Integrating the CARLA Autonomous Car Simulator with the RoTRA Highway Code tool

by

Radu Ionuț Pîrlog

Supervised by

Dr. Louise Dennis

A dissertation submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science

The University of Manchester

April 19, 2024

Abstract

This dissertation addresses the integration of the CARLA Autonomous Vehicle Simulator with the RoTRA Highway Code tool, aimed at enhancing the adherence of autonomous vehicles to the UK Highway Code recommendations. RoTRA, a Prolog-based system, analyzes traffic situations to provide relevant Highway Code recommendations. The project focuses on establishing a link between RoTRA and CARLA to identify and flag traffic rule violations within the simulation environment. Additionally, a module is developed within CARLA to enable the utilization of RoTRA recommendations to override the baseline autonomous driving algorithm when necessary, potentially enforcing compliant behaviour such as coming to a stop. The research involves the theoretical exploration on translating CARLA data into RoTRA-relevant information, and the practical implementation of the integration. This work contributes to the advancement of autonomous vehicle technology by promoting safer and more compliant driving behaviour in complex traffic scenarios. Relevant literature and methodologies from formal methods in autonomous systems and hybrid AI frameworks for self-driving vehicles are consulted. This dissertation provides a foundation for further research in the intersection of autonomous vehicle simulation and traffic law compliance.

Acknowledgements

I want to express my gratitude to Dr. Louise Dennis for her guidance, support, and mentorship throughout my dissertation. Dr. Dennis' knowledge, feedback, and encouragement have been essential in shaping the direction of my research and enhancing the quality of this work. Her dedication to excellence and passion for the subject matter have inspired me to strive for academic excellence. I also want to thank Joe Collenette for his contribution to writing the RoTRA Prolog files, which was a crucial component of this project.

Additionally, I want to thank my friends and family for their unwavering support, encouragement, and understanding throughout this academic journey. Their constant encouragement, patience, and belief in me have been a source of strength and motivation. I am truly fortunate to have such a wonderful support network, and I am deeply grateful for their love and encouragement.

Contents

Introduction

1.1 Context and Motivation

As artificial intelligence (AI) continues to advance, its integration into various industries becomes more prevalent. The transportation sector, in particular, stands to undergo significant transformation with the advent of AI-driven technologies. With the global population steadily increasing, urban areas face escalating challenges related to traffic congestion and transportation efficiency. In such a landscape, the development of self-driving vehicles emerges as a promising solution to optimize transit times and alleviate congestion [8].

However, despite the promising prospects of autonomous vehicles (AVs), current implementations still rely on human oversight and intervention in many scenarios. The transition towards fully autonomous systems is underway, but it requires addressing various challenges, including ensuring compliance with existing traffic regulations and safety standards. As AI-driven vehicles navigate complex road environments, the need to adhere to traffic laws becomes paramount to ensure the safety of both passengers and other road users.

By bridging the gap between AI-driven simulation environments and real-world traffic regulations, this research endeavour strives to pave the way for more robust and responsible autonomous vehicle technologies. Through this integration, the aim is to contribute to the ongoing efforts to realize the full potential of self-driving vehicles while ensuring their safe and efficient operation within the existing regulatory framework.

1.2 Overview

The research undertaken in this program aims to contribute to the advancement of autonomous vehicle technology by addressing potential mishaps and ensuring compliance with traffic regulations. By integrating RoTRA (Road Traffic Rule Advisor), a tool designed to interpret and apply the guidelines outlined in the UK Highway Code [3], with the CARLA Autonomous Vehicle Simulation Tool [9], this project seeks to provide a mechanism for detecting and addressing violations of traffic rules. In scenarios where human intervention may not be feasible or timely, the RoTRA system could assume control to enforce compliance with the law, thus enhancing the safety and reliability of autonomous driving systems.

The project aims to integrate two key components: the CARLA Autonomous Car Simulator and the RoTRA Highway Code tool. CARLA provides a sophisticated platform for simulating autonomous driving scenarios, while RoTRA is a tool designed to interpret and apply the guidelines outlined in the UK Highway Code. By combining these tools, the project seeks to enhance the realism and safety of autonomous vehicle simulations by ensuring compliance with traffic regulations.

The integration process involves several key steps:

1. **Implementing Communication Interface:** The project focuses on implementing a communication interface between CARLA and RoTRA. This interface enables the exchange of information between the two systems, allowing CARLA to send simulated traffic scenarios to RoTRA for analysis.

2. **Mapping CARLA Scenarios to RoTRA Input:** Next, the project involves mapping the scenarios simulated in CARLA to input formats compatible with RoTRA. This requires defining a standard representation for traffic situations within CARLA that can be interpreted by RoTRA.

3. **Interpreting RoTRA Actions:** Once RoTRA analyzes a traffic scenario, the project involves interpreting the recommendations provided by RoTRA. This includes identifying violations of traffic rules, assessing the severity of violations, and determining appropriate corrective actions.

4. **Integration with CARLA:** In the final stage of the project, the focus shifts to integrating RoTRA's recommendations with CARLA's behaviour module. This involves developing logic within CARLA to respond to RoTRA's recommendations, such as changing the actions of autonomous vehicles to comply with traffic rules or taking corrective actions to prevent violations.

1.3 Objective

The primary objective of the project is to establish a functional link between RoTRA and CARLA, to detect and address traffic rule violations within the simulated environment. This objective can be broken down into two key components:

1. **Warning System for Traffic Rule Violations:** The project aims to develop a mechanism within CARLA that utilizes RoTRA to identify instances where traffic rules are violated. This involves integrating RoTRA's rule interpretation capabilities with CARLA's simulation environment to detect deviations from the UK Highway Code. When a violation occurs, the system will generate warnings or alerts to notify users of the infraction, enabling them to take corrective action.

2. **Behavior Enforcement:** In addition to issuing warnings, the project explores the possibility of developing a subsystem within CARLA capable of enforcing compliant behaviour based on RoTRA's recommendations. This entails designing algorithms and logic within CARLA to autonomously adjust the behaviour of simulated vehicles in response to detected violations. For instance, if a vehicle runs a red light, the system may intervene to ensure the vehicle comes to a stop or takes appropriate corrective action to avoid accidents.

By achieving these objectives, the project aims to enhance the realism and safety of autonomous vehicle simulations conducted within the CARLA environment. By leveraging RoTRA's rule interpretation capabilities, the system can provide valuable feedback on adherence to traffic regulations, thereby improving the effectiveness and reliability of autonomous driving algorithms. Ultimately, the project seeks to contribute to the development of safer and more responsible autonomous vehicle technologies.

1.4 Summary of Evaluation Strategy

The evaluation strategy adopted in this research aims to comprehensively assess the effectiveness and reliability of the integrated RoTRA-CARLA system in detecting and addressing traffic rule violations across diverse driving scenarios. The strategy encompasses several key components:

1.4.1 Scenario-based Testing

Extensive scenario-based testing is conducted within the CARLA simulation environment, covering a wide range of traffic situations, including intersections, urban driving, highway driving, and rural environments. These scenarios are designed to evaluate the system's ability to detect and respond to traffic rule violations accurately.

1.4.2 Quantitative Metrics

Quantitative metrics are utilized to measure system performance, including the accuracy of rule interpretation, detection rates of rule violations, and frequency of corrective actions. These metrics provide objective assessments of the system's efficacy in enforcing compliance with traffic regulations.

1.4.3 Comparative Analysis

A comparative analysis is conducted to evaluate the impact of RoTRA integration on safety and reliability. Performance metrics of AVs with and without RoTRA oversight are compared to assess the effectiveness of RoTRA in improving compliance with traffic rules and enhancing overall safety.

1.4.4 Iterative Refinement

The evaluation process follows an iterative refinement approach, where insights from testing phases inform subsequent iterations of system development. Continuous refinement based on empirical data and user feedback ensures that the integrated RoTRA-CARLA system evolves to appropriately intervene when necessary.

The evaluation strategy was chosen based on the need to comprehensively assess the integrated RoTRA-CARLA system's performance in addressing traffic rule violations across various driving scenarios. Scenario-based testing was selected to simulate real-world driving conditions, allowing for thorough evaluation of the system's capabilities. Quantitative metrics were employed to provide objective assessments, ensuring reliable measurements of system performance. A comparative analysis was conducted to evaluate the impact of RoTRA integration on safety and reliability, providing insights into its effectiveness. The iterative refinement approach was chosen to facilitate continuous improvement based on empirical data and user feedback, ensuring that the system evolves to meet evolving demands and challenges. Overall, this strategy was selected to ensure a rigorous evaluation process that addresses the objectives of the research and contributes to the advancement of autonomous vehicle technologies.

Background

2.1 Autonomous Driving Technology

This section provides a broader context for the integration framework by discussing the advancements and challenges in autonomous driving technology. It covers key concepts such as perception, planning, and control, as well as current trends and research directions in the field.

2.1.1 Perception

Autonomous vehicles rely on a diverse array of sensors to perceive and understand their surroundings. These sensors include cameras, LiDAR (Light Detection and Ranging), radar (Radio Detection and Ranging), ultrasonic sensors, and GPS (Global Positioning System). Each type of sensor captures different aspects of the environment, such as visual information, depth, motion, and positioning data, allowing the vehicle to build a comprehensive understanding of its surroundings.

Cameras

Cameras capture high-resolution images of the vehicle's surroundings [23], providing valuable visual information about objects, road markings, traffic signs, and other relevant features. Object detection algorithms analyze these images to identify and classify objects such as vehicles, pedestrians, cyclists, and obstacles.

Figure 2.1: Google self-driving car at the Computer History Museum

LiDAR

LiDAR sensors emit laser pulses and measure the time it takes for the pulses to reflect off surrounding objects, creating a detailed 3D point cloud of the environment. LiDAR data provides precise distance measurements and 3D spatial information, allowing the vehicle to accurately perceive the shape, size, and location of objects in its vicinity [17]. Object detection algorithms process LiDAR data to detect and localize objects, even in challenging lighting conditions or adverse weather.

Radar

Radar sensors use radio waves to detect the presence and movement of objects around the vehicle. Radar sensors are particularly useful for detecting objects at longer ranges and in adverse weather conditions such as fog, rain, or snow. They provide information about the velocity and relative speed of objects, enabling the vehicle to anticipate potential collisions and adjust its trajectory accordingly [20].

Ultrasonic Sensors & GPS

Ultrasonic sensors emit high-frequency sound waves and measure the time it takes for the waves to reflect off nearby objects. Ultrasonic sensors are commonly used for short-range object detection and obstacle avoidance [16], particularly in parking and low-speed manoeuvring scenarios. GPS receivers provide accurate positioning and velocity information, allowing the vehicle to localize itself within the global coordinate system. GPS data is used for navigation, route planning, and localization, enabling the vehicle to follow predefined routes and navigate to desired destinations.

2.1.2 Planning and Decision Making

Planning and decision-making are essential for autonomous driving systems. They enable vehicles to navigate safely and efficiently in complex environments. This section discusses the processes involved in planning and decision-making, including route planning, trajectory generation, and decision-making algorithms, while exploring advanced techniques and considerations.

Route Planning

Route planning is critical for autonomous driving. It involves finding the fastest and most efficient path from the vehicle's current location to its intended destination while taking into account factors such as traffic conditions, road geometry, and legal constraints [26]. The goal is to reduce travel time, fuel consumption, and overall trip cost while ensuring passenger comfort and safety.

Various algorithms are available for route planning, each with its strengths and limitations. For offline route planning, common algorithms include A* search, Dijkstra's algorithm, and variants of the rapidly exploring random tree (RRT) algorithm. Online planning techniques like rapidly exploring random trees with reeds-shepp paths (RRT*) are used for dynamic environments where real-time adaptation is required [12].

In dynamic environments, such as areas with heavy traffic or construction zones, route planning algorithms must adapt to changing conditions in real-time [11]. Reactive planning techniques, which respond to immediate environmental changes, and online replanning, which continuously updates the vehicle's route based on new information, are es-

sential for ensuring safe and efficient navigation.

Trajectory Generation

Trajectory generation is another critical component of autonomous vehicle navigation. It involves generating a smooth and feasible trajectory for the vehicle to follow along its planned route while adhering to physical constraints such as vehicle dynamics, road boundaries, and obstacle avoidance. The trajectory must also optimize criteria such as comfort, efficiency, and safety [22].

Various trajectory generation algorithms employ different techniques, including spline interpolation, polynomial fitting, optimization-based approaches (e.g., quadratic programming, model predictive control), and sampling-based methods (e.g., probabilistic roadmaps, lattice-based planners) [5, 15, 22]. Predictive planning techniques are also used to generate robust trajectories. They anticipate future states of the environment and potential interactions with other road users to proactively plan for potential hazards and ensure safe navigation in dynamic environments [19].

Decision Making

Decision-making algorithms are at the core of how autonomous vehicles operate. They make decisions based on the vehicle's perception of the environment, its planned route, and its objectives, such as safety, efficiency, and legality. They control the vehicle's behaviours, including lane-keeping, speed control, intersection negotiation, and response to dynamic traffic situations.

One type of decision-making system is rule-based, which encodes traffic rules, regulations, and behavioural norms to guide the vehicle's actions. These systems prioritize compliance with traffic laws and safe driving practices while navigating complex traffic scenarios. They may incorporate hierarchical decision-making frameworks, where higher-level goals guide lower-level behaviours [1].

Another approach to decision-making is reinforcement learning, which allows the vehicle to learn optimal decision-making policies through interaction with the environment. Reinforcement learning algorithms enable agents to maximize rewards by selecting actions that lead to desired outcomes, such as reaching the destination while minimizing travel time

and avoiding collisions. These algorithms can be trained using simulation environments, real-world data, or a combination of both, allowing the vehicle to adapt its behaviour to different driving conditions and scenarios [1].

Human-machine interaction is also an important consideration for autonomous vehicles. Decision-making algorithms may incorporate models of human behaviour and intention estimation to anticipate and respond to interactions with other road users, including pedestrians, cyclists, and other drivers [13].

2.1.3 Control

Control algorithms play a crucial role in enabling autonomous vehicles to execute planned trajectories and navigate safely in dynamic environments. This section explores various control techniques utilized by autonomous vehicles, including PID control, model predictive control (MPC), and reinforcement learning-based control.

PID Control

PID (Proportional-Integral-Derivative) control is a widely used feedback control technique that helps autonomous vehicles perform various tasks such as speed control, lane keeping, and lateral or longitudinal vehicle control. It consists of three components: proportional, integral, and derivative terms. These components work together to adjust the vehicle's control inputs based on the error between the desired state and the actual state. Tuning the PID controllers means adjusting the gains of the proportional, integral, and derivative terms to improve the vehicle's performance [2, 21].

MPC

Model Predictive Control (MPC) is an advanced control technique that uses a predictive model of the vehicle's dynamics to optimize control inputs over a finite prediction horizon. MPC optimizes the control inputs by predicting future states based on the current state and input trajectory, and iteratively adjusting the control inputs to minimize a cost function. MPC is used in trajectory tracking, path following, and obstacle avoidance in autonomous vehicles [25]. One of the key benefits of MPC is its ability to handle nonlinear dynamics, constraints, and uncertainty in the system [21].

Reinforcement Learning-Based Control

Reinforcement learning (RL) algorithms enable autonomous vehicles to learn control policies by interacting with the environment. RL-based control is employed for adaptive cruise control, lane keeping, and decision-making tasks in autonomous vehicles. RL-based control allows self-driving cars to adapt their control policies to varying driving conditions, traffic patterns, and user preferences. However, RL-based control faces challenges such as sample inefficiency, exploration-exploitation trade-offs, and generalization to unseen scenarios. To address these challenges, careful algorithm design, training data selection, and evaluation methodologies are required to ensure safe and reliable performance in real-world environments [24].

2.2 Integration Framework

The Integration Framework section provides an overview of the project's overarching objective to link RoTRA, the CARLA Autonomous Car Simulator, and the UK Highway Code. This section serves as the foundation for understanding how these components are interconnected to achieve the project's goals. It outlines the rationale behind integrating RoTRA with CARLA, highlighting the importance of leveraging the UK Highway Code's traffic regulations to enhance the realism and safety of autonomous vehicle simulations. Additionally, the section sets the stage for discussing the individual components in detail, including their functionalities, roles within the integration framework, and the anticipated benefits of their collaboration.

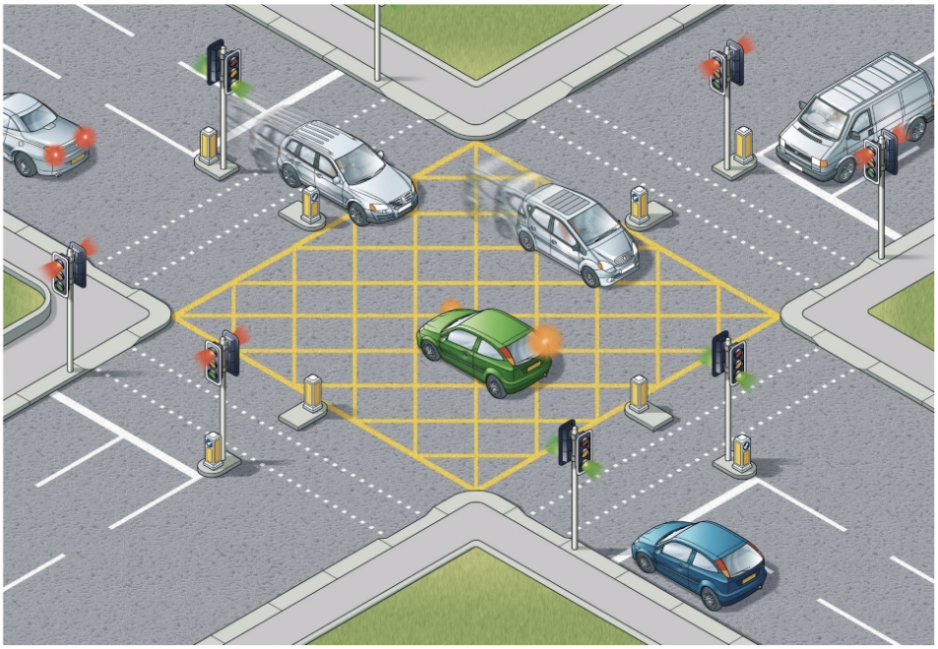
2.2.1 UK Highway Code

The UK Highway Code is an essential document that outlines the rules and regulations for using roads and traffic behaviour in the United Kingdom. It is an extensive guide covering various aspects of road usage, including rules for drivers, pedestrians, and cyclists. The Highway Code provides critical information on road signs, markings, and signals, helping road users understand their meanings and how to respond appropriately. It also establishes guidelines for safe driving practices such as keeping a safe distance from other vehicles, obeying speed limits, and using indicators when turning or changing lanes.

Rule 174

Box junctions. These have criss-cross yellow lines painted on the road (see 'Road markings'). You **MUST NOT** enter the box until your exit road or lane is clear. However, you may enter the box and wait when you want to turn right, and are only stopped from doing so by oncoming traffic, or by other vehicles waiting to turn right. At signalled roundabouts you **MUST NOT** enter the box unless you can cross over it completely without stopping.

Law TSRGD schedule 9 parts 7 and 8



Rule 174: Enter a box junction only if your exit road is clear

Figure 2.2: Example of a Highway Code Rule (Rule 174)

The Highway Code addresses specific scenarios and situations encountered on the road, such as roundabouts, pedestrian crossings, and junctions. It outlines priority rules for yielding and right of way, as well as procedures for overtaking, merging, and parking. Additionally, it emphasizes the importance of courteous and considerate driving behaviour, encouraging drivers to be aware of and respectful towards other road users.

The Highway Code is regularly updated to reflect changes in legislation, advancements in technology, and evolving road conditions. It serves as a vital resource for both experienced drivers and those learning to drive, providing guidance on road safety and legal responsibilities. By promoting adherence to its guidelines, the Highway Code aims to reduce the risk of accidents, minimize congestion, and improve overall road safety for everyone [3].

2.2.2 RoTRA Tool

RoTRA (Road Traffic Rule Advisor) [7] is an AI-powered system that uses the Belief-Desire-Intention (BDI) framework to interpret and apply the guidelines outlined in the UK Highway Code effectively. In a BDI system, beliefs represent the system's knowledge about the world, desires denote the system's goals or objectives, and intentions signify the system's planned actions based on its beliefs and desires.

RoTRA has been implemented using Prolog, a logic programming language. It maintains a knowledge base containing traffic rules and regulations derived from the Highway Code. RoTRA classifies all rules pertinent to road traffic into two primary scenarios: standard and emergency. This classification allows RoTRA to provide tailored recommendations based on the specific circumstances of a given traffic situation.

RoTRA is highly adaptable and modular. While initially designed for the UK Highway Code, RoTRA's architecture allows for seamless modification to accommodate the legal systems of other countries [18]. This versatility makes RoTRA an invaluable tool with wide-ranging applicability, enhancing its suitability for integration into various autonomous vehicle (AV) simulations.

For this project, RoTRA selectively incorporates rules from the Highway Code that are relevant to the driver's perspective, aligning with the focus on autonomous vehicle behaviour simulation. By leveraging RoTRA's capabilities, users can receive real-time advice on navigating traffic scenarios in adherence to legal requirements. The tool's ability to analyze situations and generate recommendations promotes informed decision-making and fosters compliance with traffic laws, contributing to the enhancement of road safety and the cultivation of responsible driving practices within AV simulations.

Example

Additionally, Figure 2.3 is an example of how a highway code rule is encoded in RoTRA [6]:

```
%Rule Description: Traffic Lights (Stop at red)
rule(r175a, standard, [lightRed], [approachingTrafficLight], [must-stop_at_white_line]).
```

Figure 2.3: Rule: Traffic Lights (Stop at red)

The Prolog rule for traffic lights follows the Belief-Desire-Intention (BDI) framework. It specifies conditions for stopping at red lights (`lightRed`) in standard scenarios when approaching a traffic light (`approachingTrafficLight`). The action (`must_stop_at_white_line`) dictates stopping at the white line.

2.2.3 CARLA

Description

CARLA stands as a cornerstone in the field of autonomous driving research, providing an open-source simulation platform that enables researchers and developers to test and validate their algorithms in a realistic virtual environment. Developed initially by the CARLA Team at Intel Labs and now maintained by a thriving community of contributors, CARLA offers a comprehensive suite of tools and features tailored to the needs of autonomous driving research.

At its core, CARLA provides a highly detailed and dynamic urban environment that mimics real-world conditions, complete with streets, intersections, traffic lights, pedestrians, and other vehicles. This realistic simulation environment allows researchers to replicate various driving scenarios and challenges encountered in urban settings, such as navigating through dense traffic, responding to unpredictable pedestrian behaviour, and adapting to changing weather conditions.

Its versatility extends beyond its realistic environment, offering support for a wide range of sensors commonly used in autonomous vehicles, including cameras, LiDAR, radar, and GPS. These sensors enable researchers to collect rich data streams from their simulated vehicles, facilitating the development and testing of perception, localization, and mapping algorithms [9].

Architecture

CARLA boasts a modular architecture that allows for easy integration with external systems, making it a flexible platform for conducting experiments and research. Whether it's testing novel control algorithms, evaluating sensor fusion techniques, or assessing the impact of environmental factors on driving behaviour, CARLA provides researchers with the tools they need to push the boundaries of autonomous driving technology.

Besides all of this, it not only provides a realistic simulation environment but also offers a well-structured codebase that facilitates ease of use and customization. Its modular architecture allows researchers to extend and modify the platform to suit their specific research needs. The codebase is organized into distinct modules, each responsible for different aspects of the simulation, such as vehicle dynamics, sensor simulation, and environment generation. This modular approach simplifies the integration of additional features and ensures flexibility in the simulation setup.

Traffic Manager

One of CARLA's standout features is its Traffic Manager, which governs the behaviour of AI-controlled vehicles and pedestrians in the simulation. The Traffic Manager module in CARLA serves as the simulation engine's traffic controller, orchestrating the behaviour of AI-controlled vehicles and pedestrians within the virtual environment. Through a combination of algorithms and decision-making processes, it dynamically adjusts the actions of simulated agents based on factors such as traffic density, road conditions, and user-defined parameters. This functionality enables the Traffic Manager to replicate a wide range of realistic traffic scenarios, including traffic jams, intersections, lane changes, and pedestrian crossings, providing a rich and immersive simulation experience for researchers.

Moreover, the Traffic Manager plays a crucial role in enhancing the realism of autonomous vehicle simulations by accurately simulating traffic behaviour in urban environments. By replicating real-world traffic dynamics, it allows researchers to study complex interactions between multiple agents, such as vehicle-to-vehicle interactions, pedestrian-vehicle interactions, and adherence to traffic rules and regulations. This realism enables researchers to validate and evaluate autonomous driving algorithms under various challenging conditions, leading to more robust and reliable autonomous vehicle systems.

One of the key strengths of the Traffic Manager is its flexibility and configurability, allowing researchers to customize and control various aspects of traffic simulation. Researchers can adjust parameters such as traffic density, vehicle speed limits, traffic light timings, and pedestrian behaviours to simulate specific traffic scenarios and study their impact on autonomous vehicle performance. This level of customization provides researchers with the flexibility to tailor simulations to their specific research objectives, enabling targeted experimentation and analysis.

The Traffic Manager has a significant impact on autonomous driving research and development by providing a realistic and scalable platform for testing and validating autonomous vehicle algorithms. By accurately simulating traffic scenarios, it helps researchers assess the performance of autonomous driving algorithms, identify potential weaknesses or failure modes, and iterate on algorithm design and implementation. This iterative process of testing and refinement accelerates the development of autonomous vehicle technology, ultimately leading to safer, more reliable, and more efficient autonomous vehicles.

Client-Server Connection

CARLA offers a friendly user interface (UI) that allows researchers and developers to interact with the simulation environment efficiently. The intuitive UI provides tools for configuring simulation parameters, controlling vehicles, monitoring simulation metrics, and visualizing simulation data in real-time. Additionally, CARLA's comprehensive Software Development Kit (SDK) offers a set of APIs and libraries for programmatically accessing and manipulating simulation data, enabling developers to integrate CARLA into custom software applications and workflows seamlessly.

The UI and SDK combination in CARLA serves as a powerful framework for building and extending simulation capabilities, providing researchers with the flexibility to customize simulations according to their specific requirements and objectives. The effectiveness of the client-server architecture in facilitating communication and data exchange between software components was recognized, prompting the leverage of this approach to establish a connection between CARLA and RoTRA. Through the adoption of a client-server model, real-time communication between the simulation environment in CARLA and the rule interpretation engine in RoTRA was enabled, contributing to the enhancement of the realism and effectiveness of autonomous vehicle simulations. This approach facilitates the transmission of real-time updates from CARLA to RoTRA and the reception of recommendations based on the interpretation of traffic rules, thus enabling dynamic and interactive simulations aligned with real-world scenarios.

2.2.4 Application Programming Interfaces (APIs)

APIs, or Application Programming Interfaces, help different software systems communicate and interact with each other. Think of them as a bridge between CARLA, RoTRA,

and other components. They allow these systems to exchange data seamlessly.

The custom API developed within CARLA serves as the primary communication channel between the simulation environment and RoTRA's server. This API provides real-time updates such as vehicle states, traffic conditions, and environmental cues to RoTRA. It also streamlines the reception of responses and recommendations from RoTRA, which can be interpreted and acted upon within the simulation environment.

APIs standardize communication protocols and data formats. They make sure different software modules work well together. APIs allow for simplified interfaces that exchange information and trigger actions. In the context of this integration framework, APIs streamline the integration process. They enhance the overall effectiveness and efficiency of the autonomous vehicle simulation by allowing seamless interaction between CARLA, RoTRA, and other components.

2.3 Related Work

2.3.1 Previous Integration Efforts

Integrating traffic rule interpretation tools with autonomous driving simulators has been a topic of research. The goal is to make autonomous vehicles more realistic and compliant with traffic regulations by adding rule interpretation capabilities to simulation environments. However, past attempts have encountered some challenges.

One of the biggest challenges is making sure that the traffic rules and environment states are the same in both the simulator and the interpretation tool. This is essential for seamless integration. Another big challenge is making sure that the interpretation of traffic rules is realistic and accurate within the simulation environment. This is a complex task because the simulated vehicles must comply with traffic regulations while still responding realistically to dynamic environmental conditions.

Performance and scalability are also important considerations when integrating these systems. Integration efforts need to take this into account to make sure that the interpretation of traffic rules doesn't significantly impact simulation performance or scalability. Finally, making sure that different software components, such as the autonomous driving

simulator and the rule interpretation tool, can work together is crucial for seamless integration. Standardized interfaces and communication protocols make it easier to exchange data between the integrated systems.

To address these challenges, previous integration efforts have used middleware solutions or developed custom integration frameworks. These frameworks were tailored to the specific requirements and constraints of the simulation environment and the rule interpretation tool. Researchers have learned from these experiences and understand the importance of realistic interpretation, iterative development approaches, and collaboration among researchers, developers, and domain experts. Open collaboration platforms and shared resources facilitate the exchange of ideas, methodologies, and best practices among the research community.

2.3.2 Case Studies

Zhu et al [27] created a traffic generation framework called "Realistic Interactive Traffic Flow" (RITA) for enhancing already existing autonomous driving simulations. The end scope of the project was to create high-quality traffic flows that reflect real data and present human-like reactive responses. RITA consists of two core modules, namely RITABackend and RITAKit. The former supports vehicle-wise control and provides real-world generation models from its datasets, while the latter offers simple interfaces that can modify the generation of traffic. In utilising both technologies, RITA's effectiveness in improving driving strategy evaluation is demonstrated.

Lin, Xie and Liu's research [14] proposes a method to enhance the performance of autonomous vehicle decision-making and control tasks trained in simulated environments by considering real-world applicability. It addresses the limitations of current studies, which often neglect the transition to real or near-real environments during testing, leading to potential performance degradation. The proposed method involves randomizing the driving style and behaviour of surrounding vehicles by adjusting the parameters of the car-following and lane-changing models in SUMO, a traffic simulation software. Policies are trained using deep reinforcement learning algorithms under domain-randomized rule-based microscopic traffic flow in freeway and merging scenarios. These policies are then tested separately in both rule-based and high-fidelity microscopic traffic flow envi-

ronments. The results demonstrate that policies trained under domain-randomized traffic flow exhibit significantly improved success rates and calculative rewards compared to models trained under other traffic flow conditions.

The validation of Gómez-Huélamo et al.'s fully autonomous driving architecture is conducted utilizing the National Highway Traffic Safety Administration (NHTSA) protocol within the CARLA simulator, with a specific focus on analyzing the decision-making module, which is rooted in Hierarchical Interpreted Binary Petri Nets (HIBPN). Emphasis is placed on the utilization of hyper-realistic simulators and the creation of appropriate traffic scenarios to ensure the development of safe and robust AD technology. The pipeline employed leverages standard communication in robotics through the use of ROS and Docker to provide isolation, flexibility, and portability. Validation of the architecture encompasses challenging driving scenarios, including Pedestrian Crossing, Stop, ACC, and Unexpected Pedestrian encounters. Both qualitative and quantitative results for each use case are presented, thereby validating the architecture in simulation before its implementation in the real autonomous electric car [10].

Al Shukairi and Cardoso's ML-MAS framework for fully autonomous self-driving vehicles [4] has been validated using the CARLA simulator. The focus of the analysis is on the decision-making module, which combines Machine Learning (ML) systems and rule-based systems. The goal is to develop safe and robust autonomous driving technology by using hyper-realistic simulators and creating appropriate traffic scenarios. The pipeline employed uses ROS and Docker to provide isolation, flexibility, and portability. The architecture is validated using challenging driving scenarios such as Pedestrian Crossing, Stop, ACC, and Unexpected Pedestrian encounters. The results, both qualitative and quantitative, are presented for each use case, validating the architecture in simulation before its implementation in real-world self-driving vehicles. The ML-MAS framework improves the driving score of the ML models in the CARLA simulation environment, thereby validating the architecture in simulation before its implementation in real-world self-driving vehicles.

Methodology

3.1 Architecture and Design

As shown in Figure 3.1, the integration of RoTRA with CARLA involves a comprehensive architecture that consists of three primary components: CARLA Server, Agent, and Prolog Server. Each component plays a distinct role in the system, contributing to the overall functionality and interaction between the autonomous agent and the simulation environment.

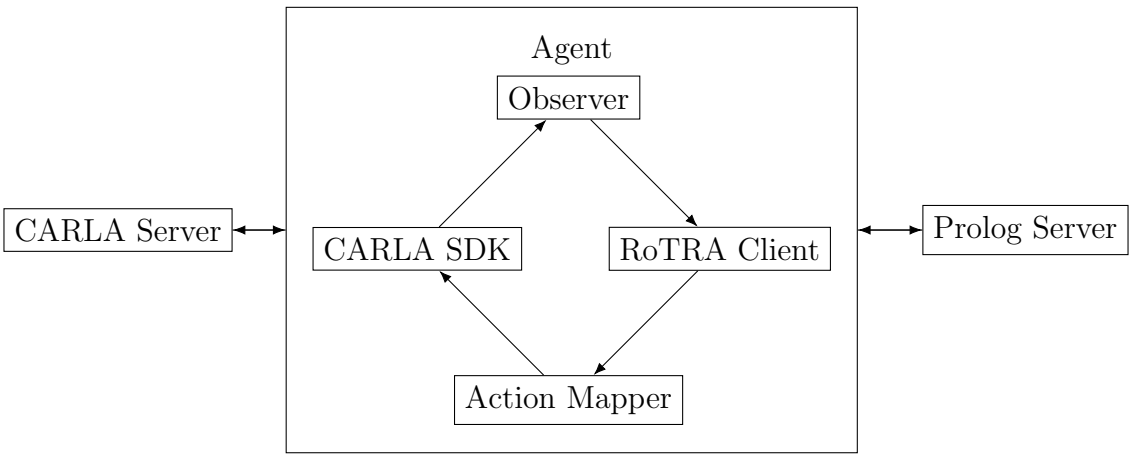


Figure 3.1: System Architecture

CARLA Server

CARLA Server (2.2.3) serves as the simulation platform. It provides a realistic environment for testing and evaluating autonomous driving algorithms, including functionalities for rendering graphics, simulating physics, and managing the dynamic environment such as traffic flow and weather conditions. CARLA Server's role remains unchanged in the in-

tegration process, as it focuses solely on providing a visual representation of the simulated world.

Agent

The Agent serves as the decision-making entity within the system, coordinating interactions between the environment and the autonomous vehicle. The Agent is composed of four key components: Observer, RoTRA Client, Action Mapper, and CARLA SDK.

1. The Observer module collects data from the CARLA environment, such as vehicle positions and traffic light states. It processes this information to derive relevant insights about the environment.

2. The RoTRA Client is responsible for transforming the observed data into a format suitable for reasoning and decision-making. It converts the gathered information into a Belief-Desire-Intention (BDI) model and communicates with the Prolog Server to obtain action recommendations.

3. The Action Mapper receives the recommended actions from the Prolog Server and translates them into executable commands for CARLA. It maps abstract actions, such as "stop" or "slow down" into specific control signals for the vehicle, such as throttle and brake values.

4. The CARLA SDK acts as the interface between the Agent and the CARLA environment. It sends the translated action commands to CARLA for execution and receives feedback on the vehicle's state.

Prolog Server

The Prolog Server (Section: 2.2.2) serves as the reasoning engine, providing logical inference and decision-making capabilities to the Agent. It receives input from the RoTRA Client, which includes the current state of the environment and the agent's goals and intentions. Using a Prolog-based inference engine, the Prolog Server evaluates the received information and generates recommended actions based on predefined rules and constraints. The recommended actions are then communicated back to the Agent for further processing and execution.

Pipeline

The integration process follows a systematic workflow: The Observer module gathers data from the CARLA environment and preprocesses it for analysis. The RoTRA Client translates the observed data into a structured format and sends it to the Prolog Server for reasoning. The Prolog Server evaluates the input data and generates recommended actions based on predefined logical rules. The recommended actions are passed to the Action Mapper, which converts them into executable commands for CARLA. The CARLA SDK executes the commands in the simulation environment, and the process repeats for each simulation step.

This architecture provides a robust framework for integrating RoTRA with CARLA, enabling seamless interaction between the autonomous agent and the simulation environment while maintaining modularity and scalability for future enhancements and optimizations.

3.2 Prolog API

The Prolog API serves as a critical component within the system architecture, enabling seamless communication between the agent and external entities through HTTP requests and responses. This section provides a detailed exploration of the Prolog API implementation, focusing on its structure, routing mechanisms, and handler logic.

3.2.1 Prolog Predicate Nomenclature

In Prolog, functions are usually referred to as predicates. A predicate is defined by its name and arity, which is the number of arguments it takes. In this paper, predicates are represented by stating the predicate name followed by a slash and the number of arguments. In the provided API implementation, predicates are named based on their functionality, for example, `get_actions/1` is used for retrieving actions, and `start_server/1` is used for initiating the HTTP server.

3.2.2 Required Modules

The Prolog API is a powerful tool that uses both built-in and external modules to manage HTTP communication and data processing. These modules work together to provide a complete solution for handling HTTP requests and responses.

HTTP Server Management

One of the key modules is `library(http/thread_httpd)`, which is responsible for creating and managing HTTP servers. This module provides a simple interface for starting and stopping servers, as well as configuring server options such as the port number and maximum number of threads.

Request Handling

Another important module is `library(http/http_dispatch)`, which dispatches HTTP requests to appropriate handler predicates based on URL paths. This module allows developers to define rules for handling different types of requests, making it easy to create custom endpoints for APIs.

Error Handling

The `library(http/http_error)` module is also essential, as it gracefully handles HTTP errors and exceptions. This module ensures that errors are reported in a clear and understandable way, making it easier for developers to debug their code.

JSON Data Processing

The Prolog API also includes modules such as `library(http/http_json)` for working with JSON data, which supports the parsing and generation of JSON data in HTTP requests and responses. The `library(http/json_convert)` module facilitates the conversion between Prolog terms and JSON data, while the `library(http/json)` provides utilities for working with JSON data in HTTP requests and responses.

Additional Functionality

Other modules included in the Prolog API are `library(http/http_parameters)`, which parses HTTP parameters from requests for further processing, `library(http/http_client)`, which allows the sending of HTTP requests to external endpoints for data retrieval or modification, and `library(http/http_header)`, which handles HTTP headers in requests and responses to ensure proper communication.

3.2.3 Defining API Routes

The Prolog API defines various routes using the `http_handler/3` predicate, specifying the URL paths and corresponding handler predicates responsible for processing incoming requests. Each route is associated with a specific HTTP method to ensure proper handling of requests:

- `/`: Handles PATCH requests to obtain recommended actions based on beliefs and intentions.
- `/actions`: Handles GET requests to retrieve actions based on contextual information.
- `/beliefs`: Handles GET requests to retrieve beliefs from the context.
- `/intentions`: Handles GET requests to retrieve intentions based on the context.

3.2.4 Implementing Handler Predicates

Handler predicates are vital components responsible for processing incoming HTTP requests, extracting relevant parameters, performing necessary computations, and generating appropriate responses. Each handler predicate corresponds to a specific route defined in the API:

- `get_actions/1`: Processes PATCH requests to obtain recommended actions based on beliefs and intentions provided in the request payload and generates a JSON response.
- `get_all_actions/1`: Retrieves actions based on the provided context and generates a JSON response.

- `get_beliefs/1`: Retrieves beliefs based on the provided context and generates a JSON response.
- `get_intentions/1`: Retrieves intentions based on the provided context and generates a JSON response.

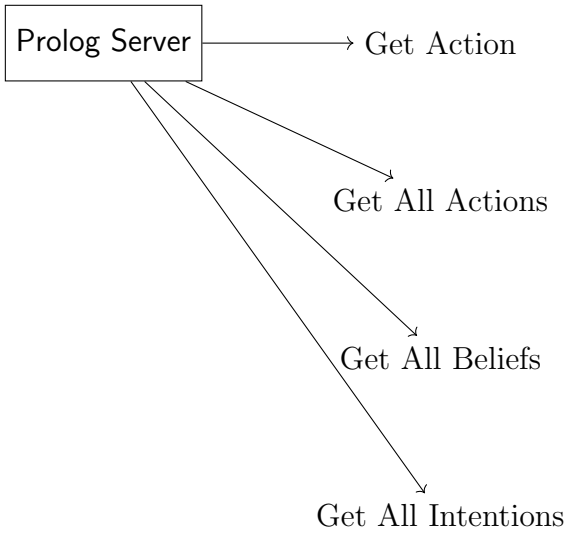


Figure 3.2: API Endpoints of the Prolog Server

3.2.5 Starting the Server

The `start_server/1` predicate initiates the HTTP server, specifying the port on which the server listens for incoming connections. Upon successful initialization, a message indicating the server's start is displayed, confirming its readiness to handle requests.

3.2.6 Overall Purpose and Significance

In contemporary knowledge-based systems, the ability to interact with external entities is crucial for real-world applicability and integration into larger software ecosystems. This Prolog API plays a pivotal role in enabling seamless communication between a Prolog-based knowledge base and external systems or applications. Here's how:

- **Facilitating Integration**: By providing a standardized HTTP interface, the API enables easy integration of Prolog-based knowledge systems with a wide range of modern software applications and services. External systems can interact with the Prolog knowledge base without requiring detailed knowledge of Prolog internals, thereby promoting interoperability.

- **Simplifying Communication**: The API abstracts away the complexity of interacting with the knowledge base, offering a simple and intuitive interface for querying beliefs, intentions, and actions. This simplification reduces the barrier to entry for developers who wish to leverage the knowledge base within their applications.
- **Enhancing Scalability and Modularity**: The use of HTTP-based communication promotes a modular architecture, allowing different components of a larger system to be developed, deployed, and scaled independently. This modular approach enhances flexibility, scalability, and maintainability, making it easier to evolve and extend the system over time.
- **Supporting Web Services and IoT Integration**: With the omnipresence of web services and the increase of Internet of Things (IoT) devices, the ability to expose the Prolog knowledge base as an HTTP API opens up opportunities for integration with web applications, mobile apps, and IoT platforms. This enables novel applications such as intelligent assistants, decision support systems, and smart IoT devices that leverage Prolog's reasoning capabilities.

3.3 Agent Module

In this section, the implementation details of important components within the agent module will be analysed (Figure: 3.3). Various aspects such as observing the world, RoTRA client functionalities, the action mapper system, and integration with the CARLA SDK will be explored. The primary focus is on adhering to RoTRA guidelines related to traffic light interpretation, proper stopping at junctions, maintenance of safe distances, lane changing, and highway behaviour, including the protocol for returning to the left lane.

3.3.1 World Observation

The Python code implements an observation mechanism for traffic lights and the surrounding world within the CARLA simulation environment. The objective of this mechanism is to enable autonomous vehicles to perceive and react to traffic signals and environmental cues effectively. This subsection discusses the functionality, purpose, benefits, disadvantages, and underlying logic of the implemented observation system.

Figure 3.3: UML Diagram

Functionality

The code consists of a few main classes: `TrafficLightObserver`, `RoadSignObserver`, `SpeedObserver` and `WorldObserver`, all inheriting from the abstract base class `Observer`. The `TrafficLightObserver` class focuses on detecting and interpreting traffic light signals. It utilizes the CARLA simulation environment's functionalities to access the state of nearby traffic lights, calculate distances to relevant waypoints, and determine the vehicle's proximity to each traffic signal. Based on this information, the `TrafficLightObserver` class generates beliefs and intentions regarding the vehicle's interaction with traffic lights, such as approaching or waiting at a red light.

The `RoadSignObserver` class focuses on detecting and interpreting various road signs present within the simulated environment. It leverages CARLA's capabilities to detect and recognize different types of road signs, including regulatory signs, warning signs, and informational signs. Upon detecting a road sign, the `RoadSignObserver` class analyzes its content and translates it into meaningful beliefs and intentions for the autonomous vehicle. These beliefs and intentions include actions such as obeying specific traffic instructions or adjusting the vehicle's behaviour based on road conditions indicated by the signs.

The `SpeedObserver` class is responsible for monitoring and interpreting the vehicle's speed-related information within the CARLA simulation environment. It utilizes functionalities provided by CARLA to access the vehicle's current speed, calculate acceleration or deceleration rates, and assess the adherence to speed limits imposed by road regulations or driving conditions.

The `WorldObserver` class acts as a higher-level perception module, coordinating multiple observation components, to provide a comprehensive understanding of the vehicle's surroundings. By aggregating beliefs and intentions from individual observers, the `WorldObserver` class generates a holistic perception context for the autonomous vehicle, enabling it to make informed decisions based on the observed environmental cues.

Logic and Reasoning

The observation mechanism employs a logic-driven approach to perceive and interpret relevant information from the simulation environment. By iteratively scanning through available traffic lights and evaluating their proximity, orientation, and state, the system determines the vehicle's spatial relationship to each traffic signal. This spatial reasoning enables the system to identify approaching traffic lights and anticipate their state changes, guiding the vehicle's decision-making process accordingly.

The modular design of the observation system promotes the separation of concerns and encapsulation of functionality, adhering to object-oriented design principles. Each observation component, such as the `TrafficLightObserver` and `SpeedObserver` classes, encapsulates specific perception tasks while maintaining loose coupling with other system components. This design facilitates code organization, testing, and extensibility, promoting a scalable and maintainable architecture.

The `TrafficLightObserver` in particular scans through available traffic lights in the

simulation environment and evaluates their proximity, orientation, and state. Based on this information, it determines the vehicle's spatial relationship to each traffic signal and identifies approaching traffic lights. Its design is driven by the requirements of RoTRA and the specific needs of the observation mechanism. It integrates with RoTRA to provide real-time updates on traffic signals and utilizes this information to guide the vehicle's decision-making process.

Purpose

The primary purpose of the observation mechanism is to facilitate decision-making for autonomous vehicles by providing them with real-time information about their surroundings, particularly regarding traffic lights. By accurately perceiving traffic signals and the surrounding environment, autonomous vehicles can make informed decisions regarding speed adjustments, lane changes, and stopping actions, contributing to safer and more efficient navigation within the simulation environment.

Disadvantages

One potential disadvantage of the observation mechanism is its reliance on predefined parameters, such as maximum observation distance and angle thresholds. While these parameters are essential for filtering out irrelevant information and optimizing computational efficiency, they may require fine-tuning to accommodate varying traffic scenarios and environmental conditions effectively. Inadequate parameter settings could lead to missed detections or false positives, impacting the system's reliability and performance. Additionally, the current implementation focuses primarily on traffic light observation, potentially overlooking other critical environmental cues, such as pedestrian crossings or dynamic obstacles. While the system's modularity allows for expansion to incorporate additional perception capabilities, such enhancements would require additional development effort and may introduce complexity to the overall architecture.

3.3.2 RoTRA Client

The RoTRA Client serves as a bridge between users and the RoTRA server, enabling the retrieval of pertinent information such as actions, beliefs, and intentions in accordance with predefined driving rules and regulations. It encapsulates a robust set of function-

alities aimed at providing users with access to the rich repository of driving rules and regulations stored within the RoTRA server.

Structure and Initialization

At the core of the RoTRA Client lies the `Client` class, designed to establish and manage connections with the RoTRA server. Upon instantiation, the `Client` class requires the server address and port number to establish a connection. This initialization process sets the foundation for subsequent interactions with the server.

Querying Actions, Beliefs, and Intentions

The primary function of the RoTRA Client is to facilitate the retrieval of actions, beliefs, and intentions based on specific contexts. The `get_action` method serves as a gateway for users to query the server for actions corresponding to provided beliefs and intentions. Belief and intention codes, represented as lists, are converted to their respective enum values before being dispatched to the server for processing.
In addition to querying individual actions, beliefs, and intentions, comprehensive methods for fetching all available data within the system are offered by the RoTRA Client. The `get_all_actions`, `get_all_beliefs`, and `get_all_intentions` methods enable users to retrieve a complete set of actions, beliefs, and intentions, respectively. These methods provide users with a holistic view of the driving rules and regulations encapsulated within the RoTRA system.

Seamless Communication with RoTRA Server

Communication with the RoTRA server is streamlined by the RoTRA Client, encapsulating HTTP requests for fetching data. By using the `requests` library, GET and PATCH requests are sent by the client to designated endpoints on the server to retrieve and update information as needed. This seamless communication ensures efficient data exchange between the client and server components of the RoTRA system.

3.3.3 Action Mapper

The Action Mapper serves as a crucial component in putting together the behaviour of autonomous vehicles within the CARLA simulation environment. It acts as a bridge

between high-level decision-making algorithms and the low-level control commands necessary for vehicle manoeuvring, ensuring safe and efficient navigation through complex traffic scenarios.

System Overview and Initialization

At the heart of the Action Mapper system lies a comprehensive set of classes and functions designed to establish connections with external sources, manage action mappings, and execute control commands on autonomous vehicles. Upon initialization, the system instantiates the `ActionMapper` class, which defines a variety of predefined control actions along with their respective priorities. For example, a `ControlAction` object with a high priority might represent an emergency braking manoeuvre, while a lower-priority action could correspond to a gentle acceleration.

Mapping Actions and Prioritization

The system offers a centralized mapping mechanism through the `ActionMapper` class, facilitating the translation of high-level action codes into actionable control objects. Consider an action code `SLOW_DOWN` indicating the need to reduce vehicle speed. The corresponding `ControlAction` object will specify a slight deceleration by adjusting throttle and brake values accordingly. Additionally, each action is assigned a priority level, allowing for dynamic prioritization based on the perceived environment and decision-making logic. For instance, in a scenario where the vehicle detects an imminent collision, actions related to collision avoidance would be prioritized over routine manoeuvres like lane changing.

Retrieval of Actions and Beliefs

The primary function of the system is to facilitate the retrieval of actions, beliefs, and intentions based on specific contexts. Users can query the system for individual actions corresponding to provided beliefs and intentions or fetch comprehensive sets of actions, beliefs, and intentions using dedicated methods. Belief and intention codes are converted to their respective enum values before being dispatched for processing. For example, if the vehicle perceives a pedestrian crossing ahead, it triggers an intention to slow down (`SLOW_DOWN`), leading to the retrieval of appropriate control actions from the system.
Action Mapper's role is to enhance the autonomy and responsiveness of autonomous

vehicles within the CARLA simulation environment. Bridging the gap between high-level decision-making and low-level control commands, the system promotes the coherent execution of control behaviours and facilitates adaptability to dynamic traffic scenarios.

Action Mapper Table

To provide insight into the mapping between high-level action codes and their corresponding low-level inputs, Table 3.1 is presented. The table outlines various action codes alongside the associated throttle, brake, steer, and hand brake inputs.

Action Code	Throttle	Brake	Steer	Hand Brake
STOP	0.0	1.0	0.0	False
SLOW DOWN	0.0	0.2	0.0	False
ACCELERATE	0.7	0.0	0.0	False
REVERSE	-0.2	0.0	0.0	False
EMERGENCY BRAKE	0.0	1.0	0.0	False
IDLE	0.0	0.0	0.0	True
NEUTRAL	0.0	0.0	0.0	False

Table 3.1: Action Codes and Control Inputs

These control inputs are utilized by the Action Mapper system to execute appropriate actions based on the current environment and decision-making logic. For instance, when an action code indicating the need to slow down is received, it is translated into specific throttle and brake values to ensure a controlled reduction in vehicle speed.

3.3.4 CARLA SDK

The CARLA Simulation Development Kit (SDK) serves as the foundation for building realistic and interactive simulations of urban environments for autonomous driving research and development. Its comprehensive set of APIs and tools enables developers to create complex scenarios, control vehicles, and simulate various aspects of urban traffic.

Integration with Observer Components

By integrating the CARLA Software Development Kit (SDK), observer components like the `TrafficLightObserver` and `WorldObserver` are significantly impacted in their design and functionality. This integration allows observers to access real-time information about the simulated environment, including traffic light status, vehicle position, and road

network layout. The `TrafficLightObserver`, for example, uses SDK functions to check for nearby traffic light statuses and calculate distances to relevant waypoints. This enables accurate perception of traffic signals and surrounding road conditions. In the same way, the `WorldObserver` uses SDK features to retrieve map data, access vehicle positions, and coordinate multiple observation components. This provides a comprehensive understanding of the vehicle's surroundings.

Integration with Action Mapper

The Action Mapper component is heavily influenced by the capabilities and features of the CARLA SDK. This component can define and map control actions based on high-level intentions and beliefs by interfacing with the SDK's APIs for vehicle control and simulation management.

For example, the Action Mapper uses SDK functions to apply throttle, brake, and steering commands to simulated vehicles in response to perceived environmental cues. It can orchestrate various manoeuvres such as accelerating, decelerating, or stopping. Additionally, the Action Mapper leverages SDK functionalities for the dynamic mapping of action codes to predefined control behaviours, which allows seamless integration with decision-making algorithms and perception modules.

3.4 Example: Autonomous Vehicle Encountering a Red Light

Let's explore a scenario where an autonomous vehicle (AV) equipped with the RoTRA system encounters a red traffic light. This example illustrates the sequence of steps involved in perceiving the environment, retrieving appropriate actions, and executing control commands to respond to the red light.

1. **Observation Phase**:
    - The `TrafficLightObserver` utilizes the CARLA SDK to gather information regarding the position and state of nearby traffic lights. It retrieves the AV's position using the `get_transform()` function and computes the distance between the vehicle and each traffic light to assess their proximity. A traffic light is considered "approached" if it satisfies the following conditions:

- It is located within a 15-meter radius of the AV.
- It is positioned on the same road as the AV.
- It is facing towards the direction of the vehicle's movement.
- It falls within a 90-degree radius cone in front of the vehicle.

Upon meeting all these criteria, the observer signals the intention as "approachingTrafficLight" and the belief as "lightRed".

- Similarly, the `WorldObserver` accesses the CARLA SDK to retrieve the AV's position, orientation, and surrounding map data. This information is used to generate a perception context.

2. **Action Retrieval**:
   - The AV's decision-making module sends a request to the RoTRA server via the RoTRA Client, specifying the detected belief (red light ahead) and the intention to stop at the intersection.
   - Upon receiving the request, the RoTRA server processes the information and maps it to an appropriate action code based on predefined rules. In this case, it returns the action code `[must-stop_at_white_line]` to instruct the AV to come to a complete stop at the red light.
3. **Response Generation**:
   - The RoTRA Client receives the action code and forwards it to the Action Mapper component.
   - The Action Mapper translates the action code into specific control inputs using predefined mappings, in thic case `STOP`. The `STOP` action is mapped to zero throttle, full brake, and straight steering values to ensure a controlled stop at the intersection.
4. **Control Execution**:
   - The AV's control system receives the generated control inputs and applies them to the vehicle's dynamics through the CARLA SDK.
   - Using the provided throttle, brake, and steering values, the AV reduces throttle to zero, engages the brakes with appropriate intensity to halt the vehicle, and adjusts steering to maintain lane alignment.
   - The AV's behavior is executed within the CARLA simulation environment, simulating the real-world response to the perceived red light.

Results and Discussion

4.1 Challenges

Developing and implementing the CARLA-based system presented several complex challenges that required attention and creative solutions. One major obstacle was that my program did not have a direct one-to-one mapping with CARLA's Autonomous Vehicle (AV) Pilot, the Traffic Manager. Unlike some systems where control can be seamlessly handed over between components, my system could only intervene indirectly by stopping and restarting the AV Pilot, effectively overriding its actions. This made implementation more complicated, requiring careful thought to ensure smooth interaction between the system and CARLA's AV Driver.

Another significant challenge was the realization that the RoTRA framework, while powerful, lacked the comprehensive framework necessary for full-fledged self-driving capabilities. RoTRA operates on predefined traffic rules and regulations but needs to be based on another AV framework to be effective. The choice of underlying AV framework could significantly influence the system's performance, leading to potential variations in results depending on the technology selected.

Additionally, RoTRA relies heavily on the same sensors used for camera detections within the CARLA environment. While this setup works well in the simulation environment, where there is a general understanding of the state of objects in the environment, it poses challenges in real-world applications. In a real-world scenario, RoTRA is dependent on the car's sensors, and if these sensors fail to detect an object or obstacle, RoTRA will fail to detect it as well. The primary purpose of RoTRA is to enhance the judgment and decision-making capabilities of autonomous vehicles rather than serving as a standalone detection system.

4.2 Simulation Environment Customization

Setting up diverse environments with varying scenarios is crucial for comprehensive testing and evaluation within the CARLA simulation environment. This section delves into the detailed process of customizing environments to replicate real-world conditions and simulate a wide range of driving scenarios.

4.2.1 Environment Configuration

The first step in environment customization involves configuring the basic parameters of the simulation, such as weather conditions, time of day, and geographical location. CARLA provides extensive options for weather simulation, including parameters for rain, fog, wind, and dynamic changes in weather patterns over time. By adjusting these parameters, a wide spectrum of weather conditions, from clear skies to heavy rainstorms, can be simulated to evaluate the system's robustness under different weather scenarios.
One critical parameter affected by weather conditions is braking distance. In adverse weather conditions such as heavy rain visibility is reduced, and road surfaces become slippery, leading to increased braking distances. This phenomenon occurs due to reduced tyre traction and decreased friction between the tyres and the road surface. As a result, vehicles require more time and distance to come to a complete stop, increasing the risk of accidents.
To illustrate the impact of weather on braking distance, the following plot compares braking distances under different weather conditions:

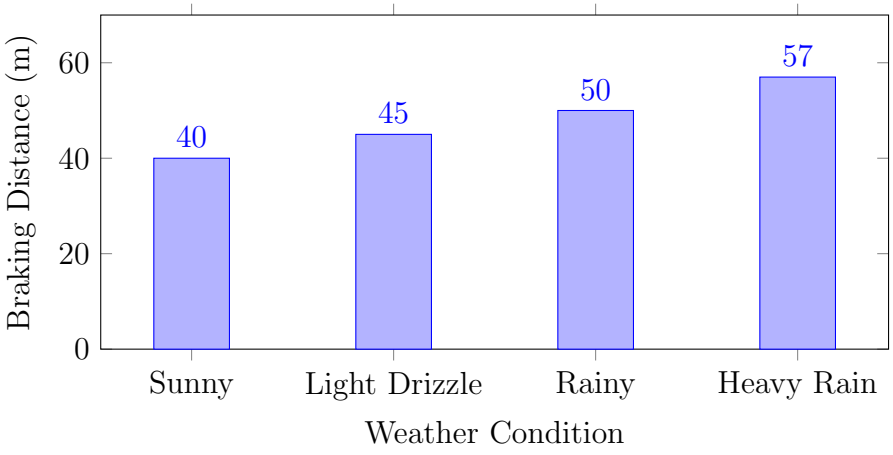


Figure 4.1: Braking Distance at 40 mph under Different Weather Conditions

Figure 4.1 demonstrates how braking distance varies under different weather conditions,

from a sunny dry day to heavy rain conditions. It's evident that adverse weather significantly increases braking distance compared to clear weather conditions. These insights are crucial for evaluating the system's performance and optimizing its responsiveness under various weather scenarios.

Geographical location settings in the CARLA simulation environment offer the flexibility to select predefined maps that represent diverse urban, suburban, and rural environments. These maps serve as virtual replicas of real-world locations, featuring unique road layouts, infrastructure, landmarks, and traffic patterns.

For example, urban maps (Figure: 4.2) may include bustling city streets with complex intersections, pedestrian crossings, and high-rise buildings. Suburban maps (Figure: 4.3) may feature residential neighbourhoods with tree-lined streets, cul-de-sacs, and suburban infrastructure such as shopping centres and schools. Rural maps (Figure: 4.4), on the other hand, may depict open countryside with winding roads, farmland, and occasional rural settlements.

To provide a visual representation of these diverse environments, three examples of maps from a top view have been attached. These maps showcase the varied terrain, road layouts, and infrastructure found in urban, suburban, and rural settings. By testing the system in these different scenarios, the program can be assessed for its adaptability to various driving environments and refined for its capabilities accordingly. These maps serve as valuable tools for validating the system's performance and ensuring its readiness for real-world deployment.

4.2.2 Scenario Design

Once the basic environment parameters were configured, the next step was to design specific driving scenarios to evaluate the system's performance in various situations. These scenarios aim to replicate real-world driving challenges, such as navigating intersections, responding to dynamic traffic conditions, and interacting with other vehicles and pedestrians.

Intersection Scenarios

RoTRA plays a critical role in intersection scenarios by intervening when traffic rules are not obeyed. For instance, if the AV fails to stop at a red light, RoTRA overrides

Figure 4.2: View of Urban Map

its actions and ensures compliance with traffic signals. Similarly, RoTRA intervenes to yield to oncoming traffic, navigate multi-lane junctions safely, and negotiate roundabouts and traffic circles. Therefore, by detecting violations and enforcing adherence to traffic regulations, RoTRA improves the system's ability to navigate intersections securely and efficiently.

Urban Driving Scenarios

In urban driving simulations, RoTRA intervenes to prevent violations of traffic rules and regulations. For example, if the AV attempts an illegal manoeuvre such as ignoring a stop sign or exceeding the speed limit in a residential area, RoTRA takes control and enforces compliance. By closely monitoring the AV's behaviour and intervening when necessary, RoTRA ensures the safety of pedestrians, and other road users while navigating through dense city environments.

Figure 4.3: View of Suburban Map

Figure 4.4: View of Rural Map

Highway and Rural Scenarios

RoTRA's intervention capabilities are also crucial in highway and rural environments to prevent deviations from highway code regulations. If the AV fails to maintain proper lane discipline, follow overtaking regulations, or adhere to speed limits, RoTRA steps in to correct its behaviour. Additionally, RoTRA monitors environmental conditions such as road surface conditions and weather, with the possibility of adjusting the AV's speed and driving strategy accordingly to ensure safe navigation in the future development of the program.

4.2.3 Iterative Testing and Refinement

Once scenarios were designed and implemented, iterative testing and refinement were conducted to identify issues, optimize system parameters, and improve overall performance. Feedback from testing sessions was used to fine-tune perception algorithms, behaviour models, and decision-making strategies, ensuring that the system can effectively handle a wide range of driving scenarios with accuracy and reliability. Customizing environments with diverse scenarios and utilizing CARLA's simulation capabilities enabled me to conduct thorough testing and evaluation of the autonomous driving system. This effort contributes to advancing the development of safe and robust autonomous vehicles for real-world deployment.

4.3 Data Analysis

4.3.1 Visualization Techniques

Visualization techniques were employed to represent the collected data in a visually intuitive manner, facilitating the identification of patterns, trends, and anomalies in system behaviour. Various types of visualizations, including scatter plots, line charts, and histograms, were utilized to explore different aspects of system performance.

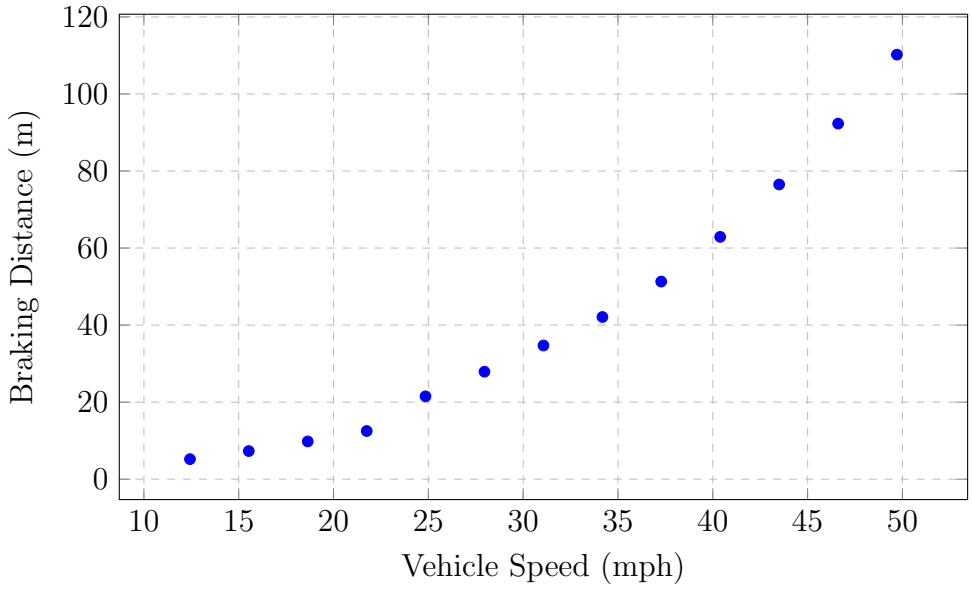


Figure 4.5: Scatter Plot: Vehicle Speed vs. Braking Distance

For example, scatter plots (Figure: 4.5) were used to visualize the relationship between

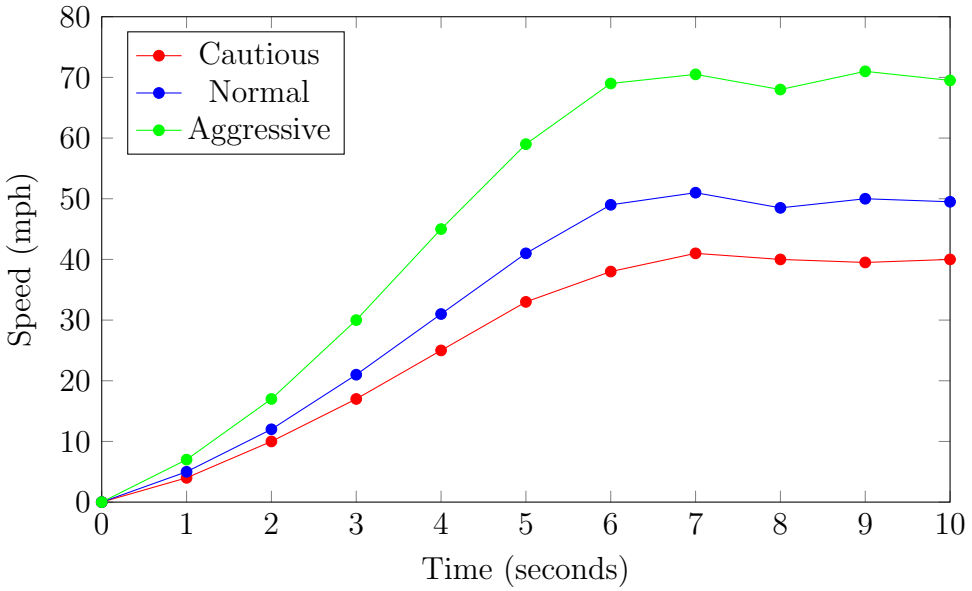


Figure 4.6: Speed of Different Behaviour Agents

input parameters (e.g., vehicle speed, distance to the nearest obstacle) and output metrics (e.g., braking distance, reaction time), allowing for the identification of potential correlations or dependencies. Line charts (Figure: 4.6) were employed to track the temporal evolution of key performance metrics over successive iterations or simulation runs, highlighting trends and fluctuations in system behaviour over time.

Histograms (Figure: 4.7) were utilized to visualize the distribution of observed behaviours or events, such as the frequency distribution of vehicle stopping distances or the distribution of observed traffic signal states (e.g., green, yellow, red).

4.3.2 Insights and Optimization Opportunities

The results of the data analysis provided valuable insights into the strengths and weaknesses of the CARLA-based system, informing potential areas for optimization and refinement. Visualizations highlighted specific areas for optimization, such as fine-tuning action mapping parameters to improve the consistency and reliability of vehicle behaviour in complex traffic scenarios.

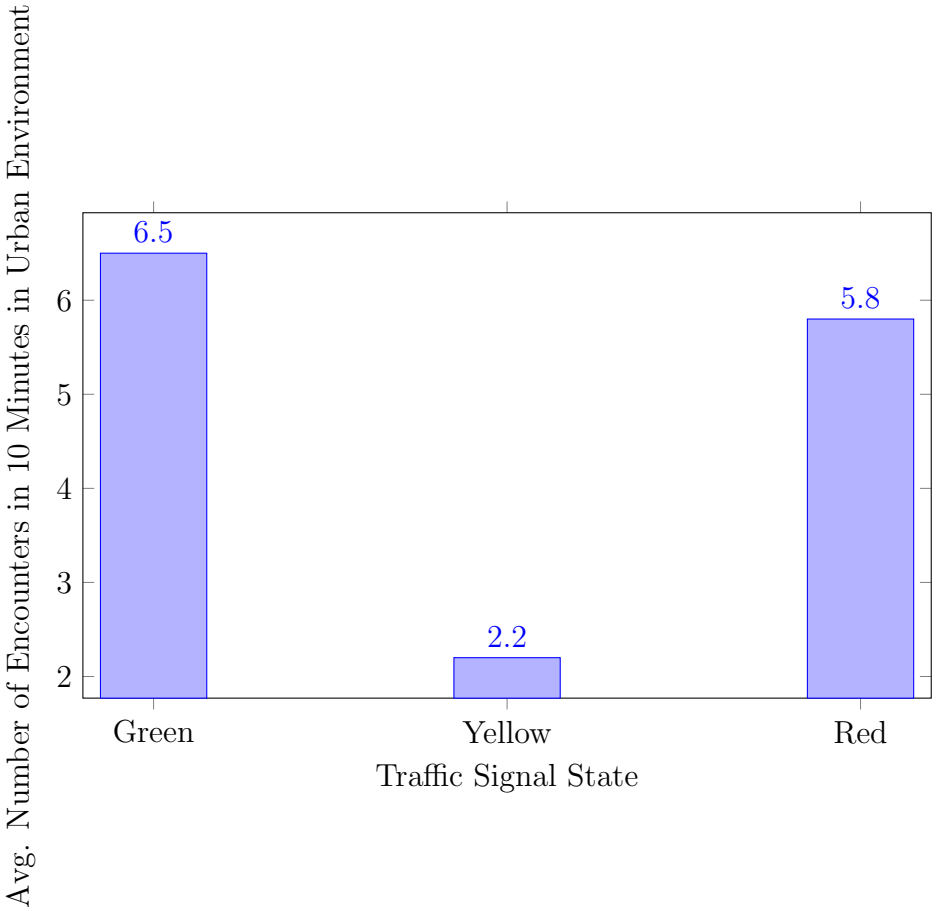


Figure 4.7: Histogram: Distribution of Traffic Signal States

4.4 Performance Evaluation

4.4.1 Manual Testing

In the manual testing phase, the CARLA system was evaluated by exposing it to diverse scenarios in various simulated environments. Different cities within the CARLA simulator were chosen, each with various road layouts, traffic conditions, and environmental factors. Additionally, the vehicle was tested at various junctions, including intersections, roundabouts, and pedestrian crossings, to assess its ability to navigate complex traffic scenarios.

The vehicle's performance was scrutinized under different driving conditions, such as varying speeds, traffic densities, and weather conditions. Scenarios involving high-speed dual carriageways, congested urban streets, and winding rural roads were simulated to evaluate the system's adaptability and responsiveness across different driving environments.

The system was subjected to a range of traffic scenarios, including scenarios with heavy traffic congestion, sudden lane changes, and pedestrian crossings. The system was exposed to these diverse scenarios to assess its ability to perceive and react to dynamic and unpredictable elements in the environment accurately.

Each testing scenario was designed to replicate real-world driving conditions as closely as possible, with careful consideration given to factors such as road signage, lane markings, traffic light timings, and the behaviour of other simulated vehicles and pedestrians. By systematically varying these parameters across different scenarios, potential weaknesses or limitations in the system's perception, decision-making, and control capabilities were aimed to be uncovered.

Manual testing was conducted to assess the observer functionality, action mapping accuracy, and overall system responsiveness of the CARLA system. During testing, usability issues related to observer perception, such as occasional misinterpretation of traffic signals, were discovered. Minor bugs in the action mapping logic were also found, resulting in inconsistent vehicle behavior in certain scenarios. The goal of manual testing was not only to evaluate the functionality of the system's implemented modules but also to identify any weak links in CARLA's AV, where RoTRA could make a difference.

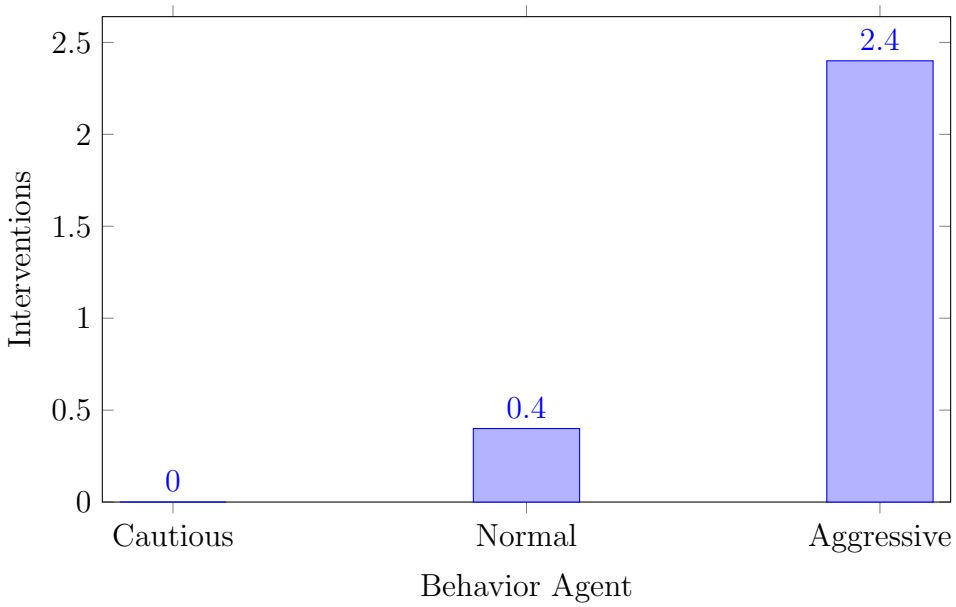


Figure 4.8: Average Number of Interventions by ROTRA for the Different Behavior Agents in 10-Minute Simulations

4.4.2 Behavior Agent Analysis

In addition to traditional testing methods, an important factor in evaluating the CARLA system's performance was the behaviour agent of the autonomous vehicles (AVs). CARLA provides three types of behaviour agents: Cautious, Normal, and Aggressive.

When AVs were set to Cautious behaviour, RoTRA had minimal impact, primarily serving as validation for the system's behaviour. Similarly, in Normal behaviour mode, RoTRA's influence was limited, although its importance slightly increased due to the higher speed at which the cars moved. However, since the AVs inherently erred on the cautious side, RoTRA did not significantly alter their behaviour.

In contrast, when AVs were set to Aggressive behaviour, RoTRA played a more significant role, particularly in stopping cars at red lights. The behaviour agent influences factors such as distance to the car in front, braking intensity for traffic lights, driving above the speed limit, tailgating, and overall aggressiveness on the road.

Graphical representations (Figure: 4.8) were utilized to illustrate the impact of different behaviour agents on the performance of the CARLA system, highlighting the effectiveness of RoTRA in influencing AV behaviour in scenarios where aggressiveness was a defining factor.

Through these graphical representations, insights into how AV behaviour was influenced by different behaviour agents, particularly in scenarios where aggressiveness played a significant role, were provided. The efficacy of RoTRA in shaping AV behaviour was observed, especially in situations such as stopping at red lights, yielding to pedestrians and overtaking when AVs were set to "Aggressive" behaviour. By comparing the performance of AVs with and without RoTRA intervention, the impact of RoTRA on improving compliance with traffic regulations and enhancing overall safety on the road could be quantified.

Due to the increased chance of the AV making a mistake in the aggressive behaviour scenario, coupled with potentially worse results in real life, from now on all testing will reflect the aggressive behaviour.

4.4.3 Safety Score Calculation

Definition

The safety score is a numerical value ranging from 0 to 100 that reflects the overall safety performance of autonomous vehicles (AVs) in a specific driving scenario. A higher safety score indicates better safety performance, while a lower score suggests potential areas for improvement in safety-related aspects.

Calculation Formula

The safety score can be calculated using a weighted combination of different safety metrics:

$$SS = w_1 \times CTR + w_2 \times CAS \tag{4.1}$$

Where:

- w_1, w_2: Weighting factors representing the relative importance of each safety metric. These weights sum up to 1.
- CTR: Compliance with Traffic Regulations, ranging from 0 to 100.
- CAS: Collision Avoidance Score, also ranging from 0 to 100.

Safety Metrics

1. **Compliance with Traffic Regulations (CTR):**

$$CTR = \frac{\text{Number of compliant actions}}{\text{Total number of actions}} \times 100 \tag{4.2}$$

To assess Compliance with Traffic Regulations (CTR), a combination of the observer module and output from RoTRA was used. The observer module continuously monitored the position, direction of travel, velocity of the AV, and the state of objects such as traffic lights. A detection algorithm was created to identify instances where traffic rules were violated based on the observed data. At the same time, RoTRA provided recommendations for actions based on driving rules, and this output was recorded.

This approach allowed for two objectives to be achieved. Firstly, it evaluated RoTRA's performance in enforcing compliance with traffic regulations. Secondly, it monitored the correctness of RoTRA's recommendations. For instance, if the observer class detected a violation such as running a red light but RoTRA did not output any action, it raised concerns about the accuracy of RoTRA's decision-making process.

2. **Collision Avoidance Score (CAS):**

$$CAS = 100 - \left(\frac{\text{Number of collision incidents}}{\text{Total number of testing scenarios}} \times 100 \right) \tag{4.3}$$

CAS was determined using a collision sensor implemented in the CARLA simulation environment. This collision sensor detects collisions per object per frame, providing real-time information on collision incidents during simulated scenarios. The number of collision incidents recorded by the collision sensor served as the basis for calculating CAS.

Weighting Factors

The weighting factors (w_1, w_2) reflect the relative importance of each safety metric in determining the overall safety score. These weights can be adjusted based on the specific priorities and objectives of the safety evaluation process. Typically, the weighting factors

are determined through judgment, iterative refinement or trial and error.

Initially, equal weighting ($w_1 = w_2 = 0.5$) was considered, as each safety metric was deemed equally important in assessing overall safety performance. However, after conducting preliminary analyses and evaluating the relative impact of each metric, adjustments were made to the weighting factors to better reflect their importance in the context of the specific driving scenarios tested.

w1 Value	**Urban SS**	**Suburban SS**	**Rural SS**
0.1	75	85	92
0.2	79	86	90
0.3	86	85	90
0.4	85	87	89
0.5	81	86	88
0.6	82	86	82
0.7	84	87	79
0.8	78	88	71
0.9	72	88	65

Table 4.1: Safety Score for Weight Calculation in Different Scenarios Averaged

Table 4.1 presents the safety scores for different scenarios averaged over various weighting factors ($w1$) ranging from 0.1 to 0.9. In conducting the evaluation, scenarios were simulated across urban, suburban, and rural environments, with routes randomized and test durations between 10 and 20 minutes.

Observations from the table suggest that a $w1$ value of 0.3 strikes a balance between compliance rate and collision avoidance. In urban environments, where traffic density and the potential for rule violations are high, safety scores peaked around 85, indicating the need for a higher emphasis on either compliance rate or collision avoidance. Conversely, in rural environments, where fewer rules are typically broken but accidents may have more severe consequences due to higher speeds, the importance of compliance rate diminishes, and collision avoidance becomes more critical.

Therefore, a bias towards the collision avoidance metric was chosen, with a $w1$ value of 0.3 and a $w2$ value of 0.7, ensuring they sum to 1. Moving forward, all tests will reflect this weighted metric, prioritizing collision avoidance while still considering compliance rate.

Example Calculation

Suppose we have the following hypothetical data for an AV tested in an urban driving scenario:

- Compliance with Traffic Regulations (CTR) = 85
- Collision Avoidance Score (CAS) = 90

Assuming equal weighting for all safety metrics ($w_1 = w_2 = 1/2$), the safety score (SS) can be calculated as follows:

$$SS = \frac{1}{2} \times 85 + \frac{1}{2} \times 90 = 87.5 \tag{4.4}$$

Therefore, the safety score for the AV in the urban driving scenario is 87.5 out of 100.

4.4.4 Dataset for Safety Score Calculation

Table 4.2 provides a comprehensive dataset illustrating the performance of CARLA's autonomous vehicles (AVs) under different driving scenarios with and without RoTRA oversight, particularly focusing on scenarios characterized by aggressive behaviour.

Scenario	**AV Behaviour**	**Safety Score**
Urban	No RoTRA	78
	RoTRA	86
Suburban	No RoTRA	79
	RoTRA	88
Rural	No RoTRA	75
	RoTRA	89

Table 4.2: Average Safety Score

1. **Scenario:** The driving scenarios tested include urban, suburban, and rural environments. These scenarios encompass diverse road conditions and traffic densities, allowing for a holistic evaluation of AV performance across various settings.
2. **AV Behavior:** The dataset distinguishes between AV behaviour with RoTRA oversight and without RoTRA oversight. This differentiation enables a comparative analysis of AV performance under different levels of regulatory supervision.

3. **Safety Score:** The safety score represents the overall safety performance of AVs in each scenario, measured on a scale from 0 to 100. A higher safety score indicates superior safety performance, reflecting factors such as adherence to traffic regulations, collision avoidance, and responsiveness to unexpected events.

The information in Table 4.2 provides valuable insights into the performance of CARLA's autonomous vehicles (AVs) in different driving scenarios. It compares scenarios with and without RoTRA oversight, particularly in situations where aggressive behaviour is prevalent.

After analyzing the data, it becomes clear that using appropriate weight values for the safety metrics leads to improvements in all tested scenarios. The introduction of RoTRA oversight significantly enhances the safety performance of AVs, resulting in higher safety scores compared to scenarios without RoTRA.

The impact of RoTRA, however, varies across different scenarios. In rural environments, where spacious road conditions can lead to exceeding the speed limit, RoTRA plays a proactive role in preventing speeding and mitigating collision risks. This proactive intervention, combined with unfavourable weather conditions such as heavy rain, results in a notable reduction in collision incidents.

On the other hand, in urban environments, where achieving higher speeds is more challenging, RoTRA's influence may be less pronounced. However, the aggressive nature of AVs in urban settings makes them prone to violations such as running red lights. In such cases, RoTRA intervenes by applying energy brake commands to prevent collisions, thereby enhancing overall safety.

Furthermore, RoTRA helps improve the quality of driving behaviour by ensuring appropriate distances between vehicles are maintained and preventing instances of speeding. This proactive approach helps prevent the creation of potentially dangerous situations on the road, further contributing to the overall safety of AV operations.

For more detailed insights into the diverse scenarios where RoTRA was employed, Tables A.1, A.2 and A.3 in the appendix provide comprehensive information about the various cases encountered. These tables offer additional context and detailed analysis of RoTRA's performance across a range of driving scenarios, including different weather conditions and intersections.

Conclusion

5.1 Summary

The main objective of this dissertation was to determine if a BDI system, specifically RoTRA (Rules of the Road Agent), can improve the behaviour of an autonomous vehicle (AV). To achieve this objective, a modular architecture was created to support the driving code for various environments, along with a Prolog server to serve as the decision engine for the AV's actions.

After implementing the four main components of the system - Observer, RoTRA Client, Action Mapper, and Carla SDK - the program was thoroughly tested in different scenarios, including different map types (urban, rural), weather conditions (sunny, rainy), and types of intersections with varying behaviour agents.

The results of the testing phase showed that a BDI system can assist an AV in making the right decisions. The safety scores obtained for different scenarios demonstrated improvements across the board, indicating that the decision engine can aid the AV in taking appropriate actions when faced with complex real-world scenarios.

5.2 Key Findings

The dissertation reveals several significant findings, shedding light on the efficacy of BDI systems in enhancing autonomous vehicle (AV) behaviour and safety. These key findings include:

- **Benefit of BDI Systems:** The research underscores the potential benefits of employing BDI systems in AVs. Incorporating rule-based decision-making and adaptive behaviour, BDI systems contribute to improved AV performance and safety across

various driving scenarios.

- **Influence of Behavior Agents:** Analysis of different behaviour agents—Cautious, Normal, and Aggressive—underscored the varying impact of RoTRA based on AV behaviour. Notably, RoTRA played a more substantial role in scenarios where AVs exhibited aggressive behaviour, particularly in ensuring compliance with traffic signals and preventing collisions.
- **Optimized Safety Metrics:** The safety score calculation method, incorporating Compliance with Traffic Regulations (CTR) and Collision Avoidance Score (CAS), provides a comprehensive assessment of AV safety performance. Through iterative refinement of weighting factors, an optimized safety metric was established, prioritizing collision avoidance while considering compliance rate.
- **Scenario-specific Impact of RoTRA:** Analysis of the safety score dataset highlighted the nuanced impact of RoTRA across different driving scenarios. In rural environments, RoTRA proactively prevents speeding and mitigates collision risks, especially in adverse weather conditions. In contrast, in urban settings, RoTRA interventions primarily focus on enforcing traffic regulations to prevent collisions.
- **Flexibility and Adaptability:** The modular architecture of the developed system allows for flexibility and adaptability to diverse driving conditions. This modular approach enables the seamless integration of new rules and behaviours, enhancing the system's responsiveness and effectiveness.

5.3 Future Work

In this section, potential avenues for future research and development to enhance the capabilities and performance of the proposed system are discussed:

- **Modular Expansion of Observer Modules:** Modularity could be enhanced by implementing additional observer modules for various other rules (e.g., pedestrian crossings) to broaden understanding of traffic regulations and environmental cues. This modular approach would allow for easy extension and customization, enabling adaptation to diverse scenarios and regulatory frameworks.
- **Expansion of Action Mapper Functionality:** Adding more actions to the action mapper would enable the interpretation of a broader range of actions returned from

the Prolog server. By expanding the repertoire of available actions, responses to complex scenarios and dynamic environments can be improved, enhancing overall decision-making and adaptability.

- **Integration of Vision Detection Algorithms:** Reliance on general map knowledge could be removed by implementing separate vision detection algorithms to improve consistency and accuracy. These algorithms could detect various world objects (e.g., traffic lights, signs, crossings) independently, enhancing perception capabilities. Introducing machine learning algorithms for vision detection, using the current output of RoTRA as part of the training data, could further improve detection accuracy and reliability.
- **Real-time Decision-Making Adaptation:** Mechanisms for real-time adaptation of decision-making strategies could be implemented to enhance responsiveness to dynamic environments. Techniques such as reinforcement learning or adaptive control algorithms could be explored to enable learning and adjustment of behaviour based on real-time feedback and environmental changes.

Bibliography

[1] Sara Abdallaoui, Halima Ikaouassen, Ali Kribèche, Ahmed Chaibet, and El-Hassane Aglzim. Advancing autonomous vehicle control systems: An in-depth overview of decision-making and manoeuvre execution state of the art. *The Journal of Engineering*, 2023(11):e12333, 2023.

[2] Sara Abdallaoui, Ali Kribèche, and El-Hassane Aglzim. Comparative study of mpc and pid controllers in autonomous vehicle application. In *International Symposium on Automation, Mechanical and Design Engineering*, pages 133–144. Springer, 2021.

[3] Driving Standards Agency. *The highway code*. The Stationery Office, 2004.

[4] Hilal Al Shukairi and Rafael C Cardoso. Ml-mas: a hybrid ai framework for self-driving vehicles. In *AAMAS'23: Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2023.

[5] Vibha Bharilya and Neetesh Kumar. Machine learning for autonomous vehicle's trajectory prediction: A comprehensive survey, challenges, and future research directions. *Vehicular Communications*, page 100733, 2024.

[6] Joe Collenette. Rulesoftheroadexperiments. `https://github.com/JoeCol/RulesOfTheRoadExperiments`, 2022.

[7] Joe Collenette, Louise A Dennis, and Michael Fisher. Advising autonomous cars about the rules of the road. *arXiv preprint arXiv:2209.14035*, 2022.

[8] Srishti Dikshit, Areeba Atiq, Mohammad Shahid, Vinay Dwivedi, and Aarushi Thusu. The use of artificial intelligence to optimize the routing of vehicles and reduce traffic congestion in urban areas. *EAI Endorsed Transactions on Energy Web*, 10, 2023.

[9] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual*

Conference on Robot Learning, pages 1–16, 2017.

[10] Carlos Gómez-Huélamo, Javier Del Egido, Luis M Bergasa, Rafael Barea, Elena López-Guillén, Felipe Arango, Javier Araluce, and Joaquín López. Train here, drive there: Ros based end-to-end autonomous-driving pipeline validation in carla simulator using the nhtsa typology. *Multimedia Tools and Applications*, pages 1–28, 2022.

[11] Tianyu Gu, John M Dolan, and Jin-Woo Lee. On-road trajectory planning for general autonomous driving with enhanced tunability. In *Intelligent Autonomous Systems 13: Proceedings of the 13th International Conference IAS-13*, pages 247–261. Springer, 2016.

[12] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[13] Xiao Li, Kaiwen Liu, H Eric Tseng, Anouck Girard, and Ilya Kolmanovsky. Decision-making for autonomous vehicles with interaction-aware behavioral prediction and social-attention neural network. *arXiv preprint arXiv:2310.20148*, 2023.

[14] Yuan Lin, Antai Xie, and Xiao Liu. Autonomous vehicle decision and control through reinforcement learning with traffic flow randomization. *arXiv preprint arXiv:2403.02882*, 2024.

[15] Vu Trieu Minh. Trajectory generation for autonomous vehicles. In *Mechatronics 2013: Recent Technological and Scientific Advances*, pages 615–626. Springer, 2014.

[16] Tommaso Nesti, Santhosh Boddana, and Burhaneddin Yaman. Ultra-sonic sensor based object detection for autonomous vehicles. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 210–218, 2023.

[17] Santiago Royo and Maria Ballesta-Garcia. An overview of lidar imaging systems for autonomous vehicles. *Applied sciences*, 9(19):4093, 2019.

[18] Inga Rüb and Barbara Dunin-Kplicz. Bdi model of connected and autonomous vehicles. In *Computational Collective Intelligence: 11th International Conference, ICCCI 2019, Hendaye, France, September 4–6, 2019, Proceedings, Part II 11*, pages 181–195. Springer, 2019.

[19] Wenbo Shao, Jiahui Xu, Zhong Cao, Hong Wang, and Jun Li. Uncertainty-aware prediction and application in planning for autonomous driving: Definitions, methods, and comparison. *arXiv preprint arXiv:2403.02297*, 2024.

[20] Arvind Srivastav and Soumyajit Mandal. Radars for autonomous driving: A review of deep learning methods and challenges. *IEEE Access*, 2023.

[21] Vagisha Vartika, Swati Singh, Subhranil Das, Sudhansu Kumar Mishra, and Sitanshu Sekhar Sahu. A review on intelligent pid controllers in autonomous vehicle. *Advances in Smart Grid Automation and Industry 4.0: Select Proceedings of ICETS-GAI4. 0*, pages 391–399, 2021.

[22] Trieu Minh Vu, Reza Moezzi, Jindrich Cyrus, Jaroslav Hlava, and Michal Petru. Feasible trajectories generation for autonomous driving vehicles. *Applied Sciences*, 11(23):11143, 2021.

[23] Chaoyang Wang, Xiaonan Wang, Hao Hu, Yanxue Liang, and Gang Shen. On the application of cameras used in autonomous vehicles. *Archives of Computational Methods in Engineering*, 29(6):4319–4339, 2022.

[24] Letian Wang, Jie Liu, Hao Shao, Wenshuo Wang, Ruobing Chen, Yu Liu, and Steven L Waslander. Efficient reinforcement learning for autonomous driving with parameterized skills and priors. *arXiv preprint arXiv:2305.04412*, 2023.

[25] Shuyou Yu, Matthias Hirche, Yanjun Huang, Hong Chen, and Frank Allgöwer. Model predictive control for autonomous ground vehicles: a review. *Autonomous Intelligent Systems*, 1:1–17, 2021.

[26] Meng-Yue Zhang, Shi-Chun Yang, Xin-Jie Feng, Yu-Yi Chen, Jia-Yi Lu, and Yao-Guang Cao. Route planning for autonomous driving based on traffic information via multi-objective optimization. *Applied Sciences*, 12(22):11817, 2022.

[27] Zhengbang Zhu, Shenyu Zhang, Yuzheng Zhuang, Yuecheng Liu, Minghuan Liu, Liyuan Mao, Ziqin Gong, Weinan Zhang, Shixiong Kai, Qiang Gu, et al. Rita: Boost autonomous driving simulators with realistic interactive traffic flow. *arXiv preprint arXiv:2211.03408*, 2022.

Appendix: Safety Score Calculation Tables

The following tables present the average safety score calculations for various driving scenarios in different environments. Each table provides insights into the impact of weather conditions and the presence of RoTRA on the safety scores obtained.

Urban Environment (Table A.1): This table illustrates the average safety scores calculated for scenarios in urban settings, including traffic light intersections, crossroad intersections, following distance, and speeding. The impact of weather conditions, such as sunny weather, light rain, and heavy rain, is examined, along with the influence of RoTRA on safety scores.

Suburban Environment (Table A.2): Similar to the urban environment, this table presents the average safety scores calculated for scenarios in suburban settings. The impact of weather conditions and the presence of RoTRA on safety scores are analyzed for crossroad intersections, following distance, and speeding scenarios.

Rural Environment (Table A.3): Focusing on rural driving scenarios, this table showcases the average safety scores calculated for crossroad intersections, following distance, and speeding. The absence of traffic lights in rural environments and the influence of weather conditions on safety scores are highlighted.

Each safety score represents an average of 30 scenarios. The car is spawned in specific conditions to test specific rules, such as starting a scenario before a highway to incentivise the AV to speed.

Scenario	**Weather**	**AV Behaviour**	**Safety Score**
Traffic Light Intersection	Sunny	No RoTRA	85
		RoTRA	88
	Light Rain	No RoTRA	80
		RoTRA	86
	Heavy Rain	No RoTRA	71
		RoTRA	82
Crossroad Intersection[1]	Sunny	No RoTRA	84
		RoTRA	84
	Light Rain	No RoTRA	78
		RoTRA	84
	Heavy Rain	No RoTRA	69
		RoTRA	82
Following Distance	Sunny	No RoTRA	94
		RoTRA	94
	Light Rain	No RoTRA	87
		RoTRA	87
	Heavy Rain	No RoTRA	63
		RoTRA	71
Speeding	Sunny	No RoTRA	81
		RoTRA	98
	Light Rain	No RoTRA	76
		RoTRA	94
	Heavy Rain	No RoTRA	68
		RoTRA	89

Table A.1: Average Safety Score Calculation for Urban Environment

[1] Includes roundabouts.

Scenario	**Weather**	**AV Behaviour**	**Safety Score**
Traffic Light Intersection	Sunny	No RoTRA	85
		RoTRA	88
	Light Rain	No RoTRA	80
		RoTRA	86
	Heavy Rain	No RoTRA	70
		RoTRA	82
Crossroad Intersection[1]	Sunny	No RoTRA	84
		RoTRA	84
	Light Rain	No RoTRA	77
		RoTRA	84
	Heavy Rain	No RoTRA	67
		RoTRA	82
Following Distance	Sunny	No RoTRA	98
		RoTRA	98
	Light Rain	No RoTRA	90
		RoTRA	90
	Heavy Rain	No RoTRA	79
		RoTRA	81
Speeding	Sunny	No RoTRA	80
		RoTRA	98
	Light Rain	No RoTRA	73
		RoTRA	94
	Heavy Rain	No RoTRA	62
		RoTRA	90

Table A.2: Average Safety Score Calculation for Suburban Environment

[1] Includes roundabouts.

Scenario	**Weather**	**AV Behaviour**	**Safety Score**
Crossroad Intersection[1]	Sunny	No RoTRA	82
		RoTRA	82
	Light Rain	No RoTRA	76
		RoTRA	82
	Heavy Rain	No RoTRA	65
		RoTRA	80
Following Distance	Sunny	No RoTRA	86
		RoTRA	99
	Light Rain	No RoTRA	80
		RoTRA	95
	Heavy Rain	No RoTRA	75
		RoTRA	86
Speeding	Sunny	No RoTRA	80
		RoTRA	96
	Light Rain	No RoTRA	70
		RoTRA	92
	Heavy Rain	No RoTRA	59
		RoTRA	88

Table A.3: Average Safety Score Calculation for Rural Environment

[1] Includes roundabouts. [2] No traffic lights in a rural environment.