InnovateX

Alset IoT Hugs the Lanes

CS347: Software Development Process

Bhagawat Chapagain, Samhita Chunduru, Neel Kulkarni, Divya Prahlad

Stevens Institute of Technology

STEVENS
INSTITUTE OF TECHNOLOGY
1870

Chapter 0 Table of Contents

Chapter 1 Introduction

About

This document is paving the road for the future with the development of a driver assisted car. The primary goal is to lay the foundation for the comprehensive software product. The main elements that are intended to be included in this first version are real-time decision-making algorithms and advanced object detection. This project embodies a crucial mission, the safety context of autonomous vehicles. The software's availability and dependability are the most important factors to consider.

Team Strengths

- **Bhagawat Chapagain** - strong writing skills, analytical, skilled in coding
- **Samhita Chunduru** - excels in project management, task-oriented, strong communication skills
- **Neel Kulkarni** - outgoing, curious, and a critical thinker
- **Divya Prahlad** - detail-oriented, patient, and excels at problem solving
- **Collectively** - experience working in Python, Java, C++, R, and Assembly languages

Mission Critical Real-Time Embedded System, Availability & Reliability

The software that will be developed is embedded within the hardware of the car and is responsible for controlling and managing its operations in real-time. The term "mission-critical" signifies that any failure or malfunction in our system can lead to critical consequences or safety risks. Therefore, our software must be designed to perform under high-stakes conditions with extreme reliability and availability. In the context of our project, availability refers to how often the service is available for use. To ensure high availability regular software updates will be implemented. Reliability pertains to the duration for which the software can run before it will fail or need to be updated. A highly reliable system operates correctly and consistently over time, even in the face of external disturbances or internal errors. To enhance reliability, rigorous testing procedures will be incorporated. We aim to maximize the time between failures to ensure that our software remains dependable throughout its lifecycle.

IoT

The incorporation of GPS technology will allow the driver-assisted car to gather data to determine its precise location and navigate effectively. This is crucial for route planning, enabling the vehicle to make informed decisions based on its current position. Advanced sensor technologies will play a pivotal role in environmental perception. Using sensors, such as LiDAR, strategically placed on the vehicle, it can capture real-time data about its surroundings, providing crucial information such as obstacles, road conditions, pedestrians, etc. Actuators will enable the vehicle to execute physical actions based on data, controlling steering, accelerating, and braking, ensuring that the vehicle responds appropriately to the data made by the IoT-driven system. It is important to note that the car will not do all movements for the driver. The system requires continuous human supervision and intervention for the decisions the car makes. Car data such as environmental observations and decisions will be securely uploaded to the company's cloud service. This repository of data allows for data analysis, monitoring, and most importantly, improvements of the system.

Software Development Process via *The Waterfall Method*

- **Roadmap Development**: Establishing a detailed plan that includes software capabilities, deadlines, and instructions. Formulating a model to guide production, incorporating budget considerations across various software layers
- **Specification Outlining**: Defining the specifications with specific tools and methods for program development in mind. Developing smaller demos, focusing on sensor and location tracking, which will be integrated into Car Control Systems (CCS) for enhanced vehicle command and safety status updates.
- **Data Collection**: Gathering data from end users to inform future Waterfall iterations and improve performance, following systematic testing phases. Utilizing data to enhance retrieval of safety status updates for the vehicle, adhering to the sequential phases of the Waterfall model.
- **Deployment**: Employing cloud services to facilitate the delivery of software updates and collect data from end users to refine future performance and support extensive testing (unit and system).

Chapter 2 Functional Architecture

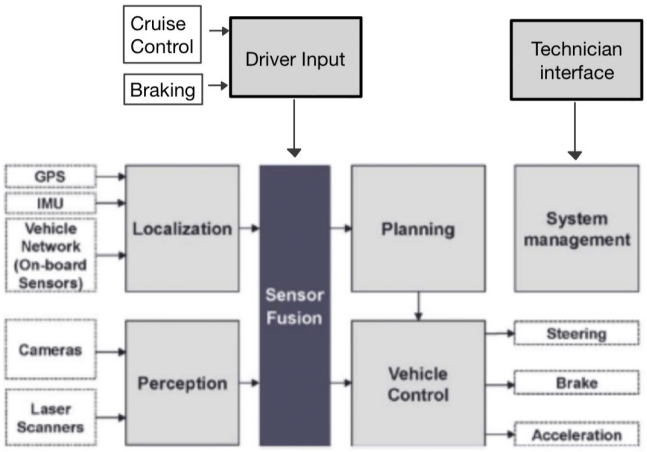
What is Functional Architecture?

The concept of functional architecture in autonomous vehicles (AVs) stems from the "functional concept" outlined by the International Functional Safety Standard for Electrical and Electronic Systems (ISO26262). This standard describes the functional concept as the system's objectives and the required interactions for achieving desired behavior. As systems become more complex, involving multiple disciplines, it's crucial to monitor how each component interacts within the system. For instance, in traditional automobiles, the invisible yet essential process of gasoline combustion in the engine drives the axles, enabling mobility. Similarly, AVs rely on an "invisible hand" — software that governs vehicle autonomy through seamless integration of sensor data and control systems.

The core of an AV's functionality lies in sensor telemetry, particularly GPS and LiDAR sensors. LiDAR sensors play a crucial role in obstacle detection, road condition assessment, and pedestrian recognition. This data, along with GPS for location and route planning, informs the vehicle's actuators. These actuators adjust the vehicle's speed, acceleration, and direction based on the sensor input. Additionally, all vehicle data is managed through cloud computing, which not only stores information but also aids in training AI to enhance vehicle autonomy continuously.

Functionality

In our functional architecture, each module has a distinct role and interfaces with others in a sequential manner. Data and control signals are passed from one module to the next, creating a smooth flow of information. This communication is depicted visually in the above data flow diagram, showing how data moves through the system. The 5 modules we will implement are Localization, Perception, Sensor Fusion, Planning, and Vehicle Control. The functionality of the Localization module is to determine the vehicle's position and orientation using GPS, IMU, and on-board sensors. The processed data is then sent to the Sensor Fusion Model. The Perception module processes visual data from cameras and laser scanners to identify objects and obstacles, and also sends its processed data to the Sensor Fusion module. The last module that sends data to Sensor Fusion is Driver Input. When the driver activates features like cruise control, turning the wheel, or applying the brakes, this input should be incorporated into the Sensor Fusion data flow.The Sensor fusion module then integrates this data to create a comprehensive understanding of the environment. Furthermore, Sensor Fusion should prioritize driver safety input over other sensor data before sending it to the Planning Module. The Planning module uses this fused data to develop driving plans and obstacle avoidance strategies, then sends planned actions to the Vehicle Control module. Vehicle Control then utilizes data from Sensor Fusion to execute these plans, by controlling the vehicle's steering, brake, and acceleration. Lastly, the Technician Interface is a new addition to our Functional Architecture. Falling under System Management, technicians must log in with a User ID and Password to retrieve log files, perform diagnostics, or update software.



Examples

Driver Assistance Systems

An example of a driver assisted system is cruise control. Cruise control assists the driver by helping them maintain a constant speed without having to constantly keep their foot on the accelerator. On long highway drives, the driver is more comfortable as there is no need for constant manual adjustment by the driver. Without the need to maintain a constant speed manually, the driver can put more focus into other things such as steering and lane changing. It is important to note the distinction between driver assisted and self driving - cruise control does not actively control steering and braking, it merely controls speed. This system can rely on network and cloud connections. More advanced systems such as adaptive cruise control may incorporate data from external sources such as GPS and traffic data. Through this, the system can recognize speed limits and anticipate changes in road conditions such as hills or curves. Thus, the system can preemptively adjust the vehicle's speed to ensure a smooth and safe driving experience. The system can also utilize cameras and sensors to track for lane departures, speed limit or stop signs, and upcoming traffic. The system will warn drivers if they are departing a lane, or if a stop sign or speed limit sign is spotted. Additionally, the system can maintain a safe following distance based on upcoming traffic.

Self-Driving

Combining cruise control, lane keeping, and automated lane changes is Tesla's Autopilot technology. When the turn signal is engaged, it changes lanes, maintains speed, and stays in its lane. By minimizing accidents and lessening fatigue during lengthy rides, this device improves safety. It performs best on highways when it has lane-keeping, braking, and accelerating capabilities. Real-time data is captured by several external cameras installed in Tesla vehicles. The onboard computer uses this data to process Autopilot functions across the local network of the car. To improve safety, add new features, and improve Autopilot operation, the camera-based system depends on cloud-based updates. Updates for the system are delivered over-the-air (OTA), eliminating the need for in-person trips to service centers to stay current. Accurate navigation, lane changes, and lane maintenance are made possible by cloud-based map data. These developments in Tesla's Autopilot system show how network and cloud connectivity can be combined to improve car performance and user experience.

System Administration

HydraNet, Tesla's neural network architecture, is split up across several parts. Without interfering with one another, each module concentrates on particular tasks, such as object detection and lane lines. Through internal communication, the network optimizes performance in a range of situations. While Hydranet itself doesn't directly rely on external cloud services, its training and fine-tuning happens on the cloud. Tesla gathers fleet data, improves neural network models and uses OTA software to deliver upgrades to cars. Continuous development and adaptation to real-world conditions are ensured by this machine learning powered by the cloud. In order to synchronize driver profiles between cars, Tesla is also transferring them to the cloud. This facilitates smooth transitions, which is advantageous for owners who own numerous Teslas or for individuals who rent or share vehicles. Personalized settings, such as seat position, climate, and media, are possible with cloud-based profiles.

Chapter 3 Requirements

3.1 Functional Requirements

3.1.1 Automatic Lane Change

Pre-Condition

- The vehicle's engine is on.
- The driver is present in the car to supervise (requires seat sensor).
- The vehicle's auto-driving mode is activated.

Requirements

- Sensor fusion detects a lane next to the vehicle is open.
- Sensor fusion detects that the car ahead is moving slow compared to the speed limit or speed of surrounding traffic.
- Vehicle control makes the car shift lanes to either left or right.

Post-Condition

- Vehicle display indicates that the vehicle has changed lanes.
- The vehicle will speed up or slow down depending on the rate of traffic ahead of it.

3.1.2 Autonomous Car Parking

Pre-Condition

- The vehicle detects it is in a parkable location.
- Requirement verification through location service data to confirm the vehicle's presence in a parkable area.
- Utilization of a surface imaging model that learns from every parking event.
- Combination of location and imaging checkers enables parking in lot areas and side street parking.

Requirements

- Detect parkable locations using location service data.
- Employ a surface imaging model for learning and accuracy improvement.
- Enable parking in diverse scenarios, including lot areas and side street parking, through location and imaging checkers combination.

Postcondition

- The vehicle successfully completes the parking maneuver in the detected parkable location using the surface imaging model and location verification, ensuring accurate and reliable parking in diverse environments.

3.1.3 Summon

Pre-condition

- The vehicle has no one in it.
- The user's location is detected outside a specified range.

Requirements

- Highly specific pathfinding.
- Utilization of Autosteering and Autonavigation.

Post-Condition

- The car will be remotely instructed to arrive at the user location.

3.1.4 Traffic Light and Stop Sign Control

Pre-Condition

- The vehicle's engine is on.
- The driver is present in the car to supervise (requires seat sensor).
- The vehicle's auto-driving mode is activated.

Requirements

- Sensor fusion detects traffic light or stop sign.
- Command center interprets the traffic light color or stop sign and sends signals to vehicle control.
- Vehicle control appropriately applies brake pressure depending on the speed of the car.

Post-Condition

- The vehicle successfully navigates through intersections according to traffic rules.

3.1.5 Autosteer on Street

Pre-Condition

- The vehicle is in driving mode on a road.
- Static obstacles (e.g., parked cars, sidewalks, street signs) are detected.
- Dynamic obstacles (e.g., other moving vehicles) are detected.

Requirements

- CCS control of steering mechanism.
- Self-correcting model to account for dynamic obstacles.

Post-Condition

- Vehicle maintains steady control and safe navigation.

3.1.6 Enhanced Cruise Control

Pre-Condition

- The vehicle's engine is on.
- The vehicle is on a road with clear lane markings.
- The vehicle is currently in motion.

Requirements

- Camera systems detect lane markings.
- Vehicle control automatically makes steering adjustments to keep the vehicle centered in its lane.

Post-Condition

- If vehicle control makes a steering adjustment, the dashboard will indicate that the driver is going close to the lane.

3.1.7 Following Distance Parameterization

Precondition

- The vehicle's engine is on.
- The vehicle is currently in motion.
- The driver is present in the car to supervise (requires seat sensor).
- The vehicle's auto-driving mode is activated.

Requirements

- CCS continuously monitors surrounding traffic conditions and calculates a safe following distance based on speed, density, and trajectory of vehicles ahead.
- Following distance adjustments should seamlessly integrate with other auto-driving features, ensuring smooth operation and enhancing overall driving experience.
- Parameterization should occur in real-time, with minimal delay, to ensure timely adjustments and prioritize safety in diverse traffic scenarios.

Postcondition

- The system adjusts the following distance based on traffic conditions.
- The vehicle maintains a safe distance from the car ahead.

3.1.8 Crash Detection

Precondition

- The vehicle's engine is on.
- The vehicle is currently in motion.

Requirements

- Prompt detection of potential collisions through analysis of surrounding objects' speed, distance, and trajectory.
- Swift calculation of required braking force based on detected collision risks and vehicle dynamics.
- Automatic application of brakes with appropriate intensity to avoid or minimize collision impact, prioritizing occupant safety.

Postconditions

- The dashboard indicates a hazard sign.
- Sensor fusion detects an imminent collision.
- Vehicle control immediately calculates the required braking force.
- Vehicle control automatically applies brakes with appropriate intensity.
- The vehicle either avoids the collision or significantly reduces speed to mitigate the impact.

3.1.9 Emergency Braking

Pre-Condition

- The vehicle's engine is on.
- The vehicle is currently in motion.

Requirements

- Dashboard indicates a hazard sign.
- Sensor fusion detects an imminent collision.
- Vehicle control immediately calculates the required braking force.
- Vehicle control automatically applies brakes with appropriate intensity.

Post-Condition

- Vehicle either ends up avoiding collision or significantly reduces speed.

3.1.10 Blind Spot Detection

Pre-Condition

- The vehicle's engine is on.
- The vehicle is currently in motion.

Requirements

- Sensor fusion monitors areas not visible to the driver.

- Vehicle control alerts driver of vehicles in blind spots during lane change attempts.

Post-Condition

- Side mirrors will light up with hazard lights.
- Vehicle audio system will make a beeping sound.

3.2 Non-functional Requirements

Reliability Requirements

NFR 1. Failure Rate - The start process implemented by our vehicle shall have a failure rate of no more than 0.0001 failure per month to ensure utmost reliability.

NFR 2. Availability - The system shall be available for use at least 99.9

NFR 3. Operating Time - Failures due to hardware or software issues should not exceed 0.1% of total operating time.

NFR 4. Fault Tolerance - The system shall detect and recover from faults such as sensor malfunctions or communication errors without ever compromising safety.

Security Requirements

SR 1. Prior to granting access to the operating controls of the vehicle, the driver's identification must be verified by the Vehicle System.
SR 2. To avoid unwanted access or tampering, the Sensor Fusion System is required to encrypt all sensor data during transmission.
SR 3. To protect passenger safety, the Vehicle System will automatically lock all of the doors if the vehicle's speed surpasses 10 mph.
SR 4. After confirming the legitimacy of the diagnostic instrument or software, the vehicle's On-Board Diagnostic System will only permit access to diagnostic data.
SR 5. In the event of a critical system failure or cyberattack, the vehicle system must put in place a fail-safe mechanism that allows the vehicle to stop safely.
SR 6. In the event that a security compromise is discovered, the car's system will notify the driver immediately.
SR 7. Every security-related incident must be recorded by the vehicle system, and only authorized persons should have access to the logs.
SR 8. All saved personal data in the vehicle system will be automatically deleted upon a transfer of ownership.

Chapter 4 Requirement Modeling

4.0.1. Use Case Scenarios

Use Case 1: Initiating Vehicle Start

Actor: Driver

Pre-Condition: The vehicle is turned off.

Post-Condition: The vehicle's systems are initialized, prepared for driving.

Trigger: The driver turns on the vehicle.

1. The driver starts the vehicle by tapping the card and pressing the start button.
2. The electrical system of the vehicle starts up.
3. The vehicle's onboard processors power up.

Exceptions:

a. If the system has problems during the initial self-checks, the engine may not start and an error alert may appear.

b. If a driver attempts to start the vehicle without tapping the card, the vehicle won't start.

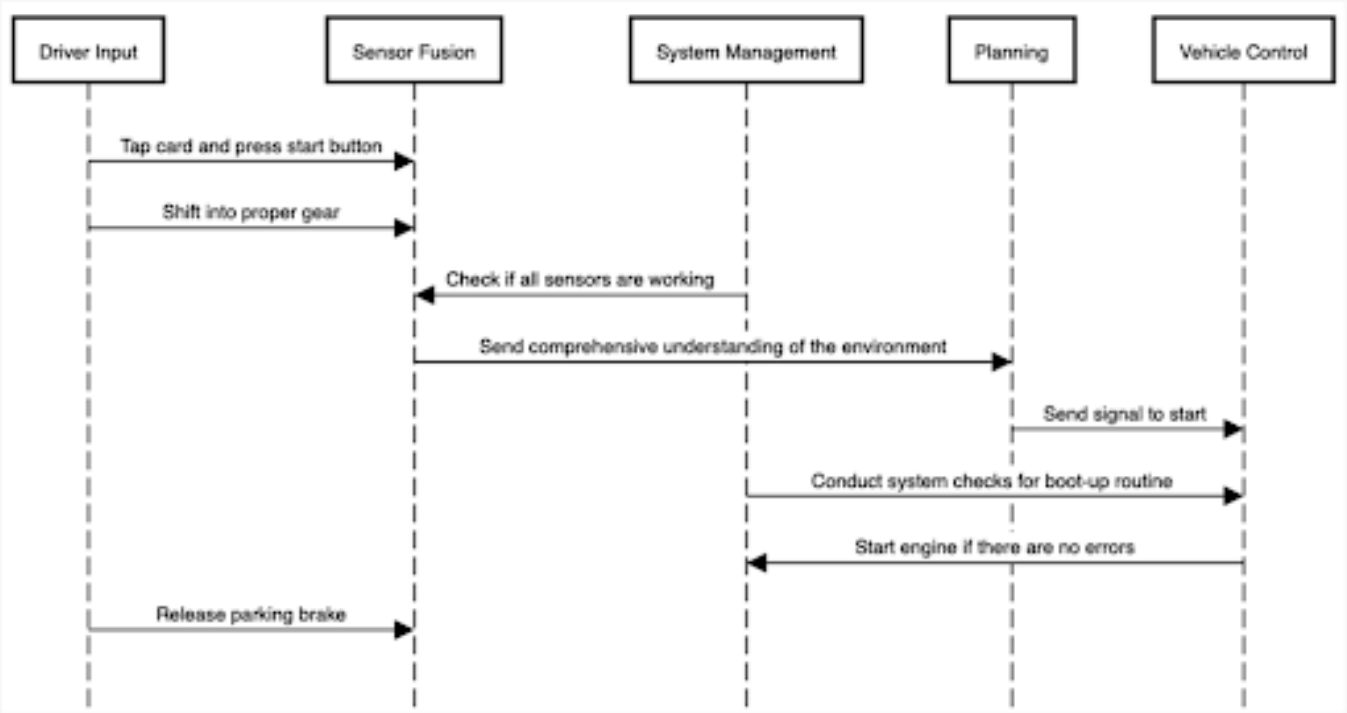


Figure 4.1: User Case 1

Use Case 2: Disengaging Cruise Control
Actor: Driver

Pre-Condition: The vehicle's cruise control is engaged and controlling its speed.

Post-Condition: Manual driving mode is resumed, and cruise control is disabled.

Trigger: The driver chooses to disengage cruise control.

1. The driver decides to disengage cruise control.
2. The driver presses the brake pedal.
3. The vehicle's cruise control system deactivates.
4. The vehicle's control is returned to the driver.
5. The dashboard displays an indicator that cruise control is off.

Exceptions:

a. If the cruise control fails to disengage, a fail-safe is activated, and manual control is restored to the driver.

b. In the event of a system error, the driver is alerted and a system check is recommended.

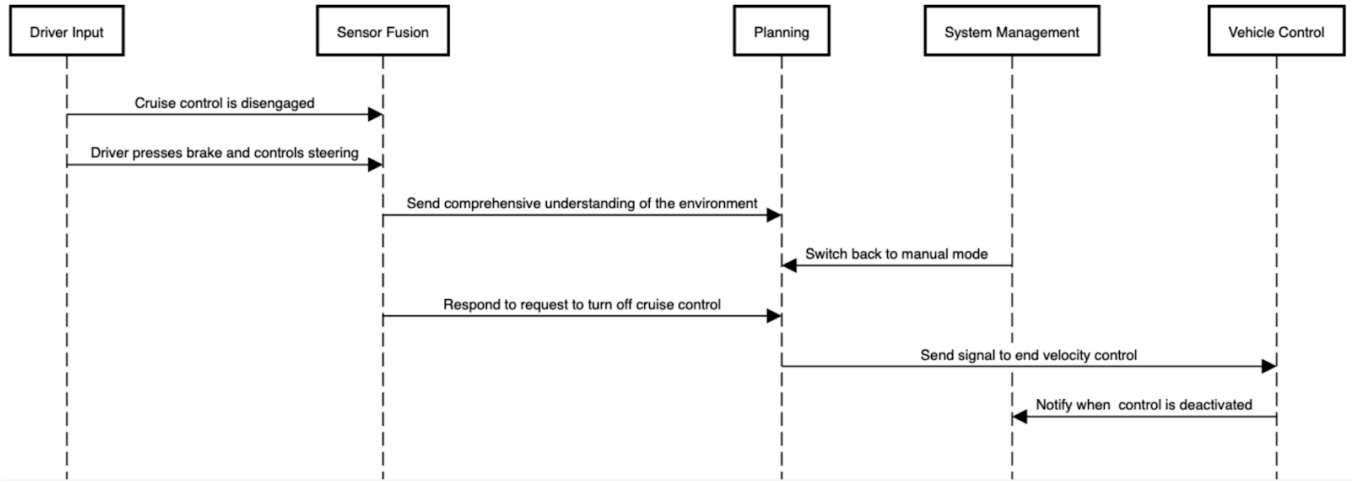


Figure 4.2: User Case 2

Use Case 3: Switching on Adaptive Cruise Control
Actor: Driver

Pre-Condition: Cruise control is not engaged; vehicle is in motion.

Post-Condition: The vehicle maintains a safe following distance automatically.

Trigger: The driver activates adaptive cruise control.

1. The driver activates the adaptive cruise control feature.
2. The vehicle's sensors determine the distance to the vehicle ahead.
3. The system sets the speed and following distance as per the driver's settings.
4. The vehicle's speed adjusts automatically to maintain the set distance.
5. The driver receives feedback via the dashboard display confirming ACC is active.

Exceptions:

a. If a sensor fails, the system alerts the driver and deactivates ACC.

b. If the vehicle in front stops suddenly, the system engages the brakes to prevent collision.

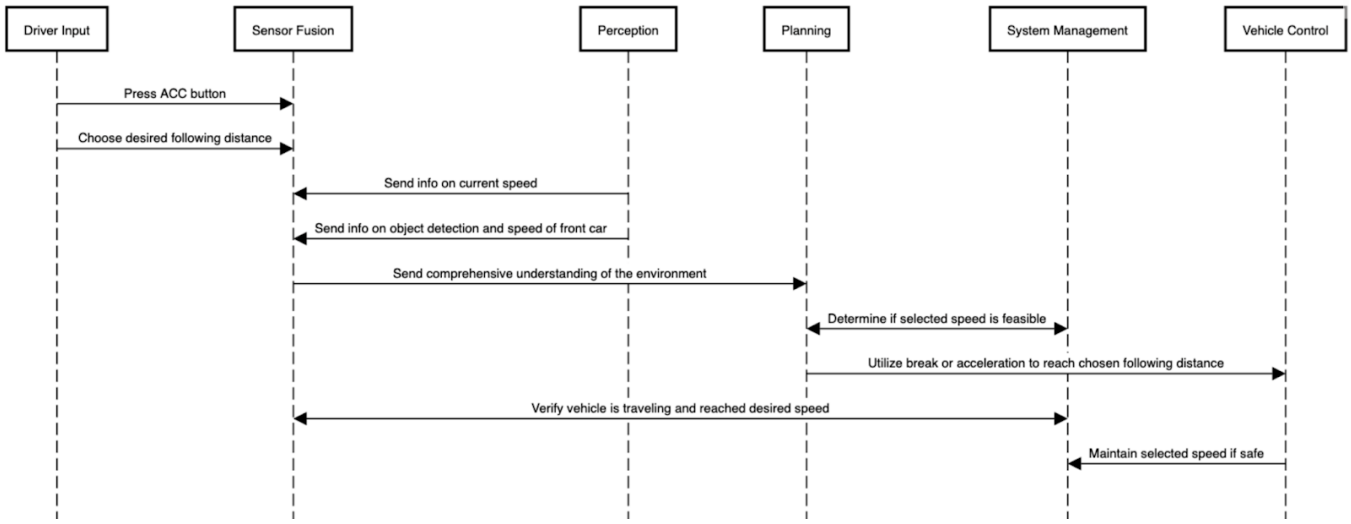


Figure 4.3: User Case 3

Use Case 4: Traffic Light and Stop Sign Control
Actor: Vehicle Control System

Pre-Condition: The vehicle is operational and in motion.

Post-Condition: The vehicle responds appropriately to traffic signals and stop signs.

Trigger: The vehicle approaches a traffic signal or stop sign.

1. The vehicle's sensors detect a traffic signal or stop sign.
2. The Sensor Fusion module interprets the traffic signal or stop sign data.
3. The vehicle's Planning module decides on the appropriate action.
4. If necessary, the vehicle slows down or stops.
5. Once the signal turns green or the way is clear at a stop sign, the vehicle proceeds.

Exceptions:

a. If the sensors fail to detect the traffic signal or stop sign, the vehicle may issue a warning and request driver intervention.

b. In the event of a malfunction, the vehicle enters a safe mode and may require manual control.

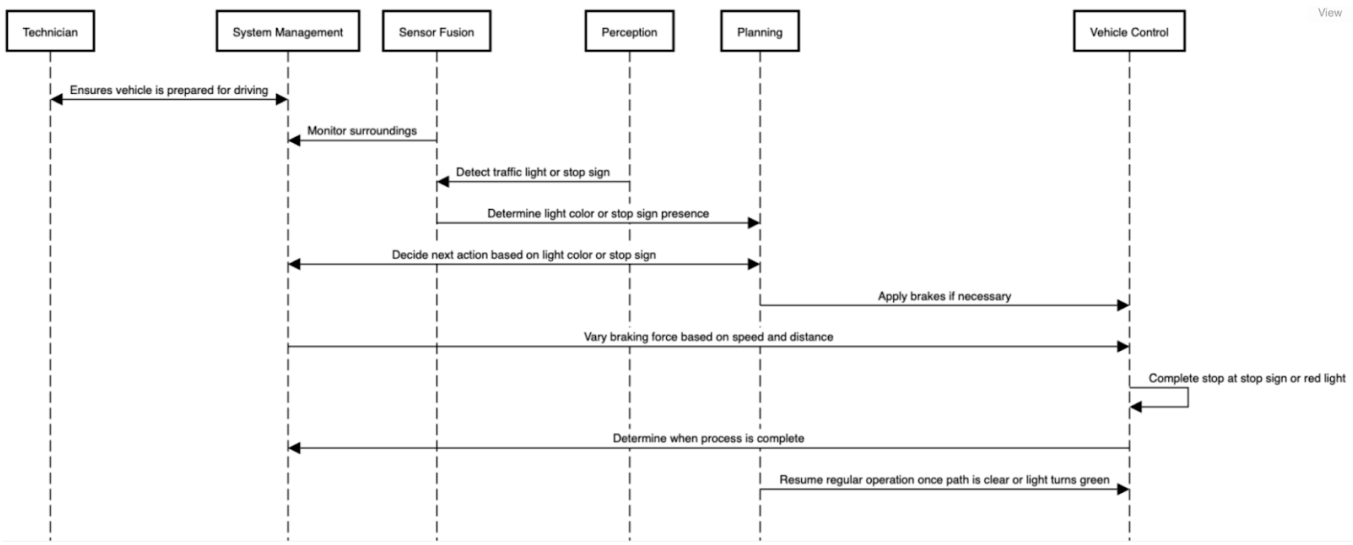


Figure 4.4: User Case 4

Use Case 5: Automated Emergency Braking
Actor: Vehicle Safety System

Pre-Condition: The vehicle's safety systems are active and operational.

Post-Condition: The vehicle takes action to prevent or mitigate a collision.

Trigger: The system detects an imminent collision risk.

1. The vehicle's sensors detect a potential collision hazard.
2. The Sensor Fusion module assesses the risk level.
3. The Planning module decides on the need for emergency braking.
4. The vehicle's braking system activates to prevent or mitigate the collision.
5. The system logs the incident for further analysis and learns from the event.

Exceptions:

a. If the hazard is detected too late for emergency braking to prevent a collision, the system may take other actions to protect occupants.

b. If a sensor or system failure occurs, the vehicle will alert the driver and may switch to manual control to ensure safety.

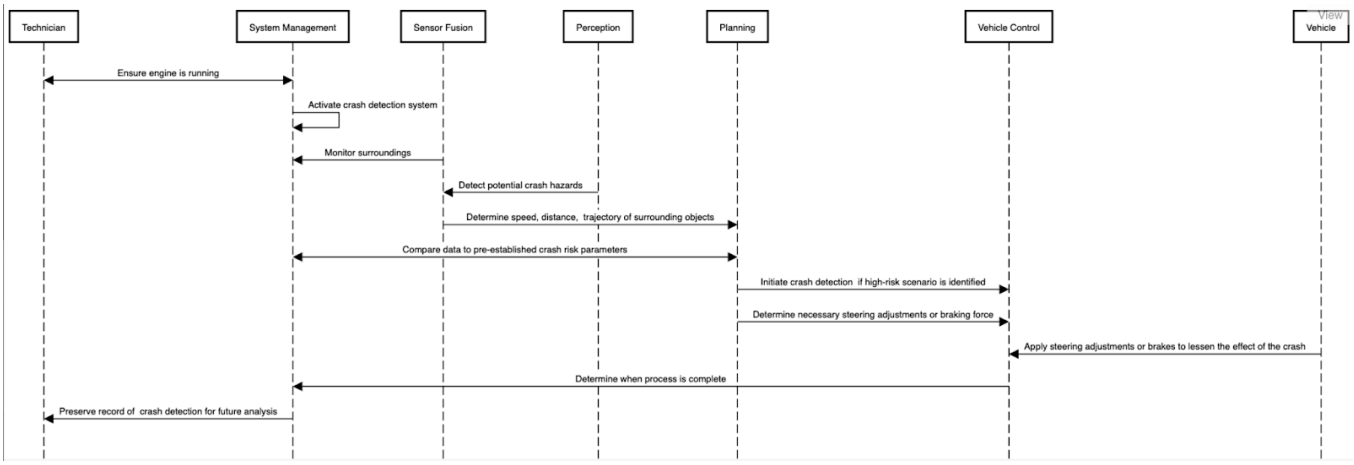


Figure 4.5: User Case 5

4.0.2. Activity Diagram

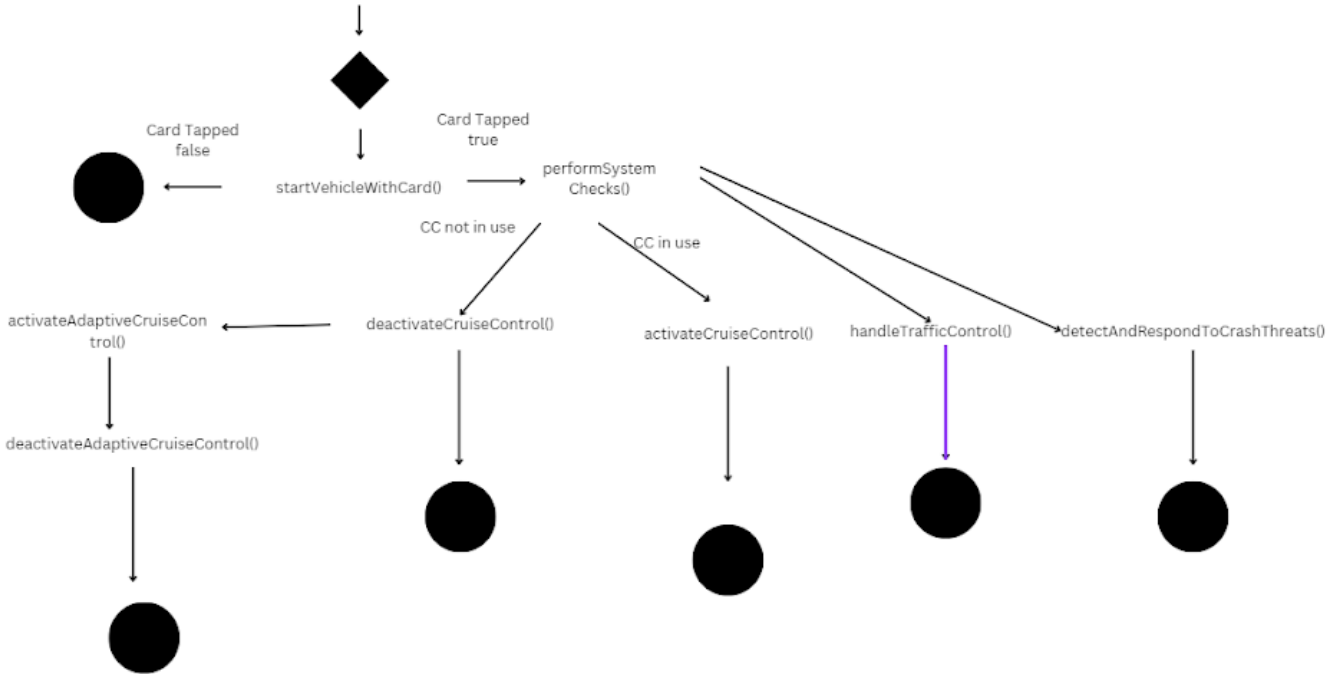


Figure 4.6: Activity Diagram

4.0.3. Classes

Classes must be defined with meaningful attributes and methods that you observed in the expected software behavior.

Listing 4.1: Vehicle class implementation

```
public class Vehicle {
    private boolean isVehicleStarted = false;
    private boolean isCruiseControlActive = false;
    private boolean isAdaptiveCruiseControlActive = false;
    private int currentSpeed = 0; // Current speed of the vehicle
    private int desiredSpeed = 0; // Desired speed set by the driver
    private int followingDistance = 0; // Desired following distance for ACC
    }

public class VehicleHealth extends Vehicle {
    private boolean isEngineHealthy = true;
    private boolean isTransmissionHealthy = true;

    public void checkEngineHealth() {
        if (!isEngineHealthy) {
            System.out.println("Engine health check failed.");
        } else {
            System.out.println("Engine is healthy.");
        }
    }

    public void checkTransmissionHealth() {
        if (!isTransmissionHealthy) {
            System.out.println("Transmission health check failed.");
        } else {
            System.out.println("Transmission is healthy.");
        }
    }
}

public class Actor {
    private String currentController; // Describes who currently controls the vehicle: "Human", "
        Autonomous System", "Remote"

    // Constructor to set the initial controller of the vehicle
    public Actor(String initialController) {
        this.currentController = initialController;
        System.out.println("Initial control set to: " + initialController);
    }



    // Method to get the current controller of the vehicle
    public String getCurrentController() {
        return currentController;
    }

    // Display current control status
    public void displayControlStatus() {
        System.out.println("The current controller of the vehicle is: " + currentController);
    }
}
```

4.0.4. State Diagrams

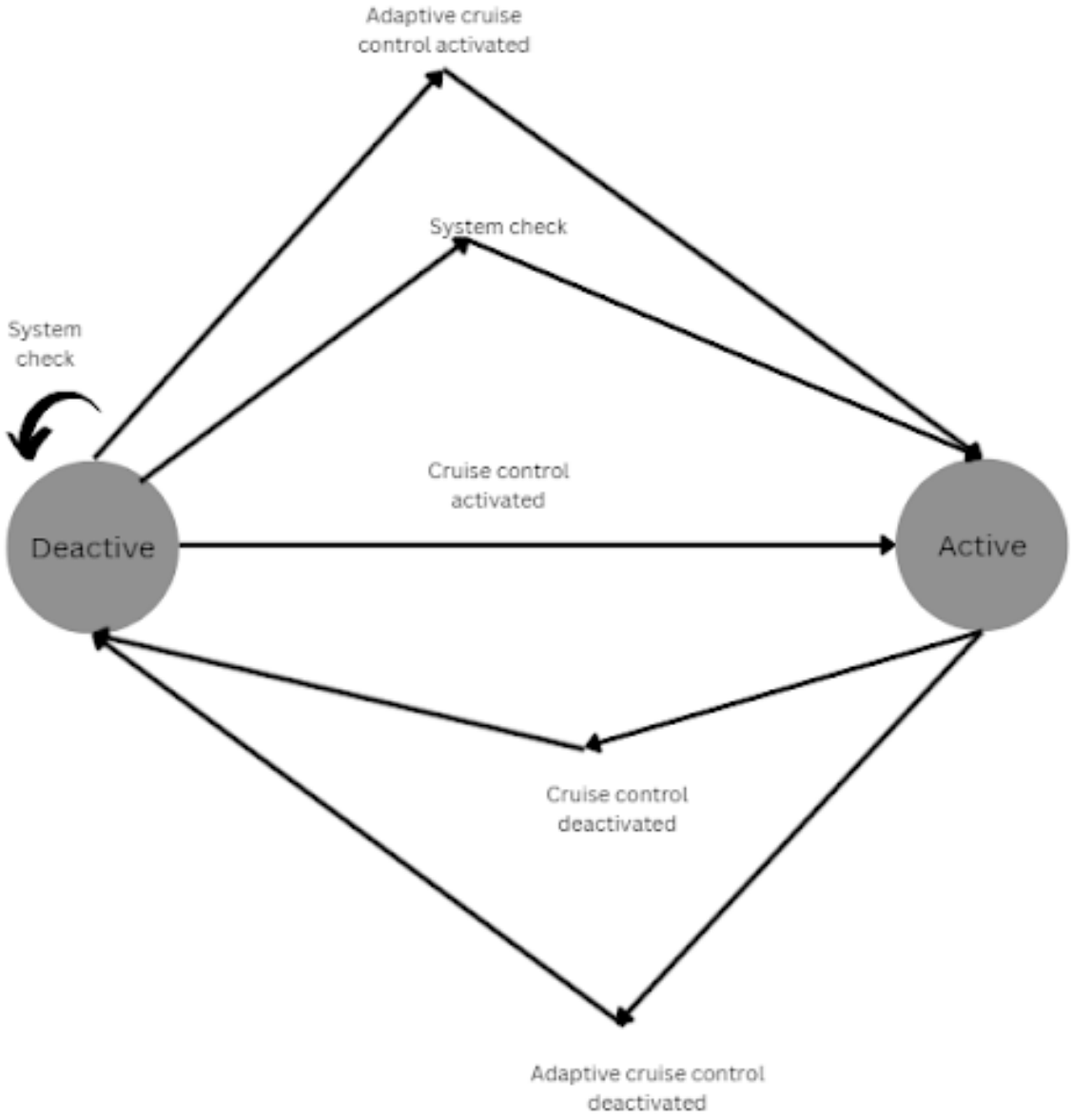


Figure 4.7: Class Diagram

Chapter 5 Design

5.1 Software Architecture

Data Centered Architecture

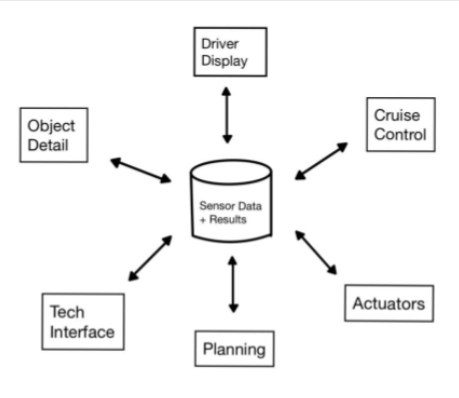
- This architecture is not fit for IOT HTL because the vehicle would require immediate, real-time processing of data to make split-second decisions. A Data Centered Architecture, particularly with its central repository model will introduce latency in data access because every decision requires communication with the central data repository.

Pros:

- Centralized security management ensures consistent security measures and updates.
- Simplifies the integration of data analysis from diverse sources such as traffic and weather conditions.

Cons:

- Potential delays due to communication with the central repository.
- Scalability issues with the central repository becoming a bottleneck as the number of vehicles increases.



Data Flow Architecture

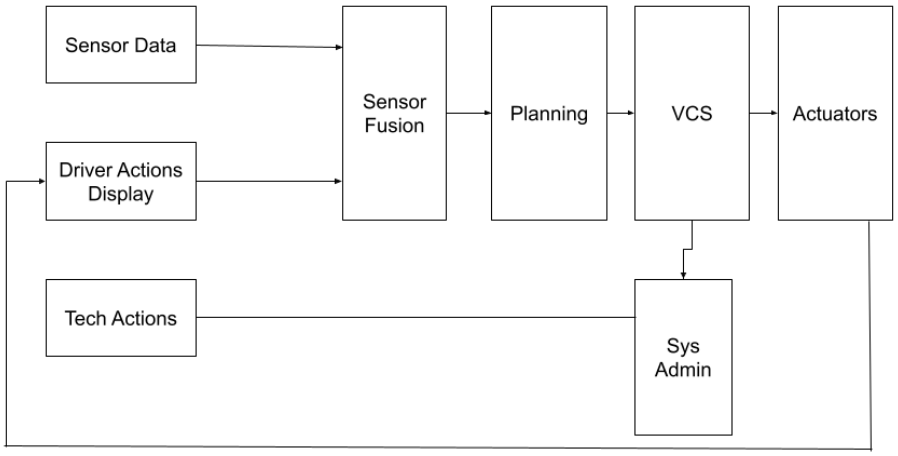
- This architecture is not fit for IOT HTL because the vehicle will operate in a highly-dynamic environment where decisions are unpredictable and need to be made in real-time. Data Flow Architecture may not be agile enough to handle the unpredictable nature of self-driving technology. The decision-making process in a self-driving vehicle involves complex algorithms that must consider multiple factors simultaneously, so it requires a more non-linear and interactive approach than Data Flow Architecture offers.

Pros:

- Provides a clear, sequential process for data handling, enhancing understandability.

Cons:

- Limited flexibility in accommodating non-linear processing.
- Sequential data processing introduces latency.



Call Return Architecture

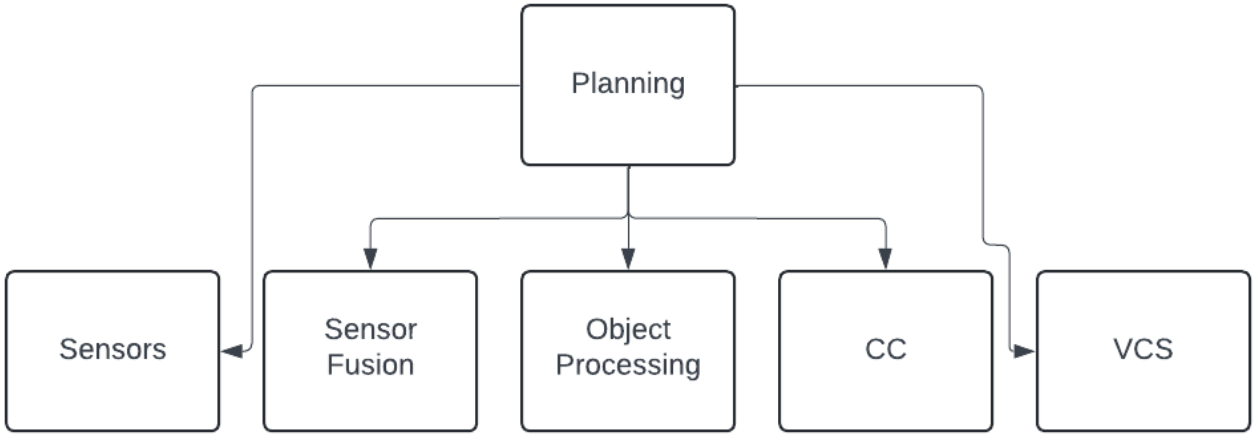
- This architecture is not fit for IOT HTL because the vehicle requires extremely fast decision making, which may not be attained with the nature of call and return operations that could introduce latency. Additionally, the vehicle's software must incorporate new features, and the static nature of this architecture with predefined calls and dependencies may limit the system's ability to adapt and scale efficiently.

Pros:

- Provides a clear and understandable flow, making the system easier to debug and test
- Encourages the development of modular components or functions that can be reused across the system, promoting code cleanliness and reducing redundancy

Cons:

- Can lead to tightly coupled system components, making it difficult to modify individual parts without affecting the others
- Handling concurrent operations efficiently can be challenging, as the architecture inherently focuses on sequential execution



Object-Oriented Architecture

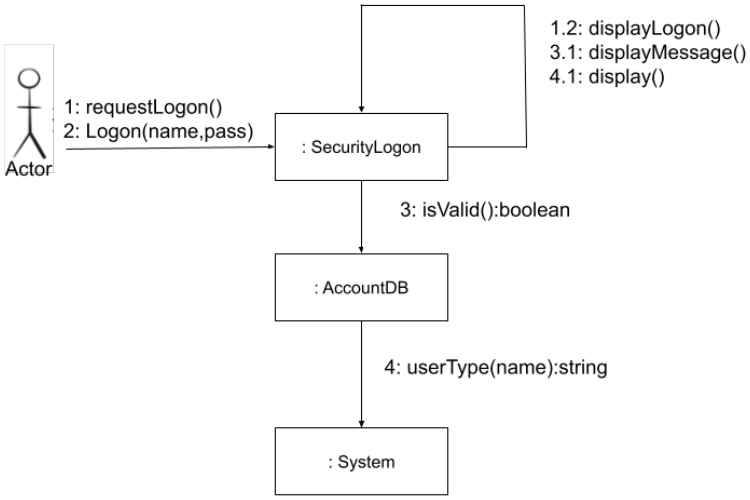
- This architecture is a fit for IOT HTL because it allows for the hierarchical structure of classes. Common functionalities can be defined in base classes, which can then be inherited and specialized by subclasses. This can speed up development significantly. Polymorphism and dynamic binding enable objects of different classes to be treated as objects of a common superclass. This allows for flexible software architecture where components can be easily swapped, extended, or modified without requiring significant changes to the overall system.

Pros:

- Objects and classes can be easily modified or extended without affecting other parts of the system, improving the system's maintainability and adaptability over time
- Inheritance allows new functionalities to be developed by extending existing classes, reducing the need to write new code from scratch and ensuring consistency

Cons:

- Understanding OOA principles and designing effective object-oriented systems can be challenging for developers new to the paradigm



Layered Architecture

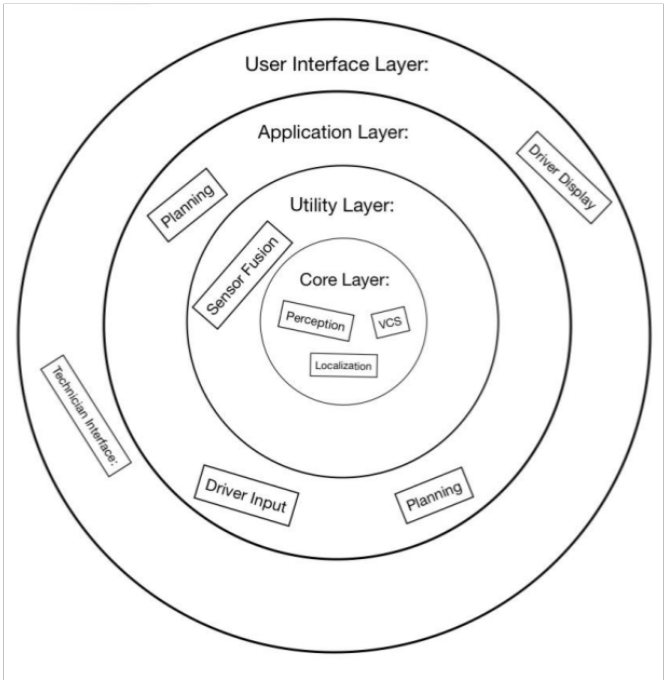
- This architecture is not a fit for IOT HTL because the vehicle integrates diverse systems like perception, decision-making, and vehicle control, which need to interact closely and efficiently. The separation enforced by Layered Architecture might hinder the high level of integration and interactivity required among these systems, limiting the vehicle's ability to function as a cohesive unit. Also, the dynamic environment and rapid technological advancements in the automotive industry require self-driving systems to be highly adaptable. Layered Architecture, with its predefined layers and communication paths, might restrict the system's ability to evolve and integrate new technologies or algorithms quickly.

Pros:

- Each layer focuses on a specific aspect of the application, making the system easier to understand, develop, and maintain
- Layers can be reused across different parts of the system or in different systems, provided they fulfill the required interfaces

Cons:

- The strict separation can make it difficult to implement features that require tight integration and coordination between different aspects of the system



Model View Controller Architecture

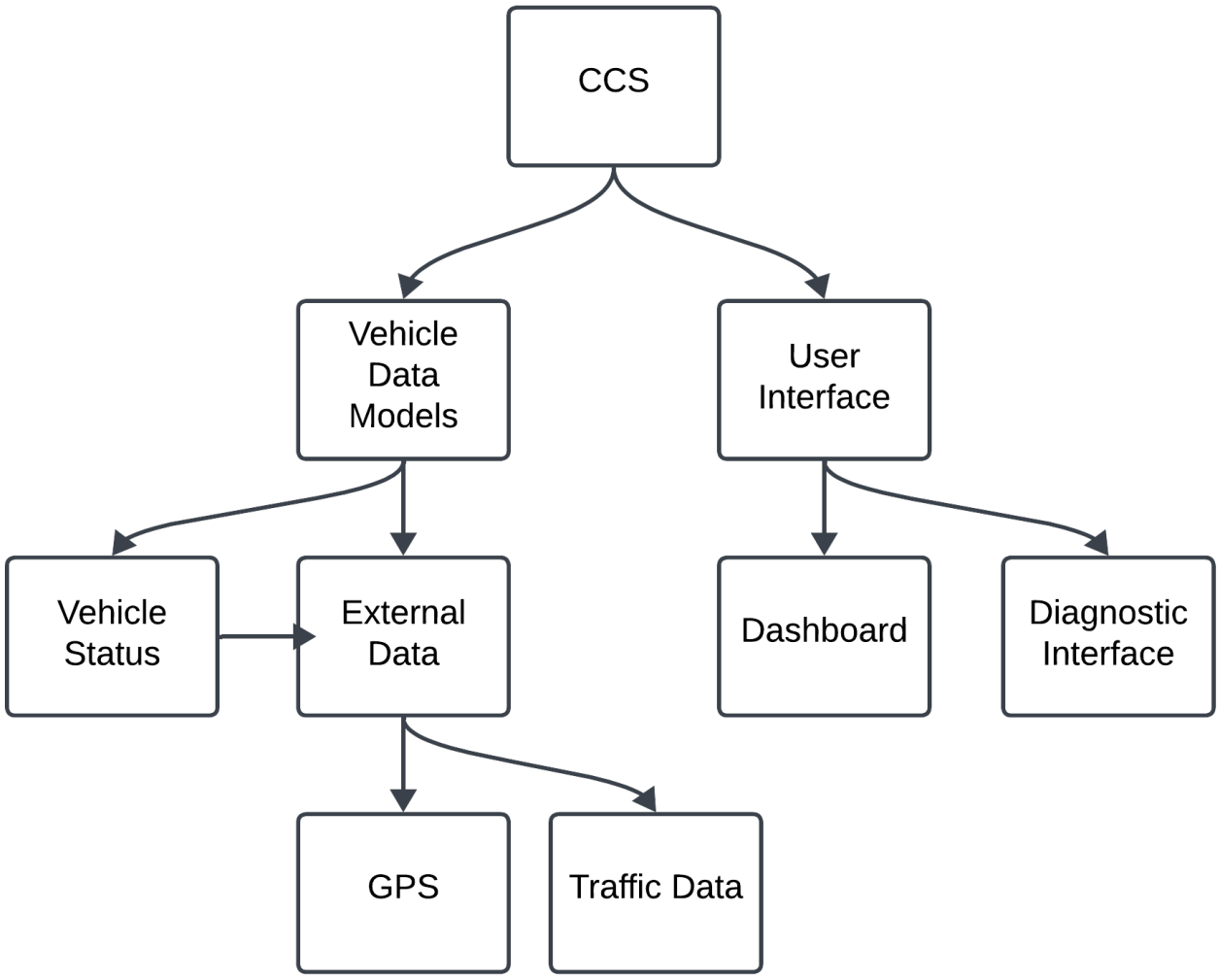
- This architecture is not a good fit because tight integration often necessitates direct communication between components that may not be sequentially aligned in a strictly separated architecture. This can complicate the implementation of features that rely on the close interplay of data and functionality across the system. In a vehicle, real-time data and features such as adaptive cruise control require seamless interaction between sensors, analysis, and actuators, which might be less efficient with tight separation.

Pros:

- Easy to maintain because it's easier to manage, update, and debug specific parts of the system without affecting others, thereby reducing the risk of introducing errors across unrelated components
- Facilitates modular development and testing, allowing teams to work on different system components independently and concurrently

Cons:

- The need for data and control messages to pass through multiple layers or components can introduce unnecessary latency and processing overhead
- Implementing and maintaining the interactions between strictly separated components can become complex, especially as the system evolves to incorporate more integrated features



Finite State Machine Architecture

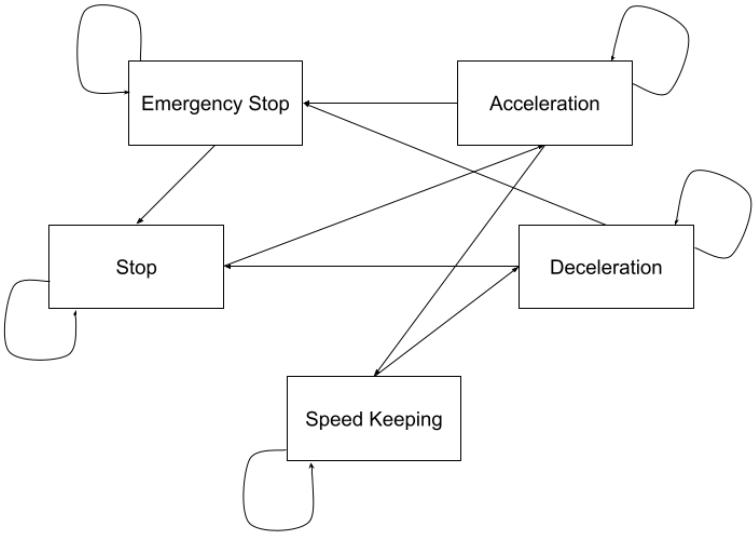
- This architecture is not a good fit because as the complexity of the system increases, the number of states and transitions in an FSM can grow exponentially, making the FSM difficult to manage, understand, and debug. Self-driving vehicles, with their myriad sensors, decision-making processes, and control actions, can quickly exceed the practical limits of an FSM in terms of complexity and scalability. Additionally, traditional FSMs are designed to be in one state at a time, which might not align well with the parallel processing requirements of self-driving vehicles that need to simultaneously monitor and respond to multiple sensors and conditions.

Pros:

- FSMs provide a clear and straightforward way to model system behavior, making it easier to predict and understand system responses to different inputs
- FSMs are effective for implementing control logic that responds to a sequence of events or conditions with a clearly defined set of responses

Cons:

- FSMs have limited ability to adapt to new or unforeseen states without significant redesign, which can be a major drawback in environments that are highly dynamic and unpredictable
- Traditional FSMs are not designed to handle multiple concurrent states or operations efficiently, which is often required in complex systems like self-driving vehicles



5.2 Interface Design

Driver Interface

public boolean startVehicleWithCard(boolean cardTapped)

- This software allows the driver to turn on the vehicle through its hardware once the card has been tapped.

public void activateCruiseControl(int speed)

- Allows the driver to engage cruise control once a speed is selected.
- Continually displays the current cruise control state.
- Vehicle sets driving speed by accessing hardware controls of the steering wheel, brake, and accelerator pedals.

public void deactivateCruiseControl()

- Allows the driver to disengage cruise control.
- Vehicle's steering, brake, and acceleration control are completely relinquished.

public void activateAdaptiveCruiseControl(int distance)

- Allows the driver to engage adaptive cruise control when a certain following distance is selected.
- Continually displays the current adaptive cruise control state.
- Vehicle sets following distance by accessing brake and accelerator hardware.

public void deactivateAdaptiveCruiseControl()

- Allows the driver to disengage adaptive cruise control.
- Vehicle's brake and acceleration controls are relinquished to the driver.

Technician Interface

private void performSystemChecks()

- Shows if there are any issues in the localization sensors, perception sensors, etc.
- Updates a log of system information that the technician can later review and use for data analysis.

public void handleTrafficControl(String signal)

- If a red light or stop sign is detected through the vehicle's perception sensors, this software will receive that input as a string literal.
- Technician's system assists in stopping the vehicle if necessary.

public void detectAndRespondToCrashThreats()

- Technician activates this setting in the software and it will continually be running in the background as the vehicle is running.

private boolean installUpdate()

- Allows technician to install the most recent version of the software to the vehicle.
- Returns True or False depending on if the update was successful or not.

5.3 Component-level Design

A driver-assisted car would have multiple parts in its system. Cameras, radar, lidar, and other sensors are utilized to sense the surroundings. These sensors are designed to accurately perceive the environment around the vehicle, collecting data about other vehicles, pedestrians, road signs, and lane markings. For instance, lidar sensors use light in the form of a pulsed laser to measure distances, creating precise 3D representations of the environment, which is crucial for navigation and obstacle detection. On the other hand, radar sensors use radio waves to determine the velocity, range, and angle of objects, which is particularly useful for adaptive cruise control and emergency braking systems. Control systems analyze and make decisions based on the data interpreted by these sensors. These choices are subsequently carried out by actuators, which modify the steering, braking, and acceleration of the car. The system

also has computing units that carry out intricate computations needed for features like object detection and course planning, as well as a database for keeping track of data such as maps and historical driving records. The Internet of Things (IoT) network plays a significant role in this system. It allows the vehicle to communicate with external systems such as traffic management systems, GPS satellites, and even other vehicles. This connectivity enhances the vehicle's perception and understanding of its environment beyond the immediate vicinity that the onboard sensors can cover. For instance, real-time traffic data can be used to optimize route planning, while GPS data is essential for localization. Numerous connectors allow these parts to cooperate and interact with one another. Sensor data and control signals can be sent between components via a data bus. The vehicle and external systems, such as traffic management and GPS satellites, can communicate due to the network protocols. A number of restrictions control how these components are integrated. Parts have to abide with car safety regulations. To achieve real-time requirements, the system needs to process data and respond under severe time constraints. Components also need to function within the vehicle's power limitations. This guarantees the system's efficiency, safety, and reliability.

- **Economy:** In software design, the economy concept highlights the importance of simplicity and efficiency. A crucial technique for accomplishing this is abstraction. We can manage complexity and concentrate on the larger picture by abstracting away the specifics. This may result in a more resource-efficient code that is also easier to maintain and comprehend.
- **Visibility:** This is the degree to which decision-makers in the system's design are easily understood by developers and other stakeholders. This entails being aware of the system's architecture, how its parts interact with each other, and the rationale behind design decisions. Increased visibility can result in higher software quality overall, simpler maintenance, and more efficient team communication.
- **Spacing:** This refers to the distinct division of various concerns inside the system. This can lessen the possibility of hidden dependencies, which could make the system more difficult to comprehend and maintain. A modular design is easier to develop, test, and maintain when the system is clearly divided into discrete sections, each with its own responsibility.
- **Symmetry:** An architecture that is balanced and consistent is easier to understand and frequently results in a more effective use of available resources.
- **Emergence:** Self-organized, emerging behavior should be supported by the architecture of the system. Software designs that are more scalable, effective, and inexpensive may result from this. Algorithms for machine learning, for instance, might be used to gain knowledge from driving data and gradually enhance system performance.

Listing 6.1: Vehicle class implementation (cont.)

```
// Start vehicle with card tap
    public boolean startVehicleWithCard(boolean cardTapped) {
        if (cardTapped) {
            isVehicleStarted = true;
            performSystemChecks();
            return true;
        }
        return false;
    }

    // Method to hand over control to another actor
    public void transferControl(String newController) {
        if (!newController.equals(currentController)) {
            System.out.println("Transferring control from " + currentController + " to " + newController)
                ;
            currentController = newController;
        } else {
            System.out.println("Control is already with the " + newController);
        }
    }

    // Perform initial system checks
    private void performSystemChecks() {
        // System c    // Activate cruise control
    public void activateCruiseControl(int speed) {
        if (!isVehicleStarted || speed <= 0) {
            System.out.println("Vehicle must be started and speed must be positive to activate cruise
                control.");
            return;
        }
        this.desiredSpeed = speed;
        this.isCruiseControlActive = true;
        System.out.println("Cruise control activated at " + desiredSpeed + " km/h.");
    }hecks logic
        System.out.println("System checks passed. Vehicle is ready to drive.");
    }



    // Deactivate cruise control
    public void deactivateCruiseControl() {
        if (isCruiseControlActive) {
            isCruiseControlActive = false;
            System.out.println("Cruise control deactivated.");
        }
    }

    // Activate adaptive cruise control
    public void activateAdaptiveCruiseControl(int distance) {
        if (!isVehicleStarted) {
            System.out.println("Vehicle must be started to activate adaptive cruise control.");
            return;
        }
        this.followingDistance = distance;
        this.isAdaptiveCruiseControlActive = true;
        System.out.println("Adaptive Cruise Control activated with a following distance of " + distance +
             " meters.");
    }

    // Deactivate adaptive cruise control
    public void deactivateAdaptiveCruiseControl() {
        if (isAdaptiveCruiseControlActive) {
            isAdaptiveCruiseControlActive = false;
            System.out.println("Adaptive Cruise Control deactivated.");
        }
    }

    // Start navigation to a specific destination
```

```
66      public void startNavigation(String destination) {
67          System.out.println("Navigation to " + destination + " started. Calculating route...");
68          // Simulated delay
69          try {
70              Thread.sleep(2000); // Simulate route calculation
71          } catch (InterruptedException e) {
72              Thread.currentThread().interrupt();
73          }
74          System.out.println("Route calculated. Proceed to destination.");
75      }
76
77      // Monitor lane position
78      public void monitorLanePosition() {
79          System.out.println("Monitoring lane position...");
80          // Simulated monitoring
81          try {
82              Thread.sleep(1000); // Simulate monitoring process
83          } catch (InterruptedException e) {
84              Thread.currentThread().interrupt();
85          }
86          System.out.println("Vehicle is correctly aligned within the lane.");
87      }
88      // Handle traffic light and stop sign control
89      public void handleTrafficControl(String signal) {
90          if (signal.equals("RED") || signal.equals("STOP_SIGN")) {
91              System.out.println("Stopping vehicle for red light or stop sign.");
92              // Logic to stop vehicle
93          } else if (signal.equals("GREEN")) {
94              System.out.println("Green light detected. Proceeding.");
95              // Logic to proceed
96          }
97          // Additional logic for YELLOW light or other conditions can be added here
98      }
99
100     // Basic crash detection and mitigation system
101     public void detectAndRespondToCrashThreats() {
102         // This method would involve complex sensor data analysis in a real-world application
103         System.out.println("Scanning for potential crash threats.");
104         boolean threatDetected = false;
105         // Placeholder for threat detection logic
106
107         if (threatDetected) {
108             System.out.println("Crash threat detected. Initiating avoidance maneuvers.");
109             // Placeholder for crash avoidance logic
110         }
111     }
112     // Install Update
113     public void installUpdate()
114     {
115         return;
116     }
117
118     // Main method for demonstration purposes
119     public static void main(String[] args) {
120         Vehicle myVehicle = new Vehicle();
121         // Example usage
122         myVehicle.startVehicleWithCard(true);
123         myVehicle.activateCruiseControl(50);
124         myVehicle.handleTrafficControl("RED");
125         myVehicle.detectAndRespondToCrashThreats();
126         myVehicle.deactivateCruiseControl();
127     }
128
129
130     public class VehicleControlSimulation {
131     public static void main(String[] args) {
132         Actor vehicleActor = new Actor("Autonomous System");
133         vehicleActor.displayControlStatus();
134
135         // Scenario: Autonomous System hands over control to Human
136         vehicleActor.transferControl("Human");
```

```
137         vehicleActor.displayControlStatus();
138
139         // Scenario: Human hands over control back to Autonomous System
140         vehicleActor.transferControl("Autonomous System");
141         vehicleActor.displayControlStatus();
142
143         // Attempt to transfer control to the current controller
144         vehicleActor.transferControl("Autonomous System");
145     }
146 }
147 }
```

Chapter 7 Testing

7.1 - Validation Testing

Reliability Requirements

7.1.1 Functional Requirements

7.1.1.1 Lane Detection

- **Pre-Condition:** Vehicle's engine is on, driver is present in the car to supervise, and the vehicle is in drive.
- **Input:** Real-time video feed from the vehicle's cameras, current speed and direction of the vehicle, data from sensors detecting road markings and surrounding conditions.
- **Output:** Identification and tracking of lane markings.
  - Continuous adjustment of the steering angle to maintain the vehicle within the lane.
  - Visual and/or audio alerts if the vehicle begins to drift without signaling.

7.1.1.2 Automatic Lane Change

- **Pre-Condition:** Vehicle's engine is on, driver is present in the car to supervise, vehicle's auto-driving mode is activated.
- **Input:** Date and time of attempt, current speed of vehicle, traffic conditions data including speeds of surrounding vehicles, data from sensors indicating an open lane.
- **Output:**
  - Vehicle changes lanes to the left or right.
  - System displays confirmation of lane change.
  - Speed of the vehicle is updated.
  - Vehicle maintains a distance proportional to 10ft for every 10 MPH.
  - Speed set to speed limit, distance calculated as $10ft \times (\text{speed}/10)$.

7.1.1.3 Autonomous Car Parking

- **Pre-Condition:** Vehicle detects it is in a parkable location, requirement verification through location service data to confirm the vehicle's presence in a parkable area, utilization of a surface imaging model that learns from every parking event, combination of location and imaging checkers enables parking in lot areas and side street parking.
- **Input:** Date and time of attempt, current location coordinates of vehicle, available data from sensors on parking space dimensions.
- **Output:**
  - Vehicle autonomously maneuvers into the parking spot.
  - System displays confirmation of parking.
  - Vehicle remains stationary.
  - Speed set to 0.

7.1.1.4 Navigation System

- **Pre-Condition:** Vehicle's navigation system is enabled, destination coordinates are input by the driver or selected via interface of the vehicle
- **Input: Current location and destination coordinates, real-time traffic data and route conditions**
- **Output**: Calculates the most efficient route to the destination
  - Provides turn-by-turn navigation instructions through visual and audio prompts
  - Adjusts route in real-time based on traffic and road conditions.

7.1.1.5 Summon

- **Pre-Condition:** The vehicle has no one in it, the user's location is detected outside a specified range.
- **Input:** Date and time of summon request, user's current location data, vehicle's current location and status data.
- **Output:**
  - Vehicle autonomously drives to the user's location.
  - System status updates are sent to the user's device during the summon process.
  - System displays feedback confirmation of arrival upon arrival; arrival distance will vary as the vehicle will search for suitable locations to park, so distance to user will vary between 0ft-100ft.
  - Speed set to 0, distance varies between 0-100ft.

7.1.2 - Non-Functional Requirements

- **7.1.2.1 Failure Rate**
  - **Pre-Condition:** Vehicle has been in use at least twice.
  - **Input:** Systems log with information on all start process successes and failures.
  - **Output:** Calculate ratio of start process successes to failures for all recorded instances of the start process. If the ratio is greater than 0.0001, an alert is raised indicating high failure rates.
- **7.1.2.2 Availability**
  - **Pre-Condition:** Vehicle has been in use for at least 30 days.
  - **Input:** Systems log information on system availability, including uptime (running time) and total time.
  - **Output:** Calculate ratio of system uptime to total time. If the ratio is less than 99.9%, an alert is raised indicating low system availability.
- **7.1.2.3 Operating Time**
  - **Pre-Condition:** All hardware and software issues are reported in the system log throughout use of the vehicle.
  - **Input:** Systems log information on hardware and software issues, as well as total time taken to fix issues.
  - **Output:** Evaluate the system log and calculate the total time taken for fixing hardware and software bugs. If failures due to hardware or software issues exceed 0.1% of total operating time, raise an alert indicating low levels of operation time due to hardware and software issues.
- **7.1.2.4 Fault Tolerance**
  - **Pre-Condition:** A fault such as sensor malfunction or communication error occurs.
  - **Input:** Systems log information or error message.
  - **Output:** Evaluate if the system response continues safe operation of the vehicle and recovers from the fault. If the system fails to recover from the fault, inform the technician, and make a report on the system log indicating the fault that the system failed to recover from.

Security Requirements

- **7.1.3.1 Identity Check**
  - **Pre-Condition:** The driver attempts to access the operating controls of the vehicle.
  - **Input:** Driver's biometric data or secure digital identification token.
  - **Output:** Verification status of the driver's identity, including a secure log entry specifying the time, date, and result of the identification attempt.
- **7.1.3.2 Sensor Data Encryption**
  - **Pre-Condition:** The Sensor Fusion System is actively transmitting sensor data.
  - **Input:** Continuous streams of raw sensor data, including telemetry and environmental metrics.
  - **Output:** Encrypted data stream with a detailed log of the encryption process, including timestamp and encryption protocol used.
- **7.1.3.3 Vehicle Speed Detection and Door Locking**
  - **Pre-Condition:** Vehicle's motion is detected by the onboard speed sensors.
  - **Input:** Real-time speed data continuously monitored by vehicle speed sensors.
  - **Output:** Automated door locking system activation upon reaching threshold speed of 10 mph, with system logs recording the action and speed at time of locking.
- **7.1.3.4 Diagnostic Data Access Control**
  - **Pre-Condition:** A request for access to the vehicle's diagnostic data is made by a diagnostic tool.
  - **Input:** Diagnostic tool credentials and access level request.
  - **Output:** Access control decision, detailed transaction logs including the identity of the diagnostic tool, time of request, and access response.
- **7.1.3.5 Fail-Safe Mechanism for Critical Failures**
  - **Pre-Condition:** Detection of a critical system failure or potential cyberattack.
  - **Input:** Alerts from system monitors, diagnostic errors, or cyber threat detection systems.
  - **Output:** Engagement of vehicle's fail-safe protocols, including detailed logs of the event, actions taken, and systems affected.
- **7.1.3.6 Immediate Notification of Security Compromises**
  - **Pre-Condition:** An unauthorized access or security compromise is detected within the vehicle's system.
  - **Input:** Alert from the vehicle's security monitoring systems, including details of the breach.
  - **Output:** Immediate notification to the driver and security team, detailed security breach report including time, nature, and severity of the compromise.
- **7.1.3.7 Recording and Access Control of Security Incidents**
  - **Pre-Condition:** A security-related incident is detected involving any part of the vehicle system.
  - **Input:** Data pertaining to the incident, including type of incident, systems involved, and any user interactions.
  - **Output:** Secure logging of the incident in an encrypted log file, access to which is strictly controlled and monitored.
- **7.1.3.8 Deletion of Personal Data Upon Ownership Transfer**
  - **Pre-Condition:** Official initiation of vehicle ownership transfer.
  - **Input:** Verification of ownership transfer from legal and system databases.
  - **Output:** Systematic and secure deletion of all personal data from the vehicle's memory, with a confirmation log entry that includes details of the data purged and timestamp.

7.2 - Scenario-based Testing

7.2.1.1 Initiating Vehicle Start

- **Pre-Condition:** The vehicle is turned on.
- **Input:** The driver taps the card and presses the start button.
- **Validation:**
  - **Test Case 1:**
    - Description: Check if the electrical system of the vehicle starts up when the driver taps the card and presses the start button.
    - Expected Result: The electrical system of the vehicle starts up.
- **Invalidation:**
  - **Test Case 1:**
    - Description: Check the system's response when there are problems during the initial self-checks.
    - Expected Result: The engine does not start and an error alert appears.

7.2.1.2 Disengaging Cruise Control

- **Pre-Condition:** The vehicle's cruise control is engaged and controlling its speed.
- **Input:** The driver decides to disengage cruise control and presses the brake pedal.
- **Validation:**
  - **Test Case 1:**
    - Description: Check if the vehicle's control is returned to the driver after cruise control is disengaged.
    - Expected Result: The vehicle's control is returned to the driver.
- **Invalidation:**
  - **Test Case 1:**
    - Description: Check the system's response when the cruise control fails to disengage.
    - Expected Result: A fail-safe is activated, and manual control is restored to the driver.

7.2.1.3 Switching on Adaptive Cruise Control

- **Pre-Condition:** Cruise control is not engaged; vehicle is in motion.
- **Input:** The driver activates the adaptive cruise control feature.
- **Validation:**
  - **Test Case 1:**
    - Description: Check if the vehicle's sensors determine the distance to the vehicle ahead when the driver activates ACC.
    - Expected Result: The vehicle's sensors determine the distance to the vehicle ahead.
- **Invalidation:**
  - **Test Case 1:**
    - Description: Check the system's response when a sensor fails.
    - Expected Result: The system alerts the driver and deactivates ACC.

7.2.1.4 Traffic Light and Stop Sign Control

- **Pre-Condition:** The vehicle is operational and in motion.
- **Input:** The vehicle approaches a traffic signal or stop sign.
- **Validation:**
  - **Test Case 1:**
    - Description: Check if the vehicle's sensors detect a traffic signal or stop sign when the vehicle approaches it.
    - Expected Result: The vehicle's sensors detect a traffic signal or stop sign.
- **Invalidation:**
  - **Test Case 1:**
    - Description: Check the system's response when the sensors fail to detect the traffic signal or stop sign.
    - Expected Result: The vehicle issues a warning and requests the driver's intervention.

7.2.1.5 Automated Emergency Braking

- **Pre-Condition:** The vehicle's safety systems are active and operational.
- **Input:** The system detects an imminent collision risk.
- **Validation:**
  - **Test Case 1:**
    - Description: Check if the vehicle's sensors detect a potential collision hazard.
    - Expected Result: The vehicle's sensors detect a potential collision hazard.
  - **Test Case 2:**
    - Description: Check if the Sensor Fusion module assesses the risk level correctly.
    - Expected Result: The Sensor Fusion module correctly assesses the risk level.
  - **Test Case 3:**
    - Description: Check if the Planning module decides the need for emergency braking correctly.
    - Expected Result: The Planning module correctly decides on the need for emergency braking.
  - **Test Case 4:**
    - Description: Check if the vehicle's braking system activates to prevent or mitigate the collision.
    - Expected Result: The vehicle's braking system activates to prevent or mitigate the collision.
- **Invalidation:**
  - **Test Case 1:**
    - Description: Check the system's response when the hazard is detected too late for emergency braking to prevent a collision.
    - Expected Result: The system takes other actions to protect occupants.
  - **Test Case 2:**
    - Description: Check the system's response when a sensor or system failure occurs.
    - Expected Result: The vehicle alerts the driver and may switch to manual control to ensure safety.
  - **Test Case 3:**
    - Description: Check the system's response when the sensors detect a false positive collision hazard.
    - Expected Result: The system does not activate the emergency braking system unnecessarily.

END OF DOCUMENT