Software Architecture Document

Architectural Design Project

Autonomous Auto

By:

Angaar Hamid

Mach Bui

Prateek Gupta

Ethan Damian Schulte

Aren Laure Lizardo

Michael Souri

Eric Cheyne

Ixis Weiss

Benjamin Filler

11/26/2022

Revision History

Date	Version	Description	Author
9/25/2022	1.0	Drafted outline	angaar
10/02/2022	1.1	add views	angaar, mach, aren
10/04/2022	2.0	modified logical view	aren, mach
10/10/2022	2.1	modified 3 views	ixis, ben, eric, prateek, angaar
10/15/2022	2.2	finalized process and deployment view	aren, ixis, mach, eric, michael, ethan
10/16/2022	3.0	finishing artifacts	ben, prateek, mach
10/17/2022	3.1	adding subtitles	aren, michael, ethan
10/18/2022	3.2	touch-ups	angaar, mach
10/20/2022	4.0	Adding communication diagram outline	mach, eric
10/21/2022	4.0.1	state diagram adjustments	mach
10/24/2022	4.1	Planning communication diagram	mach, eric
10/28/2022	4.2	Added more communication diagrams	mach, eric
11/05/2022	5.0	Fixed sequence diagrams	ethan, michael
11/07/2022	5.1	modified sequence and communication diagrams, Fixed Class and Package Diagrams	michael, ethan, eric, mach, prateek, ben
11/08/2022	5.1.1	Modified deployment diagrams	arin, ixis
11/12/2022	5.2	Modified deployment diagrams	arin, ixis
11/16/2022	6.0	Finished deployment diagrams	arin, ixis
11/18/2022	6.1	worked on architectural design principles and worked on quality page	arin, mach
11/24/2022	6.1	Work on quality page	arin, mach
11/26/2022	6.2	Added another use case diagram and fixed component diagram	angaar

Table Contents

1. Introduction

1.1 Purpose

This document is aimed to provide a comprehensive architectural overview of Autonomous Auto's Navigation and Automatic Steering System (NASS) for the Sonny Motors Company's automobile lines. It is intended to capture and convey the significant architectural decisions which have been made to system requirements provided by Sonny Motors Company to develop the NASS.

1.2 Scope

This Software Architecture Document provides an architectural overview of the NASS. This will be done by covering the architecture of the NASS through the hardware and software of its in-car components and mobile components.

1.3 Definitions, Acronyms, and Abbreviations

NASS – Navigation and Automatic Steering System

USC – Unique Identifier for Use-Case Diagram

STD - Unique Identifier for State Diagrams

CLD - Unique Identifier for Class Diagrams

COD - Unique Identifier for Component Diagrams

PAD - Unique Identifier for Package Diagrams

ACD - Unique Identifier for Activity Diagrams

SED - Unique Identifier for Sequence Diagrams

CMD - Unique Identifier for Communication Diagrams

DEP - Unique Identifier for Deployment Diagrams

1.4 References

"All You Need to Know About State Diagrams." All You Need to Know about State Diagrams, Visual Paradigm, 2022, https://www.visual-paradigm.com/guide/uml-unified-modeling-language/about-state-diagrams/.

Athuraliya, Amanda. “Activity Diagram Tutorial: How to Draw an Activity Diagram.” Creately Blog, Cinergix Pvt. Ltd, 29 Sept. 2022, https://creately.com/blog/diagrams/activity-diagram-tutorial/.

“UML Activity Diagram Tutorial.” Lucidchart, Lucid Software Inc, 2022, https://www.lucidchart.com/pages/uml-activity-diagram.

About the Unified Modeling Language Specification Version 2.1.2, https://www.omg.org/spec/UML/2.1.2/About-UML/.

Parashar, Dhruv, et al. “Self-Driving-Car/Sds.md at Master · Geekyshiva/Self-Driving-Car.” Github, NIIT University, 1 Dec. 2017, https://github.com/GeekyShiva/Self-Driving-Car/blob/master/Project%20Docs/SDS.md.

“UML Class Diagram Tutorial.” Lucidchart, https://www.lucidchart.com/pages/uml-class-diagram.

“Vehicle Management System Class Diagram.” FreeProjectz, 2 Aug. 2017, https://www.freeprojectz.com/uml/vehicle-management-system-class-diagram.

Wikipedia contributors. "Package diagram." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 2 Jun. 2021. Web. 20 Oct. 2022.

“All about UML package diagrams.” Lucidchart, https://www.lucidchart.com/pages/uml-package-diagram.

“Sequence Diagram.” Visual Paradigm, https://www.visual-paradigm.com/learning/handbooks/software-design-handbook/sequence-diagram.jsp.

1.5 Overview

This Software Architecture Document has been organized to display the “4+1” View Model of the NASS. The Use-Case view will be introduced first, where the sequence of interactions a user may have with NASS through use-case diagrams are highlighted. Then, the Logical View will be displayed, where the structure of the NASS through state and class diagrams are described. After the logical view, the Development View will be introduced, where the system from a developer’s perspective using the component and package diagrams will be illustrated. After the Development View, the Process View will be displayed. Where it is defined how the software will be executed through sequence, communication, and activity diagrams. After the Process View the Physical View will be displayed, encompassing the use cases and sequences of interactions between objects and processes through deployment diagrams. The document will be concluded by presenting the estimated Size and Performance and Quality of the NASS, that describes the characteristics of the architecture and how the software architecture contributes to all capabilities of the NASS.

2. Architectural Representation

This document presents the NASS architecture as a series of views in the following order: Use-Case View, Logical View, Development View, Process View, and Physical View. The implementations are already described in this document.The following diagrams are used to convey the views: use-case diagrams, state diagrams, class diagrams, package diagrams, component diagrams, sequence diagrams, communication diagrams, activity diagrams, and deployment diagrams.

3. Architectural Goals and Constraints

There are some key requirements and system constraints that have a significant bearing on the NASS architecture. They are:

3.1. System shall consist of reliable redundant components for fail-safe operation. In the event that one component of the NASS system malfunctions or is compromised, it will need to be immediately replaced while some features are disabled.

3.2. To attain a high-level of obstacle detection, we shall employ redundant sensor configurations all around the car along with the inside of the car. This is to ensure our NASS is as accurate and precise as possible.

3.3 System shall be designed to make the most of the existing sensor configurations in the vehicle with peripheral radars, front cameras, panoramic views, and rear-view cameras for monitoring.

3.4 The physical servers running web servers and databases responsible for the NASS must be in LINUX.

3.5 The physical servers running must be free from intrusions and have capable physical security, otherwise compromising the integrity of our cybersecurity.

3.6 Vehicle will not fight the driver for control, if the driver handles the steering wheel or steps on the pedals, all functions will cease until driver gives back control.

4. Use-Case View

4.1 Use-Case Diagrams

These use-case diagrams describe the high-level interactions of the user to the NASS. It covers the requirements listed in the requirements sheet provided by Sonny Motors Company's automobile lines. It covers the driver starting to drive, how interactions with the user might go when an obstacle is detected, when a route is changed, when a turn is required, etc. It does not describe how the system operates internally, but it describes what the NASS does and how the drivers use it.

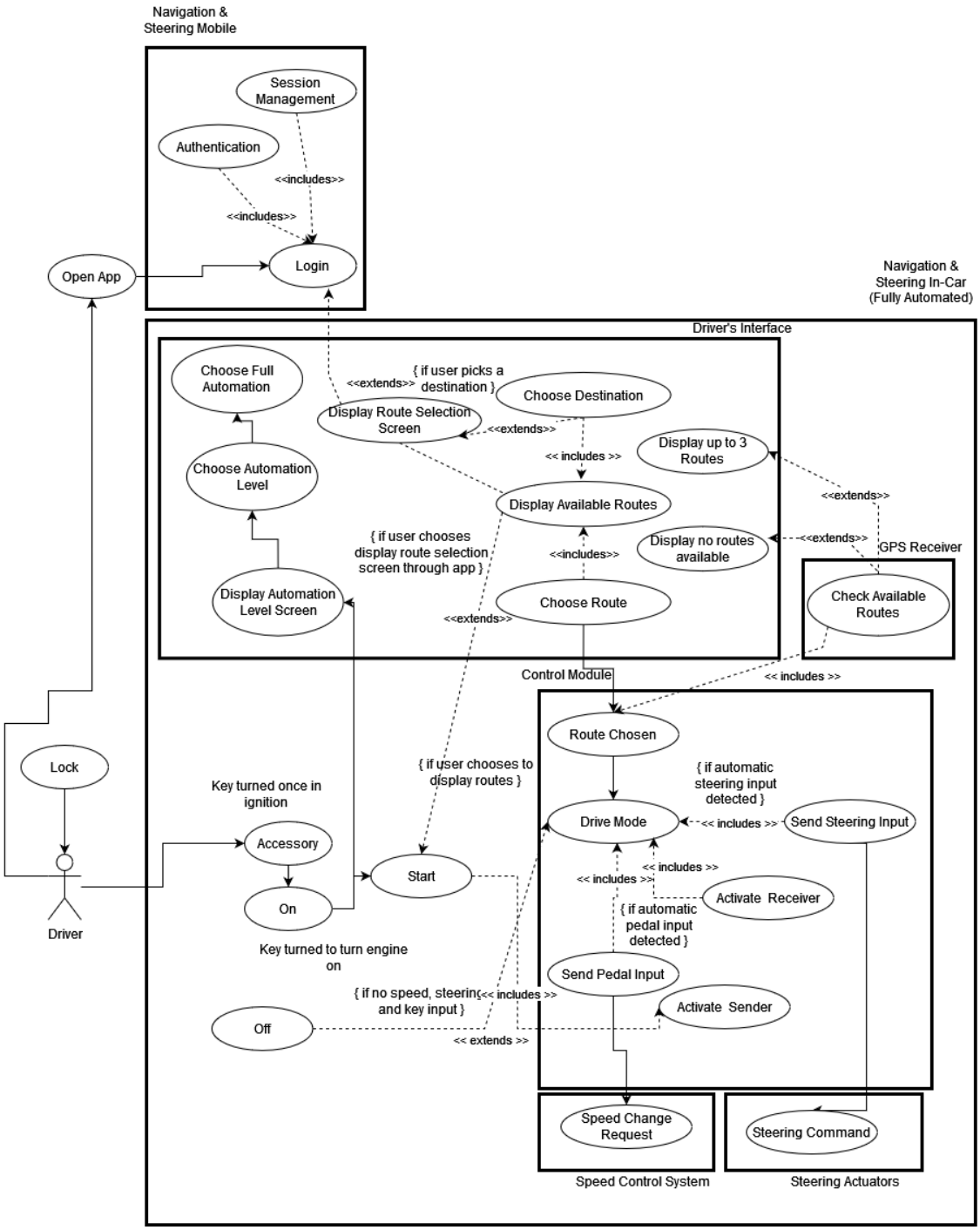

Figure 4.1.1 – UC01 (Start Driving)

4.1.1.1 – Driver

The driver will take part in a series of actions before being able to engage with the NASS. The following actions are putting the key in ignition, turning to accessors, turning the key fully, and turning the car on. The driver also has the option to open their mobile app to access the GPS interface.

4.1.1.2 – Driver starts the car

The drive will start the car here and be presented with a series of options. One, upon starting the car, the sensors will activate allowing the obstacle avoidance features to be active. Two, the driver will have routes displayed to them through the interface where they must select the routes.

4.1.1.3 – Driver chooses route

The driver will choose a route and automation level. The automation level, in this diagram, will be fully automated. The driver will have to choose a destination and, in return, will have routes displayed. The same interface here is available through the mobile app.

4.1.1.4 – Route has been chosen

Upon choosing a route, the NASS will take into effect and engage the car in steering and speed control to get to the destination. This will require the use of ‘Activating’ the receivers, sending ‘Steering Commands’, and sending ‘Speed Change Requests’.

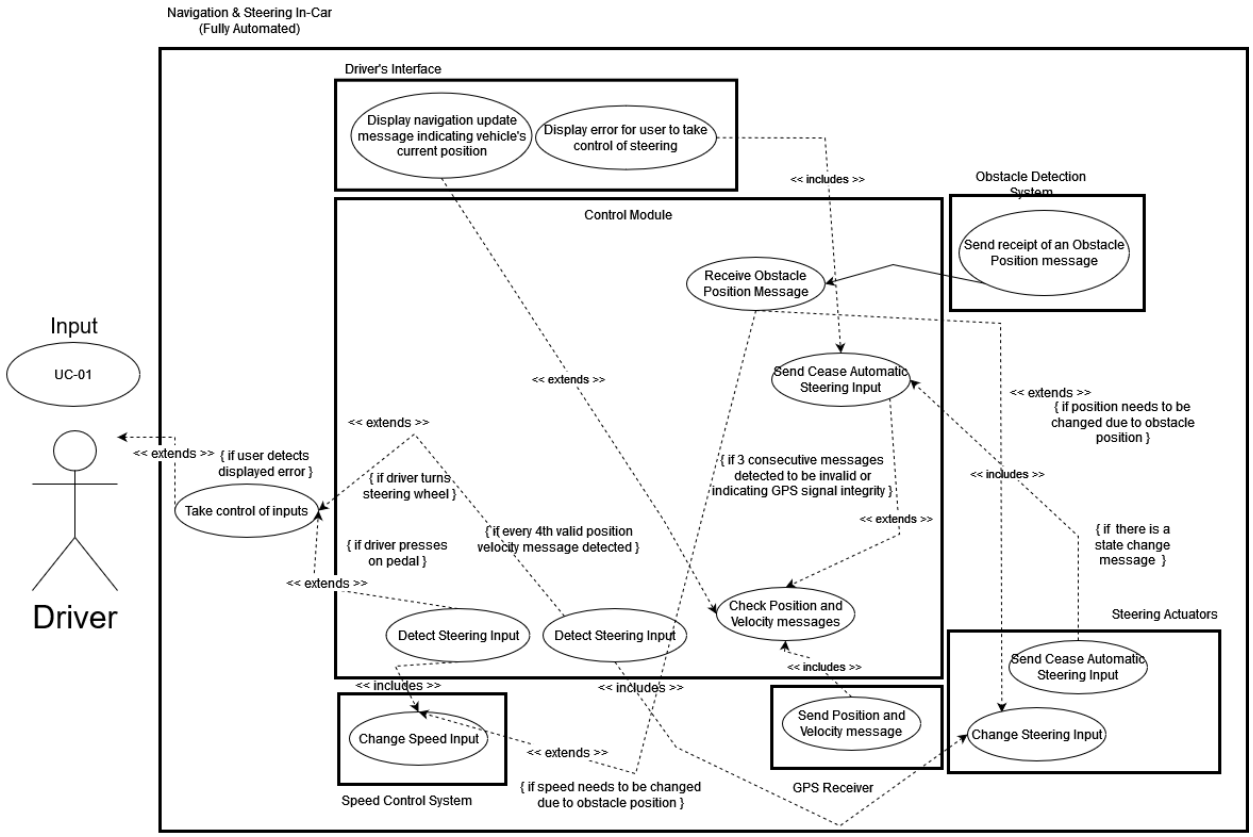

Figure 4.1.2 – UC02 (Position and Velocity Sensing)

4.1.2.1 – Driver

The prerequisites have been established above the driver. At this point, the driver should have already finished UC-01 and is in the process of driving the car.

4.1.2.2 – Check Position and Velocity Messages

While the car is driving, the control module will frequently check position and velocity messages.

4.1.2.3 – “If 3 consecutive messages decided to be invalid or indicting GPS signal integrity”

If the control module catches three consecutive invalid or low integrity position and velocity messages, the control module will crease automatic steering input. This is done by sending a “Change Steering Input” command to the steering actuators and a “Change Speed Input” to the speed control system.

4.1.2.4 – “Display error for user to take control of driving”

The NASS will notify the driver beforehand to take control of automatic steering in the event that the automatic steering ceases.

4.1.2.5 – Take control of Inputs

Driver will be expected to take control of the inputs for the car and this is detected through sensors in the steering wheel that watches out for steering input.

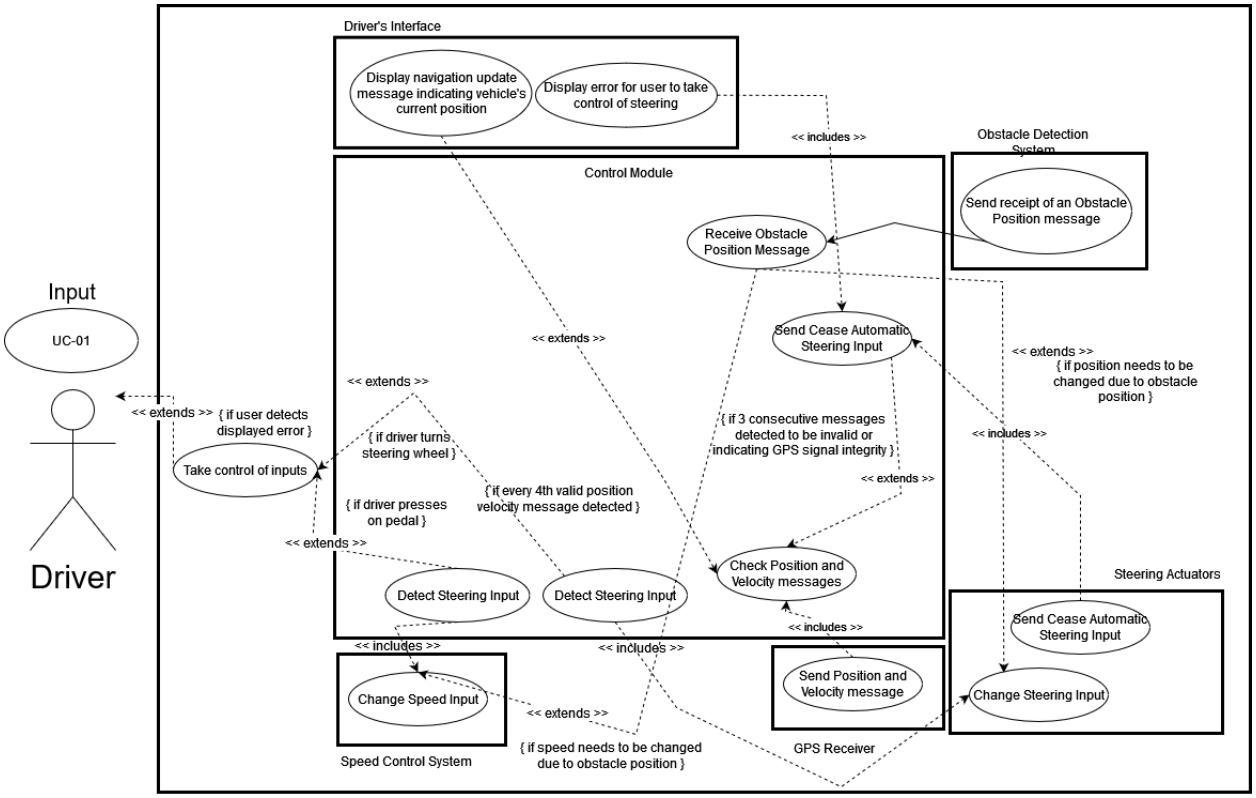

Figure 4.1.3 – UC03 (Route Selection)

4.1.3.1 – Driver

The prerequisites have been established above the driver. At this point, the driver should have already finished UC-01 and is in the process of driving the car. This use-case is like the previous one, however it addresses the Route Selection feature of the NASS.

4.1.3.2 – Display navigation update message indicating vehicle's current position.

While the car is driving, the control module will frequently check position and velocity messages. If there are four valid position velocity messages detected by the control module from the GPS Receiver, then we will display a navigation update message indicating the vehicle's current position to the Driver's Interface.

4.1.3.3 – Send receipt of an Obstacle Position Message

This state will be touched upon in UC05.

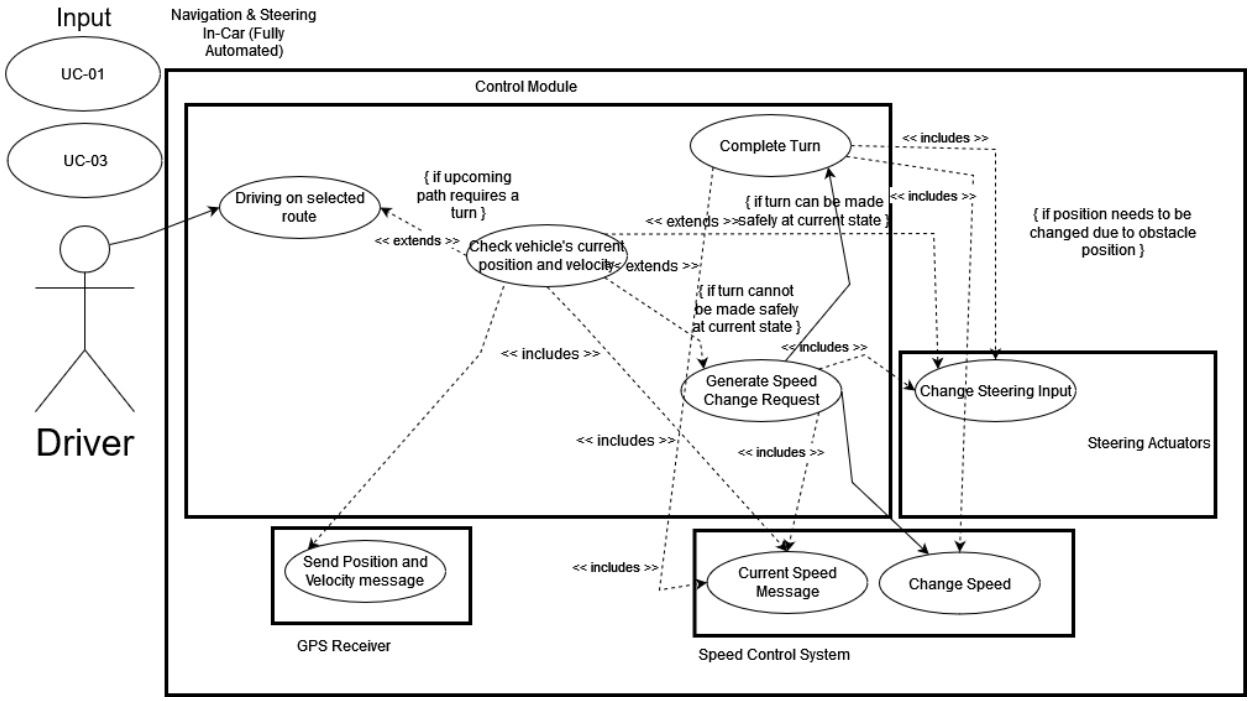

Figure 4.1.4 – UC04 (Steering)

4.1.4.1 – Driver

The prerequisites have been established above the driver. At this point, the driver should have already finished UC-01 and is in the process of driving the car. The driver must also be engaged in route selection and receiving position velocity messages.

4.1.4.2 – Check vehicle's current position and velocity

The control module will check current speed messages from the speed control system, and it will receive position and velocity messages from the GPS receiver. Upon doing so, if it detects a route that

requires a turn it will make the decision of whether it can safely turn at the current speed or not. If it can turn at the current state, then it will send a ‘Change Steering Input’ command to the steering actuators.

4.1.4.3 – Change Speed

In the event that the control module decides that the upcoming turn cannot be made safely at the current speed, then it will generate a Speed Change Request to the Speed Control System, it will note down the current speed message, and it will send a Steering Change Request to the steering actuators.

4.1.4.4 – Complete Turn

Once the control module detects that the turn has been made, it will generate one or more Speed Change Requests to return the vehicle to its appropriate speed after a turn has been completed.

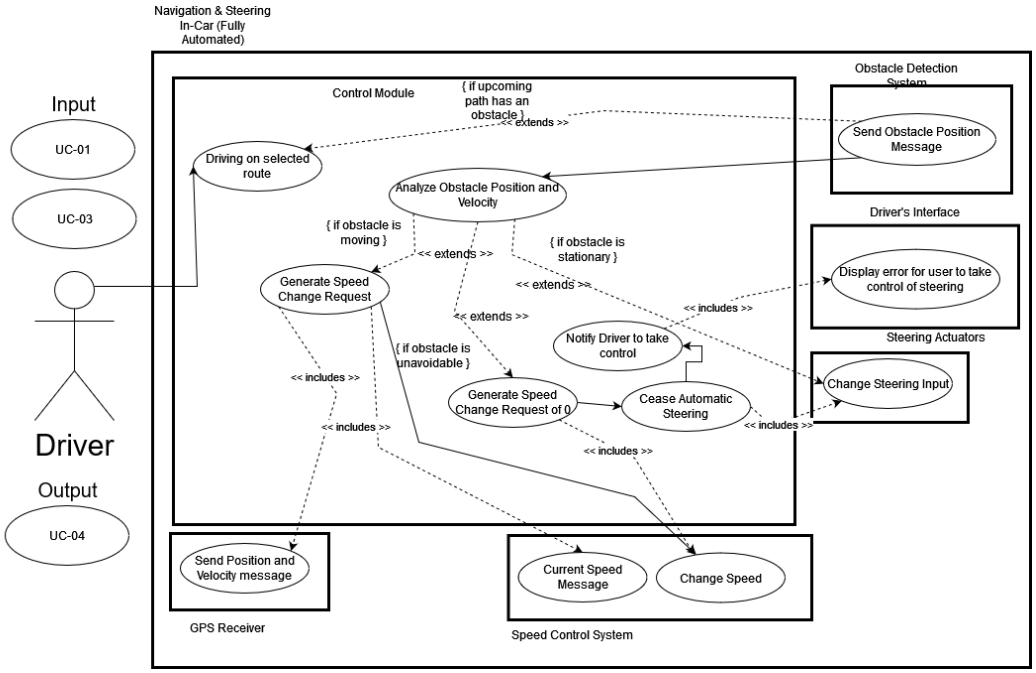

Figure 4.1.5 – UC05 (Obstacle Avoidance)

4.1.5.1 – Driver

The prerequisites have been established above the driver. At this point, the driver should have already finished UC-01 and is in the process of driving the car. The driver must also be engaged in route selection and receiving position velocity messages. The driver will be expected to take control of the steering, so the expected output, or result, of this use-case is UC04.

4.1.5.2 – Send Obstacle Position Message

The control module will receive obstacle position messages from the control module. In the event that an obstacle is detected, the control module will analyze the obstacle to determine if it is moving or avoidable. If the obstacle is moving, then the driver will generate a speed change request to reduce the vehicle's speed to keep the moving obstacle at a safe distance. If the obstacle is not moving, then the control module will send a Change Steering Input command to the steering actuators to maneuver the obstacle.

If the obstacle is unavoidable, then the control module will cease automatic steering, generate a speed change request of 0 to apply a brake, and notify the driver to take control of the manual steerigng. The control module will check current speed messages from the speed control system, and it will receive position and velocity messages from the GPS receiver. Upon doing so, if it detects a route that requires a turn it will make the decision of whether it can safely turn at the current speed or not. If it can turn at the current state, then it will send a 'Change Steering Input' command to the steering actuators.

5. Logical View

5.1 Overview

For the Navigation and Steering System (NASS) we used a state diagram to describe how the user would interact with our system, and how the system will provide information to the user. The diagram detailed the Driver's interface whilst depicting the information being exchanged through overall representatives of the different systems connected to the Control Module. We included a mobile application and how the user will interact with it to connect to the NASS.

5.2 State Diagram

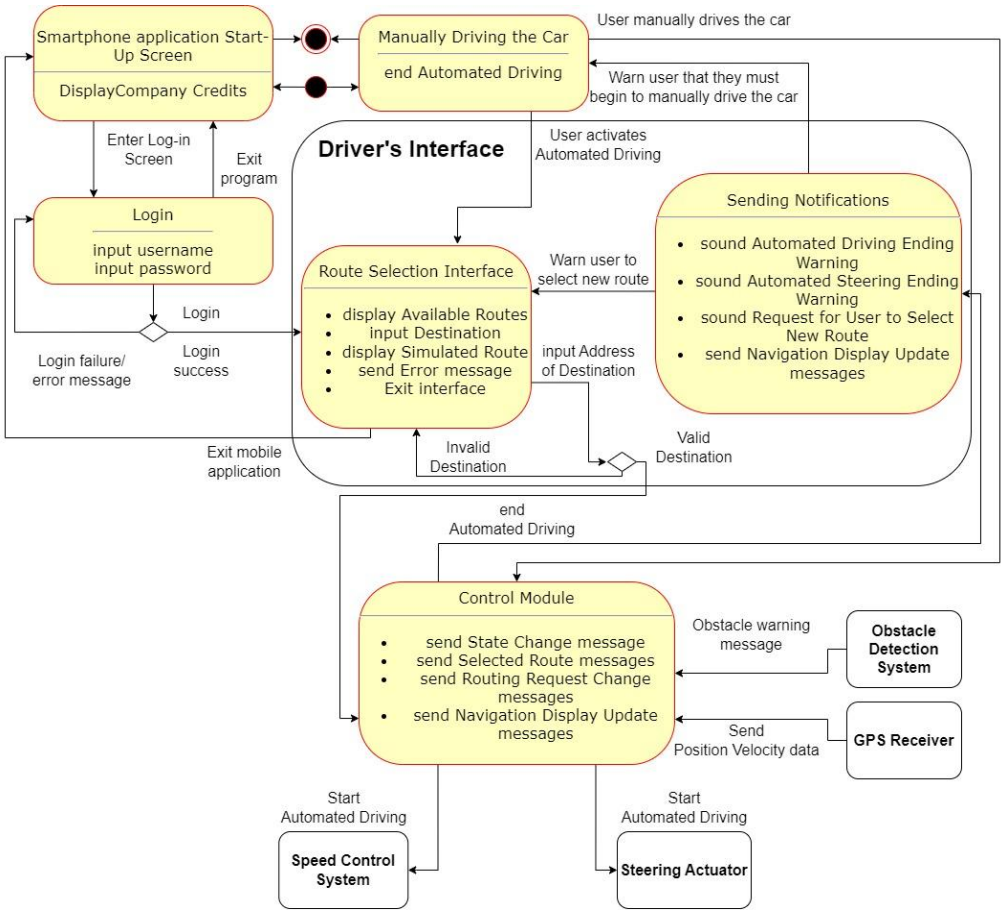

Figure 5.2.1 – SD01 (State Diagram)

5.2.1.1 – Login State

Allows the driver to login into the autonomous drive and access the GPS interface and goes into the GPS interface state. The driver has the ability to login through their smartphone and input their route and send it to the autonomous system

5.2.1.2 - GPS Interface State

This state allows the driver to input a route for the autonomous drive to simulate and execute, going into the control module state. The driver can also change the route anytime, sending Routing Request Change to the Control Module. The GPS can determine up to 3 adaptable routes when driver enters the route and sends Available Routes messages to the Control Module. If there are no routes, the Available Routes messages will tell the Control Module there are none, going into the Warning state. Once selected, a “Valid Destination” message will be sent to the control module and the car will start driving, changing the State Change message to “On” from the Control Module to indicate that the car is driving and in autonomous drive

5.2.1.3 – Warning State

This state receives messages from all the states to ensure there are no errors. If so, the system immediately goes into the warning state and displays the errors to the driver.

5.2.1.4 – Manual Mode State

Manual mode allows the driver to take full control of the vehicle. Can be accessed anytime when the driver takes the steering wheel. No exceptions. Can only be done when control module’s State Change message is “On”.

5.2.1.5 – Control Module State

The control module checks the State Change message. If the State Change message is “On”, the car is in drive. State Change simply changes the state from autonomous drive to manual drive when receiving the “Off” message. The control module receives the Selected Route, and Routing Request Change messages from the corresponding states. Selected Route message is received from the GPS interface state and Routing Change message happens when the driver wants to change the route from the GPS interface state. Available Route messages are received determining if there are routes and if not, goes into warning state.

5.3 Class Diagram

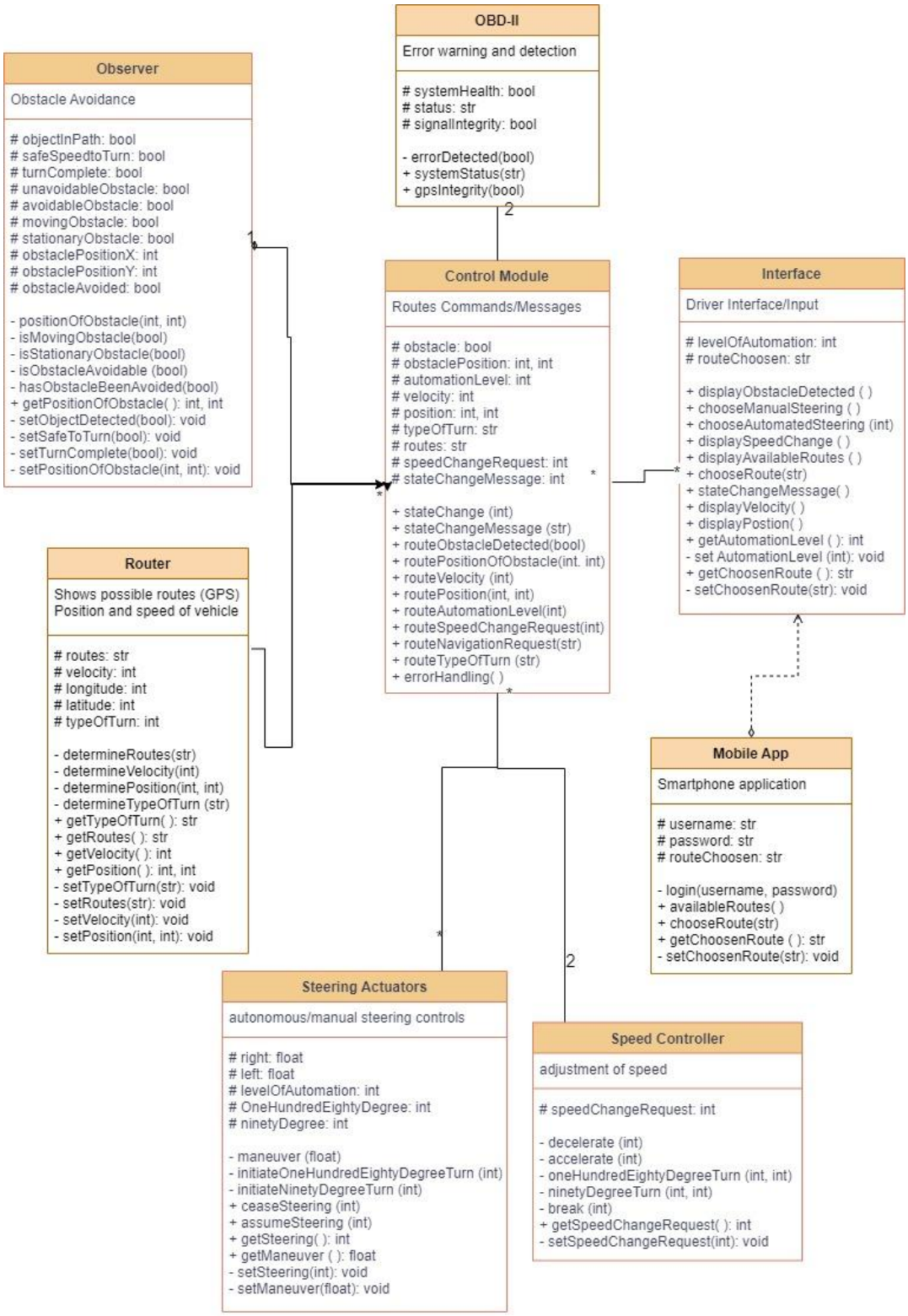

Figure 5.3.1 – CD01 (Class Diagram)

5.3.1.1 – Observer

As shown in Figure 5.3.1:

The function isObstaceAvoidable(bool) determines whether it is an avoidable or unavoidable obstacle.

The functions isStationaryObstacle(bool) or isMovingObstacleDetermine(bool) determines whether an obstacle is moving [SN-0012] or stationary [SN-0014].

The function setObjectDetected(bool) informs the driver interface that an obstacle has been detected.

The function positionOfObstacle(int, int) will send the position of the obstacle to the steering actuators and speed controller to change direction to maneuver around the obstacle should it be an avoidable obstacle.

If the function isObstaceAvoidable(bool) determines that it is an unavoidable obstacle, it will inform the driver that manual steering will be engaged [SN-0017], and the driver must take control of steering [SN-0037]. It will then inform the steering actuators to cease steering.

The function setSafeToTurn(bool) determines whether the car is at a safe speed to initiate a turn [SN-0010] and whether the turn is completed [SN-0011].

The function hasObstacleBeenAvoided(bool) determines whether an obstacle has been successfully avoided

5.3.1.2 – Router

As shown in Figure 5.3.1

The function determineRoutes(str) is called when automated driving is selected by the driver, a navigation request is sent to the router to determine three possible routes for the system to follow [SN-005]. It will then send an update to the driver interface to choose a route.

The function determineVelocity(int) and determinePostiton(int, int) will also continuously check the velocity of the car and its position using the GPS system. The function will then update the driver on the car's position and velocity [SN-001] [SN-002]. It will also continuously inform the observer of the velocity so it can determine whether it can safely make a turn

5.3.1.3 – Steering Actuators

As shown in Figure 5.3.1

The function assumeSteering(int) or ceaseSteering(int) receives a command from the driver’s interface determining the level of automation, whether they want manual or fully automatic steering [SN-008].

The function maneuver(float) makes decisions about turns based on the position, velocity, and route chosen [SN-009]. It will also make turns if the observer detects an obstacle, and a turn needs to be made to avoid it.

5.3.1.4 – Speed Controller

As shown in Figure 5.3.1

The function getSpeedChangeRequest(int) adjusts speeds using a Speed Change Request. If a turn or obstacle is detected by the observer, it will send a speed change request to the speed controller to decelerate to a determined speed [SN-0015]. When the turn is completed or the obstacle has been avoided, another speed change request will be sent to the speed controller to accelerate to the speed limit.

The functions oneHundredEightyDegreeTurn(int) or ninetyDegreeTurn(int) is used If a ninety or one hundred eighty degree turn is detected it will slow down to a specific range of speeds so the steering actuators can make a safe turn.

5.3.1.5 – OBD-II

As shown in Figure 5.3.1

The OBD-II system continuously monitor the system and its various classes to check for errors in the automatic driving system

The function systemStatus(str) will send continuous system status messages to the driver interface.

The function errorDetected(bool) is used if an error is detected and will cease automatic steering and inform the user to assume manual steering [SN-0022].

The function gpsIntegrity(bool) is used to check if an error in the GPS signal integrity is detected and automatic steering will cease [SN-003].

5.3.1.6 – Control Module

As shown in Figure 5.3.1

The function routeVelocity(int) and routePosition(int, int) routes update messages such as position and velocity every time it is updated to the driver interface [SN-004].

The functions routeAutomationLevel(int), routeSpeedChangeRequest(int), and routeNavigationRequest(str) handles routing various requests made by the system such as speed change requests [SN-0019] and the level of automated driving to the speed and steering controllers as well as navigation requests.

The functions routeObstacleDetected(bool) and routePositionOfObstacle(int, int) handle informing the drivers interface that an obstacle has been detected and inform the steering actuators and speed controllers to maneuver and change speed accordingly to avoid the obstacles.

The function routeTypeOfTurn(str) will inform the steering actuators and speed controllers whether a 90 degree or 180 degree is being initiated based on information from the observer.

The functions stateChange(int) and stateChangeMessage(str) will also generate state changes to inform the user [SN-0033] [SN-0020].

The function errorHandling() will handle error handling that is generated by the OBD-II system

5.3.1.7 – Interface

As shown in Figure 5.3.1

The functions displayObstacleDetected(), displaySpeedChanges(), displayAvailableRoutes(), stateChangeMessage(), displayVelocity(), and displayPosition() will display to the user: Obstacles detected, Speed changes, Available routes [SN-006] [SN-007], State changes [SN-0018], and Velocity/Position of the vehicle.

The function chooseRoutes allows the user to choose from 3 route of options

The function chooseAutomatedSteering(int) allows the user to determine the automation level of steering and what route to take.

The function stateChangeMessage() will also inform the user to take over manual steering [SN-0030].

Mobile App

The function login(str, str) allows for the user to login with their username and password.

The functions availableRoutes() and chooseRoutes(str) displays to the user 3 available routes and allows for them to choose routes for the automatic driving system to follow

6. Development View

6.1 Overview

The package diagram is meant to visualize the various elements of the architecture and their relationships to each other. Each box is a “package” that envelopes a group of similar classes or functions. The arrows between packages represent a dependency or a requirement that a package draws from. All together, the package diagram aims to represent the organization of the most important elements of the software architecture.

6.2 Package Diagram

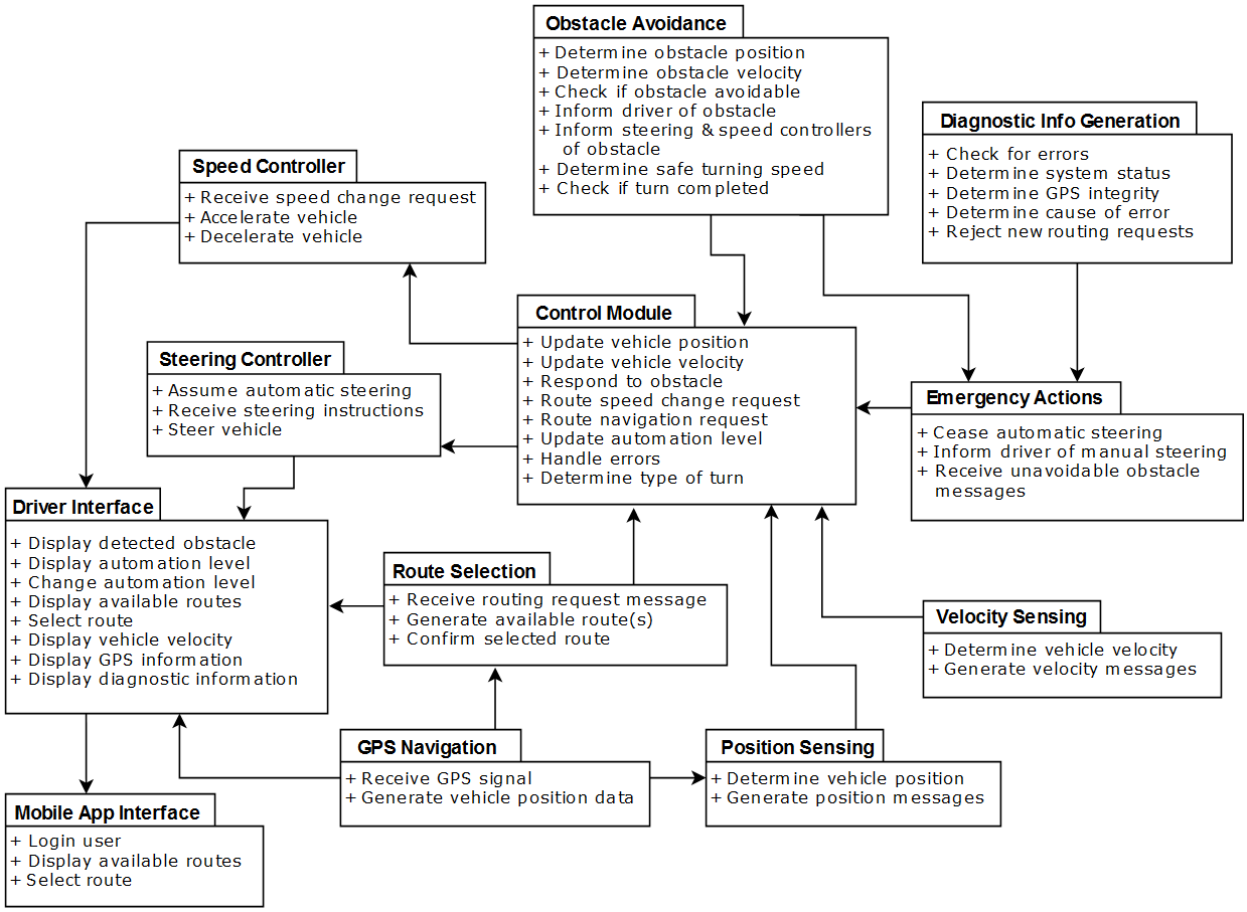

Figure 6.2.1 – PAD01 (Package Diagram)

6.2.1.1 - Control Module

As shown in Figure 6.2.1, the control module's main purpose is to route the messages sent by modules to their intended destination. The control module routes position & velocity messages sent by the position & velocity sensing packages to the driver interface, it routes changes in speed and automation level sent by the driver to the speed and steering controllers, and it routes detected obstacle messages from the obstacle avoidance package to the driver interface. Other functionalities include sending navigation requests to the steering controller, handling errors generated by the emergency actions module, and determining the type of turn to make.

6.2.1.2 - Obstacle Avoidance

As shown in Figure 6.2.1, the obstacle avoidance package detects obstacles in the path of the vehicle. It will determine the position and speed of the obstacle if it is moving, and then check if the obstacle is avoidable. It will then inform the driver interface that an obstacle has been detected. The obstacle avoidance package will send the position of the obstacle to the steering and speed controllers to maneuver around the obstacle should it be avoidable. For unavoidable obstacles, the package will inform the emergency actions package and inform the driver interface of the obstacle.

6.2.1.3 - Diagnostic Info Generation

As shown in Figure 6.2.1, the diagnostic info generation package is mainly used for error detection. This package will determine the status of the system, the integrity of the GPS navigation, and check the system for errors. If an error is found, the package will determine the cause of the error and inform the driver interface. If the error is severe, the package will notify the emergency actions package and subsequently reject any new routing requests.

6.2.1.4 - Emergency Actions

As shown in Figure 6.2.1, the emergency actions package receives messages from the obstacle avoidance or diagnostic info generation packages and responds accordingly. The package will inform the driver interface of the switch to manual steering and then deactivate the steering controllers.

6.2.1.5 - Velocity Sensing

As shown in Figure 6.2.1, the velocity sensing package detects the velocity of the vehicle and sends regular messages to the control module of the current velocity.

6.2.1.6 - Position Sensing

As shown in Figure 6.2.1, the position sensing package receives GPS data on the position of the vehicle and sends regular messages to the control module of the current position.

6.2.1.7 - Route Selection

As shown in Figure 6.2.1, the route selection package receives route information from the GPS navigation package, generates routes to the selected destination, and sends the selected route to the driver interface. The package will also send the navigation request to the control module.

6.2.1.8 - GPS Navigation

As shown in Figure 6.2.1, the GPS navigation package receives GPS satellite signal and generates vehicle position data based on it. It then sends the vehicle position data to the position sensing package.

6.2.1.9 - Steering Controller

As shown in Figure 6.2.1, the steering controller manages the automatic steering of the vehicle. The package will receive steering instructions from the control module and automatically steer the vehicle accordingly.

6.2.1.10 - Speed Controller

As shown in Figure 6.2.1, the speed controller manages the speed of the vehicle. The package will receive speed change requests from the control module and accelerate or decelerate the vehicle accordingly.

6.2.1.11 - Driver Interface

As shown in Figure 6.2.1, the drivers interface displays all of the essential & accessory information to the driver. The package will display: obstacles detected, speed changes, emergency steering notifications, error messages, available routes, vehicle velocity, vehicle position, and GPS navigation info. The driver interface also allows the driver to determine the level of automation.

6.2.1.12 - Mobile Interface

As shown in Figure 6.2.1, the mobile interface allows the user to interact with the drivers interface with their smartphone. The driver can login to the mobile app and view the available

routes on their phone. Once a route is chosen, the package sends the navigation request to the driver interface and control module.

6.3 Component Diagram

The focus of component diagrams is on self-standing, replaceable components, and how these can be assembled to make more complex systems. In this case, this diagram portrays how the modules interact with the control module and what application components and infrastructure components (COTS or not) will be required to complete such a diagram.

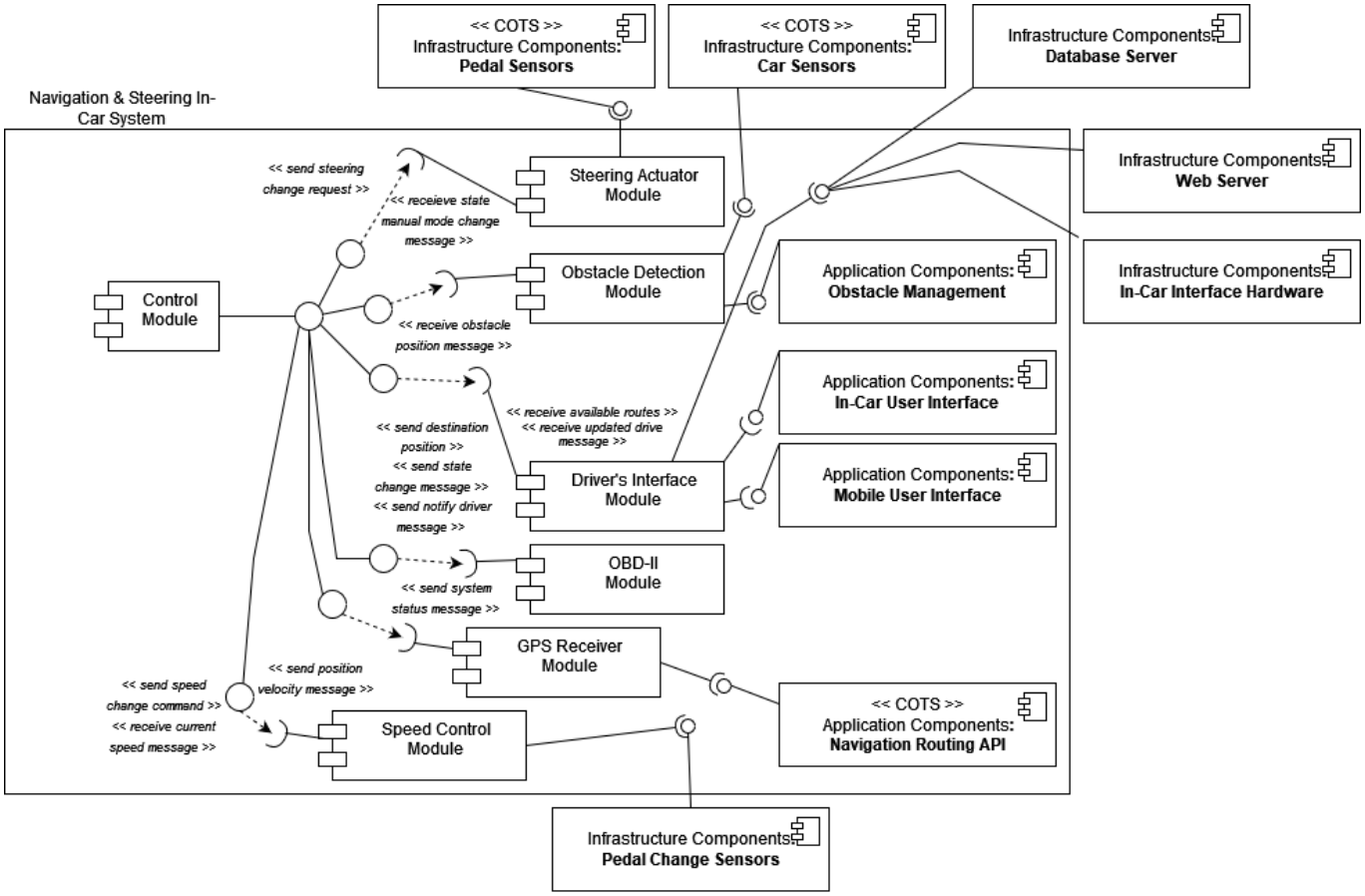

Figure 6.2.1 – CDa01 (Package Diagram)

6.2.1.1 – Control Module

The control module is the “ball” that is what all other modules requires for them to be functional. Each module is useless without the control module and the control module facilitates, and dictates, all communication between each module.

6.2.1.2 – Steering Actuator Module

The steering actuator module receives steering change requests from the control module and sends back state manual mode change requests. It is dependent on the COTS infrastructure component: Pedal Sensors. As the pedals in the car can also dictate how the steering actuator may behave. The COTS product will be outsourced to another company.

6.2.1.3 – Obstacle Detection Module

The obstacle detection module sends obstacle position messages to the control module. For this module to function, it is dependent on the application component: Obstacle Management and the COTS infrastructure component: Car Sensors. As the sensors in the car detect the obstacles and the obstacle management application manages the obstacles being detected from the back end.

6.2.1.4 – Driver's Interface Module

The driver's interface module receives destination positions, state change messages, and notify driver messages from the control module. In return, it sends, sends the available routes and updated driver messages. It is dependent on the application components: in-car user interface and mobile user interface. It is also dependent on the infrastructure components: database server, web server, and in-car hardware. The servers are used to manage information passing through the interface and the in-car hardware is used to detect all interactions of the user to the car. The application components are used to deal with the hardware from the back end.

6.2.1.5 – OBD-II Module

The OBD-II module receives system status messages from the control module. It is dependent on the infrastructure components: database server and web server. It deals with storing a lot of the status message that comes through the control module for archival and record purposes.

6.2.1.6 – GPS Receiver Module

The GPS receiver module receives position velocity messages from the control module. It is dependent on the COTS application component: Navigation API. The navigation and routing algorithms will be outsourced to a known company provided a reliable COTS algorithm that assists with these features.

6.2.1.7 – Speed Control Module

The speed control module receives speed change commands from the control module and send current speed messages to the control module. For it to function, it is dependent on the infrastructure components: Pedal Change Sensors. The pedal change sensors will affect how the speed control module behaves.

7. Process View

7.1 Overview

Over the course of the Scrum Process, we worked on the Process View. Specifically, I worked on the System Sequence Diagrams based on the SRS (3.1 - 3.3). Through a total of 9 diagrams, we used draw.io to construct them. Additionally, we wrote detailed descriptions concerning the diagrams for the S.A.D.

7.2 Activity Diagrams

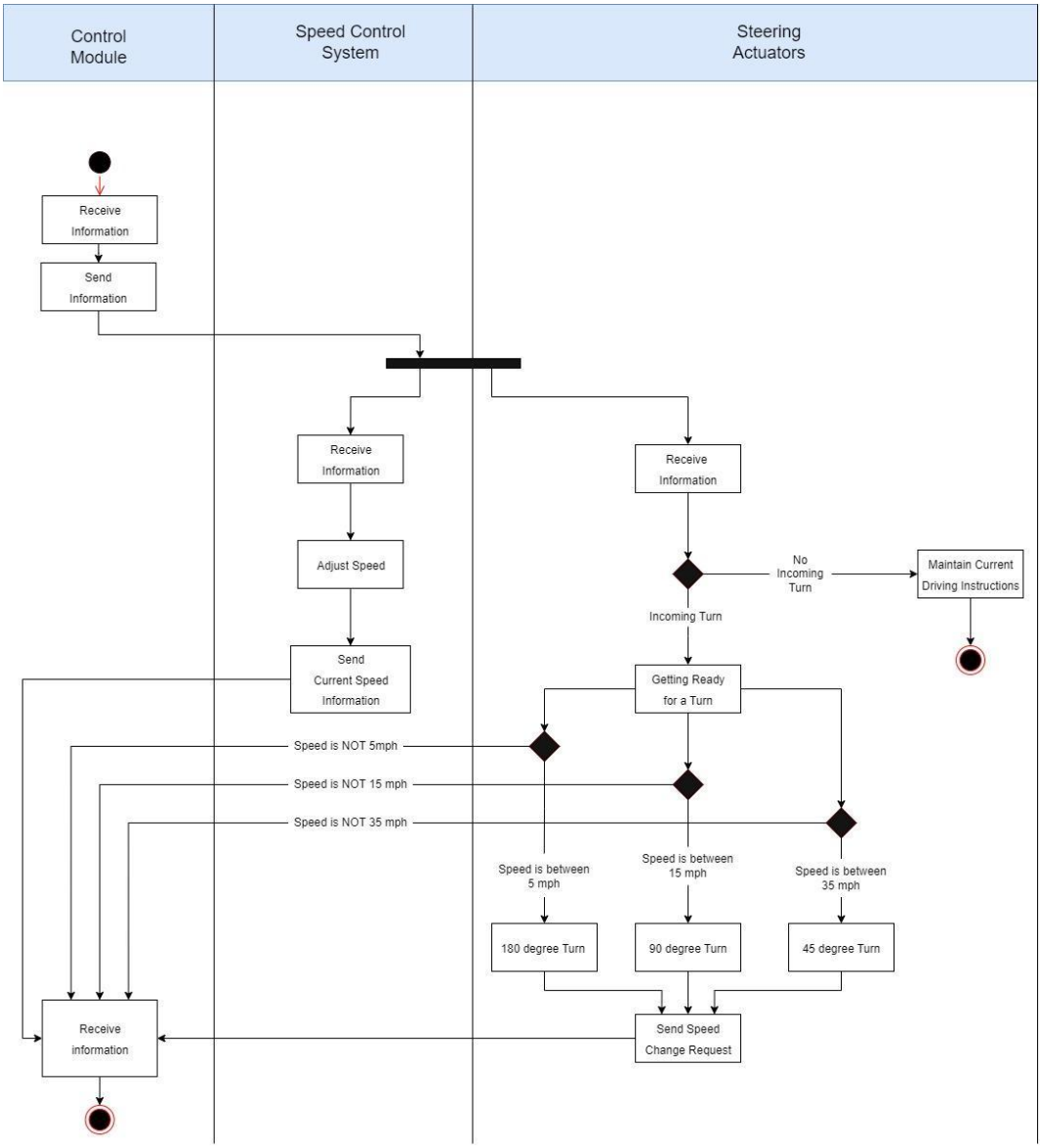

Figure 7.2.1 – AD01 (Steering Actuators & Speed Control System)

7.2.1.1 – Steering Actuators & Speed Control System

Speed Control System receives Speed Change Requests and adjusts speed. Route Selected Message sent to Steering Actuators to steer the car. GPS Receiver information influences speed and steering systems. Speed determined by upcoming turns. The speed returns to the normal speed after the turn is done.

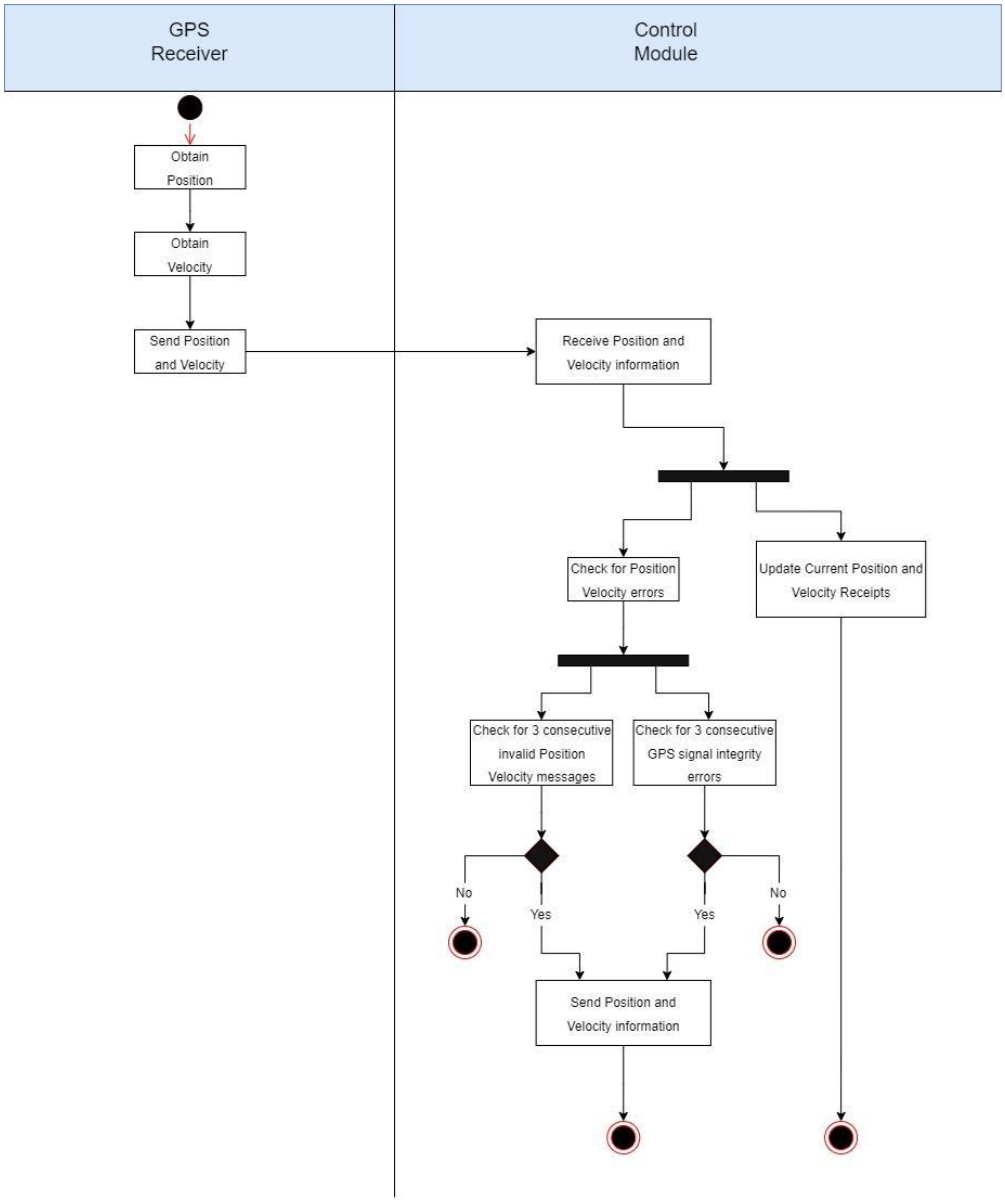

Figure 7.2.2 – AD02 (GPS Receiver)

7.2.2.1 – GPS Receiver

Update Position and Velocity data. End Automatic Steering if 3 consecutive invalid Positional Velocity messages are received within 2 seconds. End Automatic Steering if 3 consecutive GPS signal integrity errors are received within 2 seconds.

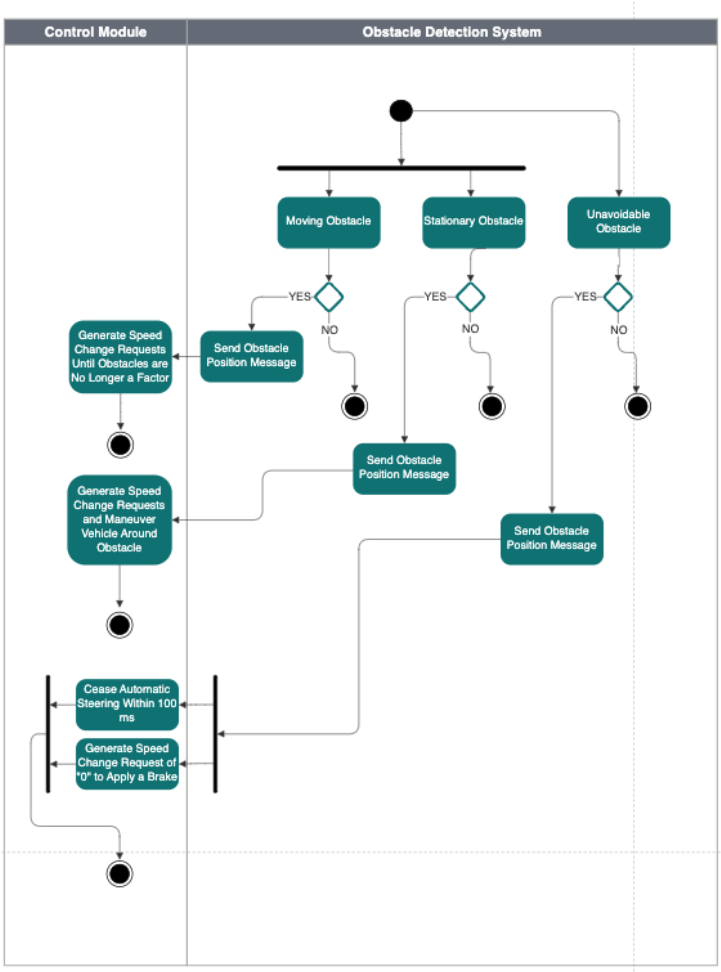

Figure 7.2.3 – AD03 (Obstacle Detection System)

7.2.3.1 – Obstacle Detection System

System detects moving obstacles, stationary obstacles, and unavoidable obstacles. If any of the obstacles are detected, obstacle position messages are sent to the control module. After the messages are sent to the control module, specific actions are enacted in order to avoid collisions.

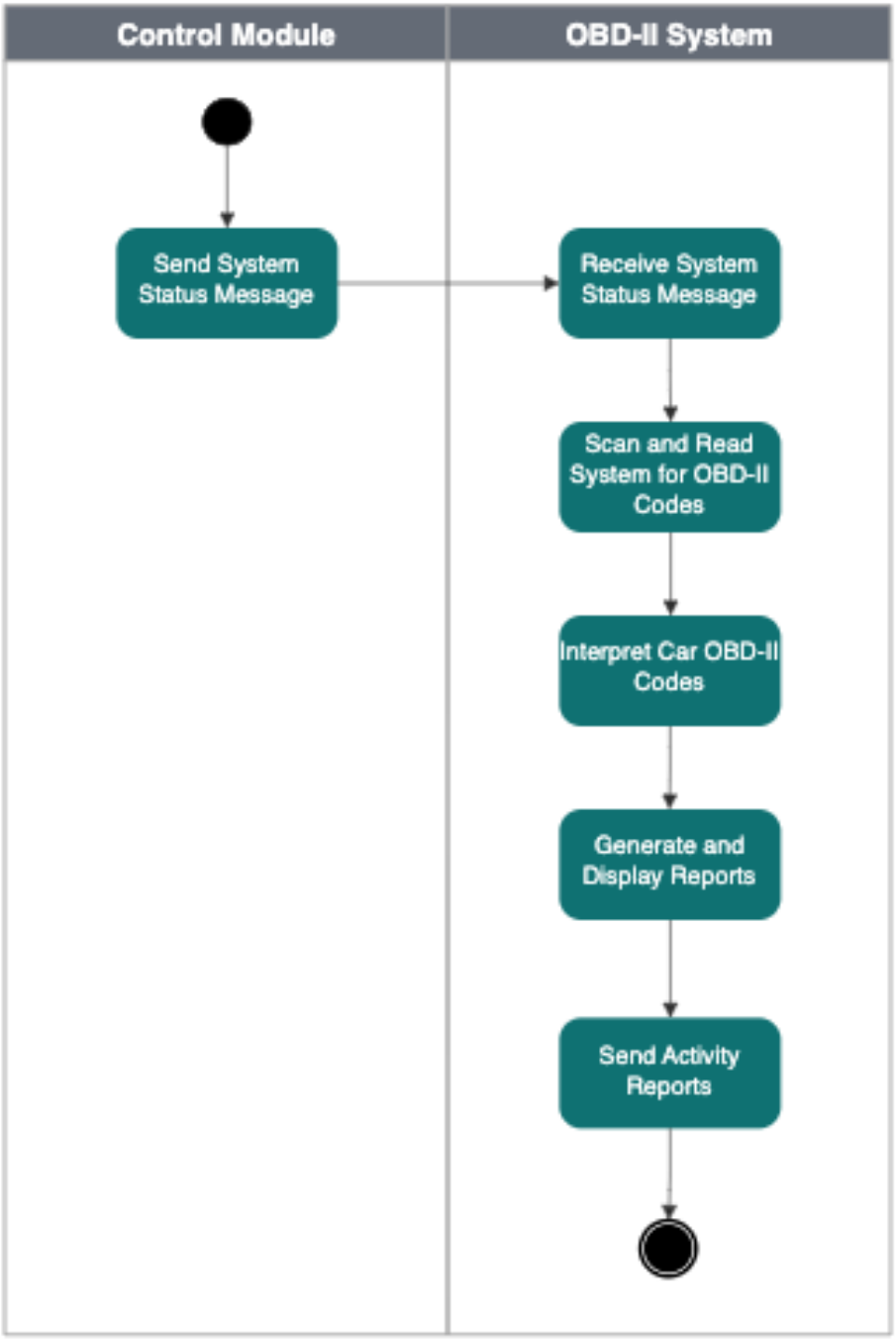

Figure 7.2.4 – AD04 (OBD-II System)

7.2.4.1 – OBD-II System

Scans and reads system for OBD-II codes. Interprets the OBD-II codes after they have been analyzed. Generates and displays reports. Sends activity reports for documentation purposes.

7.3 Sequence Diagrams

We wanted to translate some of the Functional Requirements into simple System Sequence Diagrams in order to help us visualize how information flows between all of our systems within the Navigation and Automatic Steering system (NASS). The following System Sequence Diagrams are very simple and have prerequisites for them to work as intended, such as the vehicle having a successful startup or there being an obstacle in the vehicle's path. One unique system that we added was a Counter which supports use cases by any class that needs to send multiple messages and have them be verified.

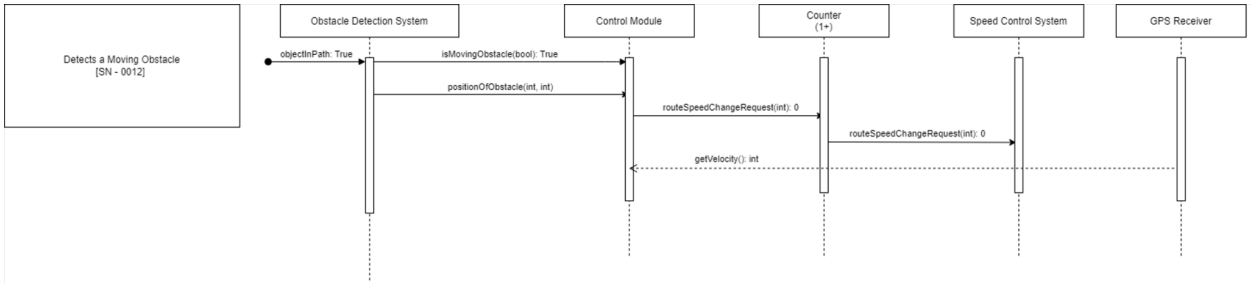

Figure 7.3.1 – SED01 (SN0012)

7.3.1.1 - SN0012

As shown in Figure 7.3.1, with a prerequisite that objectInPath is True, the Obstacle Detection System detects a Moving Obstacle and then sends the position of the obstacle to the Control Module. The Control Module will then send one or more Speed Change Requests to the Counter system which will verify the number of requests generated. The Counter will then send these Speed Change Requests to the Speed Control System. Once this has happened, the GPS Receiver will send the current velocity of the vehicle to the Control Module.

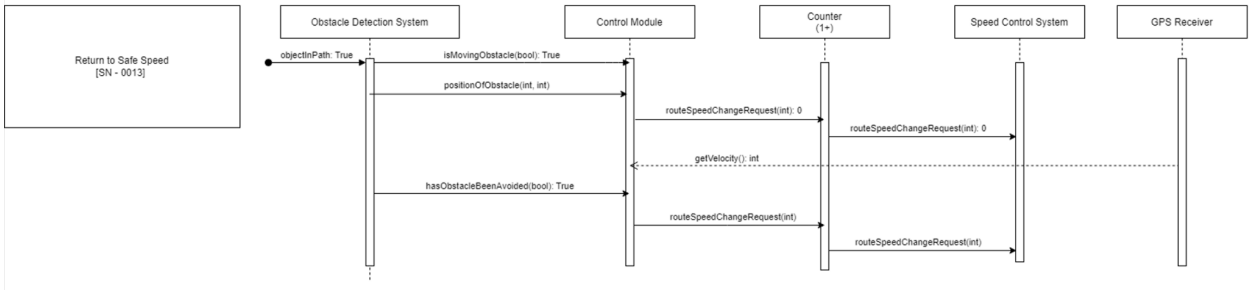

Figure 7.3.2 – SED02 (SN0013)

7.3.2.1- SN0013

As shown in Figure 7.3.2, with a prerequisite that objectInPath is true, the Obstacle Detection System detects a Moving Obstacle and then sends the position of the obstacle to the Control Module. The Control Module will then send one or more Speed Change Requests to the Counter system which will verify the number of requests generated. The Counter will then send these Speed Change Requests to the Speed Control System and it will change its speed accordingly to keep the vehicle at a safe distance. The GPS Receiver will then send the vehicle's current velocity to the Control Module. Once the Obstacle Detection System has verified that the obstacle has been avoided, it will send new Speed Change Requests to the Counter System which will again, verify the number of requests generated and send them to the Speed Control System.

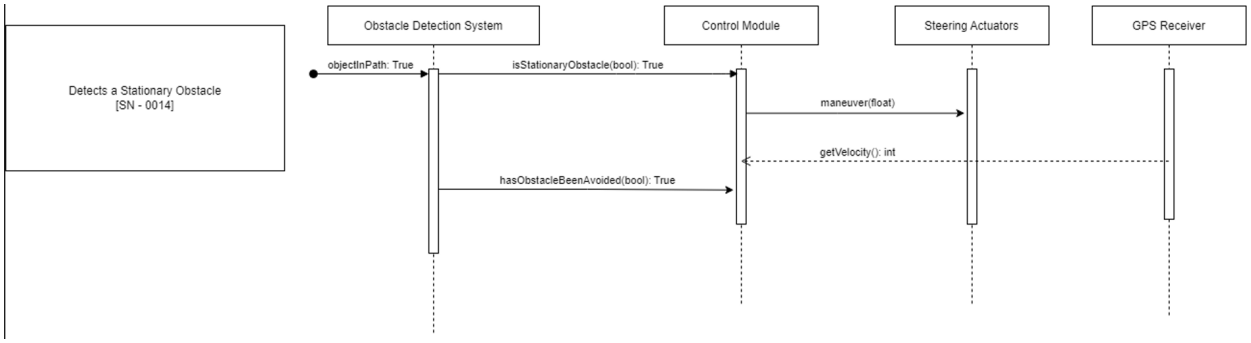

Figure 7.3.3 – SDa03 (SN0014)

7.3.3.1 - SN0014

As shown in Figure 7.3.3, with a prerequisite that objectInPath is True, the Obstacle Detection System detects that the obstacle is a Stationary Obstacle and sends an Obstacle Position Message to the Control Module. The Control Module will then send a maneuver command to the Steering Actuators in order to maneuver around the obstacle. The GPS Receiver will then send the current velocity to the Control Module. Once the Obstacle Detection System has verified that the obstacle has been avoided, it will send a message to the Control Module that the obstacle has successfully been avoided.

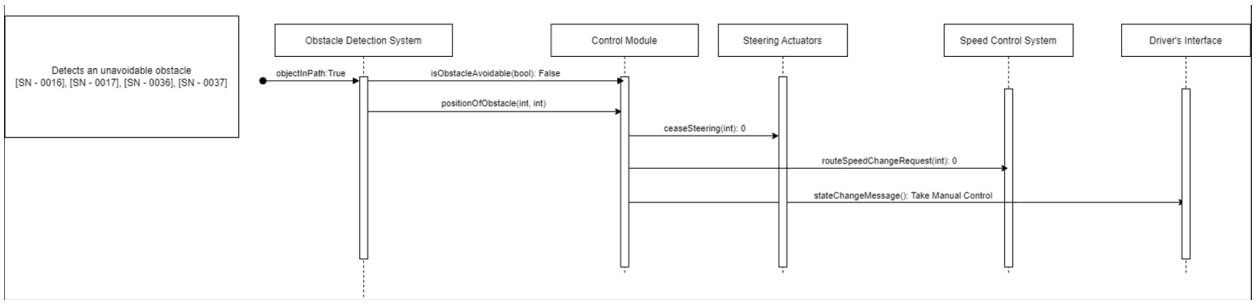

Figure 7.3.4 – SED04 (SN0016)

7.3.4.1 - SN0016

As shown in Figure 7.3.4, with a prerequisite that objectInPath is True, the Obstacle Detection System detects that the obstacle is unavoidable and sends the position of the obstacle to the Control Module. Once the Control Module has received this message, it will communicate to the Steering Actuators to cease automatic steering. The Control Module will also send a speed change request of “0” to the Speed Control System to try to stop the vehicle. Lastly, the Control Module will notify the driver to “Take Manual Control” in order to minimize damage of the situation.

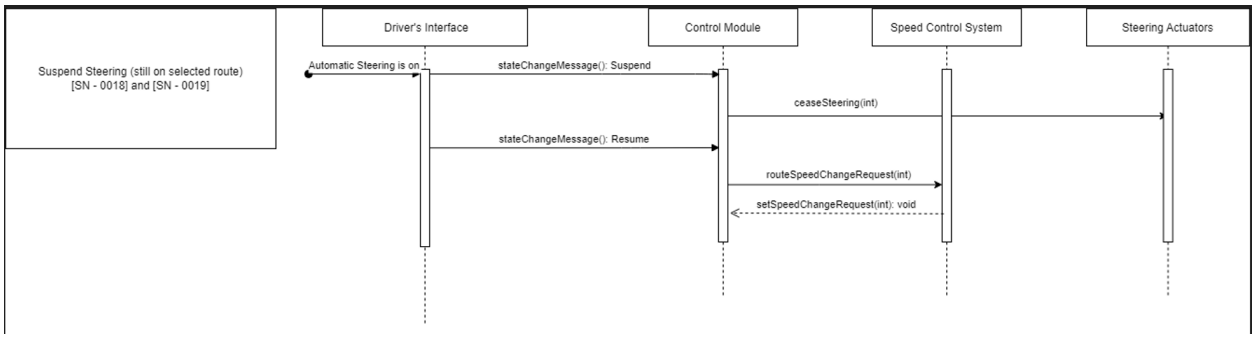

Figure 7.3.5 – SED05 (SN0018 and SN0019)

7.3.5.1 - SN0018 and SN0019:

As shown in Figure 7.3.5, with a prerequisite of that the vehicle currently has automatic steering turned on, the Driver's Interface (a system that acts as the UI for the driver) will send a State Change Message of “Suspend” to the Control Module system that will then send a ceaseSteering command to the Steering Actuators to suspend automatic steering. If the driver has decided to resume automatic steering, the Driver’s interface will send a State Change Message of “Resume” to the Control Module system that will then resume Speed Change Requests of the Speed Control System.

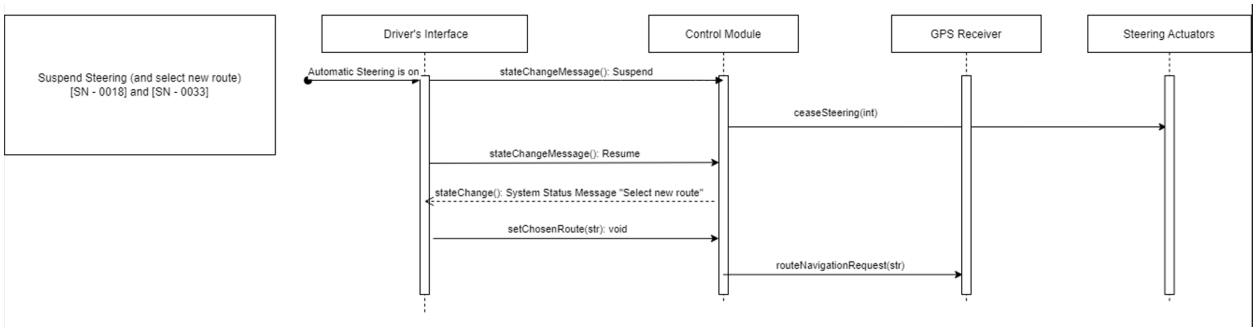

Figure 7.3.6 – SED06 (SN0018 and SN0033)

7.3.6.1 - SN0018 and SN0033:

As shown in Figure 7.3.6, with a prerequisite that the vehicle currently has automatic steering turned on, the Driver's Interface (a system that acts as the UI for the driver) will send a State Change Message of "Off" to the Control Module system that will then send a ceaseSteering command to the Steering Actuators to to suspend automatic steering. If the driver has decided to resume automatic steering, the Driver's interface will send a State Change Message of "Resume" to the Control Module system. If the current route is invalid and a new route needs to be selected, the Control Module will send a System Status Message to the Driver's Interface, asking to "Select new route." Once the driver has chosen a new route, the Driver's Interface will send the chosen route to the Control Module and will then send a Navigation Request to the GPS Receiver in order to start on this new path.

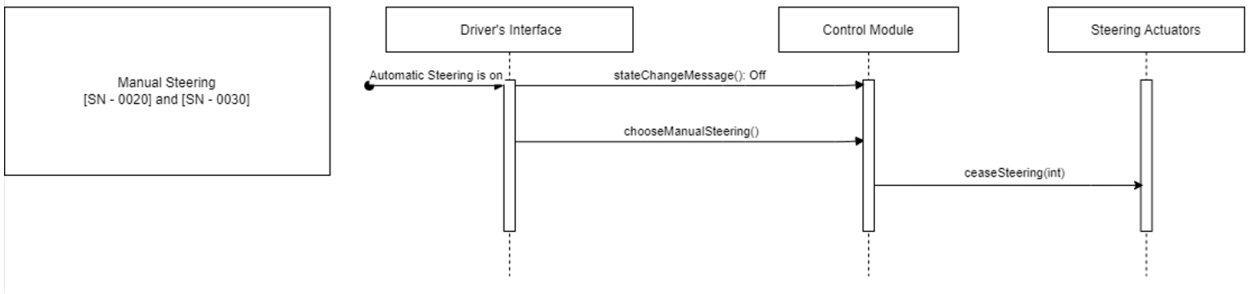

Figure 7.3.7 – SED07 (SN0020 and SN0030)

7.3.7.1 - SN0020 and SN0030

As shown in Figure 7.3.7, with a prerequisite that the vehicle currently has automatic steering turned on, the Driver's Interface (a system that acts as the UI for the driver) will send a State Change Message of "Off" to the Control Module. In addition, the Driver's Interface will communicate that the driver has chosen Manual Steering. The Control Module will then send a ceaseSteering Command to the Steering Actuators. The Steering Actuators will see this command and turn Manual Mode to "On".

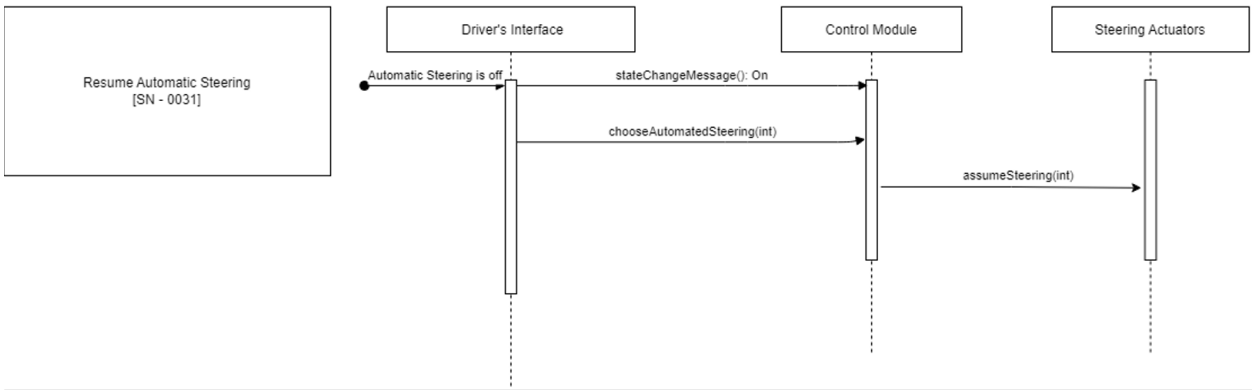

Figure 7.3.7 – SED08 (SN0031)

7.3.7.1 - SN0031:

As shown in Figure 7.3.7, with a prerequisite that the vehicle currently has automatic steering turned off, the driver can communicate through the Driver's Interface (a system that acts as the UI for the driver) and send a State Change Message of "On" to the Control Module. In addition, the Driver's Interface will communicate that the Driver has chosen Automated Steering. The Control Module will then send an assumeSteering Command to the Steering Actuators. The Steering Actuators will see this command, turn Manual Mode to "Off" to start automatic steering.

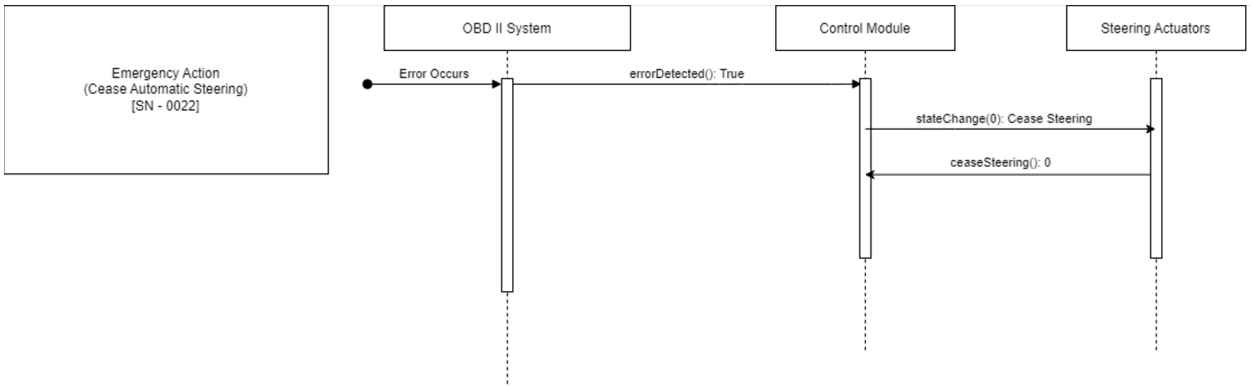

Figure 7.3.8 – SED09 (SN0022)

7.3.8.1 - SN0022:

As shown in Figure 7.3.8, with a prerequisite of an error occuring, the OBD II System will send a message to the Control Module that an error has occurred. The Control Module will then have an emergency stateChange to "Cease Steering" and will request this of the Steering Actuators. The Steering Actuators will comply and then cease automatic steering.

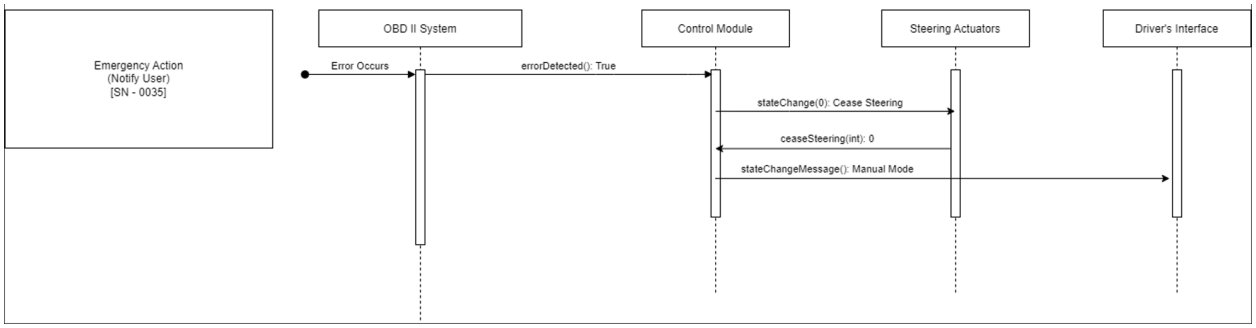

Figure 7.3.9 – SED10 (SN0035)

7.3.9.1 - SN0035:

As shown in Figure 7.3.9, with a prerequisite of an error occuring, the OBD II System will send a message to the Control Module that an error has occurred. The Control Module will then have an emergency stateChange to “Cease Steering” and will request this of the Steering Actuators. The Steering Actuators will comply and then cease automatic steering. Finally, the Control Module notifies the user that the vehicle is now in “Manual Mode” and that the user needs to take over.

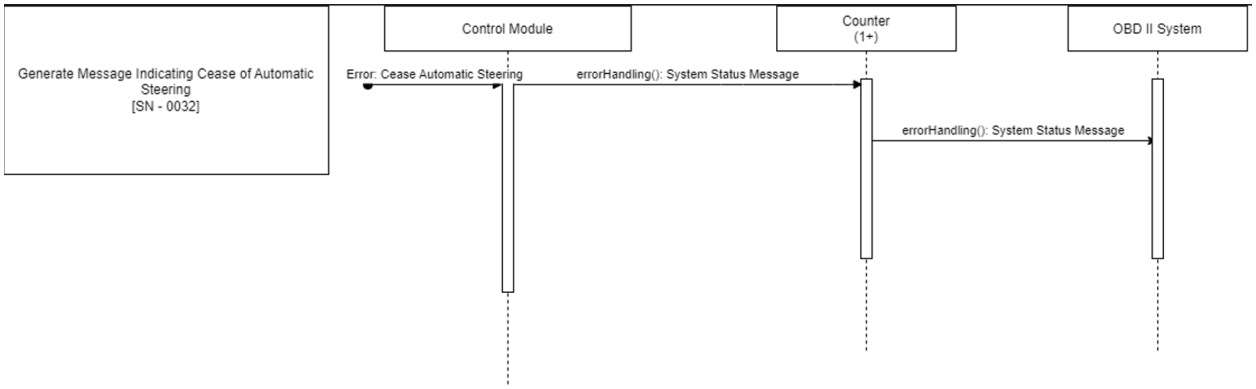

Figure 7.3.10 – SED11 (SN0032)

7.3.10.1 - SN0032

As shown in Figure 7.3.10, with a prerequisite of automatic steering being ceased due to an error, the Control Module will send a System Status Message indicating the reason for the error to the Counter system. The Counter will take one or more System Status Messages that the Control Module generates, verify the amount generated, and send them to the OBD II System.

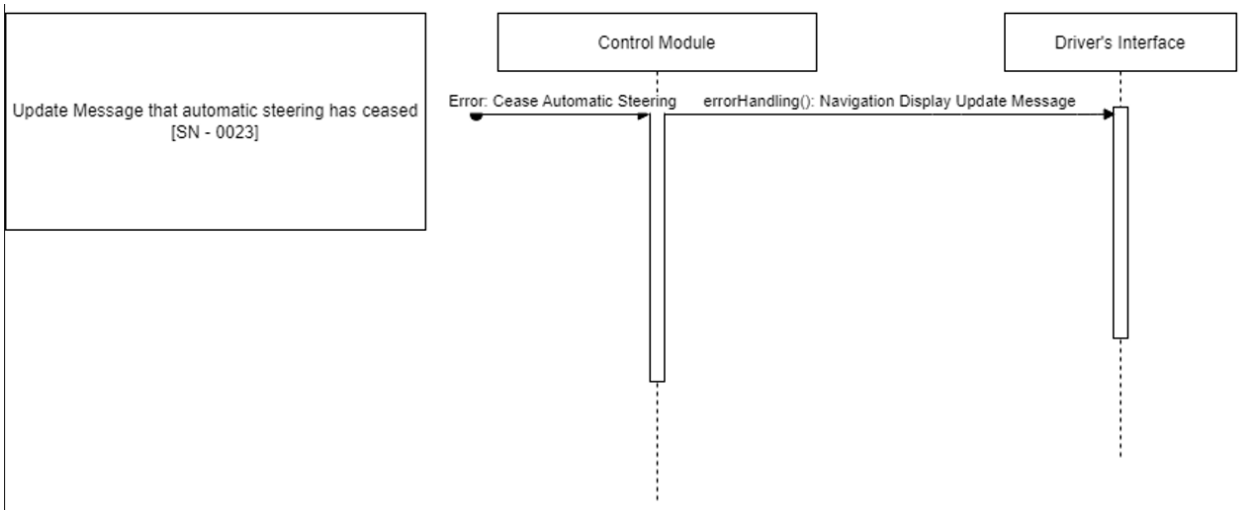

Figure 7.3.11 – SED12 (SN0023)

7.3.11 - SN0023:

As shown in Figure 7.3.11, with a prerequisite of automatic steering being ceased due to an error, the Control Module will send a Navigation Display Update Message to the Driver's Interface indicating the reason for the error.

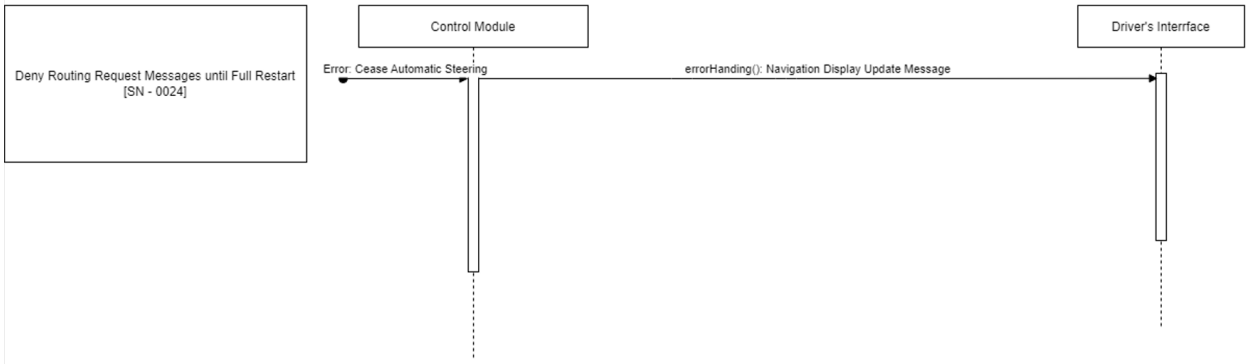

Figure 7.3.12 – SED13 (SN0024)

7.3.12.1 - SN0024:

As shown in Figure 7.3.12, with a prerequisite of automatic steering being ceased due to an error, the Control Module will send a Navigation Display Update Message to the Driver's Interface stating that the vehicle will not accept new routing request messages until the vehicle is shut off and has been restarted.

Counter:

The Counter supports use cases by any class that needs to send multiple messages and have them be verified.

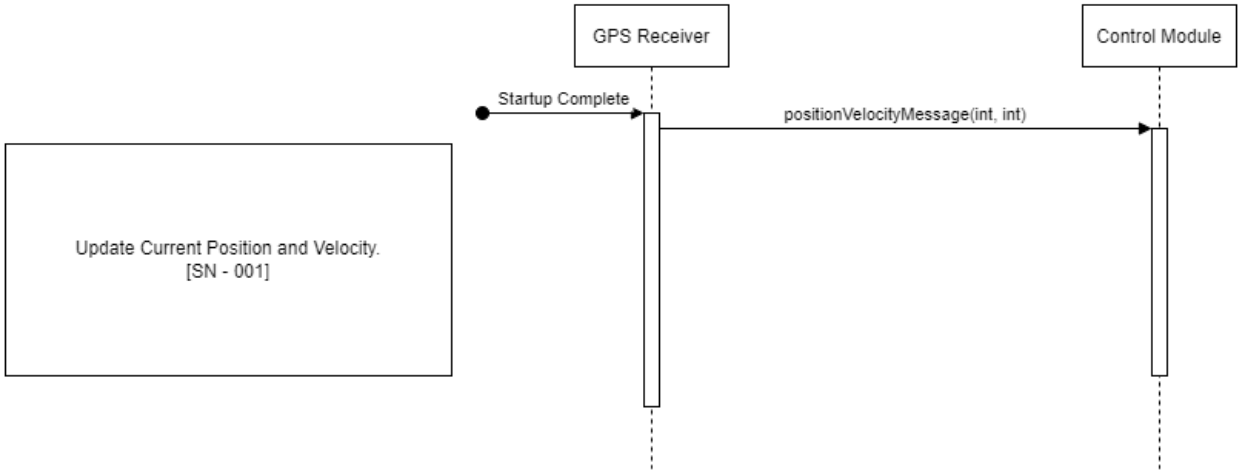

Figure 7.3.13 – SDa14 (SN001)

7.3.13.1 - SN001:

As shown in Figure 7.3.13, with a prerequisite of a successful complete startup, the GPS Receiver (a system that sends different types of Position Velocity Messages) will send its updated current Position and Velocity to the Control Module.

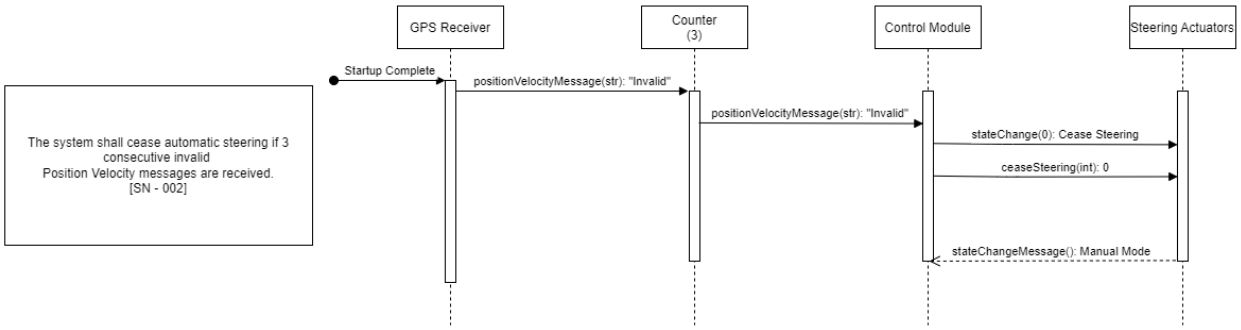

Figure 7.3.14 – SDa15 (SN002)

7.3.14.1 - SN002:

As shown in Figure 7.3.14, with a prerequisite of a successful complete startup, the GPS Receiver (a system that sends different types of Position Velocity Messages) will send an Invalid Position Velocity Message to a Counter system that, in this specific case, will send 3 consecutive (adaptable) Invalid Position Velocity messages to the Control Module system that will then communicate to the Steering Actuators system by

sending a Steering Command to “Cease Automatic Steering''. Once the Actuators receive this command, the Steering Actuators will subsequently send a command to “Activate Manual Mode” back to the Control Module.

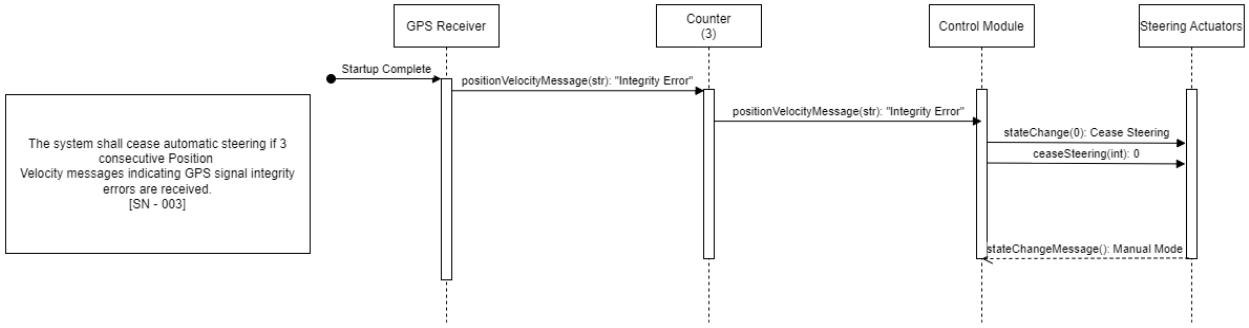

Figure 7.3.15 – SDa16 (SN003)

7.3.15.1- SN003:

As shown in Figure 7.3.15, with a prerequisite of a successful complete startup, the GPS Receiver (a system that sends different types of Position Velocity Messages) will send an “Integrity Error” Message to a Counter system that, in this specific case, will send 3 (adaptable) consecutive “Integrity Error” Messages to the Control Module system that will then communicate to the Steering Actuators system by sending a Steering Command to “Cease Automatic Steering''. Once the Actuators receive this command, the Steering Actuators will subsequently send a command to “Activate Manual Mode” back to the Control Module.

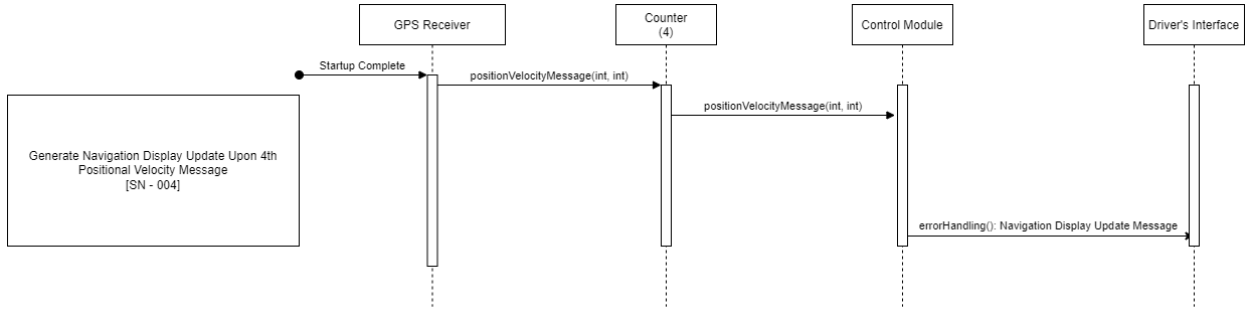

Figure 7.3.16 – SDa17 (SN004)

7.3.16.1 - SN004:

As shown in Figure 7.3.16, with a prerequisite of a successful complete startup, the GPS Receiver (a system that sends different types of Position Velocity Messages) will send its updated current Position and Velocity to a Counter system that, in this specific case, will send the current Position and Velocity to the

Control Module after the 4th (adaptable) valid Position Velocity Message which will then send a Navigation Display Update with a “Current Position” to the Driver’s Interface (a system that acts as the UI for the driver).

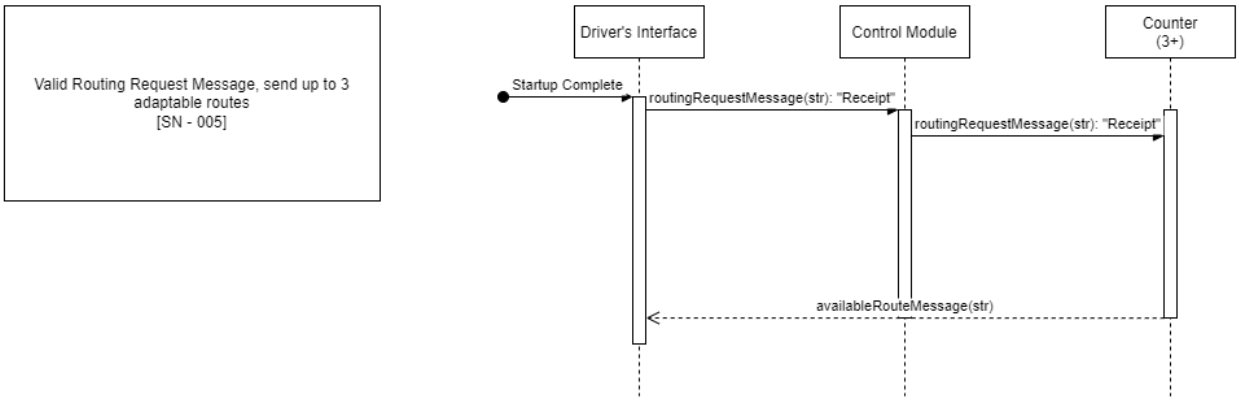

Figure 7.3.17 – SDa18 (SN005)

7.3.17.1 - SN005:

As shown in Figure 7.3.17, with a prerequisite of a successful complete startup, the Driver’s Interface (a system that acts as the UI for the driver) will send a “Receipt” of a valid Routing Request Message to the Control Module which will then send a similar “Receipt” to a Counter system that, in this specific case, shall determine up to 3 (adaptable number) of routes with an Available Route Message sent back to the Driver’s Interface.

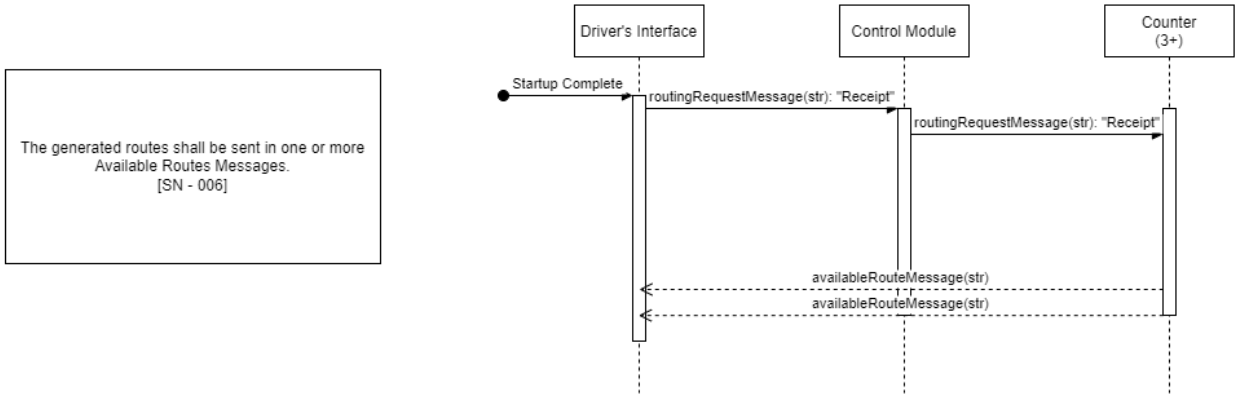

Figure 7.3.18 – SDa19 (SN006)

7.3.18.1 - SN006:

As shown in the Figure 7.3.18, with a prerequisite of a successful complete startup, the Driver’s Interface (a system that acts as the UI for the driver) will send a “Receipt” of a valid Routing Request Message

to the Control Module which will then send a similar “Receipt” to a Counter system that, in this specific case, shall determine up to 3 (adaptable number) of routes with an Available Route Message sent back to the Driver’s Interface. These generated routes shall be sent in one or more Messages.

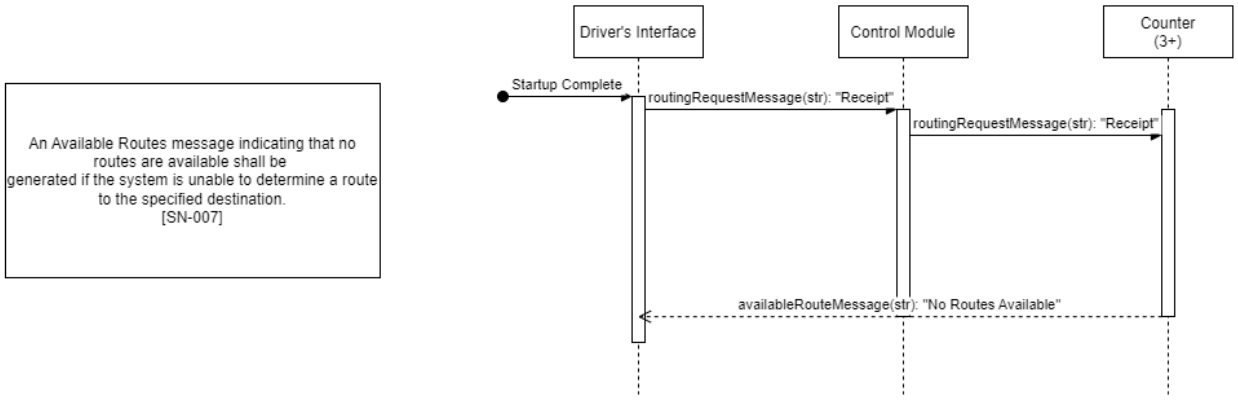

Figure 7.3.19 – SDa20 (SN007)

7.3.19.1 - SN007:

As shown in the Figure 7.3.19, with a prerequisite of a successful complete startup, the Driver’s Interface (a system that acts as the UI for the driver) will send a “Receipt” of a valid Routing Request Message to the Control Module which will then send a similar “Receipt” to a Counter system that, in this specific case, shall determine up to 3 (adaptable number) of routes with an Available Route Message with “No Routes Available” sent back to the Driver’s Interface.

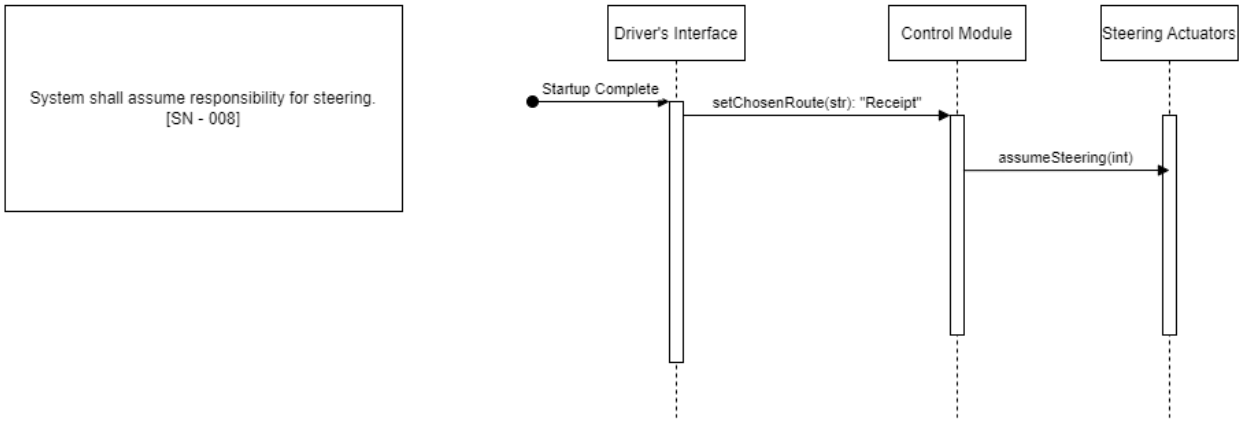

Figure 7.3.20 – SDa21 (SN008)

7.3.20.1 - SN008:

As shown in the Figure 7.3.20, With a prerequisite of a successful complete startup, the Driver's Interface (a system that acts as the UI for the driver) will send a "Receipt" of a valid Selected Route Message to the Control Module which will then a Steering Command to the Steering Actuators.

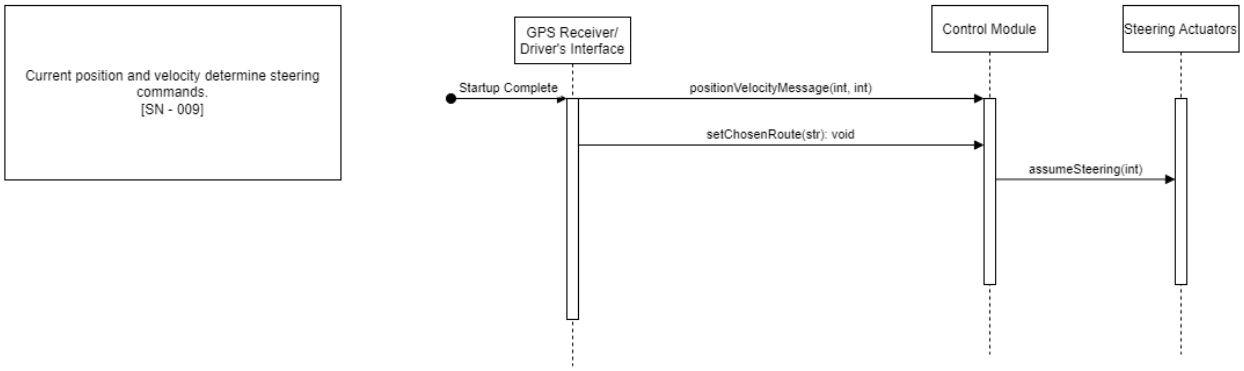

Figure 7.3.21 – SDa22 (SN009)

7.3.21.1 - SN009:

As shown in the Figure 7.3.21, with a prerequisite of a successful complete startup, the GPS Receiver (a system that sends different types of Position Velocity Messages) and the Driver's Interface (a system that acts as the UI for the driver) will send a Position Velocity Message with "Current Position" and "Current Velocity" and a Selected Route Message respectively to the Control Module. The Control Module will then send a Steering Command to the Steering Actuators.

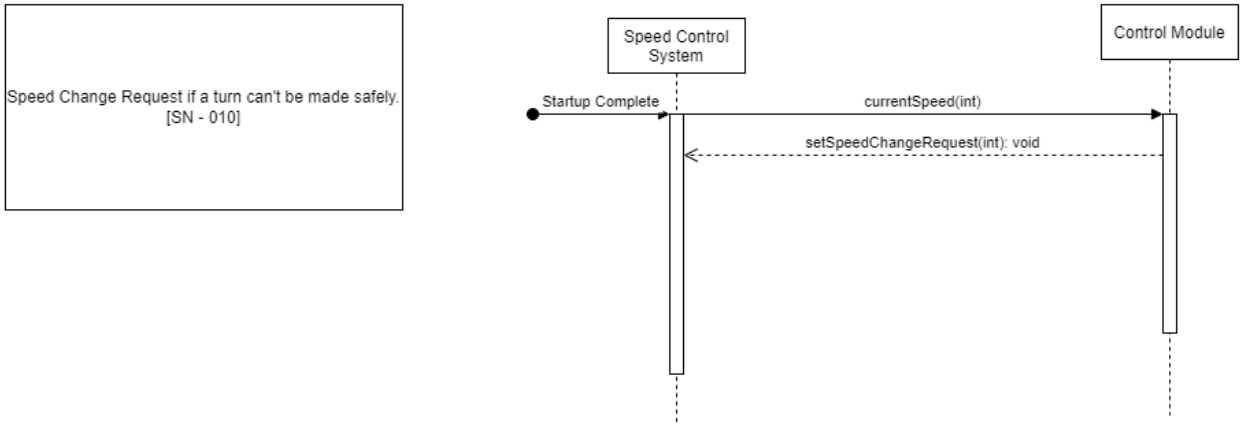

Figure 7.3.22 – SDa23 (SN010)

7.3.22.1 - SN010:

As shown in the Figure 7.3.21, with a prerequisite of a successful complete startup, the Speed Control System will send a Current Speed Message to the Control Module which will then send back a Speed Change Request to the Speed Control System.

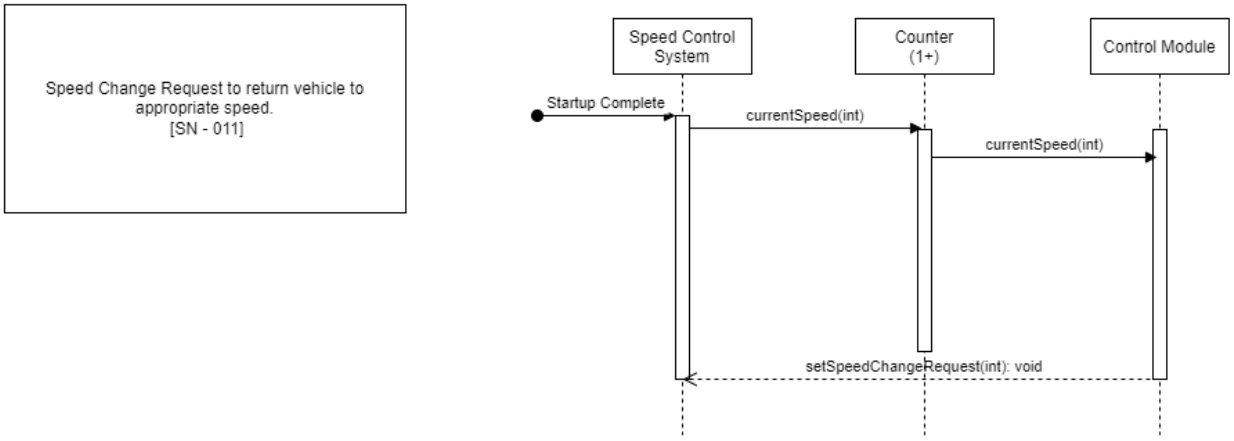

Figure 7.3.23 – SDa24 (SN011)

7.3.23.1 - SN011:

As shown in the Figure 7.3.21, with a prerequisite of a successful complete startup, the Speed Control System will send a Current Speed Message to a Counter system that, in this case, will send 1 or more Current Speed Messages to the Control Module which will then send back a Speed Change Request to the Speed Control System.

7.4 Communication Diagrams

In order to further increase the understanding of the Functional Requirements, we organized them into Communication Diagrams to convey the relationships between multiple Functional Requirements and how they flow through each of our systems of the Navigation and Automatic Steering system (NASS). These diagrams display the aforementioned flow of functions via the numbering system and arrows. The numbering system corresponds to individual Functional Requirements while the arrows showcase the flow between the systems. In order to increase legibility, the arrows are vertical and are side-by-side one another, while the functions are displayed as a list next to the arrows.

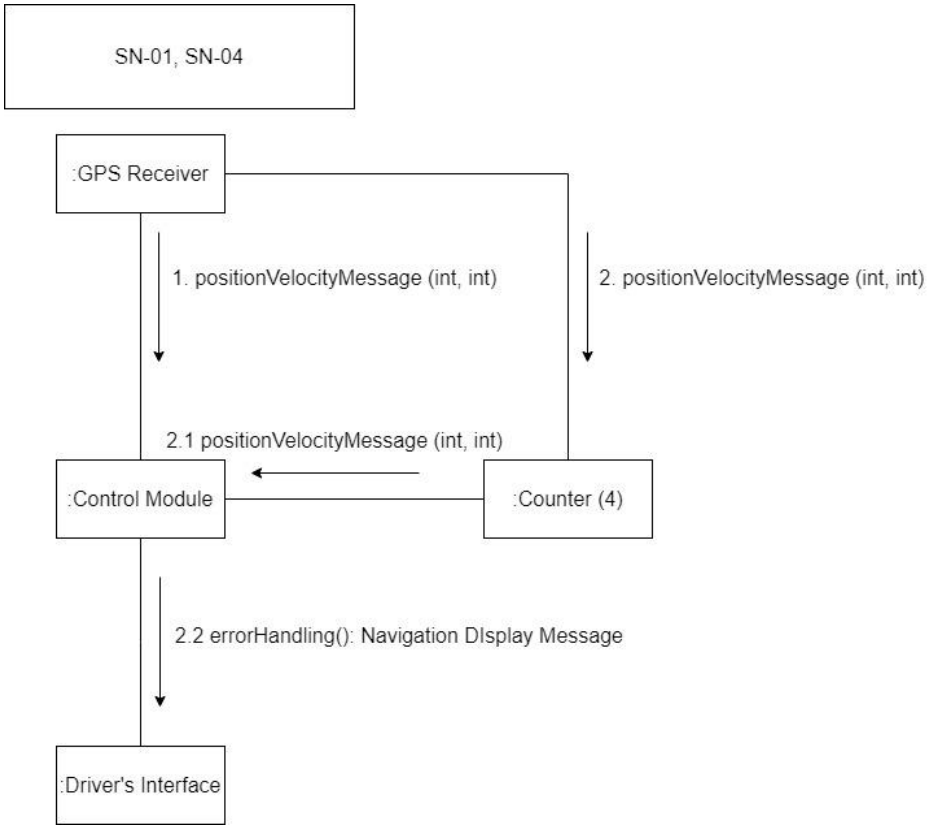

Figure 7.4.01 – CMD01 (SN0001 and SN0004)

SN – 0001 and SN – 0004

As shown in Figure 7.4.01, the GPS Receiver shall send its position velocity message to the Control Module directly and upon the fourth message sent to the Control Module, the Control Module shall hence send a navigation display message to the Driver's Interface.

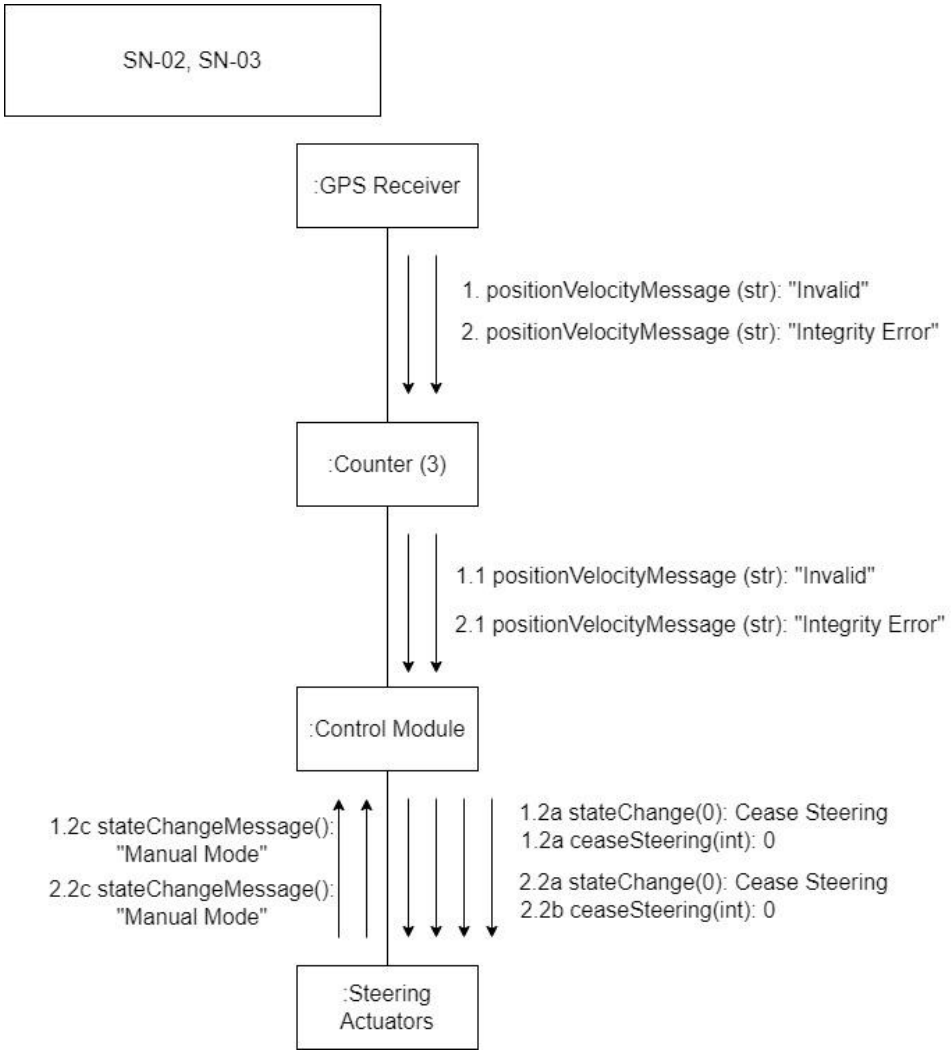

Figure 7.4.02 – CMD02 (SN0002 and SN0003)

SN – 0002 and SN – 0003

As shown in Figure 7.4.02, if the position velocity message is invalid three times, the GPS Receiver shall send a position velocity message indicating it is invalid or it is an integrity error to the Control Module. If the position velocity message sent contains the invalid message, then the Control Module shall send a state change message to the Steering Actuators and shall tell the Steering Actuators to cease steering. The Steering Actuators shall then proceed to send a state change message consisting of manual mode to the Control Module. If the position velocity message sent contains the integrity error message, then the Control Module shall send a state change message to the Steering Actuators and shall tell the Steering Actuators to cease steering. The Steering Actuators shall then proceed to send a state change message consisting of manual mode to the Control Module.

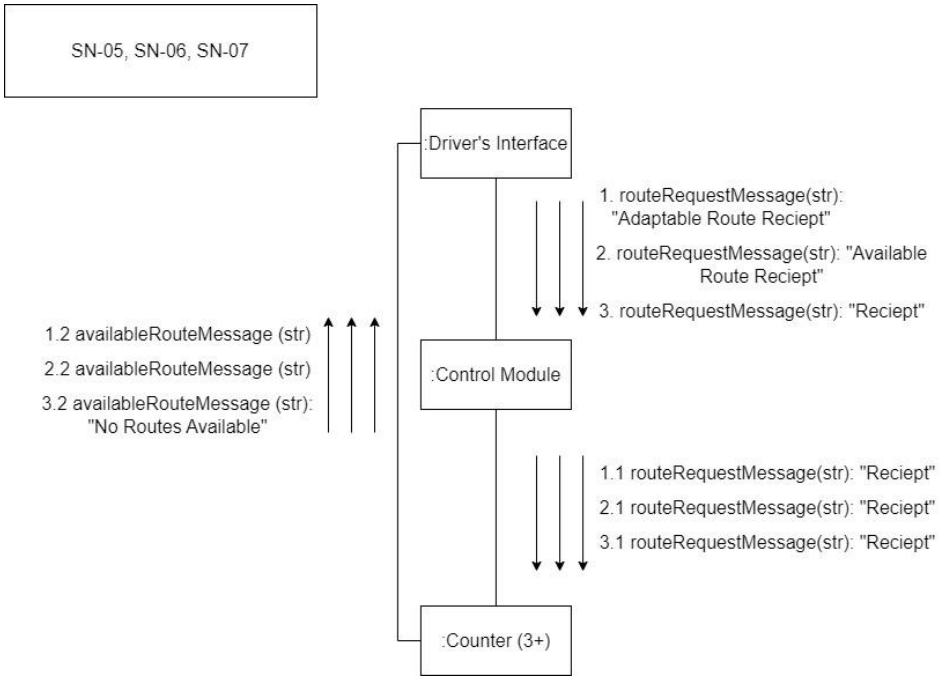

Figure 7.4.03 – CMD03 (SN0005, SN0006, and SN0007)

SN – 0005, SN – 0006, and SN – 0007

As shown in Figure 7.4.03, the Driver's Interface shall be able to send route request messages consisting of adaptable route receipt, available route receipt, and receipt to the Control Module. Once a type of message reaches a count of three, for the amount of messages sent, then the Control shall send available route messages in accordance to if the route request messages consist of adaptable route receipt and available route receipts. If the route request message consists of receipt, then, the Control Module shall send an available route message consisting of there being no route available to the Driver's Interface, thereby alerting the driver.

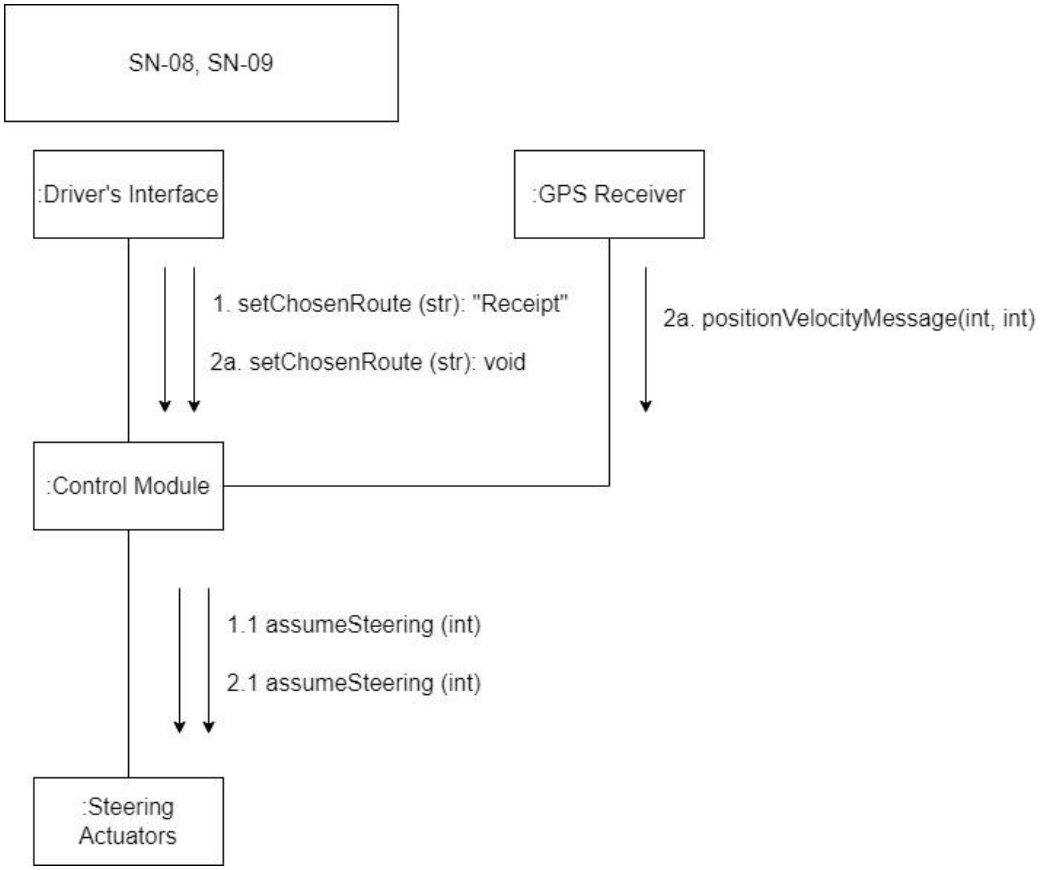

Figure 7.4.04 – CMD04 (SN0008 and SN0009)

SN – 0008 and SN – 0009

As shown in Figure 7.4.04, the Driver's Interface shall send a set chosen route message to the Control Module. If the message consists of receipt, then the Control Module will send a message to the Steering Actuators to assume steering. Otherwise, if not set chosen route message does not consist of receipt sent to the Control Module, and the GPS Receiver sends a position velocity message alongside the set chosen route message, then the Control Module will send an assume steering message to the Steering Actuators, but this message relies more upon the system to determine the best steering actions, rather than relying on input from the user.

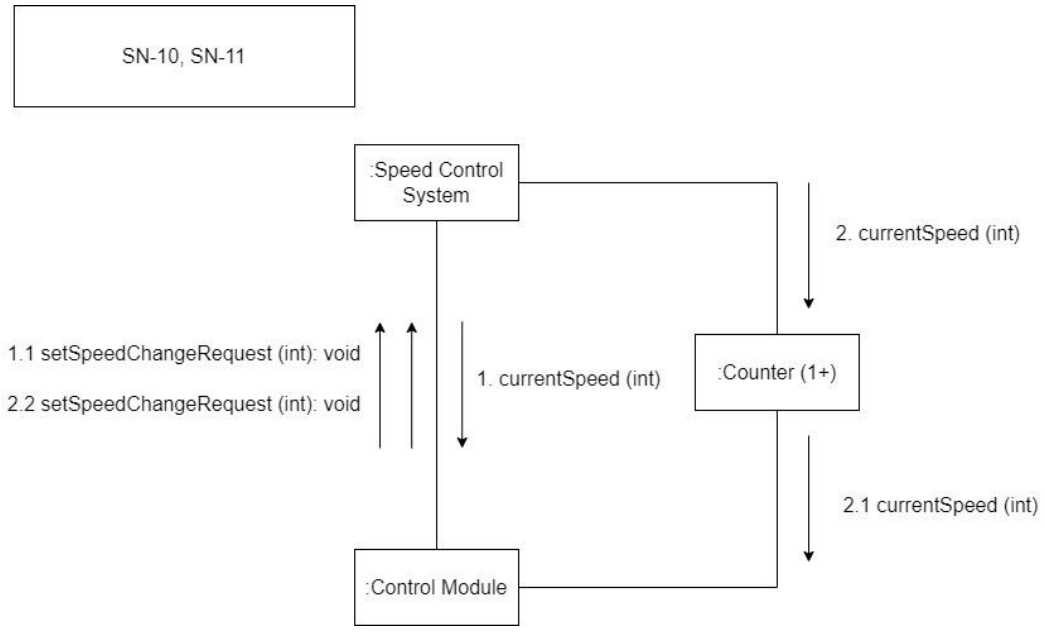

Figure 7.4.05 – CMD05 (SN0010 and SN0011)

SN – 0010 and SN – 0011

As shown in Figure 7.4.05, the Speed Control System shall send a message consisting of its current speed to the Control Module in cases where a turn cannot be made safely. The Control Module then sends a speed change request back to the Speed Control System in response. After completing a turn, one or more speed change requests shall be sent to the Control Module.

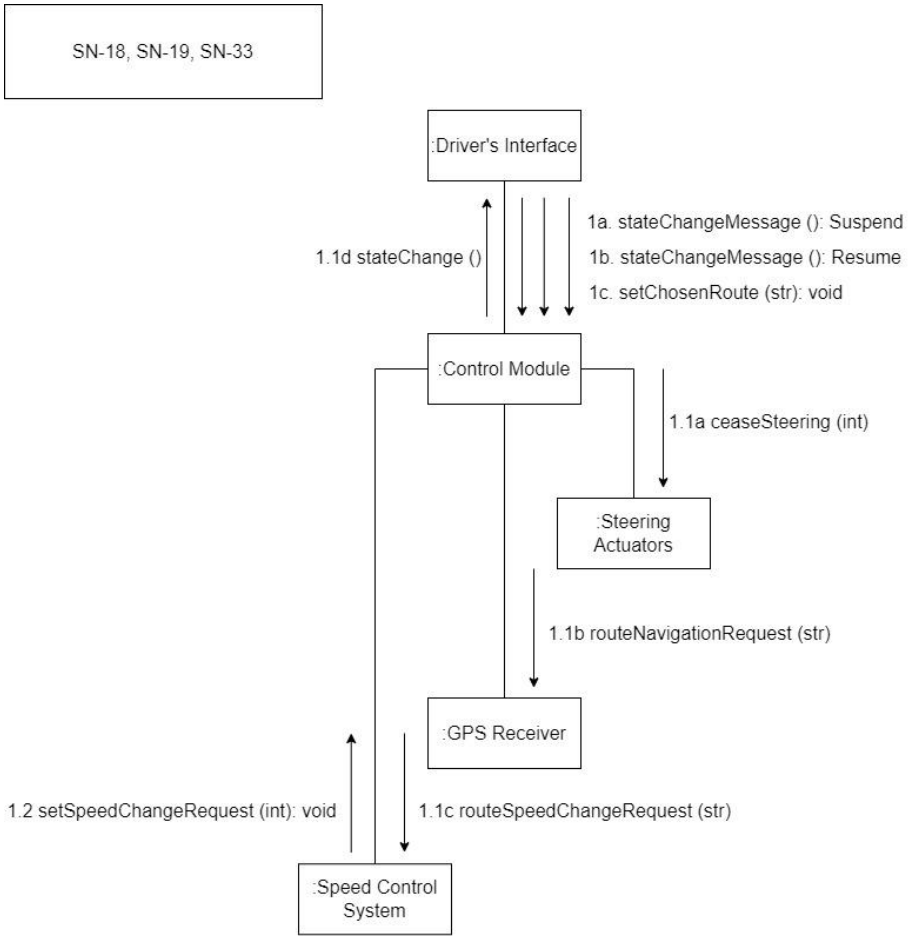

Figure 7.4.06 – CMD06 (SN0018, SN0019, and SN0033)

SN – 0018, SN – 0019, and SN – 0033

As shown in Figure 7.4.06, the Driver's Interface shall send a state change message consisting of suspend or resume, accompanied by the set chosen route message, to the Control Module. The Control Module shall then send a cease steering message to the Steering Actuators, if the previous state change message was suspend or resume. The Control Module shall also send a route speed change request to the Speed Control System to suspend steering whilst still on the selected route. The Speed Control System shall send a set speed change request to the Control Module only if the state change message consists of suspend, in order to suspend steering whilst still on a selected route. The Control Module shall also send a route navigation request to the GPS Receiver if the system is suspending steering but wants to select a new route, with a state change message to the Driver's interface to accompany.

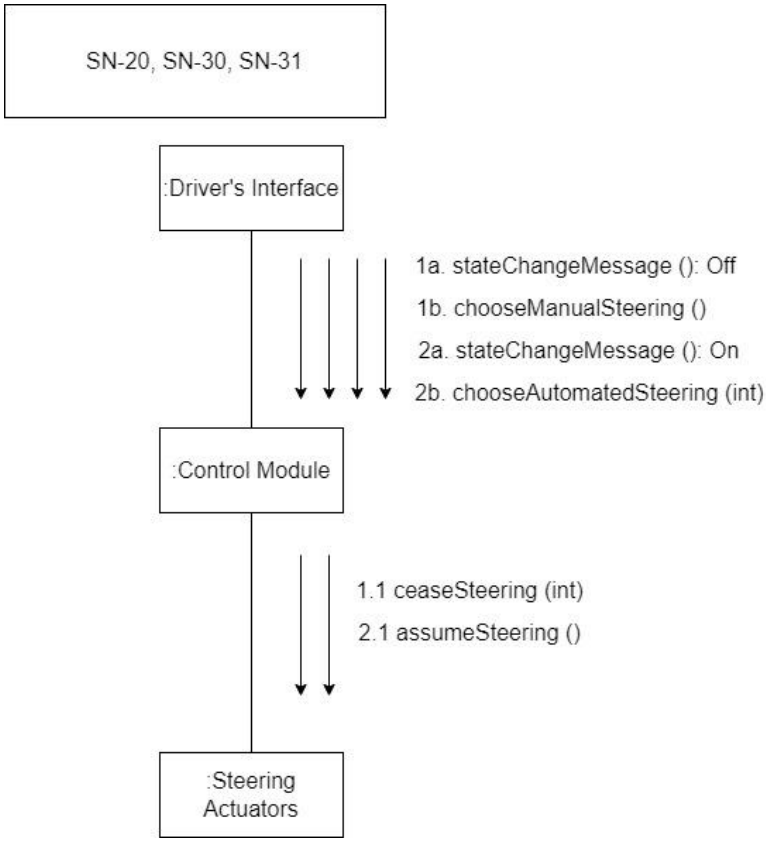

Figure 7.4.07 – CMD07 (SN0020, SN0030, and SN0031)

SN – 0020, SN – 0030, and SN – 0031

As shown in Figure 7.4.07, the Driver Interface follows two routes. The Driver Interface shall send the state change message consisting of Off accompanied by a choose manual steering message to the Control Module. The Control Module shall then send a cease steering message to the Steering Actuators. Meanwhile, the Driver Interface shall also have the ability to send a state change message consisting of On, accompanied by a chose automated steering message to the Control Module. The Control Module shall then send an assume steering message to the Steering Actuators.

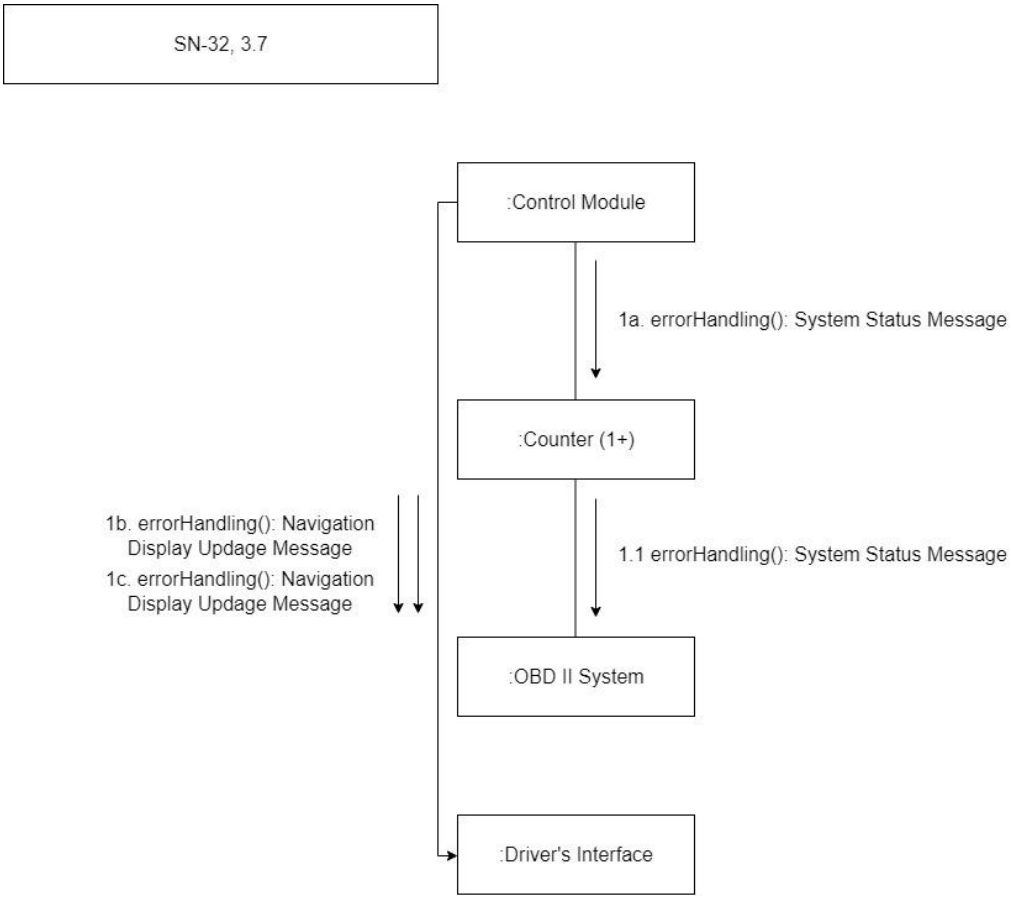

Figure 7.4.08 – CMD08 (SN0032 and 3.7)

SN – 0032 and 3.7

As shown in figure 7.4.08, the Control Module shall send either one or more system status messages to the OBD II System while the car is on. The Control Module simultaneously sends navigation display update messages to the Driver's Interface.

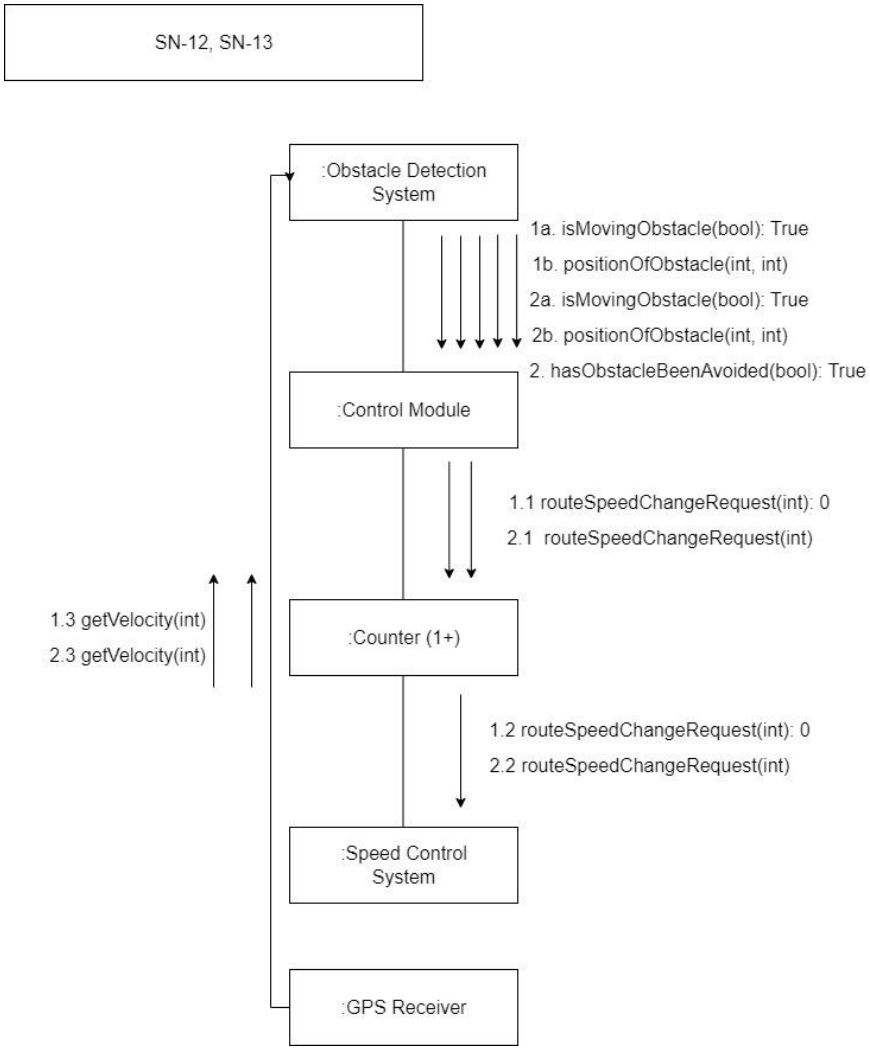

Figure 7.4.09 – CMD09 (SN0012 and SN0013)

SN – 0012, and SN – 0013

As shown in figure 7.4.09, the GPS Receiver shall send a get velocity message to the Obstacle Detection System. The system sends messages to the control module about whether there are any moving obstacles, the position of obstacles and the avoidance of the obstacles. If needed, the Control Module sends one or more speed change requests to the speed control system.

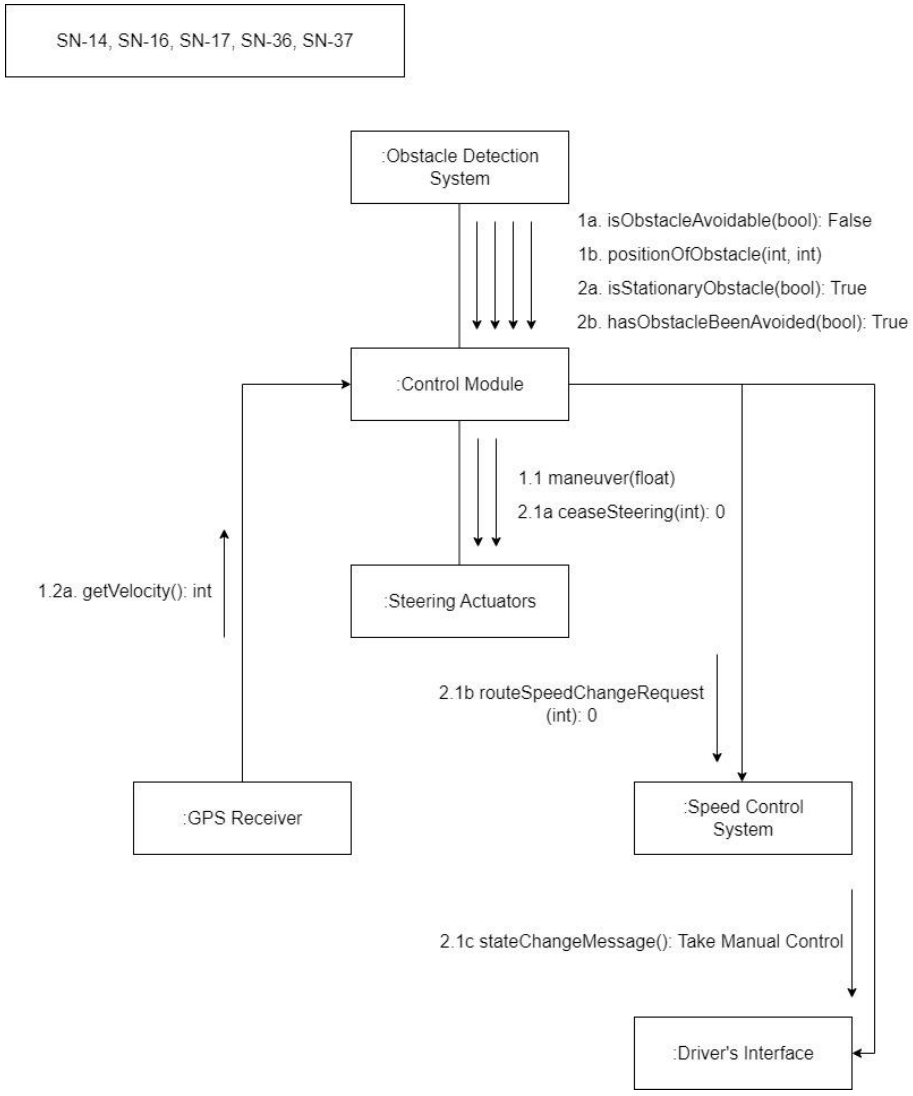

Figure 7.4.10 – CMD10 (SN0014, SN0016, SN0017, SN0036, and SN0037)

SN – 0014, SN – 0016, SN – 0017, SN – 0036, and SN – 0037

As shown in figure 7.4.10, the GPS Receiver shall send a get velocity message to the Control Module while the Obstacle Detection System sends messages to the control module about whether there are any moving obstacles, the position of obstacles and the avoidance of the obstacles. Then the Control Module shall send messages to the Steering Actuators to either maneuver around the obstacles or cease steering, and it shall send route speed change messages to the Speed Control System. The Control Module shall be able to send state change messages for the driver to take manual control of the vehicle to the Driver's Interface.

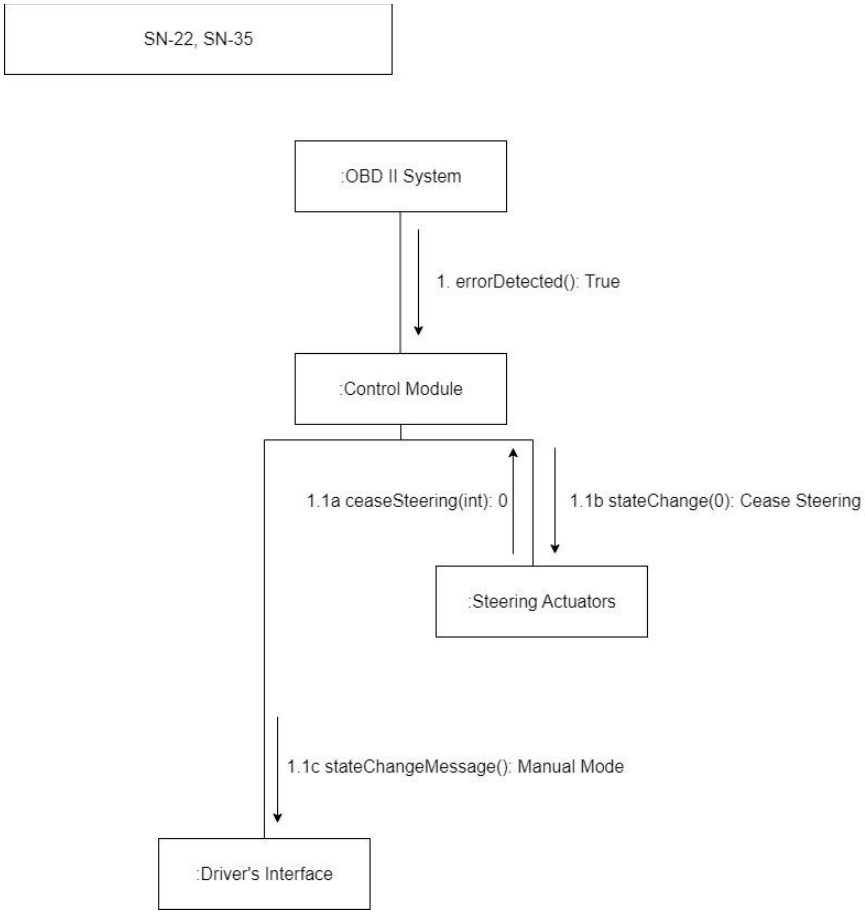

Figure 7.4.11 – CMD11 (SN0022 and SN0035)

SN – 0022 and SN – 0035

As shown in figure 7.4.11, the OBD II System shall send error detection messages to the Control Module. If the Control Module sends state change messages to cease steering to the Steering Actuators, the Steering Actuators would send a message to cease steering to the Control Module. Then the Control Module, shall send a state change message for manual mode to the Driver's Interface.

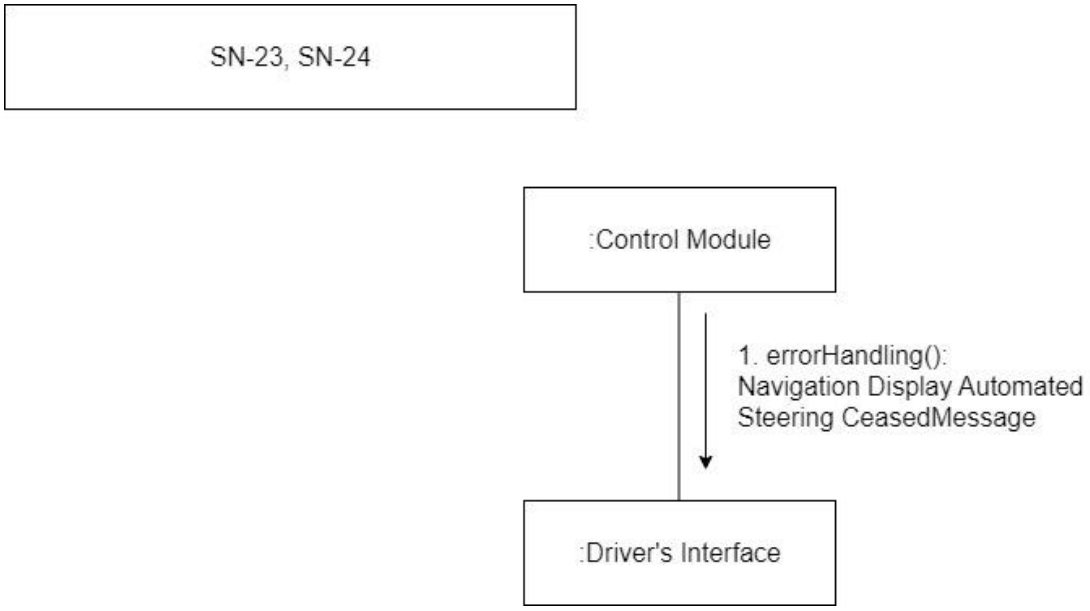

Figure 7.4.12 – CMD12 (SN0023 and SN0024)

SN – 0023 and SN – 0024

As shown in Figure 7.4.12, the Control Module shall send an error handling message to the Driver's Interface when the navigation display automates the steering ceased message.

8. Physical View

8.1 Overview

The Autonomous Auto Deployment Diagram displays the physical aspects of our system of each environment. Each box tab represents the environment of each block represented in the SRS. Each component is labeled <<device>> and <<execution>> respectively; the <<devices>> are the physical components and the <<execution>> are the software housed in.

8.2 Deployment Diagram

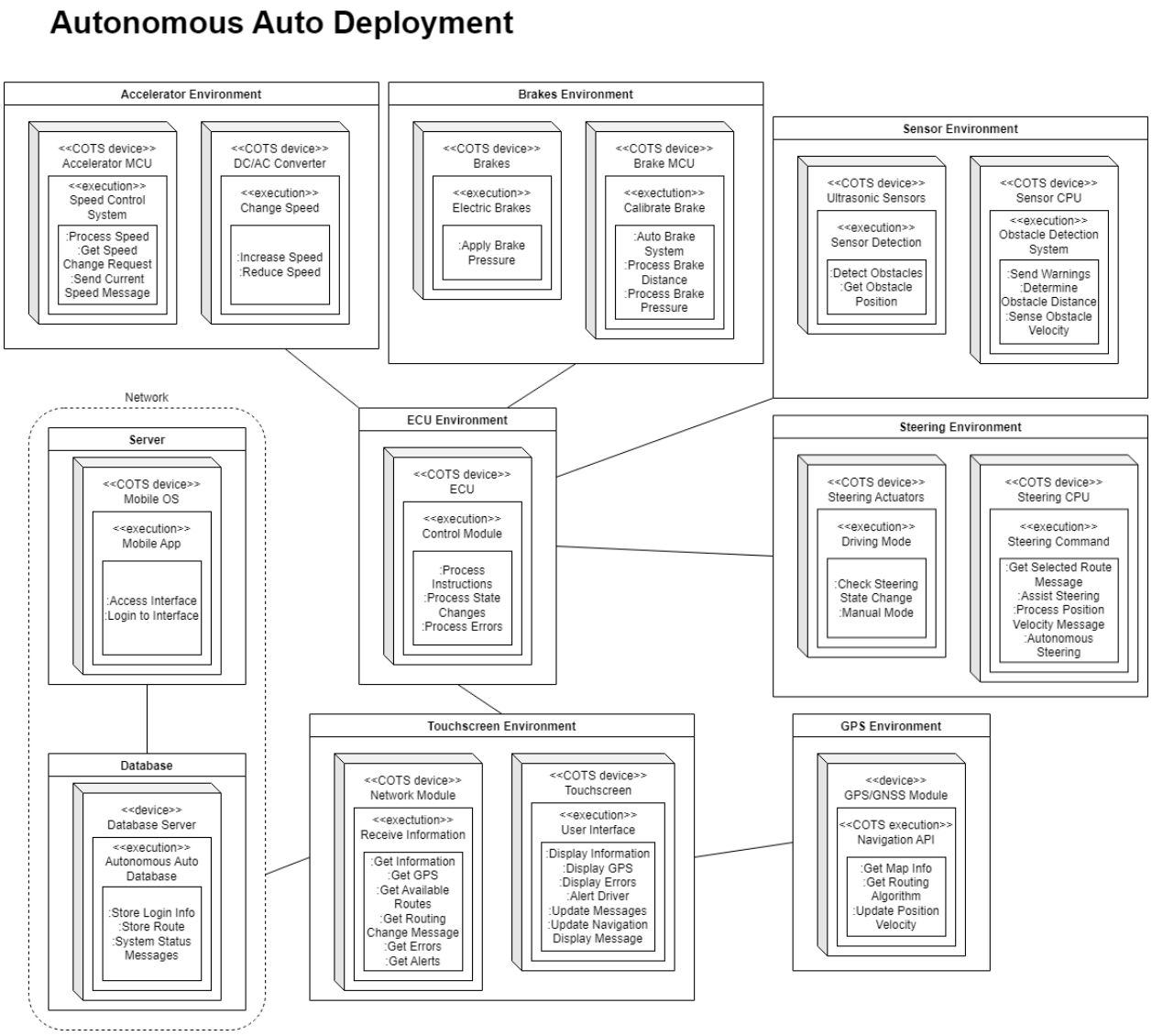

Figure 8.2.1 – DD01 (AA Deployment)

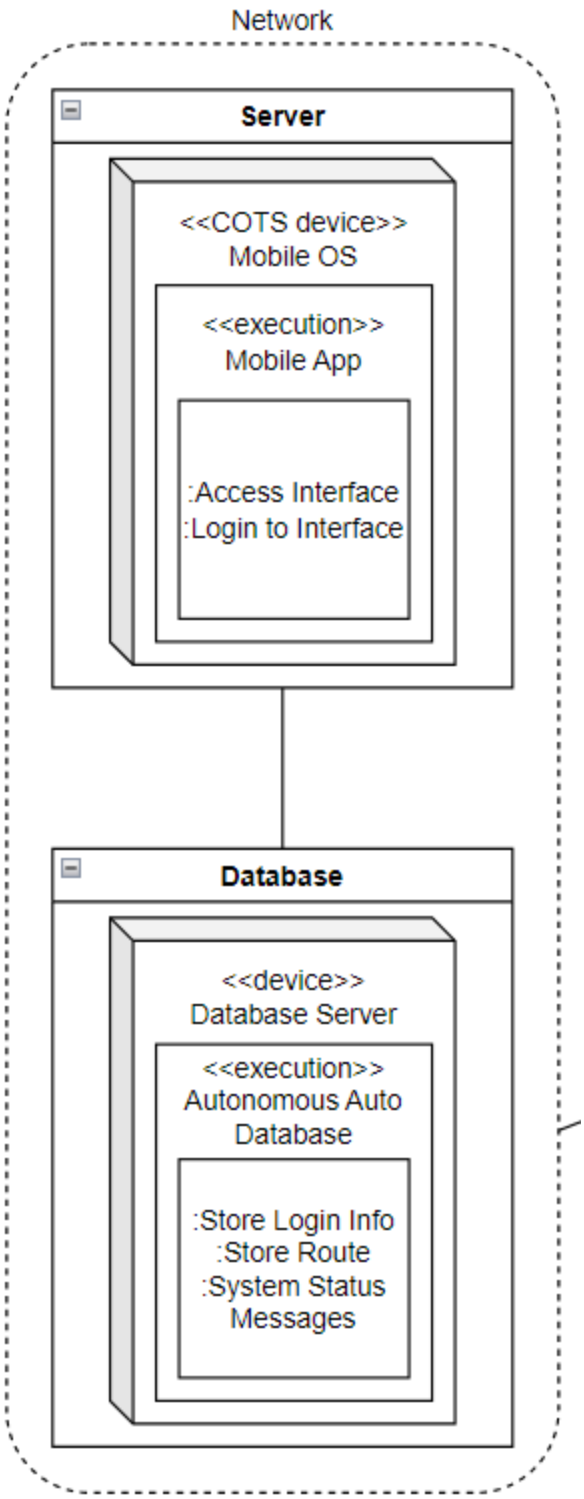

Figure 8.2.2 – DD02 (Deployment Network)

8.2.2.1 – Deployment Network

We will have our own network that will feature a server that interacts with the interface and a database server that holds the storage of all login info and their routes. Our network will also have security in accordance with [SN-0038].

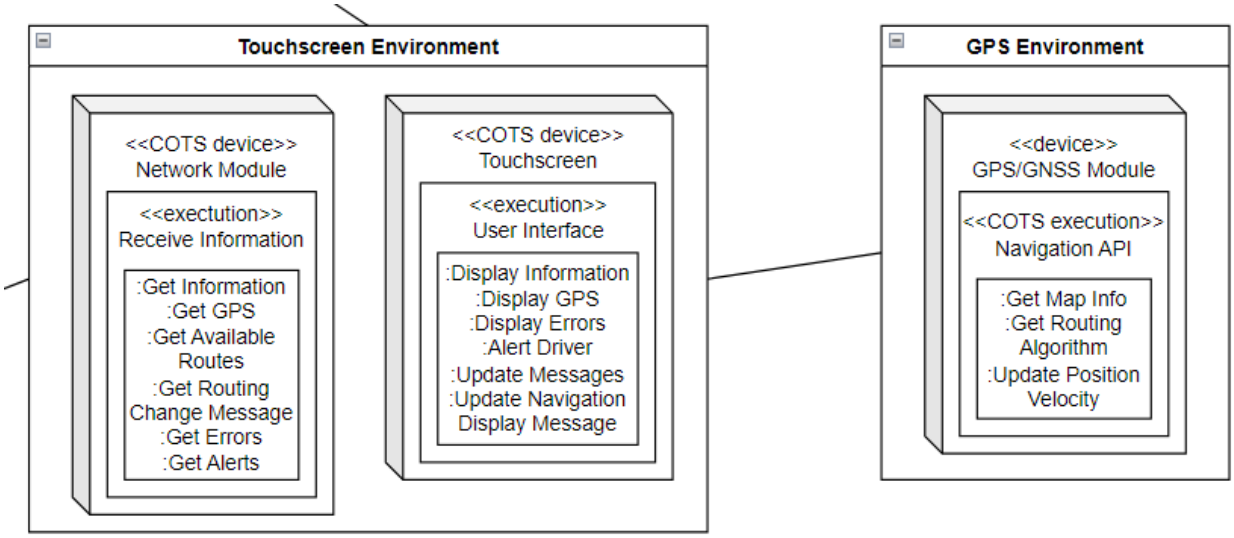

Figure 8.2.3 – DD03 (Deployment Interface)

8.2.3.1 – Deployment Interface

The interface will interact with the network to display information and various System Status Messages from the database. The network module is a COTS device that will house the software to receive the information from the said network. The touchscreen will display the information that the network module receives and relay it to the user. It also interacts with the GPS which has a navigation API to support the GPS module and gets updates relating to position velocity and routing algorithms. The Position Velocity is updated periodically.

OLD: The interface will interact with the network to display information and various System Status Messages from the database by [SN-0032][SN-0023] The GPS interacts with the interface to display its map info, routes, and position [SN-001] The interface will display errors and alert the driver from the control module and GPS according to [SN-0032][SN-0023]

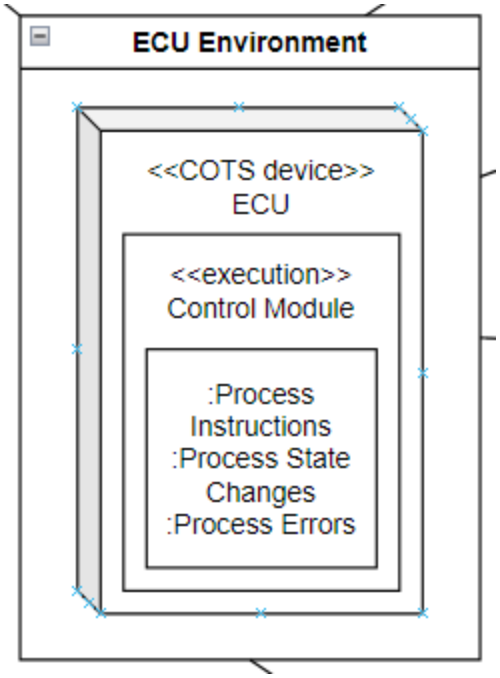

Figure 8.2.4 – DD04 (Electronic Control Unit)

8.2.4.1 – Electronic Control Unit

The ECU is the main computer in the car that uses Control Module to process instructions and various errors in [SN-0020][SN-0022][SN-0037] The ECU also processes state changes from its corresponding components [SN-0035] when changing from autonomous to manual drive.

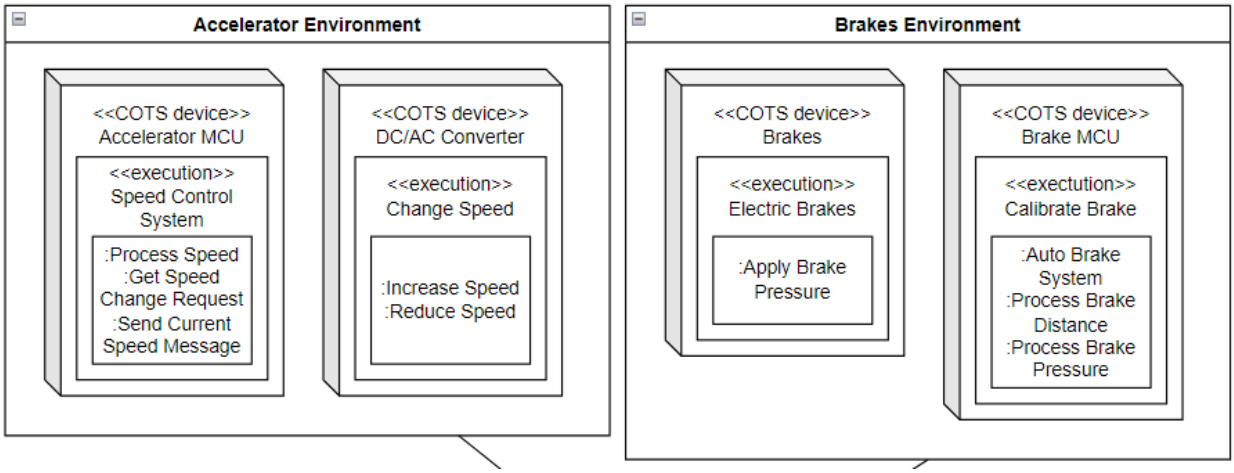

Figure 8.2.5 – DD05 (Accelerator & Brakes)

8.2.5.1 – Accelerator & Brakes

The accelerator uses the Speed Control System to process the speed and determine whether or not to increase or reduce the speed during autonomous drive due to the Speed Change Request [SN-0012] The brake MCU will calibrate the brakes during the autonomous drive and determine brake pressure needed to stop at a safe distance using the electric brakes[SN-0036].

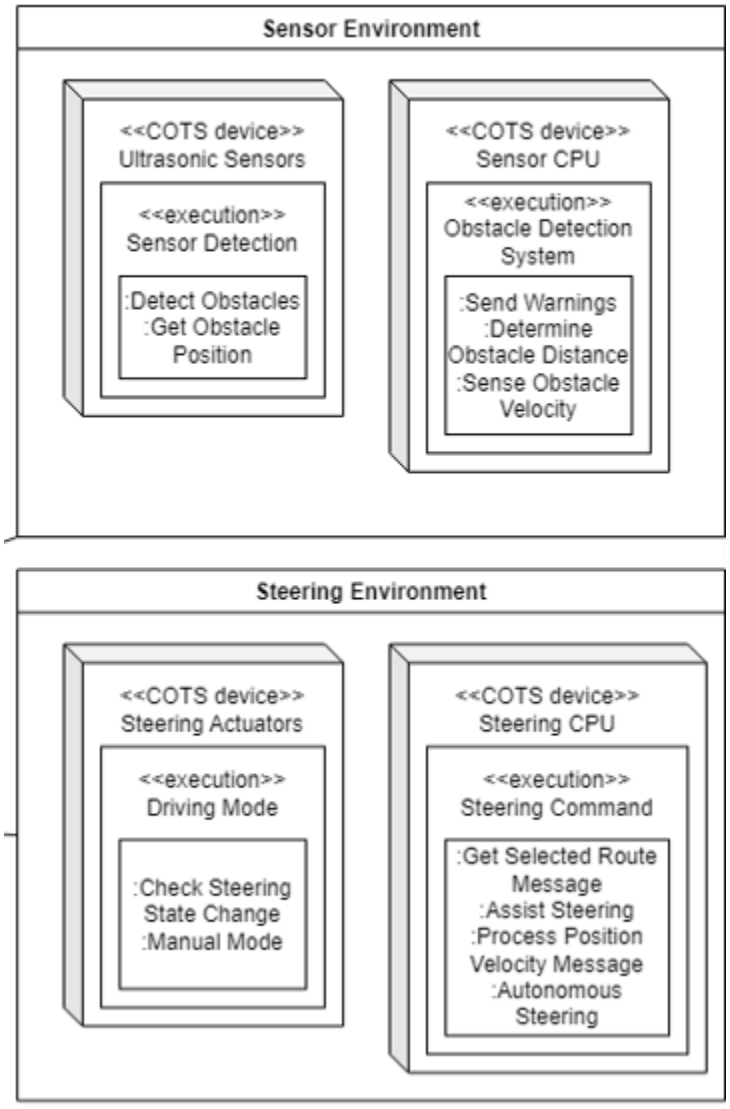

Figure 8.2.6 – DD06 (Sensors & Steering Actuators)

8.2.6.1 - Sensors & Steering Actuators

The sensors use the Obstacle Detection System to detect obstacles and determine obstacle distance to warn the driver through the control module [SN-0012] The steering actuators will use its processor to assist with driving and provide lane stability [SN-009] They will act on their own when the autonomous drive is active while in contact with the control module [SN-008]

9. Size and Performance

The software architecture supports the size and performance time requirements, as stated in the System Requirements Spec:

1. The system shall provide access to the GPS routes with no more than a 10-second latency.
2. The system must be able to communicate all message requests within 0.5 seconds.
3. The client portion shall require less than 20 MB disk space and 32 MB RAM

The selected architecture supports the size and performance requirements through the implementation of a vehicle architecture and client-server architecture. The client portion is implemented on the user's smartphones. Minimal disk and memory requirements are needed on the PC client portion. The client portion shall be a well-performing application on a smartphone that facilitates the delivery of information or requests from the user to the system. The vehicle architecture shall take in and process the information or requests, quickly sending the appropriate instructions to steer and navigate to best fulfill the user's request. The vehicle architecture shall also be able to detect hazards that get in the way of fulfilling the user's requests. The vehicle architecture shall be able to notice unintended results from in actions, reading them as errors, and send them to the system of the car.

10. Quality

The software architecture supports the quality requirements, as stated in the System Requirements Spec:

1. The interface shall feature a Linux-based operating system

2. The network shall have an average uptime of 12 months

3. The system shall update the vehicle's position and velocity to the control module about 99.999999% of the time

4. The GPS shall generate available routes at around 95%

5. When the GPS is not available, the client will use the previous map data to navigate

6. The system shall be portable whereas all updates will be compatible with previous and future models

7. The system will have security on its network and locally to protect against unauthorized activity and prevent hacking.

Appendix A: Architectural Design Principles

The “4+1” View Model is used to present the architectural overview of Autonomous Auto’s Navigation and Automatic Steering System (NASS). It features the Logical View, Process View, Physical View, Development View, and Scenarios.

The Dependency Inversion Principle is the principle that complex modules should not depend on lower-level modules. For example, our control module is in full control of the system and all modules under it had to send information to the control module only. The OBD, steering actuators, accelerators, and brakes depended on the control module as shown in the deployment view. The interface depended on the network in order to show the user information and the GPS to show the map info and routes.

The Interface Segregation Principle defines that interfaces should be as explicit, client specific, and easy to use. These clients should not depend on the functionality that they don’t use. The system sequence diagrams include this principle where the interface sends Status Change Messages to the Control Module. The interface receives messages from the network and the GPS as a singular system that won’t depend on the other modules.

The Open/Closed Principle is the principle where an extension of system behavior is brought out without modifying it. This principle is used mostly in our development view which features the packages we used in the project. Most of our modules call other modules for messages and information but won’t actually modify the variables and functions within those modules.

The Single Responsibility Principle is used when every module has one responsibility so that each intended purpose is easy to understand and leads to fewer errors. The state diagram in the logical view is a great example of this principle. Each state has its own responsibility described and its various uses that are clear and intended.