The Adam Savages

Alset

Angelo Saez, Athena Kiriakoulis, Alan Fink, Jamie Rollins

Version 1.6

Feb 2, 2022

CS 347 Section A

Table of Contents

Section 1

Introduction to the Project:

Our team, the Adam Savages, will be developing software for a self driving car for the startup company, Alset. Our task is to use an IoT engine to implement features that aid in driver assistance, all in an effort to further our goal of making the cars we work with safer, cheaper, and more efficient. Additionally, everything we develop will be from scratch. To achieve this goal we will focus on a number of features, principally a feature to provide drivers with more information about cars in the driver's blind spot and help make quick decisions when dealing with this vulnerability.

The features that will be implemented to the car first will be those of a level one driving automation, the vehicle shares control with the driver. A level one car handles features such as cruise control, controls the car's speed depending on the distance of the car in front. Park assist features would also be included in a level one car. Many safety features add to level one autonomy for cars. Later, we will move to a level two driving system, which handles steering, acceleration, and braking. This level is often referred to as "hands-off" even though the driver must keep their hands on the steering wheel at all times. Level three vehicles are called "eyes-off" because the driver is allowed to pay attention to other things besides the road, such as their phone, but they need to keep a hand on the wheel in case intervention is needed. Level four cars require no intervention at all, allowing the driver to even sleep during the drive. However a level four full self-driving system

can only be activated in specified areas or traffic jams. Finally, a level vehicle requires absolutely no human interaction in any scenario.

Our first step towards a self-driving car is addressing the management of blind spots. These are sections surrounding the car which cannot be directly observed by the driver. To tackle this issue, we will expand on the idea of front and rear cameras. Although many modern cars have built-in cameras all around the vehicle, the car only allows you to see the view in the front and the back. However, blind spots can also appear at the sides of the car, specifically the back half of the car. Therefore, we propose to add the capability for the driver to see these side blind spots as well.

When a car is detected in a blind spot, the driver should be given a view of how close the car is to them. This view from the camera will be of the back half of the car. Additionally, when a car from an adjacent lane moves into one of the blind spots, the camera feed will be shown to the driver, similar to the backup camera used in other vehicles when they are backing out or parking. This camera feed will show the portion of the car in the blind spot, so the driver can see how close they are to the approaching vehicle. If the cars are too close and the driver is too slow to react, the car will automatically slowly move away from the approaching vehicle.

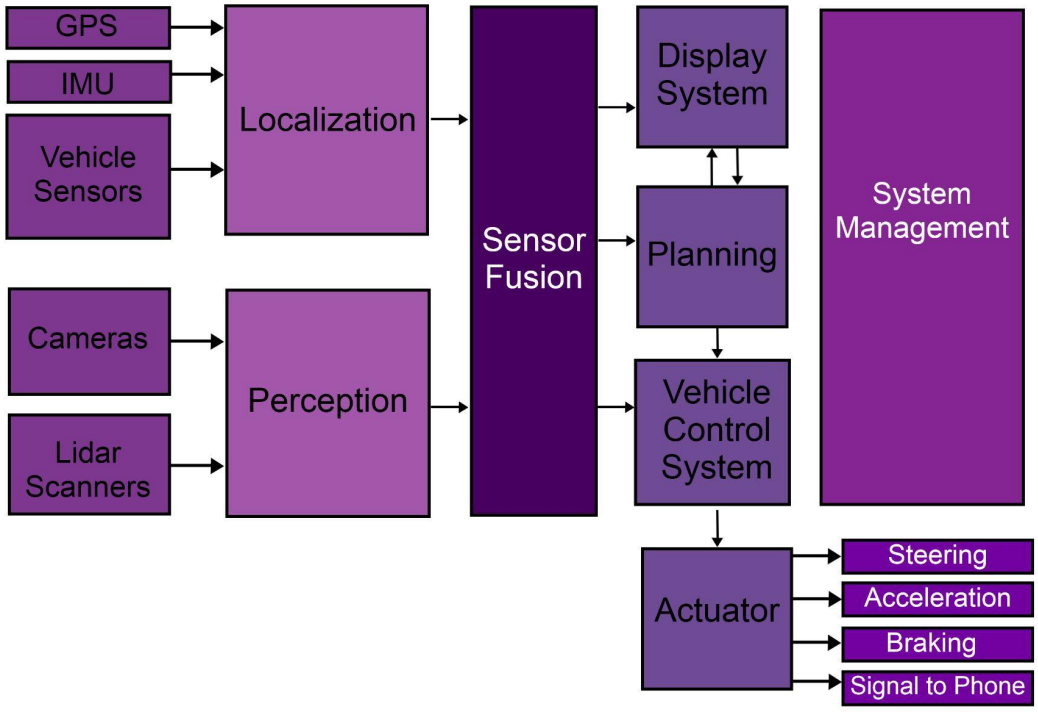
To approach the development of this software, we will use a mission critical style. Our team has a definitive end goal which we will work towards. The testing phase will be a longer phase towards the end of the project. The test cases will be created based on the designs of our features; therefore, we have to be very precise with our requirements of our software. We will be working on this project over a period of three months. Our role in this project is only as software developers, not

engineers, so we will be using the three months only for implementing features that make the car closer and closer to self- driving.

The importance of this software is to keep drivers and their vehicles safe during lane merging or backing up while parked. A driver's environment and the situation can change drastically at any moment, especially when they must deal with blind spots which can leave them vulnerable. Thus, any and all additional information is crucial to the passenger's safety. All in all, our software will build on this idea by allowing drivers to know more about the blind spot of their vehicle while driving, which will make any trip safer.

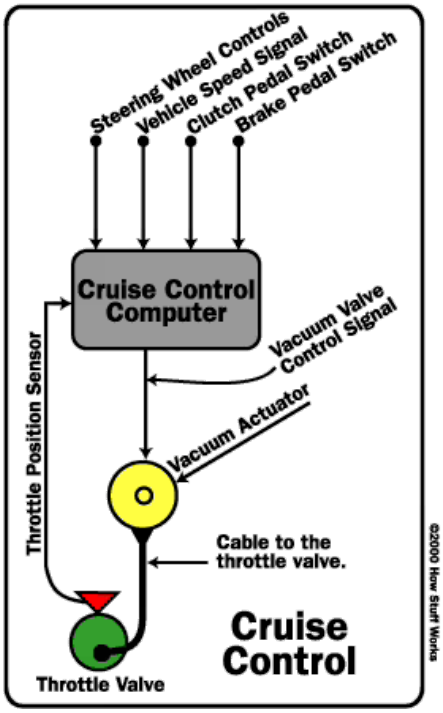
Section 2

Functional Architecture



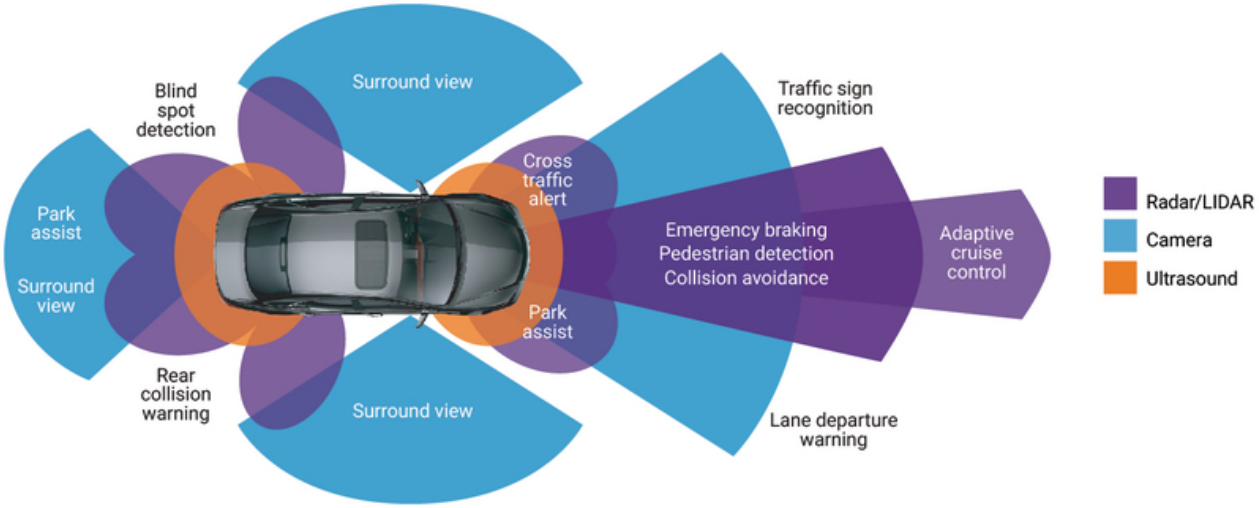
For this project our features are supposed to coordinate with each other to make a self driving car. The purpose of this section is to explain the functionality of each component as well as the input and output of each one. The components either receive input from the world around the car or the car itself. The output of all the features goes to controlling the car.

The first feature is cruise control. Its purpose is to reduce fatigue for drivers going long distances. The system imitates a human driving by automatically keeping the car driving at a set speed. This feature is controlled by a computer that gets input from the steering wheel controls, the vehicle speed signal, the clutch pedal switch, the brake pedal switch, and the throttle valve. The cruise control computer sends a vacuum valve control signal to the vacuum actuator. This means that instead of being controlled by the accelerator pedal, the speed is controlled by the vacuum actuator. Additionally, the output from the actuator, that is sent to the throttle valve, is sent back to the cruise control computer so it can track the position of the throttle. Our team will only be responsible for developing the software for the computer that controls the cruise control function, including reading the received input and giving the correct output; however, the cables and actuator would be handled by the engineering team working on the car.



The second feature is parking assistance. This component helps drivers park with greater precision. The system uses cameras in front of and in the back of the car to search for a suitable parking spot. Once a spot has been found, the software takes over steering the car into the spot. Even though the car takes over steering, the driver is still in control of acceleration, braking, and the transmission. Additionally, the software can be used to get a car out of a tight parking spot.

The next features will make the car a level two autonomous vehicle. These features are Advanced Driver Assistance Systems (A.D.A.S) in addition to automated steering, acceleration, and breaking. The goal of these functions is to make a "hands-off" vehicle. The role of A.D.A.S is to prevent deaths and injuries by reducing mistakes made by human error. Safety applications include pedestrian detection and avoidance, lane departure warning and correction, traffic sign recognition, automatic emergency braking, and blind spot detection. The system uses a combination of radar, cameras, and ultrasound to gather a three hundred sixty degree view of the outside world. Radar is used for cross traffic alerts, park assistance, emergency braking, pedestrian detection, collision avoidance, and adaptive cruise control in the front of the vehicle. In the back of the vehicle the radar is used for blind spot detection and rear collision warning. The cameras on the vehicle are placed where they can view all four sides of the car. They have surrounding views of both sides and are used for park assistance. Ultrasound aids with park assistance in the front and back of the car. All the data collected from the three types of sensors goes through the process of sensor fusion. This process is how the car is able to perceive the world and interpret the environment.



The goal of these features is to create a level five autonomous car. The functionality of these features should take control of steering, braking, speed, and parking of the vehicle. The input of these features either come from the surrounding area via cameras, radar, and ultrasonic sensors or from the car itself. When steering, parking, braking, and parking are all handled by the car the vehicle can at least be considered a level three autonomous car. When these functions can work in all situations and not just in georeferenced, specified areas the car can be considered level five.

Section 3

Requirements Analysis

1. Functional Requirements
    1.1. Cruise Control
        - Precondition: The car is on and driving at a speed s
        1.1.1. Driver turns on cruise control
            1.1.1.1. IOT HTL reads the current speed from sensor fusion and sends requested speed s to the Vehicle Control System(VCS)
        1.1.2. IOT HTL constantly verifies the current set speed
        1.1.3. In case driving speed is not s, report and aberrant (error)
        1.1.4. IOT HTL sends the cruise control speed to display
        1.1.5. IOT HTL sends the current speed to the log system
        - Postcondition: The IOT HTL maintains the speed at s mph
    1.2. Parking Assistance
        - Precondition: The car is in drive and going at a speed s < 10 mph
        1.2.1. Backing into a parking spot
            1.2.1.1. The driver goes to settings and turns on the rear camera when parking
        1.2.2. Pulling into a parking spot
            1.2.2.1. The driver turns on the front camera via settings when parking
        1.2.3. Parallel Parking

1.2.3.1. The driver goes to settings and and selects the parallel parking option so the car knows to check for obstructions in front and in back of it

1.2.4. IOT HTL reads data from sensor fusion

1.2.4.1. If valid spot is identified then it's outlined on the screen and driver is given an option to turn on Park Assist

1.2.5. IOT HTL steers car into designated spot

1.2.6. IOT HTL sends parking data to log system

- Postcondition: The IOT HTL brings the car to a complete stop and puts it in park

1.3. Advanced Driver Assistance System

1.3.1. Automated Steering

- Precondition: The car is on and in drive

1.3.1.1. Driver turns on automated steering system

1.3.1.2. IOT HTL displays automated steering symbol on screen

1.3.1.3. IOT HTL collects raw data from front cameras and sensors and puts them through the sensor fusion

1.3.1.4. IOT HTL changes car's course depending on obstructions in the cars path

- Postcondition: The car is still in drive and adjusts its own course, when necessary, without user input

1.3.2. Automated Acceleration

- Precondition: The car is on and driving at a speed s

1.3.2.1. The driver turns on automated steering

1.3.2.2. Displays that automated acceleration is on

1.3.2.3. IOT HTL verifies the constantly current speed of the vehicle

1.3.2.3.1. If the there are no obstructions in front of the car and the car is currently going under the speed limit then accelerate

- Postcondition: The IOT HTL maintains and/or adjusts the speed of the car

1.3.3. Automated Breaking

- Precondition: The car is on and in drive

1.3.3.1. The driver turns on automated breaking

1.3.3.2. Displays that automated breaking is on

1.3.3.3. IOT HTL collects data from sensor fusion

1.3.3.3.1. If there's an obstruction in the car's path then the IOT HTL will break slowing the car down and stopping if necessary

- Postcondition: The IOT HTL will slow and stop the car when necessary

2. Non-Functional Requirements

2.1. Performance Requirements

2.1.1. Reliability

2.1.1.1. Failure in time

2.1.1.1.1. No more than 3 errors in 1 million hours of operation

2.1.1.2. Runs a system check with every update

2.1.1.3. Downloads a backup of the previous system before update in the event that the update is interrupted the system has a backup to fall back on

2.1.2. Maintainability

2.1.2.1. Oil light turns on when oil needs to be changed

2.1.2.2. When check engine light is on, Alset technicians is notified so they can reach out to the vehicle's owner

2.1.3. Security

2.1.3.1. The IOT HTL can only be accessed by a certified technician

2.1.3.2. Accessed shall only be granted by user ID and password

2.1.3.3. Drivers do not have access to embedded Software

2.1.3.4. Camera/Sensor data used to create a Fortified Network Awareness Field(FNAF) around vehicle, where car is constantly monitored

2.1.3.5. Nightguard Protocol: time can be set for car to alert user when entity enters/lingers in FNAF

2.2. Operating Requirements

2.2.1. Physical constraints(size/weight)

2.2.1.1. Weight of the car: 2100 kg

2.2.1.2. Passenger Capacity: 5 adult passengers

3. User Interface

3.1. A light indicating cruise control is on

3.2. Dials

3.2.1. Displays fuel level of car

3.2.2. Displays speed of car

3.2.3. Displays warning lights

3.3. Digital Touchscreen

3.3.1. Displays weather

3.3.2. Displays radio channels

3.3.3. Displays changeable volume bar

3.3.4. Displays Air conditioning/Heating controls

3.3.5. Displays update prompt

3.3.6. Displays maps

3.3.6.1. Connects to google earth

4. Hardware IOS

4.1. The operating system

4.1.1. Runs on a custom lightweight build of Ubuntu release of Linux.

5. Software Update

- Precondition: car is on and standing(not moving), requesting permission to update software

5.1. Driver permits/denies software update

5.2. Upon permission the driver decides whether to update and restart the software or update and shutdown the car

5.3. Cloud updates the software and indicates completion

- Postcondition: new software version is installed

6. Network Connectivity

- Precondition: the car is parked in a garage with WiFi connectivity.

6.1. Network configuration

6.1.1. IOT WiFi sensor scans for available networks.

6.1.2. Userinterface allows the user to select which network to join and enter any needed passwords.

6.1.3. IOT will remember the network(s) the car has approved access to.

6.2. Automatic connection

6.2.1. On startup, the car will automatically join any nearby networks that it has the credentials for.

6.3. Cloud functionality

6.3.1. After the successful connection. The software will scan for any available updates.

6.3.2. Encryption and TLS for safe update connections.

6.3.3. The user interface will ask the users approval alongside a time estimate to download any software patches.

- Postcondition: parked cars will automatically connect to available & approved networks and will use this connection to keep the software up to date.

7. Event Logging Requirements

7.1. Car start time shall be logged in the logging system

7.1.1. The actor is the driver turning on the car because when this happens the data is logged

7.2. All sensor raw data shall be timed stamped and stored in the logging system

7.3. All drivers actions should be logged

7.4. All input / output data to modules sensor fusion, planning and ucs shall be logged

Section 4

Requirements Modeling

1. Use Case Scenarios
    a. **[Case A]:** User turns on the car (not the engine), and uses the touchscreen user interface to connect to a wifi network, and update the firmware on the vehicle. *make not of which classes are being affected and how they interact, EX: system management, sensors*
        i. Actors: driver, home network
        ii. Triggers: User turns on the car (without starting the engine)
        iii. Action: Once the network module establishes connection with the network and the Vehicle Sensors confirm that the vehicle is not moving, the components in Localization that provide cloud functionality will allow the firmware to be updated in System Management.
        iv. Alternative paths:
            1. user denies the software update
        v. Exceptions:
            1. attempted to update the software while the engine is active
            2. no available networks found
            3. failure to connect to the network
            4. software update interrupted

b. **[Case B]:** User is already driving and embarking on a cross country road trip. The user activates cruise control for a long stretch of the trip.
    i. Actors: driver
    ii. Triggers: driver activates the cruise control functionality
    iii. Action: Once triggered by buttons that communicate with Localization, the perception module will allow the user to engage in long-term cruise control. This cruise control receives input through the Perception module including the cameras and lidar scanners and adjusts the vehicle's movement using the Vehicle Control System and the Actuator.
    iv. Alternative paths:
        1. user turns off the cruise control
        2. If car speed is zero then interface notifies driver
    v. Exceptions:
        1. if the break triggers the interface will notify the driver that the cruise control has been interrupted

c. **[Case C]:** User is already driving the car and the passenger begins to operate the digital touch interface. They navigate through a variety of apps.
    i. Actors: driver, front-seat passenger
    ii. Triggers: passenger begins to operate the interface
    iii. Action: The touch display within the Localization module will communicate with the sensor fusion which works with the display system to allow passengers of the car to operate the

touch display while leaving the Vehicle Control System unchanged.

iv. Alternative paths:

1. passenger decides to stop using the touch interface

v. Exceptions:

1. unauthorized passenger tries to change system settings without the admin password

d. **[Case D]:** User is already driving and the car needs to pull off a narrow parallel park in a busy parking lot.

i. Actors: driver, nearby cars

ii. Triggers: driver activates the parallel park functionality

iii. Action: Once the feature is engaged, the sensor fusion takes input from Perception(cameras and Lidar Scanners). Because the data is being used to automatically parallel park, the sensor fusion tells the Display system to signify to the driver that the feature has been engage and provide a rear camera view of the vehicle's parking. The input data is also handed to planning from the sensor fusion so the can calculate if it is possible to park in the spot and how to maneuver into the spot if possible. Once the path the car will take has been determined, directions from Planning will be sent to the Vehicle Control System, then the Actuator, which controls steering,braking, and acceleration, to be carried out.

iv. Alternative paths:

1. driver decides to cancel the automated parking process and manually park somewhere
2. driver decides to cancel the automated parking process and reactivates it somewhere else

v. Exceptions:

1. parking process interrupted by nearby obstacle

e. **[Case E]:** User has the car in drive and engages Advanced Driver Assistance System for Collision Avoidance

i. Actors: Driver, surrounding environment

ii. Triggers: Driver activates ADAS feature

iii. Action: The user turns on the ADAS system from the digital interface, then the sensor fusion takes in input from the Perception. The sensor fusion uses the input to figure if/how many obstacles to the car they are in addition to how fast they are moving. When the car is in danger of a collision the driver is warned via auditory and visual cues; therefore from the sensor fusion data is returned to the Display System. The Display System is where the functions that play audio and visual cues are called. Meanwhile, the car automatically accelerates if the object is behind the car, breaks if the object is in front of the car, or steers away from the object if it is on one of the sides, before the data is sent to the vehicle control system it must first be sent to planning where which of the three available actions are decided and then the decision is sent to the vehicle control system to be executed.

iv. Alternative paths:

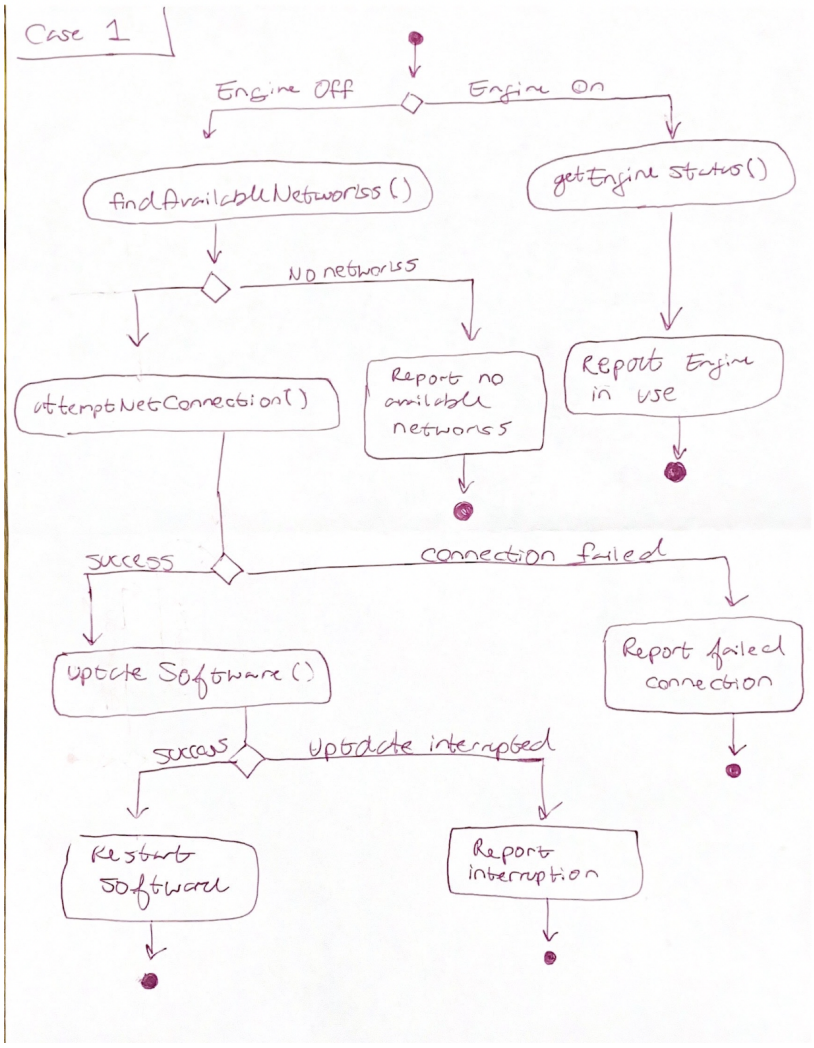
1. Driver turns off ADAS

v. Exceptions:

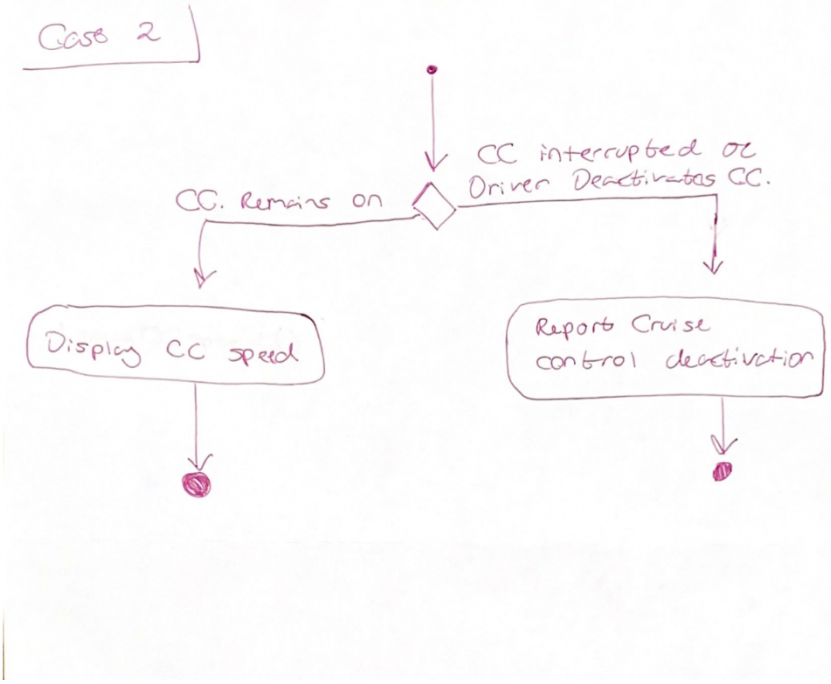
1. Car receives input from driver, which takes precedence over ADAS feature

2. Activity Diagrams

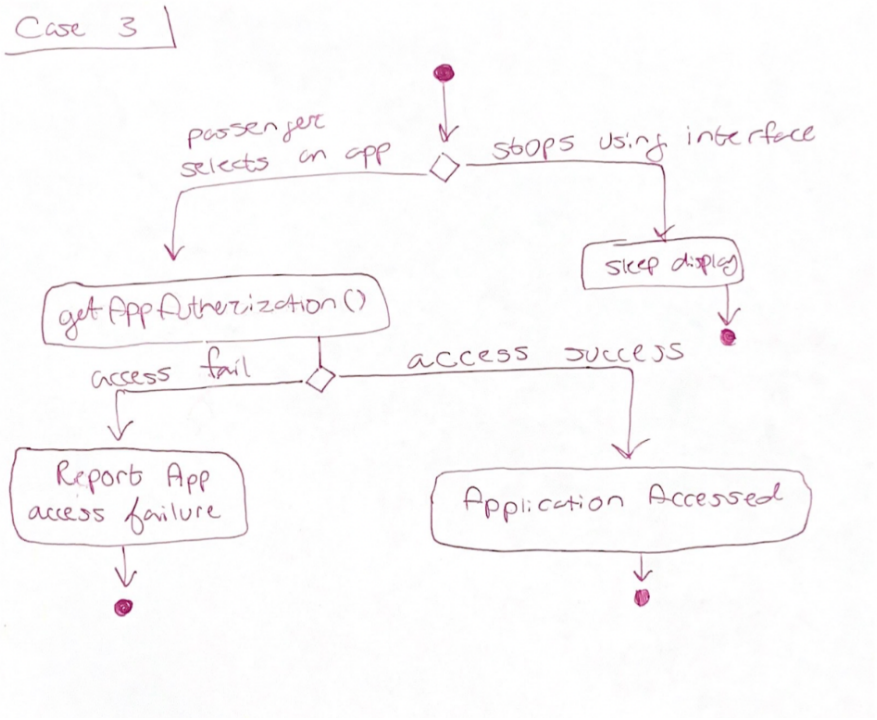
a.



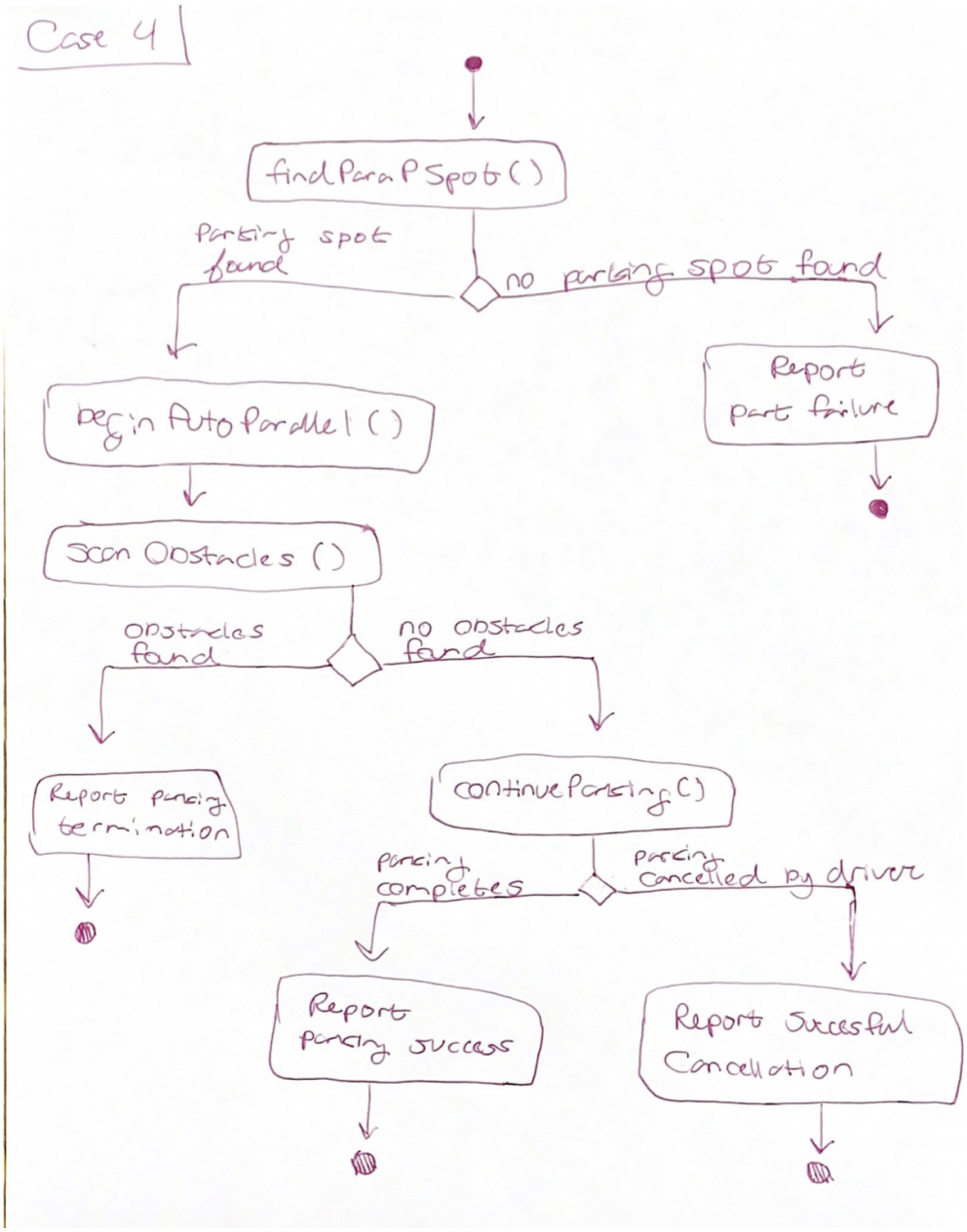
b.


Case 2
CC interrupted or
Driver Deactivates CC.
CC. Remains on
Display CC speed
Report Cruise
control deactivation

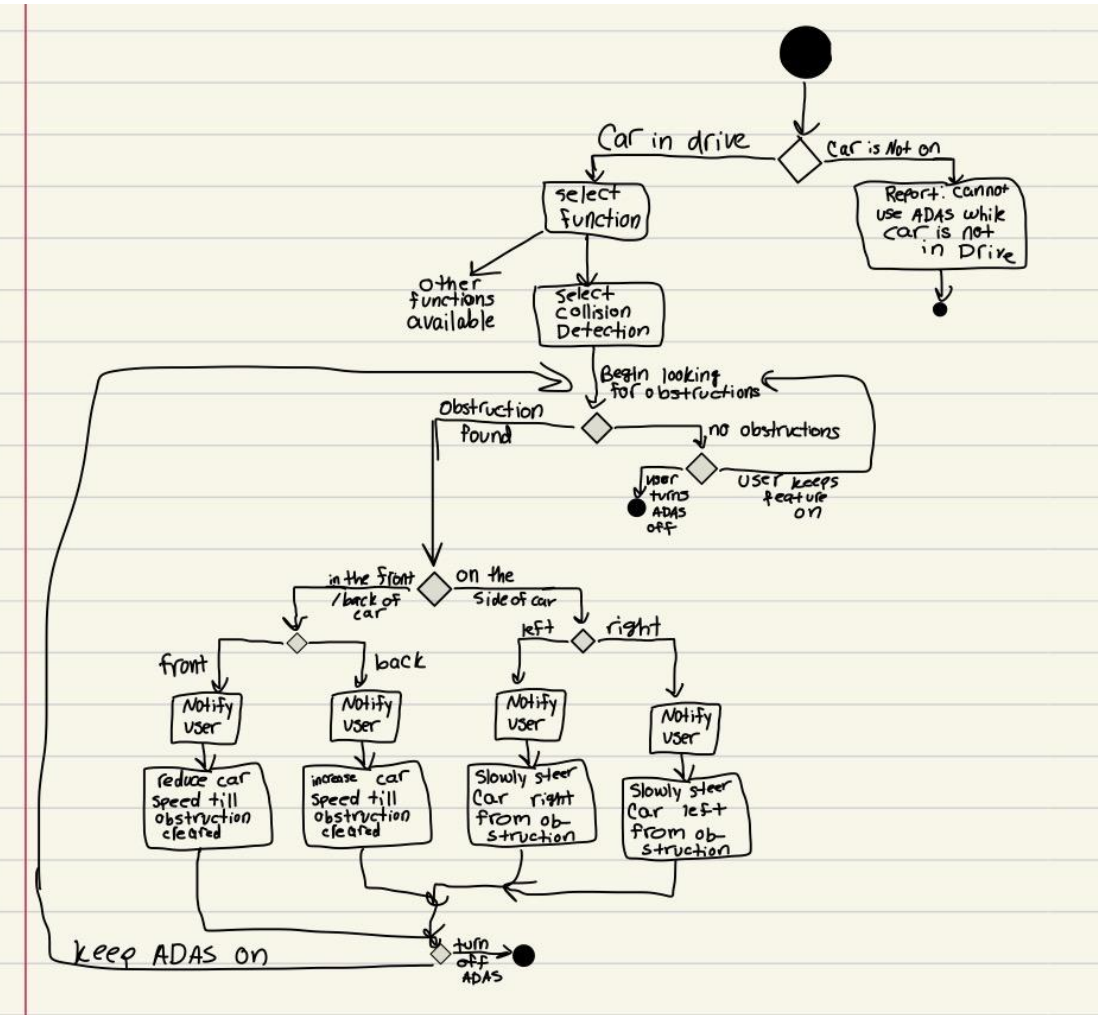
c.


Case 3
passenger
selects an app
stops using interface
Sleep display
getAppAuthorization()
access fail
access success
Report App
access failure
Application Accessed

d.

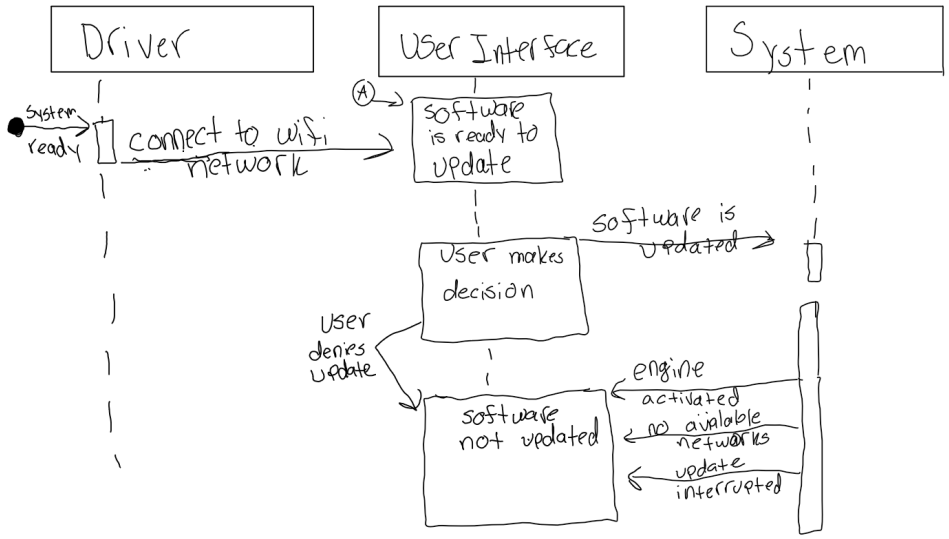

Case 4
findParaPSpot()
parking spot found
no parking spot found
Report part failure
beginAutoParallel()
scanObstacles()
obstacles found
no obstacles found
Report parking termination
continueParking()
parking completes
parking cancelled by driver
Report parking success
Report successful Cancellation

e. Case 5

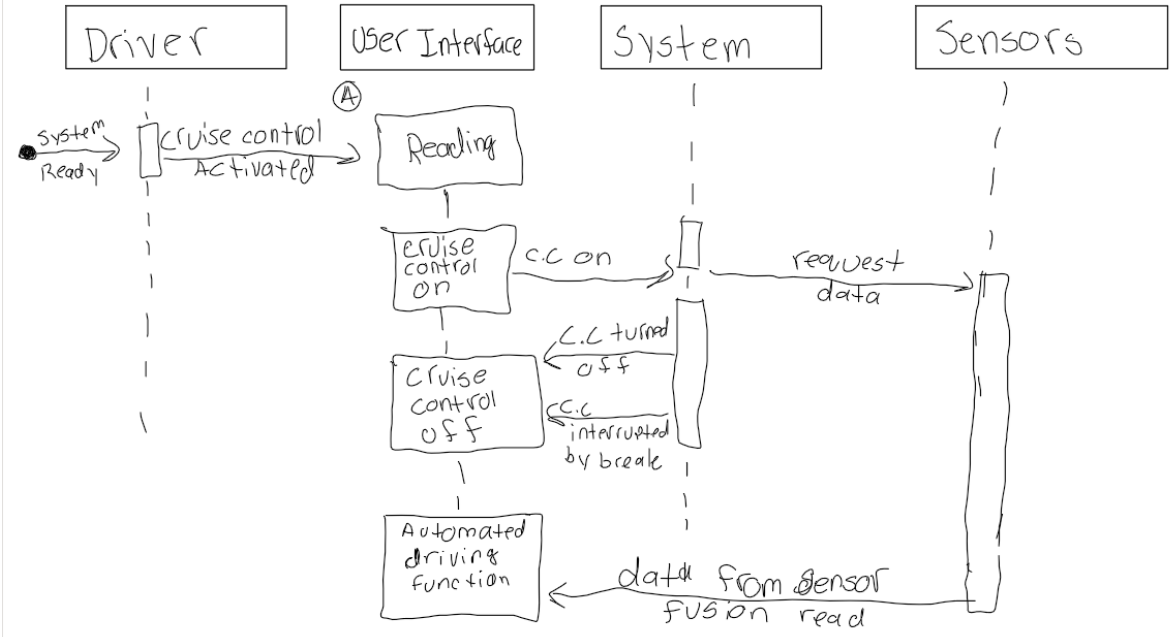


3. Sequence Diagrams

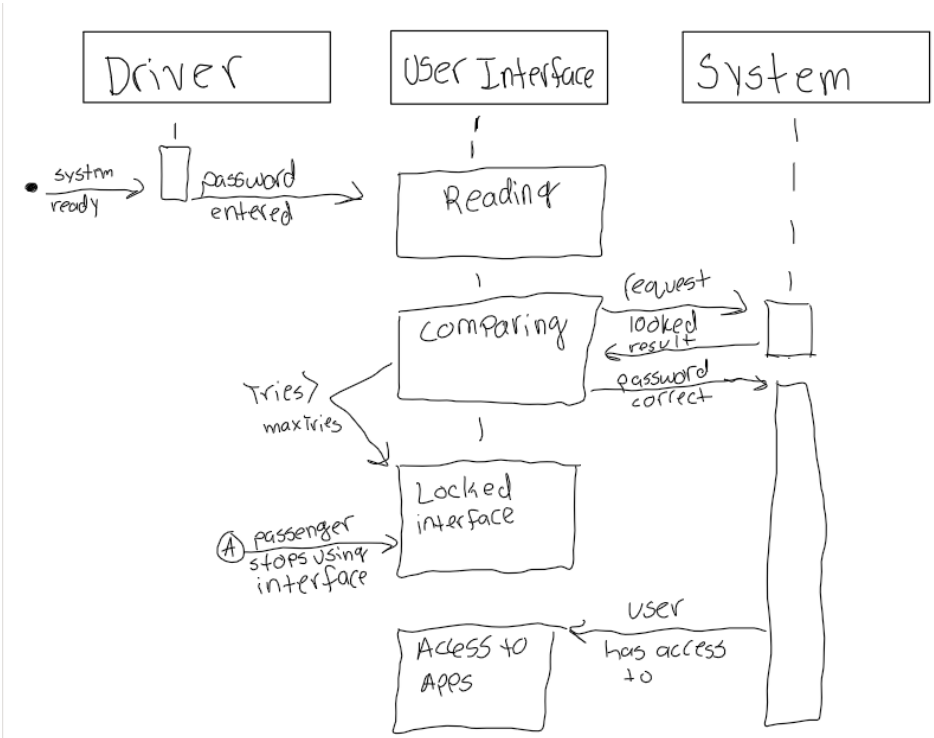
a. **Case A**



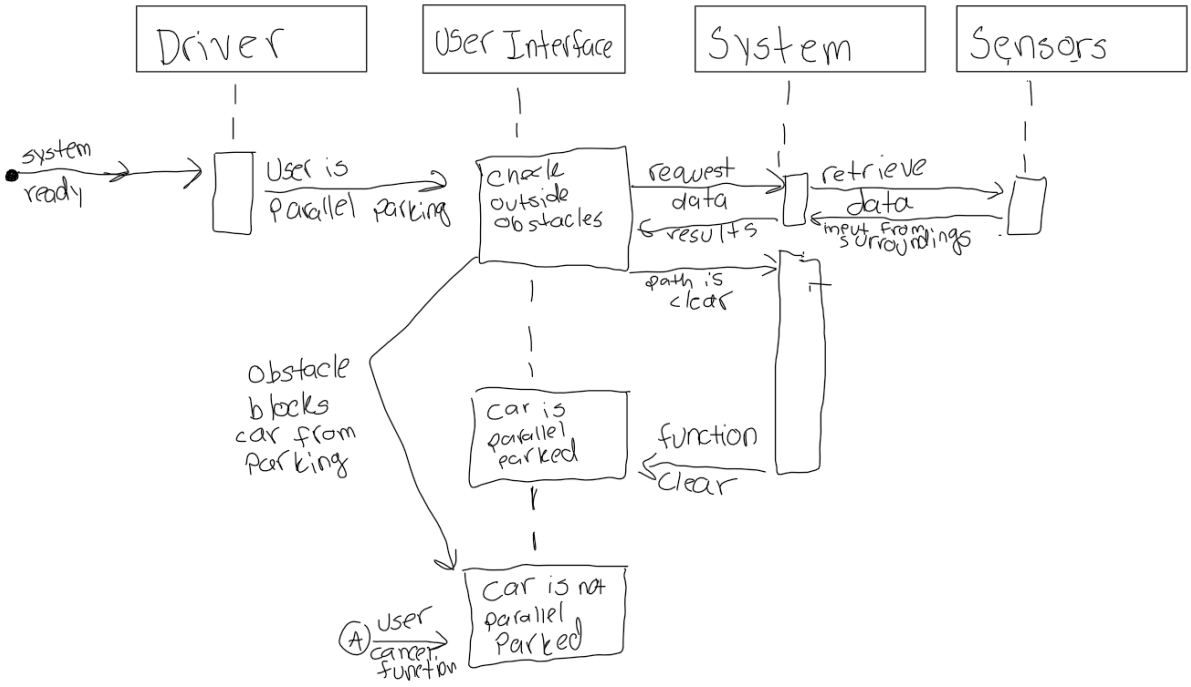
b. **Case B**



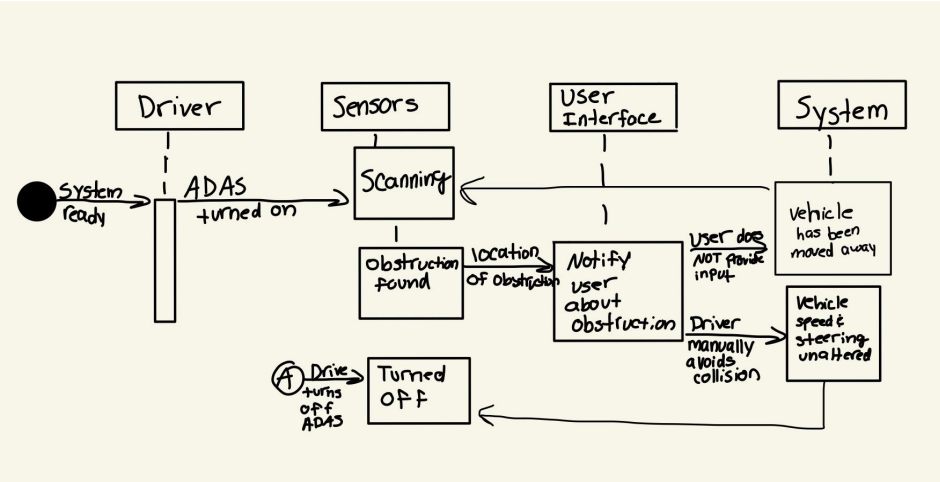
c. **Case C**



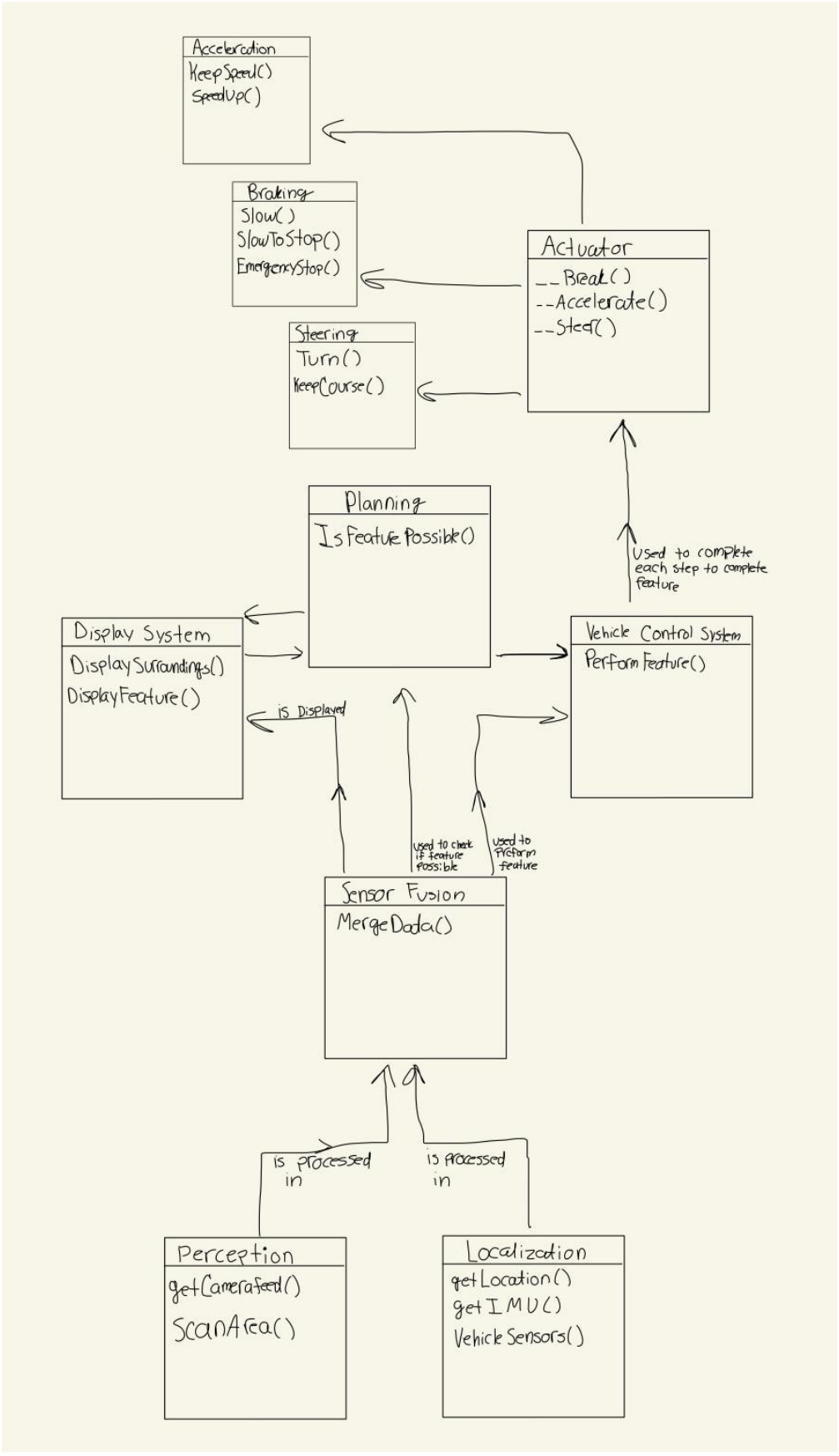
d. **Case D**



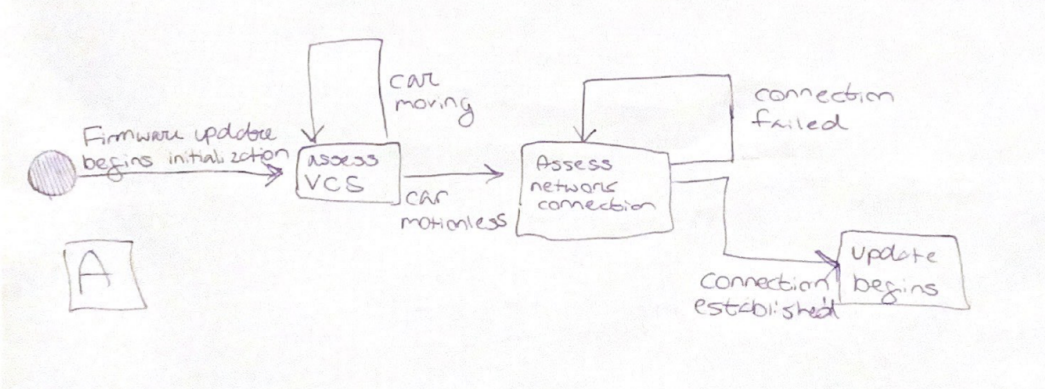
e. **Case E**


Driver
Sensors
User Interface
System
System ready
ADAS turned on
Scanning
Vehicle has been moved away
Obstruction found
location of obstruction
Notify user about obstruction
User does NOT provide input
Vehicle speed & steering unaltered
Driver manually avoids collision
Drive turns off ADAS
Turned off

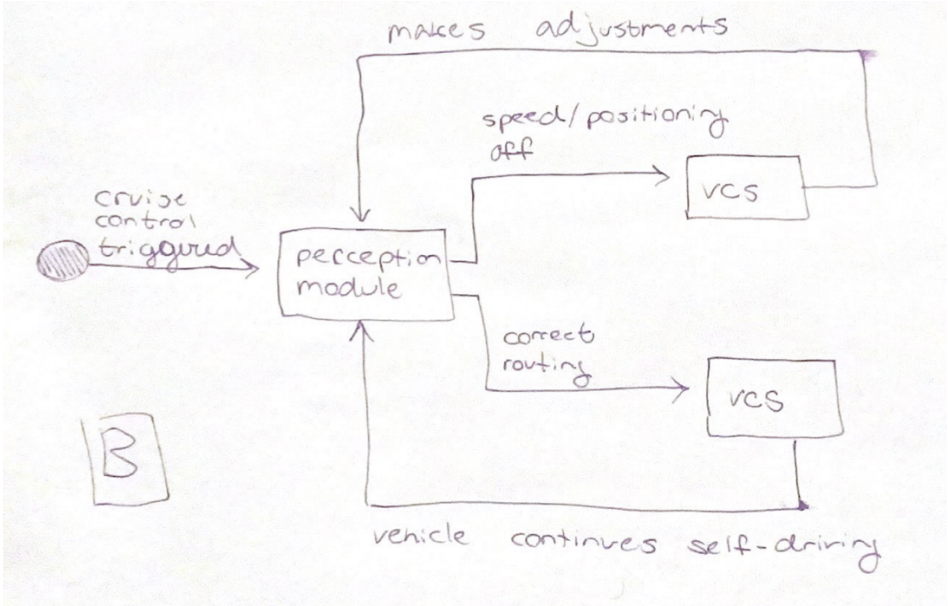
4. Class


Acceleration
KeepSpeed()
SpeedUp()
Braking
Slow()
SlowToStop()
EmergencyStop()
Steering
Turn()
KeepCourse()
Actuator
--Break()
--Accelerate()
--Steer()
Planning
IsFeaturePossible()
Used to complete each step to complete feature
Display System
DisplaySurroundings()
DisplayFeature()
Vehicle Control System
PerformFeature()
is Displayed
used to check if feature possible
used to Preform feature
Sensor Fusion
MergeData()
is processed in
is processed in
Perception
getCamerafeed()
ScanArea()
Localization
getLocation()
getIMU()
VehicleSensors()

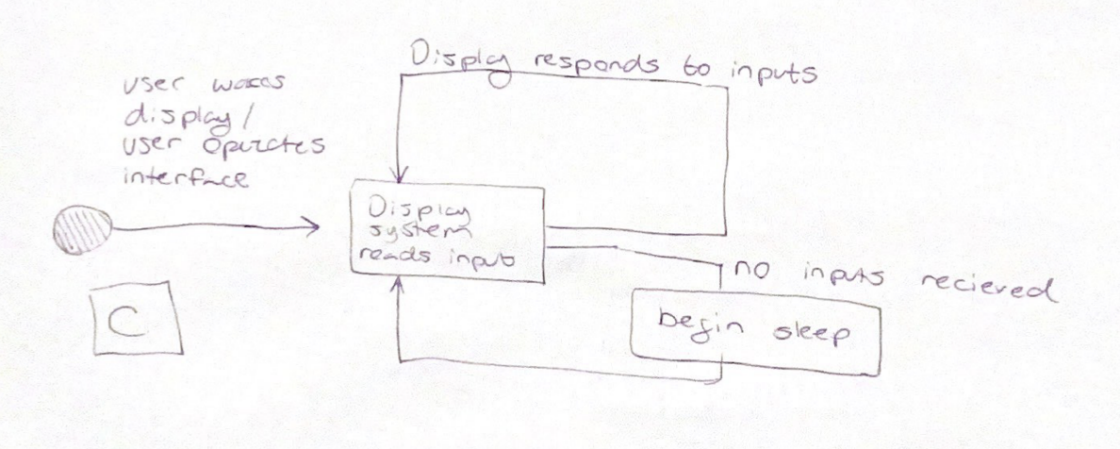
5. State Diagrams
    a. **Case A:**



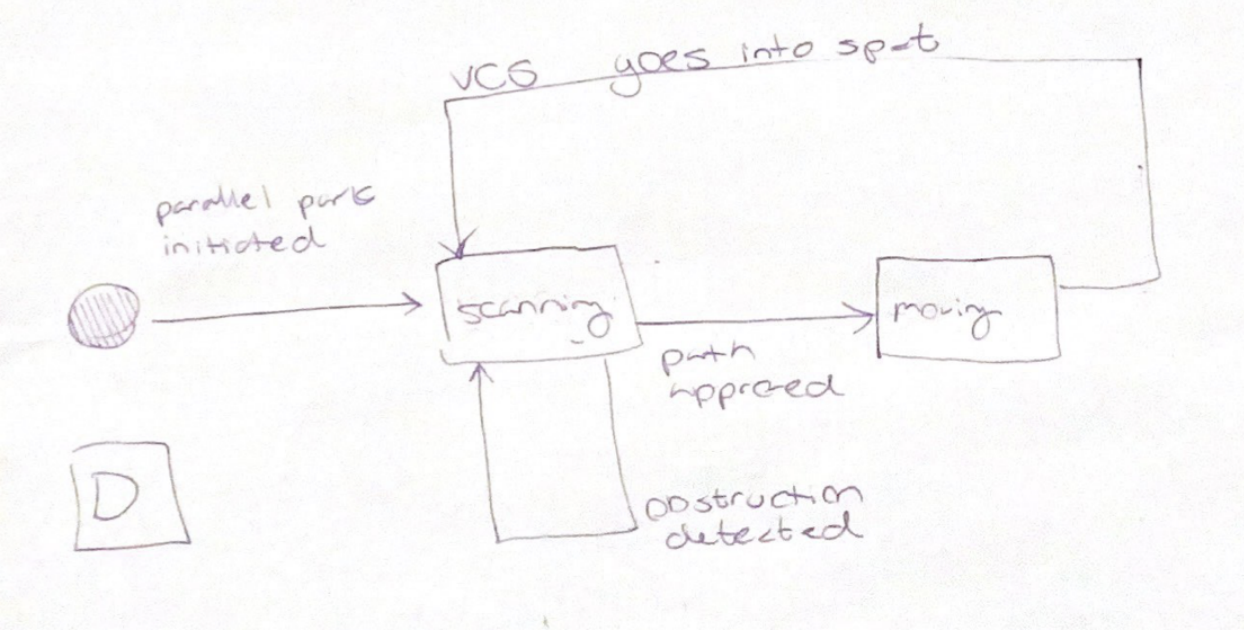
    b. **Case B:**



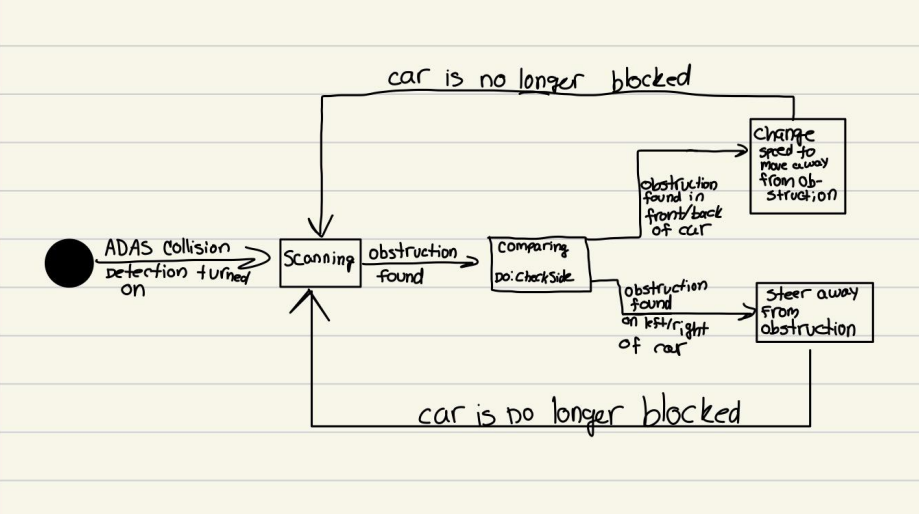
c. **Case C:**



d. **Case D:**



e. **Case E:**


car is no longer blocked
Change speed to move away from obstruction
obstruction found in front/back of car
ADAS collision detection turned on
Scanning
obstruction found
comparing
pa: checkSide
obstruction found on left/right of car
steer away from obstruction
car is no longer blocked

Section 5

Design

1. Software Architecture

1.1.1.

Architecture	**Pros**	**Cons**	**Fit or Not Fit**
Data Centered	● Components can be added/changed without concern of other clients	● Clients highly dependent on data ● Issues with Multiple client synchronization	Not a fit. Some of our components would have to be synchronized, which this does not encourage.
Data flow	● Allows for reusability since same filters can be reused	● Result needs to come from numerous pipes/filters, so interaction is difficult	Not a fit. This is not suited for the interactive software which our car relies upon.
Call return	● Program Structure easy to modify scale	● Vulnerable to internal data structure change, as it impacts the other modules	Not a fit. If we try to change the internal structure, we would have to consider all the other modules in the system.
Object oriented	● Easy to manage errors during runtime ● Reusability encouraged	● Can be difficult to determine necessary classes/objects	A fit. Our system uses many overlapping components, which make it easy to outline classes/objects, negating the only con.

Layered	• Layers help divide and conquer complex issues	• Difficult to clearly separate layers • Difficult structure for some systems	Not a fit. Our software would benefit more from an architecture which shows the communication between components, which not seen with these layers
Model View Controller	• Components can be independently tested	• Very complex • View can only get data through the controller, and cannot talk directly to the model.	Not a fit. This architecture is very complex, and it does not provide the free communication between all components, which we need.
Finite State machine	• Great for highlighting the states an application goes through	• Focuses mainly on states of software	Not a fit. This architecture focuses too much on the states the software can go through, while our team is looking for an architecture focused more on the components of the car, and how they interact.

We have decided to go with an Object-Oriented styled Architecture. First, OOA is more efficient because many of the features in our car not only require the

same if not identical input from car sensors, but also use similar functions to determine if a feature can be carried out and which of the car's hardware needs to carry out said feature. Both Perception and Localization data is handed off to the sensor fusion to be converted into an easier read data point for the rest of the functions that the features of the car use.

2. Interface Design
    2.1. Technician Interface
        2.1.1. A user interface that interacts with the system
    2.2. Driver Interface
        2.2.1. System management able to keep system up to date

There are three elements to consider when designing an interface: User Interface(UI), external interfaces to other systems/devices/networks/etc, and internal interfaces to other design components. UI focuses on creativity and user experience, while internal and external interfaces are more concerned with error checking and security features. Starting with the Driver interface, not only will this be a touch-screen that includes convenient apps, but will also be the place where the driver can turn on major features such as the A.D.A.S , software updates, network settings, etcetera. When the Driver turns on a feature the UI sends a signal to either the internal or external interface. If the feature activated requires the components of the vehicle to succeed the signal will be sent to the internal interface which will tell the sensor fusion to collect data and then send it to the functions that control the hardware components; however, if the Driver activates a feature such as network connectivity or software update the signal will be sent to the external interface where it can access the cloud system for the update. The driver will not have access to the embedded system, only the technician will have access to this system. From the Technician's Interface they will be able to access

the embedded software for update purposes and will be connected to it through the external interface.

3. Component-Level Design
    3.1. Sensor Component Design
        3.1.1. Localization Component Design
            3.1.1.1. getGPS()
            3.1.1.2. getIMU()
            3.1.1.3. getVehicleSensor()
        3.1.2. Perception Component Design
            3.1.2.1. getCameraFeed()
            3.1.2.2. ScanArea()
        3.1.3. Sensor Fusion Component Design
            3.1.3.1. getLocalization()
            3.1.3.2. getPerception()
            3.1.3.3. MergeData()
            3.1.3.4. Return combined data from perception and localization for a single environment around the car. Returns the value of MergeData
    3.2. Planning Component Design
        3.2.1. When a driver turns on a feature this is the first place that is notified, getFeature()
        3.2.2. getSensorFusion(): takes the single environment from Sensor Fusion to decide if a feature is possible
    3.3. Display Component Design
        3.3.1. getFeatureStatus()

3.3.1.1. What features are currently activated, if a feature is possible then it is activated

3.3.2. getData()

3.3.2.1. If the user uses apps for weather, music, this function

3.3.3. displayData()

3.4. Vehicle Control System Component Design

3.4.1. getData()

3.4.1.1. Calls planning function to get data

3.4.2. IsPossible()

3.4.2.1. Uses the data to calculate whether the feature can be executed successfully

3.4.2.2. If yes then send signal to actuator to complete feature else error

3.5. System Management Component Design

3.5.1. Handles updates and lets user view status of the system

3.5.2. CheckPssWrd()

3.5.2.1. Password authentication is required (can't be accessed while driving)

3.5.2.2. User inputs password

3.5.2.3. If password is correct then open system settings else if numTries < TotalTries then prompt user for password else lock system management for sent amount of time

3.5.3. IOSVer()

3.5.3.1. Returns current version of IOS

3.5.4. LoggedData()

3.5.4.1. If password entered is Technician's password then return logged data else error("Do not have access to embedded system")

3.6. Network Component Design

3.6.1. getNetwork()

3.6.1.1. Driver uses interface to choose from a list of available networks

3.6.2. Connect()

3.6.2.1. Try to connect to selected network

3.6.2.2. If successful then system is connected to network else error"Cannot connect to network"

Section 6

Coding

Main.py

```
import PySimpleGUI as sg
import os
from datetime import datetime
from test import *
"""
    Dashboard using blocks of information.


    Copyright 2020 PySimpleGUI.org
"""

theme_dict = {
    'BACKGROUND': '#437EA3',
    'TEXT': '#FFFFFF',
    'INPUT': '#F2EFE8',
    'TEXT_INPUT': '#000000',
    'SCROLL': '#F2EFE8',
    'BUTTON': ('#000000', '#C2D4D8'),
    'PROGRESS': ('#FFFFFF', '#C7D5E0'),
    'BORDER': 1,
    'SLIDER_DEPTH': 0,
    'PROGRESS_DEPTH': 0
}

# sg.theme_add_new('Dashboard', theme_dict)     # if using 4.20.0.1+
sg.LOOK_AND_FEEL_TABLE['Dashboard'] = theme_dict
```

```
sg.theme('Dashboard')

BORDER_COLOR = '#C7D5E0'
DARK_HEADER_COLOR = '#1B2838'
BPAD_TOP = ((20, 20), (20, 10))
BPAD_LEFT = ((20, 10), (0, 10))
BPAD_LEFT_INSIDE = (0, 10)
BPAD_RIGHT = ((10, 20), (10, 20))
TOP_FONT = ("Tw Cen MTfont", 15, "bold")
BUTTON_FONT = ("Bahnschrift SemiLight SemiConde", 10)
HEADER_FONT = ("HP Simplified", 20, "bold")
LEFT_FONT = ("Dubai Medium", 15, "bold")
TITLE_FONT = ("Yu Gothic UI Light", 40)

speed = 0
timer = 0
active_features = []
obstacles = []
technician_password_entry = ""
technician_password_answer = [ 1, 3]

#Obstructions
OFront = False
OBack = False
OLeft = False
ORight = False

#features
CruiseControl = False
AdvDriAsSys = False
```

```
Park = False
Sec = False

if not os.path.isdir("./logs"):
    try:
        os.mkdir("./logs")
    except OSError:
        print("Could not create logs directory")
    else:
        print("Created logs directory")

logfile = open("./logs/ALSETLogData.log", "w")

if os.path.getsize("./logs/ALSETLogData.log") > 0:
    logfile.write("\n")

logfile.write("[" + datetime.now().strftime("%H:%M:%S") + "] System Ready.\n")


#defining the various popup windows
def software_popup(win):  #popup asking to update
    layout = [
        [
            sg.Text(
                "Software update available. Would you like to proceed with software update?"
            )
        ],
        [
            sg.Button("Yes", key="SoftwareYes", enable_events=True),
            sg.Button("No", key="SoftwareNo", enable_events=True)
```

```
        ],
    ]
    win = sg.Window("My Popup",
                    layout,
                    modal=True,
                    grab_anywhere=True,
                    enable_close_attempted_event=True)
    event, value = win.read()
    if event == sg.WINDOW_CLOSE_ATTEMPTED_EVENT:
        event = "CANCEL"
    win.close()
    window.write_event_value(event, None)



def no_network_popup(win):  #no network available popup
    layout = [
        [sg.Text("No networks available. :(")],
        [sg.Button("ok", key="okButton", enable_events=True)],
    ]
    win = sg.Window("My Popup",
                    layout,
                    modal=True,
                    grab_anywhere=True,
                    enable_close_attempted_event=True)
    event, value = win.read()
    if event == sg.WINDOW_CLOSE_ATTEMPTED_EVENT:
        event = "CANCEL"
    win.close()
    window.write_event_value(event, None)
```

```
def networks_popup(win):
    layout = [
        [sg.Text("There is a network available. Connect to it?")],
        [[[sg.Button(f'{i} ', key=i)] for i in network_list],
         sg.Button("Cancel", key="ConnectNo", enable_events=True)],
    ]
    win = sg.Window("My Popup",
                    layout,
                    modal=True,
                    grab_anywhere=True,
                    enable_close_attempted_event=True)
    event, value = win.read()
    if event == sg.WINDOW_CLOSE_ATTEMPTED_EVENT:
        event = "CANCEL"
    win.close()
    window.write_event_value(event, None)



def securityTime(win):
    layout = [
        [sg.Text("How many minutes do you want the security timer to wait?")],
        [sg.Text('Time(in minutes):', size =(15, 1))],
        [sg.InputText(key='TimeyWimey')],
        [sg.Button("Submit", key="submitted", enable_events=True)]


    ]
    win = sg.Window("My Popup",
                    layout,
                    modal=True,
```

```
                grab_anywhere=True,
                enable_close_attempted_event=True)
    event, values = win.read()
    global timer
    timer += int(values['TimeyWimey'])
    if event == sg.WINDOW_CLOSE_ATTEMPTED_EVENT:
        event = "CANCEL"
    win.close()
    window.write_event_value(event, None)



def SecutityCam(win):
    layout = [
        [sg.Text("CameraFeed")],
        [sg.Image('image_file', size=(300,300))],
        [sg.Button("Submit", key="submitted", enable_events=True)]


    ]
    win = sg.Window("My Popup",
                layout,
                modal=True,
                grab_anywhere=True,
                enable_close_attempted_event=True)
    event, values = win.read()
    global timer
    timer += int(values['TimeyWimey'])
    if event == sg.WINDOW_CLOSE_ATTEMPTED_EVENT:
        event = "CANCEL"
    win.close()
    window.write_event_value(event, None)
```

```
def technician_popup(win):
    layout = [
        [sg.Text("Please enter the ALSET technician password")],
        [sg.Input(key='_PWRD_')],
        [sg.Button('Confirm'), sg.Button('Exit')]
    ]
    win = sg.Window("Technician Popup", layout, modal=True, grab_anywhere=True,
enable_close_attempted_event=True)
    event, values = win.read()
    global technician_password_entry
    technician_password_entry += values['_PWRD_']
    if event == sg.WINDOW_CLOSE_ATTEMPTED_EVENT:
        event = "CANCEL"
    win.close()
    window.write_event_value(event, None)


def getActiveFeaturesString():
    return active_features
    #return f'{active_features[i]}' for i in range(len(active_features))



def getSpeed():
    return speed



def getObstacles():
    return obstacles
```

```
top_banner = [[
    sg.Text('ASLET' + ' ' * 64,
            font=TOP_FONT,
            background_color=DARK_HEADER_COLOR),
    sg.Text('Tuesday 9 June 2020',
            size=(40, 1),
            justification='r',
            font=TOP_FONT,
            background_color=DARK_HEADER_COLOR)
]]

top = [
    [
        sg.Text('User Interface',
                size=(32, 1),
                justification='c',
                pad=BPAD_TOP,
                font=TITLE_FONT)
    ],
    [sg.T("version 1.0")],
]

block_2 = [[sg.Text('Features', font=HEADER_FONT)],
           [sg.Button('Cruise Control', font=BUTTON_FONT)],
           [sg.Button('Advanced Driver Assistance System', font=BUTTON_FONT)],
           [sg.Button('Network Settings', font=BUTTON_FONT)],
           [sg.Button('Security', font=BUTTON_FONT)],
           [sg.Button('Software', font=BUTTON_FONT)],
           [sg.Button('Parking', font=BUTTON_FONT)],
```

```
        [sg.Button('Technician\'s Interface', font=BUTTON_FONT)],
        [sg.Button('Off', font=BUTTON_FONT)]]

block_4 = [[sg.Text('Display', font=HEADER_FONT)],
        [sg.Text('Speed: ', font=LEFT_FONT)],
        [sg.Text(getSpeed(), key='-speed-', font=LEFT_FONT)],
        [sg.Text('Obstructions: ', font=LEFT_FONT)],
        [sg.Text(getObstacles(), key='-obstacles-', font=LEFT_FONT)],
        [sg.Text('Active Features: ', font=LEFT_FONT)],
        [
            sg.Text(getActiveFeaturesString(),
                key='-features-',
                font=LEFT_FONT)
        ]]

layout = [[
   sg.Column(top_banner,
          size=(960, 60),
          pad=(0, 0),
          background_color=DARK_HEADER_COLOR)
], [sg.Column(top, size=(920, 90), pad=BPAD_TOP)],
        [
          sg.Column(block_2, size=(450, 320), pad=BPAD_LEFT),
          sg.Column(block_4, size=(450, 320), pad=BPAD_RIGHT)
        ]]

window = sg.Window('Dashboard PySimpleGUI-Style',
              layout,
              margins=(0, 0),
              background_color=BORDER_COLOR,
```

```
                no_titlebar=True,
                grab_anywhere=True)



while True:  # Event Loop
    event, values = window.read()
    if event == sg.WIN_CLOSED or event == 'Off':
        break
    if event == 'Cruise Control':
        if CruiseControl == True:  #turn CC off
            CruiseControl = False
            logfile.write("[" + datetime.now().strftime("%H:%M:%S") +
                          "] Cruise Control deactivated.\n")
            active_features.remove('Cruise Control')
            window['-features-'].update(getActiveFeaturesString())


        else:  #turn CC on
             CruiseControl = True
             CC(vectorCC1)
             speed = vectorCC1[2]
             logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Cruise Control activated.\n")
             active_features.append('Cruise Control')
             window['-features-'].update(getActiveFeaturesString())
             window['-speed-'].update(getSpeed())


    if event == 'Advanced Driver Assistance System':
        if AdvDriAsSys == True:#turn ADAS off
            AdvDriAsSys = False
            logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Advanced Driver Assistance System
deactivated.\n")
```

```
            active_features.remove('ADAS')
            obstacles.remove(obstructions_to_str(obstr_dataADAS1))
            window['-features-'].update(getActiveFeaturesString())
            window['-obstacles-'].update(getObstacles())
        else: #Turn ADAS on
            ADAS(vectorADAS1)
            AdvDriAsSys = True
            logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Advanced Driver Assistance System
activated.\n")
            speed=vectorADAS1[1]
            active_features.append('ADAS')
            obstacles.append(obstructions_to_str(obstr_dataADAS1))
            window['-speed-'].update(getSpeed())
            window['-obstacles-'].update(getObstacles())
            window['-features-'].update(getActiveFeaturesString())
            window['-speed-'].update(getSpeed())
            window['-obstacles-'].update(getObstacles())
            window['-features-'].update(getActiveFeaturesString())


    if event == 'Network Settings':
        logfile.write("[" + datetime.now().strftime("%H:%M:%S") +
                      "] Network Settings opened.\n")
        sg.popup_timed('Loading available networks...')
        if not network_list:  #If no networks available
            no_network_popup(window)
        else:
            networks_popup(window)
    if event in network_list:  # Accept network connection
        sg.popup_timed('Connecting to network...')
        if isConnected:  #If car could could connect to network
            sg.popup('car connected.')
```

```
    else:
        sg.popup('car could not connect to network.')
if event == 'ConnectNo':  # Decline network connection
    sg.popup('Connection to network cancelled')
    logfile.write("[" + datetime.now().strftime("%H:%M:%S") +
                  "] Connection to network cancelled.\n")

if event == 'Security':
    if Sec == True:  #Turn FNAF off
        Sec = False
        logfile.write("[" +     datetime.now().strftime("%H:%M:%S") +
                      "] Fortified Network Assurance Field deactivated.\n")
        active_features.remove('FNAF')
        window['-features-'].update(getActiveFeaturesString())
    else:  #Turn FNAF on
        Sec = True
        logfile.write("[" + datetime.now().strftime("%H:%M:%S") +
                      "] Fortified Network Assurance Field activated.\n")
        active_features.append('FNAF')
        window['-features-'].update(getActiveFeaturesString())
        securityTime(window)

        if sus_data:
            if minutes == timer:
                active_features.append('Wowee')



if event == 'Software':
    logfile.write("[" + datetime.now().strftime("%H:%M:%S") +
```

```
                "] Software Settings opened.\n")
    window['-features-'].update(getActiveFeaturesString())
    software_popup(window)
if event == "SoftwareYes":
    for i in range(1, 10000):
        if not sg.one_line_progress_meter('Software Update', i + 1, 10000,
                                          'Software update in progress'):
            break
if event == "SoftwareNo":
    sg.popup('Software Update Cancelled')


if event == 'Parking':
    speed = vectorPark4[0]
    if Park == True: #Turn Park off
        Park = False
        logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Automated Parking deactivated.\n")
        active_features.remove('Parking')
        window['-features-'].update(getActiveFeaturesString())
    #check speed
    if(speed>10 or speed<0):
        logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Parking failed due to invalid speed. \n")
    else: #turn Park on
        Park = True
        is_parking_path_clear(vectorPark4)
        logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Automated Parking activated.\n")
        active_features.append('Parking')
        window['-speed-'].update(getSpeed())
        window['-features-'].update(getActiveFeaturesString())
if event == 'Technician\'s Interface':
    logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Technician\'s password prompted.\n")
```

```
        technician_popup(window)
        if technician_password_entry=="13":
          logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Technician\'s correct password entered.\n")
          logfile.close()
          log = open("./logs/ALSETLogData.log","r")
          print(log.read())
          logfile = open("./logs/ALSETLogData.log","w")
        else:
            logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Technician\'s wrong password entered.\n")

logfile.close()
window.close()
```

Proximity.py

```
view_size_sm = 2
view_size_lg = 3

# Simple check if two rectangles intersect
# Rectangles are a 4 tuple of (x, y, width, height)
def intersects(rect_a, rect_b):
  # separate data
  a_x, a_y, a_width, a_height = rect_a
  b_x, b_y, b_width, b_height = rect_b
  # verify intersections
  x_intersection = (a_x + a_width > b_x) and (b_x + b_width > a_x)
  y_intersection = (a_y + a_height > b_y) and (b_y + b_height > a_y)
  # return result
  return x_intersection and y_intersection
```

```
# Will return a list of the sensors that detect an obstacle
# ex: ["Front", "Right"]
def get_car_obstructions(car_bounds, obstacles):
  # local variables
  car_x, car_y, car_width, car_height = car_bounds
  car_center_x = car_x + car_width / 2
  car_center_y = car_y + car_height / 2
  # bounds calculation -- downwards y is positive
  front_bounds = (car_center_x - view_size_lg / 2,
                  car_y - view_size_sm,
                  view_size_lg,
                  view_size_sm)
  back_bounds = (car_center_x - view_size_lg / 2,
                  car_y + car_height,
                  view_size_lg,
                  view_size_sm)
  right_bounds = (car_x + car_width,
                  car_center_y - view_size_lg / 2,
                  view_size_sm,
                  view_size_lg)
  left_bounds = (car_x - view_size_sm,
                  car_center_y - view_size_lg / 2,
                  view_size_sm,
                  view_size_lg)
  # prepare result array
  result = []

  # check intersections
  for obs in obstacles:
    if intersects(front_bounds, obs):
```

```
      result.append("Front")
    if intersects(back_bounds, obs):
      result.append("Back")
    if intersects(left_bounds, obs):
      result.append("Left")
    if intersects(right_bounds, obs):
      result.append("Right")

  return result

# Returns a string interpretation of a list of the obstacles
def obstructions_to_str(obstr_list):
  if (len(obstr_list) == 0):
    return "No obstructions around the car!"
  else:
    return str(obstr_list)

# returns if the path to the goal parking spot is clear
# assuming car has been sufficiently lined up to the goal spot
def is_parking_path_clear(car_bounds, goal_bounds, obstacles):
  car_x, car_y, car_width, car_height = car_bounds
  goal_x, goal_y, goal_width, goal_height = goal_bounds
  # calculate collective bounds
  min_x = min(car_x, goal_x)
  max_x = max(car_x + car_width, goal_x + goal_width)
  min_y = min(car_y, goal_y)
  max_y = max(car_y, goal_y)
  total_width = abs(max_x - min_x)
  total_height = abs(max_y - min_y)
  collective_bounds = (min_x, max_y, total_width, total_height)
```

```
 # check collisions for each obstacle --then return result
 for obs in obstacles:
  if intersects(collective_bounds, obs):
   return False
 return True
```

ADAS.py

```
import PySimpleGUI as sg
from proximity import *
import os
from datetime import datetime


def ADAS(v):
    if not os.path.isdir("./logs"):
        try:
            os.mkdir("./logs")
        except OSError:
            print("Could not create logs directory")
        else:
            print("Created logs directory")


    logfile = open("./logs/ALSETLogData.log", "a")
    if os.path.getsize("./logs/ALSETLogData.log")>0:
        logfile.write("/n")


    c = v[0] #car on
    s = v[1] #speed
    pos = v[2] #car_bounds
    obstacles = v[3]#obstacles
    obs = v[4] #obstr_data
```

```
if(c == False):
    logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] ADAS Failed to turn on. \n")
    logfile.close()
    return "Car must be on"
if("Front" in obs):
    #decrease speed
    s = s-1
    #recalculate car position
    y = car_bounds[1]
    y = y - 1
    pos = (car_bounds[0],y,car_bounds[2],car_bounds[3])
    obstr_data = get_car_obstructions(pos,obs)
    #log
    logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Speed reduced.\n")
    #recheck if obstruction is there
    return ADAS([c,s,pos,obstacles, obstr_data])
if("Back" in obs):
    #increase speed
    s = s+1
    #recalculate car position
    y = pos[1]
    y = y + 1
    pos = (pos[0],y,pos[2],pos[3])
    #log
    logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Speed Increased.\n")
    #recheck if obstruction is there
    obstr_data = get_car_obstructions(pos,obstacles)
    return ADAS([c,s,pos,obstacles, obstr_data])
if("Left" in obs):
    #recalculate car position
```

```
        x = pos[0]
        x = x + 10
        pos = (x,pos[1],pos[2],pos[3])
        #log
        logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Moved car right.\n")
        #recheck if obstruction is there
        obstr_data = get_car_obstructions(pos,obstacles)
        return ADAS([c,s,pos,obstacles, obstr_data])
    if("Right" in obs):
        #recalculate car position
        x = pos[0]
        x = x - 10
        pos = (x,pos[1],pos[2],pos[3])
        #log
        logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Moved car left.\n")
        #recheck if obstruction is there
        obstr_data = get_car_obstructions(pos,obstacles)
        return ADAS([c,s,pos,obstacles, obstr_data])
    else:
        logfile.close()
        return [s, pos]
```

Proximity.py

```
import os
from datetime import datetime


if not os.path.isdir("./logs"):
    try:
```

```
        os.mkdir("./logs")
    except OSError:
        print("Could not create logs directory")
    else:
        print("Created logs directory")

logfile = open("./logs/ALSETLogData.log","a")

if os.path.getsize("./logs/ALSETLogData.log")>0:
    logfile.write("\n")

view_size_sm = 2
view_size_lg = 3

# Simple check if two rectangles intersect
# Rectangles are a 4 tuple of (x, y, width, height)
def intersects(rect_a, rect_b):
  # separate data
  a_x, a_y, a_width, a_height = rect_a
  b_x, b_y, b_width, b_height = rect_b
  # verify intersections
  x_intersection = (a_x + a_width > b_x) and (b_x + b_width > a_x)
  y_intersection = (a_y + a_height > b_y) and (b_y + b_height > a_y)
  # return result
```

```
  return x_intersection and y_intersection



# Will return a list of the sensors that detect an obstacle
# ex: ["Front", "Right"]
def get_car_obstructions(car_bounds, obstacles):
  # local variables
  car_x, car_y, car_width, car_height = car_bounds
  car_center_x = car_x + car_width / 2
  car_center_y = car_y + car_height / 2
  # bounds calculation -- downwards y is positive
  front_bounds = (car_center_x - view_size_lg / 2,
                  car_y - view_size_sm,
                  view_size_lg,
                  view_size_sm)
  back_bounds = (car_center_x - view_size_lg / 2,
                  car_y + car_height,
                  view_size_lg,
                  view_size_sm)
  right_bounds = (car_x + car_width,
                  car_center_y - view_size_lg / 2,
                  view_size_sm,
                  view_size_lg)
  left_bounds = (car_x - view_size_sm,
```

```
            car_center_y - view_size_lg / 2,
            view_size_sm,
            view_size_lg)
  # prepare result array
  result = []

  # check intersections
  for obs in obstacles:
    if intersects(front_bounds, obs):
      result.append("Front")
    if intersects(back_bounds, obs):
      result.append("Back")
    if intersects(left_bounds, obs):
      result.append("Left")
    if intersects(right_bounds, obs):
      result.append("Right")

  return result



# Returns a string interpretation of a list of the obstacles
def obstructions_to_str(obstr_list):
  if (len(obstr_list) == 0):
    return "No obstructions around the car!"
```

```
  else:
    return str(obstr_list)


# returns if the path to the goal parking spot is clear
# assuming car has been sufficiently lined up to the goal spot
def is_parking_path_clear(v):
  speed = v[0]
  car_bounds = v[1]
  goal_bounds = v[2]
  obstacles = v[3]
  car_x, car_y, car_width, car_height = car_bounds
  goal_x, goal_y, goal_width, goal_height = goal_bounds
  #check speed
  if(speed>10 or speed<0):
    logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Parking failed due to invalid speed. \n")
    logfile.close()
    return "Invalid Speed"
  # calculate collective bounds
  min_x = min(car_x, goal_x)
  max_x = max(car_x + car_width, goal_x + goal_width)
  min_y = min(car_y, goal_y)
  max_y = max(car_y, goal_y)
  total_width = abs(max_x - min_x)
  total_height = abs(max_y - min_y)
```

```
 collective_bounds = (min_x, max_y, total_width, total_height)

 # check collisions for each obstacle --then return result

 for obs in obstacles:

  if intersects(collective_bounds, obs):

   return False

 return True
```

CruiseControl.py

```
#keeps car at constant speed, does no accelerate or decelerate over time
def CC(v):
    car_on = v[0] #is the car on
    car_bounds = v[1] #where is the car
    speed = v[2] #Car's speed
    if(car_on == False):
        logfile.write("["+datetime.now().strftime("%H:%M:%S")+ "] Cruise Control Failed to turn on. \n")
        logfile.close()
        return "Car must be on"
    else:
        return speed
```

Test.py

```
from proximity import *
from ADAS import *
from CruiseControl import *

#######################################
# Proximity tests
```

```
# Test 1
car_bounds = (3, 5, 2, 3)
obstacles = [(6, 6, 2, 2)]
obstr_data = get_car_obstructions(car_bounds, obstacles)
print(obstructions_to_str(obstr_data))

#######################################
# Parking tests

# Test 1
speed = 5
car_bounds = (4, 2, 3, 2)
goal_bounds = (4, 7, 3, 2)
obstacles = [(1, 7, 3, 2), (7, 7, 3, 2)]
vectorPark1 = [speed,car_bounds,goal_bounds,obstacles]
path_data = is_parking_path_clear(vectorPark1)
print(f'Parking path is clear: {path_data}')

# Test 2
speed = 9
car_bounds = (4, 2, 3, 2)
goal_bounds = (4, 7, 3, 2)
obstacles = [(1, 7, 3, 2), (6, 7, 3, 2)]
vectorPark2 = [speed,car_bounds,goal_bounds,obstacles]
path_data = is_parking_path_clear(vectorPark2)
print(f'Parking path is clear: {path_data}')

# Test 3
speed = 5
```

```
car_bounds = (4, 2, 3, 2)
goal_bounds = (4, 7, 3, 2)
obstacles = [(1, 7, 3, 2), (1, 4, 3, 2)]
vectorPark3 = [speed,car_bounds,goal_bounds,obstacles]
path_data = is_parking_path_clear(vectorPark3)
print(f'Parking path is clear: {path_data}')

#Test 4
speed = 15
car_bounds = (4, 2, 3, 2)
goal_bounds = (4, 7, 3, 2)
obstacles = [(1, 7, 3, 2), (1, 4, 3, 2)]
vectorPark3 = [speed,car_bounds,goal_bounds,obstacles]
path_data = is_parking_path_clear(vectorPark3)
print(f'Parking path is clear: {path_data}')



##########################################
#ADAS tests

#Test 1
car_on = True
car_bounds = (3, 5, 2, 3)
speed = 40
obstacles = [(6, 6, 2, 2)]
obstr_dataADAS1 = get_car_obstructions(car_bounds, obstacles)
vectorADAS1 = [car_on,speed,car_bounds,obstacles,obstr_data]
spd_dir_data = ADAS(vectorADAS1)
print(f'New speed and position are : {spd_dir_data}')
```

```
#Test 2
car_on = True
car_bounds = (3, 5, 2, 3)
speed = 40
obstacles = [(4, 7, 3, 2)]
obstr_data = get_car_obstructions(car_bounds, obstacles)
vectorADAS2 = [car_on,speed,car_bounds,obstacles,obstr_data]
spd_dir_data = ADAS(vectorADAS2)
print(f'New speed and position are : {spd_dir_data}')

##########################################
# Network Test
network_list = ['PizzaPlex_Wifi', 'Exotic_Butters', 'FBI_Surveilance_Van', 'Peepaw_Aftons_WheelChair']
isConnected = True

##########################################
#Cruise Control Tests
car_on = True
car_bounds = (4,2,3,2)
speed = 35
vectorCC1 = [car_on, car_bounds, speed]
cruise = CC(vectorCC1)
print(f'Set speed is {cruise}')
```

Section 7

Testing

Req. 1.1 Cruise Control

Test 1: Turning on Cruise Control

Req 1.2 Parking Assistance

Test 1: All sensor values except set speed to 5 mph, set car bounds to (4,2,3,2), goal bounds are (4,7,3,2), and obstacles are [(1,7,3,2),(7,7,3,2)]

Input = [speed, car_bounds, goal_bounds, obstacles]

Expected Output: ‘Parking path is clear: True’

Preconditions:

- Car is on & in drive
- Speed < 10 mph

Test 2: Set speed to 9 mph, car bounds are (4,2,3,2), goal bounds set to (4,7,3,2), obstacles are [(1, 7, 3, 2), (6, 7, 3, 2)]

Input = [speed, car_bounds, goal_bounds, obstacles]

Expected Output: ‘Parking path is clear: False’

Preconditions:

- Car is on & in drive
- Speed < 10 mph

Test 3: Set speed to 5 mph, car bounds are (4,2,3,2), goal bounds set to (4,7,3,2), obstacles are [(1, 7, 3, 2), (1, 4, 3, 2)]

Input = [speed, car_bounds, goal_bounds, obstacles]

Expected Output: ‘Parking path is clear: True’

Preconditions:

- Car is on & in drive
- Speed < 10 mph

Test 4: Set speed to 15 mph, car bounds are (4,2,3,2), goal bounds set to (4,7,3,2), obstacles are [(1, 7, 3, 2), (1, 4, 3, 2)]

Input = [speed, car_bounds, goal_bounds, obstacles]

Expected Output: ‘Parking path is clear: Invalid speed’

Preconditions:

- Car is on & in drive
- Speed < 10 mph

Req 1.3 Advanced Driver Assistance System

Test 1: The car is on, set speed to 40 mph, car bounds are (3, 5, 2, 3), obstacles are [(6, 6, 2, 2)], and obstruction data is from get_car_obstructions(car_bounds, obstacles)

Input = [car_on, speed, car_bounds,obstacles,obstr_data]

Expected Output: New speed and position are

Preconditions:

- The car is on and in drive
- Adjusts cars course and speed if needed

Req 2.1.3.1 Technician Access

Test 1: Successful log-in

Input: User selects Technician Feature, and enters correct password

Expected output: the log file printed to terminal

Precondition: the car is on

Req 5 Software Update

Test 1: Successfully Update Software

Input: User selects Software feature, selects yes when prompted

Expected Output: the software is updated

Precondition: The car is on

Test 2: Decline Update

Input: User selects Software features, selects ‘no’ when told an update is available

Expected Output: the pop-up window is closed and software is not updated

Precondition: The car is on

Test 3: Cancel Update

Input: User selects Software feature, selects ‘yes’ when told an update is available, while system is updating user hits the cancel button

Expected Output: The software is not updated

Precondition: The car is on

Req 6.1 Network Configuration

Test 1: Successfully Connect to WIFI

Input: User opens Network Settings and let the system load all available networks, the user chooses a network to connect to

Expected Output: The car tries to connect to network and successfully does so

Precondition: The car is on, only show networks the user can connect to

Test 2: Cancel Connection to Network

Input: User opens network settings, once available networks have been loaded hit cancel

Expected Output: Pop-up displays that network connection has been canceled

Precondition: The car is on

Req 7 Logging

Test 1: Log the above tests

Input: Run the previous tests

Expected Output:

[20:52:20] System Ready.

[20:52:22] Cruise Control activated.

[20:52:23] Cruise Control deactivated.

[20:52:24] Advanced Driver Assistance System activated.

[20:52:25] Advanced Driver Assistance System deactivated.

[20:52:27] Automated Parking activated.

[20:52:28] Automated Parking deactivated.

[20:52:33] Fortified Network Assurance Field activated.

[20:52:33] Fortified Network Assurance Field deactivated.

[20:52:35] Network Settings opened.

[20:52:43] Software Settings opened.

[20:54:18] Technician's password prompted.

[20:54:21] Technician's correct password entered.

Precondition: The car is on, the system successfully creates or appends to log file