The DEAN's List: An IoT Driver Assistance System

Daniel Kim, Eric Park, Andrew Kim, Noah Rombough

Table of Contents

Section 1: Introduction

The concept of a driverless vehicle is quickly becoming not only mainstream, but technologically feasible, and Alset's goal of being at the forefront of this revolution is highly achievable. Alset's goal would be to not only offer functionality, but a level of safety that either matches or surpasses human driving ability. The jump from driver to driverless, however, is much more complicated than simply taking humans away from the steering wheel. Driverless vehicles means putting the lives of humans in the hands of sensors and algorithms, making reliability, effectiveness, and failsafes paramount.

In today's automobiles, cruise control is a must-have feature. Long vehicle trips would be significantly more exhausting without it, at least for the driver. The goal of a cruise control system is to control the throttle-accelerator pedal linkage to accurately maintain a speed chosen by the driver without any outside intervention. It adjusts the throttle position to regulate the speed of your car, exactly like we do ourselves. There is less possibility of the driver accelerating and driving past other drivers. It is also able to lower the amount of fuel the driver uses since keeping a set speed uses less fuel than always changing your speed, which can also allow the driver to save some money on gas and maximize fuel economy. One thing we can improve on the typical cruise control feature is implementing adaptive cruise control (ACC) to it, which is an advanced sort of cruise control that automatically slows and speeds up to keep up with the vehicle in front of the driver. The front radar sensor would detect traffic ahead of the car and control the speed of the it, adjusting itself so that it is constantly 2-3 seconds behind the car in front of it. The software automatically detects any cars around the driver and calculates and adjusts the distance and speed needed for any kind of situation, such as if another vehicle cuts in front of the driver. For safety insurance, ACC will have a visual and acoustic warning whenever a car is too close in front that will be accompanied by a brief braking jolt or a complete halt if required. Assisted/self-parking is also a feature we believe is crucial to implement. Starting off with just assisted parking, we hope to eventually evolve a self-parking feature.

In addition to sensors, we would also incorporate cameras to help improve the safety that our autonomous IoT vehicle provides. In order for our vehicle to drive safely, it would use cameras to recognize any potential hazards that threaten the safety of our passengers, such as pedestrians, deer, reckless drivers, and more. By recognizing these threats, the vehicle can then adjust its driving as necessary, which may be to slow down, stop, or even change lanes or directions to avoid any risk. This way our vehicle would be able to implement defensive driving practices at all times. Of course, the cameras would also be used to help obey traffic laws, such as recognizing traffic lights and signs.

In our iterative design, we commit to constant customer feedback being a part of our design input. The agile process perfectly accounts for this, and as we settle on a specific approach to the development process, we will abide by the principles of agile development as a whole. This will allow us to take advantage of the low cost of change, and revise our plans/objectives as optimally and as frequently as needed. The most important principle, in our opinion, being the inclusion of consistent customer feedback. We expect that including feedback as main development input will inevitably lead to the adoption of new features, or the tweaking or removal of others. While this approach may make it more difficult to predict resource costs on the offset, our philosophy is that customer satisfaction with the end product is more important than providing perfectly accurate timelines or cost estimates. Obviously, a fully autonomous car will take years to perfect, but we aim to deliver our most important features within the next 3 months, and continue from there.

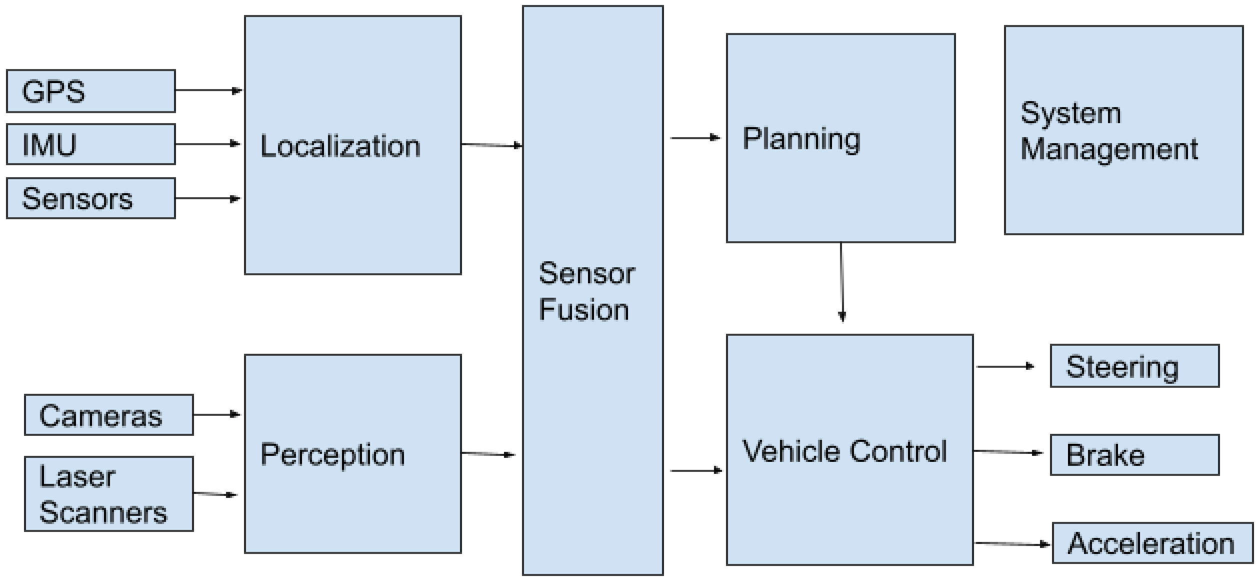
Our group members represent a broad range of skills that will aid in the project development. Noah has strengths in presenting, communication, and coding. Eric offers skills in communication, problem solving, and coding. Daniel complements the team with his communication, coding, and problem solving skills. Andrew adds to the team with his communication, patience, and detail-oriented skills.

Section 2: Functional Architecture

At its core, the vehicle is a centralized processor connected to the environment via a network of cameras and sensors for input and electric motors within the car's physical components to regulate output. These hardware and software components aim to provide two key realms of functionality to the vehicle: driver assistance and autonomous driving. Autonomous driving acts as an extension of driver assistance, using the same methods of collecting and analyzing data through hardware, but providing physical changes of the vehicle’s controls as well. With autonomous driving, we aim to eliminate the need for the active physical control of a human driver.

The architecture of autonomous vehicles can be broken down into “software architecture” and “functional architecture”. In this section we will focus on the functional side of things. Functional architecture can be defined as “specification of the intended functions and their interactions necessary to achieve the desired behavior”. Some of the core components that make up functional architecture are perception, planning, localization and vehicle control. Perception makes the use of sensors, such as cameras, LiDar, and radar to gather important environmental/ego-vehicle information. Planning takes care of the decision-making processes about vehicle behaviors and trajectories. Control is the component that actually executes planned behaviors and motions using the vehicle’s actuators. Localization references the vehicle's attempt to gain an understanding of its state real-time, including its speed, inertia, and geographic location through the use of GPS, IMU, and various other sensors.

Another equally important aspect of the vehicle is system management, which is an isolated subsystem that is responsible for overseeing the operation of the vehicle as a whole, maintaining network connections, and keeping logs of vehicle diagnostics and node failures within any internal networks.



Our driver assistance system will accept the input made by the driver using an input device on the steering wheel, which is directly part of vehicle control, instead of the need for planning. This is because direct driver input has higher precedence than algorithmic decision making. Depending on the kind of input the driver makes, the system can be altered ad-hoc. The driver will be able to function the cruise control system easily by being made aware of information given by visual feedback of the current set speed that was adjusted, how to activate/deactivate the system feature, whether it was automatically deactivated and activated again, how to adjust the speed of it, and if more speed or less speed is needed for the cruise control system to activate.

All the visual feedback will be presented in the car’s dashboard. The physical sensors on the front, rear, and sides of the car can provide input data, and after processing, limited versions of this data can be presented to the driver to aid in decision making. This data input may include, but is not limited to, lane detection and warning, hazard detection (potholes, foreign objects), the location and relative speed of other vehicles, or the presence of road signs or indicators. A heads-up windshield display can provide a simplified warning system, to draw the drivers attention to hazards they may otherwise miss.

Many of our safety features not only rely on our sensors, but the cameras as well. The functionality of our safety features is to help ensure the safety of our passengers. The features that are enabled by taking the cameras as inputs would be actively recognizing certain threats that may not be apparent or visible to the driver and taking action upon it, whether it be slowing down or presenting it to the driver so that he or she can make his or her own decision. This would work by having the cameras provide footage that would then have to undergo a software recognizing feature that would be able to differentiate certain threats to our passengers, such as pedestrians, deer, drunk driving, and so on. Depending on whether the driver wants assisted driving or autonomous driving, he would get different outputs. For assisted driving, he would get a visual notification on the dashboard which he can turn on or off in settings. For autonomous driving, the output could take many forms depending on the threat and situation. For example, seeing a deer on the side of the road might simply cause the car to slow down while a deer jumping out would cause the car to stop in the safest manner possible. To ensure the reliability of this feature, we would have to use sensor fusion by using multiple sensors and cameras for the same inputs. This way, even if one of the sensors go awry there would still be the others ones to assist with driving.

The self-driving aspect of this vehicle is very similar to the driver-assistance aspect, but instead extends the data processing and limits the data display. In this way, it takes virtually the same inputs (sensors, cameras, driver preferences, locational data), but instead of informing the driver directly, performs physical actions that are situationally dependent. In the functional architecture diagram, the flow of data from sensor fusion to planning is the path most of the data travels, as the extra level of processing means that decisions regarding vehicle control can be made without driver intervention and input. Present in the steering mechanism, accelerator, and brakes are electronic control devices that can apply or remove pressure bi-directionally depending on the decisions made by the planning system.

Section 3: Requirements

3.1) Functional Requirements

3.1.1) Cruise Control

- 3.1.1.1) Precondition: The car is on and running, with a current speed of 25-95 mph.
- 3.1.1.2) Set current speed and send speed to display
- 3.1.1.3) If car is unable to maintain speed or is interrupted, log the interrupt and notify user on display
- 3.1.1.5) Postcondition: The car maintains the set speed

3.1.2) Automatic Braking

- 3.1.2.1) Precondition: Car is in motion, localization system detects imminent collision on the front of the vehicle
- 3.1.2.2) Localization sensors will have a way to make a priority signal to the vehicle control
- 3.1.2.3) Braking is applied proportionally to imminent objects, slowing vehicles down with a safe margin of error.
- 3.1.2.4) Post condition: IoT HTL vehicle has reached a reduced physical speed that, given the current vehicle surroundings, is in no danger of front-end collision.

3.1.3) Lane Assist

- 3.1.3.1) Pre-condition: Car is on and running; intensity of lane keep assist must be set

- 3.1.3.2) Consistent veering on the drivers part results in an alert within the vehicle to ensure the drivers renewed attention.
- 3.1.3.3) Post-condition: When the vehicle encroaches on the lines of the vehicle's current lane within a distance of 4 inches, the vehicle is automatically steered back towards the center of the lane.

3.1.4) Hazard Detection (other vehicles)

- 3.1.4.1) Pre-condition: The car is driving, driver assist or self driving are activated
- 3.1.4.2) All sensors are working to determine relative position of other vehicles and their speed
- 3.1.4.3) Post-condition: Avoids collisions with other vehicles

3.1.5) Hazard Detection(road hazards/foreign objects)

- 3.1.5.1) Pre-condition: The car is driving, driver assist or self driving is activated, and localization sensors detect a dangerous irregularity in the road
- 3.1.5.2) Warnings are sent to the user display
- 3.1.5.3) Emergency braking is applied. IoT HTL avoids swerving, as that can create further danger
- 3.1.5.4) Post-condition: The vehicle is slowed to the point where the driver can react to the dangerous object

3.1.6) Road Signal Detection

- 3.1.6.1) Pre-condition: Cameras and sensors are working
- 3.1.6.2) Can differentiate between all road signals
- 3.1.6.3) Postcondition: Car will start or stop to move based on road signal

3.1.7) User Interaction and Input

- 3.1.7.1) Pre-condition: car is turned on
- 3.1.7.2) Postcondition: Displays default screen once car is turned on
- 3.1.7.3) Immediately displays any errors or malfunctions with the system or car to the user interface
- 3.1.7.4) Displays anything that the user clicks and wants to show

3.1.8) Real-Time Traffic

- 3.1.8.1) Pre-condition: Car is on and running.
- 3.1.8.2) Shall provide information about traffic conditions to the driver.
- 3.1.8.3) The driver shall receive automatic faster route alternatives, allowing him/her to bypass the traffic jam by taking a different route.
- 3.1.8.4) Post-condition: GPS data shall be collected and the information will be merged to generate real-time traffic information to the driver.

3.1.9) Proximity Access Security

- 3.1.9.1) Pre-condition: You own the car and you are near your car with the key / exit close proximity of the car
- 3.1.9.2) Postcondition: The car will open the door when you try to open it
- 3.1.9.3) The car will lock once you finish driving and leave with your key if you forget your key in the car the car will remind you to take key with you

3.1.10) Self/Assisted Parking

- 3.1.10.1) Pre-condition: Car is on; self-parking button is pressed to activate self-parking; preferred parking space should be selected when prompted; settings for assisted parking should be set in the car’s settings.
- 3.1.10.2) A button is available to engage the self-parking feature

- 3.1.10.3) Post-condition: Vehicle should now be parked, and gear should be shifted to P prior to turning the car off.

3.2) Non-functional Requirements

3.2.1) Performance

- 3.2.1.1) The IoT system will be ready to use 2 seconds after the car engine starts.
- 3.2.1.2) The IoT system will run at any speed that is in an acceptable range that is inputted by the driver.

3.2.2) Reliability

- 3.2.2.1) We aim to aim for a product failure rate to be at or below one in every 5000 operation days.
- 3.2.2.2) We shall offer services to manually inspect the car and make sure that all of its features are working properly.
- 3.2.2.3) features such as our user interface, connections, and lane recognition and correction will all have the same level of reliability as the full IoT HTL system does.
- 3.2.2.4) In the unlikely event that any feature is determined as unreliable, the driver will be notified to let him or her know to get the issue resolved as soon as possible. Issue is logged.

3.2.3) Transparency

- 3.2.3.1) Driver is able to see what kind of data is transmitted to the cloud
- 3.2.3.2) Driver can opt in/out of non-essential data sharing

- 3.2.3.3) Essential data shared includes system failures and pre-accident data, non-essential includes real-time location data. Opting out of data sharing may limit features

3.2.4) Security

- 3.2.4.1) The IoT system will only be able to be accessed or modified by our developer team or authorized technicians.
- 3.2.4.2) The IoT system will also not be able to have any kind of connectivity to any networks such as Wi-Fi, Bluetooth, etc.
- 3.2.4.3) While turned on, the IoT system will be connected to the car's speed and sensors at all times.
- 3.2.4.4) In the case where the engine shuts down, the cruise control power source will be transferred to an alternator.
- 3.2.4.5) While in operation, the car only may accept input from the local network.
    - 3.2.4.6) There cannot be any means of accessing or modifying the vehicles behavior from any external source, as security risks are too high.
- 3.2.4.7) While the driver is in and operating the vehicle, the network is virtually isolated, and only fetches data concerning traffic or GPS from the internet and satellites respectively.

3.3) User Interface

- 3.3.1) The user will have access to a steering wheel, accelerator pedal, decelerator pedal, and a means of changing the car's mode between drive, park, etc.
- 3.3.2) The steering column will have the headlight control, turn signal control, as well as cruise control, driver assist, self-driving control, and self-parking.
- 3.3.3) On the dashboard, the driver will be able to read a variety of gauges, including but not limited to speed, acceleration, fuel/power levels, and any warnings regarding immediate hazards in the road
- 3.3.3) On the infotainment system, there will be a touchscreen interface for the car that can display mobile connectivity/GPS system or show more advanced data regarding the car. The user has the ability to dive deeper into vehicle customization and settings as well.

3.4) Hardware/OS

- 3.4.1) Our vehicle will operate under the RT Linux operating system to perform the tasks from the embedded ECU's (automatic braking, airbag system, lane correct, etc.) that may have strict deadlines
- 3.4.2) A separate GPOS (General Purpose Operating System) will be used for the infotainment system.

3.5.1) Network Connectivity

- 3.5.1.1) In order to ensure that the car is driving at its optimal level of safety and functionality, we need to make sure that the system management which oversees the operation of the whole car has no problems.

- 3.5.1.2) In order to prevent any catastrophe that comes with connection failure, we need to ensure that all of the parts of the car that needs to connect to one another does not have that problem most of the time and in the worst case that it does, it would notify the driver as soon as possible through the user interface that something is not working as it should and what it could mean for the passengers in the car.

3.5.2) Physical Connections

- 3.5.2.1) While bluetooth connectivity will be offered, physical connections can be used by drivers/passengers to connect to the infotainment system for features that make use of a smartphone/mobile device if desired.
- 3.5.2.2) Devices can also be charged through physical connections.
- 3.5.2.3)

3.6) Logging

3.6.1) Logging will fall into two categories: actions and warnings.

3.6.2) Actions will represent data recorded every time that the system provided significant input on any vehicle control system.

- This data will be composed of data including time, speed, what type of action was applied, and the intensity/relative direction of the application.
- To prevent the overlogging of minor changes to the vehicles control, testing of logging sensitivity can be done to adjust the levels of adjustment that the system considers to be worthy of logging.
- The threshold should remain relatively low, to make certain that no critical data is missed.

3.6.3) Warning logs consist of hazard detection, dangerous driving situations, dangerous road conditions, and driver error.

- Data to be logged with each warning is the time, relative speed, warning type, and defining characteristics of the situational awareness system at the time of the warning.
- Hazard detection and dangerous road conditions, encompass irregular and/or dangerous items that may pose a risk to the driver, so that in the event of any damage to the vehicle, diagnostic data can aid in showing the cause of the damage.
- Dangerous driving situations are those that involve other drivers specifically. Situations that constitute this type of warning are tailgating, sudden and intense braking, swerving, or other forms of imminent collision.
- These dangerous situations will be logged to provide a record for diagnostics in the event of damage to the vehicle.

<u>Section 4: Requirement Modeling</u>

4.1: Use Case Scenarios

- 4.1.1) Technician access of vehicle
  - Technician accesses IoT HTL computer through a port in car, requires car’s Owner to give permission by giving key
  - Technician must enter a master password (Technician unique, identifies technician and prevents unauthorized access)
    - Incorrect password, reprompt
    - Multiple incorrect password attempts initiates lockout and requires re-authentication from owner
  - Technician may download diagnostic report or download past usage logs
  - If diagnostic report unavailable, Technician may run a diagnostic report to download updated data
    - If unable to fetch data that is known to exit, log error for diagnostics
- 4.1.2) Driver activating the IoT HTL system (in motion)
  - The driver must press the on/off button that is used to engage/disengage the IoT HTL system.
  - The vehicle must be driving at a minimum of 25 mph.
  - The IoT HTL system will give a visual response whenever it is ready to activate.
  - The driver must press the “+” button to increment the speed by 1 or the “-” button to decrement the speed by 1.

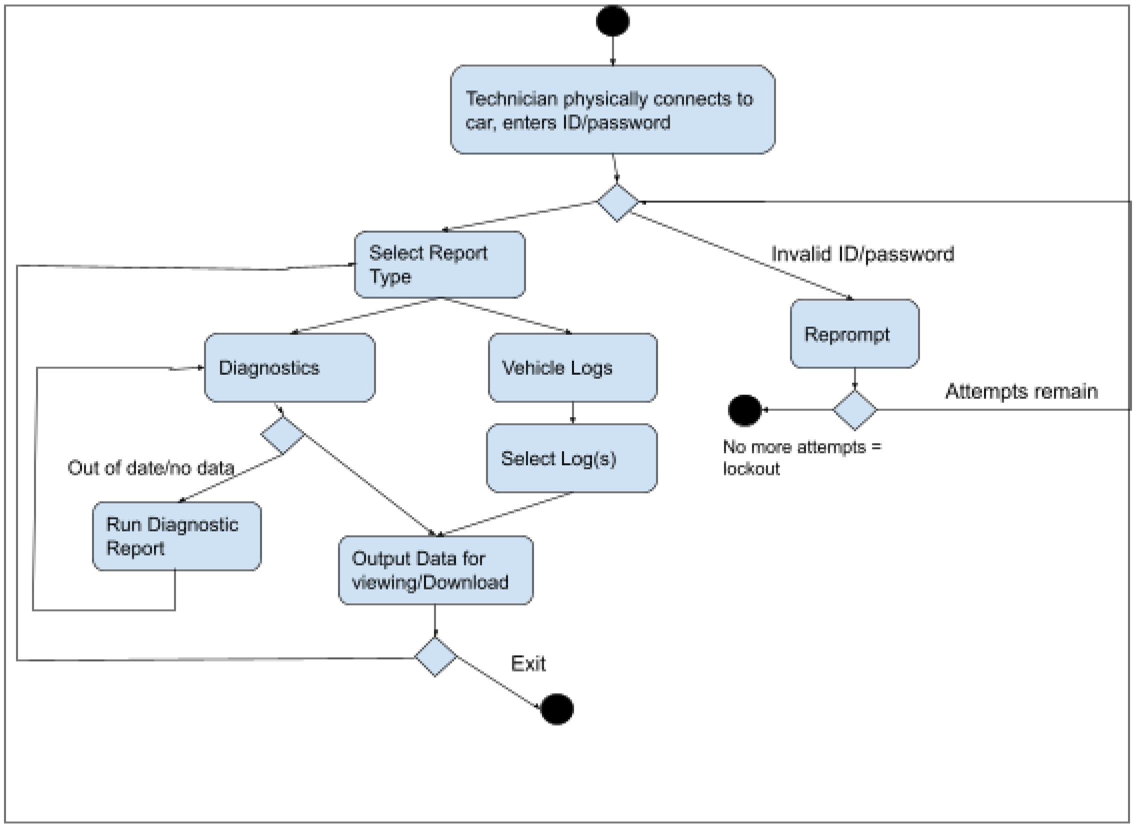
- The IoT system will give a visual representation of the new current speed that has been set.
- The IoT system will request information from Perception and Localization sensors
- Planning will make decisions with values
- The IoT System will request to match the desired speed from the vehicle control system.
- The Vehicle Control System will correspondingly modify the speed.
- The speed of the vehicle is constantly being passed on to the IoT system.

- 4.1.3) Driver initiates self-parking
  - The self-parking button is pressed
  - Vehicle must be stationary
  - Must have available parking spots open to execute action
  - Parking spot should be selected when prompted
    - If selected spot is too small/narrow, an error message will be displayed
  - Car will then park itself into the desired spot
  - If at any point the driver wishes to disengage the action, self-parking button should be pressed again
  - Once action is completed, a message will be displayed to alert the driver
- 4.1.4) Driver using lane assist
  - Minimum speed for when lane assist is going to activate is set by user (default is set to 25 mph)

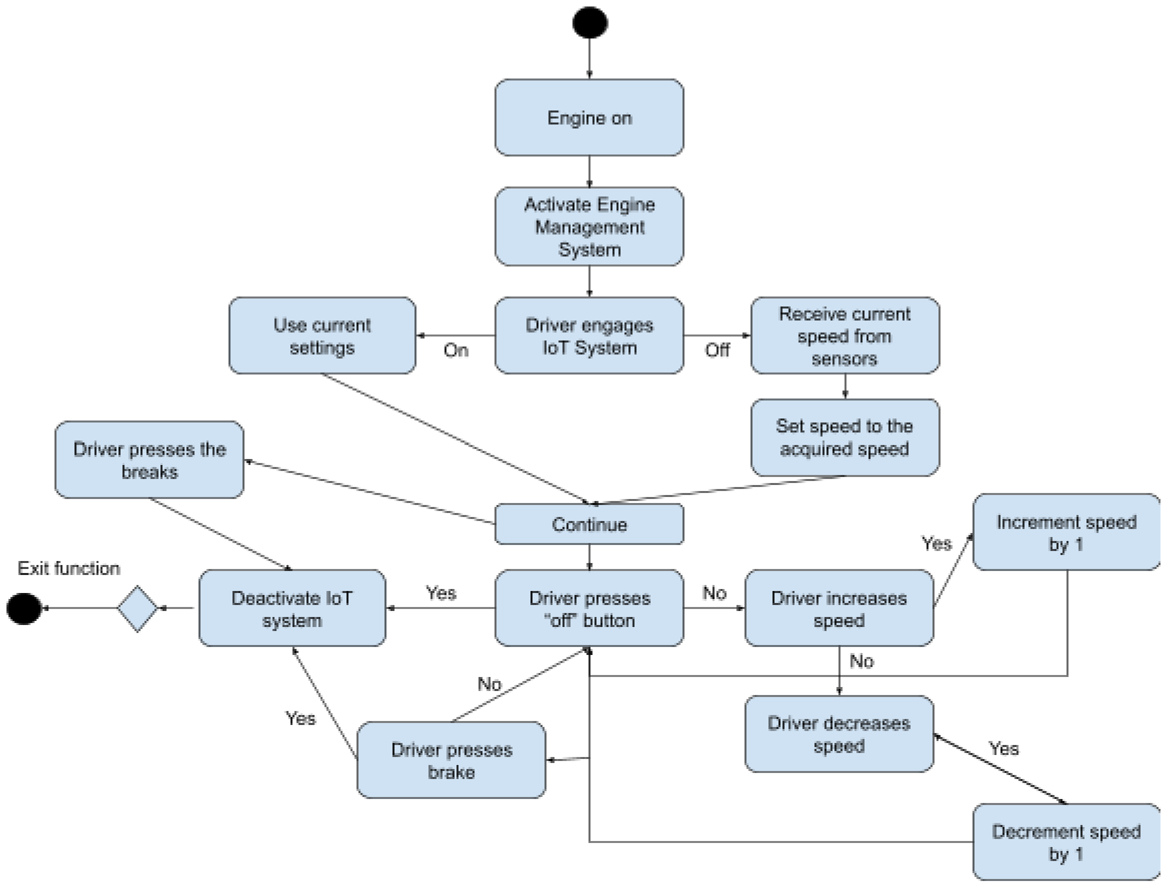
  - Must be driving in a lane for lane assist to activate
  - The user interface will display that the lane assist is on when it is currently running
    - when the driver gets close to crossing the line on his lane, a warning message will be displayed to the user interface
    - When the driver crosses the line, the car will automatically shift the vehicle back into its lane
  - If car signal is turned on, then the driver will be able to override the lane assist when the driver wants to change lanes or make turns
- 4.1.5) Hazard detection activation
  - Driver makes a decision that results in unsafe conditions, or external variables create a situation that is likely to result in an accident
  - IoT HTL receives information from Perception system sensors, such as short range sensors and long-range lasers
    - If no incoming objects at dangerous relative speed, continue course
  - If an object deemed dangerous (relative speed to current vehicle and getting closer), then send data to planning with high priority
  - Planning determines whether steering, braking, or accelerating are the best course of action
  - Planning decision passed onto vehicle control, which initiates an action
    - Sensor fusion continues to pass data from sensors to planning and adjusts in real time.

4.2: Activity Diagrams

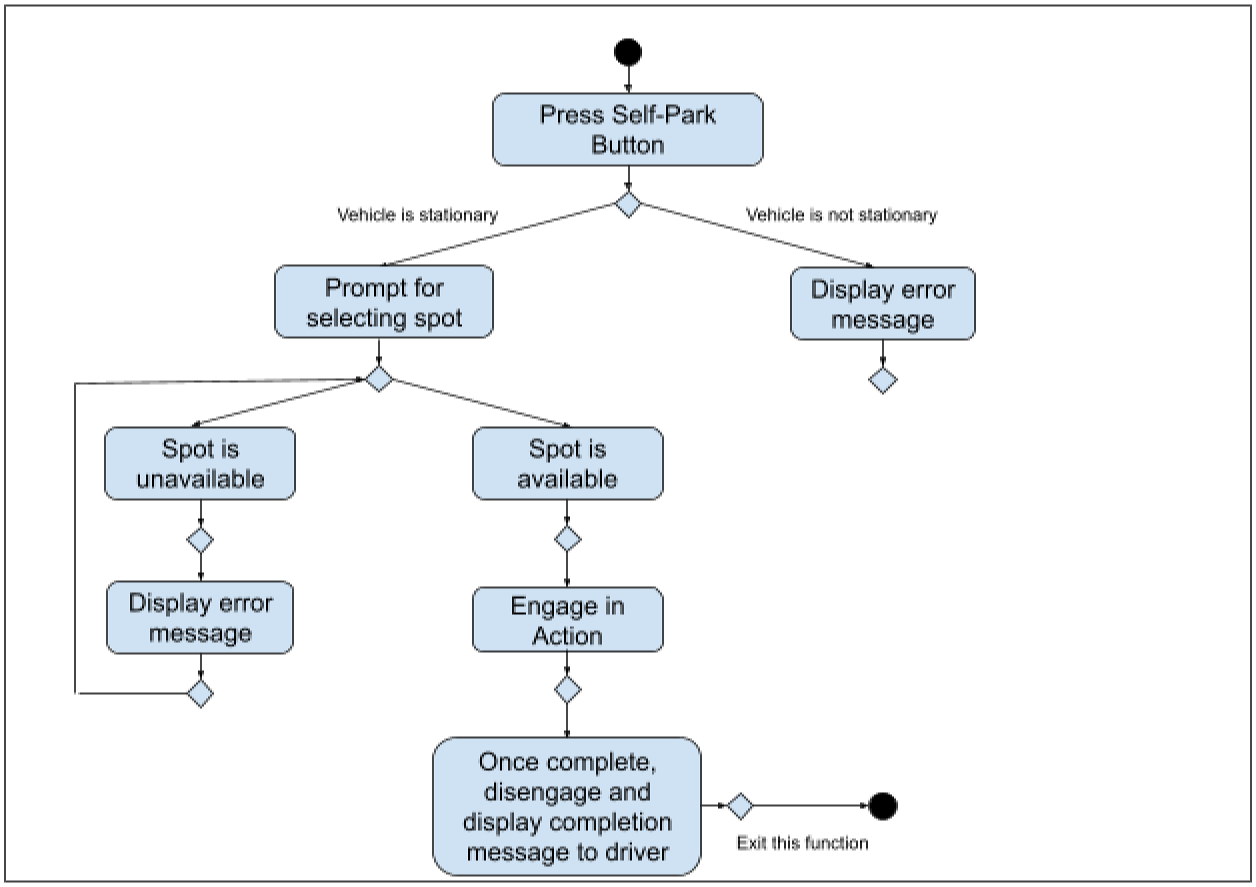
- 4.2.1) Technician access of vehicle



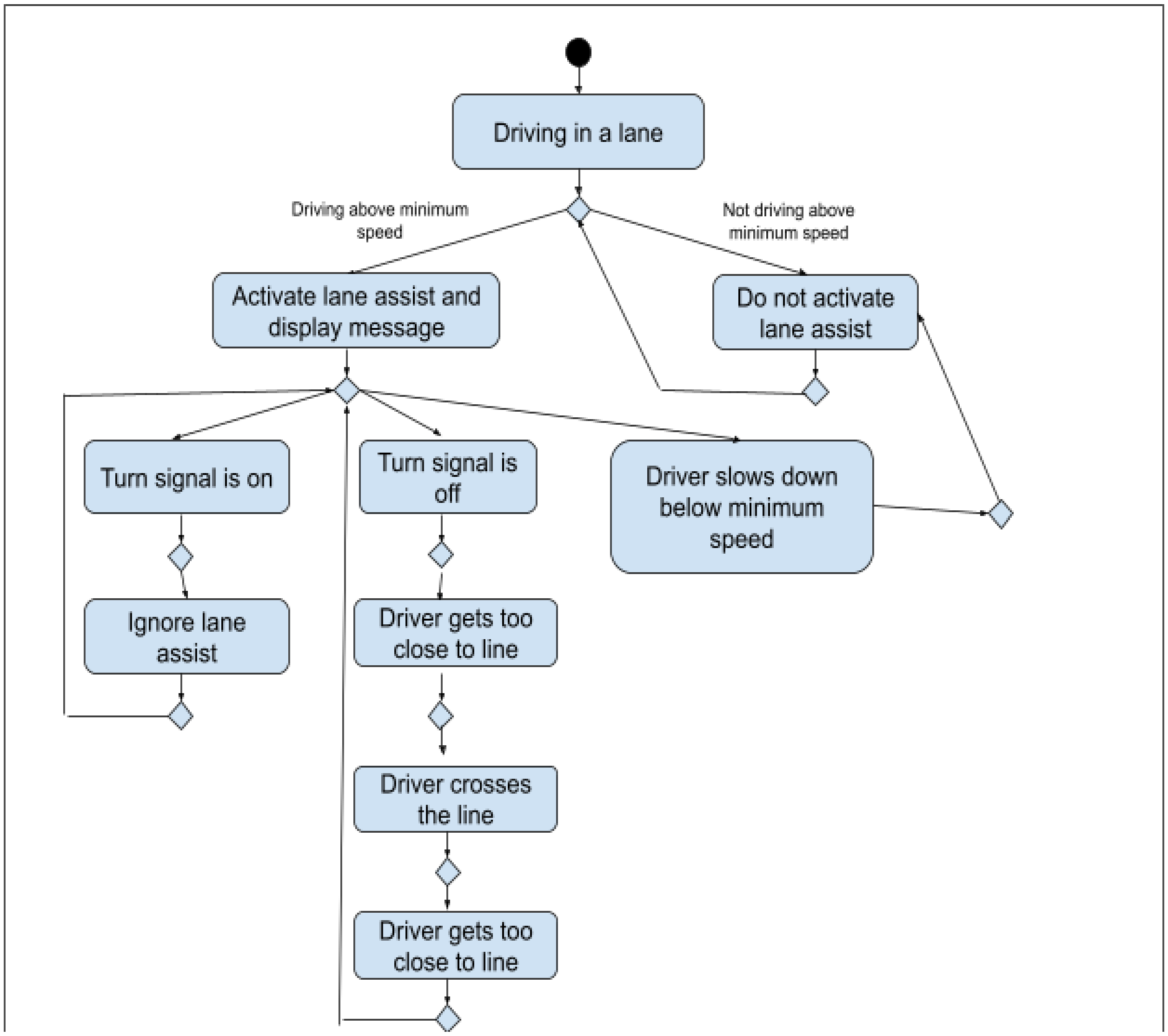
- 4.2.2) Driver activating the IoT system



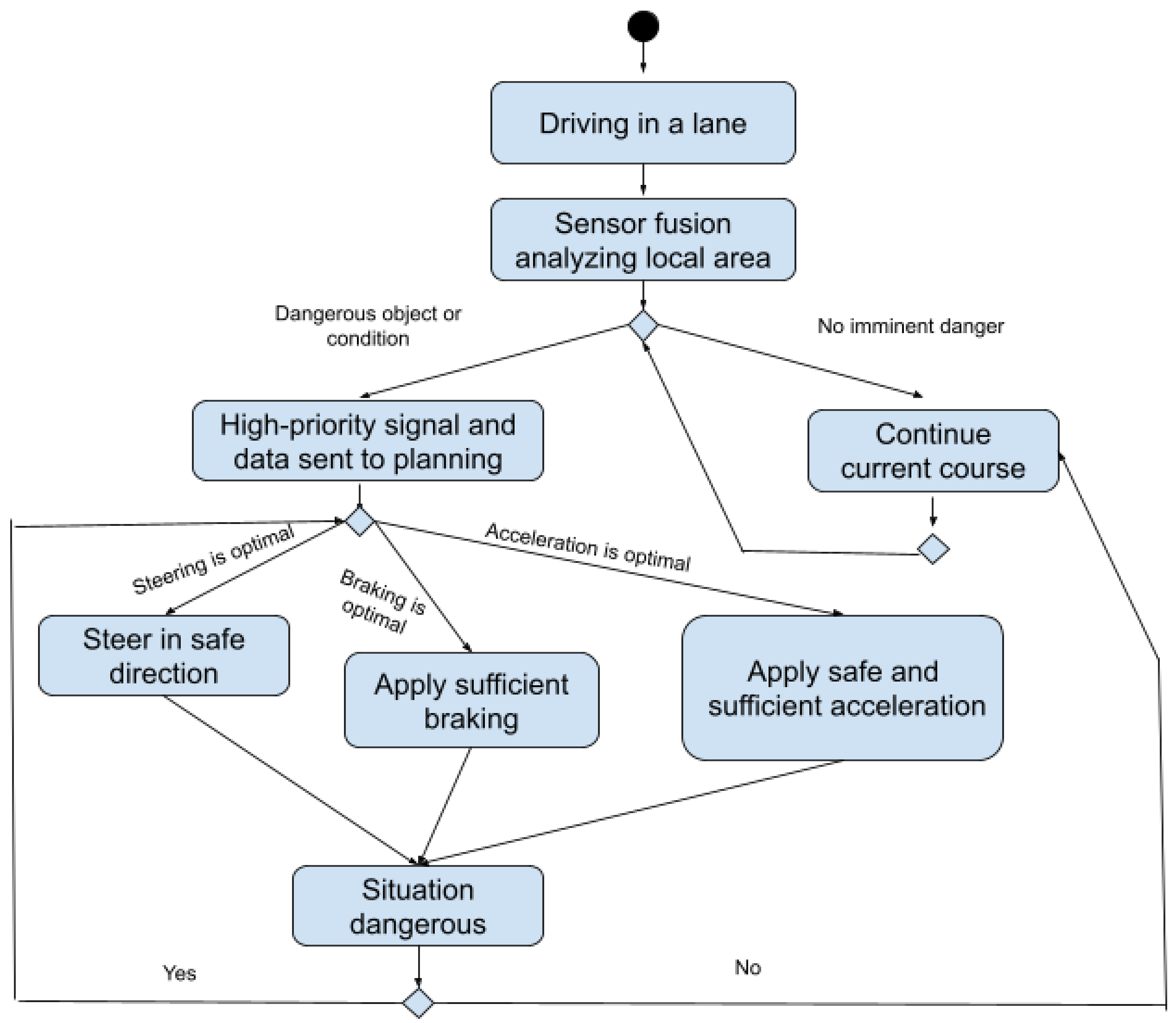
- 4.2.3) Driver initiates self-parking



- 4.2.4) Driver using lane assist

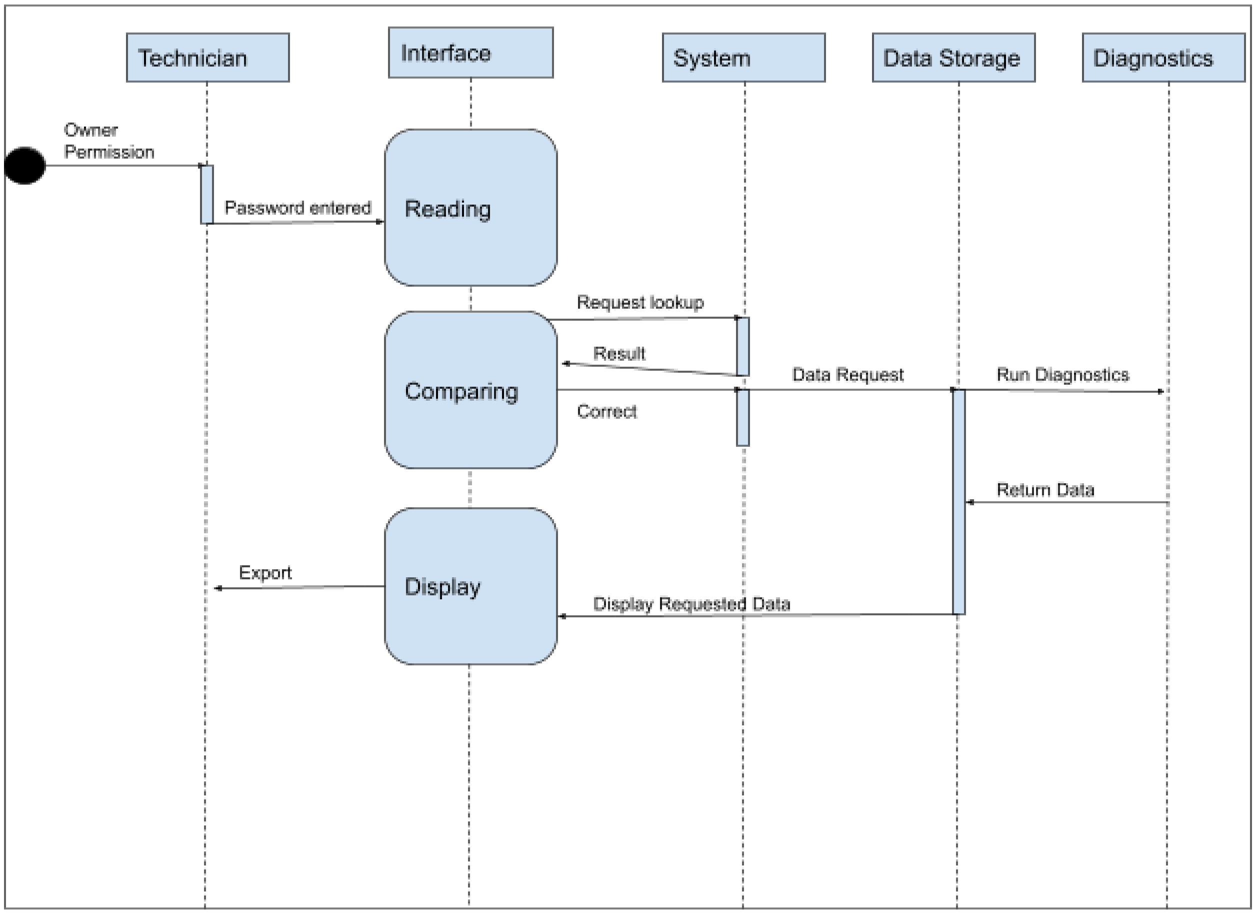

Driving in a lane
Driving above minimum speed
Not driving above minimum speed
Activate lane assist and display message
Do not activate lane assist
Turn signal is on
Turn signal is off
Driver slows down below minimum speed
Ignore lane assist
Driver gets too close to line
Driver crosses the line
Driver gets too close to line

- 4.2.5) Hazard Detection Activation

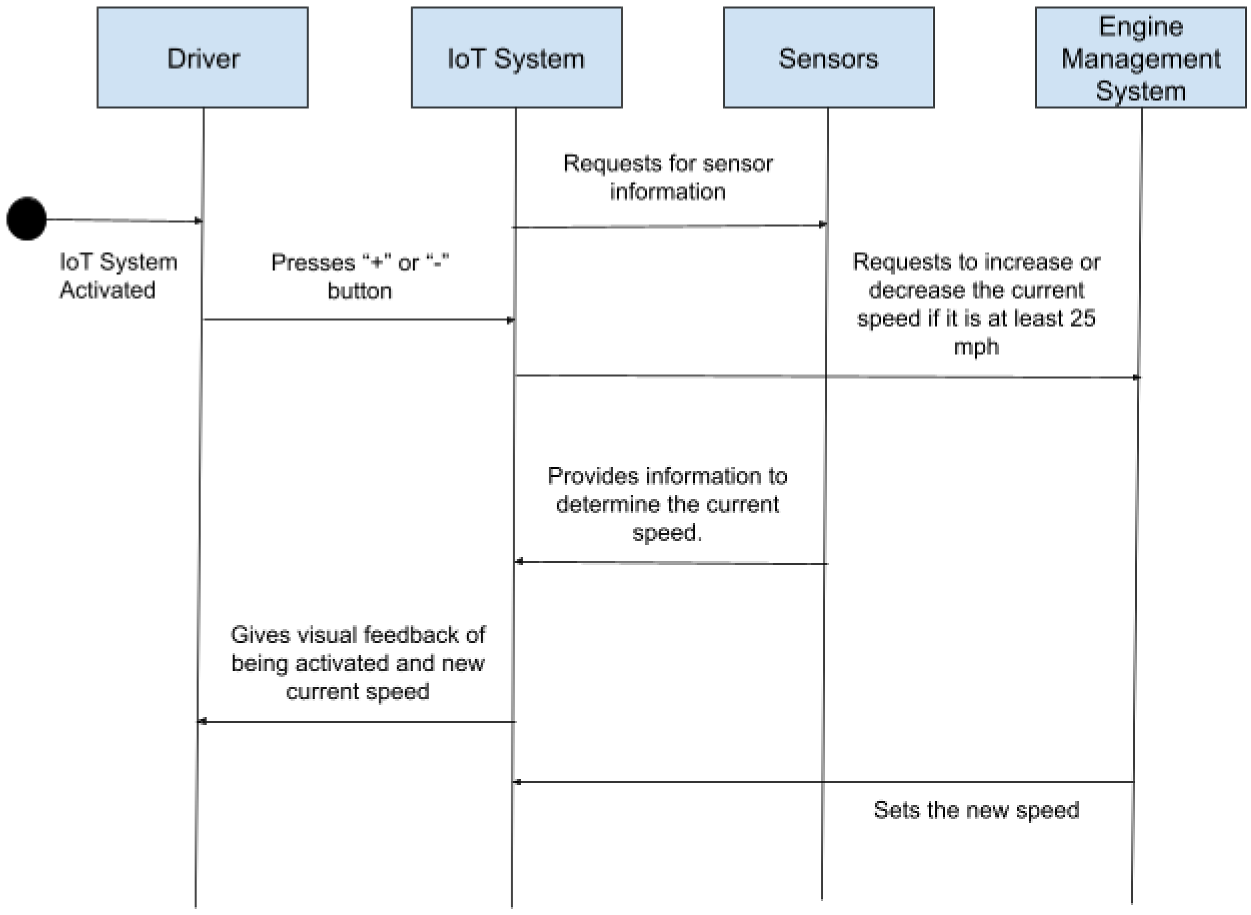


4.3: Sequence Diagrams

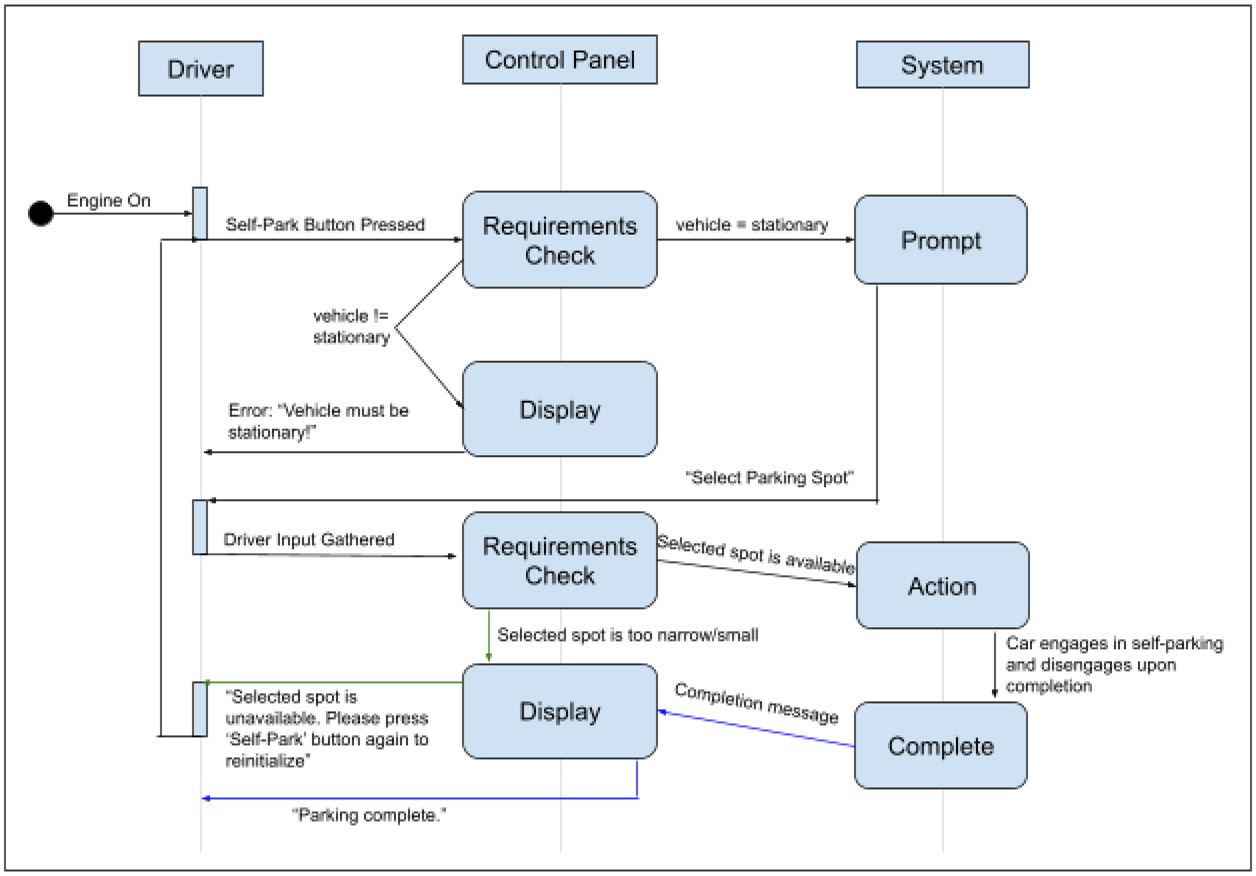
- 4.3.1 Technician Access of Vehicle



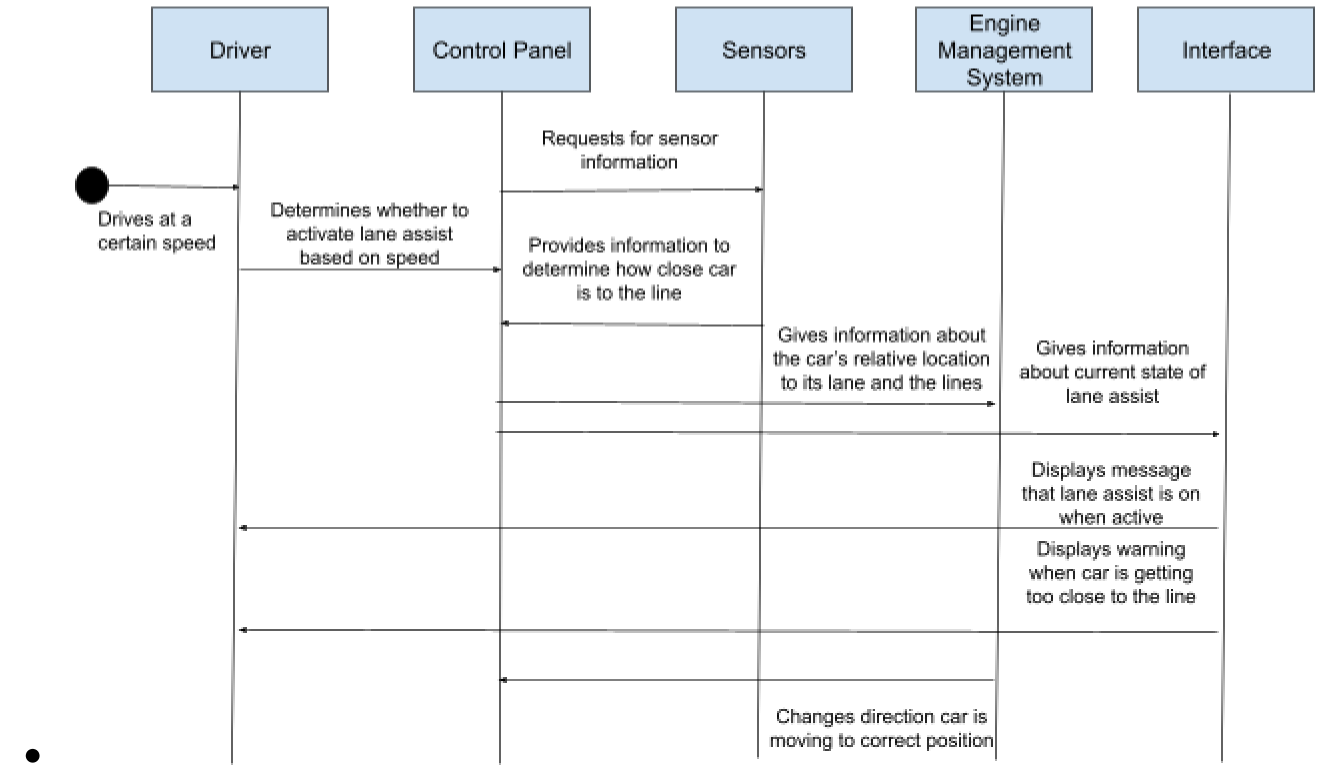
- 4.3.2) Driver activating the IoT system


Driver
IoT System
Sensors
Engine Management System
Requests for sensor information
IoT System Activated
Presses "+" or "-" button
Requests to increase or decrease the current speed if it is at least 25 mph
Provides information to determine the current speed.
Gives visual feedback of being activated and new current speed
Sets the new speed

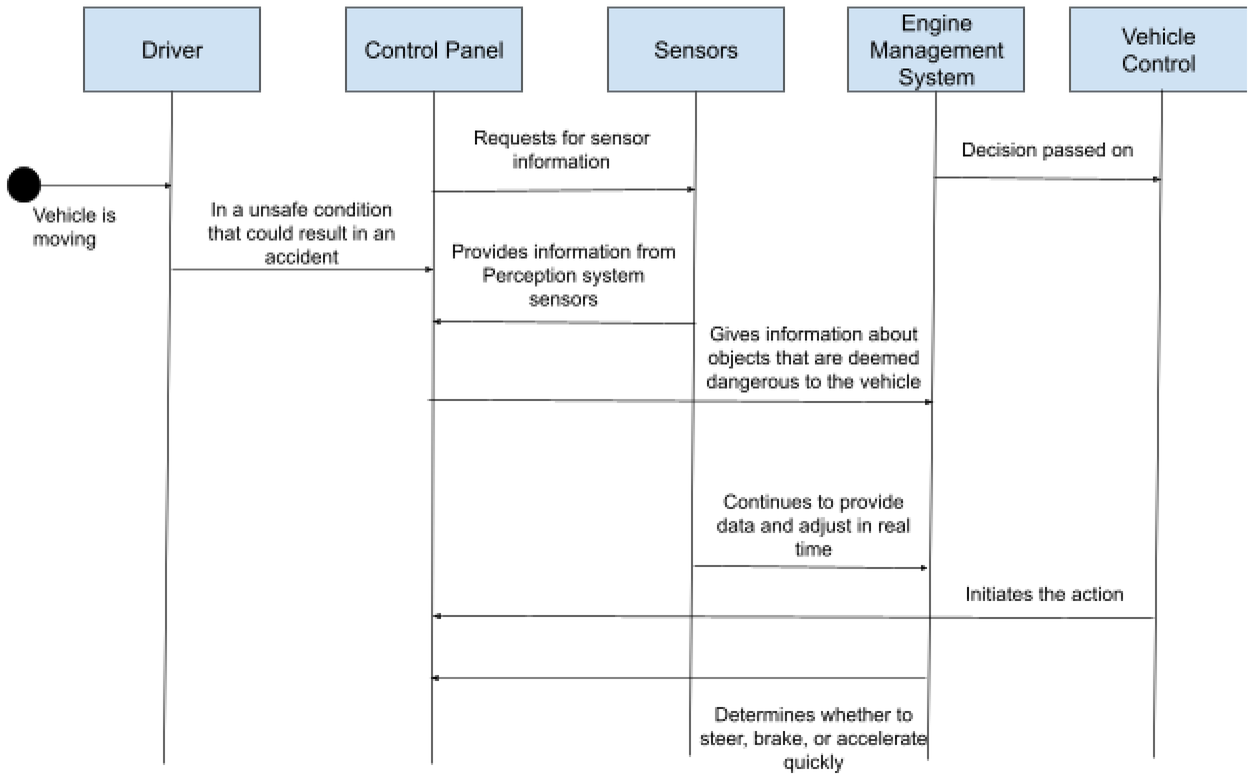
- 4.3.3 Driver initiates self-parking



- 4.3.4 Driver using lane assist


Driver
Control Panel
Sensors
Engine Management System
Interface
Requests for sensor information
Drives at a certain speed
Determines whether to activate lane assist based on speed
Provides information to determine how close car is to the line
Gives information about the car's relative location to its lane and the lines
Gives information about current state of lane assist
Displays message that lane assist is on when active
Displays warning when car is getting too close to the line
Changes direction car is moving to correct position

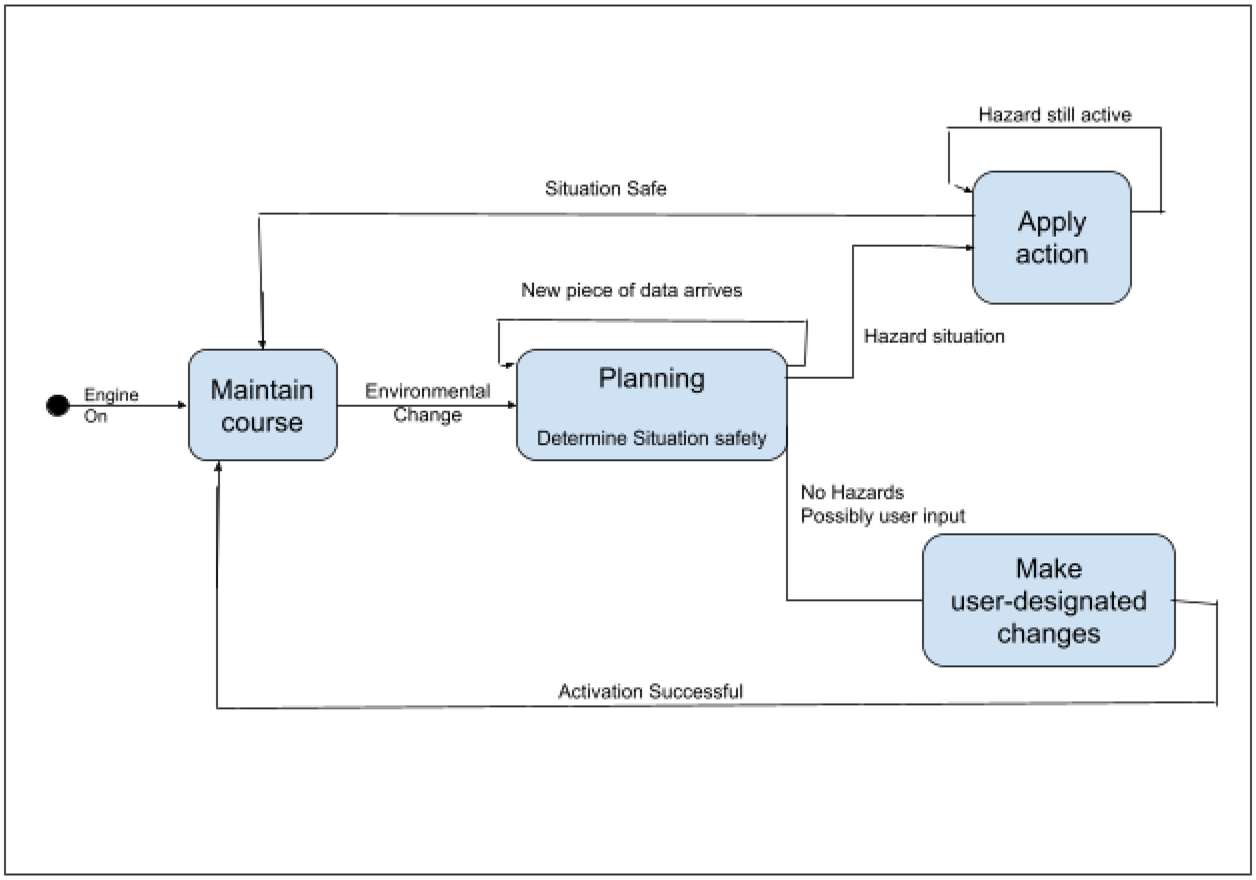
- 4.3.5 Hazard Detection Activation


Driver
Control Panel
Sensors
Engine Management System
Vehicle Control
Requests for sensor information
Decision passed on
Vehicle is moving
In a unsafe condition that could result in an accident
Provides information from Perception system sensors
Gives information about objects that are deemed dangerous to the vehicle
Continues to provide data and adjust in real time
Initiates the action
Determines whether to steer, brake, or accelerate quickly

4.4: Classes

- 4.4.1) Technician access
    - Technician
        - ID
        - Password
    - Display
        - Speed
        - Gas tank
        - Location data
    - Driver
        - Name
        - ID number
        - Age
        - Height
        - Weight
    - Sensor
        - Sensor type
        - data

4.5: State Diagrams



<u>Section 5: Design</u>

5.1: Architecture Choice

- 5.1.1) Data Centered
    - This approach would allow the IoT HTL system to act in a way that is strongly associated with IoT concepts. That is, individual embedded devices that interact with each other and a network independently, relying on retrieval of information from a centralized source.
    - Pros
        - Provides transparency among nodes and allows easy data sharing.
        - Allows for efficient parallelism.
    - Cons
        - There may be a high processing overhead, as nodes must take care of finding and processing the data they need among all other data.
        - High reliance on centralized unit may be prone to total system failure (low resistance to failure)
- 5.1.2) Data Flow
    - This approach heavily supports logical analysis, and would prevent the IoT HTL system from running into issues in logic or issues with how data is passed between systems.
    - Pros

        - Data filtering ensures that each subsystem only receives data or decisions that are relevant (a form of pre-processing). This may minimize subsystem processing overhead.
        - Designers have the power to implement strict and provable logic that minimizes unexpected output
    - Cons
        - High design overhead. Designers must ensure that the data flow has the ability to adapt to situations, and they may have to heavily premeditate most or all situations to make sure they are accounted for.
        - Flow of data may be rigid and unadaptable. Unforeseen situations may call for certain data to be available that is not part of the designed flow, possibly causing failures.
- 5.1.3) Call Return
    - This approach is often used to create a program that is easy to scale and modify
    - Pros
        - Program can be easily modified, allowing for frequent changes
        - Utilizes sub-styles or subprograms, allowing for versatility
    - Cons
        - Changes in internal data structures may lead to ripples in other modules. This may serve as a disadvantage for our purposes, since

we are likely to have many different subprograms and data structures

- 5.1.4) Object Oriented
    - This approach is the latest subtype of the call and return architecture. It is based on the division of responsibilities for a system into individual self-sufficient objects. This can allow the IoT HTL system to be divided into different classes and objects.
    - Pros
        - Allows for parallel development, which can be efficient in a team environment. Each member can work independently if they wish.
        - Object representations of internal subsystems can offer a “snapshot” of current system state, without the inflexibility of a finite state machine
        - Mutable object data can represent partial state changes for each individual system.
    - Cons
        - Classes (objects) must be heavily predefined for this to be effective. The system may suffer in terms of flexibility and scalability
        - Questions of scope and access must be considered.
            - Encapsulation of data within objects may slow down data access, sharing, or communication.
- 5.1.5) Layered

    - This approach is a system that uses a stack of layers, where each layer is only dependent on the layer below it. May not be as applicable for our IoT HTL system, as having layers may cause issues accessing parts of the system.
    - Pros
        - Efficient for incremental development
        - Layers can be replaced as long as the interface of the layer isn't modified
    - Cons
        - It is difficult to achieve clean separation between layers, as some systems are difficult to structure into layers
        - Many layers of processing may lead to performance degradation
- 5.1.6) Model View Controller
    - The MVC architecture would allow the IoT HTL system to utilize three main logical components: model, view, and controller. Each component will be able to manage the development aspect of the system.
    - Pros
        - It provides a faster development process, as one programmer can work on the view component while the other could work on another component
        - Multiple views can be provided
        - MVC works well with with asynchronous technique, which helps build quicker web applications

        - Debugging will be easier with multiple levels properly written in the application
    - Cons
        - The MVC architecture could make it hard to keep up with update requests if there are frequent changes ongoing
        - It can be complex to develop applications with this architecture
        - Must have strict rules on methods
- 5.1.7) Finite State Machine
    - The simplification of the functionality of the
    - Pros
        - FSM machines are flexible, as there are many ways to implement them. This makes them quick to develop and execute FSMs.
        - It is easy to determine the reachability of a state when the state is expressed abstractly. It is established whether or not you can obtain a state from another condition and what is required to do so.
    - Cons
        - Cannot be used for all types of domains
        - Can be too predictable
        - Large FSMs with many states and transitions can be difficult to manage and maintain
        - Non-adaptable to the possibility of unforeseen situations

Our choice for architecture is Object Oriented. A large consideration for this architecture choice is the level of abstraction that is achievable. There is a high level of independence that each system can operate with, allowing for the removal of any dependence that causes unnecessary busy waiting for subsystems. We can also abstract large portions of the system as a single object, and define methods and attributes within the objects to define the behavior of that particular portion.

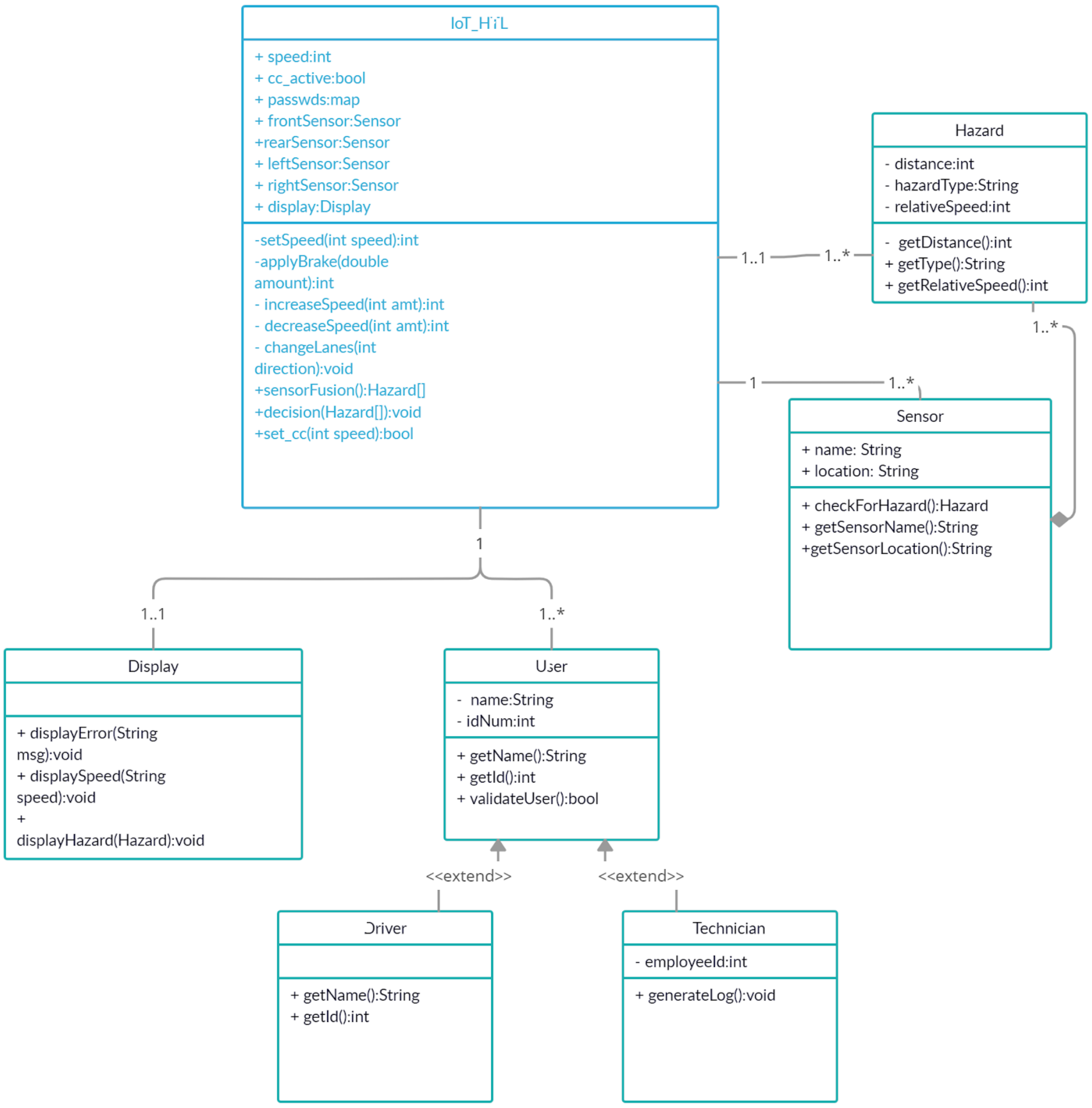
With self-driving and vehicle assist systems, response time is paramount. Object-oriented design represents the perfect tradeoff between development and system abstraction with the time-based reliability that is required of our system. Instantiated objects are self-contained, but outside signals from other objects have extreme priority. Therefore, the objects representing sensors have the ability to exert influence on vehicle systems with the expectation of no delays.

5.2: Interface Design

- 5.2.1) Driver Interface
    - The driver interface takes a more indirect role in data creation and manipulation. The Driver may directly manipulate data related to their user profile or preferences. That is, their name, profile settings (identifying user data and IoT HTL preference settings).
        - User input for this data may be input from the vehicles internal display (GUI)
        - User will input all else through the IoT HTL system, which is a hub for decision making
        - The user’s actions will alter the state of the vehicle relative to hazards, which may change the vehicle's decision making.
- 5.2.2) Technician Interface

- Direct access to the data within IoT HTL through elevated permissions.
- No need to access Driver's GUI, the methods used by the technician require special permission and can run diagnostics using these technician-specific methods

5.3: Component-level Design



Section 6: Code

Driver.java

```java
class Driver extends User {
    private int idNum;

  public Driver(String name, int idNum) {
    super(name);
    this.idNum = idNum;
  }

  public int getIdNum() {
    return idNum;
  }

}
```

Hazard.java

```java


class Hazard {

  private int distance;
  private String hazardType;
  private int relativeSpeed;
  private int side;

  public Hazard(int dist, String type, int rs, int loc) {
    this.distance = dist;
    this.hazardType = type;
    this.relativeSpeed = rs;
    this.side = loc;
  }
  public int getDistance() {
    return distance;
  }

  public String getType() {
    return hazardType;
  }

  public int getRelativeSpeed() {
    return relativeSpeed;
  }

  public int getSide() {
    return side;
  }

  public void printHazard(){
    System.out.println("Type: "
                     + getType()
                     + "\tDistance: "
                     + getDistance()
                     + "\tRelative Speed: "
                     + getRelativeSpeed());


  }



}
```

IoT_HTL.java

```
import java.util.HashMap;
```

```
class IoT_HTL {

  static final int FRONT = 0;
  static final int REAR = 1;
  static final int LEFT = 2;
  static final int RIGHT = 3;

  static final int NORMAL = 0;
  static final int ERROR = 1;

  Hazard [] detectedHazards;

  int speed;
  boolean cc_active;

  Sensor frontSensor;
  Sensor rearSensor;
  Sensor leftSensor;
  Sensor rightSensor;
  HashMap<Integer, Driver> userMap = new HashMap<Integer, Driver>();
  HashMap<Integer, Technician> technicianMap = new HashMap<Integer,
Technician>();

  /*
   * Default constructor
   */
  public IoT_HTL() {
    cc_active = false;

    frontSensor  = new Sensor(FRONT);
    rearSensor   = new Sensor(REAR);
    leftSensor   = new Sensor(LEFT);
    rightSensor  = new Sensor(RIGHT);

    detectedHazards = new Hazard[4];
  }
  User verifyUser(int pin) {
      User temp = userMap.get(pin);
      if (temp != null) {
          return temp;
      }
      return technicianMap.get(pin);

  }

  /*
   * Sets the speed of the vehicle. May be outside of the range of cruise
control,
   * As this method may be used in emergency situations.
   */
  int setSpeed(int speed) {
    this.speed = speed;
    return this.speed;
  }

  /*
   * Increases the CC speed.
   * Returns the old speed value.
   */
  int increaseSpeed(int amt) {
    int temp = this.speed;
    this.speed += amt;
    return temp;
  }

  /*
   * Decreases the CC speed.
   * Returns the old speed value.
   */
  int decreaseSpeed(int amt) {
    int temp = this.speed;
    this.speed -= amt;
    return temp;
  }

  /*
   * Sets the cruise control to a speed. Activates cruise control if not
already
   * Active.
   * Returns true if successful, false otherwise.
   *
  */
  boolean set_cc(int speed) {

    if (this.speed < 25) {
      System.out.println("Unable to set cruise control. Invalid current
speed\n");
      return false;
    }
    if (speed < 25 || speed > 95) {
      System.out.println("cc must be between 25 and 95 mph");
      return false;
    }
    return false;
  }


  int changeLanes(int direction) {
    return direction;
  }
  // Bring sensor data together into usable format.
  void sensorFusion() {
    detectedHazards[FRONT] = frontSensor.detectedHazard;
    detectedHazards[REAR] = rearSensor.detectedHazard;
    detectedHazards[LEFT] = leftSensor.detectedHazard;
    detectedHazards[RIGHT] = rightSensor.detectedHazard;
  }

  //Print some current state info about IoT_HTL
  void printIotState() {
    System.out.println("Speed: " + speed + " CC: " + cc_active);
  }

  /*
        Standardized data printout.
  */
  void displayNormal(String decision, Hazard [] currentHazards) {
    System.out.println("****************************************************");
    System.out.print("Front:\t\t");
    currentHazards[FRONT].printHazard();
    System.out.print("Rear:\t\t");
    currentHazards[REAR].printHazard();
    System.out.print("Left:\t\t");
    currentHazards[LEFT].printHazard();
    System.out.print("Right:\t\t");
    currentHazards[RIGHT].printHazard();
    System.out.println(decision);
    System.out.print("New ");
    printIotState();
    System.out.println("***************************************************");
  }

  /*
    Check for issues that require lane corrects, or for hazards on the side.
  */
  void laneCheck(StringBuilder string) {
      if (detectedHazards[LEFT].getRelativeSpeed() !=
-(detectedHazards[RIGHT].getRelativeSpeed())) {
          if (detectedHazards[LEFT].getRelativeSpeed() < 0) {
              string.append("WARNING: " + detectedHazards[LEFT].getType() + "
approaching from left.\n");
          }
          if (detectedHazards[RIGHT].getRelativeSpeed() < 0) {
            string.append("WARNING: " + detectedHazards[RIGHT].getType() + "
approaching from right.\n");
        }
      }
      else
      {
          if (detectedHazards[LEFT].getRelativeSpeed() > 0) {
              string.append("Lane Correction Left\n");
          }
          else if (detectedHazards[RIGHT].getRelativeSpeed() > 0) {
              string.append("Lane Correction Right\n");
          }
      }
  }


  /*
   * This method is where driving decisions are made.
   * Takes into account hazard data from sensors
   */
  void decision(Hazard [] hazards) {
    String frontHazardType = hazards[FRONT].getType();
    String rearHazardType = hazards[REAR].getType();
    String leftHazardType = hazards[LEFT].getType();
    String rightHazardType = hazards[RIGHT].getType();

    int frontHazardRS = hazards[FRONT].getRelativeSpeed();
    int rearHazardRS = hazards[REAR].getRelativeSpeed();
    int rightHazardRS = hazards[RIGHT].getRelativeSpeed();
    int leftHazardRS = hazards[LEFT].getRelativeSpeed();

    int frontHazardDistance = hazards[FRONT].getDistance();
    int rearHazardDistance = hazards[REAR].getDistance();
    int leftHazardDistance = hazards[LEFT].getDistance();
    int rightHazardDistance = hazards[RIGHT].getDistance();
    StringBuilder string = new StringBuilder(256);
    laneCheck(string);


    if (frontHazardType.equals("null") || frontHazardRS >= 0) {
      printIotState();
      string.append("Maintain course.");
      displayNormal(string.toString(), hazards);
      return;    //This will exit the method, avoiding any double decisions.
    }

    //Vehicle case
    if (frontHazardType.equals("vehicle")) {
      if (frontHazardDistance >= 400) {
        if (leftHazardType.equals("lane")) {
          printIotState();
          string.append("Vehicle detected: Change lanes left, maintain
speed.");
          displayNormal(string.toString(), hazards);
          return;
        }
        else if (rightHazardType.equals("lane")) {
          printIotState();
          string.append("Vehicle detected: Change lanes right, maintain
speed.");
          displayNormal(string.toString(), hazards);
          return;
        }
        else {
          printIotState();
          decreaseSpeed(frontHazardRS/3);
          cc_active = false;
          string.append("Applying brakes: Consider further braking, or lane
change if becomes possible.");
          displayNormal(string.toString(), hazards);
          return;
        }
      }
      else {
        //if the distance of the front vehicle is close (< 400), apply brakes
        if (leftHazardType.equals("lane")) {
          //apply brakes while merging left
          printIotState();
          decreaseSpeed(speed/4);
          cc_active = false;
          string.append("Close vehicle detected: Applying light brakes, change
lanes left.");
          displayNormal(string.toString(), hazards);
          return;
        }
        else if (rightHazardType.equals("lane")) {
          //apply brakes while merging right
          printIotState();
          decreaseSpeed(speed/4);
          string.append("Close vehicle detected: Applying light brakes, change
lanes right.");
          displayNormal(string.toString(), hazards);
          return;
        }
        else {
          //apply heavy brakes if driver cannot merge
          printIotState();
          decreaseSpeed(speed/3);
          cc_active = false;
          string.append("WARNING: CRASH DANGER\nApplying heavy brakes, change
lanes if becomes available.");
          displayNormal(string.toString(), hazards);
          return;
        }
      }

    }
    //Object case
    else if(frontHazardType.equals("object")) {

      // object is suddenly too close, we do not attempt to swerve (not enough
time)
      // In this case, heavily brake, disengage cc, warn user.
      if (frontHazardDistance < 200) {
        printIotState();
        decreaseSpeed(speed/3);
        cc_active = false;
        string.append("WARNING: CRASH DANGER\nApplying heavy brakes, change
lanes if becomes available.");
        displayNormal(string.toString(), hazards);
        return;
      }

      //If driver cannot merge to the right lane, merge to the left lane
      if (leftHazardType.equals("lane")) {
        printIotState();
        string.append("Object detected: Changed lanes left, maintain speed");
        displayNormal(string.toString(), hazards);
        return;
      }
      //If driver cannot merge to the left lane, merge to the right lane
      else if (rightHazardType.equals("lane")) {
        printIotState();
        string.append("Object detected: Changed lanes right, maintain speed");
        displayNormal(string.toString(), hazards);
        return;
      }

      //If the driver cannot merge lanes, use heavy brakes. I made a mistake,
we should use speed lol. Thats my bad. If we use this in some cases, it will be
negative and actually INCREASE speed. There are some cases we should use rs
speed tho but im going to go through and check them all. ohh true
      else {
        printIotState();
        decreaseSpeed(speed/3);
        cc_active = false;
        string.append("WARNING: CRASH DANGER\nApplying heavy brakes, change
lanes if becomes available.");
        displayNormal(string.toString(), hazards);
        return;
      }
```

```
    }
  }

}
```

Main.java

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

class Main {
    private static void runTests() throws FileNotFoundException{
        File dir = new File("tests");
        File[] directoryListing = dir.listFiles();
        IoT_HTL iot;
        int speed;
        boolean cc_active;
        String lht;
        int lhd;
        int lhrs;

        String rtht;
        int rthd;
        int rthrs;

        String fht;
        int fhd;
        int fhrs;

        String rht;
        int rhd;
        int rhrs;
        int count = 1;
        for (File child : directoryListing) {
            System.out.println("----------------\nTEST CASE " + count +
"\nFile: " + child.getName() + "\n----------------");
            iot = new IoT_HTL();
            Scanner scanner = new Scanner(child);

            speed = scanner.nextInt(); scanner.nextLine();
            cc_active = ((scanner.nextLine().equals("active")) ? true : false);
            iot.setSpeed(speed);
            iot.cc_active = cc_active;

            lht = scanner.nextLine();
            lhd = Integer.parseInt(scanner.nextLine());
            lhrs = Integer.parseInt(scanner.nextLine());

            rtht = scanner.nextLine();
            rthd = Integer.parseInt(scanner.nextLine());
            rthrs = Integer.parseInt(scanner.nextLine());

            fht = scanner.nextLine();
            fhd = Integer.parseInt(scanner.nextLine());
            fhrs = Integer.parseInt(scanner.nextLine());

            rht = scanner.nextLine();
            rhd = Integer.parseInt(scanner.nextLine());
            rhrs = Integer.parseInt(scanner.nextLine());

            iot.leftSensor.checkForHazard(lhd, lht, 2, lhrs);
            iot.rightSensor.checkForHazard(rthd, rtht, 3, rthrs);
            iot.frontSensor.checkForHazard(fhd, fht, 0, fhrs);
            iot.rearSensor.checkForHazard(rhd, rht, 1, rhrs);
            iot.sensorFusion();
            iot.decision(iot.detectedHazards);



            scanner.close();
            count++;
        }
    }

    private static void logonScenario(Scanner scanner) {

        IoT_HTL iot = new IoT_HTL();
        iot.userMap.put(1234, new Driver("John Doe", 8079));
        iot.technicianMap.put(555, new Technician("Professional Tech", 56789));


        System.out.print("ENTER PIN: ");
        User user = iot.verifyUser(scanner.nextInt());
        scanner.nextLine();

        if (user == null) {
            System.out.println("Invalid Pin.");
        }
        else {
            if (user.getClass() == Driver.class) {
                Driver driver = (Driver) user;
                System.out.println("Name: " + driver.getName() + "\nID: " +
driver.getIdNum());

            }
            else if (user.getClass() == Technician.class) {
                Technician tech = (Technician) user;
                System.out.println("Technician logged in.\nName: " +
tech.getName() + "\nID: " + tech.getEmployeeId());
            }
        }

    }

    public static void main(String[] args) throws FileNotFoundException {
        Scanner input = new Scanner(System.in);
        String cmd = "";
        while (!cmd.equals("quit")) {
            System.out.print("Enter Testing Scenario [driving|logon]>_");
            cmd = input.nextLine().trim().toLowerCase();
            switch(cmd) {
                case "driving":
                    runTests();
                    break;
                case "logon":
                    logonScenario(input);
                    break;
                case "quit":
                    break;
                default:
                    System.out.println("Please enter valid command, or \'quit\' to exit.");
            }

        }


        input.close();

    }


}
```

Sensor.java

```


public class Sensor {
  int side;
  Hazard detectedHazard;
  public Sensor(int side) {
    this.side = side;
  }

  public Hazard checkForHazard(int distance, String name, int location, int rs)
{
    detectedHazard = new Hazard(distance, name, rs, location);
    return detectedHazard;
  }


  public String getSensorLocation() {
    if (side == IoT_HTL.LEFT) {
      return "left";
    }
    if (side == IoT_HTL.RIGHT) {
      return "right";
    }
    if (side == IoT_HTL.FRONT) {
      return "front";
    }
    if (side == IoT_HTL.REAR) {
      return "rear";
    }
    return null;    //case should never be reached
  }


}
```

Technician.java

```
class Technician extends User {
  private int employeeId;

  public Technician(String name, int employeeId) {
    super(name);
    this.employeeId = employeeId;
  }

  public int getEmployeeId() {
    return employeeId;
  }

  public void generateLog() {

  }

}
```

User.java

```


class User {
  private String name;

  public User(String name) {
    this.name = name;

  }

  public String getName() {
    return name;
  }
  }
```

Section 7: Testing

7.1) Scenario-Based Testing

- 7.1.1) Activation of IoT HTL system
    - Input: Driver presses the “on/off” button while it is ready to be activated.
        - If the car isn’t moving at an acceptable speed, the IoT HTL system will not be able to activate.
    - Output: IoT HTL system is turned on. Visual feedback from the car’s dashboard is given, notifying to the driver that it has been activated.
- 7.1.2) Deactivation of IoT HTL system
    - Input: Driver presses the “on/off” button or presses on the car’s brakes while the IoT HTL system is activated.
    - Output: Iot HTL system is turned off. Visual feedback from the car’s dashboard is given, notifying to the driver that it has been deactivated.
- 7.1.3) Driver initiates self-parking
    - Input: Driver presses the self-park button
        - If vehicle is not stationary → display error message
        - If no nearby parking spots are available → display error message

        - If the selected spot is too narrow/small → display error message
        - If the driver reinitiates the self-park button → self-parking action will disengage
    - Output: The car is parked + a message is displayed to alert the driver
- 7.1.4) Driver using lane assist
    - Desired activation speed is set by the driver
    - If desired activation speed requirement is not met → will not utilize lane assist
    - If the car is not in a driving lane → will not utilize lane assist
    - If desired activation speed requirement is met
        - If driver gets close to crossing the lane line → display warning message
        - If driver crosses the lane line → car will be corrected back into its lane
        - If car signal is activated → lane assist is paused until signal is off
- 7.1.5) Hazard detection activation

7.2) Validation Testing

- 7.2.1) Maintenance of the desired speed set by driver
    - Input: Driver selects a suitable speed (40 mph) to cruise on and allows the vehicle to cruise at that speed.
    - Output: Both the current speed of the car and the set speed of the cruise control system match to be 40 mph after a certain amount of time.
- 7.2.2) Driver increments or decrements the speed while cruising

  - Input: Driver sets a speed of 50 mph to cruise on. The driver then increments the cruise speed by 10 mph by using the "+" button, then subsequently decrements the cruise speed by 5 mph by using the "-" button..
  - Output: Both the current speed of the car and set speed of the cruise control system are at 55 mph.
- 7.2.3) Information being displayed to the driver from the car's dashboard
  - Input: Driver performs any type of input, such as: turning on the vehicle, incrementing or decrementing the cruise control speed, and turning on/off the IoT HTL system.
  - Output: Depending on the input the driver has made, certain display messages will be presented on the car's dashboard afterwards.
    - Any errors or malfunctions with the system or car will be displayed as well.
- 7.2.4) Vehicle is not in appropriate speed range to activate IoT HTL system
  - Input: Driver cruises at a current speed of 20 mph, which is not in the acceptable range (25 mph to 95 mph) and attempts to activate the IoT HTL system.
  - Output: An error message is displayed saying "Could not activate the system. The current speed is not in an acceptable range to cruise, must go faster."
- 7.2.5) Automatic braking
  - Reaction distance = (speed ÷ 10) × 3 miles

  - If an imminent collision is detected within a reaction distance that is proportionate to the vehicle speed → partial braking is applied to slow down the vehicle
    - If driver doesn't manually apply brakes after partial braking and the reaction distance is under 100 feet → automatic braking will take over to stop the vehicle prior to collision & a warning will be displayed
- 7.2.6) Lane Assist
  - Minimum speed for activation can be set by the driver, otherwise: 25 mph
  - If vehicle exceeds activation speed for lane assist & crosses over the lane line → vehicle will be shifted back into the correct lane
- 7.2.7) Hazard Detection
  - If other vehicle is detected and a collision is expected → initiate emergency braking
- 7.2.8) Hazard Detection (road hazards / foreign objects)
  - If a road hazard is detected and a collision is expected → initiate emergency braking
- 7.2.9) Road Signal Detection
  - If signal is detected and identified as yellow or red → vehicle will slow down and come to a stop
  - Else → no altered action
- 7.2.10) Proximity Access Security

  - If the keyfob exits the set vicinity for the proximity security features → lock vehicle
  - If the keyfob enters the set vicinity for the proximity security features → driver’s door is unlocked when the handle is pulled

Text File Input Test Cases

<u>Test Case 1:</u> The car is driving with the left lane open, right lane closed, and a car mildly close in front slowing down. Cruise control is active. The car should slow down and switch to the left lane.

```
----------------
TEST CASE 1
File: test1.txt
----------------
Speed: 40 CC: true
**************************************************
Front:      Type: vehicle Distance: 300     Relative Speed: -20
Rear:       Type: vehicle Distance: 200     Relative Speed: 0
Left:       Type: lane    Distance: 2   Relative Speed: 0
Right:      Type: shoulder      Distance: 2   Relative Speed: 0
Close vehicle detected: Applying light brakes, change lanes left.
New Speed: 30 CC: false
**************************************************
```

<u>Test Case 2:</u> The car is driving with both lanes open and a car that is far in front slowing down. Cruise control is active. The car should maintain its speed and switch to the left lane.

```
----------------

TEST CASE 10
```

```
File: test2.txt

----------------
Speed: 40 CC: true
**************************************************
Front:          Type: vehicle Distance: 500        Relative Speed: -40
Rear:           Type: vehicle Distance: 400        Relative Speed: 20
Left:           Type: lane    Distance: 2   Relative Speed: 0
Right:          Type: lane    Distance: 2   Relative Speed: 0
Vehicle detected: Change lanes left, maintain speed.
New Speed: 40 CC: true
*************************************************
```

<u>Test Case 3:</u> The car is driving with closed lanes on both sides and a vehicle close in front slowing down. Cruise control is active. The car should brake and change lanes when possible.

```
----------------
TEST CASE 11
File: test3.txt
----------------
Speed: 60 CC: true
**************************************************
Front:          Type: vehicle Distance: 300        Relative Speed: -20
Rear:           Type: vehicle Distance: 500        Relative Speed: -25
Left:           Type: shoulder       Distance: 2   Relative Speed: 0
Right:          Type: vehicle Distance: 2   Relative Speed: 0
WARNING: CRASH DANGER
Applying heavy brakes, change lanes if becomes available.
New Speed: 40 CC: false
*************************************************
```

<u>Test Case 4:</u> The car is driving very fast with cars to both of its sides and a car that is very far in front slowing down. Cruise control is active. The car should slow down and turn off cruise control.

```
----------------
TEST CASE 12
File: test4.txt
----------------
```

```
Speed: 70 CC: true
**************************************************
Front:          Type: vehicle Distance: 800         Relative Speed: -20
Rear:           Type: null    Distance: 0   Relative Speed: 0
Left:           Type: vehicle Distance: 2   Relative Speed: 0
Right:          Type: vehicle Distance: 2   Relative Speed: 0
Applying brakes: Consider further braking, or lane change if becomes possible.
New Speed: 76 CC: false
*************************************************
```

<u>Test Case 5:</u> The car is driving with open lanes on both sides and no vehicle detected in front. Cruise control is active. The car should not make any changes.

```
----------------
TEST CASE 13
File: test5.txt
----------------
Speed: 30 CC: true
**************************************************
Front:          Type: null    Distance: 0   Relative Speed: 0
Rear:           Type: null    Distance: 0   Relative Speed: 0
Left:           Type: lane    Distance: 2   Relative Speed: 0
Right:          Type: lane    Distance: 2   Relative Speed: 0
Maintain course.
New Speed: 30 CC: true
*************************************************
```

<u>Test Case 6:</u> The car is driving with both sides unavailable and a car very close in front slowing down. Cruise control is active. The car should heavily brake and switch lanes if possible.

```
----------------

TEST CASE 14

File: test6.txt

----------------
Speed: 50 CC: true
**************************************************
Front:          Type: object  Distance: 150         Relative Speed: -50
```

```
Rear:       Type: null     Distance: 0   Relative Speed: 0
Left:       Type: object   Distance: 2   Relative Speed: 0
Right:      Type: vehicle  Distance: 2   Relative Speed: 0
WARNING: CRASH DANGER
Applying heavy brakes, change lanes if becomes available.
New Speed: 34 CC: false
************************************************
```

<u>Test Case 7:</u> The car is driving with the left lane occupied, right lane open, and a car that is far in front slowing down. Cruise control is active. The car should maintain speed and switch to the right lane.

```
----------------
TEST CASE 15
File: test7.txt
----------------
Speed: 30 CC: true
*************************************************
Front:      Type: vehicle  Distance: 500       Relative Speed: -20
Rear:       Type: null     Distance: 0   Relative Speed: 0
Left:       Type: shoulder       Distance: 2   Relative Speed: 0
Right:      Type: lane     Distance: 2   Relative Speed: 0
Vehicle detected: Change lanes right, maintain speed.
New Speed: 30 CC: true
************************************************
```

<u>Test Case 8:</u> The car is driving with closed lanes on both sides and no vehicle detected in front. Cruise control is active. The car should maintain speed with no changes.

```
----------------
TEST CASE 16
File: test8.txt
----------------
Speed: 55 CC: true
*************************************************
Front:      Type: null     Distance: 0   Relative Speed: 0
Rear:       Type: null     Distance: 0   Relative Speed: 0
Left:       Type: vehicle  Distance: 0   Relative Speed: 0
Right:      Type: shoulder       Distance: 2   Relative Speed: 0
Maintain course.
```

```
New Speed: 55 CC: true
*************************************************
```

Test Case 9: The car is driving with a vehicle detected in front at a far distance away with the left lane closed and the right lane open. Cruise control is active. The car should switch to the right lane and maintain its current speed.

```
----------------
TEST CASE 17
File: test9.txt
----------------
Speed: 65 CC: true
**************************************************
Front:     Type: vehicle Distance: 500     Relative Speed: 60
Rear:      Type: null    Distance: 0   Relative Speed: 0
Left:      Type: vehicle Distance: 2   Relative Speed: 0
Right:     Type: lane    Distance: 2   Relative Speed: 0
Maintain course.
New Speed: 65 CC: true
*************************************************
```

Test Case 10: The car is driving with the left lane closed and the right lane open, and there is a close vehicle detected in front. Cruise control is active. The car should apply slight brakes and change lanes to the right.

```
----------------
TEST CASE 2
File: test10.txt
----------------
Speed: 45 CC: true
**************************************************
Front:     Type: vehicle Distance: 250     Relative Speed: -50
Rear:      Type: null    Distance: 0   Relative Speed: 0
Left:      Type: vehicle Distance: 2   Relative Speed: 0
Right:     Type: lane    Distance: 2   Relative Speed: 0
Close vehicle detected: Applying light brakes, change lanes right.
New Speed: 34 CC: true
*************************************************
```

<u>Test Case 11:</u> The car is driving with the left lane open and the right lane closed, and there is an object that is far in front. Cruise control is active. The car should maintain its speed and change lanes to the left.

```
----------------
TEST CASE 3
File: test11.txt
----------------
Speed: 45 CC: true
**************************************************
Front:          Type: object  Distance: 350         Relative Speed: -45
Rear:           Type: null    Distance: 0   Relative Speed: 0
Left:           Type: lane    Distance: 2   Relative Speed: 0
Right:          Type: object  Distance: 2   Relative Speed: 0
Object detected: Changed lanes left, maintain speed
New Speed: 45 CC: true
*************************************************
```

<u>Test Case 12:</u> The car is driving with the right lane open and the left lane closed, and there is an object that is far in front. Cruise control is active. The car should maintain its speed and change lanes to the right.

```
----------------
TEST CASE 4
File: test12.txt
----------------
Speed: 65 CC: true
**************************************************
Front:          Type: object  Distance: 400         Relative Speed: -65
Rear:           Type: null    Distance: 0   Relative Speed: 0
Left:           Type: vehicle Distance: 2   Relative Speed: 0
Right:          Type: lane    Distance: 2   Relative Speed: 0
Object detected: Changed lanes right, maintain speed
New Speed: 65 CC: true
*************************************************
```

<u>Test Case 13:</u> The car is driving with both lanes open and there is an object that is close in front. The car heavily brakes and cruise control is deactivated.

```
----------------
TEST CASE 5
File: test13.txt
----------------
Speed: 50 CC: true
**************************************************
Front:          Type: object  Distance: 150         Relative Speed: -50
Rear:           Type: null    Distance: 0   Relative Speed: 0
Left:           Type: lane    Distance: 2   Relative Speed: 0
Right:          Type: lane    Distance: 2   Relative Speed: 0
WARNING: CRASH DANGER
Applying heavy brakes, change lanes if becomes available.
New Speed: 34 CC: false
**************************************************
```

<u>Test Case 14:</u> The car is driving with both lanes closed and there is an object that is close in front. The car heavily brakes and cruise control is deactivated.

```
----------------
TEST CASE 6
File: test14.txt
----------------
Speed: 45 CC: true
**************************************************
Front:          Type: object  Distance: 100         Relative Speed: -45
Rear:           Type: null    Distance: 0   Relative Speed: 0
Left:           Type: object  Distance: 2   Relative Speed: 0
Right:          Type: vehicle Distance: 2   Relative Speed: 0
WARNING: CRASH DANGER
Applying heavy brakes, change lanes if becomes available.
New Speed: 30 CC: false
**************************************************
```

<u>Test Case 15:</u> The car is driving with both lanes open and nothing in front. However, the car is slowly moving towards the right. Cruise control is active. The car should maintain its course and speed and make a lane correction to the left.

```
----------------
TEST CASE 7
File: test15.txt
----------------
```

```
Speed: 30 CC: true
**************************************************
Front:      Type: null    Distance: 0   Relative Speed: 0
Rear:       Type: null    Distance: 0   Relative Speed: 0
Left:       Type: lane    Distance: 2   Relative Speed: 1
Right:      Type: lane    Distance: 2   Relative Speed: -1
Lane Correction Left
Maintain course.
New Speed: 30 CC: true
**************************************************
```

Test Case 16: The car is driving with the left lane open and a vehicle on the right. No object or vehicle is detected in front. Vehicle is slowly moving to the left. Cruise control is active. The car should maintain its course and speed and make a lane correction to the right.

```
----------------
TEST CASE 8
File: test16.txt
----------------
Speed: 45 CC: true
**************************************************
Front:      Type: null    Distance: 0   Relative Speed: 0
Rear:       Type: null    Distance: 0   Relative Speed: 0
Left:       Type: lane    Distance: 1   Relative Speed: -1
Right:      Type: vehicle Distance: 2   Relative Speed: 3
WARNING: lane approaching from left.
Maintain course.
New Speed: 45 CC: true
**************************************************
```

Test Case 17: The car is driving with the left lane open and a vehicle approaching from the right. Cruise control is active. Gives a warning that the vehicle is approaching from the right and the car should maintain its speed.

```
----------------
TEST CASE 9
File: test17.txt
```

```
----------------
Speed: 55 CC: true
**************************************************
Front:          Type: null      Distance: 0   Relative Speed: 0
Rear:           Type: null      Distance: 0   Relative Speed: 0
Left:           Type: lane      Distance: 2   Relative Speed: 0
Right:          Type: vehicle Distance: 2   Relative Speed: -6
WARNING: vehicle approaching from right.
Maintain course.
New Speed: 55 CC: true
*************************************************
```