Software Architecture Document

BEZZLA

Bezzla's Autonomous Navigation and Driving System (BANDS)

Mar Christian Contreras, Bradley Diep, Vincent Diep, Lisa Hong, Marius Iacob, Edwin Lam, Alex Lu, and Gabriel Warkentin

10/19/2022

Revision History

Date	Version	Description	Author
09/23/2022	1.0	Added the purpose and fixed up the formatting of the document	Gabriel Warkentin
09/25/2022	1.1	Added the scope, and the definitions, acronyms, and abbreviation	Lisa Hong
09/27/2022	1.2	Added the Architecture Goals and Constraints	Gabriel Warkentin
9/30/2022	1.3	Added Use Case diagram and Use Case descriptions	Vincent Diep
10/01/2022	1.4	Added Logical & Process View Descriptions	Alex Lu
10/02/2022	1.5	Added Size and Performance descriptions	Alex Lu
10/04/2022	1.6	Created the diagram for the logical view	Gabriel Warkentin
10/05/2022	1.7	Added the diagrams for physical and development view	Lisa Hong
10/11/2022	1.8	Added Process View and Process View descriptions	Bradley Diep
10/12/2022	1.9	Revised and condensed Architectural Representation	Mar Christian Contreras
10/18/2022	2.0	Revised and condensed section 4.0 and updated the styling of the entire document	Mar Christian Contreras
10/19/2022	2.1	Revised Size and Performance descriptions, added Quality details, and References	Alex Lu
11/10/2022	2.2	Updated Figures to Links	Gabe Warkentin
11/23/2022	2.3	Included link to process view, revised “Architectural Goals and Constraints”, and revised “Size and Performance”	Bradley Diep
11/25/2022	2.4	Updated and linked the Table of	Lisa Hong

		Content	
11/27/2022	2.5	Added new high-level logical view, Messaging view, CM function view Updated table of contents and figures	Gabriel Warkentin
11/28/2022	2.6	Added CM Significant classes	Gabriel Warkentin
11/29/2022	2.7	Added the diagrams for the Logical View	Lisa Hong
11/30/2022	2.8	Updated 4.1 diagram and use cases to have reference	Alex Lu
12/1/2022	2.9	Updated section 10 to include missing points in order to fully conform to SRS	Mar Christian Contreras
12/02/2022	3.0	Updated section 9	Marius Iacob
12/03/2022	3.1	Made adjustments to Size and Performance	Alex Lu
12/04/2022	3.2	Revised Section 7 to include diagrams in the Process View	Bradley Diep
12/05/2022	3.3	Removed US National Safety Council as actor for Use Case Diagram. Mentioned Driver’s Interface package from Figure 6.1	Vincent Diep

Table of Contents

Table of Figures

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of Bezzla's Autonomous Navigation and Driving System (BANDS), using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

This Software Architecture Document provides an architectural overview of the Autonomous Navigation and Driving System. This system is currently under development by Bezzla to guide automobiles around the world from their current location to their desired destination. The Autonomous Navigation and Driving System will utilize a computer system that controls the steering actuators, GPS system, handle speed change requests, detect oncoming obstacles, and presents a simplified interface that the user can interact with the system and vice versa. Although the system has some degree of autonomy, the system is not meant to be relied on completely. The driver is still required to be focused and attentive to the road in the scenario that the system malfunctions or must be suspended.

1.3 Definitions, Acronyms, and Abbreviations

ASR - Architecturally significant requirements; the measurable requirements that affect the architecture of a system.

BANDS - Refers to the name Bezzla's Autonomous Navigation and Driving System.

Control Module (CM) - The collection of an automotive component of sensors, actuators, and basic controls that work together to operate motor vehicles.

Driver's Interface (DI) - The physical display where the driver can interact with the vehicle.

GPS - Global Positioning System; a navigational system that utilizes orbiting satellite signals to adjust the location of a radio receiver on the earth's surface.

GPS Receiver (GR) - The device that collects data on geographic locations by receiving transmissions from the GPS satellites.

OBD-II System (OS) - The On-Board Diagnostic II; is the second generation of a device that allows telematics devices to process the information of a vehicle such as the engine revolutions, fuel usage, and vehicle speed.

Obstacle Detection System (ODS) - The sensors that sense slow-moving or stationary objects in front of a vehicle that can provide warning to the control module of impending collisions; can potentially brake or change the speed of the vehicle.

Speed Control System (SCS) - The system that automatically controls the speed of a vehicle.

Steering Actuators (SA) - The device that aids the steering of a vehicle.

VIN - Vehicle identification number. The fingerprint of the car.

1.4 References

1. Backlog
    a. https://docs.google.com/spreadsheets/d/1P06jZ1lFB96F8y6WRjzsZCumeeSdUbbL8dK12LYvIUs/edit?usp=sharing

2. Presentation
    a. https://docs.google.com/presentation/d/1g0fmgVVL-Q81Yoa11hmGALrzBX8we6w8dh6oUKZRJJE/edit?usp=sharing

3. Navigation and Steering System Requirements Version 15, Aug 06, 2022

4. Software Architecture for Developers, Simon Brown

5. An Introduction to Software Architecture, David Garlan and Mary Shaw

6. Software Architecture, A. Bijlsma, B.J. Heerendr., E.E. Roubtovair, S. Stuurman

7. Software Architecture in Practice, Len Bass, Paul Clements, Rick Kazman

8. Architecture Blueprints The “4+1” View Model of Software Architecture, Phiippe Kruchten

1.5 Overview

This document serves as the documentation of Bezzla's Autonomous Navigation and Driving System software architecture. The document covers the architectural goals and constraints, the “4 + 1” model such as the use-case view, logical view, development view, process view, physical view, and data view. Covered further in the documentation includes the size and performance and quality requirements.

2. Architectural Representation

The architecture of the Autonomous Navigation and Driving System will be represented by the **4+1 views** diagram.

Use-Case View will describe the specific scenarios and interactions of the vehicle and its surrounding environment as well as how the driver will be able to operate the vehicle.

Logical View will deal with the functionalities of the Autonomous Navigation and Driving System. Some highlights of the many areas it will cover are more specifically the vehicle's state, automated steering, and its connectivity to our servers for AI data.

Development View depicts the underlying intricacies of the system. More specifically, it will go over the organization and implementation of the code by the programmer.

Process View focuses on the successful communication between the system's dynamic elements all while the system is running.

Physical View will be concerned with the implementation of the architecture onto specific and physical technologies.

3. Architectural Goals and Constraints

When driving motor vehicles, many factors inside and outside of them affect the driving experience. Anything could happen on the road and it is always a good idea to be prepared for any situation. Given the unpredictable nature of motor vehicles, the BANDS system within the BEZZLA vehicles prioritizes and enhances the user experience through safety, speed, uptime, and security.

The system will be equipped with an OBD-II system in accordance with international laws. OBD-II helps with the safety of the vehicle in that it detects errors within the car's system, whether it be regarding the drivetrain or the engine itself. The OBD-II will also detect problems including the autonomous driving system as well as safety measures such as airbag malfunctions. By providing codes and alerting the driver/user that there are issues with the vehicle, it will prevent accidents on the road in the case that the codes and vehicle starts breaking down.

The BANDS system should be REUSABLE in many different vehicles, but the computer hardware can be more limited/specific. It should be portable to future models and capable of constant remote updates. The software will locally store the connected vehicle's VIN. Whenever the software is booted up, the locally stored VIN is compared against the connected vehicle's VIN. If the VIN is not the same then the software is ported to a new vehicle and the software will reconfigure and show the setup screen.

The main functional areas should be disparate standalone components, allowing replaceable pieces, containing errors and updates, integration with COTS, and separation of concerns. BANDS will use a Message-oriented middleware to facilitate communication between modules.

Since GPS receivers are a tried and true technology, BEZZLA will be using a COTS hardware-software product.

Safety features of the ODS and SCS will all comply with the statutes of the US National Safety Council.

4. Use-Case View

The Use-Case View is important for the many sets of scenarios that will represent the core functionality of the system.

Significant use-case realizations:

- Route generating and selecting
- Current route status
- Speed Change generating and requests
- Resuming of automatic steering
- Suspension of automatic steering

4.1 Use-Case Diagrams

The use-cases explain in detail the purpose each subsystem and function serve in order to cover certain scenarios.

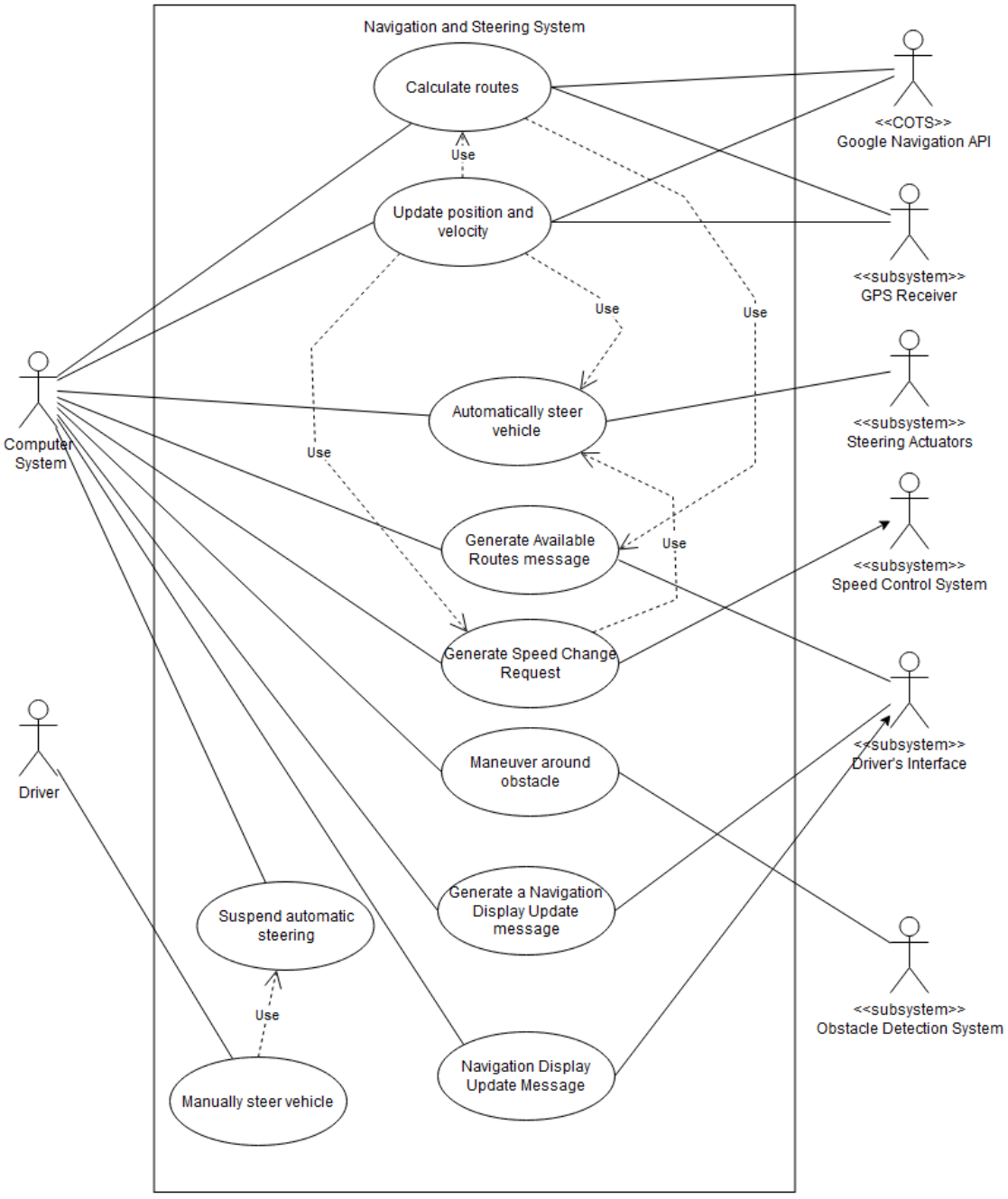

Figure 4.1 Architecturally-Significant Use Cases

4.1.1 Calculate Routes

This use case allows the computer system to calculate routes to the specified destination. Routes are calculated using Google's Navigation API. As shown in Figure 4.1, routes are calculated based on the vehicle's current position to the specified destination.

4.1.2 Update Position and Velocity

This use case allows the computer system to update the vehicle's position and velocity. The position and velocity are updated for every Position Velocity message received by the GPS Receiver. As shown in Figure 4.1, when the driver suspends autonomic steering, position and velocity will be updated.

4.1.3 Automatically Steer Vehicle

This use case allows the computer system to automatically steer the vehicle. This includes ceasing and resuming automatic steering. As shown in Figure 4.1, steering commands are sent to the steering actuator and are calculated based on the vehicle's current position and velocity.

4.1.4 Generate Available Routes Message

This use case allows the computer system to generate Available Routes Messages to the Driver's Interface for the Driver to see. This includes No Available Routes Messages if no routes could be found. As shown in Figure 4.1, when the user utilizes the calculate routes function, it will generate the available routes & send a message back to the user.

4.1.5 Generate Speed Change Request

This use case allows the computer system to generate Speed Change Requests to the Speed Control System. These Speed Change Requests are used when automatically steering the vehicle. As shown in Figure 4.1, the use-case demonstrates how when the computer system requests for a speed change request, it will receive information from the Speed Control System and return that infromation to the computer system.

4.1.6 Maneuver around Obstacle

This use case allows the computer system to maneuver around obstacles if an obstacle is detected by the Obstacle Detection System. As shown in Figure 4.1, when the computer system indicates that it needs to maneuver around an obstacle, it will go to the subsystem "Obstacle Detection System" and retrieve the algorithms in order to avoid the object indicated.

4.1.7 Generate Navigation Display Update Message

This use case allows the computer system to generate Navigation Display Update Messages to the Driver's Interface. This includes notifying the driver to take manual control of the car, notifying the driver that automatic steering has ceased, and the vehicle's current position. As shown in Figure 4.1, when the computer system wants to generate a navigation display update message, it will take information from the driver's interface and display it back to the user, in this case that the automatic steering was ceased..

4.1.8 Generate System Status Message

CR Approved SN-0033:
This use case allows the computer system to generate a Navigation Display Update Message to the Driver Interface in the event that automatic steering has ceased and when the driver is required to select a new route. The driver's interface will suggest that the driver must select a new route if it's off its selected route. Different routes will be displayed, if there are any, and the driver will be able to select one. As shown in Figure 4.1, when automatic steering was ceased, the driver will be notified that through the Navigation Display Update Message as well as indicate to the user that they will have to select a new route in the case that it is not the selected one.

4.1.9 Manually Steer Vehicle

This use case allows the driver to take control of the vehicle and manually steer the vehicle at any time when the computer system is automatically steering the vehicle. The actor in this use case is the Driver. As shown in Figure 4.1, if the driver starts manually steering the vehicle, or has both hands on the wheel, it will suspend automatic steering and send information to the computer system that it is now in manual driving mode.

4.1.10 Suspend Automatic Steering

This use case allows the computer system to suspend the automatic steering of the vehicle when the driver takes manual control over the vehicle. As shown in Figure 4.1, we see that the system will suspend automatic steering in the case that the driver turns on manual steering mode, which is indicated by the driver having both hands on the steering wheel.

5. Logical View

The Logical View covers the composition of the subsystems and packages working with each other to make the functionalities of the system.

5.1 Overview

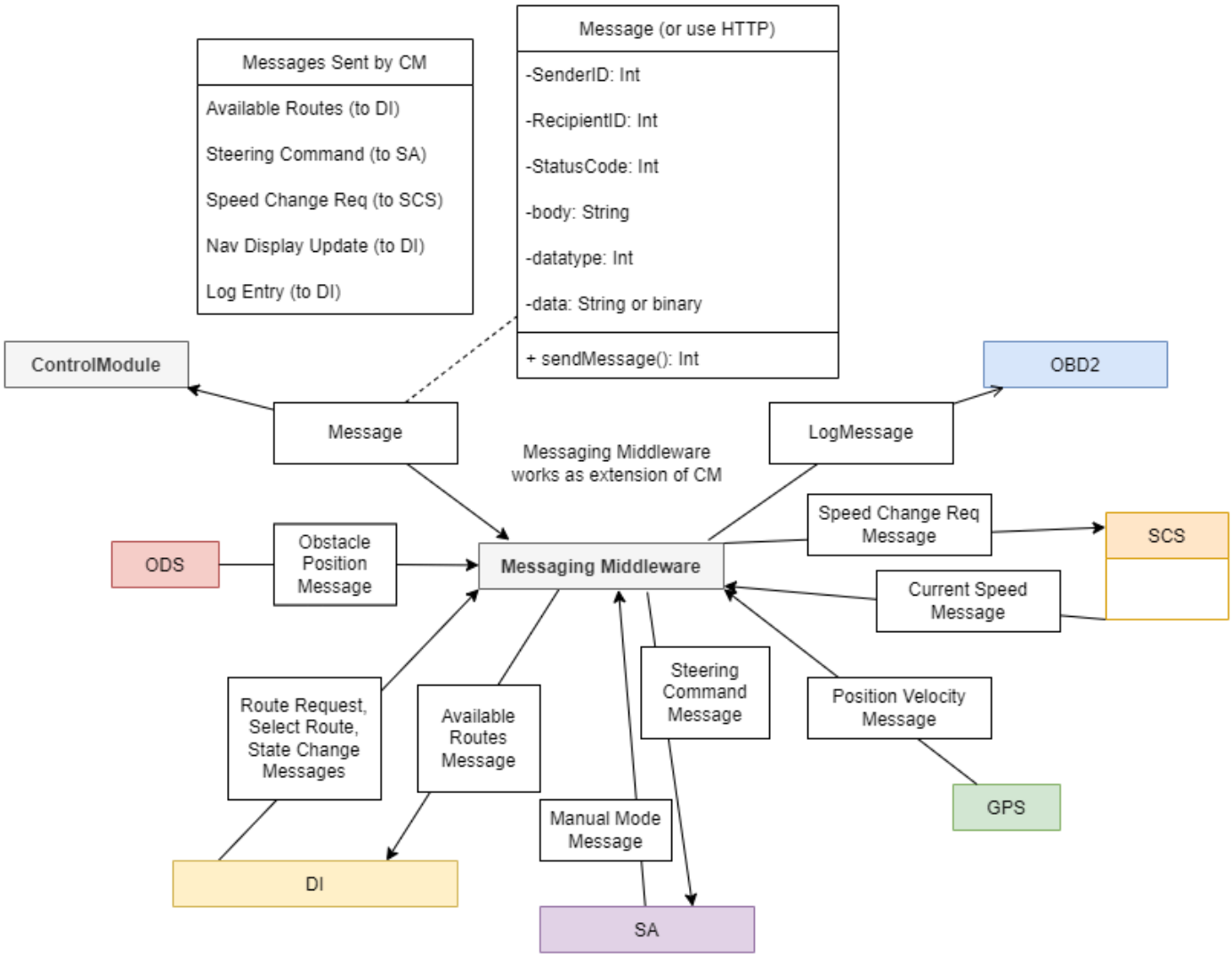

Figure 5.1 Logical View Diagram - High Level Messaging Design

The logical view of the BANDS system is operated under one main component, the Control Module, which works with its 6 subsystems. These subsystems include the OBD-II System, Speed Control System, GPS System, Steering Actuators, Driver's Interface, and Obstacle Detection System. These subsystems are separate entities that communicate via messages using a Message-oriented middleware (actually an extension of the CM) to intermediate.

5.2 Architecturally Significant Design Packages

5.2.1 Messaging-Oriented Middleware

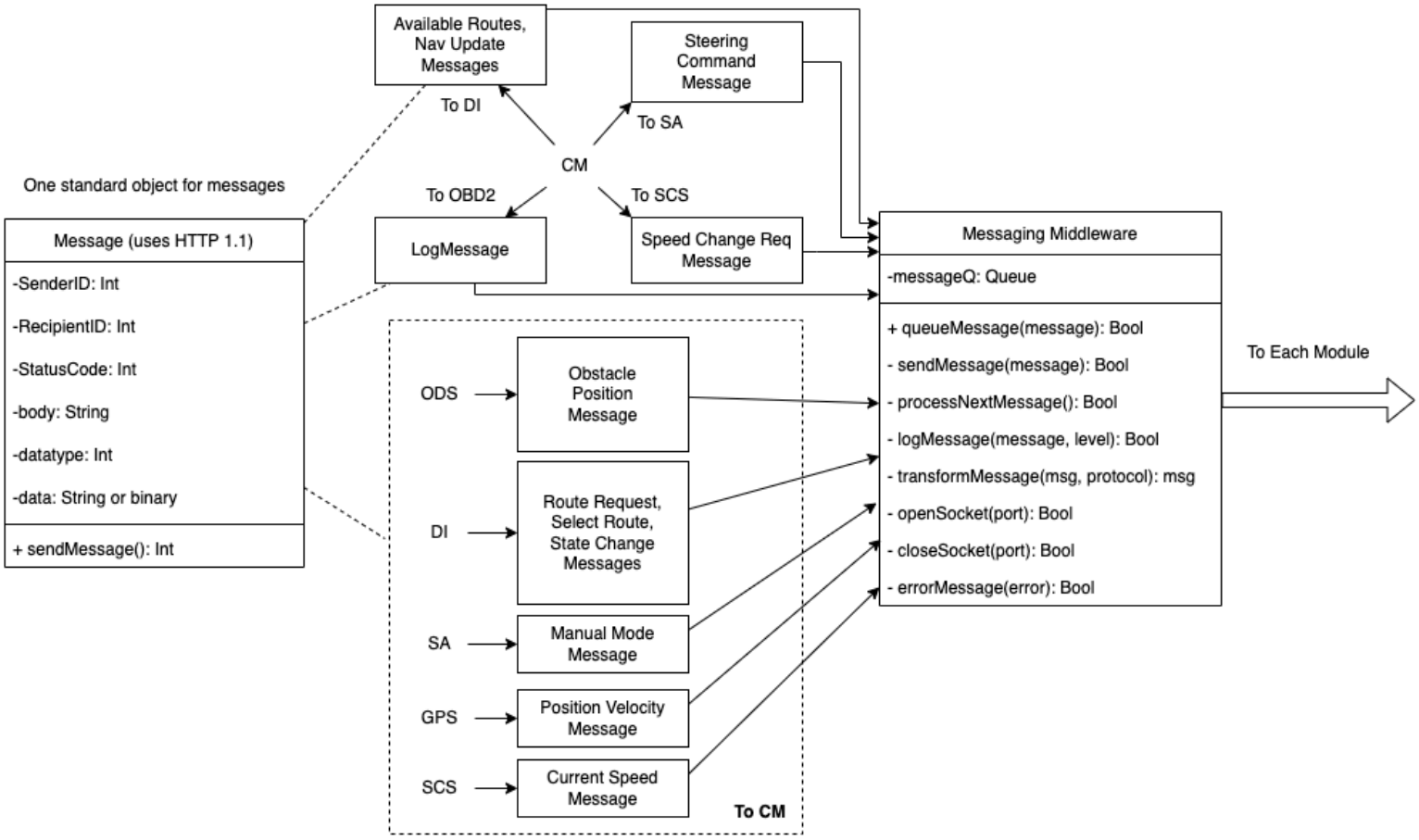

Figure 5.2.1 Logical View Diagram - Messaging Oriented Middleware

The Messaging-Oriented Middleware (MOM) receives messages from the various modules and redirects them to their appropriate sources. This helps keep the components decoupled in order to make use of COTS products and reusable software components. The MOM can translate messages between common protocols and data objects (JSON, XML, etc.) to facilitate working with a variety of softwares. It utilizes a priority queue to ensure high profile messages get through as fast as needed, and efficiently distribute messages without bottlenecks.

As seen in Figure 5.2.1, most modules only send messages to the CM, and the CM to each module in return. This ultimately functions much like a client-server model.

The core component messages utilize HTTP (Hypertext Transfer Protocol) since it is efficient, well documented, and easy to work with from a variety of languages. *Some of the key variables are outlined in the Message object in the left of Figure 5.2.1.*

5.2.2 Control Module

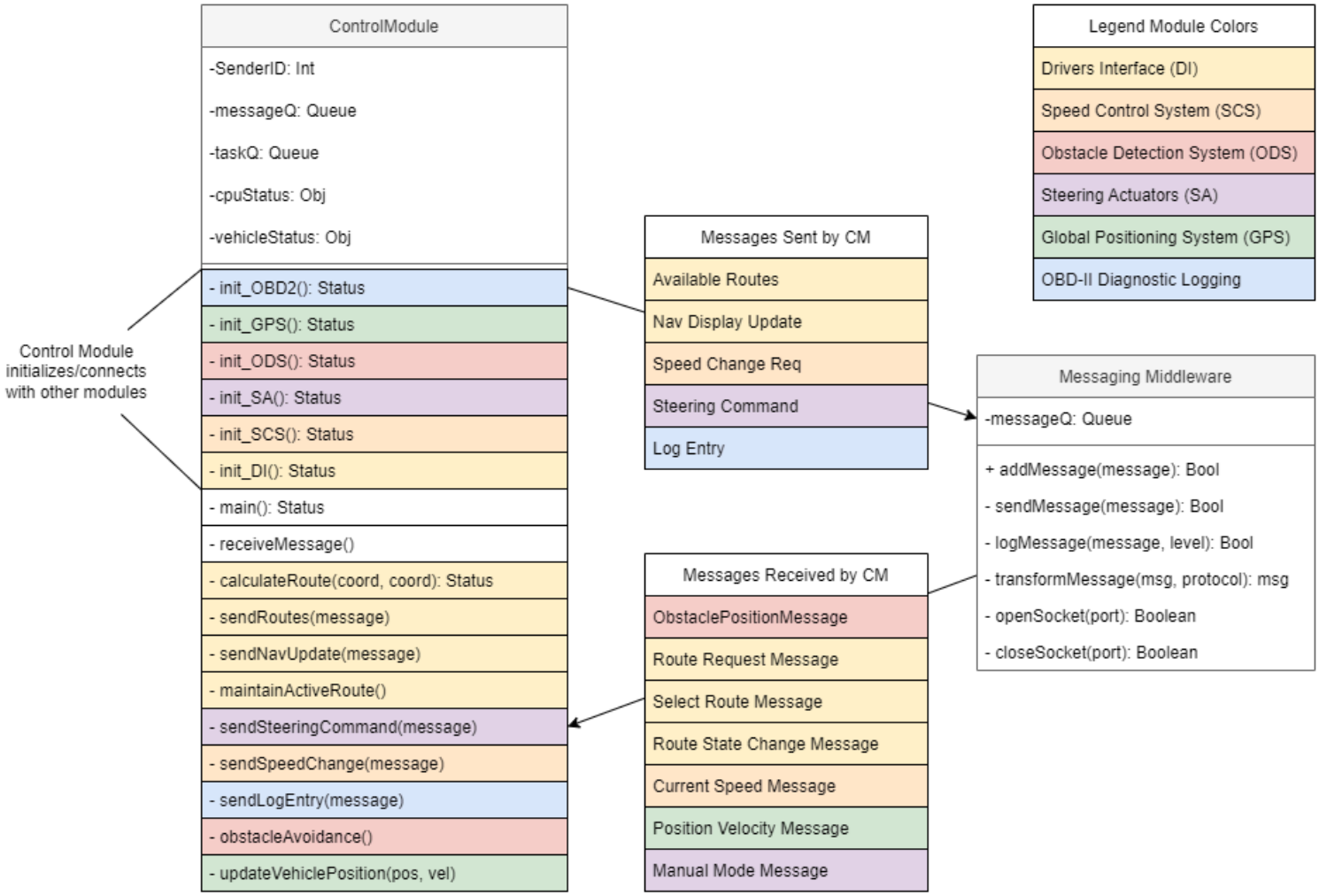

Figure 5.2.2 Logical View Diagram - Control Module Function View

The Control Module handles the most traffic and information, facilitating the entire vehicle's operation. First it initializes connections with all of the modules, either creating the processes on the host machine or finding an existing up system. Then it begins receiving status updates regularly from the GPS, SCS, ODS, and DI. Using these messages, the control module updates the vehicle status object, and uses that information to calculate routes, control steering and acceleration, avoid objects, and alert the driver of route changes and manual control requirements.

The CM regularly sends status updates to the OBD2 system with a snapshot of the current status, and what changes were made since the last update. These messages are guaranteed delivery by the Messaging Middleware to the OBD2.

5.2.3 OBD-II

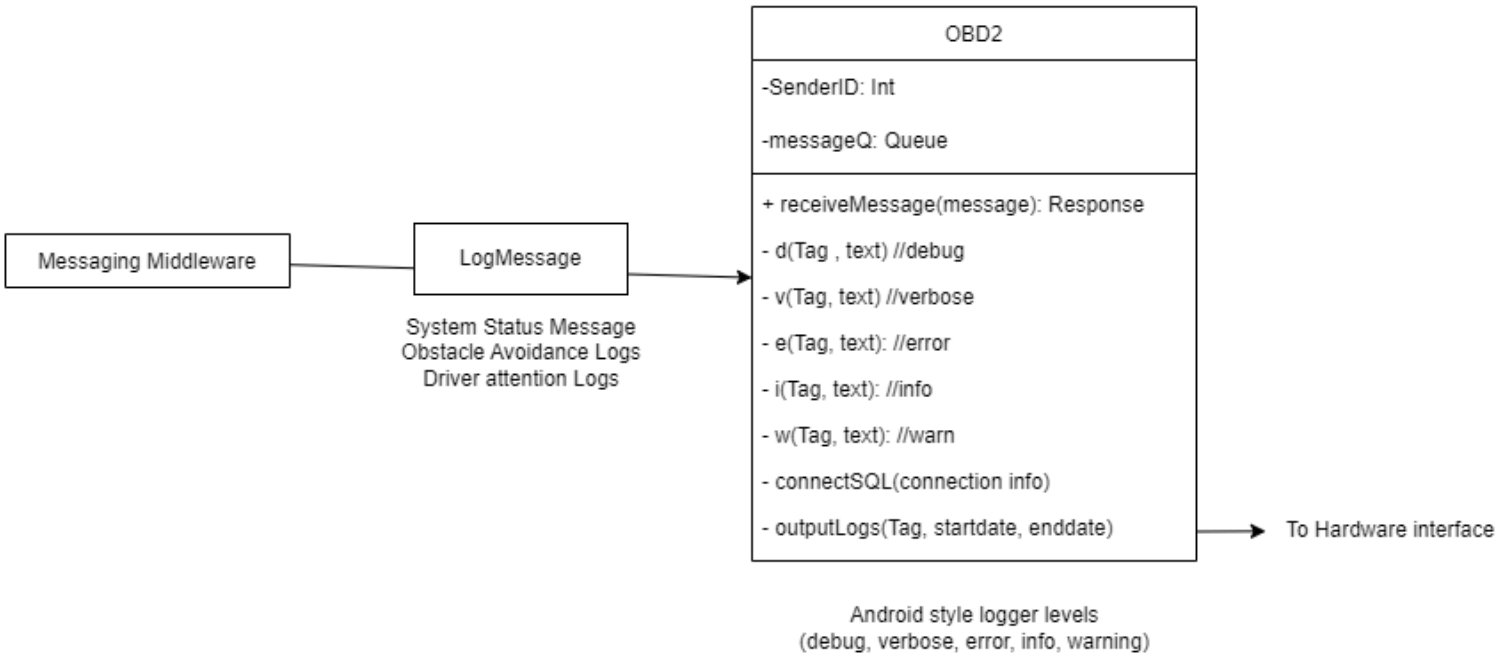

Figure 5.2.3 Logical View Diagram - OBD-II

The OBD-II System manages the diagnostics of the vehicle and BANDS system, logging information on the software and hardware systems received as System Status Messages from the Control Module. The data can be output via the standard OBD-II hardware interface in accordance with government regulations.

5.2.4 Global Positioning System (GPS)

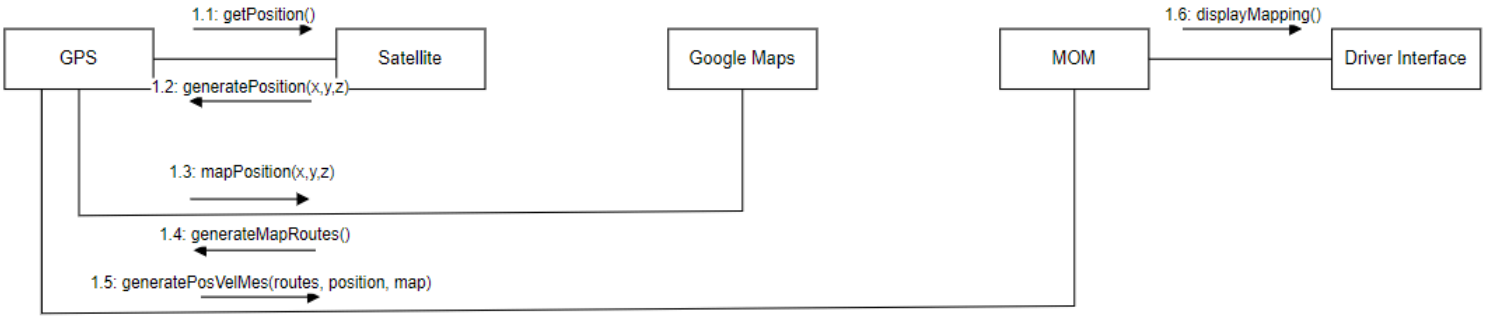

Figure 5.2.4 Logical View Diagram - GPS

The GPS System determines the current and on-going position velocity of the vehicle when taking routes manually and autonomously. As shown in Figure 5.2.4, the GPS communicates with satellites in order to determine the positional data of the vehicle. The GPS would then communicate with the Google Maps Platform API in order to generate map data that would be sent to the Messaging-Oriented Middleware Control Module extension to be sent to the Driver's Interface. This information is maintained by the Control Module to determine other important components' functionality such as speed changes & additional routing procedures.

5.2.5 Object Detection System (ODS)

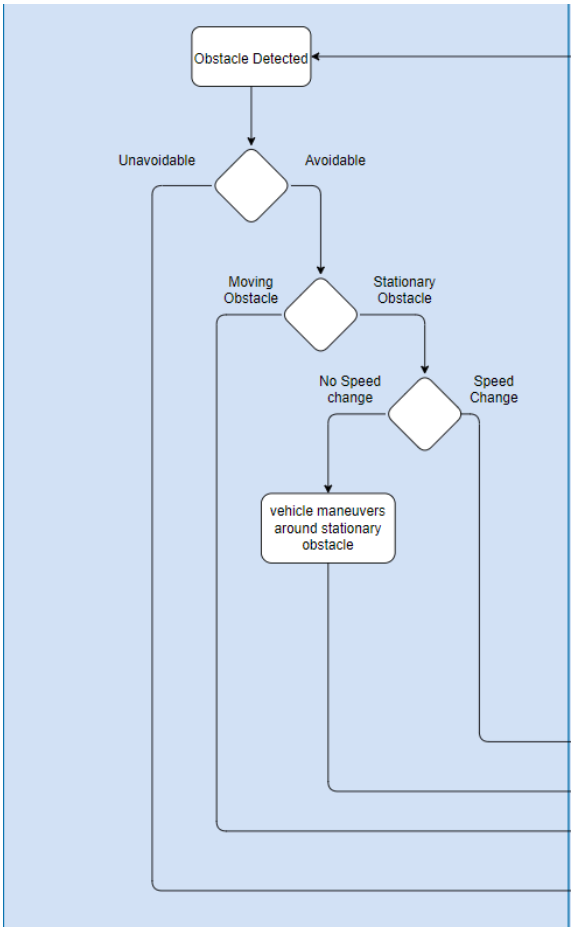

Figure 5.2.5 Logical View Diagram - ODS

The Obstacle Detection System is a software that utilizes the sensors on the car to determine whether or not obstacles on the road will make contact with the vehicle and cause potential damage/harm. The system will immediately notify the user on the Driver's Interface where they can then take over the steering, or allow the BANDS system to avoid the obstacle itself.

5.2.6 Steering Actuators (SA)

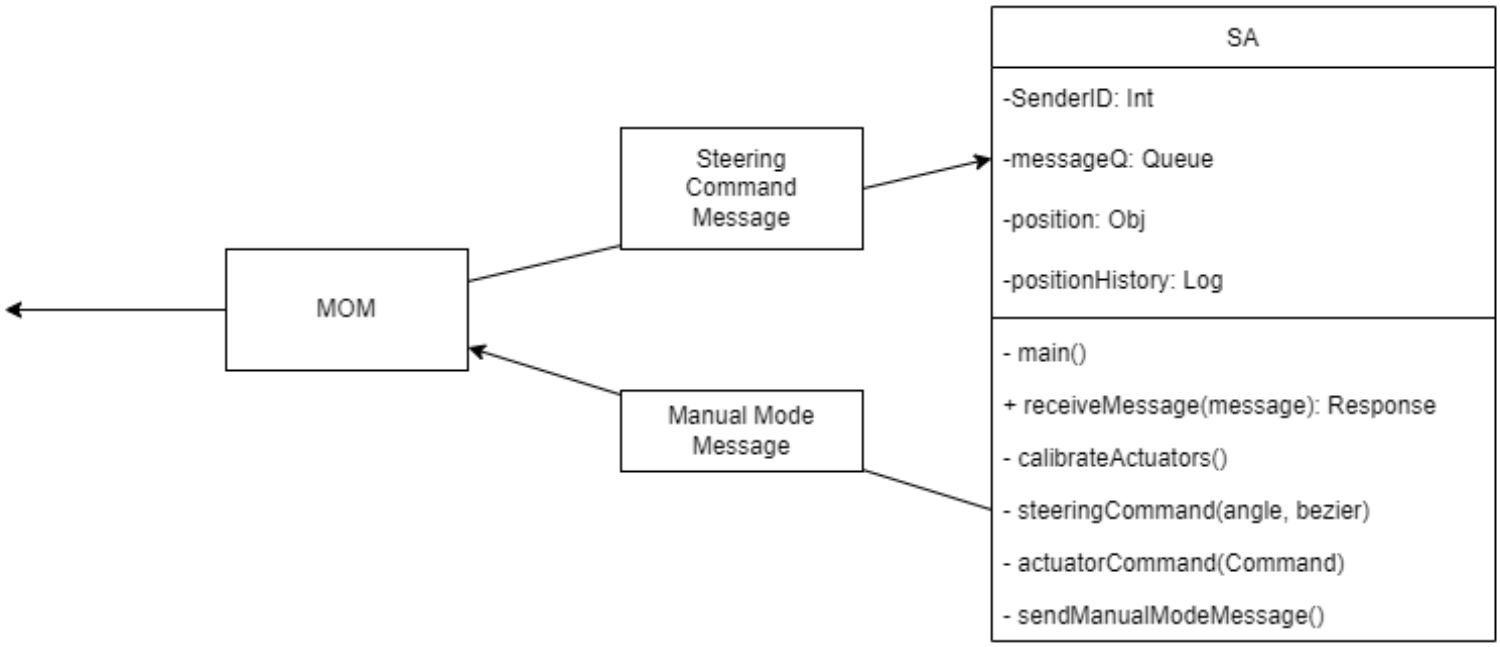

Figure 5.2.6 Logical View Diagram - SA

The Steering Actuators are components that exist in modern cars which assists in the steering of the vehicle. This is used in both manual mode where the user is in control of the vehicle as well as during the autonomous steering mode.

In terms of autonomous steering, it receives information from the Speed Control System, GPS Receiver, and other components that send information to the Control Module to determine how to steer autonomously.

5.2.7 Speed Control System (SCS)

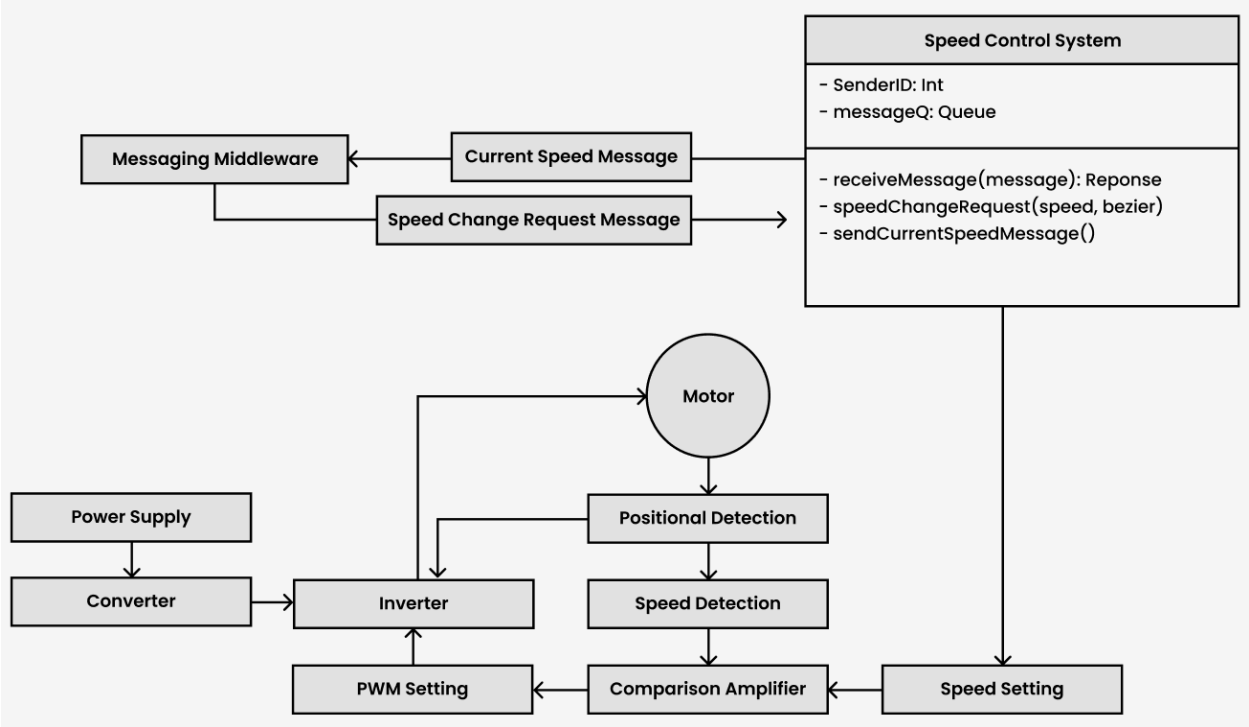

Figure 5.2.7 Logical View Diagram - SCS

The Speed Control System receives and sends data regarding the current speed that the vehicle is going. It also receives input from the user in the control module when making a speed change request and interprets that request.

5.2.8 Drivers Interface (DI)

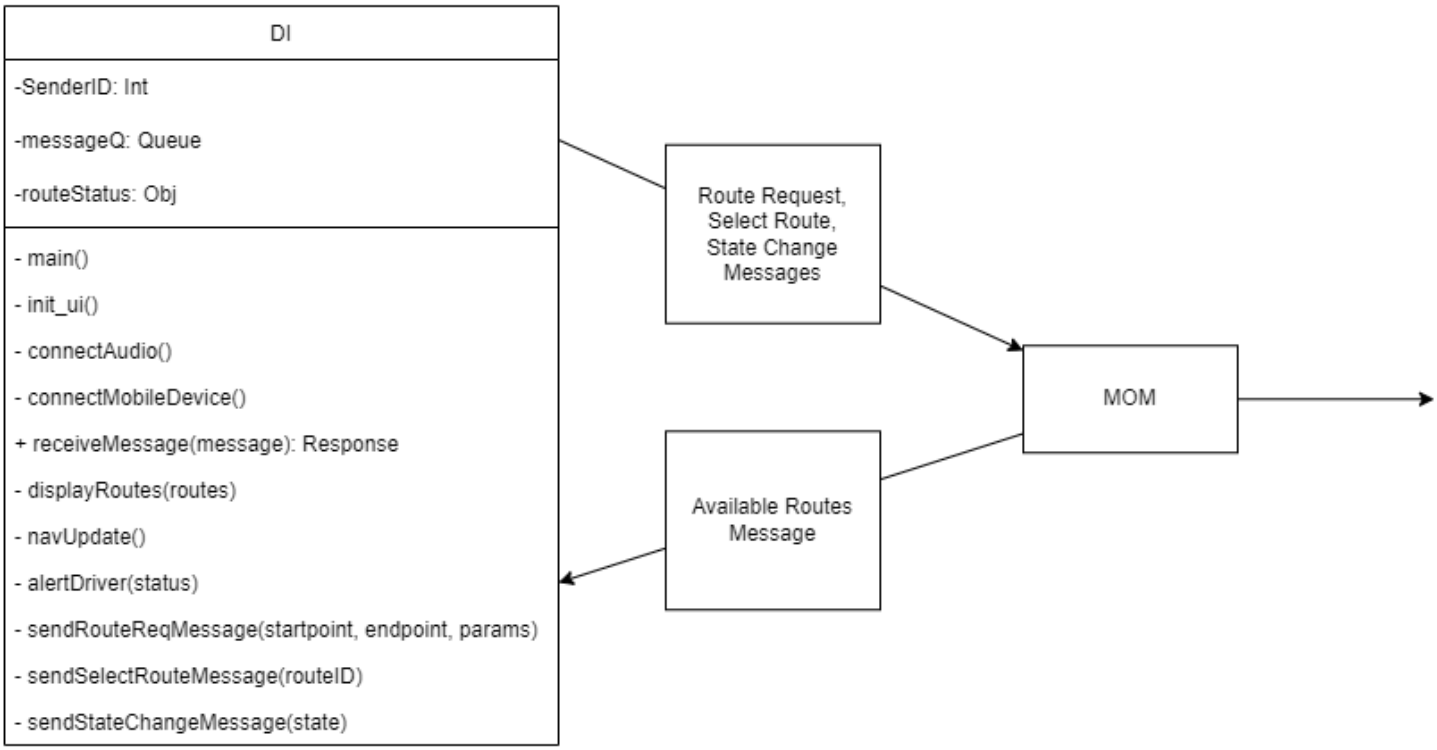

Figure 5.2.8 Logical View Diagram - DI

The Driver's Interface is the UI(User Interface) where the user is able to utilize the BANDS system. It allows the user to receive various & optimal routes to get to a specific location. The interface will have a map view with live location and different navigation routes that also gets the vehicle to the specified location. It will allow the user to engage autonomous steering and receive information regarding navigation updates and changes to the autonomous steering, such as switching to manual mode.

5.3 Architecturally Significant Design Classes

5.3.1 Control Module - Significant Classes

The Control Module itself utilizes several Classes internally to handle complex tasks. The most significant are **Routing**, **Autonomous Driving**, and **Object Avoidance**.

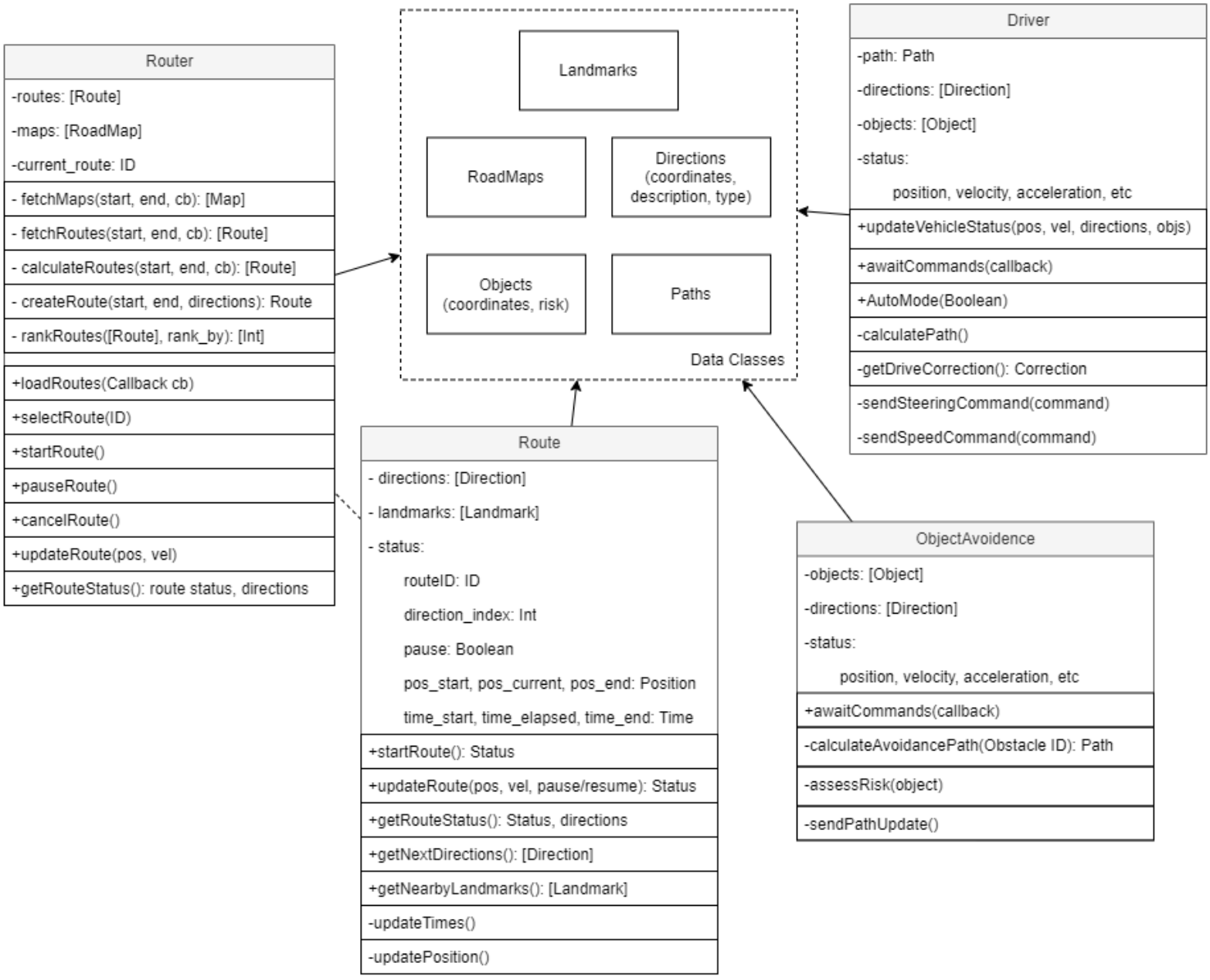

Figure 5.3.1 Control Module Classes - Routing, Driving, Object Avoidance

Routing - The Router Class uses maps and position information to create, update, pause, and cancel routes.

Autonomous Driving - The Driver Class determines paths to stay on route given the route directions, the vehicle movement status, and sends commands to the SA and SCS to direct the vehicle. Utilizes object information to keep safe distance from cars ahead.

Object Avoidance - The ObjectAvoidance Class catalogs the nearby objects and determines avoidance paths to override the standard path during an emergency situation.

REUSE - All Classes in CM share common smaller data classes as seen in *Figure 5.3.1* above.

6. Implementation/Development View

6.1 Control Module Development View Diagram

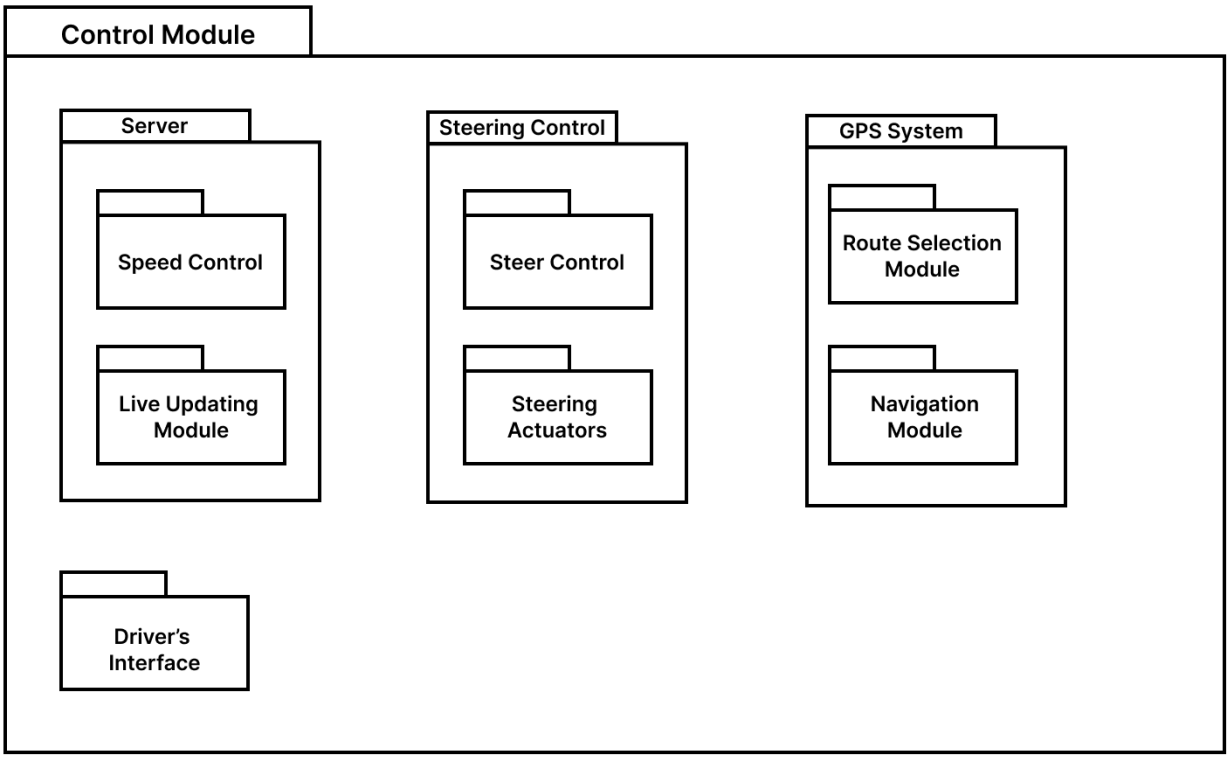

Figure 6.1 - Control Module Development View Diagram

Nested in the control module are the main modules: the Server, Steering Control, GPS System, and Driver's Interface. Within the server module are the speed control and live updating modules. In the steering control module are steer control and steering actuators. Lastly, in the GPS system module are the route selection module and navigation module. Each of the items is there to help process information for their respective modules.

6.2 BANDS Inheritance View

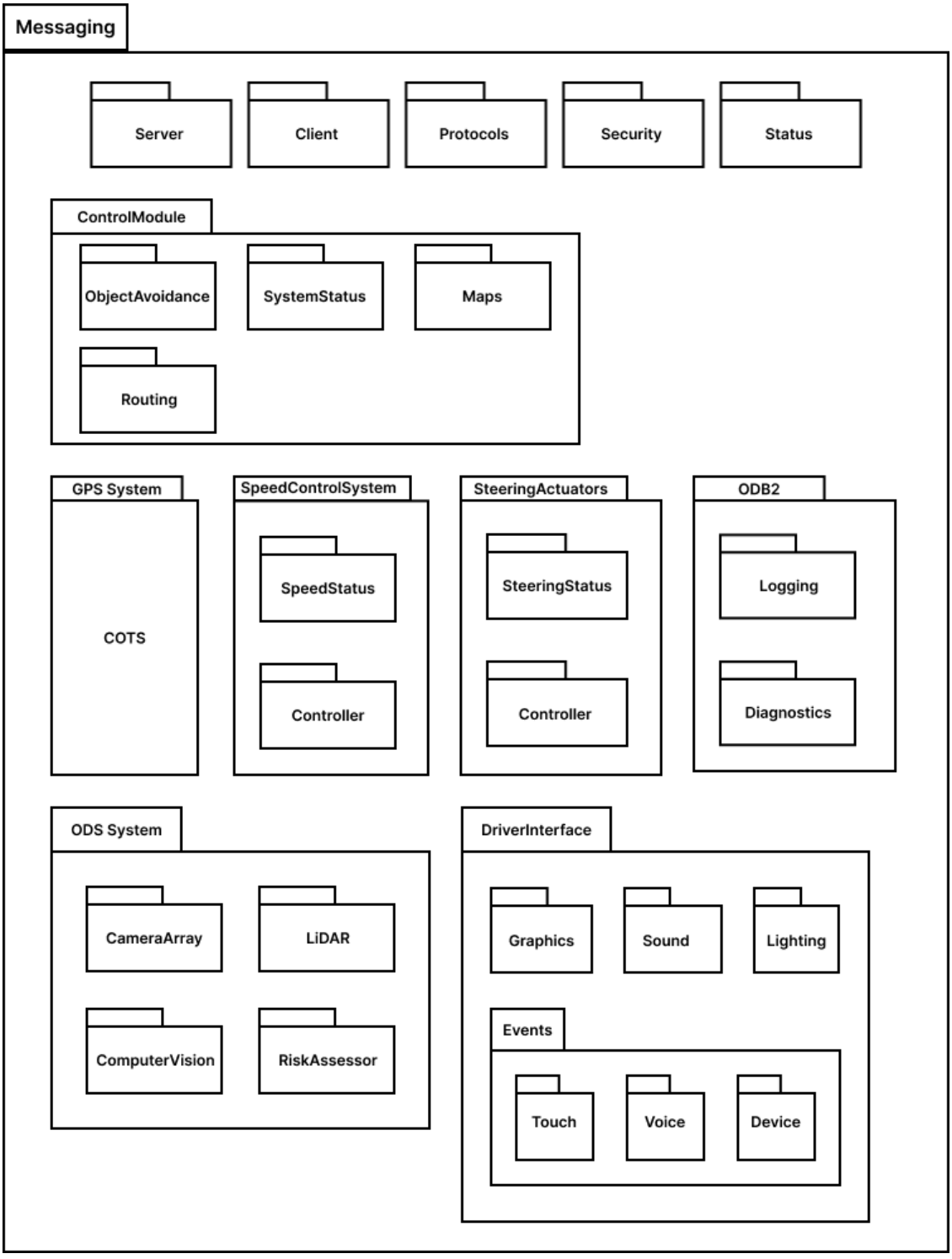

Figure 6.2 - BANDS Inheritance View Diagram

All of the components require Messaging functionality, so they all inherit from the Messaging class/library. The server, client, protocols, security, and status features are all available to the major components. The major components in turn have additional functionality like SpeedStatus and Routing. The Control Module initializes a base Messaging object as the Messaging-oriented Middleware, and then initializes the rest of the modules. The GPS system will be COTS, the MoM can handle and transform many protocols so aren't vendor locked.

The ODS ComputerVision and Risk Assessors will utilize machine learning algorithms developed in Python using scikit-learn, TensorFlow and Pandas.

The CameraArray and LiDAR will be built in C++ for performance

The Driver Interface will be built using Android Automotive. This will allow for easy updates and clean design.

The GPS is COTS

The Speed Control System and Steering Actuators will use C++ for the robotics libraries and real-time performance

The OBD-II system will be implemented in Python since performance isn't crucial

The Control Module will be a mix of Python and C++, Python for the core logic and status information, C++ for challenging calculations like Routing and ObjectAvoidance

REUSE - Modules share common messaging and security functionality

7. Process View

The Process Model illustrates the subcomponents of the BANDS system organized as executable processes that come together to form the Control Module. These processes exist to support the functionality of the subcomponents such as retrieving available & optimal routes for navigation, displaying route changes, changing speed & steering modes, and the various other features BANDS provides.

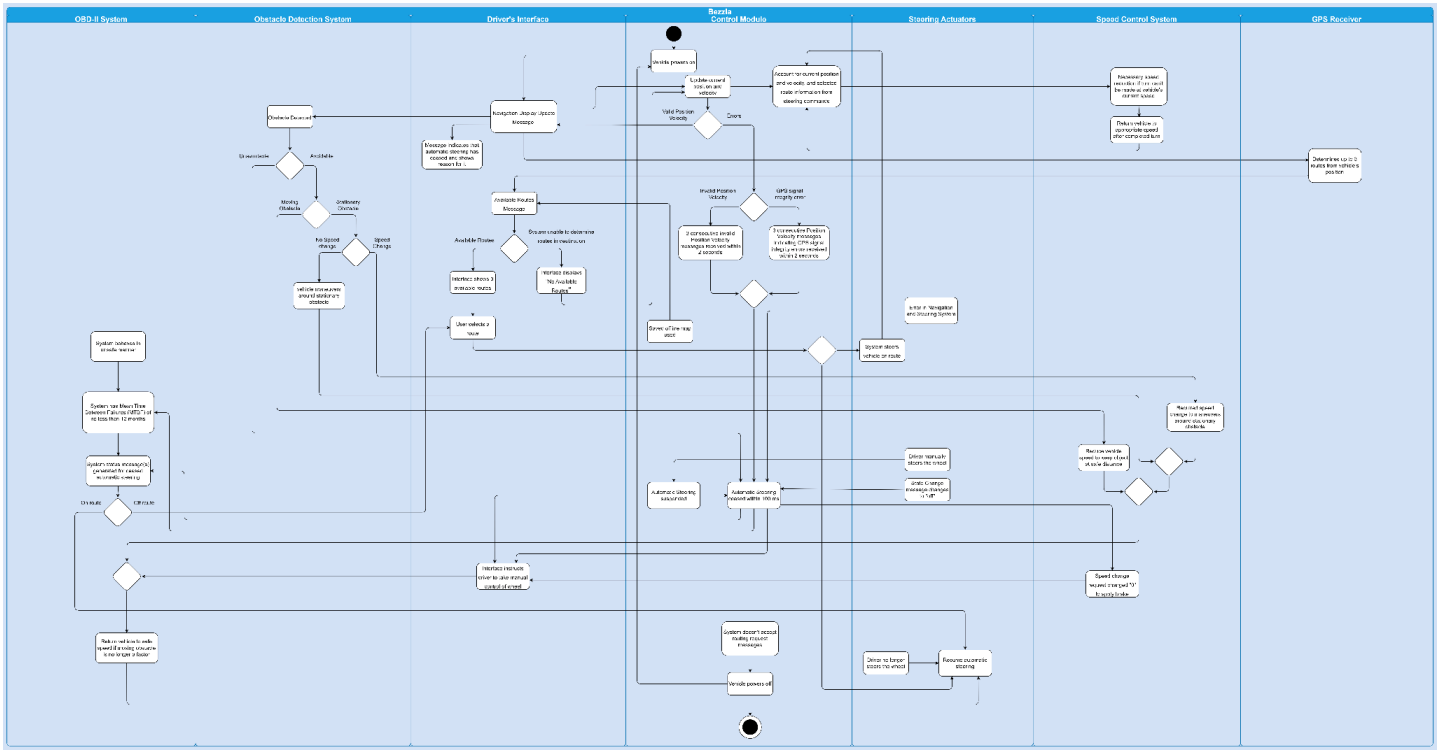

Figure 7.1 - Process View (Activity Diagram)

For a better look at the Process View, visit this link: https://tinyurl.com/bezzla-process-view.

7.1.1 OBD-II System

Upon receiving System Status Messages from the Control Module, the OBD-II System will maintain the vehicle's stability and health by detecting its current status as shown in Figure 7.1.1.

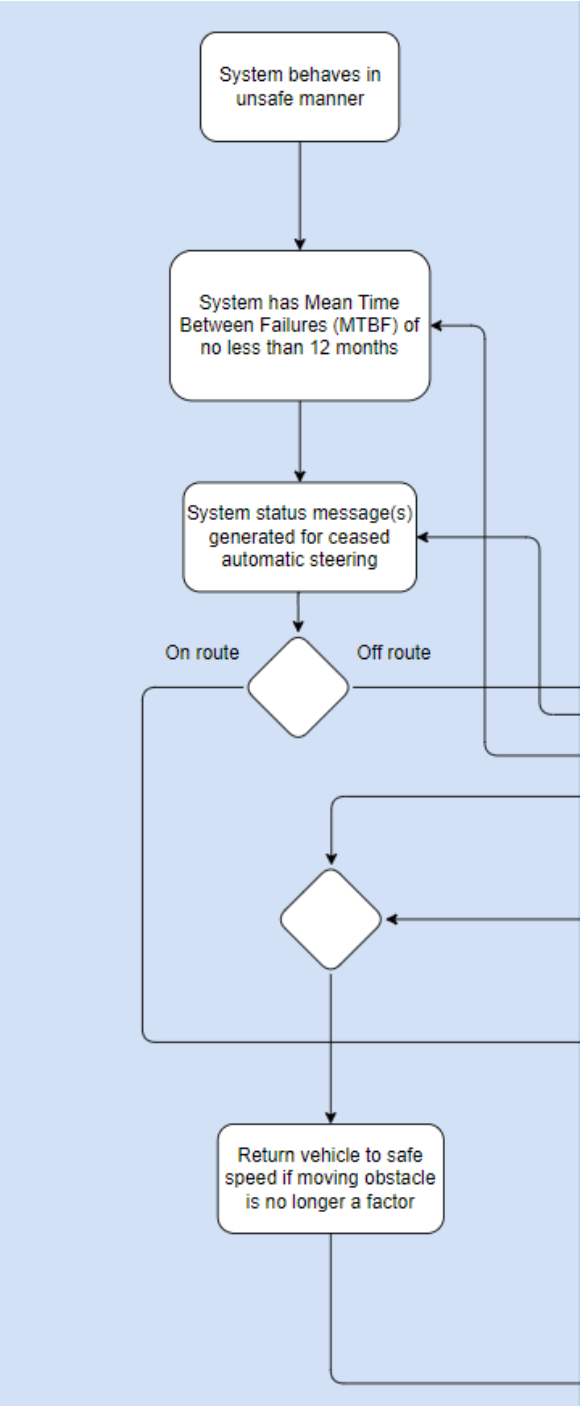

Figure 7.1.1 - Process View - OBD-II System

7.1.2 Obstacle Detection System

As shown in Figure 7.1.2, the Obstacle Detection System (ODS) will determine the vehicle's actions to avoid an obstacle by utilizing the vehicle's current position and velocity of the vehicle from the control module. Depending on the state of the vehicle and the obstacle, the ODS will plan a process for the vehicle to avoid the obstacle. Upon detection of an obstacle, the obstacle detection system will determine if the obstacle is avoidable or unavoidable. If the obstacle is avoidable, it will determine if the obstacle is moving or stationary. If the obstacle is stationary, it will determine if a speed change is necessary or not for the vehicle.

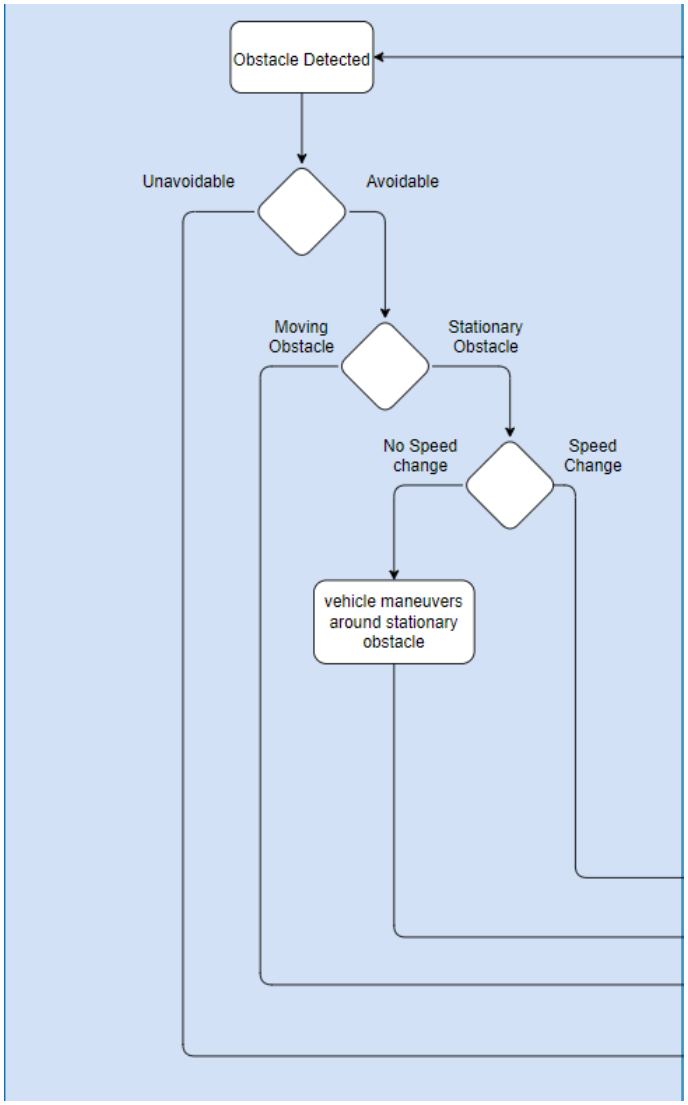

Figure 7.1.2 - Process View - Obstacle Detection System

7.1.3 Driver's Interface

The Driver's Interface will display the vehicle's current positioning and provide available routes based on that. It will also display the current status of the vehicle and instruct the user to take certain actions such as taking manual control of the wheel when automatic steering has ceased as shown in Figure 7.1.3.

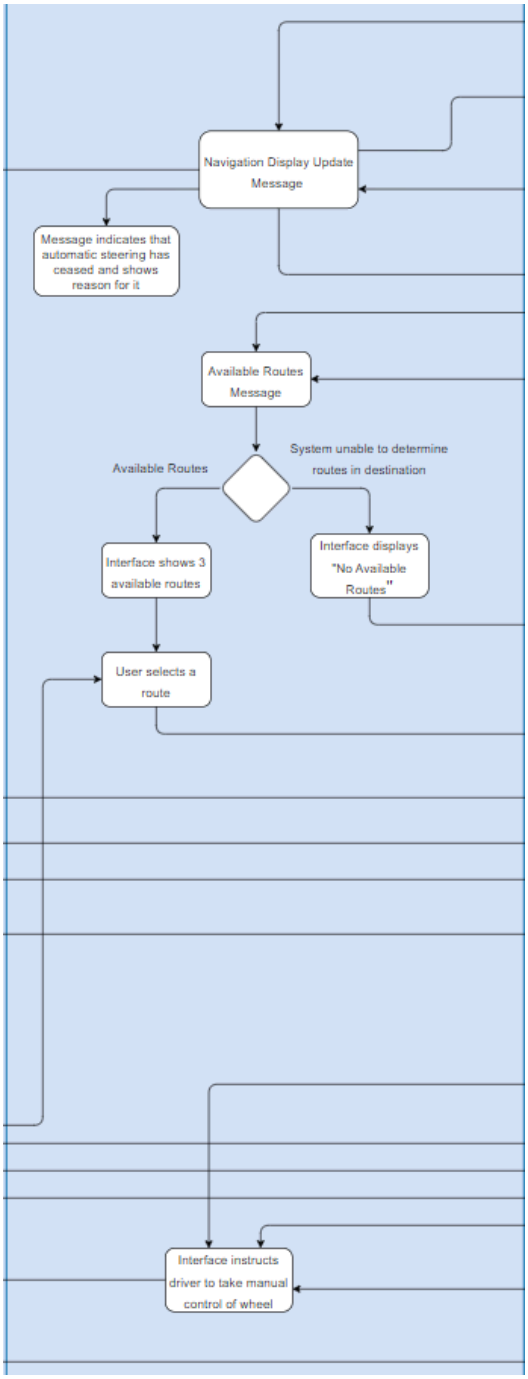

Figure 7.1.3 - Process View - Driver's Interface

7.1.4 Control Module

The Control Module will take in the processes that the other subsystems send to it and determine the states that the subsystems shall go to next as shown in Figure 7.1.4.

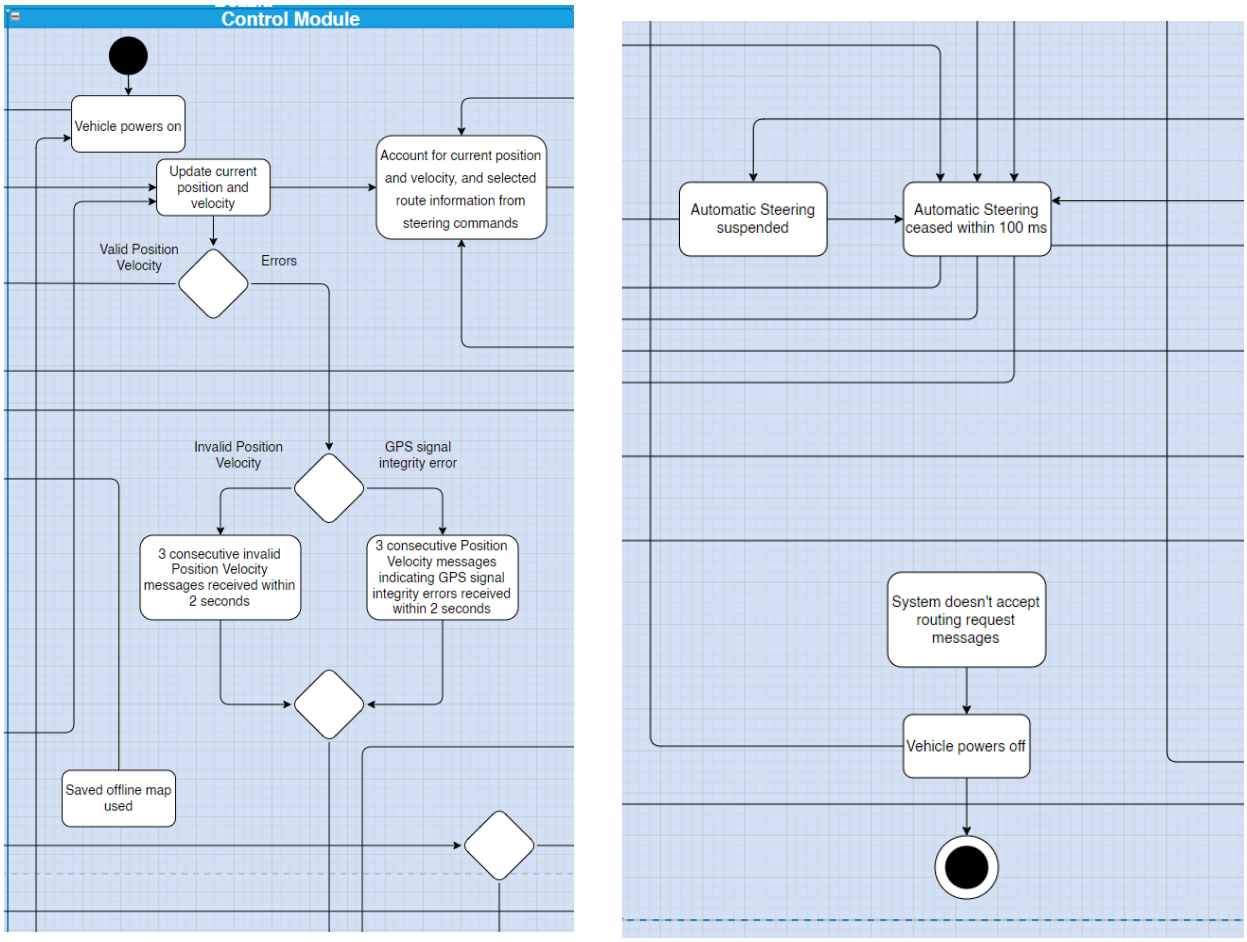

Figure 7.1.4 - Process View Control Module

7.1.5 Steering Actuators

The Steering Actuators process whether the driver is driving manually or using automatic steering. It also determines if the vehicle is steering on route as shown in Figure 7.1.5.

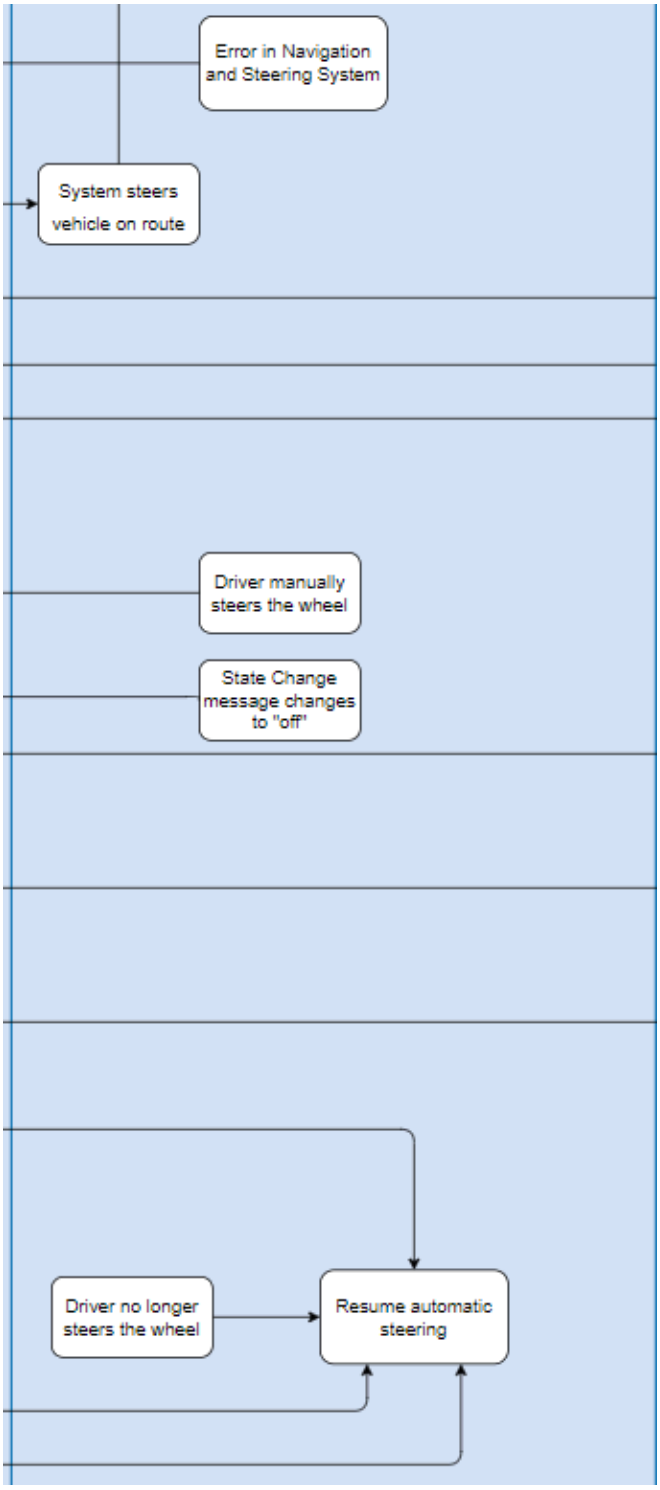

Figure 7.1.5 - Process View - Steering Actuators

7.1.6 Speed Control System

Upon receiving the vehicle's current position and velocity, the Speed Control System will determine the speed change necessary to keep the vehicle at a safe speed as shown in Figure 7.1.6.

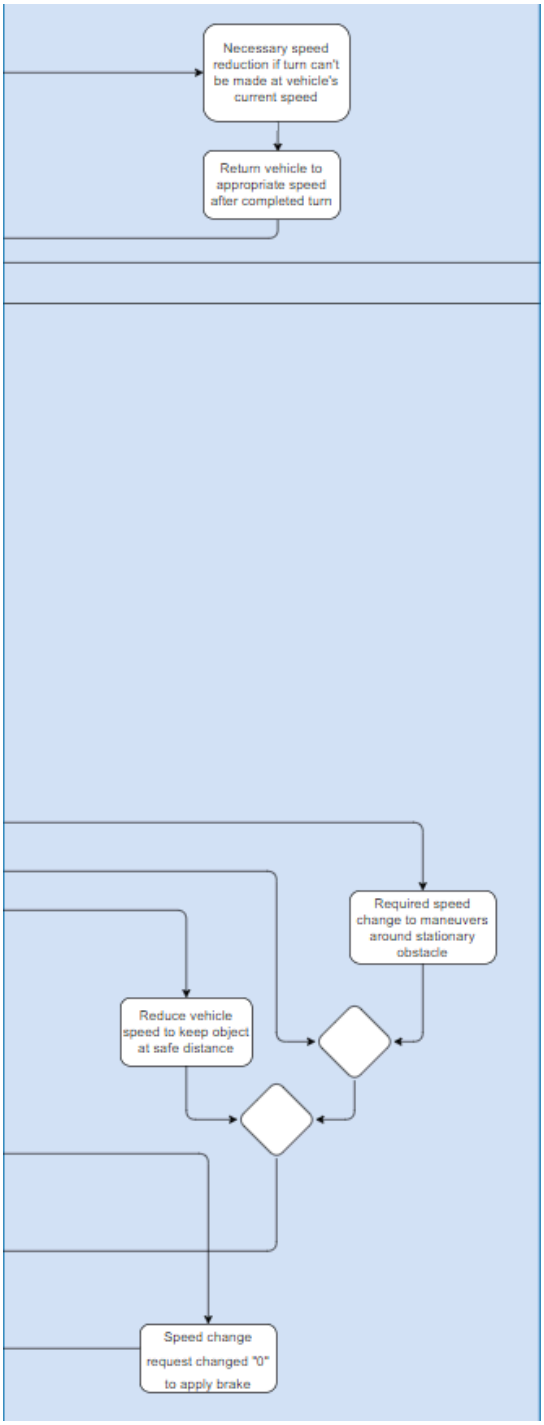

Figure 7.1.6 - Process View - Speed Control System

7.1.7 GPS Receiver

Based on the vehicle's position and velocity, the GPS Receiver will determine 3 available routes for the user to take as shown in Figure 7.1.7.

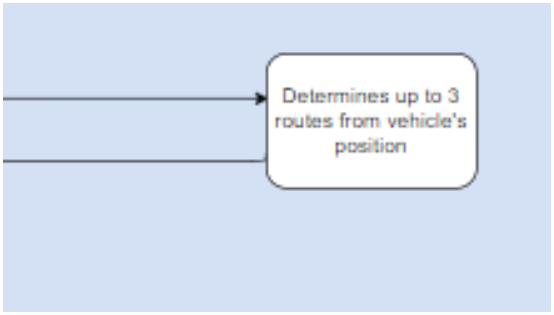

Figure 7.1.7 - Process View - GPS Receiver

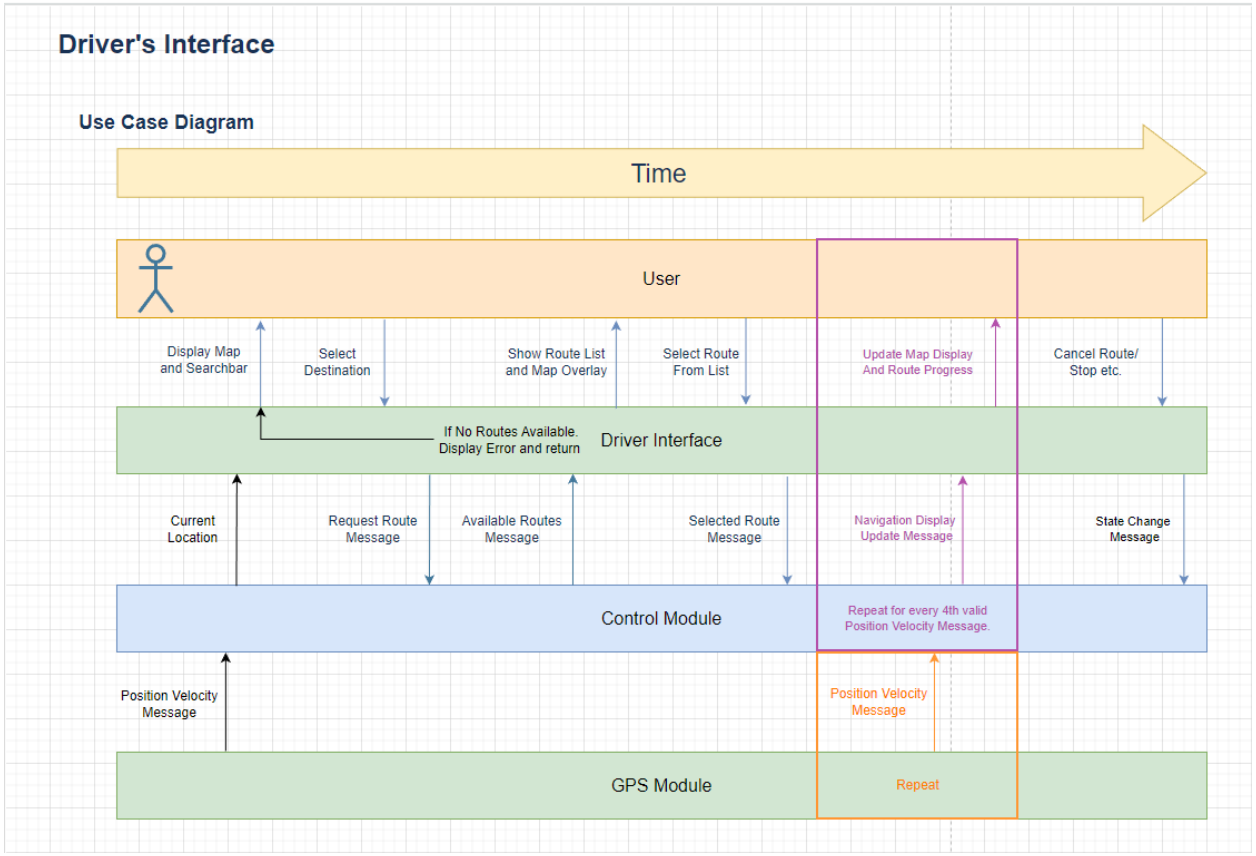

Figure 7.2 - Process View: DI - CM communication

This figure shows the timing of messages the Driver's Interface makes with the Control Module, and how this affects the state the user sees.

These modules work asynchronously, waiting for the messages to trigger callbacks without blocking the main CPU thread.

8. Deployment/Physical View

The Physical View is explained in the diagram which indicates the physical components through various nodes. These nodes showcase the mappings of processes to one another.

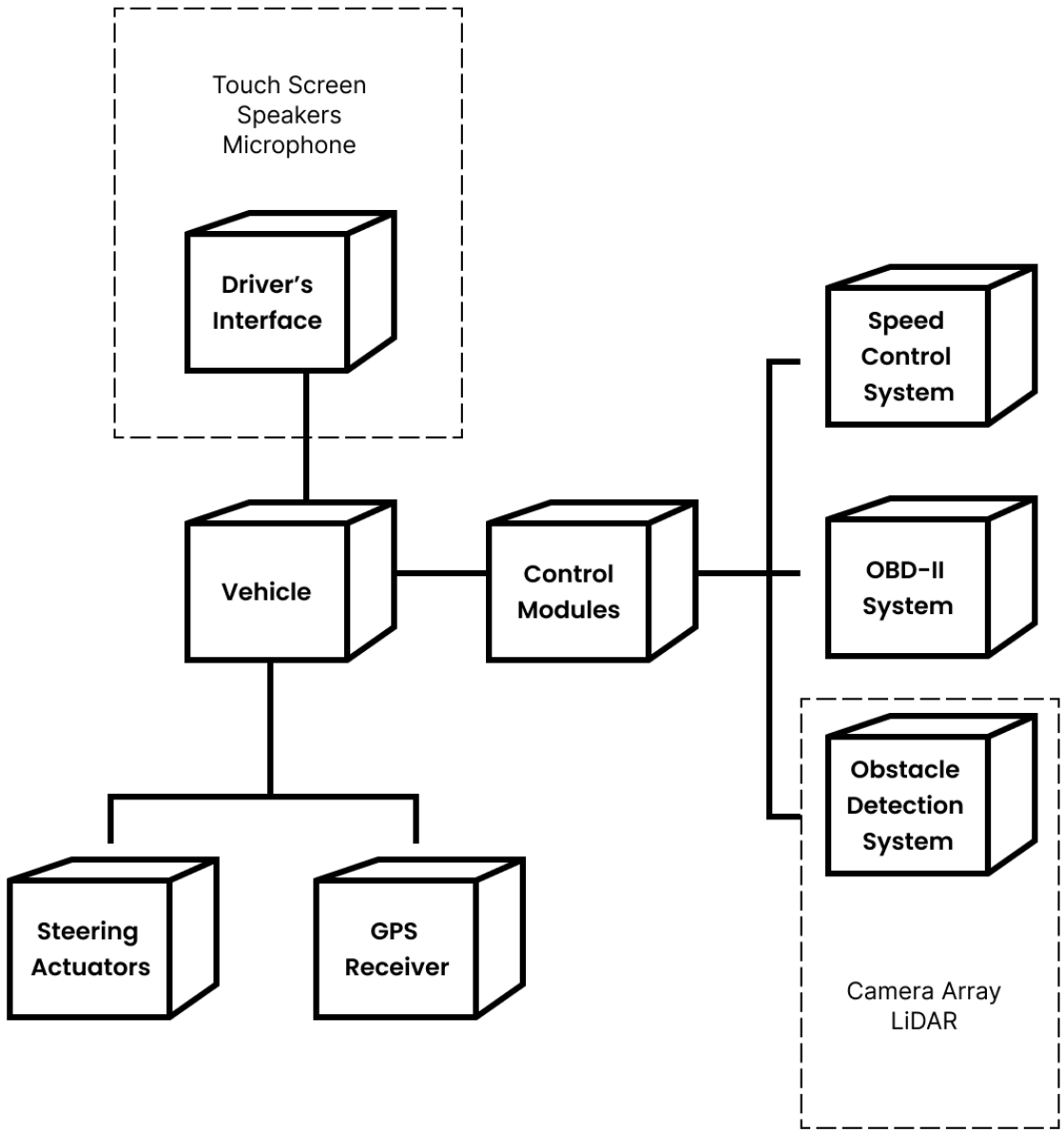

Figure 8.1 - Physical View Diagram

Driver's Interface: This display will show vehicle information and warning messages to the user. It also features a touch screen speakers microphone to allow users to utilize voice commands.

Steering Actuators: The device that aids the steering of a vehicle.

GPS Receiver: The device that collects data on geographic locations by receiving transmissions from GPS satellites.

Control Modules: The collection of an automotive component of sensors, actuators, and basic controls that work together to operate motor vehicles. It works in conjunction with the speed control system, OBD-II system, and obstacle detection system.

Speed Control System: The system that automatically controls the speed of a vehicle.

OBD-II System: The On-Board Diagnostic System is a California mandated component that tracks all components that affect emissions.

Obstacle Detection System: The sensors that sense slow-moving or stationary objects in front of a vehicle can provide a warning to the control module of impending collisions; can potentially brake or change the speed of the vehicle. The sensors work in conjunction with the camera array LiDAR.

9. Size and Performance

The chosen software architecture supports the key sizing and timing requirements:

1. The system shall support every vehicle manufactured with BEZZLA technology simultaneously with local servers & central databases. These local servers & central databases are located in two main buildings within Fullerton, CA, and Tampa, Florida. The servers and databases will be online 24/7 to support all the vehicles on the road throughout the United States. In the case that the servers are down, backup servers will be turned on immediately to ensure the safety of the customers. Servers responsible for navigation display and updates messages will be operable and provide current vehicle position 99.9999% of the time under usage.[SN-0026]

2. The system shall have extensively tested algorithms from reliable software engineers that determine optimal routes for navigating to a specific location. These optimal routes are dependent on the surrounding streets, highways, and toll roads the user is able to take. The algorithms will also take in updated/live data from Google Maps API where it recognizes streets and routes that are delayed, congested, have construction, or other variables that may cause disruption to the usual ETA. With the given API, the user is also able to register a preferred route as well as disable specific routes such as toll roads. In our product backlog, [SN-0027] states that the system shall be able to generate routes for 95% of the test cases to be supplied by the customer.

3. The systems' optimal routes will be calculated and displayed to the user in under 5 seconds through efficient algorithms processed through a Qualcomm Snapdragon CPU chip within each vehicle. The GPS receiver will utilize Google's route selection API and resources to quickly generate various optimal routes for destination navigation.[SN-005]

4. The Driver's Interface will respond to inputs for any of the features in less than 2 seconds. The Driver's Interface will utilize the operating system "Android Automotive" and is implemented with a reliable & very responsive tablet that will hold the infotainment system. This tablet has 6 GB RAM and utilizes a Qualcomm Snapdragon for fast touch screen and response. For safety reasons, if there is construction in the road or something wrong with the GPS, a message will be displayed saying, "Automatic Steering will be ceased due to objects in the road." This will be [SN-0023].

5. The Obstacle Detection System will notify the user of dangerous obstacles in less than a second for the user to respond. To detect such obstacles, the Obstacle Detection System is equipped with obstacle recognition software built within its high responsive cameras and sensors placed in the front and back of the vehicle. The obstacle recognition software determines an obstacle to be "dangerous" depending on its size and assumption for what kind of object it is. As soon as the recognition software recognizes the object within a few milliseconds, it will immediately send an alert to the instrument cluster, heads-up

display, and infotainment system about the incoming obstacle. Then, a Speed Change Request will occur to slow the vehicle to a recommended distance between the obstacles based on the US National Safety Council. [SN-0012]

6. The system will cease automatic steering within 100ms when transitioning to Manual Mode or when 3 consecutive invalid Position Velocity messages are received within 2 seconds. The system receives data from the speed control system where it considers the scenario where if the position velocity is different from that of actual speed taken from the ECU, the system will switch to manual as a precaution. [SN-004]

10. Quality

The software architecture supports the quality requirements.

1. The entire Control System & Driver Interface will be supported by Android OS, a linux-based operating system.

2. The BEZZLA technology will be usable immediately after starting the car.

3. BANDS will be closed-source software and cannot be modified by the user or hardware tampering and/or hacking with ECU components.

4. BANDS will give the user the option of disclosing navigation information of the routes the user take or sending that information to the company for diagnostics.

5. BANDS will provide clear and full detail of errors and bugs with the option to send the data to the company for diagnostics.

6. BANDS will download new software updates while charging to ensure maximum availability.

7. The system software will have maximized portability infrastructure for overall compatibility with future modeled vehicles for streamlined performance and convenience.

Appendix A: Architectural Design Principles

1. **Portability** for future car models.
2. **Scalability** to easily implement change.
3. High **availability** to ensure system reliability.
4. **Programmability** to comply with international standards/laws.
5. **Reliability** to avoid catastrophic failures.
6. **Reuse** to reduce budget and time-to-market
7. **Modularity** to allow for interoperability and updatability
8. **Buy rather than build** for solved problems with existing COTS solutions