Alset:
Hugs The Lanes

Team Frou Frou

Team Members: Trevor Mckay, Rosa Taveras, Frank Savattere, Rayyan Zafar

Current Team Lead: Frank Savattere

Version 1.7 Last updated 5/17/2022

1.1 Introduction

Here at Alset, safety is the most important factor we consider when we think about designing our vehicles. In America, 40% of car accident related fatalities are attributed to driving under the influence, resulting in the deaths of over 10,000 Americans annually. This is why one of the most important features of the newest Alset model is a built-in breathalyzer to detect any intoxicating substances before a driver is able to get behind the wheel. This feature includes a small, hands-free device that comes up to your mouth when you enter the vehicle that you can talk into while it reads your intoxication levels. Once the device reads your intoxication levels, it presents the data received onto the user interface in the center console and starts the vehicle if it determines 0 levels of intoxication. This feature makes it so that every Alset driver is guaranteed to be sober and therefore decrease the likelihood of endangering themselves or others on the road, and potentially save countless lives.

Our product aims to replace the current ignition interlock devices that drivers who have received DUI's are court mandated to install in their car. Although these devices help to prevent drunk driving they force individuals to take their hand off the steering wheel in order to blow into the device. We found that the ignition interlock devices fall into a gray area in regards to some states' distracted driving laws. For instance, New Jersey outlaws the use of handheld devices while driving. This begs the question – why are governing bodies allowed to mandate certain drivers to break these restrictions? Our new technology promises to combat this issue by keeping drivers' hands on the steering wheel and less distracted so that nobody, regardless of their previous driving record, is forced to distract themselves on the road and put others in danger.

This product will have a direct, positive impact on the dangers of driving under the influence, by acting as a deterrent.This is important to our group as we are students and residents in Hoboken, New Jersey. Hoboken is a very small city with a highly saturated population, with many narrow roads and many crosswalks. This means that the roads are very dangerous to begin with. Combine that with all of the bars and taverns Hoboken provides, and people could get hurt. To prevent this, and to help cities with similar circumstances, and all places, the built-in breathalyzer is a necessity. This will make dangerous situations safer, and will ensure the safety of others, as this will act as a massive deterrent to driving under the influence.

Other than the breathalyzer, we will include several other features that will help make the car safer as well as more autonomous. One of the features includes improved sensor technology for detecting objects in the vehicle's blindspot. Software that alerts the driver of objects they might not be able to see without distracting them from the road. Another feature is autonomous parking which with the help of the improved sensor technology, can be used even in crowded parking lots or streets, which is common here in a city like Hoboken. If, unfortunately, an accident cannot be avoided and the vehicle sustains a major crash, software will also be implemented so that authorities and other first responders are immediately alerted if airbags are

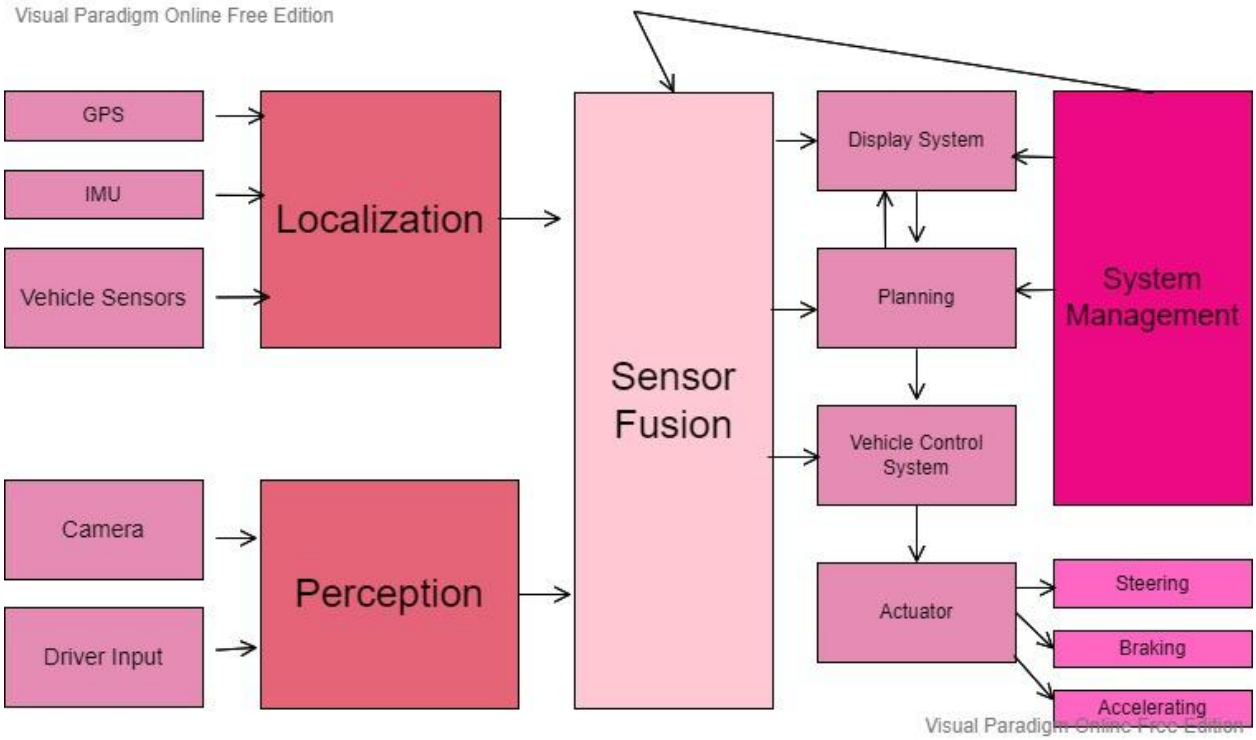
deployed. This will be done via our improved car assistant, similar to Siri on your iPhone, you will be able to access many features of the car by speaking to the driver's assistant, and this is the same assistant that will auto dial 911 and alert responders of the emergency. Sometimes people may have a hard time getting in and out of a vehicle, especially when the seat has to be near the steering wheel, so to address this problem, the Alset model will include software that turns the driver's chair towards the door when it opens, to make it easier to get in and out of the vehicle, and also automatically adjusts the seat to your preferred setting when the car starts to make it easier to drive.

Auto park is a very nice feature that will give quality parking to the driver that takes advantage of it. This feature allows the car to detect its surroundings with sensors, and then determine the correct angle to park in. For instance, if the driver wanted to pull into a standard parking spot in a parking lot, the auto park feature will allow the driver to do so. Likewise, if the driver wanted to parallel park, the car will be able to parallel park, whether the car must fit in between two cars, or not the sensors will be capable of picking up that information and the car will be able to determine the correct way to park, and will only fault when there is not enough physical space for the car to fit.

We intend to complete this project in 14 weeks. Because of this short time frame we will be using the Kanban Framework to quickly roll out our product. The Kanban Framework allows us to easily visualize our workflow, limit the amount of work in progress at any given time, and have a clear endpoint of the project. The framework depends heavily on collaboration and planning which fits our group's preferences. Because our product is mission critical, we must have fully developed features in order to release our technology. This also fits our chosen framework because the Kanban Framework does not allow unfinished tasks to be put off until later but rather encourages us to completely finish one feature before moving onto the next feature.

Our team works well together because as individuals we all offer different strengths that when put together, creates a wonderful dynamic that allows us to work efficiently and collaboratively.

1.2 Functional Architecture



As previously mentioned, the breathalyzer will take in the driver's breath as input into the device which will read the levels and report them back onto the screen of the vehicle, as well as send the signal to start the vehicle if all levels are well. As for the captain's rotating chairs, they will be triggered by the opening of the door which sends a signal to the car that the chairs must turn over towards the doors and once the person is seated, which can be detected by the weight on the chair, similar to how seatbelt alert works, the chair will turn when enough weight is detected on it, and adjust to the driver's preferred position. The output of this feature can be considered to be the movement of the chair after the input is received which would be either the opening of the door or the weight of the driver. Blindspot detection is something that has been around in other vehicles but we plan to make sure that the sensors are even more sensitive and aware so that signals may detect an object in a blindspot much faster and send the signal to the driver of the alert as output. You might ask how the car assistant is meant to work. Well, our car assistant is similar to AI technology that can take a vast amount of input, process this information and return a vast amount of output as well, whether that be in the form of voice communication, or sending signals to change the radio station. The car assistant feature goes hand in hand with our voice ID system because it is meant to replace buttons on the dashboard that might distract a driver if they are trying to change the radio station, for example, while driving. This way, the driver's voice is used as the input and the data is analyzed and sent to the car assistant which will perform the output. Similarly, in the event of an emergency, information

sent from the car's sensors would allow the car assistant to detect severe impact and just like it can send signals to change the radio, it would respond by sending a signal to 911 with the location of the accident, dispatching first responders. Our vehicle also contains a Touch ID system to unlock the door in which the input would be the driver's fingerprint onto the fingerprint sensor and output would then be the car doors unlock if the fingerprint matches one saved in the vehicle's system, and denying the driver access by keeping the doors locked if it does not. Another feature of the vehicle is automatic parking, both parallel and into a space. As input, it would take in a signal that the car needs to park sent by the driver pushing the park button, as well as the measurements of the parking space in which the vehicle is trying to park in, given by the 360 cameras, to determine if the space is big enough to park in. The input for the 360 cameras is that the vehicle is turned on because they are always running in the background while the vehicle is running and their data is sent to the cloud. If the space is big enough, then the output would be the car automatically steering, accelerating, and braking accordingly to park into the parking spot. Finally, we have our vehicle summoning system which allows the driver to call the vehicle from its parking spot to their location when it is within a certain radius. The input is a signal sent by the driver from a button on the key remote as well as the driver's location sent by the key control as well. As output, after the vehicle has calculated a safe path to the driver, it will steer, accelerate, and brake accordingly to arrive at the driver's destination.

In a UML Case Diagram the car owner/driver in our vehicle would have direct access to the inputs in the case of the breathalyzer, as it is the device that reads the BAC of the driver. However, the driver would be able to view the output, but not interact with the data in any way. The only person with such permissions would be a system administrator, and this feature can not be disabled. Like the breathalyzer, the rotating chairs will be triggered by physical movement of the driver. The driver will initiate the input and the chairs will respond to the event of the car door opening. The blindspot detection generates its own input using information that the sensors gather. The sensors will gather information and then alert the driver in a response to detecting an object. As these will be on the outside of the car, the driver does very little to interact with them. The car assistant is accessed by the driver in the car directly, the assistant will take in information given by the driver, and provide some output. The output the driver receives is influenced by the inputs given by the driver, hence the driver has an influence on both input and output. Other features like the severe collision detection and the 360 night view cameras take no input from the driver while the vehicle summoning and automatic parking require direct input from the driver in order to be actuated.

1.3 Requirements

- 3.1 Functional Requirements
    - 3.1.1 Breathalyzer
        - Must be hands-free
        - Accepts multiple phrases from the driver
        - Uses specific driver ID to tell which drivers require the use of the device while driving
        - Must connect to the ignition in order to control the vehicle's drive state based on the results
        - Preconditions
            - The engine must be off
            - The car's battery is supplying power
            - Breathalyzer must be usable on just the car's battery so that if the driver fails the breathalyzer test, they will not be able to start the car
        - Postconditions
            - Must be able to allow or deny access to the car's transmission
    - 3.1.2 Touch ID System
        - Correctly identifies if the fingerprint is valid
        - Accepts and stores up to 15 individual scans
        - Is able to be disabled within the UI
        - Built into both the inside and outside of the car so that it could be used as a key replacement
        - Connects to the Breathalyzer to authenticate the driver's identity
        - Preconditions
            - The car's battery is supplying power
        - Postconditions
            - If the fingerprint was valid, logs the time the vehicle was unlocked and who is unlocking it.
            - If it was invalid, resets to try again.
    - 3.1.3 Virtual Assistant
        - Accepts all major languages
        - Adequately responds to driver's requests as needed
        - Able to make phone calls
        - Wireless and Wired connection to a driver's personal cell phone
        - Connection to major mobile apps
        - Able to contact necessary authorities in the case of an emergency
        - Preconditions
            - The car's battery is on and supplying power
            - Bluetooth connection is active
        - Postconditions
            - Remains on standby when not in use

- 3.1.4 Severe Collision System
  - Detects significant unusual forces
  - Deploy airbags
  - Can be disabled by the driver
  - Alerts necessary authorities if the collision force threshold is met
  - Authorities can be notified via a button press
  - Preconditions
    - The vehicle is turned on
    - The vehicle's 'black box' is intact
    - Collision force threshold is surpassed
  - Postconditions
    - Constantly logging vehicle's location and damages and sending it to necessary first responders
- 3.1.5 Voice ID System
  - Recognizes all major languages
  - Uses a set of sample phrases to learn a driver's voice
  - Connected to Virtual Assistant
  - Though the Virtual Assistant connection can control certain actions within the car
  - Connected to the Breathalyzer
  - Preconditions
    - The vehicle is turned on
    - Driver's voice is a saved profile
      - If not – tell the driver that if they would like to use the features provided by the Voice ID system, they must create a voice profile.
  - Postconditions
    - Logs action performed for the driver.
- 3.1.6 Captain's chairs
  - Swivel out to allow for easy entry into the vehicle
  - Implemented into the front two seats
  - Can be controlled via a button on the side of the seat
  - Preconditions
    - The car is unlocked
    - The car is in park
    - The door is open wide enough for the chair to turn
    - The chair's path does not have obstacles
    - The person in the seat is sitting still
  - Postconditions
    - The seat will automatically return to its resting position after the person exits the vehicle
    - The car door will not close until the chair is returned to its resting position

- 3.1.7 Blindspot lane detection
  - Signals when Driver is crossing lanes without a blinker
  - If enabled, the car steering will become stiffer when approaching the adjacent lane
  - Light on the side view mirror turns on if there is an object in the vehicle's blindspot.
  - Preconditions:
    - Car must be turned on
    - Must be driving on road with clear lanes for the sensors to detect.
  - Postconditions:
    - Car is returned to the middle of the lane
- 3.1.8 Auto park
  - Preconditions:
    - The car is not moving
    - Space must be big enough for car to park in
  - Will park the car using the sensors and cameras to avoid collision
  - Postconditions:
    - The car will shut off after park is successful
- 3.1.9 Vehicle Summoning
  - Preconditions:
    - You are within a mile of your car
    - You use the summon feature in the smartphone companion app.
  - Car drives itself to the driver when signaled from the companion smartphone app, as long as the owner is within a mile radius.
  - Postconditions:
    - The car can then remain started, or turn off automatically, depending on which option is selected through the app.
- 3.1.10 Night View 360 cameras
  - Preconditions:
    - Car is on
  - Always records when the car is on and logs the data in case it ever needs to be used after an accident. Can be disabled by driver.
  - Shows on the display screen when car is in reverse or parking
  - Postconditions:
    - Car is turned off

3.2 Nonfunctional requirements

- User Interface:
  - 3.2.1 Performance Requirements
    - 3.2.1.1 Breathalyzer
      - Must be accurate within .0001 BAC
      - Must post results to User Interface within 250 milliseconds
    - 3.2.1.2 Touch ID System
      - Identifies fingerprint within 250 milliseconds.

- Success rate of fingerprint identification of at least 99.999%
- 3.2.1.3 Virtual Assistant
  - Correctly interprets commands with a success rate of 99.9%
- 3.2.1.4 Severe Collision System
  - Successfully determines whether or not a significant change in force was a collision with a success rate of 99.999%
  - The system successfully withstands 99.999% of collisions
  - Alerts the correct authorities in the event of a collision
  - Sends coordinates to authorities accurate within 100 feet
- 3.2.1.5 Voice ID System
  - Correctly identify voice by their chosen key phrases
- 3.2.1.6 Captain's Chairs
  - Rotates at a rate of .2 meters per second
  - Will not rotate unless the car is in park
  - Will stop turning if a parallel force that is opposite to the chair's movement exceeds three times the static friction force.
- 3.2.1.7 Lane Detection
  - Alerts driver of objects in blind spot
  - Steers to return the car to the center of the lane if it veers off accidentally
  - Alerts the driver if the car is trying to switch lanes without signaling
- 3.2.1.8 Auto Parking
  - Will not work unless car is initially stopped
  - Successfully determines if the parking space is big enough to initiate parking
- 3.2.1.9 Vehicle Summoning
  - Car makes it to the driver with 99.99% success rate, while avoiding cars, pedestrians, and any other obstacles.
- 3.2.1.10 Night View 360 cameras
  - Runs the entire time the car is running in the background

- 3.2.2 Reliability Requirements
  - 3.2.2.1 Breathalyzer
    - In the event where it cannot read BAC properly, display a message on the screen to contact a technician and the car remains off.
  - 3.2.2.2 Touch ID system
    - If the fingerprint for the profile that is trying to be accessed does not match after a total of 5 tries, this profile becomes disabled for 5 minutes, and an additional 5 minutes for every additional invalid entry after it is initially disabled.
  - 3.2.2.3 Virtual Assistant
    - Confirms that each command is correctly interpreted before executing it, logs the amount of the time the response is "no". Does not perform the task if the response is no.

        - 3.2.2.4 Severe Collision System
            - If the vehicle sustains a substantial change in force, but the virtual assistant is unable to alert authorities, a message is displayed to the driver to alert the authorities if an accident has occurred.
            - If a significant change in force is detected, the virtual assistant confirms with the driver which authorities need to be notified before alerting them.
        - 3.2.2.5 Voice ID System
            - If the voice for the profile that is trying to be accessed does not match after a total of 5 tries, this profile becomes disabled for 5 minutes, and an additional 5 minutes for every additional invalid entry after it is initially disabled.
        - 3.2.2.6 Captain's Chairs
            - If the turn rate falls below .2 meters per second, this feature becomes disabled and a message is displayed to the driver to contact a technician.
        - 3.2.2.7 Lane Detection
            - If the sensors become "stuck" displaying that there is always something in the blindspot, then this feature is disabled and the driver is displayed a message to contact a technician.
        - 3.2.2.8 Auto Parking
            - If it cannot accurately determine the space allocated for parking, a message is displayed to the driver that auto parking cannot occur because the vehicle was unable to evaluate the parking space chosen.
        - 3.2.2.9 Vehicle Summoning
            - If the vehicle cannot determine a safe path to the driver's location then a message is displayed to the driver through the app that a safe path cannot be determined from the vehicle to the driver.
        - 3.2.2.10 Night View 360 Cameras
            - If any camera stops working, a message is displayed to the driver that a camera has gone offline and to contact a technician.
- 3.3 User Interface
    - Can be accessed via the Virtual Assistant
    - Is able to disable/enable any of our features. Will require a passcode if the feature is deemed to be critical to driver safety.
- 3.4 Hardware IOS
    - 3.4.1 Operating System
        - Real time operating system
- 3.5 Security
    - 3.5.1 Breathalyzer
        - Driver has direct access to the input
        - Driver can only view output

- 3.5.2 Touch ID System
  - Driver inputs fingerprint directly into the system
  - Information is to be encrypted to anyone other than a System Technician
  - Can be disabled
- 3.5.3 Virtual Assistant
  - Driver's commands are direct inputs
  - Can be disabled
- 3.5.4 Severe Collision System
  - Location is encrypted until the system is triggered
- 3.5.5 Voice ID System
  - Driver's commands are direct inputs
  - Can be disabled
- 3.5.6 Captain's Chair
  - Can be disabled
- 3.5.7 Lane Detection
  - No input from driver
  - Can be disabled
- 3.5.8 Auto Park
  - Direct input from driver
  - Can be disabled
- 3.5.9 Summoning
  - Direct input from driver
  - Can be disabled
- 3.5.10 Night View 360 Cameras
  - No input from driver
  - Can be disabled

3.6 Network

- 3.6.1 Physical Connection
  - Data is temporarily stored on a hard drive in the car until it is able to upload to the cloud. At that point, the data will be removed from the hard drive
  - The driver can elect to only use physical storage
  - The car will have an internet connection via a mobile hotspot installed into the car. This can operate via a paid cell service or via the connection from the driver's phone.
- 3.6.2 Cloud Connection
  - All logged data from our features will be saved in the cloud in order to have a record of crash collision force for instance
  - Data will be stored on a physical drive in the car until it has a stable connection to upload it to the cloud
  - Can be disabled in the User Interface. The data will then be stored on the physical hard drive until it becomes filled. The driver will then be required

to either clear more space on the car's physical drive or elect to stop recording data from driving.

- 3.7 Logging Requirements
    - 3.7.1 Car Start Time
        - When driver presses push to start button, the current time is captured and uploaded to the cloud
    - 3.7.2 Sensor Logging
        - All Sensor data such as the ones used for blindspot detection are time stamped and uploaded to the cloud as well, in case of any issues.
    - 3.7.3 Camera Footage
        - Footage is constantly being uploaded to the cloud as long as the car is on.
    - 3.7.4 Virtual Assistant
        - All commands given to the virtual assistant are logged as well as if the task was performed or not.

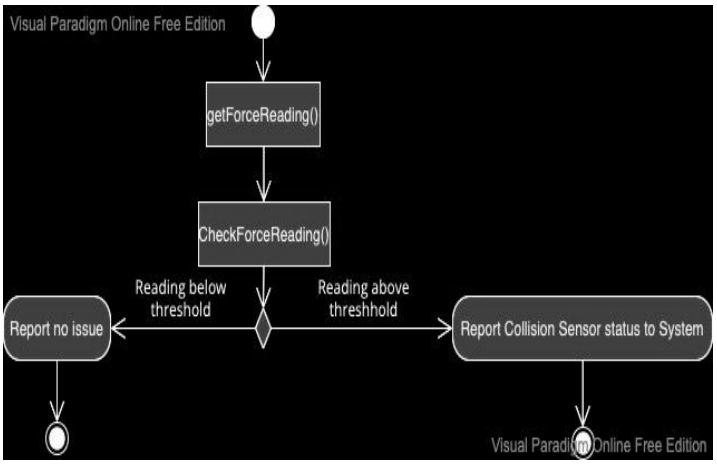
1.4 Requirements Modeling

1.4.1 Use Case Scenarios

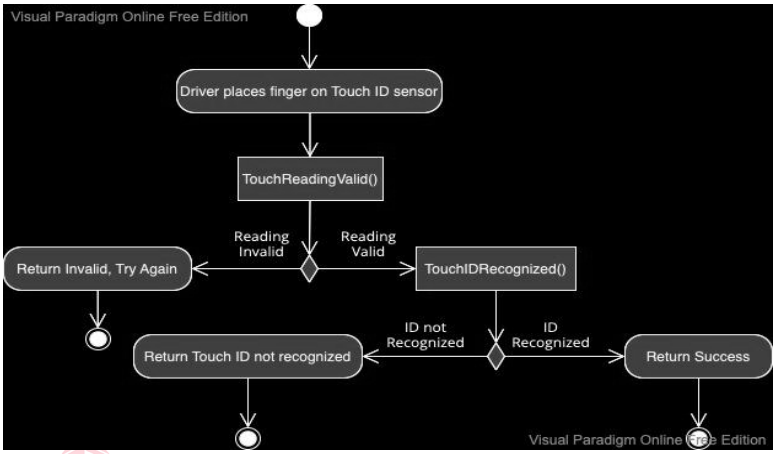
- **Severe Collision Detection**: The vehicle is on and Sensor Fusion detects that at some Sensor the external forces acting on it have exceeded the expected forces to act on it during normal driving. This force would be great enough to say with absolute certainty that the car has collided with some actor in its environment. This force limit is given by the system and when the threshold is surpassed this data is sent back through the system.
- **Virtual Assistant**: The driver needs to perform an action within the UI while driving without taking their hands off the wheel. The driver speaks, and the Sound Interpreter Sensors receive the command. The Sound Interpreter Sensors report this back to the system through Sensor Fusion. The system then deciphers the driver's command and reports back to the UI in any of the following cases:
    - Case 1: Error in interpretation.
    - Case 2: Requested function is not available
    - Case 3: Return requested function
- **Touch ID**: The driver needs to verify their ID in order to enter the vehicle. The driver places their finger on the Touch ID sensor which reads the input and sends the data to the System through Sensor Fusion. The system checks if the ID is valid and returns one of the following cases:
    - Case 1: Invalid ID, try again.
    - Case 2: Valid ID. Vehicle unlocked.
- **Voice ID:** The Driver needs to verify their ID via their voice to access the ignition through the Breathalyzer. The Sound Interpreter Sensors receive the input of the driver's voice through the Breathalyzer and then send the data to the System through Sensor Fusion. The System verifies the driver's voice through the preset profile and reports back to the Breathalyzer with one of the following cases:
    - Case 1: Voice ID invalid, car cannot be put into drive
    - Case 2: Voice ID valid, car can be put into drive.
- **Captain's Chair:** The Driver needs to enter or exit the vehicle. The driver may press the button that will begin the swiveling of the Captain's Chair. The Button sensor sends this input through Sensor Fusion to the system which begins the turning of the Captain's Chair.
- **Lane Detection**: The vehicle is on and Sensor Fusion detects an object within the car's blindspot via the Blindspot Sensors. The Driver is notified via a light indicator on the respective side view mirror as well as on the user interface. While the vehicle is on and in motion the Blindspot Sensors will report back to Sensor Fusion the following cases:
    - Case 1: Object detected by Blind Spot Sensors
    - Case 2: Object is not detected by Blind Spot Sensors
    - Case 3: Blind Spot Sensors cannot accurately detect an object's presence. Return Error.

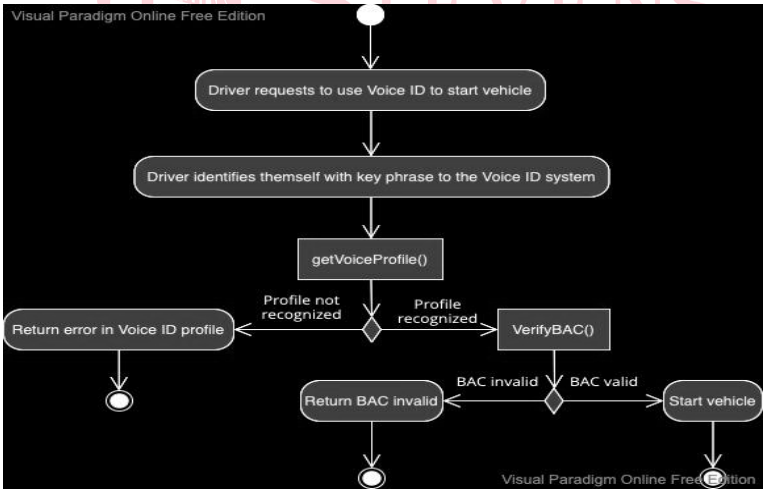
- **Summon**: The vehicle is remotely powered on via the key and in park. Sensor Fusion reports if there are any obstacles detected by the vehicle's sensors. The vehicle detects the summon destination based on the Driver's key's GPS coordinates. The vehicle begins it's summon process and the following cases may occur:
    - Case 1: Obstacle is detected by Summon Sensors. The vehicle stops until the obstacle is no longer detected.
    - Case 2: Summon Sensors do not detect any obstacles, vehicle summons
- **Auto Park**: The vehicle is on and the driver initiates the auto park sequence. Sensor Fusion scans the surrounding area of the vehicle to see if there is a valid space for the car to park. The following cases occur after this scan:
    - Case 1: No valid parking space detected. Return Error to Driver.
    - Case 2: Valid parking space detected. Begin parking sequence
- **Breathalyzer**: The Driver is in the vehicle with a valid key. The Breathalyzer is connected to the ignition. The User Interface alerts the Driver that they must use the Breathalyzer to turn on the vehicle. The Driver speaks into the Breathalyzer and the BAC sensor reports the driver's BAC to Sensor Fusion. At this time the following cases occur:
    - Case 1: Driver's detected BAC is above the legal limit. Vehicle will not start.
    - Case 2: Driver's BAC is below the legal limit. Vehicle is turned on.
    - Case 3: BAC sensor failed to detect valid BAC. Return Error.
- **Night view 360 cameras**: The vehicle is powered on and the Driver activates Night View 360 Cameras in the User Interface. Sensor Fusion detects this action and turns on the Nights View 360 Cameras and displays the 360 view on the User Interface. At this point the following cases may occur:
    - Case 1: Night View 360 Cameras are properly synced and display image on User Interface
    - Case 2: Error in Night View 360 Camera sync. Return Error.

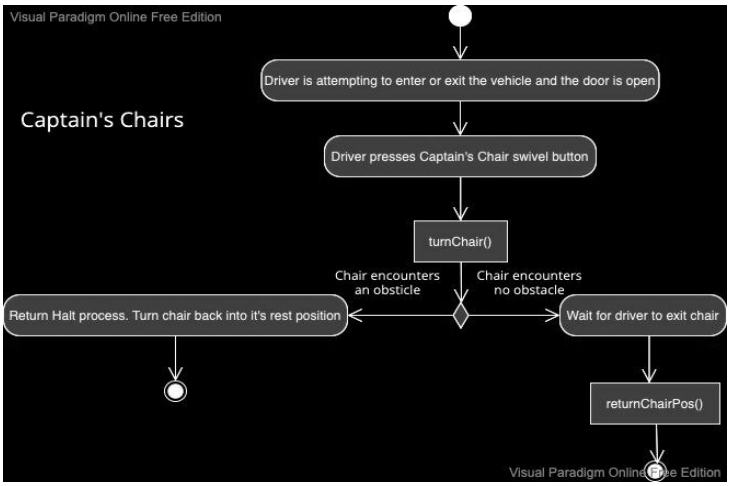
1.4.2 Activity Diagrams

- **Severe Collision Detection System**
    - 

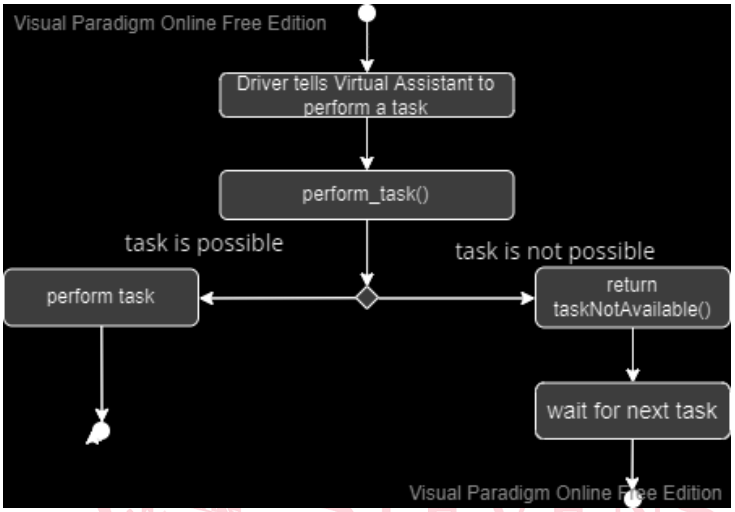
- **Touch ID**



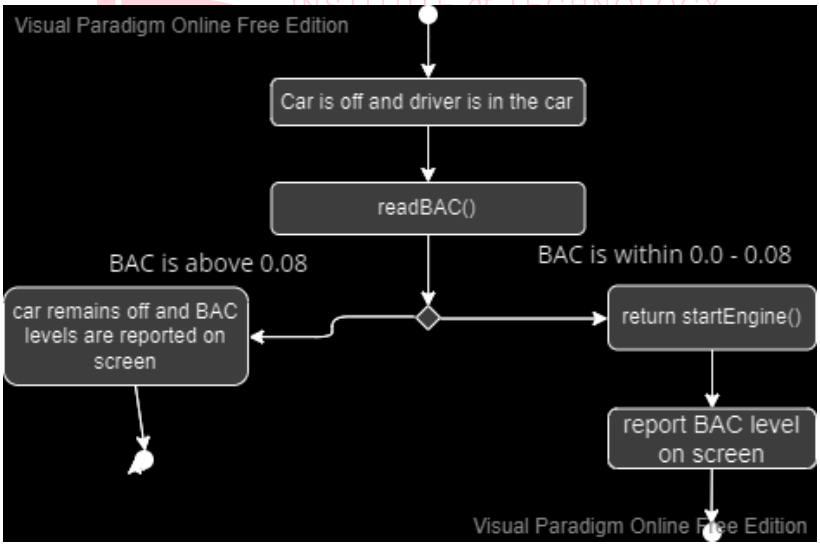
- **Voice ID**
  - 

- **Captain's Chairs**
  - 

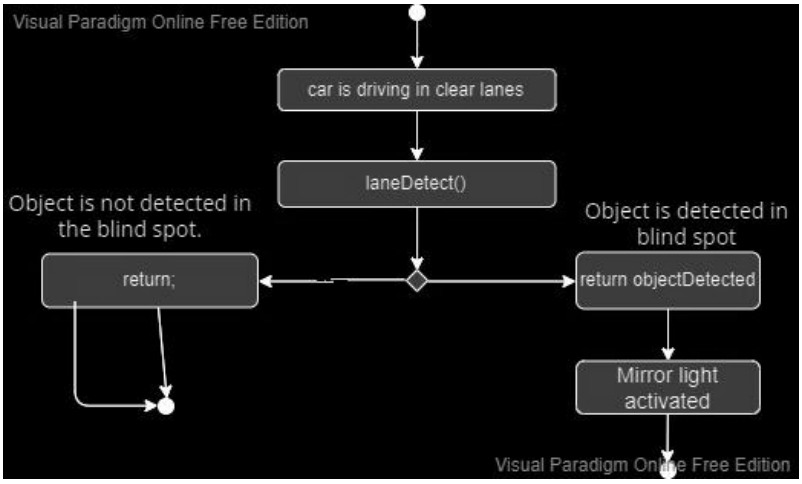
- **Virtual Assistant**



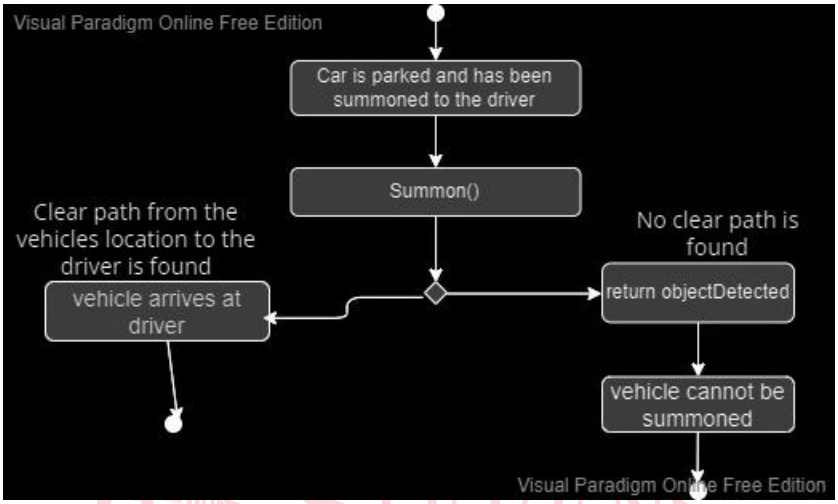
- **Breathalyzer**

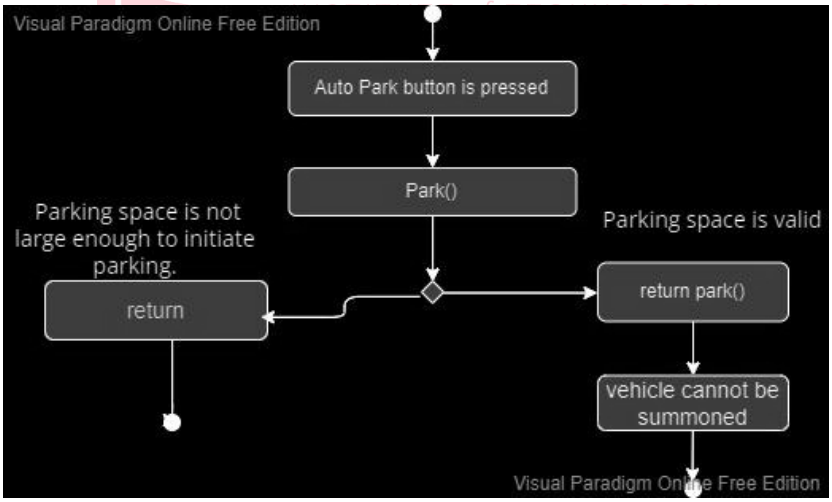


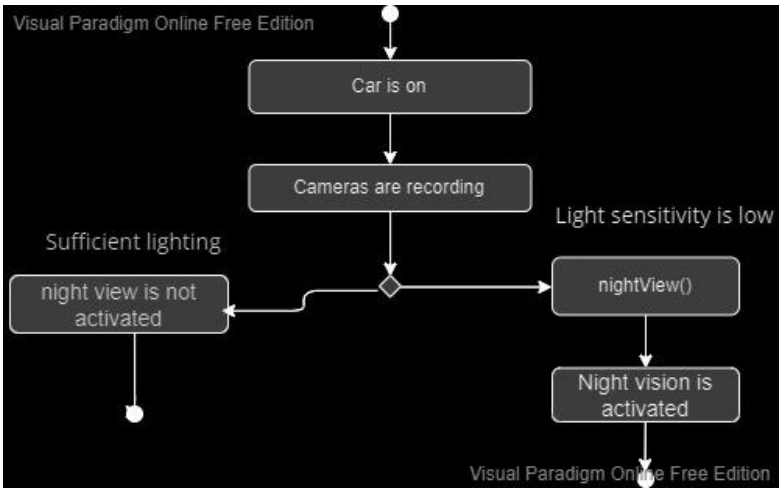
- **Lane Detection**



- **Vehicle Summoning**

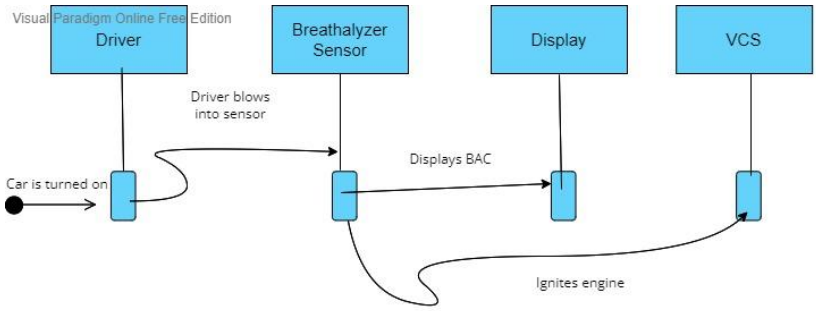
- 

- **Auto Park**
  - 

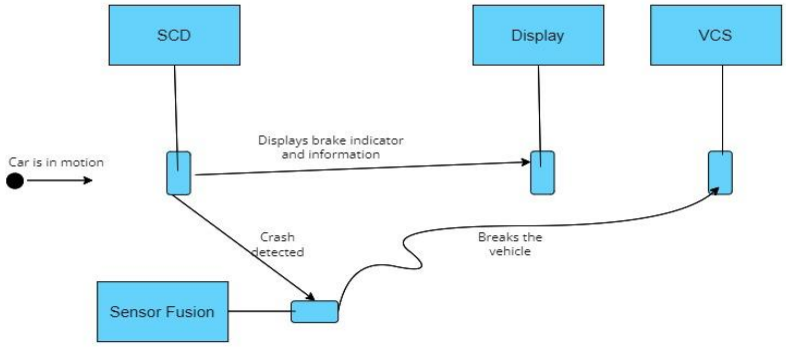
- **Night View 360 Cameras**
  - 

- 1.4.3 Sequence Diagrams

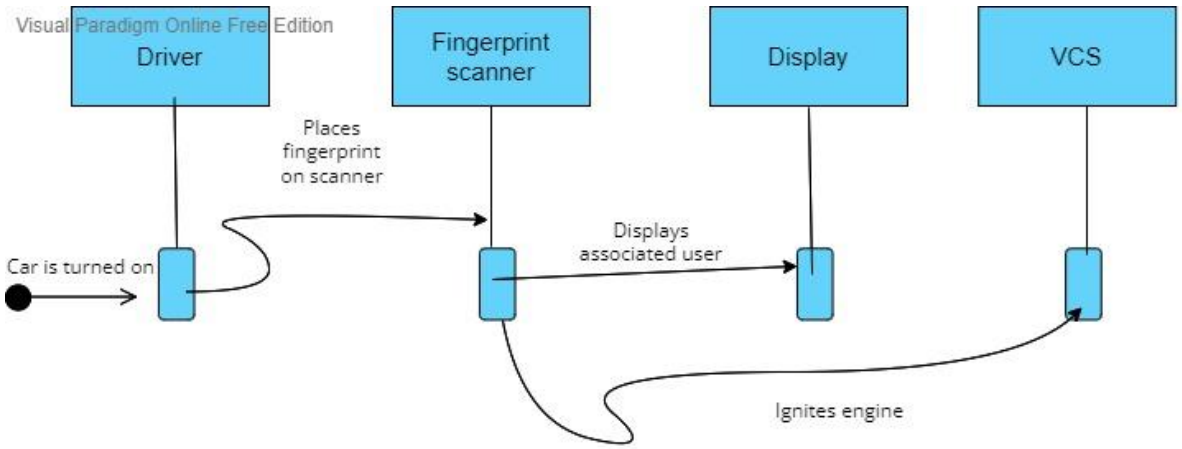
- Breathalyzer



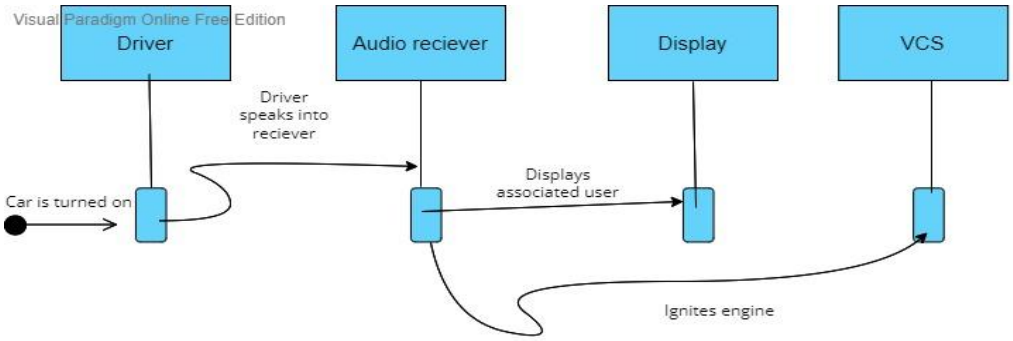
-Severe crash detection



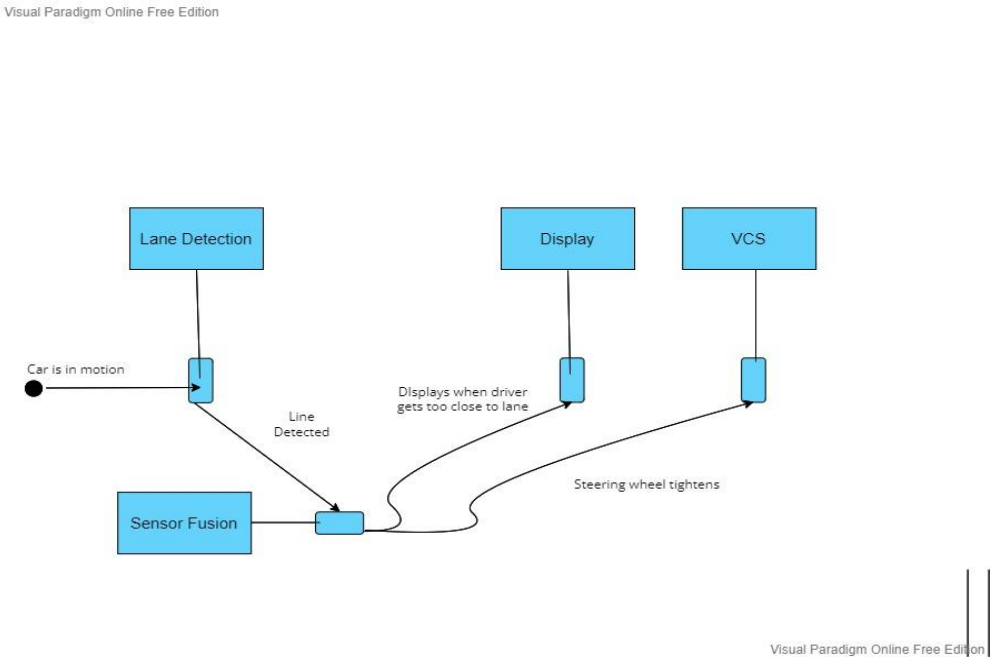
-Touch ID



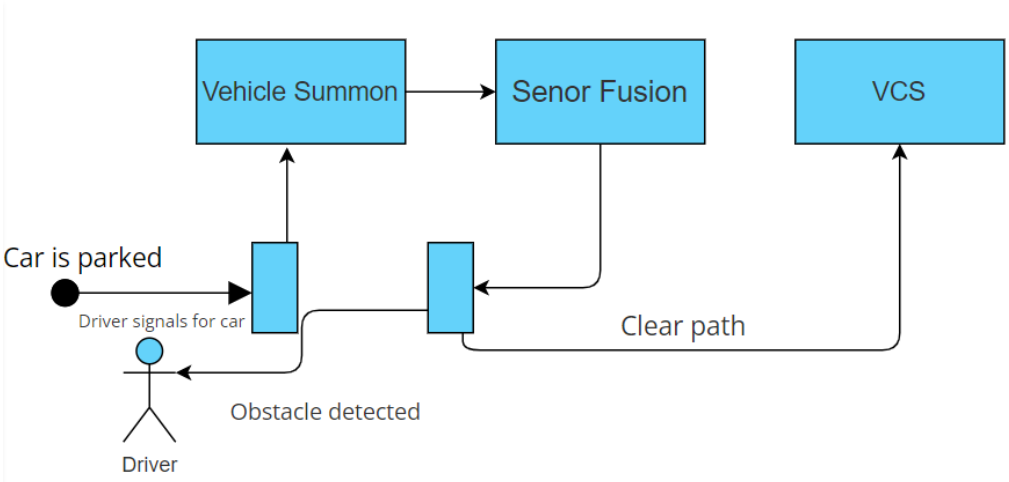
-Voice ID



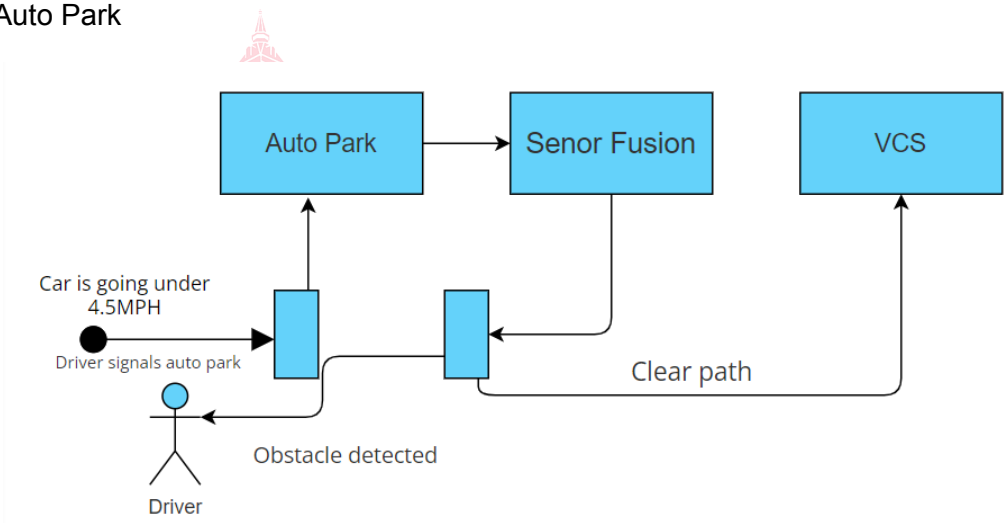
-Lane detection



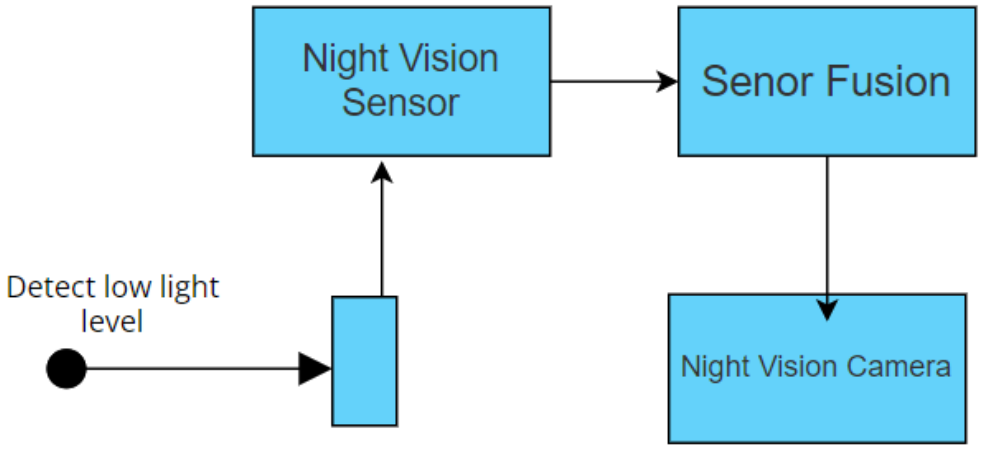
-Summon



-Auto Park



- Night View Camera



1.4.4 Classes
1.4.5 State Diagrams
Tasks for 1.4 —----------

- Complete all 10 class diagrams –
- Complete all 10 state diagrams Rayyan

1.5 Design

1.5.1 Software Architecture

- **1.5.1.1 Data Centered Architecture:** Data Centered Architecture would not be a good fit for IoT HTL because our project relies too heavily upon user input and repeated data recordings to benefit from the data storage capabilities. IoT HTL relies less on data storage and more on the processing of data that is constantly being taken in from Sensor Fusion.
- **1.5.1.2 Data Flow Architecture:** Because IoT HTL depends so heavily on taking constant data readings from sensors, having a series of filters and pipes would not be useful. The potential decrease in data transmission speed alone is enough to outweigh any pro because features like the Severe Collision Detection System must transmit data without worry of being slowed down by a series of filters and pipes.
- **1.5.1.3 Call Return Architecture:** Call Return Architecture is best suited for IoT HTL. The Controller Subprograms can manage the constant streams of sensor data from Application Subprograms. When a Sensor detects an obstacle there are limited hoops to jump through to alert the Main Program because the data alert will be passed through very few subprograms before the Main Program manages the alert. Compared to other architectures that may struggle in data flow management or speed of data transfer, Call Return Architecture serves as a middle ground that is most optimal for IoT HTL.
- **1.5.1.4 Object-Oriented Architecture:** IoT HTL relies mostly on Sensor Fusion to manage the features. The combination of many sensor readings determines most of the vehicle's actions most notably in the Auto Park and Summon features. Object-Oriented Architecture can manage requests from the user very well however it is not the most efficient in receiving a constant stream of sensor readings.
- **1.5.1.5 Layered Architecture:** Many of our features require limited interaction from the Driver to enact them. Features such as the Touch ID and Captain's Chairs only require a single interaction from the driver. In other words, our User Interface layer would hardly be utilized, limiting the effectiveness of focusing heavily on the User Interface level as the Layered Architecture does.
- **1.5.1.6 Model View Controller Architecture:** Model View Controller Architecture is beneficial for those who need to communicate through many different external ports. IoT

HTL's data is managed locally rather than in the cloud or on the internet so the main benefits of being able to handle many external requests are of no use to IoT HTL.

- **1.5.1.7 Finite State Machine Architecture:** IoT HTL requires the processing of a constant flow of data from the vehicle's many sensors. While Finite State Machines are great at managing a predictable stream of data, we have too many unknown variables to utilize this architecture's capabilities. Unknown variables such as other drivers, pedestrians, animals, and terrain are unpredictable and are best dealt with by frequent interpretation of data from our many sensors.

1.5.2 Interface Design

For breathalyzer, the interface is the screen on the dashboard that displays text to the driver which initially tells them to breathe into the device. It displays the level of the BAC and if the vehicle will start or not. This is able to be displayed because the data is sent to the User Interface. The captain's chairs do not have any type of display but the hardware which is the chair is connected to the software because it has to be told when it must turn or if there is an error and it can't turn. For auto parking, a message will be displayed on the screen if the car will begin parking or if the parking space is not adequate enough for the vehicle to initiate automatic parking, giving the driver the option to dismiss the message and find a new parking spot. Voice ID uses the screen display only when setting up a new voice profile which is interactive so that the driver may set up their name and other important information. The touch ID feature will not display anything on the screen since it is on the outside of the vehicle but it may be accessed by the driver through the settings from the display screen inside the car if they wish to set up a new fingerprint. A technician would be able to access this data as well through the special technician settings on the vehicle. The interface for the virtual assistant will be the car's speakers for when the assistant speaks back to the driver and the screen if it is required depending on what the driver is asking the virtual assistant to perform. For the summoning feature, the interface is not actually inside the vehicle but rather through the driver's phone in the app. The severe collision detection system uses the virtual assistant to assist the driver in the case of any emergency so although the data input is not the same, the interface that communicates with the driver is the same as the virtual assistant. The interface for the lane detection is the detection light on the side rear view mirrors that light up when the driver swerves out of the lane.

1.5.3 Component Level Design

The Breathalyzer can be implemented with a simple if-else conditional; if the BAC is within the acceptable range (0.0-0.8), the car's ignition can be engaged. Otherwise, the ignition will be locked. The Touch ID System can be represented by a for loop iterating over an array or vector. The aforementioned array or vector will contain data representing saved fingerprints. The user interface will allow the driver to scan their fingerprint, this will be the input. The for loop will then iterate over the array or vector, comparing each element with the input. If a data match is found, the car is unlocked, and the user may enter the car. If no match is found, the car remains locked. The Voice ID System can be represented in a similar manner to the touch id system.

Saved voice profiles can be stored as data in a data structure like an array, and when you speak, your voice is compared to these various elements to find a match. Each profile can have its own settings and personalizations that are linked to the profile, and these settings can include seat adjustment, side mirror adjustment, rear mirror adjustment, and drive mode(eco, sport, comfort). Voice profiles can also be represented with classes and objects, with different drivers each having its own customizations that are automatically implemented once the sensor recognizes a voice that is already saved. Severe Collision System can be visualized with an if conditional: If severe exterior damage is detected from the sensors, the airbags will go off, and the vehicle's current location will be sent to emergency services. Blindspot lane detection can be implemented using an if conditional within a while loop; while the car is driving on a road with multiple lanes, if the sensors detect a car approaching from the left or right blindspot, the blindspot indicator appears on the side mirror(s). The auto park can be implemented with an if statement and a function call. When the driver presses the auto parking button, if the sensors detect a valid parking space, the park function is called, otherwise, an error message is displayed.

1.6 Project Code

```
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog as fd
import tkinter.messagebox
import logging
from read import *
from input import *
import json

class tkinterApp(tk.Tk):

 def __init__(self, *args, **kwargs):

   tk.Tk.__init__(self, *args, **kwargs)

   container = tk.Frame(self)
   container.pack(side = "top", fill = "both", expand = True)

   container.grid_rowconfigure(0, weight = 1)
   container.grid_columnconfigure(0, weight = 1)
   self.frames = {}
   for F in (LandingPage, StartPage, VoiceID, TouchID, Pin, Alternative, Display,
Tech):
```

```
     frame = F(container, self)
     self.frames[F] = frame
     frame.grid(row = 0, column = 0, sticky ="nsew")
   self.show_frame(LandingPage)#selects entry page
  def show_frame(self, cont):
   frame = self.frames[cont]
   frame.tkraise()

#handles logging
logging.basicConfig(filename="gui.txt",
     format='%(asctime)s %(message)s', filemode='w')
logger=logging.getLogger()
logger.setLevel(logging.INFO)
logger.info("Vehicle turned on")
logger.info("----------------------------------")

profile =
[["pro1","3333",True,True],["pro2","4444",False,True],["pro3","5555",True,False],["pro
4","6666",False,False]]
pcurrent = []
OPTIONS = []
#initialize global variables -----
f = open('log.json')
read = json.load(f)

BACcheck = read['BACcheck']
validTID = read['TouchID']
validVID = read['VoiceID']
VA = read['VA']
SCS = read['SCS']
AutoPark = read['AutoPark']
Summoning = read['Summoning']
Cameras = read['Cameras']
Chairs = read['Chairs']
ldSensor = read['LaneDetection']

ldToggle = True
cdsToggle = True
nvcToggle = False

class LandingPage(tk.Frame):
 def __init__(self,parent,controller):
```

```
    tk.Frame.__init__(self,parent)

    for x in range(len(profile)):
        OPTIONS.append(profile[x][0])

    label = ttk.Label(self, text="Select a saved profile from the menu below")
    label.grid(row=0,column=1,padx=10,pady=10)

    #dropdown menu
    userSelect = tk.StringVar(self)
    userSelect.set(OPTIONS[0])

    dropDown=ttk.OptionMenu(self, userSelect, "select",*OPTIONS)
    dropDown.grid(row=1,column=1,padx=10,pady=10)
    '''
    update = tk.StringVar()
    def dropUpdater():
      dropDown["menu"].delete(0,"end")
      for item in OPTIONS:
        dropDown["menu"].add_command(label=item, command=lambda value=item:
update.set(value))'''

    confirmBTN = ttk.Button(self, text="Confirm
Selection",command=lambda:[confirmOnClick()])
    confirmBTN.grid(row=2,column=1,padx=10,pady=10)

    label2 = ttk.Label(self, text="Don't Have a profile? Press below to register.")
    label2.grid(row=3,column=1,padx=10,pady=10)
    #Register new profile button
    SignUpBTN = ttk.Button(self, text="Register",
                command=lambda: [controller.show_frame(Alternative),
logger.info("Registration Process Initiated")])
    SignUpBTN.grid(row=4, column=1, padx=10, pady=10)

    label2 = ttk.Label(self, text="Don't want to use a profile? Press below to use the
deafault driver display.")
    label2.grid(row=5,column=1,padx=10,pady=10)

    skipBTN = ttk.Button(self, text="skip", command=lambda: [skipVerify()])
    skipBTN.grid(row=6,column=1,padx=10,pady=10)

    def skipVerify():
```

```
    #test variable
    keyValid = True
    if keyValid == True:
      logger.info("Landing Page: Valid Key provided. Profile selection bypassed.")
      tkinter.messagebox.showinfo("Sucess!", "Press OK to continue to driver
display")
      app.title("Defualt Profile")
      controller.show_frame(Display)
    else:
      logger.info("Landing Page Error: Invalid Key provided.")
      tkinter.messagebox.showinfo("Error", "Invalid key provided. Contact technician
for more information")
  #related functions
  def confirmOnClick():
    if OPTIONS.__contains__(userSelect.get()) == True:
      global pcurrent
      pcurrent = profile[OPTIONS.index(userSelect.get())]
      print(pcurrent)
      app.title("%s's profile" %(userSelect.get()))
      controller.show_frame(StartPage)
    else:
      tkinter.messagebox.showinfo("Error", "Please select a valid profile.")
      logger.error("Invalid profile selected")

class StartPage(tk.Frame):
 def __init__(self,parent,controller):
  tk.Frame.__init__(self,parent)

  backBTN = ttk.Button(self, text="< Profile Selection",
             command=lambda: [controller.show_frame(LandingPage),
logger.info("Returned to Profile Selection")])
  backBTN.grid(row=0,column=0,padx=10,pady=10)

  label = ttk.Label(self, text="Select Driver Profile Sign in Method")
  label.grid(row=0, column=3, padx=10, pady=10)

  VoiceBTN = ttk.Button(self, text="Voice ID",
             command=lambda: [VIDcheck(), logger.info("Voice ID Login Method
Selected")])
  VoiceBTN.grid(row=1, column=2, padx=10, pady=10)

  def VIDcheck():
```

```
    if pcurrent[2] == True:
      controller.show_frame(VoiceID)
      logger.info("Voice ID: valid profile. VID ready for user.")
    else:
      logger.info("Voice ID: Voice ID not set up for this profile")
      tkinter.messagebox.showinfo("Sign In Error","Voice ID is not set up for this
profile")


  TouchBTN = ttk.Button(self, text="Touch ID",
             command=lambda: [TIDcheck(), logger.info("Touch ID Login Method
Selected")])
  TouchBTN.grid(row=1, column=3, padx=10, pady=10)

  def TIDcheck():
    if pcurrent[3] == True:
      controller.show_frame(TouchID)
      logger.info("Touch ID: valid profile. TID ready for user.")
    else:
      logger.info("Touch ID: Touch ID not set up for this profile")
      tkinter.messagebox.showinfo("Sign In Error","Touch ID is not set up for this
profile")

  PinBTN = ttk.Button(self, text="Pin",
             command=lambda: [controller.show_frame(Pin), logger.info("Pin Login
Method Selected")])
  PinBTN.grid(row=1, column=4, padx=10, pady=10)

  label2 = ttk.Label(self, text="Haven't set up a profile?")
  label2.grid(row=2, column=3, padx=10, pady=10)

  SignUpBTN = ttk.Button(self, text="Register",
             command=lambda: [controller.show_frame(Alternative),
logger.info("Registration Process Initiated")])
  SignUpBTN.grid(row=3, column=3, padx=10, pady=10)
  # to be removed. Just here for the time being to skip the verification stages
  JumpBTN = ttk.Button(self, text="jump to driver display",
             command=lambda: controller.show_frame(Display))
  JumpBTN.grid(row=5, column=5,padx=10, pady=10)

class VoiceID(tk.Frame):
 def __init__(self,parent,controller):
```

```
   tk.Frame.__init__(self,parent)

   homeButton = ttk.Button(self, text="< Back",
               command=lambda: [controller.show_frame(StartPage),
logger.info("Returned to Home Page")])
   homeButton.grid(row=0,column=0,padx=10,pady=10)

   label = ttk.Label(self, text="VoiceID")
   label.grid(row=0, column=1, padx=10, pady=10)

   def onClick():
     logger.info("Voice ID recording initiated")
     tkinter.messagebox.showinfo("Voice ID being recorded", "Press Ok when you've
completed your entry phrase")
     verifyBTN = ttk.Button(self, text="Press to verify VoiceID", command=lambda:
validID())
     verifyBTN.grid(row=1,column=2,padx=10,pady=10)

   def validID():#ehhhh
     if validVID == 'True' and BACcheck == 'True':
       logger.info("Voice ID Successful")
       controller.show_frame(Display)
     elif BACcheck == 'False':
       logger.info("Voice ID Failed. ERROR: INVALID BAC")
       tkinter.messagebox.showinfo("Invalid BAC", ("DO NOT DRIVE! BAC OVER LEGAL
LIMIT!"))
     else:
       logger.info("Voice ID Failed. ERROR: Voice ID not saved")
       tkinter.messagebox.showinfo("Voice ID","Voice ID invalid. Try again")

   startBTN = ttk.Button(self, text="Press and begin speaking", command=lambda:
onClick())
   startBTN.grid(row=1,column=1,padx=10,pady=10)

class TouchID(tk.Frame):
 def __init__(self,parent,controller):
   tk.Frame.__init__(self,parent)

   homeButton = ttk.Button(self, text="< Back",
               command=lambda: [controller.show_frame(StartPage),
logger.info("Returned to Home Page")])
   homeButton.grid(row=0,column=0,padx=10,pady=10)
```

```
   label = ttk.Label(self, text="TouchID")
   label.grid(row=0, column=1, padx=10, pady=10)

   label = ttk.Label(self, text="Press your finger below for Touch ID")
   label.grid(row = 1, column = 1, padx=10, pady=10)

   TouchBTN = ttk.Button(self, text="Press Here", command=lambda: verify())
   TouchBTN.grid(row = 2,column = 1, padx=10, pady=10)

   def verify():
     if validTID == 'True':
       logger.info("Touch ID: Touch ID entry suecessful. Displaying Driver Display")
       tkinter.messagebox.showinfo("Touch ID", "Touch ID Successful! Press Ok to
continue to Driver Display")
       controller.show_frame(Display)
     else:
       logger.info("Touch ID: Touch ID entry failed. ID invalid.")
       tkinter.messagebox.showinfo("Touch ID","Touch ID Invalid. Try again.")

class Pin(tk.Frame):
 def __init__(self,parent,controller):
   tk.Frame.__init__(self,parent)

   homeButton = ttk.Button(self, text="< Back",
               command=lambda: controller.show_frame(StartPage))
   homeButton.grid(row=0,column=0,padx=10,pady=10)

   label = ttk.Label(self, text="Pin")
   label.grid(row=0, column=1, padx=10, pady=10)

   label2 = ttk.Label(self,text="Enter PIN below")
   label2.grid(row=1, column=1, padx=10,pady=10)

   PinBTN = ttk.Button(self, text="Confirm", command=lambda: verify())
   PinBTN.grid(row=3,column=1,padx=10,pady=10)

   pinEntry = tk.StringVar()
   inputPin = ttk.Entry(self, textvariable=pinEntry)
   inputPin.grid(row = 2, column=1,padx=10,pady=10)

   def verify():
```

```
    if pinEntry.get() == "tech":
      logger.info("Pin Entry: Technician Pin entered. Displaying Tech View")
      tkinter.messagebox.showinfo("Technician Pin", "Press OK to continue to
Technician Menu")
      controller.show_frame(Tech)
    elif pinEntry.get() == pcurrent[1]:
      logger.info("Pin Entry: Driver Pin entered succesfully. Displaying Drive
Display")
      tkinter.messagebox.showinfo("Driver Pin", "Sucess! Press Ok to continue to
Driver Display")
      controller.show_frame(Display)
    else:
      logger.info("Pin Entry: Ivalid Pin entered")
      tkinter.messagebox.showinfo("Pin Entry,","Pin is invalid. Try Again")

class Tech(tk.Frame):
 def __init__(self,parent,controller):
   tk.Frame.__init__(self,parent)

   homeButton = ttk.Button(self, text="< Back", command=lambda:
controller.show_frame(StartPage))
   homeButton.grid(row=0,column=0,padx=0,pady=0)

   #button to open driver log file
   txtBTN = ttk.Button(self,text="Open Driver log file", command=lambda: [openLog()])
   txtBTN.grid(row=2,column=0)

   jsonBTN = ttk.Button(self,text="View Vehicle Data", command=lambda: [openLog()])
   jsonBTN.grid(row=3,column=0)

   #button to view/manage profiles
   '''profBTN = ttk.Button(self, text="Manage Profiles", command=lambda:[profEdit()])
   profBTN.grid(row=4, column=0)'''

   #open file code
   text = tk.Text(self, height=12)
   text.grid(row = 1, column = 1, stick='nsew', padx=10, pady=10)
   filetypes = (
       ('text files', '*.txt'),
       ('log files', '*.log'),
       ('json files', '*.json'),
       ('All files', '*.')
```

```
    )
    def openLog():
      f = fd.askopenfile(filetypes=filetypes)
      text.insert('1.0', f.readlines())

    '''def profEdit():
      pass'''

class Alternative(tk.Frame):#profile creation
 def __init__(self,parent,controller):
    #LandingPage.__init__(self, parent, controller)
    tk.Frame.__init__(self,parent)

    #app.title("Registration Page")

    homeButton = ttk.Button(self, text="< Back",
                command=lambda: controller.show_frame(StartPage))
    homeButton.grid(row=0,column=0,padx=10,pady=10)

    label = ttk.Label(self, text="Register New Profile")
    label.grid(row=0, column=1, padx=10, pady=10)

    #username and related code
    enterUser = ttk.Label(self, text="Create a username of only alphabet characters:")
    enterUser.grid(row=1, column=1, padx=10, pady=10)

    username = tk.StringVar()
    inputUser = ttk.Entry(self,textvariable=username)
    inputUser.grid(row=1,column=2,padx=10,pady=10)

    #Touch ID and related code
    enableTID = ttk.Label(self, text="Enable Touch ID?")
    enableTID.grid(row=4, column=1, padx=10, pady=10)

    TID=tk.BooleanVar()
    TIDcheck = ttk.Checkbutton(self, text="Yes", variable=TID, onvalue=True,
offvalue=False)
    TIDcheck.grid(row=4,column=2,padx=10,pady=10)

    #Voice ID and related code
    enableVID = ttk.Label(self, text="Enable Voice ID?")
    enableVID.grid(row=3, column=1, padx=10, pady=10)
```

```
   VID=tk.BooleanVar()
   VIDcheck = ttk.Checkbutton(self, text="Yes", variable= VID, onvalue=True,
offvalue=False)
   VIDcheck.grid(row=3,column=2,padx=10,pady=10)

   #Pin and related code
   enterPin = ttk.Label(self, text="Create a 4 digit Pin:")
   enterPin.grid(row=2, column=1, padx=10, pady=10)

   Pin = tk.StringVar()
   inputPin = ttk.Entry(self, textvariable=Pin)
   inputPin.grid(row = 2, column=2,padx=10,pady=10)

   validTIDscan = True
   validVIDscan = True
   def TIDscan():
     tkinter.messagebox.showinfo("Touch ID Setup","Press and hold your finger to the
TID sensor. Press OK when finished")
     validTIDscan == True
   def VIDscan():
     tkinter.messagebox.showinfo("Voice ID Setup","Repeat the following phrase: 'The
quick brown fox jumped over the lazy dog'. Press OK when finished.")
     validVIDscan = True
   #confirmation button and related code
   confirm = ttk.Button(self, text="Confirm", command=lambda: storeProfile())
   confirm.grid(row=5, column=2, padx=10, pady=10)
   #stores in the following format:
   # profiles = [[username, Pin, TID, VID],[...],...]
   def storeProfile():
     # is Pin 4 digits?, is Pin only integers?, is username a string of only alphabet
chars?
     if Pin.get().isdigit() != True and len(Pin.get()) != 4:
       logger.error("Registration Error. Invalid Pin.")
       tkinter.messagebox.showinfo("Registration Error", "Invalid Pin given. Please
try again")
     elif username.get().isalpha() != True:
       logger.error("Registration Error. Invalid username.")
       tkinter.messagebox.showinfo("Registration Error", "Invalid Username given.
Please try again.")
     else:
       temp = []
```

```
        temp.append(username.get())
        temp.append(Pin.get())
        temp.append(TID.get())
        temp.append(VID.get())
        global pcurrent
        pcurrent = temp
        profile.append(temp)
        #pstore.info(temp)
        OPTIONS.append(username.get())
        #LandingPage.dropUpdater()
        if TID.get() == True:
          TIDscan()
          if validTIDscan != True:
            logger.info("Registration: Touch ID setup failed")
            tkinter.messagebox.showinfo("Touch ID Setup", "Reading failed. Try again.")
            TIDscan()
        if VID.get() == True:
          VIDscan()
          if validVIDscan != True:
            logger.info("Registration: Voice ID setup failed.")
            tkinter.messagebox.showinfo("Voice ID Setup","VID setup failed. Try again")
            VIDscan()
        logger.info("Registration: registration complete, new profile set at pcurrent")
        tkinter.messagebox.showinfo("Registration Complete", "Registration Sucessful!
Press Ok to return to the landing page.")
        app.title("%s's profile" %(pcurrent[0]))
        controller.show_frame(StartPage)

class Display(tk.Frame):
 def __init__(self,parent,controller):
   tk.Frame.__init__(self,parent)

   homeBTN = ttk.Button(self, text="< Back",
               command=lambda: controller.show_frame(StartPage))
   homeBTN.grid(row=0,column=0,padx=10,pady=10)

   #Tree to post relevent information about driver display actions
   tree = ttk.Treeview(self)
   tree.grid(row=1,column=2,padx=10,pady=10)
   tree.insert('', 'end', 'widgets', text="Information Display")

   #Virtual Assistant Button and related functions
```

```
   vaBTN = ttk.Button(self, text="Virtual Assitant",
                command=lambda: vaOnClick())
   vaBTN.grid(row=0,column=1,padx=10,pady=10)

   def vaOnClick():#CHECK--------
     tkinter.messagebox.showinfo("Virtual Assistant Listening","Please begin giving
your command. Press Ok when finished.")
     if VA == 'True':
       tree.delete('widgets')
       tree.insert('', 'end', 'widgets', text="VA Command Processed!")
       logger.info("Virtual Assistant: Command succesfully processed.")
     else:
       tkinter.messagebox.showinfo("Virtual Assistant","Command Invalid. Please try
again or contact Technician.")
       tree.delete('widgets')
       tree.insert('', 'end', 'widgets', text="VA Command Invalid")
       logger.info("Virtual Assistant: Invalid Command")

   #Captain's Chairs Button and related functions
   ccBTN = ttk.Button(self, text="Captain's Chairs",
                command=lambda: ccOnClick())
   ccBTN.grid(row=1,column=1,padx=10,pady=10)

   def ccOnClick(): #CHECK[------------
     chairState = "Drive"
     if Chairs == 'False':
       MsgBox = tkinter.messagebox.askquestion("Captain's Chair","Are you sure you
want to turn your chair?. Select yes to begin.",icon='warning')
       if MsgBox == 'yes':
         chairState = "Entry"
         tkinter.messagebox.showinfo("Captain's Chair","Chair sucessfully turned.
Press OK to return chair to its original position")
         tree.delete('widgets')
         tree.insert('', 'end', 'widgets', text="Chair Returned!")
         chairState = "Drive"
         logger.info("Captain's Chair: Chair turn sucessful")
     else:
       chairState = "Drive"
       tree.delete('widgets')
       tree.insert('', 'end', 'widgets', text="Chair Turn Failed!")
       logger.info("Captain's Chair: Chair Turn Failed. Object detected in path")
```

```
  #AutoPark Button and related functions
  apBTN = ttk.Button(self, text="Auto Park",
              command=lambda: apOnClick())
  apBTN.grid(row=2,column=1,padx=10,pady=10)

  def apOnClick():
    validSpot = True
    apSensor = True
    if validSpot == True and apSensor == False:
      MsgBox = tkinter.messagebox.askquestion("Auto Park","Are you sure you want to
begin Auto Park?", icon='warning')
      if MsgBox == 'yes':
        tkinter.messagebox.showinfo("Auto Park","Auto Park Initiated. Please do not
touch the steering wheel.")
        logger.info("Auto Park sequence completed")
        tree.delete('widgets')
        tree.insert('', 'end', 'widgets', text="Vehicle Parked!")
    elif apSensor == True:
      tkinter.messagebox.showinfo("Auto Park","Auto Park detected an object")
      tree.delete('widgets')
      tree.insert('', 'end', 'widgets', text="Object Detected! Auto Park Failed")
      logger.info("Auto Park sequence failed. Object Detected in path.")
    elif validSpot == False:
      tkinter.messagebox.showinfo("Auto Park","Auto Park detected an invalid parking
space.")
      tree.delete('widgets')
      tree.insert('', 'end', 'widgets', text="Object Detected! AP Failed")
      logger.info("Auto Park sequence failed. Invalid parking space")
    else:
      tkinter.messagebox.showinfo("Auto Park","Auto Park Error. Please contact a
Technician")
      logger.info("Auto Park Error Unknown")

  #Lane Detection Toggle and related functions
  ldBTN = ttk.Button(self, text="Lane Detection",
              command=lambda: ldOnClick())
  ldBTN.grid(row=0,column=3,padx=10,pady=10)

  def ldOnClick():#CHECK-----------------------------
    global ldToggle
    if ldToggle == True and ldSensor == True:#toggled on, sensor detected object
      tree.insert('', 'end', 'widgets', text="Object Detected!")
```

```
      logger.info("Object detected by Lane Detection")
    elif ldToggle == False:#turn on LD
      MsgBox = tkinter.messagebox.askquestion("Lane Detection System","Select Yes to
enable Lane Detection System",icon='warning')
      if MsgBox == 'yes':
        ldToggle == True
        logger.info("Lane Detection Enabled")
        tree.delete('widgets')
        tree.insert('', 'end', 'widgets', text="LDS enabled")
    elif ldToggle == True:#turn off LD
      MsgBox = tkinter.messagebox.askquestion("Lane Detection System","Select Yes to
disable Lane Detection System",icon='warning')
      if MsgBox == 'yes':
        ldToggle = False
        logger.info("Lane Detection Disabled")
        tree.delete('widgets')
        tree.insert('', 'end', 'widgets', text="LDS disabled")
    else:
      tkinter.messagebox.showinfo("Lane Detection System","Error in Lane Detection
System. Please contact technician.")
      tree.delete('widgets')
      tree.insert('', 'end', 'widgets', text="LDS error")
      logger.info("Lane Detection: Error in detection system")

  #Vehicle Summon and related functions
  sumBTN = ttk.Button(self, text="Vehicle Summon",
             command=lambda: sumOnClick())
  sumBTN.grid(row=1,column=3,padx=10,pady=10)

  def sumOnClick():#CHECk-------------------------
    Summoning = True
    if Summoning == True:
      MsgBox = tkinter.messagebox.askquestion("Vehicle Summon","Select yes to confirm
vehicle summon",icon='warning')
      if MsgBox == 'yes':
        tkinter.messagebox.showinfo("Vehicle Summon","Summoning in progress...")
        tree.delete('widgets')
        tree.insert('', 'end', 'widgets', text="Vehicle Summoned!")
        logger.info("Summon: Sucesfully Summoned")
        Summoning = False
    else:
      tree.delete('widgets')
```

```
      tree.insert('', 'end', 'widgets', text="Already Summoned!")
      logger.info("Summon: Summon Failed.")
      Summoning = False

  #Collision Detection System and related funcitons
  cdsBTN = ttk.Button(self, text="Collision Detection System",
             command=lambda: cdsOnClick())
  cdsBTN.grid(row=2,column=3,padx=10,pady=10)

  #if collision is detected alert driver
  if SCS == 'True':
    tkinter.messagebox.showinfo("Collision Detection System","Collision Detected!")
    logger.info("Collision Detection System: Collision Detected. 911 being alerted.")
    tree.delete('widgets')
    tree.insert('', 'end', 'widgets', text="VA Alerting 911!")

  def cdsOnClick():#CHECK-------------------------------------
    global cdsToggle
    if cdsToggle == True and SCS == True:
      tkinter.messagebox.showinfo("Collision Detection System","Collision Detected!")
      logger.info("Collision Detection System: Collision Detected. 911 being
alerted.")
      tree.delete('widgets')
      tree.insert('', 'end', 'widgets', text="VA Alerting 911!")
    elif cdsToggle == True:
      MsgBox = tkinter.messagebox.askquestion("Collision Detection System","Are you
sure you want to disable the Collision Detection System's ability to automatically
contact authorities in the event of a collision? Select Yes to consent",icon =
'warning')
      if MsgBox == 'yes':
        tree.delete('widgets')
        tree.insert('', 'end', 'widgets', text="CDS Disabled")
        logger.info("Collision Detection System: System Disabled by user.")
        #tree.insert('', 'end', 'widgets', text="CDS auto alert disabled")
        cdsToggle = False
    elif cdsToggle == False:
      MsgBox = tkinter.messagebox.askquestion("Collision Detection System","Press Ok
to enable CDS 911 auto alert",icon = 'warning')
      if MsgBox == 'yes':
        tree.delete('widgets')
        tree.insert('', 'end', 'widgets', text="CDS Enabled")
        logger.info("Collision Detection System: System Enabled by User.")
```

```
        #tree.insert('', 'end', 'widgets', text="CDS auto alert enabled")
        cdsToggle = True
    else:
      tkinter.messagebox.showinfo("Collision Detection System","CDS error. Please
contact technician")

    #NightView 360 camera
  nvcBTN = ttk.Button(self, text="Night View 360 Camera",command=lambda:
nvcOnClick())
  nvcBTN.grid(row=3,column=2,padx=10,pady=10)

  def nvcOnClick():#check-------------
    global nvcToggle
    if nvcToggle == True:
      MsgBox = tkinter.messagebox.askquestion("Night Vision Camera", "Select yes to
turn off Night Vision 360 camera.",icon='warning')
      if MsgBox == 'yes':
        logger.info("Night Vision 360 camera turned off")
        nvcToggle = False
        tree.delete('widgets')
        tree.insert('', 'end', 'widgets', text="NVC turned off")
    elif nvcToggle == False:
      MsgBox = tkinter.messagebox.askquestion("Night Vision Camera","Select Yes to
turn on Night Vision 360 camera.",icon='warning')
      if MsgBox == 'yes':
        logger.info("Night Vision 360 camera turned on")
        nvcToggle = True
        tree.delete('widgets')
        tree.insert('','end','widgets',text="Displaying NVC")

app = tkinterApp()
app.geometry("800x600")
app.title("CS347")
app.mainloop()
```

1.7 Testing

1.7.1 Scenario Based Testing

Driver BAC is invalid:

Abstract: In the event the Driver's BAC is above the legal limit, the car should alert the Driver and not allow them to proceed to the driver display. Passed.

Voice ID invalid:

Abstract: If the driver's Voice ID entry phrase is not valid, the driver should not be permitted to continue to the driver display with that given method. Passed.

Touch ID invalid:

Abstract: If the driver's Touch ID finger print is not recognized by the sensors, the driver should not be permitted to continue to the driver display with the Touch ID method. Passed.

Incorrect Pin:

Abstract: If the driver's Pin is incorrect, the driver should not be permitted to continue the driver display with the Pin method. Passed.

Technician Login:

Abstract: All driver actions and sensor data should be stored into log files viewable by the technician in the technician display. This display is to be only accessed by using the technician pin "tech". Passed

Register New Profile:

Abstract: A driver should be able to create a profile and store the profile for future use. Failed – profile storage not implemented.

Severe Collision Detection Sytem Alert:

Abstract: If the driver has the Severe Collision Detection System enabled in the driver display, an alert should be shown on the driver display in the event of a collision. The VA shall also alert the nearest authorities. Passed.

Lane Detection Alert:

Abstract: If the driver has teh Lane Detection Alert enabled in the driver display, an alert shall be shown on the driver display in the event an object is detected by the blindspot sensors. Passed

1.7.2 Validation Testing

1. Severe Collision Detection System
    a. If sensor detects collision, reports information to driver.
        i. PASS
    b. Can be toggled on and off at the driver's discretion.
        i. PASS
2. Virtual Assistant
    a. System prompts user to speak
        i. PASS
    b. Determines if command was valid or invalid
        i. PASS
    c. Returns result of command processing procedure
        i. PASS
3. Touch ID
    a. Can be programmed
        i. PASS

   b. Checks validity of fingerprint and returns necessary response
      i. PASS
4. Voice ID
   a. Can be programmed
      i. PASS
   b. Checks validity and returns correct response
      i. PASS
   c. Checks BAC level and returns correct response
      i. PASS
5. Captain's Chairs
   a. Button-actuated
      i. PASS
   b. If obstacle is encountered, return to original state
      i. PASS
   c. Alerts driver of current state
      i. PASS
6. Lane Detection
   a. Can be toggled on or off at the driver's discretion
      i. PASS
   b. If object is detected by sensors, return correct response
      i. PASS
7. Summon
   a. Button-actuated
      i. PASS
   b. If sensors detect an obstacle, vehicle returns alert and waits until obstacle has been cleared
      i. PASS
8. Breathalyzer
   a. If BAC is invalid, return alert to driver
      i. PASS
   b. If BAC is valid, continue to driver display
      i. PASS
9. Night Vision 360 Camera
   a. Can be toggled on and off
      i. PASS
   b. Displays 'camera' on driver display
      i. PASS
10. Auto Park
   a. Button-actuated
      i. PASS
   b. If sensors determine spot is invalid, alert driver
      i. PASS
   c. If sensors detect sudden object interference, alert driver and stop movement
      i. PASS

All necessary requirements have been met.