Computer on Wheels

By:

Bilal Rafiq
27661
Hamza Azhar
28595
Sardar Mohsin Saghir
28016
M. Usama Nazir
30445

Supervised by:
Dr. Naveed Ikram
Dr. Rizwan Bin Faiz

Faculty of Computing
Riphah International University, Islamabad

Fall 2024

A Dissertation Submitted To

Faculty of Computing,

Riphah International University, Islamabad

As a Partial Fulfilment of the Requirement for the Award of the Degree of

Bachelors of Science in Software Engineering

Faculty of Computing
Riphah International University, Islamabad

Date: 19th November, 2024

Final Approval

This is to certify that we have read the report submitted by ***Bilal Rafiq (27661), Hamza Azhar (28595), Sardar Mohsin Saghir (28016) and M. Usama Nazir (30445),*** for the partial fulfilment of the requirements for the degree of the Bachelors of Science in Software Engineering (BSSE). It is our judgment that this report is of sufficient standard to warrant its acceptance by Riphah International University, Islamabad for the degree of Bachelors of Science in Software Engineering (BSSE).

Committee:

1 __________________________________

Dr. Naveed Ikram
(Supervisor)

2 __________________________________

Dr. Musharraf Ahmed
(Head of Department/chairman)

Declaration

We hereby declare that this document "**Computer on Wheels**" neither as a whole nor as a part has been copied out from any source. It is further declared that we have done this project with the accompanied report entirely on the basis of our personal efforts, under the proficient guidance of our teachers, especially our supervisors **Dr. Naveed Ikram** and **Dr. Rizwan Bin Faiz**. If any part of the system is proved to be copied out from any source or found to be the reproduction of any project from anywhere else, we shall stand by the consequences.

Bilal Rafiq
26771

Hamza Azhar
28595

Sardar Mohsin Saghir
28016

M. Usama Nazir
30445

Dedication

We dedicate this project to Allah Almighty our creator, our strong pillar, our source of inspiration, wisdom, knowledge and understanding. He has been the source of our strength throughout this program. Also, we dedicate our work to our family, friends and teachers. The unrivalled encouragement from our parents and outstanding support from teachers is what led to the success of this project. We also dedicate our work to our supervisors **Dr. Naveed Ikram, Dr. Rizwan Bin Faiz, Maanz AI** for their guidance and support and the faculty members.

Acknowledgement

Firstly, we are obliged to Allah Almighty the Merciful, the Beneficent, and the source of all Knowledge, for granting us the courage and knowledge to complete this Project.

We are grateful to our supervisors **Dr. Naveed Ikram and Dr. Rizwan Bin Faiz** for their enthusiasm, patience, insightful comments, helpful information, practical advice, and unceasing ideas that always helped us tremendously in our project. Without their support and guidance, this project would not have been possible. Also, a special thanks to **Maanz AI**, the faculty members, and **Mr. M. Tahir Anis**, an alumnus, for their invaluable support.

Bilal Rafiq
26771

Hamza Azhar
28595

Sardar Mohsin Saghir
28016

M. Usama Nazir
30445

Abstract

The emergence of Autonomous Vehicles (AVs) promises to revolutionize transportation by enhancing safety and efficiency. However, challenges such as human-error accidents and productivity loss during travel remain significant. This project aims to address these challenges by developing an **embedded software system for an autonomous vehicle designed for private use**, focusing on user-selected destinations. The system incorporates key functionalities, including **path following, path planning, obstacle detection and avoidance, and traffic light management.**

Utilizing machine learning for obstacle detection, the system enhances the vehicle's capabilities to navigate urban environments with precision and safety. Leveraging the **CARLA simulator for realistic vehicle simulations and ROS Noetic for robotic system** development, this project offers a novel approach to self-driving technology. The advancements in AVs by companies such as Tesla, Waymo, and Uber are paving the way for a future of transportation that promises increased global efficiency, safety, and security, which this project aims to contribute to through a robust software solution.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Chapter 1: Introduction

Computer on Wheels is an **embedded software system designed for personal vehicles** that can autonomously navigate with minimal human intervention. The system controls the vehicle's movement capabilities, including throttle, acceleration, braking, and steering. It incorporates obstacle detection mechanisms to detect and respond to obstacles, ensuring safe navigation. Additionally, path planning algorithms are utilized to calculate optimal routes from a **user-specified** starting point to a destination. By leveraging cutting-edge technologies such as the **CARLA (Car Learning to Act) simulator**, the **CARLA-ROS bridge**, and **ROS (Robot Operating System)**, this project aims to create a robust **embedded software solution that empowers personal vehicles** to navigate urban environments confidently.

1.1 Opportunity and Stakeholder

- According to a **National Highway Traffic Safety Administration (NHTSA)** study, driver error led to **94% of the crashes** examined.
- According to the **U.S. General Services Administration (GSA)**, human error causes **98% of crashes**.
- A 2017 study by **RAND Corporation** found that self-driving cars could reduce traffic fatalities by up to **25% by 2040**.
- A 2019 study by the National Highway Traffic Safety Administration (NHTSA) found that self-driving cars were involved in fewer crashes than human-driven cars per mile driven.
- A 2020 study by the **Massachusetts Institute of Technology (MIT)** found that self-driving cars could **prevent up to 90%** of crashes caused by human error.

1.1.1 Stakeholders

- Driver
- Passengers
- Vehicle Owner

1.2 Motivations and Challenges

Our project is motivated by the importance of enhancing safety for passengers, drivers, and pedestrians through autonomous vehicle technology. By alleviating the need for human drivers, we aim to enable multitasking and provide independence to individuals, including those with disabilities. Challenges such as time management and acquiring a physical model car for demonstrations were overcome by transitioning to the CARLA simulator. However, **GPU resource limitations were encountered**, which were addressed through assistance from **Maanz AI**, securing workspace and expert guidance.

1.3 Goals and Objectives

Our goals are clear: complete the project on time while ensuring high-quality deliverables and develop autonomous vehicle software to eliminate accidents caused by human error and enhance mobility for individuals with disabilities. These objectives will minimize errors, boost stakeholder productivity, and provide mobility for aged persons and people having disabilities.

1.4 Solution Overview

Our solution involves developing an embedded software system for autonomous vehicles that utilizes advanced technologies, including the CARLA simulator, ROS Noetic, the CARLA-ROS bridge, and rospy. This software enables vehicles to navigate complex environments autonomously by implementing the following key functionalities:

- **Path Planning**: Determining optimal routes from user-specified locations.
- **Path Following**: Ensuring precise vehicle navigation along the planned trajectory.
- **Obstacle Detection**: Detecting objects in the vehicle's surroundings using sensor data.

- **Obstacle Avoidance**: Executing dynamic maneuvers to circumvent detected obstacles safely.
- **Traffic Light Detection and Response**: Identifying traffic lights using sensor data, recognizing their states (red, yellow, green), and making informed decisions about vehicle behavior based on these signals.

By focusing on safety and precision, the solution aims to minimize accidents caused by human error. Through rigorous development and extensive testing, we aspire to deliver a reliable and efficient solution that revolutionizes autonomous vehicle navigation.

1.4.1 Project Scope

The scope of this project encompasses the development and implementation of key functionalities:

1.4.1.1 Integration:

- Involve integrating various sensors and algorithms to enable the vehicle to perceive its environment accurately, make decisions, and navigate safely through dynamic scenarios.

1.4.1.2 Path Planning:

- Determining a feasible and shortest path from user-specified source and destination locations
- Implementing a navigation algorithm to handle dynamic environments and potential rerouting.

1.4.1.3 Path Following:

- Implementing control algorithms for precise vehicle guidance along the planned trajectory.
- Maintaining vehicle position and orientation relative to the path using steering, acceleration, and braking control.

1.4.1.4 Obstacle Detection:

- Utilizing sensor data (such as lidar, radar or cameras) to detect objects within the vehicle's surroundings.
- Providing real-time information about detected obstacles to inform path planning and navigation decisions.

1.4.1.5 Obstacle avoidance:

- Implement reactive obstacle avoidance strategies, allowing the autonomous vehicle to dynamically adjust its trajectory based on the detected obstacles, enabling safe navigation.
- Implement algorithms/maneuver for real-time analysis of obstacle data to facilitate swift decision-making by the autonomous vehicle.

1.4.1.6 Traffic Light Detection and Response:

- Utilizing sensor data to detect the presence of traffic lights at intersections.
- Recognizing the state of traffic lights (red, yellow, green) and adjusting vehicle behavior accordingly, including stopping, slowing down, or proceeding through intersections safely.

1.5 Report Outline

This report covers all aspects of the Computer on Wheels, for understanding and clarity. This report has been divided into six chapters.

Chapter 1 serves as an introduction to our software system, encapsulating the project's opportunities, stakeholders, motivations, challenges, goals, objectives, and the proposed solution.

Chapter 2 undertakes a thorough examination of existing literature pertaining to autonomous vehicles, alongside an analysis of companies operating within this domain.

Chapter 3 outlines the essential requirements that serve as the foundation for guiding the development process and ensuring that the system meets the needs and expectations of stakeholders and end-users.

Chapter 4 comprehensively covers the design factors of the developed system, focusing on system architecture design considerations and various diagrams modelling the working behaviour of the system.

Chapter 5 includes the implementation process of our project, outlining the steps taken to achieve our goals and the integration of technologies and methodologies to ensure the successful development of our project.

Chapter 6 focus on system testing, specifically employing Black Box Testing techniques to validate the functionality and performance of the autonomous vehicle system. Black Box Testing, as a method, assesses the system's inputs and outputs without delving into the internal code structure.

Chapter 7 includes the conclusion of our project, along with a brief outlook

Chapter 2: Literature Review

Chapter 2: Literature Review

This chapter provides an overview of the current state of autonomous vehicles (AVs), including existing developments, ongoing testing, and prominent market participants. It explores the origins of autonomous vehicles and the regulatory bodies responsible for establishing rules governing their deployment.

2.1 Introduction

The concept of autonomous vehicles is well-established in the automotive industry. Companies such as Tesla, General Motors, BMW, Mercedes, Honda, KIA, and Toyota have been actively engaged in the development of AV technologies. Many have equipped vehicles with Level 2 and Level 3 autonomous systems, although not all of these have been released to the market. The Society of Automotive Engineers (SAE) has established six levels of driving automation, ranging from Level 0 (fully manual) to Level 5 (fully autonomous), providing a framework for understanding the capabilities of these systems.

2.2 Literature Review

The concept of autonomous vehicles dates back to 1918, with early attempts emerging in the 1920s. General Motors was among the pioneers, showcasing autonomous vehicle concepts at exhibitions. Significant momentum in research and development came from initiatives such as the collaboration between General Motors and RCA Sarnoff Laboratory. The Defense Advanced Research Projects Agency (DARPA) Grand Challenges Program in 2004 further accelerated research in the US.

Today, the global autonomous vehicle market features key players including AB Volvo, BMW AG, Daimler AG, Ford Motor Company, General Motors, Honda Motor Co., Ltd., Nissan Motors Co., Ltd., Tesla, Inc., Toyota Motor Corporation, and Volkswagen AG.

- **AB Volvo**: Began developing autonomous vehicles in 2006 and unveiled a fully autonomous test vehicle in 2017, though commercial availability is still pending.
- **Waymo (Google's subsidiary)**: Has logged millions of autonomous driving miles and currently offers limited commercial self-driving ride-hailing services in select locations.

- **Tesla**: Announced plans for self-driving features in their cars in 2014. However, Tesla's Autopilot is a driver-assistance system rather than fully autonomous and has faced safety criticisms.

2.2.1 Levels of Autonomous Vehicles

Understanding the different levels of autonomy set by the Society of Automotive Engineers (SAE) International is crucial before discussing existing autonomous vehicle systems. These levels explain how much control the vehicle has versus the human. The table below shows these levels, from full human control to full automation, making it easier to understand the capabilities of existing systems.

Table 2.1: Levels of taxonomy

Levels of Taxonomy	Description
Level 0 No automation	Zero autonomy; the driver performs all driving tasks.
Level 1 Driver assistance	The vehicle is controlled by the driver but driving assist features may be included in the vehicle design.
Level 2 Partial automation	Vehicles have combined automated functions, like acceleration and steering, but the driver must remain engaged with the driving task and always monitor the environment.
Level 3 Conditional automation	A driver is a necessity but is not required to monitor the environment. The driver must be ready to always take control of the vehicle with notice.
Level 4 High automation	The vehicle can perform all driving functions under certain conditions. The driver may have the option to control the vehicle.
Level 5 Full automation	The vehicle can perform all driving functions under all conditions.

2.3 Existing Level 2 Systems

Table 2.2: Existing Systems

Feature	Existing Level 2 Systems (e.g., Tesla Autopilot, GM Super Cruise)
Obstacle Detection & Avoidance	Uses **multi-sensor fusion** for advanced obstacle handling ultimately making it more **costly and complex** yet **not safe enough in adverse weather or to lose attention completely**.
Traffic Light Detection	Some systems, like Tesla, detect traffic lights but **rely heavily on driver verification**
Driver Supervision	Driver **must remain alert**; systems often include driver monitoring cameras

Currently, the automotive market provides vehicles with Levels 0, 1, and 2 of automation. Levels 3, 4, and 5 are still in the **testing phase** and not widely available for commercial use.

2.4 Summary

This chapter analyzes the current landscape of autonomous vehicles (AVs). While various companies are actively developing AV technology, commercially available vehicles primarily offer Levels 0 (no automation), 1 (driver assistance features), and 2 (partial automation) of driving autonomy as defined by the Society of Automotive Engineers (SAE). Levels 3 (conditional automation), 4 (high automation), and 5 (full automation) remain under development and testing.

Chapter 3:
Requirement Analysis

Chapter 3: Requirement Analysis

3.1 Introduction

In this chapter we will discuss the requirements of our project “Computer on Wheels”. Prior to that, we will discuss all the problem statements we have found while doing research on the project idea. These requirements are gathered using a variety of techniques, including **interviewing domain experts** and **conducting documentation analysis**. Our approach involves reviewing **existing documentation, research papers, industry standards, and guidelines** related to autonomous vehicle navigation.

Positioning: This system is designed as a **cost-efficient, driver-assistance solution within the Level 2 autonomy standard**, **not as a competitor** to more advanced autonomous systems. Instead, it focuses on delivering essential automation functions while maintaining affordability and simplicity.

3.2 Problem Scenarios

Table 3.1: problem statement 1

Problem Statement # 1: Safety Challenges Within Level 2	
The problem of	Inadequate safety measures for autonomous navigation in adverse weather conditions within level 2.
Affects	Passengers, pedestrians and other road users.
The result of which	Increased risk of accidents due to reduced visibility, leading to injuries or fatalities.
Benefits of	Improved safety protocols to ensure reliable operation of autonomous vehicles in varying environmental conditions.

Table 3.2: problem statement 2

Problem Statement # 2: Challenges in Simulation for AV Software Development	
The problem of	Unpredictable Impacts of Integrating CARLA with ROS for Autonomous Vehicle Development
Affects	Developers and researchers in autonomous vehicle systems.
The result of which	Slower development cycles, higher costs of physical testing, and potential safety risks due to insufficient validation.
Benefits of	Enhanced efficiency and safety through thorough evaluation of autonomous algorithms under varied conditions before deployment.

Table 3.3: problem statement 3

Problem Statement # 3: User Acceptance Challenges for Autonomous Vehicles.	
The problem of	Limited user acceptance of autonomous vehicle technology due to perceived safety and reliability concerns.
Affects	Potential users, stakeholders, and the overall adoption of autonomous vehicles.
The result of which	Resistance to adopting autonomous vehicles, leading to slower market penetration and reduced investment in further development and innovation.
Benefits of	Building user confidence through improved transparency, education, and demonstrations of safety features in various conditions, which can enhance the acceptance and integration of autonomous vehicles in everyday life.

3.3 Key Concepts and Terminology

To understand the requirements outlined in this chapter, it is important to be familiar with certain key concepts related to the **Robot Operating System (ROS)**, which serves as the framework for our embedded system.

3.3.1 ROS Overview

The Robot Operating System (ROS) is a middleware framework that is essential for managing complex data exchanges in autonomous systems like our **"Computer on Wheels."** It provides tools for creating **modular software components**, called **nodes**, that can communicate over defined channels known as **topics**.

3.3.2 ROS Nodes

Nodes are software modules that perform specific tasks. For example, a sensor node may detect obstacles, while a control node manages vehicle movement.

3.3.3 ROS Topics

Topics are the communication channels between nodes. Each topic is defined for a specific type of data exchange, such as publishing sensor readings or receiving control commands.

3.3.4 ROS Messages

Messages are the data structures used to communicate information over ROS topics. Each message type has a defined format and is used to transmit specific types of data, such as position coordinates or sensor measurements.

3.3.5 ROS Services

Services provide a way for nodes to request specific actions or information from each other, such as recalculating a path when an obstacle is detected.

This foundational understanding of ROS will facilitate comprehension of the requirements detailed in the subsequent sections.

3.4 Functional Requirements

3.4.1 Vehicle Control: Manages steering, throttle, and braking to execute driving commands.

Table 3.4: FR1

No	Functional Requirement	Breakdown		Description
		ID	Sub-Functionality	
1	**Vehicle Control**	**1.1**	Autonomous Navigation	The system shall be capable of autonomously navigating from a starting point to a destination using ROS-based navigation stacks.
1	**Vehicle Control**	**1.2**	Acceleration Control	The system shall control the vehicle's acceleration to maintain desired speeds along the planned trajectory, publishing commands to the topics in ROS.
1	**Vehicle Control**	**1.3**	Emergency Stop	The system shall include a mechanism for the driver to perform an immediate emergency stop, halting all vehicle operations by publishing to the dedicated ROS topic (`/emergency_stop`)
1	**Vehicle Control**	**1.4**	Throttle Control	The system shall control the throttle to regulate vehicle speed within a range of 0 to 120 km/h, adjusting for road conditions and traffic regulations, using PID controller implemented in ROS.

1	**Vehicle Control**	**1.5**	Steering Control	The system shall control the vehicle's steering to maintain a maximum lateral deviation of 0.5 meters from the planned trajectory under normal conditions, using ROS control messages.
1	**Vehicle Control**	**1.6**	Braking Control	The system shall control the vehicle's braking to safely decelerate and stop as required by the planned trajectory, publishing braking commands to ROS topic.

3.4.2 Path Planning: Determines the optimal route from the current location to the destination.

Table 3.5: FR2

No	Functional Requirement	Breakdown		Description
		ID	Sub-Functionality	
2	**Path Planning**	**2.1**	Route Calculation	The system shall calculate the most efficient route i.e. shortest path from the vehicle's current location to the driver-specified destination using ROS-based algorithms.
2	**Path Planning**	**2.2**	Lane Assignment	The system shall assign appropriate lanes for the vehicle to travel in along the calculated route, based on legal navigation rule and map data.

2	**Path Planning**	**2.3**	Waypoint Generation	The system shall generate waypoints along the calculated route to guide the vehicle towards the destination, publishing waypoints to a ROS topic (`/waypoints`).
2	**Path Planning**	**2.4**	Dynamic Obstacle Avoidance	The system shall adapt the vehicle's path in real-time to safely avoid unexpected obstacles using ROS-based path adjustment algorithms.
2	**Path Planning**	**2.5**	Map Reading	The system shall be able to read and interpret digital map data using ROS to determine the vehicle's precise location within the road network

3.4.3 Path Following: Ensures the vehicle adheres to the planned path using control algorithms.

Table 3.6: FR3

No	**Functional Requirement**	**Breakdown**		**Description**
		ID	**Sub-Functionality**	
3	**Path Following**	**3.1**	Path smoothing	The system shall apply path smoothing techniques to limit acceleration changes to within 0.3 m/s^2, ensuring a smooth ride for passengers.
3	**Path Following**	**3.2**	Lateral Control	The system shall maintain a lateral deviation of no more than 0.5 meters from the planned path under normal

				driving conditions using ROS control loops.
3	**Path Following**	**3.3**	Longitudinal Control	The system shall maintain a longitudinal deviation of no more than 1 meter from the planned path under normal driving conditions.
3	**Path Following**	**3.4**	Speed Control	The system shall control the speed to reach the destination.
3	**Path Following**	**3.5**	Waypoint Following	The system shall follow waypoints along the calculated route, using ROS topics to track progress towards each waypoint.

3.4.4 Sensor Integration: Combines data from multiple sensors for environment perception.

Table 3.7: FR4

No	**Functional Requirement**	**Breakdown**		**Description**
		ID	**Sub-Functionality**	
4	**Sensor Integration**	**4.1**	Inertial Measurement Unit Utilization	The system shall use an IMU to provide orientation and acceleration data at a frequency of 100 Hz, publishing data to ROS topics.
4	**Sensor Integration**	**4.2**	Global Positioning System Utilization	The system shall use GPS to determine the vehicle's position and publish coordinates to a ROS topic (`/gps_data`).

4	**Sensor Integration**	**4.3**	Radar/Lidar Utilization	The system shall utilize radar/lidar sensors to provide information about surrounding objects' velocity and distance, enhancing situational awareness through ROS topics

3.4.5 Trajectory Planning: Generates a feasible sequence of movements for smooth navigation.

Table 3.8: FR5

No	**Functional Requirement**	**Breakdown**		**Description**
		ID	**Sub-Functionality**	
5	**Trajectory Planning**	**5.1**	Trajectory Generation	The system shall plan a smooth and optimal trajectory, based on destination specified by user.

3.4.6 Obstacle Detection: Identifies obstacles in the vehicle's path using sensors.

Table 3.9: FR6

No	**Functional Requirement**	**Breakdown**		**Description**
		ID	**Sub-Functionality**	
6	**Obstacle Detection**	**6.1**	Detection Using Sensors	The system shall utilize various sensors to detect obstacles in the vehicle's path, integrating data through ROS topics (`/obstacle_detection`).
6	**Obstacle Detection**	**6.2**	Environmental Awareness	The system shall maintain awareness of static and dynamic objects in the vehicle's vicinity, using ROS-based perception modules.

6	**Obstacle Detection**	**6.3**	Dynamic Obstacle Tracking	The system shall continuously track moving obstacles, updating their positions through ROS messages.
6	**Obstacle Detection**	**6.4**	Destination Estimation	The system shall calculate the distance to detected obstacles and publish this data to a ROS topic (`/distance_to_obstacle`).

3.4.7 Obstacle Avoidance: Executes maneuvers to safely bypass detected obstacles.

Table 3.10: FR7

No	**Functional Requirement**	**Breakdown**		**Description**
		ID	**Sub-Functionality**	
7	**Obstacle Avoidance**	**7.1**	Maneuver Execution	The system shall execute safe and efficient avoidance maneuvers to navigate around detected obstacles, using ROS-based planning and control.
7	**Obstacle Avoidance**	**7.2**	Steering Control	The system shall dynamically adjust steering angles to guide the vehicle away from obstacles, keeping it on its intended path using ROS.
7	**Obstacle Avoidance**	**7.3**	Re-Plan Path	The system shall re-plan the path once an obstacle is detected, updating the path through ROS services.
7	**Obstacle Avoidance**	**7.4**	Trajectory Adjustment	The system shall dynamically adjust the vehicle’s trajectory to avoid obstacles in a clear environment using ROS algorithms.

7	**Obstacle Avoidance**	**7.5**	Multi-Obstacle Handling	The system shall manage avoidance of multiple obstacles simultaneously through ROS-based coordination.

3.4.8 Destination Arrival: Confirms when the vehicle successfully reaches the intended location.

Table 3.11: FR8

No	**Functional Requirement**	**Breakdown**		**Description**
		ID	**Sub-Functionality**	
8	**Destination Arrival**	**8.1**	Destination Approach	The system shall approach the driver-specified destination with a positional accuracy of within 1 meter, following the calculated trajectory and waypoints using ROS.
8	**Destination Arrival**	**8.2**	Stop at Destination	The system shall bring the vehicle to a complete stop within 1 meter of the designated destination, ensuring deceleration rates do not exceed 2 m/s^2 for passenger safety and comfort.

3.4.9 User Inputs: Captures and processes user-selected destinations or commands.

Table 3.12: FR9

No	**Functional Requirement**	**Breakdown**		**Description**
		ID	**Sub-Functionality**	
9	**User Inputs**	**9.1**	Ride Initiation	The system shall allow the user to initiate the autonomous driving process through a

				terminal command, which will start the ROS nodes required for vehicle navigation, control, and sensor integration. The command shall initiate the entire process of path planning, path following, and obstacle detection, ensuring all necessary components are activated before vehicle motion begins.
9	**User Inputs**	**9.2**	Destination Setting	The user shall be able to input the desired destination, triggering the route planning process through ROS services.

3.4.10 System Integration: Links all subsystems to function cohesively as a unified system.

Table 3.13: FR10

No	Functional Requirement	Breakdown		Description
		ID	Sub-Functionality	
10	**System Integration**	**10.1**	ROS Integration	The system shall utilize the Robot Operating System (ROS) to facilitate communication and data exchange between different software components.
10	**System Integration**	**10.2**	Simulation Environment	Development and testing of the system shall be conducted in a simulated environment (e.g., CARLA simulator) for thorough validation before real-world deployment.

3.4.11 Traffic Light Module: Detects and interprets traffic light states to guide vehicle behavior.

Table 3.14: FR11

No	Functional Requirement	Breakdown		Description
		ID	Sub-Functionality	
11	**Traffic Light Module**	**11.1**	Traffic Light Detection	The system shall detect traffic lights in the vehicle's path using camera-based sensors and publish the detected traffic light information to a ROS topic (`/traffic_light_detection`).
11	**Traffic Light Module**	**11.2**	Traffic Light State Recognition	The system shall recognize the state of detected traffic lights (red, yellow, green) using image processing algorithms within a ROS node, publishing the identified state to a topic (`/traffic_light_state`).
11	**Traffic Light Module**	**11.3**	Decision-Making Based on Traffic Light State	The system shall adjust vehicle behavior (e.g., deceleration, stopping, or proceeding) based on the recognized traffic light state, using data from the `/traffic_light_state` topic.
11	**Traffic Light Module**	**11.4**	Red Light Handling	Upon detecting a red-light state, the system shall bring the vehicle to a complete stop at a safe distance i.e., 1 meter from the traffic light, ensuring smooth deceleration.

11	**Traffic Light Module**	**11.5**	Green Light Handling	Upon detecting a green light state, the system shall resume vehicle motion and proceed along the planned path.
11	**Traffic Light Module**	**11.6**	Yellow Light Handling	Upon detecting a yellow light state, the system shall determine whether it is safe to proceed based on vehicle speed and distance to the traffic light, either decelerating to a stop or proceeding through the intersection.
11	**Traffic Light Module**	**11.7**	Traffic Light State Uncertainty	If the system cannot detect a traffic light state for more than 2 seconds, it shall trigger a safe stop and log an error message to a ROS topic (`/traffic_light_error`).

3.5 Non-Functional Requirements

Table 3.15: NFR1

No	Non- Functional Requirement	Subfactor	Verification Metric	Target Value
1	**Safety Requirement** Ensure reliable object detection in adverse weather conditions to assure safety	**Hazard Protection** The system must detect and respond to hazards arising from adverse weather conditions, such as rain, fog, or snow, which may reduce visibility.	**Detection Accuracy:** Measure the percentage of correctly detected objects in various weather conditions. **Response Time:** Time taken to respond to detected hazards. **Test Cases:** Conduct tests in simulated environments with	**Detection Accuracy:** ≥ 90% **Response Time:** ≤ 2 seconds in 95% of cases

			varying weather scenarios (e.g., rain, fog, snow)	

Table 3.16: NFR2

No	Non- Functional Requirement	Subfactor	Verification Metric	Target Value
2	**Scalability Requirement** The ROS-based architecture shall support adding new sensors (e.g., radar, additional cameras) without significant changes to the core modules.	**System Expandability**	**Integration Time:** Measure the time taken to integrate a new sensor and update existing modules. **Compatibility Tests:** Perform tests to ensure new sensors can be added without affecting existing functionality. **Modularity Assessment:** Analyze the architectural design for dependencies that may hinder expansion.	**Integration Time:** ≤ 10 minutes

Table 3.17: NFR3

No	Non- Functional Requirement	Subfactor	Verification Metric
3.1	**Modularity Requirement** The system shall maintain a modular ROS node structure, separating perception, planning, and control into distinct nodes for ease of testing and modification.	**Software Architecture**	**Node Independence:** Verify that each node can be tested independently without affecting others. **Modification Time:** Measure the time required to modify or update a specific node.
3.2	**Modularity Requirement** Each ROS node shall handle a specific task (e.g., path planning, obstacle detection) and communicate through well-defined ROS topics.	**Task Separation**	**Message Latency:** Measure the time taken for messages to be published and received between nodes (≤ 100 ms). **Task Success Rate:** Evaluate the success rate of individual nodes in completing their specific tasks (≥ 95%).

3.6 SQA activity: Defect Identification: Inspection Thought Checklist

3.6.1 Throttle Control:

Original: The system shall control the throttle for regulation of vehicle speed.

Revised: The system shall control the throttle to regulate vehicle speed within a range of 0 to 120 km/h, adjusting for road conditions and traffic regulations.

Table 3.18: Inspection Table 1 For Throttle Control

Requirement	Check List Point	Defect
The system shall control the throttle for regulation of vehicle speed.	Verifiability: Is each requirement testable or verifiable?	The requirement lacks specifics on the range of speed control and conditions under which speed regulation should be adjusted.

3.6.2 Steering Control:

Original: The system shall control the vehicle's steering to follow the planned trajectory accurately.

Revised: The system shall control the vehicle's steering to maintain a maximum lateral deviation of 0.5 meters from the planned trajectory under normal conditions.

Table 3.19: Inspection Table 2 For Steering Control

Requirement	Check List Point	Defect
The system shall control the vehicle's steering to follow the planned trajectory accurately.	Clarity: Are the requirements stated clearly so there is only one interpretation?	The term "accurately" is vague and not quantifiable.

3.6.3 Route Calculation:

Original: The system shall calculate the most efficient route i.e. shortest path from the vehicle's current location to the driver-specified destination.

Revised: The system shall calculate the most efficient route i.e. shortest path from the vehicle's current location to the driver-specified destination.

Table 3.20: Inspection Table 3 For Route Calculation

Requirement	Check List Point	Defect
The system shall calculate the most efficient route from the vehicle's current location to the driver-specified destination.	Verifiability: Does each requirement use concrete terms and measurable quantities?	"Most efficient route" is not defined; efficiency could refer to time, distance, fuel consumption, etc.

3.6.4 Path Smoothing:

Original: The system shall apply path smoothing techniques to reduce jerkiness and ensure passenger comfort.

Revised: The system shall apply path smoothing techniques to limit acceleration changes to within 0.3 m/s^2, ensuring a smooth ride for passengers.

Table 3.21: Inspection Table 4 For Path Smoothing

Requirement	Check List Point	Defect
The system shall apply path smoothing techniques to reduce jerkiness and ensure passenger comfort.	Verifiability: Is each requirement testable or verifiable?	The requirement does not define what constitutes "jerkiness" or acceptable levels of passenger comfort.

3.6.5 Lateral Deviation:

Original: The system shall minimize the lateral deviation from the path.

Revised: The system shall maintain a lateral deviation of no more than 0.5 meters from the planned path under normal driving conditions.

Table 3.22: Inspection Table 5 For Lateral Deviation

Requirement	Check List Point	Defect
The system shall minimize the lateral deviation from the path.	Clarity: Are the requirements written in user language? Do the users think so?	"Minimize" is not quantified; specific acceptable deviation limits should be stated.

3.6.6 Longitudinal Deviation:

Original: The system shall minimize the Longitudinal deviation from the path.

Revised: The system shall maintain a longitudinal deviation of no more than 1 meter from the planned path under normal driving conditions.

Table 3.23: Inspection Table 6 For Longitudinal Deviation

Requirement	Check List Point	Defect
The system shall minimize the Longitudinal deviation from the path	Clarity: Are the requirements written in user language? Do the users think so?	Similar to lateral deviation, "minimize" is not quantified, and specific limits should be provided.

3.6.7 IMU Data Usage:

Original: The system shall use IMU to provide orientation and acceleration data at some frequency.

Revised: The system shall use an IMU to provide orientation and acceleration data at a frequency of 100 Hz.

Table 3.24: Inspection Table 7 For IMU Data Usage

Requirement	Check List Point	Defect
The system shall use IMU to provide orientation and acceleration data at some frequency.	Completeness: Are all the inputs to the system specified including their source, accuracy, range of values, and frequency?	"Some frequency" is vague and should be specified clearly.

3.6.8 Trajectory Planning:

Original: The system shall plan a smooth and optimal trajectory for the vehicle to follow based on the calculated route.

Revised: The system shall plan a smooth and optimal trajectory, based on destination specified by user.

Table 3.25: Inspection Table 8 For Trajectory Planning

Requirement	Check List Point	Defect
The system shall plan a smooth and optimal trajectory for the vehicle to follow based on the calculated route.	Verifiability: Is each requirement testable or verifiable?	"Optimal trajectory" needs to be defined more concretely, considering factors like time, energy consumption, etc.

3.6.9 Destination Approach:

Original: The system shall precisely approach the driver-specified destination by following the calculated trajectory and waypoints accurately.

Revised: The system shall approach the driver-specified destination with a positional accuracy of within 1 meter, following the calculated trajectory and waypoints precisely.

Table 3.26: Inspection Table 9 For Destination Approach

Requirement	Check List Point	Defect
The system shall precisely approach the driver-specified destination by following the calculated trajectory and waypoints accurately.	Clarity: Are the requirements stated clearly so there is only one interpretation?	The terms "precisely" and "accurately" are subjective and need quantifiable measures.

3.6.10 Stop at Destination:

Original: The system shall bring the vehicle to a complete stop upon reaching the designated destination, ensuring a smooth and safe arrival.

Revised: The system shall bring the vehicle to a complete stop within 1 meter of the designated destination, ensuring deceleration rates do not exceed 2 m/s^2 for passenger safety and comfort.

Table 3.27: Inspection Table 10 For Stop at Destination

Requirement	Check List Point	Defect
The system shall bring the vehicle to a complete stop upon reaching the designated destination,	Completeness: Does each function specify the data used in the function and	"Smooth and safe arrival" should be quantified in terms of deceleration rates or stopping distance.

ensuring a smooth and safe arrival.	data resulting from the function?	

Chapter 4:
System Design

Chapter 4: System Design

This chapter focuses on how we've designed our system. Design is based upon the requirements which are gathered using a variety of techniques, including interviewing domain experts and conducting documentation analysis. Our approach involves reviewing existing documentation, research papers, industry standards, and guidelines related to autonomous vehicle navigation. We won't dive into the visual parts of our software, but we'll explore how everything in the system works together

- We adopted a **layered architecture** to ensure modularity, scalability, and separation of concerns, making it easier to maintain and expand the system.
- A **use case diagram** was created to represent interactions between the user and the system, capturing key functionalities like destination selection and navigation.
- **State charts** were developed for:
    - The entire system to model high-level behavior (idle, navigation, obstacle handling, traffic light handling).
    - Individual modules (e.g., obstacle detection and traffic light modules) to detail specific operations.
- A **system sequence diagram (SSD)** illustrates the sequence of interactions between system components, from user input to navigation and control.

These design elements ensure clear, structured, and efficient system functionality aligned with the project goals.

4.1 Introduction

The software system leverages the architecture of **ROS 1**, with outcomes visualized using the **Carla Simulator**. To enable seamless communication between Carla and ROS Noetic, we utilize the **ROS bridge as an interface** for data retrieval and command transmission. This bridge serves as a critical intermediary, facilitating integration between ROS programs and non-ROS environments.

4.2 Architectural Design

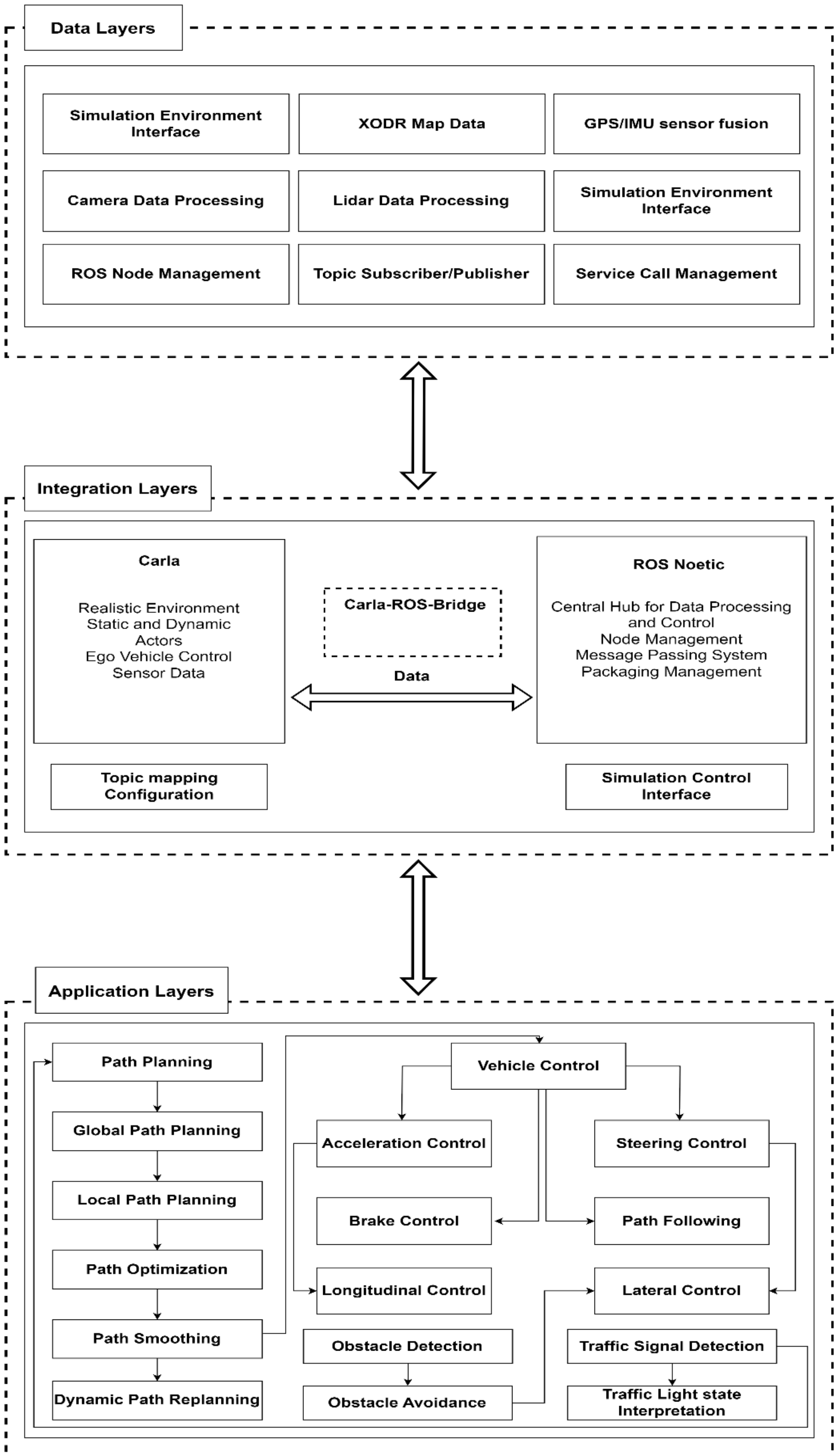

Figure 4.1: Architecture Diagram

4.3 Detailed Design

4.3.1 Use Case Design

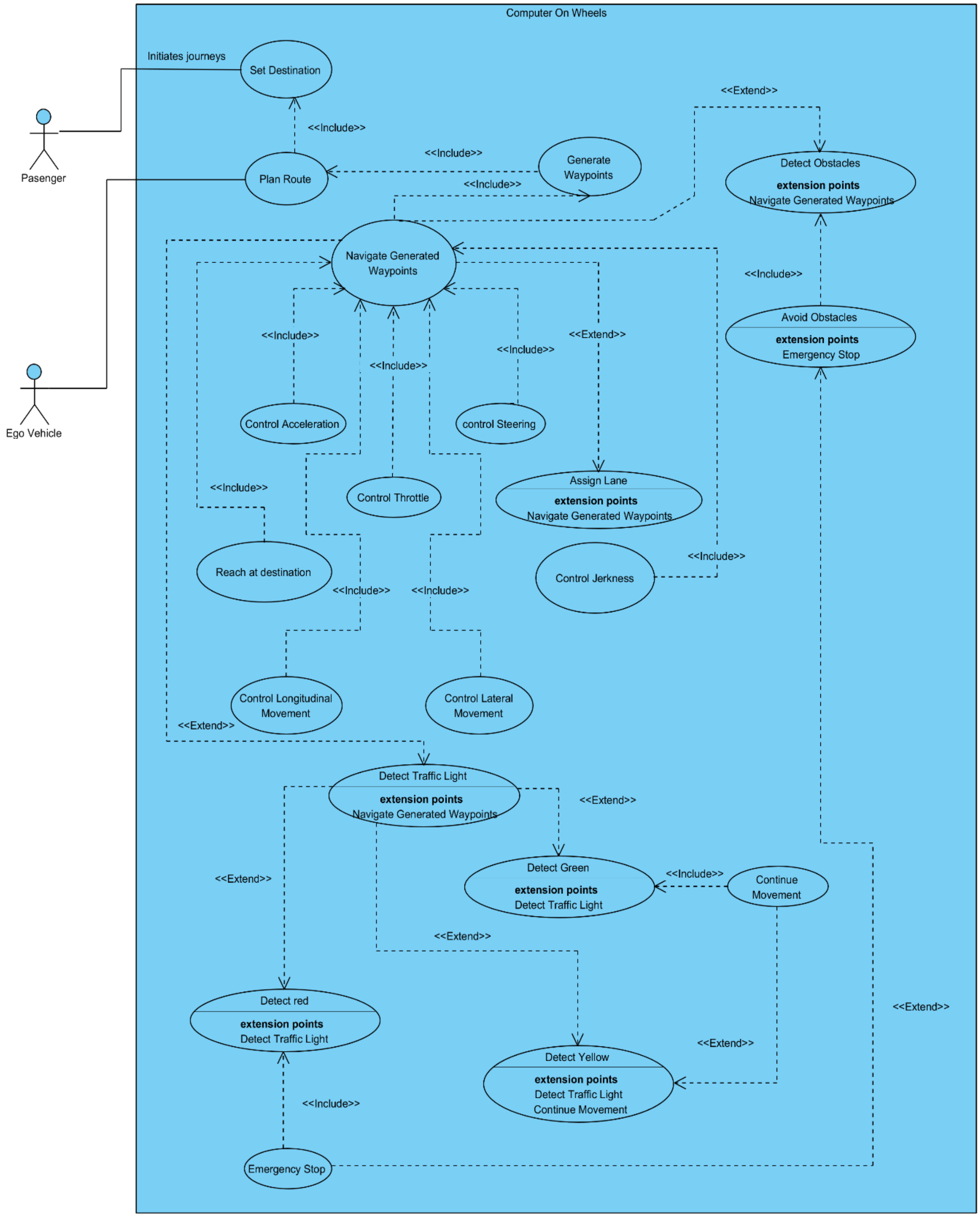

Figure 4.2: Use-case Diagram

4.3.2 Sequence Diagram

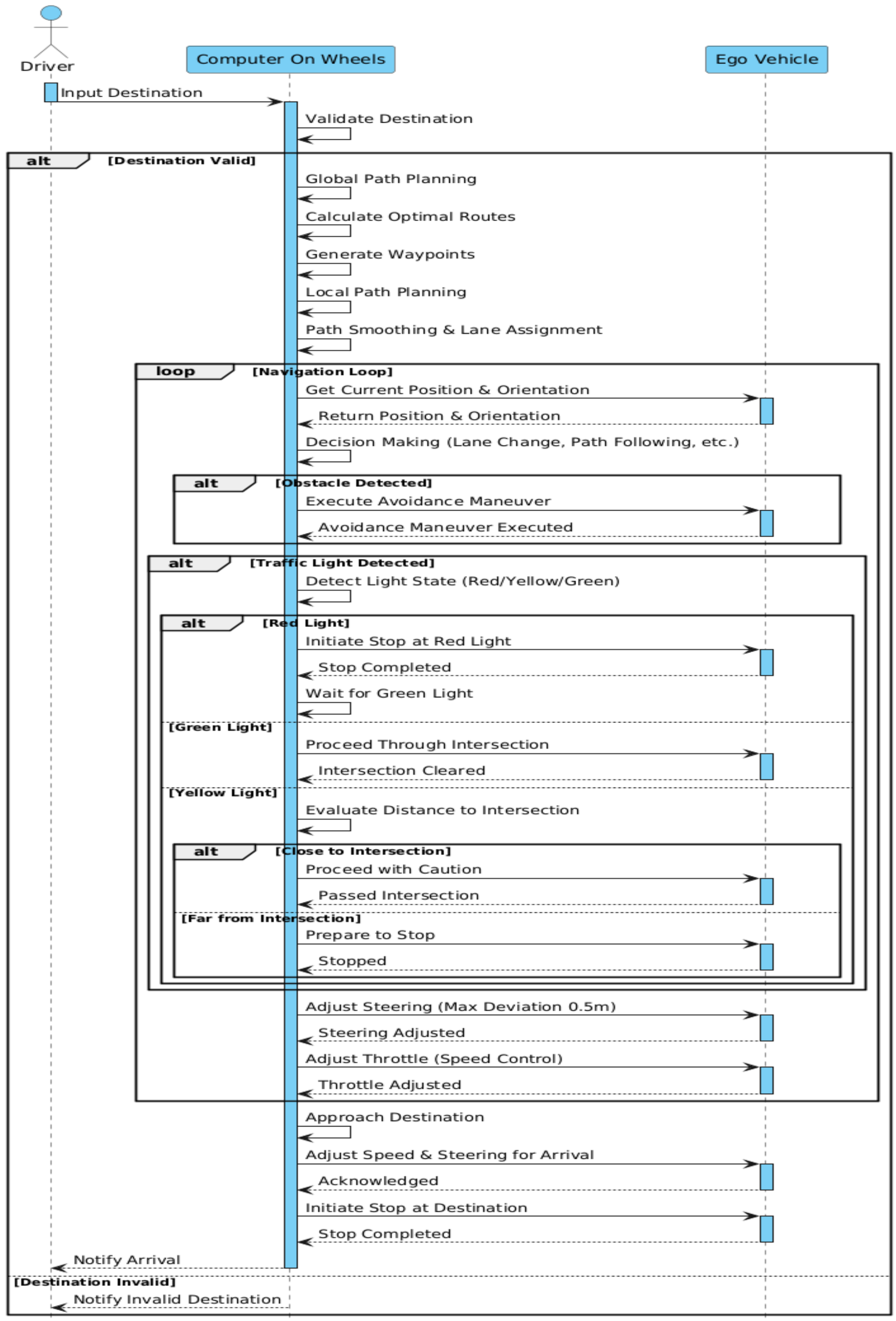

Figure 4.3: Sequence diagram

4.3.3 System State chart Diagram

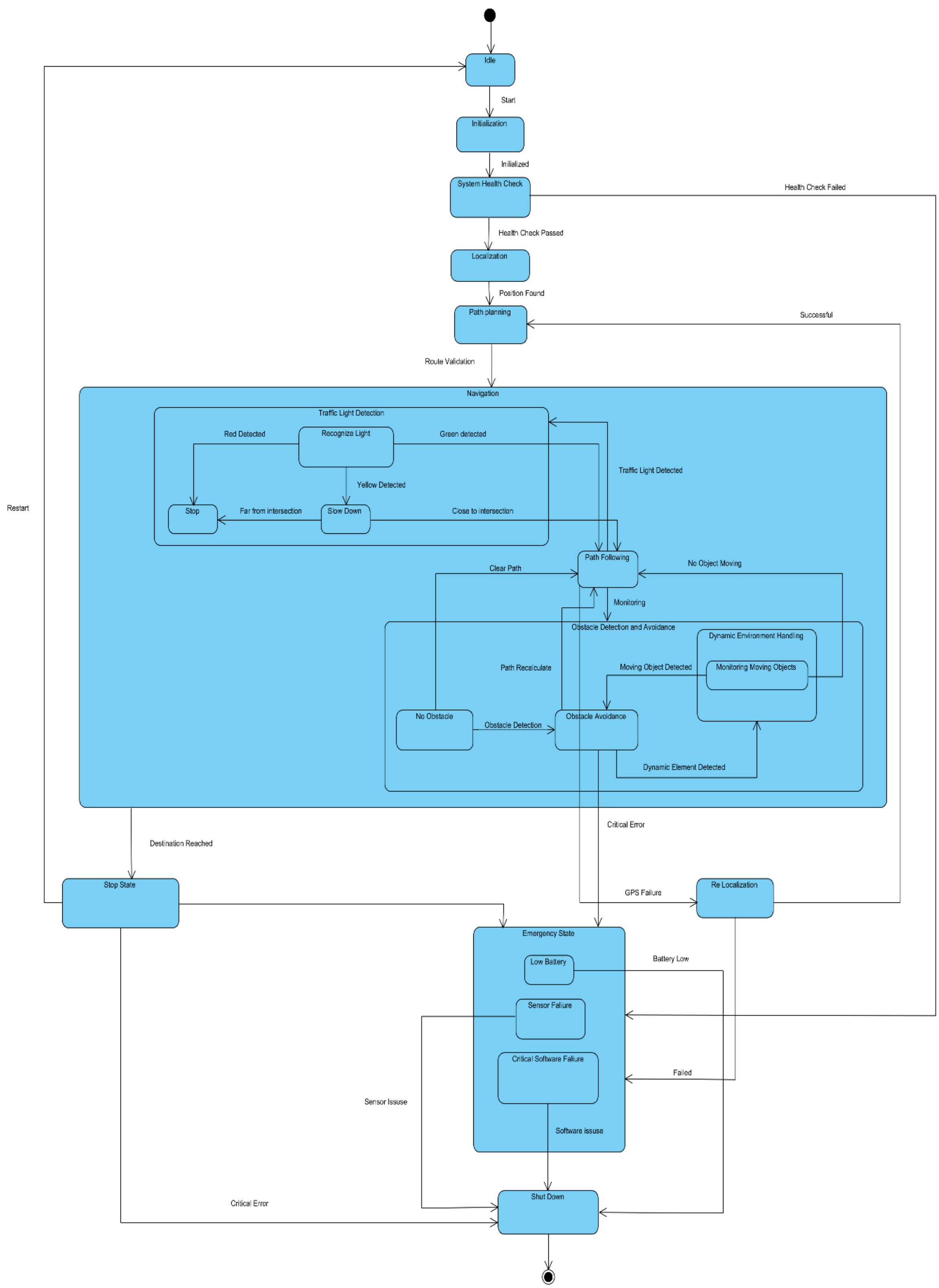

Figure 4.4: System state chart diagram

4.3.3.1 Path planning state chart diagram

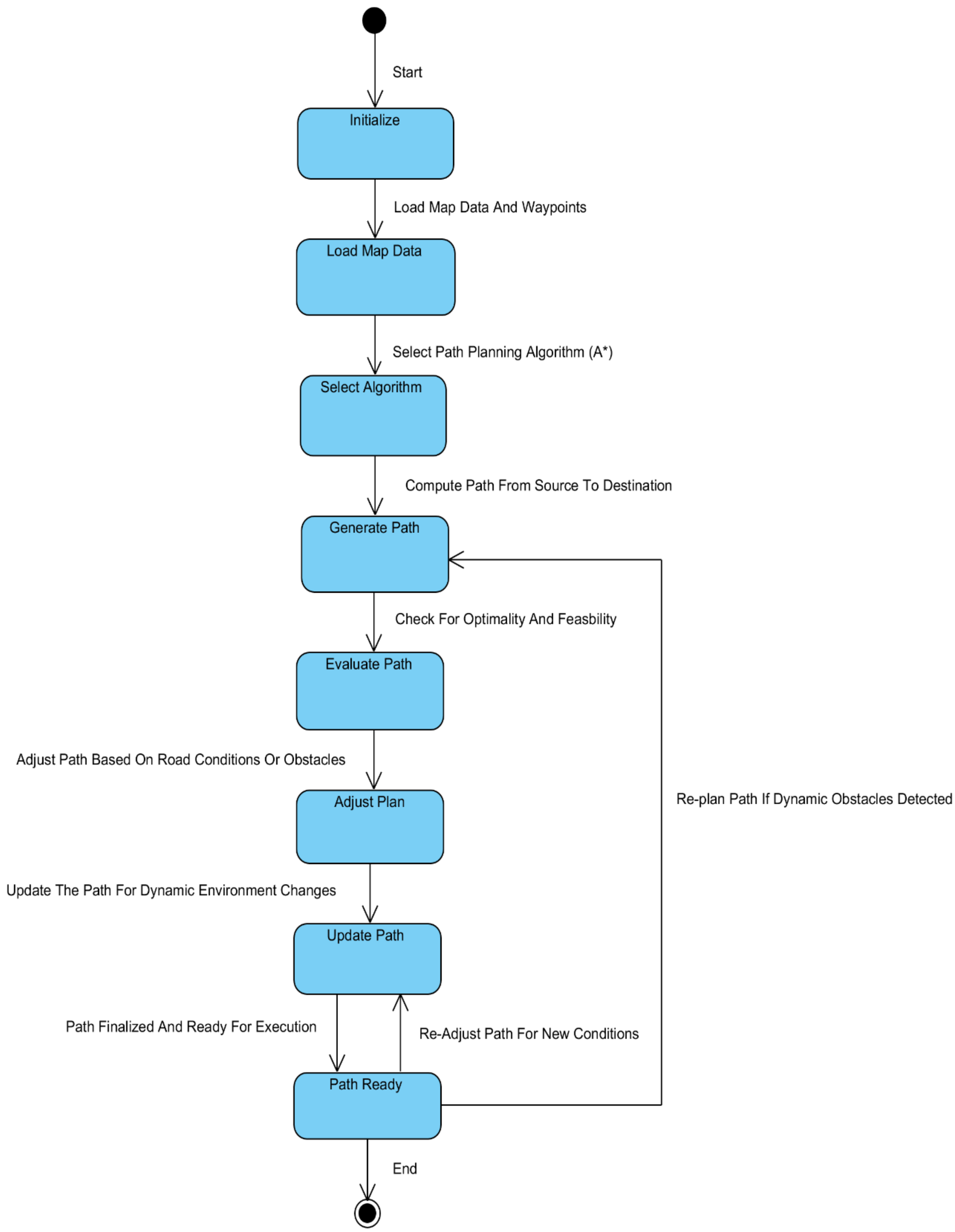

Figure 4.5: Path Planning State Chart diagram

4.3.3.2 Path following state chart diagram

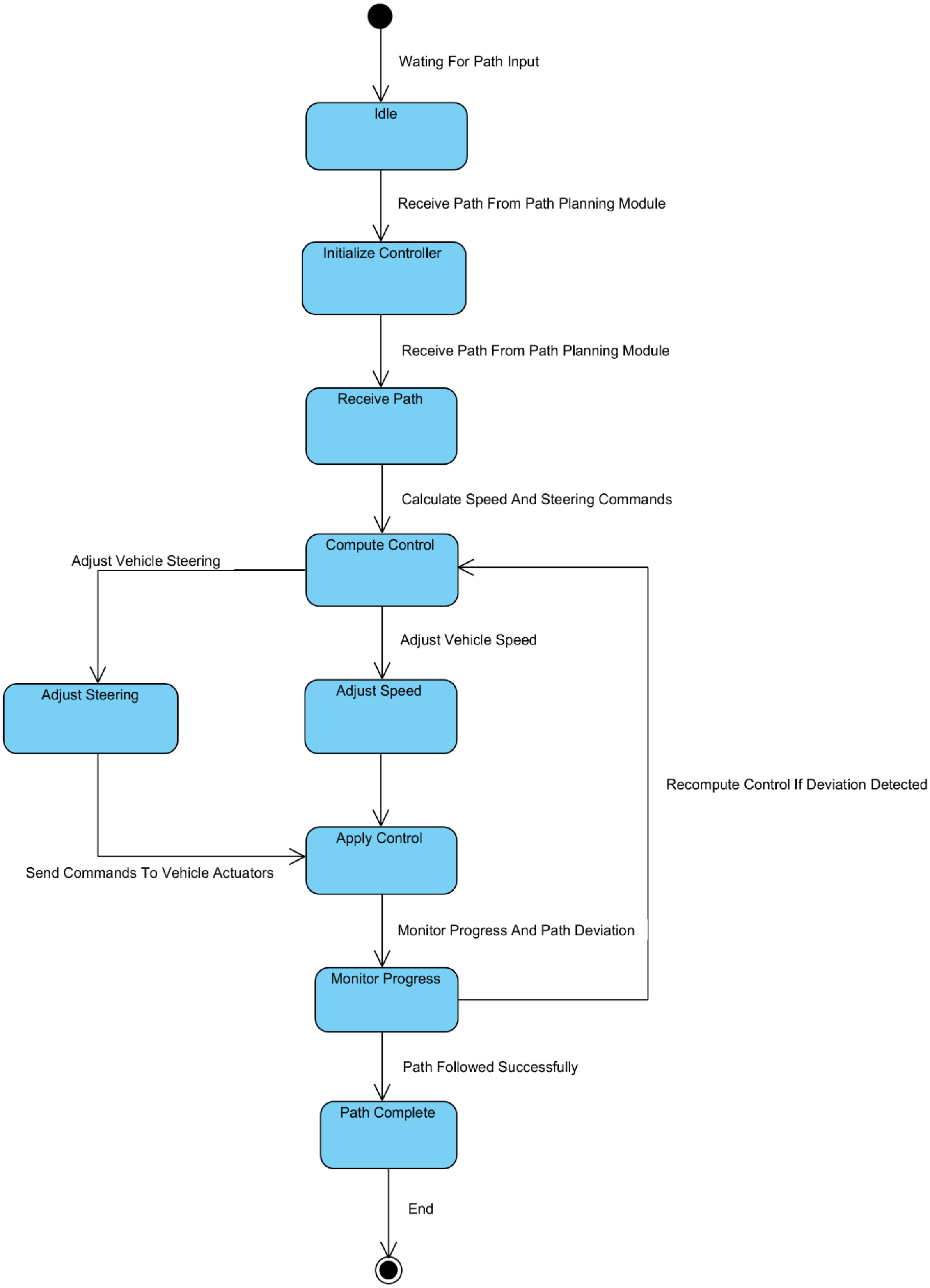

Figure 4.6: Path Following State Chart diagram

4.3.3.3 Vehicle control state chart diagram

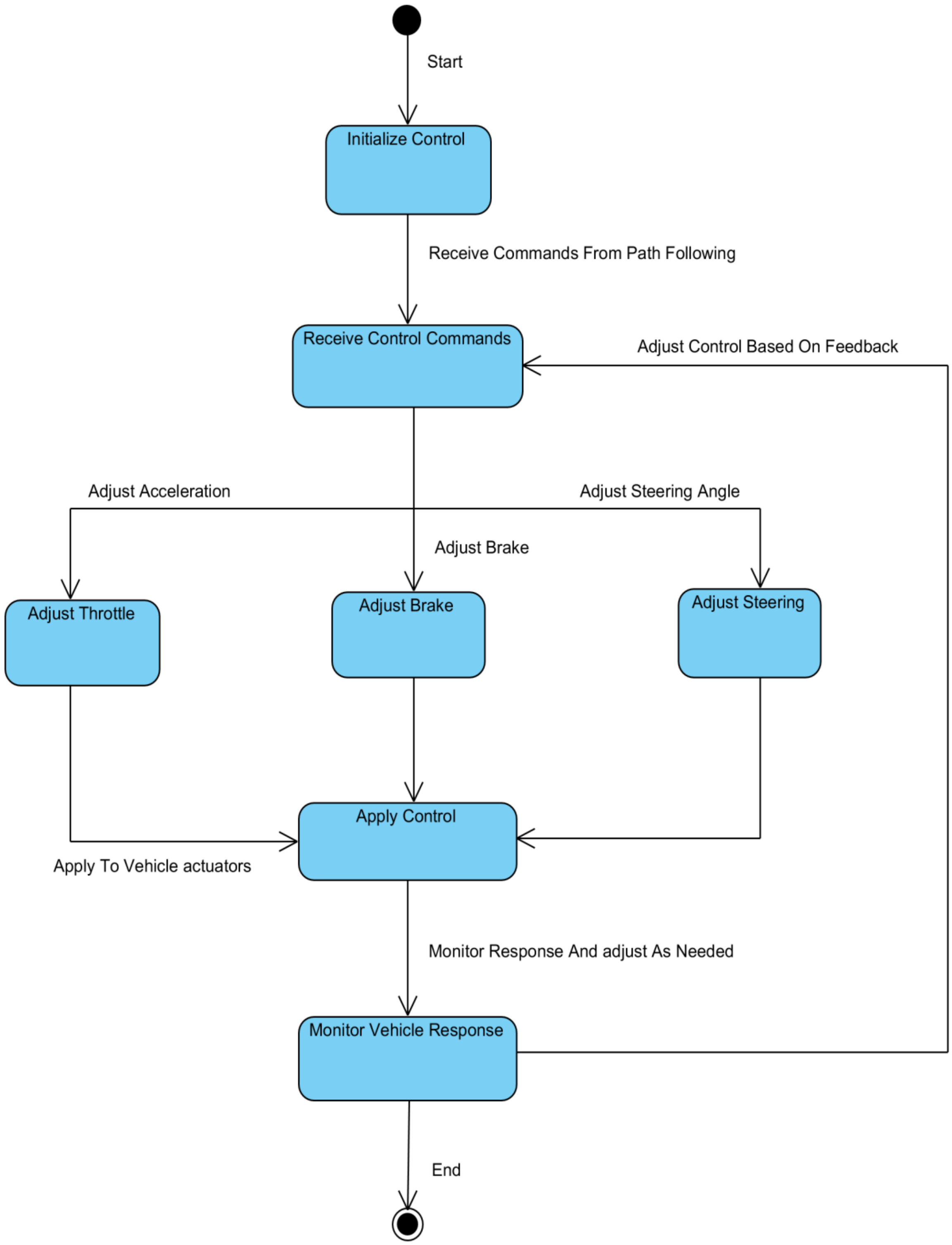

Figure 4.7: Vehicle Control State Chart diagram

4.3.3.4 Localization state chart diagram

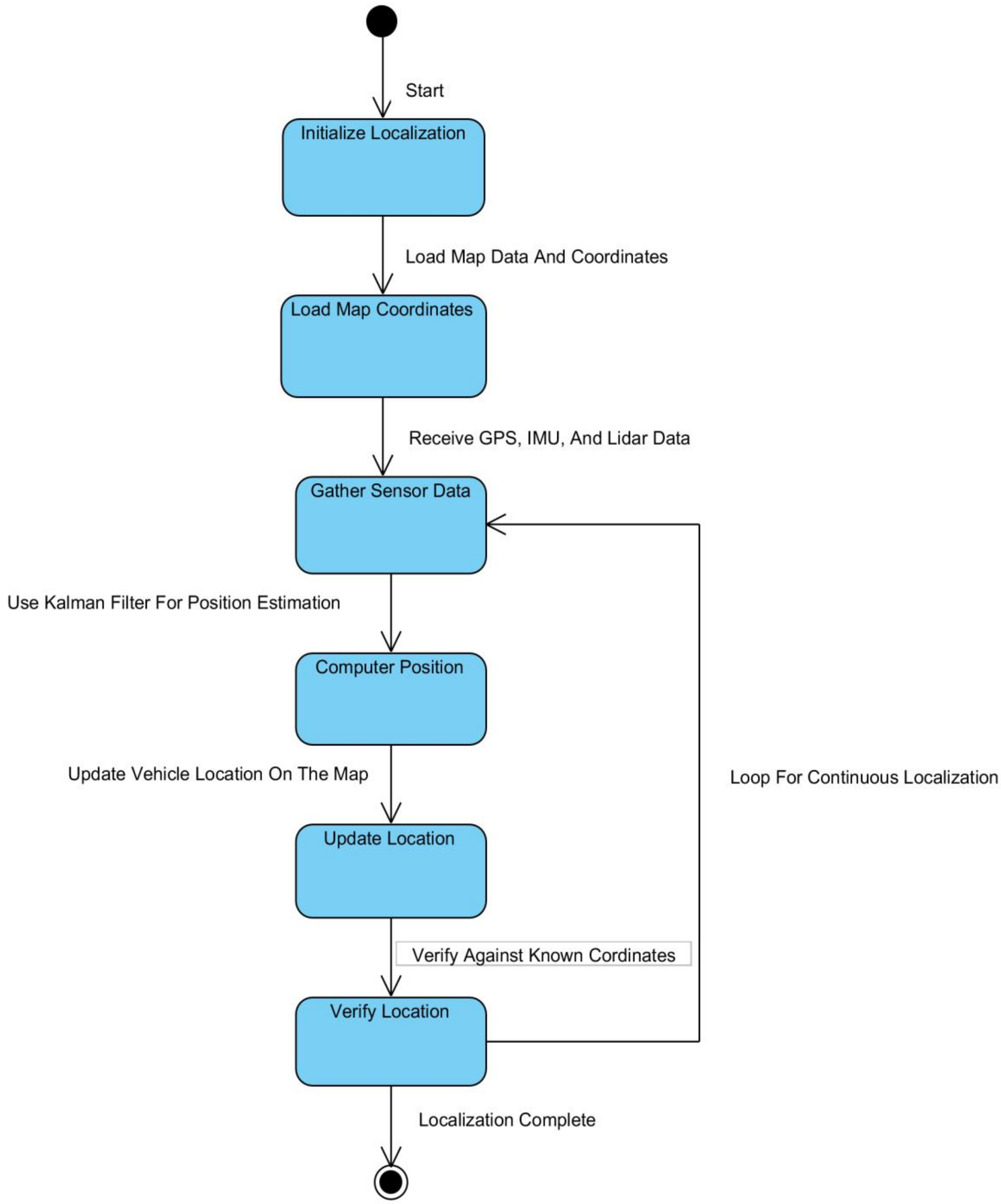

Figure 4.8: Localization State Chart diagram

4.3.3.5 Perception state chart diagram

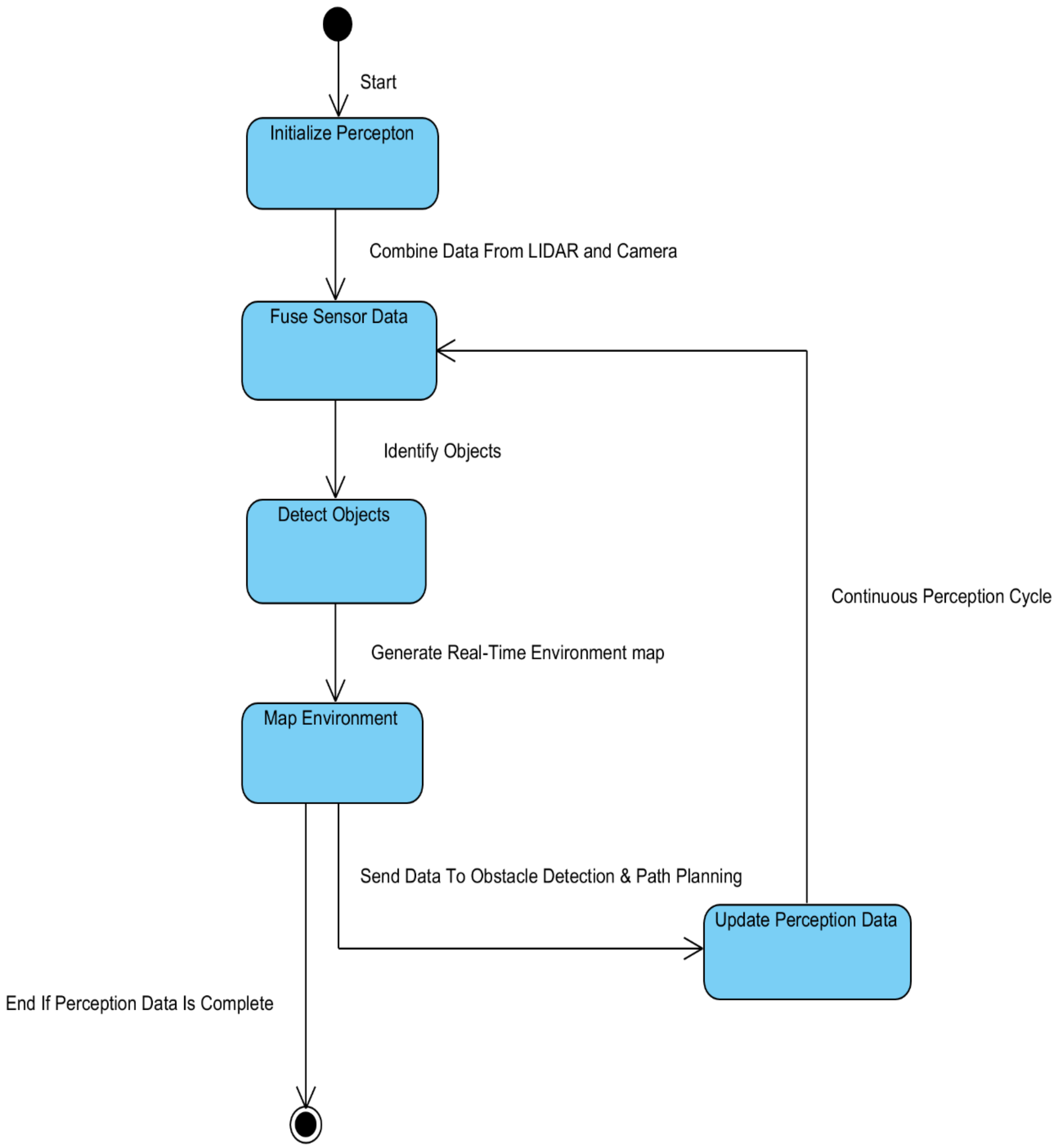

Figure 4.9: Perception State Chart diagram

4.3.3.6 Obstacle detection and avoidance state chart diagram

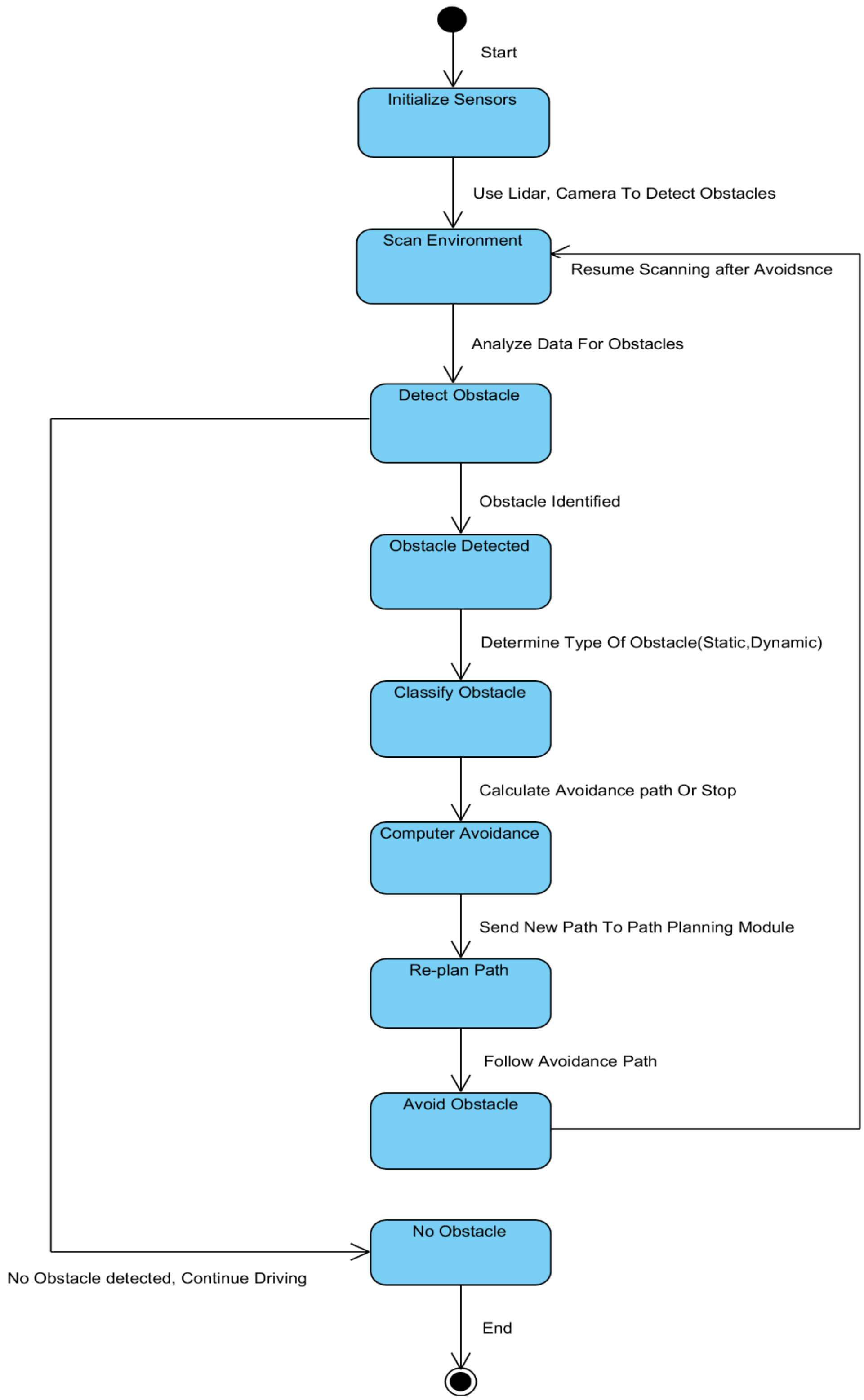

Figure 4.10: Obstacle Detection and avoidance State Chart diagram

4.3.3.7 Error handling and recovery state chart diagram

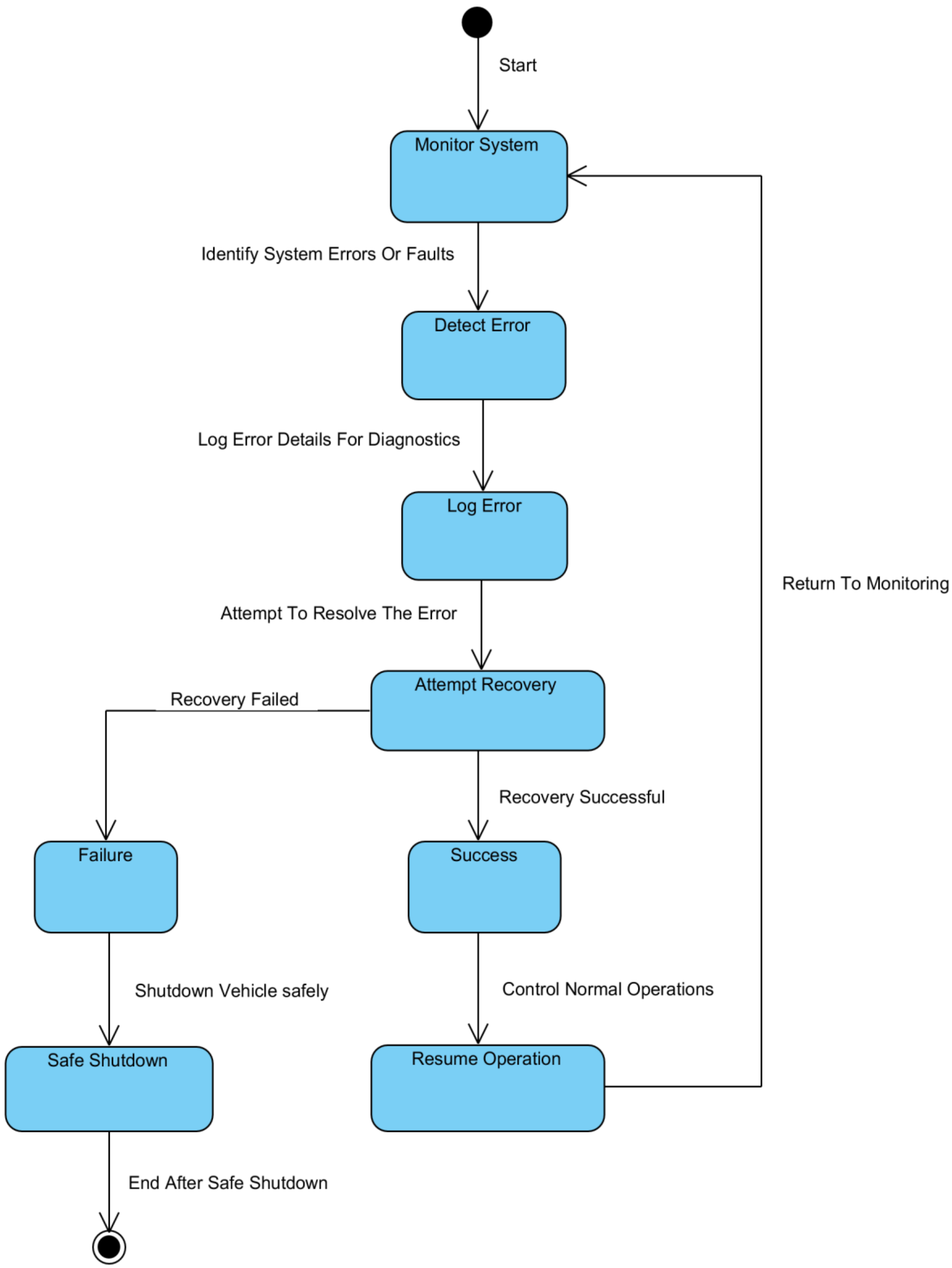

Figure 4.11: Error Handling and Recovery State Chart diagram

4.3.3.8 Traffic light detection state chart diagram

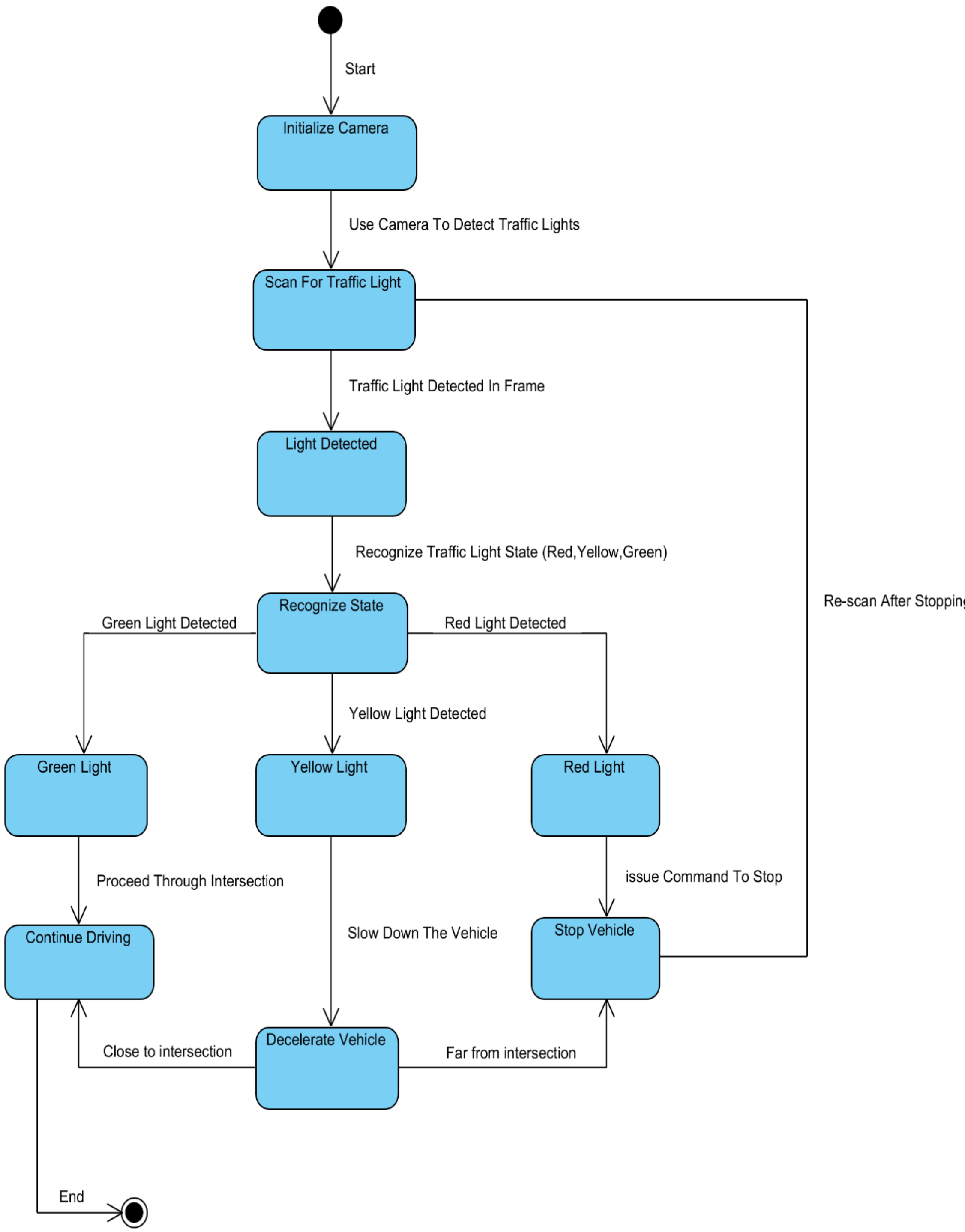

Figure 4.12: Traffic Light Detection State Chart diagram

4.3.3.9 Simulation integration state chart diagram

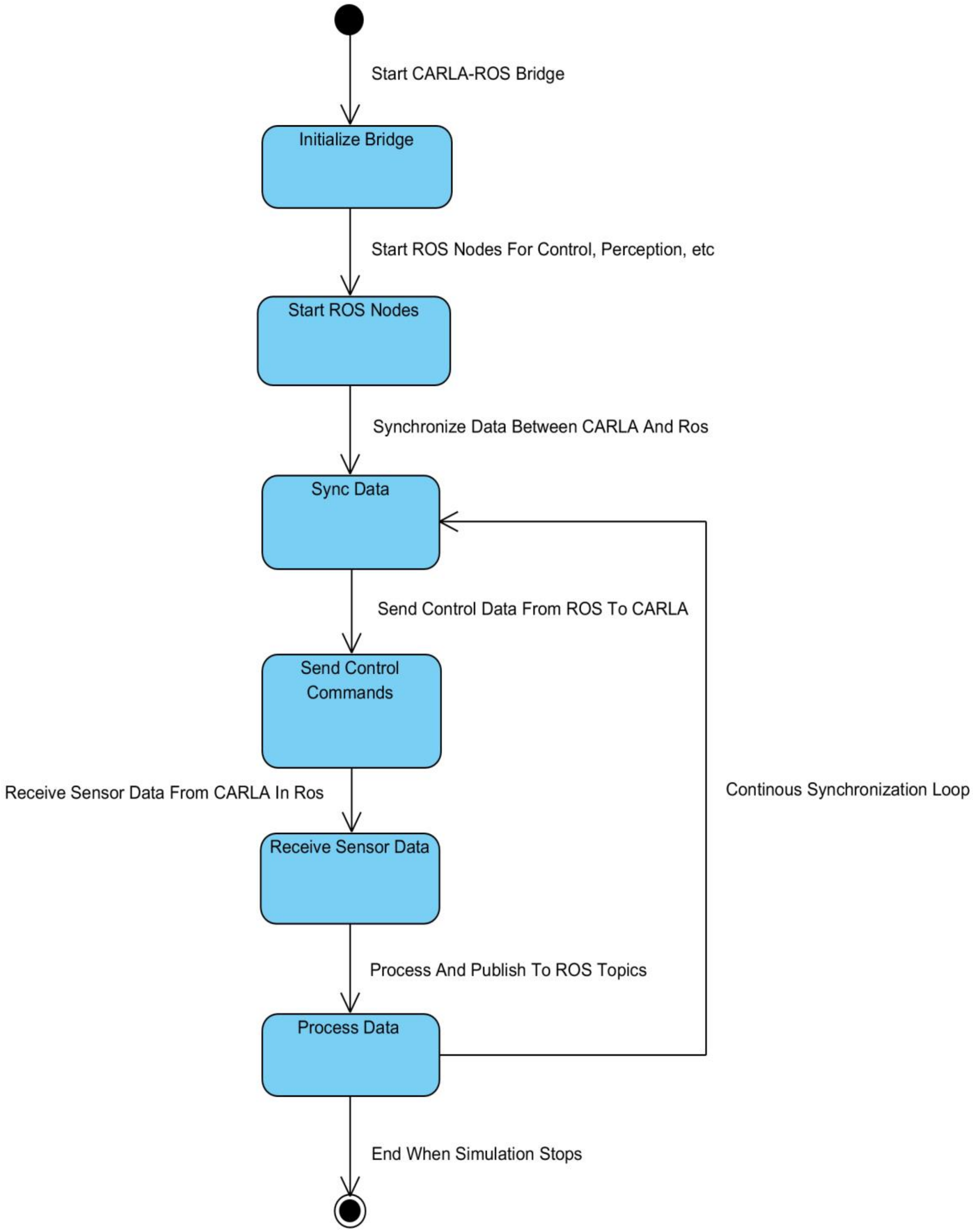

Figure 4.13: Simulation Integration State Chart diagram

4.4 SQA activity: State-Based Defect Detection Scenarios

4.4.1 Path Planning

Equivalence Class Partitioning (ECP):

- **Valid Classes**:
  - The destination is selected from the provided options.
  - The destination is entered manually and is valid (x is integer, y is integer).
- **Invalid Classes**:
  - The destination is selected but is not available (e.g., out of service area).
  - The destination coordinates are entered manually but are invalid (e.g., incorrect format, non-existent location).

Scenarios and Test Case:

Table 4.1: State-Based TC1 For Path Planning

Scenario	Input Value	ECP	Expected Output
Out of service area coordinates	x = 80.000000 y = 170.000000	Invalid	**Error**: Vehicle tries to go to the entered Coordinates, even if they are in any building

4.4.2 Path Following

Equivalence Class Partitioning (ECP):

- **Valid Classes**:
  - The vehicle's velocity and acceleration parameters are within normal operational ranges. i.e. <120 km/h
- **Invalid Classes**:
  - The vehicle's velocity or acceleration parameters are abnormal or invalid. i.e. = 120km/h

Scenarios and Test Cases:

Table 4.2: State-Based TC2 For Path Following

Test Case	Input Value	ECP	Expected Output
Abnormal Velocity Parameters	Velocity = 200 km/h	Invalid	Unexpected Error
Negative Velocity Parameters	Velocity = -20 km/h	Invalid	Unexpected Error

4.4.3 Vehicle Control

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - The vehicle's speed is within the normal operational range (i.e. 0 km/h to maximum speed limit).
  - The throttle position is within the normal operational range (i.e. 0% to 100%).
- **Invalid Classes:**
  - The vehicle's speed parameters are abnormal or invalid (i.e. speed exceeding maximum permissible limit).
  - The throttle position is abnormal or invalid (i.e. throttle position exceeding 100%).

Scenarios and Test Cases:

Table 4.3: State-Based TC3 For Vehicle Control

Test Case	Input Value	ECP	Expected Output
Negative Speed	Speed = -10 km/h	Invalid	Unexpected Error

Negative Throttle Position	Throttle = -20%	Invalid	Unexpected Error

4.4.4 Vehicle Control

Equivalence Class Partitioning (ECP):

- **Valid Classes:**

Normal Steering: Steering angle within operational range

  - -90° to 90° latitude, -180° to 180° longitude

- **Invalid Classes:**

Abnormal Steering: Steering angle outside operational range (< -30° or > +30°)

Scenarios and Test Cases:

Table 4.4: State-Based TC4 For Vehicle Control

Test Case	Input Value	ECP	Expected Output
Abnormal Orientation	Roll = -220° Pitch = of 120°	Invalid	Unexpected Error
Abnormal Steering Angle	Range = -45°, 40°	Invalid	Unexpected Error

4.4.5 Vehicle Control

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Speed: 0 km/h ≤ Speed ≤ 120 km/h
  - Distance: 2 meters ≤ Distance ≤100 meters
  - Throttle Adjustment: 0 % ≤ Throttle ≤ 80 %
  - Brake Application: 0 % ≤ Braking Force ≤ 100 %

- **Invalid Classes:**
    - Speed: > 120 km/h
    - Distance: Distance >100 meters
    - Throttle Adjustment: < 0 % or Throttle > 80 %
    - Brake Application: < 0 % or Braking Force > 100 %

Scenarios and Test Cases:

Table 4.5: State-Based TC5 For Vehicle Control

Test Case	Input Value	ECP	Expected Output
Abnormal Steering Angle	Range = -45°, 40°	Invalid	Unexpected Error
Unsafe distance	Distance = 0	Invalid	Unexpected Error
Braking force	Force = 152%	Invalid	Unexpected Error
Abnormal Speed	Speed = -15.2	Invalid	Unexpected Error

4.4.6 Vehicle Control

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
    - Lateral Position: -1.0 meters ≤ Lateral Position ≤ 1.0 meters
    - Steering Adjustment: -30° ≤ Steering Angle ≤30°
- **Invalid Classes:**
    - Lateral Position: Lateral Position > 1.0 meters
    - Steering Adjustment: Steering Angle > 30°

Scenarios and Test Cases:

Table 4.6: State-Based TC6 For Vehicle Control

Test Case	Input Value	ECP	Expected Output
Abnormal Lateral Position	Lateral Position = -2.0 meters	Invalid	Unexpected Error
Excessive Steering Adjustment	Angle = -45.23°	Invalid	Unexpected Error

4.4.7 Localization Module Test Cases

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Sensor data (GPS, IMU, LIDAR) within acceptable ranges.
  - GPS accuracy ≤ 5 meters.
  - IMU data drift ≤ 2 degrees.
  - LIDAR scan range ≥ 50 meters.
- **Invalid Classes:**
  - Sensor data outside acceptable ranges.
  - GPS accuracy > 5 meters.
  - IMU data drift > 2 degrees.
  - LIDAR scan range < 50 meters.

Scenarios and Test Cases:

Table 4.7: State-Based TC7 For Localization

Scenario	Input Value	ECP	Expected Output
GPS Signal Loss	GPS Accuracy = 15 meters	Invalid	Transition to Error State: "Localization Error"
GPS Signal Loss	GPS Status = No signal	Invalid	Unexpected Error
IMU Drift	IMU Drift = 1.5 degrees	Valid	Transition to Update Location
LIDAR Scan Range Too Short	LIDAR Range = 30 meters	Invalid	Transition to Error State: "LIDAR Range Error"
Accurate Localization	GPS Accuracy = 3 meters IMU Drift = 1 degree	Valid	Transition to VerifyLocation

4.4.8 Obstacle Detection and Avoidance Module Test Cases

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Obstacle detected within sensor range.
  - LIDAR detection distance ≤ 100 meters.
  - Obstacle size ≥ 0.5 meters.
  - Obstacle-free zone.
  - No objects detected within 100 meters.
- **Invalid Classes:**
  - Sensor fails to detect within expected range.
  - LIDAR detection distance > 100 meters for a detected obstacle.

- Obstacle size < 0.5 meters considered noise.

Scenarios and Test Cases:

Table 4.8: State-Based TC8 For Obstacle Detection and Avoidance

Scenario	Input Value	ECP	Expected Output
No Obstacle Detected	LIDAR Detection Distance = 150 meters	Invalid	Transition to No Obstacle state
LIDAR Sensor Failure	LIDAR Status = No data received	Invalid	Unexpected Error
Valid Obstacle Detected	LIDAR Detection Distance = 50 meters	Valid	Transition to Classify Obstacle
Dynamic Obstacle Within Range	LIDAR Detection Distance = 80 meters	Valid	Transition to Compute Avoidance

4.4.9 Traffic Light Detection Module Test Cases

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Traffic light detected and state correctly identified.
  - Distance to traffic light ≤ 50 meters.
  - Recognition confidence ≥ 80%.
- **Invalid Classes:**
  - Traffic light detection errors or low recognition confidence.
  - Distance to traffic light > 50 meters.
  - Recognition confidence < 80%.

Scenarios and Test Cases:

Table 4.9: State-Based TC9 For Traffic Light Detection

Scenario	Input Value	ECP	Expected Output
Traffic Light Not Detected	Detection Distance = 60 meters	Invalid	Continue scanning in ScanForTrafficLight state
Traffic Light Detected, High Confidence	Recognition Confidence = 90%	Valid	Transition to Recognize State
Camera Failure	Camera Status = No data received	Invalid	Unexpected Error
Low Confidence in Recognition	Recognition Confidence = 70%	Invalid	Re-scan for traffic light state
Traffic Light at Threshold	Detection Distance = 50 meters	Valid	Proceed with state recognition (Red/Yellow/Green)

Chapter 5: Implementation

Chapter 5: Implementation

5.1 Endeavour

In the implementation phase, our team applies rigorous software engineering principles. We plan and execute each task, adhering to industry best practices. From architectural design to testing, our approach reflects our commitment to delivering high-quality software solutions

5.1.1. Team

- Bilal Rafiq
- Hamza Azhar
- Sardar Mohsin Saghir
- Muhammad Usama Nazir

5.1.2. Work Breakdown Structure

1. **Project Management**

    1.1. Work Breakdown Structure (WBS)
    1.2. Roles & Responsibility Matrix
    1.3. Change Control System
    1.4. Meeting minutes and Progress report

2. **Reports / Documentation**

    2.1. Team Members and Project Proposal
    2.2. Project Proposal Document
        2.2.1. Opportunity and Stakeholders
        2.2.2. Challenges Goals and Objectives
        2.2.3. Solution Overview diagram
        2.2.4. Report Outline
    2.3. Literature Review
        2.3.1. Domain Expert Interview Findings

2.3.2. Questionnaire for Technical Feasibility and Risk Assessment

2.3.3. Brainstorming diagram

2.3.4. Academic Research Review

2.3.5. Gap analysis summary

2.3.6. Technology Landscape

2.3.6.1. SWOT analysis

2.3.7. Questionnaire for Selecting tools and techniques

2.3.8. Specialization - 4 courses series from Coursera

2.4. Requirement Analysis

2.4.1. Problem Scenarios

2.4.2. Requirement Elicitation

2.4.3. Questionnaire for gathering requirements

2.4.4. Functional Requirements

2.4.5. Non-Functional Requirement

2.4.6. Inspection Report

2.4.7. Software requirement specification artifact

2.5. System Design

2.5.1. Architecture Diagram

2.5.2. Use Case Diagram

2.5.3. Detail Use Cases

2.5.4. Activity Diagrams

2.5.5. System Sequence Diagram

2.6. Implementation

2.6.1. Components and Libraries

2.7. Testing and Performance Evaluation

2.7.1. Test Scenarios

2.8.Conclusion & Outlook

2.8.1. Future Recommendations

2.9.Progress Presentation

2.9.1. Slides outlining project progress

2.9.2. Updated Artifacts of part 1

2.9.2.1. Appendix-A: Software Requirements Specifications (SRS)

2.9.2.2. Appendix-B: Design Documents

2.9.2.3. Appendix-C: Coding Standards/Conventions

2.9.2.4. Appendix-D: Test Scenarios

2.9.2.5. Appendix-E: Work Breakdown Structure

2.9.2.6. Appendix-F: Roles & Responsibility Matrix

2.9.3. Answers to potential questions report

2.10. Final Presentation part 2

2.10.1. Comprehensive Slides for presentation

2.10.2. Working software system (Complete)

2.10.3. Updated Artifacts (Complete)

2.10.3.1. Appendix-A: Software Requirements Specifications (SRS)

2.10.3.2. Appendix-B: Design Documents

2.10.3.3. Appendix-C: Coding Standards/Conventions

2.10.3.4. Appendix-D: Test Scenarios

2.10.3.5. Appendix-E: Work Breakdown Structure

2.10.3.6. Appendix-F: Roles & Responsibility Matrix

2.10.4. Final Report

3. System

3.1. Development Environment

3.1.1. IDE

3.1.1.1. Visual Studio Code

3.1.1.2. PyCharm

3.1.2. Version Control

3.1.2.1. Git Hub

3.1.3. Environment Management

3.1.3.1. Anaconda Distribution

3.2. Simulation Environment Setup

3.2.1. CARLA Simulator

3.2.1.1. Carlaviz for CARLA Visualization

3.2.2. ROS Noetic Configured

3.2.3. CARLA-ROS Bridge Integrated

3.2.4. Vehicle spawn module

3.2.5. Sensor spawn module

3.2.6. Destroy Vehicle module

3.3. Path Planning component

3.3.1. Map Reading module

3.3.2. Graph of Roads

3.3.3. Graph of Lanes

3.3.4. List of Driving Lanes within map

3.3.5. Route Calculation module

3.3.6. Algorithm implementation module

3.3.7. Global route planner module

3.3.8. Axis Translation module

3.3.9. Local route planner module

3.3.10. Environment Analysis module

3.3.11. Trajectory Generation module

3.3.12. Junction handling module

3.4. Path Following component

3.4.1. Trajectory Tracking module

3.4.2. Basic agent module

3.4.3. Behaviour agent module

3.4.4. Algorithm implementation module

3.4.5. Controller module

3.4.6. Custom Destination module

3.5. Vehicle Control component

3.5.1. Throttle Control module

3.5.2. Braking Control module

3.5.3. Acceleration Control module

- 3.5.4. Steering Control module
- 3.5.5. Longitudinal Control module
- 3.5.6. Lateral Control module
- 3.5.7. Lane changing module
- 3.5.8. Jerkiness Control algorithm modules
- 3.5.9. Rotation and Translation module

3.6. Sensor Integration module

- 3.6.1. IMU integration sub-module
- 3.6.2. GPS integration sub-module
- 3.6.3. Radar integration sub-module
- 3.6.4. Lidar integration sub-module

3.7. Obstacle Detection

- 3.7.1. Sensor Fusion module
  - 3.7.1.1. Lidar-Radar Fusion sub-module
  - 3.7.1.2. Multi-sensor Data synchronization sub-module
- 3.7.2. Sensor Data Processing module
- 3.7.3. Obstacle Detection module
  - 3.7.3.1. ML based detection sub-module
- 3.7.4. Distance Estimation module
- 3.7.5. Object Classification module

3.8. Obstacle Avoidance

- 3.8.1. Dynamic Obstacle handling module
- 3.8.2. Static Obstacle handling module
- 3.8.3. Path Adjustment module
  - 3.8.3.1. Map based planning sub-module
  - 3.8.3.2. Graph based planning sub-module
- 3.8.4. Trajectory Estimation module
- 3.8.5. Maneuver Planning module
  - 3.8.5.1. Environmental evaluation sub- module
  - 3.8.5.2. Lane changes sub-module

3.8.5.3. Decelerate sub-module

3.8.5.4. Emergency Stop sub-module

3.8.6. Real-time Response module

3.8.7. Tracking module

3.8.7.1. Particle filter sub-module

4. Open House

1.1 Event Part 1

4.1.1. Standee Design

4.1.2. Printed Standee

4.1.3. Printed Broachers

4.1.4. Pre-recorded Demo video

4.2. Event Part 2

4.2.1. Standee Design

4.2.2. Printed Standee

4.2.3. Printed Broachers

4.2.4. Full Working Software

5.1.3. Roles & Responsibility Matrix:

Table 5.1: Responsibilities Assignment Matrix

WBS#	WBS Deliverable	Activity #	Activity to complete the deliverable	Duration (days)	Responsible Team Member(s) & Role(s)
1	Project Initiation Phase	1	Literature Review	7	Bilal (A) Hamza (R) Mohsin (I) Usama (R)
		2	Define project scope and objectives	5	Bilal (A/R) Hamza (C) Mohsin (C) Usama (I)

		3	Establish project team roles and responsibilities	1	Bilal (A/R) Hamza (C) Mohsin (I) Usama (I)
		4	Setup project management tools and communication channels	1	Bilal (C) Hamza (A) Mohsin (I) Usama (R)
2	Requirement Analysis	5	Research existing autonomous vehicle technologies and solutions	3	Bilal (C) Hamza (A/R) Mohsin (I) Usama (I)
		6	Gather requirements from stakeholders	5	Bilal (A) Hamza (R) Mohsin (C) Usama (C)
		7	Brainstorming	2	Bilal (R) Hamza (A) Mohsin (C) Usama (C)
		8	Define Problem Scenarios	1	Bilal (R) Hamza (A) Mohsin (C) Usama (I)
		9	Interview Domain Expert	2 Meetings per week	Bilal (A) Hamza (R) Mohsin (I) Usama (I)

		10	Define Functional Requirements	4	Bilal (R) Hamza (A) Mohsin (C) Usama (I)
		11	Specify Non-Functional Requirement	1	Bilal (A/R) Hamza (C) Mohsin (I) Usama (I)
		12	System Overview	2	Bilal (C) Hamza (R) Mohsin (I) Usama (A)
		13	Constraints	1	Bilal (A/R) Hamza (C) Mohsin (I) Usama (I)
3	System Design	14	Develop Architecture Diagram	1	Bilal (C) Hamza (A/R) Mohsin (I) Usama (C)
		15	Create Use Case Diagram	1	Bilal (C) Hamza (A/R) Mohsin (R) Usama (I)
		16	Define Detail Use Cases	3	Bilal (A) Hamza (R) Mohsin (I) Usama (C)
		17	Design Activity Diagrams	3	Bilal (C) Hamza (I) Mohsin (A/R)

					Usama (I)
		18	Construct System Sequence Diagram	1	Bilal (C) Hamza (A) Mohsin (R) Usama (I)
4	Simulation Environment Setup	19	Install and configure CARLA simulator, ROS Noetic and environment	8	Bilal (A) Hamza (C) Mohsin (I) Usama (R)
		20	Develop scripts for setting up simulation scenarios	7	Bilal (A/R) Hamza (C) Mohsin (I) Usama (I)
		21	Verify integration between CARLA and ROS	1	Bilal (A) Hamza (R) Mohsin (I) Usama (I)
5	Path Planning Algorithm Development	22	Defining algorithms for path planning considering dynamic obstacles	3	Bilal (A/R) Hamza (C) Mohsin (C) Usama (I)
		23	Path planning logic in Python using ROS	20	Bilal (A) Hamza (R) Mohsin (I) Usama (C)

		24	Route Calculation	5	Bilal (C) Hamza (A) Mohsin (I) Usama (R)
		25	Map Processing	1	Bilal (A) Hamza (I) Mohsin (C) Usama (R)
		26	Environment Analysis	2	Bilal (A) Hamza (R) Mohsin (I) Usama (C)
		27	Trajectory Generation	4	Bilal (C) Hamza (I) Mohsin (R) Usama (A)
		28	Calculating Waypoints	2	Bilal (A) Hamza (C) Mohsin (I) Usama (R)
		29	Test path planning algorithms in simulated environments	3	Bilal (A) Hamza (R) Mohsin (I) Usama (C)
6	Path Following Implementation	30	Defining control algorithms for vehicle control	2	Bilal (A/R) Hamza (R) Mohsin (I) Usama (C)

		31	Integrate path following logic/algorithm	7	Bilal (R) Hamza (A) Mohsin (C) Usama (I)
		32	Trajectory Tracking	2	Bilal (A) Hamza (R) Mohsin (C) Usama (I)
		33	Velocity Control	3	Bilal (A) Hamza (C) Mohsin (I) Usama (R)
		34	Steering Control	5	Bilal (C) Hamza (A) Mohsin (I) Usama (R)
		35	Conduct testing and validation in simulated environments	5	Bilal (C) Hamza (R) Mohsin (I) Usama (A)
7	Obstacle Detection	36	Defining strategies for detecting obstacles	3	Bilal (C) Hamza (I) Mohsin (A/R) Usama (I)
		37	Sensor Data Processing	5	Bilal (C) Hamza (I) Mohsin (A/R) Usama (I)
		38	Obstacle Detection	7	Bilal (A) Hamza (C) Mohsin (R)

					Usama (I)
		39	Distance Estimation	5	Bilal (C) Hamza (A) Mohsin (R) Usama (I)
8	Obstacle Avoidance	40	Defining avoidance Maneuver	1	Bilal (C) Hamza (A) Mohsin (R) Usama (I)
		41	Implement obstacle avoidance strategies	25	Bilal (C) Hamza (R) Mohsin (A) Usama (I)
		42	Path Adjustment	10	Bilal (C) Hamza (A) Mohsin (R) Usama (I)
		43	Maneuver Planning	5	Bilal (C) Hamza (I) Mohsin (A) Usama (R)
		44	Real Time Responding	5	Bilal (I) Hamza (C) Mohsin (A/R) Usama (C)
		45	Integrate obstacle detection and avoidance with overall system	5	Bilal (C) Hamza (I) Mohsin (R) Usama (A/R)

8	Sensor Integration and Calibration	46	Integrate sensors with the autonomous vehicle in simulation	2	Bilal (A) Hamza (C) Mohsin (I) Usama (R)
		47	Calibrate sensor data for accurate perception	6	Bilal (A) Hamza (C) Mohsin (R) Usama (I)
		48	Validate sensor data in simulated and real-world scenarios	7	Bilal (C) Hamza (A/R) Mohsin (R) Usama (I)
9	System Integration	49	Integrate all software components into the autonomous vehicle system	5	Bilal (I) Hamza (R) Mohsin (C) Usama (A/R)
10	Simulated Testing	50	Conduct comprehensive testing	6	Bilal (I) Hamza (A/R) Mohsin (C) Usama (R)
		51	Iterate on software development based on testing feedback	2	Bilal (R) Hamza (I) Mohsin (A) Usama (C)
		52	Fine-tune algorithms and	3	Bilal (C) Hamza (A/R)

			software based on testing results		Mohsin (R) Usama (I)
11	Optimization and Finalization	53	Optimize software performance and efficiency	2	Bilal (C) Hamza (R) Mohsin (A) Usama (I)
		54	Address any remaining issues or bugs	1	Bilal (I) Hamza (C) Mohsin (R) Usama (A/R)
		55	Finalize the project documentation and deliverables	2	Bilal (A/R) Hamza (C) Mohsin (C) Usama (C)

5.2 Proposed Solution

Our solution aims to enable autonomous vehicles to navigate by integrating advanced path planning, obstacle detection, and precise vehicle control. The following diagram outlines the proposed solution of our system.

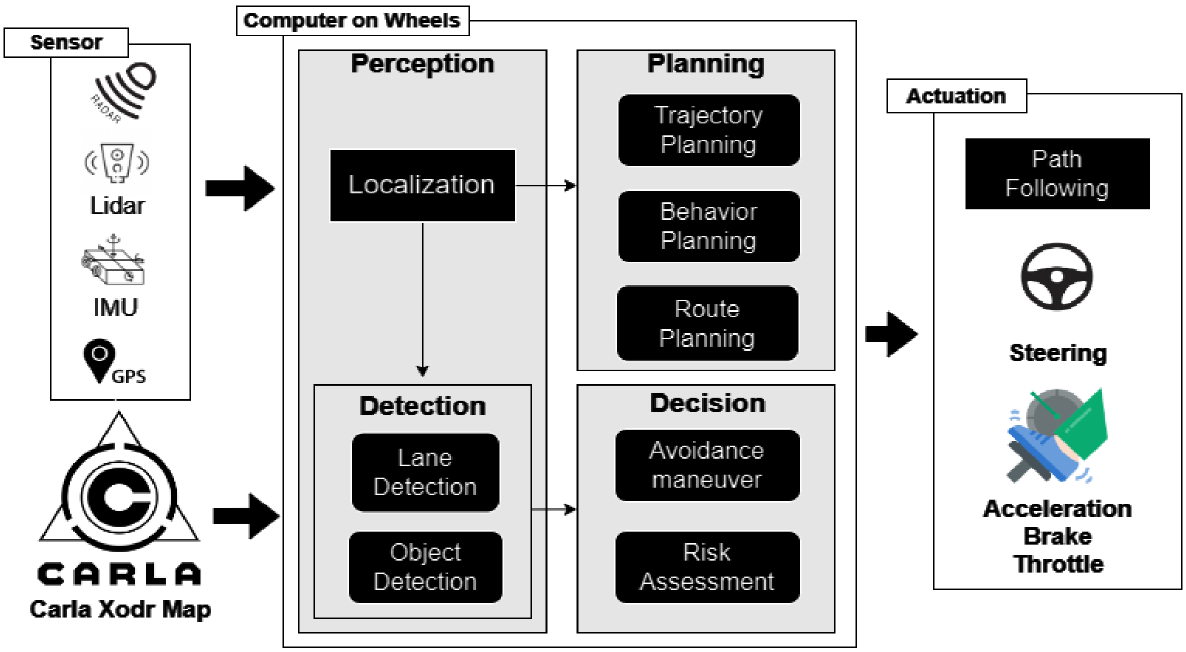

Figure 5.1: Proposed Solution

5.3 Components and Libraries

5.3.1 Components:

- Map Parser
- Traffic Generator
- Path Planner
- Trajectory Follower
- Behaviour Planner
- Environment Perception
- Obstacle Avoider
- Localization Module
- Sensor Data Fusion
- Control System
- Decision-Making Module
- Simulation Environment

- Obstacle Detector
- CARLA-ROS Bridge
- ROS Noetic Framework
- Traffic Light Detection

5.3.2 Libraries:

- **rospy:** Python client library for interacting with ROS nodes.
- **weakref:** Provides weak references to objects, helping manage memory in complex data structures.
- **NumPy:** Library for numerical computations and array manipulations.
- **pygame:** Library for creating multimedia applications, primarily games and visual simulations.
- **math:** Python's standard library for mathematical functions and operations.
- **carla:** Python API for interacting with the CARLA simulator for autonomous vehicle research.
- **xmltodict:** Converts XML data into Python dictionaries for easy data manipulation.
- **collections.deque:** Provides a double-ended queue for fast appends and pops.
- **argparse:** Parses command-line arguments to manage input parameters in scripts.
- **networkx as nx:** Library for creating and analyzing complex networks and graphs.
- **collections:** Python module with specialized data structures like namedtuple and defaultdict.
- **carla_msgs:** ROS message types specifically for CARLA-related data exchange.
- **datetime:** Library for handling date and time operations.
- **sensor_msgs:** ROS message types for data from sensors like cameras and LiDAR.
- **logging:** Standard Python library for generating log messages in applications.
- **OpenCV:** Computer vision library for image and video processing.
- **Matplotlib:** Plotting library for creating static, interactive, and animated visualizations.
- **TensorFlow:** Machine learning framework for building and training neural networks.
- **Cv2:** OpenCV's Python interface for computer vision tasks.

- **tf (ROS Transform Library):** Handles coordinate transformations in ROS.

5.4 IDE, Tools, Technologies and Development Platform

5.5.1. IDEs

- **PyCharm:** An integrated development environment (IDE) optimized for Python development with advanced code analysis and debugging tools.
- **Visual Studio Code:** A lightweight, versatile code editor with extensive language support, extensions, and debugging features.

5.5.2. Development Platform:

- **Ubuntu 20.04:** A Linux-based operating system commonly used for development and robotics applications.
- **ROS (Robot Operating System):** A flexible framework for developing and managing robotic applications, providing tools for communication, control, and simulation.

5.5.3. Tools

- **Git:** A version control system for tracking changes in code.
- **GitHub:** A cloud-based platform for hosting and collaborating on Git repositories.
- **Jira:** A project management tool for tracking tasks and issues.
- **Microsoft Office:** A suite of productivity applications, including Word, Excel, and PowerPoint.
- **Visual Paradigm:** A modelling tool for creating UML diagrams and design documentation.
- **OpenDRIVE Viewer:** A tool for visualizing road networks defined in OpenDRIVE format.
- **Carlaviz:** A web-based visualization tool for CARLA simulation data.
- **Anaconda:** A distribution for managing Python and data science packages and environments.

5.5.4. Technologies

- **Carla Simulator:** A high-fidelity simulator for autonomous driving research, providing realistic environments for testing.
- **Carla-Ros-Bridge:** A bridge enabling communication between CARLA and ROS, facilitating integration of simulation with ROS nodes.
- **OpenCV:** A computer vision library for image processing, object detection, and visual applications.
- **ROS Noetic:** The latest LTS (Long-Term Support) version of the Robot Operating System, designed for robotics development.
- **rospy:** The Python client library for interacting with ROS, enabling communication between Python programs and ROS nodes.
- **robot_localization:** A ROS package for state estimation and sensor fusion, typically used in robot navigation.
- **Python:** A high-level programming language widely used for software development, particularly in automation and robotics.

5.5 Best Practices and Coding Standards

5.5.1 Software Engineering Practice: VV Model

In our project, we adopted the **VV model** (View-View model) to ensure that both **validation** and **verification** were incorporated throughout the development process, aligning with the complex, modular nature of embedded software systems for autonomous vehicles. This approach allowed us to **test and validate** system functionality at various stages, focusing on both the design (views) and the operational behavior (viewpoints) of the system. By employing the VV model, we ensured that each system component met functional and non-functional requirements while maintaining flexibility for continuous improvements and adjustments, which is **essential in embedded systems development for autonomous vehicles.**

5.5.2 VV Model Phases

We structured our project with the **VV model**, integrating testing and validation concurrently with each phase of development. This model allowed us to iteratively assess each module’s functionality and system interactions, ensuring the software met real-time performance and safety requirements. The following key phases were part of our process:

5.5.2.1 Requirements Gathering and Analysis: We initiated the project with a comprehensive requirement gathering phase, focusing on both functional and non-functional aspects, including real-time constraints for autonomous navigation. Collaboration with domain experts and extensive documentation review ensured that all project objectives were well-defined, particularly in addressing unique requirements for personal autonomous vehicles. Concurrently, validation through early design views helped us align these requirements with realistic system expectations.

5.5.2.2 System Design: During the design phase, we translated the requirements into a layered system architecture, defining each module's functionality and its interactions. In parallel with this, we created **state charts** and **use case diagrams** as part of our **viewpoints** to ensure that the system design aligned with the functional specifications. Validation of design decisions was ongoing to verify that the system would behave as expected once implemented.

5.5.2.3 Implementation: With a finalized design, we proceeded to the implementation phase, where each component was developed according to the validated architecture. The modules, including path planning, obstacle detection, and control, were built incrementally with ongoing verification through unit testing. As development progressed, we continually tested the integration of individual components, ensuring that they performed in alignment with the original system requirements.

5.5.2.4 Integration and Testing: After the implementation phase, we focused on **integration and testing** through continuous validation of the system’s operation in various real-world scenarios. We employed **system testing**, **integration testing**, and **unit testing** to verify the behavior of individual components and their integration. **Viewpoint-based testing** ensured that system-wide behavior matched both functional and design specifications. Special attention was given to testing critical aspects like obstacle detection, traffic light recognition, and real-time system responses under different conditions, ensuring safety and reliability.

5.5.3 Documentation and Review Process

In line with the VV model's focus on ongoing verification and validation, documentation was maintained throughout each phase, ensuring traceability and continuous feedback for system improvement:

5.5.3.1 Phase Sign-Offs: At the end of each phase, a formal sign-off was conducted to verify that all objectives were met before proceeding. This process involved reviews from both development and testing perspectives to ensure the system's design and behavior were validated from all necessary viewpoints.

5.5.3.2 Detailed Documentation: Throughout the project, detailed documentation was maintained for every design decision, testing procedure, and system behavior validation. This ensured that the system's design and implementation could be referenced and updated in the future, while also providing a solid foundation for troubleshooting or extending system functionalities.

5.5.4 Python coding Standards

- Use snake_case for variable and function names. [13]
- Use CamelCase for class names. [13]
- Follow PEP 8 guidelines for code formatting. [13]
- Use meaningful variable and function names. [13]
- Keep lines of code within 79 characters. [13]
- Use comments to explain complex parts of the code. [13]
- Use docstrings to document modules, classes, and functions. [12]
- Avoid using global variables unless necessary. [13]
- Handle exceptions gracefully. [13]
- Use virtual environments to manage dependencies. [13]

5.5.5 Rospy coding Standards

- Follow Python coding standards for rospy code. [13]
- Use rospy naming conventions for nodes, topics, and services. [14]
- Utilize rospy log functions for logging messages. [14]
- Ensure ROS dependencies are properly declared in package.xml and CMakeLists.txt. [14]
- Document ROS nodes, topics, and services using ROS comments. [14]
- Use rospy's rospy.spin() to keep the node alive. [15]
- Handle ROS messages and services according to their specifications. [15]
- Use rospy's parameter server for managing node parameters. [15]

5.6 Deployment Environment

A local server hosts the CARLA simulator and the autonomous vehicle software system, facilitating communication via the CARLA-ROS bridge.

5.6.1 Deployment Diagram

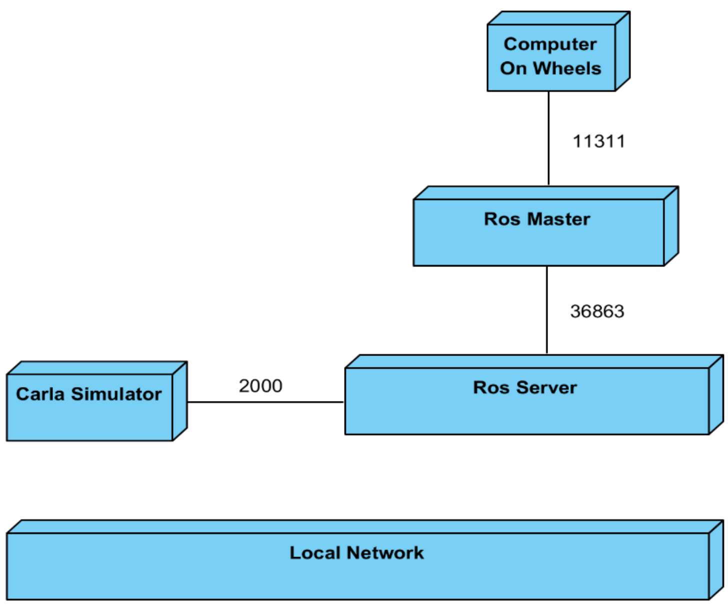

Figure 5.5: Deployment diagram

5.7 SQA activity: Defect Detection Through White Box Testing

5.7.1 Test Cases: Dijkstra vs A* with Obstacle on Path

The following test cases highlight the different behaviours of Dijkstra and A* algorithms when faced with obstacles in path planning scenarios. Dijkstra fails because of significant performance degradation due to frequent re-routing around obstacles, whereas A* demonstrates resilience by dynamically adjusting its route based on heuristic information, thereby providing reliable navigation solutions for autonomous systems.

Equivalence Class Partitioning (ECP) for start_node and end_node

- Valid Classes
  - start_node and end_node are valid integers within the grid bounds.
  - $x_{start}, x_{end} \in Z$
  - $y_{start}, y_{end} \in Z$
- Invalid Classes
  - start_node or end_node as non-integer values or out of grid bounds.
  - $x_{start}, x_{end}, y_{start}, y_{end} \notin Z$

*Table 5.2: White Box TC1 For Dijkstra vs A**

		Input Variables				
Test ID	**Algorithm**	**start_node**	**end_node**	**ECP**	**Actual Output**	**Error/Defect**
TC001	Dijkstra	(1,1)	(5,5)	Valid	Error: "Path blocked by obstacle"	Significant performance degradation due to frequent re-routing around obstacles

TC002	A*	(1,1)	(5,5)	Valid	List of node ids (int) connecting origin and destination, avoiding the obstacle.	None

5.7.2 Endless erratic behavior Around Destination

These test cases focus on the behaviour of the vehicle when it receives the same destination coordinates but its already there, in such case the vehicle starts behaving erratically

Equivalence Class Partitioning (ECP) for x and y

- Valid Classes
  - X and Y are valid integers within the grid bounds.
  - x and y ∈ Z
- Invalid Classes
  - X or Y as non-integer values or out of grid bounds.
  - X, Y ∉ Z

Table 5.3: White Box TC2 For Endless erratic behavior Around Destination

	Input Variables				
Test ID	**x**	**y**	**ECP**	**Actual Output**	**Error/Defect**
TC003	-100.00	40.00	Valid	Vehicle moves to the location (-100.00, -40.00)	None

TC004	-100.00	40.00	Valid	Error: "Stuck in endless loop around the location (-100.00, -40.00)"	Erratic behaviour of car Around Destination

5.7.3 System Shuts Down After Reaching First Destination

This test case focus on the behaviour of the system when vehicle reaches a destination and the system shuts down instead of taking the next destination.

Table 5.4: White Box TC3 For System Shuts Down After Reaching First Destination

	Input Variables				
Test ID	**x**	**y**	**ECP**	**Actual Output**	**Error/Defect**
TC005	-100.00	40.00	Valid	System shuts down after reaching destination (-100.00, -40.00)	System does not take next destination, shuts down

5.7.4 System Crashes Due to Invalid Input Types for Coordinates

These test cases focus on invalid input types for coordinates which results in crashing/shutting down the system.

Table 5.5: White Box TC4 For System Crashes

	Input Variables				
Test ID	**x**	**y**	**ECP**	**Actual Output**	**Error/Defect**
TC006	"abc"	"def"	Invalid	Error	System crashes when given string inputs
TC007	"@#$"	"%^&"	Invalid	Error	System crashes when given

					special characters

5.7.5 Spawn Point for Vehicle

These test cases focus on validating the error handling of the system when given invalid input variables for spawn points, including None values, excessively large coordinates, and non-integer inputs.

Table 5.6: White Box TC5 For Spawning Points

	Input Variables				
Test ID	**x**	**y**	**ECP**	**Actual Output**	**Error/Defect**
TC008	None	-133.808	Invalid	Error	Spawn point with None x coordinate causes failure
TC009	-2.0230992692528655	None	Invalid	Error	Spawn point with None y coordinate causes failure
TC010	-2.0230992692528655	-133.808	Invalid	Error	Invalid actor type causes service call failure
TC011	999999999999999999	999999999999999999	Invalid	Error	Large positive x,y coordinate causes service call failure
TC012	“ ”	sdsd	Invalid	Error	Empty or non-integer causes failure

5.7.6 Steering Control

These test cases focus on validating the robot's behavior is either as expected under these conditions or not.

Equivalence Class Partitioning (ECP) for target_linear_speed and target_angular_speed

- Valid Classes
  - Positive integers only
- Invalid Classes
  - Negative Integers
  - None
  - String

Table 5.7: White Box TC6 For Steering Control

	Input Variables				
Test ID	**target_linear_speed**	**target_angular_speed**	**ECP**	**Actual Output**	**Error/Defect**
TC013	-1.0	0.0	Invalid	Robot moves backward	Negative linear speed does not stop robot moving backward
TC014	0.5	None	Invalid	Robot turns in unpredictable motion	Missing angular speed does led to unpredictable turns

5.7.7 previous_destination is not initialized or updated correctly

Equivalence Class Partitioning (ECP) for previous_destination

- Valid Classes
  - $\in Z$
- Invalid Classes
  - $\notin Z$
  - Empty

Table 5.8: White Box TC7 For Destination

	Input Variables			
Test ID	previous_destination	**ECP**	**Actual Output**	**Error/Defect**
TC015	None	Invalid	Error	System crash

5.7.8 Jerkiness

This test case focuses on observing the vehicle's behavior for jerkiness and sudden movements on tight curves.

Table 5.9: White Box TC8 For Jerkiness

	Input Variables				
Test ID	**x**	**y**	**ECP**	**Actual Output**	**Error/Defect**
TC016	-150.0	45.0	Valid	Jerkiness (Sudden Movements) especially on curves	Vehicle exhibits significant jerky motion on tight curves

5.7.9 PID Controllers to perform longitudinal control

Equivalence Class Partitioning (ECP) for target_speed and waypoint

- Valid Classes
  - target_speed > 0
  - waypoint $\in$ Z
- Invalid Classes
  - target_speed <= 0
  - waypoint $\notin$ Z

Table 5.10: White Box TC9.1 For PID Controller

	Input Variables			
Test ID	target_speed	**ECP**	**Actual Output**	**Error/Defect**
TC017	0	Invalid	Vehicle oscillates/does not stop	Improper handling of zero target speed

Table 5.11: White Box TC9.2 For PID Controller

	Input Variables			
Test ID	waypoint	**ECP**	**Actual Output**	**Error/Defect**
TC018	None	Invalid	Vehicle does not steer or crashes randomly	None waypoint not handled properly

5.7.10 Ackermann Steering Model

This test case focuses on testing for ZeroDivisionError when calculating the turning radius with a zero inner wheel angle.

Equivalence Class Partitioning (ECP) for wheel_base and inner_wheel_angle ∈ R

- Valid Classes
  - wheel_base > 0
  - inner_wheel_angle ∈ R
- Invalid Classes

→ wheel_base ≤ 0, inner_wheel_angle: ∅

Table 5.12: White Box TC10 For Ackermann Steering Model

	Input Variables				
Test ID	**wheel_base**	**inner_wheel_angle**	**ECP**	**Actual Output**	**Error/Defect**
TC019	-2	0.5	Invalid	ZeroDivisionError	Given inner_wheel_angle = 0, the calculation for the turning radius results in a division by zero. This will cause the program to raise a ZeroDivisionError. Therefore, the actual output in this case is an error rather than a valid pair of steering angles.

5.7.11 Spawning the vehicle

This test case focuses on verifying the system's behavior when given a valid vehicle name, ensuring that the success flag accurately reflects whether the vehicle was actually spawned.

Equivalence Class Partitioning (ECP) for vehicle_name

- Valid Classes
  - vehicle_name: String
- Invalid Classes
  - vehicle_name: None

Table 5.13: White Box TC11 For Spawning Vehicle

	Input Variables			
Test ID	vehicle_name	**ECP**	**Actual Output**	**Error/Defect**
TC020	Car1	Valid	True	The success flag is always set to True without verifying if the vehicle was actually spawned.

5.8 Summary

In this chapter we have provided a list of components and libraries that we have used in our project for better user experience. We have mentioned Work breakdown structure WBS and Control flow diagram. We have also mentioned tools and IDEs and best practices and coding standards of software engineering.

Chapter 6:
System Testing

Chapter 6: System Testing

This chapter presents a comprehensive overview of the system-level testing procedures conducted for the autonomous vehicle software developed in this project. The primary objective of system testing is to verify the end-to-end functionality, robustness, and safety of the autonomous system in real-world scenarios. Our approach ensures that the software components function cohesively, meeting both user requirements and safety standards for personal autonomous vehicle systems.

6.1 Objectives of System Testing

The objectives of system testing in this project are:

6.1.1 Validate Functional Requirements: Ensure that all components work together to meet specified functionalities, such as navigation, obstacle avoidance, and traffic light recognition.

6.1.2 Safety and Reliability Testing: Test the system's response to hazardous situations, such as obstacle detection failures or adverse weather conditions.

6.1.3 Performance and Environmental Testing: Assess the system's capability under various environmental conditions to ensure reliability in practical usage.

6.1.4 Simulation Integration Testing: Validate the interaction between ROS Noetic and the CARLA simulator, ensuring accurate data exchange and feedback.

6.2 System Testing Methodology

We adopted a rigorous, systematic testing approach tailored to the specifics of our autonomous vehicle software. Given the complexity and safety-critical nature of autonomous systems, we utilized a **black-box testing methodology**. This approach focuses on evaluating the system's external behavior without requiring knowledge of its internal workings, ensuring that the software meets the specified requirements from an end-user perspective. The methodology involved creating realistic test scenarios using the CARLA simulator, setting up test cases for each subsystem, and observing the interactions between components.

6.3 Functional System Test Cases

This section documents the functional system test cases, focusing on core functionalities like navigation, traffic light detection, and obstacle avoidance.

6.3.1 Path Planning

Equivalence Class Partitioning (ECP):

- **Valid Classes**:
    - The destination is selected from the provided options.
    - The destination is entered manually and is valid (x is integer, y is integer).
- **Invalid Classes**:
    - The destination is selected but is not available (e.g., out of service area).
    - The destination coordinates are entered manually but are invalid (e.g., incorrect format, non-existent location).

Test Case:

Table 6.1: Black Box TC1 For Path Planning

Scenario	Input Value	ECP	Expected Output
User enters the destination from given destinations	Home	Valid	Vehicle navigates successfully to the entered destination.
User enters the destination from given destinations	x = -60.000000 y = 60.000000	Valis	Vehicle navigates successfully to the entered destination.

6.3.2 Path Following

Equivalence Class Partitioning (ECP):

- **Valid Classes**:
    - The vehicle's velocity and acceleration parameters are within normal operational ranges. i.e. <120 km/h

- **Invalid Classes**:
  - The vehicle's velocity or acceleration parameters are abnormal or invalid. i.e. = 120km/h

Test Cases:

Table 6.2: Black Box TC2 For Path Following

Scenario	Input Value	ECP	Expected Output
Normal Velocity Parameters	Velocity = 200 km/h	Valid	Vehicle accelerates smoothly within limits and maintains a steady trajectory

6.3.3 Vehicle Control

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - The vehicle's speed is within the normal operational range (i.e. 0 km/h to maximum speed limit).
  - The throttle position is within the normal operational range (i.e. 0% to 100%).
- **Invalid Classes:**
  - The vehicle's speed parameters are abnormal or invalid (i.e. speed exceeding maximum permissible limit).
  - The throttle position is abnormal or invalid (i.e. throttle position exceeding 100%).

Test Cases:

Table 6.3: Black Box TC3 For Vehicle Control

Scenarios	Input Value	ECP	Expected Output
Normal Speed	Speed = 50 km/h	Valid	Vehicle operates smoothly at the given speed.
Normal Throttle Position	Throttle = 50%	Valid	Vehicle accelerates normally without issues.

6.3.4 Vehicle Control

Equivalence Class Partitioning (ECP):

- **Valid Classes:**

Normal Steering: Steering angle within operational range

  - -90° to 90° latitude, -180° to 180° longitude

- **Invalid Classes:**

Abnormal Steering: Steering angle outside operational range (< -30° or > +30°)

Test Cases:

Table 6.4: Black Box TC4 For Vehicle Control

Scenarios	Input Value	ECP	Expected Output
Normal Orientation	Roll = 10° Pitch = 5°	Valid	Vehicle maintains stable orientation.
Acceptable Steering Angle	Angle = 15°	Valid	Vehicle steers appropriately without error.

6.3.5 Vehicle Control

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Speed: 0 km/h ≤ Speed ≤ 120 km/h
  - Distance: 2 meters ≤ Distance ≤100 meters
  - Throttle Adjustment: 0 % ≤ Throttle ≤ 80 %
  - Brake Application: 0 % ≤ Braking Force ≤ 100 %
- **Invalid Classes:**
  - Speed: > 120 km/h
  - Distance: Distance >100 meters
  - Throttle Adjustment: < 0 % or Throttle > 80 %
  - Brake Application: < 0 % or Braking Force > 100 %

Test Cases:

Table 6.5: Black Box TC5 For Vehicle Control

Scenarios	Input Value	ECP	Expected Output
Normal Speed	Speed = 100 km/h	Valid	Vehicle operates efficiently within speed limits.
Safe distance	Distance = 50 meters	Valid	Vehicle maintains safe distance without issue.
Acceptable Throttle	Throttle = 60%	Valid	Vehicle accelerates smoothly as expected.
Normal Brake Application	Force = 75%	Valid	Vehicle decelerates smoothly without issues.

6.3.6 Vehicle Control

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
    - Lateral Position: -1.0 meters ≤ Lateral Position ≤ 1.0 meters
    - Steering Adjustment: -30° ≤ Steering Angle ≤30°
- **Invalid Classes:**
    - Lateral Position: Lateral Position > 1.0 meters
    - Steering Adjustment: Steering Angle > 30°

Test Cases:

Table 6.6: Black Box TC6 For Vehicle Control

Scenarios	Input Value	ECP	Expected Output
Valid Lateral Position	Lateral Position = 0.5 meters	Valid	Vehicle maintains intended lane position.
Normal Steering Adjustment	Angle = 15°	Valid	Vehicle steers correctly without issue.

6.3.7 Localization Module Test Cases

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
    - Sensor data (GPS, IMU, LIDAR) within acceptable ranges.
    - GPS accuracy ≤ 5 meters.
    - IMU data drift ≤ 2 degrees.
    - LIDAR scan range ≥ 50 meters.
- **Invalid Classes:**
    - Sensor data outside acceptable ranges.
    - GPS accuracy > 5 meters.

- IMU data drift > 2 degrees.
- LIDAR scan range < 50 meters.

Test Cases:

Table 6.7: Black Box TC7 For Localization

Scenario	Input Value	ECP	Expected Output
Accurate GPS Signal	GPS Accuracy = 3 meters	Valid	Vehicle update's location correctly.
Valid IMU Data	IMU Drift = 1 degree	Valid	Vehicle continues to maintain accurate positioning.
Adequate LIDAR Scan Range	LIDAR Range = 60 meters	Valid	Vehicle successfully detects surroundings.

6.3.8 Obstacle Detection and Avoidance Module Test Cases

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Obstacle detected within sensor range.
  - LIDAR detection distance ≤ 100 meters.
  - Obstacle size ≥ 0.5 meters.
  - Obstacle-free zone.
  - No objects detected within 100 meters.
- **Invalid Classes:**
  - Sensor fails to detect within expected range.
  - LIDAR detection distance > 100 meters for a detected obstacle.
  - Obstacle size < 0.5 meters considered noise.

Test Cases:

Table 6.8: Black Box TC8 For Obstacle Detection and Avoidance

Scenario	Input Value	ECP	Expected Output
No Obstacle Detected	LIDAR Detection Distance = 150 meters	Valid	Vehicle proceeds without any detection issues.
Valid Obstacle Detected	LIDAR Detection Distance = 50 meters	Valid	Vehicle transitions to classify detected obstacle.
Clear Path Ahead	LIDAR Detection Distance = 80 meters	Valid	Vehicle successfully computes navigation path.

6.3.9 Traffic Light Detection Module Test Cases

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Traffic light detected and state correctly identified.
  - Distance to traffic light $\leq$ 50 meters.
  - Recognition confidence $\geq$ 80%.
- **Invalid Classes:**
  - Traffic light detection errors or low recognition confidence.
  - Distance to traffic light > 50 meters.
  - Recognition confidence < 80%.

Test Cases:

Table 6.9: Black Box TC9 For Traffic Light Detection

Scenario	Input Value	ECP	Expected Output
Traffic Light Detected	Detection Distance = 30 meters	Valid	Vehicle identifies traffic light state correctly.
Traffic Light Detected, High Confidence	Recognition Confidence = 90%	Valid	Transition to Recognize State
Low Confidence in Recognition	Recognition Confidence = 70%	Invalid	Re-scan for traffic light state
Traffic Light at Threshold	Detection Distance = 50 meters	Valid	Proceed with state recognition (Red/Yellow/Green)

6.3.10 Functional Requirement Testing

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Destination coordinates within defined boundaries and obstacles.
  - Setpoints defined and within ±0.5 meters of the planned path.
  - Traffic light present and recognized properly.
  - Obstacle within detection range and accurately identified.
- **Invalid Classes:**
  - Destination coordinates outside operational boundaries or near non-navigable areas.
  - Setpoints defined with deviations greater than ±0.5 meters from the planned path.

- No traffic light present or misidentified state (e.g., recognition error).
- Obstacle detected outside the 5-meter range or misclassified as safe.

Test Cases:

Table 6.10: Black Box TC10

Scenario	Input/Conditions	ECP	Expected Outcome
Path planning to destination	Input destination coordinates	Valid	Vehicle plans a feasible route considering obstacles and road boundaries
Path following accuracy	Input setpoints for route	Valid	Vehicle follows the planned path accurately within ±0.5 meters deviation
Traffic light detection	Approach intersection with traffic light	Valid	Vehicle detects traffic light state and acts accordingly (stop/go)
Obstacle detection and avoidance	Obstacle within 5 meters ahead	Valid	Vehicle decelerates and navigates around obstacle safely

6.4 Safety and Hazard Response Testing

Testing the system's response to hazardous scenarios ensures that the vehicle can handle safety-critical situations without compromising passenger safety.

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Sensors functioning normally, detection within acceptable limits
  - Rain simulation executed without sensor errors.
  - GPS operational, providing accurate positioning data.
  - Obstacles detected within the specified range.
- **Invalid Classes:**
  - Sensor failure not triggering safe mode or alerts.

- Vehicle fails to adjust speed or braking distance during heavy rain.
- GPS failure leading to a localization error without fallback.
- Emergency braking not performed within 1 meter or failure to detect obstacle.

Test Cases:

Table 6.11: Black Box TC11 For Safety and Hazards Responses

Scenario	Input/Conditions	ECP	Expected Outcome
Obstacle detection failure	Simulated sensor failure	Valid	Vehicle enters safe mode, slows down, and alerts user
Adverse weather (rain simulation)	Heavy rain simulation in CARLA	Valid	Vehicle adjusts speed, increases braking distance
GPS signal loss	Disable GPS in CARLA	Valid	System initiates localization using IMU and LIDAR data
Emergency braking	Immediate obstacle within 2 meters	Valid	Vehicle performs emergency brake within 1 meter distance

6.5 Performance Testing

Performance testing evaluates the system's responsiveness, computational efficiency, and resource usage to ensure it meets operational requirements.

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - System response time meets specified performance criteria.
  - System resource usage remains under limits during full load.
  - Localization accuracy is maintained within ±2 meters.
- **Invalid Classes:**
  - Response time exceeds 0.5 seconds for obstacle detection.
  - CPU or memory usage exceeds acceptable limits (>80%).

- Localization accuracy exceeds ±2 meters from input coordinates

Test Cases:

Table 6.12: Black Box TC12 For Performance Testing

Scenario	Input/Conditions	ECP	Expected Outcome
Response time for obstacle detection	Moving obstacle appears within range	Valid	System detects and begins avoidance maneuver within 0.5 seconds
Computational load	Full system simulation with all modules active	Valid	CPU and memory usage within acceptable limits (<80%)
Localization accuracy	Test using varied GPS coordinates	Valid	Vehicle stays within ±2 meters accuracy to input coordinates

6.6 Environmental Testing

Environmental testing assesses the system's reliability and response under various simulated weather conditions, including rain, fog, and low visibility.

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Vehicle responds appropriately to fog conditions.
  - Vehicle adapts correctly to night driving conditions.
  - Vehicle maintains safe operation during heavy rain.
- **Invalid Classes:**
  - Vehicle fails to slow down or activate fog lights in fog.
  - Vehicle does not adjust to reduced visibility.
  - Vehicle does not adjust speed or distance during heavy rain.

Test Cases:

Table 6.13: Black Box TC13 For Environmental Testing

Scenario	Input/Condition	ECP	Expected Outcome
Fog simulation	High-density fog in CARLA	Valid	Vehicle slows to maintain safe distance, activates fog lights
Night-time driving	Night mode with reduced lighting	Valid	Vehicle adjusts to reduced visibility, activates headlights
Heavy rain	Rain simulation	Valid	Vehicle adjusts speed, maintains safe distance from other objects

6.7 Simulation Integration Testing

Testing integration between CARLA and ROS is critical for verifying that the data exchange is accurate and that system responses reflect real-time simulation inputs.

Equivalence Class Partitioning (ECP):

- **Valid Classes:**
  - Accurate and timely data exchange occurs between CARLA and ROS.
  - Command successfully transmitted and executed.
  - Continuous feedback allows for effective vehicle control adjustments.
- **Invalid Classes:**
  - Data exchange delayed or inaccurate.
  - Command transmission fails to stop the vehicle.
  - Feedback loop not resulting in smooth control transitions.

Test Cases:

Table 6.14: Black Box TC14 For Simulation Integration

Scenario	Input/Conditions	ECP	Expected Outcome
CARLA-ROS data exchange	Position and velocity data updates	Valid	ROS receives accurate, real-time data from CARLA without delay
Action command transmission	Send stop command to CARLA vehicle	Valid	CARLA vehicle stops immediately on command
Feedback loop validation	Continuous commands based on feedback	Valid	Smooth transitions and adjustments in vehicle control

6.8 Summary and Observations

System testing revealed that the autonomous vehicle software met its functional and safety requirements across various scenarios. However, adjustments may be necessary in the areas of environmental robustness and sensor failure handling to further enhance safety in unpredictable conditions.

Chapter 7:
Conclusion and Outlook

Chapter 7: Conclusion and Outlook

7.1 Introduction

This chapter concludes the development of an embedded software system for autonomous navigation within a simulated personal vehicle environment. The system was designed to ensure safe and efficient travel from a designated source to a destination, focusing on path planning, obstacle detection, and traffic light response. The following sections outline the specific achievements, challenges, and recommendations for enhancing the system's embedded software capabilities in future implementations.

7.2 Achievements and Improvements

The embedded software system accomplished its core objectives, successfully implementing key navigation features crucial for autonomous personal vehicles. Key achievements include:

7.2.1 Integration of Embedded Software Technologies: The system seamlessly integrated CARLA for vehicle simulation, ROS Noetic for robotic operation, and the CARLA-ROS bridge for communication between the two. This combination provided a platform that mimicked real-world scenarios, allowing a detailed assessment of the system's embedded capabilities.

7.2.2 Functional Software Modules in a personal Vehicle Context: The software includes specialized modules for critical navigation tasks tailored to personal vehicle requirements, including Path Planning, Path Following, Obstacle Detection and Avoidance, and Traffic Light Recognition. These modules were specifically configured to prioritize personal safety, reliable state transitions, and responsive control, fundamental to any embedded system in a personal vehicle.

7.2.3 Simulation Testing for Real-Time Responsiveness: Extensive testing in CARLA allowed the system to demonstrate real-time response in critical situations, such as rapid deceleration for obstacles or timely stopping at traffic lights. The simulator provided a controlled environment to refine the software's timing constraints and real-time control functions, essential features for embedded systems in automotive safety.

7.2.4 Optimized Code for Embedded System Constraints: To align with typical embedded system requirements, the software was designed with efficient code and resource management in mind. Optimizations in the path-following and obstacle avoidance algorithms reduced processing demands, a significant accomplishment given the limited computational resources common in embedded systems.

7.3 Critical Review

While the embedded software system achieved its primary goals, several challenges were encountered:

7.3.1 Limitations of Simulation-Only Testing: While CARLA's simulation is robust, the exclusive reliance on software simulation introduced limitations in verifying hardware performance. Embedded systems in personal vehicles often require hardware testing to validate real-time performance, response to environmental factors, and sensor accuracy—areas not fully replicated in simulation.

7.3.2 High GPU Usage and Computational Demands: The system's heavy reliance on GPU resources, especially during intensive simulation tasks, posed a challenge. For an embedded software system, particularly in automotive applications, high GPU dependency is impractical, as most onboard computing units prioritize low-power, efficient processing. Optimizing the software for lower GPU requirements or identifying alternatives to GPU-heavy processes would be crucial for a viable real-world deployment.

7.3.3 Traffic Light Detection Limitations Without AI: The traffic light module relied on rule-based methods rather than machine learning, restricting its adaptability. An embedded personal vehicle system may benefit from machine learning models that provide more accurate and context-aware recognition, especially in varied or ambiguous lighting conditions.

7.4 Future Recommendations/Outlook

Future advancements can build on this embedded software system by addressing its current limitations and expanding its capabilities for real-world application. Recommendations include:

7.4.1 Transition to Real-World Hardware Testing: Integrating physical hardware components (such as LIDAR, cameras, and sensors) into the testing process would be invaluable for refining the system's embedded software behavior under real-world conditions. This step would help verify the system's robustness and validate its real-time performance, essential in an embedded software context for personal vehicles.

7.4.2 Machine Learning Integration for Enhanced Detection: Implementing machine learning in the traffic light and obstacle detection modules could improve accuracy and adaptability, essential for handling complex, dynamic environments in personal vehicles. Machine learning models can provide more reliable context awareness, which is critical for personal safety in real-time embedded systems.

7.4.3 Optimization for Memory Efficiency and Real-Time Performance: Improving the system's handling of high-volume data streams would enhance memory efficiency and overall real-time performance, making it more suitable for the constraints of embedded automotive systems. Future iterations could focus on data compression techniques, efficient data storage, and optimized sensor fusion strategies to maintain responsiveness within strict memory limits.

7.4.4 Scalability and Extended Scenario Testing: To prepare for broader deployment, the system can be further optimized for scalability. Enhanced scenario testing with larger and more complex datasets would allow verification of system stability across varied conditions, an important factor in embedded systems for personal safety.

7.5 Summary

In conclusion, this project successfully developed an embedded software system tailored for autonomous navigation in a personal vehicle context. Through extensive testing, the system demonstrated reliable path planning, obstacle detection, and traffic light recognition, all critical features for autonomous operation. The system's simulation-based approach provided a foundation for real-time responsiveness, while code optimizations supported the resource efficiency required for embedded applications. Though challenges remain, particularly in real-world testing and adaptability, this project lays the groundwork for further advancements. By incorporating hardware validation, machine learning, and advanced memory management, future work can refine this embedded software system, advancing its potential for autonomous personal vehicles.

References and Bibliography

[1] SAE International. (2023). SAE Standards News: J3016 automated-driving. Retrieved from https://www.sae.org/blog/sae-j3016-update

[2] Nilsson, N. J., & Kanal, L. N. (1969). A generalization of the A algorithm*. AI Memo 84, Stanford University.

[3] Ackermann, R. A. (1906). Die Wissenschaftliche Lenkung des Motors [The Scientific Steering of the Motor]. Zweite Mitteilung. S. Hirzel Verlag, Leipzig.

[4] Lee, D. T., & Schachter, B. J. (1980). Two algorithms for constructing a Delaunay triangulation. International Journal of Computer & Information Sciences, 9(3), 219-242.

[5] ROS-bridge. (2023). CARLA Simulator Documentation. Retrieved from https://carla.readthedocs.io/projects/ros-bridge/en/latest/

[6] CARLA Team. (2021). Integrating CARLA with ROS: A practical guide. CARLA Simulator White Paper.

[7] LaValle, S. M. (2006). Planning Algorithms. Cambridge University Press.

[8] Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010). Practical search techniques in path planning for autonomous driving. Robotics and Autonomous Systems, 58(5), 476-492.

[9] Correll, N., Wise, M., & van de Panne, M. (2016). Analysis of ROS-based platforms for autonomous vehicle applications. Robotics and Autonomous Systems, 75, 552-562.

[10] Hoenig, W., Jadhav, S., & Kolski, S. (2018). ROS-Industrial: Open-source robot operating system meets automation. IEEE Robotics & Automation Magazine, 25(1), 13-21.

[11] Python Software Foundation. (2020). PEP 8 – Style Guide for Python Code. Retrieved from https://peps.python.org/pep-0008/

[12] Python Software Foundation. (2020). PEP 257 – Docstring Conventions. Retrieved from https://peps.python.org/pep-0257/

[13] Python Software Foundation. (2020). Python Documentation. Retrieved from https://docs.python.org/3/

[14] ROS Wiki. (2020). ROS Documentation. Retrieved from http://wiki.ros.org/

[15] ROS Wiki. (2020). rospy Documentation. Retrieved from http://wiki.ros.org/rospy

Appendices

Appendix-A: Software Requirements Specifications (SRS)

Appendix-B: Design Artifact

Appendix-C: Coding Standards/Conventions

Appendix-D: SQA Activities

Appendix-E: Work Breakdown Structure

Appendix-F: Roles & Responsibility Matrix