Version 1.7

ALSET - Best Team

Dylan Edelman, Christopher Horn, Isabel Hughes, Steven Truong, Michael Mancino

17 May 2022

Team 2 CS 347-B

Table Of Contents

ALSET - Best Team 0

Section 1: Introduction 2

Section 2: Functional Architecture 7

Section 3: Requirements 10

Section 4: Requirement Modeling 19

Section 5: Design 35

Section 6: Use-Case Testing 47

Section 7: Code 52

Section 1: Introduction

IoT (Internet of Things) is essentially a collection of sensors, mini processors and physical objects embedded with software/technologies that allow for the exchange and processing of information between other devices, systems and the internet. Consider that the majority of vehicle accidents are human error. Consider how often people go to mechanics to diagnose their vehicles only to leave still confused. Now imagine a system that could drive you from place to place in a safe and fast manner. A system that communicates with other vehicles to orchestrate lane changes and turns while also being able to see upcoming road conditions while having the capability to self-diagnose and attempt repairs using information straight from the manufacturer. All of this is possible when using IoT edge devices to capture information and IoT engines to process said information. As this is a mission critical real-time embedded system, availability, reliability and efficiency are of great importance to us.

As a self-organized team, one of our biggest goals in this project is to work efficiently and effectively towards a successful project completion in the time provided to us. Furthermore, we are committed to high quality software development, in which we are determined to create our software with an emphasis on security and debugging. Following the Agile Process Model, we first begin by understanding the problem and gathering all necessary requirements to communicate our collective vision for the project. After which, we will plan out our major features and devise their capabilities within the system, estimating the amount of work needed for integration. Next, we will model out concept designs and analyze real-world advantages and disadvantages to our proposed sensory features. Executing on our plan, we will test if our solution is one that effectively answers the problem of creating a safe and inexpensive vehicle by utilizing frequent and constructive customer feedback. If it is not a valid

solution, we will return back to the drawing board to introspectively reflect on our previous work and reconstruct the software so it is better suited for our next iterative design. We will continue to repeat these steps until we, as a team, are unanimously confident with our project and that our primary goal, satisfaction from the customer, is reached as well. With a timeframe of February 1st to May 1st and our very different sets of qualifications, we can effectively move through the problem solving process, with our individual skills helping in each step.

- Isabel - People-Oriented, Diligent Worker, Coding
- Michael - Efficient, Coding, Goal-Oriented
- Steven - Analysis, Problem Solving, Reserved
- Christopher - Coding, Project Management, Task-Oriented
- Dylan - Communication, Detail-Oriented, Writing Skill

Altogether, we are oriented towards finishing our own individual tasks and reaching our goals in a manner where every detail is examined for flaws. We also are able to communicate effectively with one another and efficiently complete every aspect of our project. Isabel, Michael, and Christopher excel in programming and can ensure all software is working correctly in a technical mindset. Dylan is able to ensure our documentation is clear and communicates all aspects of the software that we have detailed. Lastly, Steven joins the entire team together and keeps us on track, while also providing solutions for parts of the project we may find challenging.

Features:

1. **Flexible Map routing**
    - While we are all well known of GPS apps, such as Google Maps or Waze, there are several downsides associated with them. First off is the distraction of the phone itself. By integrating the GPS routing into the vehicle, it allows for distraction less driving and keeps the driver's eyes on the road {HUD #3}. Secondly, GPS apps strive for the fastest arrival, which navigates drivers

through neighborhoods and side streets which are not meant to accommodate mass amounts of traffic. Our flexible map routing will take into consideration the volume of traffic, construction, weather conditions, and other environmental factors that would affect time of arrival. To combat the congestion of neighborhoods and other small streets, our service will attempt to avoid all of those roads and still get the driver there in an optimal amount of time. Throughout the trip if any of these statuses change, the driver will be informed and re-routed with the best possible course of action.

2. **Lane mitigation**
    - Lane mitigation uses a detection system with forward and side facing cameras to continually monitor the lines of the lanes surrounding the vehicle. The vehicle usually produces visual and audible warnings with a display on the HUD and a distinct beeping noise to alert the driver that they have left the lane. In our vehicle, we are going to take lane departure warnings to a new level and implement lane keep and centering assistance. If the driver is ever to leave the lane, keep assistance will provide slight steering and/or braking to continue to keep the vehicle in its lane. Centering assistance will provide the same as keep assistance, but will also center the vehicle within the two lanes. As this safety feature can seem unnecessary to some, it will also be able to be disabled or re-enabled at any time.
3. **HUD**
    - Head-up display, or HUD, is software that projects the dashboard onto the windshield, just below the driver's line of sight. The features on the HUD include current speed, the speed limit of the road, navigation information, alerts from the vehicles running systems, and the state of the battery. The biggest reason to use a HUD is for safety. According to the National Highway and Traffic Safety Administration, in 2019, 3,142 people were killed due to distracted driving. By using the HUD, the driver can receive notifications from your phone by Bluetooth, and have navigation directions and speed directly in front of them. This allows for the overall and continual safety of the driver and those on the road with them.
4. **Virtual side mirrors**
    - Virtual side mirrors, while a newer technology, can extensively improve the safety of a vehicle. By using cameras on either side of the vehicle, the virtual side mirrors are projected on either side of the vehicle, and on the HUD if desired. These mirrors can also help in guidance with turning and adjusting the vehicle in a lane. Due to heated and water-resistant lenses, the e-mirrors will never freeze or be misty, with perfect side views at all times. The last feature of these mirrors is that they help in blind spot detection, by allowing the driver to move the camera's view around. This will also allow for a green overlay for safe

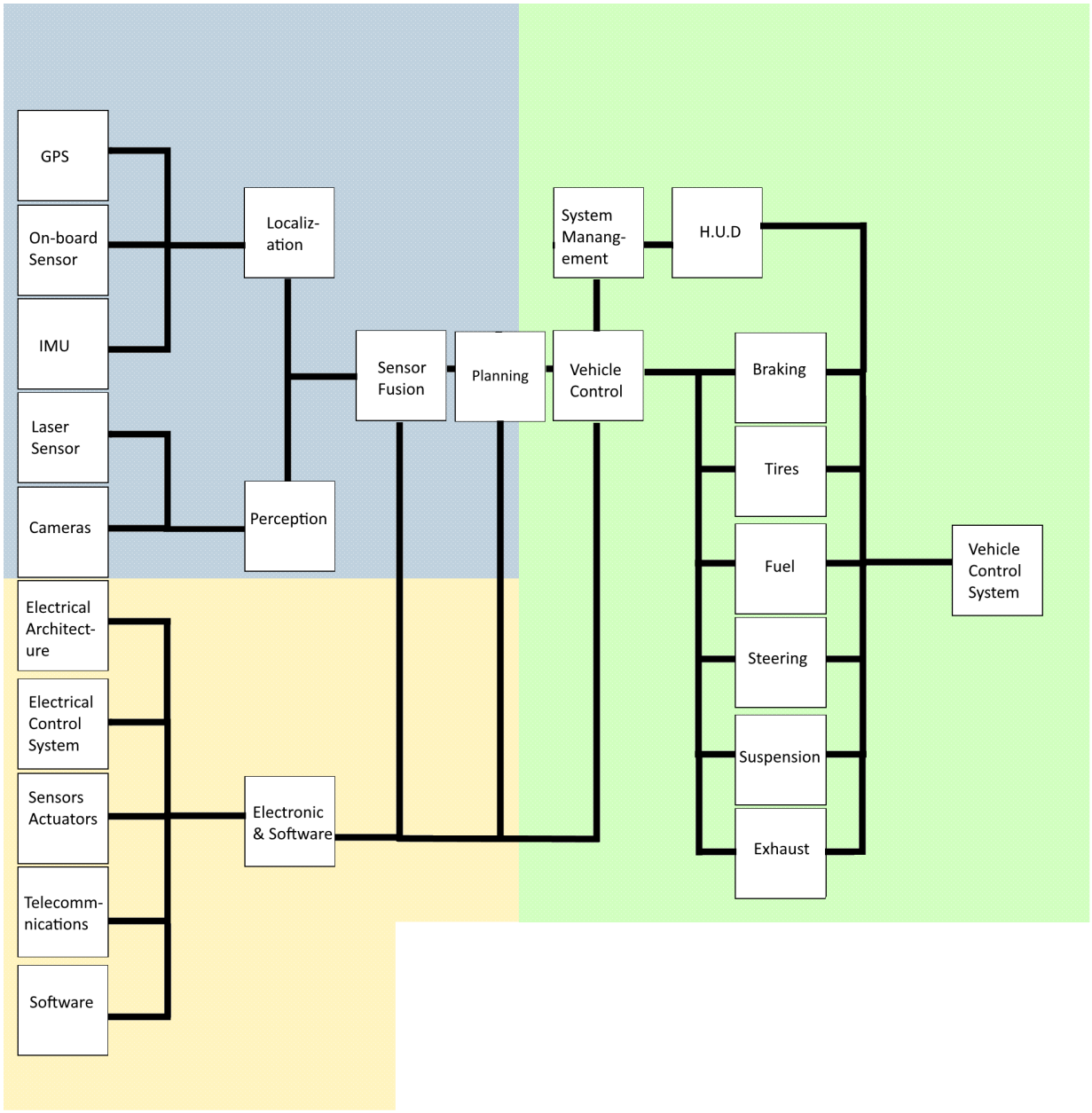
lane switching, and a red overlay for unsafe lane switching by using the AI sensors around the vehicle.

5. **Solar panels**
    - One of the bigger complaints about electric vehicles is the concept of phantom drainage. Phantom drain is the drain of energy which naturally occurs in batteries due to lack of use or driving in the cold. To combat this issue, solar cells will be placed into the roof of the vehicle to charge the battery when the vehicle is not in use. Another feature additional to the solar panels is a newer, Zinc-ion battery for the vehicle. Zinc-Ion batteries are lighter, more affordable, and more green than normal lithium ion batteries. They also increase safety as Zinc does not cause a heat reaction, which would combat phantom drainage when driving in a cold environment.
6. **AI sensors**
    - What good is a self-driving vehicle if it can't actually drive itself? Using our detection sensors, the vehicle will automatically accelerate and stop based on the current situation. In conjunction with our previous features, we can determine when to make turns, switch lanes and come to stops. Using map information and sensors, we will be able to stop at intersections and in conjunction with recognition software to see any crossing pedestrians. Information will be sent to us so we can improve our driving and mapping software.
7. **Auto parking/auto summon**
    - Even with parking cameras, drivers may still find it difficult to park in certain positions. Taking it to the next level, we will simply tell the vehicle to park itself instead of relying on the driver. Using sensors on the vehicle, we can determine valid parking positions and attempt to self park using a virtual representation of the current situation to calculate the actions and movements needed to park. Using an app or a feature implemented in the keys, we will allow the driver to automatically summon the vehicle. Simply using your phone’s GPS coordinates, the vehicle will drive there safely and wait for the driver to unlock and enter the vehicle. Of course the vehicle will stay on the road and not just drive into a random building.
8. **Automatic emergency contact**
    - Should the vehicle get in a serious crash, the vehicle will automatically contact emergency services and send information such as coordinates, where the crash happened such as right & left side and estimation of the force of impact. Any people designated as emergency contacts will also be informed. To prevent false positives, if the driver fails to respond, emergency services will be contacted. Logs will also be sent to us to determine what exactly happened and how we can use this information to update our software.

9. **Self-diagnosis/repair**
    - The ability to self-diagnose problems and possibly attempt repairs is very handy for all drivers as not everyone will be savvy enough to understand the inner-workings of a vehicle. Unlike other vehicles with on-board diagnostics, we wish to inform the driver of the problem, possible solutions and urgency of said problem without a special reader. Though regulation may require us to have OBDII for now, we will still implement our own diagnostic system in conjunction with OBDII to collect vehicle metrics and determine abnormalities. If the vehicle is parked and there is a possibility of self-repair, the vehicle will attempt to do so when given permission. If self-repair is not feasible, the driver will be informed of the problem and be asked to take the vehicle to the nearest mechanic/dealer. Logs will be sent to use so we can determine error rates and so we can take steps to improve our systems.
10. **Internet integration**
    a. As this vehicle will use IoT, internet integration is extremely important in order to facilitate our previously mentioned features and keep our software updated. Using an embedded chipset and antenna allows for the vehicle to communicate with us to receive updates, diagnosis information, road/weather information and allow the driver to connect the internet for services like Spotify.

Section 2: Functional Architecture

Diagram:



2.1 Diagram Explanation

- When the vehicle is managing itself, whether in an auto-summon, lane mitigation, or driving autonomously, the blue side of the graph (sensor fusion) it activated to pick up the precise location of the vehicle and what it is doing, where the yellow side of the graph (electronics and software) provides the means to keep the vehicle in its action. For example, in auto-summoning, the sensor fusion displays the vehicles location in actuality to it's person and relays that to the software which will navigate the vehicle in

the right direction, using sensors to avoid any obstacles. Together, sensor fusion and the software allow for the access of system management and vehicle control by the driver.

2.2 Localization

- GPS
    - Built specifically for Hug the Lanes, puts a greater weight on taking routes through high density roadways than through neighborhoods and backstreets to minimize traffic in those areas. Allows for greater flexibility in map routing.
- On-board Sensor
    - Detects passengers onboard and sends information to make sure the vehicle never makes any maneuvers that might cause harm to any of them.
- Inertial measurement unit
    - Perceives the rotational momentum and force of the vehicle and can detect when rates increase to dangerous levels as well as prevent them.

2.3 Perception

- Laser Sensors
    - Uses lasers to create virtual representation of space around the vehicle. Used in self-driving, auto-parking, and summoning.
- Cameras
    - Rearview, Front View, Blind spot detection, and Virtual side Mirrors.

2.4 Electronics and Software

- Electrical Architecture
    - Convergence of electrical hardware such as solar panels and dashboard touchscreen with software that ties everything together.
- Electrical Control System
    - The control unit of the vehicle. It’s responsible for all the other electrical components and lets them run as intended.
- Sensors/Actuators
    - The main reason the vehicle’s software has visualization capabilities.
- Telecommunication
    - Telecommunications and Internet Connections within the vehicle allows for the vehicle to receive software updates, also allowing the vehicle to self-diagnose issues and attempt to repair or contact for repairs. Lastly, telecommunications will alert emergency contacts and road assistance if an accident or emergency occurs with the vehicle. Utilizes network and cloud connection.
- Software
    - Handles driver input and sends it to vehicle control.

2.5 Sensor Fusion

- Combination of perception and localization in order to give the vehicle an accurate understanding of its surroundings.

2.6 System Management

- Collects information & data logs of systems inside the vehicle and sends them to Alset.

2.7 Vehicle Control

- Combines Sensor Fusion, Perception, Electronics & Software and System Management to decide the best course of action in different parts of the vehicle such as:
    - Braking and Steering - Allows the vehicle to stop or turn if it is needed, such as for sensing a vehicle in front of it going slower than it's current speed or that the road ends ahead and the vehicle must change its current trajectory.
    - Exhaust, Suspension, and Tires - Allows the vehicle to be able to tell if anything is going wrong with these systems, and will warn the driver or stop the vehicle if help is required.
    - Fuel - Allows the vehicle to check the amount of charge left in the battery and inform the driver if it is low and needs to be recharged.

2.8 HUD

- Uses Vehicle control and System Management to give driver information about the drive, such as road conditions and the state of the vehicle's battery.
- Can also display side view cameras to allow the driver to see blindspots.

2.9 Vehicle Driving System

- The combination of all of the features we listed, the end goal of a fully self-driving vehicle.
- Flexible Map Routing, Lane mitigation, HUD, Virtual Side Mirrors, Solar Panels, AISensors, Auto Parking/Auto Summon, Automatic Emergency Contact, Self-Diagnosis/Repair, Internet Integration, Self-driving capability.

Section 3: Requirements

3.1 Functional Requirements

3.1.1 Engaging the Cruise Control

Pre-conditions

- The vehicle must not be in park, neutral, or reverse
- The driver cannot have their foot on the brake pedal
- The vehicle's current speed is not over 90 mph or under 30 mph
- Self-Diagnostics detect no fault in the cruise control systems
    - Blown Fuse
    - Faulty Brake Pedal/Speed Sensor/Control Switch
- The set speed must be within 15 mph of the vehicle's current speed
- The speed sensor must confirm the current speed of the vehicle

Post-conditions

- The vehicle continues to cruise at the inputted speed, until compression from the brakes is received.

3.1.1.1

- The driver engages cruise control, the button sends the inputted speed to Sensor Fusion

3.1.1.2

- Sensor Fusion sends speed request (1) to IoT HTL for cruise control.

3.1.1.3

- IoT HTL continually sends speed requests to the Vehicle Control System.

3.1.1.4

- IoT HTL sends a request to turn on the cruise control display on the HUD.

3.1.1.5

- IoT HTL logs the event in System Management and the vehicle continues to cruise at the speed given in the request.

3.1.2 Disengaging the Cruise Control

Pre-conditions

- Vehicle is driving with Cruise Control engaged. Ensured by the usage light on HUD. IoT is sending continual speed requests to the Vehicle Control System.

Post-conditions

- The cruise control disengages and the light turns off. The driver regains speed control of the vehicle.

3.1.2.1

- The driver compresses the brake, vehicle control sends a 0 signal to IoT HTL for sensor fusion to disengage the cruise control.

3.1.2.2

- Sensor fusion sends back a signal to vehicle control to turn off the cruise control display from the HUD.
- Sensor fusion stops sending signals to navigate the vehicle at the set speed *s*, allowing the driver to regain control of speed.

3.1.3 Flexible Routing

Pre-conditions

- Engine must be running but the vehicle must be stopped to ensure undistracted driving.
- Must be in a location where internet and location services are available.

Post-conditions

- Informs the driver that they have arrived at the destination.

3.1.3.1

- Driver uses touch screen or app.

3.1.3.2

- The IoT engine communicates with our servers to determine the optimal route.
- Planning constantly references GPS and map data to dynamically optimize routes depending on road conditions.

3.1.3.3

- When the GPS matches the selected location, it displays to the driver on the HUD or app that they have arrived at their destination.

3.1.3.4

- IoT HTL logs the event in System Management.

3.1.4 Lane Mitigation

Pre-conditions

- Vehicle must be driving on a road with visible lane markings.
- Turn signals must be disengaged.

Post-conditions

- If the steering wheel is turned, the system disengages to avoid over-alerting or or over-turning the steering wheel, causing the vehicle to cross over lane markings.

3.1.4.1

- Cameras send an alert to Perception that lane markings are in sight.

3.1.4.2

- Perception sends a request to sensor fusion to alert the driver with a sound.
- Sensor Fusion passes information to Planning.

3.1.4.3

- If continuous alerts of the lane markings are too close to the vehicle, Planning directs the driver's vehicle control and shifts the vehicle back into the lane.

3.1.4.4

- IoT HTL logs the event in System Management.

3.1.5 HUD

Pre-conditions

- The vehicle is on.

Post-conditions

- The HUD is turned, allowing the driver to see the information provided by it.

3.1.5.1

- HUD communicates with Self-Diagnostics to alert the driver about any issues and display options regarding self-repair.
- Display to driver a message that that vehicle should be taken to the repair center if the self-repair attempt failed or no self-repair options are available.

3.1.5.2

- Communicate with systems such as VCS and Electronics/Software to relay information to the driver. Includes information such as speed, battery charge, and more.

3.1.6 Virtual Side Mirrors

Pre-conditions

- The vehicle is on.

Post-conditions

- The virtual side mirrors are accessible by the driver, and are able to be used as a blind spot detector.

3.1.6.1

- When the vehicle is on, cameras activate and send their feed to Sensor Fusion.
- Cameras continually send their live feed to Sensor Fusion, so the driver is never put into an unsafe position.

3.1.6.2

- Sensor Fusion sends the live camera feed to Vehicle Control through Planning.

3.1.6.3

- Vehicle control sends the feed to System Management, where it can be viewed directly on either side of the HUD, and can be moved with the side mirror joystick.

3.1.7 Solar Panels

Pre-conditions

- Vehicle is turned off.
- Self-Diagnostics detect no fault in the Solar Panel System.
- The driver has not specified that they do not want the solar panels to activate.
- Solar panels are in sunlight ranges.
- The vehicle is not currently charging via its charging port.

Post-conditions

- The vehicle gains *X*% of charge from the solar panels.

3.1.7.1

- When pre-conditions are met, the vehicle communicates with the IoT engine to determine weather conditions and determine if solar panels would be effective.
- Informs driver about current weather conditions if they are bad and confirms with the driver if the solar panels should be exposed via the hud.

3.1.7.2

- Planning tells electronic and software modules to expose the solar panels to the sky and collect energy.

3.1.7.3

- IoT HTL logs the event in System Management.

3.1.8 AI Sensor

Pre-conditions

- Self-Diagnostics detect no fault in any of the sensors.

Post-conditions

- Relays sensor information to necessary modules.

3.1.8.1

- While the vehicle is on, sensor data is recorded and sent to the Sensor Fusion module.

3.1.8.2

- Sensor Fusion will collect sensor data and the localization and perception modules will pass this information to the Planning Module.

3.1.8.3

- IoT HTL logs the event in System Management.

- This process repeats itself while the vehicle stays in use to ensure the AI sensors are always on unless they are functioning improperly.

3.1.9 Auto Parking/Summon

Pre-conditions

- Driver has enabled auto summon via app or keys or has turned on auto parking.
- For auto summoning, the vehicle must have the ability to find the location of the driver using our app or keys.
- For auto parking/summoning, the vehicle must be at a stop.
- Driver is inside the vehicle and does not have a foot on the accelerator pedal.
- Self-Diagnostics detect no fault in the auto park/summon systems or sensors.

Post-conditions.

- Vehicle has come to a complete stop at the designated location.

3.1.9.1

- Auto park request is sent to Planning.
- Keys or App will send auto summon requests to Planning.

3.1.9.2

- Planning will take the auto park request and use information from sensor data alongside the IoT engine to determine valid parking spots.
- Planning will take the auto summon request and use information from sensor data alongside the IoT engine to determine the driver's current position or designated summon location.
- Vehicle speed will not exceed 5 mph.

3.1.9.3

- VCS will manipulate the vehicle's speed, wheel angle and more to drive the vehicle into the valid parking spot or driver’s summon location.

3.1.9.4

- Planning and VCS will tell the vehicle to come to a stop and relay to the driver that the auto park/summon has been completed in either the HUD or app.

3.1.9.5

- IoT HTL logs the event in System Management.

3.1.A Self-diagnosis/Repair

Pre-conditions

- None.
- This feature will run in the background regardless of whether the driver is driving the vehicle or not.
- Feature will run unless the driver decides to turn the vehicle completely off.

Post-conditions

- Vehicle will either attempt to do vehicle repair and/or inform the driver of the diagnosis and to bring the vehicle to a repair shop. Create a log for repair attempt and diagnosis in System Management.

3.1.A.1

- Reads logs from System Management and will check any abnormalities or errors in various vehicle systems.
- Reports and reads data from modules if no log is available to read.
- If at any point, a system reports an error, Self-Diagnosis will immediately begin to assess the log and the error.
- If logs or data don't have any errors, repeat the process of checking other systems.

3.1.A.2

- If logs or data do have errors, determine the corresponding error codes and communicate with our servers for possible repair solutions using Electronic/Software.
- Informs driver on the HUD. Gives them the error code, brief description and depending on availability, will ask if the vehicle can perform self repair.

3.1.A.3

- If repair was selected, it will inform the driver to drive the car to a stop at a safe location and once stopped. Then Planning will perform the necessary repair steps using information from our servers once the car has come to a stop. If this fails, inform the driver and suggest taking the vehicle to a repair center.
- If repair was selected or is unavailable, inform the driver to take the vehicle to a repair center.

3.1.A.4

- Keeps the error message on the HUD until the problem is fixed.
- IoT HTL logs the event in System Management.

3.1.B Vehicle Drive System

Pre-conditions

- Self-Diagnostics detect no faults in all other systems and modules
- Driver has enabled self driving via a button or HUD.
- Driver has selected a destination.

Post-conditions

- Vehicle navigates itself from point A to point B in a safe and efficient manner.

3.1.B.1

- Grabs location requested from driver.

- Repeats location requests constantly in order to make sure the drive system is accurate.

3.1.B.2

- Communicates with Flexible Map Routing module to determine path.
- Gets traffic data from various possible routes to the destination and calculates which path results in the least amount of overall trip time, as well as whether or not the route contains tolls.
- Sends all the information and information from Flexible Map Routing to Planning.

3.1.B.3

- Communicates with Planning to drive the vehicle.
- Utilizes other functional requirements and processes previously integrated in order to effectively drive the vehicle to its destination in a safe and timely manner.

3.1.B.4

- Communicates with the Auto Park module to park the vehicle.
- Once securely parked, Flexible Map Routing, Line Mitigation, and other functionally used software is told to turn off in order to preserve battery life by limiting the running programs to only the necessary ones.

3.1.B.5

- IoT HTL logs the event in System Management.

3.2 Non-Functional Requirements

3.2.1 Performance

3.2.1.1

- All response and execution times will not take longer than 10 ns.

3.2.1.2

- IoT HTL will be able to process a thousand requests per second.

3.2.2 Reliability

3.2.2.1

- CC does not fail more than once per 300,000 hours during operation.

3.2.2.2

- Processors will not fail more than 3 minutes per year.

3.2.2.3

- Each System will not fail more than once per year.

3.2.2.4

- Total System Failure will be impossible excluding crashes or unforeseeable events.

3.2.3 Security

3.2.3.1

- Access to IoT HTL will be secured by driver ID passwords.

3.2.3.2

- Data sent through IoT HTL will be encrypted to protect against cyber attacks on the vehicle.

3.2.3.3

- App communicates with vehicle on an encrypted network. Outside drivers will not be able to eavesdrop or manipulate communication lines.

3.2.4 Software Update

3.2.4.1

- Software update by cloud services will make sure the vehicle is always up to date, so long as the connection is stable.

3.2.4.2

- Software updates will happen automatically as long as the vehicle is not being used.

3.2.4.3

- Software updates will be done with encryption. Update will have a verifiable signature to guarantee that the vehicle will not download a rogue update.

3.2.5 Network

3.2.5.1

- Connects to the IoT and allows the rest of the vehicle to function.

3.2.5.2

- Network must be reliable to facilitate software updates and vehicle features.

3.2.5.3

- Network must be fast to run our systems and ensure drivers can enjoy consistent access to the internet.

3.2.5.4

- Network must be encrypted.

3.2.6 driver Interface

3.2.6.1

- Usability for the driver.
- Will not have too much abstraction so that the driver can access permitted information to assess and fix software and hardware issues should they arise.

3.2.6.2

- UI must be intuitive. Buttons must be labeled properly and sized & spaced for efficient usage.

3.2.6.3

- UI must be reactive and responsive. Needs to be clean, sleek and quick.

3.2.7 HUD/Display

3.2.7.1

- Used by the driver to see insights about the vehicle, but without means of distraction.

3.2.7.2

- Needs to be non-invasive. Needs to not distract or block the driver's view of the road.

3.2.8 Hardware

3.2.8.1

- Physical interface for technician access.

Section 4: Requirement Modeling

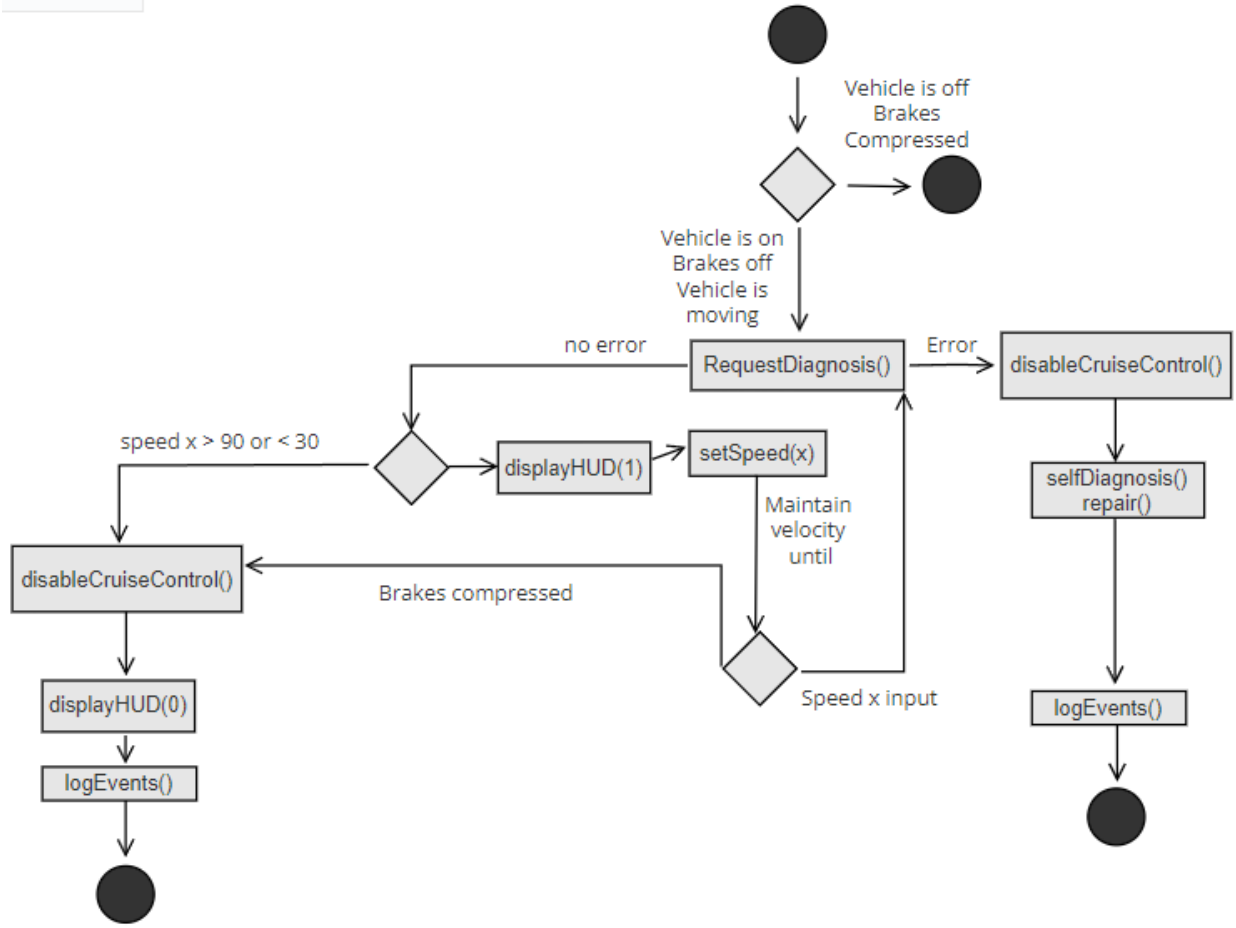
4.1 Cruise Control

4.1.1 Use Case - driver Sets Cruise Control

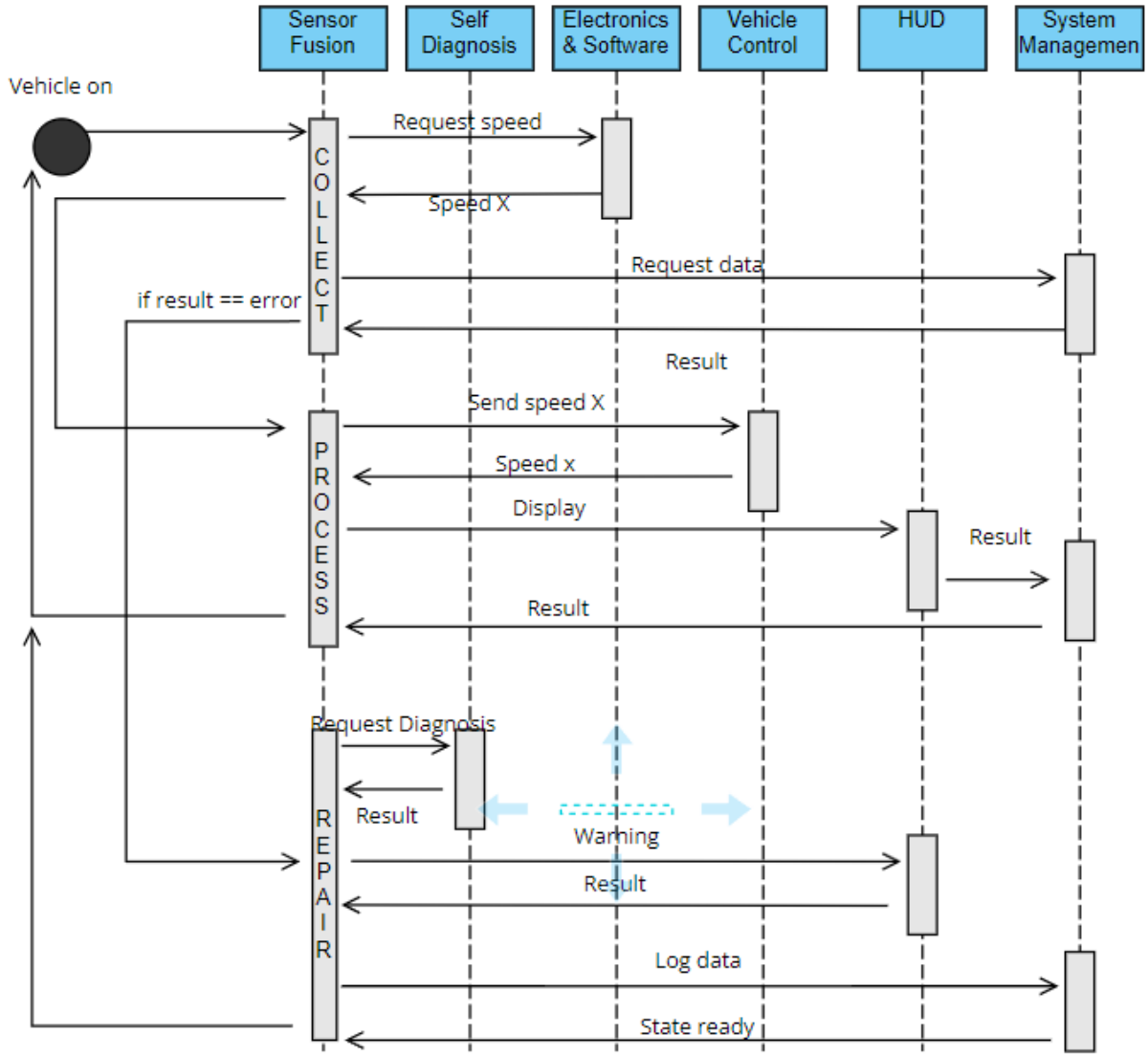
- Pre-Condition: Vehicle is on, moving, and break is not being compressed.
- Post-Condition: Vehicle moves at set cruise control speed. HUD Display shows that cruise control is in use.

1. The driver presses the Cruise Control button in the vehicle.
2. Inputted speed x is requested from Electrictronics and Software to Sensor Fusion.
3. The Internet of Things (IoT) receives the request from Sensor Fusion, which prompts it to send a request for x speed to the Vehicle Control System.
4. The Vehicle Control System turns on the Cruise Control display with the current speed for the driver to see.
5. IoT continually sends requests to drive at speed x until the speed is changed or cruise control is disengaged.
6. All events are logged in System Management as to if the system is functioning properly
    a. If not, an alert is sent to the driver through the HUD and cruise control will not activate until self-diagnosis/repair is done.
    b. If yes, the cruise control will remain on at speed x.
7. The vehicle cruises at speed x safely.

4.1.2 Activity Diagram



4.1.3 Sequence Diagram

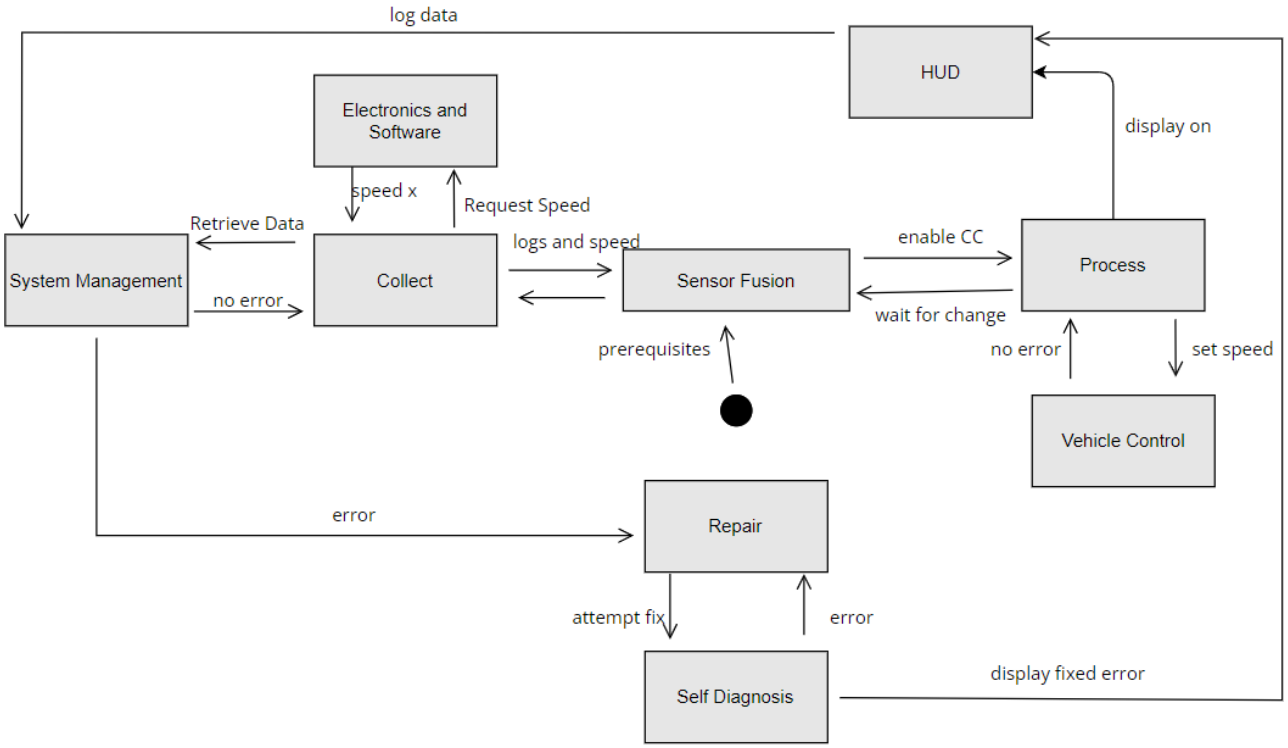


4.1.4 Classes

- Collect
  - Retrieves the inputted speed x from electronics and software.
  - Retrieves system management data to ensure that the last log on Cruise Control was working correctly
  - Proceed to processes if system management's result isn't an error, otherwise call on repairs.
- Process
  - Sends out speed x to vehicle control to result in the car achieving that constant velocity.
  - Displays to HUD that Cruise Control is active with the current speed.
  - All events are logged in system management, and the process repeats, waiting for a brake compress or change in speed.

- Repair
    - Alerts the driver with a HUD warning that Cruise Control failed and self-diagnosis is in process
    - If function is fixed, the car becomes operational and returns to the collect class operationally.

4.1.5 State Diagrams



4.2 Auto Park/Summon

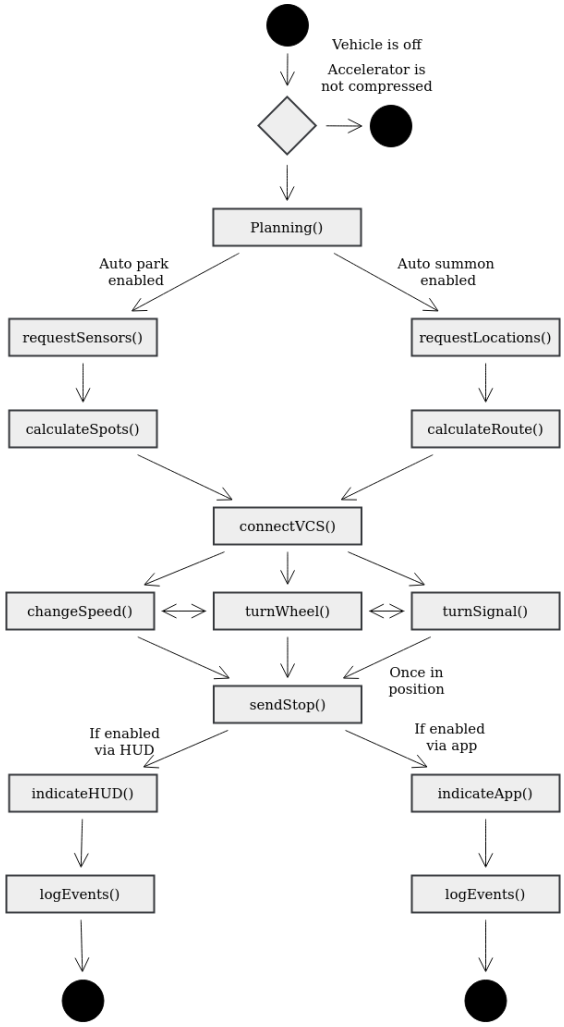
4.2.1 Use Case

- Pre-Condition: Vehicle is stopped, auto park/summon is enabled, and accelerator is not being compressed.
- Post-Condition: Vehicle is completely stopped at the decided location.

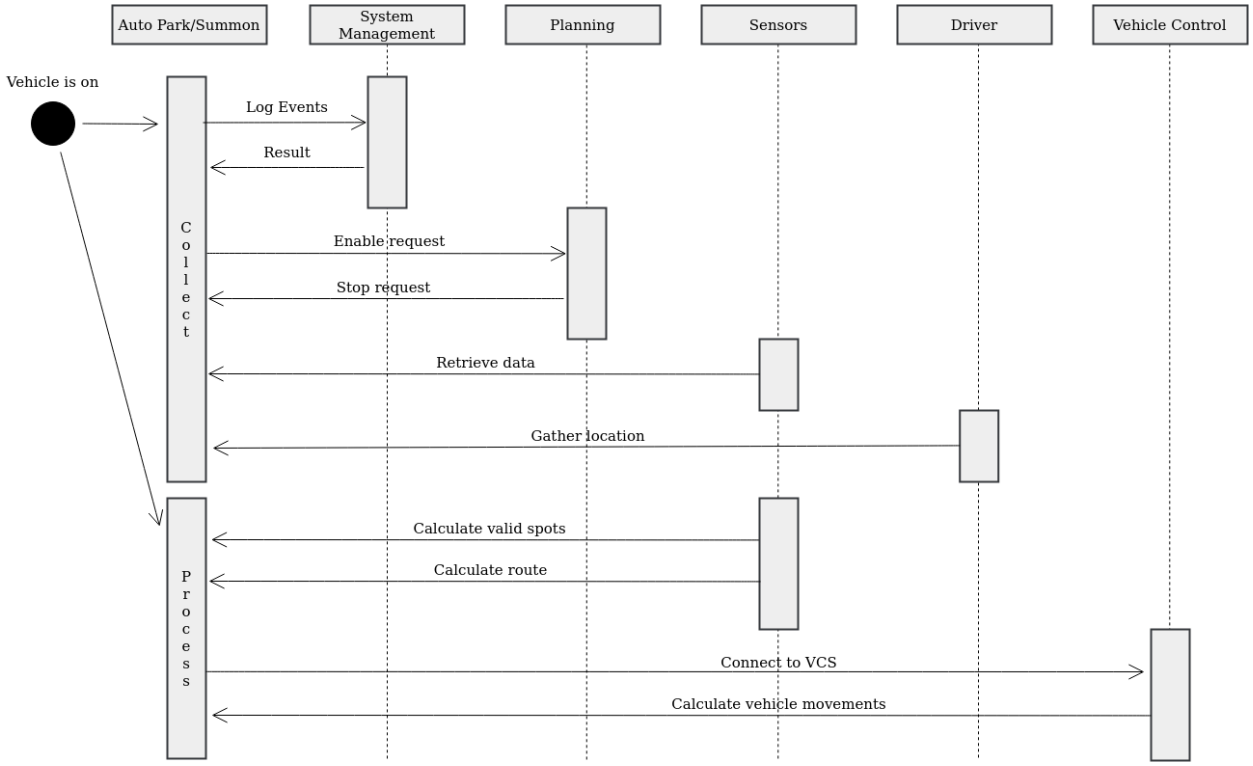
1. Driver requestSensors auto park/summon.
2. The keys or the app sends request to Planning.
3. Planning takes request and gathers data from sensors along the IoT engine.
4. Calculates which parking spots are valid via sensors.
5. Calculates where the driver is currently located or where the summon is located.
6. VCS manipulates speed of the vehicle, wheel turning, etc. in order to guide the vehicle into its designated spot or location.
    a. Vehicle speed will not exceed 5 mph

7. Once in position, VCS and Planning will send a signal to the vehicle to stop and will indicate to the driver (HUD or app respectively) that the auto park/summon has finished.
8. All event data is then logged into System Management.

4.2.2 Activity Diagram



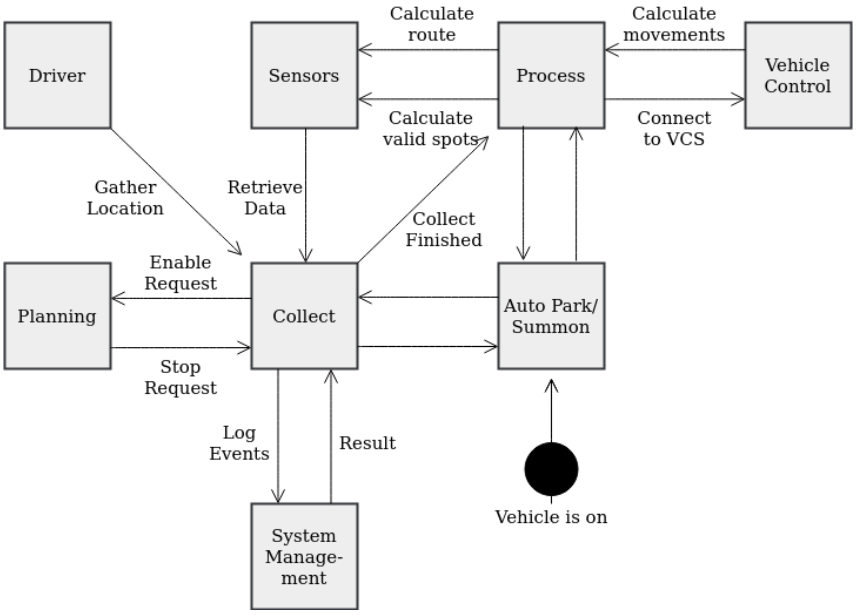
4.2.3 Sequence Diagram



4.2.4 Classes

- Collect
    - Receives request that auto park/summon is enabled.
    - Retrieves sensor data from the IoT engine.
    - For auto summon, gather where the driver's and vehicle's location is.
    - Start with Process unless locations are unable to be acquired or no parking spots.
    - Log event data to System Management.
- Process
    - For auto park, calculate which parking spots are valid for the vehicle to park in.
    - For auto summon, calculate the best route to return the vehicle to the driver.
    - Connect to VCS and process correct and accurate movements of the wheel and pedals.
    - When the Process is finished, proceed to Store unless an error is thrown.

4.2.5 State Diagrams



4.3 Self-Diagnosis/Repair

4.3.1 Use Case

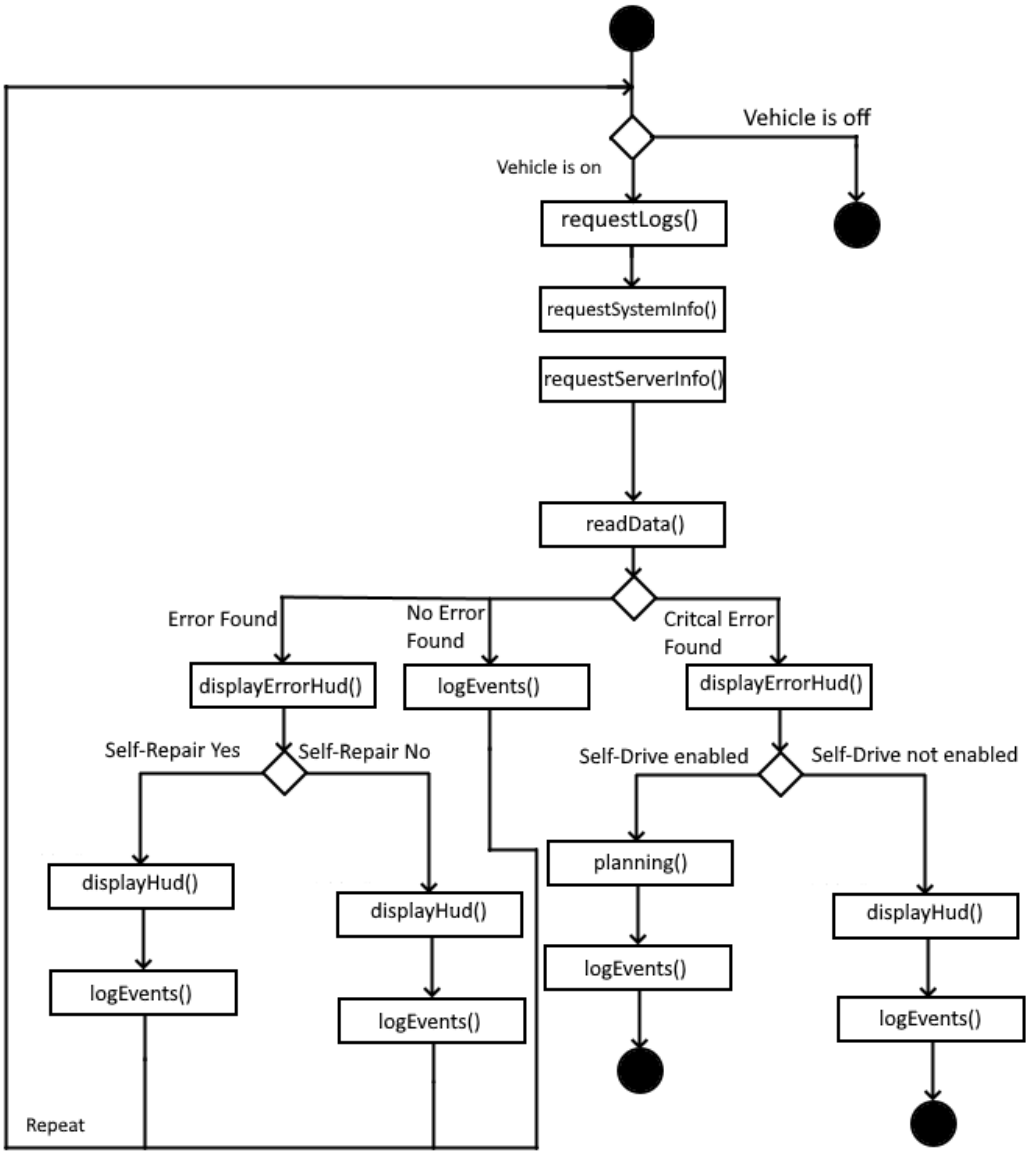
- Pre-Condition: Vehicle is on.
- Post-Condition: Self-Diagnosis has read logs and data from other systems. Has attempted self-repair if the driver selected yes. Display message to driver on HUD.

1. Self-Diagnosis requests logs from Systems Management for system x.
2. Self-Diagnosis requests information from system x.
3. Requests technical information about system x from Electronics and Software.
4. Read the data or logs and detect any abnormalities in conjunction with server information.
5. If critical failure, send a request to HUD to display to the driver to stop the vehicle in a safe location as soon as possible.
    a. Display Error message and inform Driver to ask for assistance once vehicle has stopped.
    b. If the vehicle is self driving, send a request to HUD to display to driver an Error Message and attempt to stop in a safe location (Ex:shoulder of a highway).
    c. Log events
6. If an error is detected, send a request to HUD to display an Error Message that informs the Driver of a possible error and depending on server information, the vehicle may be able to self-repair. Display a choice of yes or no for the vehicle to self repair.

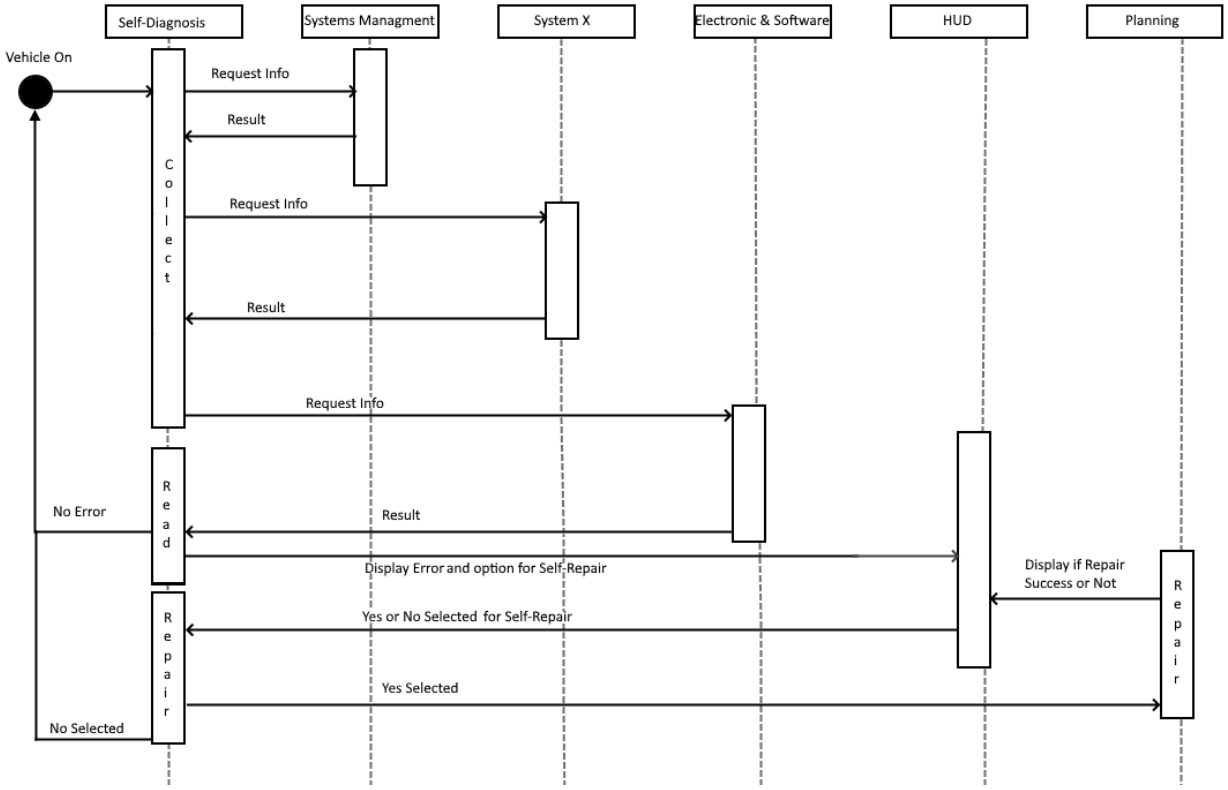
a. If yes, send a request to planning and any necessary server information to Planning.
    i. Planning will attempt to follow server instructions and attempt self-repair. Send a request to HUD displaying information if repair was successful or not.
    ii. Repeat the process for every other system.
b. If no, display an error onto the HUD. Display to driver that vehicle should be looked at by a mechanic.
c. Log events and repeat the process for every other system.

7. If no error is detected, log events and repeat the process for every other system.

4.3.2 Activity Diagram



4.3.3 Sequence Diagram

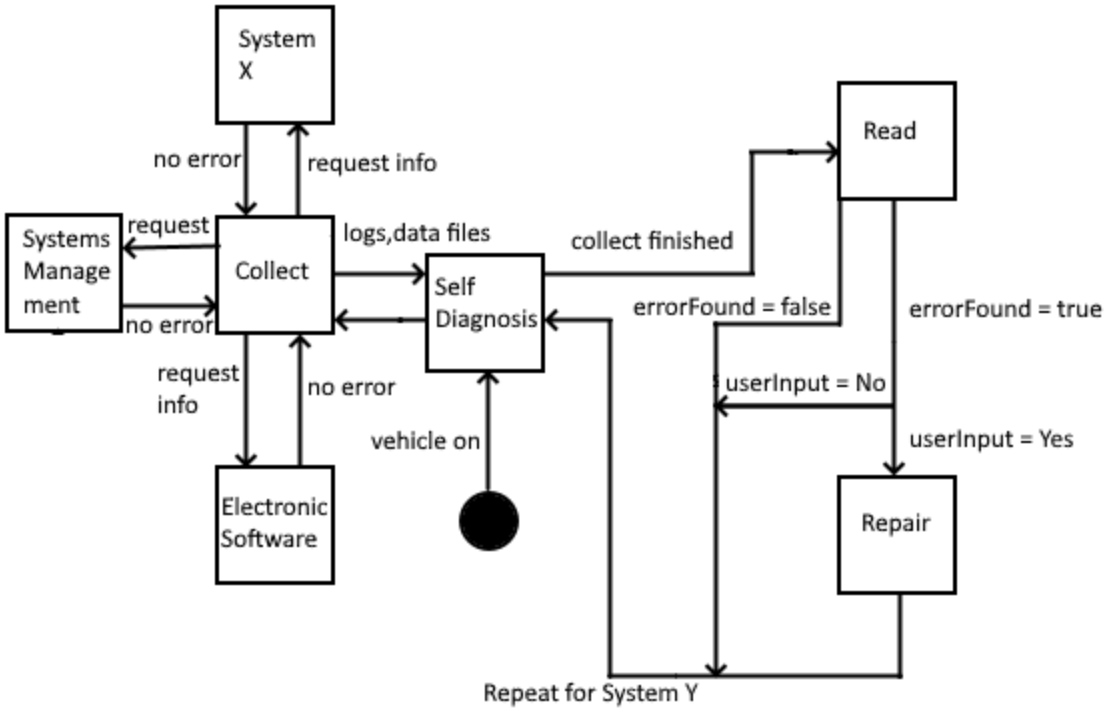


-

4.3.4 Classes

- Collect
  - Retrieves information on system X through logs from Systems Management and directly from system x.
  - Retrieves technical information from our servers about system X.
  - Proceed with Read unless an Error was thrown out somewhere.
- Read
  - Takes logs and information from System X and reads data.
  - Detects and flags if an error has been found.
  - Display error message and prompts driver.
  - Proceed with Repair.
- Repair
  - Depending on driver input, will send requests to Planning to continue with the repair request.

4.3.5 State Diagrams



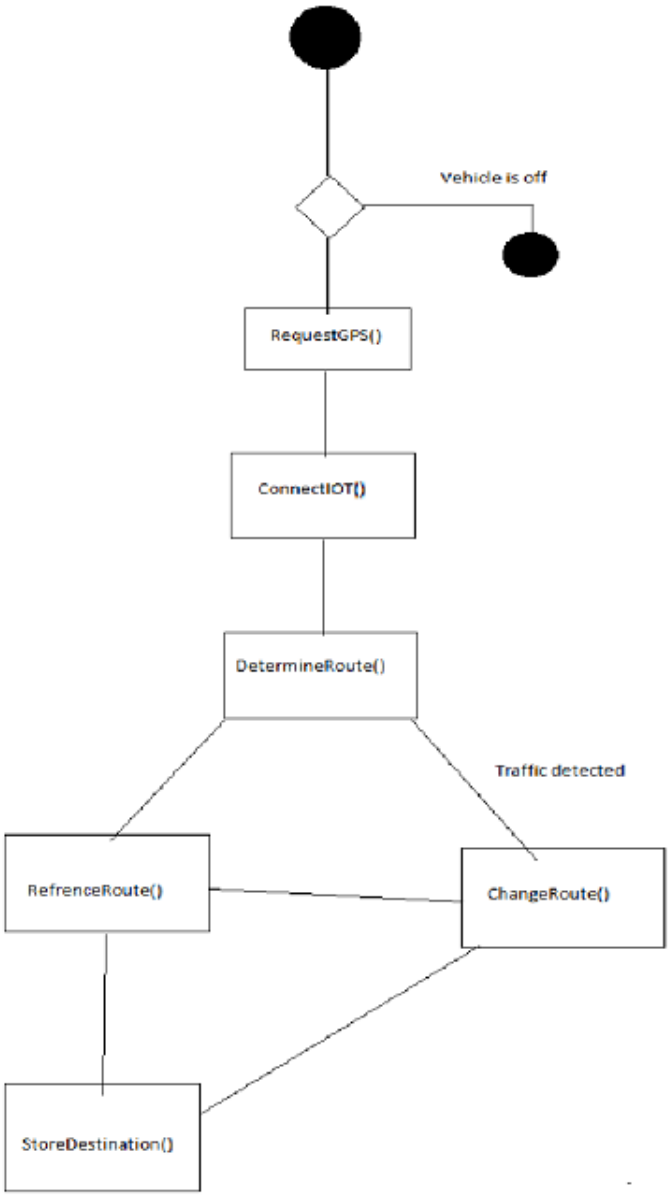
4.4 Flexible Routing

4.4.1 Use Case

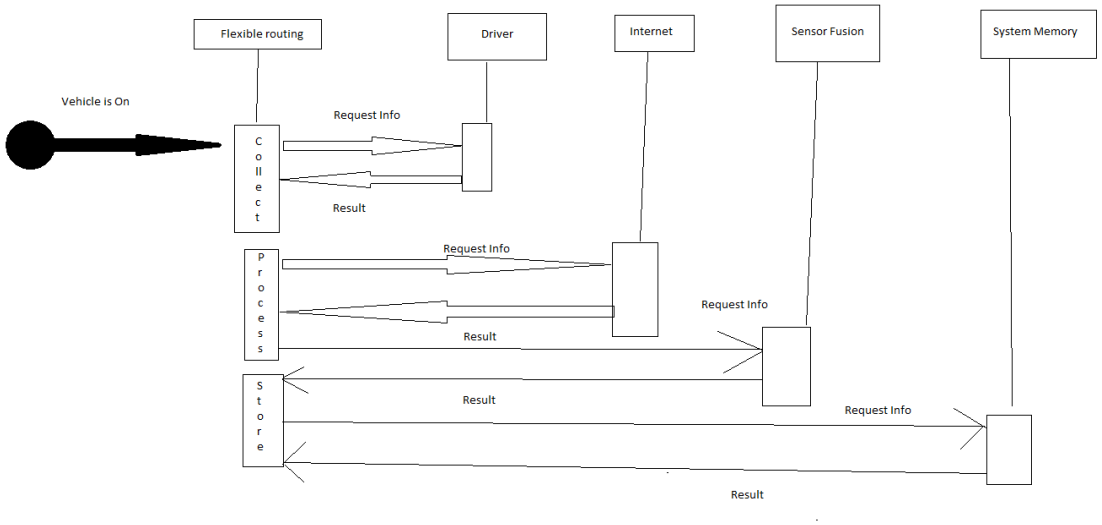
- Pre-Condition: Vehicle is turned on, and has a stable internet connection.
- Post-Condition: Informs the driver they have arrived at their destination.

1. The driver uses the touch screen or app to request the routing feature.
2. Connects the routing system to the IoT.
3. Determine the fastest possible route to the destination.
    a) If there is traffic, inform the driver and change path accordingly.
4. The system will continue to reference the path during the drive, and if conditions change midway through, the path will be changed accordingly.
5. Driver is informed that they have arrived at their requisition destination.
6. Data is logged in the system memory, in case the destination is needed again.

4.4.2 Activity Diagram



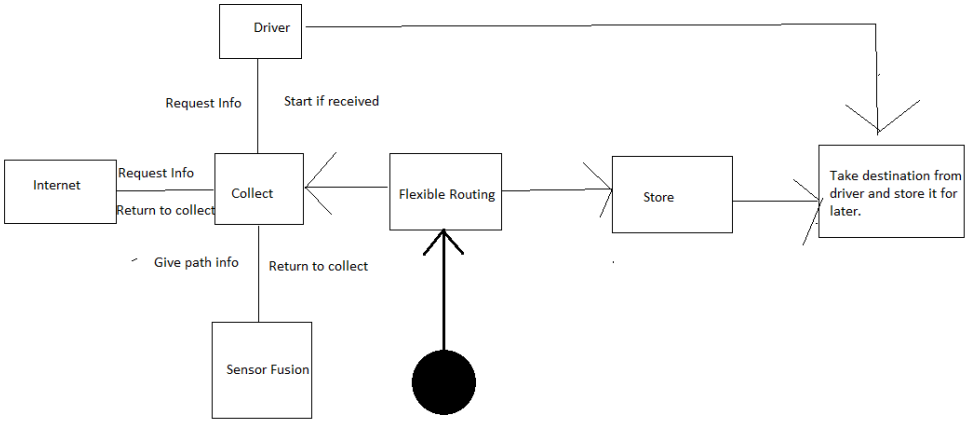
4.4.3 Sequence Diagram



4.4.4 Classes

- Collect
  - Retrieves intended destination from the driver.
  - Receives the data on possible paths from the internet
  - Proceeds to process unless an error is raised.
- Process
  - Determines the fastest possible path, and sends that to sensor fusion to take that path.
  - When the path changes, notify Sensor Fusion to change the path of the car.
  - When the path is finished, proceed to Store unless errors arise.
- Store
  - Stores the destination into system memory in case the driver would like to go to the same destination again.

4.4.5 State Diagrams



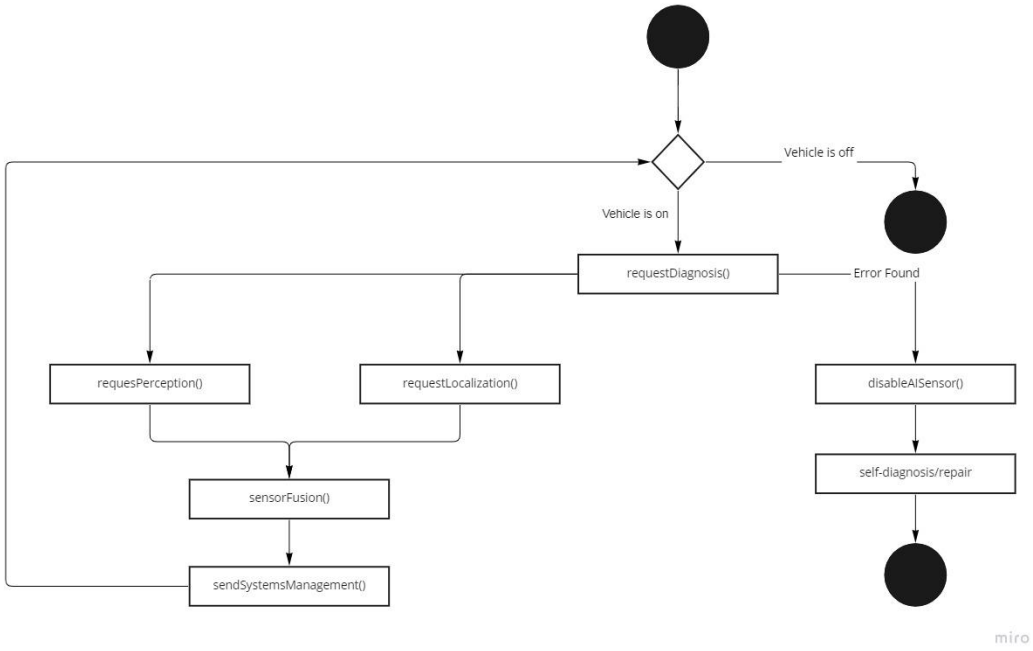
4.5 AI Sensor

4.5.1 Use Case

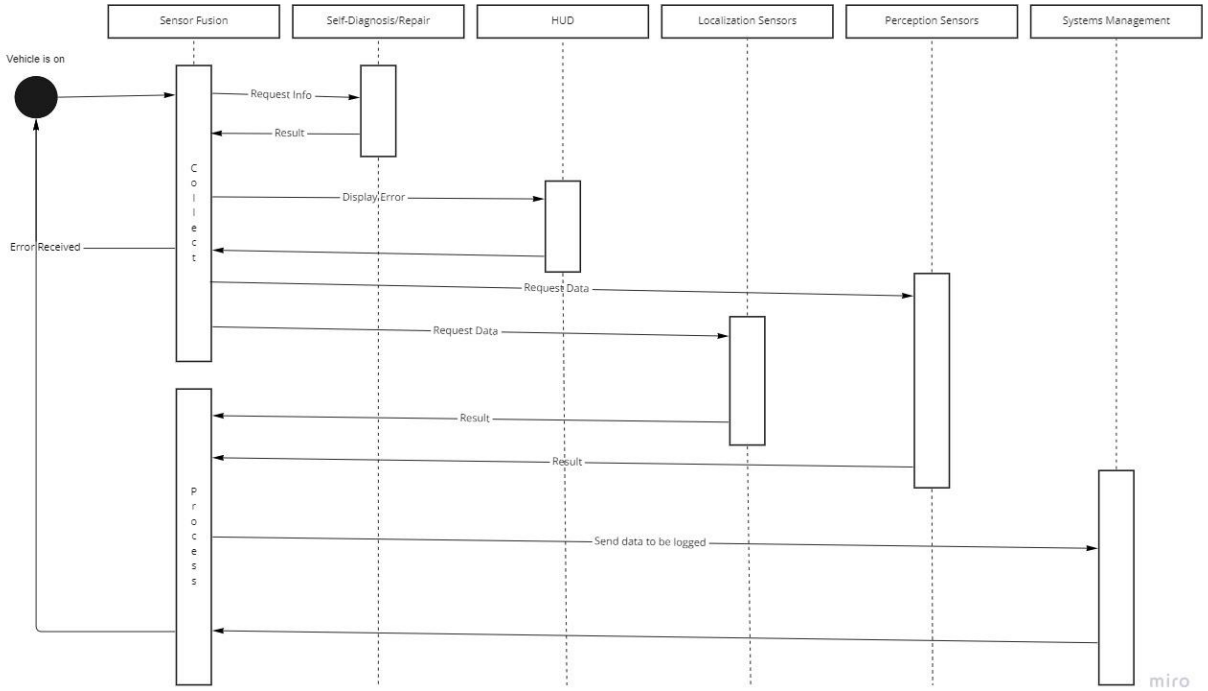
- Pre-Condition: The car is running and Self-Diagnostics detect no fault in any of the sensors.
- Post-Condition: Retrieves information accumulated by sensors and relays them to the appropriate modules.

1. Sensor Fusion requests a self diagnosis of each sensor in both localization and perception.
    a. If there is an error, display an error message on the HUD to alert the driver to follow the self-diagnosis/repair procedure and disable the AI sensors.
2. Sensor Fusion requests data from localization sensors and perception sensors
3. All sensors record their information and send it back to the localization module or perception module which in turn send it to the Sensor Fusion
4. Sensor Fusion combines the information given into a single format to send off to the planner
5. Sensor Fusion sends event information to the Systems Management for it to be logged for later use.
6. The process is repeated until the car is no longer in use and is turned off.

4.5.2 Activity Diagram



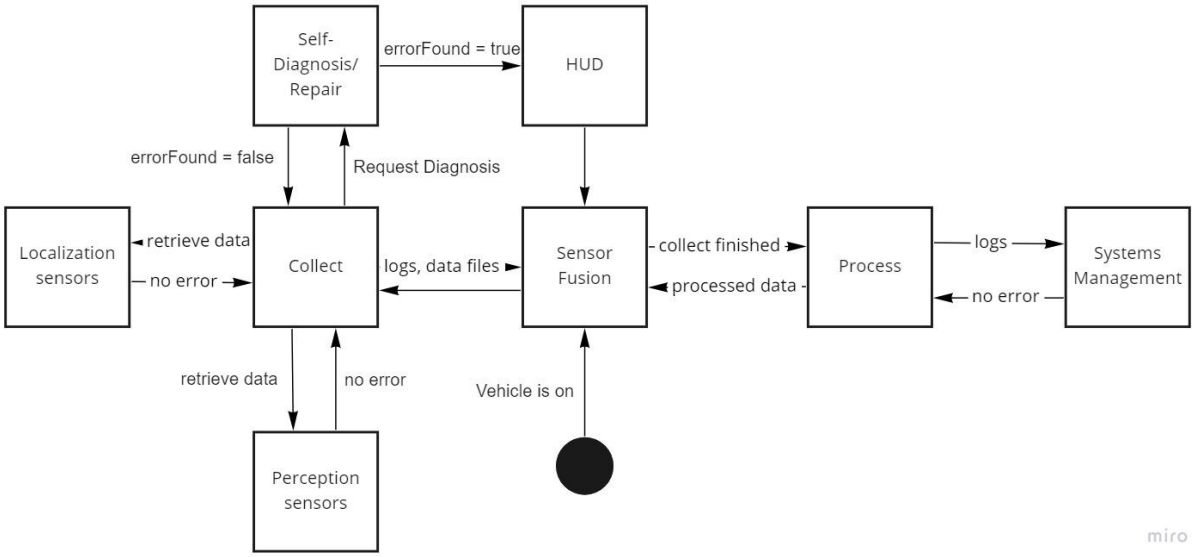
4.5.3 Sequence Diagram



4.5.4 Classes

- **Collect**
    - Retrieve error diagnostics and process result
    - Retrieves sensor data from localization sensors and perception sensors to send to Process class
    - Proceeds to Process unless an error was raised
- **Process**
    - Processes result of localization and perception sensors and is picked up by planner when sensor fusion is called
    - Sends log data to Systems Management

4.5.5 State Diagrams



Section 5: Design

5.1 Software Architecture

5.1.1 Data Centered Architecture

- What is Data Centered Architecture
    - In a Data Centered Architecture, data is centralized and continually accessed by other components within the system. These components also have access to modify data within the system. This architecture focuses heavily on ensuring integrality of data.
- Pros and Cons
    - Pros
        - Reduces short term memory between the software components.
        - Centralized management allows for Sensor Fusion to control all aspects of IoT.
        - Allows for data to be stored efficiently in System Management.
    - Cons
        - Providing updates to certain softwares will be difficult because all of the data systems are connected.
        - Depends heavily on the data being stored correctly and all components also cooperating with one another.
        - May have issues synchronizing all data from multiple components.
- Suitability
    - Because Data Centered Architecture has such a big risk sharing all data to all components, it is not fit for our IoT infrastructure. Our software needs to be able to manage anything the driver might incur. If there are data inconsistencies or vulnerabilities in our system, it would work negatively in our software layout.

5.1.2 Data Flow Architecture

- What is Data Flow Architecture
    - In Data Flow Architecture, the software is seen as a series of transformations done on data. Input data is transformed into output data which is then connected to a pipe. Pipes transfer data from component to component which are called filters. Filters expect a certain type of input and outputs the data into a type that the next filter will expect. Filters work independently of each other and do not require the knowledge of the inner-working of neighboring filters.

- Pros and Cons
    - Pros
        - Allows for lots for data to be processed.
        - Allows for simpler system maintenance.
        - Allows for filters and pipes to be be reused for multiple tasks.
        - Allows for clarity as it shows the distinction between filters.
        - Allows for sequential and parallel execution.
    - Cons
        - Not suitable for dynamic interactions.
        - Difficult to turn architecture into a dynamic one.
        - Filters operate independently, meaning no built-in way for them to work together.
- Suitability
    - Based on the information given, Data Flow Architecture is not suitable for IoT HTL as we need software to be dynamic and responsive to situations on the road. Filters being unable to cooperate is not an acceptable downside for our software.

5.1.3 Call Return Architecture

- What is Call Return Architecture
    - In Call Return Architecture, the software is seen as a program divided into different subprograms. These controller subprograms are then divided further into application subprograms. Essentially, it's a simple divide-and-conquer algorithm for an architecture process.
- Pros and Cons
    - Pros
        - Allows for easy split-up of work.
        - Allows for simple debugging, letting you always know what is wrong due to the split-ups.
        - Allows for both sequential and parallel execution.
        - High clarity - Easy to tell what is doing what.
    - Cons
        - Very hard to go back once a sub-program has been created.
        - Can be hard to merge programs into one at the end.
- Suitability
    - This architecture could be suitable for IoT HTL. Our design is similar to a lot of different systems being merged into one big project at the end, and it would be easy for us to merge the sensors into sensor fusion, all the GPS systems into dynamic routing, etc.

5.1.4 Object-Oriented Architecture

- What is Object-Oriented Architecture

    - Object Oriented Architecture divides the responsibilities and processes of the application into individual and reproducible objects that can be quickly instantiated.
- Pros and Cons
    - Pros
        - Repetitive use allows the architecture to be easily tested and maintained.
        - Allows for high levels of abstraction.
        - Implementing new features is not invasive to the rest of the application.
        - Reduces development time and cost.
    - Cons
        - Determining the necessary objects and classes is difficult.
        - Successful reuse on a large scale is improbable without an explicit reuse procedure.
- Suitability
    - Object-Oriented Architecture could be a suitable fit for IoT HTL due to the many processes and individual features of the system. This architecture allows for dynamic communication between objects and piping information between them is supported. All of the operations of the system can be implemented as objects easily since they are already planned.

5.1.5 Layered Architecture

- What is Layered Architecture
    - Layered Architecture is where each component of the project is put into different categories of abstraction. Components closer to the driver layer is for the driver, and components closer to the utility layer are for the developers.
- Pros and Cons
    - Pros
        - Large amounts of abstraction built-in.
        - Driver will never see the insides of the project, even during design.
        - Can often predict where everything goes beforehand.
    - Cons
        - Hard to split up work as something in the UI layer might interfere with something in the utility layer.
        - Abstraction can lead to a lot of confusion when debugging.
        - Hard to make a functioning prototype as you need all 4 layers before it somewhat works.
- Suitability
    - This would not work for IoT HTL. We wouldn't be able to manage building everything from the ground up like this, as a lot of the UI interfaces also double in the utility layer (E.g Flexible Routing).

5.1.6 Model View Controller Architecture

- What is Model View Controller Architecture
    - Model View Controller Architecture is where a web server links different parts of the architectural design. In this case, it links the model with the view and controller of the model. This design also has external data easily accessible from the model.
- Pros and Cons
    - Pros
        - Great for web-based design
        - Easy to communicate between parts
        - Outside data built-in
        - Very easy to constantly update through something like GitHub
    - Cons
        - Horribly inefficient for in-person projects
        - Useless without internet access
        - Relies on HTML data
        - Requires external data to be fully effective
- Suitability
    - This definitely doesn't work for IoT HTL. This isn't a web-based project - We're designing a car's software which will have in-person aspects. We can't do it this way because this requires our software to be completely online. Even if that wasn't the case, we have too many moving parts for this architecture to work.

5.1.7 Finite State Machine Architecture

- What is Finite State Machine Architecture
    - Finite State Machine Architecture involves building a finite state machine to design your project around. You have multiple nodes of the project that are connected to each other through actions taken by the driver.
- Pros and Cons
    - Pros
        - Easy to see what path gets you where.
        - Good for architecture with lots of driver input.
        - Great for design in a coding environment due to the widespread use of if/else.
    - Cons
        - Inefficient and impractical for some real-life applications.
        - Hard to fully understand what's going and where in the case with a bigger version of the architecture.
        - You must have parts of your project that can be put into the nodes of the finite state machine which isn't always the case.
- Suitability

- This would not work for IoT HTL. We would have so many nodes it would very quickly break down. There's too many moving parts in our software to make a finite state machine that's readable. We also have a lot of "dead ends" and a lot of parts that are seperate from the rest.

5.1.8 Finalized Architecture

- The finalized architecture that we are going to use is Object-Oriented Architecture.
  - This is due to the fact that this type of architecture best fits our proposed design model and implements our many interconnected devices together in the most efficient way possible.

5.2 Interface Design

5.2.1 Driver Interface

- Driver turns on the vehicle and can access the interface. Now they can…
  - Define locations and Flexible Routing
    - Driver selects Home, School or enters new destination
    - Software takes that and starts map routing.
      - str address;
      - bool error;
      - bool arrived;
      - getAddress();
      - collect();
        - getData();
          - getGps();
        - getInternetContent();
      - flexibleRouting();
        - determineRoute();
        - referenceRoute();
        - changeRoute();
        - storeDestination();
      - displayErrorHud();
      - displayHud();
  - Activate/Deactivate cruise control
    - Driver presses the button to enable cruise control.
    - Software
      - bool error;
      - int speed;
      - collect();
        - getData();
          - getSFData();
        - getGps();

        - cruiseControl();
            - process();
            - setSpeed();
        - displayErrorHud();
        - displayHud();
    - Communicates with Sensor Fusion to obtain vehicle information and Planning to set cruising speeds.
- Control virtual side mirror cameras
    - Driver touches the screen to reposition the virtual side mirrors
    - Software
        - bool error;
        - displayCamera();
        - collect();
            - getData();
                - userInput();
                    - adjustCameraDisplay();
        - displayErrorHud();
        - displayHud();
- Toggle when solar panels are active.
    - Driver selects when to have solar panels activated.
    - Software processes request
        - bool error;
        - collect();
            - getData();
                - toggleSolar();
        - displayErrorHud();
        - displayHud();
- Answer prompts for Self-Repairs/Updates
    - Driver receives a question for when to perform a software update.
    - Software
        - bool error;
        - collect();
            - getData();
                - Yes
                    - performUpdate();
                    - log();
                - No
                    - log();
                - Schedule Time
                    - delayUpdate();

                - performUpdate();
                - log();
            - displayErrorHud();
            - displayHud();
    - Establish Bluetooth connection
        - Driver turns on pairing mode and uses their phone to connect to the vehicle.
        - Software runs btConnect which will pair the vehicle to the phone and throw an error message if need be.
            - bool error;
            - btConnect();
            - displayErrorHud();
            - log();
            - displayHud();
        - Electronics and Software will pair with the phone and store the necessary information for bluetooth pairing.

- **5.2.2 Technical Interface**
    - Technician accesses interface on the vehicle and has to enter login information. Passing login, Technician now can …
        - Gain access to the entire Vehicle Control System.
            - System Management Logs
                - Technician selects which module or log they want to read.
                - Interface takes module or log information and requests the specified log or data from the specified module.
                    - bool error;
                    - str logName;
                    - collect();
                        - requestDiagnostics();
                        - getData();
                        - getLog();
                    - log();
                    - displayHud();
                    - displayErrorHud();
                - Systems Management and Planning will work together to collect logs,data and any information the Technician will need and display it.
        - Debug/Test Software
            - Technician selects modules and starts tests.
            - Software will run tests on selected modules.
                - bool error;

        - requestDiagnostics();
        - log();
        - displayHud();
        - displayErrorHud();
    - Selected module will run built-in diagnostic tests and display data onto the hud so Technician can monitor the module as it runs. Logs event.
- Manually implement updates/repairs to software
    - Technician chooses to update software or clicks on repair.
    - Software will check if software is up to date and reports back to Technician that software is already up to date. If repair is clicked, will proceed to repair using the Self-Repair module.
        - bool repaired;
        - bool error;
        - collect();
            - requestDiagnostics();
            - getData();
        - log();
        - repair();
        - displayHud();
        - displayErrorHud();
    - The IoT engine downloads necessary information from our server and updates the software. Depending on repair needed, the Planning module may be used to facilitate repairs.

5.3 Component Level Design

- enableCruiseControl(int active)
    - Called upon when the driver presses the cruise control button. If cruise control is already active, disableCruiseControl() is called upon. Else, setSpeed() constantly waits for speed input and then is called upon.
    - setSpeed(int chosen, int current)
        - Takes in the inputted speed and the current speed of the vehicle
        - Ensures that the chosen speed is within the pre-conditions of the current speed
        - If there is no error, setSpeed() changes the vehicle's speed to chosen.
            - setSpeed() then waits again for another driver input, or for cruise control to be disabled.
        - On error, disableCruiseControl() and logEvents().
    - disableCruiseControl()

        - Called from enableCruiseControl(). Disables cruise control and allows the driver to manually control the speed. calls upon logEvents() to store the event in System Management.
- startFlexibleRouting()
    - RequestGPS()
        - Requests the GPS to allow flexible routing to begin
        - Doesn't take anything in
    - ConnectIOT()
        - Connects to the Internet of Things
        - Doesn't take anything in
    - DetermineRoute()
        - Takes in the internet connection and the GPS
        - Determines the best possible route
        - Calls other methods if redirection is needed
    - RefrenceRoute()
        - Takes in the route from above
        - If any change is needed, go to ChangeRoute()
    - ChangeRoute()
        - Takes in the route and changes it if needed
        - If no change is required, simply leave the method
    - StoreDestination()
        - Takes the destination as input
        - Prompts the driver to store the destination if they want to

- startAISensor()
    - sensorDiagnosis()
        - Requests a diagnosis of the perception and localization sensors
        - Returns and integer to the AI Sensor component
            - 0 tells the AI Sensor component that there are no errors in any of the sensors and that it should continue to operate
            - 1 tells the AI Sensor to shutdown so the sensor with an error can be repaired while it is not in use.
            - The SensorDiagnosis() is further handled by selfDiagnosisRepair() and thus does not need a reference to the sensor(s) that have the error
        - The input is void as the function itself is a call to the selfDiagnosisRepair()
    - requestSensorData()
        - The function splits between localization and perception sets of sensors and recursively adds sensor data to a dynamically sized array of strings

        - Returns an array of strings that hold the concatenated data retrieved by from the sensors
        - No input is needed as the sensors themselves are not receiving any information
    - exportSensorData(str[] data)
        - Pipes the information retrieved from RequestSensorData() to the control component of the system so it can be processed for other components to use
        - Returns an integer indicating whether or not the function succeeded
            - A 0 means the the function completed successfully and the component continues and calls logEvents()
            - A 1 means the function failed. displayErrorHUD() is called to inform the user and logEvents() is called to log the event. selfDiagnosisRepair() is then called to handle the problem
        - The input is an array of strings and each index is the data of a certain sensor

- selfDiagnosisRepair()
    - requestLogs(str module);
        - Given module name, requests logs of that module from Systems Managements. Returns logs if available.
    - getData(str module);
        - Given module name, requests data specifically from the module. Returns data if available.
    - getServerData(str module);
        - Given module name, requests data specifically from the server about the module. Returns data if available.
    - readData(str[] log, str[] data, str[] serverData);
        - Reads logs and data and detects any abnormalities using serverData. If abnormalities are found, call displayErrorHUD() informing the driver that an error has been found. Information displayed on hud will vary depending on the severity of error found.
    - repair(int input, str error);
        - If input = 1 then user does not want repair else repair will take str err and attempt to follow instructions from serverData based on str err to perform repair vehicle. If repair is successful then displayHud() else displayErrorHud().
    - logEvents(str event) if diagnostics found an error or not and if repair was successful or not.

- autoSummon/Parking()
    - autoSummon(int x, int y, int z);
        - Ensures that the caller is within a certain range to navigate to.
        - Takes in the driver's current coordinates, and calls upon startFlexibleRouting() to map how the vehicle should get to those coordinates.
        - Calls upon connectVCS(int array route) to take full control of the vehicle to follow navigation
    - autoPark();
        - When called, something something find valid parking spots
        - requestSensorData();
            - The function splits between localization and perception sets of sensors and recursively adds sensor data to a dynamically sized array of strings
            - Returns an array of strings that hold the concatenated data retrieved by from the sensors
            - No input is needed as the sensors themselves are not receiving any information
        - calculateSpots(int[] sensorData);
            - Will use sensorData to find valid parking spots near the vehicle and once the best spot has been found, will call connectVCS() with the parking spot destination as a parameter.
    - connectVCS(int array route);
        - Allows for the vehicle control system to be fully controlled by the route given from Flexible routing. Anytime in the route speed, direction, or signals need to be activated, it will call upon one of its helper functions. Disconnects when sendStop() is called.
        - changeSpeed(int speed);
            - Communicates with Planning to set speed of vehicle
        - turnWheel();
            - Communicates with Planning to manipulate vehicle's wheels
        - turnSignal(int turn);
            - If turn == 0 then signalLeft() else signalRight().
        - sendStop();
            - If autoSummon was made from phone then indicateApp else indicateHud
            - indicateApp(str notification);
                - displayApp(notification)
            - indicateHud(str notification);
                - displayHUD(notification)

- disconnectVCS();
    - Disables override from summon and restores control and access to the driver.
    - Calls autoPark() to stop the car in a secure manner.

Section 6: Use-Case Testing

6.1 Lane Mitigation

6.1.1 Normal Testing

- A Lane Mitigation object is created whenever the turn signal isn't activated and the vehicle is currently driving. If the vehicle detects you are over the line by a certain amount it will adjust to the right spot. If the steering wheel is ever detected in motion then lane mitigation will disable and the driver will have full control.

6.1.2 Error Testing

- Errors in this section stem from the failure of prerequisites if the vehicle currently has its blinkers on or it isn't on a valid road. If either of these occur lane mitigation reassesses itself until both of these measures are met.

6.1.3 Test Cases

- Prerequisites aren't met
- Enabling mitigation and having it realign the vehicle in a lane
- Disabling mitigation on a wheel turn

6.2 Self-Diagnosis/Repair

6.2.1 Normal Testing

- Self-Diagnosis object is created, with the module M which is to be self diagnosed. The car then checks connection to IOT, and connects if it isn't. Then data is read from Server logs, Module logs, and Module data. If no error is found, return good otherwise.

6.2.2 Error Testing

- If an error is found, we call upon repair which attempts the repair within the system. If another error occurs and the self-repair fails, urge the driver to take it to a mechanic.

6.2.3 Test Cases

- Module M being checked requires no repair
- Module M has an error in Module Data/Module Logs/Server Logs
- IOT cannot find logs in the system.

6.3 Flexible Routing

6.3.1 Normal Testing

- A Flexible Routing object is created, with the location that the driver wants to travel to. The car then checks connection to IOT and GPS in order to travel. Then the location is taken and a route to that location is either found in the guide book or errors out. If the location is in the guide book, then the route is referenced and directions are given until the destination is reached.

6.3.2 Error Testing

- If an error is found in IOT or GPS connection, the vehicle will continue to try and reconnect them until a connection is found. If an error is found in the location guide book, the HUD will display that the location is unknown and will disable routing.

6.3.3 Test Cases

- Location L isn't in the guide book.
- GPS & IOT aren't connected
- Location L is in the guide book and the driver navigates there.

6.4 Auto Summon/Auto Parking

6.4.1 Normal Testing

- The SummonParking object is created with one of the two constructors. One for summoning, which takes the input of current x,y, and z coordinates, and the other for parking, which receives in an int[][] of sensor data. The two different constructors call upon summoning and parking respectively.
- In Parking, an xyz position is created based on sensor data and then connected to the Vehicle control for AutoPark with current xyz. If autopark, the vehicle simply parks at that position.
- In summoning, we also connect to the Vehicle control system where GPS provides directions to get to that xyz location. Once it's arrived, auto parking is called to park the car for the driver to safely enter. Once the vehicle is parked, the Vehicle Control System is disconnected.

6.4.2 Error Testing

- The errors in this section are to the timing of whether or not the directions print out at the right time. If the directions are missed for some reason, an Exception e will be thrown and displayed to the HUD.

6.4.3 Test Cases

- Auto Parking without a Summon
- Auto Parking with a Summon
- Directions cannot be displayed

6.5 Emergency Contact

6.5.1 Normal Testing

- Emergency contact object is created when the menu to add contacts is accessed for the first time. From there, contacts can be added or removed as long as the driver isn't actively driving. On crash detection, if a driver fails to put in an input, then all emergency services and contacts are directed. If the driver does put in an input, services are not contacted and they can continue on their way.

6.5.2 Error Testing

- If an error is found in IOT or GPS connection, the vehicle will continue to try and reconnect them until a connection is found. This is the only source of error as it wouldn't allow us to send emergency contact out to first responders.

6.5.3 Test Cases

- Add/remove/printing out contacts to emergency contact list
- Crash detected and contacts Emergency Services and Contacts
- Driver puts in input to state they are okay.

6.6 Internet Integration

6.6.1 Normal Testing

- An internet integration object is set up whenever the driver inputs that they want to do something that requires an internet connection, for example checking the weather or using Spotify. For anything to work, the vehicle must firstly be connected to the internet. After that, any feature can be connected, and the vehicle can also connect to the ALSET servers to receive software updates.

6.6.2 Error Testing

- Most errors in the internet integration section stem from not currently being connected to the internet. The only other error that is thrown is when the vehicle cannot connect to the internet, where it then needs to be taken in for repairs.

6.6.3 Test Cases

- If internet connectivity ever fails for repair
- Connecting to different features (Spotify and Weather)
- Connecting to the ALSET server
- Connecting without internet connectivity

6.7 HUD

6.7.1 Normal Testing

- The HUD is set up to display all the necessary information that pertains to the driver's ride status and any necessary warnings on the vehicle's condition. By utilizing parameters with given sensor values, the HUD is able to show what relevant gauges are needed for the driver.

6.7.2 Error Testing

- Mainly the HUD checks whether any alerts have been enabled via other parts of the car's systems and will immediately display that to the driver. Furthermore, it tests whether the remaining battery percentage is empty or not and will throw an error accordingly.

6.7.3 Test Cases

- Alert status checks.
- Speedometer display.
- Cruise Control indicator.
- Remaining battery percentage and emptiness.

6.8 Virtual Side Mirrors

6.8.1 Normal Testing

- A virtual side mirror object is created with parameters to determine the angle the side view mirrors the left and right side have, and if the camera feed is being shown on the HUD. From here we can choose to adjust each mirror and either turn on/off the camera feed on the hud.

6.8.2 Error Testing

- If the mirror angle is too obtuse or acute, attempt to shift the mirror to the default angle of 65 degrees.

6.8.3 Test Cases

- Displaying camera feed on HUD.
- Shutting off camera feed on HUD
- Adjusting angle of left mirror
- Adjusting angle of right mirror

6.9 AI Sensor

6.9.1 Normal Testing

- An AI Sensor object is set up with parameters to determine the status of the sensors in the car. If the sensors are connected and the diagnosis proves they are functional then the object handles pulling the data off of each sensor and

compiling them into a single string array which can be used by the car. The car repeatedly calls this object method so long as it's functional

6.9.2 Error Testing

- If the sensor AI encounters an error while connecting to the IoT, then the sensor AI turns itself off and the car repeatedly attempts to reconnect.
- If one of the sensors is damaged or if the self diagnosis catches an error, then the sensor AI turns off

6.9.3 Test Cases

- Successfully send sensor data to IoT
- Failure to connect to IoT
- Self Diagnostics find error in sensors

6.10 Cruise Control

6.10.1 Normal Testing

- When a cruise control object is created, it takes in the speed S which we want to set the vehicle's current speed to. The vehicle ensures safety of cruise control by checking logs, and if all is well it sets the requested speed to the Vehicle Control System.

6.10.2 Error Testing

- If connecting to IOT errors, then the car attempts to reconnect until the connection is successful. If the logs read in error from Cruise Control, then it is disabled and Self-Diagnosis/Repair is called upon.

6.10.3 Test Cases

- Changing speed successfully
- Failure to connect to IOT
- CruiseControl safety check fails

Section 7: Code

7.1 Car.java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Car{
    public static void main(String[] args) throws IOException{
        int GPS = 0, IOT = 0;
        boolean on = true;
        String[] data = {"Summon to: 35 -10 23",
                                    "Park at: 123 -123 0",
                                    "HUD Speed: 35 38",
                                    "Start routing: School",
                                    "Cruise Control: 17",
                                    "HUD CC: 17",
                                    "Internet Integration: Spotify",
                                    "Solar Panels: 0 75",
                                    "Lane Mitigation: 0",
                                    "Self Diagnosis: Planning",
                                "Mirror: 0",
                                "Left Mirror: 34.0",
                                "Right Mirror: 26.0",
                                "Technician Entry",
                                "Sensor AI: 1 1",
                                "HUD Battery: 35",
                                "Technician Done",
                                "Set Contact: Steve",
                                "Set Contact: Izzi",
                                "Remove Contact: Steve",
                                "CRASH"};

        emergencyContact b = new emergencyContact();
```

```
        virtualSideMirrors c = new virtualSideMirrors(65.0, 65.0, 0);
        HUD d = new HUD();
        for(int i = 0; i < data.length; i++) {
            //SUMMON PARKING - DONE
            if(data[i].contains("Summon to: ")) {
                  data[i] = data[i].substring(11);
                  int x = Integer.parseInt(data[i].substring(0,
data[i].indexOf(' ')));
                  data[i] = data[i].substring(data[i].indexOf(' ') + 1);
                  int y = Integer.parseInt(data[i].substring(0,
data[i].indexOf(' ')));
                  data[i] = data[i].substring(data[i].indexOf(' ') + 1);
                  int z = Integer.parseInt(data[i].substring(0));
                  SummonParking a = new SummonParking(x, y, z);
            }
            else if(data[i].contains("Park at: ")) {
                  data[i] = data[i].substring(9);
                  int x = Integer.parseInt(data[i].substring(0,
data[i].indexOf(' ')));
                  data[i] = data[i].substring(data[i].indexOf(' ') + 1);
                  int y = Integer.parseInt(data[i].substring(0,
data[i].indexOf(' ')));
                  data[i] = data[i].substring(data[i].indexOf(' ') + 1);
                  int z = Integer.parseInt(data[i].substring(0));

                  int[][] inputs = {{x-1, x, x+1}, {y-1, y, y+1}, {z-1, z,
z+1}};
                  SummonParking a = new SummonParking(inputs);
            }

            //FLEXIBLE ROUTING - DONE
            if(data[i].contains("Start routing: ")) {
                  FlexibleRouting a = new FlexibleRouting("driver",
data[i].substring(15));
                  IOT = a.connectIOT(IOT);
                GPS = a.requestGPS(GPS);
                a.determineRoute(IOT, GPS, a.location);
            }


            //CRUISE CONTROL - DONE
            if(data[i].contains("Cruise Control: ")) {
```

```
                CruiseControl a = new
CruiseControl(Integer.parseInt(data[i].substring(17)));
                IOT = a.ConnectToIOT();
                a.setSpeed(a.speed);
          }

          //SOLAR PANELS - DONE
          if(data[i].contains("Solar Panels: ")) {
                data[i] = data[i].substring(14);
                int onoff = Integer.parseInt(data[i].substring(0,
data[i].indexOf(' ')));
                data[i] = data[i].substring(data[i].indexOf(' ') + 1);
                double battery = Integer.parseInt(data[i].substring(0));
                solarPanels a = new solarPanels(battery, onoff);
                a.checkBattery();
                a.toggleSolar();
                a.charge();
                a.checkBattery();
                a.toggleSolar();
          }

          //INTERNET - DONE
          if(data[i].contains("Internet Integration: ")) {
                internetIntergration a = new internetIntergration();
                a.connectInternet();
              a.connectServer();
              a.connectFeature(data[i].substring(22));
              a.disconnectInternet();
          }

          //SELF DIAGNOSIS/REPAIR - DONE
          if(data[i].contains("Self Diagnosis: ")) {
                 selfDiagnosisRepair a = new
selfDiagnosisRepair(data[i].substring(16));
                 IOT = a.connectIOT(IOT);
                 a.readData(IOT);
          }


          //LANE MITIGATION - DONE
          if(data[i].contains("Lane Mitigation: ")) {
                laneMitigation a = new
```

```
laneMitigation(Integer.parseInt(data[i].substring(17)));
                for(int j = 0; j < 7; j++) {
                      a.checkAlign();
                }
                a.DisableMitigation(1);
          }

          //EMEREGENCY CONTACT - DONE
          if(data[i].contains("Set Contact: ")) {
              IOT = b.connectIOT(IOT);
              GPS = b.connectGPS(GPS);
              b.setContact(data[i].substring(13));
              b.printContacts();
          }
          else if(data[i].contains("Remove Contact: ")) {
              IOT = b.connectIOT(IOT);
              GPS = b.connectGPS(GPS);
              b.removeContact(data[i].substring(16));
              b.printContacts();
          }
          else if(data[i].equals("CRASH")) {
                b.crashes();
          }


          //VIRTUAL SIDE MIRRORS - DONE
          if(data[i].contains("Mirror: ") && !data[i].contains("Right")
&& !data[i].contains("Left")) {
                c.displayCamera();
          }
          else if(data[i].contains("Left Mirror: ")) {
                c.adjustCamera(Double.parseDouble(data[i].substring(13)),
0);
          }
          else if(data[i].contains("Right Mirror: ")) {
                c.adjustCamera(Double.parseDouble(data[i].substring(14)),
1);
          }

          //HUD
          if(data[i].contains("HUD Battery: ")) {
```

```
d.displayBattery(Integer.parseInt(data[i].substring(13)));
            }
            else if(data[i].contains("HUD Speed: ")) {
                        data[i] = data[i].substring(11);
                  int x = Integer.parseInt(data[i].substring(0,
data[i].indexOf(' ')));
                  data[i] = data[i].substring(data[i].indexOf(' ') + 1);
                  int y = Integer.parseInt(data[i].substring(0));
                  d.displaySpeed(x, y);
                  }
            else if(data[i].contains("HUD CC: ")) {

d.displayCruiseControl(Integer.parseInt(data[i].substring(8)));
                  }


            //AI SENSOR - DONE
            if(data[i].contains("Sensor AI: ")) {
                  data[i] = data[i].substring(11);
                  int x = Integer.parseInt(data[i].substring(0,
data[i].indexOf(' ')));
                  data[i] = data[i].substring(data[i].indexOf(' ') + 1);
                  int y = Integer.parseInt(data[i].substring(0));
                  sensorAI a = new sensorAI(x, y);
                  a.isConnected();
                 a.isFunctional();
            }


            //TECHNICIAN ACCESS - DONE
            if(data[i].equals("Technician Entry")) {
                  String password;
                  int tech;
              System.out.println("Enter Technician Password:"); //Prompt
user for response and read console line
              BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
              password = in.readLine();
              if(password.equals("1234"))
              {
                  tech = 1;
              }
```

```
            else
            {
                tech = 0;
            }
            if(tech == 1)
            {
                System.out.println("Technician Menu");
            }
            else
            {
                System.out.println("User Menu");
            }
        }
        else if(data[i].equals("Technician Done")) {
                System.out.println("User Menu");
        }

        //SLEEP FOR 5 SECONDS BETWEEN EACH COMMAND
        try {
                Thread.sleep(3000);
                System.out.println("\n\n");
        }
        catch (Exception e) {
                System.out.println(e);
        }
    }
  }
}
```

7.2 CruiseControl.java

```
import java.lang.Thread;

public class CruiseControl extends Car {

    int isOn = 0; //0 off, 1 on
    int speed;
    int connected;
    int isSafe;

    public CruiseControl(int s){ //set fields
        isOn = 1;
```

```
        speed = s;
        connected = 0;
        isSafe = 1;
        System.out.println("Enabling Cruise Control.");
    }

    public int ConnectToIOT(){
        if(isOn == 1 && connected == 0){
            System.out.println("Connecting to IOT.");
            connected = 1;
            return connected;
        }
        if(connected == 0){
            System.out.println("Connection failed, trying to reconnect.");
            return ConnectToIOT();

        }
        return 0;
    }

    public void setSpeed(int newSpeed){ //takes speed (int) as a parameter
for our crusing speed
      checkSafe();
        System.out.println("Changing to Requested speed.");
        try { //try to change speed and throw out an error if we can't
            Thread.sleep(5000);
        }
        catch (Exception e) {
            System.out.println(e);
        }
        System.out.println("Speed Set!"); //inform user
    }

    public void checkSafe(){ //check if safe to be on cruise control
        if(isSafe != 1){
            isOn = 0;
            System.out.println("Car needs to be fixed, disabling Cruise
Control.");
        }
    }
}
```

7.3 FlexibleRouting.java

```
import java.lang.Thread;
import java.util.Random;

public class FlexibleRouting extends Car{
    //onoff -> (0 turn off) (1 turn on)
    int onoff;
    //caller -> either 'autosummon' or 'driver'
    //location -> looks up in our 'GPS Book'
    String caller, location;

    //constructor
    public FlexibleRouting(String c, String l){
        System.out.println("Turning on maps...\n");
        onoff = 1;
        caller = c;
        location = l;
    }

    //takes current status of gps, if off turns on, else turn off
    //used for enabling and disabling the GPS system
    public int requestGPS(int GPS){
            System.out.println("Connecting to GPS...");
        if(GPS == 0){
            System.out.println("Connected!\n");
            return 1;
        }
        System.out.println("Connection failed.. reconnecting");
        return requestGPS(GPS);
    }

    //takes current status of iot, if off turns on, else turn off
    //used for connecting to IOT
    public int connectIOT(int IOT){
            System.out.println("Connecting to IOT...");
        if(IOT == 0){
            System.out.println("Connected!\n");
            return 1;
        }
```

```
        System.out.println("Connection failed.. reconnecting");
        return connectIOT(IOT);
    }

    //takes status of IOT and GPS, ensures they are both on, and then looks for location in the GPS book
    //either throws
    public int determineRoute(int IOT, int GPS, String location){
        String[][][] locations = {{{"School"}, {"Turn Right", "Go Straight", "Turn Left", "Arrived"}}, {{"Home"}, {"Turn Right", "Go Straight", "Turn Left", "Arrived"}}};
        //IOT and GPS are on and working
        if(IOT == 1 && GPS == 1){
            for(int i = 0; i < locations.length; i++){
                //if place is in our location list
                if(locations[i][0][0] == location){
                    //return directions to location
                    System.out.println("Mapping to " + location + "!");
                    referenceRoute(locations[i][1]);
                    return 0;
                }
            }
            System.out.println("Error: Location unknown.");
            return -1;
        }
        return -1;
    }

    public void referenceRoute(String[] curroute){
        for(int i = 0; i < curroute.length; i++){
            //catching if sleep errors
            Random rand = new Random();
            try {
                    Thread.sleep(rand.nextInt(3000, 10000));
            }
            catch (Exception e) {
                    System.out.println(e);
            }

            //print instruction and check if process is ended
```

```
        System.out.println(curroute[i]);
        if(curroute[i] == "Arrived"){
            this.onoff = 0;
            return;
        }
    }
  }
}
```

7.4 HUD.java

```
public class HUD extends Car{

    int alertState = 0;         // 0 = off (default), 1 = on.

    public HUD() {

    }

    // Checks status of alerts.
    public int checkStatus() {
        if (alertState == 1) {      // If alert is detected, returns 1 and prints a
warning.
            System.out.println("Alert: System Diagnostics has detected a
warning!");
            return 1;
        } else {
            System.out.println("Status is stable");     // If no alert is detected,
then returns 0.
            return 0;
        }
    }

    // Displays current speed and speed limit.
    public void displaySpeed(int currentSpeed, int speedLimit) {
        System.out.print("Current Speed: " + currentSpeed);
        System.out.println(" MPH");
        System.out.print("Speed limit: ");
        System.out.print(speedLimit);
        System.out.println(" MPH");
    }

    // Checks if cruise control is enabled and if so, prints the cruise control
```

```
speed.
    public void displayCruiseControl(int cruiseControl) {    // 0 = off (default),
# = speed of cruise control.
        if (cruiseControl > 0) {
            System.out.print("Cruise Control enabled: ");
            System.out.print(cruiseControl);
            System.out.println(" MPH");
        } else {
            System.out.println("Cruise Control disabled");
        }
    }

    // Displays battery percentage
    public int displayBattery(int batteryState) {           // 0-100
        System.out.print(batteryState);
        System.out.println("% battery remaining");

        if (batteryState <= 20) {       // If battery is less than or equal to 20%,
then it prints a warning message.
            if (batteryState == 0) {    // If the battery is empty, it prints an
alert and changes the alert state to 1.
                System.out.println("Alert: Battery is empty!");
                alertState = 1;
                return 1;
            }
            System.out.println("Battery is low!");
        }
        return 0;
    }
}
```

7.5 SummonParking.java

```
import java.util.Random;
import java.lang.Thread;

public class SummonParking extends Car{
      //to summon -> gives x,y,z coordinates
      Random rand = new Random();
      int sx, sy, sz;
      public SummonParking(int x, int y, int z){
            sx = x;
            sy = y;
            sz = z;
            autoSummon();
```

```
	}

	//array of different x,y,z positions [[234, 235, 236], [450,451,452]
....
	int[][] sensor;
	//to autopark -> gives sensor data
	public SummonParking(int[][] s) {
		sensor = s;
		autoPark();
	}

	//takes in sensor data and uses it to calculate a parking spot
	public void autoPark() {
		int[] temp = calculateSpots(this.sensor);
		this.sx = temp[0];
		this.sy = temp[1];
		this.sz = temp[2];
		System.out.println("AutoParking at x: " + this.sx + " y: " +
this.sy + " z: " + this.sz);
		connectVCS(this.sx, this.sy, this.sz, "AutoPark");
	}

	public void autoSummon() {
		System.out.println("AutoSummoning to x: " + this.sx + " y: " +
this.sy + " z: " + this.sz);
		connectVCS(this.sx, this.sy, this.sz, "AutoSummon");
	}

	//takes in sensor data and calculates and x,y,z position array to
park at
	public int[] calculateSpots(int[][] sensorData) {
		int[] returndata = new int[3];
		for(int i = 0; i < 3; i++) {
			returndata[i] = sensorData[i][rand.nextInt(0, 3)];
		}
		return returndata;
	}

	public void connectVCS(int x, int y, int z, String caller){
		if(caller.equals("AutoPark")){
			System.out.println("Parking vehicle...");
			try {
```

```
                  Thread.sleep(5000);
            }
            catch (Exception e) {
                  System.out.println(e);
            }
                  System.out.println("Parked!");
            }
            else {
                  int temp = 0;
                  String[] commands = {"Stop", "Turn wheel right", "Go
forward", "Turn wheel left", "Slow down", "Speed up"};
                  for (int i = 0; i < rand.nextInt(3,9); i++){
                        try {
                        Thread.sleep(rand.nextInt(3000, 10000));
                  }
                  catch (Exception e) {
                        System.out.println(e);
                  }

                      int picker = rand.nextInt(0,6);
                      if(temp == picker){
                        if(picker == 5){
                              picker--;
                        }
                        else{
                              picker++;
                        }
                      }
                      temp = picker;
                      System.out.println(commands[picker]);
                  }
                  System.out.println("Arrived.");
            }
            //add else for autosummon commands
            disconnectVCS(caller);
      }

      public void disconnectVCS(String caller){
            if(caller.equals("AutoPark")) {
                  System.out.println("Disconnecting VCS...");
                  System.out.println("Disconnected!");
            }
```

```
            else {
                  int[][] sensorData = new int[3][3];
              for (int r = 0; r < sensorData.length; r++){
                  for (int c = 0; c < sensorData[0].length; c++){
                      if(r == 0){
                        sensorData[r][c] = rand.nextInt(this.sx-5,
this.sx+6);
                      }
                      else if(r==1) {
                        sensorData[r][c] = rand.nextInt(this.sy-2,
this.sy+3);
                      }
                      else {
                        sensorData[r][c] = rand.nextInt(this.sz-5,
this.sz+6);
                      }
                  }
              }
                  this.sensor = sensorData;
                  autoPark();
            }
      }
```

7.6 EmergencyContact.java

```
import java.util.Scanner;
import java.util.Timer;
import java.util.TimerTask;
import java.io.*;


/*
 *      emergencyContact a = new emergencyContact();
        IOT = a.connectIOT(IOT);
        IOT = a.connectGPS(GPS);
        a.setContact("Stev");
        a.setContact("Izz");
        a.setContact("Dill");
        a.setContact("Chips");
        a.setContact("Michael \"The Man\" Mancino");
        a.printContacts();
        a.removeContact("Steven Truong");
        a.removeContact("Stev");
        a.printContacts();
        a.crashes();
 */

public class emergencyContact extends Car{

      int index = 0; // index to keep track of errorList
      int gps,  iot;
      public static String[] contactList = new String[1024]; // global
array to contain all errors

      public emergencyContact()
      {

      }

      private String str = ""; //timer task code. Section will run after x
time.
      TimerTask task = new TimerTask() {
            public void run() {
                  if (str.equals("")) {
                        System.out.println("No user input ... Calling
Emergency Services and Contacts");
```

```
                        contact();
                        System.exit(0);
                  }
            }
      };

      public void getInput() throws Exception {
            Timer timer = new Timer(); //setup timer
            timer.schedule(task, 10 * 500);

            System.out.println("Crash detected! Input a string within 5
seconds!: "); //Prompt user for response and read console line
            BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
            str = in.readLine();

            timer.cancel(); //cancel timer once we get user input.
            System.out.println(
                        "User input detected ... All clear. Would you like
to contact Emergency Serives and Contacts? (0/1) (0 for no. 1 for yes.)");

            Scanner sc = new Scanner(System.in); //reads from console
        int temp = sc.nextInt();
            if (temp == 1) {
                  contact();
            }
      }


      public void crashes() // crash detection
      {
            try
        {
             getInput();
        }
        catch( Exception e )
        {
            System.out.println( e );
        }
      }

      public void contact()  //contact emergency services and contacts
```

```
	{
		this.iot = 1;
		System.out.println("Contacting Emergency Services...");
		if(iot == 1)
		{
			System.out.println("Contacted Emergency Services
sucessful! \n");
			System.out.println("Sending GPS information ...");
			if (gps == 1)
			{
				System.out.println("Sending GPS sucessful!\n");
			}
			else
			{
				System.out.println("Sending GPS failed!\n");
			}
		}
		else
		{
			System.out.println("Contacted Emergency Services failed!
\n");
		}
		System.out.println("Contacting Emergency Contacts ...");
		if(iot == 1)
			{for (int i = 0; i < index; i++) // goes contact list and
blanks out contact
		{
			String x = contactList[i];
			if (x != "") // if blank, don't print
			{
				System.out.print("Contacting ");
				System.out.println(x);
			}

		}
		}
	}

	public void setContact(String Contact) // tacks contact into array
and increases index
	{
		contactList[index] = Contact;
```

```
            index++;
      }

      public void removeContact(String Contact) {
            for (int i = 0; i < index; i++) // goes throughout contact list
and blanks out contact
            {
                  String x = contactList[i];
                  if (x.equals(Contact)) {
                        contactList[i] = "";
                  }
            }
      }

      public void printContacts() {
            System.out.println("Emergency Contact List:");
            for (int i = 0; i < index; i++) // goes contact list and blanks
out contact
            {
                  String x = contactList[i];
                  if (x != "") // if blank, don't print
                  {
                        System.out.println(x);
                  }

            }
            System.out.println("\n");
      }

      public int connectGPS(int GPS) {
            System.out.println("Connecting to GPS...");
            if (GPS == 0) {
                  System.out.println("Connected!\n");
                  gps = 1;
                  return 1;
            }
            System.out.println("Connection failed.. reconnecting");
            gps = 0;
            return connectGPS(gps);
      }

      public int connectIOT(int IOT) {
```

```
            if (IOT != 1)
            {
            System.out.println("Connecting to IOT...");
            if (IOT == 0) {
                  System.out.println("Connected!\n");
                  iot = 1;
                  return 1;
            }
            System.out.println("Connection failed.. reconnecting");
            return connectIOT(IOT);
            }
            return 0;
      }

}
```

7.7 internetIntegration.java

```
import java.util.concurrent.TimeUnit;

/*
 *         internetIntergration a = new internetIntergration();
        a.connectFeature("Spotify");
        a.connectServer();
        a.connectInternet();
        a.connectInternet();
        a.connectFeature("Spotify");
        a.connectFeature("Weather");
        a.connectServer();
        a.disconnectInternet();
        a.disconnectInternet();
 */
public class internetIntergration {

      int connected;
      public internetIntergration()
      {

      }

      public int connectInternet()
      {
```

```
            if (connected == 0)
            {
            System.out.println("Connecting ...");
            try {
                  TimeUnit.SECONDS.sleep(3);
            } catch (InterruptedException e) {
                  System.out.print("Attempt failed! Please take to
mechanic! \n");
                  return 0;
            }
            System.out.println("Connected! \n");
            connected = 1;
            return 1;
            }
            else
            {
                  System.out.println("Already connected \n");
                  return 1;
            }
      }

      public void disconnectInternet()
      {
            if(connected == 1)
            {
            System.out.println("Disconnecting ...");
            try {
                  TimeUnit.SECONDS.sleep(3);
            } catch (InterruptedException e) {
                  System.out.print("Attempt failed! Please take to
mechanic! \n");
            }
            System.out.println("Disconnected! \n");
            connected = 0;
      }
            else
            {
                  System.out.println("Already disconnected \n");
            }
      }
      public void connectFeature(String feature) //Connect to Spotify,
Check Mail, Check weather
```

```
	{
		System.out.print("Connecting to ");
		System.out.print(feature);
		System.out.println("...");
		if (connected == 1)
		{
			System.out.println("Sucess!");
			if(feature.equals("Spotify") ||
feature.equals("spotify"))
			{
				System.out.println("Now Playing: Beets by Tray
\n");
			}
			else if (feature.equals("weather") ||
feature.equals("Weather"))
			{
				System.out.println("73F today in Hoboken. Small
Chance of Rain Later Tonight. \n");
			}
		}
		else
		{
		System.out.println("Failed! Check if vehicle is connected to
the internet! \n");
		}
	}

	public int connectServer()  //Connect with our servers to check for
updates and download data
	{
		System.out.println("Connecting to ASLET Servers...");
		if (connected == 1)
		{
			System.out.println("Sucess! \n");
			return 1;
		}
		else
		{
		System.out.println("Failed! Check if vehicle is connected to
the internet!\n");
		return 0;
		}
```

```
        }

}
```

7.8 selfDiagnosisRepair.java

```
import java.util.*;
public class selfDiagnosisRepair {

int errorFound; //way to determine if we have encountered an error
String module;

int index = 0; //index to keep track of errorList
public static String[] errorList = new String[1024]; //global array to contain all errors

/*
 *        selfDiagnosisRepair a = new selfDiagnosisRepair("Planning");
 *        IOT = a.connectIOT(IOT);
 *         a.readData(IOT);
 */
public selfDiagnosisRepair(String m)
{
        module = m;
}

public int connectIOT(int IOT){
        System.out.println("Connecting to IOT...");
        IOT = 0;
   if(IOT == 0){
      System.out.println("Connected!\n");
      return 1;
   }
   System.out.println("Connection failed.. reconnecting");
   return connectIOT(IOT);
}

public String[] requestLogs()
{
        System.out.println("Requesting Logs...");
```

```
		String[] logs = {"Planning","Working","Working", "Error: Engine Temps Too
High","Working"};
		if(logs[0].equals(module))
		{
			System.out.println("Logs found! \n");
			return logs;
		}
		System.out.println("Logs not found! \n");
		return new String[0];
}
public String[] getServerData()
{
		System.out.println("Requesting Server Data...");
		String[] logs = {"Planning","Working","Working", "Working","Working","Base Data" };
		if(logs[0].equals(module))
		{
			System.out.println("Server Data found! \n");

			return logs;
		}
		System.out.println("Server Data not found! \n");
		return new String[0];
}

public String[] getData()
{
		System.out.println("Requesting Module Data...");
		String[] logs = {"Planning","Working","Working", "Working","Working", "Error: Hi"};
		if(logs[0].equals(module))
		{
			System.out.println("Module Data found! \n");
			return logs;
		}
		System.out.println("Module Data not found! \n");
		return new String[0];
}

public int readData(int IOT)
{
```

```
        String[] moduleLogs = getData();
        String[] moduleData = requestLogs();
        if(IOT == 1) //need to be connected to IOT for server data
        {
        String[] serverLogs = getServerData();
        if(serverLogs[0].equals(module))
        {
                System.out.println("Reading Server Logs");
                 for (int i = 0; i < serverLogs.length; i++) //goes through logs and looks for
errors. If error found, add to array and continue.
                 {
                String x = serverLogs[i];
                if(x.contains("Error"))
                {
                        errorList[index] = x;
                        index++;
                        errorFound = 1;
                }
            }
        }
        }
        if(moduleLogs[0].equals(module))
        {
                 System.out.println("Reading Module Logs");
                 for (int i = 0; i < moduleLogs.length; i++)  //goes through logs and looks for
errors. If error found, add to array and continue.
                 {
                String x = moduleLogs[i];
                if(x.contains("Error"))
                {
                        errorList[index] = x;
                        index++;
                        errorFound = 1;
                }
            }
        }
        if(moduleData[0].equals(module))
        {
                 System.out.println("Reading Module Data");
```

```
                for (int i = 0; i < moduleData.length; i++)  //goes through logs and looks for
errors. If error found, add to array and continue.
                {
                String x = moduleData[i];
                if(x.contains("Error"))
                {
                        errorList[index] = x;
                        index++;
                        errorFound = 1;
                }
            }
        }
        if (errorFound == 1) //prompt user if they want to do repairs
        {
                System.out.println("Error detected ... \n");
                 for (int i = 0; i < index; i++)
                 {
                         System.out.println(errorList[i]);
            }

                System.out.print("\nWould you like to repair? \n(0/1. 0 for no repair. 1 for
repair) \n");
                Scanner sc = new Scanner(System.in); //reads from console
      int repair = sc.nextInt();
                repair(repair);
                return 1;
        }
        else //print out we have no error
        {
                System.out.print("No Error Found in ");
                System.out.print(module);
                return 0;
        }
}


public int repair(int repairYes) //if repair is 1, that means repair if possible
{
        if (repairYes == 1)
```

```
        {
                String[] logs = getServerData();
                System.out.println("Attempting Repair ...");
                if(logs[0].equals(module))
                {
                        for (int i = 0; i < logs.length; i++)
                        {
                        String x = logs[i];
                        if(x.contains("Repair"))
                        {
                                System.out.println("Reapir Sucessful! \n");
                                return 0;
                        }
                    }
                                System.out.println("Reapir Failed! Take to mechanic when
possible!\n");
                                return -1;
                }
                else
                {
                        System.out.println("Reapir Failed! Take to mechanic when
possible!\n");
                        return -1;
                }
        }
        return 0;
}

}
```

7.9 sensorAi.java

```
public class sensorAI {

    int isOn = 0; //0 = off, 11 = on, off is default
    String[] compiledSensorData = new String[50]; // allocated space for
sensor data to be exported
    int Connected = 0; // 0 = not connected, 1 = connected
    int Functional = 0; // 0 = not functional, 1 = functional
```

```
    public sensorAI(int connected, int functional) {
        isOn = 1;
        Connected = connected;
        Functional = functional;
        System.out.println("Enabling Sensor AI...");
        // call methods to determine if Sensor AI should turn off
        isConnected;
        isFunctional;
    }

    public int isConnected() {
        if (Connected == 0) {
            System.out.println("Error: Sensor AI failed to connect to
system.");
            isOn = 0;
            return 1;
        } else {
            System.out.println("Sensor AI successfully connected to
system.");
            return 0;
        }
    }

    public int isFunctional() {
        if (Functional == 0) {
            System.out.println("Error: Problem occurred in Sensor AI damage
diagnosis.");
            isOn = 0;
            return 1;
        } else {
            System.out.println("Sensor AI passed diagnosis examination.");
            return 0;
        }
    }

    public String[] sensorDataFusion(string[] localizationSensors, string[]
perceptionSensors) {
        // pulls array of functional sensors and array of perception
sensors into a single array to export
        // to lane mitigation, self driving, self parking, etc.
        for (int i = 0; i < localizationSensors.length; i++) {
```

```
            compiledSensorData[i] = localizationSensors[i];
        }
        for (int j = 0; j < perceptionSensors.length; j++) {
            compiledSensorData[j+i] = perceptionSensors[j];
        }
        return compiledSensorData;

}
```

7.A solarPanels.java

```
import java.util.concurrent.TimeUnit;
import java.text.DecimalFormat;
import java.util.*;

public class solarPanels extends Car{
      int enabled; // panels. 0 = off. 1 = on
      double battery; // battery levels

      /*
       * solarPanels a = new solarPanels(75,0); a.checkBattery();
a.toggleSolar();
       * a.toggleSolar(); a.charge(); a.checkBattery();
       */
      public solarPanels(double c, int e) {
            enabled = e;
            battery = c;
      }

      public void toggleSolar() // checks global variable and switches
them. Waits for panels to retract/move.
      {
            if (enabled == 1) {
                  System.out.println("Retracting Solar Panels ...");
                  try {
                        TimeUnit.SECONDS.sleep(3);
                  } catch (InterruptedException e) {
                        System.out.print("Attempt failed! Please take to
mechanic! \n");
                  }
                  System.out.println("Solar Panels retacted! \n");
```

```
                enabled = 0;
            } else if (enabled == 0) {
                System.out.println("Exposing Solar Panels ...");
                try {
                    TimeUnit.SECONDS.sleep(3);
                } catch (InterruptedException e) {
                    System.out.print("Attempt failed! Please take to
mechanic! \n");
                }
                System.out.println("Solar Panels exposed! \n");
                enabled = 1;
            } else {
                System.out.println("Error detected! Attempting to reset
Solar Panels ...");
                try {
                    TimeUnit.SECONDS.sleep(3);
                } catch (InterruptedException e) {
                    System.out.print("Attempt failed! Please take to
mechanic! \n");
                }
                enabled = 0;
                System.out.println("Attempt sucessful! \n");
            }
        }

        public void charge() {
            Random rand = new Random();
            double charge = rand.nextDouble(); // generate random double
charge and rounds/formats it to two decimal points.
            charge = charge * 10;
            DecimalFormat df = new DecimalFormat("#.##");
            charge = Double.valueOf(df.format(charge));

            if (enabled == 1) // depending on enabled, will either add or
subtract charge from battery. Will
                                        // check if batter becomes > 100
or < 0
            {
                System.out.print("Charing ...");
                try {
                    TimeUnit.SECONDS.sleep(3);
                } catch (InterruptedException e) {
```

```
                        System.out.print("Charge failed! Please take to
mechanic! \n");
                }
                System.out.print(" Charged ");
                System.out.print(charge);
                System.out.print("% \n");
                if (battery + charge > 100) {
                        battery = 100;
                        System.out.println("Battery Charged");
                } else {
                        battery = battery + charge;
                }

          } else {
                System.out.print("Consuming ...");
                try { // "consumes" charge
                        TimeUnit.SECONDS.sleep(3);
                } catch (InterruptedException e) {
                        System.out.print("Attempt failed! Please take to
mechanic! \n");
                }

                System.out.print(" Consumed ");
                System.out.print(charge);
                System.out.print("% \n");
                if (battery - charge < 0) // check if battery would be
less than 0 and throws out message
                {
                        battery = 0;
                        System.out.println("Please Charge Battery!");
                } else {
                        battery = battery - charge;
                }
          }
     }

     public double checkBattery() {
          System.out.print("Battery Levels ");
          System.out.print(battery);
          System.out.print("% \n");
          return battery;
     }
```

```
}
```

7.B VirtualSideMirrors.java

```
import java.util.concurrent.TimeUnit;
import java.util.*;

/*
            virtualSideMirrors a = new virtualSideMirrors(115.5,1211.5,0);
        a.displayCamera();
        a.displayCamera();
        a.adjustCamera(-100.2, 0);
        a.adjustCamera(100.2, 0);
        a.adjustCamera(-100.2, 1);
        a.adjustCamera(100.2, 1);
 */
public class virtualSideMirrors extends Car{ //If angle > 100 or < 30
resets angle to default for both left and right. resets cameraOn if not a 0
or 1. cameraOn refers to on hud screen.

      double cleftAngle;
      double crightAngle;
      int cameraOn;
      double defaultAng = 65.0;
      public virtualSideMirrors(double camAngle1, double camAngle2, int
camera)
      {
            cleftAngle = camAngle1;
            crightAngle = camAngle2;
            cameraOn = camera;
            if(cleftAngle > 100)
            {
                  System.out.println("Designated angle for Left Mirror is
too wide! Resetting Mirror to defualt standard ... \n");
                  try {
                        TimeUnit.SECONDS.sleep(3);
                  } catch (InterruptedException e) {
                        System.out.print("Attempt failed! Please take to
mechanic! \n");
                  }
                  System.out.println("Left Side Mirror is now at a 65.0
```

```
degree angle! \n");
                  cleftAngle = defaultAng;
            }
            if(cleftAngle < 30)
            {
                  System.out.println("Designated angle for Left Mirror is
too acute! Resetting Mirror to defualt standard ... \n");
                  try {
                        TimeUnit.SECONDS.sleep(3);
                  } catch (InterruptedException e) {
                        System.out.print("Attempt failed! Please take to
mechanic! \n");
                  }
                  System.out.println("Right Side Mirror is now at a 65.0
degree angle! \n");
                  cleftAngle = defaultAng;
            }
            if(crightAngle > 100)
            {
                  System.out.println("Designated angle for Right Mirror is
too wide! Resetting Mirror to defualt standard ... \n");
                  try {
                        TimeUnit.SECONDS.sleep(3);
                  } catch (InterruptedException e) {
                        System.out.print("Attempt failed! Please take to
mechanic! \n");
                  }
                  System.out.println("Virtual Side Mirrors are now at a
65.0 degree angle! \n");
                  crightAngle = defaultAng;
            }
            if(crightAngle < 30)
            {
                  System.out.println("Designated angle for Right Mirror is
too acute! Resetting Mirror to defualt standard ... \n");
                  try {
                        TimeUnit.SECONDS.sleep(3);
                  } catch (InterruptedException e) {
                        System.out.print("Attempt failed! Please take to
mechanic! \n");
                  }
                  System.out.println("Virtual Side Mirrors are now at a
```

```
65.0 degree angle!");
                  crightAngle = defaultAng;
            }
            if(cameraOn != 0)
            {
                  if (cameraOn != 1)
                  {
                  System.out.println("State of Virtual Side Mirrors
Unknown! Resetting Mirrors to defualt ... \n");
                  try {
                        TimeUnit.SECONDS.sleep(3);
                  } catch (InterruptedException e) {
                        System.out.print("Attempt failed! Please take to
mechanic! \n");
                  }
                  System.out.println("Virtual Side Mirrors are now
disabled. Please re-enable if wanted! \n");
                  cameraOn = 0;
                  }
            }

      }

      public int adjustCamera(double angle, int cameraLR)  //takes double
angle and int CameraLR (0 = left, 1 = right) and +- camera angle. If angle
> 100 or < 30 resets angle to defualt
      {
            if(cameraLR == 0) //adjust left
            {
                  System.out.println("Adjusting Left Mirror ... \n");
                  cleftAngle = cleftAngle + angle;
                  if(cleftAngle > 100)
                  {
                        System.out.println("Designated angle for Left
Mirror is too wide! Resetting Mirror to defualt standard ... \n");
                        try {
                              TimeUnit.SECONDS.sleep(3);
                        } catch (InterruptedException e) {
                              System.out.print("Attempt failed! Please take
to mechanic! \n");
                        }
                        System.out.println("Virtual Side Mirrors are now at
```

```
a 65.0 degree angle! \n");
                        cleftAngle = defaultAng;
                  }
                  if(cleftAngle < 30)
                  {
                        System.out.println("Designated angle for Left
Mirror is too acute! Resetting Mirror to defualt standard ... \n");
                        try {
                              TimeUnit.SECONDS.sleep(3);
                        } catch (InterruptedException e) {
                              System.out.print("Attempt failed! Please take
to mechanic! \n");
                        }
                        System.out.println("Virtual Side Mirrors are now at
a 65.0 degree angle! \n");
                        cleftAngle = defaultAng;
                  }
                  System.out.print("Left mirror adjusted. Now at an angle
of ");
                  System.out.println(cleftAngle);
                  System.out.println("");
            }
            else //adjust right
            {
                  crightAngle = crightAngle + angle;
                  if(crightAngle > 100)
                  {
                        System.out.println("Designated angle for Right
Mirror is too wide! Resetting Mirror to defualt standard ... \n");
                        try {
                              TimeUnit.SECONDS.sleep(3);
                        } catch (InterruptedException e) {
                              System.out.print("Attempt failed! Please take
to mechanic! \n");
                        }
                        System.out.println("Virtual Side Mirrors are now at
a 65.0 degree angle! \n");
                        crightAngle = defaultAng;
                  }
                  if(crightAngle < 30)
                  {
                        System.out.println("Designated angle for Right
```

```
Mirror is too acute! Resetting Mirror to defualt standard ... \n");
                        try {
                              TimeUnit.SECONDS.sleep(3);
                        } catch (InterruptedException e) {
                              System.out.print("Attempt failed! Please take
to mechanic! \n");
                        }
                        System.out.println("Virtual Side Mirrors are now at
a 65.0 degree angle! \n");
                        crightAngle = defaultAng;
                  }
                  System.out.print("Right mirror adjusted. Now at an angle
of ");
                  System.out.println(crightAngle);
                  System.out.println("");

            }
            return 0;
      }

      public int displayCamera() //0 = off. 1 = on. Takes current status of
cameraOn and switches them.
      {
            if (cameraOn == 0)
            {
                  System.out.println("Displaying mirrors to HUD ... \n");
                  try {
                        TimeUnit.SECONDS.sleep(3);
                  } catch (InterruptedException e) {
                        System.out.print("Attempt failed! Please take to
mechanic! \n");
                        return 1;
                  }
                  System.out.println("Virtual Side Mirrors are being
displayed on the hud! \n");
                  cameraOn = 1;
                  return 0;
            }
            else
            {
                  System.out.println("Turning off mirror feed on HUD ...
\n");
```

```
                  try {
                        TimeUnit.SECONDS.sleep(3);
                  } catch (InterruptedException e) {
                        System.out.print("Attempt failed! Please take to
mechanic! \n");
                        return 1;
                  }
                  System.out.println("Virtual Side Mirrors are no longer
being displayed on the HUD! \n");
                  cameraOn = 1;
                  return 0;
            }
      }
}
```