

Hug The Lanes IoT Project

Stevens Institute of Technology

Team SWD

Team Leader:

Matthew Yap, myap2@stevens.edu

Students:

Malak Abdelhakim, mabdelha@stevens.edu

Isabella Baratta, ibaratta@stevens.edu

Joshua Prasad, jprasad2@stevens.edu

Date: 01/31/2024

Table of Contents

Section 1: Introduction	3
• Section 1.1: IoT Architecture	3
• Section 1.2: Features	3
• Section 1.3: Software Development Process	4
• Section 1.4: Project Team	4
Section 2: Functional Architecture	5
• Section 2.1: Functional Architecture Application	6
• Section 2.2: Module Functionality	6
• Section 2.3: Architecture Validity	7
• Section 2.4: Functional Distribution	8
Section 3: Requirements	10
• Section 3.1: Functional Requirements	10
• Section 3.2: Non-Functional Requirements	19
Section 4: Requirement Modeling	20
• Section 4.1: Use Case Scenarios	20
• Section 4.2 Activity Diagrams	24
• Section 4.3: Sequence Diagrams	28
• Section 4.4: Classes	31
• Section 4.5: State Diagrams	34
Section 5: Design	41
• Section 5.1: Software Architecture	41
• Section 5.2: Interface Design	44
• Section 5.3: Component Level Design	46
Section 6: Code	47
• Section 6.1: Driver Interface	48
• Section 6.2 Technician Interface	49
Section 7: Testing	52
• Section 7.1: Validation Testing	52
• Section 7.2 Scenario Testing	53

Section 1: Introduction

Section 1.1: IoT Architecture

IoT stands for the Internet of Things and it's used to describe physical devices that are embedded with sensors and technologies for the purpose of exchanging data and information with other devices over the internet. IoT has transformed the way we interact with physical objects because now we have the ability to control them through the digital world. IoT has been made possible by the recent access to cost-effective, low-power sensor technology, artificial intelligence and machine learning, cloud computing, etc.

We plan on using IoT to create a new self-driving car of the future. This car will include several features such as: the ability for the driver to enter commands and receive status on the execution, capture data from both the car and the environment, and option to allow the car to make decisions and provide information to the driver. We will be relying on IoT to be the most advanced architecture for this product.

Section 1.2: Features

The implementation of a new self-driving car can warrant some skepticism as malfunctions can result in dangerous accidents, possibly resulting in the injury or death of the driver, passengers, pedestrians, and anyone else caught in the accident. As a result, it will be imperative to develop a reliable product that is highly immune to any sort of malfunction rooted from insufficient testing. We do acknowledge the nature of this project as a mission critical real-time embedded system. As such, it is in our best interest to keep the customer and the people on the road safe and away from danger, hence the availability and reliability of it.

Section 1.3: Software Development Process

To complete this project successfully, properly, and efficiently we will be following a waterfall model of the software development process. Seeing as this project has very clear requirements it will be easy to establish and communicate exactly what is desired by customers and stakeholders. After planning and modeling it will be clear to see the path of the project before construction and easy to communicate with the customer to see if this is/will be what they are looking for.

We will be using an incremental release process, due to the likelihood of flaws within the first release of the project. After performing the first version, there will be following updates to further build and refine our project to become a more complete product.

Section 1.4: Project Team

Our team is comprised of four members who's qualifications and experience varies in order to provide the most well-rounded team. Isabella is task-oriented, patient, and detail-oriented; Joshua is fluent in coding, has strong writing skills, and is reserved; Matthew is outgoing, has strong communication skills, and is task-oriented; Malak has strong project management skills, is patient, and is good at problem solving. Each member is knowledgeable about a wide array of programming languages and development techniques in order to ensure the success and reliability of this critical-natured project.

Section 2: Functional Architecture

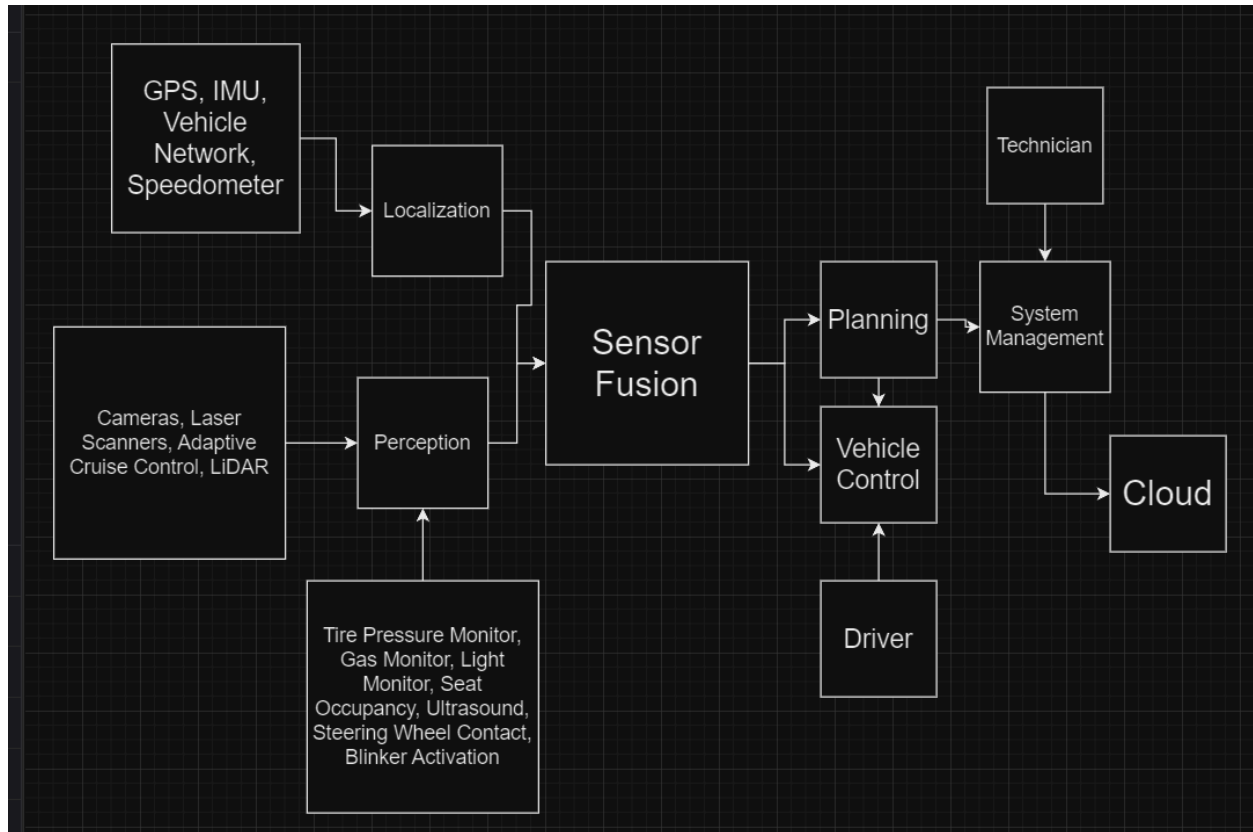


Figure 1 - Functional architecture of the self-driving car

Section 2.1: Functional Architecture Application

The functional architecture of our self-driving car of the future integrates different IoT technologies to allow for more efficient and faster communication between the car and its external surroundings. Integrating functional architecture into the self-driving car

allows us to split the system into components and subcomponents. This establishes clarity of the purpose that each piece serves in the whole implementation of a self-driving car, in addition to the data that flows in between them.

Section 2.2: Module Functionality

The key components of this architecture would be sensors and actuators, a data processing unit, a user interface (UI), as well as modules for communication and decision-making. The sensors to be included in the car are cameras, GPS, ultrasonic sensors, radar, etc, and their key role is to be the car's eyes in a sense that they will capture data of the surrounding environment, such as obstacles, traffic signs, road conditions, pedestrians, and so on.

Actuators, such as motors, would translate the decisions made due to the surroundings into actions that the car can take, such as braking, accelerating, or steering. These decisions are made by algorithms that act upon information about the surroundings and data in order to optimize safety and efficiency.

The data processing unit will make use of artificial intelligence and machine learning to allow the car to process data, act upon the decisions made from processing that information, and then using those past decisions to predict more responsible future actions.

The user interface will serve as the interface between the car and people riding in it, allowing the user(s) to input commands and interact with the car's features overall. The user interface can take on many forms, whether it be touchscreens, mobile applications, or even voice commands. As for the communication module, it will allow communication between the car and other systems, such as other vehicles, cloud servers, mobile devices, etc. It will do this using WiFi for example and will exchange information to allow for a more organized and safer driving experience.

Section 2.3: Architecture Validity

Our AAV (Assisted Autonomous Vehicle) will include different aspects to complement the integration of our project, Driver Assistance, Self-Driving, and a System Admin. Driver Assistance will use GPS and Inertial Measurement Units (IMUs) to determine the vehicle's position and orientation. For example, for lane-assist, it is critical for our system to be able to determine where the lines that differentiate the lanes are to prevent crossover. Furthermore, our system would have to take into account other cars in the vicinity of our AAV, to which we will use a Vehicle Network to create a mapping of other vehicles nearby. Doing so provides the Driver Assistance with the ability to avoid other cars and prevent crashes.

Self-Driving utilizes cameras and other sensors to interpret the environment. This will be necessary for identifying objects, such as walls, cones, or pedestrians. The laser scanners can be used when the cameras are unable to detect objects, possibly due to inclement weather or insufficient lighting. Self-Driving will take over if it detects that the driver has made an error. This will only occur when the AAV has checked Perception for any mistakes. For example, when the driver attempts to change lanes when a car is in the way, the vehicle will check its cameras and scanners for any objects. If the car was in a position where changing lanes would cause an accident, it would prevent the driver from moving over. However, if a proper and swift analysis was done, and the car determined there was enough space to change lanes, it would relinquish control to the driver.

The System Admin is critical in establishing a connection between the network and the cloud to allow the computer system to store and send data over the internet to provide the sensor fusion with quick updates on the road, or to document data about the performance of the AAV. The System Admin logs data for the technician to use, additionally providing updates when older versions become outdated.

Section 2.4: Functional Distribution

In Figure 1 you can see the overall design of the functional architecture. The main components are Localization, Perception, Planning, and Vehicle Control. Perception and Localization are intended to be the components needed to get information about the environment of the vehicle. This information will then be fused by the sensor fusion to create a more accurate depiction of its surroundings. This can use sensors such as cameras, GPS, IMU, and LiDAR. Some of these sensors provide similar data but have different weaknesses. For example, IMU/GPS and LiDAR can both be used to provide localization but IMU/GPS can fail to work under areas such as tunnels. LiDAR can be impacted by rain and dust. LiDAR uses pulsed lasers to measure distances and for the instance of a self-driving car, it is useful to calculate distances of other cars and objects that may be on or near the road. These two components are simply the components used to gather information about the vehicle and its positioning.

Planning and Vehicle Control are the components that take in the information provided by the sensors in Perception and Localization. The information from the sensors are used to make decisions for the vehicle. These two areas consist of all the necessary functions to ensure that when self-driving it is making the proper decision based on provided information. For example, information gathered by LiDAR may show that there is an object 20ft up ahead, possibly a pedestrian, and planning would then make the decision that from that information the vehicle needs to start slowing down.

The supporting components consist of the sensors, the Technician, System Management, the Driver, and the Cloud. The sensors that lead into Perception are fed into the sensor fusion in order to a better understanding of both the physical surroundings of the car, and the conditions of the inside of the car.

Some sensors, for example, the Adaptive Cruise Control, detects whether the Cruise Control button is turned on, if it is, it utilizes this data to maintain a safe distance from other vehicles, automatically adjusting speed as needed. Others may monitor the vehicle's condition, such as Tire Pressure Monitors ensure optimal tire performance or

Gas Monitors keep track of fuel levels to name a few. The sensor fusion will properly take in these inputs from the sensors and formulate a plan in the Planning module to take a specified course of action.

The driver remains an integral part of this system, as with any car. The system will take the driver's safety as its utmost priority when entering the Planning module and ordering commands to the vehicle control system. In these cases, the system will take over the driver's capability of driving the vehicle to ensure the safety of others and the driver.

Cloud connectivity plays a pivotal role, offering access to a wealth of data and computational power for real-time updates, navigation, and system enhancements. System Management serves as the central command, orchestrating the various components and sensors, ensuring they work in concert and respond appropriately to any given situation.

In conjunction, Technicians, with the aid of System Management insights and Cloud analytics, carry out maintenance and updates, keeping the system finely tuned and up-to-date. This intricate architecture, with its multiple layers of technology and human oversight, ensures a seamless and dynamic driving experience, prioritizing safety and reliability at every turn.

Section 3: Requirements

Section 3.1: Functional Requirements

1. Blind Spot Monitoring

1.1. Description

- 1.1.1. The Blind Spot Monitoring system alerts drivers of vehicles that are in their blind spots

1.2. Functional Architecture

- 1.2.1. Input: Data from side-facing cameras and sensors that are monitoring the blind spot areas
- 1.2.2. Processing: Object detection algorithms that determine the presence of vehicles
- 1.2.3. Output: Visible or audible alerts to the driver of the vehicle to indicate the presence of a vehicle

1.3. Pre-Conditions

- 1.3.1. Driver is in the seat
- 1.3.2. Car is in drive
- 1.3.3. Car can be in motion or stationary

1.4. Requirements

- 1.4.1. Sensor Fusion receives data from side-facing cameras and sensors, value of 1 to indicate “vehicle present” in the blind spot
- 1.4.2. Sensor Fusion passes “vehicle present” to planning
- 1.4.3. Planning sends request to vehicle control system to provide an audible and visible cue
- 1.4.4. VCS sends “audible cue” to speakers and “visible cue” to “vehicle present indicator”
- 1.4.5. Speakers emit an audible signal, and “vehicle present indicator” turns on
- 1.4.6. Planning sends all data to System Admin for logging

1.5. Post-Conditions

- 1.5.1. Driver is notified of any vehicles not visible in blind spots

2. Parking Assistance

2.1. Description

- 2.1.1. Assist the driver in parking the vehicle

2.2. Functional Architecture

- 2.2.1. Input: Data from sensors and cameras to detect the intended parking space and to detect any potential obstacles
- 2.2.2. Processing: Trajectory algorithms to determine the optimal parking path
- 2.2.3. Output: In a self-driving case, the result is the car will apply the trajectory to the vehicle control for automated parking maneuvers. In a driver-assisted case, guidance will prompt the driver with audible and visible cues

2.3. Pre-Conditions

- 2.3.1. Driver is in the seat
- 2.3.2. Car is on

2.4. Requirements

- 2.4.1. Driver identifies a parking spot they would like to park in and activates the parking assistance
- 2.4.2. Sensor Fusion receives data from rear, front, and side sensors and cameras, and identifies an eligible parking spot, value of 1 to indicate “open parking spot”
- 2.4.3. Sensor Fusion passes “open parking spot” to planning
- 2.4.4. Planning sends request to vehicle control system to adjust the steering, braking, and acceleration
- 2.4.5. VCS adjusts steering in order to park the car
- 2.4.6. Planning sends all data to System Admin for logging

2.5. Post-Conditions

- 2.5.1. Vehicle will be parked

3. Lane Departure Warning

3.1. Description

- 3.1.1. Lane departure warning helps to keep the driver within its lane by providing alerts and steering assistance

3.2. Functional Architecture

- 3.2.1. Input: Data from cameras or sensors that monitor lane markings and surrounding vehicles
- 3.2.2. Processing: Image processing algorithms to detect lane boundaries and to calculate any potential steering corrections
- 3.2.3. Output: In a self-driving case, signals to the vehicle control to adjust the vehicle's steering controls. In a driver-assisted case, provides an auditory cue that the driver has crossed outside of his lanes

3.3. Pre-Conditions

- 3.3.1. Driver is in the seat
- 3.3.2. Car is on
- 3.3.3. Car is in motion
- 3.3.4. Blinker Activation has not detected blinker

3.4. Requirements

- 3.4.1. Sensor Fusion receives data from side sensors and cameras, value of 1 if the vehicle "leaves lane"
- 3.4.2. Sensor Fusion passes "leaves lane" to planning
- 3.4.3. Planning sends request to vehicle control system to adjust the steering
- 3.4.4. VCS adjusts the steering and sends "audible cue" to speakers
- 3.4.5. Speakers emit audible signal
- 3.4.6. Planning sends all data to System Admin for logging

3.5. Post-Conditions

- 3.5.1. Sends a notification to the driver they are drifting to another lane
- 3.5.2. Steers the car back into designated lane

4. **Collision Avoidance Systems**

4.1. Description

- 4.2. Detects potential collisions with vehicles, pedestrians, or other road objects and initiates an action to avoid collision

4.3. Functional Architecture

- 4.3.1. Input: Data from cameras and sensors that detect objects in the vehicle's vicinity
- 4.3.2. Processing: Collision detection algorithms to determine collision risk and plan any potential vehicle control maneuvers

- 4.3.3. Output: Commands to certain systems depending on the situation, e.g the steering if there is ample room to maneuver, braking if space is limited, or accelerating if lateral movement is restricted, but forward movement is free

4.4. Pre-Conditions

- 4.4.1. Driver is in the seat
- 4.4.2. Car is on
- 4.4.3. Car is in motion
- 4.4.4. Car detects objects through sensors

4.5. Requirements

- 4.5.1. Sensor Fusion receives data from rear, front, and side sensors and cameras, value of 1 if the vehicle “detects obstacle”
- 4.5.2. Sensor Fusion passes “detects obstacle” to planning
- 4.5.3. Planning sends request to vehicle control system to adjust the steering, braking, and acceleration
- 4.5.4. VCS overrides driver input with the steering, braking, and acceleration adjustments
- 4.5.5. Planning sends all data to System Admin for logging

4.6. Post-Conditions

- 4.6.1. Vehicle will take specific maneuvers to avoid collision, including steering, decelerating, or accelerating.

5. **Automatic Emergency Braking**

5.1. Description

- 5.1.1. A system that detects potential collisions and automatically applies brakes to prevent or reduce the damage of any accidents

5.2. Functional Architecture

- 5.2.1. Input: Data from front-facing cameras and sensors to detect any objects in the vehicle’s path
- 5.2.2. Processing: Collision detection algorithms to determine the need for an intervention by the system
- 5.2.3. Output: Similar to the collision avoidance systems, it provides commands to braking systems to override driver input and reduce damage from collisions

5.3. Pre-Conditions

- 5.3.1. Driver is in the seat
- 5.3.2. Car is on
- 5.3.3. Car is in motion
- 5.3.4. Car detects object in front through sensors

5.4. Requirements

- 5.4.1. Sensor Fusion receives data from front sensors and cameras, value of 1 if the vehicle “detects obstacle”
- 5.4.2. Sensor Fusion passes “detects obstacle” to planning
- 5.4.3. Planning sends request to vehicle control system to apply brakes
- 5.4.4. VCS applies breaks
- 5.4.5. Planning sends all data to System Admin for logging

5.5. Post-Conditions

- 5.5.1. Car will brake immediately

6. **Driver Adaptive Cruise Control**

6.1. Description

- 6.1.1. This system automatically adjusts the vehicle’s speed in order to maintain a safe distance from the vehicle ahead

6.2. Functional Architecture

- 6.2.1. Input: Data from forward-facing sensors and cameras, and speedometer readings from the vehicle’s own system
- 6.2.2. Processing: Algorithm to determine the distance between this vehicle and the vehicle in front and to adjust the throttle and brakes
- 6.2.3. Output: Provides commands to the accelerating and braking systems, depending on the distance of the car in front and the speed limit

6.3. Pre-Conditions

- 6.3.1. Driver is in the seat
- 6.3.2. Car is on
- 6.3.3. Car is in motion
- 6.3.4. Driver activates adaptive cruise control

6.4. Requirements

- 6.4.1. Sensor Fusion receives data from speedometer, front sensors and cameras

- 6.4.2. Sensor Fusion continuously passes data to vehicle control system
- 6.4.3. VCS maintains the current speed if safe, and decreases speed if a vehicle ahead comes to within a certain distance
- 6.4.4. Planning sends all data to System Admin for logging
- 6.5. Post-Conditions
 - 6.5.1. Car will automatically take control of the car and adjust speed accordingly

7. Weather and Road Adaptation

- 7.1. Description
 - 7.1.1. Detects weather conditions such as rain and snow as well as road conditions, in case it's wet, icy, or unsafe.
- 7.2. Functional Architecture
 - 7.2.1. Input: Data from cameras, LiDar, GPS, radars, and speed sensors.
 - 7.2.2. Processing: Image processing algorithms to detect potholes and weather checking algorithms to drive accordingly based on weather conditions.
 - 7.2.3. Output: Provides commands to accelerating and braking systems if certain speeds are unsafe, collision avoidance systems, as well as windshield wiper systems when raining and snowing.
- 7.3. Pre-Conditions
 - 7.3.1. Driver is in the seat
 - 7.3.2. Car is on
 - 7.3.3. Car is in motion
 - 7.3.4. Driver activates weather/road adaptation
- 7.4. Requirements
 - 7.4.1. Sensor Fusion receives data from cameras, LiDar, GPS, radar, and speed sensors
 - 7.4.2. Planning determines whether windshield wipers are on, and speed/path of car
 - 7.4.3. VCS changes speed of car/path and windshield wipers to on/off depending on Planning
 - 7.4.4. Planning sends system logs to System Management
- 7.5. Post-Conditions
 - 7.5.1. Car will take safety precautions to ensure safety of the driver in inclement weather

8. Intersection Assistance

8.1. Description

- 8.1.1. Detects lights changing, and other cars in the intersection. Deals with the safety of turning when there is no green arrow or when lights are turning yellow.

8.2. Functional Architecture

- 8.2.1. Input: Data from LiDar, radar, GPS, and cameras
- 8.2.2. Processing: Algorithms that take in images/location from sensors to determine course of action at intersections
- 8.2.3. Output: Provides commands to steering, braking, and acceleration systems depending on traffic lights, other vehicles, and pedestrians

8.3. Pre-Conditions

- 8.3.1. Driver is in the seat
- 8.3.2. Car is on
- 8.3.3. Car is in motion
- 8.3.4. Car detects intersection via sensors

8.4. Requirements

- 8.4.1. Sensor Fusion receives data from cameras, LiDar, and the vehicle network
- 8.4.2. Planning determines when to go, stop, and to turn
- 8.4.3. VCS accelerates, slows down, stops, or turns depending on Planning
- 8.4.4. Planning sends system logs to System Management

8.5. Post-Conditions

- 8.5.1. Car will cross intersection

9. Maintenance Monitoring

9.1. Description

- 9.1.1. This keeps track of any maintenance warnings that may need to have the car pulled over or make a stop. Keeps track of air in tires and gas in the car.

9.2. Functional Architecture

- 9.2.1. Input: Data from Tire Pressure, Gas, and Light Monitor

9.2.2. Processing: Monitoring Algorithm that take in levels from sensors to notify driver when the car needs to be taken in for maintenance

9.2.3. Output: Provides notification to Dashboard when a safety issue occurs, and every time the car starts if it has not been fixed.

9.3. Pre-Conditions

9.3.1. Driver is in the seat

9.3.2. Car is on

9.4. Requirements

9.4.1. Sensor Fusion receives data from Tire Pressure, Gas, and Light Monitors

9.4.2. Planning sends a message to VCS to display the need for a maintenance inspection for the driver, and notifies System Management that the car needs to be taken in

9.4.3. Planning sends system logs to System Management

9.5. Post-Conditions

9.5.1. Driver will be notified of insufficient conditions and will be suggested to take it to the technician for proper repair

10. Driver Monitoring

10.1. Description

10.1.1. Driver monitoring is meant to keep track of the driver, ensuring there is a driver in the car, as well as that they are aware in case of emergency.

10.2. Functional Architecture

10.2.1. Input: Data from Seat Occupancy, Ultrasound, Steering Wheel Contact

10.2.2. Processing: Monitoring Algorithm that determines current condition of the driver based on specified sensors

10.2.3. Output: Alerts Vehicle to slow down and pull over if the driver is not in condition to drive or warn the driver to focus on the road

10.3. Pre-Conditions

10.3.1. Driver is in the seat

10.3.2. Car is on

10.3.3. Car is in motion

10.4. Requirements

10.4.1. Sensor Fusion receives data from Seat Occupancy, Ultrasound, Steering Wheel Contact

- 10.4.2. Planning determines what actions to ensure the safety of the driver.
- 10.4.3. VCS pulls over if driver is incapable of driving and alerting driver if driver is capable of driving
- 10.4.4. Planning sends system logs to System Management
- 10.5. Post-Conditions
 - 10.5.1. The car will take necessary action to ensure the safety of the driver and any passengers

11. Technician

- 11.1. Description
 - 11.1.1. The Technician is responsible for maintaining the car and providing any manual updates
- 11.2. Function Architecture
 - 11.2.1. Input: Technician receives logs from system management and data from the cloud
 - 11.2.2. Processing: Technician performs routinely check
 - 11.2.3. Output: Conducts repair and maintenance
- 11.3. Pre-Conditions
 - 11.3.1. Car has been taken into repair shop
 - 11.3.2. Technician signs in with User ID and Password to login
- 11.4. Requirements
 - 11.4.1. Technician performs diagnostic
 - 11.4.2. Perform software updates when needed
 - 11.4.3. All noted repairs, system updates, and damage will be uploaded to the cloud
- 11.5. Post-Conditions
 - 11.5.1. Car will return on the road in working condition with the latest updates

Section 3.2: Non-Functional Requirements

1. Reliability

- 1.1. The system hardware, which are the sensors of the system, shall achieve a five-nine reliability (0.99999).

- 1.2. The system software shall achieve a five-nine reliability (0.99999) as well.
- 1.3. Redundant systems shall be placed to ensure non-stop operation in the event of sensor or software failures.

2. Performance

- 2.1. The time the system takes to receive information and execute the associated actions shall be minimal.
- 2.2. The duration of a message from a sensor to a VCS actuator shall be less than 10 nanoseconds.
- 2.3. The system shall be scalable to account for increased data volume and processing demands as the environment complexity or the number of users increases.

3. Security

- 3.1. The system management interface shall ensure that no data from the log file gets breached by unauthorized personnel.
 - 3.1.1. Door Access
 - 3.1.2. Technician login and password access, which should allow them access to different privileges based on their roles within the workflow.
- 3.2. Regular security updates shall be put in place to ensure that there aren't any vulnerabilities and to protect against possible rising threats.
- 3.3. The system shall keep an audit trail of all access and modifications to data, in order to ensure traceability and accountability.

4. Software Update

- 4.1. The system shall support software updates, which can be conducted either via cloud-based updates or through a technician.
 - 4.1.1. For cloud-based updates, there shall be a direct arrow from the system management component to indicate the update process. (this can be indicated directly on the picture of the process from section 2)
- 4.2. Users shall be notified in advance of schedule changes and updates in order to minimize any type of disruption to operations.

- 4.3. The system shall support rollback mechanisms to revert to a previous version of in cases of issues with the latest update.

Section 4: Requirement Modeling

Section 4.1: Use Case Scenarios

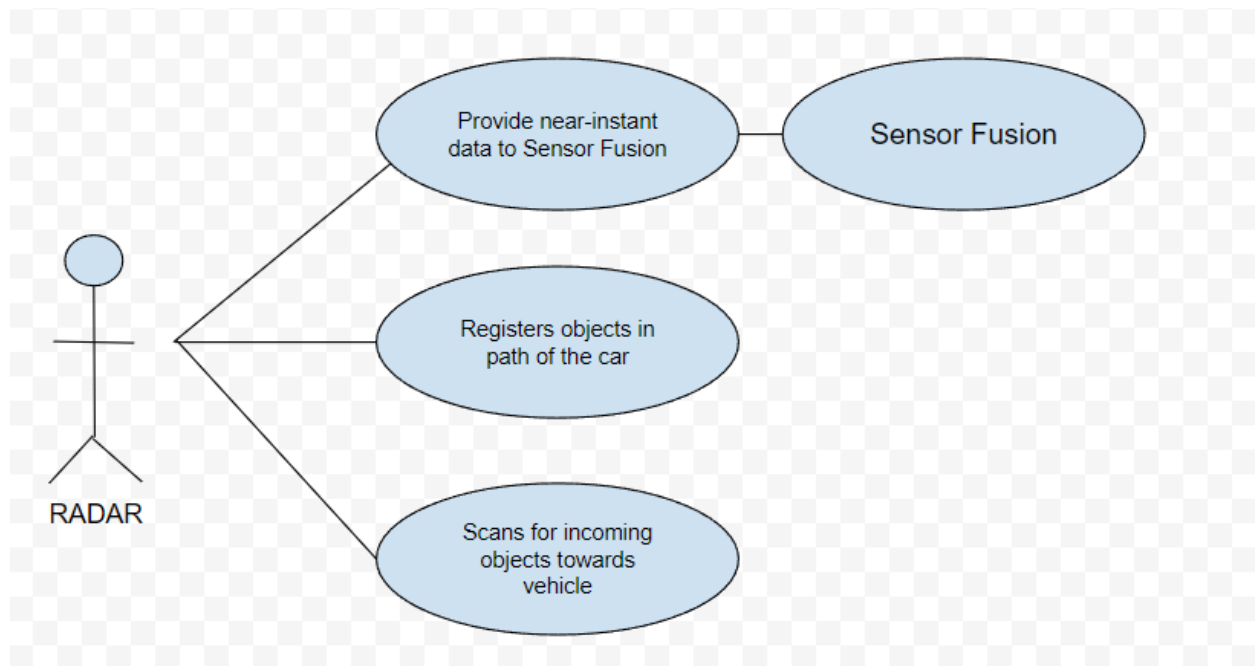


Figure 2 - UML Diagram of the actor (RADAR) and its purpose

Use Case 1: Object Detection and Avoidance

- Pre-Condition: The car is on and is removed from park.
- Post-Condition: The car is successful in avoiding collisions.
- Trigger: An object is detected within 50 Feet of the car and in its path.
 1. RADAR detects any obstacle in the car's way.
 2. Sensor Fusions sends that obstacle data to Planning.
 3. Planning decides the avoidance strategy based on that obstacle data and if necessary, it sends avoidance commands to the Vehicle Control System.
 - a. If the vehicle or object is detected to be from 26-50 feet away and unmoving, it decides to apply brake level 1.

- b. If it's between 15-25 feet, apply brake level 2.
 - c. If the object is under 15 feet apply brake level 3.
- 4. The VCS then executes avoidance reactions such as steering or braking.
- 5. Planning logs all data related to the obstacle analysis and VCS logs all the avoidance actions taken.

Use Case 2: Lane Departure Notification

- Pre-Condition: The car is on the road and is in drive, on a road that has lines.
- Post-Condition: The car successfully stays within the lines of a lane.
- Trigger: Cameras detects that a road line (lane) is being crossed.
 - 1. If any of the cameras detect that the car is going over some external road line, as well as take into account the car's position.
 - 2. Sensor Fusions detects the lane departure and sends it to Planning.
 - 3. Planning looks over the received data and prompts a notification.
 - a. If it is determined that a blinker was turned on during or prior to the lane departure then the notification isn't sent and nothing happens.
 - 4. This notification is then sent to the driver either through audio or visuals.
 - 5. Planning logs all these events.

Use Case 3: Driver Awareness

- Pre-Condition: There is a driver in the driver's seat and the car is removed from park position.
- Post-Condition: The driver has been alerted and is now aware of the surroundings again. Their hands are placed back on the wheel and/or they are awake.
- Trigger: The driver is asleep or has taken their hands off the wheel. The steering wheel sensor detects that the driver's hands have fallen off or the cameras facing the driver detect eyes are closed.
 - 1. Sensors pick up on either the drowsiness or hands off the wheel and send data to planning.
 - 2. When planning receives the data, if either of the actions has gone on for longer than 5 seconds then the signal is sent to VCS.
 - 3. VCS then will trigger an alarm in the front of the car.

4. The alarm stops once the driver is awakened or hands are back on the wheel.
5. If the driver's hands are still not detected or they are not awakening and its been over 15 seconds, then planning should analyze data.
6. Planning should find a safe way to return to a shoulder or safe location on the side of the road.
7. All data is sent to and logged in the log file.

Use Case 4: Parking Assistance

- Pre-Condition: The driver activates parking assistance.
- Post-Condition: The car is successfully parked.
- Trigger: The driver turns on the parking assistance.
 1. The driver activates parking assistance and selects the spot they would like to park the car in.
 2. Using RADAR and cameras the system determines the nearby parking space.
 3. This data gets sent to Planning.
 - a. If it's parallel parking that is needed then, it determines which side of the road they are parking on.
 - b. If it's regular parking, determine which side of the road.
 4. VCS executes the appropriate parking maneuvers to park successfully by maneuvering the steering wheel.
 - a. For parallel parking: Start backing into the spot until halfway in and at a 45-degree angle. Then straightening the wheel and backing up further, turn the wheel again and finish backing in. Once you can't go further, straighten the wheel and move forward.
 - b. Regular: Turns into the spot and recalculates if adjustments are needed.
 5. Planning logs all of this information.

Use Case 5: Weather Adaptation

- Pre-Condition: The car must be turned on and driving. Weather adaptations are activated by the driver.
- Post-Condition: The car is properly adapted/slowed due to the weather conditions.
- Trigger: Windshield wipers are turned on/Weather adaptation mode.

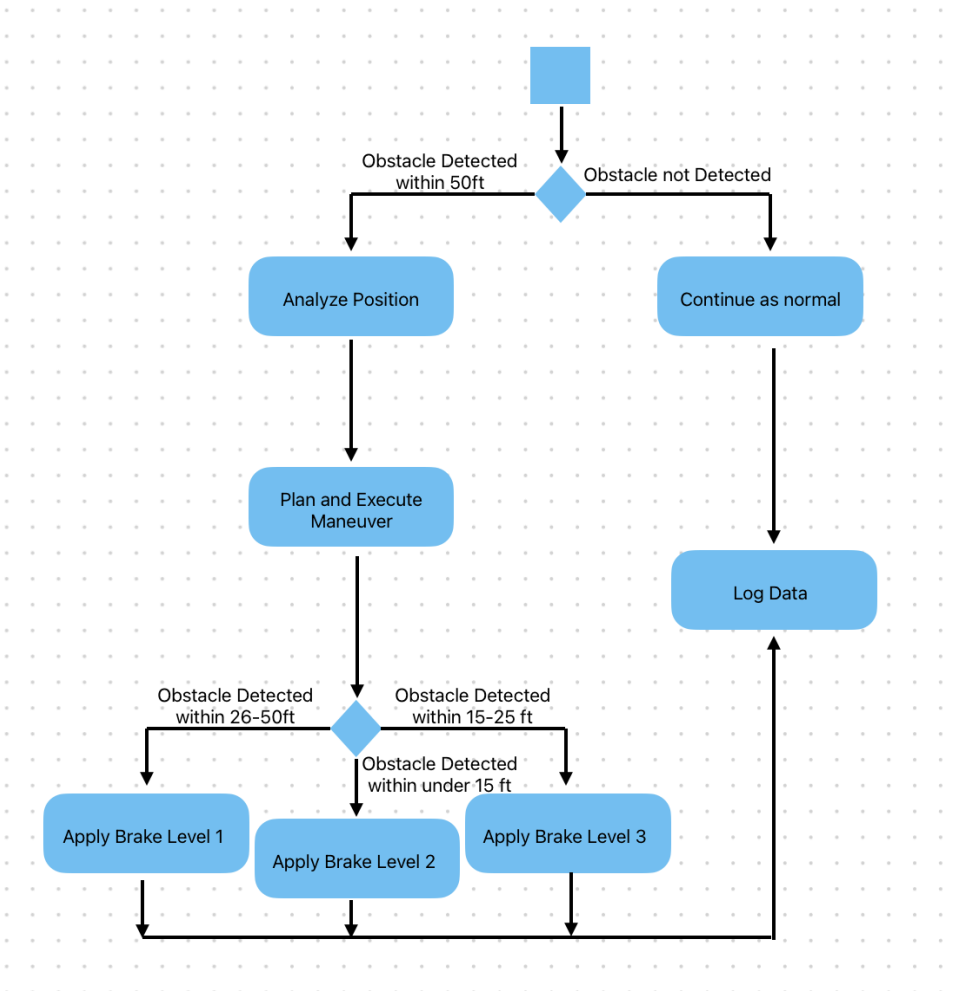
1. Windshield wipers turn on.
2. The speed is then determined.
 - a. If there is rain falling at 15-39 drops per minute then level 1 speed.
 - b. If the rain is falling at 40-59 drops per minute then level 2 is engaged.
 - c. If the rain is falling at anything 60 or above then level 3 is engaged.
3. Planning determines the necessary adjustments needed to speed, braking, and distance between cars/objects.
 - a. Adjusts so that braking distances increase by 5 feet.
4. VCS makes adjustments to car systems.
5. All information is logged in the log file.

Use Case 6: Maintenance Monitoring

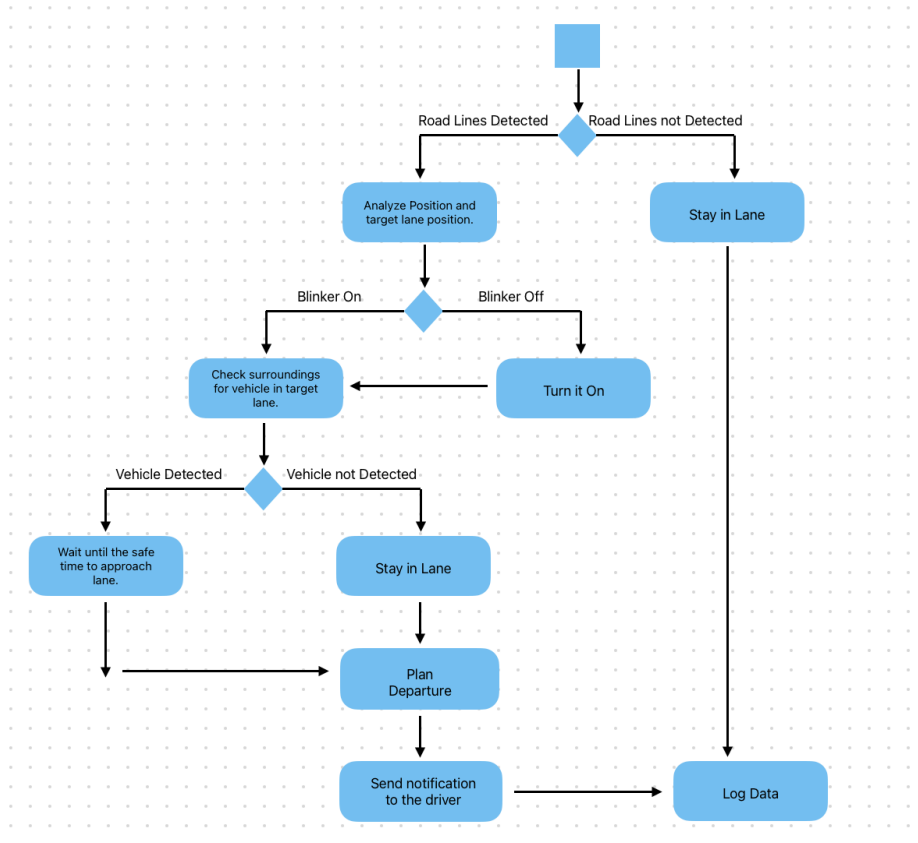
- Pre-Condition: The car is operating.
- Post-Condition: Any maintenance issues are found and logged for technician review.
- Trigger: The car sensors detect low air in a tire.
 1. If the tire pressure system detects a loss of air data gets sent to Planning
 2. Planning determines a course of action:
 - a. If the tire pressure falls between 20-30 psi then the signal is sent to the driver's dashboard signaling a low air pressure.
 - b. If it drops to anything below 20 psi the car sends signals to VCS determining the car needs to safely pull over and alerts a signaling to the driver of a flat tire.
 3. VCS executes the proper signals determined by planning.
 4. The driver dashboard receives and displays proper messages about the car's status.
 5. All actions are logged in the log file.

Section 4.2: Activity Diagrams

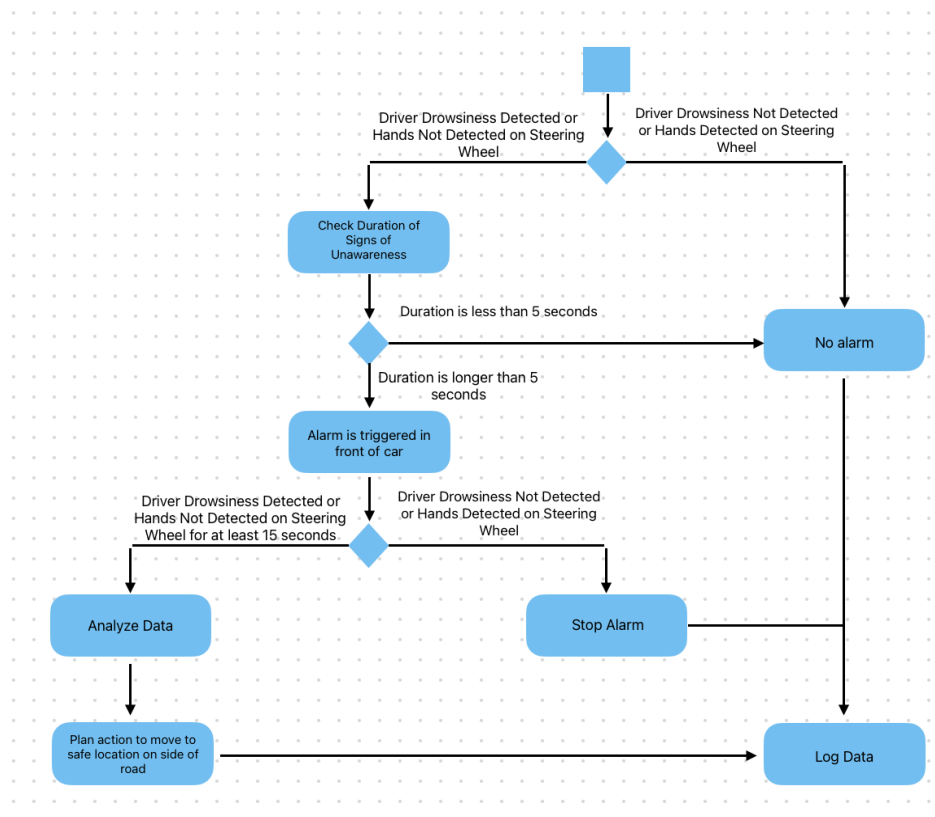
Use Case 1:



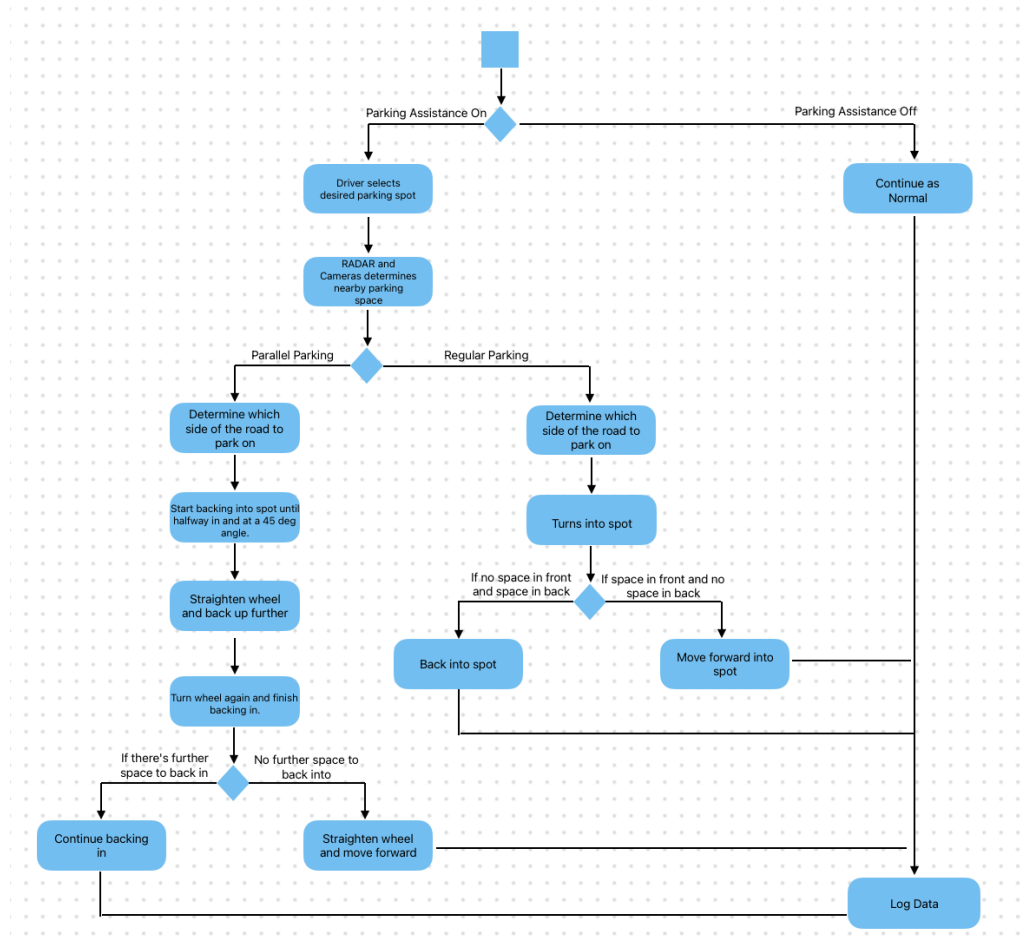
Use Case 2:



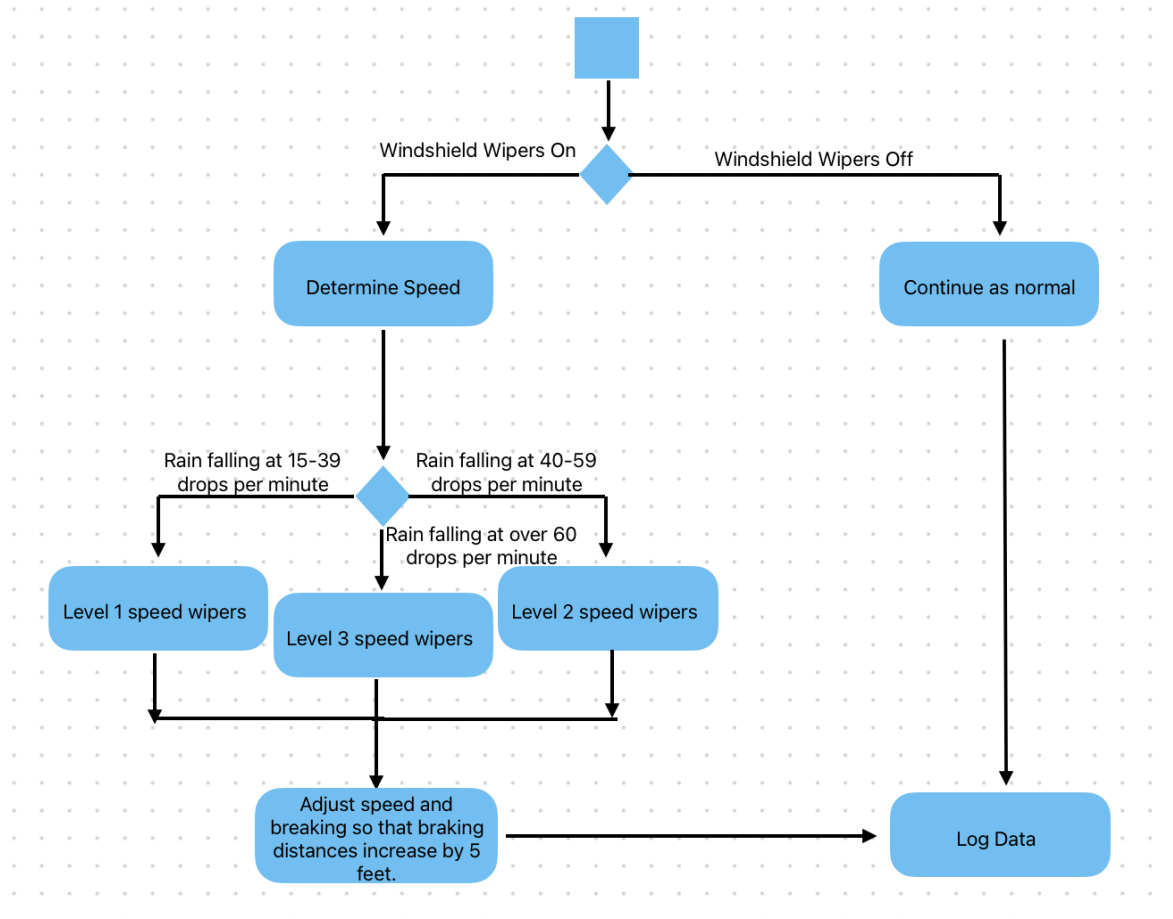
Use Case 3:



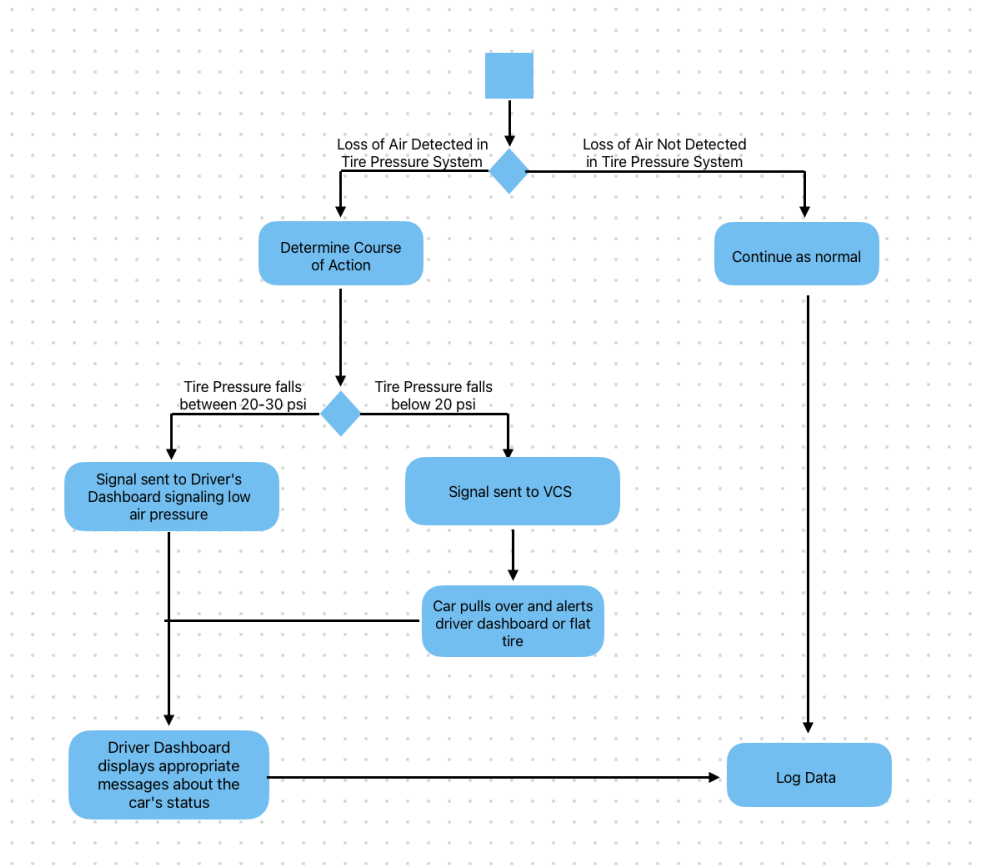
Use Case 4:



Use Case 5:

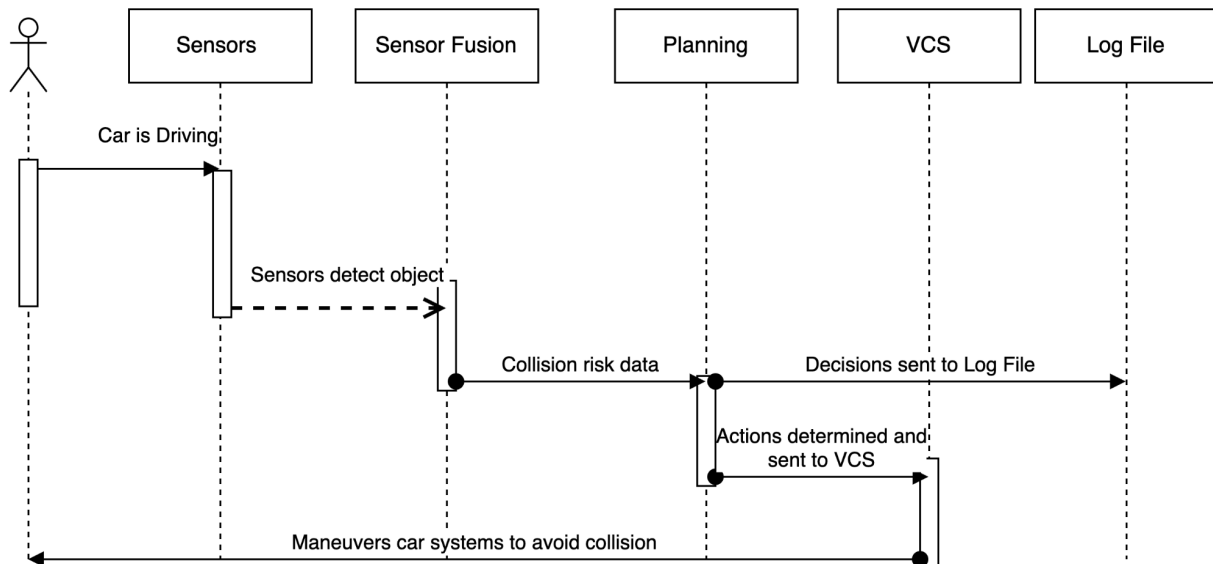


Use Case 6:

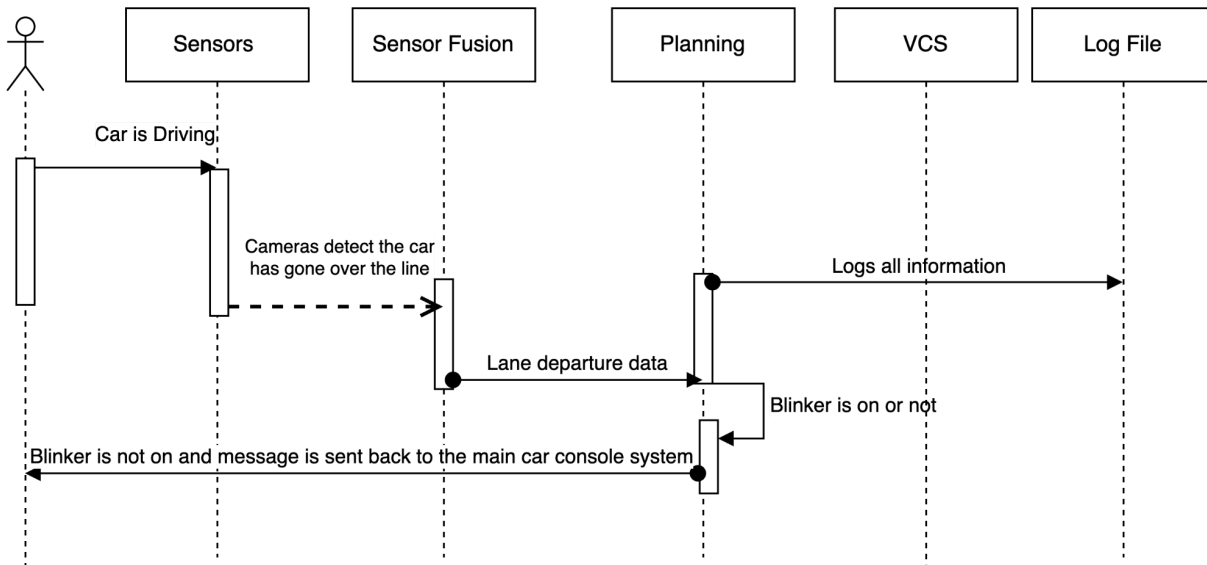


Section 4.3: Sequence Diagrams

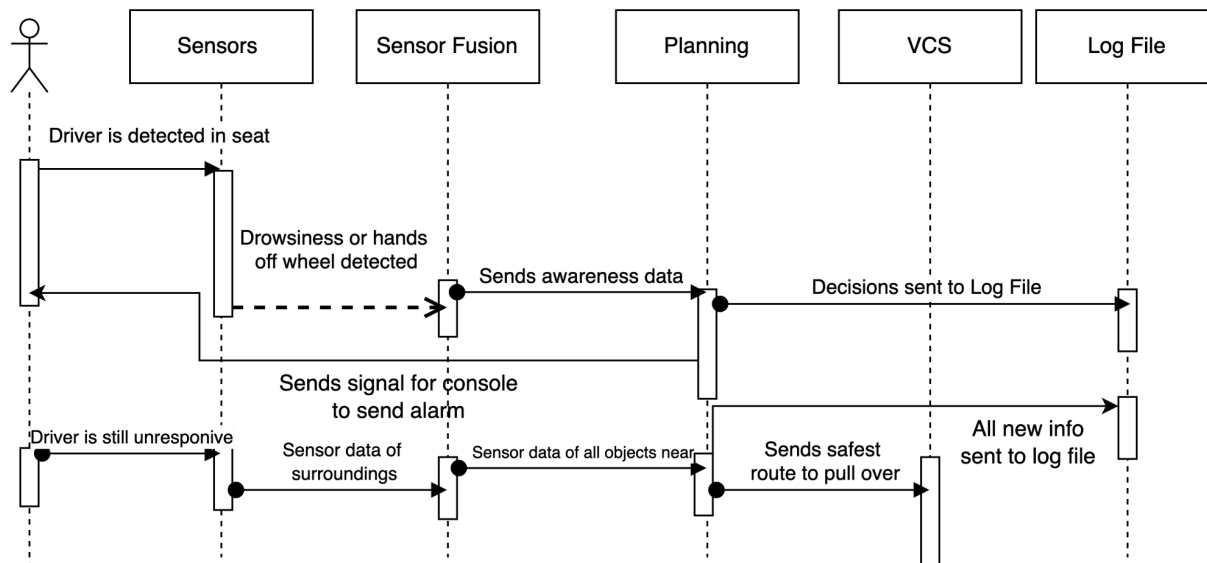
Use case 1:



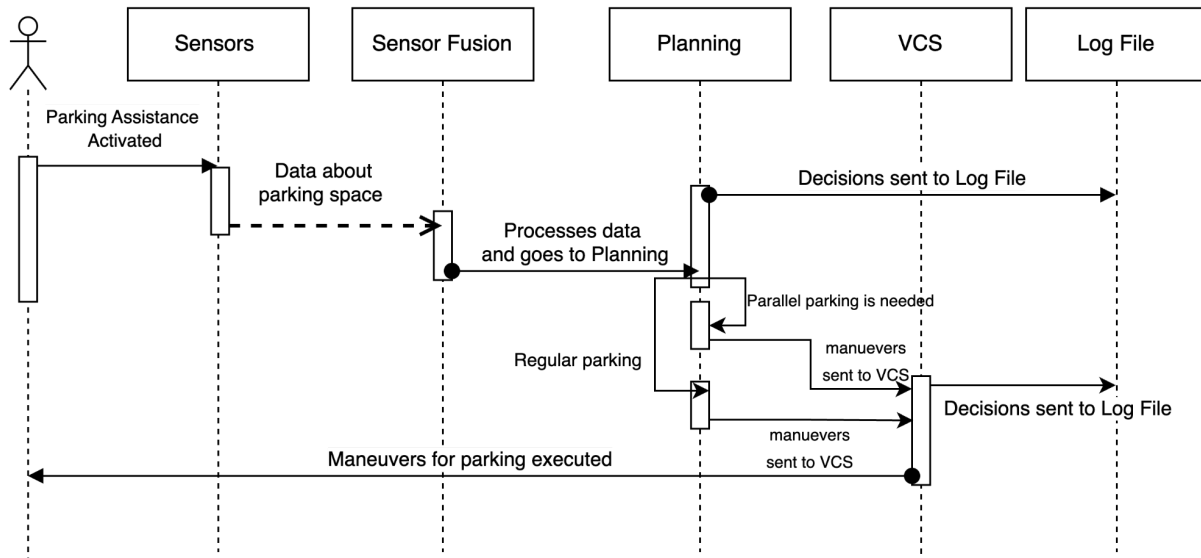
Use Case 2:



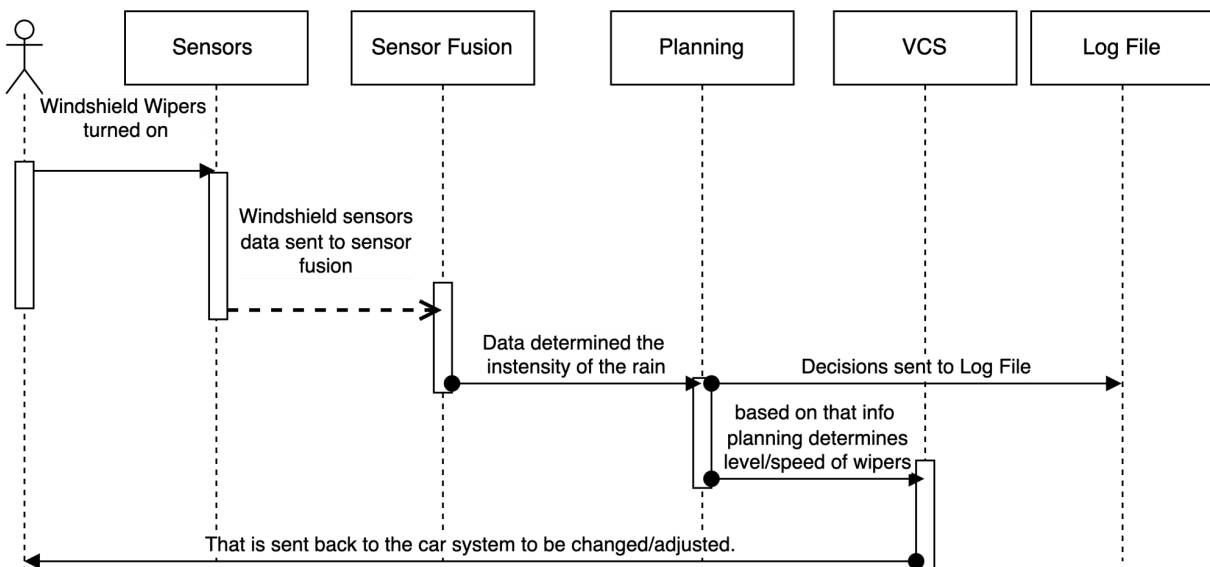
Use Case 3:



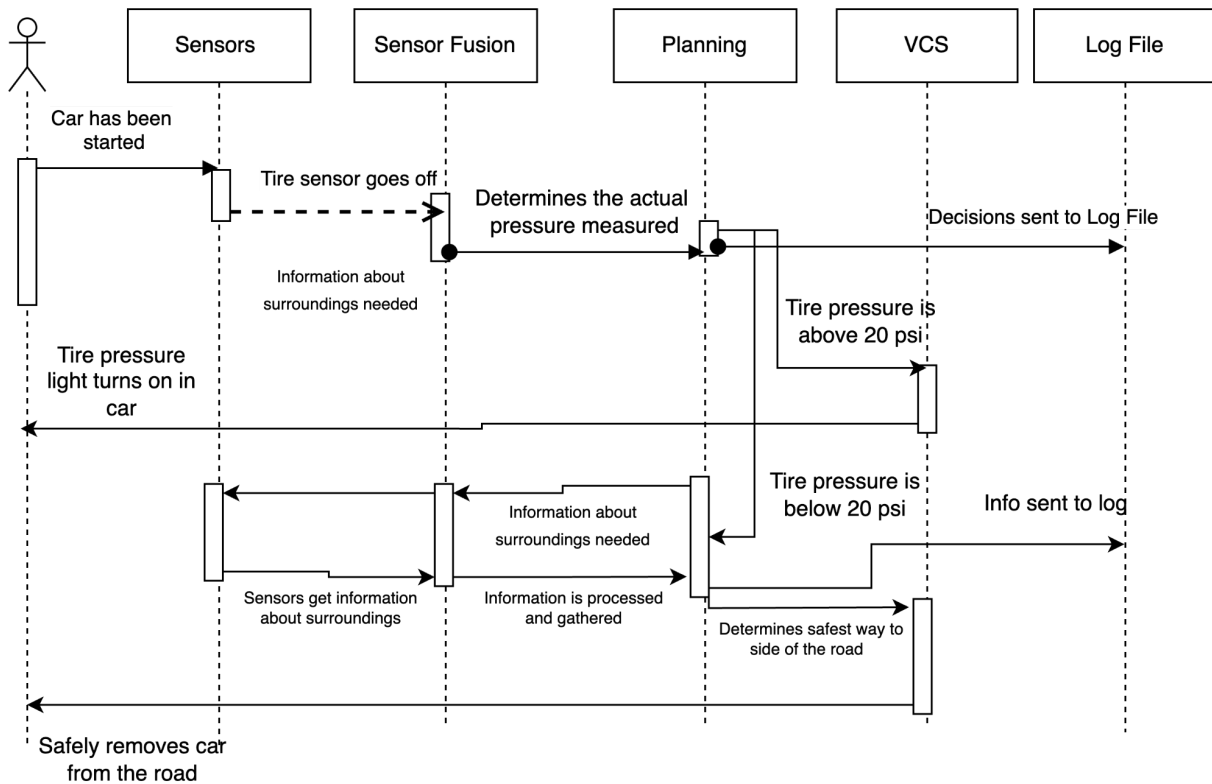
Use Case 4:



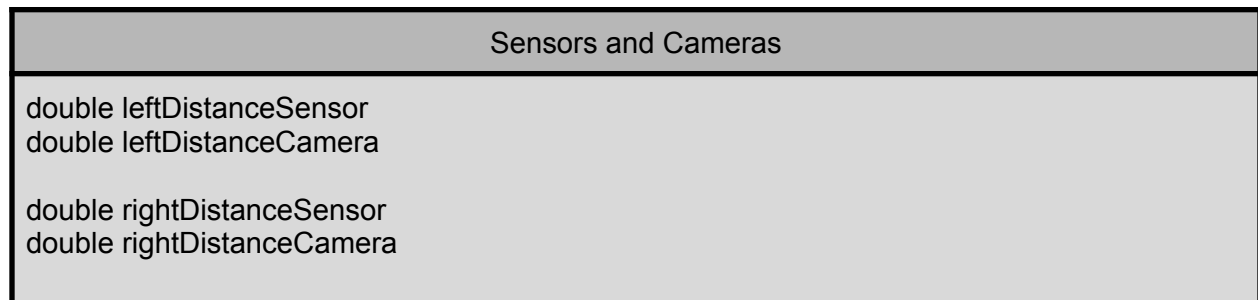
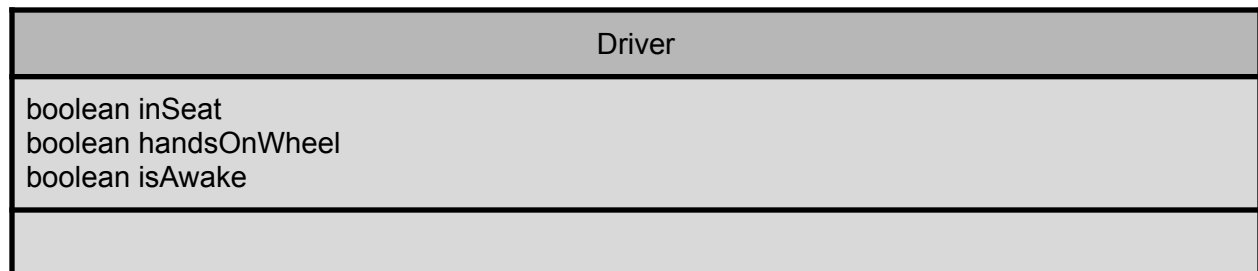
Use Case 5:



Use Case 6:



Section 4.4: Classes



```

double frontDistanceSensor
double frontDistanceCamera

double rearDistanceSensor
double rearDistanceCamera

double tirePressure
boolean steeringWheelSensor
boolean driverAwakeCamera

double gasInTank

double lightLevels

boolean adaptiveCruiseControlSwitch
boolean blinker

boolean weatherCondition
boolean weatherAndRoadAdaptationSwitch

```

```

boolean handsOnWheel() //Returns true if the driver's hands are on the wheel, false otherwise
String driverAlert() //Sends alert to driver

int getBlinker() //Returns 0 if no blinker is on, 1 if left blinker is on, or 2 if right blinker is on

double getDistance(String sensor) //Returns the distance of the specified sensor

int getGasInTank() //Returns current gas level
double getTirePressure() //Returns tire pressure
double getHeadLightLevel() //Returns current head light level

boolean getAdaptiveCruiseControl() //Returns true if adaptive cruise control is on, false otherwise

boolean getAwakeStatus() //Returns true if driver is awake, false otherwise
boolean getHandsOnWheelStatus() //Returns true if hands are on wheel, false otherwise
boolean getInSeat() //Returns true if driver is in seat, false otherwise

```

Vehicle Control System

```

boolean isOn
boolean inSeat
boolean inMotion

int vehicleSpeed
int obstacleDistance
int accelerationLevel

```


int brakeLevel

String trafficLightColor

bool leavingLane

int steerAngle

int selfVehicleSpeed

int otherVehicleSpeed

String weatherCondition

int windShieldLevel

boolean blindSpotMonitor() //Returns true if performed blind spot monitor procedure, false otherwise

boolean vehicleInBlindSpot() //Returns true if there is a vehicle in the blindspot, false otherwise

void parkingAssist()

boolean parkingAssistStatus() //Returns true if performed parking assist, false otherwise

boolean withinLines() //Returns true if the car is within the lines, false otherwise

boolean openParkingSpot() //Returns true if there is an open parking spot, false otherwise

double getObstacleDistance(String direction) //Returns the specified obstacle distance

boolean laneDeparture() //Returns true if performed lane departure procedure, false otherwise

boolean isLeavingLane() //Returns true if driver is leaving lane, false otherwise

boolean emitLeaveLaneSound() //Returns true if blinker was not on and attempting to leave lane, false otherwise

void collisionAvoidance()

boolean collisionAvoidanceStatus() //Returns true if performed collision avoidance, otherwise false

int leftSteer(int degree) //Returns number of degree turned left

int rightSteer(int degree) //Returns number of degree turned right

void setAccelerationLevel() //Sets acceleration level

int getAccelerationLevel() //Returns current acceleration level

void setBrakeLevel() //Set brake level

int getBrakeLevel() //Returns current brake level

void autoBrake()

boolean autoBrakeStatus //Return true if performed emergency auto brake, otherwise false

void adaptiveCruiseControl() //If adaptive cruise control takes over

boolean adaptiveCruiseControlStatus() //If adaptive cruise control takes over

```

int getOtherVehicleSpeed() //Gets speed of another vehicle in MPH
int getSelfVehicleSpeed() //Returns speed of self car in MPH

void weatherAndRoadApadatation()
boolean weatherAndRoadApadatationStatus() //If weather and road adaptation takes over
void setWindShieldWiperLevel() //Sets windshield wiper level
int getWindShieldWiperLevel() //Gets windshield wiper level
String getWeather() //Returns type of weather

void intersectionAssist() //Intersection Assist function
Boolean
String getTrafficLight() //Gets the light of the traffic light facing car

String maintenanceMonitor() //Prints what needs to be fixed to dashboard, checks all levels
when run

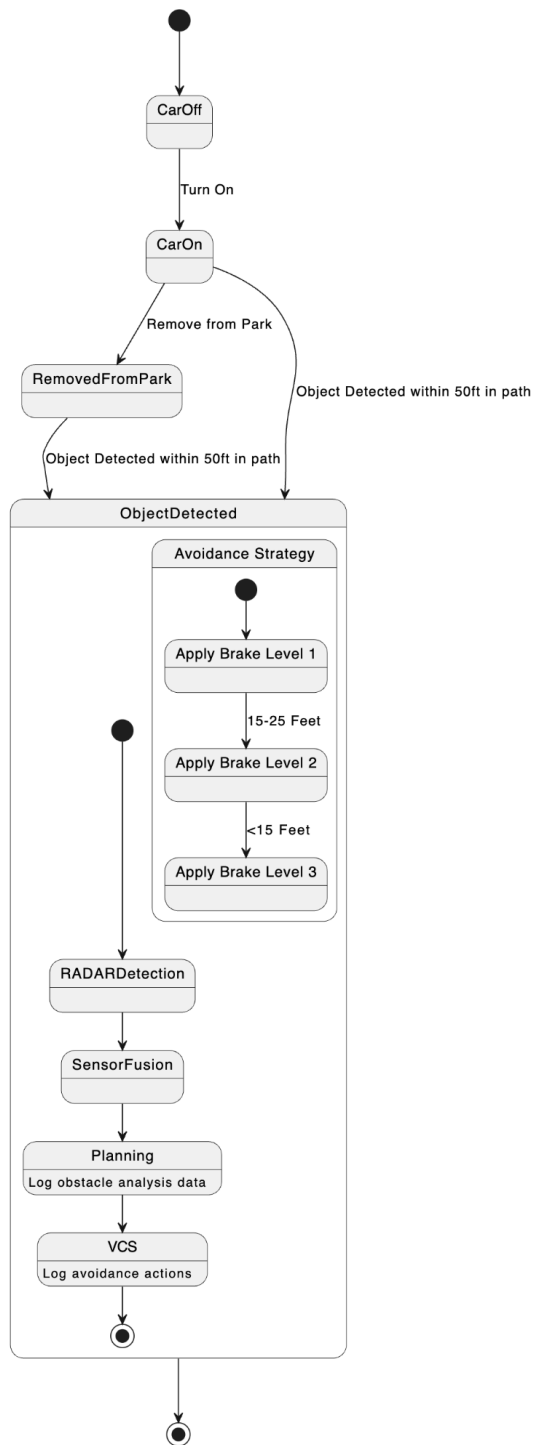
void driverMonitor()
boolean driverMonitor() //If driver is safe, false, else true

```

Section 4.5: State Diagrams

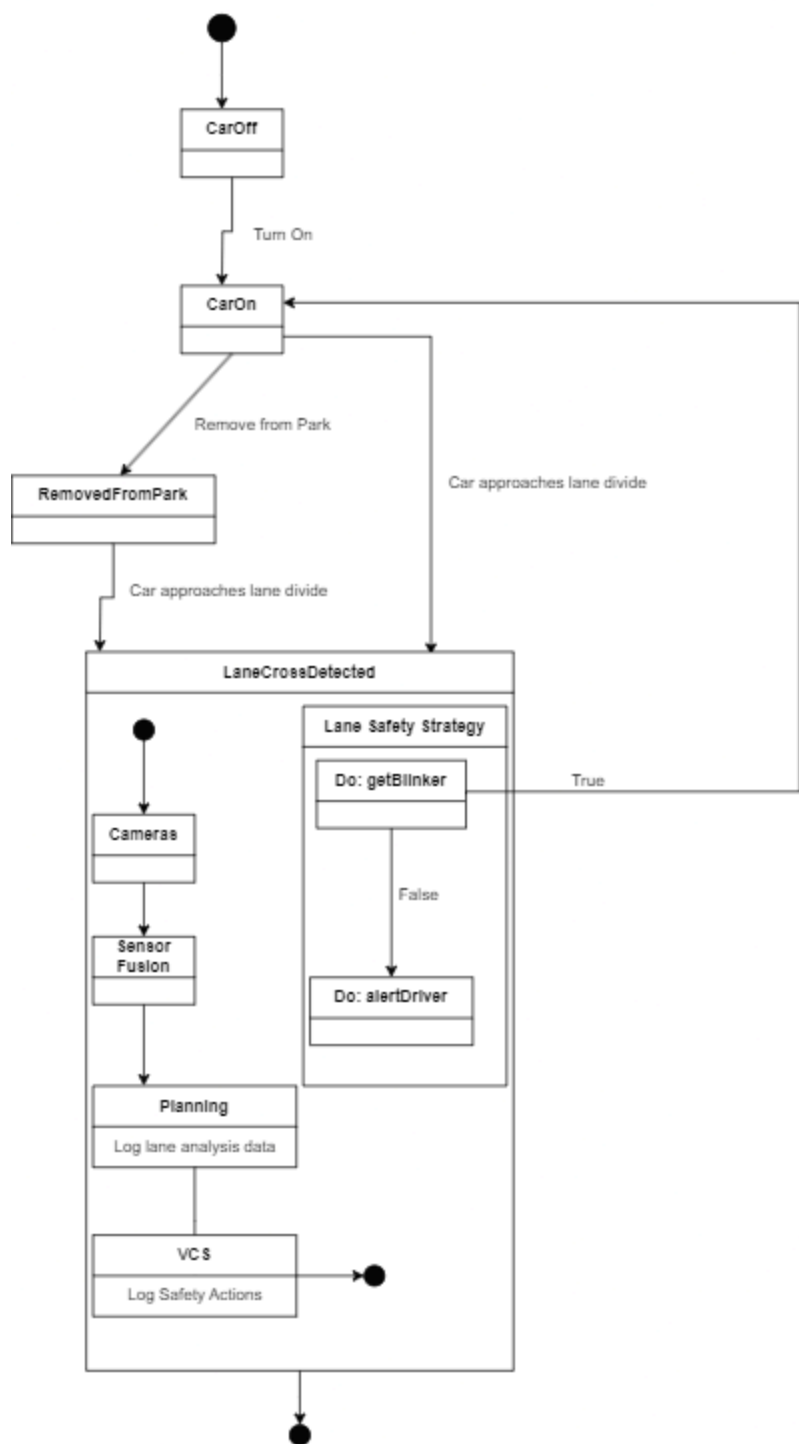
Use Case 1:

Self Driving Car System - Object Detection and Avoidance

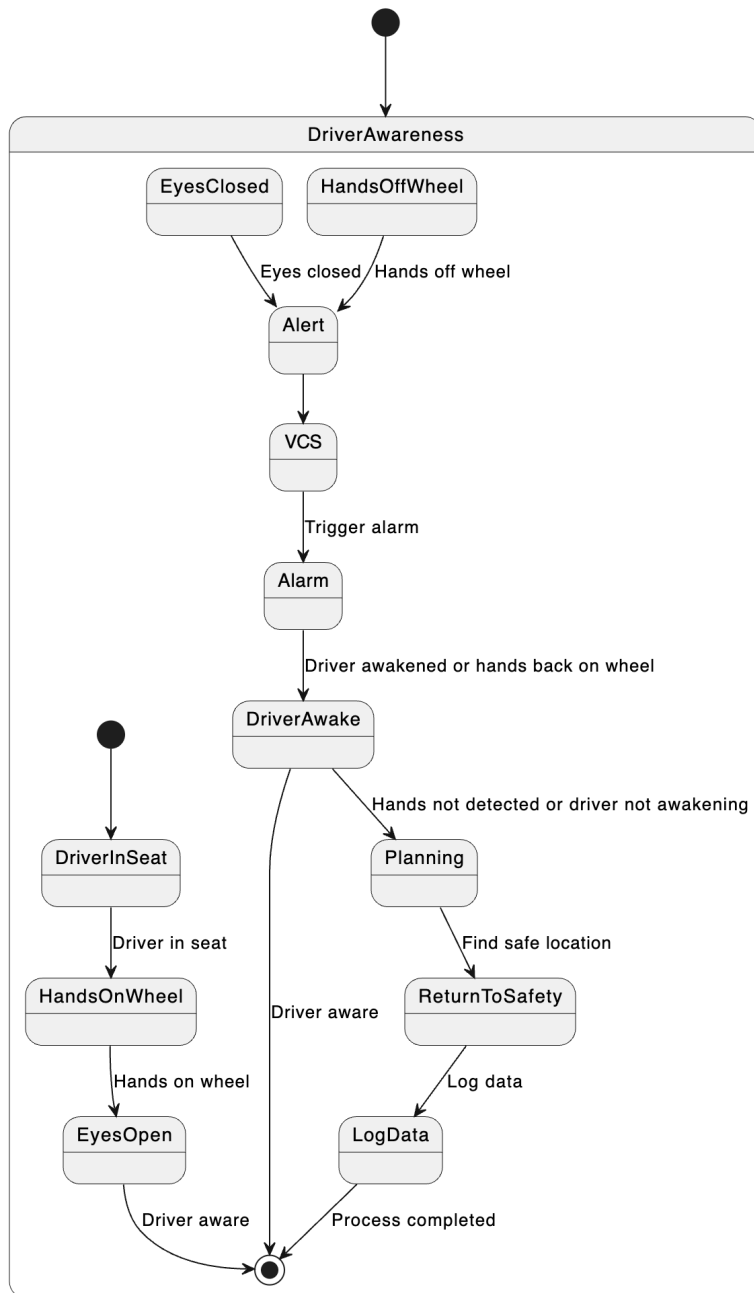


Use Case 2:

Self Driving Car System - Lane Departure Notification

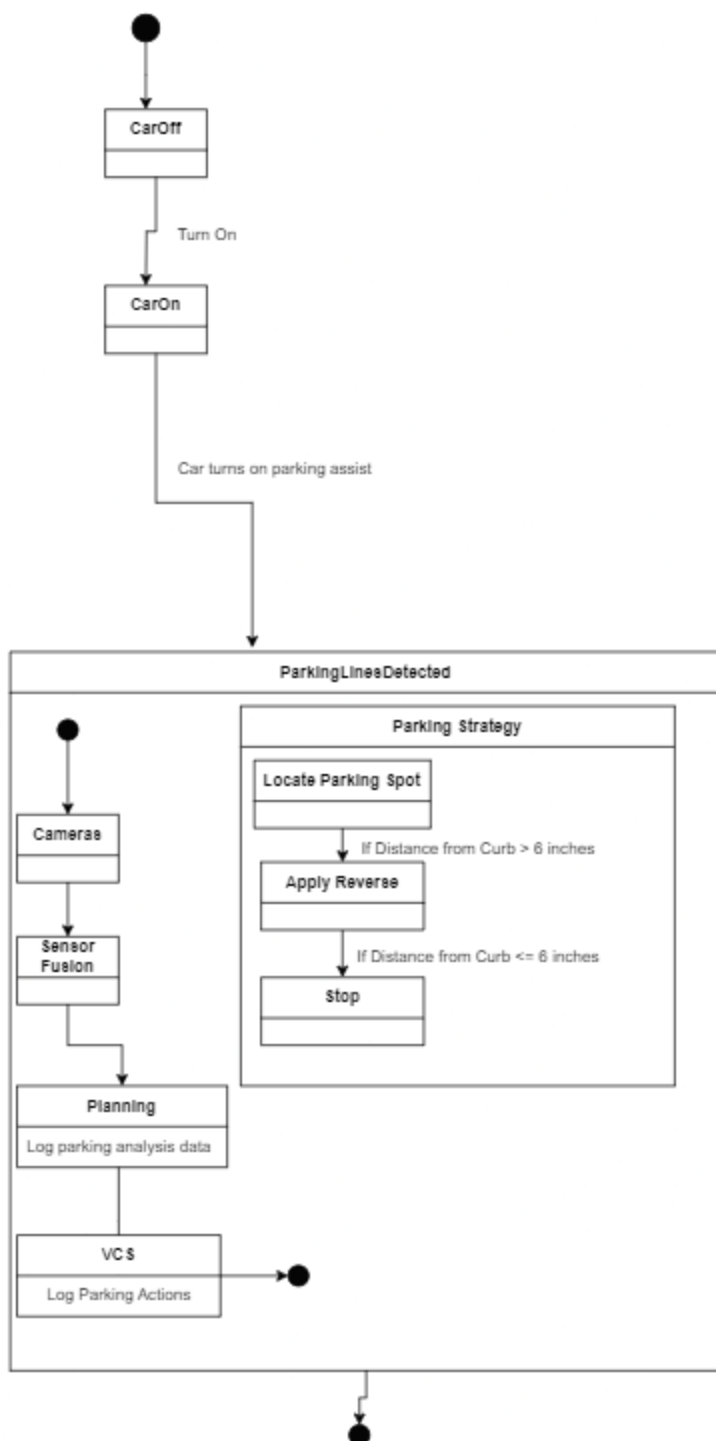


Use Case 3:

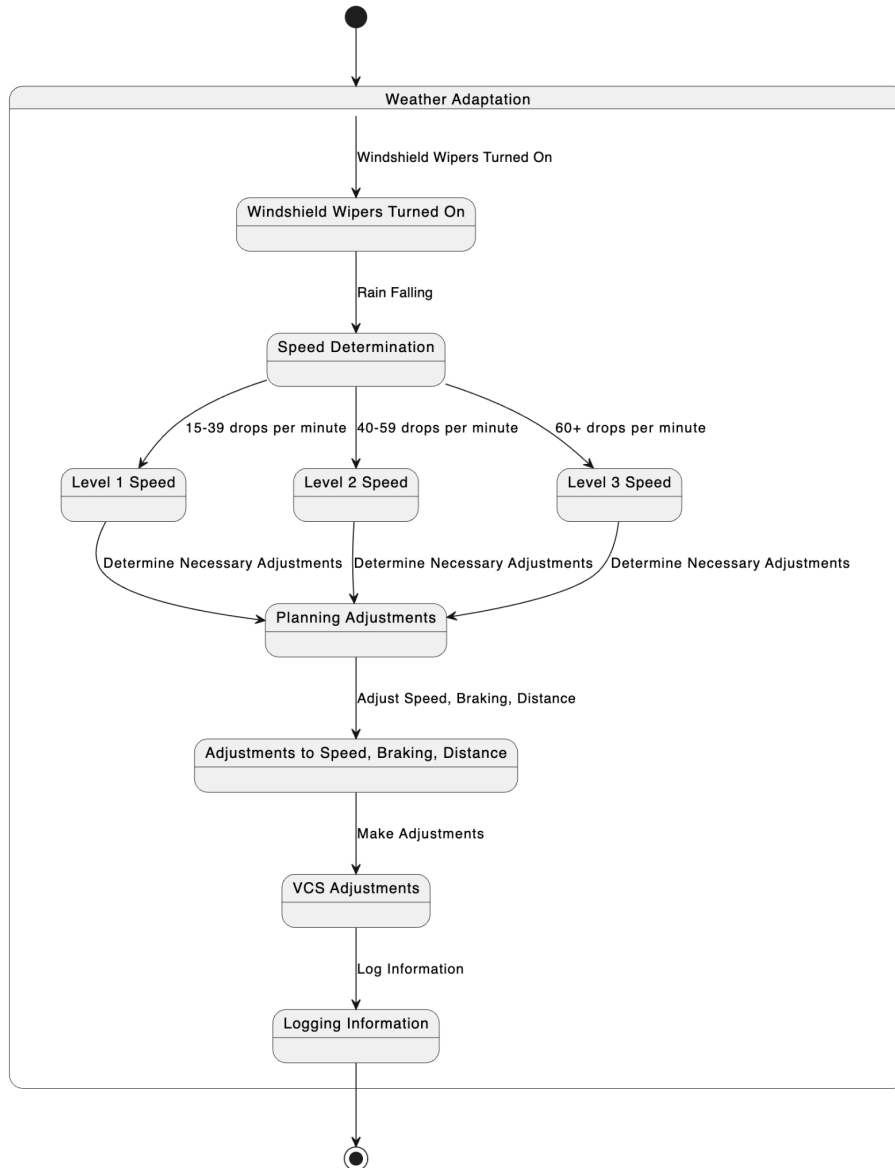


Use Case 4:

Self Driving Car System - Parking Assistance

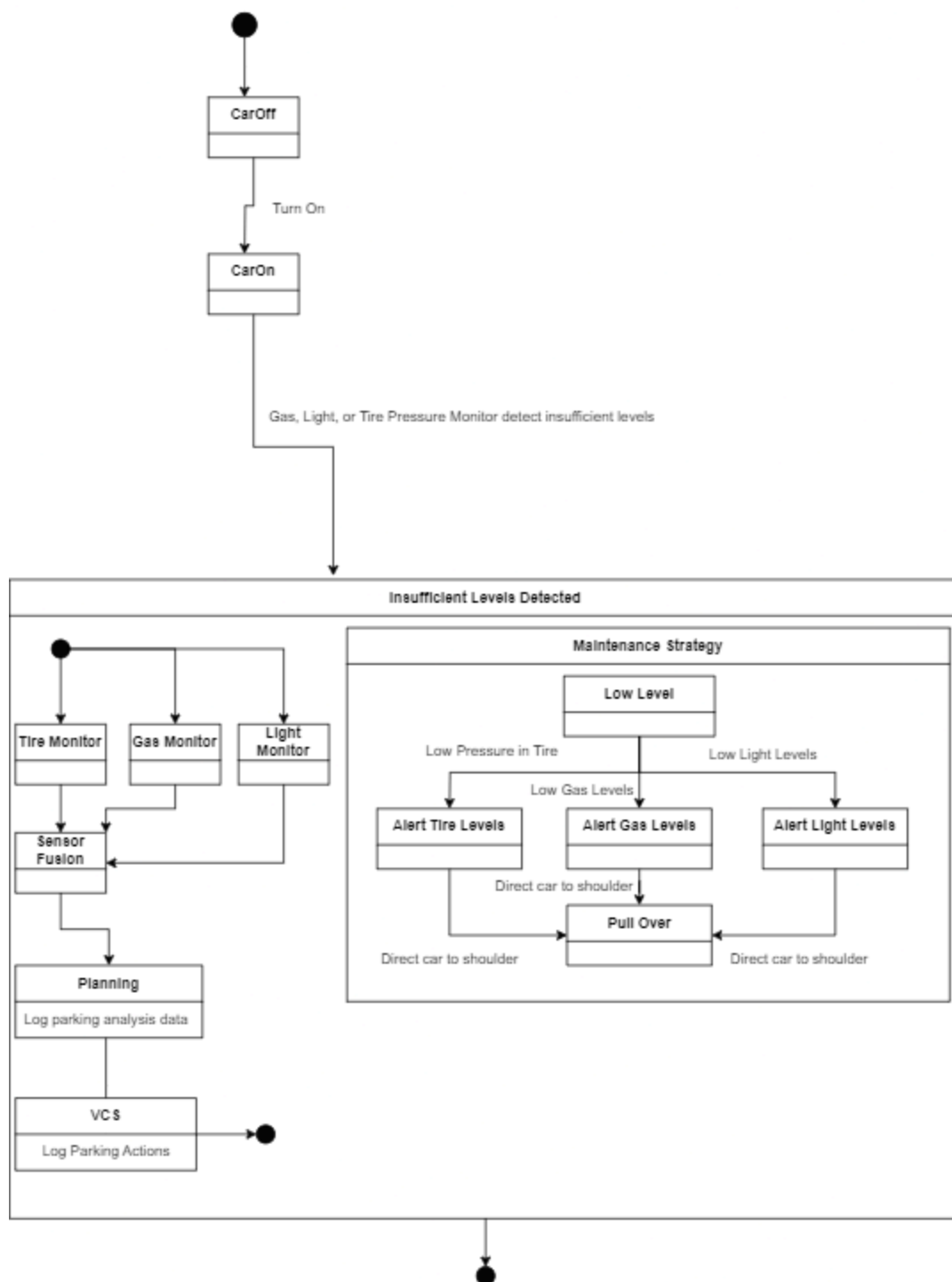


Use Case 5:



Use Case 6:

Self Driving Car System - Maintenance Monitoring



Section 5: Design

Section 5.1: Software Architecture

The software architecture is a representation of the operational software and is a crucial aspect of the communication among stakeholders in the development. Lots of early design decisions are determined in this stage; the software architecture needs to convey how the system is structured and how the components function together. There are multiple different styles of software architecture, and more than one may be applicable to this software.

1. Data Centered

- 1.1. The primary focus of the Data Centered architecture is data. This architecture is made to separate the data from the processes that interact with it, therefore all data for clients will write to and read from a singular shared data store. The pros of using this model are that it emphasizes data integrity, which is crucial for storing and processing sensor and camera data; it allows for efficient data organization; and it can provide an easy location to send all log data for further use. The cons are that it doesn't support handling complex control logic; all logic would need to be done as part of the client software.
- 1.2. This would be a fit for IoT HTL, since its main purpose is to have exceptional compatibility with sensor processing, which, in our case, is necessary for high speed processing.

2. Data Flow

- 2.1. The primary focus of the Data Flow architecture is to model the flow of data through a system. This architecture will track the movement of data and how the data is changed or used. It is common in data processing systems such as real-time systems or signal processing systems. The pros of this model are that it models the flow of sensor and camera data, which is important for real-time decisions; and it supports parallel

processing, which means that computational resources are being fully utilized. The cons of this model are that it can lead to complexities in managing the different data flow paths, and complexities in synchronization.

2.2. Similar to Data Centers, this would also be fit for IoT HTL.

3. Call Return

3.1. The primary focus of the Call Return architecture is to manage control flow through the use of the call and return of subprograms. In this architecture, a program will call a subprogram and wait for it to resolve before continuing with its own execution. The pros of this model are that it provides a structured approach to managing control flow, and it facilitates a modular approach, thus making the program easier to write. The cons of this model are that it may impact real-time responsiveness due to extensive function calls.

3.2. Due to its slow response time compared to other software architectures, Call Return would not be fit for IoT HTL.

4. Object-Oriented

4.1. The primary focus of the Object-Oriented architecture is to design using objects, which are singular instances of classes. These classes are made up of behavior functions and data. Because the objects are instances of one universal class, it allows for reusability if software components are properly organized. The pros of this model are that it allows for code reusability, and it allows sensor and camera data and algorithms to remain together. The cons of this model are that object creation and method-calling can impact performance in time-sensitive tasks.

4.2. Object oriented architecture may impact performance on time-sensitive tasks, but its ability to allow sensor data and algorithms to remain together, alongside code reusability allows coders to construct a program that can be read and understood easily. For the purpose of this project, object-oriented architecture is fit for IoT HTL.

5. Layered

- 5.1. The primary focus of the Layered architecture is to organize software components into distinct layers, with each layer being responsible for a specific set of functions. Layers will only interact with adjacent layers, which enforces a strict hierarchy. The pros of this model are that it promotes separation of concerns, and it facilitates scalability by allowing new layers to be added. The cons of this model are that there may be increased complexity in managing interactions between layers, especially in real-time systems.

6. Model View Controller

- 6.1. The primary focus of the Model View Controller architecture is to divide an application into three interconnected components. The model is the data and business logic, the view is the presentation layer, and the controller is the logic for handling user input. The pros of this model are that it separates the different layers which promotes modularity and maintainability, and it facilitates iterative development and testing by removing the interface from the underlying logic. The cons are that it may introduce overhead in managing communication between all three components, which could impact real-time responsiveness.
- 6.2. Impacting several components in responsiveness could lead to delays

7. Finite State Machine

- 7.1. The primary focus of the Finite State Machine is to represent a system with a finite number of states and transitions between these states based on inputs. They are used to model the behavior of systems that exhibit sequential behavior. The pros of this model are that it provides a clever and structured way to model a certain behavior, and it facilitates formal verification and validation of the system. The cons of this model are that it has limited expressiveness in handling complex behaviors that cannot be easily represented as finite states and transitions.

7.2. Based on the above pros and cons, we plan to use the following software architectures:

- 7.2.1. Finite State Machine

- 7.2.1.1. This architecture can be adopted for the technician login, because there is a clear finite number of states and there is a sequential behavior, i.e. the user logging into the system. This architecture cannot be used for the self-driving features as there is not a finite amount of states due to complexity.

7.3. Object-Oriented

- 7.3.1. OO architecture will be adopted for the driver interface. This will allow different aspects of the interface to be grouped into different classes, providing an organized method of coding.

Section 5.2: Interface Design

1. Technician Interface:

- 1.1. The main purpose of the Technician interface is to monitor the vehicle and to check what needs to be fixed, replaced, or updated. The Technician UI will comprise a lot of technical components needed for maintaining the car to modern standards in comparison to the Driver Interface. It is critical for the Technician to be able to access all parts of the car to be able to fix anything that can go wrong anywhere. All features of the car will be available in the Technician's UI menu.
- 1.2. Features
 - 1.2.1. Monitor Menu
 - 1.2.1.1. maintenanceMonitor
 - 1.2.1.2. Tire Pressure Gauge
 - 1.2.1.2.1. tirePressure
 - 1.2.1.2.2. Current Pressure in tire
 - 1.2.2. Gas Levels
 - 1.2.2.1. gasInTank

- 1.2.2.2. Current Gas Level
 - 1.2.3. Light Levels
 - 1.2.3.1. lightLevels
 - 1.2.3.2. Current light levels
 - 1.2.4. Tire Grip Levels
 - 1.2.4.1. Tire grip
 - 1.2.4.2. Consists of percentage of tire grip from original
 - 1.2.5. Damage to Vehicle
 - 1.2.5.1. Physical or electrical damage to the vehicle
 - 1.2.6. Battery Levels
 - 1.2.6.1. Car battery level
 - 1.2.7. Engine Condition
 - 1.2.7.1. Current condition of vehicle engine
 - 1.2.8. Oil Condition
 - 1.2.8.1. If vehicle needs motor oil replacement
 - 1.2.9. Mirror Condition
 - 1.2.9.1. Any mirror are broken or blurred
 - 1.2.10. Camera Condition
 - 1.2.10.1. Any camera is broken, blurred, or not functional
 - 1.2.11. Ventilation System Condition
 - 1.2.11.1. Current condition of air flow, AC, and Heater
 - 1.2.12. Speaker Condition
 - 1.2.12.1. If speakers are emitting sound properly
 - 1.2.13. Brake Condition
 - 1.2.13.1. Current condition of brakes
 - 1.2.14. Suspension Conditions
 - 1.2.14.1. Current condition of suspension
- 1.3. Software Updates
 - 1.3.1. Consists of any and all software updates needed to keep the vehicle's system up to date and functioning properly
- 1.4. Hardware Updates

- 1.4.1. Consists of any and all hardware components that need to be updated, replaced, or repaired to keep vehicle working
- 1.5. Available Car Upgrades
 - 1.5.1. Any upgrades to the car, including tire upgrades, engine upgrades, camera upgrades, etc.
- 1.6. Driving History
 - 1.6.1. Includes map of driving history on days used and any documentation/log files of accidents, faulty features, and overall vehicle performance

2. Driver Interface

- 2.1. The driver interface design serves as the primary means for the driver to interact with and control the vehicle. It needs to provide essential feedback and information to the driver, and it also needs to allow the driver to provide inputs and make decisions. The interface will consist of both physical controls and digital displays, but will be designed with simplicity in mind. Critical features such as speed, temperature, and fuel level will be larger, and less critical features will be smaller.
- 2.2. Features:
 - 2.2.1. Speedometer
 - 2.2.2. Temperature
 - 2.2.3. Tire Pressure Gauge
 - 2.2.4. Blind Spot Indicators
 - 2.2.5. Lane Departure Warning
 - 2.2.6. Cruise Control Status
 - 2.2.7. Navigation and Maps

Section 5.3: Component-level Design

1. Intelligence Control Component

- 1.1. This component would include Sensor Fusion, Planning, and VCS. These components use sensors, cameras, data processors, and decision-makers

to work with and gather data. Data is collected from all sensors in sensor fusion. Sensor fusion collects data from LIDAR, RADAR, and other internal sensors. Data is then integrated to formulate a collective image of the car's surroundings. This is pinpointing and identifying all surrounding objects that may be of importance. This keeps track of all possible dangers (blind spots, obstacles, etc). Planning receives this data on the car image and surroundings. This is where the procedure is to study the surrounding area and determine the exact dangers around the car. Once Planning has identified the danger (lane departure, pedestrian, etc), it then uses the system to determine what exact maneuvers the car needs to do to avoid the danger. This can be braking unexpectedly, steering to the side, and so on. This exact solution is then organized and sent to VCS. Which then sends all those actions to where they need to go. If it tells the car to brake then it sends those signals to the brakes and slows the car down. All three areas take in outside information and process them before executing them. They have different outputs as VCS alters the actual car, Planning outputs decisions that VCS must execute and Sensor Fusion sends an image of the surrounding area to Planning for processing.

2. Driver Interface

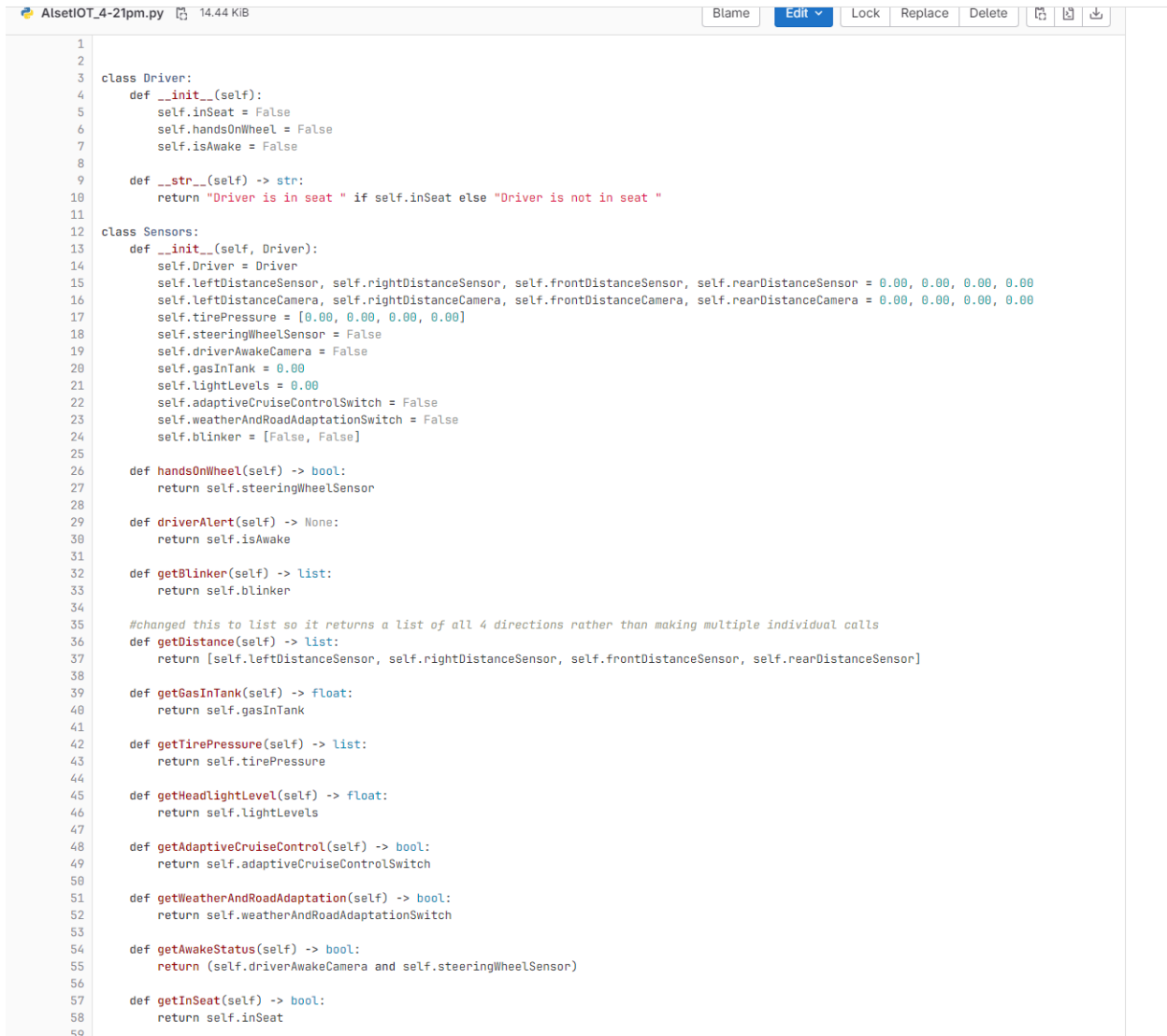
2.1. This is all the Driver Commands and the dashboard, anything the drivers can use/interact with. Upon turning the car on these interfaces will turn on to provide a user-friendly environment. All of these components are only responsive when interacted with or used by the user. Otherwise, they wait ready to be engaged with. Once the interface or command is interacted with it should immediately execute. For a command such as turning on the windshield wipers, upon being turned on the wipers she immediately becomes engaged in whatever setting is being used. If it is later turned off it should also happen immediately.

3. Technician Interface

- 3.1. This is for those who work on the vehicle itself. It creates an interface for them to assess the car as easily as possible. This means all functions apart from the interface will display information as needed. They will receive information from the correlating sensors and report back what this data means. For example, tire pressure will be taken from sensors and display information such as tire pressure being too low and needing maintenance.

Section 6: Code

Section 6.1: Driver Interface



```

1
2
3 class Driver:
4     def __init__(self):
5         self.inSeat = False
6         self.handsOnWheel = False
7         self.isAwake = False
8
9     def __str__(self) -> str:
10        return "Driver is in seat " if self.inSeat else "Driver is not in seat "
11
12 class Sensors:
13     def __init__(self, Driver):
14         self.Driver = Driver
15         self.leftDistanceSensor, self.rightDistanceSensor, self.frontDistanceSensor, self.rearDistanceSensor = 0.00, 0.00, 0.00, 0.00
16         self.leftDistanceCamera, self.rightDistanceCamera, self.frontDistanceCamera, self.rearDistanceCamera = 0.00, 0.00, 0.00, 0.00
17         self.tirePressure = [0.00, 0.00, 0.00, 0.00]
18         self.steeringWheelSensor = False
19         self.driverAwakeCamera = False
20         self.gasInTank = 0.00
21         self.lightLevels = 0.00
22         self.adaptiveCruiseControlSwitch = False
23         self.weatherAndRoadAdaptationSwitch = False
24         self.blinker = [False, False]
25
26     def handsOnWheel(self) -> bool:
27         return self.steeringWheelSensor
28
29     def driverAlert(self) -> None:
30         return self.isAwake
31
32     def getBlinker(self) -> list:
33         return self.blinker
34
35     #changed this to list so it returns a list of all 4 directions rather than making multiple individual calls
36     def getDistance(self) -> list:
37         return [self.leftDistanceSensor, self.rightDistanceSensor, self.frontDistanceSensor, self.rearDistanceSensor]
38
39     def getGasInTank(self) -> float:
40         return self.gasInTank
41
42     def getTirePressure(self) -> list:
43         return self.tirePressure
44
45     def getHeadLightLevel(self) -> float:
46         return self.lightLevels
47
48     def getAdaptiveCruiseControl(self) -> bool:
49         return self.adaptiveCruiseControlSwitch
50
51     def getWeatherAndRoadAdaptation(self) -> bool:
52         return self.weatherAndRoadAdaptationSwitch
53
54     def getAwakeStatus(self) -> bool:
55         return (self.driverAwakeCamera and self.steeringWheelSensor)
56
57     def getInSeat(self) -> bool:
58         return self.inSeat
59

```

Figure 3 - Full Details on Gitlab

Section 6.2: Technician Interface

```

AlsetIoT-Technician.py 2.81 KiB
Blame Edit Lock Replace Delete

1 import sys
2
3
4 vehicle_conditions = {
5     "tire_pressure": 32, # PSI
6     "gas_level": 50, # Percentage
7     "light_levels": 50, # Percentage
8     "tire_grip": 80, # Percentage from original
9     "damage": "None", # Damage to car
10    "battery_level": 75, # Percentage
11    "engine_condition": "Good", # Condition of engine
12    "oil_condition": "Change Needed", # Oil condition
13    "mirror_condition": "Clear", # Mirror clarity
14    "camera_condition": "Operational", # Camera condition
15    "ventilation_system": "Functional", # Vent system functionality
16    "speaker_condition": "Clear", # Clear or static
17    "brake_condition": "Good", # Brake functionality
18    "suspension_condition": "Okay" # Condition of suspension
19 }
20
21 software_update = False
22
23 hardware_update = False
24
25
26 def login(): # Check credentials for login
27     max_attempts=3
28     username = "Username"
29     password = "password"
30     attempt_count = 0
31     while attempt_count < max_attempts:
32         userInput = input("Enter username: ")
33         pwdInput = input("Enter password: ")
34         if userInput == username and pwdInput == password:
35             print("Login successful")
36             return True
37         else:
38             print("Incorrect credentials")
39             attempt_count += 1
40     print("Too many incorrect attempts. Exiting...")
41     return False
42
43 def menu_selection(): # Ask for selected menu
44     menu_options = ["Software Updates", "Hardware Updates", "Monitor Menu"]
45     while True:
46         state = input("Please select from the following menus: Software Updates, Hardware Updates, Monitor Menu: ")
47         if state in menu_options:
48             return state
49         print("Please select a valid option")
50
51 def monitor_menu(): # Return all values of vehicle conditions
52     return vehicle_conditions
53
54 def software_updates(): # Return bool value of software_update
55     return software_update

```

Figure 4 - Full Details on Gitlab

Section 7: Testing

Section 7.1: Validation Testing

1. Object Detection Test
 - a. Precondition: Car is on, speed > 0
 - b. Postcondition: car stops 10 feet from object
 - c. Inputs: speed, distance from object = 100ft,
 - d. Output: car stops at 10 feet from the object
2. Parking Assistance Test
 - a. Precondition: Car is in reverse gear.
 - b. Postcondition: Car successfully parks within the designated space.
 - c. Inputs: Space available for parking; obstacles are detected by front and rear cameras
 - d. Output: Car steers according to obstacle positions to park within the target space.
3. Weather and Road Adaptation Test
 - a. Precondition: Weather conditions include precipitation.
 - b. Postcondition: System adjusts windshield wiper level according to weather conditions.
 - c. Inputs: Weather conditions; speed and level of precipitation; command to adjust wiper level.
 - d. Output: Windshield wiper level is appropriately chosen for the current weather.
4. Collision Avoidance Test
 - a. Precondition: Car is in motion
 - b. Postcondition: System applies appropriate brakes or steering to avoid collision.
 - c. Inputs: Distance to obstacle < 50ft, vehicle speed, obstacle distance.
 - d. Output: Appropriate action is taken to avoid collision, such as braking or steering.
5. Auto Brake Activation Test
 - a. Precondition: Car is in motion; auto brake system is active.
 - b. Postcondition: Appropriate braking levels are activated based on obstacle distance.
 - c. Inputs: Car speed, obstacle distance.
 - d. Output: Appropriate braking level is applied to prevent collision.
6. Blind Spot Monitoring Test
 - a. Precondition: Car is in motion and in a multi lane road.
 - b. Post Condition: Driver stays in lane and aware of vehicles in blind spot
 - c. Inputs: Camera distance and location of vehicles to the right and left
 - d. Outputs: Beeping to alert driver and car stays in current lane.

Section 7.2: Scenario Testing

7. Technician Login

- a. Precondition: laptop connected to system admin, login/pwd page active
 - b. Postcondition: technician access to system admin or login denied
 - c. Inputs: technician enters username, password
 - d. Output: if correct, access provided; if incorrect, access denied
8. Driver Alertness Check
- a. Precondition: Driver is seated and hands are on the wheel.
 - b. Postcondition: System detects the driver's level of alertness.
 - c. Inputs: Driver's position; Driver's activity
 - d. Output: System provides alert if driver is drowsy or unattentive.
9. Maintenance Alert Test
- a. Precondition: Tire pressure is low or gas level is low.
 - b. Postcondition: Driver is alerted about these unfavorable conditions.
 - c. Inputs: Tire pressure readings; gas level readings.
 - d. Output: System provides alert if any of these readings are lower than expected.
10. Intersection Assistance Test
- a. Precondition: Car is approaching an intersection.
 - b. Postcondition: Car successfully navigates through the intersection.
 - c. Inputs: Blinker status, traffic light color.
 - d. Output: System provides appropriate steering or braking assistance based on inputs.
11. Adaptive Cruise Control Activation
- a. Precondition: Car is in motion and ACC is not activated.
 - b. Postcondition: ACC is activated.
 - c. Inputs: Command to activate ACC.
 - d. Output: System confirms ACC activation.
12. Lane Departure Test
- a. Precondition: Car is in motion; on a road with lanes/lines
 - b. Postcondition: Driver is alerted and car is steered back properly in the lane.
 - c. Inputs: Blinker, lane lines, car distance from lines.
 - d. Outputs: Noise alerting driver of lane departure.