Computer on Wheels

****

**By:**

**Bilal Rafiq**

**27661**

**Hamza Azhar**

**28595**

**Sardar Mohsin Saghir**

**28016**

**M. Usama Nazir**

**30445**

**Supervised by:**

**Dr. Naveed Ikram**

**Faculty of Computing**

**Riphah International University, Islamabad**

**Fall 2024A 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: 25th June, 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 AV software system** utilizing **machine learning**. Through the integration of path planning and dynamic obstacle avoidance algorithms, the system aims to enhance AVs' capabilities to navigate urban environments with precision and safety. By implementing these algorithms and leveraging low-cost solutions, 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.

**Table of Contents**

[Chapter 1: Introduction 14](#_Toc180593808)

[1.1 Opportunity and Stakeholder 14](#_Toc180593809)

[1.2 Motivations and Challenges 15](#_Toc180593810)

[1.3 Goals and Objectives 15](#_Toc180593811)

[1.4 Solution Overview 16](#_Toc180593812)

[1.5 Report Outline 18](#_Toc180593813)

[1.5.1 Chapter 1 18](#_Toc180593814)

[1.5.2 Chapter 2 18](#_Toc180593815)

[1.5.3 Chapter 3 18](#_Toc180593816)

[1.5.4 Chapter 4 18](#_Toc180593817)

[1.5.5 Chapter 5 18](#_Toc180593818)

[1.5.6 Chapter 6 19](#_Toc180593819)

[Chapter 2: Literature/Market Survey 21](#_Toc180593820)

[2.1 Introduction 21](#_Toc180593821)

[2.2 Literature Review / Technology Overview 21](#_Toc180593822)

[2.3 Brainstorming 24](#_Toc180593823)

[2.4 Existing Systems 25](#_Toc180593824)

[2.5 Summary 26](#_Toc180593825)

[Chapter 3: Requirement Engineering 28](#_Toc180593826)

[3.1 Introduction 28](#_Toc180593827)

[3.2 Problem Scenarios 28](#_Toc180593828)

[3.3 Key Concepts and Terminology 30](#_Toc180593829)

[3.4 Functional Requirements 31](#_Toc180593830)

[3.5 Non-Functional Requirement 41](#_Toc180593831)

[3.6 SQA activity: Defect Identification: Inspection Thought Checklist 43](#_Toc180593832)

[Chapter 4: System Design 49](#_Toc180593833)

[4.1 Introduction 49](#_Toc180593834)

[4.2 Architectural Design 50](#_Toc180593835)

[4.3 Detailed Design 51](#_Toc180593836)

[4.3.1 Use Case Design 51](#_Toc180593837)

[4.3.2 Sequence Diagram 52](#_Toc180593838)

[4.4 SQA activity: State-Based Defect Detection Scenarios 53](#_Toc180593839)

[Chapter 5: Implementation 63](#_Toc180593840)

[5.1 Endeavour 63](#_Toc180593841)

[5.2 Proposed Solution 76](#_Toc180593842)

[5.3 Pseudo codes 76](#_Toc180593843)

[5.4 Components and Libraries 86](#_Toc180593844)

[5.5 IDE, Tools, Technologies and Development Platform 87](#_Toc180593845)

[5.6 Best Practices / Coding Standards 87](#_Toc180593846)

[5.7 Deployment Environment 89](#_Toc180593847)

[5.8 SQA activity: Defect Detection Through White Box Testing 90](#_Toc180593848)

[5.9 Summary 98](#_Toc180593849)

[Chapter 6: Conclusion and Outlook 100](#_Toc180593850)

[6.1 Introduction 100](#_Toc180593851)

[6.2 Achievements and Improvements 100](#_Toc180593852)

[6.3 Critical Review 102](#_Toc180593853)

[6.4 Future Recommendations/Outlook 103](#_Toc180593854)

[6.5 Summary 104](#_Toc180593855)

[References and Bibliography 105](#_Toc180593856)

[Appendices 106](#_Toc180593857)

[Appendix-A: Software Requirements Specifications (SRS) 106](#_Toc180593858)

[Appendix-B: Design Artifact 106](#_Toc180593859)

[Appendix-C: Coding Standards/Conventions 106](#_Toc180593860)

[Appendix-D: SQA Activities 106](#_Toc180593861)

[Appendix-E: Work Breakdown Structure 106](#_Toc180593862)

[Appendix-F: Roles & Responsibility Matrix 106](#_Toc180593863)

# 

**List of Figures**

|  |  |  |
| --- | --- | --- |
| 2.1 | Brainstorming Diagram | 24 |
| 4.1 | Architecture Diagram | 45 |
| 4.2 | Use-case Diagram | 46 |
| 4.15 | Sequence diagram | 71 |
| 5.1 | Proposed Solution | 90 |
| 5.2 | A\* diagram | 91 |
| 5.3 | Ackermann diagram | 95 |
| 5.4 | Cubic Spline diagram | 98 |
| 5.5 | Deployment diagram | 103 |

# 

**List of Tables**

|  |  |  |
| --- | --- | --- |
| 2.1 | Levels of taxonomy | 23 |
| 2.2 | Existing Systems | 25 |
| 3.1 | problem statement 1 | 28 |
| 3.2 | problem statement 2 | 29 |
| 3.3 | problem statement 3 | 29 |
| 3.4 | FR1 | 30 |
| 3.5 | FR2 | 31 |
| 3.6 | FR3 | 32 |
| 3.7 | FR4 | 33 |
| 3.8 | FR5 | 33 |
| 3.9 | FR6 | 34 |
| 3.10 | FR7 | 34 |
| 3.11 | FR8 | 35 |
| 3.12 | FR9 | 36 |
| 3.13 | FR10 | 36 |
| 3.14 | NFR1 | 37 |
| 3.15 | Inspection Table 1 | 37 |
| 3.16 | Inspection Table 2 | 38 |
| 3.17 | Inspection Table 3 | 38 |
| 3.18 | Inspection Table 4 | 39 |
| 3.19 | Inspection Table 5 | 39 |
| 3.20 | Inspection Table 6 | 40 |
| 3.21 | Inspection Table 7 | 40 |
| 3.22 | Inspection Table 8 | 41 |
| 3.23 | Inspection Table 9 | 41 |
| 3.24 | Inspection Table 10 | 42 |
| 4.13 | Scenario Based TC1 | 72 |
| 4.14 | Scenario Based TC2 | 73 |
| 4.15 | Scenario Based TC3 | 73 |
| 4.16 | Scenario Based TC4 | 74 |
| 4.17 | Scenario Based TC5 | 75 |
| 4.18 | Scenario Based TC6 | 75 |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 5.1 | Responsibilities Assignment Matrix | 82 |
| 5.2 | White Box TC1 | 104 |
| 5.3 | White Box TC2 | 105 |
| 5.4 | White Box TC3 | 106 |
| 5.5 | White Box TC4 | 106 |
| 5.6 | White Box TC5 | 107 |
| 5.7 | White Box TC6 | 108 |
| 5.8 | White Box TC7 | 109 |
| 5.9 | White Box TC8 | 109 |
| 5.10 | White Box TC9.1 | 110 |
| 5.11 | White Box TC9.2 | 110 |
| 5.12 | White Box TC10 | 111 |
| 5.13 | White Box TC11 | 112 |

**Chapter 1:**

**Introduction**

# Chapter 1: Introduction

Computer on Wheels, is an **embedded software system** for a car that can drive themself with minimal intervention. The system is capable of autonomously controlling the vehicle's movement, including **throttle control**, **acceleration, braking, and steering**. Furthermore, it incorporates obstacle detection capabilities to detect and respond to obstacles ensuring safe navigation. Moreover, path planning algorithms are used to determine optimal routes from point A to point B. By leveraging state-of-the-art technologies such as the **CARLA (Car Learning to Act) simulator, CARLA-ROS bridge, and ROS (Robot Operating System)**, our project endeavours to create an embedded software solution capable of empowering autonomous vehicles to navigate urban environment with confidence.

## 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. **Stakeholders**
* Driver
* Passengers

## 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.

## 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.

## Solution Overview

Our solution includes developing an embedded autonomous vehicle software utilizing cutting-edge technologies like the CARLA simulator, ROS Noetic, CARLA-ROS bridge, and rospy. This software will enable vehicles to autonomously navigate complex environments by implementing key functionalities:

* Path Planning
* Path Following
* Obstacle Detection
* Obstacle Avoidance

Also, ensuring safety and precision while minimizing accidents caused by human error. Additionally, our solution prioritizes accessibility, aiming to provide mobility for individuals with disabilities and the elderly. Through rigorous development and testing, we endeavour to deliver a reliable and efficient solution that revolutionizes autonomous vehicle navigation.

* + 1. **Project Scope**

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

* + - 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. **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. **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. **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. **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.

## 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

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

### Chapter 2

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

### Chapter 3

This chapter 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

This chapter 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

This chapter 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

This chapter includes the conclusion of our project, along with a brief outlook

Chapter 2:

**Literature/Market Survey**

# Chapter 2: Literature/Market Survey

This chapter aims to provide an overview of the current state of autonomous vehicles, including existing developments and ongoing testing. It will explore the origins of autonomous vehicles and the regulatory bodies responsible for establishing rules. Furthermore, it will identify prominent market participants involved in advancing autonomous vehicle technologies.

## Introduction

The concept of autonomous vehicles is not fresh in the automotive industry. Companies such as Tesla, General Motors, BMW, Mercedes, Honda, KIA, Toyota, among others, have been actively involved in this field. While many have developed vehicles equipped with level 2 and level 3 autonomous systems, not all have released them 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).

## Literature Review / Technology Overview

The concept of autonomous vehicles traces back to 1918, with early attempts in the 1920s. General Motors was among the pioneers, showcasing autonomous vehicle concepts at exhibitions. The research and development efforts for autonomous vehicles gained momentum with initiatives like General Motors and Radio Corporation of America Sarnoff Laboratory's collaboration. Notably, the Defense Advanced Research Projects Agency (DARPA) Grand Challenges Program in 2004 accelerated autonomous vehicle research in the US.

Today, the global autonomous vehicle market boasts 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 autonomous vehicle development in 2006 and unveiled a fully autonomous test vehicle in 2017, though commercially available self-driving cars from Volvo are still pending.
* **Waymo** (Google's subsidiary): Made significant progress, logging millions of autonomous driving miles. Currently offers limited commercial self-driving ride-hailing services in specific locations.
* **Tesla**: Announced plans for self-driving features in their cars in 2014, promoting them as standard. Notably, Tesla's Autopilot is a driver-assistance system rather than fully autonomous, and has faced safety criticisms.

AVs operate themselves and execute necessary functions without human intervention. This is achieved through their ability to sense their surroundings using advanced technologies such as artificial intelligence (AI) software, light detection and ranging (LiDAR), radio detection and ranging (RADAR), and cameras. These sensors enable the vehicle to form an active 3D map of its environment, allowing it to navigate safely and efficiently.

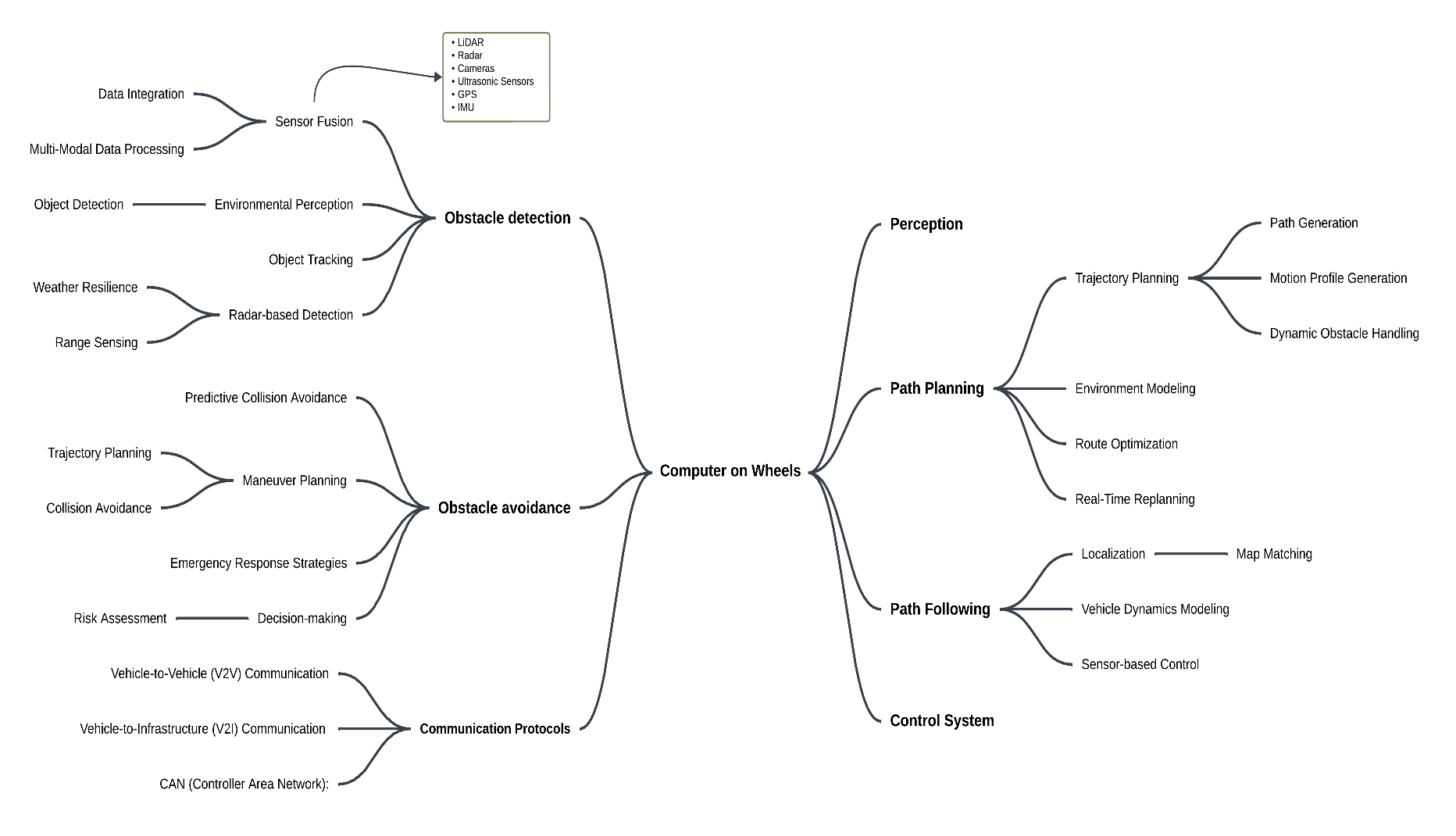
* + 1. **Levels of Autonomous Vehicles**

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. |

## Brainstorming



*Figure 2.1: Brainstorming Diagram.*

## Existing Systems

*Table 2.2: Existing Systems*

|  |  |  |
| --- | --- | --- |
| **Company** | **Target Level** | **Key Features** |
| **Tesla, Ford, Toyota** | Level 2 (Autopilot) | * Lane keeping * automatic emergency braking * traffic light and stop sign recognition * highway driving assist * self-parking (Level 2) * Navigate on Autopilot |
| **BMW, Nissan** | Level 2 | * Adaptive cruise control with stop-and-go * lane departure warning * lane change assist |
| **Honda, Mercedes-Benz** | Level 3 (conditional) | * Hands-free driving at up to 60 km/h on specific highways * automatic lane changes * traffic jam assist * emergency stop assists |
| **Way-mo** | Level 4 | * LiDAR-based system for navigating complex * extensive real-world testing * millions of miles driven |
| **Cruise** | Level 5 | * Fully autonomous robo-taxi |

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.

## 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 Engineering

## 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.

## Problem Scenarios

*Table 3.1: problem statement 1*

|  |  |
| --- | --- |
| **Problem Statement # 1: Hazards caused by human errors** | |
| The problem of | Hazards caused by human errors |
| Affects | Passengers, drivers, and pedestrians |
| The result of which | More injuries/deaths, Damage to property and Emotional stress |
| Benefits of | Mitigation of human errors thus reduces accidents and fatalities |

*Table 3.2: problem statement 2*

|  |  |
| --- | --- |
| **Problem Statement # 2: Limited driver productivity** | |
| The problem of | Driver's unproductiveness while driving |
| Affects | Drivers |
| The result of which | It decreased efficiency |
| Benefits of | Increased productivity by doing other important tasks |

*Table 3.3: problem statement 3*

|  |  |
| --- | --- |
| **Problem Statement # 3: Poor mobility for the aged, disabled, and children.** | |
| The problem of | Transportation reliance for vulnerable groups such as children, the elderly, and the disabled |
| Affects | Passenger. |
| The result of which | Increased risks to the health vulnerable individuals, potentially leading to delayed or insufficient medical care in emergency situations |
| Benefits of | Accessibility, independence and reduced burden on caregivers |

## ****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.

* + 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**.

* + 1. **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.

* + 1. **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.

* + 1. **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.

* + 1. **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.

## Functional Requirements

* + 1. **Vehicle Control:**

*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. |

* + 1. **Path Planning:**

*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 |

* + 1. **Path Following:**

*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/s², 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. |

* + 1. **Sensor Integration:**

*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 |

* + 1. **Trajectory Planning:**

*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. |

* + 1. **Obstacle Detection:**

*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). |

* + 1. **Obstacle Avoidance:**

*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. |

* + 1. **Destination Arrival:**

*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/s² for passenger safety and comfort. |

* + 1. **User Inputs:**

*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. |

* + 1. **System Integration:**

*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. |

* + 1. **Traffic Light Module:**

*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). |

## Non-Functional Requirement

*Table 3.15: NFR1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Non- Functional Requirement** | **ID** | **Description** | **Subfactor** |
| **1** | **Safety Requirement** | **1.1** | 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. |

*Table 3.16: NFR**2*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Non- Functional Requirement** | **ID** | **Description** | **Subfactor** |
| **2** | **Scalability Requirement** | **2.1** | The ROS-based architecture shall support adding new sensors (e.g., radar, additional cameras) without significant changes to the core modules. | **System Expandability** |

*Table 3.17: NFR3*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Non- Functional Requirement** | **ID** | **Description** | **Subfactor** |
| **3** | **Modularity Requirement** | **3.1** | 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** |
| **3** | **Modularity Requirement** | **3.2** | Each ROS node shall handle a specific task (e.g., path planning, obstacle detection) and communicate through well-defined ROS topics. | **Task Separation** |

## 

## SQA activity: Defect Identification: Inspection Thought Checklist

* + 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*

|  |  |  |
| --- | --- | --- |
| **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. |

* + 1. **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*

|  |  |  |
| --- | --- | --- |
| **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. |

* + 1. **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*

|  |  |  |
| --- | --- | --- |
| **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. |

* + 1. **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/s², ensuring a smooth ride for passengers.

*Table 3.21: Inspection Table 4*

|  |  |  |
| --- | --- | --- |
| **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. |

* + 1. **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*

|  |  |  |
| --- | --- | --- |
| **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. |

* + 1. **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*

|  |  |  |
| --- | --- | --- |
| **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. |

* + 1. **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*

|  |  |  |
| --- | --- | --- |
| **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. |

* + 1. **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*

|  |  |  |
| --- | --- | --- |
| **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. |

* + 1. **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*

|  |  |  |
| --- | --- | --- |
| **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. |

* + 1. **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/s² for passenger safety and comfort.

*Table 3.27: Inspection Table 10*

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

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

## Introduction

The software system is founded upon the architecture and framework of ROS 1, with outcomes visualized through the Carla Simulator. Facilitating seamless communication between Carla and ROS Noetic, we employ the ROS bridge as our interface for data retrieval and command transmission. Functioning as a crucial intermediary, the ROS bridge facilitates integration between ROS programs and non-ROS environments.

## Architectural Design

*Figure 4.1: Architecture Diagram*

## Detailed Design

### Use Case Design

*Figure 4.2: Use-case Diagram.*

### Sequence Diagram

*Figure 4.15: Sequence diagram*

## SQA activity: State-Based Defect Detection Scenarios

* + 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*

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

* + 1. **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*

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

* + 1. **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*

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case** | **Input Value** | **ECP** | **Expected Output** |
| Negative Speed | Speed = -10 km/h | Invalid | Unexpected Error |
| Negative Throttle Position | Throttle = -20% | Invalid | Unexpected Error |

* + 1. **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*

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

* + 1. **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*

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

* + 1. **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*

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

* + 1. **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*

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

* + 1. **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*

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

* + 1. **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*

|  |  |  |  |
| --- | --- | --- | --- |
| **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

## Endeavour

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

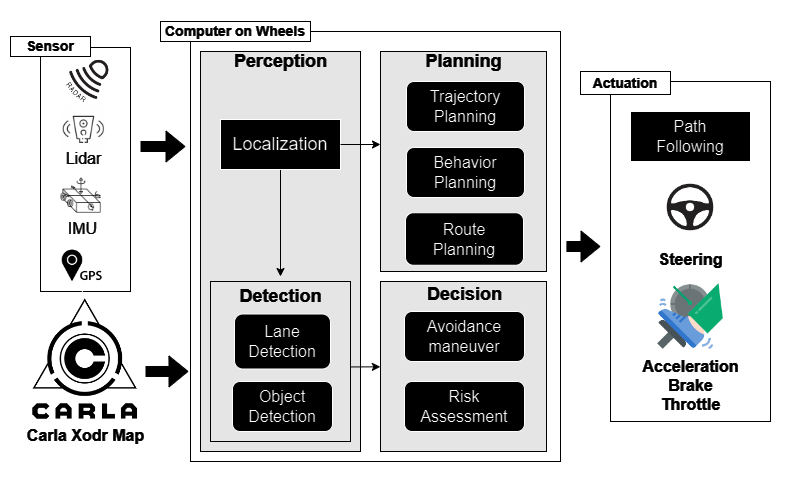
* + 1. **Team**
* Bilal Rafiq
* Hamza Azhar
* Sardar Mohsin Saghir
* Muhammad Usama Nazir
  + 1. **Work Breakdown Structure**

1. **Project Management**
   1. Work Breakdown Structure (WBS)
   2. Roles & Responsibility Matrix
   3. Change Control System
   4. Meeting minutes and Progress report
2. **Reports / Documentation**
   1. Team Members and Project Proposal
   2. Project Proposal Document
      1. Opportunity and Stakeholders
      2. Challenges Goals and Objectives
      3. Solution Overview diagram
      4. Report Outline
   3. Literature / Market Survey
      1. Domain Expert Interview Findings
      2. Questionnaire for Technical Feasibility and Risk Assessment
      3. Brainstorming diagram
      4. Academic Research Review
      5. Gap analysis summary
      6. Technology Landscape
         1. SWOT analysis
      7. Questionnaire for Selecting tools and techniques
      8. Specialization - 4 courses series from Coursera
   4. Requirement Analysis
      1. Problem Scenarios
      2. Requirement Elicitation
      3. Questionnaire for gathering requirements
      4. Functional Requirements
      5. Non-Functional Requirement
      6. Inspection Report
      7. Software requirement specification artifact
   5. System Design
      1. Architecture Diagram
      2. Use Case Diagram
      3. Detail Use Cases
      4. Activity Diagrams
      5. System Sequence Diagram
   6. Implementation
      1. Components and Libraries
   7. Testing and Performance Evaluation
      1. Test Scenarios
   8. Conclusion & Outlook
      1. Future Recommendations
   9. Progress Presentation
      1. Slides outlining project progress
      2. Updated Artifacts of part 1
         1. Appendix-A: Software Requirements Specifications (SRS)
         2. Appendix-B: Design Documents
         3. Appendix-C: Coding Standards/Conventions
         4. Appendix-D: Test Scenarios
         5. Appendix-E: Work Breakdown Structure
         6. Appendix-F: Roles & Responsibility Matrix
      3. Answers to potential questions report
   10. Final Presentation part 2
       1. Comprehensive Slides for presentation
       2. Working software system (Complete)
       3. Updated Artifacts (Complete)
          1. Appendix-A: Software Requirements Specifications (SRS)
          2. Appendix-B: Design Documents
          3. Appendix-C: Coding Standards/Conventions
          4. Appendix-D: Test Scenarios
          5. Appendix-E: Work Breakdown Structure
          6. Appendix-F: Roles & Responsibility Matrix
       4. Final Report
3. **System**
   1. Development Environment
      1. IDE
         1. Visual Studio Code
         2. PyCharm
      2. Version Control
         1. Git Hub
      3. Environment Management
         1. Anaconda Distribution
   2. Simulation Environment Setup
      1. CARLA Simulator
         1. Carlaviz for CARLA Visualization
      2. ROS Noetic Configured
      3. CARLA-ROS Bridge Integrated
      4. Vehicle spawn module
      5. Sensor spawn module
      6. Destroy Vehicle module
   3. Path Planning component
      1. Map Reading module
      2. Graph of Roads
      3. Graph of Lanes
      4. List of Driving Lanes within map
      5. Route Calculation module
      6. Algorithm implementation module
      7. Global route planner module
      8. Axis Translation module
      9. Local route planner module
      10. Environment Analysis module
      11. Trajectory Generation module
      12. Junction handling module
   4. Path Following component
      1. Trajectory Tracking module
      2. Basic agent module
      3. Behaviour agent module
      4. Algorithm implementation module
      5. Controller module
      6. Custom Destination module
   5. Vehicle Control component
      1. Throttle Control module
      2. Braking Control module
      3. Acceleration Control module
      4. Steering Control module
      5. Longitudinal Control module
      6. Lateral Control module
      7. Lane changing module
      8. Jerkiness Control algorithm modules
      9. Rotation and Translation module
   6. Sensor Integration module
      1. IMU integration sub-module
      2. GPS integration sub-module
      3. Radar integration sub-module
      4. Lidar integration sub-module
   7. Obstacle Detection
      1. Sensor Fusion module
         1. Lidar-Radar Fusion sub-module
         2. Multi-sensor Data synchronization sub-module
      2. Sensor Data Processing module
      3. Obstacle Detection module
         1. ML based detection sub-module
      4. Distance Estimation module
      5. Object Classification module
   8. Obstacle Avoidance
      1. Dynamic Obstacle handling module
      2. Static Obstacle handling module
      3. Path Adjustment module
         1. Map based planning sub-module
         2. Graph based planning sub-module
      4. Trajectory Estimation module
      5. Maneuver Planning module
         1. Environmental evaluation sub- module
         2. Lane changes sub-module
         3. Decelerate sub-module
         4. Emergency Stop sub-module
      6. Real-time Response module
      7. Tracking module
         1. Kalman filter sub-module
         2. Particle filter sub-module
4. **Open House**
   1. Event Part 1
      1. Standee Design
      2. Printed Standee
      3. Printed Broachers
      4. Pre-recorded Demo video
   2. Event Part 2
      1. Standee Design
      2. Printed Standee
      3. Printed Broachers
      4. Full Working Software
      5. **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 software based on testing results | 3 | Bilal (C)  Hamza (A/R)  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) |

## 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*

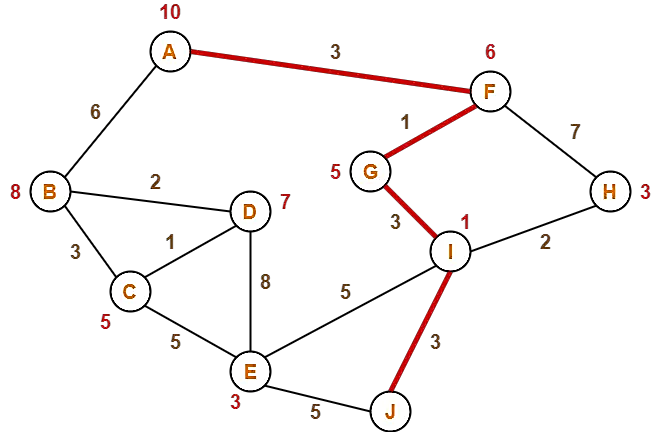
## Pseudo codes

**5.3.1. A\* algorithm:**

The A\* algorithm is a widely used informed search algorithm in computer science and artificial intelligence. It efficiently finds the shortest path from a given start node to a goal node in a weighted graph, where each edge has a non-negative cost. A\* combines elements of **Dijkstra's algorithm and greedy best-first search** by using a heuristic function to guide its search process.

**5.3.1.1 How A\* Works - Mathematical Foundation**

A\* selects nodes to expand based on an evaluation function f(n)=g(n)+h(n):

* g(n): Cost of the path from the start node to node n.
* h(n) Heuristic estimate of the cost from node n to the goal.
* **f(n): Total estimated cost of the path through node n.

*Figure 5.2: A\* diagram*

**5.3.1.2 Pseudo-code**

function A\_star(start, goal, graph):

**// Initialize a priority queue for nodes to be evaluated**

open\_list = PriorityQueue()

**// Add the start node to the open list with a priority of 0**

open\_list.push(start, 0)

**// Dictionary to map nodes to their predecessors**

came\_from = {}

**// Dictionary to map nodes to their cost from the start node**

g\_score = {node: infinity for node in graph.nodes}

**// The cost of the start node is 0**

g\_score[start] = 0

**// Dictionary to map nodes to their estimated total cost (start to goal)**

f\_score = {node: infinity for node in graph.nodes}

**// The estimated total cost for the start node is calculated using the heuristic**

f\_score[start] = heuristic\_cost\_estimate(start, goal)

**// Loop until there are no nodes to evaluate**

while not open\_list.isEmpty():

**// Get the node in the open list with the lowest f\_score value**

current = open\_list.pop()

**// If the current node is the goal, reconstruct the path and return it**

if current == goal:

return reconstruct\_path(came\_from, goal)

**// Loop through each neighbor of the current node**

for neighbor in graph.neighbors(current):

**// Calculate the tentative g\_score for the neighbor**

tentative\_g\_score = g\_score[current] + cost(current, neighbor)

**// If the tentative g\_score is better than the known g\_score**

if tentative\_g\_score < g\_score[neighbor]:

**// Set the predecessor of the neighbor to the current node**

came\_from[neighbor] = current

**// Update the g\_score of the neighbor**

g\_score[neighbor] = tentative\_g\_score

**// Update the f\_score of the neighbor**

f\_score[neighbor] = g\_score[neighbor] + heuristic\_cost\_estimate(neighbor, goal)

**// If the neighbor is not in the open list, add it with its f\_score**

if neighbor not in open\_list:

open\_list.push(neighbor, f\_score[neighbor])

**// If the goal is not reached, return failure**

return "No path found"

function reconstruct\_path(came\_from, current):

**// Initialize the path with the goal node**

total\_path = [current]

**// Loop to construct the path by tracing predecessors**

while current in came\_from.keys():

**// Move to the predecessor node**

current = came\_from[current]

**// Add the node to the path**

total\_path.append(current)

**// Reverse the path to start from the initial node**

return total\_path.reverse()

function heuristic\_cost\_estimate(node, goal):

**// Implement a heuristic function (e.g., Euclidean distance, Manhattan distance)**

return distance(node, goal)

**5.3.2. Ackermann:**

**Ackermann Steering Geometry:** A steering mechanism where all wheels of a vehicle have different turning radii, designed to ensure that all wheels track the same path center, allowing for smoother turns.

* + - 1. **How Ackermann Works - Mathematical Formulation**

The Ackermann steering model uses the following relationships to determine the angles of the wheels based on the desired turning radius and vehicle dimensions:

1. **Turning Radius (R)**: The radius of the circle along which the vehicle is turning.
2. **Wheelbase (L)**: The distance between the front and rear axles of the vehicle.
3. **Track Width (W)**: The distance between the left and right wheels on the same axle.

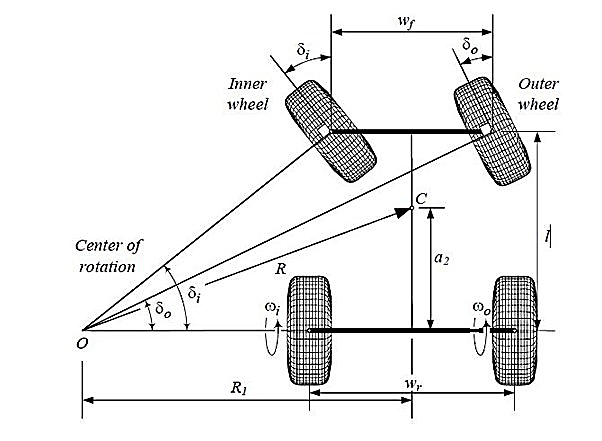
For a vehicle turning with a radius R:

* **Inside Wheel Angle (delta\_i)**: Angle of the inside wheel relative to the vehicle's longitudinal axis.
* **Outside Wheel Angle (delta\_o)**: Angle of the outside wheel relative to the vehicle's longitudinal axis.

The angles delta\_i and delta\_o are calculated using:

* **tan (δi) =**
* **tan (δ0) =**

These equations ensure that when the vehicle turns, the wheels follow paths that intersect at a common turning center, preventing tire scrubbing and maintaining stability

*Figure 5.3: Ackermann diagram*

**5.3.2.2** **Pseudo-code**

class AckermannSteering:

**// Constants and parameters**

L: float **// Wheelbase (distance between front and rear axles)**

W: float **// Track width (distance between left and right wheels on the same axle)**

constructor(L, W):

this.L = L

this.W = W

function calculate\_ackermann\_angles(R):

**// Calculate steering angles for a given turning radius R**

delta\_i = atan(this.L / (R - this.W / 2))

delta\_o = atan(this.L / (R + this.W / 2))

return delta\_i, delta\_o

function move\_vehicle(speed, delta):

**// Simulate movement of the vehicle based on speed and steering angle delta**

**// Assume delta is the angle of the front wheels (steering angle)**

**// Implement vehicle movement simulation here (e.g., update position and orientation)**

**// For simplicity, assume the vehicle moves in a straight line or turns with a fixed radius**

**// Update vehicle position and orientation based on speed and steering angle**

**// Update position (x, y) and orientation (theta) of the vehicle**

x = x + speed \* cos(theta)

y = y + speed \* sin(theta)

theta = theta + speed / this.L \* tan(delta)

**// Normalize theta to keep it within [-pi, pi)**

if theta >= pi:

theta = theta - 2 \* pi

elif theta < -pi:

theta = theta + 2 \* pi

**// Return updated position (x, y) and orientation (theta)**

return x, y, theta

**5.3.3. Cubic Splines:**

Cubic splines are **a type of interpolation technique** used to create smooth curves through a set of given points (called waypoints or control points). They are particularly useful in path planning and smoothing for autonomous vehicles, where it is essential to ensure smooth transitions between waypoints to avoid abrupt changes in direction or speed, which can **lead to jerkiness and discomfort.**

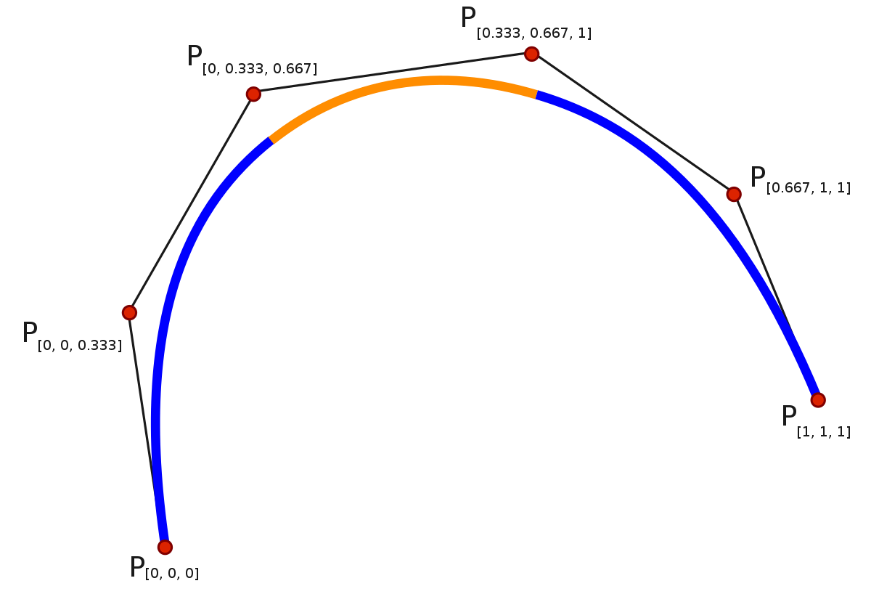
**5.3.3.1 How Cubic Splines Work - Mathematical Formulation**

For a set of n waypoints (xi, yi) a cubic spline constructs n − 1 cubic polynomials, one for each interval between waypoints. Each cubic polynomial Si(x) is defined as:

**Si(x) = ai + bi (x − xi) + ci (x − xi)2 +di (x − xi)3**

The coefficients ai, bi, ci and di are determined such that the spline:

1. Passes through all the waypoints.
2. Has continuous first and second derivatives at each waypoint.
3. (Optionally) has specified boundary conditions at the endpoints.



*Figure 5.4: Cubic Spline diagram*

**5.3.3.2 Pseudo-code**

function cubic\_spline\_path\_smoothing(points):

n = points.length

**// Initialize variables**

A = create\_matrix(n, n) **// Coefficient matrix**

b = create\_vector(n) **// Right-hand side vector**

h = create\_vector(n-1) **// Distances between points**

**// Calculate distances between consecutive points**

for i from 0 to n-2 do:

h[i] = points[i+1].x - points[i].x

**// Populate coefficient matrix A**

A[0][0] = 1

A[n-1][n-1] = 1

for i from 1 to n-2 do:

A[i][i-1] = h[i-1]

A[i][i] = 2 \* (h[i-1] + h[i])

A[i][i+1] = h[i]

**// Populate right-hand side vector b**

for i from 1 to n-2 do:

b[i] = 3 \* ((points[i+1].y - points[i].y) / h[i] - (points[i].y - points[i-1].y) / h[i-1])

**// Solve the system for the second derivatives (coefficients** of the cubic spline)

M = solve\_system(A, b)

**// Generate cubic spline segments**

splines = []

for i from 0 to n-2 do:

a = points[i].y

b = (points[i+1].y - points[i].y) / h[i] - h[i] \* (2 \* M[i] + M[i+1]) / 3

c = M[i]

d = (M[i+1] - M[i]) / (3 \* h[i])

**// Generate points along the cubic spline segment**

for t from 0 to 1 with step size delta\_t do:

y = a + b \* t + c \* t^2 + d \* t^3

splines.append((points[i].x + t \* h[i], y))

return splines

## Components and Libraries

**Components:**

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

**Libraries:**

|  |  |
| --- | --- |
| * rospy | * weakref |
| * NumPy | * pygame |
| * math | * carla |
| * xmltodict | * collections.deque |
| * argparse | * networkx as nx |
| * collections | * carla\_msgs |
| * datetime | * sensor\_msgs |
| * logging | * OpenCV |
| * Matplotlib | * TensorFlow |

## IDE, Tools, Technologies and Development Platform

* + 1. **IDEs**
* PyCharm
* Visual Studio Code
  + 1. **Development Platform:**
* Ubuntu
* ROS (Robot Operating System)
  + 1. **Tools**
* GitHub
* Jira
* Microsoft office
* Visual Paradigm
* OpenDRIVE viewer
* Carlaviz
* Anaconda
  + 1. **Technologies**
* Carla Simulator
* Carla-Ros-Bridge
* ROS Noetic
* Rospy
* Robot\_localization
* Python

## Best Practices / Coding Standards

* + 1. **Software Engineering Practice**

In our project, we adopted a comprehensive and systematic approach to software engineering practices to ensure the delivery of a scalable and maintainable autonomous vehicle software system. Our methodology was **influenced by industry best practices and tailored** to meet the specific needs of our project. Key practices included:

* + 1. **Feature-Driven Development (FDD)**

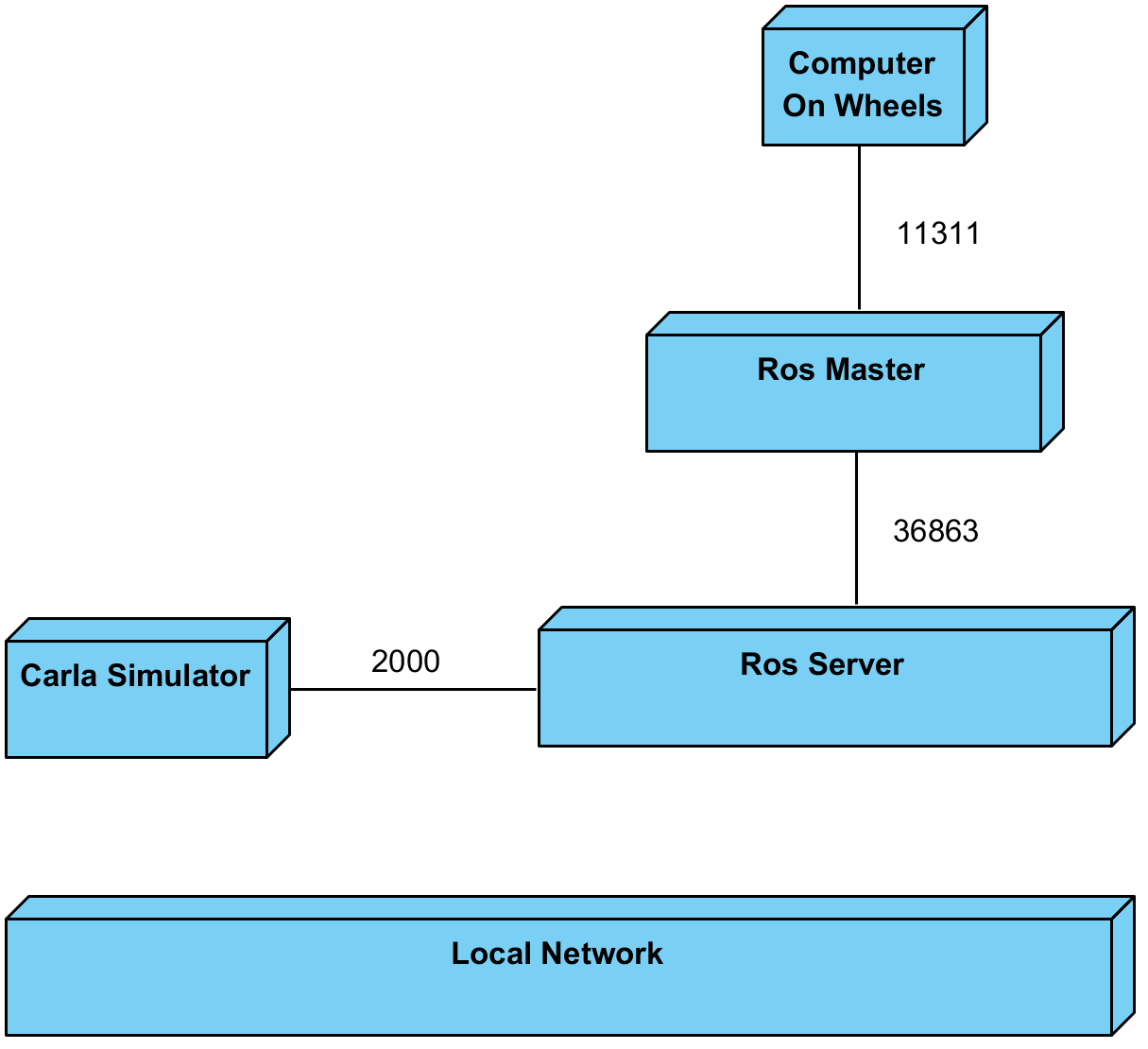
We implemented the Feature-Driven Development (FDD) methodologywhichfalls under the umbrella of **Agile methodologies** to manage our workflow efficiently and adapt to changing requirements. **Utilizing Jira** as our project management tool, we maintained a visual feature list and tracked progress seamlessly. Our agile approach included:

* **Regular Supervisor Meetings**: Conducted weekly meetings with supervisors to review progress, address challenges, and incorporate feedback. These meetings ensured alignment with project goals and facilitated timely decision-making.
* **Feature List Management**: Created and maintained a comprehensive feature list that broke down the system into small features. This list served as the backbone of our development process, guiding incremental and iterative development.
* **Incremental Development**: Emphasized continuous and iterative development, delivering small, functional parts of the project regularly. This approach allowed for frequent validation, adjustment, and integration of new requirements.
  + 1. **Python coding Standards**
* Use snake\_case for variable and function names.
* Use CamelCase for class names.
* Follow PEP 8 guidelines for code formatting.
* Use meaningful variable and function names.
* Keep lines of code within 79 characters.
* Use comments to explain complex parts of the code.
* Use docstrings to document modules, classes, and functions.
* Avoid using global variables unless necessary.
* Handle exceptions gracefully.
* Use virtual environments to manage dependencies.
  + 1. **Rospy coding Standards**
* Follow Python coding standards for rospy code.
* Use rospy naming conventions for nodes, topics, and services.
* Utilize rospy log functions for logging messages.
* Ensure ROS dependencies are properly declared in package.xml and CMakeLists.txt.
* Document ROS nodes, topics, and services using ROS comments.
* Use rospy's rospy.spin() to keep the node alive.
* Handle ROS messages and services according to their specifications.
* Use rospy's parameter server for managing node parameters.
* Implement proper error handling for ROS communication.

## Deployment Environment

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

* + 1. **Deployment Diagram**



*Figure 5.5: Deployment diagram*

## SQA activity: Defect Detection Through White Box Testing

* + 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.
* xstart, xend ∈ Z
* ystart, yend ∈ Z
* Invalid Classes
* start\_node or end\_node as non-integer values or out of grid bounds.
* xstart, xend, ystart, yend ∉ Z

*Table 5.2: White Box TC1*

|  |  |
| --- | --- |
| **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 |

* + 1. **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∈ Z
* 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*

|  |  |
| --- | --- |
| **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 |

* + 1. **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*

|  |  |
| --- | --- |
| **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 |

* + 1. **System Crashes Due to Invalid Input Types for Coordinates**

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

|  |  |
| --- | --- |
| **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 |

*Table 5.5: White Box TC4*

* + 1. **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*

|  |  |
| --- | --- |
| **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 |

* + 1. **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

|  |  |
| --- | --- |
| **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 |

*Table 5.7: White Box TC6*

* + 1. **previous\_destination is not initialized or updated correctly**

**Equivalence Class Partitioning (ECP) for previous\_destination**

* Valid Classes
* ∈ Z
* Invalid Classes
* ∉ Z
* Empty

|  |
| --- |
| **Input Variables** |
| **Test ID** | previous\_destination | **ECP** | **Actual Output** | **Error/Defect** |
| TC015 | None | Invalid | Error | System crash |

*Table 5.8: White Box TC7*

* + 1. **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*

|  |  |
| --- | --- |
| **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 |

* + 1. **PID Controllers to perform longitudinal control**

**Equivalence Class Partitioning (ECP) for target\_speed and waypoint**

* Valid Classes
* target\_speed > 0
* waypoint ∈ Z
* Invalid Classes
* target\_speed <= 0
* waypoint ∉ Z

*Table 5.10: White Box TC9.1*

|  |
| --- |
| **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*

|  |
| --- |
| **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 |

* + 1. **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*

|  |  |
| --- | --- |
| **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. |

* + 1. **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

|  |
| --- |
| **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. |

*Table 5.13: White Box TC11*

## 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:**

**Conclusion and Outlook**

# Chapter 6: Conclusion and Outlook

## Introduction

In this chapter, we will conclude our project on autonomous vehicle navigation by summarizing the key achievements and improvements based on the requirements implemented. We will conduct a critical review, discuss the challenges faced, and highlight the limitations of our current system. Finally, we will provide future recommendations and outlooks for further development in this field, followed by a concise summary.

## Achievements and Improvements

* + 1. **Achievements**
* **Vehicle Control**
  + **Autonomous Navigation**: Successfully implemented autonomous navigation, enabling the vehicle to navigate from a starting point to a destination autonomously.
  + **Throttle Control**: Developed a throttle control system that regulates vehicle speed within the range of 0 to 120 km/h, adjusting for road conditions and traffic regulations.
  + **Steering Control**: Achieved precise steering control, maintaining a maximum lateral deviation of 0.5 meters from the planned trajectory under normal conditions.
* **Path Planning**
  + **Route Calculation**: Implemented an efficient route calculation algorithm to determine the shortest path from the vehicle's current location to the specified destination.
  + **Lane Assignment**: Successfully assigned appropriate lanes for the vehicle along the calculated route.
  + **Waypoint Generation**: Generated waypoints along the calculated route to guide the vehicle towards the destination effectively.
* **Trajectory Planning**
  + **Trajectory Generation**: Planned smooth and optimal trajectories, balancing between minimum travel time and energy efficiency while considering real-time traffic data and road conditions.
* **Sensor Integration**
  + **Inertial Measurement Unit Utilization**: Utilized an IMU to provide orientation and acceleration data at a frequency of 100 Hz.
  + **Global Positioning System Utilization**: Employed GPS to determine the vehicle’s position accurately.
* **Path Following**
  + **Path Smoothing**: Applied path smoothing techniques to limit acceleration changes to within 0.3 m/s², ensuring a smooth ride for passengers.
  + **Lateral Control**: Maintained a lateral deviation of no more than 0.5 meters from the planned path under normal driving conditions.
  + **Longitudinal Control**: Ensured a longitudinal deviation of no more than 1 meter from the planned path under normal driving conditions.
  + **Speed Control**: Effectively controlled the speed to reach the destination.
  + **Waypoint Following**: Followed waypoints along the calculated route to guide the vehicle towards the destination.
* **Destination Arrival**
  + **Destination Approach**: Approached the driver-specified destination with a positional accuracy of within 1 meter, following the calculated trajectory and waypoints precisely.
  + **Stop at Destination**: Brought the vehicle to a complete stop within 1 meter of the designated destination, ensuring deceleration rates did not exceed 2 m/s² for passenger safety and comfort.
* **User Input**
  + **Destination Setting**: Enabled the driver to input the desired destination, triggering the route planning process.
* **System Integration**
  + **ROS Integration**: Utilized the Robot Operating System (ROS) to facilitate communication and data exchange between different software components.
  + **Simulation Environment**: Conducted development and testing in the CARLA simulator for thorough validation before real-world deployment.
    1. **Improvements**
* **Algorithm Optimization**: Enhanced the efficiency of path planning, trajectory generation, and path smoothing algorithms, resulting in reduced computational load and faster execution times.
* **Sensor Fusion**: Improved sensor fusion techniques to combine data from IMU and GPS, increasing the accuracy and reliability of environmental perception.
* **User Interface**: Developed a user-friendly interface for monitoring and controlling the autonomous vehicle, allowing for better interaction and real-time adjustments.

## Critical Review

* + 1. **Strengths**
* **Comprehensive Framework**: The integration of CARLA, ROS Noetic, and the CARLA-ROS bridge provided a comprehensive framework for autonomous vehicle development and testing.
* **Realistic Simulation**: Using the CARLA simulator enabled realistic and diverse testing scenarios, critical for evaluating the robustness of our navigation algorithms.
* **Modular Design**: The modular design of our software architecture allowed for easy updates and extensions of individual components without affecting the entire system.
  + 1. **Weaknesses**
* **Simulation Constraints**: While CARLA provides a realistic simulation environment, it still lacks some real-world complexities, which might affect the transition from simulation to real-world deployment.
* **Computational Resources**: High computational resources were required for running the simulations and algorithms, which could limit scalability and real-time performance on less powerful hardware.
* **Limited Testing Scenarios**: Although extensive, our testing scenarios did not cover all possible real-world conditions, leaving some edge cases untested.

## Future Recommendations/Outlook

**Enhancing Realism in Simulation**

Future work should focus on enhancing the realism of the simulation environment by incorporating more complex and varied scenarios, including adverse weather conditions, varied traffic patterns, and unpredictable pedestrian behaviour.

**Real-World Testing**

Transitioning from simulation to real-world testing is crucial. Developing a robust testing framework that allows for safe and controlled real-world experiments will help validate the algorithms' performance in practical conditions.

**Advanced Machine Learning Techniques**

Incorporating advanced machine learning techniques for perception and decision-making can significantly enhance the vehicle's ability to handle complex environments and unforeseen obstacles.

**Edge Computing Integration**

To address the challenge of high computational resource requirements, integrating edge computing solutions can distribute the processing load and enable real-time performance on resource-constrained platforms.

**Collaboration and Open-Source Contribution**

Engaging with the broader research and developer community through collaborations and contributing to open-source projects can accelerate the development and refinement of autonomous vehicle technologies.

## Summary

In conclusion, our project has successfully demonstrated the potential of using CARLA, ROS Noetic, and the CARLA-ROS bridge for developing robust autonomous vehicle navigation systems. We have achieved significant milestones in **vehicle control, path planning, trajectory planning, sensor integration, path following, destination arrival, user input, and system integration**. Despite the challenges and limitations faced, our work lays a strong foundation for future research and development in this field. By focusing on enhancing realism in simulations, conducting real-world testing, leveraging advanced machine learning, integrating edge computing, and fostering collaboration, we can continue to advance the capabilities of autonomous vehicles, moving closer to their safe and efficient deployment in real-world scenarios.

# 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.

# Appendices

## Appendix-A: [Software Requirements Specifications (SRS)](file:///D:\Final%20Year%20Project\Appendix-A%20Software%20Requirements%20Specifications%20(SRS)6.0.pdf)

## Appendix-B: [Design Artifact](file:///D:\Final%20Year%20Project\Appendix-B%20Design%20Document.pdf)

## Appendix-C: [Coding Standards/Conventions](file:///D:\Final%20Year%20Project\Appendix-C.pdf)

## Appendix-D: [SQA Activities](file:///D:\Final%20Year%20Project)

## Appendix-E: [Work Breakdown Structure](file:///D:\Final%20Year%20Project\Appendix-E%20Work%20Breakdown%20Structure.pdf)

## Appendix-F: [Roles & Responsibility Matrix](file:///D:\Final%20Year%20Project\Appendix-F%20Roles%20&%20Responsibility%20Matrix.pdf)