating the Vehicle
- Vehicle is beginning to Drift

Trigger: Driver has not corrected the Drifting

Scenario:

1. Driver is operating the Vehicle
2. Sensor Fusion continuously sends data from the positional Sensors to Planning
3. Planning detects that the Vehicle is Drifting
4. Digital Dashboard emits an audible, visible, and tactile alert to the Driver (chirping sound for audible, "DRIFTING ALERT" for visual, and a buzz through the Steering Wheel for tactile)
    a. If Driver reacts immediately, Lane Mitigation does not continue
5. Planning calculates the direction and distance the Vehicle must change to remain in its lane
6. Planning sends a Lane Mitigation and Tracking request to the Vehicle Control System
7. Vehicle Control System takes action to rectify the position as recommended by Planning
8. System Management logs the Lane Mitigation Event

Exceptions:

- Driver manually corrects the drifting

Section 4.1.5: Use Case 5 - Traffic Sign Recognition

Primary Actor: Vehicle

` **Goal in Context:** Vehicle implements action directed by Traffic Sign

Preconditions:

- Vehicle is on a Road with Traffic Signs

Trigger: Vehicle is approaching a Traffic Sign

Scenario:

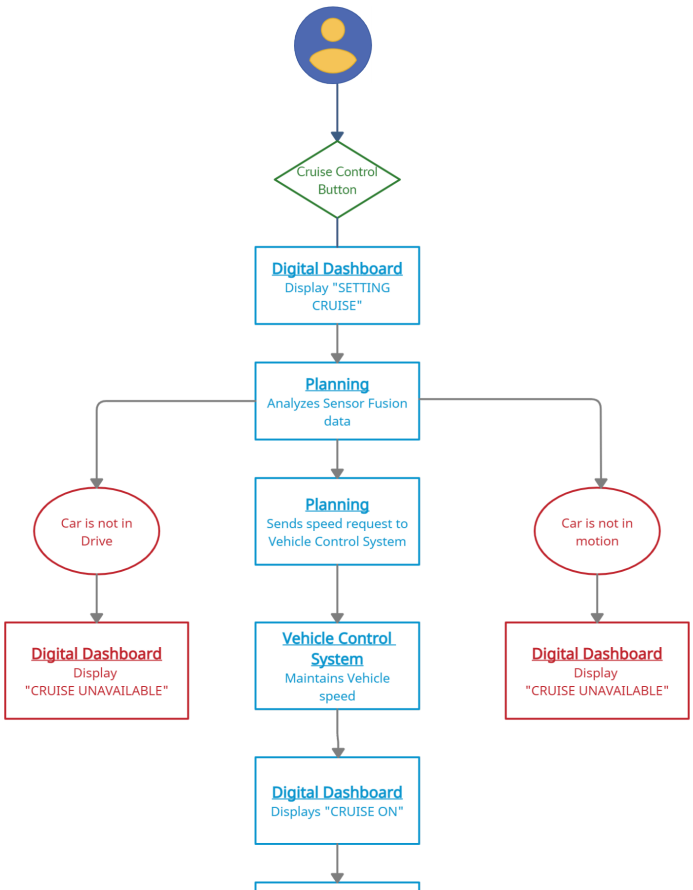
1. Sensor Fusion continuously sends data from Front-Facing Cameras and GPS Coordinates to Planning
2. Planning uses this data to determine if there are surrounding Traffic Signs
    a. If none are detected, Traffic Sign Recognition is not enabled
3. Planning sends Traffic Sign information to the Vehicle Control System
4. Vehicle Control System adjusts the Vehicle depending on the information (e.g. increases or decreases speed depending on the limit)
5. Digital Dashboard displays a miniature copy of the Traffic Sign to the Driver

Exceptions:

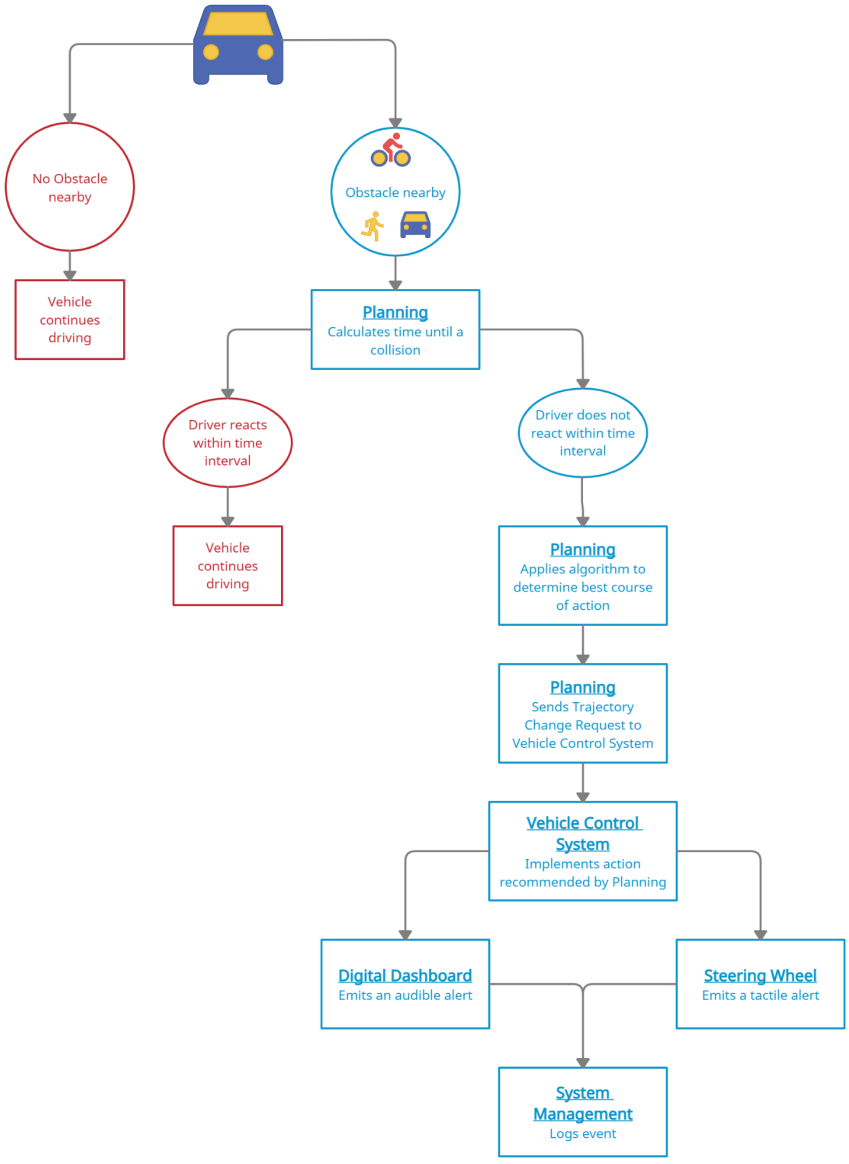
- No Traffic Signs are detected

Section 4.2: Activity Diagrams

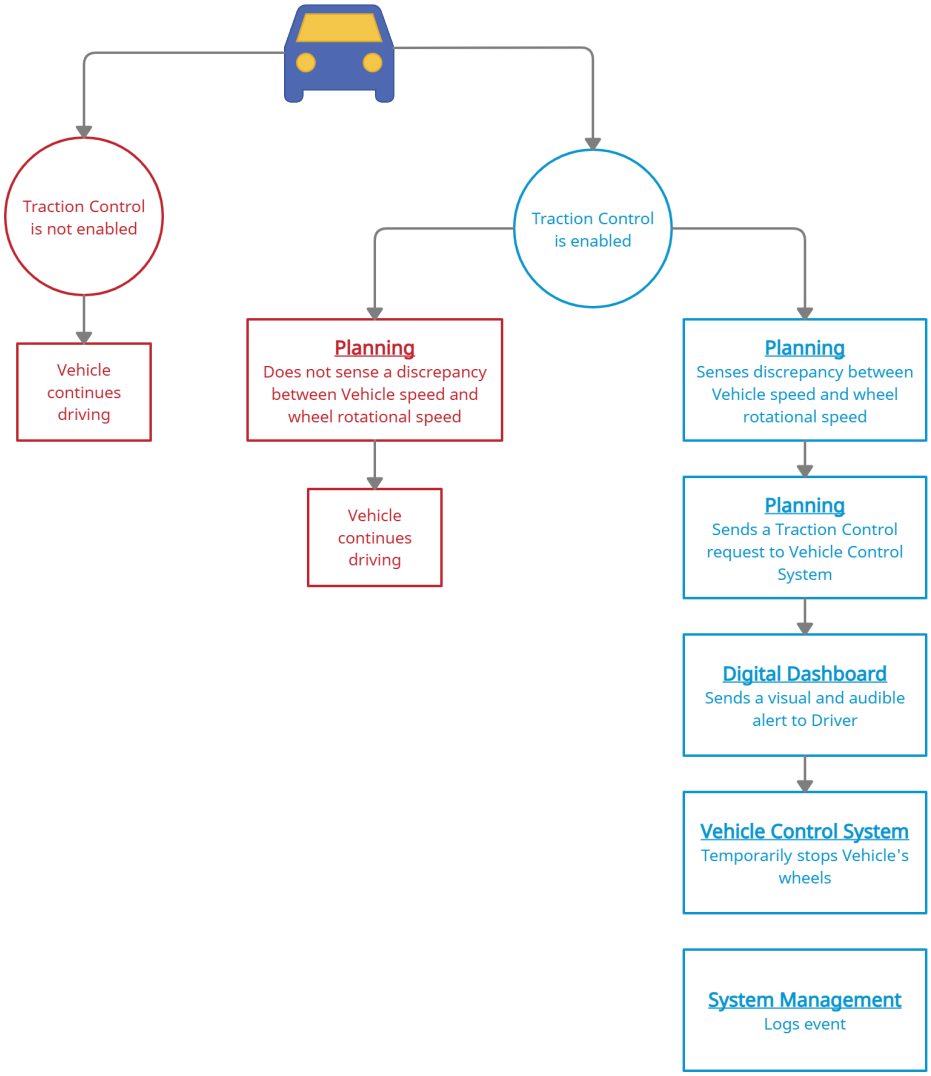
Section 4.2.1: Activity Diagram 1 - Enabling Cruise Control



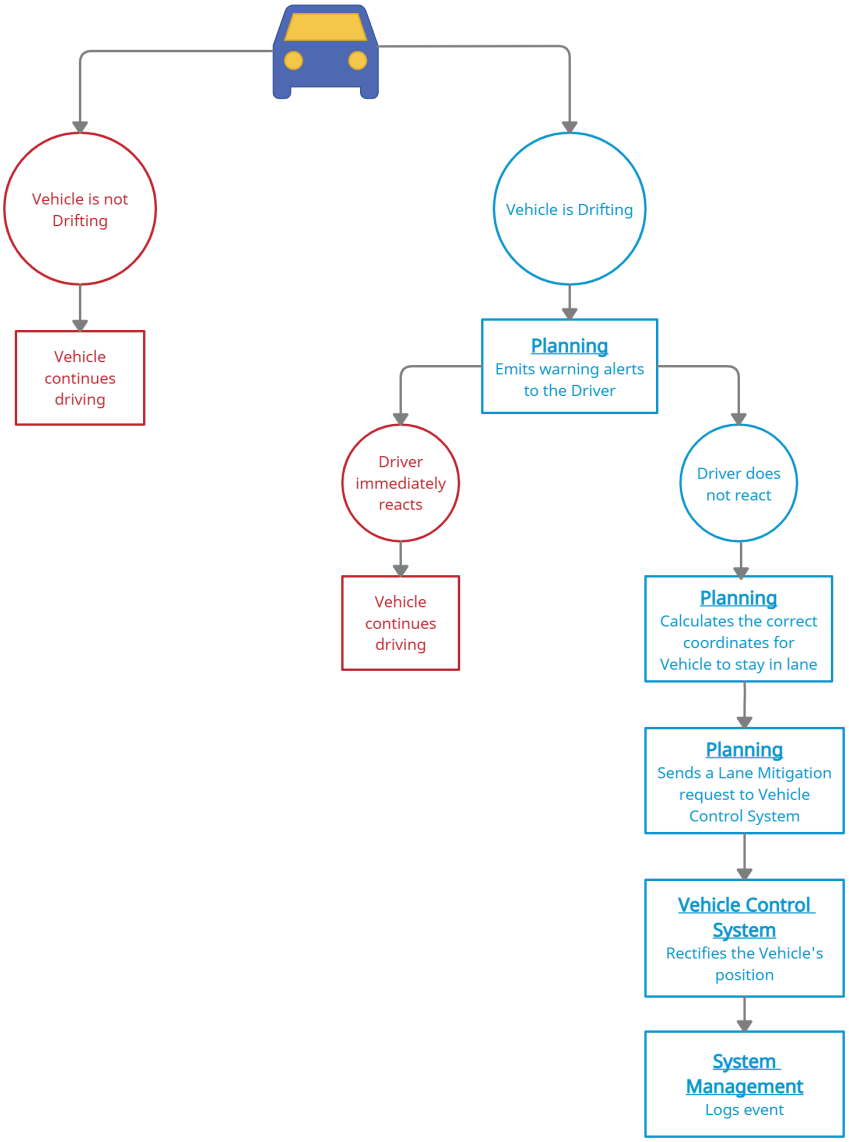
Section 4.2.2: Activity Diagram 2 - Collision Prediction



Section 4.2.3: Activity Diagram 3 - Traction Control

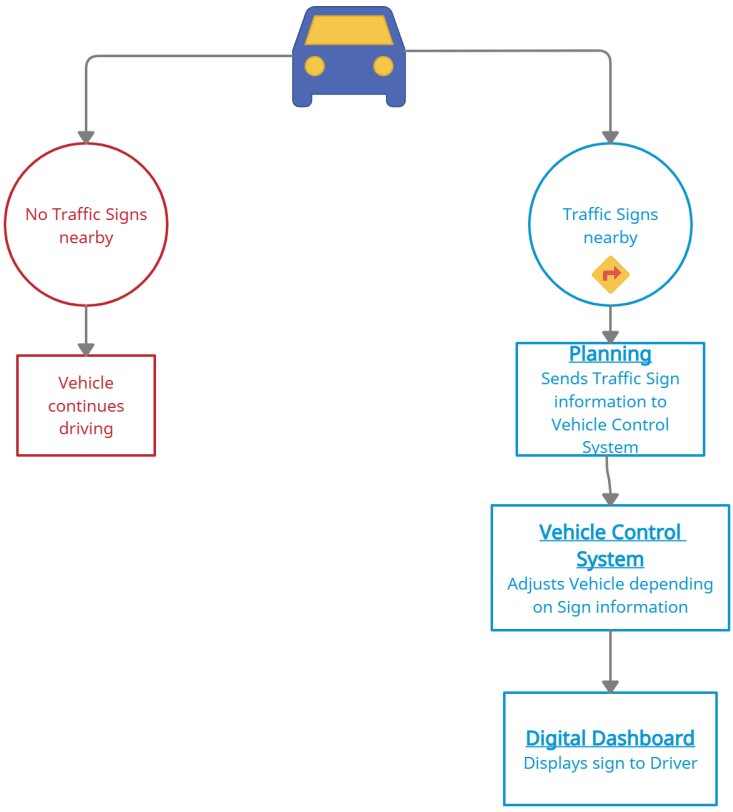

Traction Control is not enabled
Vehicle continues driving
Traction Control is enabled
Planning
Does not sense a discrepancy between Vehicle speed and wheel rotational speed
Vehicle continues driving
Planning
Senses discrepancy between Vehicle speed and wheel rotational speed
Planning
Sends a Traction Control request to Vehicle Control System
Digital Dashboard
Sends a visual and audible alert to Driver
Vehicle Control System
Temporarily stops Vehicle's wheels
System Management
Logs event

Section 4.2.4: Activity Diagram 4 - Lane Mitigation and Tracking



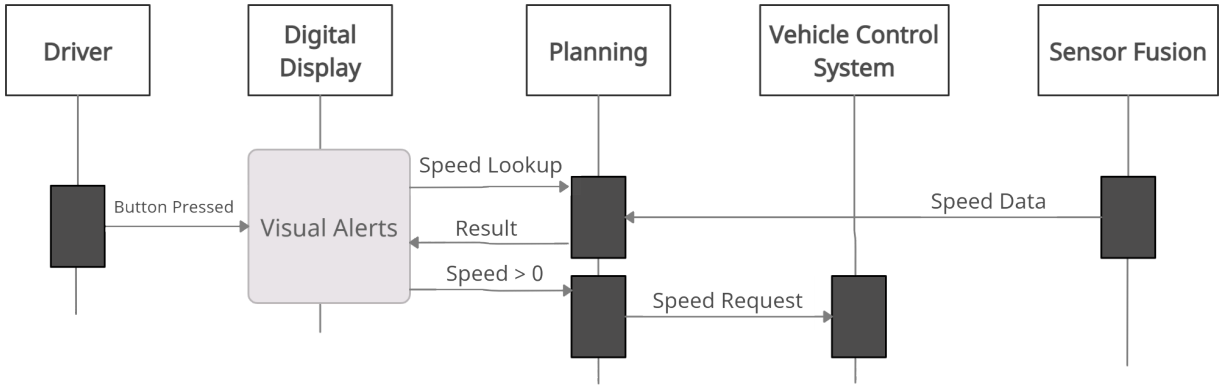
‘

Section 4.2.5: Activity Diagram 5 - Traffic Sign Recognition

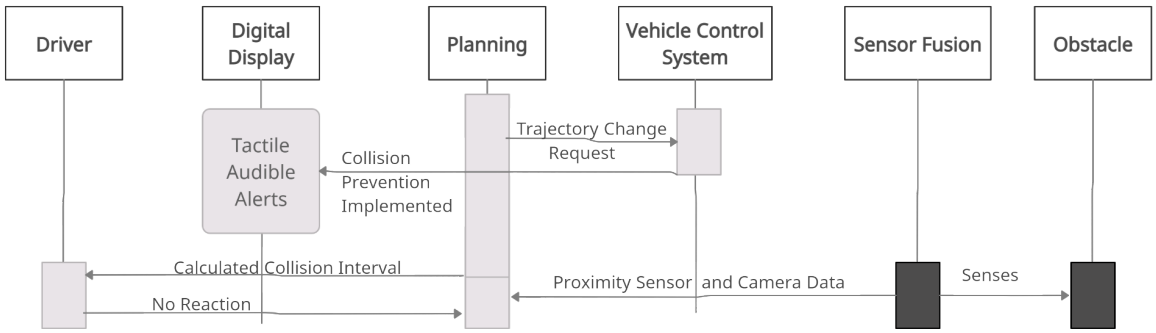


Section 4.3: Sequence Diagrams

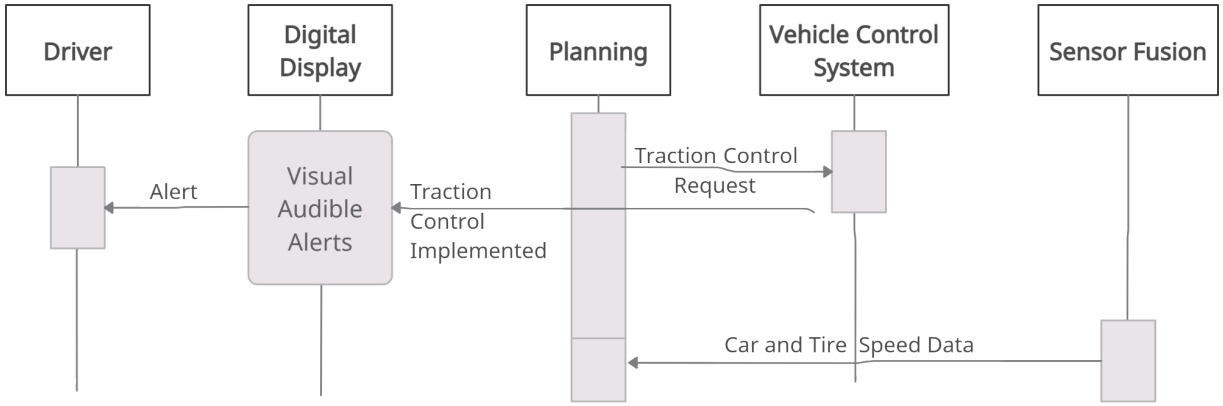
Section 4.3.1:Sequence Diagram 1 - Enabling Cruise Control



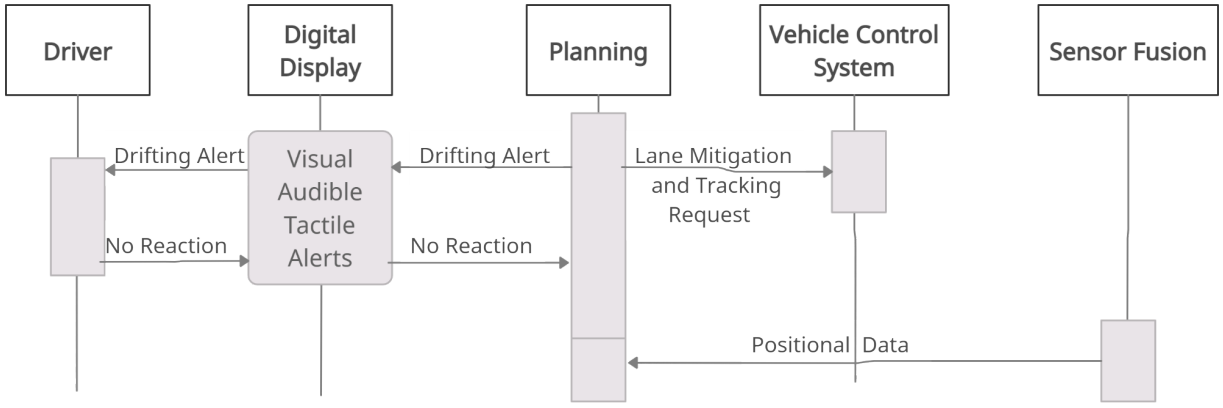
Section 4.3.2: Sequence Diagram 2 - Collision Prediction



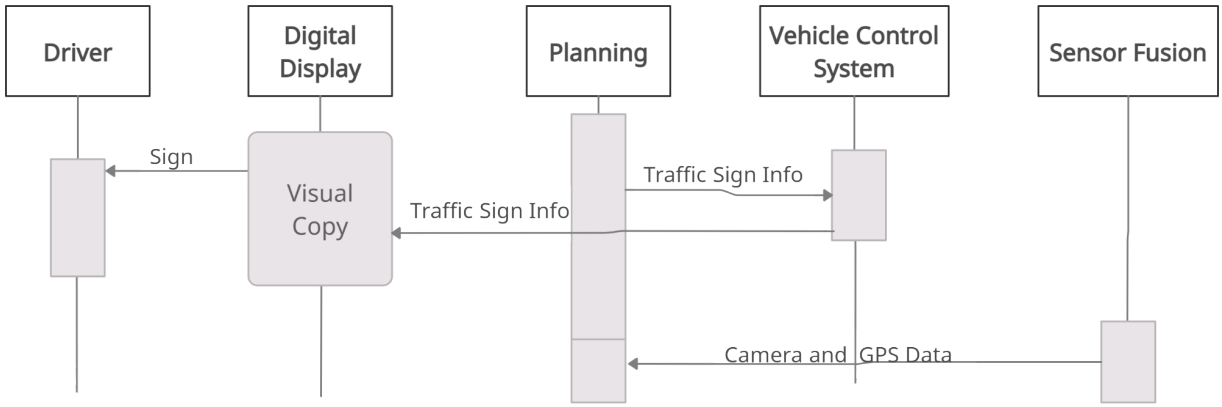
Section 4.3.3: Sequence Diagram 3 - Traction Control



Section 4.3.4: Sequence Diagram 4 - Lane Mitigation and Tracking



Section 4.3.5: Sequence Diagram 5 - Traffic Sign Recognition



Section 4.4: Classes

Classes must be defined with meaningful attributes and methods that you observed in the expected software behavior (e.g. Use Cases)

Potential Class	*Methods*	*Expected Behavior (Use Cases)*

Driver / Interface	brake(x)
	accelerate(x)
	steer(x,y)
	cruiseControl();
	tractionControl();
	setDestination();
	autoPilot();
	parallelPark();
	anglePark(x);
		Controls basic operation of the vehicle. Activates different assisted and self driving features from the interface.

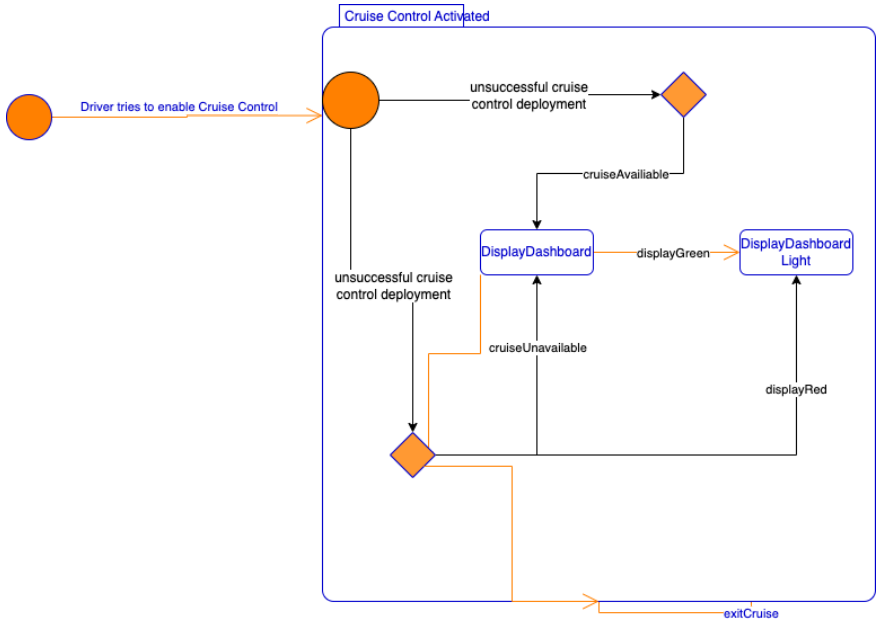
GPS Sensor	getLocation()	Returns the exact location of the vehicle.
Proximity Sensor	getDistance() getAngle()	Returns the distance between objects in close proximity to the vehicle.
LIDAR	detect()	Detects objects around the vehicle.
Speedometer	getSpeed()	Returns current speed of vehicle.
RPM Sensor	getRPM() tractionControl(x)	Returns RPM and activates tractionControl when necessary.
Side Cameras	getColor() detect()	Detects and records objects, actions, and colors to the sides of the car.

Rear Cameras	getColor() detect() record() log()	Detects and records objects, actions, and colors to the rear of the car.
Temperature Sensor	getTemperature()	Returns outdoor and interior temperatures.
Brake Sensor	isBraking(x)	Detects pressure of the brake.
Display	showSpeed() showTemperature() showSurroundings() showLocation()	Displays vital information to the driver.
System Logs	upload() download() log()	Stores, logs, and uploads system logs locally and to the cloud.
User Credentials	login(Username, Password) logout()	Stores user information for system and cloud access.
Planning Unit		
Vehicle Control	brake(x) accelerate(x) steer(x,y)	Receives commands from planning and adjusts vehicle speed and position accordingly.

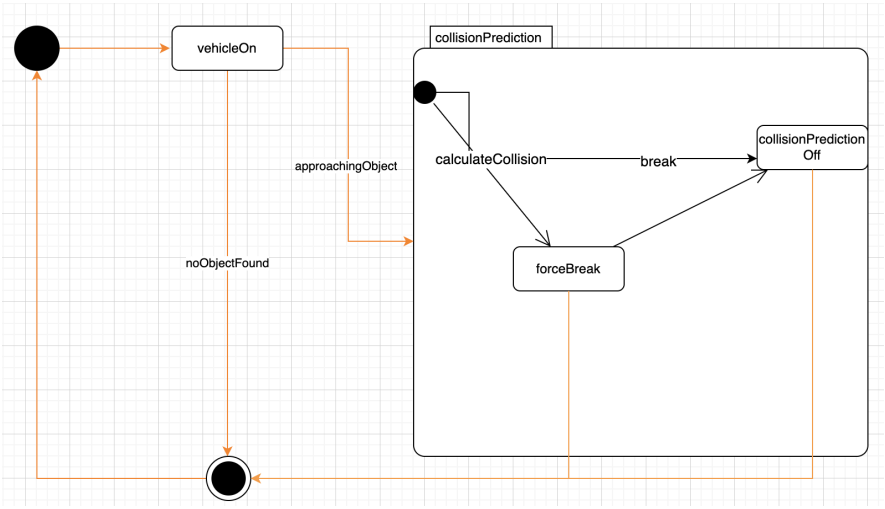
Section 4.5: State Diagrams

Nodes must be the states that the software creates, not processes (such as Normal State, Orange State, Red State of the software/car). The arrowed lines must be the triggers that cause state changes.

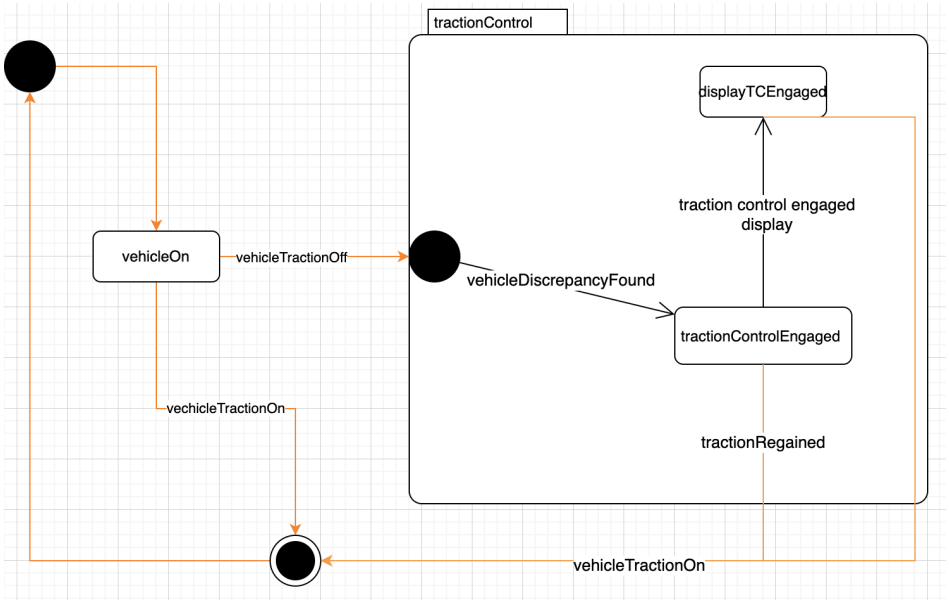
Section 4.5.1: State Diagram 1 - Enabling Cruise Control



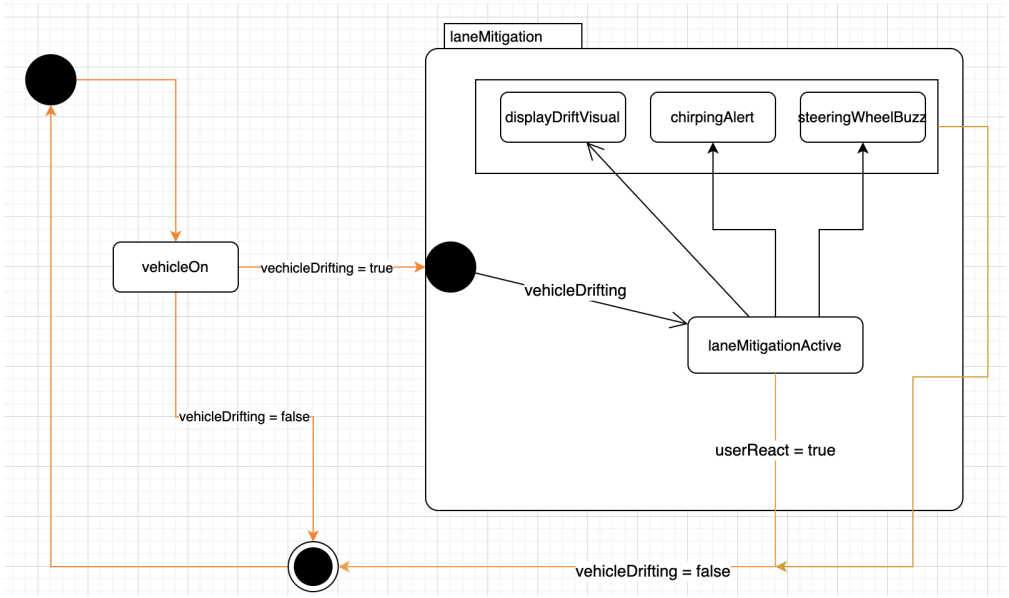
Section 4.5.2: State Diagram 2 - Collision Prediction



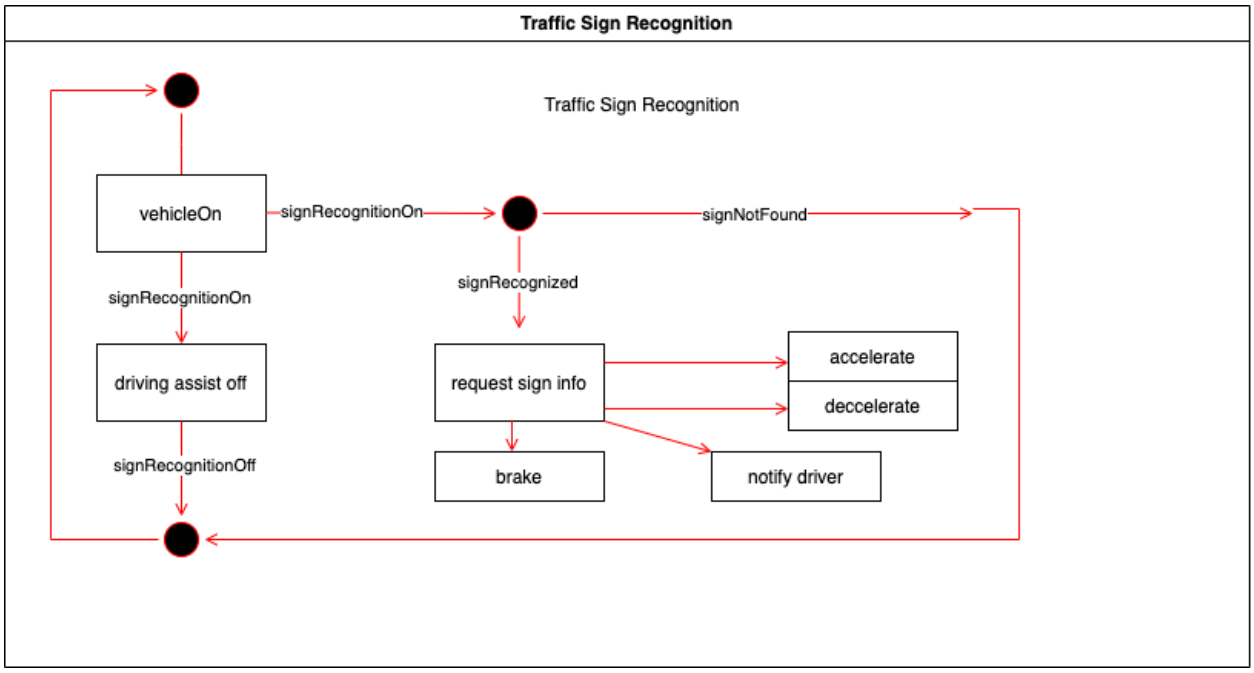
Section 4.5.3: State Diagram 3 - Traction Control



Section 4.5.4: State Diagram 4 - Lane Mitigation and Tracking



Section 4.5.3: State Diagram 5 - Traffic Sign Recognition



Section 5: Design

Section 5.1: Software Architecture

that defines the relationship among the major software structural elements.
Examine every Software Architecture model (discussed in the Lecture slides) feasibility for IoT HTL.
State how the architecture would be a fit or not for IoT HTL.
State the specific pros and cons of the model as it applies to the IoT HTL
Finalize your choice of software architecture based on the above pros and cons and state it clearly.

5.1.1: Data Centered Architecture

In Data Centered Architecture, Client Software sends requests to a Data Store, which provides permanent data storage, in order to perform actions. A centralized Data Store would fit IOT HTL, as it incorporates many sensors and cameras that collect data to be processed. However, the components must operate around an agreed repository data model, so it would be necessary to compromise between the specific needs of each component, which is difficult for a vehicle that relies on a wide variety of data.
Pros:

- Components can be independent
- Changes made by one Component can be propagated to all Components
- All data can be managed consistently

Cons:

- Any failure in the Data Store affects the entire system
- Possible inefficiencies in organizing all communication through the Data Store
- Difficult to integrate new components
- Components must operate around an agreed repository data model

Conclusion: Bad Fit

5.1.2: Data Flow Architecture

In Data Flow Architecture, data flows and is transformed as it moves through a sequence. Each step in processing is implemented as a transform , so input data flows through transforms until it is converted into output. The flow of this data architecture is similar to the flow of the Vehicle's processes, but it is more suited to simple inputs and outputs. IOT HTL's constant stream of data may be too complex for this model.
Pros:

- Easy to understand and supports transformation reuse
- Workflow style matche s the Vehicle's processes

- Easy to integrate new components and functions
- Can be either sequential or concurrent

Cons:

- Data transfers must parse input and unparse output, which increases system overhead
- May be difficult to reuse architectural components
- Model is not well suited for interactive systems

Conclusion: Possible Fit

5.1.3: Call Return Architecture

In Call Return Architecture, a main program is divided into smaller pieces throughout a hierarchy. The main program divides into controller subprograms, which are in turn divided into application subprograms. This style of architecture could match the Vehicle's functionalities, where different sensor data is required for specific functions, but the functions and activities are styled in a different way, without much of a hierarchy.

Pros:

- Easy to modify and scale
- Easy to integrate new components and functions
- Analyzing control flow is simple

Cons:

- Parallel processing may be difficult to achieve
- Operation exceptions may be difficult to implement

Conclusion: Possible Fit

5.1.4: Object-Oriented Architecture

In Object-Oriented Architecture, components (objects) operate through procedure calls (methods). This style makes sense for certain Vehicle functions, where an actor (either the Driver of the Vehicle) can call a method, but may be difficult to implement during other functions that rely on sensor data.

Pros:

- System structure is easy to understand
- Easy to modify objects and their implementations without affecting Clients
- Inheritance and polymorphism

Cons:

- Need to explicitly reference names and interface of objects
- High overhead to maintain objects
- Different uses of an object may cause unexpected side effects

Conclusion: Bad Fit

5.1.5: Layered Architecture

In Layered Architecture, the system is organized into layers, each of which has a functionality associated with it. The lowest layer represents the core services used throughout the system, and each layer provides services to the layer above it. It could be possible to split the Vehicle's Functional Architecture into separate layers, but there is some overlap between the responsibilities in each layer.

Pros:

- Layers can be easily replaced if interface is maintained
- Redundant facilities can be provided in each layer to increase system dependability

Cons:

- A clean separation between layers is difficult to achieve
- Performance may be affected by multiple levels of interpretation for each layer

Conclusion: Bad Fit

5.1.6: Model View Controller Architecture

In Model View Controller Architecture, the system is separated into three logical components that interact with each other: Model, View, and Controller. Model manages system data and associated data operations, View defines and manages how the data is presented to the user, and the Controller manages user interaction and passes them to View and Model. This split between Model, View, and Controller would work well with the functionalities implemented in the Vehicle, but there may be more of a burden placed on the Model component.

Pros:

- Data can change independently of its representation
- Same data can be presented in different ways

Cons:

- May involve additional code and code complexity when the data model and interactions are simple

Conclusion: Good Fit

5.1.7: Finite State Machine

In a Finite State Machine Architecture, the system is defined in states, initial states, and transitions. The system begins at an initial state and undergoes transitions until it goes into the desired state. While this style of architecture would allow us to define the states and necessary transitions of the vehicle, the wide variety of states would be difficult to implement.

Pros:

- Stable and easy to maintain
- Easy to track transitions and current conditions

Cons:

- Difficult to perform asynchronous background executions

- Very data driven, states may require multiple transitions with specific preconditions

Conclusion: Possible Fit

Our Architecture Design: Model View Controller

Section 5.2: Interface Design

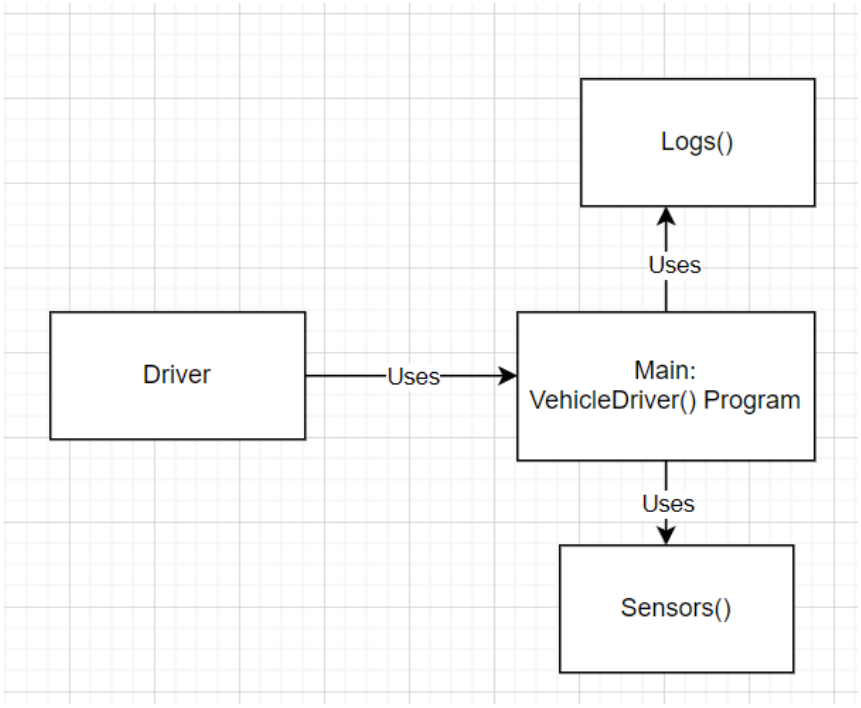
Defines how software elements, hardware elements, and end-users communicate. Consider both Technician and Driver interfaces. Specify exactly what will appear in each of the user interfaces with names, class, data, etc.

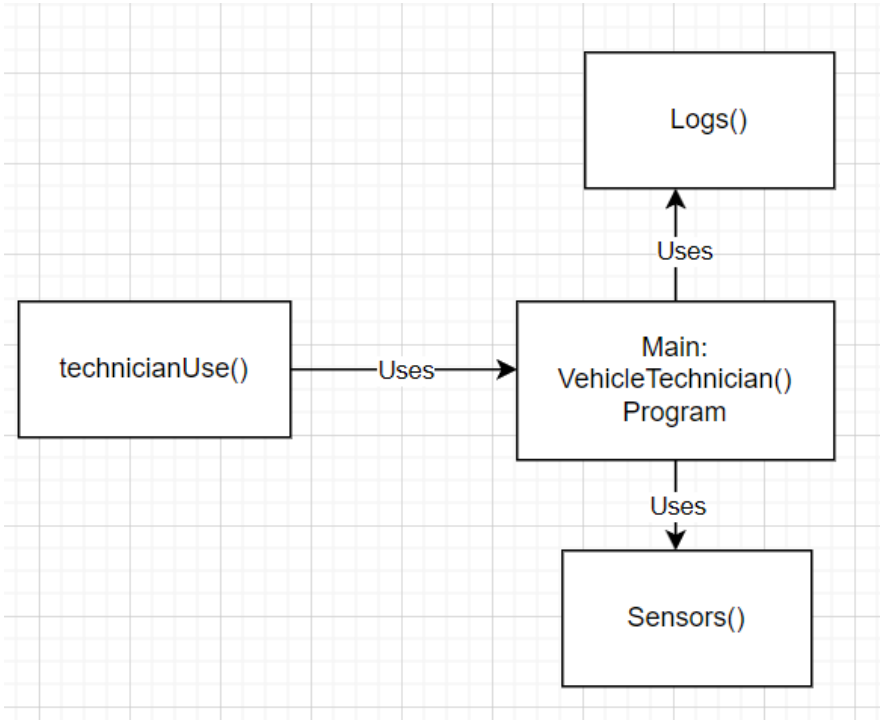
The below table states each interface, which user interacts with it, and the elements, data, and methods that will be programmed. Some elements, such as the Alerts/Errors, only return and store boolean values. Most interface elements store values that are triggers for other elements. Following the Model View Controller Architecture, the Controller manages user interaction, sensor data, and planning module decisions to trigger events and methods. For example, climate conditions and camera clarity can automatically turn off lane mitigation, surroundings render, and collision prediction and notify the user. Cruise control and self driving always receive data from the speedometer and distance sensors. Traction control is reliant on the RPM meter. Logs are accessible through the Technician and Driver toolkits in the Central Display. All events and driving mode elements are logged.

Interface	*End User*	*Element*	*Data*	*Methods*
Dashboard	Driver / Technician	Speedometer	Speed	get_current_speed() log_speed()
Dashboard	Driver / Technician	RPM Meter	RPM	get_current_rpm() log_rpm()
Dashboard	Driver / Technician	Battery Level	Battery Battery Temperature	get_battery_level() get_battery_temp() get_battery_status() log_battery_level() log_battery_temp() log_battery_status()
Dashboard	Driver / Technician	Alerts / Errors	Motor status Auto Glide Control Warning Eco Mode General Electric Issue General Issue	

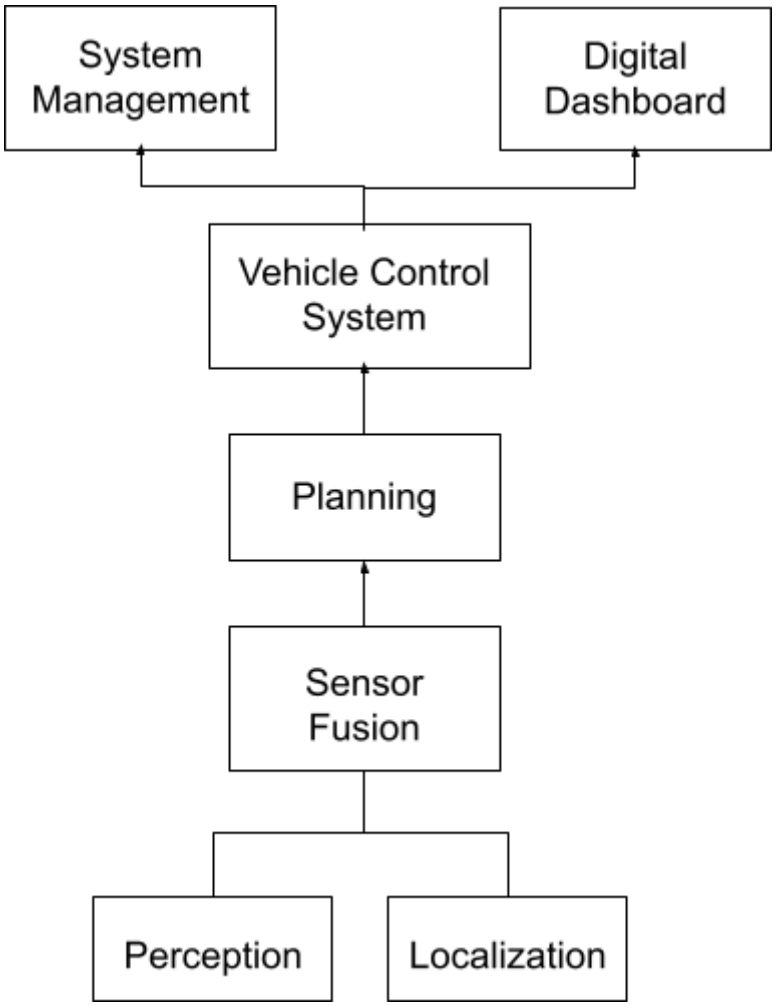
			Pedestrian Warning Limited Power Ready to Drive Regenerative Braking Regenerative Braking Warning External Sound Issue	
Central Display	Driver	Camera viewfinders	Color Objects Alignment Lane line Clarity	get_color() get_distance() get_clarity()
Central Display	Driver	Surroundings Real-time Render	Traffic lights Signs Pedestrians Other cars Lane line	alert() log_alerts()
Central Display	Driver	Climate Conditions	Temperature Wind Speed Ice conditions	forceBrake() log_temp() log_wind() log_ice() alert()
Central Display	Driver	Driving Mode	Driver assist Fully automated Fully manual	get_driving_mode() log_driving_mode()
Central Display	Driver	Collision Prediction	Distance	forceBrake() get_cp_status() log_event()
Central Display	Driver	Driver Toolkit	Logs	login() logout() get_logs()
Central Display	Technician	Technician Toolkit	Logs	login() logout() get_logs()
Steering Wheel Console	Driver	Cruise control activation	On/Off Starting Speed Cruising Speed	increment() decrement() set_cc_speed() increment_cc_speed() decrement_cc_speed()

				get_cc_status() deactivate() log_event()
Steering Wheel Console	Driver	Lane mitigation activation	On/Off Cruising Speed	get_lm_status() steerLeft() steerRight() deactivate() log_event()
Steering Wheel Console	Driver	Self driving activation	On/Off Cruising Speed	increment() decrement() set_sd_speed() increment_sd_speed() decrement_sd_speed() get_sd_status() deactivate() log_event()
Steering Wheel Console	Driver	Traction control activation	On/Off	get_tc_status() deactivate() log_event()
Sounds	Driver	Lock/Unlock	Locked/Unlocked	log_event()
Sounds	Driver	Blind spot detection	Blind spot present	alert()
Sounds	Driver	Seatbelt Warning	Seatbelt On/Off	alert()
Sounds	Driver / Technician	Errors and Alerts		alert()


Logs()
Uses
Driver
Uses
Main:
VehicleDriver() Program
Uses
Sensors()


Logs()
Uses
technicianUse()
Uses
Main:
VehicleTechnician()
Program
Uses
Sensors()

Section 5.3: Component-level Design



Section 6: Project Code

Section 6.1: Main

```
// global variables:
var speed = 0;
var rpm = 0;
var longitude = 0;
var latitude = 0;
var btemp = 0;
var otemp = 0;
var clicked = false;
var itemp = 0;
var etemp = 0;
var wspeed = 0;
var humid = 0;
var visib = 0;
var frontProx = 0;
var rearProx = 0;
var blindS = false;
var cruiseStart = 0;
var cruiseSpeed = 0;
var cruiseOn = false;
var leftLineDist = 0;
var rightLineDist = 0;
var whatSignal = "";


function check(form) {
  if (form.userid.value == "mechanic" && form.password.value ==
"password") {
    window.location.href = "main.html";
  } else {
    alert("Invalid Username or Password.");
  }
}

/*
 * getRandInput: Generates random numbers since we do
```

```
 *                not have the sensor input. You may change
 *                the global vairables if you want to test
 *                your own input.
 */

function getRandInput() {
  // Speed: GPS generated speed.
  speed = Math.floor(Math.random() * 100); // speed = <number>
  document.getElementById("carSpeed").innerHTML = speed; // do not change.

  // RPM: Rotations Per Minute on the car wheels
  rpm = Math.floor(Math.random() * 1000); // rpm = <number>
  document.getElementById("rpm").innerHTML = rpm; // do not change.

  // Battery Temp: Battery Temperature of the car.
  btemp = Math.floor(Math.random() * 190); // etemp = <number>
  document.getElementById("batteryTemp").innerHTML = btemp; // do not
change.

  // Otemp: Battery Level of the car.
  otemp = Math.floor(Math.random() * 100); // otemp = <number>
  document.getElementById("batteryLevel").innerHTML = otemp; // do not
change.

  // ActualSpeed: Calculates the speed from RPM.
  actualSpeed = Math.floor(((2.89 * rpm) / 1609) * 60); // actualSpeed =
<number>
  document.getElementById("calcSpeed").innerHTML = actualSpeed; // do not
change.

  // External Temp: Outdoor Temperature
  etemp = Math.floor(Math.random() * (90 - 30) + 30); // etemp = <number>
  document.getElementById("externalTemp").innerHTML = etemp; // do not
change.

  // Internal Temp: Cabin Temperature of the car.
  itemp = etemp - 10;
  document.getElementById("internalTemp").innerHTML = itemp; // do not
change.
```

```
// Wind Speed
wspeed = Math.floor(Math.random() * 11);
document.getElementById("windSpeed").innerHTML = wspeed;

// Humidity
humid = Math.floor(Math.random() * 100);
document.getElementById("humidity").innerHTML = humid;

// visibility
visib = Math.floor(Math.random() * 100);
document.getElementById("visibility").innerHTML = visib;

// Distance between car and object in front
frontProx = Math.floor(Math.random() * 20);
document.getElementById("frontDistance").innerHTML = frontProx;

// Distance between car and object behind
rearProx = Math.floor(Math.random() * 20);
document.getElementById("rearDistance").innerHTML = rearProx;

// Blind Spot yes or no
blindS = Math.random() < 0.5;
document.getElementById("blindSpot").innerHTML = blindS;

// Cruise control on or not
cruiseOn = Math.random() < 0.5;
document.getElementById("cruiseControlStatus").innerHTML = cruiseOn;

// starting speed for cruise control
if (cruiseOn) {
  cruiseStart = speed;
} else {cruiseStart = 0}
document.getElementById("startingSpeed").innerHTML = cruiseStart;

// Cruising speed
document.getElementById("cruisingSpeed").innerHTML = cruiseStart;

// distance from left lane line
```

```
  leftLineDist = Math.floor(Math.random() * 30);
  document.getElementById("distanceLeftLine").innerHTML = leftLineDist; //
do not change.

  // distance from right lane line
  rightLineDist = Math.floor(Math.random() * 30);
  document.getElementById("distanceRightLine").innerHTML = rightLineDist;
// do not change.

  // var redLight = false;
  signIndex = Math.floor(Math.random() * 5);
  if (signIndex == 1){
    whatSignal = "No nearby traffic signal";
    document.getElementById("whatSign").innerHTML = whatSignal;
  }
  if (signIndex == 2){
    whatSignal = "Red Light";
    document.getElementById("whatSign").innerHTML = whatSignal;
  }
  if (signIndex == 3){
    whatSignal = "Stop Sign";
    document.getElementById("whatSign").innerHTML = whatSignal;
  }
  if (signIndex == 4){
    whatSignal = "All Way Stop Sign";
    document.getElementById("whatSign").innerHTML = whatSignal;
  }
  if (signIndex == 5){
    whatSignal = "Yield Sign";
    document.getElementById("whatSign").innerHTML = whatSignal;
  }

  throwWarnings();
  putInTable();
}

function putInTable() {
  let table = document.getElementById("Logs");
  let row = table.insertRow();
```

```
  var today = new Date();

  let dateCell = row.insertCell(0);
  dateCell.innerHTML =
    today.getMonth() + 1 + "-" + today.getDate() + "-" +
today.getFullYear();

  let timeCell = row.insertCell(1);
  timeCell.innerHTML =
    today.getHours() + ":" + today.getMinutes() + ":" +
today.getSeconds();

  let speedCell = row.insertCell(2);
  speedCell.innerHTML = speed;

  let rpmCell = row.insertCell(3);
  rpmCell.innerHTML = rpm;

  let engTCell = row.insertCell(4);
  engTCell.innerHTML = btemp;

  //battery level
  let outTCell = row.insertCell(5);
  outTCell.innerHTML = otemp;

  let itempCell = row.insertCell(6);
  itempCell.innerHTML = itemp;

  let etempCell = row.insertCell(7);
  etemp.innerHTML = etemp;

  let wspeedCell = row.insertCell(8);
  wspeedCell.innerHTML = wspeed;

  let humidCell = row.insertCell(9);
  humidCell.innerHTML = humid;

  let visibCell = row.insertCell(10);
```

```
  visibCell.innerHTML = visib;

  let frontProxCell = row.insertCell(11);
  frontProxCell.innerHTML = frontProx;

  let rearProxCell = row.insertCell(12);
  rearProxCell.innerHTML = rearProx;

  let blindSCell = row.insertCell(13);
  blindSCell.innerHTML = blindS;

  let cruiseStartCell = row.insertCell(14);
  cruiseStartCell.innerHTML = cruiseStart;

  let cruiseSpeedCell = row.insertCell(15);
  cruiseSpeedCell.innerHTML = cruiseStart;

  let cruiseOnCell = row.insertCell(16);
  cruiseOnCell.innerHTML = cruiseOn;

  let leftLineCell = row.insertCell(17);
  leftLineCell.innerHTML = leftLineDist;

  let rightLineCell = row.insertCell(18);
  rightLineCell.innerHTML = rightLineDist;

  let whatSignalCell = row.insertCell(19);
  whatSignalCell.innerHTML = whatSignal;

  let condCell = row.insertCell(20);
  condCell.innerHTML = "Admin";
}

function throwWarnings() {
  if (speed < 80) {
    document.getElementById("speedingAlert").innerHTML =
        "SPEED OK";
```

```
    document.getElementById("speedingAlert").style.backgroundColor =
"green";
    document.getElementById("speedingAlert").style.color = "white";
    document.getElementById("speedingAlert").style.padding =
      "10px 100px 10px 100px";
  }
  if (speed > 80) {
    document.getElementById("speedingAlert").innerHTML =
        "SPEED WARNING";

    document.getElementById("speedingAlert").style.backgroundColor =
"yellow";
    document.getElementById("speedingAlert").style.color = "black";
    document.getElementById("speedingAlert").style.padding =
      "10px 100px 10px 100px";
  }
  if (speed > 90) {
      document.getElementById("speedingAlert").innerHTML =
          "SPEEDING";

    document.getElementById("speedingAlert").style.backgroundColor =
"red";
    document.getElementById("speedingAlert").style.color = "black";
    document.getElementById("speedingAlert").style.padding =
      "10px 100px 10px 100px";
  }

  if (etemp < 200) {
      document.getElementById("tempAlert").innerHTML =
          "GREEN NO ALERT ENGINE OK";

    document.getElementById("tempAlert").style.backgroundColor = "green";
    document.getElementById("tempAlert").style.color = "white";
    document.getElementById("tempAlert").style.padding =
      "10px 100px 10px 100px";
  }
  if (etemp > 200) {
      document.getElementById("tempAlert").innerHTML =
          "YELLOW ALERT ENGINE TMP";
```

```
  document.getElementById("tempAlert").style.backgroundColor = "yellow";
  document.getElementById("tempAlert").style.color = "black";
  document.getElementById("tempAlert").style.padding =
    "10px 100px 10px 100px";
}
if (etemp > 250) {
    document.getElementById("tempAlert").innerHTML =
        "RED ALERT ENGINE TMP";

  document.getElementById("tempAlert").style.backgroundColor = "red";
  document.getElementById("tempAlert").style.color = "black";
  document.getElementById("tempAlert").style.padding =
    "10px 100px 10px 100px";
}
if (whatSignal == "No nearby traffic signal") {
  document.getElementById("signAlert").innerHTML =
      "NO SIGNS";

document.getElementById("signAlert").style.backgroundColor = "green";
document.getElementById("signAlert").style.color = "white";
document.getElementById("signAlert").style.padding =
  "10px 100px 10px 100px";
}
if (whatSignal == "Red Light"){
  document.getElementById("signAlert").innerHTML =
      "RED LIGHT";

document.getElementById("signAlert").style.backgroundColor = "red";
document.getElementById("signAlert").style.color = "black";
document.getElementById("signAlert").style.padding =
  "10px 100px 10px 100px";
}
if (whatSignal == "Stop Sign"){
  document.getElementById("signAlert").innerHTML =
      "STOP";

document.getElementById("signAlert").style.backgroundColor = "red";
document.getElementById("signAlert").style.color = "black";
```

```
document.getElementById("signAlert").style.padding =
  "10px 100px 10px 100px";
}
if (whatSignal == "All Way Stop Sign"){
  document.getElementById("signAlert").innerHTML =
      "STOP";

document.getElementById("signAlert").style.backgroundColor = "red";
document.getElementById("signAlert").style.color = "black";
document.getElementById("signAlert").style.padding =
  "10px 100px 10px 100px";
}
if (whatSignal == "Yield Sign"){
  document.getElementById("signAlert").innerHTML =
      "YIELD";

document.getElementById("signAlert").style.backgroundColor = "yellow";
document.getElementById("signAlert").style.color = "black";
document.getElementById("signAlert").style.padding =
  "10px 100px 10px 100px";
}
if((speed > (actualSpeed + 5)) || (speed < (actualSpeed -5))){
  document.getElementById("tractAlert").innerHTML =
      "TRACTION LOSS";

document.getElementById("tractAlert").style.backgroundColor = "red";
document.getElementById("tractAlert").style.color = "black";
document.getElementById("tractAlert").style.padding =
  "10px 100px 10px 100px";
}
if(rightLineDist > 45 || leftLineDist >45){
  document.getElementById("driftAlert").innerHTML =
      "DRIFTING";

document.getElementById("driftAlert").style.backgroundColor = "red";
document.getElementById("driftAlert").style.color = "black";
document.getElementById("driftAlert").style.padding =
  "10px 100px 10px 100px";
}
```

```
if(rightLineDist <= 45 && leftLineDist <=45){
  document.getElementById("driftAlert").innerHTML =
      "NOT DRIFTING";

document.getElementById("driftAlert").style.backgroundColor = "green";
document.getElementById("driftAlert").style.color = "white";
document.getElementById("driftAlert").style.padding =
  "10px 100px 10px 100px";
}
if(blindS == 1){
  document.getElementById("blindAlert").innerHTML =
      "OBJECT IN BLINDSPOT";

document.getElementById("blindAlert").style.backgroundColor = "red";
document.getElementById("blindAlert").style.color = "black";
document.getElementById("blindAlert").style.padding =
  "10px 100px 10px 100px";
}
if(blindS == 0){
  document.getElementById("blindAlert").innerHTML =
      "NO OBJECT IN BLINDSPOT";

document.getElementById("blindAlert").style.backgroundColor = "green";
document.getElementById("blindAlert").style.color = "white";
document.getElementById("blindAlert").style.padding =
  "10px 100px 10px 100px";
}
if(cruiseOn == 0){
  document.getElementById("cruiseAlert").innerHTML =
      "CRUISE OFF";

document.getElementById("cruiseAlert").style.backgroundColor = "red";
document.getElementById("cruiseAlert").style.color = "black";
document.getElementById("cruiseAlert").style.padding =
  "10px 100px 10px 100px";
}
if(cruiseOn == 1){
  document.getElementById("cruiseAlert").innerHTML =
      "CRUISE ON";
```

```
  document.getElementById("cruiseAlert").style.backgroundColor = "green";
  document.getElementById("cruiseAlert").style.color = "white";
  document.getElementById("cruiseAlert").style.padding =
    "10px 100px 10px 100px";
  }

  let isMoving = Math.floor(Math.random() * 2);
  let objectDist = Math.floor(Math.random() * 50);
  if (isMoving == 1) {
    // object IS moving
    if (objectDist <= 10) {
      document.getElementById("objectAlert").innerHTML =
          "COLLISION";

      document.getElementById("objectAlert").style.backgroundColor =
"red";
      document.getElementById("objectAlert").style.color = "black";
    } else if (objectDist <= 30) {
      document.getElementById("objectAlert").innerHTML =
          "OBJECT APPROACHING";

      document.getElementById("objectAlert").style.backgroundColor =
"yellow";
      document.getElementById("objectAlert").style.color = "black";
    } else {
      document.getElementById("objectAlert").innerHTML =
          "NO OBJECT";

      document.getElementById("objectAlert").style.backgroundColor =
"green";
      document.getElementById("objectAlert").style.color = "white";
    }
  } else {
    // object is NOT moving
    if (objectDist <= 5) {
      document.getElementById("objectAlert").innerHTML =
          "COLLISION";
```

```
      document.getElementById("objectAlert").style.backgroundColor =
"red";
      document.getElementById("objectAlert").style.color = "black";
    } else if (objectDist <= 10) {
      document.getElementById("objectAlert").innerHTML =
          "OBJECT NEAR";

      document.getElementById("objectAlert").style.backgroundColor =
"yellow";
      document.getElementById("objectAlert").style.color = "black";
    } else {
      document.getElementById("objectAlert").innerHTML =
          "NO OBJECT";

      document.getElementById("objectAlert").style.backgroundColor =
"green";
      document.getElementById("objectAlert").style.color = "white";
    }
  }
}
```

```
<!DOCTYPE html>
<html>
<title>IoT Details</title>

<head>
  <script src="main.js"></script>
</head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Montserrat">
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-a
wesome.min.css">
<!-- Navbar on small screens -->
<div>
```

```
  <a href="login.html">LOGOUT</a>
</div>
<!-- MAIN SECTION -->
<div id="Dashboard">
  <h2>DASHBOARD</h2>
</div>
<section>
  <div></div>
  <span id="carSpeed">0</span> MPH<br>Speed</div>
  <div><span id="rpm">0</span> RPM<br>RPM</div>
  <div><span id="batteryLevel">0</span> %<br>Battery Level</div>
  <div><span id="batteryTemp">0</span> °F<br>Battery Temperature</div>
  <div><span id="calcSpeed">0</span> MPH<br>Speed from RPM</div>
</section>
<div id="driveMode">
  <h2>DRIVING MODE</h2>
</div>
<section>
  <div></div>
  <span id="drivingMode">Driver Assisted</span></div>
</section>
<div id="Climate">
  <h2>CLIMATE CONDITIONS</h2>
</div>
<section>
  <div></div>
  <span id="internalTemp">0</span> °F<br>Internal Temperature</div>
  <div><span id="externalTemp">0</span> °F<br>External Temperature</div>
  <div><span id="windSpeed">0</span> MPH<br>Wind Speed</div>
  <div><span id="humidity">0</span> %<br>Humidity</div>
  <div><span id="visibility">0</span> %<br>Visibility</div>
</section>
<div id="CollisionPrediction">
  <h2>COLLISION PREDICTION</h2>
</div>
<section>
  <div></div>
  <span id="frontDistance">20</span> ft<br>Front Proximity</div>
  <div><span id="rearDistance">20</span> ft<br>Rear Proximity</div>
```

```
  <div><span id="blindSpot">No</span> <br>Blindspot</div>
</section>
<div style="margin: 3rem" class="-center">
  <span id="tractAlert"
    style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bott
om:10px;">Traction
    Status: --</span>
</div>
<div style="margin: 3rem" class="-center">
  <span id="blindAlert"
    style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bott
om:10px;">BlindSpot
    Status: --</span>
</div>
<div style="margin: 3rem" class="-center">
  <span id="speedingAlert"
    style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bott
om:10px;">Speed
    Status: --</span>
</div>
<div id="CruiseControl">
  <h2>CRUISE CONTROL</h2>
</div>
<section>
  <div></div>
  <span id="startingSpeed">60</span> MPH<br>Starting Speed</div>
  <div><span id="cruisingSpeed">65</span> MPH<br>Cruising Speed</div>
  <div><span id="cruiseControlStatus">Yes</span><br>Cruise Control
On</div>
</section>
<div style="margin: 3rem" class="-center">
  <span id="cruiseAlert"
    style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bott
om:10px;">Cruise
    Status: --</span>
```

```
</div>
<div id="LaneMitigation">
  <h2>LANE MITIGATION</h2>
</div>
<section>
  <div></div>
  <span id="distanceLeftLine">20</span> in<br>Distance from Left
Line</div>
  <div><span id="distanceRightLine">20</span> in<br>Distance from Right
Line</div>
</section>
<div style="margin: 3rem" class="-center">
  <span id="driftAlert"
    style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bott
om:10px;">Drifting
    Status: --</span>
</div>
<div id="SignRecognition">
  <h2>TRAFFIC LIGHT AND SIGN RECOGNITION</h2>
</div>
<section>
  <div></div>
  <span id="whatSign">Red Light</span><br>Traffic Signal</div>
</section>
<br>
<div style="margin: 3rem" class="-center">
  <span id="tempAlert"
    style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bott
om:10px;">Temperature
    Status: --</span>
</div>
<div style="margin: 3rem" class="-center">
  <span id="objectAlert"
    style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bott
om:10px;">Object
    Status: --</span>
```

```
</div>
<div style="margin: 3rem" class="-center">
  <span id="signAlert"
    style="background-color:
green;padding-left:100px;padding-right:100px;padding-top:10px;padding-bott
om:10px;">Sign
    Status: --</span>
</div>
<div class="-center" style="padding-bottom:15px;">
  <button type="submit" onclick="getRandInput()" class="nicebutton">
TESTING </button>
</div>
<!-- STYSTEM LOGS SECTION -->
<div>
  <h2>SYSTEM LOGS</h2>
  <table id="Logs" class="syslogs">
    <tr>
      <th>Date</th>
      <th>Time</th>
      <th>Speed</th>
      <th>RPM</th>
      <th>Bat. Temp.</th>
      <th>Bat. Level</th>
      <th>Int. Temp</th>
      <th>Ext. Temp</th>
      <th>Wind Speed</th>
      <th>Humidity</th>
      <th>Fog Visibility</th>
      <th>Front Prox.</th>
      <th>Rear Prox.</th>
      <th>Blind Spot?</th>
      <th>CC Start Speed</th>
      <th>CC Speed</th>
      <th>CC On?</th>
      <th>Left Line Dist.</th>
      <th>Right Line Dist.</th>
      <th>Sign?</th>
      <th>Conductor</th>
    </tr>
```

```
    <tr>
      <td>MM-DD-YYYY</td>
      <td>HH:MM:SS</td>
      <td>MPH</td>
      <td>RPM</td>
      <td>°F</td>
      <td>%</td>
      <td>°F</td>
      <td>°F</td>
      <td>MPH</td>
      <td>%</td>
      <td>%</td>
      <td>ft</td>
      <td>ft</td>
      <td>T/F</td>
      <td>MPH</td>
      <td>MPH</td>
      <td>T/F</td>
      <td>in</td>
      <td>in</td>
      <td>Sign</td>
      <td>Mechanic</td>
    </tr>
  </table>
  <br>
</div>
</div>
<footer>
  Drive Smarter, with ALSET
</footer>
</body>

</html>
```

Section 6.2: Login

```
<!DOCTYPE html>
<html>

<head>
  <title>W3.CSS Template</title>
  <script src="main.js"></script>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="style.css">
  <link rel="stylesheet"
href="https://fonts.googleapis.com/css?family=Inconsolata">

  <style>
    body,
    html {
      height: 100%;
      font-family: "Inconsolata", sans-serif;
    }

    .bgimg {
      background-position: center;
      background-size: cover;
      background-image:
url("https://mediapool.bmwgroup.com/cache/P9/202202/P90450623/P90450623-bm
w-super-bowl-spot-2022-2396px.jpg");
      min-height: 75%;
    }
  </style>
</head>

<body>

  <!-- Links (sit on top) -->
  <div class="w3-top">
    <div class="w3-row w3-padding w3-black">
      <div class="w3-col s3">
        <a href="#" class="w3-button w3-block w3-black">LOG IN</a>
      </div>
```

```
      <div class="w3-col s3">
        <a href="#about" class="w3-button w3-block w3-black">ABOUT</a>
      </div>
      <div class="w3-col s3">
        <a href="#troubleshoot" class="w3-button w3-block
w3-black">TROUBLESHOOT</a>
      </div>
      <div class="w3-col s3"> </div>
    </div>
  </div>

  <!-- Header with image -->
  <header class="bgimg w3-display-container w3-grayscale-min" id="home">
    <div class="w3-display-bottomleft w3-center w3-padding-large
w3-hide-small">
      <span class="w3-tag">Drive Smarter, with ALSET.</span>
    </div>
    <div class="w3-display-middle w3-center">
      <form name="login" style="font-size: 25px;">
        <h3>ALSET Log In</h3>

        <label for="username" style="padding-bottom:10px;"></label>
        <input type="text" placeholder="username" id="username"
name="userid">
        <br>
        <label for="password"></label>
        <input type="password" placeholder="Password" id="password"
name="password">

        <div>
          <button type="reset" onclick="check(this.form)"> Login </button>
        </div>
      </form>
    </div>
  </header>

  <!-- Add a background color and large text to the whole page -->
  <div class="w3-sand w3-grayscale w3-large">
```

```
    <!-- About Container -->
    <div class="w3-container" id="about">
      <div class="w3-content" style="max-width:700px">
        <h5 class="w3-center w3-padding-64"><span class="w3-tag
w3-wide">ABOUT</span></h5>
        <p>lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore
          magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo
          consequat</p>
        <p>lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore
          magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo
          consequat</p>
        <div class="w3-panel w3-leftbar w3-light-grey">
          <p><i>mission statement here lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod tempor
              incididunt ut labore et dolore magna aliqua.</i></p>
          <p>-Jessica Noel, Jack Chen, Sophia Roper</p>
        </div>
      </div>

      <!-- Contact/Area Container -->
      <div class="w3-container" id="troubleshoot"
style="padding-bottom:30px;">

        <p><strong>TroubleShoot Services</strong></p>
        <form action="/action_page.php" target="_blank">
          <p><input class="w3-input w3-padding-16 w3-border" type="text"
placeholder="Name" required name="Name"></p>
          <p><input class="w3-input w3-padding-16 w3-border" type="text"
placeholder="Message \ Special requirements"
              required name="Message"></p>
          <p><button class="w3-button w3-black" type="submit">SEND
MESSAGE</button></p>
        </form>
      </div>
    </div>
```

```
    <!-- End page content -->
  </div>
</body>

</html>
```

Section 7: Testing

Tests are generated by the getRandInput() function in the code.

8	TEST LOG	5/13/2022	20:34:22	17	992	169	98	53	63	1	0	42	2	1	TRUE	17	17	TRUE	0	12	RED LIGHT
	EXPECTED	5/13/2022	20:34:22	NO ALERT, ENGINE OK						TRACTION LOSS				COLLISION	COLLISION!!	SPEED OK	SPEED OK	CRUISE ON	NOT DRIFTING	NOT DRIFTING	RED LIGHT
	RESULT	5/13/2022	20:34:22	NO ALERT, ENGINE OK						TRACTION LOSS				COLLISION	COLLISION!!	SPEED OK	SPEED OK	CRUISE ON	NOT DRIFTING	NOT DRIFTING	RED LIGHT
9	TEST LOG	5/13/2022	20:37:45	89	982	28	59	78	39	9	78	32	3	15	FALSE	89	89	TRUE	15	13	RED LIGHT
	EXPECTED	5/13/2022	20:37:45	NO ALERT, ENGINE OK						TRACTION LOSS				NO OBJECT NEAR	NO OBJECT IN BLINDSPOT	SPEED WARNING	SPEED WARNING	CRUISE ON	NOT DRIFTING	NOT DRIFTING	RED LIGHT
	RESULT	5/13/2022	20:37:45	NO ALERT, ENGINE OK						TRACTION LOSS				NO OBJECT NEAR	NO OBJECT IN BLINDSPOT	SPEED WARNING	SPEED WARNING	CRUISE ON	NOT DRIFTING	NOT DRIFTING	RED LIGHT
10	TEST LOG	5/13/2022	20:39:35	1	174	91	76	38	48	0	76	3	16	7	TRUE	1	1	TRUE	8	21	RED LIGHT
	EXPECTED	5/13/2022	20:39:35	NO ALERT, ENGINE OK						TRACTION GOOD					OBJECT IN BLINDSPOT	SPEED OK	SPEED OK	CRUISE ON	NOT DRIFTING	NOT DRIFTING	RED LIGHT
	RESULT	5/13/2022	20:39:35	NO ALERT, ENGINE OK						TRACTION GOOD					OBJECT IN BLINDSPOT	SPEED OK	SPEED OK	CRUISE ON	NOT DRIFTING	NOT DRIFTING	RED LIGHT

Test	RESULTS	Date	Time	Speed (MPH)	RPM	Battery Temp (°F)	Battery Level (%)	Internal Temp °F	External Temp °F	Wind (MPH)	Humidity (%)	Fog Visiblity (%)	Front Proximity (Ft.)	Rear Proximity (Ft.)	Blind Spot? (Boolean)	CC Start (MPH)	CC Speed (MPH)	CC On? (Boolean)	Left Line Dist. (Ft.)	Right Line Dist. (Ft.)	Sign?
1	TEST LOG	4/29/2022	20:58:33	23	704	131	87	79	67	4	34	24	16	1	FALSE	23	23	TRUE	16	17	No Sign
	EXPECTED	4/29/2022	20:58:33	NO ALERT, ENGINE OK						TRACTION LOSS				COLLISION	NO OBJECT IN BLINDSPOT	SPEED OK	SPEED OK	CRUISE CONTROL ON	NOT DRIFTING	NOT DRIFTING	GREEN SIGN
	RESULT	4/29/2022	20:58:33	NO ALERT, ENGINE OK						TRACTION LOSS				COLLISION	NO OBJECT IN BLINDSPOT	SPEED OK	SPEED OK	CRUISE CONTROL ON	NOT DRIFTING	NOT DRIFTING	GREEN SIGN
2	TEST LOG	5/1/2022	16:13:30	73	23	69	12	62	34	7	26	10	7	17	TRUE	73	73	TRUE	11	4	RED LIGHT
	EXPECTED	5/1/2022	16:13:30	NO ALERT, ENGINE OK						TRACTION GOOD					COLLISION!!	SPEED OK	SPEED OK	CRUISE CONTROL ON	NOT DRIFTING	NOT DRIFTING	RED LIGHT
	RESULT	5/1/2022	16:13:30	NO ALERT, ENGINE OK						TRACTION GOOD					COLLISION!!	SPEED OK	SPEED OK	CRUISE CONTROL ON	NOT DRIFTING	NOT DRIFTING	RED LIGHT
3	TEST LOG	5/5/2022	12:01:10	37	158	158	26	41	51	9	2	95	8	16	TRUE	37	37	TRUE	24	11	RED LIGHT
	EXPECTED	5/5/2022	12:01:10	NO ALERT, ENGINE OK						TRACTION GOOD					COLLISION!!	SPEED OK	SPEED OK	CRUISE CONTROL ON	NOT DRIFTING	NOT DRIFTING	RED LIGHT
	RESULT	5/5/2022	12:01:10	NO ALERT, ENGINE OK						TRACTION GOOD					COLLISION!!	SPEED OK	SPEED OK	CRUISE CONTROL ON	NOT DRIFTING	NOT DRIFTING	RED LIGHT
4	TEST LOG	5/6/2022	16:45:50	71	736	100	37	75	29	1	33	39	13	0	FALSE	0	0	FALSE	9	17	NO NEARBY SIGN
	EXPECTED	5/6/2022	16:45:50	NO ALERT, ENGINE OK						TRACTION LOSS				OBJECT NEAR	NO OBJECT IN BLINDSPOT	SPEED OK	SPEED OK	CRUISE OFF	NOT DRIFTING	NOT DRIFTING	NO SIGNS
	RESULT	5/6/2022	16:45:50	NO ALERT, ENGINE OK						TRACTION LOSS				OBJECT NEAR	NO OBJECT IN BLINDSPOT	SPEED OK	SPEED OK	CRUISE OFF	NOT DRIFTING	NOT DRIFTING	NO SIGNS
5	TEST LOG	5/9/2022	19:21:39	55	974	53	70	50	46	2	42	72	17	3	TRUE	0	0	FALSE	14	16	RED LIGHT
	EXPECTED	5/9/2022	19:21:39	NO ALERT, ENGINE OK						TRACTION LOSS				NO OBJECT NEAR	OBJECT IN BLINDSPOT	SPEED OK	SPEED OK	CRUISE OFF	NOT DRIFTING	NOT DRIFTING	RED LIGHT
	RESULT	5/9/2022	19:21:39	NO ALERT, ENGINE OK						TRACTION LOSS				NO OBJECT NEAR	OBJECT IN BLINDSPOT	SPEED OK	SPEED OK	CRUISE OFF	NOT DRIFTING	NOT DRIFTING	RED LIGHT
6	TEST LOG	5/10/2022	10:54:00	89	97	88	35	44	54	10	70	65	4	4	FALSE	89	89	TRUE	18	1	ALL WAY STOP
	EXPECTED	5/10/2022	10:54:00	NO ALERT, ENGINE OK						TRACTION GOOD				OBJECT APPROACHING	NO OBJECT IN BLINDSPOT	SPEED WARNING	SPEED WARNING	CRUISE ON	NOT DRIFTING	NOT DRIFTING	STOP
	RESULT	5/10/2022	10:54:00	NO ALERT, ENGINE OK						TRACTION GOOD				OBJECT APPROACHING	NO OBJECT IN BLINDSPOT	SPEED WARNING	SPEED WARNING	CRUISE ON	NOT DRIFTING	NOT DRIFTING	STOP
7	TEST LOG	5/10/2022	10:58:59	64	873	109	21	27	37	1	97	52	7	7	FALSE	0	0	FALSE	14	3	STOP SIGN
	EXPECTED	5/10/2022	10:58:59	NO ALERT, ENGINE OK						TRACTION LOSS				NO OBJECT NEAR	NO OBJECT IN BLINDSPOT	SPEED OK	SPEED OK	CRUISE OFF	NOT DRIFTING	NOT DRIFTING	STOP
	RESULT	5/10/2022	10:58:59	NO ALERT, ENGINE OK						TRACTION LOSS				NO OBJECT NEAR	NO OBJECT IN BLINDSPOT	SPEED OK	SPEED OK	CRUISE OFF	NOT DRIFTING	NOT DRIFTING	STOP

7.2 TEST FILE/CODE WE USED FOR TESTING SOFTWARE

SPEED

```
if (speed < 80) {
    document.getElementById("speedingAlert").innerHTML =
        "SPEED OK";
if (speed > 80) {
    document.getElementById("speedingAlert").innerHTML =
        "SPEED WARNING";
if (speed > 90) {
      document.getElementById("speedingAlert").innerHTML =
          "SPEEDING";
```

ENGINE TEMPERATURE

```
  if (etemp < 200) {
      document.getElementById("tempAlert").innerHTML =
          "GREEN: NO ALERT, ENGINE OK";
  }
  if (etemp > 200) {
      document.getElementById("tempAlert").innerHTML =
          "YELLOW ALERT ENGINE TMP";
  }
  if (etemp > 250) {
      document.getElementById("tempAlert").innerHTML =
          "RED ALERT ENGINE TMP";
```

TRAFFIC SIGNS

```
if (whatSignal == "No nearby traffic signal") {
    document.getElementById("signAlert").innerHTML =
        "NO SIGNS";
  }
  if (whatSignal == "Red Light"){
    document.getElementById("signAlert").innerHTML =
        "RED LIGHT";
  }
  if (whatSignal == "Stop Sign"){
```

```
    document.getElementById("signAlert").innerHTML =
        "STOP";
  }
  if (whatSignal == "All Way Stop Sign"){
    document.getElementById("signAlert").innerHTML =
        "STOP";
  }
  if (whatSignal == "Yield Sign"){
    document.getElementById("signAlert").innerHTML =
        "YIELD";
  }
```

RPM

```
if(rpm > 500){
    document.getElementById("tractAlert").innerHTML =
        "TRACTION LOSS";
  }
  else {
    document.getElementById("tractAlert").innerHTML =
        "NO TRACTION LOSS";
  }
  if(rightLineDist > 45 || leftLineDist >45){
    document.getElementById("driftAlert").innerHTML =
        "DRIFTING";
  }
  if(rightLineDist <= 45 && leftLineDist <=45){
    document.getElementById("driftAlert").innerHTML =
        "NOT DRIFTING";
  }
```

BLINDSPOT

```
 if(blindS == 1){
    document.getElementById("blindAlert").innerHTML =
        "OBJECT IN BLINDSPOT";
  }
```

```
if(blindS == 0){
  document.getElementById("blindAlert").innerHTML =
      "NO OBJECT IN BLINDSPOT";
}
```

CRUISE CONTROL ACTIVATED?

```
if(cruiseOn == 0){
  document.getElementById("cruiseAlert").innerHTML =
      "CRUISE OFF";
}
if(cruiseOn == 1){
  document.getElementById("cruiseAlert").innerHTML =
      "CRUISE ON";
}
```

COLLISION DETECTOR

```
let isMoving = Math.floor(Math.random() * 2);
 let objectDist = Math.floor(Math.random() * 50);
 if (isMoving == 1) {
   // object IS moving
   if (objectDist <= 10) {
     document.getElementById("objectAlert").innerHTML =
         "COLLISION";
   } else if (objectDist <= 30) {
     document.getElementById("objectAlert").innerHTML =
         "OBJECT APPROACHING";
   } else {
     document.getElementById("objectAlert").innerHTML =
         "NO OBJECT";
   }
 } else {
   // object is NOT moving
   if (objectDist <= 5) {
```

```
      document.getElementById("objectAlert").innerHTML =
          "COLLISION";
    } else if (objectDist <= 10) {
      document.getElementById("objectAlert").innerHTML =
          "OBJECT NEAR";
    } else {
      document.getElementById("objectAlert").innerHTML =
          "NO OBJECT";
    }
  }
```

Citations

[1] Serban, Alexandru Constantin, Erik Poll, and Joost Visser. "A standard driven software architecture for fully autonomous vehicles." *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2018.
[2] Jo, Kichun, et al. "Development of autonomous car—Part I: Distributed system architecture and development process." *IEEE Transactions on Industrial Electronics* 61.12 (2014): 7131-7140.
[3] Kukkala, Vipin Kumar, et al. "Advanced driver-assistance systems: A path toward autonomous vehicles." *IEEE Consumer Electronics Magazine* 7.5 (2018): 18-25.